Commodore

DX-64

MANUAL

C64DX SYSTEM SPECIFICATION

o Design Concepts
o Hardware Specifications
o Software Specifications

Requires ROM Version 0.9A.910228 or later.

```
 CCCC       666     555555
C    C      6       5
C           6       5
C           6       55555
C          66666    5    5
C           6   6        5
C           6   6        5
C    C      6   6   5    5
 CCCC       6666     5555
```

This system specification contains the contributions of several people
including: Fred Bowen, Paul Lassa, Bill Gardei, and Victor Andrade.

Portions of the BASIC ROM code are Copyright 1977 Microsoft.


```
PPPP   RRRR   EEEE  L      I  M   M  I  N   N    A     RRRR  Y   Y
P   P  R   R  E     L      I  MM MM  I  NN  N   A A     R   R  Y Y
PPPP   RRRR   EEE   L      I  M M M  I  N N N   AAAAA   RRRR    Y
P      R   R  E     L      I  M   M  I  N  NN  A   A    R   R   Y
P      R   R  EEEE  LLLL   I  M   M  I  N   N  A   A    R   R   Y
```

Revision 0.2   (pilot release)   January 31, 1991

At this time, Pilot Production, the C65 system consists of either
revision 2A or 2B PCB, 4510R3, 4567R5 (PAL only), F011B/C FDC, and 018
DMAgic chips. There will be changes to all these chips before
Production Release.

This work is by:

        Fred Bowen   System Software -   C65
        Paul Lassa   Hardware engineer - C65, DMagic
        Bill Gardei   LSI engineer -   4567, FDC
        Victor Andrade   LSI engineer -   4510

Included are contributions by contractors hired by Commodore for the
C65 project. These contributions include the DOS, Graphics, Audio, and
Memory management areas.

Several 4502 assembler systems are available:

        VAX, Amiga, and PC based BSO-compatible cross assemblers.

        PC based custom cross assembler by Memocom, compatible
        with Memocom 4502 emulator and Mem-ulator systems.

        C128-based BSO compatible cross assembler by Commodore.

Custom software support is available for the following logic
analyzers:

        Hewlett Packard HP655x A and B logic analyzers.

Table of Contents

1.0. Introduction

This specification describes the requirements for a low-cost 8-bit
microcomputer system with excellent graphic capabilities.

1.1. System Concept

The C65 microcomputer is a low-cost, versatile, competitive product designed for the international home computer and game market.

The C65 is well suited for first time computer buyers, and provides an excellent upgrade path for owners of the commercially successful C64. The C65 is composed of concepts inherent in the C64 and C128.

The purpose of the C65 is to modernize and revitalize the 10 year old C64 market while still taking advantage of the developed base of C64 software. To accomplish this, the C65 will provide a C64 mode of operation, offering a reasonable degree of C64 software compatibility and a moderate degree of add-on hardware and peripheral compatibility. Compatibility can be sacrificed when it impedes enhanced functionality and expandability, much as the C64 sacrificed VIC-20 compatibility.

It is anticipated that the many features and capabilities of the new C65 mode will quickly attract the attention of developers and consumers alike, thereby revitalizing the low-end home computer market. The C65 incorporates features that are normally found on today's more expensive machines, continuing the Commodore tradition of maximizing performance for the price. The C65 will provide many new opportunities for third party software and hardware developers, including telecommunications, video, instrument control (including MIDI), and productivity as well as entertainment software.

1.2. System Overview

   o CPU -- Commodore CSG4510 running at 1.02 or 3.5 Mhz

         o New instructions, including Rockwell and GTE extensions

         o Memory Mapper supporting up to 1 Megabyte address space

         o R6511-type UART (3-wire RS-232) device, programmable baud
            rate (50-56K baud, MIDI-capable), parity, word size, sync
            and async. modes. XD/RD wire ORed/ANDed with user port.

         o Two CSG6526-type CIA devices, each with 2 I/O ports
            programmable TOD clocks, interval timers, interrupt control

   o Memory

         o RAM -- 128K bytes (DRAM)
            Externally expandable from additional 512K bytes to 4MB
            using dedicated RAM expansion port.

         o ROM -- 128K bytes
            C64 Kernel and BASIC 2.2
            C65 Kernel, Editor, BASIC 10.0, ML Monitor (like C128)
            DOS v10 (1581 subset)
            Multiple character sets: 40 and 80 column versions
            National keyboards/charsets for foreign language systems
            Externally expandable by conventional C64 ROM cartridges
            via cartridge/expansion port using C64 decodes.
            Externally expandable by additional 128K bytes or more
            via cartridge/expansion port using new system decodes.

o   DMA -- Custom DMAgic controller chip built-in
      Absolute address access to entire 8MB system map
      including I/O devices, both ROM & RAM expansion ports.
      List-based DMA structures can be chained together
      Copy (up,down,invert), Fill, Swap, Mix (boolean Minterms)
      Hold, Modulus (window), Interrupt, and Resume modes,
      Block operations from 1 byte to 64K bytes
      DRQ handshaking for I/O devices
      Built-in support for (optional) expansion RAM controller

o Video -- Commodore CSG 4567 enhanced VIC chip

    o   RGBA with sync on all colors or digital sync

    o   Composite NTSC or PAL video, separate chroma/luma

    o   Composite NTSC or PAL digital monochrome

    o   RF TV output via NTSC or PAL modulator

    o   Digital foreground/background control (genlock)

    o   All original C64 video modes:
        40x25 standard character mode
        Extended background color mode
        320x200 bitmap mode
        Multi-color mode
        16 colors
        8 sprites, 24x21

    o   40 and 80 character columns by 25 rows:
          Color, blink, bold, inverse video, underline attributes

    o   True bitplane graphics:
            320 x 200 x 256 (8-bitplane) non-interlaced
            640 x 200 x 16* (4-bitplane) non-interlaced
           1280 x 200 x 4*   (2-bitplane) non-interlaced
            320 x 400 x 256 (8-bitplane) interlaced
            640 x 400 x 16* (4-bitplane) interlaced
           1280 x 400 x 4*   (2-bitplane) interlaced
                   *plus sprite and border colors

    o   Color palettes:
          Standard 16-color C64 ROM palette
          Programmable 256-color RAM palette, with 16 intensity
          levels per primary color (yielding 4096 colors)

    o   Horizontal and vertical screen positioning verniers

    o   Display Address Translator (DAT) allows programmer to
        access bitplanes easily and directly.

    o   Access to optional expansion RAM

    o   Operates at either clock speed without blanking

o Audio -- Commodore CSG8580 SID chips

o   Stereo SID chips:
        Total of 6 voices, 3 per channel
        Programmable ADSR envelope for each voice
        Filter, modulation, audio inputs, potentiometer
        Separate left/right volume, filter, modulation control

o Disk, Printer support --

    o FDC custom MFM controller chip built in, with 512-byte
       buffer, sector or full track read/write/format, LED and
       motor control, copy protection.

    o Built-in 3.5" double sided, 1MB MFM capacity drive

    o Media & file system compatible with 1581 disk drive

    o Supports one additional "dumb" drive externally.

    o Standard CBM bus serial (all modes, about 4800 baud)

    o Fast serial bus (C65 mode only, about 20K baud)

    o Burst serial (C65 mode only, about 50K baud)

o External ports --

    o 50-pin Cartridge/expansion port (ROM cartridges, etc.)

    o 24-pin User/parallel port (modem (1670), RS-232 serial)

    o Composite video/audio port (8-pin DIN)

    o Analog RGB video port (DB-9)

    o RF video output jack

    o Serial bus port (disks (1541/1571/1581), printers, etc.)

    o External floppy drive port (mini DIN8)

    o 2 DB9 control ports (joystick, mouse, tablets, lightpen)

    o Left and right stereo audio output jacks

    o RAM expansion port, built-in support for RAM controller

o Keyboard -- 77 keys, including standard C64 keyboard plus:

    o Total of 8 function keys, F1-F16, shifted and nonshifted

    o TAB, escape, ALT, CAPS lock, no scroll, help (F15/16)

    o Power, disk activity LEDs

o Power supply -- external, brick type

    o +5VDC at 2.2A and +12VDC at .85A

1.3. System Components

Microcontroller: 4510 (65CE02, 2x6526, 6511 UART, Mapper,
                            Fast serial)

Memory: 4464   DRAM (128K bytes)
                       271001 ROM (128K bytes)

Video controller: 4567 (extended VIC, DAT, PLA)

Audio controllers: 6581 (SID)

Memory control: 41xx-F018 (DMA)

Disk controller: 41xx-F011 (FDC, supports 2 DSDD drives, MFM,
                              RAM buffer)


```
KEYS
 + USER PORT
 | + CONTROL PORTS                              EXPANSION PORT
 | | +                                      + + +  +---+
 | | |                                      | | |  |   | +MOD-> RFOUT
++-+-++                                      | | |  |   +-+----> COMP,CHROMA/LUMA
|     |                                      | | |  |   +------> RGBA
|     +-------------------------------------+-----+  |      +---+
|     +-------------------------------------+  +--...-+ R +--+
|     |                                  | | |  |        E   | EXPANSION
|     |   +--+  +--+  +--+  +--+  +--+ | | |  +--...-+ C +--+   MEMORY
| 4   |   |  |  |  |  |  |  |  |  |  | | | |  | 4 |     +---+
| 5   +-----+ D +--+ F +--+ S +--+ S +--+ R +-+----+ 5 |  +--+  +--+  +--+  +--+
| 1   +-----+ M +--+ D +--+ I +--+ I +--+ O +------+ 6 |  |  |  |  |  |  |  |  |
| 0   | ADR | A |  | C |  | D |  | D |  | M |   | 7 +--+  +--+  +--+  +--+  |
|     |     | G |  |  |  |  |  |  |  |  |  |   |  |  +--+  +--+  +--+  +--+  |
|     +-----+ I +--+   +--+   +--+   +--+   +---+--+  +--+  +--+  +--+  +--+  |
|     +-----+ C +--+   +--+   +--+   +--+   +------+  +--+  +--+  +--+  +--+  |
|     | DAT |  |  |  |  |  |  |  |  |  |  |   |   |  |  |  |  |  |  |  |  |  |  |
+--+--+     +---+ +++-+ +-+-+ +-+-+ +---+       +---+  +--+  +--+  +--+  +--+
  |         |       ||     |       |                               128K
  +         |       ||     R       L                           RAM INTERNAL
SERIAL BUS  |       ||        SPEAKERS
                    ++
              FLOPPY PORT
```


1.4. System Concerns

1.4.1. C64 Compatibility Issues

1.4.1.1. Software

C64 software compatibility is an important goal. To this end, when the
system is in "C64 mode" the processor will operate at average 1.02MHz
speed and dummy "dead" cycles are emulated by the processor. The C64
ROM is the same except for patches to serial bus routines in the
kernel (to interface built-in drive), the removal of cassette code
(there is no cassette port), and patches to the C64 initialization

routines to boot C65 mode if there is no reason (eg., cartridges) to stay in C64 mode.

Compatibility with C64 software that uses previously unimplemented 6502 opcodes (often associated with many copy-protection schemes) or that implements extremely timing dependent "fast loaders" is iherently impossible. Because the VIC-III timing is slightly different, programs that are extremely timing dependant may not work properly. Also because the VIC-III does not change display modes until the end of a character line, programs that change displays based strictly upon the raster position may not display things properly. The aspect ratio of the VIC-III display is slightly different than the VIC-II. The use of a 1541-II disk drive (optional) will improve compatibility. C64 BASIC 2.2 compatibility will be 100% (within hardware constraints). C128 BASIC 10 compatibility will be moderate (graphic commands are different, some command parameters different, and there are many new commands).

1.4.1.2. Hardware

C64 hardware compatibility is limited. Serial bus and control port devices (mouse, joysticks, etc.) are fully supported. Some user port devices are supported such as the newer (4-DIP switch) 1670 modems, but there's no 9VAC so devices which require 9VAC won't function correctly. The expansion port has additional pins (50 total), and the pin spacing is closer than the C64 (it's like the PLUS/4). An adaptor ("WIDGET") will be necessary to utilize C64 cartridges and expansion port devices. Furthermore, timing differences between some C64 and C65 expansion port signals will affect many C64 expansion devices (such as the 1764).

1.4.2. DOS Compatibility

The built-in C65 DOS is a subset of Commodore 1581 DOS. There is no track cache, index sensor, etc. To load and run existing 1541-based applications, the consumer must add a 1541 drive to the system. Many commercial applications cannot be easily ported from 1541/5.25" media to 1581/3.5" media, due to copy protection or "fast loaders". Most C64 applications that directly address DOS memory, specific disk tracks or sectors, or rely on DOS job queues and timing characteristics will not work with the built-in drive and new DOS.

1.4.3. Operating Modes

The C65 powers up in the C64 mode. If there are no conditions present which indicate that C64 mode is desired, such as the C= key depressed or a C64 cartridge signature found, then C65 mode will be automatically brought into context. Unlike the C128, "C6 4 mode" is escapable. Like the C128, all of the extended features of the C65 system are accessible from "C64 mode" via custom software. Whenever the system initiates C64 mode, new VIC mode is always disabled except when the DOS is required.

1.5. System Maps

1.5.1. Composite System Memory Map

C64 CARTRIDGES	C64	C65	RAM-LO	RAM-HI

```
$FFFF+----------+ +----------+ +----------+ +----------+ +----------+
     |          | |          | |          | |          | |COLOR NYBS|
$F800|  GAME    | | KERNEL   | | KERNEL   | |          | +----------+
     |          | |   &      | |   &      | |          | |          |
     |  CARD    | | EDITOR   | | EDITOR   | |          | |          |
     |          | |          | |          | |..........| |..........|
$E000+----------+ +----------+ +----------+ | C65 EVEN | | C65 ODD  |
                  |COLOR NYBS| |COLOR NYBS| | BITPLANES| | BITPLANES|
                  |I/O & CHARS| |I/O & CHARS| |..........| | ........ |
$D000 ----------- +----------+ +----------+ |          | |          |
                               |          | |          | |          |
                               | KERNEL   | |          | |          |
                               |          | | C65 BASIC| | C65 VARS &|
$C000+----------+ +----------+ +----------+ |  TEXT    | | STRINGS  |
     |          | |          | |          | |$2000-$FEFF| |$2000-$F7FF|
     |APPLICATION| |          | |          | |          | |          |
     |          | | BASIC    | |          | |          | |          |
     | CARD _ HI| |          | | BASIC    | |          | |          |
     |          | |          | | GRAPHICS | |          | |          |
$A000+----------+ +----------+ |          | +----------+ |          |
     |          |              |          | |          | |          |
     |APPLICATION|             |  DOS     | |          | |          |
     |          |              | (MAPPED) | |          | |          |
     | CARD _ LOW|             |          | | C64 VARS &| |          |
     |          |              |          | | STRINGS  | |          |
$8000+----------+ ------------ +----------+ | TEXT-$BFFF| |          |
                               |COLOR NYBS| |          | |          |
                               |I/O & CHARS| | C64 BASIC| |          |
$6000 ------------------------- +----------+ |  TEXT    | |          |
                               |          | |$0800-VARS| |          |
                               |          | |          | |          |
                               |          | |          | |          |
                               |          | |          | |          |
                               | BASIC    | |          | |          |
                               |          | |          | |          |
                               |          | |          | |          |
                               |          | |          | |          |
$2000 ------------------------- +----------+ +----------+ +----------+
                                            | C65 SYSTEM| | C64 & C65|
                                            |TEXTSCREENS| |   DOS    |
$0000 --------------------------------------- +----------+ +----------+
```

1.5.2. C65 System Memory Map

```
        MAPPER BANK
        -----+-----
             |
             |
    1M    $F,FFFF  +------------+ ----------
                   |            |
                   +-        -+
                   |   RAM    |      512K BLOCK APPEARING
   768K   $C,0000  +-        -+      HERE IS DETERMINED BY
                   | EXPANSION|      THE RAM EXPANDER CTLR
                   +-        -+      (UP TO 8MB TOTAL MAP)
                   |            |
   512K   $8,0000  +------------+ ----------
                   |            |
                   +- RESERVED -+    FUTURE CARTRIDGES
                   |            |
```

```
  256K  $4,0000  +------------+ ----------
                 | SYSTEM ROMS |
  128K  $2,0000  +------------+    SEE SYSTEM MEMORY
                 | SYSTEM ROMS |    LAYOUT, BELOW
        $0,0000  +------------+ ----------
```

1.5.3. C65 System Memory Layout

```
              BANK 0           BANK 1           BANK 2           BANK 3
              RAM-LO           RAM-HI           ROM-LO           ROM-HI
$FFFF  +------------+ +------------+ +------------+ +------------+
$F800  |            | | COLOR NYBS | |    C64     | |    C65     |
       |            | +------------+ |   KERNEL   | |   KERNEL   |
$E000  | BITPLANES  | |            | +------------+ +------------+
       |   (EVEN)   | |            | | C64 CHRSET | |            |
$D000  |            | | BITPLANES  | +------------+ |  RESERVED  |
       |            | |   (ODD)    | | INTERFACE  | |            |
$C000  +............+ +............+ +------------+ +------------+
       |            | |            | |    C64     | |            |
       |            | |            | |   BASIC    | |            |
$A000  | STRUCTURES | |  STRINGS   | +------------+ |  GRAPHICS  |
       |    ???     | |            | |    C65     | |            |
       |            | |            | |   CHRSET   | |            |
$8000  +............+ +............+ +------------+ +------------+
       |            | |            | |            | |            |
       |            | |            | |            | |            |
       |            | |            | |            | |            |
       |            | |            | |            | |            |
       |   BASIC    | |   BASIC    | |  RESERVED  | |  C65 BASIC |
       |   TEXT     | | VARIABLES  | |            | |            |
       |            | |            | |            | |            |
       |            | |            | |            | |            |
$4000  |            | |            | +------------+ |            |
       |            | |            | |            | |            |
       |            | |            | |            | |            |
       |            | |            | |            | |            |
$2000  +------------+ +------------+ |            | +------------+
       | TEXT SCREEN | |    DOS     | |    DOS     | |  MONITOR   |
       +------------+ |            | |            | |            |
       |            | |  BUFFERS   | |  (MAPS TO  | |  (MAPS TO  |
       | SYSTEM VARS | |  & VARS    | |   $8000)   | |   $6000)   |
       |            | |            | |            | |            |
$0000  +------------+ +------------+ +------------+ +------------+
```

What does this mean?   Here is what the 64K memory map looks like in
various configurations (i.e., as seen by the processor):

```
C64 mode:      $E000-$FFFF    Kernel, Editor, Basic overflow area
---------      $D000-$DFFF    I/O and Color Nybbles, Character ROM
               $C000-$CFFF    Application RAM
               $A000-$BFFF    BASIC 2.2
               $0002-$9FFF    RAMLO.  VIC screen at $0400-$07FF
                              BASIC program & vars from $0800-$9FFF

C65 mode:      $E000-$FFFF    Kernel, Editor ROM code
---------      $D000-$DFFF    I/O and Color Bytes (CHRROM at $29000)
               $C000-$CFFF    Kernel Interface, DOS ROM overflow area
               $8000-$BFFF    BASIC 10.0 Graphics & Sprite ROM code
               $2000-$7FFF    BASIC 10.0 ROM code
```

```
            $0002-$1FFF       RAMLO.  VIC screen at $0800-$0FFF
                              BASIC prgs mapped from $02000-$0FF00
                              BASIC vars mapped from $12000-$1F7FF


C65 DOS mode:  $E000-$FFFF    Kernel, Editor ROM code
-------------  $D000-$DFFF    I/O (CIA's mapped out), Color Bytes
               $C800-$CFFF    Kernel Interface
               $8000-$C3FF    DOS ROM code
               $2000-$7FFF      (don't care)
               $0000-$1FFF    DOS RAMHI


C65 Monitor:   $E000-$FFFF    Kernel, Editor ROM code
------------   $D000-$DFFF    I/O and Color Bytes
               $C000-$CFFF    Kernel Interface
               $8000-$BFFF      (don't care)
               $6000-$7FFF    Monitor ROM code
               $0002-$5FFF    RAMLO
```

It's done this way for a reason. The CPU MAPPER restricts the
programmer to one offset for each 32Kbyte half of a 64Kbyte segment.
For one chunk of ROM to MAP in another chunk with a different offset,
it must do so into the other half of memory from which it is
executing. The OS does this by never mapping the chunk of ROM at
$C000-$DFFF, which allows this chunk to contain the Interface/MAP code
and I/O (having I/O in context is usually desirable, and you can't map
I/O anyhow). The Interface/MAP ROM can be turned on and off via VIC
register $30, bit 5 (ROM @ $C000), and therefore does not need to be
mapped itself. Generally, OS functions (such as the Kernel, Editor,
and DOS) live in the upper 32K half of memory, and applications such
as BASIC or the Monitor) live in the lower 32K half. For example,
when Monitor mode is entered, the OS maps out BASIC and maps in the
Monitor. Each has ready access to the OS, but no built-in access to
each other. When a DOS call is made, the OS overlays itself with the
DOS (except for the magical $C000 code) in the upper 32K half of
memory, and overlays the application area with DOS RAM in the lower
32K half of memory.

1.5.4. C65 System I/O Memory Map


```
         +-------------+
 $DF00   |    I/O-2    |      EXTERNAL I/O SELECT
 $DE00   |    I/O-1    |      EXTERNAL I/O SELECT
         +-------------+
 $DD00   |    CIA-2    |      SERIAL, USER PORT
 $DC00   |    CIA-1    |      KEYBOARD, JOYSTICK, MOUSE CONTROL
         +-------------+
 $D800   |  COLOR NYB  |      COLOR MATRIX (*FROM $1F800-$1FFFF)
         +-------------+
 $D700   |     DMA     |      *DMA CONTROLLER
         +-------------+
 $D600   |    UART     |      *RS-232, FAST SERIAL, NEW KEY LINES
         +-------------+
 $D440   |   SID (L)   |      AUDIO CONTROLLER (LEFT)
 $D400   |   SID (R)   |      AUDIO CONTROLLER (RIGHT)
         +-------------+
 $D300   | BLU PALETTE |
 $D200   | GRN PALETTE |      *COLOR PALETTES (NYBBLES)
 $D100   | RED PALETTE |
         +-------------+
 $D0A0   |     REC     |      *RAM EXPANSION CTRL (OPTIONAL)
```

```
            +-------------+
  $D080     |     FDC     |      *DISK CONTROLLER
            +-------------+
  $D000     |  VIC-4567   |       VIDEO CONTROLLER
            +-------------+
                  .
                  .
                  .
            +-------------+
  $0000     |    4510     |       MEMORY CONTROL FOR C64 MODE
            +-------------+       (this register is actually in
                                   the VIC-4567 in the C65)
```

*NOTE: VIC must be in "new" mode to address these devices

2.0. C65 System Hardware

2.1.1. Keyboard Layout

```
+----+   +----+----+----+----+   +----+----+----+----+   +----+----+----+----+
|RUN |   |ESC |ALT |ASC | NO |   | F1 | F3 | F5 | F7 |   | F9 | F11| F13|HELP|
|STOP|   |    |    |DIN |SCRL|   | F2 | F4 | F6 | F8 |   | F10| F12| F14|    |
+----+   +----+----+----+----+   +----+----+----+----+   +----+----+----+----+


+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
| <- | !  | "  | #  | $  | %  | &  | '  | (  | )  |    |    |    |    |CLR |INST|
|    | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 0  | +  | -  | œ  |HOME|DEL |
+----+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+----+----+
|  TAB   |    |    |    |    |    |    |    |    |    |    |    |    | ã  | RSTR |
|        | Q  | W  | E  | R  | T  | Y  | U  | I  | O  | P  | @  | *  | ^  |      |
+----+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+------+
|CTRL|SHFT|    |    |    |    |    |    |    |    | [  | ]  |    |   RETURN   |
|    |LOCK| A  | S  | D  | F  | G  | H  | J  | K  | L  | :  | ;  | =  |            |
+----+----+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+----+----+
| C= | SHIFT  |    |    |    |    |    |    | <  | >  | ?  | SHIFT|CRSR|
|    |        | Z  | X  | C  | V  | B  | N  | M  | ,  | .  | /  |      | UP |
+----+-------+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+--+-+----+----+----+
         |                    SPACE                    |   |CRSR|CRSR|CRSR|
         |                                             |   |LEFT|DOWN|RITE|
         +---------------------------------------------+   +----+----+----+
```

Notes:

1/ The cursor keys are special -- the shifted cursor keys appear as
   separate keys, but in actuality pressing them generates a SHIFT
   plus the normal cursor code, making them totally compatible with,
   and therefore functional in, C64 mode.

2/ There are a total of 77 keys, two of which are locking keys.

3/ The NATIONAL keyboards are similar, and their layout and operation
   is identical to their C128 implementation.

2.1.2. Keyboard Matrix

```
        +-----+-----+-----+-----+-----+-----+-----+-----+-----+        +--
---+
```

```
                | C0  | C1  | C2  | C3  | C4  | C5  | C6  | C7  | C8  |              |
GND |
                |PIN20|PIN19|PIN18|PIN17|PIN16|PIN15|PIN14|PIN13|PIN-4|              |
|PIN-1|
                +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+           +--
+--+
                 |     |     |     |     |     |     |     |     |
|
                 |     |     |     |     |     |     |     |     |
|
                 V     V     V     V     V     V     V     V     V
|
+-----+      +-----+-----+-----+-----+-----+-----+-----+-----+-----+
|
| R0  |<----+ INS | #   | %   | '   | )   | +   | œ   | !   | NO  |
|
|PIN12|     | DEL | 3   | 5   | 7   | 9   |     |     | 1   | SCRL|
|
+-----+      +-----+-----+-----+-----+-----+-----+-----+-----+-----+
|
| R1  |<----+ RET | W   | R   | Y   | I   | P   | *   | <-- | TAB |
|
|PIN11|     |     |     |     |     |     |     |     |     |     |
|
+-----+      +-----+-----+-----+-----+-----+-----+-----+-----+-----+
|
| R2  |<----+ HORZ| A   | D   | G   | J   | L   | ]   | CTRL| ALT |
|
|PIN10|     | CRSR|     |     |     |     |     | ;   |     |     +----------+
|
+-----+      +-----+-----+-----+-----+-----+-----+-----+-----+-----+          |
|
| R3  |<----+ F8  | $   | &   | {   | 0   | -   | CLR | "   | HELP|          |
|
|PIN-9|     | F7  | 4   | 6   | 8   |     |     | HOM | 2   |     |          |
|
+-----+      +-----+-----+-----+-----+-----+-----+-----+-----+-----+          |
|
| R4  |<----+ F2  | Z   | C   | B   | M   | >   |RIGHT|SPACE| F10 |          |
|
|PIN-8|     | F1  |     |     |     |     | .   |SHIFT| BAR | F9  |          |
|
+-----+      +-----+-----+-----+-----+-----+-----+-----+-----+-----+          |
|
| R5  |<----+ F4  | S   | F   | H   | K   | [   | =   | C=  | F12 |          |
|
|PIN-7|     | F3  |     |     |     |     | :   |     |     | F11 |          |
|
+-----+      +-----+-----+-----+-----+-----+-----+-----+-----+-----+          |
|
| R6  |<----+ F6  | E   | T   | U   | O   | @   | ã   | Q   | F14 |          |
|
|PIN-6|     | F5  |     |     |     |     |     | ^   |     | F13 |          |
|
+-----+      +-----+-----+-----+-----+-----+-----+-----+-----+-----+          |
|
| R7  |<----+ VERT|LEFT | X   | V   | N   | <   | ?   | RUN | ESC +------+   |
|
|PIN-5|     | CRSR|SHIFT|     |     |     | ,   | /   | STOP|     +--+   |   |
|
+-----+      +--+--+--+--+-----+-----+-----+-----+--+--+-----+-----+  |   |   |
```

```
|
                |       |                               |                   |   |   |
|
                |       |                               |                   |   |   |
|
                |    +--+--+     /         (LOCKING)    |                   |   |   |
|
                |    |SHIFT+----+  +------------------------------------+   |   |
|
                |    | LOCK|                            |                   |   |
|
                |    +-----+                            |                   |   |
|
                |                            +-----+-----+                  |   |
|
           +--+--+                           |         |                   |   |
|
           |CRSR +------------+------------+           +---------------+   |
|
           | UP  |      K1 PIN-21          |           |                   |
|
           +--+--+                         |   4066    |                   |
|
           |                               | DECODER   |                   |
|
           +--+--+                         |           |                   |
|
           |CRSR +------------+------------+           +------------------+
|
           |LEFT |      K2 PIN-22          |           |
|
           +-----+                         +-----------+
|

|
+-----+            +-----+     /
|
| NMI | <---------+RESTR+----+  +---------------------------------------------------
+
|PIN-3|           |    |
|
+-----+           +-----+
|

|

|
+-----+            +-----+     /         (LOCKING)
|
| R8  | <---------+CAPS +----+  +---------------------------------------------------
+
|PIN-2|           |LOCK |
+-----+           +-----+
```
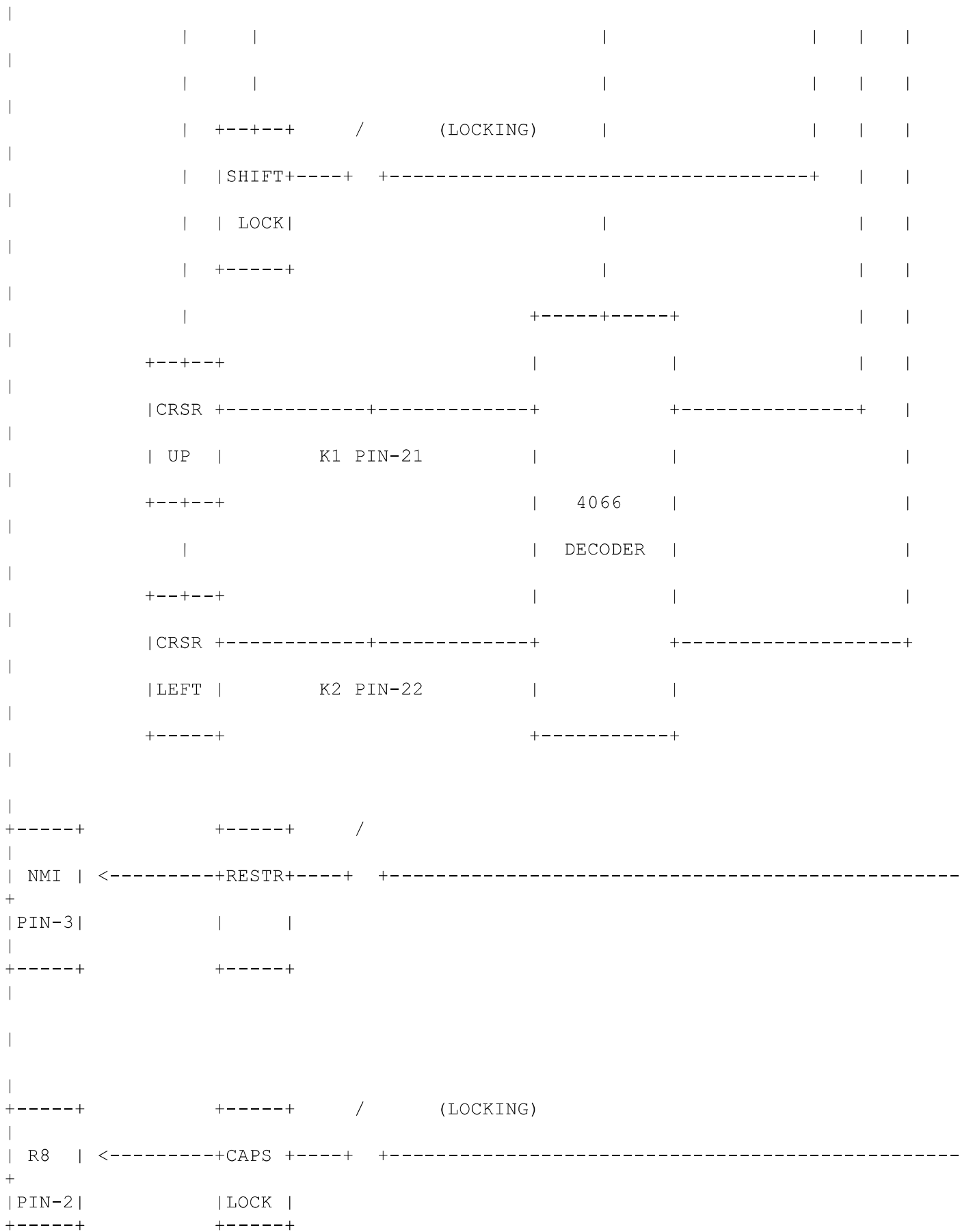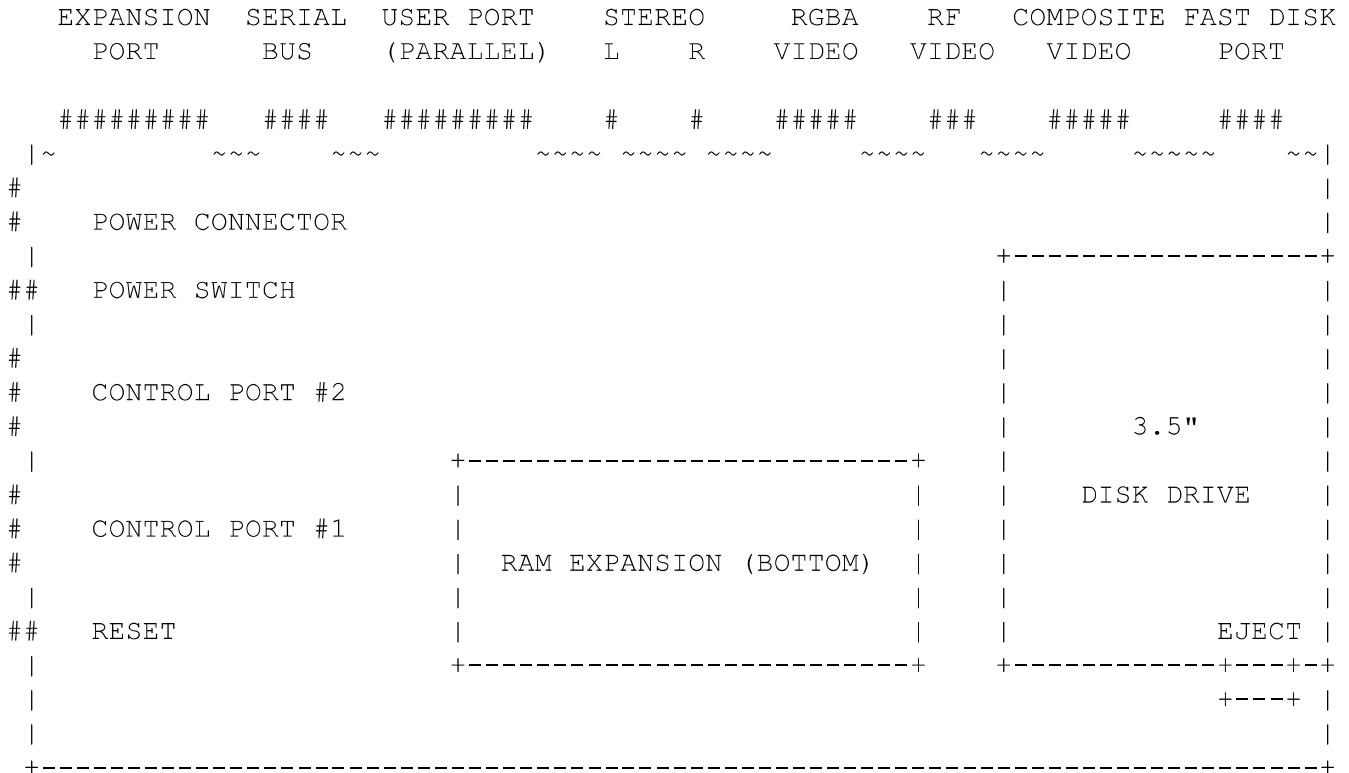
Keyboard Notes:

1/ The 64 keys under C0 through C7 occupy the same matrix position as
   in the C/64, as does the RESTORE key. Including SHIFT-LOCK, there
   are 66 such keys.

2/ The extended keyboard consists of the 8 keys under the C8 output.

Counting the CAPS-LOCK key, there are 9 new keys. The C/64 does not
scan these keys.

3/ The new CURSOR LEFT and CURSOR UP keys simulate a CURSOR plus RIGHT
SHIFT key combination.

4/ The keyboard mechanism will be mechanically similar to that of the
C128.

2.2. Form Factor

```
   EXPANSION   SERIAL   USER PORT     STEREO      RGBA     RF   COMPOSITE FAST DISK
     PORT       BUS     (PARALLEL)    L    R      VIDEO   VIDEO   VIDEO     PORT


   #########   ####    #########    #    #     #####    ###     #####      ####
  |~           ~~~      ~~~        ~~~~ ~~~~ ~~~~    ~~~~    ~~~~     ~~~~~     ~~|
 #                                                                              |
 #     POWER CONNECTOR                                                          |
  |                                                     +------------------+
 ##     POWER SWITCH                                    |                  |
  |                                                     |                  |
 #                                                      |                  |
 #     CONTROL PORT #2                                  |                  |
 #                                                      |        3.5"      |
  |                          +-------------------------+ |                  |
 #                           |                         | |   DISK DRIVE     |
 #     CONTROL PORT #1       |                         | |                  |
 #                           |   RAM EXPANSION (BOTTOM)| |                  |
  |                          |                         | |                  |
 ##     RESET                |                         | |          EJECT   |
  |                          +-------------------------+ +-----------+---+-+
  |                                                            +---+ |
  |                                                                  |
  |                                                                  |
  +---------------------------------------------------------------------------+
```

NOTES:

1. Dimensions: about 18" wide, 8" deep, 2" high.
2. Disk unit faces forward.

2.3. The CSG 4510 Microcontroller Chip

2.3.1. Description

    This specification describes the requirements for a single chip
8-bit microcontroller   unit fabricated in 2U CMOS double-metal
technology for high speed and low power consumption.
    The IC is a fully static device that contains an enhanced 6502
microprocessor (65CE02), four independent 16-bit interval timers/two
24-hour (AM/PM) time of day clocks each with programmable alarm,
full-duplex serial I/O (UART) channel with programmable baud rate
generator, built-in memory map function to access up to 1 megabyte of
memory, two 8-bit shift registers for synchronous serial I/O, and 30
individually programmable I/O lines.

2.3.2. Configuration

This IC device shall be configured in a standard, 84-pin plastic chip carrier package.   [*** Pinout below will change for 4510R5 ***]

```
                    A A A F S C S C S V V C C R E R I N R T T
                    2 1 0 L R N P N P C S O A E X S R M X X E
                        A Q T 1 T 2 C S L P S T T Q I D D S
                        G I 1   2       8 S E R R * *     T
                        2 N             L T * *
                        * *             K *

                    1 1                 8 8 8 8 8 7 7 7 7 7
                    1 0 9 8 7 6 5 4 3 2 1 4 3 2 1 0 9 8 7 6 5
    A3      12  +---------------------------------------+ 74  C7MHZ
    A4      13  |                                       | 73  SRQDAT
    A5      14  |                                       | 72  SRQCLK
    A6      15  |                                       | 71  SRQATN
    A7      16  |                                       | 70  PA2
    A8      17  |                                       | 69  COL7
    A9      18  |                                       | 68  COL6
    A10     19  |                                       | 67  COL5
    A11     20  |                                       | 66  COL4
    A12     21  |                                       | 65  COL3
    A13     22  |               CSG 4510                | 64  COL2
    A14     23  |                                       | 63  COL1
    A15     24  |                                       | 62  COL0
    A16     25  |                                       | 61  ROW7
    A17     26  |                                       | 60  ROW6
    A18     27  |                                       | 59  ROW5
    A19     28  |                                       | 58  ROW4
    PSYNC   29  |                                       | 57  ROW3
    AEC     30  |                                       | 56  ROW2
    DMA*    31  |                                       | 55  ROW1
    NOIO    32  +---------------------------------------+ 54  ROW0

                    3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 5 5 5 5
                    3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3

                    N D D D D D D D D V P R P P P P P P P P P
                    O B B B B B B B B C H / B B B B B B B B C
                    M 7 6 5 4 3 2 1 0 C O W 0 1 2 3 4 5 6 7 2
                    A
                    P
```

2.3.3. Functional Description

2.3.3.1. Pin Description

PIN NAME	PIN NUMBER	SIGNAL DIRECTION	DESCRIPTION
----	------	---------	
VSS	1	IN	This is the power ground signal (0 volts).
VCC	2,42	IN	This is the power supply signal (+5 volts).
SPB,	3	I/O	The SPA and SPB signals are open-drain

SPA	5	I/O	and bidirectional, each with a 3Kohm (min.) passive pull-up. The SPA and SPB signals are the data lines used by the two 8-bit synchronous serial port registers. In input mode, SPA and SPB are clocked into the device on the rising edge of the CNTA and CNTB clocks, respectively. In the output mode, SPA and SPB change on the falling edge of the CNTA and CNTB clocks, respectively.
CNTB, CNTA	4 6	I/O I/O	The CNTA and CNTB signals are open-drain and bidirectional, each with a 3K ohm (min.) passive pull-up. These pins are internally synchronized to the PH0 clock and then used to clock the synchronous serial registers, in input mode. In output mode, each pin will reflect the clock signal derived from the corresponding timer.
FLAGA/ FLAGB/	1 8	I/O IN	The FLAGA/ and FLAGB/ inputs are negative edge sensitive input signals. A passive pull-up (3Kohm min.) is tied on each of these pins. They are internally synchronized to the PH0 clock and are used as general purpose interrupt inputs. Any negative transition on either of these signals will cause the device to start an interrupt sequence, provided that the proper bit is set in each of the interrupt mask registers. The device-will drop the IRQ/ line to indicate that an interrupt sequence is underway.

```
*** When the FAST SERIAL MODE is enabled the CNTA, SPA and ***
*** FLAGA/ lines will not function as described above. See ***
*** section 2.5.6. for FAST SERIAL MODE description.      ***
```

A0-A19	9 thru 28	I/O	Address Bus - This is a 20 bit bi-directional bus with tri-state outputs. The output of each address line is TTL compatible, capable of driving two standard TTL loads and 55 pf. When the AEC or DMA/ line goes low, the bus goes tri-state. If AEC only is low, A17, A18 and A19 will each reflect the state of the A16 line. During an I/O access (IO/ is low), A0-A3, A8 and A9 are used to select an internal I/O register. If AEC is high, the bus will be driven by the CPU and A16-A19 will point to a mapped memory location (if MAP/ is low). If memory is not mapped (MAP/ is high), A16-A19 will be low.
PSYNC	29	OUT	This output line is provided to identify those cycles in which the microprocessor is doing an OP CODE fetch. The PSYNC line goes high during PHI of an OP CODE fetch and stays high for the remainder of that cycle. If AEC or DMA/ is low during the rising edge of PHI, in which pulse PSYNC went high, the processor will stop in its current state and will remain, in the state until

| | | | either AEC or DMA/ goes high. In this manner, the SYNC signal can be used to control either the AEC or DMA/ line to cause single instruction execution. |

Signal	Pin	Dir	Description
AEC	30	IN	This input signal is the Address Enable Control line. When high, the address bus, R/W are valid. When low, the address bus, R/W and MAP/ are in a high-impedance state except for A17, A18 and A19 each of which will be connected to the A16 line.
DMA/	31	IN	This signal is connected to a 3K passive pull-up. When this signal is low the address bus and R/W will be tri-stated. This will allow external DMA devices to assume control of the system bus lines.
(READY)	Internal Signal		This signal is generated internally via the AEC and DMA/ lines. The READY signal goes high when both AEC and DMA/ are high. It goes low if either AEC or DMA/ goes low. The READY signal allows the user to single-cycle the microprocessor on all cycles including write cycles. A low state on either DMA/ or AEC during the rising transition of phase one (PHI) will deassert the READY line and halt the micro processor with the output address lines holding the current address. This feature allows microprocessor interfacing with low speed memory as well as fast (max 2 cycle) Direct Memory Access (DMA).
IO/	32	IN	This input signal is used to select the internal registers of the device, provided memory is not being mapped by the CPU.
MAP/	33	OUT	This signal is passively pulled-up (3 Kohm) whenever DMA/ or AEC is pulled low. This output signal is used to indicate whether or not memory is being mapped by the device. If the CPU is addressing a mapped memory region the MAP/ line will go low and will inhibit the IO/ line from selecting an internal register. If the CPU is not mapping memory the MAP/ line will be-high and A16-A19 will be kept low.
DB7-DB0	34 thru 41	I/O	D0-D7 form an 8 bit bi-directional data bus for data exchanges to and from the internal CPU (the 65CE02) and the device internal registers. It is also used to communicate with external peripheral devices. The output buffers are capable of driving two standard TTL loads and 55pf.
R/W	43	I/O	This signal is generated by the CPU to control the direction of data transfers on the data bus. This line is high except when the CPU is writing to memory, an internal I/O register or an external device. When the AEC or DMA/ signal is low, the R/W becomes tri-state.

```
PH0          44        IN    This clock is a TTL compatible input used for.
                             internal device operation and as a timing
                             reference for communicating with the system
                             data bus. Two internal clocks are generated by
                             the device; phase two (PH2) is in phase with PH0,
                             and phase one (PH1) is 180 degrees out of phase
                             with PH0.

PC/          53        OUT   This output line is a strobe signal and is
                             Centronics interface compatible. The signal goes
                             low following a read or write access of PORT D.

PRD0-PRD7  45 thru 52  I/O   These are three 8-bit ports with each of their
PRB0-PRB7  54 thru 61  I/O   lines having a passive pull-up  (min. 3K ohm)
PRA0-PRA7  62 thru 69  I/O   as well as active pull-up and pull-down
                             transistors. Each individual port line may be
                             programmed to be either input or output.

PRC2         70        I/O   This line corresponds to PORT C, bit 2.
                             It has passive pull-up (min. 3k ohm) as
                             well as active pull-up and pull-down
                             transistors. The line can be configured
                             as input or output. PRC2 becomes the
                             external shift register clock when the
                             UART is configured to operate in the
                             synchronous mode, otherwise PRC2
                             operates as normal.

PRC3         71        OUT   This signal is an open drain output with a
                             passive pull-up (1K ohm min). It corresponds to
                             bit 3 of PORT C. When this  port bit is set as
                             an input, the PRC3 line is driven low; reading
                             the port bit will give a high. If configured as
                             an output, reading this port bit will not give
                             the-status of the PRC3 line  but the value
                             previously written on the PORT C data reg. bit 3.

PRC46        72        I/O   This is an open drain bi-directional signal with
                             a passive pull-up (1K ohm min). Bit 6 of PORT C
                             is always configured as an input; the bit will
                             give the status of the PRC46 line anytime the
                             the port is read,  regardless of what is written
                             in the data direction register.
                             If bit 4 of PORT C is set as an input, the PRC46
                             line will be pulled low; reading the port bit
                             will give a high.  If bit 4 is configured as
                             an output, PRC46 will be pulled low if bit 4 in
                             the port data register is high, otherwise the
                             PRC46 line will float to a high.

PRC57        73        I/O   This is an open drain bi-directional signal with
                             a passive pull-up (1K ohm min). Bit 7 of PORT C
                             is always configured as an input; the bit will
                             give the status of the PRC57 line anytime the
                             the port is read,  regardless of what is written
                             in the data direction register.
                             If bit 5 of PORT C is set as an input, the PRC57
                             line will be pulled low; reading the port bit
                             will give a high.  If bit 5 is configured as
                             an output, PRC57 will be pulled low if bit 5 in
                             the port data register is high, otherwise the
```

PRC57 line will float to a high.

PRE0,PRE1   83, 84   I/O   This a 2-bit port with each line having a
                             passive pull-up (min. 3K ohm) as well as active
                             pull-up and pull-down transistors. Each indi-
                             vidual port line may be programmed to be eithi.
                             input or output.

BAUDCLK   74   IN   This Input is a 7MHz clock used to drive the
                             UART Baud Rate Generator, and is assumed to
                             be synchronous with the PHO clock. This clock
                             is also divided down to 1MHz to drive the
                             interval timers, and down to 10Hz to drive the
                             TOD timers. This clock is also used to time out
                             the FOR and RESTORE (RSTR*) circuits.

TEST   75   IN   When this input goes to a high state, the device
                             will operate in a test mode. The test mode will
                             allow the BAUDCLK dividers to be initialized and
                             the TOD and interval timers to be driven
                             directly by the BAUDCLK clock, bypassing all the
                             dividers.

TXD   76   OUT   This is the UART transmit data output line.
                             The LSB of the Transmit Data Register is the
                             first data bit transmitted. The data transmission
                             rate (baud rate) is determined by the
                             value written to the Baud Rate Timer latches.

RXD   77   IN   This is the UART receive data input line and
                             is connected to a passive pull-up (1K ohm min)
                             The first data bit received is loaded into the
                             LSB of the Receive Data Register. The receiver
                             data rate must be the same as that determined
                             by the value written to the Baud Rate Timer
                             latches.

NMI/   78   I/O   The NMI/ pin is an open drain bi-directional
                             signal. A passive pull-up (3K ohms minimum) is
                             tied on this pin, allowing multiple NMI/ sources
                             to be tied together. A negative transition on
                             this pin requests a non-maskable interrupt
                             sequence to be generated by the microprocessor.
                             The interrupt sequence will begin with the first
                             PSYNC after a multiple-cycle opcode. NMI/ inputs
                             cannot be masked by the processor status
                             register I flag. The two program counter bytes
                             PCH and PCL, and the processor status register
                             P, are pushed onto the stack. Then the program
                             counter bytes PCL and PCH are loaded from memory
                             addresses FFFA and FFFB/ respectively.

                             NOTE: Since this interrupt is non-maskable,
                             another NMI/ can occur before the first is
                             finished. Care should be taken to avoid this.
                             The NMI/ line is normally off (high impedance)
                             and the device will activate it low as described
                             in the functional description. AEC and DMA/ must
                             be high for any interrupt to be recognized.

IRQ/   79   I/O   The Interrupt Request line (IRQ/) is an open

drain bi-directional signal. A passive pull-up (3K Ohms minimum) is tied on this pin/ allowing multiple IRQ/ sources to be connected together. This pin is sampled during PH2 and when a negative transition is detected an interrupt will be activated, only if the mask flag (I) in the status register is low. The interrupt sequence will begin with the first PSYNC after a multiple-cycle opcode. The two program counter bytes PCH and PCL, and the processor status register P, are stored-onto the stack; the interrupt mask flag is set high so that no. further IRQ/'s may occur. At the end of this cycle, the program counter low byte (PCL) will be loaded from address FFFE/ and the high byte (PCH) from FFFF, thus transferring program control to the vector located at this addresses. The IRQ/ line is normally off (high impedance) and the device will activate it low as described in the functional descriptioni AEC and DMA/ must be high for any interrupt to be recognized.

RESTR/   80   IN   This input is tied to a 3K ohm (min.) passive pull-up. A bounce eliminator circuit is used on this pin to remove any bounce during its falling transition, if the pin is tied to a contact closure. If the device sees a negative transition on this pin, it will immediately

assert the NMI/ line to start a Non-Maskable Interrupt sequence. The device will ignore any subsequent transitions on the RESTR/ line until 4.2ms has elapsed, at which time the NMI/ line is deasserted.

EXTRST/   81   OUT   This output is an open drain output with a min. 1K ohm pull-up. This pin will only go to a low state during power-up, and will stay low until .9 seconds after VDD has reached its operating voltage.

RESET/   82   I/O   The Reset line (RESET/) is an open drain bi-directional signal. A passive pull-up (1K ohm minimum) is tied on this pin, allowing any external source to initialize the device. A low on RESET/ will instantly initialize the internal 65CE02 and all internal registers. All port pins are set as inputs and port registers to zero (a read of the ports will return all highs because of passive pull-ups); all timer control registers are set to zero and all timer latches to ones. All other registers are reset to zero. During power-up RESET/ is held low and will go high .9 seconds after VDD reaches the operating voltage. If pulled low during operation, the currently executing opcode will be terminated. The B and 2 registers will be cleared. The stack pointer will be set to "byte" mode, with the stack page set to page 1. The processor status bits E and I will be set. When the high transition is detected/the reset sequence begins

on the CPU cycle. The first four cycles of the
reset sequence do nothing. Then the program
counter bytes PCL and PCH are loaded from memory
addresses FFFC and FFFD, and normal program
execution begins.

