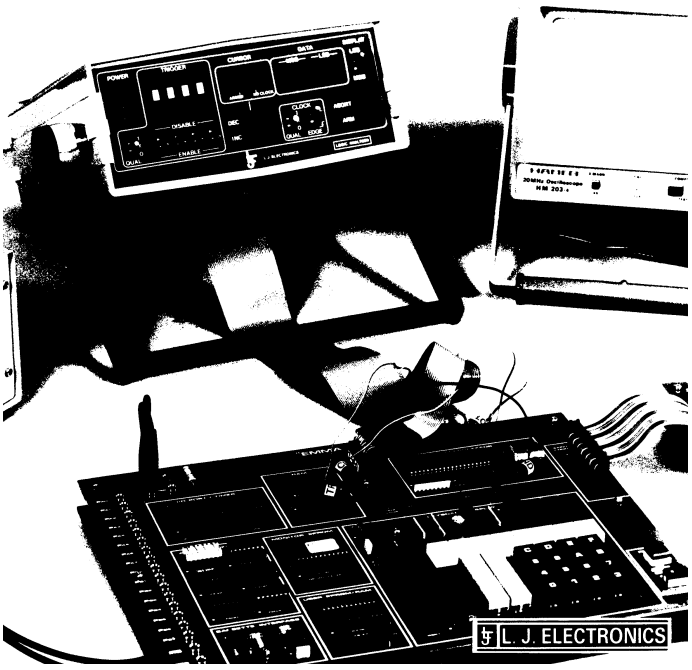
L.J. Technical Systems

EMMA II
User Manual



L.J. Electronics

EMMA II

User Manual

Written by LJ Technical Publications Dept.

Designed, Typeset and Produced by LJ Publicity Dept. 1986, second issue.

Issue Number MP116/E

Contents

EMMA II User Manual

Introduction to EMMA II

The EMMA II User Manual provides an introduction to the LJ EMMA, an educational microcomputer system, based on a 6502 microprocessor.

Designed to introduce the EMMA system in a straight forward step by step manner, this manual will make the user familiar with the instructions used to program the 6502, together with providing an awareness of system architechture. It also illustrates the many possibilities for further applications using this expandable system.

The EMMA II Technical Manual provides the user with the more detailed information on the system architecture and the software operation.

As the user becomes familiar with the EMMA it will become evident that applications control is a major function of the system.
LJ Electronics provide a wide range of equipment designed for more advanced studies.These include

- A range of Applications Hardware Modules: simple advanced, digital control.
- VISA Expansion unit - providing Video Interface, ASSEMBLY language and BASIC language programming, EPROM programming, Floppy Disk interface.
- 6502 Trouble Shooting System
- 6502 Development System
- Robotic Teaching Systems with trouble shooting

General Description

EMMA II is a fully assembled and tested microcomputer requiring only a +5V, approximately 700mA, regulated d.c. supply to commence computing.

The system is built around a 6502 microprocessor, and has a crystal controlled clock operating at 1 MHz.

A monitor program and useful sub-routines are stored in a 2716 EPROM.

User memory is available using a 2k byte RAM.

An important feature is the Input/Output port (I/O Port) capability which is provided by the 6522 Versatile Interface Adaptor (VIA) and includes, amongst other features, two 8-bit programmable I/O ports.

Keyboard and display interface is provided by the 6821 Interface controller.

The 6502 microprocessor is capable of addressing 64K memory. On the EMMA II only a limited amount is decoded. The required address decode is selected by links placed in an i.c. Header. This gives the user access to modify the address decoding if required. The arrangement is shown below for the unexpanded EMMA II.

Wire Link Decode Select

(As viewed from bottom of board)

The hardware of EMMA II is arranged to allow the easy identification of sections as shown opposite on page 3.

The cassette interface provides a facility for the rapid retention of programs on a standard cassette recorder. Connection to the recorder is made via 0.1" pins on EMMA II and the cassette DIN input/output socket or the external microphone/earphone connectors.

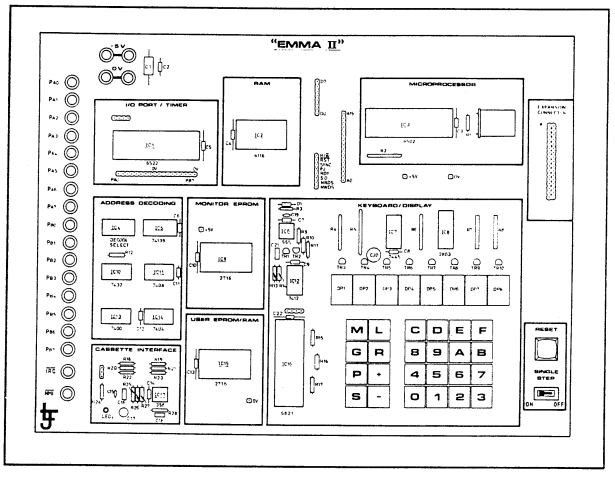
Communication with EMMA II is through the keyboard/display although you will soon make use of the I/O port which is brought out to 4mm sockets on the left-hand side of the microcomputer board. These sockets are designated PA0-PB7 and provide the input/output connections to I/O Ports A and B respectively. Also available are 4mm sockets for interrupt facilities - these are used extensively in work associated with Application Modules.

Power supply to the board is made via 4mm sockets, two sockets being provided for the +5V connection and two for the 0V connection. The provision of two sockets for each line facilitates the looping of supplies to other system items.

Access to all bus lines and the various control signals are made through 0.1" printed circuit board (p.c.b.) pins. These are used for System Diagnosis and Fault Finding exercises.

The single step facility on the bottom right hand side of the board is a **Debug Facility**, and will be described later in the manual. For normal use this switch must be in the 'OFF' position.

Before switching ON familiarize yourself with the control key functions.



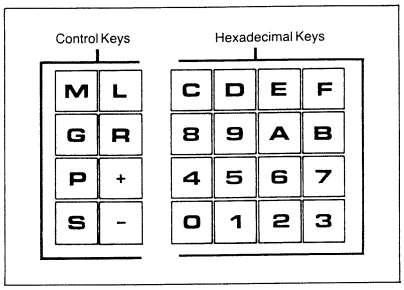
Chapter 1 EMMA II Key Functions

The EMMA II keyboard/display

The EMMA II keyboard/display unit provides all the necessary controls for EMMA II except for the Address Decode Patching Header and the Reset Pushbutton.

The keyboard is split into two well defined key groupings:

- Hexadecimal Keys - These are the matrix of sixteen keys marked 0 - 9 and A - F. They are used to input all data, example: user programs, data tables.
- Control Keys - These are the matrix of eight keys marked, M,G,P,S,L,R,+ and −. These keys have well defined functions as below.



[M] M key - used to select the memory mode. For example the seven segment display may show either a memory address or a memory address and its content. Depression of M will alternate these modes.

[+] [−] +(plus) and − (minus) - used to increment or decrement the program entry address. For example if the current address being displayed is 0020, pressing the + (plus) key will increment this to 0021. The use of these keys greatly facilitates program entry and subsequent checking before running a program.

[G] Program Run - The G key causes the program to **Run** (be executed). In practice having entered a program, G will be pressed and the start address of the program then keyed in. Pressing G again will cause the program to be executed.

[R]
[P] Program Debug - Key R provides a single step feature and Key P a means of inserting a forced break into a user program. Both these keys are essentially for program debug and are fully discussed later in this manual.

[S]
[L] Program Dump - **EMMA II** provides a feature for dumping programs onto magnetic tape using a conventional cassette tape recorder. Keys S and L are used to control this feature, S being for store onto tape, while L loads the microcomputer memory from tape. Both are discussed later.

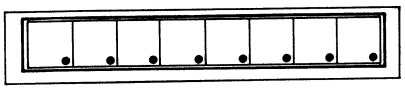
These key functions are designated by the microcomputer monitor program. They may, however, be redesignated temporarily under user program control. They will assume normal function upon returning to the monitor program.

We will now follow a switch-on routine and familiarize ourselves with the actual **EMMA II** keyboard/display.

Chapter 2 EMMA II Switch-on

Connect EMMA II to the 5V, 3A supply outlet of an LJ Electronics System Power 90 carefully ensuring that the polarity is correct.

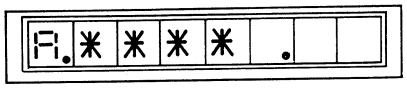
Press the **Reset** button (bottom right-hand of the microcomputer board) and notice that the display shows eight decimal points.



This indicates that the monitor program is running and the microcomputer is ready to accept information from the keyboard.

It is also reasonable to assume that the microcomputer system is operating correctly.

Press the control key 'M' (memory key). The display will now indicate:

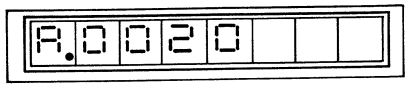


Where **** is a 4 digit hexadecimal address between the values 0000 and FFFF.

This display can be modified by depressing the desired sequence of hexadecimal coding keys.

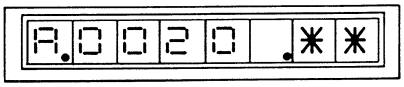
e.g. press 0 0 2 0

The display now shows:



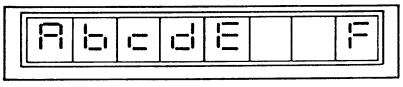
This is the address of the lowest user memory in EMMA II.

Now press the 'M' key again. The display will now show the data stored at the location indicated by illuminating the two right-most 7-segment displays. Example, if the address part of the display shows A.0020 then pressing 'M' will cause the display to show:



where ** are any two hexadecimal digits. Pressing any of the hexadecimal coding keys will modify this data.

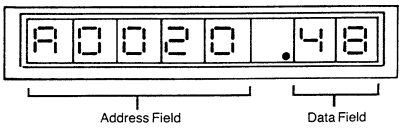
Now press keys A, B, C, D, E and F and observe that some characters on the display are in capitals and some in lower case.



Note: the letter b and the number 6 are similar and care must be taken not to misinterpret these two characters.

We refer to the two parts of the display (as used above) as the **Address Field** and **Data Field** respectively.

Example:



Now perform the following operation:

	Operation	Display	COMMENTS
●	Press Reset	. . . . .	Monitor running
●	Press Control key M	A.**** .	Address field only illuminated and showing a random address.
●	Press Hex keys 0020	A.0020.	Address field indicates address high byte (00) address low byte (20)
●	Press Hex keys 0238	A.0238 .	Address modified to 0238
●	Press M	A.0238 .**	Address field and data field illuminated Data field shows random data stored at location 0238.
●	Press Hex keys 45	A0238 .45	Data field shows data (45) input to location 0238.

The function of the control keys plus (+) and minus (−) can now be explored. These increment (increase by one) and decrement (decrease by one) the displayed address field when in the data mode. They provide a convenient way of sequentially moving through a series of addresses when entering a program or simply checking a program already entered without having to continually use the M control key.

Before we can actually write a program we need to have some knowledge of the machine **Instruction Set**. There is an instruction set for the 6502 in Appendix A, since the EMMA II is based upon this microprocessor.

However, for convenience, we will reproduce part of an instruction that we will be using to form our first simple program.

LDA **Load Accumulator with Memory** **LDA**

Operation: (M)→A

N Z C I D V
✓ ✓ - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Absolute	LDA Oper	AD	3	4

Instructions are as shown, from the instruction set.

We should notice the following:

- The operation is clearly stated as:
  "Load accumulator with memory"

  It is also symbolized by: (M) → A which means: "Transfer the contents of memory M to the accumulator A"

- A Mnemonic is given:

  LDA

  meaning: "Load Accumulator"

- An addressing mode is given:

  Absolute

  This indicates that the data is to be found at the absolute address specified by the two bytes following the Op. Code.

- The format of the Assembly Language instruction is given:

  LDA Oper

  where LDA specifies the Operation to be performed and is to be followed by an operand. In our case the operand is a two byte absolute address.

Now let's see how we can employ this instruction and others in a simple program.

A Simple Program Example

- Program Task
  Transfer the contents of memory M1 to memory M2.

  All data movements must be made through the accumulator; hence the TASK is executed by

- LOADING the data in memory M1 into the accumulator A.
- STORING the data transferred from M1 into memory M2.

  Our program, using mnemonics, is

  LDA M1
  STA M2

  If we now assign addresses for M1 and M2 we get

  LDA 0080
  STA 0081

  where 0080 and 0081 are two absolute addresses with the high byte (00) specified before the low byte (80).

  Note: It is normal to refer to addresses high byte first followed by low byte e.g. 0080 and 0081. However, when entering a program in memory, the machine requires that we reverse this order.

Using the STA instruction to store accumulator in memory location 0081 and assigning addresses we get:

0020	LDA	0080	
0023	STA	0081	
0026			Next instruction op code

Our program (using a standard programming sheet) would look like this:

Standard Programming Form

Programmer: Program Title: 1st Program

Hexadecimal				Symbolic Assembler Instructions			
ADDR	1	2	3	LABEL	MNEM	OPERAND	COMMENTS
				Program			
0200	AD	80	00		LDA	M1	Address of M1 is 0080
0203	8D	81	00		STA	M2	Address of M2 is 0081
0206	4C	06	02		JMP	0206	Terminates program
				Data			
0080	?						Single byte of unspecified data to be transferred to memory
0081							0081

You will notice that we have terminated our program using a **Jump** instruction.

Memory locations 0080 can be loaded with data to be transferred and 0081 with 00 to be written over.

Data Address	Bytes 1 2 3	Comments
0080	FF	Data (FF) to be transferred
0081	00	Data (00) will be over written

You may notice that we are going to load memory location 0080 and 0081 with FF and 00 respectively. We have deliberately put 00 in 0081 so that we will positively know that FF has been transferred.

We will now try entering an actual program.

Program Entry

RESET **EMMA II**	-	Press Reset
Obtain Address Field	-	Press M Key
Set 1st Address (0200)	-	Press 0, 2, 0 and 0
Obtain Data Field	-	Press M Key
Modify Data Field	-	Set AD
Increment Program	-	Press + Key
Modify Data Field	-	Set to 80
Increment Program	-	Press + Key

Continue until program is entered.

Now **Load** Data.

Obtain Address Field	-	Press M
Set Address (0080)	-	Press 0, 0, 8 and 0
Obtain Data Field	-	Press M
Enter Data	-	Press FF

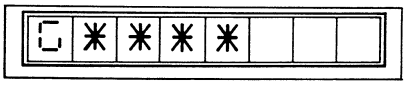
Continue for 0081

You should now have entered the whole of the program and the data. It is advisable to check this by looking at each loaded location. You may do this by incrementing or decrementing through the locations and the respective data.

Now let's run the program!

Program Run

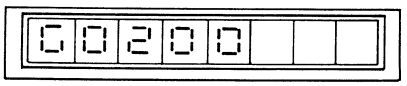
Press control key G (G stands for 'GO') - the display should now show:



where **** is some random address.

Using the hexadecimal coding keys, modify this random address to the start address of the program e.g. 0200

Display:



Press the go key, G, again.

You will notice that the display has gone blank. To check whether the program has indeed run successfully we now need to inspect the memory location into which we transferred the contents of location 0080. Remember - the contents of 0080 was FF and should have been transferred to 0081.

Symbolically:
(0080) → 0081
where ( ) indicates contents of
and → transfer to.

To inspect the memory contents, press **Reset** followed by M, key in address 0081, press M again and data field of display should indicate FF. Transferring contents of a memory does not destroy it.

Change contents of 0080 and run again and see if your data is transferred.

Chapter 3 User Memory

Within the microcomputer system some memory locations may not be available to the user, either because they are used for special purposes or simply because memory devices do not exist at these addresses. As mentioned earlier the 6502 can address 64K of memory.

Our major concern at this point is to see what memory locations are available to us to store our program. You should observe that available memory space is:

0020 – 00FF = 224 bytes for the user

0200 – 03FF = 512 bytes for the user

0C00 – 0FFF = 1024 bytes for the user

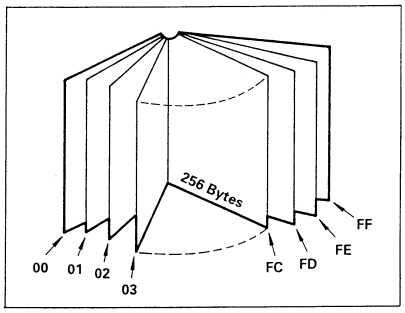
001C – 001F = 4 bytes used by the user in conjunction with useful routines etc.

Now is an opportune moment for us to look at the two bytes which constitute an address.

The Concept of Paging

In common with most 8-bit microprocessors, **EMMA II** has an address capability of 64K (65,536) memory locations. These are organised into **Pages** where a page is 256 consecutive memory locations. With this size of page, there are a possible 256 pages in 64K of memory.

Schematically we could show these as:



Each location could have a two-byte unique address, example:

HIGH BYTE	LOW BYTE

where the high byte is the **Page Reference Number** and the low byte is the **Location on Page**

Hence the address 0020(HEX) is:

00 page and 20(HEX) location on that page.

Page 00 and page 01 have instructions which are special to those pages.

Zero Page

Zero page may be seen as comprising a set of working registers upon which any instruction will be executed in a shorter time than if any other page had been used. Since the time saving in executing a single instruction can be as much as 33.33% it is worthwhile reserving zero page for essential data that needs to be retrieved at high speed.

The Stack

The stack is designated by the microprocessor as **page one**. Special instructions exist which operate only on the stack and serve to transfer and retrieve data 'pushed' onto and 'pulled' from the stack. None of these instructions specify a particular location on the stack - the location of the last data item pushed to the stack is 'remembered' in a register termed the **Stack Pointer**.

Chapter 4 Instruction Addressing Modes

Each instruction also has an Addressing Mode. The 6502 can perform 56 different operations, some of which can be executed in as many as eight different ways so producing 150 variations.

These addressing modes can be summarised as:

- **Implied**
  Implied addressing uses a single byte instruction which operates on registers whose address is implied by the particular OP. Code used. These registers are those internal to the microprocessor - index registers, status register, stack pointer and external to the microprocessor (in memory) - the stack and interrupt vector locations.

- **Immediate**
  All instructions in immediate addressing mode are two bytes long. The first is the OP. Code and the second specifies a constant or literal which is to be loaded into an internal register or external memory location.

- **Absolute**
  You have already met this type of addressing mode. Instructions require three bytes of which the second two specify the location of the operand.

- **Zero Page**
  Requires two bytes. Zero page is implied in Op. Code and therefore not specified implicitly. Only the location on zero page is required.

- **Relative**
  Requires two bytes. Instructions using this addressing mode are of the Branch type. They cause the microprocessor to branch to another part of the user program rather than execute the next instruction in sequence. The branch is taken upon the result of a test performed on the condition of flags within the status register. The second byte specifies the extent of the 'branch' that is the amount of program displacement and its direction relative to the address of the Op. Code of the instruction following the branch instruction.

- **Indexed**
  The 6502 is equipped with two index registers, X and Y. The contents of an index register is added to a base address specified in the address field to modify that address.

The method of addressing enables data tables to be sequentially accessed by performing increment (or decrement) operations on the index registers.

- **Indirect**
  The concept of indirect addressing enables the address field following an Op. Code to specify an address which in turn specifies the address of the data required.

- **Indexed X, indirect**
  This mode adds the contents of the index register X to the zero page address specified immediately following the Op. Code.

- **Indirect, indexed Y**
  This mode adds the contents of the index register Y to the data base address.

Chapter 5 Writing a Program

We will now construct a program using more commonly used instructions and addressing modes, maintaining the principle of:

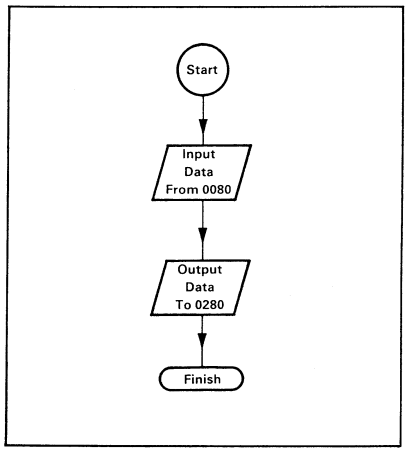
- Stating the task
- Constructing a flow chart
- And writing the program.

Task

Move a single byte of data from memory location 0080 to memory location 0280. Logically:

(0080) → 0280
Note: the brackets mean "contents of".

Flow chart



Program

This demands three basic operations to effect the transfer of data.

1. LOAD data from memory into the accumulator.
   (0080) → A
2. STORE data in accumulator in memory
   (A) → 0280
3. TERMINATE program.