2.3.3.2. 4510R3 Timing Description

```
            +---+                                +---+
-----------|   |--------------------------------|   |---------AEC, DMA
            +---+                                +---+
    TAES---| + |--TAEH            |------ TPWH -----|
-------------+                    +---------------+
             |                    |                |          PH0
             +----------------+                    +----------
             |----- TPWL -----|
                       TAIS--| |--              --|  |--TAIH
----------------+             +--------------------+          NOIO,R/W
          +----------+              VALID         +-------- A0-A19/NOMAP
----------------+             +--------------------+          (INPUT)
          ---|TAOS |---                          ---|  |--TAOH
----------------+      +--------------------------+    +---- PSYNC,R/W
          +---+             VALID                 +---+      A0-A19/NOMAP
----------------+      +--------------------------+    +---- (OUTPUT)
    TDIS|- -+- -|TDIH            |--TDOS--|       |- -|TDOH
         +-------+                        +-----------+
--------+ VALID +----------------------+    VALID   +------ D0-D7
         +-------+                        +-----------+


------+                 +---------
      |                 |              AEC, DMA
      +---------------+
    --| |--TAZ         --| |--TZA
--------+                 +-------
    ON  +---------------+  ON   D0-D7,R/W/A0-A15(AEC, DMA)
--------+                 +-------A16-A19 (DMA)
              |--- TCH ---|
-------+          +-----------+
      |           |           |          C7MHZ
      +---------+               +------
      |---TCL---|          --| |--TCCL
------------------------+
                        |           PH0
              +--------
```
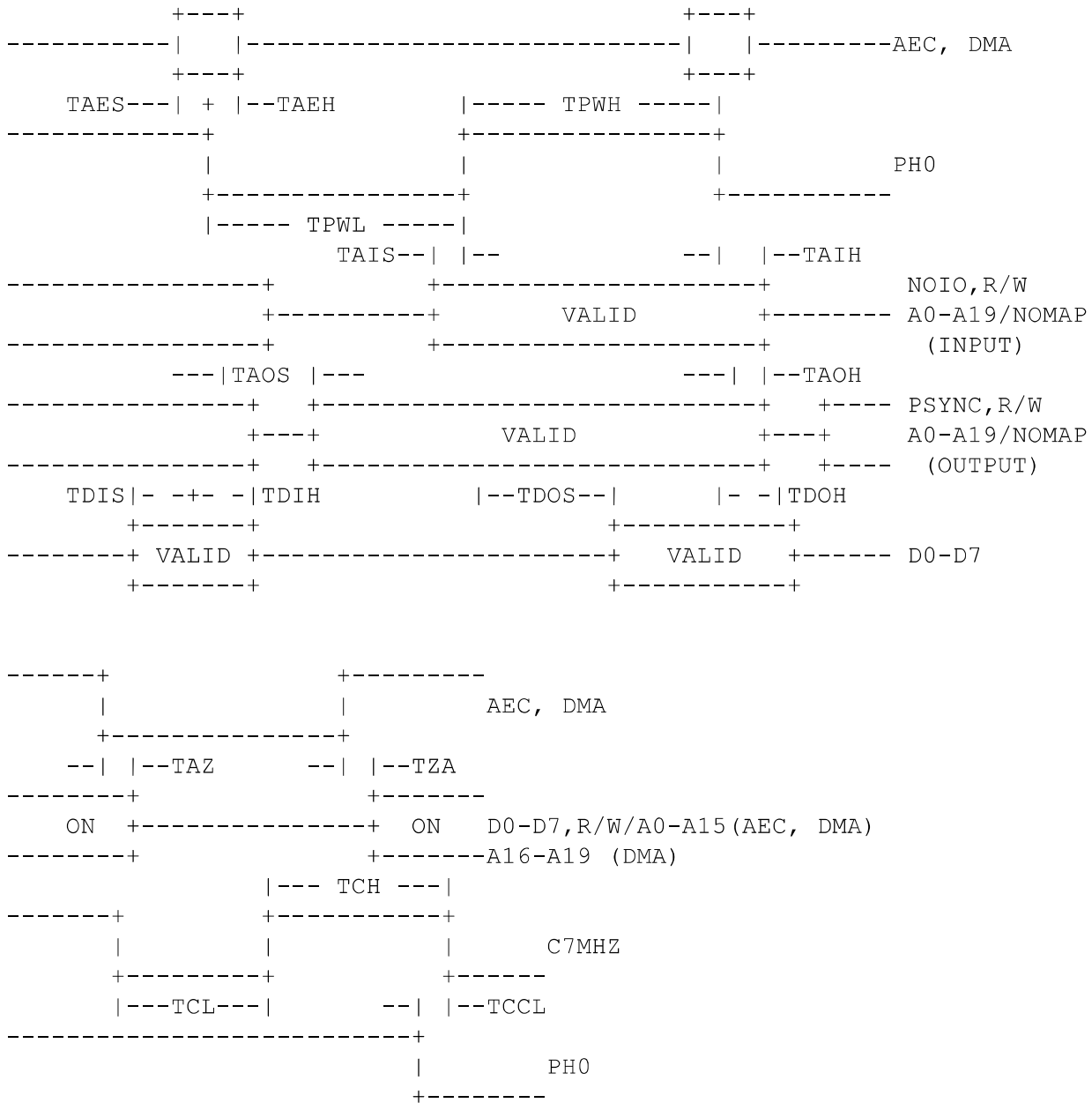
Param	Description	MIN	TYP	MAX
Tpwh	PH0 clock high time	65	135	-
Tpwl	PH0 clock low time	65	135	-
Taes	AEC, DMA setup to PH0 falling	30	-	-
Taeh	AEC, DMA hold from PH0 falling	10	-	-
Tais	address input setup to PH0 rising	20	-	-
Tain	address input hold from PH0 falling	10	-	-
Taos	address output setup from PH0 falling	-	-	50
Taoh	address output hold from PH0 falling	15	-	-
Tdis	data input setup to PH0 falling	40	-	-
Tdih	data input hold from PH0 falling	10	-	-

```
Tdos    data output setup from PH0 rising        -       -       50
Tdoh    data output hold from PH0 falling        30      -       -
Taz     address off from AEC or DMA falling      0       15      20
Tza     address on from AEC and DMA rising       15      -       30
Tch     C7MHZ clock high time                    65      -       -
Tcl     C7MHZ clock low time                     65      -       -
TccL    C7MHZ delay from PH0                      0       -       50
```

2.3.3.3. Register Description

     This device contains a total of 41 I/O peripheral registers which
can be accessed after the following conditions are met. In a an access
cyclethe device must be in a non-mapped mode (MAP/ line is not
asserted), the IO/ line must be in an active low state and the A0-A3,
A8 and A9 address-lines must contain the valid address of the register
to be accessed. In addition the state of the R/W line will indicate
whether a read (R/W is "high") write (R/W is "low") cycle is under
way.

```
  A9 A8...A3 A2 A1 A0   HEX ADD   REG SYMBOL       REGISTER NAME
+--------------------+-------+-----------+---------------------------+
| 0  0   0  0  0  0  | 0X0   |  PRA      |  Peripheral Data  Reg A   |
| 0  0   0  0  0  1  | 0X1   |  PRB      |  Peripheral Data  Reg B   |
| 0  0   0  0  1  0  | 0X2   |  DDRA     |  Data Direction Reg A     |
| 0  0   0  0  1  1  | 0X3   |  DDRB     |  Data Direction Reg B     |
| 0  0   0  1  0  0  | 0X4   |  TA LO    |  Timer A  Low Register    |
| 0  0   0  1  0  1  | 0X5   |  TA HI    |  Timer A  High Register   |
| 0  0   0  1  1  0  | 0X6   |  TB LO    |  Timer B  Low Register    |
| 0  0   0  1  1  1  | 0X7   |  TB HI    |  Timer B  High Register   |
| 0  0   1  0  0  0  | 0X8   |  TODATS   |  TODA 10ths Sec Register  |
| 0  0   1  0  0  1  | 0X9   |  TODAS    |  TODA Seconds Register    |
| 0  0   1  0  1  0  | 0XA   |  TODAM    |  TODA Minutes Register    |
| 0  0   1  0  1  1  | 0XB   |  TODAH    |  TODA Hours-AM/PM Reg.     |
| 0  0   1  1  0  0  | 0XC   |  SDRA     |  SERIALA Data Register    |
| 0  0   1  1  0  1  | 0XD   |  ICRA     |  INTERRUPTA Control Reg.  |
| 0  0   1  1  1  0  | 0XE   |  CRA      |  Control Register A       |
| 0  0   1  1  1  1  | 0XF   |  CRB      |  Control Register B       |
| 0  1   0  0  0  0  | 1X0   |  PRC      |  Peripheral Data Reg. C   |
| 0  1   0  0  0  1  | 1X1   |  PRD      |  Peripheral Data Reg. D   |
| 0  1   0  0  1  0  | 1X2   |  DDRC     |  Data Direction Reg C     |
| 0  1   0  0  1  1  | 1X3   |  DDRD     |  Data Direction Reg D     |
| 0  1   0  1  0  0  | 1X4   |  TC LO    |  Timer C  Low Register    |
| 0  1   0  1  0  1  | 1X5   |  TC HI    |  Timer C  High Register   |
| 0  1   0  1  1  0  | 1X6   |  TD LO    |  Timer D  Low Register    |
| 0  1   0  1  1  1  | 1X7   |  TD HI    |  Timer D  High Register   |
| 0  1   1  0  0  0  | 1X8   |  TODBTS   |  TODB 10ths of Sec Reg.   |
| 0  1   1  0  0  1  | 1X9   |  TODBS    |  TODB Seconds Register    |
| 0  1   1  0  1  0  | 1XA   |  TODBM    |  TODB Minutes Register    |
| 0  1   1  0  1  1  | 1XB   |  TODBH    |  TODB Hours-AM/PM Reg.     |
| 0  1   1  1  0  0  | 1XC   |  SDRB     |  SERIALB Data Register    |
| 0  1   1  1  0  1  | 1XD   |  ICRB     |  INTERRUPTB Control Reg.  |
| 0  1   1  1  1  0  | 1XE   |  CRC      |  Control Register C       |
| 0  1   1  1  1  1  | 1XF   |  CRD      |  Control Register D       |
| 1  0   0  0  0  0  | 2X0   |  DREG     |  Receive/Transmit Data Reg|
| 1  0   0  0  0  1  | 2X1   |  URSR     |  UART Status Register     |
| 1  0   0  0  1  0  | 2X2   |  URCR     |  UART Control Register    |
| 1  0   0  0  1  1  | 2X3   |  BRLO     |  Baud Rate Timer LO Reg.  |
| 1  0   0  1  0  0  | 2X4   |  BRHI     |  Baud Rate Timer HI Reg.  |
| 1  0   0  1  0  1  | 2X5   |  URIEN    |  UART IRQ/NMI Enable Reg. |
| 1  0   0  1  1  0  | 2X6   |  URIFG    |  UART IRQ/NMI Flag Reg.    |
```

REGISTER ADDRESS ALLOCATION
TABLE 1

The functional description of the memory mapper follows in section
2.3.4. The Fast Serial register is described in section 2.3.5.6.

2.3.3.3.1. REGISTER BIT ALLOCATION

R/W	REG	NAME		D7	D6	D5	D4	D3	D2	D1	D0
R/W	0X0	PRA		PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
R/W	0X1	PRB		PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
R/W	0X2	DDRA		DPA7	DPA6	DPA5	DPA4	DPA3	DPA2	DPA1	DPA0
R/W	0X3	DDRB		DPB7	DPB6	DPB5	DPB4	DPB3	DPB2	DPB1	DPB0
READ	0X4	TA LO	T	TAL7	TAL6	TAL5	TAL4	TAL3	TAL2	TAL1	TAL0
READ	0X5	TA HI	I	TAH7	TAH6	TAH5	TAH4	TAH3	TAH2	TAH1	TAH0
READ	0X6	TB LO	M E	TBL7	TBL6	TBL5	TBL4	TBL3	TBL2	TBL1	TBL0
READ	0X7	TB HI	R	TBH7	TBH6	TBH5	TBH4	TBH3	TBH2	TBH1	TBH0
WRITE	0X4	TA LO	P R	PAL7	PAL6	PAL5	PAL4	PAL3	PAL2	PALI	PAL0
WRITE	0X5	TA HI	E S	PAH7	PAH6	PAH5	PAH4	PAH3	PAH2	PAH1	PAHO
WRITE	0X6	TB LO	C A	PBL7	PBL5	PBL5	PBL4	P3L3	PBL2	PBL1	PBL0
WRITE	0X7	TB HI	L E R	PBH7	PBH6	PBH5	PBH4	PBH3	PBH2	PBH1	PBH0
READ	0X8	TODATS	T O D	0	0	0	0	TA8	TA4	TA2	TA1
READ	0X9	TODAS		(*) 0	SAH4	SAH2	SAH1	SAL8	SAL4	SAL2	SAL1
READ	0XA	TODAM	T I M	(*) 0	MAH4	MAH2	MAH1	MAL8	MAL4	MAL2	MAL1
READ	0XB	TODAH	E	APM	0	0	HAH	HAL8	HAL4	HAL2	HAL1
				(*) IN TEST MODE: WILL READ DIVIDER STAGE OUTPUTS							
WRITE	0X8	TODATS	T O	0	0	0	0	TA8	TA4	TA2	TA1
WRITE	0X9	TODAS		0	SAH4	SAH2	SAH1	SAL8	SAL4	SAL2	SAL1
WRITE	0XA	TODAM	L A T	0	MAH4	MAH2	MAH1	MAL8	MAL4	MAL2	MAL1
WRITE	0XB	TODAH	C	APM	0	0	HAH	HAL8	HAL4	HAL2	HAL1

```
|     |   |          | H |                                                                  |
|     |   |          | E |   IF CRB ALARM BIT=1 , ALARM REGISTER IS WRITTEN                  |
|     |   |          | S |   IF CRB ALARM BIT=0 , TOD REGISTER IS WRITTEN                    |
+-----+---+----------+-----+------+------+------+------+------+------+------+------+
| R/W |0XC|   SDRA   | SRA7 | SRA6 | SRA5 | SRA4 | SRA3 | SRA2 | SRA1 | SRA0 |
+-----+---+----------+-----+------+------+------+------+------+------+------+------+
| READ|0XD|   ICRA   | IRA  |  0   |  0   | FLGA | SPA  | ALRMA|  TB  |  TA  |
|     |   |(INT DATA)|      |      |      |      |      |      |      |      |
+-----+---+----------+-----+------+------+------+------+------+------+------+------+
|WRITE|0XD|   ICRA   |AS/C~ |  --  |  --  | FLGA | SPA  | ALRMA|  TB  |  TA  |
|     |   |(INT MASK)|      |      |      |      |      |      |      |      |
+-----+---+----------+-----+------+------+------+------+------+------+------+------+
| R/W |0XE|   CRA    | TODA | SPA  | TMRA |LOADA | RUN-A| OUT-A| PRB6 |STARTA|
|     |   |          |  IN  | MODE |INMODE|      | MODE | MODE | ON   |      |
+-----+---+----------+-----+------+------+------+------+------+------+------+------+
| R/W |0XF|   CRB    |ALARM |TIMERB|INMODE|LOADB | RUN-B| OUT-B| PRB7 |STARTB|
|     |   |          |(TODA)| CRB6 | CRB5 |      | MODE | MODE | ON   |      |
+-----+---+----------+-----+------+------+------+------+------+------+------+------+
| READ|1X0|   PRC    | PC7  | PC6  | PC5  | PC4  | PC3  | PC2  | PC1  | PC0  |
+-----+---+----------+-----+------+------+------+------+------+------+------+------+
| R/W |1X1|   PRD    | PD7  | PD6  | PD5  | PD4  | PD3  | PD2  | PD1  | PD0  |
+-----+---+----------+-----+------+------+------+------+------+------+------+------+
| R/W |1X2|   DDRC   | DPC7 | DPC6 | DPC5 | DPC4 | DPC3 | DPC2 | DPC1 | DPC0 |
+-----+---+----------+-----+------+------+------+------+------+------+------+------+
| R/W |1X3|   DDRD   | DPD7 | DPD6 | DPD5 | DPD4 | DPD3 | DPD2 | DPD1 | DPD0 |
+-----+---+------+---+-----+------+------+------+------+------+------+------+------+
| READ|1X4| TC LO|   | TCL7 | TCL6 | TCL5 | TCL4 | TCL3 | TCL2 | TCL1 | TCL0 |
+-----+---+------+ T +------+------+------+------+------+------+------+------+
| READ|1X5| TC HI| I | TCH7 | TCH6 | TCH5 | TCH4 | TCH3 | TCH2 | TCH1 | TCH0 |
+-----+---+------| M +------+------+------+------+------+------+------+------+
| READ|1X6| TD LO| E | TDL7 | TDL6 | TDL5 | TDL4 | TDL3 | TDL2 | TDL1 | TDL0 |
+-----+---+------| R +------+------+------+------+------+------+------+------+
| READ|1X7| TD HI|   | TDH7 | TDH6 | TDH5 | TDH4 | TDH3 | TDH2 | TDH1 | TDH0 |
+-----+---+------+---+------+------+------+------+------+------+------+------+
|     |   |      | P |      |      |      |      |      |      |      |      |
|WRITE|1X4| TC LO| R | PCL7 | PCL6 | PCL5 | PCL4 | PCL3 | PCL2 | PCL1 | PCL0 |
+-----+---+------+ E +------+------+------+------+------+------+------+------+
|WRITE|1X5| TC HI| S | PCH7 | PCH6 | PCH5 | PCH4 | PCH3 | PCH2 | PCH1 | PCH0 |
+-----+---+------+ C +------+------+------+------+------+------+------+------+
|WRITE|1X6| TD LO| A | PDL7 | PDL6 | PDL5 | PDL4 | PDL3 | PDL2 | PDL1 | PDL0 |
+-----+---+------+ L +------+------+------+------+------+------+------+------+
|WRITE|1X7| TD HI| E | PDH7 | PDH6 | PDH5 | PDH4 | PDH3 | PDH2 | PDH1 | PDH0 |
|     |   |      | R |      |      |      |      |      |      |      |      |
+-----+---+------+---+------+------+------+------+------+------+------+------+
|     |   |      | T |      |      |      |      |      |      |      |      |
|READ |1X8|TODBTS| O |  0   |  0   |  0   |  0   | TB8  | TB4  | TB2  | TB1  |
+-----+---+------+ D +------+------+------+------+------+------+------+------+
|READ |1X9|TODBS |   |(*) 0 | SBH4 | SBH2 | SBH1 | SBL8 | SBL4 | SBL2 | SBL1 |
+-----+---+------+ T +------+------+------+------+------+------+------+------+
|READ |1XA|TODBM | I |  0   | MBH4 | MBH2 | MBH1 | MBL8 | MBL4 | MBL2 | MBL1 |
+-----+---+------+ M +------+------+------+------+------+------+------+------+
|READ |1XB|TODBH | E | BPM  |  0   |  0   | HBH  | HBL8 | HBL4 | HBL2 | HBL1 |
|     |   |      | R |      |      |      |      |      |      |      |      |
|     |   |      |   |  (*)  IN TEST MODE: WILL READ DIVIDER STAGE OUTPUT        |
+-----+---+------+---+------+------+------+------+------+------+------+------+
|WRITE|1X8|TODBTS| T |  0   |  0   |  0   |  0   | TB8  | TB4  | TB2  | TB1  |
+-----+---+------+ O +------+------+------+------+------+------+------+------+
|WRITE|1X9|TODBS | D |  0   | SBH4 | SBH2 | SBH1 | SBL8 | SBL4 | SBL2 | SBL1 |
+-----+---+------+ L +------+------+------+------+------+------+------+------+
|WRITE|1XA|TODBM | A |  0   | MBH4 | MBH2 | MBH1 | MBL8 | MBL4 | MBL2 | MBL1 |
+-----+---+------+ T +------+------+------+------+------+------+------+------+
```

WRITE	1XB	TODBH	CHES	BPM	0	0	HBH	HBL8	HBL4	HBL2	HBL1

IF CRD ALARM BIT=1 , ALARM REGISTER IS WRITTEN
IF CRD ALARM BIT=0 , TOD REGISTER IS WRITTEN

R/W	1XC	SDRB	SRB7	SRB6	SRB5	SRB4	SRB3	SRB2	SRB1	SRB0
READ	1XD	ICRB (INT DATA)	IRB	0	0	FLGB	SPB	ALRMB	TD	TC
WRITE	1XD	ICRB (INT MASK)	BS/C~	--	--	FLGB	SPB	ALRMB	TD	TC
R/W	1XE	CRC	TODB IN	SPB MODE	TMRC INMODE	LOADC	RUN-C MODE	OUT-C MODE	PRD6 ON	STARTC
R/W	1XF	CRD	ALARM (TODB)	TIMERD CRD6	INMODE CRD5	LOADD	RUN-D MODE	OUT-D MODE	PRD7 ON	STARTD
READ	2X0	DREG (RECEIVE DATA REG)	RCV7	RCV6	RCV5	RCV4	RCV3	RCV2	RCV1	RCV0
WRITE	2X0	DREG (TRANSMIT DATA REG)	XMT7	XMT6	XMT5	XMT4	XMT3	XMT2	XMT1	XMT0
READ	2X1	URSR	TDONE	EMPTY	ENDT	IDLE	FRME	PRTY	OVR	FULL
WRITE	2X1	URSR	--	--	ENDT	IDLE	--	--	--	--
R/W	2X2	URCR	XMITR EN	RCVER EN	UART UM1	MODE UM0	CHAR LENGTH CH1	CH0	PARITY EN	PARITY EVEN
R/W	2X3	BRL0	BRL7	BRL6	BRL5	BRL4	BRL3	BRL2	BRL1	BRL0
R/W	2X4	BRHI	BRH7	BRH6	BRH5	BRH4	BRH3	BRH2	BRH1	BRH0
R/W	2X5	URIEN	XDIRQ	RDIRQ	XDNMI	RDNMI	--	--	--	--
READ	2X6	URIFG	XDIRQ	RDIRQ	XDNMI	RDNMI	--	--	--	--
R/W	2X7	PRE	--	--	--	--	--	--	PE1	PE0
R/W	2X8	DDRE	--	--	--	--	--	--	DPE1	DPE0
R/W	2X9	FSERIAL	*DMODE	*FSDIR	--	--	--	--	--	--

REGISTER BIT ALLOCATION
TABLE 2

2.3.4. Memory Mapper

The microprocessor core is actually a C4502R1 with some addittional instructions, used to operate the memory mapper.

The former AUG (augment) opcode has been changed to MAP (mapper), and the former NOP (no-operation) has been changed to EOM (end-of-mapping-sequence).

The 4510 memory mapper allows the microprocessor to access up to

1 megabyte of memory. Here's how. The 6502 microprocessor can only
access 64K bytes of memory because it only uses addresses of 16 bit's.
The 4502 is not different, nor is the 4510. But the 4510 memory mapper
allows these addresses to be redirected to new physical addresses
to access different parts of a much larger memory, within the 64K byte
confinement window.

   The 64K window has been divided into eight blocks, and two
regions, with four blocks in each region. Blocks 0 through 3 are in
the "lower" region, and blocks 4 through 7 are in the "upper" region,
as shown...

```
                  +-      +----------+FFFF
                  |       |  BLK 7   |
                  |       +----------+E000
                  |       |  BLK 6   |
 UPPER REGION -+          +----------+C000
                  |       |  BLK 5   |
                  |       +----------+A000
                  |       |  BLK 4   |
                  +-      +----------+8000
                  |       |  BLK 3   |
                  |       +----------+6000
                  |       |  BLK 2   |
 LOWER REGION -+          +----------+4000
                  |       |  BLK 1   |
                  |       +----------+2000
                  |       |  BLK 0   |
                  +-      +----------+
```
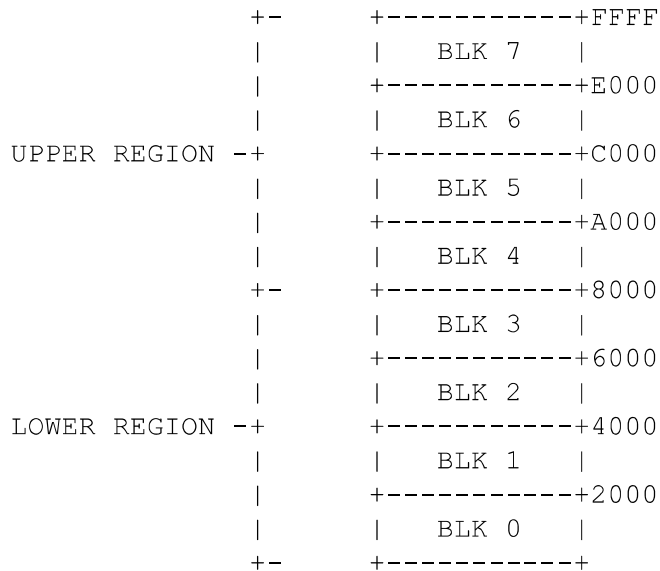
   Each block can be programmed to be "mapped", or "non-mapped" via
bits in the mapper's "mask" registers. NON-MAPPED means, simply,
address out equals address in. Therefore, there are still only 64K
bytes of non-mapped memory. MAPPED means that address out equals
address in plus some offset. The offset is programmed via the mapper's
"offset" registers.

There are two "offset" registers. One is for the lower region, and one
is for the upper region.

   The low-order 6 addresses are never mapped. The offsets are only
added to the 12 high-order addresses. This means the smallest unit you
can map to is 256 bytes, or one page.

   The 4510 has an output (NOMAP) which lets the outside world know
when the processor is accessing mapped (0) or non-mapped (1) address.
This is useful for systems where you may want I/O devices to be at
fixed (non-mapped) addresses, and only memory at mapped addresses.

   It is possible, and likely, to have mapped, and unmapped memory
at the same physical address. And, with offset registers set to zero,
mapped addresses will match unmapped ones. The only difference is the
NOMAP signal to tell whether the address is mapped or unmapped.

   To program the mapper, the operating system must load the A, X,
Y, and Z registers with the following information, and execute a MAP
opcode.

                        Mapper Register Data

     7   6   5   4   3   2   1   0   BIT

```
+-------+-------+-------+-------+-------+-------+-------+-------+
| LOWER | LOWER | LOWER | LOWER | LOWER | LOWER | LOWER | LOWER | A
| OFF15 | OFF14 | OFF13 | OFF12 | OFF11 | OFF10 | OFF9  | OFF8  |
+-------+-------+-------+-------+-------+-------+-------+-------+
| MAP   | MAP   | MAP   | MAP   | LOWER | LOWER | LOWER | LOWER | X
| BLK3  | BLK2  | BLK1  | BLK0  | OFF19 | OFF18 | OFF17 | OFF16 |
+-------+-------+-------+-------+-------+-------+-------+-------+
| UPPER | UPPER | UPPER | UPPER | UPPER | UPPER | UPPER | UPPER | Y
| OFF15 | OFF14 | OFF13 | OFF12 | OFF11 | OFF10 | OFF9  | OFF8  |
+-------+-------+-------+-------+-------+-------+-------+-------+
| MAP   | MAP   | MAP   | MAP   | UPPER | UPPER | UPPER | UPPER | Z
| BLK7  | BLK6  | BLK5  | BLK4  | OFF19 | OFF18 | OFF17 | OFF16 |
+-------+-------+-------+-------+-------+-------+-------+-------+
```

After executing the MAP opcode, all interrupts are inhibited. This is done to allow the operating system is complete a mapping sequence without fear of getting an interrupt. An interrupt occurring before the proper stack-pointer is set will cause return address data to be written to an undesired area.

Upon completing the mapping sequence, the operating system must remove the interrupt inhibit by executing a EOM (formerly NOP) opcode. Note that application software may execute NOPs with no effect.

2.3.5. Peripheral Control Functions

2.3.5.1. I/O Ports

Ports A, B and D each consist of an 8-bit Peripheral Data Register (PR) and an 8-bit Data Direction Register (DDR). Port E consists of a 2-bit PR and DDR registers. If a bit in the DDR is set to one, the corresponding bit in the PR is an output, if a DDR bit is set to a zero, the corresponding PR bit is defined as an input. On a READ, the PR bit reflects the information present on the actual port pins (PRA0-PRA7, PRB0-PRB7, PRC2, PRD0-PRD7, PRE0-PRE1) for both input and output bits. All ports have passive pull-up devices as well as active pull-ups, providing both CMOS and TTL compatibility.

In addition to normal I/O operation, PRB6, PRB7, PRD6 and PRD7 also provide timer output functions (refer to Control Register section, 2.5.8.).

Only bit PC2 and DPC2 of PORT C meet the above description. The other bits function as described in the following.

PC0,PC1   These signals are simply a register bits. When read, they will reflect the value previously written to the PRC register.

PC4   This bit is a "high" if it's configured as input (DPC4 is a "low"). If configured as output (DPC4 is a "high"), the bit will reflect its previous written value when PORT C is read. Then the PRC46 pin is pulled "low" if PC4 is "high"; otherwise, PRC46 is pulled-up through passive resistor.

PC5   This bit is a "high" if it's configured as input (DPC5 is a "low"). If configured as output (DPC5 is a "high"), the bit will reflect its previous written value when PORT C is read. Then the PRC57 pin is pulled "low" if PC5 is "high";

otherwise, PRC57 is pulled-up through passive resistor.

PC6,PC7   These bits are always configured as inputs. When PORT C
           (PRC) is read, PC6 and PC7 will reflect the values on the
           PRC46 and PRC57 pins, respectively.

2.3.5.2. Handshaking

     Handshaking on data transfers can be accomplished using the PC/
output pin and either the FLAGA/ or FLAGB/ input pin. The PC/ line
will go low and stay low for two cycles, two cycles after a read or
write to PORT D. This is required to meet Centronics Parallel
Interface specs. The PC/ line can be used to indicate "data ready" at
PORT D or "data accepted" from PORT D. Handshaking on 16-bit data
transfers (using either PORT A or B and then PORT D) is possible by
always reading or writing PORT A or PORT B first. The FLAG/ lines are
negative edge sensitive inputs which can be used for receiving the PC/
output from other 4510 devices, or as general purpose interrupt
inputs. A negative transition on FLAGA/ or FLAGB/ will set the FLAGA
or FLAGB interrupt bits, respectively.

2.3.5.3. Interval Timers (Timer A, Timer B, Timer C, Timer D)

     Each interval timer consists of a 16-bit read-only Timer Counter
and a 16-bit write-only Timer Latch (prescaler). Data written to the
timer are latched in the Timer Latch, while data read from the timer
are the present contents of the Timer Counter. The timers can be used
independently or linked in pairs for extended operations (TIMER A may
be linked with Timer B; TIMER C may be linked with TIMER D). The
various timer modes allow generation of long time delays, variable
width pulses, pulse trains and variable frequency waveforms. Utilizing
the CNT inputs, the timers can count external pulses or measure
frequency, pulse widtdh and delay times of external signals. Each
timer has an associated control register, providing independent
control of the following functions (see bits functional description in
section 2.5.8 below):

Start/Stop

     Each timer may be started or stopped by the microprocessor at
any time by writing to the START/STOP bit of the corresponding control
register (CRA, CRB, CRB or CRC).

PRB, PRD On/Off

     Control bits allow any of the timer outputs to appear on a PORT
B or PORT D output line (PRB6 for TIMER A, PRB7 for TIMER B, PRD6 for
TIMER C and PRD7 for TIMER D). Note that this function overrides the
DDRB control bit and forces the appropriate PB or PC line to be an
output.

Toggle/Pulse

     Control bits select the ouputs applied to PORT B and PORT D. On
every timer underflow the ouput can either toggle or generate a single
positive pulse of one cycle duration. The Toggle output is set high
whenever the appreciate timer is started and is set low by RESET/.

One-Shot/Continuous

Control bits select-either timer mode. In one-shot mode, the
timer will count down from the latched value to zero, generate an
interrupt, reload the latched value, then stop. In continuous mode,
the timer will count from the latched value to zero, generate an
interrupt, reload the latched value and repeat the procedure
continuously.

Force Load

A strobe bit allows the timer latch to be loaded into the timer
counter at any time, whether the timer is running or not.

Input Mode

Control bits allow selection of the clock used to decrement the
timer. TIMER A or TIMER C can count C1MHZ clock pulses or external
pulses applied to the CNTA or CNTB, respectively. The C1MHZ clock is
obtained after internally dividing the C7MHZ by a factor of seven.

TIMER B can count C1MHZ clock pulses, external pulses applied to
the CNTA input, TIMER A underflow pulses or TIMER A underflow pulses
while the CNTA pin is held high.

TIMER D can count C1MHZ clock pulses, external pulses applied to
the CNTB input, TIMER C underflow pulses or TIMER C underflow pulses
while the CNTB pin is held high.

The timer latch is loaded into the timer on any timer underflow,
on a force load or following a write to the high byte of the prescaler
while the timer is stopped. If the timer is running, a write to the
high byte will load the timer latch, but not reload the counter.

2.3.5.4. Time of Day Clocks (TODA, TODB)

The TODA and TODB clocks are special purpose timers for
real-time applications. Each clock, TODA or TODB, consists of a
24-hour (AM/PM) clock with 1/10th second resolution. Each is organized
into four registers: 10ths of seconds (TODATS, TODBTS), Seconds
(TODAS, TODBS)/Minutes (TODAM, TODBM) and Hours (TODAH, TODBH). The
AM/PM flag is in the MSB of the Hours register for easy testing. Each
register reads out in BCD format to simplify conversion for driving
displays, etc. Each TOD requires a 10HZ clock input to keep accurate
timing. This 10HZ clock is generated by dividing the C7MHz clock input
by a factor of 102273 for NTSC (60Hz) applications, or a factor of
101339 for PAL (50Hz) applications. The divider ratio is selected by
the TODA IN and the TODB IN bits of the Control Registers, CRA and
CRC, respectively (see 2.5.8).

In addition to time-keeping, a programmable ALARM is provided
for generating an interrupt at the desired time, from either of the
TOD clocks. The ALARM registers registers are located at the same
addresses as the corresponding TODA and TODB registers. Access to the
ALARM is governed by bit 7 in the Control Registers CRB and CRD. The
ALARM registers are write-only; any read of a TOD address will read
time regardless of the state of the ALARM access control bits.

A specific sequence of events must be followed for proper
setting and reading of each TOD. A TOD is automatically stopped
whenever a write to the corresponding Hours register occurs. The TOD

will not start again until after a write to the proper 10ths of
seconds register. This assures that a TOD will always start at the
desired time. Since a carry from one stage to the next can occur at
any time with respect to a read operation, a latching function is
included to keep all Time of Day information constant during a read
sequence. All four registers of each TOD latch on a read of the
corresponding Hours register and remain latched until after a read of
the corresponding 10ths of second register. A TOD continues to count
when the output registers are latched. If only one register is to be
read, there is no carry problem and the register can be read "on the
fly", provided that any read of the Hours register if followed by a
read of the proper 10ths of seconds, to disable the latching.

2.3.5.5. Serial Ports (SDRA, SDRB)

        Each serial port is a buffered, 8-bit synchronous shift register
system. A control bit (CRA SPA bit, CRC SPB bit) selects input or
output mode for either the SDRA or SDRB port.

        In input mode, data on the SPA or SPB pin is shifted into the
corresponding shift register on the rising edge of the signal applied
to the CNTA or CNTB pin, respectively. After 8 CNTA pulses, the data
in the shift register is dumped into the SERIALA Data Register (SDRA)
and an interrupt is generated, SPA bit is set in register ICRA. After
8 CNTB pulses, the data in the shift register is dumped into the
SERIALB Data Register (SDRB) and an interrupt is generated, SPB bit is
set in register ICRB.

        In the output mode, TIMER A is used for the baud rate generator
of serial port A, Timer C for serial port B. Data is shifted on an SP
pin at half the underflow rate of the TIMER used. The maximum baud
rate possible is C1MHz divided by four, but the maximum useable baud
rate will be determined by line loading and the speed at which the
receiver responds to input data. Transmission will start following a
write to Serial Data Register (provided the proper TIMER used is
running and in continuous mode). The clock signal derived from TIMER A
would appear as an output on the CNTA pin; the one from TIMER C would
appear on the CNTB pin. The data in the Serial Data Register will be
loaded into its corresponding shift register then shift out to the SPA
or SPB pin when a CNTA or CNTB pulse occurs, respectively.

        Data shifted out becomes valid on the falling edge of its CNT
clock and remains valid until the next falling edge. After 8 CNT
pulses, an interrupt is generated to indicate more data can be sent.
If the Serial Data Register was loaded with new information prior to
this interrupt, the new data will automatically be loaded into the
shift register and transmission will continue. If the microprocessor
stays one byte ahead of the shift register, transmission will be
continuous. If no further data is to be transmitted, after the 8th CNT
pulse, CNT will return high and SP will remain at the level of the
last data bit transmitted. SDR data is shifted out MSB first and
serial input data should also appear on this format.

        The bidirectional capability of each of the Serial Ports and CNT
clocks allows many 4510 to be connected to a common serial
communication bus on which one Serial Port would act as a master,
sourcing data and shift clock, while the other Serial Port (and all
other ports from other 4510 devices) would act as slaves. All the CNT
and SP outputs are open drain to allow such a common bus. Protocol for
master/slave selection can be transmitted over the serial bus, or via

dedicated handshaking lines.

2.3.5.6. FAST SERIAL MODE

     The FAST SERIAL logic consists of a 2-bit write-only register,
which resides in location 0001 (hex). This register may only be
accessed by the CPU if neither the AEC or DMA/ line is low. Upon
reset, both bits in the register are forced low which allows the
device to operate as normal (the CNTA, SPA, PRC57 and FLAGA/ lines
will not be affected).

     Bit 1 of the FAST SERIAL register is the Fast Serial Mode
disable bit (DMODE* bit).

     Bit 6 of the FAST SERIAL register is the FSDIR* bit. When the
DMODE* bit is set high, the FSDIR* bit will be used as an output to
control the fast serial data direction buffer hardware, and as an
input to sense a fast disk enable signal. This function will affect
the CNTA, SPA, PRC57 and FLAGA/ lines as summarized in the following
table.

 DMODE* FSDIR*   FUNCTION

   0   0   Fast Serial mode is disabled.

   x   1   Both the FLAGA/ and the PRC57 lines will behave as
                   outputs. The FLAGA/ output will reflect the state of
                   the CNTA pin, whereas the PRC57 output will reflect
                   the state of the SPA pin.

   1   0   Both the CNTA and SPA lines will behave as outputs.
                   The CNTA output will reflect the state of the FLAGA/
                   pin, whereas the SPA output will reflect that of the
                   PRC57 pin.

2.3.5.7. Interrupt Control Registers (ICRA, ICRB)

These registers control the following sources of interrupts:

  i. Underflows from TIMER A, TIMER B, TIMER C and TIMER D.
 ii. TODA ALARM and TODB ALARM.
iii. SERIALA and SERIALB Port full/empty conditions.
 iv. FLAGA/ and FLAGB/ low transitions.

     The ICRA and ICRB registers each provides masking and interrupt
information. ICRA and ICRB each consists of a write-only MASK register
and   a   read-only-DATA   register.   Any   interrupt will set the
corresponding bit in the DATA register. Any interrupt which is enabled
by the MASK register will set the IR bit (MSB) of its corresponding
DATA register and bring the IRQ/ pin low. In a multi-chip system, the
IR bit (IRA of ICRA or IRB of ICRB) can be polled to detect which chip
has generated an interrupt request. The interrupt DATA register is
cleared and the IRQ/ line returns high following a read of the DATA
register. Since each interrupt sets and interrupt bit regardless of
the MASK, and each interrupt bit can be selectively masked to
prevent the generation of a processor interrupt, it is possible to
intermix polled interrupts with true interrupts. However, polling
either of the IR bits will cause its corresponding DATA register to
clear, therefore, it is up to the user to preserve the information

contained in the DATA registers if any polled interrupts were present.

Both MASK (ICRA, ICRB) registers provide convenient control of individual mask bits. When writing to a MASK register, if bit 7 of the data written (corresponding to AS/C in ICRA, or BS/C in ICRB) is a ZERO, any mask bit written with a one will be cleared, while those bits written with a zero will be unaffected. In order for an interrupt flag to set the IR bit and generate an Interrupt Request, the corresponding MASK bit must be set in the corresponding MASK Register.

2.3.5.8. Control Registers (CRA, CRB, CRC, CRD)

CRA (0XE):

BIT	Bit Name	Function
0	STARTA	1=START TIMER A, 0=STOP TIMER A. This bit is automatically reset when TIMER A underflow occurs during one-shot mode.
1	PRB6 ON	1=TIMER A output appears on PRB6, 0=PRB6 normal port operation.
2	OUT-A MODE	1=TOGGLE output applied on port PRB6, 0=PULSE output applied on port PRB6.
3	RUN-A MODE	1=ONE-SHOT TIMER A operation, 0=CONTINUOUS TIMER A operation.
4	LOADA	1=FORCE LOAD on TIMER A (this is a STROBE input, there is no data storage, bit 4 will always read back a zero and writing a zero has no effect).
5	TMRA INMODE	1=TIMER A counts positive CNTA transitions, 0=TIMER A counts internal C1MHZ pulses.
6	SPA MODE	1=SERIAL A PORT output mode (CNTA sources shift clock), 0=SERIAL A PORT input mode (external shift clock on CNTA).
7	TODA IN	1=50 Hz operation. C7MHZ divided down by 101339 to generate TODA input of 10 Hz. 0=60 Hz operation. C7MHZ divided down by 102273 to generate TODA input of 10 Hz.

CRB (0XF):

BIT	Bit Name	Function

(Bits 0-4 of the CRB register operate identically to bits 0-4 of the CR7 register, except that functions now apply to TIMER B and bit 1 control the output of TIMER B on PRB7).

| 5,6 | TIMERB INMODE | Bits 5 and 6 select one of four input modes for TIMER B as follows: |

CRB6 CRB5

CRB6	CRB5	
0	0	TIMER B counts C1MHz pulses.
0	1	TIMER B counts positive CNTA transitions.

```
                   1    0      TIMER B counts TIMERA underflow pulses.
                   1    1      TIMER B counts TIMERA underflows while
                               CNTA is high.

   7     ALARM TODA      1=writing to TODA registers sets ALARM,
                         0=writing to TODA registers sets TODA clock.


CRC (1XE):
BIT   Bit Name          Function


   0     STARTC          1=START TIMER C, 0=STOP TIMER C. This bit is
                         automatically reset when TIMER C underflow occurs
                         during one-shot mode.


   1     PRD6 ON         1=TIMER C output appears on PRD6, 0=PRD6 normal port
                         operation.


   2     OUT-C MODE      1=TOGGLE output applied on port PRD6,
                         0=PULSE output applied on port PRD6.


   3     RUN-C MODE      1=ONE-SHOT TIMER C operation,
                         0=CONTINUOUS TIMER C operation.


   4     LOADC           1=FORCE LOAD on TIMER C (this is a STROBE input,
                         there is no data storage, bit 4 will always read
                         back a zero and writing a zero has no effect).


   5     TMRC INMODE     1=TIMER C counts positive CNTB transitions,
                         0=TIMER C counts internal C1MHZ pulses.


   6     SPB MODE        1=SERIAL B PORT  output mode (CNTB sources shift
                         clock),
                         0=SERIAL B PORT  input mode (external shift clock
                         on CNTB).


   7     TODB IN         1=50 Hz operation.  C7MHZ divided down by 101339 to
                         generate TODB input of 10 Hz.
                         0=60 Hz operation.  C7MHZ divided down by 102273 to
                         generate TODB input of 10 Hz.


CRD (1XF):
BIT    Bit Name          Function


(Bits  0-4  of the CRD register operate identically to bits 0-4 of the
CRD  register,  except  that  functions now apply to TIMER D and bit 1
controls the output of TIMER D on PRD7).


5,6   TIMERD           Bits 5 and 6 select one of four input modes for
      INMODE           TIMER D as follows:


                       CRD6 CRD5


                   0    0      TIMER D counts C1MHz pulses.
                   0    1      TIMER D counts positive CNTB transitions.
                   1    0      TIMER D counts TIMERC underflow pulses.
                   1    1      TIMER D counts TIMERC underflows while
                               CNTB is high.


   7     ALARM TODB      1=writing to TODB registers sets ALARM,
```

0=writing to TODB registers sets TODA clock.

                    C65 Peripheral Control Utilization

6526 cia complex interface adapter #1

keyboard / joystick / paddles / mouse / lightpen / fast serial

pra0 keybd output c0 / joystick #1 up   / mouse right button
pra1 keybd output c1 / joystick #1 down
pra2 keybd output c2 / joystick #1 left / paddle "A" fire button
pra3 keybd output c3 / joystick #1 right / paddle "B" fire button
pra4 keybd output c4 / joystick #1 fire / mouse left button
pra5 keybd output c5 /
pra6 keybd output c6 /   / select port #1 paddles|mouse
pra7 keybd output c7 /   / select port #2 paddles|mouse

prb0 keybd input r0 / joystick #2 up   / mouse right button
prb1 keybd input r1 / joystick #2 down / paddle "A" fire button
prb2 keybd input r2 / joystick #2 left / paddle "B" fire button
prb3 keybd input r3 / joystick #2 right
prb4 keybd input r4 / joystick #2 fire / mouse left button
prb5 keybd input r5 /
prb6 keybd input r6 / timer b: toggle/pulse output
prb7 keybd input r7 / timer a: toggle/pulse output

timer 1 & cra : fast serial
timer 2 & crb :

tod   :
sdr   :
icr   :

6526 cia complex interface adapter #2

user port / rs232 / serial bus / VCC bank / NMI

pra0 va14   VIC   16K bank select
pra1 va15
pra2 rs232 DATA   output   (C64 mode only)
pra3 serial ATN   output
pra4 serial CLK   output
pra5 serial DATA   output
pra6 serial CLK   input
pra7 serial DATA   input

prb0 user port / rs232 received data   (C64 mode only)
prb1 user port / rs232 request to send
prb2 user port / rs232 data terminal ready
prb3 user port / rs232 ring indicator
prb4 user port / rs232 carrier detect
prb5 user port
prb6 user port / rs232 clear to send
prb7 user port / rs232 data set ready

timer 1 & cra :   rs232 baud rate   (C64 mode only)
timer 2 & crb :   rs232 bit check   (C64 mode only)

tod   :

```
sdr   :
icr   : nmi (/irq)
```

2.3.6. UART Operation

The device contains seven registers to control the different UART modes of operation. Section 2.2 describes how to access these registers.

The UART modes can be programmed by accessing the UART control register, URCR, whose bits function as described below.

2.3.6.1. UART Control Register (DRCR)

BIT   Bit Name   Function

  0   PARITY EVEN   1 = Even Parity. If parity is   enabled,   the
                       transmitter will assert the parity bit (P) to a low
                       when "even" parity data is transmitted, otherwise
                       it will pull it high. The receiver checks that the
                       parity bit is asserted, or low, if the data
                       received has even parity; if the bit is not
                       asserted, the device will indicate a parity error.

                       0=Odd Parity. If parity is enabled, the transmitter
                       will pull the parity bit (P) low when "odd" parity
                       data is transmitted, otherwise it will pull it
                       high. The receiver checks that the parity bit is
                       asserted if the data received has odd parity; if
                       the bit is not asserted when data had odd parity,
                       the device will indicate a parity error.

  1   PARITY EN   1 = Parity Enabled.

                       0 = Parity Disabled. The transmitter and receiver
                       will not allocate a parity bit in the data, instead
                       a stop bit will be used in its place. See the Data
                       Configuration chart below.

2,3   CHAR LENGTH   These two bits are used to select the number of
                       bits per character to be transmitted or received.
                       5, 6, 7 or 8 bits per character may be selected as
                       follows:

                               CH1   CH0
                               ---   ---
                                0   0   eight bits per character
                                0   1   seven bits per character
                                1   0   six   bits per character
                                1   1   five bits per character

4,5   UART MODE   These two bits select whether operations will be
                       asynchronous or synchronous for the transmitter
                       and/or receiver. The actual selection is done as
                       follows:

                       DM1   DM0
                       ---   ---
                        0   0   both transmitter and receiver operate
```

```
                        in asynchronous mode.
            0     1     receiver operates in synchronous mode,
                        transmitter in asynchronous mode.
            1     x     receiver operates in asynchronous mode,
                        transmitter in synchronous mode.


6   RCVR EN       0 = Receiver is disabled.

                 1 = Receiver is Enabled. To provide noise immunity,
                 the duration of a bit interval  is  segmented  into
                 16  subintervals.  This is also used to verify that
                 a high to low  transition  (START bit)  on  the RXD
                 line is  valid  (stays low)  at the half point of a
                 bit  duration;  if  not  valid,  operation will not
                 start.

                 If   after  an idle period, a high to low transition
                 is   detected  on the RXD line and is verified to be
                 low,  the  receiver will synchronized itself to the
                 incoming   character   for   the  duration  of  the
                 character. Received data is then sampled or latched
                 in  the center of a bit time to determine the value
                 of the remaining bits.  The LSB of the data is  the
                 leading   bit   received.  Any  unused  high  order
                 register  bits  will  be  set  "high".  The receiver
                 expects  the data to have only one parity bit (when
                 parity is enabled) and one stop bit.  At the end of
                 the  character  reception,  the receiver will check
                 whether any errors have occured and will update the
                 status register (URSR) accordingly. In addition, if
                 no   errors  were encountered the receiver will load
                 the   contents  of  the  shift  register  into  the
                 Receiver Data Register, eliminating parity and stop
                 bits.

                 In   synchronous mode, the receiver will reconfigure
                 its Data Register and Shift Register so that only 8
                 data bits are always accepted on the RXD line. This
                 mode  only works if an external clock is applied on
                 the  PRC2  input  line,  which is used to shift the
                 bits into the Receiver Shift register.  Data on the
                 RXD  is  latched at the rising edge of the external
                 clock applied in PRC2.

7   XMITR EN      0 = Transmitter is disabled.

                 1 = Transmitter is Enabled.  Transmitter will start
                 operation  once  the  microprocessor writes data to
                 the  transmitter data register (DREG),  after which
                 the  Transmitter  Shift  Register is loaded and the
                 start bit is placed on the TXD line. The LSB of the
                 data  is  the  leading  bit being transmitted.  The
                 Transmitter  is "doubled buffered" which means that
                 the  CPU  can  load  a new character as soon as the
                 previous one starts transmission. This is indicated
                 by the status register, bit 6  (URSR6 -- Empty Data
                 Register),  which  when  set, it indicates that the
                 data   register  is  ready  to  accept  the  next
                 character. The character data format is illustrated
                 by figure 1.3. In synchronous mode, the transmitter
                 will  reconfigure  its  Data  Register  and  Shift
```

Register so that only 8 data bits are always transmitted on the TXD line, eliminating all parity and stop bits. The external clock output will be placed in the PRC2 line and will shift the data out of the transmitter shift register. Data on the TXD line will change on the falling edge of the PRC2 signal, the external clock.

2.3.6.2. UART Status Register (URSR)

BIT	Bit Name	Function
0	FULL	Receiver Data Register Full bit. This bit is forced to a low upon reset, or after the data register (DREG) is read. This bit is enabled only if the RCVER EN bit is set in the URCR register. The FULL bit is set when the character being received is transferred from the receiver shift register into the receiver data register. If an error is encountered in the character data, this bit will not be set and the proper error bit will be set in the URSR register.
1	OVR	Receiver Over-Run Error bit. This bit is cleared upon reset or after reading the receiver data register. This bit is set if the new received charater is attempted to be transferred from the receiver shift register before reading the last character from the data register. Therefore, the last character is preserved in the data register while the new received character is lost.
2	PRTY	Receiver Parity Error bit. This bit is cleared upon reset or after reading the receiver data register. The PRTY bit will be set when a parity error is detected on the received character, provided the PARITY EN bit is set and receiver is running asynchronously.
3	FRME	Receiver Frame Error bit. This bit is cleared upon reset or after reading the receiver data register. The FRME bit is set whenever the received character contains a low in the first stop-bit slot.
4	IDLE	Receiver Idle bit. When this bit is written to a "high", the status register bits 0-3 are disabled until the receiver detects 10 consecutive marks, highs, on the RXD line, at which time the IDLE bit is cleared. This bit is also cleared upon reset. This bit allows the microprocessor, or any external microprocessor device, to ignore the transmission of a character until the start of the next character.
5	ENDT	Transmitter End of Transmission bit. This bit is cleared upon reset or whenever data is written into the transmitter data register, DREG. Setting this bit would disable the Transmitter Empty bit, EMPTY, until device completes transmission.

2.3.6.3. Character Configuration

```
       ASYNC MODE


     S
     T B                   P = PARITY BIT
     A I                 STP = STOP BIT
     R T
     T   LSB                                                        --+
MARK>-+    +---+---+---+---+---+---+---+---+                         |   P
      |    | D0| D1| D2| D3| D4| P |STP|STP|   <--   5-BIT/CHARACTER  |   A
      |    +---+---+---+---+---+---+---+---+                         |   R
                                                                    |   I
      --+    +---+---+---+---+---+---+---+---+---+                   |   T
        |    | D0| D1| D2| D3| D4| D5| P |STP|STP|   <--  6-BIT/CHARACTER  |   Y
        |    +---+---+---+---+---+---+---+---+---+                   |
                                                                 +-> E
      --+    +---+---+---+---+---+---+---+---+---+---+               |   N
        |    | D0| D1| D2| D3| D4| D5| D6| P |STP|STP|   <--  7-BIT/CHARACTER |   A
        |    +---+---+---+---+---+---+---+---+---+---+               |   B
                                                                    |   L
      --+    +---+---+---+---+---+---+---+---+---+---+               |   E
        |    | D0| D1| D2| D3| D4| D5| D6| D7| P |STP|   <--  8-BIT/CHARACTER |   D
        |    +---+---+---+---+---+---+---+---+---+---+---+           |
                                                                    --+




      --+    +---+---+---+---+---+---+---+                           --+
        |    | D0| D1| D2| D3| D4|STP|STP|   <--  5-BIT/CHARACTER     |   P
        |    +---+---+---+---+---+---+---+                            |   A
                                                                     |   R
                                                                     |   I
      --+    +---+---+---+---+---+---+---+---+                        |   T
        |    | D0| D1| D2| D3| D4| D5|STP|STP|   <--  6-BIT/CHARACTER  |   Y
        |    +---+---+---+---+---+---+---+---+                         |
                                                                  +-> D
      --+    +---+---+---+---+---+---+---+---+---+                     |   I
        |    | D0| D1| D2| D3| D4| D5| D6|STP|STP|   <--  7-BIT/CHARACTER |   S
        |    +---+---+---+---+---+---+---+---+---+                     |   A
                                                                      |   B
      --+    +---+---+---+---+---+---+---+---+---+---+                 |   L
        |    | D0| D1| D2| D3| D4| D5| D6| D7|STP|STP|   <--  8-BIT/CHARACTER |   E
        |    +---+---+---+---+---+---+---+---+---+---+                 |   D
                                                                      --+
                       CHARACTER CONFIGURATION
                              TABLE 3
```

2.3.6.4. Register Map

```
                          C65 UART
```

R/W	REG	NAME	D7	D6	D5	D4	D3	D2	D1	D0
R/W	0	DATA	R/X7	R/X6	R/X5	R/X4	R/X3	R/X2	R/X1	R/X0

R/W	#	Name	b7	b6	b5	b4	b3	b2	b1	b0
READ	1	STATUS	XMIT DONE	XMIT EMPTY	ENDT (R/W)	IDLE (R/W)	FRAME	PARITY	OVER RUN	RCVR FULL
R/W	2	CONTROL	XMIT ON	RCVR ON	UART MODE		WORD LENGTH		PARITY ON	EVEN
R/W	3	BAUD LO	BRL7	BRL6	BRL5	BRL4	BRL3	BRL2	BRL1	BRL0
R/W	4	BAUD HI	BRH7	BRH6	BRH5	BRH4	BRH3	BRH2	BRH1	BRH0
R/W	5	INT MASK	XMIT IRQ	RCVR IRQ	XMIT NMI	RCVR NMI	--	--	--	--
READ	6	INT FLAG	XMIT IRQ	RCVR IRQ	XMIT NMI	RCVR NMI	--	--	--	--

The BAUD RATE can be generated using the following formulas:

$$BaudRate = \frac{URCLK}{16 \times (COUNT+1)} \quad or, \quad COUNT = \frac{URCLK}{16 \times BaudRate} - 1$$

Where: COUNT = value loaded into BAUD RATE register
            URCLK = C7Mhz input,   7.15909 MHz NTSC
                                     7.09375 MHz PAL

The following tables show some of the most common data rates. Data rate errors of less than +/-1.5% are acceptable for most purposes.

    A. NTSC   URCLK = 7.15909 MHZ

BR #	BAUD RATE REQUIRED	COUNT (HEX)	BAUD RATE OBTAINED	PERCENT ERROR	
1	50	22F4	49.999	.0015	
2	75	174D	74.999	.0015	
3	110	0FE3	109.991	.0080	
4	134.5	0CFE	134.488	.0090	
5	150	0BA6	149.998	.0015	
6	300	05D2	299.895	.035	
7	600	02E9	599.79	.035	
8	1200	0174	1199.58	.035	
9	1800	00F8	1796.96	.17	
10	2400	00B9	2392.74	.30	
11	3600	007B	3608.41	.23	
12	4800	005C	4811.22	.23	
13	7200	003D	7216.82	.23	
14	9600	002E	9520.07	.83	
15	19200	0016	19454.0	1.323	
16	31250	000D	31960.2	1.023	(MIDI)
0	56000	0007	55930.4	.124	

B. PAL   URCLK = 7.09375 MHZ

```
+----+----------+---------+----------+---------+
| BR | BAUD RATE |  COUNT  | BAUD RATE | PERCENT |
| #  |  REQUIRED |  (HEX)  | OBTAINED  | ERROR   |
+----+----------+---------+----------+---------+
|  1 |       50 |   22A2  |   50.001 |  .0020  |
|  2 |       75 |   1716  |   75.005 |  .0080  |
|  3 |      110 |   0FBE  |  109.987 |  .010   |
|  4 |    134.5 |   0CDF  |  134.514 |  .010   |
|  5 |      150 |   0B8B  |  149.986 |  .009   |
|  6 |      300 |   05C5  |  299.973 |  .009   |
|  7 |      600 |   02E2  |  599.75  |  .009   |
|  8 |     1200 |   0170  | 1198.27  |  .144   |
|  9 |     1800 |   00F5  | 1802.27  |  .126   |
| 10 |     2400 |   00B8  | 2396.54  |  .144   |
| 11 |     3600 |   007A  | 3604.55  |  .126   |
| 12 |     4800 |   005B  | 4819.12  |  .398   |
| 13 |     7200 |   003D  | 7150.96  |  .68    |
| 14 |     9600 |   002D  | 9638.25  |  .40    |
| 15 |    19200 |   0016  | 19276.5  |  .40    |
| 16 |    31250 |   000D  | 31668.5  |  1.01   | (MIDI)
|  0 |    56000 |   0007  | 55419.9  |  1.04   |
+----+----------+---------+----------+---------+
```

2.3.7. CPU

2.3.7.1. Introduction

    The 4502, upon reset, looks and acts like any other CMOS 6502
processor, with the exception that many instructions are shorter or
require less cycles than they used to. This causes programs to execute
in less time that older versions, even at the same clock frequency.

    The, stack pointer has been expanded to 16 bits, but can be used
in two different modes. It can be used as a full 16-bit (word) stack
pointer, or as an 8-bit (byte) pointer whose stack page is
programmable. On reset, the byte mode is selected with page 1 set as
the stack page. This is done to make it fully 65C02 compatible.

    The zero page is also programmable via a new register, the "B" or
"Base Page" register. On reset, this register is cleared, thus giving
a true "zero" page for compatability reasons, but the user can define
any page in memory as the "zero" page.

    A third index register, "Z", has been added to increase
flexibility in data manipulation. This register is also cleared, on
reset, so that the STZ instructions still do what they used to, for
compatibility.

    This is a list of opcodes that have been added to the 210
previously defined MOS, Rockwell, and GTE opcodes.

1. Branches and Jumps

```
        BCC label        word-relative
        BCS label        word-relative
        BEQ label        word-relative
        BMI label        word-relative
        BNE label        word-relative
        BPL label        word-relative
        BRA label        word-relative
        BVC label        word-relative
        BVS label        word-relative


        BSR label        Branch to subroutine (word relative)
        JSR (ABS)        Jump to subroutine absolute indirect
        JSR (ABS,X)      Jump to subroutine absolute indirect, X
        RTN #            Return from subroutine and adjust stack pointer
```

2. Arithmetic Operations

```
        NEG A            Negate (or 2's complement) accumulator


        ASR A            Arithmetic Shift right accumulator or memory
        ASR ZP
        ASR ZP,X


        INW ZP           Increment Word
        DEW ZP           Decrement Word


        INZ              Increment and
        DEZ              Decrement Z register


        ASW ABS          Arithmetic Shift Left Word
        ROW ABS          Rotate Left Word


        ORA (ZP),Z       These were formerly (ZP) non-indexed
        AND (ZP),Z       now are indexed by Z register
        EOR (ZP),Z       (when .Z=0, operation is the same)


        ADC (ZP),Z
        CMP (ZP),Z
        SBC (ZP),Z


        CPZ IMM          Compare Z register with memory immediate,
        CP2 ZP           zero page, and
        CPZ ABS          absolute.
```

3. Loads, Stores, Pushes, Pulls and Transfers

```
        LDA (ZP),Z       formerly (ZP)


        LDZ IMM          Load Z register immediate,
        LDZ ABS          absolute,
        LDZ ABS,X        absolute,X.


        LDA (d,SP),Y     Load Accu via stack vector indexed by Y
        STA (d,SP),Y     and Store


        STX ABS,Y        Store X Absolute,Y
        STY ABS,X        Store Y Absolute,X


        STZ ZP           Store Z register (formerly store zero)
        STZ ABS
```

```
        STZ ZP,X
        STZ ABS,X

        STA (ZP),Z      formerly  (ZP)


        PHD IMM         Push Data Immediate (word)
        PHD ABS         Push Data Absolute (word)


        PHZ             Push Z register onto stack
        PLZ             Pull Z register from stack


        TAZ             Transfer Accumulator to Z register
        TZA             Transfer Z register to Accumulator


        TAB             Transfer Accumulator to Base page register
        TBA             Transfer Base page register to Accumulator


        TSY             Transfer Stack Pointer High byte to Y register
                          and set "byte" stack-pointer mode
        TYS             Transfer Y register to Stack Pointer High byte
                          and set "word" stack-pointer mode
```

2.3.7.2. CPU Operation

The 4502 has the following 8 user registers:

```
        A           accumulator
        X           index-X
        Y           index-Y
        Z           index-Z

        B           Base-page
        P           Processor status
        SP          Stack pointer
        PC          Program counter
```

Accumulator

     The   accumulator is the only general purpose computational
register. It can be used for arithmetic functions add, subtract,
shift,   rotate,   negate,   and   for Boolean functions and, or,
exclusive-or, and bit operations. It cannot, however, be used as an
index register.

Index X

     The index register X has the largest number of opcodes pertaining
to, or using it. It can be incremented, decremented, or compared, but
not used for arithmetic or logical (Boolean) operations. It differs
from other index registers in that it is the only register that can be
used in indexed-indirect or (bp,X) operations. It cannot be used in
indirect-indexed or (bp),Y mode.

Index Y

     The index register Y has the same computational constraints as
the X register, but finds itself in a lot less of the opcodes, making
it less generally used. But the index Y has one advantage over index
X, in that it can be used in indirect-indexed operations or (bp),Y
mode.

Index Z

     The index register Z is the most unique, in that it is used in
the smallest number of opcodes. It also has the same computation
limitations as the X and Y registers, but has an extra feature. Upon
reset, the Z register is cleared so that the STZ (store zero) opcodes
and non-indexed indirect opcodes from previous 65C02 designs are
emulated. The Z register can also be used in indirect-indexed or
(bp),Z operations.

Base page B register

     Early versions of 6502 microprocessors had a special subset of
instructions that required less code and less time to execute. These
were referred to as the "zero page" instructions. Since the addressing
page was always known, and known to be zero, addresses could be
specified as a single byte, instead of two bytes.

     The CSG4502 also implements this same "zero page" set of
instructions, but goes one step further by allowing the programmer to
specify which page is to be the "zero page". Now that the programmer
can program this page, it is now, not necessarily page zero, but
instead, the "selected page". The term "base page" is used, however.

     The B register selects which page will be the "base page", and
the user sets it by transferring the contents of the accumulator to
it. At reset, the B register is cleared, giving initially a true "zero
page".

Processor status P register

     The processor status register is an 8-bit register which is used
to indicate the status of the microprocessor. It contains 8 processor
"flags". Some of the flags are set or reset based on the results of
various types of operations. Others are more specific. The flags
are...

     Flag   Name   Typical indication

     N   Negative   result of operation is negative
     V   Overflow   result of add or subtract causes signed overflow
     E   Extend   disables stack pointer extension
     B   Break   interrupt was caused by BRK opcode
     D   Decimal   perform add/subtract using BCD math
     I   Interrupt   disable IRQ interrupts
     Z   Zero   result of Operation is zero
     C   Carry   operation caused a carry

Stack Pointer SP

     The stack pointer is a 16 bit register that has two modes. It can
be programmed to be either an 8-bit page programmable pointer, or a
full 16-bit pointer. The processor status E bit selects which mode
will be used. When set, the E bit selects the 8-bit mode. When reset,
the E bit selects the 16-bit mode.

     Upon reset, the CSG 4502 will come up in the 8-bit page-
programmable mode, with the stack page set to 1. This makes it
compatible with earlier 6502 products. The programmer can quickly
change the default stack page by loading the Y register with the

desired page and transferring its contents to the stack pointer high
byte, using the TYS opcode. The 8-bit stack pointer can be set by
loading the X register with the desired value, and transferring its
contents to the stack pointer low byte, using the TXS opcode.

       To select the 16-bit stack pointer mode, the user must execute a
CLE (for CLear Extend disable) opcode. Setting the 16-bit pointer is
done by loading the X and Y registers with the desired stack pointer
low and high bytes, respectively, and then transferring their contents
to the stack pointer using TXS and TYS. To return to 8-bit page mode,
simple execute a SEE (SEt Extend disable) opcode.

```
       *************************************************************
       *                         WARNING                          *
       *                                                          *
       *  If you are using Non-Maskable-Interrupts, or Interrupt  *
       *  Request is enabled, and you want to change BOTH stack   *
       *  pointer bytes, do not put any code between the TXS and  *
       *  TYS opcodes. Taking this precaution will prevent any    *
       *  interrupts from occuring between the setting of the two *
       *  stack pointer bytes, causing a potential for writing    *
       *  stack data to an unwanted area.                         *
       *************************************************************
```

Program Counter PC

       The program counter is a 16-bit up-only counter that determines
what area of memory that program information will be fetched from. The
user generally only modifies it using jumps, branches, subroutine
calls, or returns. It is set initially, and by interrupts, from
vectors at memory addresses FFFA through FFFF (hex). See "Interrupts"
below.

2.3.7.3. 65CE02 Interrupts

       There are four basic interrupt sources on the CSG 4502. These are
RES*, IRQ*, NMI*, and SO, for Reset, Interrupt Request, Non-Maskable
Interrupt, and Set Overflow. The Reset is a hard non-recoverable
interrupt that stops everything. The IRQ is a "maskable" interrupt, in
that its occurance can be prevented. The MMI is "non-maskable", and if
such an event occurs, cannot be prevented. The SO, or Set Overflow, is
not really an interrupt, but causes an externally generated condition,
which can be used for control of program flow.

       One important design feature, which must be remembered is that no
interrupt can occur immediately after a one-cycle opcode. This is very
important, because there are times when you want to temporarily
prevent interrupts from occurring. The best example of this is, when
setting a 16-bit stack pointer, you do not want an interrupt to occur
between the times you set the low-order byte, and the high-order byte.
If it could happen, the interrupt would do stack writes using a
pointer that was only partially set, thus, writing to an unwanted
area.

IRQ*

       The IRQ* (Interrupt ReQuest) input will cause an interrupt, if it
is at a low logic level, and the I processor status flag is reset. The
interrupt sequence will begin with the first SYNC after a
multiple-cycle opcode. The two program counter bytes PCH and PCL, and

the processor status register P, are pushed onto the stack. (This causes the stack pointer SP to be decremented by 3.) Then the program counter bytes PCL and PCH are loaded from memory addresses FFFE and FFFF, respectively.

An interrupt caused by the IRQ* input, is similar to the BRK opcode, but differs, as follows. The program counter value stored on the stack points to the opcode that would have been executed, had the interrupt not occurred. On return from interrupt, the processor will return to that opcode. Also, when the P register is pushed onto the stack, the B or "break" flag pushed, is zero, to indicate that the interrupt was not software generated.

NMI*

The NMI* (Non-Maskable Interrupt) input will cause an interrupt after receiving high to low transition. The interrupt sequence will begin with the first SYNC after a multiple-cycle opcode. NMI* inputs cannot be masked by the processor status register I flag. The two program counter bytes PCH and PCL, and the processor status register P, are pushed onto the stack. (This causes the stack pointer SP to be decremented by 3.) Then the program counter bytes PCL and PCH are loaded from memory addresses FFFA and FFFB.

As with IRQ*, when the P register is pushed onto the stack, the B or "break" flag pushed, is zero, to indicate that the interrupt was not software generated.

RES*

The RES* (RESet) input will cause a hard reset instantly as it is brought to a low logic level. This effects the following conditions. The currently executing opcode will be terminated. The B and Z registers will be cleared. The stack pointer will be set to "byte" mode/with the stack page set to page 1. The processor status bits E and I will be set.

The RES* input should be held low for at least 2 clock cycles. But once brought high, the reset sequence begins on the CPU cycle. The first four cycles of the reset sequence do nothing. Then the program counter bytes PCL and PCH are loaded from memory addresses FFFC and FFFD, and normal program execution begins.

SO

The SO (Set Overflow) input does, as its name implies, set the overflow or V processor status flag. The effect is immediate as this active low signal is brought or held at a low logic level. Care should be taken if this signal is used, as some of the opcodes can set or reset the overflow flag, as well. NOTE: The SO pin has been removed for C65.

2.3.7.4. 65CE02 Addressing Modes

It should be noted that all 8-bit addresses are referred to as "byte" addresses, and all 16-bit addresses are referred to as "word" addresses. In all word addresses, the low-order byte of the address is fetched from the lower of two consecutive memory addresses, and the high-order byte of the address is fetched the higher of the two. So, in all operations, the low-order address is fetched first.

Implied   OPR

     The register or flag affected is identified entirely by the
opcode in this (usually) single cycle instruction. In this document,
any implied operation, where the implied register is not explicitly
declared, implies the accumulator. Example: INC with no arguments
implies "increment the accumulator".

Immediate (byte, word)   OPR #xx

     The data used in the operation is taken from the byte or bytes
immediately following the opcode in the 2-byte or 3-byte instruction.

Base Page   OPR bp   (formerly Zero Page)

     The second byte of the two-byte instruction contains the
low-order address byte, and the B register contains the high-order
address byte of the memory location to be used by the operation.

Base Page, indexed by X   OPR bp,X   (formerly Zero Page,X)

     The second byte of the two-byte instruction is added to the X
index register to form the low-order address byte, and the B register
contains the high-order address byte of the memory location to be used
by the operation.

Base Page, indexed by Y   OPR bp,Y   (formerly Zero Page,Y)

     The second byte of the two-byte instruction is added to the Y
index register to form the low-order address byte, and the B register
contains the high-order address byte of the memory location to be used
by the operation.

Absolute   OPR abs

     The second and third bytes of the three-byte instruction contain
the low-order and high-order address bytes, respectively, of the
memory location to be used by the operation.

Absolute, indexed by X   OPR abs,X

     The second and third bytes of the three-byte instruction are
added to the unsigned contents of the X index register to form the
low-order and high-order address bytes, respectively, of the memory
location to be used by the operation.

Absolute, indexed by Y   OPR abs,Y

     The second and third bytes of the three-byte instruction are
added to the unsigned contents of the Y index register to form the
low-order and high-order address bytes, respectively, of the memory
location to be used by the operation.

Indirect (word)   OPR (abs)   (JMP and JSR only)

     The second and third bytes of the three-byte instruction contain
the low-order and high-order address bytes, respectively, of two
memory locations containing the low-order and high-order JMP or JSR
addresses, respectively.

Indexed by X, indirect (byte)   OPR (bp,X)   (formerly (zp,X))

     The second byte of the two-byte instruction is added to the
contents of the X register to form the low-order address byte, and the
contents of the B register contains the high-order address byte, of
two memory locations that contain the low-order and high-order address
of the memory location to be used by the operation.

Indexed by X, indirect (word)   OPR (abs,X)   (JMP and JSR only)

     The second and third bytes of the three-byte instruction are
added to the unsigned contents of the X index register to form the
low-order and high-order address bytes, respectively, of two memory
locations containing the low-order and high-order JMP or JSR address
bytes.

Indirect, indexed by Y   OPR (bp),Y   (formerly (zp),Y)

     The second byte of the two-byte instruction contains the
low-order byte, and the B register contains the high-order address
byte of two memory locations whose contents are added to the unsigned
Y index register to form the address of the memory location to be used
by the operation.

Indirect, indexed by Z   OPR (bp),Z   (formerly (zp))

     The second byte of the two-byte instruction contains the
low-order byte, and the B register contains the high-order address
byte of two memory locations whose contents are added to the unsigned
Z index register to form the address of the memory location to be used
by the operation.

Stack Pointer Indirect, indexed by Y   OPR (d,SP),Y   (new)

     The second byte of the two-byte instruction contains an unsigned
offset value, d, which is added to the stack pointer (word) to form
the address of two memory locations whose contents are added to the
unsigned Y register to form the address of the memory location to be
used by the operation.

Relative (byte)   Bxx LABEL   (branches only)

     The second byte of the two-byte branch instruction is sign-
extended to a full word and added to the program counter (now
containing the opcode address plus two). If the condition of the
branch is true, the sum is stored back into the program counter.

Relative (word)   Bxx LABEL   (branches only)

     The second and third bytes of the three-byte branch instruction
are added to the low-order and high-order program counter bytes,
respectively. (the program counter now contains the opcode address
plus two). If the condition of the branch is true, the sum is stored
back into the program counter.

2.3.7.5. 65CE02 Instruction Set

Add memory to accumulator with carry   ADC

A=A+M+C

```
   Addressing Mode                   Abbrev.    Opcode

   immediate                         IMM        69
   base page                         BP         65
   base page indexed X               BP,X       75
   absolute                          ABS        6D
   absolute indexed X                ABS,X      7D
   absolute indexed Y                ABS,Y      79
   base page indexed indirect X      (BP,X)     61
   base page indirect indexed Y      (BP),Y     71
   base page indirect indexed Z      (BP),Z     72


Bytes  Cycles   Mode
  2       2     immediate
  2       3     base page non-indexed, or indexed X or Y
  3       4     absolute  non-indexed, or indexed X or Y
  2       5     base page indexed indirect X, or indirect indexed Y or Z
```

The ADC instructions add data fetched from memory and carry to the contents of the accumulator. The results of the add are then stored in the accumulator. If the "D" or Decimal Mode flag, in the processor status register, then a Binary Coded Decimal (BCD) add is performed.

The "N" or Negative flag will be set if the sum is negative, otherwise it is cleared. The "V" or Overflow flag will be set if the sign of the sum is different from the sign of both addends, indicating a signed overflow. Otherwise, it is cleared. The "Z" or Zero flag is set if the sum (stored into the accumulator) is zero, otherwise, it is cleared. The "C" or carry is set if the sum of the unsigned addends exceeds 255 (binary mode) or 99 (decimal mode).

```
      Flags
  N V E B D I Z C
  N V - - - - Z C
```


And memory logically with accumulator   AND

A=A.and.M

```
   Addressing Mode                   Abbrev.    Opcode

   immediate                         IMM        29
   base page                         BP         25
   base page indexed X               BP,X       35
   absolute                          ABS        2D
   absolute indexed X                ABS,X      3D
   absolute indexed Y                ABS,Y      39
   base page indexed indirect X      (BP,X)     21
   base page indirect indexed Y      (BP),Y     31
   base page indirect indexed 2      (BP),Z     32


Bytes  Cycles   Mode
  2       2     immediate
  2       3     base page non-indexed, or indexed X or Y
  3       4     absolute non-indexed, or indexed X or Y
  2       5     base page indexed indirect X, or indirect indexed Y or Z
```

The AND instructions perform a logical "and" between data bits

fetched from memory and the accumulator bits. The results are then stored in the accumulator. For each accumulator and corresponding memory bit that are both logical 1's, the result is a 1. Otherwise it is 0.

The "N" or Negative flag will be set if the bit 7 result is a 1. Otherwise it is cleared. The "Z" or Zero flag is set if all result bits are zero, otherwise, it is cleared.

```
      Flags
 N V E B D I Z C
 N - - - - - Z -
```

Arithmetic shifts, memory or accumulator, left or right   ASL ASR ASW

```
    ASL    Arithmetic shift left A or M       A<A<<1 or M<M<<1
    ASR    Arithmetic shift right A or M      A<A>>1 or M<M>>1
    ASW    Arithmetic shift left M (word)     Mx<Mw<<1
```

| | | Opcodes | | |
Addressing Mode	Abbrev.	ASL	ASR	ASW
register (A)		0A	43	
base page	BP	06	44	
base page indexed X	BP,X	16	54	
absolute	ABS	0E		CB
absolute indexed X	ABS,X	1E		

```
 Bytes  Cycles    Mode
   1       1      register (ASL)
   1       2      register (ASR)
   2       4      base page (byte) non-indexed, or indexed X
   3       5      absolute non-indexed, or indexed X
   3       7      absolute (ASW)
```

The ASL instructions shift a single byte of data in memory or the accumulator left (towards the most significant bit) one bit position. A 0 is shifted into bit 0.

The "N" or Negative bit will be set if the result bit 7 is (operand bit 6 was) a 1. Otherwise, it is cleared. The "Z" or Zero flag is set if ALL result bits are zero. The "C" or Carry flag is set if the bit shifted out is (operand bit 7 was) a 1. Otherwise, it is cleared.

The ASR instructions shift a single byte of data in memory or the accumulator right (towards the least significant bit) one bit position. Since this is an arithmetic shift, the sign of the operand will be maintained.

The "N" or Negative bit will be set if bit 7 (operand and result) a 1. Otherwise, it is cleared. The "Z" or Zero flag is set if ALL result bits are zero. The "C" or Carry flag is set if the bit shifted out is (operand bit 0 was) a 1. Otherwise, it is cleared.

The ASW instruction shifts a word (two bytes) of data in memory left (towards the most significant bit) one bit position. A zero is shifted into bit 0.

The "N" or Negative bit will be set if the result bit 15 is

(operand bit 14 was) a 1. Otherwise, it is cleared. The "Z" or Zero
flag is set if ALL result bits (both bytes) are zero. The "C" or Carry
flag is set if the bit shifted out is (operand bit 15 was) a 1.
Otherwise, it is cleared.

```
        Flags
   N V E B D I Z C
   N - - - - - Z C
```


Branch conditional or unconditional   BCC BCS BEQ BMI BNE
                                                        BPL BRA BVC BVS

```
   Opcode   Opcode Byte  Opcode Word  Opcode
   Title    Relative     Relative     Purpose

    BCC        90           93        Branch if Carry Clear
    BCS        B0           B3        Branch if Carry Set
    BEQ        F0           F3        Branch if EQual (2 flag set)
    BMI        30           33        Branch if Minus (N flag set)
    BNE        D0           D3        Branch if Not Equal (Z flag clear)
    BPL        10           13        Branch if PLus (N flag clear)
    BRA        80           83        BRanch Always
    BVC        50           53        Branch if overflow Clear
    BVS        70           73        Branch if overflow Set

   Bytes    Cycles     Mode
    2         2      byte-relative
    3         3      word-relative
```

     All branches of this type are taken, if the condition indicated
by the opcode is true. All branch relative offsets are referenced to
the branch opcode location+2. This means that for byte-relative, the
offset is relative to the location after the two instruction bytes.
For word-relative, the offset is relative to the last of the three
instruction bytes.

```
        Flags
   N V E B D I Z C
   - - - - - - - -
```


Break: (force an interrupt)   BRK

```
   Bytes    Cycles     Mode        Opcode
    2         7       implied        00      (stack)<PC+1w,P SP<SP-2
```

     The BRK instruction causes the processor to enter the IRQ or
Interrupt ReQuest state. The program counter (now incremented by 2),
bytes PCH and PCL, and the processor status register P, are pushed
onto the stack. (This causes the stack pointer SP to be decremented by
3.) Then the program counter bytes PCL and PCH are loaded from memory
addresses FFFE and FFFF, respectively.

     The BRK differs from an externally generated interrupt request
(IRQ) as follows. The program counter value stored on the stack is
PC+2, or the address of the BRK opcode+2. On return from interrupt,
the processor will return to the BRK address+2, thus skipping the
opcode byte, and a following "dummy" byte. A normal IRQ will not add
2, so that a return will execute the interrupted opcode. Also, when
the P register is pushed onto the stack, the B or "break" flag is set,

to indicate that the interrupt was software generated. All outside
interrupts push P with the B flag cleared.

```
      Flags
 N V E B D I Z C
 - - - - - - - -
```

Branch to subroutine   BSR

```
 Bytes   Cycles      Mode          Opcode
   3        5      word-relative     63      (stack)<PC+2w SP<SP-2
```

     The BSR Branch to SubRoutine instruction pushes the two program
counter bytes PCH and PCL onto the stack. It then adds the
word-relative signed offset to the program counter. The relative
offset is referenced to the address of the BSR opcode+2, hence, it is
relative to the third byte of the three-byte BSR instruction. The
return address, on the stack, also points to this address. This was
done to make it compatible with the RTS functionality, and to be
consistant will other word-relative operations.

```
      Flags
 N V E B D I Z C
 - - - - - - - -
```


Clear processor status bits   CLC CLD CLE CLI CLV

```
                                  Opcode  Cycles       Flags

                                                   N V E B D I Z C
 CLC   Clear the Carry bit          18      1     - - - - - - - R
 CLD   Clear the Decimal mode bit   D8      1     - - - - R - - -
 CLE   Clear stack Extend disable bit 02    2     - - R - - - - -
 CLI   Clear Interrupt disable bit  58      2     - - - - - R - -
 CLV   Clear the Oveflow bit        B8      1     - R - - - - - -
```

```
 Bytes     Mode
   1       implied
```

     All of the P register bit clear instructions are a single byte
long. Most of them require a single CPU cycle. The CLI and CLE require
2 cycles. The purpose of extending the CLI to 2 cycles, is to enable
an interrupt to occur immediately, if one is pending. Interrupts
cannot occur after single cycle instructions.

Compare registers with memory   CMP CTX CPY CPZ

```
 CMP   Compare  accumulator with memory  (A-M)
 CPX   Compare  index X with memory      (X-M)
 CPY   Compare  index Y with memory      (Y-M)
 CPZ   Compare  index Z with memory      (Z-M)
```

```
                                          Opcodes
 Addressing Mode              Abbrev.  CMP  CPX  CPY  CPZ

 immediate                    IMM      C9   E0   C0   C2
 base page                    BP       C5   E4   C4   D4
 base page indexed X          BP,X     D5
 absolute                     ABS      CD   EC   CC   DC
```

```
   absolute indexed X                ABS,X       DD
   absolute indexed Y                ABS,Y       D9
   base page indexed indirect X  (BP,X)      C1
   base page indirect indexed Y  (BP),Y      D1
   base page indirect indexed Z  (BP),Z      D2


Bytes  Cycles   Mode
  2       2     immediate
  2       3     base page non-indexed, or indexed X or Y
  3       4     absolute non-indexed, or indexed X or Y
  2       5     base page indexed indirect X, or indirect indexed Y or Z
```

     Compares are performed by subtracting a value in memory from the
register being tested. The results are not stored in any register,
except the following status flags are updated.

     The "N" or Negative flag will be set if the result is negative
(assuming signed operands), otherwise it is cleared. The "Z" or Zero
flag is set if the result is zero, otherwise it is cleared. The "C" or
carry flag is set if the unsigned register value is greater than or
equal to the unsigned memory value.

```
      Flags
  N V E B D I Z C
  N - - - - - Z C
```


Compare registers with memory   CMP CPX CPY CPZ

```
     CMP    Compare accumulator with memory  (A-M)
     CPX    Compare index X with memory      (X-M)
     CPY    Compare index Y with memory      (Y-M)
     CPZ    Compare index Z with memory      (Z-M)
```

| | | | Opcodes | | |
Addressing Mode	Abbrev.	CMP	CPX	CPY	CPZ
immediate	IMM	C9	E0	C0	C2
base page	BP	C5	E4	C4	D4
base page indexed X	BP,X	D5			
absolute	ABS	CD	EC	CC	DC
absolute indexed X	ABS,X	DD			
absolute indexed Y	ABS,Y	D9			
base page indexed indirect X	(BP.X)	C1			
base page indirect indexed Y	(BP),Y	D1			
base page indirect indexed Z	(BP),Z	D2			

```
Bytes  Cycles   Mode
  2       2     immediate
  2       3     base page non-indexed, or indexed X or Y
  3       4     absolute  non-indexed, or indexed X or Y
  2       5     base page indexed indirect X, or indirect indexed Y or Z
```

     Compares are performed by subtracting a value in memory from the
register being tested. The results are not stored in any register,
except the following status flags are updated.

     The "N" or Negative flag will be set if the result is negative
(assuming signed operands), otherwise it is cleared. The "Z" or Zero
flag is set if the result is zero, otherwise it is cleared. The "C" or
carry flag is set if the unsigned register value is greater than or

equal to the unsigned memory value.

```
     Flags
 N V E B D I Z C
 N - - - - - Z C
```

Exclusive OR accumulator logically with memory   EOR

A=A.or.M.and..not.(A.and.M)

Addressing Mode	Abbrev.	Opcode
immediate	IMM	49
base page	BP	45
base page indexed X	BP,X	55
absolute	ABS	4D
absolute indexed X	ABS,X	5D
absolute indexed Y	ABS,Y	59
base page indexed indirect X	(BP,X)	41
base page indirect indexed Y	(BP),Y	51
base page indirect indexed Z	(BP),Z	52

Bytes	Cycles	Mode
2	2	immediate
2	3	base page non-indexed, or indexed X or Y
3	4	absolute non-indexed, or indexed X or Y
2	5	base page indexed indirect X, or indirect indexed Y or Z

     The EOR instructions perform an "exclusive or" between bits
fetched from memory and the accumulator bits. The results are then
stored in the accumulator. For each accumulator or corresponding
memory bit that are different (one 1, and one 0) the result is a 1.
Otherwise it is 0.

     The "N" or Negative flag will be set if the bit 7 result is a 1.
Otherwise it is cleared. The "Z" or Zero flag is set if all result
bits are zero, otherwise, it is cleared.

```
     Flags
 N V E B D I Z C
 N - - - - - Z -
```

Jump to subroutine   JSR

Addressing Mode	Abbrev.	Opcode	bytes	cycles
absolute	ABS	20	3	5
absolute indirect	(ABS)	22	3	7
absolute indexed indirect X	(ABS,X)	23	3	7

     The JSR Jump to SubRoutine instruction pushes the two program
counter bytes PCH and PCL onto the stack. It then loads the program
counter with the new address. The return address, stored on the stack,
is actually the address of the JSR opcode+2, or is pointing to the
third byte of the three-byte JSR instruction.

```
     Flags
 N V E B D I Z C
 - - - - - - - -
```

```
   LDA    Load Accumulator from memory    A<M
   LDX    Load index X from memory        X<M
   LDY    Load index Y from memory        Y<M
   LDZ    Load index Z from memory        Z<M
```

Addressing Mode	Abbrev.	LDA	LDX	LDY	LDZ
immediate	IMM	A9	A2	A0	A3
base page	BP	A5	A6	A4	
base page indexed X	BP,X	B5		B4	
base page indexed Y	BP,Y		B6		
absolute	ABS	AD	AE	AC	AB
absolute indexed X	ABS,X	BD		BC	BB
absolute indexed Y	ABS,Y	B9	BE		
base page indexed indirect X	(BP,X)	A1			
base page indirect indexed Y	(BP),Y	B1			
base page indirect indexed Z	(BP),Z	B2			
stack vector indir indexed Y	(d,SP),Y	E2			

Bytes	Cycles	Mode
2	2	immediate
2	3	base page non-indexed, or indexed X or Y
3	4	absolute non-indexed, or indexed X or Y
2	5	base page indexed indirect X, or indirect indexed Y or Z
2	6	stack vector indirect indexed Y

     These instructions load the specified register from memory. The
"N" or Negative flag will be set if the bit 7 loaded is a 1. Otherwise
it is cleared. The "Z" or Zero flag is set if all bits loaded are
zero, otherwise, it is cleared.

```
     Flags
  N V E B D I Z C
  7 - - - - - Z -
```

Negate (twos complement) accumulator   NEG

A=-A

```
  Addressing Mode  Opcode  Bytes  Cycles
    implied           42      1      2
```

     The NEG or "negate" instruction performs a two's-complement
inversion of the data in the accumulator. For example, 1 becomes -1,
-5 becomes 5, etc. The same can be achieved by subtracting A from
zero.

     The "N" or Negative flag will be set if the accumulator bit 7
becomes a 1. Otherwise it is cleared. The "2" or Zero flag is set if
the accumulator is (and was) zero.

```
     Flags
  N V E B D I Z C
  N - - - - - Z -
```

No-operation   NOP

```
Addressing Mode  Opcode  Bytes  Cycles

   implied          EA      1       1
```

The NOP no-operation instruction has no effect, unless used following a MAP opcode. Then its is interpreted as a EOM end-of-map instruction. (See EOM)

```
      Flags
 N V E B D I Z C
 - - - - - - - -
```

Or memory logically with accumulator   ORA

A=A.or.M

```
 Addressing Mode                  Abbrev.    Opcode

 immediate                        IMM        09
 base page                        BP         05
 base page indexed X              BP,X       15
 absolute                         ABS        0D
 absolute indexed X               ABS,X      ID
 absolute indexed Y               ABS,Y      19
 base page indexed indirect X     (BP,X)     01
 base page indirect indexed Y     (BP),Y     11
 base page indirect indexed Z     (BP),2     12
```

```
Bytes   Cycles    Mode
  2       2      immediate
  2       3      base page non-indexed, or indexed X or Y
  3       4      absolute  non-indexed, or indexed X or Y
  2       5      base page indexed indirect X, or indirect indexed Y or Z
```

The ORA instructions perform a logical "or" between data bits fetched from memory and the accumulator bits. The results are then stored in the accumulator. For either accumulator or corresponding memory bit that is a logical 1's, the result is a 1. Otherwise it is 0.

The "N" or Negative flag will be set if the bit 7 result is a 1. Otherwise it is cleared. The "Z" or Zero flag is set if all result bits are zero, otherwise, it is cleared.

```
      Flags
 N V E B D I Z C
 N - - - - - Z -
```

Pull register data from stack   PLA PLP PLX PLY PLZ

```
                                        Opcode

   PLA    Pull Accumulator from stack       68
   PLX    Pull index X from stack           FA
   PLY    Pull index Y from stack           7A
   PLZ    Pull index Z from stack           FB
   PLP    Pull Processor status from stack  28
```

```
   Bytes    Cycles      Mode
```

```
         1         3        register
```

The Pull register operations, first, increment the stack pointer
SP, and then, load the specified register with data from the stack.

Except in the case of PLP, the "N" or Negative flag will be set
if the bit 7 loaded is a 1. Otherwise it is cleared. The "Z" or Zero
flag is set if all bits loaded are zero, otherwise, it is cleared.

In the case of PLP, all processor flags (P register bits) will be
loaded from the stack, except the "B" or "break" flag, which is always
a 1, and the "E" or "stack pointer Extend disable" flag, which can
only be set by SEE, or cleared by CLE instructions.

```
        Flags
  N V E B D I Z C
  N - - - - - Z -   (except PLP)
  7 6 - - 3 2 1 0   (PLP only)
```

Push registers or data onto stack   PHA PHP PHW PHX PHY PHZ

```
    PHA    Push Accumulator onto stack
    PHP    Push Processor status onto stack
    PHW    Push a word from memory onto stack
    PHX    Push index X onto stack
    PHY    Push index Y onto stack
    PHZ    Push index Z onto stack
```

| | | Opcodes | | | | | |
Addressing Mode	Abbrev.	PHA	PHP	PHW	PHX	PHY	PHZ
register		48	08		DA	5A	DB
word immediate	IMMw			F4			
word absolute	ABSw			FC			

```
    Bytes    Cycles      Mode
      1        3        register
      3        5        word immediate
      3        7        word absolute
```

These instructions push either the contents of a register onto
the stack, or push two bytes of data from memory (PHW) onto the stack.
If a register is pushed, the stack pointer will decrement a single
address. If a word from memory is pushed ([SP]<-PC(LO),
[SP-1]<-PC(HI)), the stack pointer will decrement by 2. No flags are
changed.

```
        Flags
  N V E B D I Z C
  - - - - - - - -
```

Reset memory bits   RMB

M=M.and.-bit

```
          Opcode to reset bit
       0    1    2    3    4    5    6    7

      07   17   27   37   47   57   67   77
```

```
   Bytes    Cycles       Mode
     2        4        base-page
```

These instructions reset a single bit in base-page memory, as
specified by the opcode. No flags are modified.

```
       Flags
  N V E B D I Z C
  - - - - - - - -
```


Rotate memory or accumulator, left or right   ROL ROR ROW

   ROL   Rotate memory or accumulator left throught carry
   ROR   Rotate memory or accumulator right throught carry
   ROW   Rotate memory (word) left throught carry

                                          Opcodes
  Addressing Mode   Abbrev.   ROL ROR ROW

  register (A)   2A   6A
  base page   BP   26   66
  base page indexed X   BP,X   36   76
  absolute   ABS   2E   6E   EB
  absolute indexed X   ABS,X   3E   7E

```
  Bytes    Cycles       Mode
    1        1        register
    2        4        base page (byte) non-indexed, or indexed X
    3        5        absolute  non-indexed, or indexed X
    2        6        absolute (word)
```

The ROL instructions shift a single byte of data in memory or the
accumulator left (towards the most significant bit) one bit position.
The state of the "C" or "carry" flag is shifted into bit 0.

The "N" or Negative bit will be set if the result bit 7 is
(operand bit 6 was) a 1. Otherwise, it is cleared. The "Z" or Zero
flag is set if ALL result bits are zero. The "C" or Carry flag is set
if the bit shifted out is (operand bit 7 was) a 1. Otherwise, it is
cleared.

The ROR instructions shift a single byte of data in memory or the
accumulator right (towards the least significant bit) one bit
position. The state of the "C" or "carry" flag is shifted into bit 7.

The "N" or Negative bit will be set if bit 7 is (carry was) a 1.
Otherwise, it is cleared. The "Z" or Zero flag is set if ALL result
bits are zero. The "C" or Carry flag is set if the bit shifted out is
(operand bit 0 was) a 1. Otherwise, it is cleared.

The ROW instruction shifts a word (two bytes) of data in memory
left (towards the most significant bit) one bit position. The state of
the "C" or "carry" flag is shifted into bit 0.

The "N" or Negative bit will be set if the result bit 15 is
(operand bit 14 was) a 1. Otherwise, it is cleared. The "Z" or Zero
flag is set if ALL result bits (both bytes) are zero. The "C" or Carry
flag is set if the bit shifted out is (operand bit 15 was) a 1.
Otherwise, it is cleared.

Return from BRK, interrupt, kemal, or subroutine   RTI RTN RTS

Operation   description   Opcode bytes cycles

   RTI   Return from interrupt   40   1   5   P,PCw<(SP),SP<SP+3
   RTN #n Return from kernal   62   2   7   PCw<(SP)+1,SP<SP+2+N
   RTS   Return from subroutine 60   1   4   PCw<(SP)+1,SP<SP+2

     The RTI or ReTurn from Interrupt instruction pulls P register
data and a return address into program counter bytes PCL and PCH from
the stack. The stack pointer SP is resultantly incremented by 3.
Execution continues at the address recovered from the stack.

       Flags
   N V E B D I Z C
   7 6 - - 3 2 1 0 (RTI only)

     The RTS or ReTurn from Subroutine instruction pulls a return
address into program counter bytes PCL and PCH from the stack. The
stack pointer SP is resultantly incremented by 2. Execution continues
at the address recovered + 1, since BSR and JSR instructions set the
return address one byte short of the desire return address.

     The RTN or ReTurn from kerNal subroutine is similar to RTS,
except that it contains an immediate parameter N indicating how many
extra bytes to discard from the stack. This is useful for returning
from subroutines which have arguments passed to them on the stack. The
stack pointer SP is incremented by 2 + N, instead of by 2, as in RTS.

       Flags
   N V E B D I Z C
   - - - - - - - - (RTN and RTS)
   7 6 - - 3 2 1 0 (RTI)

Set memory bits   SMB

M=M.or.bit

         Opcode to set bit
      0   1   2   3   4   5   6   7
     87 97 A7 B7 C7 D7 E7 F7

   Bytes   Cycles   Mode
     2   4   base-page

     These instructions set a single bit in base-page memory, as
specified by the opcode. No flags are modified.

       Flags
   N V E B D I Z C

Store registers   STA STX STY STZ

```
STA    Store Accumulator to memory    M<A
STX    Store index X to memory        M<X
STY    Store index Y to memory        M<Y
STZ    Store index Z to memory        M<Z
```

```
                                             Opcodes
   Addressing Mode                  Abbrev.   STA  STX  STY  STZ

   base page                        BP        85   86   84   64
   base page indexed X              BP,X      95        94   74
   base page indexed Y              BP,Y           96
   absolute                         ABS       8D   8E   8C   9C
   absolute indexed X               ABS,X     9D        SB   9E
   absolute indexed Y               ABS,Y     99   9B
   base page indexed indirect X     (BP,X)    81
   base page indirect indexed Y     (BP),Y    91
   base page indirect indexed Z     (BP),Z    92
   stack vector indir indexed Y     (d,SP),Y  82
```

```
Bytes  Cycles   Mode
  2       3     base page non-indexed, or indexed X or Y
  3       4     absolute  non-indexed, or indexed X or Y
  2       5     base page indexed indirect X, or indirect indexed Y or Z
  2       6     stack vector indirect indexed Y
```

These instructions store the specified register to memory. No flags are affected.

```
      Flags
 N V E B D I Z C
 - - - - - - - -
```

Transfers (between registers)   TAB TAX TAY TAZ
                                                 TBA TSX TSY TXA
                                                 TXS TYA TYS TZA

Operation		Flags		Transfer	
Symbol	Code	N V E B D I Z C	from	to
TAB	5B	- - - - - - - -	accumulator	base page reg
TAX	AA	N - - - - - Z -	accumulator	index X reg
TAY	A8	N - - - - - Z -	accumulator	index Y reg
TAZ	4B	N - - - - - Z -	accumulator	index Z reg
TBA	7B	N - - - - - Z -	base page reg	accumulator
TSX	BA	N - - - - - Z -	stack ptr low	index X reg
TSY	0B	N - - - - - Z -	stack ptr high	index Y reg
TXA	8A	N - - - - - Z -	index X reg	accumulator
TXS	9A	- - - - - - - -	index X reg	stack ptr low
TYA	98	N - - - - - Z -	index Y reg	accumulator
TYS	2B	- - - - - - - -	index Y reg	stack ptr high
TZA	6B	N - - - - - Z -	index Z reg	accumulator

These instructions transfer the contents of the specified source register to the specified destination register. Any transfer to A, X, Y, or Z will affect the flags as follows. The "N" or "negative" flag will be set if the value moved is negative (bit 7 set), otherwise, it is cleared. The "Z" or "zero" flag will be set if the value moved is zero (all bits 0), otherwise, it is cleared. Any transfer to SPL or SPH will not alter any flags.

```
      ***********************************************************
      *                        WARNING                          *
      *                                                          *
      *  If you are using Non-Maskable-Interrupts, or Interrupt  *
      *  Request is enabled,  and you want to change BOTH stack  *
      *  pointer bytes, do not put any code between the TXS and  *
      *  TYS  opcodes.  Taking this precaution will prevent any  *
      *  interrupts  from  occuring  between the setting of the  *
      *  two  stack  pointer  bytes,  causing  a  potential for  *
      *  writing stack data to an unwanted area.                 *
      ***********************************************************
```

```
    Bytes    Cycles      Mode
      1         1       register
```


Test and reset or set memory bits   TRB TSB

TRB   Test and reset memory bits with accumulator (M.or.A),M<M.and.-A
TSB   Test and set memory bits with accumulator   (M.or.A),M<M.or.A


```
                                    Opcodes
   Addressing Mode            Abbrev.   TRB   TSB

   base page                  BP        14    04
   absolute                   ABS       1C    OC
```

     These instructions test and set or reset bits in memory, using
the accumulator for both a test mask, and a set or reset mask. First,
a logical AND is performed between memory and the accumulator. The "Z"
or "zero" flag is set if all bits of the result of the AND are zero.
Otherwise it is reset.

     The TSB then performs a logical OR between the bits of the
accumulator and the bits in memory, storing the result back into
memory.

     The TRB, instead, performs a logical AND between the inverted
bits of the accumulator and the bits in memory, storing the result
back into memory.

```
    Bytes    Cycles      Mode
      2         4       base page non-indexed
      3         5       absolute  non-indexed
```

```
      Flags
   N V E B D I Z C
   - - - - - - Z -
```


2.3.7.6. 4502 Opcode Table


```
    0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
  +----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
  |BRK |ORA |CLE*|SEE*|TSB |ORA |ASL |RMB0|PHP |ORA |ASL |TSY*|TSB |ORA |ASL |BBR0|
  |    |INDX|    |    |ZP  |ZP  |ZP  |ZP  |    |IMM |    |    |ABS |ABS |ABS |ZP  |
 0
  +----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
  |BPL |ORA |ORA |BPL*|TRB |ORA |ASL |RMB1|CLC |ORA |INC |INZ*|TRB |ORA |ASL |BBR1|
  |REL |INDY|INDZ|WREL|ZPX |ZPX |ZPX |ZP  |    |ABSY|    |    |ABS |ABSX|ABSX|ZP  |
 1
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	JSR ABS	AND INDX	JSR* IND	JSR* INDX	BIT ZP	AND ZP	ROL ZP	RMB2 ZP	PLP	AND IMM	ROL	TYS*	BIT ABS	AND ABS	ROL ABS	BBR2 ZP
3	BMI REL	AND INDY	AND INDZ	BMI* WREL	BIT ZPX	AND ZPX	ROL ZPX	RMB3 ZP	SEC	AND ABSY	DEC	DEZ*	BIT ABSX	AND ABSX	ROL ABSX	BBR3 ZP
4	RTI	EOR INDX	NEG*	ASR* ZP	ASR* ZP	EOR ZP	LSR ZP	RMB4 ZP	PHA	EOR IMM	LSR	TAZ*	JMP ABS	EOR ABS	LSR ABS	BBR4 ZP
5	BVC REL	EOR INDY	EOR INDZ	BVC* WREL	ASR* ZPX	EOR ZPX	LSR ZPX	RMB5 ZP	CLI	EOR ABSY	PHY	TAB*	MAP*	EOR ABSX	LSR ABSX	BBR5 ZP
6	RTS	ADC INDX	RTN*	BSR* WREL	STZ ZP	ADC ZP	ROR ZP	RMB6 ZP	PLA	ADC IMM	ROR	TZA*	JMP IND	ADC ABS	ROR ABS	BBR6 ZP
7	BVS REL	ADC INDY	ADC INDZ	BVS* WREL	STZ ZPX	ADC ZPX	ROR ZPX	RMB7 ZP	SEI	ADC ABSY	PLY	TBA*	JMP INDX	ADC ABSX	ROR ABSX	BBR7 ZP
8	BRU REL	STA INDX	STA* IDSP	BRU* WREL	STY ZP	STA ZP	STX ZP	SMB0 ZP	DEY	BIT IMM	TXA	STY* ABSX	STY ABS	STA ABS	STX ABS	BBS0 ZP
9	BCC REL	STA INDY	STA INDZ	BCC* WREL	STY ZPX	STA ZPX	STX ZPY	SMB1 ZP	TYA	STA ABSY	TXS	STX* ABSY	STZ ABS	STA ABSX	STZ ABSX	BBS1 ZP
A	LDY IMM	LDA INDX	LDX IMM	LDZ* IMM	LDY ZP	LDA ZP	LDX ZP	SMB2 ZP	TAY	LDA IMM	TAX	LDZ* ABS	LDY ABS	LDA ABS	LDX ABS	BBS2 ZP
B	BCS REL	LDA INDY	LDA INDZ	BCS* WREL	LDY ZPX	LDA ZPX	LDX ZPY	SMB3 ZP	CLV	LDA ABSY	TSX	LDZ* ABSX	LDY ABSX	LDA ABSX	LDX ABSY	BBS3 ZP
C	CPY IMM	CMP INDX	CPZ* IMM	DEW* ZP	CPY ZP	CMP ZP	DEC ZP	SMB4 ZP	INY	CMP IMM	DEX	ASW* ABS	CPY ABS	CMP ABS	DEC ABS	BBS4 ZP
D	BNE REL	CMP INDY	CMP INDZ	BNE* WREL	CPZ* ZP	CMP ZPX	DEC ZPX	SMB5 ZP	CLD	CMP ABSY	PHX	PHZ*	CPZ* ABS	CMP ABSX	DEC ABSX	BBS5 ZP
E	CPX IMM	SBC INDX	LDA* IDSP	INW* ZP	CPX ZP	SBC ZP	INC ZP	SMB6 ZP	INX	SBC IMM	EOM NOP	ROW* ABS	CPX ABS	SBC ABS	INC ABS	BBS6 ZP
F	BEQ REL	SBC INDY	SBC INDZ	BEQ* WREL	PHD* IMM	SBC ZPX	INC ZPX	SMB7 ZP	SED	SBC ABSY	PLX	PLZ*	PHD* ABS	SBC ABSX	INC ABSX	BBS7 ZP

2.4. The CSG 4567 System/Video Controller

2.4.1. Description

The CSG 4567 is a low-cost high-peformance system/video controller, designed to be used in a wide variety of low-end home-computer type systems ranging from joystick controlled video games to high-end home-productivity machines with built-in disk drives and monitor. The 4567 was designed with Commodore-64 (C64) architecture as a subset of its advanced features. In addition to having all of the C64 video modes, it also supports the character attributes -- blink, bold, reverse video, and underline, and can display any of the new or old video modes in 80 column or 640 horizontal pixel format, as well as the older 40 column 320 pixel format.

A new "bitplane" video mode was added to allow the displaying of true bitplane type video, with up to eight bitplanes in 320 pixel mode and up to four in 640 pixel mode. The 4567 can also time-multiplex the bitplanes to give a true four-color 1280 pixel picture. Vertical resolution is maintained at 200 lines as standard, but can be doubled to 400 with interlace.

2.4.2. CSG 4567 Pin Assignments

```
                  (*** Pinout will change with 4567R7 ***)


            R B G C S F S R R I I N A R N E G E S V
            V V V V Y G I O O O O O E W O X A X I C
            I I I N B D M M M 2 1 I C   M R M P D C
            D D D D C G * L H * * * 0   A O E N C
            E E E E       * *           P M   D L
            0 0 0 0                         * K


            7 7 7 7 7 8 8 8 8                   1 1
            5 6 7 8 9 0 1 2 3 4 1 2 3 4 5 6 7 8 9 0 1
  PSYNC   74 +-------------------------------------+ 12 CAS*
  CASS    73 |                                     | 13 CASB*
  DISK*   72 |                                     | 14 CASA*
  MEMCLK  71 |                                     | 15 RAS*
  VCC     70 |                                     | 16 CPUCLK
  D0      69 |                                     | 17 DOTCLK
  E0      68 |                                     | 18 XTAL14
  D1      67 |                                     | 19 XTAL17
  E1      66 |                                     | 20 MA0
  D2      65 |                                     | 21 MA1
  E2      64 |              CSG 4567               | 22 MA2
  D3      63 |                                     | 23 MA3
  E3      62 |                                     | 24 MA4
  D4      61 |                                     | 25 MA5
  E4      60 |                                     | 26 MA6
  D5      59 |                                     | 27 MA7
  E5      58 |                                     | 28 MB7
  D6      57 |                                     | 29 MB6
  E6      56 |                                     | 30 MB5
  D7      55 |                                     | 31 A16
  E7      54 +-------------------------------------+ 32 A15
            5 5 5 5 4 4 4 4 4 4 4 4 4 4 3 3 3 3 3 3 3
            3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3


            V I R L E E A A A A A A A A A A A A A A A
```

```
           S R E P X X 0 1 2 3 4 5 6 7 8 9 1 1 1 1 1
           S Q S * T T                   0 1 2 3 4
             * *   H V
                   * *
```

2.4.3. CSG 4567 Operation

     The 4567 accesses two 8-bit memory blocks, which are up to 64K
each, via two 8-bit bidirectional busses. These are D0-D7 and E0-E7.
The D0-D7 bus is common with the CPU chip, ROM, SID, and the expansion
port, and is used for system memory and bitplanes. The E0-E7 port is
only connected to RAM. This RAM is used for COLOR RAM, attribute RAM,
system memory, and bitplanes.

     To access these RAMs, the 4567 has two multiplexed address
busses. These are MA0-MA7, and MB5-MB7. Lines MA0-MA4 are common to
both 64K banks of RAM, but MA5-MA7 go only to bank A, and MB5-MB7 go
only to bank B.