We will now look at each of these steps in turn:

1. The data is on ZERO PAGE.
   00 80
   Page Location

   Now ask yourself if a LOAD instruction is available which operates directly on Zero Page (Consult Instruction Set - Appendix A)

   The Instruction Set should reveal the following:

Addressing Mode	- Zero Page
Mnemonic	- LDA
Op Code	- A5
Number of bytes	- 2

   We can write the instruction in **either:**

- Symbolic Machine Code:

Operation (Mnemonic)	Operand (Address)
LDA	80

   Where 80 is the zero page memory location.

- Hexadecimal Machine Code:

Op Code (Hexadecimal)	Operand (Address)
A5	80

2. Now let's look for a STORE instruction.

   The Instruction Set will reveal:

   Addressing Mode - Absolute
   Mnemonic - STA
   Op. Code - 8D
   Number of Bytes - 3

   We cannot use zero page addressing mode since the data is stored on page 02.

   Again we can write the instruction in two ways.

- Symbolic Machine Code

Operation (Mnemonic)	Operand (Address High Byte)	(Address Low Byte)
STA	02	80

- Hexadecimal Machine Code -

Operation (Mnemonic)	Operand (Address Low Byte)	(Address High Byte)
8D	80	02

Note the way the operand has been written. When writing addresses it is normal to write HIGH BYTE followed by LOW BYTE. However, we **Enter** the **Hexadecimal Codes** into the machine (6502) LOW BYTE first. If you follow this practice when using the **Standard Programming Form** you are less likely to make mistakes when entering your program using the hexadecimal keyboard.

3. Now let's look at terminating the Program.

   If you scan the Instruction Set you will not find an instruction which you can use directly for this purpose. The 6502 Instruction Set does not have an instruction such as **Stop, Halt** or indeed **Finish.**

First let's consider what would happen if we did not bother, after all our **two** instructions will effectively complete the task! Unless we tell the machine to stop processing when the transfer is complete it will continue to fetch, sequentially, data from memory. Unfortunately, this may be either another program or, as is more likely, rubbish (the memories will always have something in them; they cannot be 'empty').

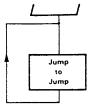
A simple way round the problem is to cause the machine to enter a 'program loop' for which it can escape only by the user pressing the RESET pushbutton. A JUMP instruction will do this.

The instruction set shows:

Addressing Mode	- Absolute
Mnemonic	- JMP
Op Code	- 4C
Number of Bytes	- 3

We will use this instruction to jump back to itself.

Example:



- Using symbolic machine code

Operation (Mnemonic)	Operand (Address)
JMP	Terminate

In the operand field we have used a **Label**. We can do this using symbolic notation. The label stands in place of a Program Address. We cannot enter the program address that we wish to jump back to because we have not yet allocated memory space for our program.

It is worthwhile mentioning here that the programmer invents his own labels for programming convenience. However, a simple guide to labelling is **DO NOT use labels that look like or contain Op Code Mnemonics**. We can now collate our instructions and enter them on a programming sheet.

Hexadecimal				Symbolic Assembler Instructions			
ADDR	1	2	3	LABEL	MNEM	OPERAND	COMMENTS
0200	A5	80			LDA	80	
0202	8D	80	02		STA	0280	
0205	4C	05	02	Term	JMP	Term	

You will notice that we have assigned memory addresses to our program. This has also enabled us to assign a program address to the label TERM.

Enter the program in EMMA II and execute. Do not forget to enter appropriate data in memory locations 0080 and 0280. If you have any doubts regarding program entry procedure, go back and study the section on program loading on page 12.

After running your program examine the location 0280 to see if your data from 0080 has been stored in that location.

If your program has not functioned as expected go back and check that the program is correctly entered.

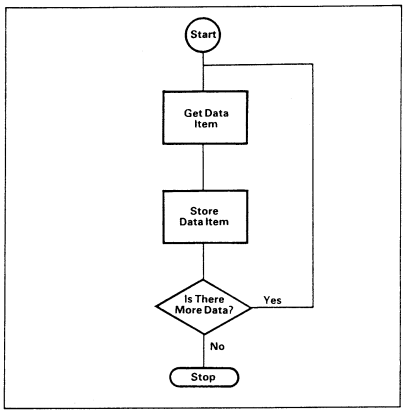
Example Program

Assuming you have successfully completed the program lets try a more complex program.

Task

The following is an example of performing a data block move.

Flow diagram



Example: Move data from data block addresses 0020-0024 to addresses 02F0-02F4.

Main Program

Hexadecimal				Symbolic Assembler Instructions			
ADDR	1	2	3	LABEL	MNEM	OPERAND	COMMENTS
0200	A2	05			LDX#	05	Set up data pointer
0202	CA				DEX		Set up data item
0203	B5	20		*	LDA	20,X	Transfer data
0205	9D	F0	02		STA	02F0,X	
0208	D0	F8			BNE	*	Get next valid data item
020A	4C	0A	02		JMP		FINISH

Data to be moved:

0020	00	
0021	01	
0022	02	
0023	03	Data to be transferred
0024	04	

Locations for data to be shifted to:

02F0	
02F1	
02F2	
02F3	Memory space reserved for data
02F4	

By studying the program you will see that it contains the following address modes:

- "LDX#05" This is an immediate instruction to load 'X' register with 05. This has the effect of setting up the data pointer in this program.
- "DEX" An implied instruction to decrement the contents of the 'X' register.

- "LDA 20,X" A ZERO PAGE instruction; to load accumulator with the contents of address 0020+'X', where 'X' is 4. Hence the instruction loads accumulator with data from 0024.
- "STA 02F0,X" An absolute instruction to store accumulator at address 02F0+X, which is 02F4. Hence data from 0024 is stored at 02F4.
- "BNE" After the decrement instruction the BNE instruction is asking to branch if X is not equal to zero. At this point in the program X is now 4 and not equal to zero and the branch is taken back to address 0202. The data F8 in the Hexadecimal region of the program is a negative number giving the backward displacement of the branch.
- "JMP" This is the next instruction after X reaches zero. The JMP instruction is the same as in the previous program and jumps back on itself and terminates the program.

Load the program; Load information to be transferred, and run the program from 0200.

Inspect reserved space after the program has been run to see if the data has been transferred.

If the program does not operate correctly, check to see if your program has been entered correctly.

Terminating a Program with a Reset

A better way of terminating the above program is to jump to the monitor of the machine to effect a reset condition. The location of the reset condition in the monitor program is FEE0.

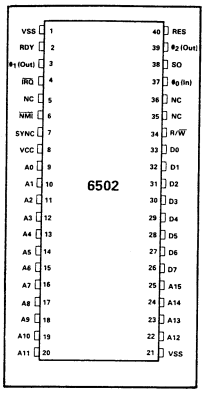
After the program has run a row of dots will appear on the display showing that the program has run and reset.

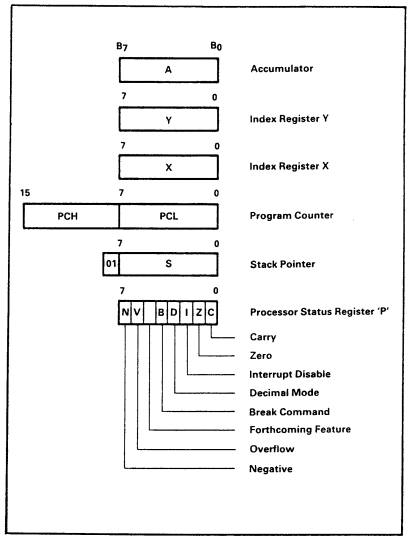
To terminate the program change the JMP instruction from 4C 0A 02 in the hexadecimal region to 4C E0 FE.

A Programming Model

To be competent at programming with EMMA II the user must be aware of the internal architecture of the 6502 microprocessor. To help with this, a programming model can be used.

This model is a diagram of all the internal registers of the microprocessor.





Status Register 'P'

The Status register comprises a number of flags some of which are set or reset by the result of operations involving the arithmetic unit. The testing of these flags is an important part of any programming task. Below is a brief description of each flag.

The programmer has control over some of these flags, he can set or reset them as required by the logic of his program.

The Status flags are:

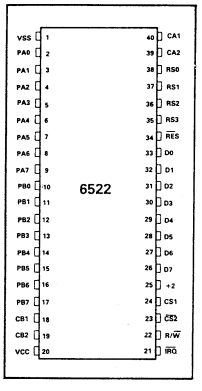
- [N] - set if the most recent operation performed in the arithmetic unit gave a negative result.
- [V] - this flag indicates when the 7-bit result of a signed number arithmetic operation overflows.
- [B] - this is a break command flag. It is set by the microprocessor when an interrupt is caused by a break command.
- [D] - when this flag is set any arithmetic operations will be performed in binary coded decimal. With this flag cleared the arithmetic unit operates in true binary.
- [I] - is the interrupt disable flag. When this flag is set the $\overline{IRQ}$ input will not interrupt the microprocessor.
- [Z] - set if the operation in the arithmetic unit gave a zero result.
- [C] - is a carry input to the arithmetic unit. If set, it will apply a '1' to the least significant bit of an arithmetic operation.

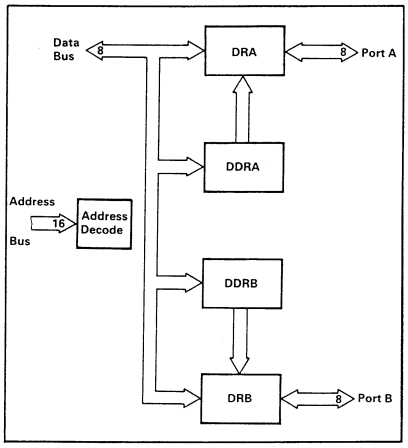
Chapter 6 Use of Input/Output Ports

The next stage in using the EMMA II is to utilise its facilities such as the I/O ports.

The I/O ports are contained in chip 6522 on the EMMA II board which is referred to as a Versatile Interface Adaptor (VIA). As its name implies, it is versatile in as far as it is capable of numerous functions. It is also an interface because it provides a means of connecting EMMA II to the outside world. The 6522 is physically connected between the 6502 (microprocessor chip) and external devices such as applications hardware modules.

The component parts of the I/O port which we will initially consider are two ports designated Port A and Port B. Schematically the appropriate architecture is:





The VIA is far more comprehensive than depicted opposite but we will consider this subsequently.