     There are four types of DMA accesses which the 4567 can perform.
Remember that RAS* is asserted on every memory clock cycle. These
are...

mode	operation	CASA*	CASB*	ROM*
1.	4567 reading both banks	X	X	
2.	4567 reading bank "A"	X		
3.	4567 reading ROM			X
4.	4567 doing refresh			

     There are six types of CPU routings to RAM and peripheral devices
that are handled by the 4567.

mode	operation	CASA*	CASB*	ROM*
2.	CPU reading bank "A"	X		
3.	CPU reading bank "B"		X	
4.	CPU writing bank "A"	X		
5.	CPU writing bank "B"		X	
6.	CPU reading ROM			X
7.	CPU accessing I/O1, I/O2, SID, ROMH, ROML			

     There are four basic data routings through the 4567 chip. Three
internal signals route the data busses. WTREG (write 4567 register)
enables routing the external D0-D7 bus to the internal register data
bus. It is normally a logic 1. When it is brought low, the internal
bus disconnects, and the DO-D7 bus output drivers turn on. This is for
CPU reads of 4567 registers or "B" bank RAM. RDBMEM (read "B" bank
memory) routs the E0-E7 data bus to the inputs of the D0-D7 bus output
drivers when at logic level 1. This is for CPU reads of "B" bank RAM.
When 0, (normal) the internal register data bus is routed to the
D0-D7 bus output driver inputs, instead. WTBMEM (write B" bank memory)
turns on the E0-E7 bus drivers, which directly routs the D0-D7 data
bus to the E0-E7 bus when 1. This is for CPU writes to the "B" bank
RAM. When 0, (normal) the E0-E7 bus is input only.

mode	operation	Wtreg	RdBmem	WtBmem
1.	CPU write 4567 register, CPU access external, 4567 DMA, etc	1	0	0 (default)
2.	CPU read 4567 register	0	0	0
```

```
     3.   CPU read B RAM              0       1      0
     4.   CPU write B RAM             1       0      1
```

                    DMA and Special CPU Accesses

VMF -- Video Matrix Fetch

     The 4567 performs Video Matrix Fetches, during displayed video
times, in all of the original VIC-II modes (SCM, MCM, ECM, BMM). This
is true for both 40 and 80 column (320 and 640 pixel) modes. During
VMF, the 4567 reads -- both banks (A & B) of memory over both data
busses D0-D7 and E0-E7. The D0-D7 bus provides the video matrix data,
E0-E3 provides color data, and E4-E7 provides character attribute
data.

CDF -- Character Data Fetch

     The 4567 performs Character Data Fetches immediately after each
Video Matrix Fetch in the original VIC-II modes except bitmap mode.
During this fetch Character image data is fetched from ROM or RAM bank
A over the D0-D7 bus.

BMF -- BitMap Fetch

     The 4567 performs Bitmap Data Fetches immediately after each
Video Matrix Fetch, only in the bitmap mode. During this fetch. Bitmap
image data is fetched from RAM bank A over the D0-D7 bus.

BPF -- BitPlane Fetch

     The 4567 can perform Bitplane image fetches during displayed
video times, if the Bitplane mode (BPM) is enabled. The number and
position of these fetches is determined by which bitplanes are
enabled. During bitplane fetches, even numbered bitplane data is
fetched over D0-D7 and odd numbered bitplane data is fetched over
E0-E7.

RF -- RAM refresh

     The 4567 performs six cycles of dynamic RAM refresh every scanned
video line. During this time no data is fetched and CASA* and CASB*
are not activated.

SPF -- Sprite Pointer Fetch

     Up to eight Sprite Pointer Fetches can occur each scanned video
line. One SPF is generated for each sprite that is enabled and
currently being displayed. During an SPF, the pointer to the sprite
image data is fetched from the video matrix area of memory for the
sprite in question over the D0-D7 data bus.

SDF -- Sprite Data Fetch

     Three Sprite Data Fetches follow each Sprite Pointer Fetch.
During this time, sprite image data for the sprite in question is
fetched over the D0-D7 data bus.

DAT -- Display Address Translation

     Display Address Translation, or DAT fetches, are not actually

DMA-type accesses, but rather CPU address redirections to RAM. In this
case, the unmultiplexed address bus is totally separated from the
multiplexed address bus.

COL -- Color RAM accesses

        Color RAM is also accessed by the CPU via an address translation.
This is because color RAM would otherwise be located in the I/O area.

                Contents of the Internal A and B Memory
                   Address Busses Prior to Multiplexing

Signal	"VMF"	"CDF"	"BMF"	"BPF"	"RF"	"SPF"	"SDF"	"DAT"	"COL"
IA0	VC0	RC0	RC0	RC0	RF0	SF0	SD0	DT0	A0
IA1	VC1	RC1	RC1	RC1	RF1	SF1	SD1	DT1	A1
IA2	VC2	RC2	RC2	RC2	RF2	SF2	SD2	DT2	A2
IA3	VC3	D0	VC0	VC0	RF3	1	SD3	DT3	A3
IA4	VC4	D1	VC1	VC1	RF4	1	SD4	DT4	A4
IA5	VC5	D2	VC2	VC2	1	1	SD5	DT5	A5
IA6	VC6	D3	VC3	VC3	1	1	DO	DT6	A6
IA7	VC7	D4	VC4	VC4	1	1	Dl	DT7	A7
IA8	VC8	D5	VC5	VC5	1	1	D2	DT8	A8
IA9	VC9	D6	VC6	VC6	1	1	D3	DT9	A9
IA10	VM0/VC10	D7	VC7	VC7	RF5	VM0/1	D4	DT10	A10
IA11	VM1	CBO	VC8	VC8	RF6	VM1	D5	DT11	1
IA12	VM2	CB1	VC9	VC9	RF7	VM2	D6	DT12	1
IA13	VM3	CB2	CB2/VC10	BE13/VC10	1	VM3	D7	BE13/DT13	1
IA14	VBO	VBO	VBO	BE14	1	VBO	VBO	BE14	1
IA15	VB1	VB1	VB1	BE15	1	VB1	VB1	BE15	1
IA16	A16	A16	A16	A16	RF8	A16	A16	DT16	1
IB10	0/*	*	*	*	*	*	*	*	*
IB11	1	*	*	*	*	*	*	*	1
IB12	1	*	*	*	*	*	*	*	1
IB13	1	*	*	B013/*	*	*	*	B013/*	1
IB14	1	*	*	B014	*	*	*	B014	1
IB15	1	*	*	B015	*	*	*	B015	1
DMA	1	1	1	1	1	1	1	0	0

Legend :

VC = Video Matrix Counter
RC = Row Counter
VM = Video Matrix Address
VB = Video Bank Address
CB = Character Generator Bank Address
RF = Refresh Counter
SF = Sprite Pointer Fetch Counter
SD = Sprite Data Fetch Counter
DT = Display Address Translator
BE = Bitplane Even Pointer
BO = Bitplane Odd Pointer
A = Address Out = Address In
D = Data fetched from previous fetch
* = "B" bus contents, same as "A" bus
xxx/yyy = contents for 320/640 pixel modes

Multiplexed Address Bus Generation

The A and B memory address busses are multiplexed 2:1 to generate the MA and MB multiplexed address busses. Listed below are the primary addresses used to generate the multiplexed row and column addresses.

signal	row	column
MA0	A0	A5
MA1	A1	A6
MA2	A2	A7
MA3	A3	A8
MA4	A4	A9
MA5	A10	A13
MA6	A11	A14
MA7	A12	A15
MB5	B10	B13
MB6	B11	B14
MB7	B12	B15

ROM physical addresses

0000	New area A
2000	Basic
4000	New area B
5000	Character sets
6000	Kernal

ROM can appear (to the 4567) at 1000-1FFF (bank 0)

and 9000-9FFF (bank 2)
The ROM address translates to   5000-5FFF

Contents of Memory map based on Loram, Hiram, Game, and Exrom

L O R A M	H I R A M	G A M E	E X R O M	0000-7FFF	8000-9FFF /ROML	A000-BFFF /ROMH	C000-CFFF-	D000-DFFF	E000-FFFF /ROMH	Area
X	X	0	1	4KRAM	EXT	NADA	NADA	I/O	EXT	
0	0	1	X	RAM	RAM	RAM	RAM	RAM	RAM	
0	0	X	0	RAM	RAM	RAM	RAM	RAM	RAM	
0	1	0	0	RAM	RAM	EXT	RAM	I/O	ROM	
0	1	1	X	RAM	RAM	RAM	RAM	I/O	ROM	
1	0	0	0	RAM	RAM	RAM	RAM	I/0	RAM	* CG ROM off
1	0	1	X	RAM	RAM	RAM	RAM	I/O	RAM	
1	1	0	0	RAM	EXT	EXT	RAM	I/O	ROM	
1	1	1	0	RAM	EXT	ROM	RAM	I/O	ROM	
1	1	1	1	RAM	RAM	ROM	RAM	I/O	ROM	

Color Palette ROM Programming

index	red	green	blue	fg/bg	I	Q	Y	color
0	0	0	0	0	0	0	0	black
1	15	15	15	1	0	0	1.0	white

```
    2     15      0      0     1      .60    .21    .30   red
    3      0     15     15     1     -.60   -.21    .70   cyan
    4     15      0     15     1      .28    .52    .41   magenta
    5      0     15      0     1     -.28   -.52    .59   green
    6      0      0     15     1     -.32    .31    .11   blue
    7     15     15      0     1      .32   -.31    .89   yellow
    8     15      6      0     1      .49   0      -.54   orange
    9     10      4      0     1      .33   0       .36   brown
   10     15      7      7     1      .32    .11    .63   pink
   11      5      5      5     1      0      0       .33   dark grey
   12      8      8      8     1      0      0       .53   medium, grey
   13      9     15      9     1     -.11   -.21    .84   light green
   14      9      9     15     1     -.13   -.12    .64   light blue
   15     11     11     11     1      0      0       .73   light, grey
```

                  Horizontal Sync Counter Events (assuming HPOS reg=0)

       For NTSC the first 390 HCOUNT steps are at half the primary clock
rate, and 390 are at the primary clock rate, giving 520 counts for 910
clocks. For PAL the first 388 HCOUNT steps are at the slow rate, and
132 are at the faster clock rate, giving 520 counts for 908 clocks.

```
           EVENT           Clock    +256      /2        HCOONT   Duration
           -----           -----    ----      ----      ------   --------
           VSYNC1 START    513      769       384       384      846 59us
           VSYNC1 STOP     449      705       352       352
           VSYNC2 START    58       314       157       157      846 59us
           VSYNC2 STOP     904      250 W     125       125
           HSYNC START     513      769       384       384      63  4.4us
           HSYNC STOP      576      832       416       442
           HEQU1 START     513      769       384       384      36  2.5us
           HEQU1 STOP      549      805       402       414
           HEQU2 START     58       314       157       157      36  2.5us
           HEQU2 STOP      94       350       175       175
           BURST START     576      832       416       442      47  3.3us
           BURST STOP      623      879       439       488
           HBLANK START    478      734       367       367      175 12.2us
           HBLANK STOP     653      909       454       518
```

                      Horizontal DMA Counter Events
                  (these are actual counts - decode 1 count earlier)

```
           EVENT                   HCOUNT
           -----                   ------
           HDMAEN START            15       19   (640 mode)
           HDMAEN STOP             335      339   (640 mode)
           HDEW START              25       32   (38 col)
           HDEN STOP               345      336   (38 col)
           HPIXEN START            24
           HPIXEN STOP             344
           SPR GO                  358
           SPR STOP                359
           SPR CLOCK DIS           360
           SPR CLOCK ENA           488
           SPR DMA START           372  (and EOL)
           SPR DMA STOP            482
           REFRESH START           482
           REFRESH STOP            506
           VINC                    370
```

```
          HRES                    15
          DOG START              16
          DOG STOP              376
          SYNCO                  0
          SYNC1                  1
          SYNC2                  3
          FAST                 390 NTSC        388  PAL
```

Vertical Timings

      When the vertical position register VPOS is set to zero by the
CPU, it actually is storing a compare value of 128, since the MSB of
VPOS is inverted. This actually corresponds to raster count 256, since
the vertical event counter is counting half-lines. When the vertical
event counter matches the VPOS register, the vertical sync. counter is
reset to zero. Multiply the desired line for each event by 2 and
subtract the nominal VPOS value of 256 to get the desired decode. If
the result is negative add the modulo of the vertical event counter,
which is 525 for NTSC and 625 for PAL. The "line" in these tables
refer to raster lines, where line 50. is the first displayed line in a
25 row display.

NTSC

Event	line	v count	- vpos	decode
VSYNC START	11	22	-234	291
VSYNC STOP	14	28	-228	297
VEQU START	8	16	-240	285
VEQU STOP	17	34	-222	303
VBLANK START	8	16	-240	285
VBLANK STOP	28	56	-200	325
EARLY START	64	128	-128	397
EARLY STOP	11	22	-234	291
LATE START	11	23	-233	292
LATE STOP	3	6	-250	275

PAL -- timings begin 25 lines before NTSC because of 50 extra lines

Event	line	v count	- vpos	decode	
VSYNC START	-14	-29	-285	340	
V&YNC STOP	-11	-24	-280	345	
VEQU START	-17	-34	-290	335	*equ/sync is 15 half-lines
VEQU STOP	-9	-19	-275	350	*for pal
VBLANK START	-17	-34	-290	335	
VBLANK STOP	3	6	-250	375	
EARLY START	39	78	-176	447	
EARLY STOP	-14	-29	-285	340	
LATE START	-14	-28	-284	341	
LATE STOP	-22	-44	-300	325	

Note : EARLY and LATE active concurrently indicate GROSS.

Divide ratios (including external sync values)

Counter	Normal	Early	Late	Gross
NTSC vertical	525	524	526	540
```

```
        PAL vertical          625     624     626        640
        NTSC horizontal       910     908     912
        PAL horizontal        908     906     910
        horizontal counter    520     519     521
```

Number of cycles per line

In "slow" CPU mode...

```
        Total cycles no video                     65
        40 column SCM, MCM, ECM, BMM,
        320 pixel BPM, BP0-BP3 only, or
        640 pixel BPM, BP0-BP1 only     no cost   0
        80 column SCM, MCM, ECM, BMM,
        320 pixel BPM, BP0-BP7, or
        640 pixel BPM, BP0-BP3          subtract 40
        Sprites                         subtract  2 per active sprite
```

Examples...

```
 No video on line                                 65
 40 column text or equiv. BPs (no sprites)        65
 80 column text or equiv. BPs (no sprites)        25
 80 column text or equiv. BPs, all sprites         9 -- worst case
```

```
        227 memory cycles/line
        6 refresh
        0-32 sprite
        0,40,80,120,160 video
```

```
        fast    slow
        ----    ----
        277     138     total cycles/line
        -6      -3      refresh
        -32     -16     sprites
        ----    ----
        239     119     avail CPU cycles/line (no video)
```

```
        no      1       2       3       4
        video   fetch   fetch   fetch   fetch
        ------  ------  ------  ------  ------
cpucyc  239     199     159     119     79      all sprites (fast)
cpucyc  271     231     191     151     111     no sprites (fast)
cpucyc  119     79      79      39      39      all sprites (slow)
cpucyc  135     95      95      55      55      no sprites (slow)
```

2.4.4. Programming the new VIC (4567)

The C4567R6 is a high performance single chip video controller
designed to bring exceptional graphics to low cost computer and game
systems. It presently is available in NTSC and PAL versions to match
European and North American television standards.

The following are new features that are added as a superset of
the old VIC-II video controller functions incorporated in the C4567R6.

```
    a. NewVic mode
    b. 80 column character and 640 horizontal pixel mode
    c. Scan interlace and 400 line mode
```

d. Character attributes (blink, highlight, underline, reverse).
        e. Fast clock mode (3.58 vs. 1.02 MHz)
        f. Bitplane mode
        g. Color palettes
        h. Additional ROM
        i. 1280H pixel mode
        j. Display Address Translator (DAT)
        k. Horizontal and vertical positioning
        1. External sync (Genlock)
        m. Alternate character set
        n. Chroma killer

NewVic Mode

        After power-up and reset, the C4567R6 performs as if it were the
"old" VIC chip. In this mode, none of the new features are accessible.
The old VIC II registers appear at addresses $D000-$D3FF, echoed 16
times, every 64 addresses, and any new registers within the 64 byte
block will not exist.

        To put the C4567R6 into "NewVic" mode, the user must write first
an $A5 and then a $96 to the KEY register ($D02F). Once these values
have been entered the C4567R6 will be in "NewVic" mode, and access to
the "NewVic" registers and modes will be possible.

        To take the C4567R6 out of "NewVic" mode, simply write any value
to the KEY register. After doing this, all of the new modes will be
disabled. The registers that were programmed in "NewVic" mode will
retain their current values. It should be noted, however, that since
all old modes are available in new mode, there is little reason to
exit new mode.

80 Column (character) or 640 Pixel (bitmap or bitplane) Mode.

        You can put the C4567R6 into "80 Column Mode" or "640 horizontal
pixel mode" by setting the H640 bit in control register "B". The
normal horizontal rendering is 40 columns or 320 pixels.

        In 80 column character mode, several things change. The Video
Matrix becomes 2K bytes long, where it used to be 1K in 40 column
mode. The character color RAM also becomes 2K bytes long. The
locations of these areas do not change from the prior convention,
except that the low order video matrix address bit is not used in 80
column mode. Where the programmer used to have 16 choices for locating
the Video Matrix within a video bank, in 80 column mode there are only
8 choices.

        Although the color RAM doubles in size to 2K bytes, the area
provided for color RAM in the I/O map only allows for 1K of color RAM.
To read or write the second 1K of color RAM requires that you move
CIA1, CIA2, I/O1, and I/O2 out of the way. To do this, set the "COLOR
RAM @DC00" bit in Control Register "A".

        In 640 pixel bitmap mode, similar changes occur. The video matrix
and color RAM double in size and are positioned in the memory map
exactly as is done in 80 column character mode. The bitmap must now
also double in size from 8K to 16K bytes. Because the total memory
that the video matrix and the bitmap would require now exceeds the
normal 16K byte video bank size, the video bank size has been doubled

from 16K to 32K for the bitmap only. The least significant video bank
bit is ignored, and the high order character generator bank bit
selects which half of the 32K video bank that bitmaps will be fetched
from. The video matrix is still fetched from the normal 16K video
bank.

In 80 column or 640 pixel mode, the sprite pointers are at the
end of the 2K byte video matrix, where they used to be at the end of
the 1K byte video matrix, in 40 column or 320 pixel mode. The size,
location, and resolution of sprites are not affected by any of the
mode switches.

Interlace, and 400 Line Vertical mode

The C4567R6 can interlace scan lines to give a true NTSC, 525
line screen (625 lines on PAL versions), although the default,
however, is a 262 line non-interlaced screen (312 lines on PAL
versions). Set the INT bit in control register "B" to a "1" if you
want interlacing.

The C4567R6 can also give a 400 line vertical resolution, which
is useful in the new Bitplane mode. Set the V400 bit, and the INT bit
in control register "B" to a "1" to enable 400 line bitplanes. (see
Bitplanes, below) The V400 switch will have no effect if the display
is not interlacing. Also, although interlacing is permitted in all of
the old video modes, the same data will appear on both odd and even
rasters, even if the V400 switch is on.

1280 Horizontal Pixel mode

The C4567R6 supports ultra-high resolution graphics by permitting
the programmer to use 1280 pixel lines. This is enabled by setting the
H1280 and H640 bits in control register "B" to a "1".

The 1280 pixels are acheived by time-multiplexing bitplane bits.
This is done by substituting the pixel clock for bitplane 7. This
means that for the first half of each pixel, the color palette will be
fed the normal color index. For the second half of the same pixel, it
will fed the normal index, plus 128. To utilize this feature, the user
must program the color palette to perform the multiplexing function.

The H1280 bit can also be set H640 off. This is a unique mode
that allows the use of 320 and 640 horizontal pixel bitplanes
simultaneously.

Character Attributes

In NewVic mode, the C4567R6 supports four new character
attributes which can be enabled by setting the ATTR bit in Control
Register "B". These are Blink, Highlight, Underlined, and Reverse
Video characters. Any combination of these attributes can be enabled
on a character by character basis, at any time. Certain combinations
will have varying effects. (See table below) Attributes can also be
applied to bitmap mode, and, to a limited extent, to the new bitplane
mode. (see Bitplanes, below)

Blink is enabled by setting bit 4 of the Color RAM location for
each character requiring this attribute. The Blink attribute will

either flash the character on and off, or will alternately enable and
disable the other attributes, if any are selected. The blink rate is
approximately 1 Hz.

     Reverse Video is enabled by setting bit 5 of the Color RAM
location for each character requiring this attribute. Reverse Video is
achieved by simply complementing the character image data for each
character with this attribute. If the character is also underlined,
the underline will be reversed, as well. Highlighted characters also
will reverse. Blink, if enabled, will alternately enable and disable
this attribute.

     Highlight is enabled by setting bit 6 of the Color RAM location
for each character requiring this attribute. Highlight is achieved by
adding 16 to the color index value. As in the past, the character
color is determined by the index value stored in bits 0-3 of the color
RAM. In many respects, bit 6 is merely another color select bit. What
differs is that the Blink attribute can be used to blink between the
"normal" color, and the "highlight" color. Both the character image,
and its background can have unique highlight colors.

     To use the highlight attribute, effectively, color palette
locations 16 through 31 should be programmed to "highlight" colors.
(see Palette, below). Highlight colors don't have to be related to
normal colors, but can be anything.

     Underline is enabled by setting bit 7 of the Color RAM location
for each character requiring this attribute. Underline is accomplished
by forcing "1" character image data on the eighth raster line for each
character with this attribute. If the Blink attribute is also
selected, the underline will blink.

          Summary of Character Attributes and their Effects

Underline Hilite Reverse Blink   Effect
--------- ------ ------- ----- -----------------------
  off   off   off   off   normal character
  off   off   off   on   blinking character
  off   off   on   off   reverse video character
  off   off   on   on   alternate reverse/normal
  off   on   off   off   highlight character
  off   on   off   on   alternate highlight/normal
  off   on   on   off   highlight, reverse video
  off   on   on   on   alternate highlight-reverse/normal
  on   off   off   off   underlined character
  on   off   off   on   normal char with blinking underline
  on   off   on   off   underlined reverse-video
  on   off   on   on   alternate underline-reverse/normal
  on   on   off   off   highlight underlined character
  on   on   off   on   alternate highlight-underline/normal
  on   on   on   off   highlight underlined reversed
  on   on   on   on   alternate hilite-underlined-rev/normal

Fast Clock

     To permit the new system to run certain types of the old C64
software, the C4567R6 provides a normal (slow) CPU clock with a long
term (63us) average of 1.02 Mhz (exactly the C64 clock rate). This is
accomplished by setting up a pattern of 1.79Mhz (560ns) cycles to give

a total of 65 cycles be horizontal scanning line (also, like C64). In addition/logic is provided on the C4567R6 to determine when the microprocessor chip is executing an enhanced opcode, and, if so, subtracts a clock cycle from it.

By setting the FAST bit in Control Register "B", you can instruct the C4567R6 to clock the CPU at 3.58 Mhz, and permit the microprocessor to execute its enhanced instructions at full speed. This can increase CPU speed up to 400%.

BitPlane mode

In addition to the usual video modes provided by the old VIC chip, the C4567R6 provides a bitpiane mode, which allows up to eight bitplanes to be used in the 320, or up to four bitplanes to be used in the 640 horizontal pixel modes.

Enabling BitPlane mode is done by setting the BPM bit in Control Register "B". Doing this will override all of the other video modes. To specify which bitplanes (0-7) to use, set the corresponding bit for each bitplane you want, in the Bitplane Enable register. Bitplane mode may be used with sprites. Bitplane 2 is the foreground/background plane used for sprite/background collision detection and priority.

The bitplanes, whether enabled, or not, provide the eight color value bits used to define what color will be displayed for any pixel on the screen. Bitplane 0 provides the least significant bit of the color value, and bitplane 7 provides the most significant bit. Bitplanes that are not enabled will contribute a "0" to their bit position in the color select code, unless the complement bit for that bitplane, in the complement register, is set.

Any bitplane's data can be inverted, whether or not the bitplane is enabled by setting its respective bit in the Bitplane Complement register. Inversion on unenabled bitplanes will cause them to contribute a "1" instead of their usual "0".

In BitPlane mode, the C4567R6 does not use the Video Bank select bits, like the old VIC chip did. Instead, you can specify which 8k block (in 320 mode), or which 16k block (in 640 mode) of memory you want a bitplane to come out of. Specify where you want the bitplanes to be fetched from, using Bitplane Address registers 0 through 7. Note, however, that the least significant bits of these registers are ignored in 640 pixel mode, and that register 4 through 7 are never used in 640 pixel mode. Even numbered bitplanes can only be fetched from memory bank 0 (addresses 0-FFFF hex), and odd numbered bitplanes can only be fetched from memory bank 1 (addresses 10000-1FFFF hex). So, the bitplane pointers define which section within the confined bank that bitplane data will be fetched from.

In the Bitplane address registers, there are two bit-fields. One field of bits is for the even vertical scan, and the other field of bits is for the odd scan. The odd scan bits are not used unless both INT and V400 bits are set in control register "B".

Attributes can be enabled in bitplane mode by setting the ATTR bit in control register "B". If this is done, the most significant nybble of bytes fetched for bitplane 3 will contain the attribute specification for each 8 by 8 pixel cell, exactly as is done in character modes. One exception is that the "hilite" attribute will be

disabled. The attributes are only applied to bitplane 2, which is also the foreground/background plane for sprite collisions and priority purposes.

    To properly utilize this feature, bitplane 2 must be enabled to provide attributed bitplane data, and bitplane 3 must be disabled, since it will be providing attribute data. Data fetches for the attribute data will occur, because bitplanes 2 and 3 are both fetched in the same memory cycle. You may also enable any other bitplanes as needed. Bitplane 2, and any other bitplane may be complemented, but complementing bitplane 3 will only cause its bit weight to contribute a "1", and will not invert the attribute data.

Note:

    Addresses 1F800-1FFFF hex are the Color and Attribute RAM used in the old video modes. You can use this area for bitplane if you do not plan on switching between old and new video modes and expect the data for both modes to be there.

Color Palette

    The C4567R6, allows the programmer to use the sixteen standard "C64" colors, or define up to 256 custom colors and/or use the palette to perform boolean operations on the bitplane data. The C4567R6 incorporates a 16 word palette ROM and a has a 256 word palette RAM. Each palette location is an index, which can specify one of sixteen possible intensity values (4 bits) each, of Red, Green, and Blue primary colors, plus a single control bit (FGBG) which can be used for foreground/background control for video mixing applications, or to drive a separate monochrome screen.

    The first 16 locations of the palette default to the C64 colors in ROM. The remaining 240 locations are programmable RAM. The first 16 locations can also be replaced with RAM, however, by setting the PAL bit in control register "B". All old video modes, including sprites and exterior, can only access the lowest 16 palette locations (except hilite cells), so you may want to reserve these indices for such features.

    Only bitplane mode can make full use of all palette locations. Even when less than eight bitplanes are used, the bitplane complement bits of the unused bitplanes can be used to specify which part of the palette is to be used. This feature allows the programmer to define multiple sub-palettes, which can be switched between quickly, or to specify an offset in the color table for the bitplanes, allowing separate colors for exterior and sprites.

    To set the color palette, the user must simply write to the color palette RAM. Addresses D100-D1FF (hex) program the 256 Red values, addresses D200-D2FF (hex) program the 256 Green values, and addresses D300-D3FF(hex) program the 256 Blue values. All 256 locations of both the blue and green palettes are only 4 bits wide, so the upper four data bits do nothing. Bit 4 of every red palette location is the FGBG programming bit, the remaining 3 bits are not used. The palette locations are not readable by the CPU.

                        C4567R6 Registers

```
              MEMORY MAP SELECT AND ENABLE REGISTERS
              ---------------------------------------
              (EN BIT MUST BE 1 FOR SELECT TO BE 0)
                        "4510" PORT

+-------+-------+-------+-------+-------+-------+-------+-------+
|       |       |       |       |       |  EN2  |  EN1  |  EN0  | 0000
+-------+-------+-------+-------+-------+-------+-------+-------+
|       |       |       |       |       | CHREN | HIRAM | LORAM | 0001
+-------+-------+-------+-------+-------+-------+-------+-------+


                    VIC-II MODE REGISTERS

+-------+-------+-------+-------+-------+-------+-------+-------+ $D000+
| S0X7  | S0X6  | S0X5  | S0X4  | S0X3  | S0X2  | S0X1  | S0X0  | 00 SPRITE 0 X
+-------+-------+-------+-------+-------+-------+-------+-------+
| S0Y7  | S0Y6  | S0Y5  | S0Y4  | S0Y3  | S0Y2  | S0Y1  | S0Y0  | 01 SPRITE 0 Y
+-------+-------+-------+-------+-------+-------+-------+-------+
| S1X7  | S1X6  | S1X5  | S1X4  | S1X3  | S1X2  | S1X1  | S1X0  | 02 SPRITE 1 X
+-------+-------+-------+-------+-------+-------+-------+-------+
| S1Y7  | S1Y6  | S1Y5  | S1Y4  | S1Y3  | S1Y2  | S1Y1  | S1Y0  | 03 SPRITE 1 Y
+-------+-------+-------+-------+-------+-------+-------+-------+
| S2X7  | S2X6  | S2X5  | S2X4  | S2X3  | S2X2  | S2X1  | S2X0  | 04 SPRITE 2 X
+-------+-------+-------+-------+-------+-------+-------+-------+
| S2Y7  | S2Y6  | S2Y5  | S2Y4  | S2Y3  | S2Y2  | S2Y1  | S2Y0  | 05 SPRITE 2 Y
+-------+-------+-------+-------+-------+-------+-------+-------+
| S3X7  | S3X6  | S3X5  | S3X4  | S3X3  | S3X2  | S3X1  | S3X0  | 06 SPRITE 3 X
+-------+-------+-------+-------+-------+-------+-------+-------+
| S3Y7  | S3Y6  | S3Y5  | S3Y4  | S3Y3  | S3Y2  | S3Y1  | S3Y0  | 07 SPRITE 3 Y
+-------+-------+-------+-------+-------+-------+-------+-------+
| S4X7  | S4X6  | S4X5  | S4X4  | S4X3  | S4X2  | S4X1  | S4X0  | 08 SPRITE 4 X
+-------+-------+-------+-------+-------+-------+-------+-------+
| S4Y7  | S4Y6  | S4Y5  | S4Y4  | S4Y3  | S4Y2  | S4Y1  | S4Y0  | 09 SPRITE 4 Y
+-------+-------+-------+-------+-------+-------+-------+-------+
| S5X7  | S5X6  | S5X5  | S5X4  | S5X3  | S5X2  | S5X1  | S5X0  | 0A SPRITE 5 X
+-------+-------+-------+-------+-------+-------+-------+-------+
| S5Y7  | S5Y6  | S5Y5  | S5Y4  | S5Y3  | S5Y2  | S5Y1  | S5Y0  | 0B SPRITE 5 Y
+-------+-------+-------+-------+-------+-------+-------+-------+
| S6X7  | S6X6  | S6X5  | S6X4  | S6X3  | S6X2  | S6X1  | S6X0  | 0C SPRITE 6 X
+-------+-------+-------+-------+-------+-------+-------+-------+
| S6Y7  | S6Y6  | S6Y5  | S6Y4  | S6Y3  | S6Y2  | S6Y1  | S6Y0  | 0D SPRITE 6 Y
+-------+-------+-------+-------+-------+-------+-------+-------+
| S7X7  | S7X6  | S7X5  | S7X4  | S7X3  | S7X2  | S7X1  | S7X0  | 0E SPRITE 7 X
+-------+-------+-------+-------+-------+-------+-------+-------+
| S7Y7  | S7Y6  | S7Y5  | S7Y4  | S7Y3  | S7Y2  | S7Y1  | S7Y0  | 0F SPRITE 7 Y
+-------+-------+-------+-------+-------+-------+-------+-------+
| S7X8  | S6X8  | S5X8  | S4X8  | S3X8  | S2X8  | S1X8  | S0X8  | 10 SPRITE 8 X
+-------+-------+-------+-------+-------+-------+-------+-------+
|  RC8  |  ECM  |  BMM  |  BLNK |  RSEL | YSCL2 | YSCL1 | YSCL0 | 11 Y SCROLL
+-------+-------+-------+-------+-------+-------+-------+-------+
|  RC7  |  RC6  |  RC5  |  RC4  |  RC3  |  RC2  |  RC1  |  RC0  | 12 RASTER CNT
+-------+-------+-------+-------+-------+-------+-------+-------+
| LPX7  | LPX6  | LPX5  | LPX4  | LPX3  | LPX2  | LPX1  | LPX0  | 13 LITEPEN X
+-------+-------+-------+-------+-------+-------+-------+-------+
| LPY7  | LPY6  | LPY5  | LPY4  | LPY3  | LPY2  | LPY1  | LPY0  | 14 LITEPEN Y
+-------+-------+-------+-------+-------+-------+-------+-------+
|  SE7  |  SE6  |  SE5  |  SE4  |  SE3  |  SE2  |  SE1  |  SE0  | 15 SPRITE ENA
+-------+-------+-------+-------+-------+-------+-------+-------+
|       |       |  RST  |  MCM  |  CSEL | XSCL2 | XSCL1 | XSCL0 | 16 X SCROLL
+-------+-------+-------+-------+-------+-------+-------+-------+
```

SEXY7	SEXY6	SEXY5	SEXY4	SEXY3	SEXY2	SEXY1	SEXYO	17 SPR EXP Y
VS13	VS12	VS11	VS10	CB13	CB12	CB11		18 VS/CB BASES
IRQ				LPIRQ	ISSC	ISBC	RIRQ	19 INTERRUPTS
				MLPI	MISSC	MISBC	MRIRQ	1A INT MASKS
BSP7	BSP6	BSP5	BSP4	BSP3	BSP2	BSP1	BSP0	1B BK/SPR PRI
SCM7	SCM6	SCM5	SCM4	SCM3	SCM2	SCM1	SCM0	1C MC SPR
SEXX7	SEXX6	SEXX5	SEXX4	SEXX3	SEXX2	SEXX1	SEXX0	1D SPR EXP X
SSC7	SSC6	SSC5	SSC4	SSC3	SSC2	SSC1	SSC0	1E SPR-SPR COL
SBC7	SBC6	SBC5	SBC4	SBC3	SBC2	SBC1	SBC0	1F SPR-BK COL
				BORD3	BORD2	BORD1	BORD0	20 EXT COLOR
				BK0C3	BK0C2	BK0C1	BK0C0	21 BK0 COLOR
				BK1C3	BK1C2	BK1C1	B10C0	22 BK1 COLOR
				BK2C3	BK2C2	BK2C1	BK2C0	23 BK2 COLOR
				BK3C3	BK3C2	BK3C1	BK3C0	24 BK3 COLOR
				SM0C3	SM0C2	SM0C1	SM0C0	25 SPR MC0
				SM1C3	SM1C2	SM1C1	SM1C0	26 SPR MC1
				S0C3	S0C2	S0C1	S0C0	27 SPR0 COLOR
				S1C3	S1C2	S1C1	S1C0	28 SPR1 COLOR
				S2C3	S2C2	S2C1	S2C0	29 SPR2 COLOR
				S3C3	S3C2	S3C1	S3C0	2A SPR3 COLOR
				S4C3	S4C2	S4C1	S4C0	2B SPR4 COLOR
				S5C3	S5C2	S5C1	S5C0	2C SPR5 COLOR
				S6C3	S6C2	S6C1	S6C0	2D SPR6 COLOR
				S7C3	S7C2	S7C1	S7C0	2E SPR7 COLOR

VIC-III MODE REGISTERS

								D000+
KEY7	KEY6	KEY5	KEY4	KEY3	KEY2	KEY1	KEY0	2F KEY
ROM @E000	CROM @9000	ROM @C000	ROM @A000	ROM @8000	PAL	EXT SYNC	CRAM @DC00	30 CONTROL A
H640	FAST	ATTR	BPM	V400	H1280	MONO	INT	31 CONTROL B
BP7EN	BP6EN	BP5EN	BP4EN	BP3EN	BP2EN	BP1EN	BP0EN	32 BP ENABS

```
+-------+-------+-------+-------+-------+-------+-------+-------+
|B0AD15 |B0AD14 |B0AD13 |       |B0AD15 |B0AD14 |B0AD13 |       | 33 BITPLANE 0
| ODD   | ODD   | ODD   |       | EVEN  | EVEN  | EVEN  |       |    ADDRESS
+-------+-------+-------+-------+-------+-------+-------+-------+
|B1AD15 |B1AD14 |B1AD13 |       |B1AD15 |B1AD14 |B1AD13 |       | 34 BITPLANE 1
| ODD   | ODD   | ODD   |       | EVEN  | EVEN  | EVEN  |       |    ADDRESS
+-------+-------+-------+-------+-------+-------+-------+-------+
|B2AD15 |B2AD14 |B2AD13 |       |B2AD15 |B2AD14 |B2AD13 |       | 35 BITPLANE 2
| ODD   | ODD   | ODD   |       | EVEN  | EVEN  | EVEN  |       |    ADDRESS
+-------+-------+-------+-------+-------+-------+-------+-------+
|B3AD15 |B3AD14 |B3AD13 |       |B3AD15 |B3AD14 |B3AD13 |       | 36 BITPLANE 3
| ODD   | ODD   | ODD   |       | EVEN  | EVEN  | EVEN  |       |    ADDRESS
+-------+-------+-------+-------+-------+-------+-------+-------+
|B4AD15 |B4AD14 |B4AD13 |       |B4AD15 |B4AD14 |B4AD13 |       | 37 BITPLANE 4
| ODD   | ODD   | ODD   |       | EVEN  | EVEN  | EVEN  |       |    ADDRESS
+-------+-------+-------+-------+-------+-------+-------+-------+
|B5AD15 |B5AD14 |B5AD13 |       |B5AD15 |B5AD14 |B5AD13 |       | 38 BITPLANE 5
| ODD   | ODD   | ODD   |       | EVEN  | EVEN  | EVEN  |       |    ADDRESS
+-------+-------+-------+-------+-------+-------+-------+-------+
|B6AD15 |B6AD14 |B6AD13 |       |B6AD15 |B6AD14 |B6AD13 |       | 39 BITPLANE 6
| ODD   | ODD   | ODD   |       | EVEN  | EVEN  | EVEN  |       |    ADDRESS
+-------+-------+-------+-------+-------+-------+-------+-------+
|B7AD15 |B7AD14 |B7AD13 |       |B7AD15 |B7AD14 |B7AD13 |       | 3A BITPLANE 7
| ODD   | ODD   | ODD   |       | EVEN  | EVEN  | EVEN  |       |    ADDRESS
+-------+-------+-------+-------+-------+-------+-------+-------+
|BP7COMP|BP6COMP|BP5COMP|BP4COMP|BP3COMP|BP2COMP|BP1COMP|BP0COMP| 3B BP COMPS
+-------+-------+-------+-------+-------+-------+-------+-------+
| BPY8  | BPX6  | BPX5  | BPX4  | BPX3  | BPX2  | BPX1  | BPX0  | 3C BITPLANE X
+-------+-------+-------+-------+-------+-------+-------+-------+
| BPY7  | BPY6  | BPY5  | BPY4  | BPY3  | BPY2  | BPY1  | BPY0  | 3D BITPLANE Y
+-------+-------+-------+-------+-------+-------+-------+-------+
| HPOS7 | HPOS6 | HPOS5 | HPOS4 | HPOS3 | HPOS2 | HPOS1 | HPOS0 | 3E HORIZ POS
+-------+-------+-------+-------+-------+-------+-------+-------+
| VPOS7 | VPOS6 | VPOS5 | VPOS4 | VPOS3 | VPOS2 | VPOS1 | VPOS0 | 3F VERT POS
+-------+-------+-------+-------+-------+-------+-------+-------+
```

                      DAT MEMORY PORTS

```
+-------+-------+-------+-------+-------+-------+-------+-------+ D000+
|B0PIX7 |B0PIX6 |B0PIX5 |B0PIX4 |B0PIX3 |B0PIX2 |B0PIX1 |B0PIX0 | 40 BITPLANE 0
+-------+-------+-------+-------+-------+-------+-------+-------+
|B1PIX7 |B1PIX6 |B1PIX5 |B1PIX4 |B1PIX3 |B1PIX2 |B1PIX1 |B1PIX0 | 41 BITPLANE 1
+-------+-------+-------+-------+-------+-------+-------+-------+
|B2PIX7 |B2PIX6 |B2PIX5 |B2PIX4 |B2PIX3 |B2PIX2 |B2PIX1 |B2PIX0 | 42 BITPLANE 2
+-------+-------+-------+-------+-------+-------+-------+-------+
|B3PIX7 |B3PIX6 |B3PIX5 |B3PIX4 |B3PIX3 |B3PIX2 |B3PIX1 |B3PIX0 | 43 BITPLANE 3
+-------+-------+-------+-------+-------+-------+-------+-------+
|B4PIX7 |B4PIX6 |B4PIX5 |B4PIX4 |B4PIX3 |B4PIX2 |B4PIX1 |B4PIX0 | 44 BITPLANE 4
+-------+-------+-------+-------+-------+-------+-------+-------+
|B5PIX7 |B5PIX6 |B5PIX5 |B5PIX4 |B5PIX3 |B5PIX2 |B5PIX1 |B5PIX0 | 45 BITPLANE 5
+-------+-------+-------+-------+-------+-------+-------+-------+
|B6PIX7 |B6PIX6 |B6PIX5 |B6PIX4 |B6PIX3 |B6PIX2 |B6PIX1 |B6PIX0 | 46 BITPLANE 6
+-------+-------+-------+-------+-------+-------+-------+-------+
|B7PIX7 |B7PIX6 |B7PIX5 |B7PIX4 |B7PIX3 |B7PIX2 |B7PIX1 |B7PIX0 | 47 BITPLANE 7
+-------+-------+-------+-------+-------+-------+-------+-------+
```

                      COLOR PALETTES

```
+--------+--------+--------+--------+--------+--------+-------+-------+
```

```
|       |       |       |       | FG/BG | RED3  | RED2  | RED1  | RED0  | 100-1FF RED
+-------+-------+-------+-------+-------+-------+-------+-------+-------+
|       |       |       |       |       | GRN3  | GRN2  | GRN1  | GRN0  | 200-2FF GREEN
+-------+-------+-------+-------+-------+-------+-------+-------+-------+
|       |       |       |       |       | BLU3  | BLU2  | BLU1  | BLU0  | 300-3FF BLUE
+-------+-------+-------+-------+-------+-------+-------+-------+-------+
```

                      COLOR/ATTRIBUTE RAM

```
+-------+-------+-------+-------+-------+-------+-------+-------+
| UNDER | HILIT | REVRS | BLINK | INDX3 | INDX2 | INDX1 | INDX0 | D800-DBFF
+-------+-------+-------+-------+-------+-------+-------+-------+ (DC00-DFFF)
```

              VIDEO BANK SELECT AND ENABLE
              (EN BIT MUST BE 1 FOR VB TO BE 0)

```
+-------+-------+-------+-------+-------+-------+-------+-------+
|       |       |       |       |       |       | VB1   | VB0   | DD00 (WRITE)
+-------+-------+-------+-------+-------+-------+-------+-------+
|       |       |       |       |       |       | EN1   | EN0   | DD02 (WRITE)
+-------+-------+-------+-------+-------+-------+-------+-------+
```

              Limitations of the C4567R6 and How to Avoid Them

     Watch carefully, when particular mode Changes take effect. You
may change PAL, H1280, V400, BPM, ATTR, and H640 modes anytime.
However, the new mode selection will not take effect until after the
last line of the current character row. This is intended to simplify
split-screen programming. But, if you are using the DAT to access
bitmaps or bitplanes, you must wait long enough after selecting a new
H640 or V400 mode to guarantee that the C4567R6 is in the mode you
intended before doing, any DAT accesses. The DAT uses these bits to
determine how to draw the image.

     If you want to use all four 640x400 bitplanes, you will be
limited to a maximum of 5 sprites having unique data. You can have
more sprites, if they are permitted to share data. This limitation is
due to the fact that sprite pointers and data must be fetched from the
16K video matrix which must also be shared with one of the bitplanes.
The bitplane will use 16000 of the 16384 bytes. This leaves 384 bytes,
which would support 6 sprite data blocks of 64 bytes, each. But the
sprite pointers must come out of the highest addressed block, thus
leaving only 5 sprite data blocks available.

     If you really need 8 unique sprites, you can use four 640x384
pixel bitplanes. This is done by setting the row select bit to 24 row
mode. This will give you a total of 16 blocks of 64. This is more than
enough, so you can even have alternate sprite data blocks.

     Note that Sprites and Sprite coordinates are unaffected by screen
resolution, meaning that in 640x400 screens, for example, the sprites
are still the same size on the screen and are still positioned as if
the display map were 320x200. In an 80-column text, or 640-wide
bitplane, screen a "dot" on a sprite will cover 2 pixels.

     Note also that, in bitplane, mode, sprites will only collide with
"background" data which has bits "on" in bitplane 2. All other

bitplanes will NOT cause a sprite-to-background-data collision.

                An Example of How to Program the Color Palette
                   for 1280 Pixel Resolution and Driving FGBG

     In 1280 mode you must use 2 bitplanes to time-multiplex into 1.
So, for example, lets use BP0 for "early" bytes and BP1 for "late"
bytes.

```
      +-----+-----+-----+-----+-----+-----+-----+-----+
      | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |  early BP0
      +-----+-----+-----+-----+-----+-----+-----+-----+


      +-----+-----+-----+-----+-----+-----+-----+-----+
      | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |  early BP1
      +-----+-----+-----+-----+-----+-----+-----+-----+
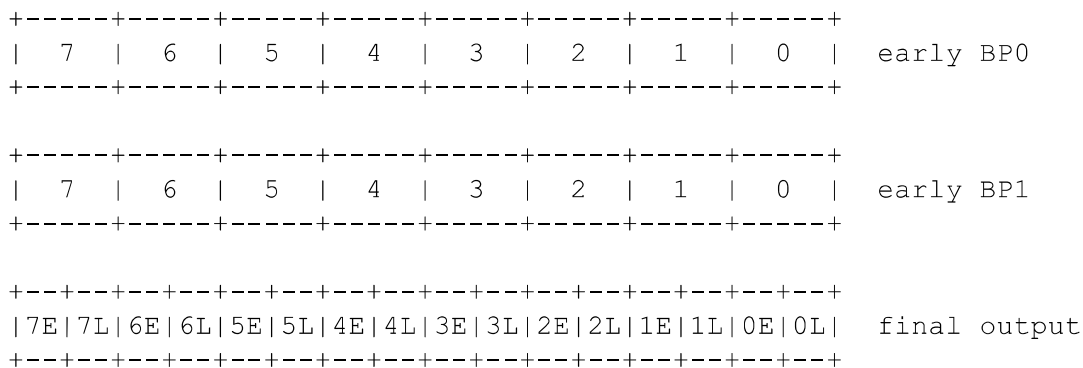

      +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
      |7E|7L|6E|6L|5E|5L|4E|4L|3E|3L|2E|2L|1E|1L|0E|0L|  final output
      +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

The early pixels will be interleaved with the late ones, as shown. So,
if you want to alter 1 pixel, you must decide which bitplane it will
be in, and operate on its byte.

     Make sure the H1280 control bit is set. If it is, BP7 will be
forced low for an early pixel, and high for a late pixel. Let's
program the palette to multiplex BP0 early and BP1 late and ignore BP2
and BP3. I want my background to be black, and image to be white, and,
at the same time have BP3 drive a 640 pixel monochrome screen with the
FGBG pin. (it too could be 1280 pixels).

                    Display Address Translator (DAT)

     The C4567R6 contains a special piece of hardware, known as the
Display Address Translator, or DAT, which allows the programmer to
access the bitplanes directly. In the old VIC configuration, the
bitmap was organized as 25 rows of 40 stacks of 8 sequential bytes.
This is great for displaying 8 x 8 characters, but difficult for
displaying graphics.

     The DAT overcomes the original burden by allowing the programmer
to specify the (X,Y) location of the byte of bitplane memory to be
read, modified, or written. This is done by writing the (X,Y)
coordinates to the BPX and BPY register, respectively. The user can
then read, modify, or write the specified location by reading,
modifying, or writing one of the eight Bitplane registers. There is
one bitplane register for each bitplane.

     The DAT automatically determines whether to use 320 or 640 pixel
mode, and whether to use 200 or 400 line mode. It will also use the
areas specified for the bitplanes, using the Bitplane Address
registers.

Horizontal and Vertical Positioning

     The C4567R6 has two registers to allow the programmer to alter
the positioning of the display relative to the borders of his CRT

(television or monitor). Initially the positioning registers are set to zero, to give C64 standard positioning. These registers are signed, two's complement values which specify an offset from the default positions.

Chroma Killer

The C4567R6 provides analog RGB video, with sync on all colors, an analog luminance output, with sync, and an analog NTSC (or PAL on PAL versions) chrominance output. It also provides a separate digital video signal, and a separate digital sync. When using the C4567R6 with a black and white television receiver, it may be best to suppress the chrominance information. This can be done by setting the MONO bit in control register "B".

Additional ROM

The C4567R6 does all decoding for ROMs. It supports a total of 32K of ROM, which is 12K over what the C64 is configured for. This 12K of extra ROM is available in one 8K block at 8000 (hex), and one 4K block at C000 (hex). To enable ROM at these areas, set the ROM@8000 or ROM@COOO bits in Control Register "A". (Note that there are other chips in the C65 which extend this addressing limitation. The C65 has a 1MB ROM built-in.)

Alternate Character Set

Ordinarily, the C4567R6 will always fetch ROM-based character data from addresses D000-DFFF. If the CROM@9000 bit is set in control register "A", ROM-based character data will be fetched from addresses 9000-9FFF. This allows for an alternate ROM-based character set.

Future Document Topics

At a later time, this document may also describe the following C4567R6 enhancements and features...

    Weatherfax Mode
    Multiple (2-8) playfields
    Playfield prioritization
    Multiple CRT configurations using the digital and analog video
    Multiple sub-palettes
    Mixing 1280 pixel and 640 pixel bitplanes
    Using all 272 palette locations
    Transparency, highlighting, and palette logic functions
    Use of the priority/collision bitplane with the sprites
    Use of external Video RAM

palette addresses	palette outputs			
				F
BBBBBBBB				G
PPPPPPPP	RRRR	GGGG	BBBB	B
76543210	3210	3210	3210	G
--------	----	----	----	-

```
00000000    0000    0000    0000    0    Since BP7 is low,
00000001    1111    1111    1111    0    the early pixel matters.
00000010    0000    0000    0000    0    Only care about BP0 data,
00000011    1111    1111    1111    0    since it supplies the
00000100    0000    0000    0000    1    early data. Notice how
00000101    1111    1111    1111    1    the RGB output is all 1's
00000110    0000    0000    0000    1    only when BP0 is a 1,
00000111    1111    1111    1111    1    regardless of what the
00001000    0000    0000    0000    0    other BP's are doing.
00001001    1111    1111    1111    0    This is how you program
00001010    0000    0000    0000    0    the palette to ignore
00001011    1111    1111    1111    0    certain bitplanes.
00001100    0000    0000    0000    1
00001101    1111    1111    1111    1    Did you see how FGBG is
00001110    0000    0000    0000    1    a 1 only when BP3 is a 1
00001111    1111    1111    1111    1    regardless of other BPs?

10000000    0000    0000    0000    0    Now BP7 is high. The late
10000001    0000    0000    0000    0    pixels are being output.
10000010    1111    1111    1111    0    Now, the RGB output is all
10000011    1111    1111    1111    0    1's only when BP1 (the
10000100    0000    0000    0000    1    late BP) is a 1, regardless
10000101    0000    0000    0000    1    of what the other BPs are
10000110    1111    1111    1111    1    doing. This is how to time
10000111    1111    1111    1111    1    multiplex between planes.
10001000    0000    0000    0000    0
10001001    0000    0000    0000    0    Notice, now, that FGBG is
10001010    1111    1111    1111    0    still a 1 only if BP3 is
10001011    1111    1111    1111    0    a 1, regardless of the
10001100    0000    0000    0000    1    other BPs, like before.
10001101    0000    0000    0000    1    This makes FGBG immune to
10001110    1111    1111    1111    1    the mutiplexing. It also
10001111    1111    1111    1111    1    shows how you can mix
                                         modes on the same screen!
```

Note that BP4, BP5, and BP6 will be zero unless I specifically ask
them to be set to 1 in the Bitplane Complement register. So if they
are zero, I do not need to program the rest of the palette. But I
can program the other parts of the palette, and use the bitplane
complements for BP4, BP5, and BP6 to switch between sub-palettes!

VIC-II modes, enhanced VIC-II modes, and VIC-III modes.

    The VIC-III supports, what are called, "VIC-II" video modes. It
also supports enhancements to the basic VIC-II modes. There are, also
a variety of all-new VIC-III modes. In order to utilize any enhanced
VIC-II mode, or any VIC-III mode, a special keying sequence is
required.

    VIC-II modes

            Standard Character Mode
            Multi-Color Character Mode
            Extended Color Mode
            Bit Map Mode
            Sprites

    Enhancements available to VIC-II modes

            80 column character modes (vs standard 40 columns)

640 x 200 pixel bit maps (vs standard 320 x 200)
            Programmable colors
            Character attributes -- Underline, Blink, Reverse, Hilight
            Alternate character set
            Interlace

      VIC-III video modes

            Bitplane modes
            1280 pixel ultra-high resolution
            400 line operation

Location of VIC-II video data in memory (Video Bank selection)

     The VIC-II modes can only access a maximum of 16K bytes of
memory, out of a total of 64K of potentially available display memory.
To select which fourth of the 64K memory will be available for VIC-II
video accesses, the user must specify which Video Bank to use. This is
done by setting bits 0 and 1 in the Bank Select register (location
DD02 hex) as shown.

                    Bit   Video   Address
                    1 0   Bank   Range
                    - -   ----   -------
                    0 0   3   C000-FFFF
                    0 1   2   8000-BFFF
                    1 0   1   4000-7FFF
                    1 1   0   0-3FFF

The   same   two bits must be set to a 1 in an enable register (location
DD00 hex) in order for a 0 data bit to be recognized. Both of these
registers, though write only, may have bits shared, elsewhere in the
application system. If this is the case, care must be taken to
preserve the other port bits not shown, here.

The Video Matrix

     The   Video   Matrix   is   a block of memory used to store
character-organized display data. Depending on whether the chip is in
40 column or 80 column display mode, it is 1024 or 2048 bytes long.
Since the VIC-II modes can only access 16K bytes of memory, this means
there are "16 or 8 places that the video matrix can appear within the
16K Video Bank, depending on whether 40 or 80 column mode is selected.
The location of the Video matrix is chosen by bits 4 through 7 of the
Memory Pointers register (address D018 hex). Bit 4 has no effect in 80
column mode.

The Character Memory Block

     The Character Memory is a 2048 byte block of memory that contains
character image data. Each character definition requires 8 bytes in
order to display a 8 x 8 bit character image. And there are 256
possible values for each character code, so 8 x 256, or 2048 locations
are required. For each character definition stored in the character
memory, the lowest of the eight memory addresses used by the character
represents the top one of eight scan lines of the character. The
leftmost pixel of each character is the most significant bit (bit 7)
of the respective character memory byte.

Since the VIC-II modes can only access 16K bytes of memory, there are only eight choices where the Character Memory Block can be located. That location is selected by bits 1-3 of the Memory Pointers register (address D018 hex). Special combinations of Character Memory Block and Video Bank selections determine whether the character image data is fetched from RAM or from ROM, as shown below.

```
CB bit  VB bit  Image   hex
3 2 1   1 0     source  address
- - -   - -     ------  -------
0 0 0   x x     RAM     (0-7FF)+VB
0 0 1   x x     RAM     (800-FFF)+VB
0 1 0   x 0     ROM     D000-D7FF (C000-C7FF if CROM@C000)
0 1 0   x 1     RAM     (1000-17FF)+VB
0 1 1   x 0     ROM     D000-D7FF (C000-C7FF if CROM@C000)
0 1 1   x 1     RAM     (1800-1FFF)+VB
1 0 0   x x     RAM     (2000-27FF)+VB
1 0 1   x x     RAM     (2800-2FFF)+VB
1 1 0   x x     RAM     (3000-37FF)+VB
1 1 1   x x     RAM     (3800-3FFF)+VB
```

Color/Attribute Memory

The VIC-II modes have a 1024 or 2048 byte color and attribute memory, depending on whether 40 columns or 80 columns are selected. This memory is used to determine what color and what attributes are to be applied to each character in the video matrix. Color/Attribute RAM is immovable. Physically, it is located at RAM locations 1F800-1FFFF. The CPU, however can access the 1024 byte portion at addresses D800-DBFF. It can access the entire 2048 byte block from D800-DFFF if the COL@DC00 bit is set in control register A. The CPU can also access Color/Attribute RAM directly at addresses 1F800- 1FFFF.

Standard Character Mode

Standard Character Mode is selected by writing 0 to the ECM and BMM bits in Mode Register A (location D011 hex), writing 0 to the MCM bit in Mode Register B (location D016 hex), and by writing 0 to Control Register B (location D031 hex).

2.5. CSG F011x -- C65 Disk Controller Chip gate array (preliminary)

2.5.1. Description

CSG4171-F011 Revision C

The CSG4171-F011 is a low cost MFM disk interface. It requires the use of an external 512 byte RAM as a data cache buffer. This interface can perform reads from and writes to MFM formatted diskettes, as well as free-format full track reads and writes. It can also format diskettes. Logic is also provided for timed head stepping and for motor spin-up. The F011 provides for expansion drive interconnect using a serial protocol for control and status signals. It also incorporates an index pulse simulator for drives that do not have an index sensor.

Unlike its predecessors, the "C" revision provides:

    a. Active high local LED output.
    b. Correct remote DSKCHG status.
    c. Protection of control bits when changing drive selects.
    d. IRQ cleared on reset.
    e. Blinking of the local LED.
    f. Swapping of buffer halves for CPU access.
    g. Two new Digital Phase Locked Loop (DPLL) read recovery methods
       in addition to the original Full Correction (FC) algorithm.
    h. Improved capture range in Full Correction.
    i. Decoding for external disk registers.
    j. A one line to two line active low decoder for external hardware.

Read recovery options

    The F011 now provides 3 methods for recovering MFM formatted disk
data. Each method has its own advantages and tradeoffs. This is how
they work...

    The read-recovery, or dibit counter divides the dibit period into
sixteen partitions or counts assuming no read data pulses occur or
correctly positioned read pulses occur. When a read data pulse with
less-than-ideal positioning occurs, the dibit counter will modify its
count depending on whether Full Correction (FC), Digital Phase Locked
Loop (DPLL) or Alternate Phase Locked Loop (ALT) recovery methods are
selected.