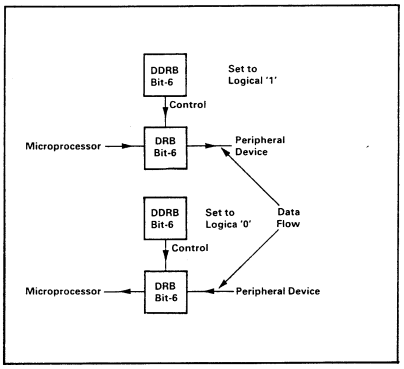
The data registers hold the data which is being transferred from the microprocessor to some outside peripheral device or from some outside peripheral device to the microprocessor, that is to say they are bidirectional. Needless to say data cannot pass through them in both directions simultaneously. They have to be set to operate in the required direction. **An important feature of this particular VIA is that each of the eight bits of both ports can be directionally set independantly.** It is the function of the data direction registers to accomplish this.

The setting up of the ports is termed **initialisation** and must be done by the programmer before the port is used.

Port Initialisation

The data direction registers are both eight bit registers with each bit being associated with a corresponding bit in the data register. For example, if bit 6 of data direction register Port B is set to logical one, then bit 6 of data register Port B will be set to 'output' while logical zero will set it to 'input'.

The diagram below indicates:

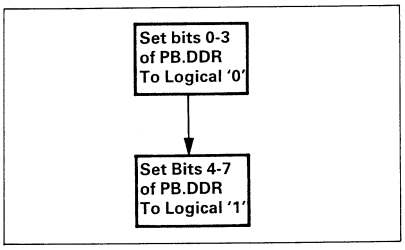


The blocks in the diagram are fully addressable and are identifiable as below:

Label	Designation	Address
DRB	Port B, Data Register	0900
DRA	Port A, Data Register	0901
DDRB	Port B, Data Direction Register	0902
DDRA	Port A, Data Direction Register	0903

Each of these registers can be separately addressed and simply appear to the microprocessor as a memory location.

We will now write a program which will initialise Port B so that bits B0-B3 are configured as input and bits B4-B7 as output.



We can perform our setting of the data direction register simply by loading the DDRB with F0 and using the instructions:

LDA# F0
STA DDRB

where:

F sets bits B4 - B7 to 1's
0 clears bits B0 - B3 to 0's

Simply DDRB is the label for the address of Port B - Data Direction Register. In hexadecimal notation this is 0900.

Using the Data Registers

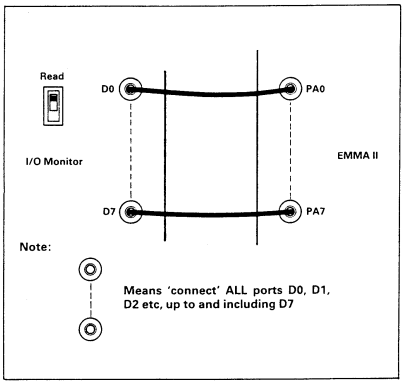
Once the ports have been initialised the data registers can be used. We will consider the Data Registers as being 'transparent', that is, any data appearing at the register in the correct direction (as determined by the DDR) will pass through it. I/O Ports are covered in more detail in the **EMMA II Technical Manual.**

The control of any system configuration is the responsibility of the microprocessor and its program. The data registers therefore appear to the microprocessor as memory locations, data can be 'stored' to them or 'loaded' from them.

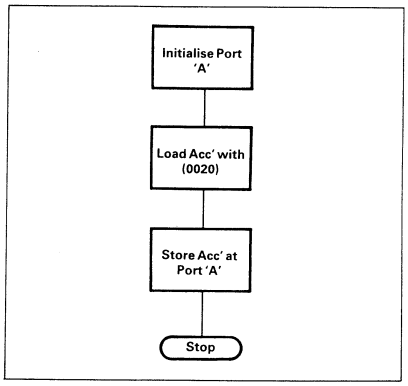
Using the I/O Port with I/O Monitor

- Consider the program to transfer data to the I/O port
- Take a standard programming form and assemble a program which transfers data from memory 0020 to Port A.
- Connect the I/O port monitor to Port A, and set the I/O monitor to READ.

Connection Diagram:



Flow chart for this procedure is:



The program listing is:

	Hexadecimal			Symbolic Assembler Instructions			
ADDR	1	2	3	LABEL	MNEM	OPERAND	COMMENTS
0200	A9	FF			LDA#	FF	Load accumulator with FF
0202	8D	03	09		STA	0903	Stores acc at DDRA
0205	A5	20			LDA	20	Load acc with contents of 0020
0207	8D	01	09		STA	0901	Stores acc at DRA
020A	4C	0A	02		JMP	020A	Terminates program

Load your program into **EMMA II**. Also load memory 0020 with AA.

RUN program and observe results.

◆ ***Exercise***

Write a program which READS data from a peripheral device and stores in memory 0020. Keep connection of I/O port monitor to Port A and switch its mode selector switch to WRITE.

Chapter 7 Interrupts

Interrupts

An interrupt allows the program currently being run on the microprocessor to be interrupted (it's execution temporarily suspended), so that a more important task can be attended to.

There are four possible conditions which may interrupt the microprocessor; these are considered in turn.

Interrupt Request (IRQ)

IRQ Vectors ØEFE Low byte
ØEFE High byte

This is a level sensitive input to the 6502. When a logical '0' is sensed at the $\overline{IRQ}$ pin (the bar means enabled low), the processor will complete its current operation. It will then 'read' the Interrupt Disable Flag (flag I in status register) and, if clear, will implement an interrupt sequence of operations.

- Store Program Counter high byte (PCH) on the stack
- Store Program Counter low byte (PCL) on the stack
- Store status register contents on the stack
- Load PCL from address ØEFE
- Load PCH from address ØEFF
- Set Interrupt Disable Flag (flag I in status register).
- Program execution now continues from the memory VECTOR held at ØEFE and ØEFF

If the Interrupt Disable Flag is set when the $\overline{IRQ}$ line goes low, the interrupt will be ignored.

ØEFE Low byte	IRQ Vectors
ØEFF High byte	

If only one device were connected to the $\overline{IRQ}$ line it would be serviced by what is known at an 'interrupt service routine'. This routine would effectively be the final sequence of instructions above, namely program execution now continues from the memory VECTOR held at ØEFF and ØEFE. However, it is more likely that a number of devices would be connected to the same $\overline{IRQ}$ line, each capable of bringing it low. A software routine would then have to determine which device had caused the interrupt before it could execute an appropriate set of instructions to service that particular device. Various methods are available not least of which is termed **Polling**.

Polling

This is a technique whereby all devices connected to the $\overline{IRQ}$ line are 'polled' or interrogated to determine whether they are asking for service. Since two or more devices may be requesting an interrupt, the polling may be performed to some order of priority.

Once an interrupt service routine is being executed it is possible for the programmer to allow a further interrupt to take place since the interrupt disable flag is under his control. In this way a number of interrupts may be in a state of being serviced at any one time.

At the end of an interrupt service routine the instruction, Return From Interrupt (RTI) must be used.

A further point to note is that if any other registers such as the accumulator, X or Y hold data, at the instant of interrupt, which needs to be remembered, these must be pushed to the stack at the start of the routine and pulled from the stack prior to RTI.

One last point-it is important that before executing the RTI, the Device Interrupt Flag which pulled the $\overline{IRQ}$ low is set since RTI will have the effect of clearing the Interrupt Disable Flag when the status register is restored; eg., the flag must have been clear to allow the interrupt in the first place. If this action is not taken then a series of interrupts will be attempted although the device has, in fact, been serviced!

Non Maskable Interrupt ($\overline{NMI}$)

ØEFC Low byte	NMI Vectors
ØEFD High byte	

This is an edge sensitive input to the 6502. When a logical '1' to logical '0' transition takes place at the $\overline{NMI}$ pin, the microprocessor will complete its current operation and set an internal flag such that no matter what state the interrupt disable flag is in, the microprocessor performs the interrupt sequence outlined under $\overline{IRQ}$. The only exception is that the memory vectors are taken from ØEFC and ØEFD.

The $\overline{NMI}$, through the way it has been implemented, has priority over the $\overline{IRQ}$ at all times.

It is possible to connect more than one device to the $\overline{NMI}$, but if a subsequent interrupt occurs while servicing the first, it will be ignored. Further, it will not be serviced when the initial service routine is completed. The implications are that multiple interrupt lines connected to the $\overline{NMI}$ require careful servicing.

Break Command (BRK)

The break command is a software interrupt. It is primarily used to cause the microprocessor to go to a halt condition during program debugging but can equally be used in other useful ways.

The break command sequence is similar to the hardware interrupt $\overline{IRQ}$ except that it cannot be masked by the interrupt disable flag. Also when the break command is 'fetched' the Break Command Flag (flag B in status register) is set, this enables the programmer to check whether the interrupt was caused by the software BRK or the hardware $\overline{IRQ}$. Both use the same memory vectors 0EFE and 0EFF.

RESET ($\overline{RES}$)

This is an edge sensitive input to the 6502 when a logical '0' to logical '1' transition takes place at the $\overline{RES}$ pin, the microprocessor will begin a 'reset' sequence.

It is used to reset or start the microprocessor from a power-down condition. With $\overline{RES}$ held low, read/write operations are inhibited. In the case of EMMA II, $\overline{RES}$ is held high via a 4.7kΩ resistor; it is pulled low when the reset pushbutton is depressed. Depression of the reset pushbutton and then its release will start the reset sequence. After a system initialisation period of six clock cycles, the interrupt disable flag will be set and the microprocessor will load the program counter with the reset vectors FEE0.

Programming Interrupt Service Routines

We will now design a program which will demonstrate the use of interrupt service routines.

Task

Our program will repeatedly increment the VIA User Port A and the number of times a full count is achieved will be indicated at VIA User Port B. The program will include a time waste sub-routine so that Port A can be easily observed on the I/O Port Monitor.

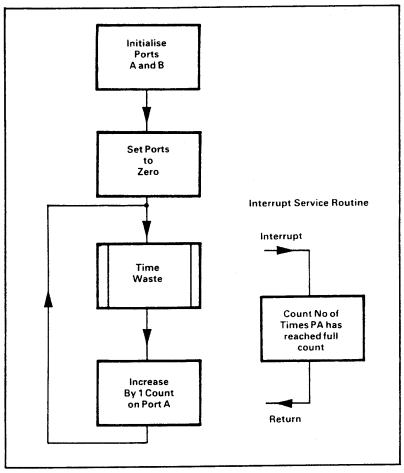
The program is in four parts:

- Main Program
- Interrupt Service Routine
- Time Waste Sub-Routine
- Interrupt Vector Loading

We will look at each part separately.