```
                  |----         DIBIT CELL        ----|
     -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
      | | | |0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F| | | |DIBIT COUNTER
     -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-


     ---------------+ +----------------------------
                    | |          READ DATA PULSE
                    +-+
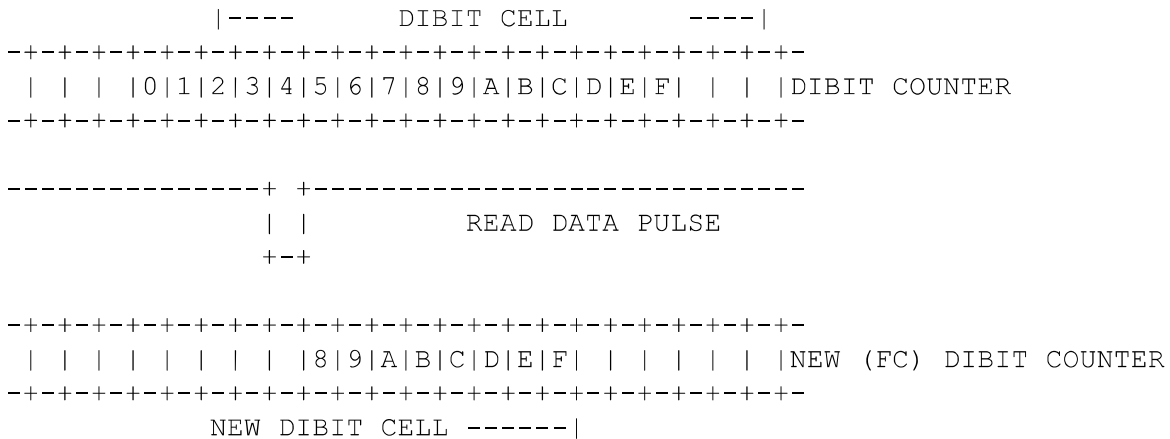

     -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
      | | | | | | | | | |8|9|A|B|C|D|E|F| | | | | | |NEW (FC) DIBIT COUNTER
     -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
              NEW DIBIT CELL ------|
```

    In Full Correction (FC) the dibit counter is forced to count
eight after a read pulse is received. This is the equivalent of
forcing the read pulse to the center of the bit cell. This method
fully compensates for phase and frequency variation. It will tolerate
a considerable range of bit frequency error at the cost of permitting
a limited range of bit phase error.


```
     ---------------+ +----------------------------
                    | |          READ DATA PULSE
                    +-+
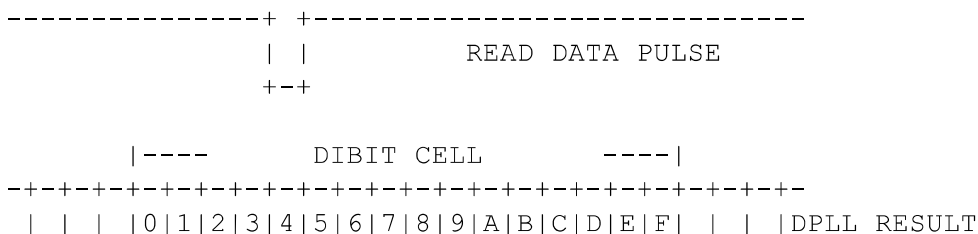

         |----         DIBIT CELL        ----|
     -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
      | | | |0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F| | | |DPLL RESULT
```

```
 -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
       |--- ADD 1 ---| |---- SUB 1 ----|
```

In Digital Phase Locked Loop (DPLL) recovery, the dibit counter
is incremented if a read pulse occurs early (before a dibit cell
center), decremented if a read pulse is late (after a dibit cell
center), or counts normally if no read pulse occurs, or if a pulse
occurs within a dibit cell center. This method has the ability to
track a large range of bit phase error, but, unfortunately can only
handle a very narrow frequency error range.

```
--------------+ +---------------------------
             | |          READ DATA PULSE
             +-+
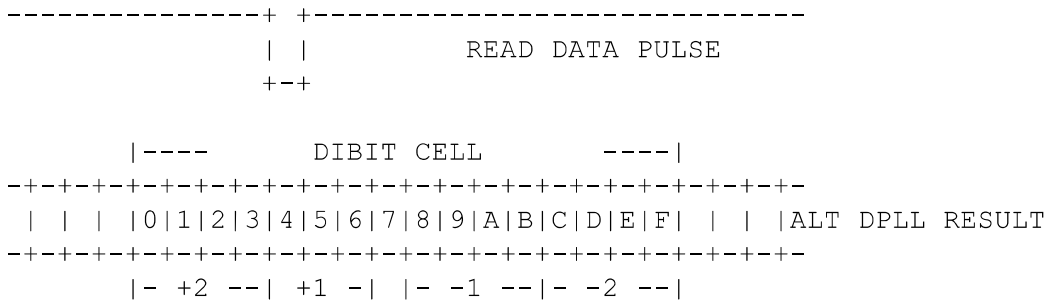
      |----       DIBIT CELL       ----|
 -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
  | | | |0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F| | | |ALT DPLL RESULT
 -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
       |- +2 --| +1 -| |- -1 --|- -2 --|
```

In Alternate Digital Phase Locked Loop (ALT) recovery, the dibit
counter behaves exactly as it does in standard DPLL mode, except that
if a read pulse occurs more than 3 counts early, or 4 counts late, the
counter is incremented or decremented by 2. Like DPLL, this method can
tolerate a large range of bit phase error, but can also compensate for
a larger frequency error range.

2.5.2. Configuration

2.5.2.1. Pinout

Pin	Name	Active	Dir	Type	Description
1	RD	low	input	disk	read-data
2	MOT	low	output	disk	motor on
3	SIDE	low	output	disk	side select
4	WPROT	low	input	disk	write protect
5	TK0	low	input	disk	track 0
6	WGATE	low	output	disk	write gate
7	RW		input	cpu	read/write
8	WD	low	output	disk	write data
9	A0		input	cpu	address
10	A1		input	cpu	address
11	A2		input	cpu	address
12	A3		input	cpu	address
13	INDEX	low	input	disk	index
14	GND				
15	D0		I/O	cpu	data
16	D1		I/O	cpu	data
17	D2		I/O	cpu	data
18	D3		I/O	cpu	data
19	D4		I/O	cpu	data
20	D5		I/O	cpu	data
21	D6		I/O	cpu	data
22	D7		I/O	cpu	data

```
23      IRQ       low     out oc   cpu     interrupt request
24      VCC
25      VCC
26      RA8               output   ram     address
27      RA7               output   ram     address
28      RA6               output   ram     address
29      RA5               output   ram     address
30      RA4               output   ram     address
31      RA3               output   ram     address
32      RA2               output   ram     address
33      RA1               output   ram     address
34      RA0               output   ram     address
35      CS        low     input    cpu     chip select
36      RRW               output   ram     read/write
37      RCS       low     output   ram     chip select
38      RD7               I/O      ram     data
39      RD6               I/O      ram     data
40      RD5               I/O      ram     data
41      RD4               I/O      ram     data
42      RD3               I/O      ram     data
43      RD2               I/O      ram     data
44      RD1               I/O      ram     data
45      RD0               I/O      ram     data
46      SERIO     low     I/O      exp     serial control/status
47      LD        low     output   exp     direction of serio
48      CLK       low     output   exp     shift clock
49      LOCAL     low     input    disk    local drive available
50      TSTCLK            input    test    test clock
51      EXTREG    low     output           to external registers
52      A4                input    cpu     address
53      DR0       low     output   disk    drive select 0
54      CS1       low     input    cpu     chip select external logic
55      LED       high    output   disk    panel LED
56      DIR               output   disk    stepping direction
57      STEP      low     output   disk    stepping command
58      PH0               input    cpu     clock
59      DSKIN     low     input    disk    disk inserted
60      RES       low     input    cpu     reset
61      XTAL1             input            crystal
62      XTAL2             output           crystal
63      VENDOR    low     input            vendor
64      VCC
65      CSLO      low     output           to external logic
66      CSHI      low     output           to external logic
67      GND
68      GND
```

2.5.2.2. Signal Descriptions

Processor Interface Lines

A0-A4   These five address inputs select which internal or external
        register is to be read or written by the processor.

RW   The RW input determines whether a register will be written
        (RW=low) or read (RW=high) by the processor.

D0-D7   Eight bi-directional lines which transfer data to and from
        the processor during register reads and writes. These are
        normally inputs, but become driven outputs when CS and PH0

are true.

CS   The Chip Select is a low-true input that determines that a
           register read or write will occur when PH0 becomes true.

CS1   External hardware chip select input. This low-true signal,
           when asserted, will cause CSLO to go true (low) if A4 is low,
           or CSHI to go true (low) if A4 is high.

PH0   A high-true input that must be driven high by the processor
           to indicate that A0-A4, RW, and CS are valid.

IRQ   The Interrupt Request is an open-drain output that will sink
           current when an interrupt is requested by the F011. IRQ will
           go low (true) when the BUSY status bit changes from true to
           false if IRQ is enabled.

RES   The Reset is a low-true input used to reset internal events.
           When RES goes low (true) any command in progress will be
           terminated. RES will not, however, affect any control
           register bits.

                    Buffer RAM Interface Lines

RA0-RA8   These nine RAM Address outputs must be connected directly to
           nine of the external buffer RAM chip address inputs. These
           may be scrambled for PCB simplification.

RD0-RD7   These eight bi-directional lines must be connected to the
           eight bi-directional data lines of the external buffer RAM.
           These may be scrambled for PCB simplification. RDO-RD7 are
           inputs except when RRW and PCS are low. Then they become
           driven outputs.

RRW   The RAM Read/Write output must be connected to the R/W input
           of the external buffer RAM to control reading and writing.

RCS   The RAM Chip Select is a 1.0 Mhz clock of 50% duty cycle, and
           is low at a time when RA0-RA8, RRW, and RCS are valid. It
           must be connected to the CS input of the external buffer RAM.

                    Disk Drive Interface Lines
                    (All disk signals are low-true)

RD   The Read Data input expects a series of low-going pulses
           from the currently selected disk drive.

WD   The Write Data output provides a series of low-going pulses
           at all times to all drives. It represents the MFM encoded
           data stream used for disk writes.

WGATE   The Write Gate output, when true, causes the Write Data to
           be written to the diskette in the currently selected drive.

WPROT   The Write Protect input must indicate, when true, that the
           present diskette in the local drive must not be written to.
           The F011 will not assert WGATE if WPROT is true, and will not
           execute any write related commands.

LOCAL   The Local Drive Available input must be grouded in systems
          that have a resident local drive 0, and must be tied to Vcc
          in systems that are diskless. This will permit drive 0 to be
          configured externally.

DR0   This output, when low, indicates that the local drive
          (Drive 0) is the currently selected drive.

DISKIN   The Disk In Input must indicate when a diskette is physically
          in the local drive, and the drive is available for use.

MOT   The Motor On output, when true, turns on the motor of the
          local disk drive only. (Also turns on local LED).

LED   The LED output, when true turns on the panel Light-emitting-
          diode of the local disk drive only. (Causes LED to BLINK).

SIDE   The Side select output determines which side of the media
          is to be read or written. It is high (false) for side 0, and
          low (true) for side 1. This output reflects the status of the
          SIDE control bit regardless of which drive is selected.

STEP   The Step output provides a low-going pulse when a head
          stepping command is executed, regardless of which drive is
          selected.

DIR   The Direction output indicates to the drives whether the
          read/write head is to step toward track 0 (DIR=high) or away
          from track 0 (DIR=low) when a step pulse is received. This
          output reflects the status of the DIR command register bit
          regardless of which drive is selected.

TK0   The Track Zero input must determine when the read/write head
          of the local drive is positioned over track zero. This input
          will not suppress stepping pulses.

INDEX   The Index pulse input must provide a low going pulse for each
          spindle rotation of the local drive, if the local drive has
          an index sensor. This input must be tied low if the local
          drive has no index sensor.

                    Expansion Drive Interface Lines
                    (all expansion lines are low-true)

SERIO   The Serial I/O line is a bi-directional signal that is used
          to pass control to all external disk drives, and to receive
          status information from them. It is a driven, output when LD
          is high, and an input, otherwise.

LD   The Load Data output tells the external expansion drives
          when to update control information shifted off of the serio
          line, when to load status information for shifting, and when
          to drive the SERIO line. (This is discussed later.)

CLK   The Clock output provides a 50% duty cycle clock at 250Khz
          to be used by the external expansion drives for shifting
          control and status information in and out.

                          Other Signals

XTAL1   These two lines form two poles of a series-resonant crystal
XTAL2   oscillator circuit. XTAL1 is an input, and XTAL2 is an
         output. An 8.0000Mhz crystal should be used.

VENDOR   The software Vendor identifier input determines whether the
         F011 will be capable of generating protect marks within the
         the sector headers. Production units will not have this
         signal bonded, except those shipped to software vendors. This
         pin should be grounded at all times.

TSTCLK   The Test Clock input is used to reduce F011 test times. This
         pin should be grounded at all times.

CSLO   External hardware active-low chip select output. Goes low
         when CS1 and A4 are both low.

CSHI   External hardware active-low chip select output. Goes low
         when CS1 is low and A4 is high.

EXTREG   External register active-low chip select output. Goes low
         when CS is low and A4 is high.

2.5.3. Registers

                        C4171-F011C Registers

                  7   6   5   4   3   2   1   0
          +-------+-------+-------+-------+-------+-------+-------+-------+
CONTROL | IRQ | LED | MOTOR | SWAP | SIDE | DS2 | DS1 | DS0 | 0 RW
          +-------+-------+-------+-------+-------+-------+-------+-------+

          +-------+-------+-------+-------+-------+-------+-------+-------+
COMMAND | WRITE | READ | FREE | STEP | DIR | ALGO | ALT | NOBUF | 1 RW
          +-------+-------+-------+-------+-------+-------+-------+-------+

          +-------+-------+-------+-------+-------+-------+-------+-------+
STAT A | BUSY | DRQ | EQ | RNF | CRC | LOST | PROT | TKQ | 2 R
          +-------+-------+-------+-------+-------+-------+-------+-------+

          +-------+-------+-------+-------+-------+-------+-------+-------+
STAT B | RDREQ | WTREQ | RUN | NGATE | DSKIN | INDEX | IRQ | DSKCHG| 3 R
          +-------+-------+-------+-------+-------+-------+-------+-------+

          +-------+-------+-------+-------+-------+-------+-------+-------+
TRACK | T7 | T6 | T5 | T4 | T3 | T2 | T1 | T0 | 4 RW
          +-------+-------+-------+-------+-------+-------+-------+-------+

          +-------+-------+-------+-------+-------+-------+-------+-------+
SECTOR | S7 | S6 | S5 | S4 | S3 | S2 | S1 | S0 | 5 RW
          +-------+-------+-------+-------+-------+-------+-------+-------+

          +-------+-------+-------+-------+-------+-------+-------+-------+
SIDE | S7 | S6 | S5 | S4 | S3 | S2 | S1 | S0 | 6 RW
          +-------+-------+-------+-------+-------+-------+-------+-------+

          +-------+-------+-------+-------+-------+-------+-------+-------+
DATA | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | 7 RW
          +-------+-------+-------+-------+-------+-------+-------+-------+

```
         +-------+-------+-------+-------+-------+-------+-------+-------+
CLOCK    |  C7   |  C6   |  C5   |  C4   |  C3   |  C2   |  C1   |  C0   | 8 RW
         +-------+-------+-------+-------+-------+-------+-------+-------+


         +-------+-------+-------+-------+-------+-------+-------+-------+
STEP     |  S7   |  S6   |  S5   |  S4   |  S3   |  S2   |  S1   |  S0   | 9 RW
         +-------+-------+-------+-------+-------+-------+-------+-------+


         +-------+-------+-------+-------+-------+-------+-------+-------+
P CODE   |  P7   |  P6   |  P5   |  P4   |  P3   |  P2   |  P1   |  P0   | A R
         +-------+-------+-------+-------+-------+-------+-------+-------+
```

Control Register

Data from the control register is sent to both the local drive (DR0) and all of the serially connected expansion drives (DR1-DR7). The MOTOR and LED signals will be held for the local drive while other drives are selected.

IRQ   When set, enables interrupts to occur, when reset clears and
          disables interrupts.

LED   These two bits control the state of the MOTOR and LED
MOTOR   outputs. When both are clear, both MOTOR and LED outputs will
          be off. When MOTOR is set, both MOTOR and LED Outputs will be
          on. When LED is set, the LED will "blink".

SWAP   swaps upper and lower halves of the data buffer
          as seen by the CPU.

SIDE   when set, sets the SIDE output to 0, otherwise 1.

DS2-DS0   these three bits select a drive (drive 0 thru drive 7). When
          DS0-DS2 are low and the LOCAL input is true (low) the DR0
          output will go true (low).

Command Register

WRITE   must be set to perform write operations.
READ   must be set for all read operations.
FREE   allows free-format read or write vs formatted
STEP   write to 1 to cause a head stepping pulse.
DIR   sets head stepping direction
ALGO   selects read and write algorithm. 0=FC read, 1=DPLL read,
          0=normal write, 1=precompensated write.

ALT   selects alternate DPLL read recovery method. The ALG0 bit
          must be set for ALT to work.
NOBUF   clears the buffer read/write pointers

Status Registers

The appropriate status bits are sampled from the local status inputs if the local drive (DR0) is selected. Otherwise, those bits are

sampled from the serially connected expansion drive (DR1-DR7).

```
BUSY      command is being executed
DRQ       disk interface has transferred a byte
EQ        buffer CPU/Disk pointers are equal
RNF       sector not found during formatted write or read
CRC       CRC check failed
LOST      data was lost during transfer
PROT      disk is write protected
TK0       head is positioned over track zero

RDREQ     sector found during formatted read
WTREQ     sector found during formatted write
RUN       indicates successive matches during find operation
WGATE     write gate is on
DSKIN     indicates that a disk is inserted in the drive
INDEX     disk index is currently over sensor
IRQ       an interrupt has occurred
DSKCHG    the DSKIN line has changed
          this is cleared by deselecting drive
```

Track Register

Sector Register

Side Register

    The Track, Side and Sector registers are used in FIND operations
to locate a given sector on a given track on a given side.

Data Register

    The data register is the CPU gateway to the data buffer for both
read and write operations.

Clock Register

    The clock register is used to define the clock pattern to be used
to write address and data marks. This register should normally be
written to FF (hex).

Step Register

    The step register is used to time head stepping. This register is
compared to a counter, which is clocked at 16Khz, giving a time of
62.5 microseconds per count, allowing a maximum of 16 milliseconds of
step time per step operation.

Protect Code Register

    The Protection Code register is a read-only register that
contains the protect code of the last sector read. If the last sector
read does not contain a Protect Mark in its header, then this register
will contain zero.

Legal commands are...

```
  hexcode  notes    macro    function
  -------  -----    -----    --------
     40    1,4,5    RDS      Read Sector
     80    1,2      WTS      Write Sector
     60    1,4,5    RDT      Read Track
     A0    1,2      WTT      Write Track (format)
     10    3        STOUT    Head Step Out
     14    3        TIME     Time 1 head step interval (no pulse)
     18    3        STIN     Head Step In
     20    3        SPIN     Wait for motor spin-up
     00    3        CAN      Cancel any command in progress
     01             CLB      Clear the buffer pointers


Notes:     1. Add 1 for nonbuffered operation
           2. Add 4 for write precompensation
           3. Add 1 to clear buffer pointers
           4. Add 4 for DPLL recovery instead of FC recovery
           5. Add 6 for Alternate DPLL recovery
```

2.5.4. Command Descriptions

     Execution of any legal command will cause the BUSY status to be
set, and the IRQ, RNF, CRC, and LOST flags to be cleared. Execution of
the CANcel or CLearBuffer commands, or any write operation command
with, the WPROT status set, or any illegal command, will not cause a
normal BUSY condition. However, any write to either the Command
Register or the Control register will automatically cause BUSY to be
set for at least one round trip delay of transmission and reception of
the serialized control and status signals. When BUSY gets reset,
either by successful command completion, error termination, round trip
completion, or by user cancellation, the IRQ flag will be set, and an
interrupt generated, unless interrupts are disabled.

     The user may CANcel any operation in progress at any time using
the CAN command to can it. Use of this command during write operations
is not advised.

Unbuffered operations

     If the buffer pointers are held clear by setting bit 0 in the
command register while issuing a command, unbuffered operations will
result. These are most useful for formatting a diskette. The DRQ flag
in status register A indicates when a transfer has occurred to or from
the disk.

     For read operations, DRQ set, indicates that a byte of data has
been read from disk, and must be read by the CPU. Reading the data
with the CPU will clear the DRQ flag. If the data is not read by the
time another byte is read from the disk, the old data will be
overwritten and the LOST status flag will be set. The LOST flag will
remain set until the next command is written.

     For write operations, the user should supply the first byte of
data either before, or shortly after issuing a write command. The DRQ
flag set indicates that the byte has been written to disk, and the CPU
must supply the next byte. When the CPU supplies a byte the DRQ flag
will be cleared. If the CPU does not supply a new byte in the time

that it is required by the disk interface, the previous byte data will
be written, and the LOST flag will be set. The LOST flag will remain
set until the next command is written.

Buffered operations

     Buffered operations can be monitored by reading status register
A. The DRQ and EQ bits indicate the immediate status of the buffer
pointers. During any operation, the EQ bit, when set, indicates that
both the disk and CPU buffer pointers are pointing to the same
location. This can mean that the buffer is full or empty, depending on
what operation is, or will be performed. The DRQ bit set indicates
that the disk was last to access the buffer, and clear indicates that
CPU was last to access the buffer.

     For read operations, the disk interface will read bytes from disk
into the buffer. This will set DRQ and clear EQ. The CPU may read data
from the buffer at any time after this occurs, and can continue to
read data until EQ goes high, indicating that the buffer is empty. CPU
reads from the data buffer will clear DRQ. If data is read from disk,
setting DRQ, and EQ also gets set, this indicates that the buffer is
now full. One more byte read from disk will set the LOST flag. The
LOST flag will remain set until the next command is written. This
condition will not usually occur when performing sectored reads of 512
bytes or less, since that is the buffer size.

     For write operations, CPU data may be written to the buffer
before executing a write command, but may also be supplied during the
transfer. If the EQ flag is set after the CPU writes to the buffer,
clearing DRQ, this indicates that the buffer is now full, and that the
CPU should wait before stuffing more data. The the EQ flag goes high
with DRQ high, this indicates that the disk interface has used all of
the available data in the buffer. If one more byte is written to the
disk, the LOST flag will be set, indicating old buffer data has been
written to disk. The LOST flag will remain set until the next command
is written.

Data Transfer Commands

     Execution of any of the Data Transfer Commands must be performed
assuming that the correct drive has been selected, the proper side has
been selected, and the drive's motor is on and has had time to spin
up. The read/write head(s) must be positioned over the track that data
is to be transferred to or from. If the status of the buffer pointers
is not as expected or required, a buffer pointer clear should be
performed before writing data or issuing commands.

     All write commands should be performed with all bits in the clock
register set to a "1" (FF hex). This register is used only for
formatting diskettes. For all write operations, the WGATE status flag
indicates when data is actually being written to the diskette.

Sectored or formatted operations

     These operations differ from free-format commands in that the use
of sectors is expected. Sectors are of fixed length, and are located
and read or written automatically. The disk control logic will verify
that the track/sector/side read from the address marks on the disk

match the track/sector/side register contents before transferring any
data. If the address marks do not match the address information
supplied by the user within 6 index pulses, the command will
terminate, BUSY will be reset, and the RNF (record not found) flag
will be set. The RNF flag will remain set until the next command is
issued. The RUN flag, when set, indicates that so far, the sector
being accessed appears to be correct. This flag will reset when any
part of the address mark does not match the expected data, or a
successful completion occurs. Therefore, RUN can change states several
times over single track.

RDS   Read a Sector

     Writing a 40 (hex) to the"command register will cause the
controller to execute a buffered RDS (read sector) command. Writing a
41 (hex) will execute an unbuffered RDS command. Add 4 to either
command to select DPLL data recovery instead of the normal FC method.
Add 6 to either command to select Alternate DPLL recovery instead of
the FC method.

     The RDREQ flag, when set, indicates that the requested sector has
been found, and is now being read into the buffer. RDREQ will reset
after the last byte of the sector is read.

WTS   Write a Sector

     Writing a 80 (hex) to the command register will execute a
buffered WTS (write a sector) command. Add 1 to this command for
unbuffered operation, and add 4 if write pre compensation is desired.

     The WTREQ flag, when set, indicates that the requested sector has
been found, and is now being written from the buffer. WTREQ will reset
after the last byte of the sector is written.

RDT   Read a track

     Writing a 60 (hex) to the command register will initiate an
unformatted buffered disk read. Add 1 to the command for unbuffered
operation. Reading will begin immediately, and will continue until
user cancellation. The data recovery logic will use address and data
marks to align data to byte boundaries. Add 4 to either command to
select DPLL data recovery instead of the normal FC method. Add 6 to
either command to select Alternate DPLL recovery instead of the FC
method.

WTT   Write a track

     Writing an A0 (hex) to the command register will initiate a
buffered write track operation. Add 1 to this command for unbuffered
operation, and add 4 to enable write precompensation.

     The Write Track feature is usually only used for formatting
diskettes, and will most likely be used in the unbuffered mode, since
both data and clock must be supplied on a byte by byte basis. Write
normal data with the clock register set to FF hex. Write special marks
with missing clocks by writing an FB hex to the clock register.

Writing actually begins with the first index pulse after the command is issued, and continues until the next index pulse.

STIN, STOUT   Step In and Step Out

Writing a 10 (hex) or 18 (hex) to the command register will initiate a Step-In or Step-Out operation, respectively. The stepping pulse will be generated immediately, and BUSY will remain set for the duration of the stepping time specified in the STEP register.

TIME   General purpose timer

Writing a 14 (hex) to the command register will initiate a TIME operation. BUSY will remain set for the duration of the time specified in the STEP register. No stepping pulse will be generated.

SPIN   Wait for motor spin-up

Writing a 20 (hex) to the command register will cause BUSY to be set, and stay set for six index pulses. The RNF flag will be set at the end of this operation.

CAN   Cancel or "Can" the current operation

Writing a 0 to the command register will force cancellation of any command in progress, and force BUSY to be reset after at least one round-trip serial control and status transmission and reception.

CLB   Clear buffer pointers

Writing a 1 to the command register will unconditionally reset the buffer pointers. This should be considered a buffer clear operation, although the contents of the buffer are not affected. The BUSY flag will be set for at least one round-trip serial control and status transmission and reception.

Full Track Writing and Formatting Diskettes

Writing full-track data and formatting are very similar. Both will require that you generate the appropriate SYNC bytes, so that the read data recovery logic can align the serial bitstream to byte boundaries. Both descriptions, below, will assume that the spindle motor is on, and up to speed, and that the read/write head is positioned over the track and side to be written.

Track Writes

Full-track writes can be done, either buffered or unbuffered, however, the CLOCK pattern register has no buffer, and writes to this register must be done "one on one".

Write track Buffered

issue "clear buffer" command

```
        write FF hex to clock register
        issue "write track buffered" command
        write FF hex to data register
        wait for first DRQ flag
        write A1 hex to data register
        write FB hex to clock register
        wait for next DRQ flag
        write A1 hex to data register
        wait for next DRQ flag
        write A1 hex to data register
        wait for next DRQ flag
        write FF hex to clock register
        write your first data byte to the data register
          you may now use fully buffered operation.


    Write Track Unbuffered


        write FF hex to clock register
        issue "write track unbuffered" command
        write FF hex to data register
        wait for first DRQ flag
        write A1 hex to data register
        write FB hex to clock register
        wait for next DRQ flag
        write A1 hex to data register
        wait for next DRQ flag
        write A1 hex to data register
        wait for next DRQ flag
        write FF hex to clock register
  loop: write data byte to the data register
        check BUSY flag for completion
        wait for next DRQ flag
        go to loop
```

Formatting a track

     In order to be able to read or write sectored data on a diskette,
the diskette MUST be properly formatted. If, for any reason, marks are
missing or have improper clocks, track, sector, side, or length
information are incorrect, or the CRC bytes are in error, any attempt
to perform a sectored read or write operation will terminate with a
RNF error.

     Formatting a track is simply writing a track with a strictly
specified series of bytes. A given track must be divided into an
integer number of sectors, which are 128, 256, 512, or 1024 bytes
long. Each sector must consist of the following information. All
clocks, are FF hex, where not specified. Data and clock values are in
hexadecimal notation. Fill any left-over bytes in the track with 4E
data.

```
  quan       data/clock       description
  ----       ----------       -----------
   12        00               gap 3*
   3         A1/FB            Marks
             FE               Header mark
             (track)          Track number
             (side)           Side number
             (sector)         Sector number
             (length)         Sector Length (0=128,1=256,2=512,3=1024)
```

```
2        (crc)          CRC bytes
23       4E             gap 2
12       00             gap 2
3        A1/FB          Marks
         FB             Data mark
128,
256,
512, or
1024     00             Data bytes (consistent with length)
2        (crc)          CRC bytes
24       4E             gap 3*
```

* you may reduce the size of gap 3 to increase diskette capacity,
  however the sizes shown are suggested.

Generating the CRC

     The CRC is a sixteen bit value that must be generated serially,
one bit at a time. Think of it as a 16 bit shift register that is
broken in two places. To CRC a byte of data, you must do the following
eight times, (once for each bit) beginning with the MSB or bit 7 of
the input byte.

     1. Take the exclusive OR of the MSB of the input byte and CRC
        bit 15. Call this INBIT.
     2. Shift the entire 16 bit CRC left (toward MSB) 1 bit position,
        shifting a 0 into CRC bit 0.
     3. If INBIT is a 1, toggle CRC bits 0, 5, and 12.

     To Generate a CRC value for a header, or for a data field, you
must first initialize the CRC to all 1's (FFFF hex). Be sure to CRC
all bytes of the header or data field, beginning with the first of the
three A1 marks, and ending with the before the two CRC bytes. Then
output the most significant CRC byte (bits 8-15) and then the least
significant CRC byte (bits 7-0). You may also CRC the two CRC bytes.
If you do, the final CRC value should be 0.

     Shown below is an example of code required to CRC bytes of data.

```
;
; CRC a byte. Assuming byte to CRC in accumulator and cumulative
;             CRC value in CRC (lsb) and CRC+1 (msb).

        CRCBYTE LDX  #8           ; CRC eight bits
                STA  TEMP
        CRCLOOP ASL  TEMP         ; shift bit into carry
                JSR  CRCBIT       ; CRC it
                DEX
                BNE  CRCLOOP
                RTS


;
; CRC a bit. Assuming bit to CRC in carry, and cumulative CRC
;            value in CRC (lsb) and CRC+1 (msb).

        CRCBIT  ROR
                EOR CRC+1         ; MSB contains INBIT
                PHP
                ASL CRC
```

```
              ROL CRC+1          ; shift CRC word
              PLP
              BPL RTS
              LDA CRC            ; toggle bits 0, 5, and 12 if INBIT is 1.
              EOR #$21
              STA CRC
              LDA CRC+1
              EOR #$10
              STA CRC+1
        RTS   RTS
```

2.5.5. F011 Disk Expansion Port Serial Protocol

```
   +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +--
   | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | CLK
 --+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+
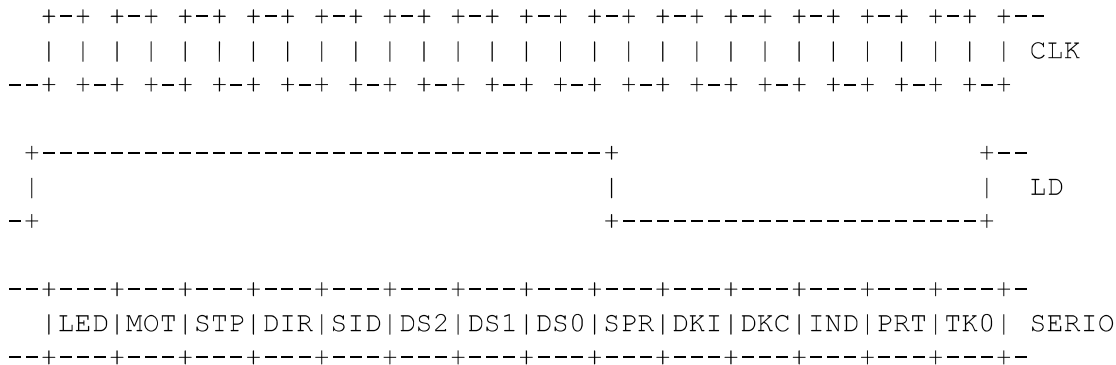

   +-------------------------------+                +--
   |                               |                |  LD
 -+                                +----------------+


 --+---+---+---+---+---+---+---+---+---+---+---+---+---+---+-
   |LED|MOT|STP|DIR|SID|DS2|DS1|DS0|SPR|DKI|DKC|IND|PRT|TK0|  SERIO
 --+---+---+---+---+---+---+---+---+---+---+---+---+---+---+-
```

        Legend:

        Outputs...   Inputs...
        LED   Panel LED On   TK0   Track Zero
        MOT   Spindle Motor On   DKI   Disk Inserted
        STP   Step Pulse   DKC   Disk Changed
        DIR   Step Direction   IND   Index
        SID   Side Select   PRT   Write Protect
        DS2-DS0 Drive Unit Select   SPR   Spare input

     The SERIO pin is bi-directional/and is used for both transmission
of drive control signals, and reception of drive status signals. The
F011 will drive SERIO when LD is high. The selected remote unit must
drive SERIO when LD is low. All SERIO bits are low-true. SERIO will
float high for non-existant drives, making all inputs look false.

     All remote units must clock in serial data on the falling edge of
CLK. The remote units must update their control information on LD
falling if the DS bits match the given unit. All remote units may load
their status inputs when LD is high. Remote units shift out serial
status on the rising edge of CLK. The F011 will not change LD
coincident with CLK, nor will it drive SERIO when LD is changing.

2.5.6. F011 Disk Timing

                  UNBUFFERED WRITE

```
        +-+           +-+
        | |           | |                    CTAK
     ----+ +---------+ +--------------------


          +-----+      +---------------
          |     |      |                      DRQ
     -----------+      +-----+
```

```
                    +-+             +-+   +-+
                    | |             | |   | |         DTAK
         ----------+ +--------+ +---+ +-------
  

                                    +---------
                                    |               LOST
         --------------------------+


                   UNBUFFERED READ

          +-+             +-+   +-+
          | |             | |   | |                   DTAK
       ----+ +--------+ +----+ +------------

          +-----+      +-------------------
          |     |      |                             DRQ
       ----+    +-----+

          +-+
          | |                                         DTAK
       ----------+ +------------------------

                                    +---------
                                    |               LOST
         --------------------------+


                    BUFFERED READ

        +-+         +-+   +-+        +-+       +-+   +-+
        | |         | |   | |        | |       | |   | |     DTAK
     ---+ +-------+ +---+ +-- --+ +-------+ +--+ +---

        +----+      +--------- -------+     +----------
        |    |      |                 |     |               DRQ
     ---+    +----+                    +----+

     ----+    +----+              +----+    +----+
         |    |    |              |    |    |    |          EQ
         +----+    +--------- ---+    +----+    +----

          +-+                     +-+
          | |                     | |                        CTAK
     -------+ +-------------- ------+ +--------------

                                        +-----
                                        |                    LOST
     ----------------------- ----------------+


                    BUFFERED WRITE

        +-+         +-+                      +-+   +-+
        | |         | |                      | |   | |       CTAK
     ---+ +-------+ +----------------- ---+ +--+ +---

          +----+    +-------------- ---+
          |    |    |                  |                      DRQ
     --------+    +-----+              ---+---------
```

```
   ----+    +----+    +----+                     +----
       |  |    |  |    |                          |     EQ
     +----+    +----+    +-------- ----------+


       +-+         +-+   +-+    +-+
       | |         | |   | |    | |                        DTAK
   --------+ +--------+ +--+ +---+ +-- --------------
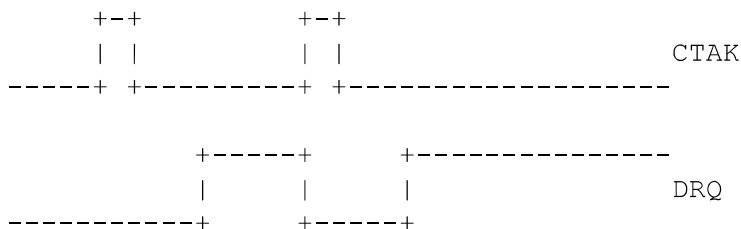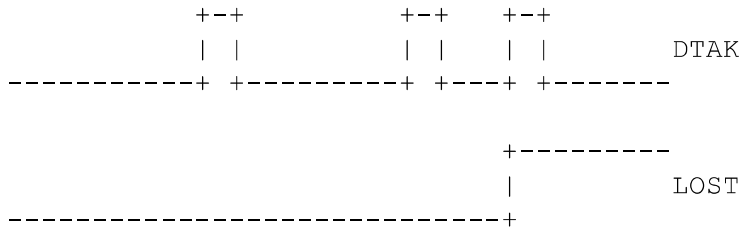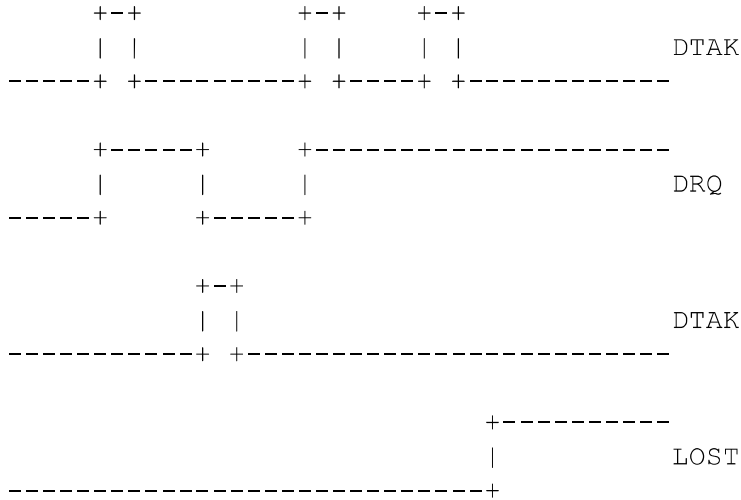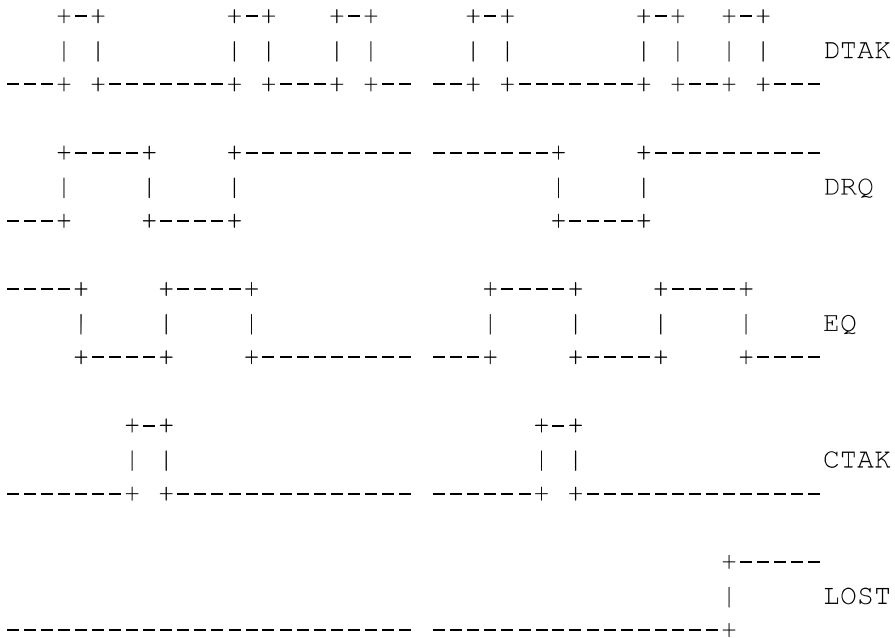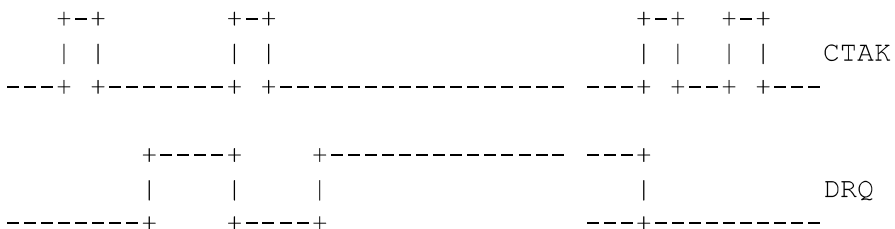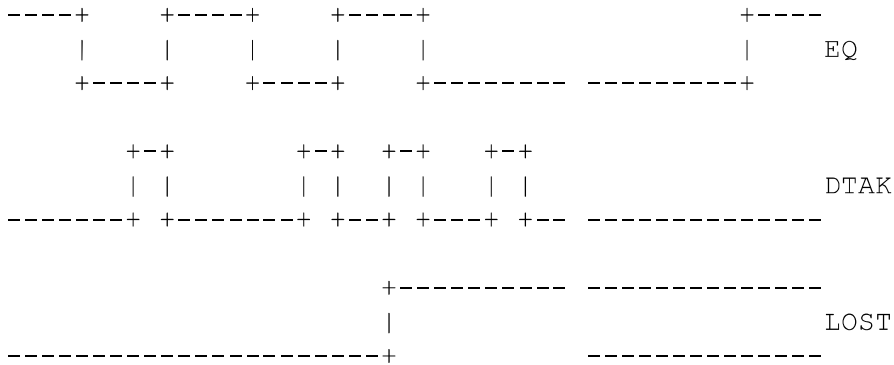

                   +---------- --------------
                   |                                LOST
   --------------------+            --------------
```


2.6. F016 Expansion Drive Controller

2.6.1. Description

     The CSG4101-F016 is a disk expansion interface that is compatible
with the CSG4181-F011B disk controller. With the use of the F016, up
to seven external drives can be added to a base F011B system. Drive 0
is the main unit and is controlled entirely by the F011B. Drives 1
thru 7 are external drives, an each must be connected to the F011B
with a separate F016.

*** NOTE THAT THE C65 DOS SUPPORTS ONLY ONE EXTERNAL F016 EXPANSION DRIVE ***

     CSG4101-F016 Pinout:

     Pin   Name   Active   Dir   Type   Description
     ---   ----   ------   ---   ----   -----------
      1   DS   low   output   drive selected
      2   MOT   low   output   motor on
      3   SIDE   low   output   side select
      4   WPROT   low   input   write protect
      5   TKO   low   input   track 0
      6   INDEX   low   input   index
      7   DR2   low   input   pullup   drive assign dipswitch
      8   DR1   low   input   pullup   drive assign dipswitch
      9   DR0   low   input   pullup   drive assign dipswitch
     10   GND   power
     11   RES   low   input   master reset
     12   LED   low   output   panel LED
     13   DIR   output   stepping direction
     14   STEP   low   output   stepping command
     15   SPARE   input
     16   DSKIN   low   input   disk inserted
     17   SERIO   low   I/O   bidir   serial data
     18   CLK   input   serial data clock
     19   LD   input   shift/load command
     20   VCC   power

Signal descriptions:

RES   The Reset is a low-true input used to reset internal flip-
        flops. The DS (drive selected) output will go false (high)
        when RES is asserted (low).

WPROT   The Write Protect input must indicate, when true, that the
        diskette in the attached drive must not be written to (the
```

drive itself will inhibit writing, as well).

DR   This output, when low, indicates that the attached drive is
          the currently selected drive. This signal will become false
          (high) upon RESet and when another drive is selected.

DSKIN   The Disk In Input must indicate when a diskette is physically
          in the attached drive, and the drive is available for use.

MOT   The Motor On output, when true, turns on the motor of the
          attached disk drive.

LED   The LED output, when true turns on the panel Light-emitting-
          diode of the attached disk drive.

SIDE   The Side select output determines which side of the media is
          to be read or written. It is high (false) for side 0, and low
          (true) for side 1.

STEP   The Step output provides a low-going pulse when a head step
          operation is required, assuming DS is true (low).

DIR   The Direction output indicates to the drives whether the read/
          write head is to step toward track 0 (DIR=high) or away from
          track 0 (DIR=low) when a step pulse is received, assuming DS
          is true (low).

TK0   The Track Zero input must determine when the read/write head
          of the attached drive is positioned over track zero.

INDEX   The Index pulse input must provide a low going pulse for each
          spindle rotation of the attached drive, if it has an index
          sensor. The F016 will latch index pulses until they are sent
          out via the SERIO line. This input must be tied low if the
          attached drive has no index sensor.

SERIO   The Serial I/O line is a bi-directional signal that is used
          to receive control information from the main disk controller,
          and return status information to the main controller, assuming
          the DS output is true (low). It is a driven output when LD and
          DS are low, and an input, otherwise.

LD   The Load Data input tells when to update control information
          shifted over the SERIO line, when to load status information
          for shifting, and when to drive the SERIO line.

CLK   The Clock input is used for shifting control and status
          information.

2.6.2. Expansion Port Timing

                    (used by all F016 chips)

        +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +--
        | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | CLK
     --+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+ +-+

        +----------------------------------+   +--
        | | | LD
      -+   +-------------------+

```
--+---+---+---+---+---+---+---+---+---+---+---+---+---+---+-
  |LED|MOT|STP|DIR|SID|DS2|DS1|DS0|SPR|DKI|DKC|IND|PRT|TK0|  SERIO
--+---+---+---+---+---+---+---+---+---+---+---+---+---+---+-
```

Legend:

Outputs...		Inputs...	
LED	Panel LED On	TK0	Track Zero
MOT	Spindle Motor On	DKI	Disk Inserted
STP	Step Pulse	DKC	Disk Changed
DIR	Step Direction	IND	Index
SID	Side Select	PRT	Write Protect
DS2-DS0	Drive Unit Select	SPR	Spare Input

The SERIO pin is bi-directional, and is used for both transmission of drive control signals, and reception of drive status signals. The F011B will drive SERIO when LD is high. Any selected F016 will drive SERIO when LD is low. All SERIO bits are low-true. SERIO will float high for nonexistant drives, making ail inputs look false.

All F016 chips clock in serial data on the falling edge of CLK. They update their control information on LD falling if the DS bits match the DS0-DS2 switch settings. All F016 chips load their status inputs when LD is high, and shift out serial status on the rising edge of CLK.

2.7. DMAgic DMA CONTROLLER F018 (Preliminary)

2.7.1. F018 DESCRIPTION

DMAGIC is a custom DMA Gate array IC used in the C65. It functions as a DMA controller with a few tricks up its sleeve. Specifically, DMAgic provides the following commands:

* COPY -- Copy a block of memory to another area in memory.

* MIX   -- Perform a boolean Minterm mix of a source block of memory with a destination block of memory.

* SWAP -- Exchange the contents of two blocks of memory.

* FILL -- Fill a block of memory with a source byte.

Special features include:

    * List-based fetching of DMA command sequences.

    * Ability to CHAIN multiple DMA command sequences.

    * Absolute Address access to entire System Memory (8MB).

    * Blocks can be up to 64K bytes long.

    * Windowed Block capability using MODulus function.

    * DMAgic operations yield to VIC video and external DMA accesses.

* DMAgic operations can optionally yield to system interrupts.

        * Interrupted DMAgic operations can be continued/resumed, or
          cancelled.

        * Data ReQuest handshaking support for 10 devices.

        * Independent memory/mapped IO selection for source and destination.

        * Independent memory tranfer DIRection for source and destination.

        * Independent MODulus enable for source and destination.

        * Independent HOLD (fixed pointer) for source and destination.

The DMA controller has 4 registers:

        0   DMA List address low,   Triggers DMA   (write only)
        1   DMA List address high   (write only)
        2   DMA List address bank   (write only)
        3   DMA Status (b7=busy, b0=chained)   (read only)
                        (a read will restart an INTerupted DMA operation)

Note: Minterms & Subcommand will not be implemented until F018A, at
        which time the register map will be reorganized & support for
        the REC added.

        dma_ctlr   = $D700   ;DMA Controller

2.7.2. F018 REGISTERS.

                              F018 DMA CONTROLLER

REG NAME	R #	B7	B6	B5	B4	B3	B2	B1	B0
COMMAND	0	SADA	SADA	SADA	SADA	INT	CHAIN	OPERATION	
CNT LO (COL)	1	C7	C6	C5	C4	C3	C2	C1	C0
CNT HI (ROW)	2	C15	C14	C13	C12	C11	C10	C9	C8
SRC LO (FILL)	3	SA7	SA6	SA5	SA4	SA3	SA2	SA1	SA0
SRC HI	4	SA15	SA14	SA13	SA12	SA11	SA10	SA9	SA8

```
                        |       |       |       |       |       |       |       |       |
SRC BANK    5 |  I/O  |  DIR  |  MOD  | HOLD  | SA19  | SA18  | SA17  | SA16  |
                        |       |       |       |       |       |       |       |       |
                        +-------+-------+-------+-------+-------+-------+-------+-------+


                        +-------+-------+-------+-------+-------+-------+-------+-------+
                        |       |       |       |       |       |       |       |       |
DEST LO     6 |  DA7  |  DA6  |  DA5  |  DA4  |  DA3  |  DA2  |  DA1  |  DA0  |
                        |       |       |       |       |       |       |       |       |
                        +-------+-------+-------+-------+-------+-------+-------+-------+
                        |       |       |       |       |       |       |       |       |
DEST HI     7 |  DA15 |  DA14 |  DA13 |  DA12 |  DA11 |  DA10 |  DA9  |  DA8  |
                        |       |       |       |       |       |       |       |       |
                        +-------+-------+-------+-------+-------+-------+-------+-------+
                        |       |       |       |       |       |       |       |       |
DEST BANK   8 |  I/O  |  DIR  |  MOD  | HOLD  | DA19  | DA18  | DA17  | DA16  |
                        |       |       |       |       |       |       |       |       |
                        +-------+-------+-------+-------+-------+-------+-------+-------+



                        +-------+-------+-------+-------+-------+-------+-------+-------+
                        |       |       |       |       |       |       |       |       |
MOD LO      9 |  M7   |  M6   |  M5   |  M4   |  M3   |  M2   |  M1   |  M0   |
                        |       |       |       |       |       |       |       |       |
                        +-------+-------+-------+-------+-------+-------+-------+-------+
                        |       |       |       |       |       |       |       |       |
MOD HI     10 |  M15  |  M14  |  M13  |  M12  |  M11  |  M10  |  M9   |  M8   |
                        |       |       |       |       |       |       |       |       |
                        +-------+-------+-------+-------+-------+-------+-------+-------+
```

```
        OPERATIONS:         0 0     COPY
                            0 1     MIX   (MINTERMS ACTIVE)
                            1 0     SWAP
                            1 1     FILL  (SRC LO = FILL BYTE)


        PARAMETERS:         INT    0   NO INTERRUPTION
                                   1   IRQ/NMI INTERRUPTION

                            CHAIN  0   LAST COMMAND IN LIST
                                   1   PERFORM NEXT COMMAND


        BOOLEAN MINTERMS:                   DA
                                      0    |   1
                                   +-------+-------+
                                   |       |       |
                            0      | SADA  | SADA  |
                                   |0      |1      |
                            SA     +-------+-------+
                                   |       |       |
                            1      | SADA  | SADA  |
                                   |2      |3      |
                                   +-------+-------+
```

THE ABOVE COMMANDS ARE NOT YET IMPLEMENTED, AND SOME OF THE REGISTER
BITS DEFINED ARE DIFFERENT IN THE PILOT VERSIONS.

2.8. RAM Expansion Controller

2.8.1. Functional Specification

            C65 RAM EXPANSION FUNCTIONAL SPECIFICATION

          *** THIS IS PRELIMINARY AND WILL BE CHANGING ***

     The C65 RAM Expansion Card (REC) provides 1 megabyte of expansion
RAM for the C65 computer. The C65 4510/VIC-III provides 1MB of address
space, but rudimentary banking capability is provided by the REC to
allow several different memory configurations for both the CPU and the
VIC-III via available chip selects.