The first function is to draw up a flow chart of the task.

Flow chart



Main program

Hexadecimal				Symbolic Assembler Instructions			
ADDR	1	2	3	LABEL	MNEM	OPERAND	COMMENTS
0200	A9	FF			LDA#	FF	Initialise ports
0202	8D	02	09		STA	DDRB	A and B to
0205	8D	03	09		STA	DDRA	output
0208	A9	00			LDA#	00	Set DRA
020A	8D	00	09		STA	DRB	and DRB to zero
020D	8D	01	09		STA	DRA	
0210	EE	01	09	NEXT	INC	DRA	
0213	20	80	02		JSR	0280	Time waste subroutine
0216	4C	10	02		JMP	NEXT	Continue count

Enter the main program starting at 0200. Examination of the main program is shown below:

Initialisation of Ports 'A' and 'B' is done by setting the data direction registers (DDRA) to FF, giving all 1's; hence output conditions exists. Setting of the ports to zero is done by loading (DRA) direction registers to '00'. Then incrementing of port 'A' begins at 0210. A time waste routine is needed to give the LED displays time to illuminate and time for you to see the count. As can be seen the instruction at 0213 is to jump to subroutine at 0280. The time waste routine needs to be entered as shown below:

Hexadecimal				Symbolic Assembler Instructions			
ADDR	1	2	3	LABEL	MNEM	OPERAND	COMMENTS
0280	A9	FF			LDA#	FF	
0282	8D	20	00		STA	0020	
0285	A9	FF		LOOP2	LDA#	FF	
0287	8D	21	00		STA	0021	
028A	CE	21	00	LOOP1	DEC	0021	
028D	D0	FB			BNE	LOOP1	
028F	CE	20	00		DEC	0020	
0292	D0	F1			BNE	LOOP2	
0294	60				RTS		

Time Waste Sub-Routine

Examination of this program shows that it consists of two loops counting down from FF in each case.

Interrupt Service Routine

Hexadecimal				Symbolic Assembler Instructions			
ADDR	1	2	3	LABEL	MNEM	OPERAND	COMMENTS
0250	EE	00	09		INC	DRB	Increase by 1 count at Port B
0253	40				RTI		Continue counting at Port A

Interrupt Vector Loadings

The interrupt vectors (start address of the interrupt service routines) are located at:

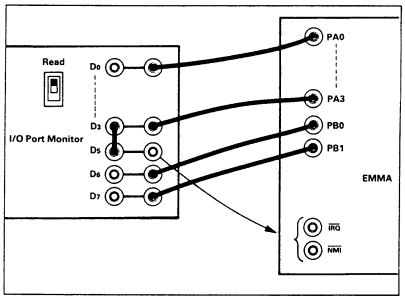
0EFC	Low byte	$\overline{\text{NMI}}$ Vector
0EFD	high byte	
0EFE	low byte	$\overline{\text{IRQ}}$ Vector
0EFF	high byte	

These locations must be loaded with the start address (interrupt vectors) of its interrupt service routine (0250). Hence:

Hexadecimal				Symbolic Assembler Instructions			
ADDR	1	2	3	LABEL	MNEM	OPERAND	COMMENTS
0EFC	50						$\overline{\text{NMI}}$ Vectors
0EFD	02						
0EFE	50						$\overline{\text{IRQ}}$ Vectors
0EFF	02						

Also enter the interrupt service routine as can be seen above.

We now require a connection diagram.



The arrangement is such that Port A will provide a binary count up to 15 (Denary) which will be indicated on the LED s of the I/O Port Monitor. D5 will also indicate the status of the Interrupt Line and D6 and D7 will indicate the number of interrupts. This connection will firstly be taken to the $\overline{\text{NMI}}$ and secondly to the $\overline{\text{IRQ}}$.

- Run the main program from 0200 and observe the I/O Port Monitor.

We have already stated that the $\overline{\text{NMI}}$ interrupt is enabled low and on the negative edge (transition from high to low). You should observe this at the instant when Port B is incremented immediately following the reset of Port A to zero from a full count of 15 (denary).

- Now consider the Interrupt Request $\overline{\text{IRQ}}$. This signal differs from the $\overline{\text{NMI}}$ in that the $\overline{\text{NMI}}$ is negative edge enabled while the $\overline{\text{IRQ}}$ is level (low) enabled. We will keep to the same basic program except that some modifications will be necessary due to the difference in the interrupt enabling signals. We will present the programs and then discuss the modifications.

Main program

Hexadecimal				Symbolic Assembler Instructions			
ADDR	1	2	3	LABEL	MNEM	OPERAND	COMMENTS
0200	A9	FF			LDA#	FF	Initialise Ports
0202	8D	02	09		STA	DDRB	A and B to
0205	8D	03	09		STA	DDRA	output
0208	A9	00			LDA#	00	Resets Port B
020A	8D	00	09		STA	DRB	to zero
0200	A9	10			LDA#	10	Sets Port A
020F	8D	01	09		STA	DRA	bit 4 to 'high'
0212	58						
0213	EE	01	09	NEXT	INC	DRA	
0216	20	80	02		JSR	0280	Timewaste SR
0219	4C	13	02		JMP	NEXT	

Interrupt Service Routine:

Hexadecimal				Symbolic Assembler Instructions			
ADDR	1	2	3	LABEL	MNEM	OPERAND	COMMENTS
0250	A9	10			LDA#	10	Resets
0252	0D	01	09		ORA	DRA	Port A
0255	8D	01	09		STA	DRA	bit 4
0258	EE	00	09		INC	DRB	
0258	40				RTI		

The Time Waste Sub-Routine and the Interrupt Vectors remain unchanged.

- The program will require that the $\overline{IRQ}$ signal is obtained from Port A, bit 4 rather than bit 3 as in previous exercise. Now let's consider the modifications to the program.
- If Port A is set originally to zero, an interrupt will be enabled immediately the Interrupt Flag in the microprocessor Status Register is cleared. Obviously this must be disallowed since we have not yet started counting! Bit 4 (connected to interrupt $\overline{IRQ}$) is thus set high by storing 10 to DRA.
- When Port A is incremented it will now start counting at 10 (Hex) rather than 00 (Hex). When F (HEX) is reached (this will represent a denary count of 15 on Bits 0-3 at Port A) a further increment should cause an interrupt (to increment Port B) and reset Bits 0 - 3 at Port A to zero. The actual output at Port A will go from 1F to 20, so causing the required interrupt (bit 4 goes low).
- Before leaving the interrupt routine we must set Bit 4 high otherwise upon return we will immediately break to interrupt and continue to do so without further counts at Port A taking place. The setting of bit 4 has been done by loading the accumulator with 10, performing a logical OR on the accumulator with the actual output of Port A, and storing the result back to Port A.

Exercise

Change the interconnections between the I/O Port Monitor and EMMA II, enter the new program (not forgetting the Time Waste and Interrupt Vectors if the machine has been switched off) and run the program.

You should observe that the effect is exactly the same as before.

Chapter 8 Hardware Timers (VIA)

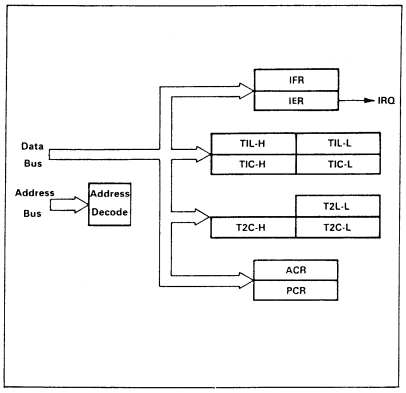
In the previous programs we used a time waste sub-routine as shown below:

Hexadecimal				Symbolic Assembler Instructions			
ADDR	1	2	3	LABEL	MNEM	OPERAND	COMMENTS
0260	A9	FF			LDA#	FF	
0262	8D	FE	00		STA	00FE	
0265	A9	FF		LOOP2	LDA#	FF	
0267	8D	FF	00		STA	00FF	Time Waste
026A	CE	FF	00	LOOP1	DEC	00FF	subroutine
026D	D0	FB			BNE	LOOP1	
026F	CE	FE	00		DEC	00FE	
0272	D0	F1			BNE	LOOP2	
0274	60				RTS		

This is not a particularly efficient way to use the microprocessor since during a software generated time waste the processor is unable to perform any other work. In control systems many such time wastes are required and hence the processor could be very inefficiently used. A better solution to this time waste/efficiency problem is to use hardware timers. These can be triggered off by the microprocessor and configured to interrupt the processor upon time out. Meanwhile the microprocessor can be performing other useful functions.

The Versatile Interface Adaptor (6522) has two timers which can be programmed to count out predetermined periods. They can be programmed to interrupt the microprocessor upon count out or the microprocessor can be programmed to read the timer at intervals and take appropriate action when the timer has timed out.

Simplified Architecture 6522 Interval Timers



Each of the blocks in the diagram are fully addressable and are identifiable as shown below:

Label	Designation	Address
T1C-L	Timer 1 low-order latch (WRITE) Timer 1 low-order counter (READ)	0904
T1C-H	Timer 1 high-order counter (WRITE)	0905
T1L-L	Timer 1 low-order latch (WRITE)	0906
T1L-H	Timer 1 high-order latch (WRITE)	0907
T2C-L	Timer 2 low-order latch (WRITE) Timer 2 low-order counter (READ)	0908
T2C-H	Timer 2 high-order counter (WRITE)	0909
ACR	Auxiliary Control Register	090B
IFR	Interrupt Flag Register	090D
IER	Interrupt Enable Register	090E
PCR	Peripheral Control Register	090C

You should observe that there is a slight difference between the two timers.

We will now look at each of the registers, considering timer 1 only.

Auxiliary Control Register (ACR)

Two bits 6 and 7 of the Auxiliary Control Register determine the four operating modes of Timer 1. Each mode will affect both the Interrupt Flag Register (bit 6 in particular) and bit 7 of output Port B. The four modes are outlined in the table below and are selected when the appropriate code is written into bits 6 and 7 of ACR.