The REC presumes the following system memory map:

          $00000-$1FFFF   128K internal RAM
          $20000-$3FFFF   128K for internal System ROM
          $40000-$7FFFF   256K reserved for cartridge expansion
          $80000-$FFFFF   512K reserved for RAM expansion

     The REC contains a four-bit write-only register. Data is read
from the four low-order bits of the data bus. Reset forces all of
these bits into the reset (low) state. The four bits are defined as:

        ____CPU bank select
      /____VIC access enable
     //____VIC address range
    ///____VIC Bank select
   ////
   3210   VIC sees:
   ----   --
   x0xx   Internal RAM $00000-$1FFFF
   x100   Expansion RAM bank 0, physical address $C0000-$DFFFF
   x110   Expansion RAM bank 0, physical address $E0000-$FFFFF
   x101   Expansion RAM bank 1, physical address $C0000-$DFFFF
   x111   Expansion RAM bank 1, physical address $E0000-$EFFFF

           CPU sees (note that DMA and VIC-DAT access see this too):
   ----   --------------------
   0xxx   Expansion RAM bank 0
   1xxx   Expansion RAM bank 1

/* Inputs */

PIN 1   = MEMCLK ;   /* System memory clock   */
PIN 2   = !CAS ;   /* Correct timing for CAS signal */
PIN 3   = AEC ;   /* The VIC is in town   */
PIN 4   = B3 ;   /* bit to control CPU accesses */
PIN 5   = A19 ;   /* high order address lines */
PIN 6   = A18 ;
PIN 7   = A17 ;
PIN 8   = A16 ;
PIN 9   = A7 ;
PIN 10 = RW ;
PIN 11 = !SID ;   /* Chip select for SID. Used as a decode */
PIN 13 = B2 ;   /* bits to control VIC accesses */
PIN 14 = B1 ;

```
PIN 23  = B0 ;

* Outputs */

PIN 15  = 'CAS0B ;         /* Cases for the DRAMS */
PIN 16  = !CAS0A ;
PIN 17  = 'CAS1B ;
PIN 18  = !CAS1A ;
PIN 19  = !EXPAND ;        /* Signal to system to allow internal ram out */
PIN 20  = MA8 ;            /* High order Memory address line DRAMS */
PIN 21  = !BRDGOE ;        /* Enable for the Gardei Bridge */
PIN 22  =  EX_LATCH ;      /* Strobe for user write to control latch */


VIC     = !AEC ;
RAST    = !MEMCLK ;
CAST    =  MEMCLK ;


EXVIC   = B0 ;
VICSEL0 = B1 ;
VICSEL1 = B2 ;
CPUBANK = B3 ;


EX_LATCH = CAS & SID & A7 & !RW ;        /* location of control register */
         /* latch data on cas fall to avoid the phi-2 hold time problem */


BRDGOE  = EXPAND & A16 & !VIC ;          /* CPU accessing E bank side */


EXPAND  = !VIC & A19                      /* ram area */
        #  VIC & EXVIC ;                  /* external vie accesses allowed */


MA8     = VIC  & RAST 6  ;A16             /* Ras time, keep upper */
        # VIC  & CAST & VICSEL0           /* Cas time, programable */
        # !VIC & RAST &  A18              /* ras time */
        # !VIC & CAST &  A17 ;            /* cas time */

/* bank 0 drams */

CAS0A   = CAS & EXPAND & ( !VIC & !CPUBANK & !A16 # VIC & !VICSEL1 ) ;
CAS0B   = CAS & EXPAND & ( !VIC & !CPUBANK &  A16 # VIC & !VICSEL1 ) ;

/* bank 1 drams */
CAS1A   = CAS & EXPAND & ( !VIC & CPUBANK & !A16 # VIC & VICSEL1 );
CAS1B   = CAS & EXPAND & ( !VIC & CPUBANK &  A16 # VIC & VICSEL1 );



2.9. 8580 SID REGISTER MAP
```

	7	6	5	4	3	2	1	0	
0	F7	F6	F5	F4	F3	F2	F1	F0	FREQUENCY LO VOICE-1
1	F15	F14	F13	F12	F11	F10	F9	F8	FREQUENCY HI
2	PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0	PULSE WIDTH LO
3					PW11	PW10	PW9	PH8	PULSE WIDTH HI
4	NOISE	PULSE	SAW	TRI	TEST	RING	SYNC	GATE	CONTROL REGISTER
5	ATK3	ATK2	ATK1	ATK0	DCY3	DCY2	DCY1	DCY0	ATTACK / DECAY
6	STN3	STN2	STN1	STN0	RLS3	RLS2	RLS1	RLS0	SUSTAIN / RELEASE
7	F7	F6	F5	F4	F3	F2	F1	F0	FREQUENCY LO VOICE-2
8	F15	F14	F13	F12	F11	F10	F9	F8	FREQUENCY HI
9	PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0	PULSE WIDTH LO
10					PW11	PW10	PW9	PH8	PULSE WIDTH HI

```
11 | NOISE| PULSE|  SAW  |  TRI  | TEST  | RING  | SYNC  | GATE  | CONTROL REGISTER
12 | ATK3 | ATK2 | ATK1  | ATK0  | DCY3  | DCY2  | DCY1  | DCY0  | ATTACK / DECAY
13 | STN3 | STN2 | STN1  | STN0  | RLS3  | RLS2  | RLS1  | RLS0  | SUSTAIN / RELEASE
   +------+------+------+------+------+------+------+------+
14 |  F7  |  F6  |  F5   |  F4   |  F3   |  F2   |  F1   |  F0   | FREQUENCY LO  VOICE-3
15 |  F15 |  F14 |  F13  |  F12  |  F11  |  F10  |  F9   |  F8   | FREQUENCY HI
16 |  PW7 |  PW6 |  PW5  |  PW4  |  PW3  |  PW2  |  PW1  |  PW0  | PULSE WIDTH LO
17 |      |      |       |       | PW11  | PW10  |  PW9  |  PH8  | PULSE WIDTH HI
18 | NOISE| PULSE|  SAW  |  TRI  | TEST  | RING  | SYNC  | GATE  | CONTROL REGISTER
19 | ATK3 | ATK2 | ATK1  | ATK0  | DCY3  | DCY2  | DCY1  | DCY0  | ATTACK / DECAY
20 | STN3 | STN2 | STN1  | STN0  | RLS3  | RLS2  | RLS1  | RLS0  | SUSTAIN / RELEASE
   +------+------+------+------+------+------+------+------+
21 |      |      |       |       |       | FC2   | FC1   | FC0   | FREQUENCY LO  FILTER
22 | FC10 |  FC9 |  FC8  |  FC7  |  FC6  | FC5   | FC4   | FC3   | FREQOENCY HI
23 | RES3 | RES2 | RES1  | RES0  |FILTEX | FILT3 | FILT2 | FILT0 | RESONANCE / FILTER
24 | 3 OFF|  HP  |  BP   |  LP   | VOL3  | VOL2  | VOL1  | VOL0  | MODE / VOLUME
   +------+------+------+------+------+------+------+------+
25 |  PX7 |  PX6 |  PX5  |  PX4  |  PX3  |  PX2  |  PX1  |  PX0  | POT X           MISC.
26 |  PY7 |  PY6 |  PY5  |  PY4  |  PY3  |  PY2  |  PY1  |  PY0  | POT Y
27 |  O7  |  O6  |  O5   |  O4   |  O3   |  O2   |  O1   |  O0   | OSCILLATOR 3
28 |  E7  |  E6  |  E5   |  E4   |  E3   |  E2   |  E1   |  E0   | ENVELOPE 3
   +------+------+------+------+------+------+------+------+
```

Notes:

1. CIA#1 ports PRA6 and PRA7 select which control port POT line
   is routed to SID.

2. While there are 2 SIDs in the C65, the POT lines are still
   routed to SID#1 for C64 compatibility reasons.

3.0. System Software

3.1. BASIC 10.0

                     C64DX BASIC 10.0

3.1.1. INTRODUCTION

    This section lists BASIC 10.0 commands, statements, and functions
in alphabetical order. It gives a complete list of the rules (syntax)
of BASIC 10.0, along with a concise description of each.

COMMAND AND STATEMENT FORMAT

    The commands and statements presented in this section are
governed by consistent format conventions designed to make them as
clear as possible. In most cases, there are several actual examples to
illustrate what the actual command looks like. The following example
shows some of the format conventions that are used in the BASIC
commands:

    EXAMPLE: DLOAD <"program name"|(file_name_var)> [,U#] [,D#]
                     | | | |
                     | | | |
              keyword   argument (if any)   optional arguments

    The parts of the command or statement that the user must type in
exactly as they appear are in capital letters. Words that don't have

to be typed exactly, such as the name of the program, are not
capitalized. When quote marks (" ") appear (usually around a program
or file name), the user should include them in the appropriate place
according, to the format example.

KEYWORDS, also called RESERVED WORDS, appear in uppercase
letters. THESE KEYWORDS MUST BE ENTERED EXACTLY AS THEY APPEAR.
However, many keywords have abbreviations that can also be used.

Keywords are words that are part of the BASIC language that the
computer understands. Keywords are the central part of a command or
statement. They tell the computer what kind of action to take. These
words cannot be used as variable names.

ARGUMENTS (also called parameters) appear in lower case.
Arguments are the parts of a command or statement; they complement
keywords by providing specific information about the command or
statement. For example, a keyword tells the computer to load a
program, while the argument tells the computer which specific program
to load and a second argument specifies which drive the disk
containing the program is in. Arguments include filenames, variables,
line numbers, etc.

SQUARE BRACKETS [] show OPTIONAL arguments. The user selects any
or none of the arguments listed, depending on the requirements.

ANGLE BRACKETS <> indicates that the user MUST choose one of the
arguments listed.

VERTICAL BAR | separates items in a list of arguments when the
choices are limited to those arguments listed, and no other arguments
can be used. Then the vertical bar appears in a list enclosed in
SQUARE BRACKETS, the choices are limited to the items in the list, but
still have the option not to use any arguments.

ELLIPSIS ..., a sequence of three dots, means that an option or
argument can be repeated more than once.

QUOTATION MARKS " " enclose character strings, filenames, and
other expressions. When arguments are enclosed in quotation marks in a
format, the quotation marks must be included in a command file or
statement. Quotation marks are not conventions used to describe
formats; they are required parts of a command or statement.

PARENTHESES () When arguments are enclosed in parentheses in a
format, they must be included in a command or statement. Parentheses
are not conventions used to describe formats; they are required parts
of a command or statement.

VARIABLE refers to any valid BASIC variable name such as X, A$,
or T%.

EXPRESSION means any valid BASIC expression, such as A+B+2 or
.5*(X+3).

3.1.2. ALPHABETICAL LIST OF COMMANDS, FUNCTIONS, and OPERATORS

```
*           Token = AC        multiplication
+           Token = AA        addition
```

```
-            Token = AB        subtraction
/            Token = AD        division
<            Token = B3        less-than
=            Token = B2        equal
>            Token = B1        greater-than
^            Token = AE        exponentiation
(PI)         Token = FF        return value of PI


ABS          Token = B6        absolute function
AND          Token = AF        logical AND operator
APPEND       Token = FE,0E     append file
ASC          Token = C6        string to PETSCII function
ATN          Token = C1        trigonometric arctangent function
AUTO         Token = DC        auto line numbering
BACKGROUND   Token = FE,3B     background color
BACKUP       Token = F6        backup diskette
BANK         Token = FE,02     memory bank selection
BEGIN        Token = FE,18     start logical program block
BEND         Token = FE,19     end logical program block
BLOAD        Token = FE,11     binary load file from diskette
BOOT         Token = FE,1B     load & run ML, or BASIC autoboot
BORDER       Token = FE,3C     border color
BOX          Token = E1        draw graphic box
BSAVE        Token = FE,10     binary save to disk file
BUMP         Token = CE,03     sprite collision function
BVERIFY      Token = FE,28     verify memory to binary file
CATALOG      Token = FE,0C     disk directory
CHANGE       Token = FE,2C     edit program
CHAR         Token = E0        display characters on screen
CHR$         Token = C7        PETSCII to string function
CIRCLE       Token = E2        draw graphic circle
CLOSE        Token = A0        close channel or file
CLR          Token = 9C        clear BASIC variables, etc.
CMD          Token = 9D        set output channel
COLLECT      Token = F3        validate diskette (chkdsk)
COLLISION    Token = FE,17     enable BASIC event
COLOR        Token = E7        set screen colors
CONCAT       Token = FE,13     concatenate two disk files
CONT         Token = 9A        continue BASIC program execution
COPY         Token = F4        copy a disk file
COS          Token = BE        trigonometric cosine function
CUT          Token = E4        cut graphic area
DATA         Token = 83        pre-define BASIC program data
DCLEAR       Token = FE,15     mild reset of disk drive
DCLOSE       Token = FE,0F     close disk channel or file
DEC          Token = D1        decimal function
DEF          Token = 96        define user function
DELETE       Token = F7        delete BASIC lines or disk file
DIM          Token = 86        dimension BASIC array
DIR          Token = EE        disk directory
DISK         Token = FE,40     send disk special command
DLOAD        Token = F0        load BASIC program from disk
DMA          Token = FE,1F     define & execute DMA command
DMA          Token = FE,21     "
DMA          Token = FE,23     "
DMODE        Token = FE,35     set graphic draw mode
DO           Token = EB        start BASIC loop
DOPEN        Token = FE,0D     open channel to disk file
DPAT         Token = FE,36     set graphic draw pattern
DSAVE        Token = EF        save BASIC program to disk
DVERIFY      Token = FE,14     verify BASIC memory to file
```

```
ELLIPSE       Token = FE,30    draw graphic ellipse
ELSE          Token = D5       if/then/else clause
END           Token = 80       end of BASIC program
ENVELOPE      Token = FE,0A    define musical instrument
ERASE         Token = FE,2A    delete disk file
ERRS          Token = D3       BASIC error function
EXIT          Token = FD       exit BASIC loop
EXP           Token = BD       exponentiation function
FAST          Token = FE,25    set system speed to maximum
FILTER        Token = FE,03    set audio filter parameters
FIND          Token = FE,2B    hunt for string in BASIC program
FN            Token = A5       define user function
FOR           Token = 81       start BASIC for/next loop
FOREGROUND    Token = FE,39    set foreground color
FRE           Token = B8       available memory function
GCOPY         Token = FE,32    graphic copy
GENLOCK       Token = FE,38    set video sync mode
GET           Token = Al       receive a byte of input
GO            Token = CB       program branch
GOSUB         Token = 8D       program subroutine call
GOTO          Token = 89       program branch
GRAPHIC       Token = DE       set graphic mode
HEADER        Token = F1       format a diskette
HELP          Token = EA       display BASIC line causing error
HEX$          Token = D2       return hexadecimal string function
HIGHLIGHT     Token = FE,3D    set highlight color
IF            Token = 8B       if/then/else conditional
INPUT         Token = 85       receive input data from keyboard
INPUT#        Token = 84       receive input data from channel (file)
INSTR         Token = D4       locate a string within a string
INT           Token = B5       integer function
JOY           Token = CF       joystick position function
KEY           Token = F9       define or display function key
LEFT$         Token = C8       leftmost substring function
LEN           Token = C3       length of string function
LET           Token = 88       variable assignment
LINE          Token = E5       draw graphic line, input line
LIST          Token = 9B       list BASIC program
LOAD          Token = 93       load program from disk
LOCATE        Token = E6               (currently unimplemented)
LOG           Token = BC       natural log function
LOOP          Token = EC       end of do/loop
LPEN          Token = CE,04    lightpen position function
MID$          Token = CA       substring function
MONITOR       Token = FA       enter ML Monitor mode
MOUSE         Token = FE,3E    set mouse parameters
MOVSPR        Token = FE,06    set sprite position and speed
NEW           Token = A2       clear BASIC program area
NEXT          Token = 82       end of for-next loop
NOT           Token = A8       logical complement function
OFF           Token = FE,24            (subcommand)
ON            Token = 91       multiple branch or subcommand
OPEN          Token = 9F       open I/O channel
OR            Token = B0       logical or function
PAINT         Token = DF       graphic flood-fill
PALETTE       Token = FE,34    set palette color
PASTE         Token = E3       draw graphic area from cut buffer
PEEK          Token = C2       return memory byte function
PEN           Token = FE,33    set graphic pen color
PIC           Token = FE,37    graphic subcommand
PLAY          Token = FE,04    play musical notes from string
```

```
POINTER      Token = CE,0A      address of string var function
POKE         Token = 97         change memory byte
POLYGON      Token = FE,2F      draw graphic polygon
POS          Token = B9         text cursor position function
POT          Token = CE,02      return paddle position
PRINT        Token = 99         display data on text screen
PRINT#       Token = 98         send data to channel (file)
PUDEF        Token = DD         define print-using symbols
QUIT         Token = FE,1E              (currently unimplemented)
RCLR         Token = CD                 (currently unimplemented)
RDOT         Token = D0                 (currently unimplemented)
READ         Token = 87         read program pre-defined program data
RECORD       Token = FE,12      set relative disk file record pointer
REM          Token = 8F         BASIC program comment
RENAME       Token = F5         rename disk file
RENUMBER     Token = F8         renumber BASIC program lines
RESTORE      Token = 8C         set DATA pointer, subcommand
RESUME       Token = D6         resume BASIC program after trap
RETURN       Token = 8E         end of subroutine call
RGR          Token = CC                 (currently unimplemented)
RIGHT$       Token = C9         rightmost substring function
RMOUSE       Token = FE,3F      read mouse position
RND          Token = BB         pseudo random number function
RREG         Token = FE,09      return processor registers after SYS
RSPCOLOR     Token = CE,07      return sprite color function
RSPPOS       Token = CE,05      return sprite position function
RSPRITE      Token = CE,06      return sprite parameter function
RUN          Token = 8A         run BASIC program from memory or disk
RWINDOW      Token = CE,09      return text window parameter function
SAVE         Token = 94         save BASIC program to disk
SCALE        Token = E9                 (currently unimplemented)
SCNCLR       Token = E8         erase text or graphic display
SCRATCH      Token = F2         delete disk file
SCREEN       Token = FE,2E      set parameters or open graphic screen
SET          Token = FE,2D      set system parameter, subcommand
SGN          Token = B4         return sign of number function
SIN          Token = BF         trigonometric sine function
SLEEP        Token = FE,0B      pause BASIC program for time period
SLOW         Token = FE,26      set system speed to minimum
SOUND        Token = DA         perform sound effects
SPC          Token = A6         skip spaces in printed output
SPRCOLOR     Token = FE,08      set multicolor sprite colors
SPRDEF       Token = FE,1D              (currently unimplemented)
SPRITE       Token = FE,07      set sprite parameters
SPRSAV       Token = FE,16      set or copy sprite definition
SQR          Token = BA         Square root function
STEP         Token = A9         for-next step increment
STOP         Token = 90         halt BASIC program
STRS         Token = C4         string representation of number function
SYS          Token = 9E         call ML routine
TAB          Token = A3         tab position in printed output
TAN          Token = C0         trigonometric tangent function
TEMPO        Token = FE,05      set tempo (speed) of music play
THEN         Token = A7         if/then/else clause
TO           Token = A4                 (subcommand)
TRAP         Token = D7         define BASIC error handler
TROFF        Token = D9         BASIC trace mode disable
TRON         Token = D8         BASIC trace mode enable
TYPE         Token = FE,27      display sequential disk file
UNTIL        Token = FC         do/loop conditional
USING        Token = FB         define print output format
```

```
USR            Token = B7          call user ML function
VAL            Token = C5          numeric value of a string function
VERIFY         Token = 95          compare memory to disk file
VIEWPORT       Token = FE,31             (currently unimplemented)
VOL            Token = DB          set audio volume
WAIT           Token = 92          pause program pending memory condition
WHILE          Token = ED          do/loop conditional
WIDTH          Token = FE,1C            (currently unimplemented)
WINDOW         Token = FE,1A       set text screen display window
XOR            Token = CE,08       logical xor function
```

3.1.3. BASIC 10.0 COMMAND AND FUNCTION DESCRIPTION

ABS -- Absolute value function

                          ABS (expression)

The ABSolute value function returns the unsigned value of the numeric
expression.

                X = ABS(1)   Result is X = 1
                X = ABS(-1)   Result is X = 0

AND -- Boolean operator

                     expression AND expression

The AND operator returns a numeric value equal to the logical AND of
two numeric expressions, operating on the binary value of signed
16-bit integers in the range (-32768 to 32767). Numbers outside this
range result in an 'ILLEGAL QUANTITY' error.

                X = 4 AND 12   Result is X=4
                X = 8 AND 12   Result is X=8
                X = 2 AND 12   Result is X=0

In the case of logical comparisons, the numeric value of a true
situation is -1 (equivalent to 65535 or $FFFF hex) and the numeric
value of a false situation is zero.

  X = ("ABC"="ABC") AND ("DEF"="DEF") Result is X=-1 (true)
  X = ("ABC"="ABC") AND ("DEF"="XYZ") Result is X= 0 (false)

APPEND -- Open a disk file and prepare to append data to it

   APPEND# logical_file_number, "filename" [,Ddrive] [<ON|,>Udevice]

Opens filename for writing, and positions the file pointer at the end
of the file. Subsequent PRINT# statements to the logical_file_number
will cause data to be appended to the end of this file. If the file
does not exist, it will be created.

APPEND#1, "filename"
APPEND#1, (file$), ON U(unit)
```

ASC -- PETSCII value function

                          ASC (string)

This  function  returns  the  PETSCII  numeric  value  of  the  first
character of a string.  The PETSCII value of an empty (null) string is
zero. This function is the opposite of the CHR$ function. Refer to the
Table of PETSCII Character Codes.

            X = ASC("ABC")           Result is X=65
            X = ASC("")              Result is X=0


ATN -- Arc tangent function

                          ATN (expression)

This  function  returns  the  angle  whose tangent is the value of the
numeric  expression,  measured  in radians. The result is in the range
of -PI/2 to PI/2 radians.

            X = ATN(45)              Result is X=1.54050257

To  get  the arc tangent of an angle measured in degrees, multiply the
numeric expression by PI/180.


AUTO -- Enable or disable automatic line numbering

                          AUTO [increment]

Turns  on  the automatic line numbering feature which eases the job of
entering  programs  by  typing  the line numbers for the user. As each
program  line  is entered by pressing <RETURN> the next line number is
printed  on  the  screen,  with the cursor in position to begin typing
that  line.  The  increment  parameter refers to the increment between
line  numbers.  AUTO  with  no  increment  given  turns  off auto line
numbering.  AUTO  mode is also turned off automatically when a program
is RUN. This statement is executable only in direct mode.

  AUTO 10   automatically numbers line in increments of ten.
  AUTO 50   automatically numbers line in increments of fifty.
  AUTO      turns off automatic line numbering.


BACKGROUND -- Set the background color of the display

                          BACKGROUND color

Sets  the  screen background color to the given color. The color given
must be in the range (0-15). See the Color Table.


BACKUP -- Backup an entire disk from one drive to another

BACKUP Dsource_drive TO Ddestination_drive [<ON|,>Udevice]

This  command  copies all the files on a diskette to another on a dual
drive  system  only.  It  cannot backup diskettes using CBM serial bus
type  drives, for example. If the destination diskette is unformatted,
BACKUP  will  automatically  format it. BACKUP copies every sector, so

any data already on the destination diskette will be overwritten. To
copy specific files from one drive to another, use the COPY command.

NOTE: This command can only be used with a dual disk drive, such as
the built-in C64DX drive and optional F016-type expansion drive. To
backup diskettes using different drives, such as the built-in drive
and a 1581-type serial bus drive, use a utility program.

```
BACKUP D0 to D1              Copies all files from the disk in
                             drive 0 to the disk in drive 1.

BACKUP D0 TO D1, ON U9       Copies all files from drive 0 to
                             drive 1 in disk drive unit 9.
```


BANK -- Set the memory bank number for PEEK, POKE, SYS, WAIT, LOAD, SAVE

                        BANK memory_bank

              [*** THIS COMMAND MIGHT CHANGE ***]

This command should be used before and BASIC command that has an
address parameter. The address parameters are limited to the range
(0-65535, $0000-$FFFF hex). The BANK command tells the computer which
64K byte memory bank the location you want is in.

The memory_bank parameter is number from 0-255. Refer to the System
memory map to see what is in each bank. A BANK number greater than
127 (i.e., has its most significant bit set) means "use the current
system configuration", and must be used to access an I/O location.
BASIC defaults to BANK 128.

For examples, see PEEK, POKE, etc.

BEGIN/BEND -- Extend an IF clause over more than one line

BEGIN/BEND are used to define a block of code which is considered by
the IF statement to be one statement.

The normal usage of IF/THEN/ELSE would be along the following lines:

        IF boolean THEN statement(s) : ELSE statement(s)

The main restriction is that the entire body of the IF/THEN/ELSE
construct can only occupy one line. BEGIN/BEND allows either the
'THEN' or the 'ELSE' clause to run on for more than one line.

        IF boolean THEN BEGIN: statements...
                              statements...
                              statements...   BEND : ELSE BEGIN
                              statements...
                              statements...   BEND

Remember, however, that this is only a way to extend the body for more
than one line: all other 'IF/THEN' rules apply. For example:

```
100 IF x=1 THEN BEGIN : a=5
110 :  b=6
120 :  c=7
130 BEND : print "ah-ha!"
```

In the above example, "ah-ha!" would be printed ONLY if the expression 'x=1' is TRUE, because the print statement is on the same logical line as the THEN clause.

It is bad practice to GOTO a line in the middle of a BEGIN-BEND block. If BEGIN or BEND is encountered outside of an active IF statement, it is ignored.

BLOAD -- loads a binary disk file into memory

        BLOAD "filename" [,Bbank] [,Paddress] [<ON|,>Udevice]

Used to load a machine language program or other binary data (such as display pictures or sprite data) into memory. If a load address is not given, the load address given in the disk file will be used. If a bank number is not given, the bank given in the last BANK statement will be used. If a load overflows a bank (that is, the load address exceeds 65535 ($FFFF)), an 'OUT OF MEMORY' error is reported. Also see the LOAD command.

        BLOAD "sprites", P(dec("600")), B0

BOOT -- Load and execute a program

    BOOT
    BOOT SYS
    BOOT filename [,Bbank] [,Paddress] [,Ddrive] [<ON|,>Udevice]

BOOT without a filename given causes the computer to look for a BASIC program called AUTOBOOT.C65* on the indicated diskette, LOAD it and RUN it (just like RUN "AUTOBOOT.C65*").

BOOT with a filename given will cause the executable binary file to be BLOADed and executed beginning at the load address. If a load address is not given, the file will be loaded and execution begun at the address stored on disk.

BOOT SYS is a special command that copies the "home" sector (the very track and sector) of the C64DX built-in drive into memory at address $400 to $5FF (one physical sector, 512 bytes) and perform a machine language JSR (Jump SubRoutine) to it. It has the same function as turning on your C64DX while holding down the ALT key. It is used to boot an alternate operating system from either a CBM 3.5" diskette or an MSDOS (720K) diskette. If used in a BASIC program, and it fails, the system can be corrupted. BOOT SYS does *not* use the normal DOS to access the disk.

        BOOT   Loads & runs BASIC program called
                        AUTOBOOT.C65* on system disk.

        BOOT U9   Loads & runs BASIC program called
                        AUTOBOOT.C65* on disk unit 9.

        BOOT "ml"   Load & executes machine language
                        program called ML, starting at address
                        stored on disk.

BORDER -- Set the exterior border color of the display

                            BORDER color

Sets the screen border color to given color. The color must be in the
range (0-15). See the Color Table.

BOX -- Draw a 4-sided graphical shape

                BOX x0,y0, x1,y0, x0,y1, x1,y1 [,solid]

Requires two line segments to be specified, the order of which
determines the shape drawn. The shape is drawn in the currently
specified PEN color, on the currently specified SCREEN. The above
command will draw the following shape:

        |0,<=0 +--------------------+ |1,<=0
              | |
              | |
        |0,<=1 +--------------------+ |1,<=1

But if the order of the coordinates were given as:

                BOX x0,y0, x1,y0, x1,y1, x0,y1

a "bowtie" shape would be drawn. See the sample program at SCREEN.

BSAVE -- Save an area of memory in binary disk file

    BSAVE "[@]filename", Pstart_adr TO Pend_adr [,Bbank] [,Ddrive]
          [<ON|,>Udevice]

BSAVE copies an area of memory into a binary disk file called
"filename", starting at start_adr and ending at end_adr-1 (i.e.
end_adr must be one more than actual last address saved). If a bank
number is not given, the bank given in the last BANK statement will
be used. end_adr must be greater than start_adr, and area to be saved
must be limited to the indicated memory bank. You cannot save data
from more than one bank at a time. start_adr is saved on disk as the
load address. If filename already exists on the designated diskette,
memory is NOT saved and a 'FILE EXISTS' error is reported. Preceding
the filename with an '@'-sign will allow you to overwrite an existing
file, but see the cautions at DSAVE.

      BSAVE "sprites", P(dec("600")) TO P(dec("800")), B0

BUMP -- Sprite collision function

                            BUMP (type)

This function return a numeric summary of sprite collisions
accumulated since the last time the BUMP function was used.

You can use the COLLISION command to set up a special routine in your
program to receive control whenever a sprite BUMPs into something, but
a particular COLLISION does not have to be enabled to use BUMP. See
the COLLISION command.

To evaluate sprite collisions, where a BIT position (0-7) in the
numeric result corresponds to a sprite number (0-7):

```
     BIT position:          7 6 5 4 3 2 1 0
                            | | | | | | | |
     BUMP value in binary:  0 0 0 0 0 1 0 1   = 5 decimal
```

BUMP(1) returns a value representing sprite-to-sprite collisions.
BUMP(2) returns a value representing sprite-to-data collisions.

```
     X = BUMP(1)      Result is X=3 if sprites 0 & 1 collided,
                      as shown above. (binary 101 = 5 decimal).
```

Note that more than one collision can be recorded, in which case you
should evaluate a sprite's position using the RSPPOS function to
figure out which sprite collided with what. BUMP is reset to zero
after each use.

BVERIFY -- Compare a binary disk file to an area of memory

  BVERIFY "filename" [,Paddress] [,Bbank] [,Ddrive] [<ON|,>Udevice]

BVERIFY compares a binary disk file called "filename" to an area of
memory. In direct mode, if the areas contain the same data the message
"OK" is displayed, and if the data differs the message 'VERIFY ERROR'
is displayed.

In program mode, an error is generated if a mismatch is found
otherwise the program continues normally. The comparison starts with
the address given, else it starts at the address stored on disk. The
comparison ends when the last byte is read from the disk file.

If a bank number is not given, the bank given in the last BANK
statement will be used. The ending address is determined by the
length of the disk file. The comparison halts on the first mismatch or
at the end of the file. The area to be compared must be confined to
the indicated memory bank.

                  BVERIFY "sprites", P(dec("600")), B0

CATALOG -- see DIR (DIRECTORY) command

CHANGE -- Find text in a BASIC program and change it.

              CHANGE :string1: TO :string2: [,line_range]
              CHANGE "string1" TO "string2" [,line_range]

This is a direct (edit) mode command. CHANGE looks for all occurrences
of string1 in the program, displays each line containing string1 with
the target string highlighted, and prompts the user for one of the
following:

        Y<RETURN>   Yes, change it and look for more
        N<RETURN>   No, don't change it, but look for more
        *<RETURN>   Yes, change all occurrences from here on
        <RETURN>   Exit command now, don't change anything

Any character can be used for the string delimiter, but there are side

effects: see comments at FIND command. If the line number range is not
given (see LIST for description of range parameter), the entire
program is searched.

CHAR -- Draw a character string on a graphic screen

   CHAR column, row, height, width, direction, "string" [,charsetadr]

                   [*** THIS IS SUBJECT TO CHANGE ***]

CHAR displays text on a graphic screen at a given location. The
character height, width, and direction are programmable. The
parameters are defined as:

        column: Character position:
                               For 320 wide screens, 0-39
                               For 640 wide screens, 0-79
        row: Pixel line:
                               For 200 line screens, 0-199
                               For 400 line screens, 0-399
        height: Multiple of 8-bit character height:
                               1= 8 pixels high, 2= 16 pixels, etc.
        width: Multiple of 8-bit character width:
                               1= 8 pixels high, 2- 16 pixels, etc.
        direction: Bit mask: B0= up
                                        B1= right
                                        B2= down
                                        B3= left

The string can consist of any printable character, as defined by the
VIC character set. Non-text characters are ignored. If the address
of the character set is not given, the upper/lower ROM character set
is used ($29800).

               CHAR 18,96, 1,1,2, "C65D", DEC("9000")

The above example will draw the characters "C65D" in the center of a
320x200 pixel screen using the system's uppercase/graphic character
set.

CHR$ -- Character string function

                          CHR$ (value)

This function returns a string of one character having the PETSCII
value specified. This function is the opposite of the ASC function.
It's often used in PRINT strings to output data that is not visible,
such as control codes and escape sequences. Refer to the Table of
PETSCII Character Codes.

     PRINT CHR$(27)"Q";   CHR$(27) is the escape character.
                            This statement performs the
                            clear-to-end-of-line escape function.

CIRCLE -- Draw a circle on a graphic screen

              CIRCLE x_center, y_center, radius [,solid]

The CIRCLE command will draw a circle with the given radius centered
at (x_center,y_center) on the current graphic screen. The circle will
be filled (i.e., a disc) if SOLID is non-zero.

                          CIRCLE 160,100,50

The above example will draw a circle in the center of a 320x200 pixel
screen (160,100) having a radius of 50 pixels. The aspect ratio of the
screen may cause it to appear as an ellipse, however. See also the
ELLIPSE command.

CLOSE -- Close a logical I/O channel

                     CLOSE logical_channel_number

This command closes the input/output channel associated with the given
logical_channel_number, established by an OPEN statement. In the case
of buffered output (such as the serial bus or RS232) any data in the
device's buffer will be transmitted before the channel is closed.
Refer to specific I/O operations for details.

The logical_channel_number is required; to close all channels on a
given device, use the DCLOSE command. Note that RUN, NEW, and CLR
commands will initialize the logical channel tables but will not
actually close any channels.

CLR -- Clear program variables

                                 CLR

This statement initializes BASIC's variable list, setting all numeric
variables to zero and string variables to null. It also initializes
the DATA pointer, BASIC runtime stack pointer (i.e., clears all
GOSUBs, DO/LOOPs, FOR/NEXT loops, etc.), and clears any user functions
(DEF FNx). Any OPEN channels are forgotten (but a CLOSE is not
performed; don't use if there are any open disk output files). A CLeaR
is automatically performed by a RUN or a NEW command.

CMD -- Set default output channel

                   CMD logical_channel_number [,string]

CMD changes the default output device, normally the screen, to that
specified. The logical_channel_number can be any previously OPENed
write channel, such as one to a disk file, printer, or RS232.

When redirected via CMD, all output which normally would go to the
screen (such as PRINT commands, LIST output, DIRECTORY lists, etc.)
is sent to another device or file.

The redirection is terminated by CLOSE-ing the CMD channel or
executing a PRINT# to the CMD channel. Some output devices require a
PRINT# to be performed before the CMD channel is closed, such as
printers, to cause the device's buffer to be flushed (i.e.,
displayed).

Any system error will redirect output back to the system default,
normally the screen, but will not flush nor close the output channel.

If the optional string is given, it is output immediately after the
CMD device is established. This feature is normally used to set up
printers (eg., set printer modes via escape codes) or to identify the
output (eg., title printouts).

```
        OPEN 4,4        OPENS device #4, which is the printer.
        CMD 4           All normal output now goes to the printer.
        LIST            The LISTing goes to the printer.
        PRINT#4         Set output back to the screen.
        CLOSE 4         Close the printer channel.
```

COLLECT -- Check (validate) disk, delete bad files and free lost sectors

                    COLLECT [Ddrive] [<ON|,>Udevice]

Refer to the DOS 'V'alidate command. This command will cause the DOS
to recalculate the Block Availability Map (BAM) of the diskette in the
indicated drive, allocating only those sectors being used by valid,
properly closed files. All other sectors are marked as "free" and
improper files are automatically deleted.

Note: COLLECT should be used with extreme care, and MUST NOT be used
on diskettes with special boot sectors or direct access (eg., random)
files. In any case, be sure the diskette has been BACKUP-ed first.

COLLISION -- Setup subroutine to handle special events

                    COLLISION type [,linenumber]

                    [*** THIS MIGHT CHANGE ***]

COLLISION is used to handle "interrupt" situations in BASIC, such as
sprites bumping into things or lightpen triggers. When the specified
situation occurs, BASIC will finish processing the currently executing
instruction and perform an automatic GOSUB to the linenumber given.

When the subroutine terminates (it must end with a RETURN) BASIC will
resume processing where it left off. Interrupt handling continues
until a COLLISION of the same type but without any linenumber is
specified. More than one type interrupt may be enabled at the same
time, but only one interrupt can be handled at a time (i.e., no
recursion and no nesting of interrupts). The type interrupt can be:

                1 = Sprite to sprite collision
                2 = Sprite to display data collision
                3 = Light pen

Note that what caused an interrupt may continue causing interrupts for
some time unless the situation is altered or the interrupt is
disabled. This is especially true for BASIC, which is slow to respond
to interrupts. Use the BUMP and RSPPOS functions to evaluate the
results of sprite collisions, and the LPEN function to evaluate the
position of a light pen.

```
        10 COLLISION 1,90
        20 SPRITE 1,1 : MOVSPR 1,100,100 : MOVSPR 1,0#5
        30 SPRITE 2,1 : MOVSPR 2,100,150 : MOVSPR 2,180#5
        40 DO : PRINT : LOOP
```

```
         50 END
         90 PRINT"BUMP! ";:RETURN
```

In this example, sprite-to-sprite collisions are enabled (line 10),
and two sprites are turned on, positioned, and made to move (lines
20 & 30). One sprite moves up and the other moves down while the
program does nothing other than print blank lines to the screen (line
40). When the sprite collide, the subroutine at line 90 is called, it
prints "BUMP!", and the computer goes back to printing blank lines.

COLOR -- Enable or disable screen color (character attribute) control

                         COLOR <ON|OFF>

COLOR turns on or turns off the screen editor's attribute handler.
When colors are turned off, whatever character attributes are being
currently displayed (text color, underline, flash, etc.) are "stuck".
The main purpose for doing this is to speed up screen handling
(writing to the screen or scrolling the screen) about two times, since
the screen editor no longer has to manipulate the attributes. Note
that only FOREGROUND colors (and special VIC attributes) are affected.

To change screen colors, use the following commands:

        FOREGROUND color#   Set Foreground color (text)
        HIGHLIGHT color#   Set Highlight color (text)
        BACKGROUND color#   Set VIC background color
        BORDER color#   Set VIC border color

CONCAT -- Concatenate (merge) two sequential disk files

    CONCAT "file1"[,Ddrive1] TO "file2"[,Ddrive2] [<ON|,>Udevice]

CONCAT merges two SEQuential files, appending the contents of "file1"
to "file2". Upon completion, "file2" contains the data of both files,
and "file1" is unchanged. Both files must exist on drives of the the
same unit, and pattern matching is not allowed.

Some disk drives handle CONCAT differently; refer to the DOS manual
for specific details.

CONT -- Continue program execution

                              CONT

CONTinue is used to re-start a BASIC program that was halted by a STOP
or END statement, or interrupted by the <STOP> key. The program will
resume at the statement following the STOP or END instruction, or at
the statement after the one that was interrupted by the <STOP> key.
CONT is typically used during program debugging. You can look at and
alter variables while the program is halted.

Programs halted as a result of an untrapped error condition cannot be
CONTinued. Programs that have been edited in any way cannot be
restarted. Any error condition that occurs since the program was
halted will prevent it from being restarted. Programs that cannot be
restarted via CONT can be restarted with a GOTO, as long as you don't
need to resume execution in the middle of a line of commands and you

recall where the halt occurred.

Note that the <STOP> key can interrupt some commands in mid-execution, such as file I/O, drawing commands, etc. In such cases, programs may not run correctly after a CONTinue.

COPY -- Copy disk files

        COPY ["file1"][,Dd1] TO ["file2"][,Dd2] [<ON|,>Udevice]

COPYs a disk file to another disk file. On single drive units, the filenames must be different. On dual drive units, copying can be done between two drives on the same unit, and the filenames can be the same or different. Pattern matching can be used. Copying files from one unit to a different unit cannot be done: use a copy utility program in such cases. Only legal type files can be copied; direct access data, boot sectors, and partitions cannot be copied.

Refer to the DOS manual for your disk drive specific details.

        COPY "file1" TO (F2$)   Copies "file1" to another file
                                       whose name is in F2$ on the
                                       same drive. Names must differ.

        COPY "file1",D0 TO D1,U9   Copies "file1" from unit 9
                                       drive-0 to unit 9 drive-1.

        COPY D0 TO D1   Copies all files from drive-0
                                       to drive-1 on the same unit.

        COPY "???.src",D0 TO "*",D1   Copies all files on drive-0
                                       matching the pattern to a file
                                       of the same name on drive-1.

COS -- Cosine function

                        COS (expression)

This function returns the cosine of X, where X is an angle measured in radians. The result is in the range -1 to 1.

         X = COS (pi)   Result is X=-1

To get the cosine of an angle measured in degrees, multiply the numeric expression by pi/180.

CUT -- Cut a graphic area into a temporary structure

                        CUT x,y,dx,dy

                   [*** NOT YET IMPLEMENTED ***]

DATA -- Define program constant data to be accessed by READ command

                   DATA [list of constants]

DATA statements store lists of data that will be accessed during

program execution by a READ statement. The DATA statement can appear
anywhere in the program, and it is never executed. BASIC keeps a
pointer to the earliest un-READ DATA statement, and data is read
sequentially from first item in a DATA statement to the last item,
from the earliest DATA statement in the program to the last DATA
statement in the program.

The list of constants can contain both numeric data (integer or
floating point) and string data, but cannot contain expressions which
must be evaluated (such as 1+2, DEC("1234"), or CHR$(13)). Items are
separated by commas. String data need not be enclosed in quotes unless
it contains certain characters, such as spaces, commas, colons,
graphic characters, or control codes. If two commands have nothing
between them, the data will be READ as 0 if numeric or a null string.

The RESTORE command allows you to position BASIC's data pointer to a
specific line number. If the program tries to read more DATA than
exists in the program, an 'OUT OF DATA' error results. If a READ
statement's variable type does not agree with the DATA being read, a
'TYPE MISMATCH' error results.

        DATA 100, 200, FRED, "HELLO, MOM", , 3.14, ABC123, -1.7E-9

DCLEAR -- Clear all open channels on disk drive

                DCLEAR [Ddrive] [<ON|,>Udevice]

DCLEAR sends the indicated disk drive an 'I'nitialize command. This
clears all open channels, closes all open files, and causes the DOS to
re-read the diskette's Block Allocation Map (BAM). Note that DCLEAR
DOES NOT close open channels on the computer's side (see the DCLOSE
command). There are some other side affects caused by this command
with different types of drives -- refer the DOS manual for your disk
drive for specific details.

DCLOSE -- Close a disk file, or close all channels on a device

                DCLOSE [#logical_file_number] [<ON|,>Udevice]

DCLOSE is intended to close a file opened with the DOPEN command.
Specific files can be closed by specifying a logical_file_number, or
all files on a particular drive can be closed by not specifying a
particular logical_file_number.

It is possible to close channels on non-disk devices with this command
by specifying only the device number.

        DCLOSE#1   Closes the file associated with logical
                        logical file number 1.
        DCLOSE   Closes all files currently open on the
                        default system drive.
        DCLOSE U(U2)   Closes all channels open to device U2.

DEC -- Decimal value function

                DEC (hex_string)

This function return the decimal value of a string representing a

hexadecimal number in the range "0000" to "FFFF". The result is in the range 0-65535. If the string contains a non-hexadecimal digit or is more than four (4) characters in length an 'ILLEGAL QUANTITY' error is reported.

```
        VIC = DEC("D000")        Result is VIC=53248,
                                 the address of the VIC chip
```

DEF FN -- Define function

```
        DEF FN name(numeric_variable) = numeric_expression
```

Define a user-written numeric function. The DEF FNx statement must be executed before the function can be used. Once a function has been defined, it can be used like any other numeric variable. The function name is the letters FN followed by any legal floating point (non-integer) variable name. A function can be defined only in a program.

The numeric_variable is a "dummy" variable. It names the variable the numeric_expression which will be replaced when the function is used. It's not required to be used in the numeric_expression, and its value won't be changed by the function call.

The numeric_expression performs the calculations of the function. It is any legal numeric expression that fits on one line. Variables used in the expression have their value at the time the function is used.

Functions can be used only by the program which defines them. If one program chains to another program, the first program's functions cannot be used (usually a 'SYNTAX ERROR' results). Similarly, if the program is moved in any way after the function is defined, the function cannot be used.

```
10 DEF FN R(MAX) = INT(RND(0)*MAX)+1
20 INPUT "MAXIMUM"; MAX
30 PRINT FN R(MAX)
```

In this example, we've defined a function which will return a pseudo random number between 1 and whatever MAX is. Instead of using the expression INT(RND(0)*MAX)+1 every time a random number is needed, we can now use FN R(MAX). When we use FN R(x), the value of 'x' will be substituted everywhere MAX is used in the function definition.

```
10 DEF FN I(X) = X+1
20 DEF FN L(Z) = LEN(A$)
30 DEF FN AVG(N) = (TOT*CNT+N)/(CNT+1)
```

DELETE -- Delete lines of BASIC program, or
         Delete disk files

```
        DELETE [startline] [-[endline]]
        DELETE "filespec" [,Ddrive] [<ON|,>Udevice] [,R]
```

There are two forms of DELETE. The first form is used in direct mode to remove lines from a BASIC program:

```
   DELETE 75      Deletes line 75
   DELETE 10-50   Deletes line 10 through 50 inclusive.
```

```
    DELETE -50      Deletes all lines from the beginning of
                    the program up to and including line 50.
    DELETE 75-      Deletes all lines from 75 to the end of
                    the program.
```

The second form is used in program or direct mode to delete a disk
file. See the SCRATCH command.

```
    DELETE "myfile"   Deletes the file MYFILE on the system drive.
```


DIM -- Declare array dimensions

```
        DIM variable(subscripts) [,variable(subscripts)]...
```

Before arrays of variables can be used, the program must first execute
a DIM statement to establish DIMensions of that array (unless there
are 11 or fewer elements in the array). The statement DIM is followed
by the name of the array, which may be any legal variable name. Then,
enclosed in parentheses, put the number (or numeric variable) of
elements in each dimension. An array with more than one dimension is
called a matrix. Any number of dimensions may be used, but keep in
mind that the whole list of variables being created takes up space in
memory, and it is easy to run out of memory if too many are used. To
figure the number of variables created with each DIM, multiply the
total number of elements in each dimension of the array. Note: each
array starts with element 0, and integer arrays take up 2/5ths of the
space of floating point arrays.

More than one array can be dimensioned in a DIM statement by
separating the arrays by commas. If the program executes a DIM
statement for any array more than once, the message 'REDIM'D ARRAY' is
reported. It is good programming practice to place DIM statements
near the beginning of the program.

```
        10 DIM A$(40),B7(15),CC%(4,4,4)
                   |       |          |
           41 elements   16 elements  125 elements
```


DIRECTORY -- List the files of a diskette
DIR

```
        DIRECTORY ["filespec"] [,R] [,Ddrive] [<ON|,>Udevice]
```

A directory is a list of the names of the files that are on a
diskette. The directory listing consists of the name of the diskette,
the names, sizes, and filetypes of all the files on a diskette, and
the remaining free space on the diskette. The filespec is used to
specify a pattern match string to view selected files. Not all disk
drives support the same options or filespecs; refer to your DOS manual
for details. The C64DX allows you to print DIR listings without having
to 'load' the directory; see example below.

The commands DIR, DIRECTORY, and CATALOG have the exact same function.
They can be used in direct or program mode.

```
    DIRECTORY               List all files on the diskette
                            in the default system drive
    DIR "*.src", U9         Lists the all the files ending with
                            ".src" on unit 9.
```

```
    DIR "*,=p",R                    List all the deleted but recoverable
                                    PRG-type files on the system drive.

  OPEN 4,4:CMD 4:DIR:CLOSE 4    Print DIR listing to printer unit 4.
```

The following program can be used to load the directory into variables
for use within a program. In this case, the filename is simply printed
to the screen:

```
    10 OPEN 1,8,0,"$0:*,P,R"                open dir as a file
    20 : IF DS THEN PRINT DS$: GOTO 100    abort if error
    30 GET#1,X$,X$                          trash load address
    40 DO                                   read each line
    50 : GET#1,X$,X$: IF ST THEN EXIT      trash links, check EOF
    60 : GET#1,BL$,BH$                      get file size
    70 : LINE INPUT#1, F$                   get filename & type
    80 : PRINT LEFT$(F$,18)                 print filename
    90 LOOP                                 loop until EOF
    100 CLOSE 1                             close dir
```

DISK -- Send a disk command

                DISK "command_string" [<ON|,>Udevice]

The DISK command is used to send special commands to the DOS via the
disk drive's command channel. The DISK command is analogous to the
following BASIC code:

                OPEN 1,n,15: PRINT#1,"command_string": CLOSE 1

Not all disk drives understand the same commands. Refer to your DOS
manual for commands and command syntax for your drive. Note that the
drive number, if any, must be included in the command_string.

```
    DISK "U0>10"          Renumber system drive to 10.
    DISR "U0>V"+chr$[0)    Turn off write verify
    DISK "S0:file",U(n)    Scratch "file" on unit n
```

DLOAD -- Load a BASIC program file from disk

                DLOAD "filename" [,Ddrive] [<ON|,>Udevice]

This command copies a BASIC program from disk into the BASIC program
area of the computer. It can then be edited, DSAVEd, or RUN.

Used in program mode, it overlays the current program in memory and
begin execution automatically at the first line of the new program.
Variable definitions will be left intact, but any open data files and
the disk command channel will be automatically closed. This is called
CHAINING.

See also RUN. Use BLOAD to load binary or machine language data.

```
    DLOAD "myprogram"       Searches the default system disk drive
                            for the BASIC program "myprogram",
                            loads it, and relinks it.
    DLOAD (F$),U9           LOADs a program whose name is in F$
                            from disk unit 9.
```

```
DMA -- Perform a DMA operation

DMA command [,length,source(l/h/b),dest(l/h/b),subcmd,mod(l/h) [,...]]

                [*** THIS COMMAND IS SUBJECT TO CHANGE ***]

The DMA command defines and executes a Direct Memory Access operation.
The parameters are used to construct a DMA list,  which is then passed
to  the   DMA   processor  for  execution. Refer  to  the  DMA  chip
specification for details.  Chained DMA commands are not allowed,  but
multiple  DMA  commands  can  be given and the DMA handler will set up
and execute each one, one at a time. Refer to the system memory map to
find out where things are.

Because  this  command  directly  accesses system memory, extreme care
should  be  taken in its use.  Changing the wrong memory locations can
crash the computer (press the reset button to reboot).

 DMA 3, 2000, ASC("+"), 0, DEC("800"), 0        Fill screen with '+'
 DMA 0, 2000, DEC("800"), 0, DEC("8000"), 1     Copy screen to $18000


DMODE -- Set graphic display mode

        DMODE jam, comp, inverse, stencil, style, thickness

            [*** THIS COMMAND IS SUBJECT TO CHANGE ***]

                    jam                 0-1
                    complement          0-1
                    inverse             0-1
                    stencil             0-1
                    style               0-3
                    thickness           1-8


DO/LOOP/WHILE/UNTIL/EXIT -- Program loop definition and control

      DO   [UNTIL boolean_expression | WHILE boolean_expression]
              .
              .           statements [EXIT]
              .
      LOOP [UNTIL boolean_expression | WHILE boolean_expression]

Performs  the  statements  between  the  DO  statement  and  the  LOOP
statement.  If  no  UNTIL  or WHILE modifies either the DO or the LOOP
statement,  execution  of  the  intervening  statements  continues
indefinitely.  If an EXIT statement is encountered in the body of a DO
loop,  execution  is  transferred to the first statement following the
nearest  LOOP  statement.  Do loops may be nested, following the rules
defined for FOR-NEXT loops. If the UNTIL parameter is used the program
continues  looping  until  the boolean argument is satisfied  (becomes
true).  The  WHILE  parameter  is  basically the opposite of the UNTIL
parameter:  the  program  continues  looping  as  long  as the boolean
argument is TRUE. An example of a boolean argument is A=1, or G>65.

  DO UNTIL X=0 or X=1                     This loop will continue
        : statements                      until X=0 or X=1. If
  LOOP                                     X=0 or 1 at beginning
                                          the loop won't execute.
```

```
  10 A$="": DO GETKEY A$: LOOP UNTIL A$="Q"   This will loop until
                                              the user types 'Q'


  10 DOPEN#1,"FILE"                           This program will
  20 C=0                                      count the number of
  30 DO: LINEINPUT#1,A$: C=C+1: LOOP UNTIL ST    lines in FILE
  40 DCLOSE#1
  50 PRINT"FILE CONTAINS";C;" LINES."
```

DOPEN -- Open a disk file

DOPEN#lf,"filename[,<S|P>]"[,L[reclen]] [,W] [,Ddrive] [<ON|,>Udevice]

This command OPENs a file on disk for reading or writing. If is the
logical file number, which you will use in PRINT#, INPUT#, GET#,
RECORD#, and DCLOSE# commands to reference the channel to your file.
The filename is required. The defaults are to OPEN a SEQuential file
for Reading, in which case the file must exist or a 'FILE NOT FOUND'
error results. To create an file and write to it, use the 'W'rite
option. 'FILE EXISTS' error is report if an output file already
exists. To read or write a RELative file, use the 'L'ength option. The
'reclen' record length is required only when creating a relative file.
For more information regarding Relative files, see the RECORD command
and refer to your DOS manual. See also APPEND.

See the OPEN command for a discussion about channel and device
numbers.

```
DOPEN#1,"readfile"      Opens sequential READFILE for reading.
DOPEN#1,"writefile",W   Creates & opens seq WRITEFILE for writing.
DOPEN#1,"file,P",U(u)   Opens a PRoGram type file for reading on unit U
DOPEN#1,(rf$),L         Open existing relative file whose name's in RF$
DOPEN#a,"rel",L80       Create a relative file with record length of 80
```


DPAT -- Set graphic draw pattern

         DPAT type [, # bytes, byte1, byte2, byte3, byte4]

            [*** THIS COMMAND IS SUBJECT TO CHANGE ***]

                 type   0-63
                 # bytes   1-4
                 byte1   0-255
                 byte2   0-255
                 byte3   0-255
                 byte4   0-255

DSAVE -- Save a BASIC program into a disk file

         DSAVE "[@]filename" [,Ddrive] [<ON|,>Udevice]

This command copies a BASIC program in the computer's BASIC memory
area into a PRoGram-type disk file. If the file already exists, the
program is NOT stored and the error message 'FILE EXISTS' is reported.
If the filename is preceded with an '@', then if the file exists it
will be replaced by the program in memory. Because of some problems
with the 'save-with-replace' option on older disk drives, using this

option is not recommended if you do not know what disk drive is being
used. Use the DVERIFY to compare the program in memory with a program
on disk.

To save a binary program, use the BSAVE command.

     DSAVE "myprogram"   Creates the PRG-type file MYPROGRAM
                                   on the default system disk and copies
                                   the BASIC program in memory into it.

     DSAVE "@myprogram"   Replaces the PRG-type file MYPROGRAM
                                   with a new version of MYPROGRAM. If
                                   MYPROGRAM doesn't exist, it's created.

     DSAVE (F$),U9   Saves a program whose name is in F$
                                   on disk unit 9.

DVERIFY -- Compare a program in memory with one on disk

            DVERIFY "filename" [,Ddrive] [<ON|,>Udevice]

This command is just like a DLOAD, but instead of LOADing the BASIC
program file into computer memory the data is read from disk and
compared to computer memory. If there's any difference at all a
'VERIFY ERROR' is reported.

Note: If the BASIC program in memory is not located at the same
address as the version on disk was SAVEd from, the files will not
match even if the program is otherwise identical. The comparison ends
when the last byte is read from the disk file.

Use the BVERIFY command to compare memory with binary files.

                 DVERIFY "myprogram"

Good: SEARCHING FOR 0:myprogram   Bad: SEARCHING FOR 0:myprogram
       VERIFYING   VERIFYING
       OK   ?VERIFY ERROR

ELLIPSE -- Draw an ellipse on a graphic screen

      ELLIPSE x_center, y_center, x_radius, y_radius [,solid]

The ELLIPSE command will draw an ellipse with the given radii centered
at (x_center,y_center) on the current graphic screen. The ellipse will
be filled (i.e., a disc) if SOLID is non-zero.

                ELLIPSE 160,100,65,50

The above example will draw an ellipse in the center of a 320x200
pixel screen (160,100) having radii of (65,50) pixels. The aspect
ratio of the screen may cause it to appear as an circle, however. See
also the CIRCLE command.

ELSE -- See IF/THEN/ELSE

```
END -- Define the end of program execution

                              END

The  END  statement  terminates  program execution.  It does not close
channels  or  files,  and it does not clear any variables or reset any
pointers. An END statement does not need to be put at the last line of
a program.

The  CONTinue  command  can  be used to resume execution with the next
statement following the END statement. See also the STOP command.