Mode	ACR 7	Bit No 6	Operation	Port B Bit 7
1	0	0	One-shot Mode. A single interrupt occurs upon time-out of timer 1	DISABLED
2	0	1	Free Run Mode. Upon time-out the counter is automatically reloaded and a new time-out period begins. An interrupt occurs upon each time-out.	DISABLED
3	1	0	As for Mode 1	Goes low for duration of timed period.
4	1	1	As for Mode 2	Output inverted upon each time-out producing a square wave of equal mark/space ratio. Unless counter is re-loaded from latches creating a new time period.

Interrupt Flag Register (IFR)

Bit 6 of the Interrupt Flag Register is set upon time-out of Timer 1. This bit is cleared by either reading Timer 1, low-order counter (T1C-L), by writing Timer 1, high-order counter (T1C-H) or by writing a '1' directly to the flag.

Interrupt Enable Register (IER)

Bit 6 of the Interrupt Enable Register corresponds to bit 6 of the Interrupt Flag Register. A '1' in this bit will enable the interrupt while a '0' will disable. However, bits in this register are under program control as follows:

With bit 7 at '0', a '1' in bit 6 will CLEAR interrupt enable, while a '0' will leave it unaffected.

With bit 7 at '1', a '1' in bit 6 will SET interrupt enable, while a '0' will leave it unaffected.

Note: both IFR and IER are 8-bit registers and provide flags for other modes of operation of the VIA as well as for Timer 1.

Latches/Counters

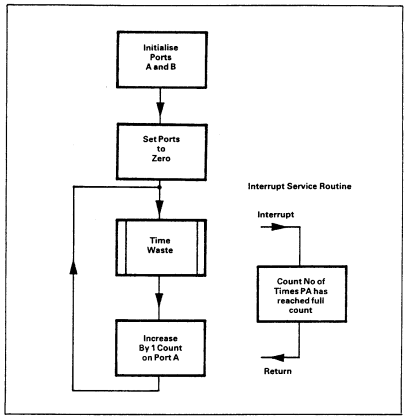
Two 8-bit latches designated low-order and high-order respectively are provided for Timer 1. Associated with these are two 8-bit counters, also designated low-order and high-order respectively. The latches are used to store data which is to be loaded into the counter. After loading, the counter is decremented at phase 2 (Ø2) clock rate (1 micro second). Upon reaching zero, an interrupt flag (bit 6 of IFR) is set and the $\overline{IRQ}$ line will go low if the interrupt is enabled (bit 6 of IER set to '1'). Further interrupts will be disabled by the timer unless programmed to automatically transfer the contents of the latches into the counter and begin to decrement again.

Programming Hardware Timer 1

Task

We will write a simple program which, with minor modifications, will demonstrate each of the modes of operation of the timers.

Flow Chart



The program is effectively in four parts:

- Main Program
- Interrupt Service Routine
- Time Waste Sub-Routine
- Interrupt Vector Loadings

Main Program

The main program is designed to initialise the microprocessor interrupt flags, the VIA interrupt flags, the VIA mode of operation and finally to set Timer 1 time interval.

The program is:

	Hexadecimal			Symbolic Assembler Instructions			
ADDR	1	2	3	LABEL	MNEM	OPERAND	COMMENTS
0200	58				CLI		Enables Processor $\overline{IRQ}$
0201	A9	C0			LDA#	C0	Enables VIA $\overline{IRQ}$
0203	8D	0E	09		STA	IER	
0206	A9	*			LDA	*	Sets mode of timer 1
0208	8D	0B	09		STA	ACR	operation
0208	A9	FF			LDA#	FF	Sets
020D	8D	04	09		STA	TIC-L	Timer 1
0210	A9	FF			LDA#	FF	time
0212	8D	05	09		STA	TIC-H	interval
0215	4C	15	02	WAIT	JMP	WAIT	Program End

* Mode of operation Codes. (Enter CØ for the present).

Code	Operation	PB7
00	Single Shot	Disabled
40	Free Run	Disabled
80	Single Shot	Enabled
C0	Free Run	Enabled

Interrupt Service Routine

The interrupt service routine is designed to initialise Port A bits 0-6 to output, increment this port to a full count on bits 0-6, reset the VIA interrupt flag and then return from interrupt.

The program is:

	Hexadecimal			Symbolic Assembler Instructions			
ADDR	1	2	3	LABEL	MNEM	OPERAND	COMMENTS
0250	A9	FF			LDA#	FF	Initiali e Port A,
0252	8D	03	09		STA	DDRA	bits Ø-7 to output
0255	A9	ØØ			LDA#	ØØ	Sets Port A
0257	8D	01	09		STA	DRA	to zero
025A	EE	01	09	REP	INC	DRA	
025D	20	80	02		JSR	0280	Time waste SR
0260	A9	7F			LDA#	7F	Tests DRA for
0262	CD	01	09		CMP	DRA	full count
0265	D0	F3			BNE	REP	
0267	A9	40			LDA#	40	Clears VIA
0269	8D	0D	09		STA	IFR	Timer 1 $\overline{IRQ}$ flag
026C	40				RTI		

Time Waste Sub-Routine

The time waste subroutine is simply included to allow the I/O Port Monitor to be used to monitor Port A, bits 0-6. It is a straight forward double-loop routine which you will be familiar with.

The program is:

	Hexadecimal			Symbolic Assembler Instructions			
ADDR	1	2	3	LABEL	MNEM	OPERAND	COMMENTS
0280	A9	FF			LDA#	FF	
0282	85	20			STA	20	
0284	A9	FF		LOOP2	LDA#	FF	
0286	85	21			STA	21	
0288	C6	21		LOOP1	DEC	21	
028A	D0	FC			BNE	LOOP1	
028C	C6	20			DEC	20	
028E	D0	F4			BNE	LOOP2	
0290	60				RTS		

Interrupt Vector Loadings

The VIA Timer 1 causes an Interrupt request ($\overline{IRQ}$) at the VIA and hence the microprocessor itself (the VIA $\overline{IRQ}$ output pin is connected on the printed circuit board, to the microprocessor $\overline{IRQ}$ input pin) breaks to locations 0EFE and 0EFF.

Hence 0EFE and 0EFF must be loaded with the start address of the Interrupt Service Routine.

0EFE - Low Byte
0EFF - High Byte

(In this instance, the start address, and hence the vectors of the interrupt service routine are 0250.) These must be loaded before the program is run.

Now let's enter these four components of our TIMER 1 program. But first let's decide on our mode of operation, (see * on Main Program).

- Mode 1 (Code 00)
  The program will operate to increment Port A bits 0-6 to a single full count only.
- Mode 2 (Code 40)
  The program will operate to increment Port A, bits 0-6 reset Port A to zero and continue to increment. The microprocessor will continue this operation repeatedly.
- Mode 3 (Code 80)
  The program will operate as in Mode 1 above, but bit 7 of Port B will also be enabled. PB7 will be pulsed low for the duration of the Timer 1 time interval (approx, 0.6 second).
- Mode 4 (Code C0)
  The program will operate as in Mode 2 above but bit 7 of Port B will also be enabled. PB7 will generate a continuous square wave.

Load into **EMMA II** the four components of the program and run the program for each of the mode of operation codes. Observe results.

With the Timer 1 program entered and with mode 4 selected, change the main program such that the microprocessor $\overline{IRQ}$ is disabled (change location 0200 from CLI, op. code, to SEI, op. code 78). Observe results. Note that a similar result can be obtained by disabling the interrupt at the VIA (load IER with 80)

Run the program in mode 1 but enter No Operations (op. code EA) at memory locations 0267-026B inclusive. This will show the importance of clearing a **Device Interrupt Flag** before returning from interrupt. Although Timer 1 is in a single-shot mode, the VIA interrupt request line $\overline{IRQ}$ will only be cleared (taken high) if the counters are reloaded. Since mode 1 does not do this, the Timer 1 interrupt flag (bit 6 in IFR), must be cleared. If this is not done, the microprocessor will continue to be interrupted upon return from each interrupt although the timer itself is not producing interrupts.

The Timer 2 is slightly different from Timer 1. You will find technical details of this timer in the **EMMA II Technical Manual**.

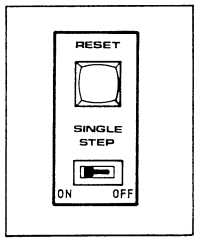
Chapter 9 Program Debugging

If the program does not operate when RUN it is more likely to be your program than the hardware that is at fault! When this happens, as surely it will, you will appreciate the care (or lack of) that you took in firstly constructing your flow chart and secondly providing adequate comments on your assembly listing. Assuming you have met these documentation requirements then you may proceed to use one or other of the debug features provided by the **EMMA II** monitor.

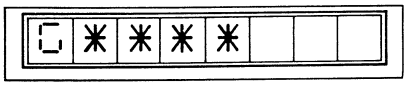
Single Step Mode

This causes the microprocessor to be interrupted after each instruction has been executed, enabling the user to inspect various internal registers. The contents of these registers indicate the result of the last operation performed by the microprocessor.

To use the single step facility.



- Switch single step switch to 'ON'
- Press Reset
- Press [R] key

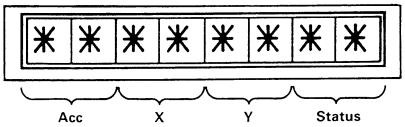


- Set START address
- Press [R] key. This will cause the first instruction to be executed.

The display will now show the contents of the:

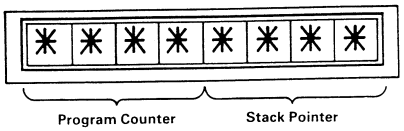
ACCUMULATOR, X-REGISTER, Y-REGISTER and the STATUS REGISTER

Two hexadecimal digits will be devoted to each and displayed on the **EMMA II** keyboard as follows:



- Press the [R] key.

The display will now show the contents of the **Program Counter** and the **Stack Pointer.** Four hexadecimal digits being devoted to each. The program counter gives the address of the next instruction to be executed.



If you now press [R] again the next instruction will be executed and the display will again show the current contents of Acc. X. Y and STATUS. Press [R] again to give PC and SP.

Repeatedly pressing [R] will thus step the program through instruction by instruction, allowing you to inspect each of the processor registers after every instruction.

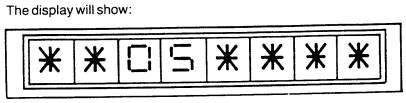
To return to the normal program RUN mode, the single step switch must be returned to 'OFF'. Press the [G] key twice and the program will run normally from its start address.

As an example of single step function enter the following program:

Hexadecimal				Symbolic Assembler Instructions			
ADDR	1	2	3	LABEL	MNEM	OPERAND	COMMENTS
0200	A2	05			LDX#	05	
0202	A0	06			LDY#	06	
0204	A9	07			LDA#	07	
0206	4C	00	02		JMP	0200	

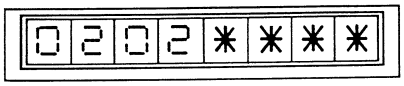
Set the machine single step mode as explained earlier, and set the start address for 0200. Upon the next press of [R] you will see the first instruction has been carried out.

The display will show:



05 has been loaded into the X register

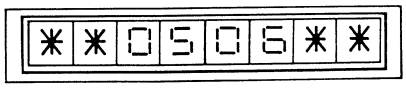
Press [R] again:



Note that the program counter is at 0202.

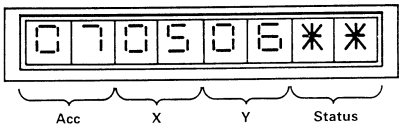
Press [R] again, and the instruction at address 0202 is carried out.

The display will show this:



Press [R] again the program counter shifts to 0204.

Press [R] and the instruction load accumulator is carried out and can be seen on the display as:



Press [R] again, and the program counter is displayed, shifted to 0206.

Press [R] again; the jump instruction is carried out. Note that the registers Acc, X, Y still contain the data previously entered.

Press [R] and the program counter displays the address 0200 indicating that the jump instruction has been carried out.

Obviously this program is a simple operation to show how data is manipulated. The single step facility is of great benefit when fault finding in a program.

The single step mode can be a time consuming process especially if the program is large. The following procedure allows the program to be 'inspected' at set points in the program.

Breakpoint Mode

A breakpoint will interrupt the microprocessor, enabling the user to inspect the state of the internal registers at any point within the program.

Assume that the user suspects that there are problems occuring within a particular part of his program, the breakpoint mode can be used.

```
0200  A2  05          LDX#  05
0202  A0  06          LDY#  06
0204  A9  07          LDA#  07
0206  4C  00  02      JMP   0200
```

Using the previous program we wish to insert a break command at location 0204. Now the break command code is 00 and it is this that must replace A9 at 0204.

Now:

- Press [P] key The display will read: **P . * * * ***
- Set **P . 0 2 0 4** using hexadecimal keys.
- Press P. The display now shows that 00 (Break command code) has been loaded into location **0204** .
- Note that pressing [P] again (twice) restores the data A9 LDA and pressing [P] again (twice) re-introduces the break command.
- With display showing **P . 0 2 0 4 0 0**, press [G] and set display to **G . 0 2 0 0** where 0200 is the start of the program.
- Press G

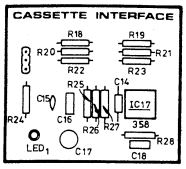
  The program will now RUN to address 0204 and break to the monitor subroutine used for the single step mode. The display will now show: Acc. X, Y and STATUS.

  In this example ****0506****

- Press P. The display will now show:
  PROGRAM COUNTER AND STACK POINTER.

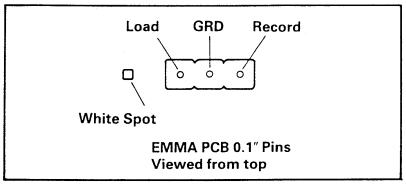
  In this example **0 2 0 4 * * * ***
- Press P twice.
  The break command is removed and the LDA code is restored.
- Press R.
  The program continues to RUN from location 0204.

Chapter 10 Using the Cassette Interface



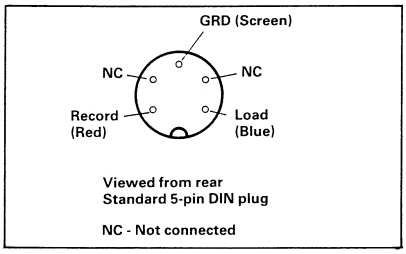
A cassette interface is provided on the 'EMMA II' to allow the use of a normal commercial cassette recorder for the storage of programs. It is recommended that good quality tapes are used and that the tape head is kept clean with the occasional use of a head cleaning tape. Short duration tapes will be less likely to stretch with repeated use and for this reason C120 cassette tapes are not recommended.

Connection to the recorder is best facilitated by the hi-fi DIN connector. If the recorder does not have a DIN facility, then connection can be made via external microphone and ear-phone jack sockets. The cassette connector on the 'EMMA II' is a 3-way pcb plug on the left of the cassette interface block.



Care must be taken to insert the P.C.B. connector correctly i.e. white side towards the white spot on the P.C.B.

Pin connections for the lead are:



Program Save on Cassette

- Press save key [S]
  The display asks for 'FROM' address **F** * * * *
- Press [S] again.
  The display will ask for 'TO' address **t** * * * *
  Enter last address of program +1
- Press [S] again and the display will ask for a baud (speed of transmission) rate. **BAUD** -
  Pressing the '1' key selects 1200 baud
  Pressing the '0' key selects 300 baud
  Do not select this until the cassette is running.
- Press RECORD and PLAY on the cassette recorder.
- Select BAUD RATE; a header in tone is recorded for approx 4 sec (2.4kHz) followed by data. During this time LED 1 is illuminated.
- When the last address is reached LED 1 is extinguished and the last address appears on the display.
- Press STOP on recorder.

Summary

- Press [S] enter start address
- Press [S] enter last address +1
  Press [S]
- Start recorder
- Press [0] or [1] to select baud rate and initiate data transfer.

Program Load from Cassette

The EMMA II interface is tolerant of the amplitude of the cassette output and it is generally acceptable to have the volume and tone controls set to approximately mid-position.

- RESET the microprocessor
- PRESS [L] The display will show: **BAUD**
- Find the program header tone, either with tape counter, or by listening for the 2.4KHz tone.
- During the header tone select baud rate i.e.
  [1] = 1200 baud
  [0] = 300 baud
- As the data is loaded LED 1 is illuminated.
- When the program has loaded the row of dots will reappear on the display.
- Stop the recorder and run the program in the normal way.

Note: The baud rate must be the same as the recorded program when loading.

Summary

- Press [L] "BAUD-" displayed.
- Start recorder.
- During header tone press [1] or [0] to select appropriate baud rate.
- Wait for row of dots on display and then stop the recorder.

NB It is important when saving a program to label the tape containing your program for future reference.

- Title
- Start Address
- Baud Rate
- Tape Counter Location
- Checksum

A 'Checksum' is formed by adding all the data bytes and arriving at a total. This figure can be used to check if data has been corrupted during transmission. Refer to checksum routine page 61.

Chapter 11 Useful Routines/Subroutines

User Routines

Included in the monitor program of EMMA II are sub-routines that perform functions to assist the programmer. This section of the manual will explain how to use these routines.

- **Hexadecimal to Decimal/Decimal to Hexadecimal**
  Routine Start Address = D900
  A mode signal 'H' will appear indicating that the routine is in HEX→DEC mode. The 'H' is asking for a HEX number up to 4 digits to be entered. After a HEX number has been entered pressing [P] "PROCESS", will change the state of the display to decimal and display 'D' in the mode character location and the decimal equivalent of the HEX number on the right of the display.

e.g. H	F5
D	245

- **Branch Offset Calculator**
  Routine Start Address = D980
  This sub-routine does branch calculations for the programmer. Example:

  Run the program from D980; the mode digit displays 'F' asking for a "FROM" address to be entered.

  e.g. Enter 0045 as "FROM" address by pressing [P] 'Process' The mode character 't' will appear asking for a "TO" address. For this example enter 0060.
  Press [P] again and the mode character will display a '0' meaning that it is in the offset mode and display a two digit offset, '19' in this example.

  If the branch is out of range, the mode character will show 'E' meaning ERROR.

  In both of these cases pressing [R] (Re-Run) will re-run the program.

  i.e. Return to the "FROM" situation for the next calculation. By pressing [L] the monitor program runs and a reset situation exists.

- **Relocator**
  Routine Start Address = D9C9
  This program is designed to move blocks of program to new address locations. To use this program run from "D9C9". The mode digit means "F" asking for start address of block to be moved for an example enter 0245 as start address. Pressing [P] "PROCESS" the mode digit will display 't' indicating the need for an end address 1 of the block to be moved, for this example enter 0260.

  To relocate do NOT overlap the shift block and the new block location (ie will corrupt if overlapped).

  Pressing [P] key again the command will be 'd' asking for a destination address, for the example enter 0800.

  By pressing the [P] key the program will now transfer data from 0245 - 025F to a new address 0800; and return to the monitor program (a row of dots will appear on the display.)

- **Checksum Routine**
  Routine Start Address = DA50
  A checksum is done by adding a group of data bytes (e.g. a program) together and forming a total to be used for comparison; for example after transmission etc a bit could be corrupted hence the program would have a different checksum. To use this routine:

  Go from DA50
  The mode character displays 'F' asking for the start address of the block to be checksummed.

  Enter address.
  Press [P] the mode displays 'N' asking for the number of bytes.
  Enter number of bytes.

  Press [P]
  The mode character will display 'C' indicating a checksum and the checksum appears on the right of the display.

  By pressing [0] the program returns to the monitor and a reset condition.

Useful Subroutines

- **Data Insert Routine**
  Routine Start Address = D805
  This routine can be used in a program to ask the user to insert data to be used as a function of the program. To utilise the program the 'Y' register needs to be set to the number of characters of the data. The start address of the sub-routine is D805. For example enter the small program.

```
0200  A0  05      LDY # 05  Number of characters
0202  20  05  D8  JSR D805  'Insert' subroutine
0205  4C  E0  FE  JMP FEE0  Reset
```

Run the program from 0200.
The display will show G 0200.
Now you can enter data up to 5 HEX characters. The displayed characters will shift left as this is being loaded. For this example, enter 01234. By pressing PROCESS key [P] the program is executed. The data has been converted into a 4 byte code and stored at 001C - 001F by examination.

```
1C = 34
1D = 12
1E = 00
1F = 00
```

From the example the routine uses both the X and the Y registers, which are restored on exit.