ENVELOPE -- Define musical instrument envelopes

     ENVELOPE n, [,[atk] [,[dec] [,[sus] [,[rel] [,[wf] [,pw] ]]]]]


                    n............... Envelope number       (0-9)
                    atk ............ Attack rate           (0-15)
                    dec ............ Decay rate            (0-15)
                    sus ........... Sustain rate           (0-15)
                    rel ........... Release rate           (0-15)
                    wf ............ Waveform:   0 = triangle
                                                1 = sawtooth
                                                2 = pulse (square)
                                                3 = noise
                                                4 = ring modulation
                    pw ............ Pulse width            (0-4095)

              [*** THIS COMMAND IS SUBJECT TO CHANGE ***]

A parameter that is not specified will retain its current value. Pulse
width  applies  to  pulse  waves  (wf=2)  only and is determined by the
formula (pwout = pw/40.95 %), so that pw = 2048 produces a square wave
and  values  of  0  or  4095  produce  constant  DC output.  The C64DX
initializes the ten (10) tune envelopes to:


                    n A  D  S  R  wf  pw          instrument
                    ------------------          -------------
          ENVELOPE  0, 0, 9, 0, 0, 2, 1536       piano
          ENVELOPE  1,12, 0,12, 0, 1             accordion
          ENVELOPE  2, 0, 0,15, 0, 0             calliope
          ENVELOPE  3, 0, 5, 5, 0, 3             drum
          ENVELOPE  4, 9, 4, 4, 0, 0             flute
          ENVELOPE  5, 0, 9, 2, 1, 1             guitar
          ENVELOPE  6, 0, 9, 0, 0, 2, 512        harpsichord
          ENVELOPE  7, 0, 9, 9, 0, 2, 2048       organ
          ENVELOPE  8, 8, 9, 4, 1, 2, 512        trumpet
          ENVELOPE  9, 0, 9, 0, 0, 0             xylophone



ERASE -- Delete disk files

          ERASE "filespec" [,Ddrive] [<ON|,>Udevice] [,R]

This  command  is  identical  to  DELETE  and SCRATCH. See the SCRATCH
command for details.

     ERASE "myfile" Deletes the file MYFILE on the system drive.
```

ERR$ -- Error message function

                          ERR$ (error_number)

This function returns a string which is the BASIC error message
corresponding to the given error_message. If the given number is too
small (less than 1) or too large (greater than 41) an 'ILLEGAL
QUANTITY' error is reported.

This function is usually used to display a BASIC error condition in a
TRAP routine, using the BASIC error word ER as the error_number.
Note that when ER=-1, no BASIC error has occurred and ERR$(-1) results
in an 'ILLEGAL QUANTITY' error.

See the example at TRAP.

EXIT -- See DO/LOOP/WHILE/UNTIL/EXIT

EXP -- Function to return e^x

                           EXP (number)

This function returns the numeric value of e (2.71828183), the base of
natural logarithms) raised to the power of given number. If the
number is greater than 88.0296919 an 'OVERFLOW' error is reported.

          X = EXP(4)   Result is X=54.5981501

FAST -- Set system speed to 3.58MHz

FAST is the default state of the system. FAST is used to restore this
state following direct access of "slow" I/O devices such as the SID
sound chips.

FETCH -- (see the DMA command)

FILTER -- Define sound filter parameters

          FILTER [freq] [,[lp] [,[bp] [,[hp] [,res]]]]

          freq ..... Filter cut-off frequency (0-2047)
          lp .... Low pass filter on (1) off (0)
          bp .... Band pass filter on (1), off(0)
          hp .... High pass filter on (1), off(0)
          res ...... Resonance (0-15)

            [*** THIS COMMAND IS SUBJECT TO CHANGE ***]

Unspecified parameters result in no change to the current value. The
filter output modes are additive. For example, how low pass and
high pass filters can be selected to produce a notch (or band reject)
filter response. For the filter to have an audible effect at least
one filter output mode must be selected and at least one voice must be
routed through the filter.

FIND -- Find text in a BASIC program.

                        FIND :string: [,line_range]
                        FIND "string" [,line_range]

This is a direct (edit) mode command. FIND looks for all occurrences
of string in the program and displays each line containing string,
with string highlighted. Use the <C=> key to slow the display, or the
<NO-SCROLL> key to pause the display. Press <STOP> to cancel.

Any character can be used for the string delimiter, but there are side
effects. Using a non-quote delimiter will cause the string to be
tokenized, and FIND will find only tokenized strings in the program
that match. Using a quote character as the delimiter will cause the
string to be interpreted as plain PETSCII, and any matches found will
therefore be plain PETSCII. Searching for some tokens such as DATA
statements may require the use of colons as delimiters due to the
special affect these commands have upon the interpreter.

If the line number range is not given (see LIST for description of
range parameter), the entire program is searched.

FN xx -- User defined function

                        FN xx (expression)

The result of this numeric function is determined by the BASIC program
in a DEF FN statement. See the example at DEF FN.

FOR/TO/STEP/NEXT -- Program loop definition and control

                FOR index = start TO end [STEP increment]
                 |
                NEXT index [,index]

This command group performs a series of instructions a given number of
times. The loop index is a floating point (non-integer) variable
which will initially be set to the start value and be incremented by
the STEP increment when the NEXT statement is encountered. The loop
continues until the index exceeds the end value at the NEXT statement.

The start, end, and increment values can be numeric variables or
expressions. If the STEP increment is not specified, it is assumed to
be one (1). The STEP increment can be any value, positive, negative,
or non-integer. If the STEP increment is negative, the loop continues
until the index is less than the end value at the NEXT statement.

Note that, regardless of the start, end, or increment values, the loop
will always execute at least once. The index can be modified within
the loop, but it is bad practice to do so. It is also bad practice to
GOTO a line inside a loop structure, or to similarly jump out of a
loop structure (which can cause an 'OUT OF MEMORY' error).

Loops may be nested. If too many are nested, an 'OUT OF MEMORY' error
is reported (depends upon stack size, room for about 28 nested loops).

The index variable can be omitted from the NEXT statement, in which
case the NEXT will apply to the most recent FOR statement. If a NEXT
statement is encountered and there is no preceding FOR statement, the

error 'NEXT WITHOUT FOR' is reported.

```
10 FOR L = 1 TO 10
20 PRINT L
30 NEXT L
40 PRINT "I'M DONE! L = "L
```

This program prints the numbers from one to ten, followed by the message I'M DONE! L = 11.

```
10 FOR L = 1 TO 100
20 FOR A = 5 TO 11 STEP .5
30 NEXT A
40 NEXT L
```

This program illustrates a nested loop.

FOREGROUND -- Set the text color of the display

                        FOREGROUND color

Sets the text color to the given color index. Color must be in the range (0-15). See the Color Table. COLOR must be ON (see the COLOR command).

FRE -- Free byte function

                          FRE (x)

This function returns the number of available ("free") bytes in a specified area.

  PRINT FRE(0)   Shows the amount of memory left in the program area,
                 C64DX bank 0

  X = FRE(1)   X= the amount of available memory in variable area
                 C64DX bank 1. This causes a "garbage collect" to
                 occur, a process which compacts the string area.

  X = FRE(2)   X= the number of expansion RAM banks present.

GCOPY -- Copy a graphic area

                  [*** NOT YET IMPLEMENTED ***]

GENLOCK -- Enable or disable video sync mode & colors

                    GENLOCK ON [,color#]...
                    GENLOCK OFF [,color#,R,G,B]...

To enable video sync mode and specify which colors are affected, use the GENLOCK ON command, and list the palette color indices (0-255) which will display external video.

To disable video sync mode and restore the associated palette colors use the GENLOCK OFF command, and list the color index and its RGB values to restore them (see the SET PALETTE command for details). Also

see the PALETTE RESTORE command.

GET -- Get input data from the keyboard

                        GET variable_list

The GET statement is a way to get data from the keyboard one character
at a time. When the GET is executed, the character that was typed is
received. If no character was typed, then a null (empty) character is
returned, and the program continues without waiting for a key. There
is no need to hit the <RETURN> key, and in fact the <RETURN> key can
be received with a GET. The word GET is followed by a variable name,
usually a string variable. If a numeric were used and any key other
than a number was hit, the program would stop with an error message.
The GET statement may also be put into a loop, checking for an empty
result, that waits for a key to be struck to continue. The GETKEY
statement could also be used in this case. This statement can only be
executed within a program.

            10 DO: GET A$: LOOP UNTIL A$ ="A"

        This line waits for the A key to be pressed to continue.

GETKEY -- Get input character from keyboard (wait for key)

                        GETKEY variable_list

The GETKEY statement is very similar to the GET statement. Unlike the
GET statement, GETKEY waits for the user to type a character on the
keyboard. This lets it be used easily to wait for a single character
to be typed. This statement can only be executed within a program.

                        10 GETKEY A$

This line waits for a key to be struck. Typing any key will continue
the program.

GET# -- Get input data from a channel (file)

            GET# logical_channel_number, variable_list

Used with a previously OPENed device or file to input one character at
a time. Otherwise, it works like the GET statement. This statement can
only executed within a program.

                        10 GET#1,A$

GO64 -- Exit C64DX mode and switch to C64 mode

                            GO64

This statement switches from C64DX mode to C64 mode. The question 'ARE
YOU SURE?' (in direct mode only) is posted for the user to respond to.
If Y and return is typed then the currently loaded program is lost and
control is given to C64 mode. This statement can be used in direct
mode or within a program.

GOSUB -- Call a BASIC subroutine

                          GOSUB line

This statement is like the GOTO statement, except that the computer
remembers from where it came. When a line with a RETURN statement is
encountered, the program jumps back to the statement immediately
following the GOSUB. The target of a GOSUB statement is called a
subroutine. A subroutine is useful if there is a section of the
program that can be used by several different parts of the program.
Instead of duplicating the section over and over, it can be set up as
a subroutine and called with a GOSUB statement from different parts of
the program. This also make the main part of your program much more
readable. See also the RETURN statement.

Variables are shared with the main program and all subroutines. You
can pass information to, and get information back from, subroutines
by using variables as messengers.

GOSUB statements can be nested. That is, one subroutine can call
another subroutine, and the computer automatically keeps track of
all the calls. It's important not to jump into or out of subroutines
since this can confuse the computer. If too many GOSUBs are nested
(usually cause by jumping out of them) an 'OUT OF MEMORY' error is
reported because the computer ran out of room to keep track of all
the calls.

        10 DIR : GOSUB 100   show directory, check status
        20 GOSUB 200   print gap
        30 LIST "PROGRAM": GOSUB 100   show listing, check status
        40 GOSUB 200   print gap
        50 etc...
        90 END
        99 :
        100 REM SUBROUTINE TO CHECK DISK STATUS
        110 IF DS THEN GOSUB 200: PRINT "DISK ERROR: ";DS$
        120 RETURN
        199 :
        200 REM SUBROUTINE TO PRINT A SPACER ON THE SCREEN
        210 PRINT
        220 FOR I=1 TO 39: PRINT"-";: NEXT
        230 PRINT
        240 RETURN

GOTO -- Transfer program execution to specified line number

                     GOTO line_number
                     GO TO line_number

After a GOTO statement is executed, the next line to be executed will
be the one with the line number following the word GOTO. When used in
direct mode, GOTO line number allows starting of execution of the
program at the given line number without clearing the variables.

                     10 PRINT"COMMODORE"
                     20 GOTO 10

The GOTO in line 20 makes line 10 repeat continuously until STOP is
pressed.

GRAPHIC -- select graphic mode

                        GRAPHIC CLR
                        GRAPHIC command# [,args]

Basically this is a modified C64-type SYS command, minus the address.
In the C64DX system, this will represent the ML interface, not the
BASIC 10.0 interface which is implemented in the development system.

                [*** THIS COMMAND IS SUBJECT TO CHANGE ***]

GRAPHIC CLR initializes (warm-starts) the BASIC graphic system. It
clears any existing graphic modes, screens, etc. and allows a program
to commence graphic operations from scratch.

HEADER -- Format a diskette

          HEADER "diskname" [,Iid] [,Ddrive] [<ON|,>Udevice]

The HEADER command prepares a new diskette for use, sometimes called
FORMATting a diskette. There are two types of "newing" a diskette -- a
long form and a quick (or short) form. You must use the long form when
preparing a new diskette for its first use. Thereafter you can use the
quick form.

WARNING: Formatting a diskette (long or short) will destroy all
existing data on the diskette! In direct mode, you are asked to
confirm what you are doing with 'ARE YOU SURE?'. Type 'Y' and press
return to proceed, or TYPE ANY OTHER CHARACTER AND PRESS RETURN TO
CANCEL the command. In program mode there is no confirmation prompt.

The long HEADER form requires a diskname and an ID. The diskette will
be completely (re)sectored, zeros written to all blocks, and a new
system track (directory, BAM, etc.) will be created.

        HEADER "newdisk",I01   prepares a new diskette

The short HEADER form is performed when the ID option is omitted. The
diskette is assumed to have been previously formatted, and only a new
system track (directory, BAM, etc.) is installed. This is roughly
equivalent to deleteing all the files, but much quicker.

        HEADER "makelikenew"   re-news an working diskette

The diskname is limited to 16 characters and the ID string to two
characters. The same rules apply for the diskname as for a filename.
Some Disk Systems use the ID string to tell if you have swapped a
diskette in a drive, so it's recommended that the ID string be unique
for each of your diskettes. Some more examples:

                    HEADER "QUICK"
                    HEADER "MYDISK", I23
                    HEADER "RECS", I"FB", U9
                    HEADER (FILES), I(ID$), U(UNIT)

HELP -- Show the BASIC line that cause the last error

The HELP command is used after an error has been reported in a
program. When HELP is typed, the line where the error occurred
listed, with the portion containing the error highlighted. Print
ERR$(ER) for the error message, and print EN or EL for the error
number and error line, respectively. HELP can be used in direct mode
or in program mode. Note that, in the case of many I/O errors, there
is no associated BASIC error. Check ST or DS$ errors in these cases.

HEX$ -- Hexadecimal value function

                    HEX$ (decimal_expression)

This function returns a 4-character string that represents the
hexadecimal value of the numeric decimal expression. The expression
must be in the range (0-65535, $0000-$FFFF hex) or an 'ILLEGAL
QUANTITY' error is reported.

```
    PRINT HEX$(10)              The string "000A" is printed.
    PRINT RIGHT$(HEX$(10),2)    The string  "0A" is printed.
```


HIGHLIGHT -- Set the text highlight color of the display

                        HIGHLIGHT color

Sets the highlight color to the given color index. The color value
must be in the range (0-15). See the Color Table. COLOR must be ON
(see the COLOR command). The highlight color is used in HELP messages
and FIND/CHANGE strings.

IF/THEN/GOTO/ELSE -- Conditional program execution

    IF expression <GOTO line | THEN then_clause> [:ELSE else_clause]

IF...THEN lets the computer analyze a BASIC expression preceded by IF
and take one of two possible courses of action. If the expression is
true, the statement following THEN is executed. This expression can be
any BASIC statement. If the expression is false, the program goes
directly to the next line, unless an ELSE clause is present. The ELSE
clause, if present, must be in the same line as the IF-THEN part. When
an ELSE clause is present, it is executed when the THEN clause isn't
executed. In other words, the ELSE clause executes when the expression
is FALSE. See BEGIN/BEND to spread the IF statement out over several
lines. An ELSE statement is matched to the closest THEN statement in
the case of nested IF/THEN statements.

The expression being evaluated may be a variable or formula, in which
case it is considered true if nonzero, and false if zero. Usually
expressions involve relational operators =, <, >, <=, >=, <>.

            50 IF X>0 THEN PRINT "X>0": ELSE PRINT "X<=0"

If X is greater than 0, the THEN clause is executed, and the ELSE
clause isn't. If X is less than or equal to 0, the ELSE clause is
executed and the THEN clause isn't.

INPUT -- Get input from the keyboard

                [LINE] INPUT ["prompt"<,1;>] variable_list

The INPUT statement pauses the BASIC program, prints the prompt string
if present, prints a question mark and a space, and waits for data to
be typed by the user, terminated by a return character. If the prompt
string ends with a comma instead of a semicolon, a question mark and
space is not printed.

Input is gathered and assigned to variables in the variable_list. The
type of variable must match the type of input typed or a 'TYPE
MISMATCH' error is reported. Separate data items typed by the user
must be separated with commas. String data with imbedded spaces or
commas must be surrounded with quotes. If insufficient data to satisfy
the variable-list is typed, two question marks are displayed by the
computer to prompt for additional data to be input. If the computer
does not understand the input (such as the user typing cursor up or
down keys) the computer responds with the message 'REDO FROM START?'
and waits for acceptable data to be entered. Input is limited to 160
characters (two screen lines in 80-column mode), which is the size of
the input buffer.

The INPUT statement can only be executed from within a program.

LINE INPUT allows the program to input a string which includes any
PETSCII character (including colons, commas, imbedded spaces, etc.)
up to but not including a null or return character. There should be
only one string-type variable name in the variable_list in this case,
but if there are more the computer prompts as usual with two question
marks for more data to assign to the additional variables.

        10 INPUT "WHAT'S YOUR FIRST NAME AND AGE"; NA$,A
        20 PRINT "YOUR NAME IS ";NAS;" AND YOU ARE";A;" YEARS OLD"

The above INPUT is the traditional BASIC form.

        10 LINE INPUT "WHAT'S YOUR ADDRESS"; AD$
        20 PRINT "YOUR ADDRESS IS: ";AD$

The above INPUT allows an entire line of data to be assigned to a
string variable, including commas and other common punctuation marks.

        10 INPUT "ENTER YOUR NAME HERE: ", NA$

The above INPUT suppresses the traditional '? ' prompt by using a
comma instead of a semicolon after the prompt string. To suppress the
'?' without a prompt string, make the prompt string null.

INPUT# -- Input data from an I/O channel (file)

            [LINE] INPUT#logical_channel_number, variable_list

The INPUT# command works like the INPUT command, except no prompt
string is allowed and input is gathered from a previously OPENed
channel or file. This command can only be used in a program.

The logical channel number is the number assigned to the device (file)
in an OPEN (or DOPEN) statement. Items in the variable list must agree
with the type of data input, or a 'FILE DATA ERROR' will resuit.

On the C64DX, an End Of File (EOF) condition or bad I/O status will
terminate input, as if a return character was received. It's good
practice to examine the I/O status byte (and the DS disk status for
file I/O) after every I/O instruction to check for problems or errors.

```
10 DOPEN#1, "FILE"                            This program will
20 C=0                                        count the number of
30 DO: LINE INPUT#1,AS: C=C+1: LOOP UNTIL ST  lines in FILE
40 DCLOSE#1
50 PRINT"FILE CONTAINS";C;" LINES."
```


INSTR -- Get the location of one string inside another string

            INSTR (string_1, string_2 [,starting_position])

This function searches for the first occurrence of string_2 in
string_1 and returns its location. A value of zero (0) is returned if
no match is found, if either string is null (empty), or if string_2
is longer than string_1.

If the starting position is given, the search begins at that location,
otherwise the search begins at the first character of string_1.

The strings can be literals, variables, or string expressions.

```
X = INSTR("123456","4")        Result is X=  0
X = INSTR("123456","X")        Result is X= -1
X = INSTR("123123","2")        Result is X= 123
X = INSTR("123123","2",3)      Result is X=-124
```


INT -- Greatest integer function

                        INT (expression)

This function returns the greatest integer less than or equal to the
numeric expression.

JOY -- Joystick function

                        JOY (port)

This function returns the state of a joystick controller in the
specified port.

        When port=1 returns position of joystick 1
        When port=2 returns position of joystick 2

The value returned is encoded as follows:

    Fire = 128 +   1

                        8   2

                7   0   3

                        6   4

A value of zero (0) means that the joystick is not being manipulated.
A value of 128 or more means that the fire button is being pressed.
The possible vales returned are:

0	No activity	128	fire
1	up	129	fire + up
2	up + right	130	fire + up + right
3	right	131	fire + right
4	right + down	132	fire + right + down
5	down	133	fire + down
6	down + left	134	fire + down + left
7	left	135	fire + left
8	left + up	136	fire + left + up

KEY -- Enable, disable, display, or define function keys

                    KEY ON
                    KEY OFF
                    KEY [key#, string]

There are 14 function keys available on the C64DX (seven unshifted and
seven shifted). The user can assign a string consisting of BASIC
commands, control codes, escape functions, or a combination of each to
function key. The data assigned to a key is typed out when that key is
pressed, just as if the characters were typed one by one on the
keyboard. The user can enable ("turn on") or disable ("turn off") the
function keys. When they are disabled, pressing a function key return
that key's normal character code instead of the string assigned to it.
This includes the HELP and (shifted) RUN keys. It is also possible to
redefine the HELP and (shifted) RUN keys, as function keys 15 and 16,
respectively. The system has default assignments for all function
keys. KEY with no parameters displays a listing of the current
assignments for all the function keys.

The maximum length for all the definitions together is 240 characters.
If an assignment would be too big to fit, an 'OUT OF MEMORY' error is
reported and the assignment is not made.

        KEY 2, "DIR U9"+CHR$(13)

This causes the computer to display the directory from disk unit #9
when function key 2 is pressed. This is equivalent to typing 'DIR U9'
and pressing the <RETURN> key directly. The CHR$(13) is the character
for <RETURN>. Other often used control codes are CHR$(141) for
'shifted RETURN', CHR$(27) for 'ESCape', and CHR$(34) to incorporate a
double quote into a KEY string.

        KEY 2, "DIR"+CHR$(34)+"*=P"+CHR$(34)+CHR$(13)

This is equivalent to typing DIR"*=P" and pressing <RETURN> at the
keyboard. Note the way quotes can be incorporated into an
assignment. When function key 2 is pressed, a directory of all program
files on the default system disk will be displayed.

        KEY OFF

This turns off function key strings. Pressing a function key now would
return the character codes associated with F-keys as on the VIC-20 and

C64 computers. KEY ON would re-enable function key strings, unchanged
from their previous assignments. To restore the system default
assignments, reset the computer.

LEFT$ -- Get the leftmost characters of a string

                        LEFT$ (string,count)

This function returns a string containing the leftmost 'count' number
of characters of the string expression. Count is an numeric
expression in the range (0-255). If count is greater than the length
of the string, the entire string will be returned. If count is zero,
a null (empty) string will be returned.

        A$ = LEFT$("123ABC",3)   Result is A$="123"

LEN -- Get the length of a string

                          LEN (string)

This function returns the number of characters in a string expression.
Nonprinting characters and blanks are counted.

        A = LEN("ABC")   Result is A=3

LET -- Assign a value to a variable

                    [LET] variable = expression

The LET command is optional, since the equal sign by itself is
understood by the computer to mean assignment. Multiple assignments
on LET statements are not allowed.

        10 LET A=1: LET B=A+1: LET C$=" THREE"
        20 : D=1:   E=D+1:   F$=" THREE"
        30 PRINT A;B;C$
        40 PRINT D;E;F$

     Output: 1 2 THREE
                     1 2 THREE

LINE -- Draw a line on a graphic screen

                     LINE x0, y0, x1, y1

LINE draws a line on the currently defined graphic screen with the
currently defined draw modes. The line is draw from (x0,y0) to
(x1,y1).

LIST -- List a BASIC program from memory or disk

                  LIST [startline] [- [endline]]

                LIST "filename" [,Ddrive] [<,|ON>Udevice]

LIST is used to view part or all of a BASIC program in memory or all
of a BASIC program on disk (without affecting the program that is

currently in memory).

The display can be slowed down by holding down the <C=> key or it can
be paused by pressing the <NO-SCROLL> key or <CONTROL><S>. A listing
that is paused can be restarted by pressing <NO-SCROLL> again or by
pressing <CONTROL><Q>. The display can be stopped by pressing <STOP>.

If the word LIST is followed by a line number, the computer shows only
that line number. If LIST is typed with two numbers separated by a
dash, the computer shows all lines from the first to the second line
number. If LIST is typed followed by a number and just a dash, it
shows all lines from that number to the end of the program. And if
LIST is typed, a dash, and then a number, all lines from the beginning
of the program to that line number are LISTed. By using these
variations, any portion of a program can be examined or easily brought
to the screen for modification. LIST can be used in direct mode or in
a BASIC program.

     LIST   Shows entire program.
     LIST 100-   Shows from line 100 until the end of the program.
     LIST 10   Shows only line 10.
     LIST -100   Shows lines from the beginning until line 100.
     LIST 10-200 Shows lines from 10 to 200, inclusive.

LOAD -- Load a program or data into memory from disk

          LOAD "filename" [,device_number [,relocate_flag]]

This command loads a file into the computer's memory. The filename
must be given, and pattern matching may be used. In the case of dual
drive systems, the drive number must be part of the filename. If a
device number is given, the file is sought on that unit, which must be
a disk drive. If a device number is not given, the default system
drive is used. See also DLOAD and RUN commands.

The relocate_flag is used to LOAD binary files. If the relocate_flag
is present and non-zero, the file will be copied into memory starting
at the address stored on disk when the file was SAVEd. See BLOAD. Do
not use the relocate_flag to load BASIC programs: they will be
automatically relocated to the start of the BASIC program area and
relinked.

To compare a program in memory to a disk file, use the VERIFY or
DVERIFY command. To compare a binary file, use BVERIFY.

See the discussion at DLOAD regarding CHAINING programs.

  LOAD "PROG"   Loads BASIC program PROG from the system drive.
  LOAD FILE$,DRV   Loads a program whose name is in the variable
                     called F$ from the unit whose number is in DRV.
  LOAD "0:PROG" 8   Loads BASIC program PROG from unit 8, drive 0.
  LOAD "BIN",8,1   Loads a binary file into memory.

LOCATE -- [*** NOT YET IMPLEMENTED ***]

LOG -- Get the natural logarithm of a number

               LOG (number)

This function returns the natural logarithm of a numeric expression. A
natural log is a log to the base e (2.71828183). See the EXP function.
To convert to log base 10, divide by LOG(10).

```
        A = LOG(123)                    Result is A=4.81218436
        A = LOG(123) / LOG(10)          Result is A=2.08990511
```


LOOP -- See DO/LOOP/WHILE/UNTIL/EXIT

LPEN -- Get the position of a lightpen

                        PEN (position)

This function returns the current position of a lightpen on the
screen. When position=0, the X position is returned, and when
position=1 the Y position is returned. Note that lightpen coordinates,
like sprite coordinates, are offset from the normal graphic coordinate
map. This means you have to calculate where the lightpen is with
respect to the screen display. The electronics of each lightpen also
introduces a skew which must be factored into your calculations.

The X resolution is limited to every 2 pixels, and will always be an
even number in the approximate range (60-320). The Y position is in
the approximate range (50-250). If either the X or the Y position is
zero, the lightpen is off-screen.

Note that a lightpen COLLISION need not be enabled to use LPEN. A
bright background color, such as white, is usually required to
stimulate the light pen. Lightpens only work in game port 1.

```
        10 TRAP 40                      We're done if STOP key
        15 BACKGROUND 1                 Make background color white
        16 FOREGROUND 0                 Make text color black
        20 COLLISION 3,100              Enable lightpen interrupt
        30 DO:LOOP                      Hang here until done
        40 END                          Done
       100 COLLISION 3                  Got one, don't want more
       110 PRINT LPEN(0),LPEN(1)        Display lightpen position
       120 COLLISION 3,100              Re-enable interrupt
       130 RETURN
```


MID$ -- Substring function

                    MID$ (string, position [,length])

This function can appear on the left or the right side of an
assignment statement:

Case 1:   string_var = MID$ (string_expression, position [,length])

This form returns a piece of another string. The function returns a
string of the specified length taken from the string_expression
beginning at the indicated position. The position must be in the range
(1-255), one (1) being the first character. The length can be any
number in the range (0-255), or it can be omitted. If the position
specified is greater than the number of characters in the
string_expression, a null (empty) string is returned. If the length is

greater than the number of characters from the given position to the
end of the string_expression, or the length is omitted, then all the
rightmost characters beginning at the position are returned.

```
        A$ = MID$("TICTACTOE",4,3)        Result is A$="TAC"
        A$ = MID$("TICTACTOE",4)          Result is A$="TACTOE"
        A$ = MID$("TICTACTOE",10,1)       Result is AS="" (empty)
```

Case 2:   MID$ (string_var, position [,length]) = string_expression

This form replaces a portion of the string contained in string_var
with data from another string_expression, beginning at the specified
position in the string_var. If the length is given only, that many
characters from the string_expression are taken, otherwise all the
characters in the string_expression will replace characters in the
string_var beginning at the position specified. The there are too many
characters to fit in the string_var, an 'ILLEGAL QUANTITY' error is
reported. If the length given is zero, no characters will be replaced.

```
 A$="TICTACTOE": MID$(A$,4,3)="123456"   Result is A$="TIC123TOE"
 A$="TICTACTOE": MID$(A$,4)  ="123456"   Result is A$="TIC123456"
 A$="TICTACTOE": MID$(A$,5)  ="123456"   Result is 'ILLEGAL QUANTITY'
```


MONITOR -- Enter the built-in machine language monitor

              SEE SECTION 3.2 ON THE C64DX MONITOR.

MOUSE -- Enable or disable the mouse driver

```
                MOUSE ON [,port [,sprite [,position] ] ]
                MOUSE OFF
```

```
    port       = joyport 1, 2, or either (both)        (1-3)
    sprite     = sprite pointer                        (0-7)
    position   = initial pointer location              (x,y)
                 normal, relative, or angular coordinate

                 defaults to sprite 0, port 2
                 ????    add min/max x/y positions
```

              [*** THIS COMMAND IS SUBJECT TO CHANGE ***]

Mouse ON enables the built-in mouse driver. The user must load a
pointer into the proper sprite area ($600-$7FF). The driver assumes
the "hot point" is the top left corner of the sprite, and does not
allow this point to leave the screen.

Mouse OFF will turn off the driver and the currently associated
sprite.

Use the RMOUSE function to get the current pointer position and button
status. See the sample program at RMOUSE.

MOVSPR -- Position sprite or set sprite in motion

                      MOVSPR sprite <,x,y>

Use the SPRITE command to turn on a sprite and MOVSPR to position it.

Sprites are numbered 0-7. The sprite's position can be specified using
one of the following coordinate types:

```
        [+/-]x, [+/-]y = [relative] position
               x#y   = angle and speed
               x;y   = distance and angle
```

Angles are specified as 0-360 degrees, with 0 being straight up.
Speeds are specified as a number of pixels per frame, 0-255. Sprites
are moved through each pixel so that collisions are accurately
detected.

NEW -- Delete program in memory and clear all variables

                        NEW [RESTORE]

This command erases the entire program in memory and clears all
variables and open channels (but it does NOT properly close open
disk write files -- used DCLOSE or DCLEAR beforehand). NEW also resets
the runtime stack pointer (clears GOSUB & FOR/NEXT stacks), the DATA
pointer, and the PRINTUSING characters.

The BASIC program in memory is lost unless it was previously SAVEd to
disk. If you have not entered or loaded any BASIC programs since
typing NEW, the RESTORE option will recover the BASIC program in
memory. But if the BASIC environment has been changed in any way, the
program may not be restored correctly. If BASIC can tell something's
wrong, it will report 'PROGRAM MANGLED'.

NEW can be used in direct (edit) mode or in a program. When it's
encountered in a program, the program terminates.

NEXT -- See FOR/NEXT/STEP and RESUME

NOT -- Get the complement of a number

                        NOT (expression)

The NOT function returns the complement of an integer in the range
(-32768 to 32767). The function operates on the binary value of signed
16-bit integers. An expression outside of this range will cause an
'ILLEGAL QUAUTITY' error.

```
            X = NOT(5)              Result is X=-6
            X = NOT(-6)             Result is X=5
```

NOT is often used in logical comparisons (such as an IF statement) to
invert the result, since -1 (true) is the result of NOT(0) (false),
and 0 (false) is the result of NOT(-1) (true).

```
  X = NOT("ABC"="ABC") AND ("DEF"="DEF")      Result is X= 0 (false)
  X = NOT("ABC"="ABC") AND ("DEF"="XYZ")      Result is X=-1 (true)
```


OFF -- Subcommand used with various BASIC commands.

ON -- Computed GOTO/GOSUB

```
                ON expression <GOTO|GOSUB> line_number_list
```

This is a variation of the IF <expression> GOTO statement that
branches to one of several line numbers based upon the value of an
expression. The integer value of the evaluated expression determines
which line number in the line_number_list gets control.

If the expression evaluates to one, the first line number in the list
gets control, if it's two the second line number gets control, and so
on. Fractional parts of the value are truncated (for example, 2.9
becomes 2). If the value is zero or greater than the number of items
in the list the computer takes none of the branches and continues on
with the next statement. If the value is negative, an 'ILLEGAL
QUANTITY ERROR' is reported.

The ON/GOSUB statement must call the first line number of a subroutine
and the subroutine must end with a RETURN statement. After executing
the subroutine, control is returned to the statement following the
ON/GOSUB statement.

```
              10 INPUT"ENTER A NUMBER 1-3: ",X
              20 ON X GOTO 100, 200, 300
              30 PRINT"TOO LOW OR TOO HIGH": RUN
             100 PRINT"ONE":   RUN
             200 PRINT"TWO":   RUN
             300 PRINT"THREE": RUN
```


OPEN -- Open a channel to a device or disk file

```
    OPEN logical_chnl_num, device_number [,secondary_adr
                                      [,<filespec|command>]]
```

Before a program can access a device or a file, an I/O channel must be
opened to it to communicate through. When something is opened, you
associate a logical channel number with it, and it is with this number
that all other I/O statements access the device or file. The OPEN
command can be used in direct (edit) mode or in a program.

The channel number, device number, and optional secondary address are
integers from 0-255. Refer to the device's manual for more
information about what (if any) secondary addresses it uses.


```
     channel:    0-127    return = output return character only
               128-255    return = output return + linefeed

     device:        0     Keyboard
                    1     Default system drive
                           whatever its number is (see SET DEF)
                    2     RS232
                    3     Screen
                   4-7    Serial bus
                           (usually reserved for printers)

                   8-31   Serial bus
                           (usually reserved for disk drives)
```

The filespec is the file name in the case of disk files (refer to your
DOS manual for details). Typically, the filename is a string having

the the following form:

```
             [[@|S]drive:] filename [,type] [,mode]
```

An example would be 0:MYFILE,SEQ,READ to open the sequential file
MYFILE for reading on drive 0. Disk drives usually support some kind
of filename pattern matching. Most disk drives support the following
file types and modes (can be abbreviated to first character):

```
    types:            'S'equential
                      'P'rogram
                      'R'elative
                      'U'ser

    modes:            'R'ead
                      'W'rite
                      'L'ength (for relative type files)
```

Some channels or devices accept a command string instead of a filename
when they are opened. An example would be the disk command channel or
the RS232 open/setup command. Refer to the device's documentation.

```
    OPEN 1,8,15,"I"         Open CBM disk command channel & send
                                it the 'I'nitialize command.
    OPEN 4,4,7             Open CBM printer channel in upper/lower
                                case mode.
    OPEN 128,2,2,CHR$(14)  Open a 9600 8N1 RS232 channel and
                                translate CR into CRLF on output.
```

See also DOPEN, DCLOSE, CLOSE, CMD, GET#, INPUT#, and PRINT#
statements and I/O status variables ST, DS, and DS$.

OR -- Boolean operator

```
                    expression OR expression
```

The OR operator returns a numeric value equal to the logical OR of two
numeric expressions, operating on the binary value of signed 16-bit
integers in the range (-32768 to 32767). Numbers outside this range
result in an 'ILLEGAL QUANTITY' error.

```
              X = 4 OR 8              Result is X=12
```

In the case of logical comparisons, the numeric value of a true
situation is -1 (equivalent to 65535 or $FFFF hex) and the numeric
value of a false situation is zero.

```
    X = ("ABC"="ABC") OR ("DEF"="DEF")      Result is X=-1 (true)
    X = ("ABC"="ABC") OR ("DEF"="XYZ")      Result is X=-1 (true)
    X = ("ABC"="XYZ") OR ("DEF"="XYZ")      Result is X= 0 (false)
```


PAINT -- Fill a graphics area with color

```
                    PAINT x,y, mode [,color]
```

```
       x,y    coordinate to begin fill at
       mode   0: fill area to edge = color
              1: fill area to edge=same as color at x,y
```

PAINT fills an enclosed graphic area starting at the given coordinate
with the color of the currently defined PEN. The mode parameter
identifies the region to be filled.

                [*** THIS COMMAND IS NOT YET IMPLEMENTED ***]

PALETTE -- Define a color

          PALETTE [screen#|COLOR], color#, red, green, blue
          PALETTE RESTORE

          screen# 0-1
          color# 0-255
          red   0-15
          green   0-15
          blue   0-15

The PALETTE command can be used to define a color for a logical
graphic screen, set an absolute color, or restore the C64DX VIC-III
default colors. PALETTE can be used in direct mode or in a program.

The VIC-III pre-defines the first 16 colors to the usual C64-type
colors, but you can change them with the PALETTE COLOR command or
restore them all with the PALETTE RESTORE command.

See the sample program after the SCREEN command.

PASTE -- Put a CUT graphic area on the screen

                              PASTE x,y

                     [*** NOT YET IMPLEMENTED ***]

PEEK -- Function returning the contents of a memory location

                            PEEK (address)

This function returns the contents of a memory location. The address
must be an integer in the range of 0-65535 ($0-$FFFF) and the value
returned will be an integer in the range of 0-255 ($0-$FF).

Use the BANK command to specify which 64K memory bank the address is
in. Note that a BANK number greater than 127 (i.e., a bank number
with the most significant bit set) must be used to address an I/O
location, such as the VIC chip or color memory. Refer to the system
memory map for details. PEEK uses the DMA device to access memory.

Use the POKE command to change the contents of a memory location.

  BANK 0: X = PEEK (208)   Reads the keyboard buffer index. If
                              it's empty, X will be zero, otherwise X
                              will be the number of characters in it.

PEN -- Specify a pen color for drawing on graphic screen

                            PEN pen, color

```
         pen      0-2
         color    0-255
```

Before you can draw anything on a graphic screen, you have to tell
BASIC what color your PENs are. You should first define what your
colors are using the PALETTE command, then use PEN to associate those
colors with a PEN. Whatever graphic commands you use after a PEN
command will use the PEN you specified.

         PEN 0,1   Put color 1 "ink" into draw pen 0

See the sample program after the SCREEN command.

PIC -- Graphic picture subcommand

PLAY -- Play a musical string

                  PLAY "[Vn,On,Tn,Un,Xn,elements]"

              [*** WILL CHANGE TO ADD 2ND SID SUPPORT ***]

The PLAY command lets you select a voice, octave, instrument, volume,
filter, and musical notes. All these parameters are packed into a
string (spaces are allowed for readability).

              On = Octave (n=0-6)
              Tn = Tune envelope # (n=0-9)
                      0= piano   (defaults)
                      1= accordion
                      2= calliope
                      3= drum
                      4= flute
                      5= guitar
                      6= harpsichord
                      7= organ
                      8= trumpet
                      9= xylophone
              Un = Volume (n=0-9)
              Vn = Voice (n=1-3)
              Xn = filter on (n=1), off (n=0)

        Elements:
              A,B,C,D,E,F,G   ... Notes, may be preceded by:
              # .... Sharp
              S .... Flat
              . ... Dotted
              W .... Whole note
              H .... Half note
              Q .... Quarter note
              I .... Eighth note
              S .... Sixteenth note
              R .... Rest
              M .... Wait for all voices playing to end
                                  (a measure)

Once the music string starts PLAYing, the computer will continue with
the next statement. The music will continue to play automatically.
Using the 'M'easure command will cause the computer to wait until the
music has up to that point has been played out.

Use the TEMPO command to alter the tempo (speed) of PLAY. Note that
the VOLume command can change a PLAY string's volume setting.

POINTER -- Get the address of a variable descriptor

                        POINTER (variable_name)

This function returns the address of an entry in the variable table.
If the value returned is zero, the variable is currently undefined.
The variable table is normally in the second RAM bank (BANK 1). See
the section on variable storage for details.

Note that, while the location of a string descriptor will not change,
the location of the actual string in memory changes all the time.
Also, when working with an array name you must specify a particular
element, to which POINTER will return a pointer to that element's
descriptor and not to the array descriptor.

        10 A$="FRED"   Define A$
        20 DESC=POINTER (A$)   Lookup A$ in variable table
        30 BANK1: PRINT PEEK(DESC)   Displays the length of A$

PORE -- Write a byte to memory location

                    POKE address, byte [,byte ...]

POKE is used to write one or more bytes into one or more memory
locations. The address must be an integer in the range of 0-65535
($0-SFFFF) and the value to be written must be an integer in the range
of 0-255 ($0-$FF). If more than one byte is given, it will be written
into successive memory locations.

Use the BANK command to specify which 64K memory bank the address is
in. Note that a BANK number greater than 127 (i.e., a bank number
with the most significant bit set) must be used to address an I/O
location, such as the VIC chip or color memory. Refer to the system
memory map for details. Also note that, unlike previous CBM computers,
POKEs to a ROM location will not "bleed through" into a corresponding
RAM location. POKE uses the DMA device to access memory.

Use the PEEK function to read a byte from a memory location.

Because this command directly accesses system memory, extreme care
should be taken in its use. Altering the wrong memory location can
crash the computer (press the reset button to reboot).

 BANK 0: POKE 208,0   Resets location 208 ($000D0),
                                       clearing the keyboard buffer.

 BANK 128: POKE DEC("D023"),1,2,3 Sets the VIC extended background
                                       colors to 1, 2, and 3 respectively

POLYGON -- Draw a regular n-sided figure on a graphic screen

 POLYGON x,y, xradius,yradius, [solid], angle,drawsides,sides,subtend

        x,y   = center of polygon

```
        x,yradius    = radii of polygon
        solid        = solid flag              (0-1)
        angle        = starting angle          (0-360)
        drawsides    = # of sides to draw      (3-127)
        sides        = # sides of polygon      (drawsides<=sides)
        subtend      = subtend flag            (0-1)
```


POS -- Get the column number of the cursor

                        POS(0)

This function returns the current text column the cursor is in, with
respect to the currently defined window (see RWINDOW). It's usually
used to format text printed to the screen. The argument (0) is not
used for anything. POS will not work as expected if text output is
redirected to a disk file or the printer.

```
        10 MAXCOL = RWINDOW(1)
        20 FOR ADR=DEC("600") TO DEC("7FF")
        30 PRINT HEX$(PEEK(ADR));" ";
        40 IF POS(0) > (MAXC0L-5) THEN PRINT
        50 NEXT
```

This example illustrates one way to format output to the screen,
keeping the last item on a line from being split between two lines,
regardless of the window size (as long as the window size is at least
4 characters wide). It dumps the data for the first sprite in hex.

POT -- Paddle function

                        POT (paddle)

This function returns the state of a game paddle (POTentiometer)
controller in one of the two game ports.

```
     paddle=1 ..... Position of paddle #1 (port 1, paddle "A")
     paddle=2 ..... Position of paddle #2 (port 1, paddle "B")
     paddle=3 ..... Position of paddle #3 (port 2, paddle "A")
     paddle=4 ..... Position of paddle #4 (port 2, paddle "B")
```

The value returned by POT ranges from 0 to 255. Any value greater than
255 means that the fire button is also pressed. Paddles are read
"backwards" from normal things like volume knobs or faucets. A value
of 255 means the paddle has been turned counterclockwise as far as it
will go ("off"), and a value of 0 means the paddle has been turned
clockwise as far as it will go "on").

Note that some paddles are "noisy" and their output must be averaged
or "damped" to prevent whatever they are controlling from jittering.

```
     10 SPRITE 1,1                      Turn on a sprite
     20 DO                              Begin a loop
     30 X=POT(3)                        Read paddle "A" in port 2
     40 MOVSPR 1,300-(X AND 254),200    Move the sprite
     50 LOOP UNTIL X>255                Loop until button pressed
     60 SPRITE 1,0                      Turn off sprite
```

This sample program turns on a sprite and lets you move it
horizontally with a paddle. If you press the paddle's fire button, it

turns off the sprite and the program ends. The calculations in line 40
do several things all at once -- they mask the fire button and "damp"
the output to reduce jitter by masking the least significant bit (the
X AND 254 part) and invert the output so that turning the paddle to
the right makes the sprite go right (subtracting result from 300).

PRINT -- Display data on text screen

                    PRINT [expression_list] [<,|;>]

PRINT will evaluate each item in the expression_list and pass the
results to the system screen editor to display on the screen. If a
screen window is defined, the output will be confined to the window.
PRINT can be used to send control codes and escape sequences to the
screen editor to do such things as set windows, change TAB stops,
change text colors or set reverse field, or choose cursor styles. See
the section on Editor modes for details.

PRINT can be followed by any of the following:

    Numeric or string expressions   12, "HELLO", 1+1, "S"+STR$(I)
    Variable names   A, B, A$, X$
    Functions   ABS(33), HEX$(160)
    Punctuation marks   ;,
    Nothing

Numeric values are always followed by a space. Positive numbers are
preceded by a space, and negative numbers are preceded by a minus sign
('-'). Scientific notation is used when a number is less than 0.01 or
greater than or equal to 999999999.2 .

A semicolon (';') or space between list items causes the next item to
be printed immediately following the previous item. A comma (',')
causes the next item to be printed at the next comma stop (similar to
TAB stops, but every 10 spaces). These rules apply to the next print
statement, if the expression_list ends with either a semicolon or a
comma, otherwise a return is printed. Note that floating point
variable names should not be separated from the next variable name
with a space, and constants should not be preceeded or followed by a
space.

For formatted PRINT output, see the PRINT USING command.

   PRINT "HELLO"   HELLO

   A$="THERE": PRINT "HELLO ";A$ HELLO THERE

   A=4:B=2: PRINT A+B   6

   J=41: PRINT J;: PRINT J-1   41 40

   C=A+B:D=B-A: PRINT A;B;C;D   4 2 6 -2
   C=A+B:D=B-A: PRINT A,B,C,D   4   2   6   -2

   A=1:B=2:AB=3: PRINT A B   3
   PRINT 1 2 3, 1 2 3 +1   123   124

   PRINT 0.009, 0.01   9E-03   .01
   PRINT 999999999; 999999999.2   999999999 1E+09

The CMD command can be used to redirect PRINT output to a device or file. Also see the POS, SPC, TAB functions, CHAR and PRINT USING.

PRINT# -- Send data to an I/O channel (file)

        PRINT#logical_channel_number [,expression_list] [<,|;>]

This command is used to send (transmit) data to a device or file. The logical_channel number is the number assigned to the device (file) in an OPEN (or DOPEN) statement. The output is otherwise identical to that of a PRINT statement, including the comma and semicolon conventions. Note that certain screen-oriented functions, such as TAB and SPC do not have the same effect as they do with screen I/O.

It's good practice to examine the I/O status byte (and the DS disk status for file I/O) after every I/O instruction to check for problems or errors.

For formatted output, use the PRINT# USING command.

```
    10 OPEN 1,8,15                  Initialize disk drive
    20 PRINT#1,"I"                  (same as DCLEAR)
    30 CLOSE 1

    10 DOPEN#1,"NEWFILE",W          Create a SEQ file
    20 FOR I=1 TO 10
    30 PRINT#1, I, STR$(I)          Write numbers 1-10 to it
    40 NEXT
    50 DCLOSE#1

    10 OPEN 2,2,2,CHR$(12)          Open 1200 baud RS232 channel
    20 PRINT#2, "ATDT,5551212"      Send modem a Hayes dial command
```


PRINT USING -- Output formatted data to the screen, device, or file

PRINT [#logical_channel_number,] USING format; expression_list [<,|:>]

Read about the PRINT and PRINT# commands first for information regarding the syntax of the expression list and, for device output, establishing the logical_channel_number.

The items in the expression list must be separated by commas (',').

The format is defined in a string literal or string variable and is described below. See the PUDEF command for specifing special formatting characters. The various formatting characters are:

CHARACTER	SYMBOL	NUMERIC	STRING
Pound sign	#	X	X
Plus sign	+	X	
Minus sign	−	X	
Decimal Point	.	X	
Comma	,	X	
Dollar Sign	$	X	
Four Carets	^^^^	X	
Equal Sign	=		X
Greater Than Sign	>		X

The pound sign ('#') reserves room for a single character in the
output field. If the data item contains more characters than the
number of pound signs in the format field, the entire field will be
filled with asterisks ('*').

```
                    10 PRINT USING "####";X
```

For these values of X, this format displays:

```
                    A = 12.34       12
                    A = 567.89      568
                    A = 123456      ****
```

For a STRING item, the string data is truncated at the bounds of the
field. Only as many characters are printed as there are pound signs
in the format item. Truncation occurs on the right.

The plus ('+') and minus ('-') signs can be used in either the first
or last position of a format field but not both. The plus sign is
printed if the number is positive. The minus sign is printed if the
number is negative.

If a minus sign is used and the number is positive, a blank is printed
in the character position indicated by the minus sign.

If neither a plus sign nor a minus sign is used in the format field
for a numeric data item, a minus sign is printed before the first
digit or dollar symbol if the number is negative and no sign is
printed if the number is positive. This means that one more character
is printed if the number is positive. If there are too many digits to
fit into the field specified by the pound sign and +/- signs, then an
overflow occurs and the field is filled with asterisks ('*').

A decimal point ('.') symbol designates the position of the decimal
point in the number. There can be only one decimal point in any format
field. If a decimal point is not specified in the format field, the
number is rounded to the nearest integer and printed without any
decimal places.

When a decimal point is specified, the number of digits preceding the
decimal point (including the minus sign, if the number is negative)
must not exceed the number of pound signs before the decimal point. If
there are too many digits an overflow occurs and the field is filled
with asterisks ('*').

A comma (',') allows placing of commas in numeric fields. The position
of the comma in the format list indicates where the commas appears in
a printed number. Only commas within a number are printed. Unused
commas to the left of the first digit appear as the filler character.
At least one pound sign must precede the first comma in a field.

If commas are specified in a field and the number is negative, then a
minus sign is printed as the first character even if the character
position is specified as a comma.

FIELD	EXPRESSION	RESULT	COMMENT
##.#	-.1	-0.1	Leading zero added
##.#	1	1.0	Trailing zero added
####	-100.5	-101	Rounded to no decimal places
###.	10	10.	Decimal point added

```
    #$##          1          $1         Leading dollar sign
    ####         -1000      ****        Overflow because 4 digits and
                                        minus sign don't fit in field
```

A dollar sign ('$') symbol shows that a dollar sign will be printed in
the number. If the dollar sign is to float (always be placed before
the number), specify at least one pound sign before the dollar sign.
If a dollar sign is specified without a leading pound sign, the dollar
sign is printed in the position shown in the format field. If commas
and/or a plus or minus sign is specified in a format field with a
dollar sign, the program prints a comma or sign before the dollar
sign. The four up arrows or carets symbol is used to specify that the
the number is to be printed in E format (scientific notation). A pound
sign must be used in addition to the four up arrows to specify the
field width. The arrows can appear either before or after the pound
sign in the format field. Four carats must be specified when a number
is to be printed in E format. If more than one but fewer than four
carats are specified, a syntax error results. If more than four carats
are specified only the first four are used. The fifth carat is
interpreted as a no text symbol. An equal sign ('=') is used to
center a string in a field. The field width is specified by the number
of characters (pound sign and =) in the format field. If the string
contains fewer characters than the field width, the string is centered
in the field. If the string contains more characters that can be fit
into the field, then the rightmost characters are truncated and the
string fills the entire field. A greater than sign ('>') is used to
right justify a string in a field.

```
        5 X=32: Y=100.23: A$="TEST"
        10 PRINT USING "$##.## ";13.25,X,Y
        20 PRINT USING "###>#";"CBM",A$
```

When this is RUN, the following output appears on the screen:

```
        $13.25 $32.00 $*****
          CBM TEST
```

$***** is printed instead of Y because Y has 5 digits, which exceeds
the format specification. The second line asks for the strings to be
right justified, which they are.

PUDEF -- Redefine PRINT USING symbols

                    PUDEF definition_string

PUDEF allows redefinition of up to 4 symbols in the PRINT USING
statement. Blanks, commas, decimal points, and dollar signs can be
changed into some other character by placing the new character in the
correct position in the PUDEF definition_string.

Position 1 is the filler character. The default is a space character.
Place another character here to be used instead of spaces. Similarly,

Position 2 is the comma character. Default is a comma.
Position 3 is the decimal point.
Position 4 is the dollar sign.

```
        10 PUDEF "*"          PRINTs * in the place of blanks.
        20 PUDEF " @"         PRINTs @ in place of commas.
```

QUIT -- [*** UNIMPLEMENTED ***]

RCLR -- Get the current screen color

                        RCLR(source)

              [*** CURRENTLY UNIMPLEMENTED ***]

This function returns the color assigned to source as an number in the
range of 0-15. The color sources are:

                    0 = background
                    1 = foreground
                    2 = multicolor 1
                    3 = multicolor 2
                    4 = border
                    5 = highlight color

RDOT -- Get the current position or color of the pixel cursor

                        RDOT(source)

              [*** CURRENTLY UNIMPLEMENTED ***]

This function returns information about the current pixel location.

                    0 = current X position
                    1 = current Y position

READ -- Read data from DATA statements

                      READ variable_list

READ statements are used along with DATA statements. READ statements
read data from DATA statements into variables, just like an INPUT
statement reads data typed by the user. READ statements can be used in
direct or program mode, but DATA statements must be in a program.

The variable types in the variable_list must match the type of DATA
being read, or a 'TYPE MISMATCH' error is reported. If there are
insufficient data in the program's DATA statements to satisfy all
of the variables in the READ statement, an 'OUT OF DATA' error is
reported.

The computer maintains a pointer to the next DATA item to be read by a
READ statement. Initially this pointer points to the beginning of
the program. As each variable in a READ statement is filled, the
computer moves the DATA pointer to the next DATA item. If all of a
READ statement's variables are filled before all of the data has been
read from a DATA statement, the next READ statement will begin reading
data at the point where the previous READ stopped.

The DATA pointer can be changed by the RESTORE command. It can be
reset back to the beginning of the program, or pointed to a specific
line number. See RESTORE.

    10 DATA 100, 200, FRED, "HELLO, MOM", , 3.14, ABC123, -1.7E-9
    20 READ X,Y

```
 30 READ NAME$, MSG$, NULL$
 40 READ PI, JUNK$, S
 50 RESTORE
```


RECORD -- Specify a relative disk file record number

          RECORD #logical_channel_number, record [,byte]

This command allows you to access any part of any record in a RELative
type disk file. If the byte parameter is omitted, the access pointer
is pointed at the first byte of the specified record number.

Before you can use RECORD, you must OPEN a file. See OPEN and DOPEN
for instructions. Also refer to your DOS manual for an explanation of
RELative type files.

```
 10 INPUT"ENTER RELATIVE FILENAME: ",F$  get name of existing file
 20 DOPEN#1, (F$),L: PRINT DS$           open it & display disk status
 30 R=1: INPUT"ENTER RECORD NUMBER: ",R  get a record number
 40 B=1: INPUT"ENTER BYTE (RETURN): ",B  get byte number, if any
 50 RECORD#1, R,B                        position file pointer
 60 INPUT#1, REC$                        read the record
 70 PRINT REC$                           display the record
 80 PRINT "CONTINUE? (Y/N)"
 90 GETKEY A$: IF A$="Y" THEN 30
100 DCLOSE#1                             close the file
```


REM -- Place an explanatory remark or comment in a program

                    REM plain text message

The REMark command is just a way to leave a note to whomever is
reading a LISTing of the program. It might explain a section of the
program, give information about the author, etc.

REM statements in no way effect the operation of the program, except
to add length to it (and therefore slow it down a little). No other
executable statement can follow a REMark on the same line.

```
 10 REM  THIS PROGRAM WAS WRITTEN ON 2/14/91 BY F.BOWEN
 20 REM  SAMPLE PROGRAM
 30 :
 40 DIR                    :REM  DISPLAY THE DISK DIRECTORY
 50 LIST "SAMPLE PROGRAM"  :REM  DISPLAY THIS PROGRAM
 60 END
```


RENAME -- Rename a disk file

        RENAME "oldname" TO "newname" [,Ddrive] [<ON|,>Udevice]

The RENAME command changes the name of a file in the disk directory.
Pattern matching is not allowed, and "newname" must be a valid
filename that does not already exist on the disk. The file being
renamed does not need to be open.

```
        RENAME "TEST" TO "FINALTEST"
        RENAME (OLD$) TO (OLD$+".OLD") ON U(DEV)
```

RENUMBER -- Renumber the lines of a BASIC program

   RENUMBER [new_starting_line [,[increment] [,old_starting_line]]]

Renumber is used to resequence the line numbers of a BASIC program in
memory. All or part of a program can be renumbered. The RENUMBER
command first scans the program to make sure all the line numbers
referenced in commands (such as GOTO, GOSUB, TRAP, etc.) exist, that
new line numbers are in the legal range, and that changing the program
would not overflow the available memory. An 'UNRESOLVED REFERENCE',
'LINE NUMBER TOO LARGE', or 'OUT OF MEMORY' error is reported if
there's a problem, and RENUMBER is automatically canceled without
having changed anything.

If the program passes all the checks, RENUMBER changes the specified
line numbers and updates all references to the old numbers throughout
the program and relinks the program.

The new_starting_line is the number of the first line in the program
after renumbering. It defaults to 10. The increment is the spacing
between line numbers (eg., 10, 20, 30 would mean an increment of 10).
It also defaults to 10. The old_starting_line is the line number in
the program where you want renumbering to begin.

RENUMBER can be used in direct (edit) mode only. Note that line number
zero (0) is a valid line number.

        RENUMBER   Renumbers the entire program. After
                                    renumbering, the first line will be 10
                                    the second 20, etc. through the end
                                    of the program.

        RENUMBER ,1   Renumbers the entire program as above,
                                    but in increments of one. The first
                                    line will be 10, the second 11, etc.

        RENUMBER 100, 5, 80   Starting at line 80, renumbers the
                                    program. Line 80 becomes line 100,
                                    and lines after that are numbered in
                                    increments of 5, through the end of
                                    the program.

        RENUMBER ,,65   Starting at line 65, renumbers lines
                                    in increments of 10, starting at line
                                    10 through the rest of the program.

RESTORE -- Position READ pointer at specific DATA statement

                        RESTORE [line]

The computer maintains a pointer to the next DATA item to be read by a
READ statement. Initially this pointer points to the beginning of
the program. The DATA pointer can be changed by the RESTORE command.

Using RESTORE without specifying a line number will reset the DATA
pointer back to the beginning of the program. If a line number is
specified, the DATA pointer is pointed to that line. The line does not
have to contain a DATA statement. When the computer executes the
next READ statement, it will look for the next DATA item starting at

the line the DATA pointer is at.

See the READ command an example.

RESUME - Resume program execution after error TRAP

                         RESUME [line|NEXT]

Used to return to execution after TRAPping an error. If a line number
is given, the computer performs a 'GOTO line' and resumes execution at
that line. RESUME NEXT resumes execution at the statement following
the one that cause the error. RESUME without any parameters will
resume execution at the statement that cause the error.

If the computer encounters a RESUME statement outside of a TRAP
routine or if a TRAP was not in effect a 'CAN'T RESUME' error is
reported. RESUME can only be used in program mode.

                    10 TRAP 90
                    20 FOR I=-5 TO 5
                    30 PRINT 5/I
                    40 NEXT
                    50 END
                    60 :
                    90 PRINT ERR$(ER): RESUME NEXT

RETURN -- Return from subroutine or event handler

                              RETURN

This statement is associated with the GOSUB (GO SUBroutine) statement.
When a subroutine is called by a GOSUB statement, the computer
remembers where it's at before it calls the subroutine. When the
computer encounters a RETURN statement, it returns to the place it
last encountered a GOSUB and continues with the next statement.

If there wasn't a previous GOSUB, then a 'RETURN WITHOUT GOSUB' error
is reported.

RETURN is also used by event handlers, set up by the COLLISION
command. See COLLISION.

RGR -- Get the current graphic mode

                              RGR(0)

                   [*** CURRENTLY UNIMPLEMENTED ***]

This function returns current graphic mode. A result of zero means the
display is text, a non-zero result means it's graphic.

RIGHT$ -- Get the rightmost characters of a string

                        RIGHT$ (string,count)

This function returns a string containing the rightmost 'count' number
of characters of the string expression. Count is an numeric expression

in the range (0-255). If count is greater than the length of the
string, the entire string will be returned. If count is zero, a null
(empty) string will be returned.

```
        A$ = RIGHT$("123ABC",3)         Result is A$="ABC"
```


RMOUSE -- Get the mouse position and button status

```
            RMOUSE [Xposition [,Yposition [,button]]]

            X,Yposition = current position of mouse pointer sprite
            Button      = current status of mouse buttons
                            0   = no button
                            1   = right button
                            128 = left button
                            129 = both buttons
```
RMOUSE is a command which retrieves a mouse's current position and the
state of its buttons, and places this information into the specified
numeric variables. If a mouse is not installed, "-1" is returned for
all variables. If both ports are enabled, buttons from each port are
merged. Use the MOUSE command to turn a mouse on or off.

```
   10 MOUSE ON, 2, 1           Turn mouse on, port 2, sprite 1
   20 DO                       Begin loop
   30 RMOUSE X, Y, B           Get mouse position & buttons
   40 PRINTUSING"### ";X,Y,B   Show   "         "         "
   50 LOOP UNTIL B=129         Loop until user presses both buttons
   60 MOUSE OFF                Turn mouse off
```


RND -- Get a pseudo-random number

```
                    RND (type)
```

The RND function returns a pseudo RaNDom number between 0 and 1. The
random sequence returned is determined by the type parameter:

```
  type = 0    Returns a random number based upon the system clock.
  type < 0    Negative numbers "seed" the random number generator,
              defining a new but reproducible random sequence.
  type > 0    Positive numbers draw the next random number from the
              sequence defined by the last "seed" value.
```

This lets a programmer use a reproducible sequence while debugging
(fixing) a program, so that random errors can be easily reproduced.
Once the program has been fixed, it can be "seeded" such that a random
sequence is used every time the program is run.

```
            10 DO
            20 INPUT "SEED"; S
            30 IF S=0 THEN END
            40 FOR I=1 TO S
            50 PRINT INT(RND(1)*6)+1, INT(RND(1)*6)+1
            60 NEXT
            70 LOOP
```

The above program will demonstrate the results of seeding the random
number generator. It lets you specify a positive or negative seed
value, and then prints the first S random pairs of that sequence.
Enter a zero to end the program. The calculations in line 50 make the

random numbers be integers from 1 to 6, like dice. Type in a negative
dice from that sequence. Every time you enter "-1", for example, you
will roll the same numbers:

                      first roll   2 and 6
                      second   6 and 1
                      third   1 and 1
                      fourth   1 and 4
                      fifth   5 and 5

Games and statistical programs should use RND(0) for true randomness
or seed the generator with a random number, such as RND(-TI).

The general form for getting random integers using RND is:

                    INT(RND(0) * MAX) + 1

where MAX is the highest number you can get. This gives you numbers as
low as 1 and as high as MAX. For dice, MAX is 6 (or 12 if you want
to simulate rolling two dice at once). For cards, MAX is 52.

                    INT(RND(0) * 16)

This form will return integers from zero to 15, which is useful for
generating random colour values, for example.

RREG -- Get register data after a SYS call

        RREG [a_reg] [,[x_reg] [,[y_reg] [,[z_reg] [,status]]]]

Following a SYS call, the RREG command retrieves the contents of the
microprocessor's registers and puts them into the specified numeric
variables. See the sample program at SYS.

RSPCOLOR -- Get multicolor sprite colors

                    RSPCOLOR (multicolor#)

Returns the current colors for multicolor sprites. Color values range
from 0-15. Use RSPRITE function to get the foreground sprite color.

                multicolor# = 1 gets multicolor #1
                multicolor# = 2 gets multicolor #2

See SPRITE and SPRCOLOR.

RSPPOS -- Get the location and speed of a sprite

                    RSPPOS (sprite,parameter)

The RSPPOS function returns the current X or Y position of a sprite
and its speed, set by the MOVSPR command. A sprite does not have to
be on to use RSPPOS. The sprite number must be in the range of 0-7,
and the parameter is:

                0 to get current X position
                1 to get current Y position
                2 to get current speed (0-255)

RSPRITE -- Get information about a sprite

                    RSPRITE (sprite,parameter)

The RSPRITE function returns the current state of a sprite, set by the
SPRITE command. The sprite number must be in the range of 0-7, and the
parameter is:

        0   to see if it's turned on   (1)=yes (0)=no
        1   to get sprite foreground color   (0-15)
        2   to get priority over background   (1)=yes (0)=no
        3   to get X-expansion factor   (1)=yes (0)=no
        4   to get Y-expansion factor   (1)=yes (0)=no
        5   to get multicolor factor   (1)=yes (0)=no

RUN -- execute BASIC program

                RUN [line #]
                RUN "filename" [,Ddrive] [<On|,>Udevice]

RUN executes the BASIC program that is currently in memory. The
program has to be LOADed (DLOAD) or manually typed in before it can
be executed. If a line number is specified, execution begins at that
line. If a filename is specified, the program is automatically loaded
from disk into memory and executed. RUN can be used in a program.

RUN clears all variables and open channels (but it does NOT properly
close open disk write files -- used DCLOSE or DCLEAR beforehand). RUN
also resets the runtime stack pointer (clears GOSUB & FOR/NEXT stacks)
the DATA pointer, and the PRINT USING characters. To start a program
without initializing everything, use GOTO.

    RUN   Starts the program at the first line.
    RUN 100   Starts the program at line 100.
    RUN "TEST"   Loads the program TEST from the, default system
                     disk and starts the program at the first line.

RWINDOW -- Get information about the current text window

                    RWINDOW (parameter)

This is a function that returns information about the current console
text display. The parameter is specified as:

        0   to get the maximum line # in the current window
        1   to get the maximum column # in the current window
        2   to get the screen size, either 40 or 80 columns

SAVE -- Save a BASIC program in memory to disk

            SAVE "[[@]drive:]filename" [,device_number]

This   command copies a BASIC program in the computer's BASIC memory
area   into a PRoGram-type disk file. If the file already exists, the
program is NOT stored and the error message 'FILE EXISTS' is reported.
If the filename is preceded with an '@0:', then if the file exists it

will be replaced by the program in memory. Because of some problems
with the 'save-with-replace' option on older disk drives, using this
option is not recommended if you do not know what disk drive is being
used (DELETE the file before SAVEing). Pattern matching is not
allowed. In the case of dual drive systems, the drive number must be
part of the filename.

Use the VERIFY or DVERIFY command to compare the program in memory
with a program on disk. To save a binary program, use the BSAVE
command.

    SAVE "myprogram"   Creates the PRG-type file MYPROGRAM
                            on the default system disk and copies
                            the BASIC program in memory into it.

    SAVE "@0:myprogram"   Replaces the PRG-type file MYPROGRAM
                            with a new version of MYPROGRAM. If
                            MYPROGRAM doesn't exist, it's created.

    SAVE F$,9   Saves a program whose name is in F$
                            on disk unit 9.

SCALE -- Set the logical dimension of the graphic screen

                    [*** NOT YET IMPLEMENTED ***]

SCNCLR -- Clear a text or graphic screen

                        SCNCLR [color]

This command will clear the current text window if [color] omitted,
otherwise it will clear the current graphic screen using the given
color value. See also SCREEN CLR.

    SCNCLR   Clears the text screen. If a window is defined
                    it clears only the window area.

    SCNCLR 0   Clears the current graphic screen with color 0.

SCRATCH -- Delete files from disk directory
           Recover accidentally deleted files

            SCRATCH "filespec" [,Ddrive] [<ON|,>Udevice] [,R]

SCRATCH, ERASE, or DELETE are different names of the same command.
They are used to delete a file from a disk directory, or optionally to
recover if possible an accidentally deleted file. The diskette must
not be 'write protected', or a 'WRITE PROTECT ON' error is reported.

WARNING: Deleting a file will destroy all existing data in that file.
Be extremely careful if you are using pattern matching, which can
delete any or all files. In direct mode, you are asked to confirm what
you are doing with 'ARE YOU SURE?'. Type 'Y' and press <RETURN> to
proceed, or type any OTHER CHARACTER and press <RETURN> to cancel the
command. In program mode there is no confirmation prompt.

Upon completion, in direct mode only, the computer will display the
number of files deleted.

Refer to your disk manual for other details. Different disk drives
implement slightly different pattern matching rules or support
features such a specially protected files.

If the 'R'ecover option is present and the DOS supports it, a deleted
file can be recovered if nothing else has been written to the diskette
since the file was accidentally deleted. You will still be asked to
confirm the operation, and upon completion the computer will display
the number of files restored.

    SCRATCH "oldfile"   Deletes the file OLDFILE from the disk
                             in the default system drive.

    SCRATCH "file.*"   Deletes all files beginning with FILE.

    SCRATCH (F$), U(DD)   Deletes the file whose name is in F$
                             from the disk in device DD.

    SCRATCH "SAVEME", R   Attempt to recover the program SAVEME.

SCREEN -- Graphic command

The SCREEN command is used to initiate a graphic command. It always
precedes another command word which identifies the graphic operation
to be performed:

SCREEN CLR   - Set graphic screen color

                        SCREEN CLR color#

Clears (erases) the currently opened graphic screen using the given
color value. Use SCNCLR to clear a text screen. See also SCNCLR.

SCREEN DEF   - Define a graphic screen

                   SCREEN DEF screen#, width, height, depth

                    screen#   0-1
                    width   0=320, 1=640, 2=1280
                    height   0=200, 1=400
                    depth   1-8 bitplanes (2-256 colors)

Defines a logical screen (numbered 0 or 1), specifies its size and how
many colors (bitplanes) it has. It does not allow access to the
screen and it does not display the screen. The screen must be defined
before it is opened for viewing and/or drawing to.

SCREEN SET   - Set draw and view screens

                   SCREEN SET DrawScreen#, ViewScreen#

                        draw screen #   0-1
                        view screen #   0-1

This command specifies which logical screen is to be viewed and which
logical screen is to be accessed by the various draw commands. The
screen must be defined and opened first. Both the draw and the view
screen can be, and usually are, the same logical screen. For double
buffering, they are different.

```
SCREEN OPEN      - Open a screen for access

                    SCREEN OPEN  screen# [,error_variable]

                    screen#          0-1
                    error_variable  [*** NOT YET IMPLEMENTED ***]

This  command  actually sets up the screen and allocates the necessary
memory for it.  If it's the view screen it will be displayed.  If it's
the draw screen, it can now be drawn to. If there is not enough memory
for the screen,  'NO GRAPHICS AREA'  is reported and the screen is not
opened.

SCREEN CLOSE     - Close a screen

                        SCREEN CLOSE  screen#

                        screen#          0-1

This command closes a logical screen,  ending access to it by the draw
commands if it's the draw screen and restoring the text screen if it's
the view screen.  SCREEN CLOSE deallocates any memory reserved for the
screen.

                    SAMPLE GRAPHIC PROGRAM:


  1 TRAP 170                          in case of error want text screen
 10 GRAPHIC CLR                       initialize graphics
 20 SCREEN DEF 1,0,0,2                define a 320x200x2 graphic screen
 30 SCREEN OPEN 1                     open it
 40 PALETTE 1,0,  0, 0, 0             define screen 1 color 0 = black
 50 PALETTE 1,1, 15, 0, 0             define screen 1 color 1 = red
 55 PALETTE 1,2,  0, 0,15             define screen 1 color 2 = blue
 60 PALETTE 1,3,  0,15, 0             define screen 1 color 3 = green
 70 SCREEN SET 1,1                    make it the view screen
 80 SCNCLR 0                          clear screen with palette color 0
 90 BORDER 0                          set border color to color 0
100 PEN 0,1                           make draw pen = color 1 (red)
110 LINE 100,100, 150,150             draw a diagonal red line
120 PEN 0,2                           make draw pen = color 2 (blue)
130 BOX 50,50, 50,80, 80,50, 80,80   draw a blue box
140 PEN 0,3                           make draw pen = color 3 (green)
150 CHAR 25,50, 1,1,2, "WORDS"        draw green text
160 SLEEP 5                           pause for 5 seconds
170 SCREEN CLOSE 1                    close graphic, get text screen
180 PALETTE RESTORE                   restore normal system colors
190 BORDER 6                          restore normal border color
200 END



SET -- Set various system parameters

The SET command is used to set a system parameter.  It always precedes
another command word which identifies the parameter to be changed:

SET DEF          - Set default system disk drive

                        SET DEF device

The BASIC  DOS commands default to disk unit 8.  Use SET DEF to change
```

which device these commands default to. This command does not renumber
a disk device, use SET DISK for that. Commands which specify a device
will still access the device they specified. A program can be made
more "user friendly" by either not specifying a drive (thus using the
user's preferred drive) or by specifying device 1. Device number 1
means "use the system default drive, whatever its number is."

```
     10 DIR                   gets directory of device 8
     20 DIR U1                gets directory of device 8
     30 DIR U10               gets directory of device 10
     40 SET DEF 10            change the default drive to unit 10
     50 DIR                   gets directory of device 10
     60 DIR U1                gets directory of device 10
     70 DIR U8                gets directory of device 8
```

SET DISK   - Change a disk device number

                    SET DISK oldnumber TO newnumber

Use this command to renumber (change) a disk drive's unit number. Not
all drives can be renumbered -- refer to your disk drive manual for
details. This command sends to the disk's command channel the
conventional CBM serial disk drive "M-W" command. See also the DISK
command, which lets you send any command to a disk drive.

        SET DISK 8 TO 10   Change unit 8's number to 10

Because the built-in C64DX drives always take precedence over serial
bus drives, this is one way to get the built-in drive "out of the way"
so that you can access a serial bus drive #8.

SGN -- Get the sign of a number

                        SGN (expression)

The SiGN function returns the sign of a numeric expression as follows:

        If the expression is < 0 (negative) .... returns -1
        If the expression is = 0 (zero) .... returns 0
        If the expression is > 0 (positive) .... returns 1

SIN -- Sine function

                        SIN (expression)

This function returns the sine of X, where X is an angle measured in
radians. The result is in the range -1 to 1.

        X = SIN (pi/4)   Result is X=0.707106781

To get the sine of an angle measured in degrees, multiply the numeric
expression by pi/180.

SLEEP -- Pause program execution of a specified period of time

                        SLEEP seconds

Temporarily suspends execution of your program for 1 to 65535 seconds.

SLOW -- Set system speed to 1.02MHz

SLOW is used primarily to directly access "slow mode only" devices
such as the SID sound chips. FAST is the default system speed.

SOUND -- Produce sound effects

            SOUND v, f, d [,[dir] [,[m] [,[s] [,[w] [,p]]]]]

  v   = voice   (1-6)
  f   = frequency   (0-65535)
  d   = duration   (0-32767)
  dir = step direction (0(up), 1(down), or 2(oscillate))   default=0
  m   = min frequency   (0-65535)   default=0
  s   = sweep   (0-65535)   default=0
  w   = waveform   (0=triangle,1=saw,2=square,3=noise) default=2
  p   = pulse width   (0-4095)   50% duty cycle=default=2048

The sound command is a fast and easy way to create sound effects and
musical tones. The first three parameters are required to select the
voice, frequency, and duration of the tone. The duration is specified
in "jiffies" (60 jiffies = 1 second).

Optionally, you can specify a waveform and, for square waves, the
pulse width. The SOUND command can sweep a voice through a series of
equally-spaced frequencies. The direction of the sweep, minimum and
maximum frequencies can be programmed. If time expires before the
sweep is done, the sound stops. If the minimum or maximum frequency is
reached before time expires, the sound repeats.

For programming details, refer to the SID hardware documentation. Use
the VOLume command to change the volume of the sound. Note that the
TEMPO command affects PLAY strings only, not SOUND effects.

                    FREQout = (f * 0.0596) Hz
                    PWout = (p / 40.95) %

Each voice can be programmed separately and played simultaneously for
a wide variety of sound effects. Once a sound effect is initiated,
BASIC execution continues with the next statement while the sound
plays out, allowing you to combine and control graphics, animation,
and sound from a BASIC program. The examples below include information
about how to generate precise tones for exact times, but for most
casual users trial and error are perfectly acceptable! (Note that the
values used are for 60Hz (NTSC) systems):

Using voice 1, emit a square-wave, 440Hz tone for 1 second. Note that
440Hz = 7382 * 0.0596 using the above formula.

                    SOUND 1, 7382, 60

Using voice 2, sweep from 100Hz (m=1638) to 440Hz (f=7382) in
increments of 1Hz (s=17). The time required to do this can be
calculated as t=(f-m)/s, so t=336 jiffies.

                    SOUND 2, 7382, 336, 0, 1678, 17

Using voice 3, make a neat sound using an oscillating sweep (dir=2)

and a sawtooth waveform (w=1) for 3 seconds (t=180).

                    SOUND 3, 5000, 180, 2, 3000, 500, 1

SPC -- Space PRINT output

                         SPC (number)

The SPaCe function is used to format PRINTed data to the screen, a
printer, or a file. It specifies the number of spaces to be skipped,
from 0 to 255. A semicolon (';') is always assumed to follow SPC, even
if it appears at the end of a print line.

The SPC function works a little differently on screen, printer and
disk output. On the screen, SPC skips over characters already on the
screen, which is not the case with printer and disk output. On
printers, if the last character on a line is skipped, the printer
will automatically perform a carriage return and linefeed.

        PRINT "123";SPC(3);"456"   Displays '123   456'

        PRINT "X";SPC(5) :PRINT"X"   Displays 'X   X'

See also the TAB function. A better way to format PRINT output is with
PRINT USING.

SPRCOLOR -- Set multicolor sprite colors

                SPRCOLOR [sprite_mc1] [,sprite_mc2]

Use the SPRITE command to set up a multicolor sprite, and use SPRCOLOR
to set the additional colors. Note that these colors are common to all
multicolor sprites. The color values must be in the range (0-15). Use
the RSPCOL0R function to get the current multicolor sprite colors, and
RSPRITE to get the current sprite foreground color.

SPRDEF -- Define a sprite pattern

                [*** NOT EXPECTED TO BE IMPLEMENTED ***]

SPRITE -- Turn a sprite on or off, and set its characteristics

SPRITE number [,[on] [,[color] [,[priority] [,[x_exp] [,[y_exp] [,mode]]]]]]

The SPRITE command allows you set all of the characteristics of a
sprite. Use the MOVSPR command to position it or set it in motion.
Use the SPRCOLOR to set the multicolor sprite colors, if you are using
multicolor sprites.

All the parameters except the sprite number are optional. If you don't
specify a parameter then it won't be changed.

        number = sprite number   (0-7)
        on   = enable (1) or disable(0)
        color   = sprite foreground color   (0-15)
        priority= sprite to display data priority:
                    0 means sprite goes over screen data

```
                      1 means sprite goes under screen data
        x,y_exp = sprite expansion on (1) or off (0)
        mode    = sprite mode:
                      0 high resolution
                      1 multicolor
```

The SPRITE command does not define a sprite. The sprite definitions
must be loaded into the sprite area first ($600-$7FF). Use the BLOAD
and BSAVE commands. [*** THIS MAY CHANGE ***] A sprite is 24 pixels
wide and 21 pixels high. Each sprite definition requires 63 ($40 hex)
bytes:

```
                    $600     Sprite 0 definition
                    $640    Sprite 1 definition
                    $680     Sprite 2 definition
                    $6C0     Sprite 3 definition
                    $700     Sprite 4 definition
                    $740     Sprite 5 definition
                    $780     Sprite 6 definition
                    $7C0     Sprite 7 definition
```

Use the RSPRITE function to read a sprite's characteristics, or the
RSPPOS function to read a sprite's position. The RSPCOLOR function
is used to get the current multicolor sprite colors.

```
 10  BLOAD"sprite 1 data",          Load sprite 1's definition
     P(DEC("640"))
 20  SPRITE 1, 1, 2                  Turn it on, make it red
 30  MOVSPR 1, 24, 50               Put it at top-leftmost corner
 40  SPRSAV 1, 2                     Copy sprite 1 definition to 2
 50  SPRITE 2, 1, 7                  Turn on sprite 2 make it yellow
 60  MOVSPR 2, 320, 229             Put it at bottom-rightmost corner
 70  BSAVE"sprite 2 data",          Save sprite 2
     P(DEC("680")) TO P(DEC("6C0"))
 80  SPRITE 1, 0                     Turn off sprite 1
 90  SPRITE 2, 0                     Turn off sprite 2
```

SPRSAV -- Copy a sprite definition

                      SPRSAV source, destination

Use this command to copy a sprite's data (shape) to another sprite or
into a string variable, or copy a shape from a string variable into a
sprite. You can have many different sprite shapes in memory at one
time, all stored in strings. This makes it possible to animate
sprites from BASIC by quickly "flipping through" shapes, using each
shape like a frame from a movie film.

     SPRSAV 0, A$ copy the data (shape) of sprite 0 into A$

     SPRSAV A$, 2   copy the data (shape) in A$ into sprite 2

     SPRSAV 1, 2   copy the data (shape) in sprite 1 to sprite 2

STASH -- (see the DMA command)

SQR -- Square root function

SQR (number)

This function returns the SQuare Root of the given numeric expression.
The numeric expression must not be negative or an 'ILLEGAL QUANTITY'
error is reported.

              A = SQR(10)   Result is A = 3.16227766

STEP -- See FOR/NEXT/STEP

STOP -- Halt program execution

When STOP is executed, the computer immediately stops running the
program and reports 'BREAK IN LINE xx'. No variables are cleared and
files are not closed.

This command is usually used while debugging (fixing) a BASIC program,
since it lets you stop at a specific place, examine variables, change
variables, and restart the program where it was halted (see CONTinue
command) or some other line (see GOTO). In many cases, you can even
change the program and use GOTO to resume execution with variables
and open channels intact.

SWAP -- (see the DMA command)

STR$ -- Get the string representation of a number

                        STR$ (number)

The STRing function returns a string identical to PRINT's output of
the given numeric expression. See PRINT for details regarding the
format of numeric output. STR$ is the opposite of VAL.

            A$ = STR$(123)   Result is A$ = " 123"
            A$ = STR$(-123)   Result is A$ = "-123"
            A$ = STR$(.009)   Result is A$ = " 9E-03"

SYS -- Call a ROM routine or user machine language routine

          SYS address [,[a] [,[x] [,[y] [,[z] [,s]]]]]

This statement performs a call to a machine language routine at the
specified address (range 0-65535, $3000-$FFFF) in a memory bank set
up previously by the BANK command.

The microprocessor's registers are loaded with the values specified in
the parameters following the address (if given) and a JSR (Jump
SubRoutine) instruction is performed. When the called routine ends
with an RTS (ReTurn from Subroutine), the microprocessor's registers
are saved and control is returned to the BASIC program. The
microprocessor's registers can be examined with the RREG command.

Because this command instructs the computer's microprocessor (CPU) to
perform something, extreme care should be taken in its use. It can
easily crash the computer if you do something wrong (press the reset
button to reboot). Also see the BOOT SYS command.

```
BANK 128: SYS DEC("FF5C")        Call the Kernel's PHOENIX routine.

BANK 128: SYS DEC("FF81")        Reset the Screen Editor


10 BANK 128
20 BLOAD"user routine",P(DEC("1800"))   Load a user routine
30 SYS DEC("1800"), areg, xreg          Call it with args in A and X
40 RREG areg, xreg, , , sreg            Get args back in A, X, and S
50 carry = (sreg AND 1)                 Get carry flag from S
60 PRINT "ACCUMULATOR = ";HEX$(areg)    Display registers
70 PRINT "X REGISTER  = ";HEX$(xreg)
80 PRINT "CARRY FLAG  = ";carry
```

See the USR function for another way to call machine language
routines.

TAB -- Space PRINT output

                          TAB (number)

The TAB function is used to format PRINTed data to the screen, a
printer, or a file. It's primarily for screen text output, moving
the cursor to the specified column (plus one) as long as the current
print position is not already beyond that point (for example, if the
current print position is the first column, TAB(1) would print
subsequent text beginning in column 2). If the current print position
is already beyond the column specified by the TAB function, nothing is
done. For disk and printer output, TAB works exactly like the SPC
function (see SPC).

A semicolon (';') is always assumed to follow TAB, even if it appears
at the end of a print line.

    PRINT "TEXT";TAB(10);"HERE"   Result is 'TEXT   HERE'

    PRINT "TEXT";SPC(10);"HERE"   Result is 'TEXT   HERE'

The above examples illustrate the difference between TAB and SPC. See
also the SPC function. A better way to format PRINT output is with
PRIUT USING. Don't confuse the TAB function with the TAB character,
CHR$(9), which is used to format data using the programmable TAB
stops.

TAN -- Tangent function

                          TAN (expression)

This function returns the tangent of the numeric expression, measured
in radians. If the result overflows, TAN(pi/2) for example, an
'OVERFLOW' error is reported.

          X = TAN(1)   Result is X=1.55740772

To get the tangent of an angle measured in degrees, multiply the
numeric expression by pi/180.

TEMPO -- Set the tempo (speed) of a PLAY string

TEMPO rate

Use this command to adjust the tempo (speed) of music playback by the
PLAY command. The rate determines the duration of a whole note. The
default is 12, making a whole in 4/4 time last 2 seconds. The formula
is:
                        duration = 24/rate

The higher the rate, the faster the note. The range is (1-255).

THEN -- See IF/THEN/ELSE

TO -- See FOR/NEXT/STEP. Also used as a subcommand.

TRAP -- Define an BASIC error handler

                        TRAP [line_number]

When turned on, TRAP intercepts all BASIC execution error conditions
except 'UNDEF'D STATEMENT ERROR'. Even the STOP key can be TRAPped.

When an error occurs, BASIC saves the error's location, line number,
and error number. If TRAP is not set, BASIC returns to direct mode
and displays the error message and line number. If TRAP is set, BASIC
performs a GOTO to the line number specified in the TRAP statement and
continues executing.

Your BASIC error handling routine can examine the error number,
message, and the line number where the error occurred and determine
the proper course of action. The system error words are:

        ER   Error Number
        EL   Error Line (line where the error occurred)
        ERR$() Error Message

If ER is -1, then a BASIC error did not occur. The error routine
should check the disk status words, in case they were the cause of
the error:

        DS   Disk Error Number
        DS$ Disk Error Message

Refer to the list of BASIC and Disk error messages in the appendix.

Note that an error in your TRAP routine cannot be trapped. The RESUME
statement can be used to resume execution -- see RESUME.

TRAP with no line number specified turns off error TRAPping.

        10 TRAP 90   enable trapping
        20 FOR I=-5 TO 5
        30 PRINT 5/I   error when I=0
        40 NEXT
        50 TRAP   turn trapping off
        60 END
        70 :
        90 PRINT ERR$(ER): RESUME NEXT   error routine

```
TROFF -- Turn off trace mode
TRON  -- Turn on  trace mode
```

```
                        TROFF
                        TRON
```

Trace mode is used while debugging (fixing) a BASIC program. TRON
enables tracing, and TROFF disables tracing. When the program is run
and trace mode is on, the line number of the command that is being
executed is displayed on the screen. If there are three commands on
the line, the line number will be displayed three times, once each
time one of the commands is executed. Trace mode lets you know what
the computer is doing.

Trace mode works even when a graphic screen is being displayed, but
the line number is still displayed on the text screen so you won't be
able to see it until the graphic screen is turned off. If your program
is doing alot of PRINT statements, the display can seen a little
confusing.

Trace mode can be set in direct mode to trace the entire program, or
it can be turned on and off from within your program to let you trace
only selected portions of the program.

Trace mode has no effect on commands entered in direct (edit) mode.
The NEW command disables trace mode, but RUN and CLR do not.

```
                    10 FOR I=-5 TO 5
                    15 TRON
                    20 PRINT 5/I
                    25 TROFF
                    30 NEXT
```


TYPE -- Display the contents of a sequential disk file

```
            TYPE "filename" [,Ddrive] [<,|ON>Udevice]
```

Use this command to print the contents of a PETSCII data file on the
screen. The file must contain lines no longer than 255 characters long
and terminated by a return character (CHR$(13)). Lines too long result
in a 'STRING TOO LONG' error.

TYPE "readme"   display the contents of the README file on the screen

The command sequence below will print the contents of the README file
on a CBM serial bus printer in upper/lower case mode.

```
        OPEN 4,4,7: CMD4: TYPE"readme": CLOSE4
```


UNTIL -- See DO/LOOP/WHILE/UNTIL/EXIT

USR - Call a user defined machine language function

```
                    USR (expression)
```

When this function is used, the program jumps to a machine language

subroutine whose starting address must be POKEd into system memory (BANK 128) at address 760 (low byte) and 761 (high byte), or $2F8 hex. The floating point value of the numeric expression is passed to the routine in the Floating point ACCumulator (FACC), and the value to be returned is taken from the FACC when the routine ends.

If the USR vector is not set up prior to making the USR call, an 'UNDEF'D FUNCTION' error is reported. The routine must be located in the system bank. The BANK command does not affect USR.

Using this method of calling a machine language routine requires a fair amount of set up and a good knowledge of the lower level math routines built into BASIC. See the SYS command, which is more commonly used to call a machine language routine.

The following program illustrates the basic steps required for installing a USR routine and calling it:

```
10 BANK 128                          System bank for poke & load
20 UV = DEC("1800")                  Where my routine is
30 BLOAD "my user routines",P(UV)    Load my routine
40 POKE DEC("2F8"), UV AND 255, UV / 256  Set up USR address
50 x = USR(123): PRINT X             Call my routine with the
                                     the value 123, get back and
                                     print whatever my routine
                                     leaves in FACC
```

The following program actually works. It points the USR vector to the BASIC math jump table entry for the routine which inverts the sign of the number in the FACC. Type in positive & negative numbers:

```
10 BANK 128                          System bank for poke
20 POKE DEC("2F8"), DEC("33"), DEC("7F")  Set up USR address
30 DO: INPUT"SIGNED NUMBER"; N        Get number input
40 : PRINT USR(N)                     Display USR output
50 : LOOP UNTIL N=0                   End if user types zero
```


USING -- See PRINT USING

VAL -- Get the numerical value of a string

                         VAL (string)

The VALue function converts a string into a number. The conversion starts with the first character and ends at the end of the string or the first character that is not allowed in normal number input. Spaces are ignored. If the first character of the string is not a legal character, a zero is returned.

The VAL function works the same way the INPUT and READ commands do. VAL is the opposite of STR$.

             X = VAL(" 123")   Result is X = 123
             X = VAL("-123")   Result is X = -123
             X = VAL(" 9E-02")   Result is X = .09

VERIFY -- Compare a program or data in memory with a disk file

```
                VERIFY "filename" [,device_number [,relocate_flag]]
```

This command is just like a LOAD command, except instead of putting
the data read from a file into memory, the computer compares it to
what is already in memory. If there's any difference at all a 'VERIFY
ERROR' is reported.

The filename must be given, and pattern matching may be used. In the
case of dual drive systems, the drive number must be part of the
filename. If a device number is given, the file is sought on that
unit, which must be a disk drive. If a device number is not given,
the default system drive is used. See also DVERIFY.

Note: If the BASIC program in memory is not located at the same
address as the version on disk was SAVEd from, the files will not
match even if the program is otherwise identical.

The relocate_flag is used to VERIFY binary files. If the relocate_flag
is present and non-zero, the file will be compared to memory starting
at the address stored on disk when the file was SAVEd. The memory bank
used is the bank given in the last BANK statement. The ending address
is determined by the length of the disk file. The comparison halts
on the first mismatch or at the end of the file. The area to be
compared must be confined to the indicated memory bank. Do not use
the relocate_flag to verify BASIC programs. See also BVERIFY.

                            VERIFY "myprogram"

Good: SEARCHING FOR 0:myprogram   Bad: SEARCHING FOR 0:myprogram
         VERIFYING   VERIFYING
         OK   ?VERIFY ERROR

  VERIFY "PROG"   Compares BASIC program in memory to file PROG
                         on the default system disk.

  VERIFY FILE$,DRV   Compares program in memory to a program whose
                         name is in the variable FILE$ on the unit
                         whose number is in DRV.

  VERIFY "0:PROG",8   Compares memory to BASIC program PROG on unit
                         8, drive 0.

  BANK 128   Compares a binary file into memory. The
  VERIFY "BIN",8,1   address used comes from the disk file, but
                         you must specify the memory bank.

VIEWPORT --   [*** CURRENTLY UNIMPLEMENTED ***]

VOL -- Set audio volume level

                            VOL volume

                 [*** THIS COMMAND WILL CHANGE ***]

This statement sets the volume level for SOUND and PLAY statements.

VOLUME can be set from 0 to 15, where 15 is the maximum volume. A
volume of 0 turns sound output off. VOLume affects all 3 voices.
Note that PLAY strings can change the volume, too.

WAIT -- Pause BASIC program until a memory state satisfied

                    WAIT address, and_mask [,xor_mask]

The WAIT statement causes program execution to be suspended until data
at a specified memory location matches a given bit pattern. It's used
to pause your program until an event occurs.

The event could be an I/O state (such as a fire button or peripheral
port change), a hardware state (such as the raster position or RS232
status), or memory change caused by an interrupt event (such as a
keyboard scan).

The WAIT statement tells the computer to read (PEEK) a memory location
(0-65535) and AND the value it got with the number in and_mask
(0-255). If the result is zero, repeat the operation until the result
is not zero. This is like the following BASIC instructions, but much
faster:

  DO: result = PEEK(address): LOOP UNTIL (result AND and_mask) <> 0

This works if the state you are WAITing for is non-zero (a one or
"high" state). If you want to wait for a zero state (a "low" state),
you need to use the xor_mask option to "flip" the bits of the result.

Note that it's possible to "hang" your program indefinitely if the
state you are waiting for never happens or you specify the wrong data.
Press the STOP and RESTORE keys at the same time to get control back.

Be sure to use the BANK command before you tell the computer to WAIT
to specify which 64K memory bank the address is in. Note that a BANK
number greater than 127 (i.e., a bank number with the most significant
bit set) must be used to address an I/O location, such as the VIC
chip. Refer to the system memory map for details.

     10 BANK 128   Wait for the VIC raster to be
     20 WAIT DEC("D011"), 128   offscreen (want RC8 = 1)

     10 BANK 128   Wait for the VIC raster to be
     20 WAIT DEC("D011"), 128, 128 onscreen (want RC8 = 0)

     10 BANK 128
     20 WAIT DEC("D3"), 1   Wait for user to press <SHIFT> key
     30 WAIT DEC("D3"), 2   Wait for user to press <C=> key
     40 WAIT DEC("D3"), 4   Wait for user to press <CTRL> key
     50 WAIT DEC("D3"), 8   Wait for user to press <ALT> key

WHILE -- See DO/LOOP/WHILE/UNTIL/EXIT

WIDTH --   [*** CURRENTLY UNIMPLEMENTED ***]

WINDOW -- Set a text window

   WINDOW left_column, top_row, right_column, bottom_row [,clear]

This command defines a logical text screen window. All text I/O will

be confined to this window. The row parameters must be in the range
(0-24), and the column parameters must be in the range (0-79) for
80-column screens or (0-39) for 40-column screens. The parameters are
always referenced to the physical screen (i.e., you cannot define a
window within a window). If the clear flag is given, the new window
area will be cleared after it's set up.

Use the RWINDOW function to get the current window size.

You are responsible for saving and restoring screen data in all
windows because the WINDOW command simply sets the window margins. The
WINDOW command does not draw a border around a window. All color
commands and screen modes (such as scroll disable, TAB stops, etc.)
are global.

Two consecutive "home" characters will reset the window definition
back to the physical screen.

      WINDOW 0,0,39,24   Define a window in 80-column mode
                                  that is the left half of the screen.

      WINDOW 40,0,79,24   Define a window in 80-column mode
                                  that is the right half of the screen.

      WINDOW 0,0,79,12   Define a window in 80-column mode
                                  that is the top half of the screen.

      WINDOW 0,13,79,24   Define a window in 80-column mode
                                  that is the bottom half of the screen.

      WINDOW 20,6,59,12,1   Define a window in 80-column mode in
                                  the center of the screen and clear it.
                                  The window is 12 characters high and
                                  40 characters wide.

      PRINT CHR$(19)CHR$(19);   Reset the window back to full screen
                                  in either 40 or 80-column mode and put
                                  the cursor in top left corner.

XOR -- Exclusive-Or function

                        XOR (number,number)

The XOR function returns a numeric value equal to the logical XOR of
two numeric expressions, operating on the binary value of the unsigned
16-bit integers in the range (0 to 65535). Numbers outside this range
result in an 'ILLEGAL QUANTITY' error.

              X = XOR(4,12)   Result is X= 8
              X = XOR(2,12)   Result is X=14

3.1.4. VARIABLES

The C64DX uses three types of variables in BASIC:

                    floating point   X
                    integer   X%
                    string   X$

Normal NUMERIC VARIABLES, also called floating point variables, can have any from up to nine digits of accuracy. When a number becomes larger than nine digits can show, as in +10 or -10, the computer displays it in scientific notation form, with the number normalized to 1 digit and eight decimal places, followed by the letter E and the power of ten by which the number is multiplied. For example, the number 12345678901 is displayed as 1.23456789E+10.

INTEGER VARIABLES can be used when the number is a signed whole number from +32767 to -32768. Integer data is a number like 5, 10, or -100. Integers take up less space than floating point variables, particularly when used in an array.

STRING VARIABLES are those used for character data, which may contain numbers, letters, and any other character that the computer can make. An example of string data is "Commodore C64DX".

VARIABLE NAMES may consist of a single letter, a letter followed by a number, or two letters. Variable names may be longer than 2 characters, but only the first two are significant. An integer is specified by using the percent (%) sign after the variable name. String variables have a dollar sign ($) after their names.

EXAMPLES:

Numeric Variable Names: A, A5 , BZ
Integer Variable Names: A%, A5%, BZ%
String Variable Names : A$, A5$, BZ$

ARRAYS are lists of variables with the same name, using an extra number (or numbers) to specify an element of the array. Arrays are defined using the DIM statement, and may be floating point, integer, or string variable arrays. The array variable name is followed by a set of parentheses () enclosing the number of the variable in the list.

EXAMPLE:

A(7), BZ%(11), A$(87)

Arrays can have more than one dimension. A two dimensional array may be viewed as having rows and columns, with the first number identifying the row and the second number identifying the column (as if specifying a certain grid on the map).

EXAMPLE:

A(7,2), BZ%(2,3,4), Z$(3,2)

RESERVED VARIABLE NAMES are names that are reserved for use by the computer, and may not be used for another purpose. These are the variables DS, DS$, ER, ERR$, EL, ST, TI, and TI$. KEYWORDS such as TO and IF or any other names that contain KEYWORDS, such as RUN, NEW, or LOAD cannot be used.

ST is a status variable for input and output (except normal screen/keyboard operations). The value of ST depends on the results of the last I/O operation. In general, if the value of ST is 0 then the operation was successful.

TI and TI$ are variables that relate to the real-time clock built into the C64DX. The system clock is reset to zero when the system is powered up or reset, and can be changed by the user or a program.

TI$="hh:mm:ss.t"   Allows optional colons to delimit parameters and allows input to be abbrieviated (eg., TI$="h:mm" or even TI$=""), defaulting to "00" for unspecified parameters. 24-hour clock (00:00:00.0 to 23:59:59.9).

TI   24-hour TOD converted into tenths of seconds.

The value of the clock is lost when the computer is turned off. It starts at zero when the computer is turned on, and is reset to zero when the value of the clock exceeds 23:59:59.9.

The variable DS reads the disk drive command channel, and returns the current status of the drive. To get this information in words, PRINT DS$. These status variables are used after a disk operation, like DLOAD or DSAVE, to find out why the error light on the disk drive is blinking.

ER, EL, and ERR$ are variables used in error trapping routines. They are usually only useful within a program. ER returns the last error encountered since the program was RUN. EL is the line where the error occurred. ERR$ is a function that allows the program to print one of the BASIC error messages. PRINT ERR$(ER) prints out the proper error message.

3.1.5. OPERATORS

The BASIC OPERATORS include ARITHMETIC, RELATIONAL, and LOGICAL OPERATORS. The ARITHMETIC operators include the following signs:

```
+ addition
- subtraction
* multiplication
/ division
^ raising to a power (exponentiation)
```

On a line containing more than one operator, there is a set order in which operations always occur. If several operators are used together, the computer assigns priorities as follows. First exponentiation, then multiplication and division, and last, addition and subtraction. If two operators have the same priority, then calculations are performed in order from left to right. If these operations are to occur in a different order, BASIC 10.0 allows giving a calculation a higher priority by placing parentheses around it. Operations enclosed in parentheses will be calculated before any other operation. Make sure that the equations have the same number of left and right parentheses, or a SYNTAX ERROR message is posted when the program is run.

There are also operators for equalities and inequalities, called RELATIONAL operators. Arithmetic operators always take priority over relational operators.

```
=           is equal to
<           is less than
```

```
        >              is greater than
        <= or =<       is less than or equal to
        >= or =>       is greater than or equal to
        <> or ><       is not equal to
```

Finally, there are three LOGICAL operators, with lower priority than both arithmetic and relational operators:

```
        AND
        OR
        NOT
```

These are most often used to join multiple formulas in IF...THEN statements. When they are used with arithmetic operators, they are evaluated last (i.e., after + and -). If the relationship stated in the expression is the true the result is assigned an integer of -1 and if false an integer of 0 is assigned. There is also an XOR function.

<div align="center">EXAMPLES:</div>

```
IF A=B AND C=D THEN 100      requires both A=B & C=D to be true
IF A=B OR C=D THEN 100       allows either A=B or C=D to be true
A=5:B=4:PRINT A=B            displays  0
A=5:B=4:PRINT A>3            displays -1
PRINT 123 AND 15:PRINT 5 OR 7   displays 11 and 7
```

3.1.6. ERROR MESSAGES

3.1.6.1. BASIC ERROR MESSAGES

The following error messages are displayed by BASIC. Error messages can also be displayed with the use of the ERR$() function. The error number refers only to the number assigned to the error for use with this function. In direct mode, DOS error messages (DS$) are automatically displayed. They are described in the section after this one.

```
ERROR#    ERROR NAME                      DESCRIPTION
-----------------------------------------------------------------
 1  TOO MANY FILES          There is a limit of 10 files OPEN at one
                            time.

 2  FILE OPEN               An attempt was made to open a file using
                            the number of an already open file.

 3  FILE NOT OPEN           The file number specified in an I/O
                            statement must be opened before use.

 4  FILE NOT FOUND          No file with that name exists on the
                            specified drive.

 5  DEVICE NOT PRESENT      The required I/O device not available.

 6  NOT INPUT FILE          An attempt made to read data from a file
                            that was opened for writing.

 7  NOT OUTPUT FILE         An attempt was made to write data to a
                            file that was opened for reading.
```

```
 8   MISSING FILE NAME        Filename was missing in command.

 9   ILLEGAL DEVICE NUMBER    An attempt was made to use a device
                              improperly (SAVE to the screen, etc) or
                              an illegal device number was specified.

10   NEXT WITHOUT FOR         Either loops are nested incorrectly, or
                              there is a variable name in a NEXT
                              statement that doesn't correspond with
                              one in FOR.

11   SYNTAX ERROR             A statement is unrecognizable by  BASIC.
                              This could be  because of missing or
                              extra parenthesis, parameters, delimiters,
                              or a misspelled keyword.


12   RETURN WITHOUT GOSUB     A RETURN statement was encountered when no
                              GOSUB statement was active.

13   OUT OF DATA              A READ statement was encountered with no
                              DATA left unREAD.

14   ILLEGAL QUANTITY         A number used as an argument is  outside
                              the allowable range (too big or too small)

15   OVERFLOW                 The result of a computation is larger than
                              the largest number allowed (1.701411834E+38)

16   OUT OF MEMORY            There is not enough memory for the program,
                              or variables, or there are too many DO, FOR
                              or GOSUB statements in effect.

17   UNDEF'D STATEMENT        A line number referenced doesn't exist.

18   BAD SUBSCRIPT            The program tried to reference an element
                              of an array out of the range specified by
                              a DIM statement, a missing DIM statement
                              or a mistyped function name.

19   REDIM'D ARRAY            An array can only be DIMensioned once.

20   DIVISION BY ZERO         Division by zero is illegal.

21   ILLEGAL DIRECT           Command is only allowed to be used in a
                              program.

22   TYPE MISMATCH            A numeric variable was used in place of
                              a string variable or vice versa.

23   STRING TOO LONG          An attempt was made to assign more than
                              255 characters to a string, or enter more
                              than 160 characters from the keyboard, or
                              to input more than 255 characters from a
                              file.

24   FILE DATA                The wrong type of data was read from a
                              file.

25   FORMULA TOO COMPLEX      An expression is too complicated for BASIC
```

```
                            to process all at one time. Break it into
                            smaller pieces or use fewer parentheses.

26  CAN'T CONTINUE          The CONT command does not work if the
                            program was not RUN, there was an error
                            or a line has been edited.

27  UNDEFINED FUNCTION      An attempt was made to use a user defined
                            function that was never defined.

28  VERIFY                  The program on disk does not match the
                            program in memory.

29  LOAD                    There was a problem loading.

30  BREAK                   The program was halted by the STOP key or
                            a STOP statement.

31  CAN'T RESUME            A RESUME statement was encountered without
                            a TRAP in effect, or an error occurred in
                            the trap handler itself.

32  LOOP NOT FOUND          The program encountered a DO statement and
                            cannot find the corresponding LOOP.

33  LOOP WITHOUT DO         A LOOP was encountered without a DO
                            statement active.

34  DIRECT MODE ONLY        A command was used in a program that can
                            only be used in direct mode.

35  NO GRAPHICS AREA        A graphics command was used before a
                            graphics screen was defined and opened.

36  BAD DISK                A BOOT SYS command failed because the disk
                            could not be read.

37  BEND NOT FOUND          A BEND statement not found for BEGIN.

38  LINE NUMBER TOO LARGE   A line number cannot exceed 64000.

39  UNRESOLVED REFERENCE    Renumber failed because a referenced line
                            number does not exist.

40  UNIMPLEMENTED COMMAND   The given command is not currently
                            implemented in this computer.

41  FILE READ               There was a problem reading data from a
                            disk file. Similar to LOAD ERROR.
```

3.1.6.2. DOS ERROR MESSAGES

    The following error messages are returned through the DS and DS$
variables. If a disk command is type in direct mode, these messages
will be displayed automatically. NOTE: DOS message numbers less than
20 are advisory and are not necessarily errors. DOS messages may vary
slightly depending upon the drive model. Refer to your DOS manual for
details.

```
ERROR #                              DESCRIPTION
```

```
-------  ----------------------------------------------------------
  00:   OK (no error)

  01:   FILES SCRATCHED (not an error)
        The following number (track) tells how many files were deleted
        by the scratch command.

  02:   PARTITION SELECTED (not an error)
        The requested disk partition (subdirectory) has been selected.

  03:   FILES LOCKED
        The requested file(s) have been locked.

  04:   FILES UNLOCKED
        The requested file(s) have been unlocked.

  05:   FILES RESTORED
        The requested file(s) have been recovered (undeleted).

  20:   READ ERROR (block header not found)
        The disk controller is unable to locate the header of the
        requested data block. Caused by an illegal sector number, or
        the header has been destroyed.

  21:   READ ERROR (no sync character)
        The disk controller is unable to detect a sync mark on the
        desired track. Caused by misalignment of the read/write head,
        no diskette is present, or unformatted or improperly seated
        diskette. Can also indicate a hardware failure.

  22:   READ ERROR (data block not present)
        The disk controller has been requested to read or verify a
        data block that was not properly written. This error occurs in
        conjunction with the BLOCK commands and indicates an illegal
        track and/or sector request.

  23:   READ ERROR (checksum error in data block)
        This error message indicates that there is an error in one or
        more of the data bytes. The data has been read into the DOS
        memory, but the checksum over the data is in error. This
        message may also indicate grounding problems.

  24:   READ ERROR (byte decoding error)
        The data or header has been read into the DOS memory, but a
        hardware error has been created due to an invalid bit pattern
        in the data byte. This message may also indicate grounding
        problems.

  25:   WRITE ERROR (write-verify error)
        This message is generated if the controller detects a mismatch
        between the written data and the data in the DOS memory.

  26:   WRITE PROTECT ON
        This message is generated when the controller has been
        requested to write a data block while the write protect switch
        is depressed.

  27:   READ ERROR
        This message is generated when a checksum error is in the
        header.
```

28:    WRITE ERROR
       This error message is generated when a data block is too long.

29:    DISK ID MISMATCH
       This message is generated when the controller has been
       requested to access a diskette which has not been initialized.
       The message can also occur if a diskette has a bad header.

30:    SYNTAX ERROR (general syntax)
       The DOS cannot interpret the command sent to the command
       channel. Typically, this is caused by an illegal number of
       file names, or patterns are illegally used. For example, two
       file names appear on the left side of the COPY command.

31:    SYNTAX ERROR (invalid command)
       The DOS does not recognize the command. The command must start
       in the first position.

32:    SYNTAX ERROR (invalid command)
       The command sent is longer than 58 characters.

33:    SYNTAX ERRROR (invalid file name)
       Pattern matching is invalidly used in the OPEN or SAVE
       command.

34:    SYNTAX ERROR (no file given)
       The file name was left out of the command or the DOS does not
       recognize it as such.

39:    SYNTAX ERROR (invalid command)
       This error may result if the command sent to the command
       channel (secondary address 15) is unrecognized by the DOS.

40:    UNIMPLEMENTED COMMAND
       Command is not implemented at this time.

41:    FILE READ
       The file cannot be read.

50:    RECORD NOT PRESENT
       Result of disk reading past the last record through INPUT# or
       GET# commands. This message will also occur after positioning
       to a record beyond end_of_file in a relative file. If the
       intent is to expand the file by adding the new record (with a
       PRINT# command), the error message may be ignored. INPUT and
       GET should not be attempted after this error is detected
       without first repositioning.

51:    OVERFLOW IN RECORD
       PRINT# statement exceeds record boundary. Information is
       truncated. Since the carriage return which is sent as a record
       terminator is counted in the record size, this message will
       occur if the total characters in the record (including the
       final carriage return) exceeds the defined size.

52:    FILE TOO LARGE
       Record position within a relative file indicates that disk
       overflow will result.

53:    BIG RELATIVE FILES DISABLED