- **Display 8**
  Routine Start Address = D849
  This routine transfers data from 001C - 001F to the display buffer in seven segment form. Zero suppression is also performed.
  For example load locations 001C - 001F with the following:

```
001C  67
001D  45
001E  23
001F  01
```

Run from D849 and the displayed data will be [1234567]

- **Display**
  Routine Start Address = D84D
  This routine is similar to 'DISPLAY 8'; the only difference is the data transferred for display is from 1C - 1E. Digit 1 of the display is cleared and digit 0 is loaded from 1B. 1B can be loaded to give a symbol or letter as a code of information. The data in our example will display length **L** at 7500.

```
001B   38   Seven segment code for 'L'

001C   00
001D   75   Data (7500)
001E   00
```

Run from D84D.
The display will read **L** 7500

- **Multiply By 10**
  Routine Start Address = D89D
  This routine multiplies the 3 byte number in 0A - 0C by 10_{10} and stores the result in HEX back in 0A - 0C. Care must be taken to ensure that the monitor program does not corrupt the answer. (i.e. if the monitor uses 0A - 0C)

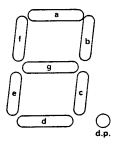
- **Multiply by 16**
  Routine Start Address = D8B3
  This routine multiplies the 3 byte number in 0A - 0C by 16_{10} and stores the result at 0A - 0C.

- **Display Memory Contents**
  Routine Start Address = FE0C
  This routine displays seven segment codes stored at: 0010 - 0017. If the contents of 000E is less than 80 the display is scanned once and returns from the sub-routine. If the contents of 000E are greater than 80 the display is continuously scanned until a key is operated, at which point the accumulator and location 000D are loaded with the hex value of the key and a return from subroutine is performed.

To show how this sub-routine can be used enter the following program which loads the contents of 0020 to 0027 into 0010 to 0017 and then displays them.

	Hexadecimal			Symbolic Assembler Instructions			
ADDR	1	2	3	LABEL	MNEM	OPERAND	COMMENTS
0300	A9	80			LDA	#80	
0302	85	0E			STA	0E	
0304	A2	00			LDX	#00	
0306	B5	20		*	LDA	20,X	
0308	95	10			STA	10,X	
030A	E8				INX		
030B	E0	08			CPX	#08	
030D	D0	F7			BNE	*	
030F	20	0C	FE		JSR	FE0C	
0312	8D	00	02		STA	0200	
0315	4C	E0	FE		JMP	FEE0	

Then calculate the data to display a message and load this into locations 0020 to 0027.
The codes required to illuminate each segment are shown below:



Segment	Code
a	01
b	02
c	04
d	08
e	10
f	20
g	40
d.p	80

Run the program and observe that the message is displayed.

Operate one of the keys and note that control is returned to the monitor program. Examine location 0200 which will contain the hex value of the key used to exit the subroutine.

Write 4 Characters
Routine Start Address = FE00

Displays contents of (0000 + X − 3) (0000 + X − 2) (0000 + X − 1) and (0000 + X), from left to right, on the display.

For example: Load 01 into location 002D, 23 into location 002E, 45 into location 002F, 67 into location 0030.

Load the following simple program starting at location 0300:-

```
0300   A2,30      LDX#
0302   20,00,FE   JSR  FE00
```

Run the program from 0300, i.e. press 'G', 0300, 'G'.

The display will show the contents of locations 002D to 0030 inclusive. This subroutine drops through to subroutine FE0C to display the information.

Note: The subroutine uses the accumulator, X register and Y register, so if information in these registers is needed, then it must be stored elsewhere before entering the subroutine.

Display Memory Contents
Routine Start Address = FE5E

Converts the contents of the indirect address formed from locations (0000 + X) and (0000 + X + 1) into seven segment codes, and stores them in locations 0016 and 0017. Jumping to subroutine FE0C will cause the information to be displayed on the right hand side of the display.

Load the following program starting at 0300:

```
0300  A2,00      LDX#00
0302  8A         TXA
0303  95,10      * STA  10,X   clears
0305  E8         INX           locations
0306  E0,08      CPX#08        0010-0017
0308  D0,F9      BNE  *
030A  A9,80      LDA#80
030C  85,0E      STA  0E
030E  A2,20      LDX#20
0310  20,5E,FE   JSR  FE5E
0313  20,0C,FE   JSR  FE0C
```

Load 0020 with 30 and 0021 with 00. Load location 30 with AB.

Run the program from 0300; the display will show AB on the two righthand digits. The subroutine starting at FE5E uses the numbers stored in 0020 and 0021 (X = 20) to form an indirect address (in this case 0030) and then converts the contents of this location into seven segment codes and stores them at 0016 and 0017. Jumping to the subroutine at FE0C displays these characters.

The subroutine uses the accumulator and Y register.

Display Accumulator Contents
Routine Start Address = FE60

Converts the contents of the accumulator into seven segment codes and stores them in locations 0016 and 0017. To display the contents of these locations, jump to subroutine FE0C on returning from the subroutine at FE60.

To show how these subroutines can be used, load the following program:-

```
0300  A2,00          LDX#00
0302  8A             TXA
0303  95,10       *  STA  10,X   clears
0305  E8             INX         locations
0306  E0,08          CPX#08      0010-0017
0308  D0,F9          BNE *
030A  A9,80          LDA#80
030C  85,0E          STA  0E
030E  A5,30          LDX  0030
0310  20,60,FE       JSR  FE60
0313  20,0C,FE       JSR  FE0C
```

In this program the contents of 0030 are loaded into the accumulator. On jumping to subroutine FE60, this information is converted into seven segment codes and stored at 0016 and 0017. Jumping to subroutine FE0C displays this information on the right of the display.

The subroutine uses the accumulator and Y register.

Display 4 Characters
Routine Start Address = FE64

Converts the contents of locations (0000 + X + 1) and (0000 + X) into seven segment codes and stores them at locations 0011, 0012, 0013 and 0014 respectively. Jumping to subroutine FE0C will display this information on the left hand centre of the display.

To show how this subroutine can be used, load the following program starting at 0300:

```
0300  A2,00          LDX#00
0302  8A             TXA
0303  95,10       *  STA  10,X
0305  E8             INX
0306  E0,08          CPX#08
0308  D0,F9          BNE *
030A  A9,80          LDA#80
030C  85,0E          STA  0E
030E  A2,30          LDX#30
0310  20,64,FE       JSR  FE64
0313  20,0C,FE       JSR  FE0C
```

Load location 0030 with 23 and 0031 with 01. Run the program and 0123 will be displayed.

The program displays the contents of locations 0030 and 0031 by first jumping to the subroutine at FE64. This converts the information into seven segment codes and stores these at location 0011, 0012, 0013 and 0014. The program then jumps to the subroutine at FE0C to display the information.

The subroutine uses the accumulator and Y register.

Read Hexadecimal Keys
Routine Start Address = FE88

Shifts data entered on the keyboard hexadecimal keys into memory locations (0000 + X + 1) and (0000 + X), the subroutine then jumps to the subroutines starting at addresses FE64 and FE0C to display the information in the second, third, fourth and fifth digit positions (from the left side of the display). When a command key is pressed the subroutine is exited with the value of the command key stored in the accumulator and at address 000D. Also, on exiting the subroutine, the information which has been entered will be stored in locations(0000 + X + 1) and (0000 + X); in this case,0031 and 0030.

Load the following program:-

```
0300  A2,00          LDX#00
      8A             TXA
      95,10       *  STA 10,X
      E8             INX
      E0,08          CPX#08
      D0,F9          BNE *
      A2,30          LDA#30
      95,00          STA 00,X
      95,01          STA 01,X
      20,88,FE       JSR FE88
      4C,13,03    ** JMP **
```

Run the program from 0300 and enter hexadecimal information via the keyboard. Notice that the information enters the display. Press a command key and note that the display goes blank. Press RESET and then examine memory locations 0030 and 0031; these will contain the last four digits entered via the keyboard.

The subroutine uses the accumulator and the Y register.

Key values are:

M	10	L	14	0	00	4	04	8	08	C	0C
G	11	R	15	1	01	5	05	9	09	D	0D
P	12	+	16	2	02	6	06	A	0A	E	0E
S	13	−	17	3	03	7	07	B	0B	F	0F

Output Data Through The Cassette Interface (To Tape)
Routine Start Address = FEB1

This subroutine takes an 8-bit parallel code from the accumulator and outputs this byte as a serial code through the cassette interface.

The following program illustrates the use of subroutine FEB1:-

```
0300  A2,00          LDX#00
0302  BD,00,02    *  LDA  0200,X
0305  20,B1,FE       JSR  FEB1
0308  E8             INX
0309  D0,F7          BNE *
030B  4C,E0,FE       JMP  FEE0
```

Press reset before running the program and the program will output all of page 02 through the cassette interface.

If required, the following program can be run to store AA and 55 in alternate locations on page 02. The program given above can then be used to store this data on tape.

```
0350  A2,00          LDX#00
0352  A9,AA          LDA#AA
0354  85,20          STA  20
0356  A5,20          LDA  20
0358  49,FF          E0R#FF
035A  85,20          STA  20
035C  9D,00,02       STA  0200,X
035F  E8             INX
0360  D0,F4          BNE  F4
0362  4C,E0,FE       JMP  FEE0
```

The subroutine at address FEB1 uses the accumulator and Y register.

Input Data Through the Cassette Interface (from tape)
Routine Start Address = FEDD

Loads a byte from tape into the accumulator. In loading this byte, the subroutine takes a serial 8-bit code from the cassette interface and converts it to a parallel code, placing this parallel byte in the accumulator.

The following program illustrates the use of this subroutine:-

```
0300    A2,00       LDX#00
0302    20,DD,FE    JSR  FEDD
0305    9D,00,02    STA  0200,X
0308    E8          INX
0309    D0,F7       BNE  F7
030B    4C,FE,E0    JMP  FEE0
```

This program loads 256 bytes of data from the tape and stores it on page 02.

If the program given in the TO TAPE routine is run first, then the FROM TAPE routine can be used to read the data back from the tape.

The subroutine uses the accumulator and Y register.

Appendix A 6502 Instruction Set

The following notation applies to this summary:

A	Accumulator
X,Y	Index Registers
M	Memory
P	Processor Status Register
S	Stack Pointer
√	Change
-	No Change
+	Add
∧	Logical AND
–	Subtract
∀	Logical Exclusive Or
↑	Transfer from Stack
↓	Transfer to Stack
→	Transfer to
←	Transfer from
V	Logical OR
PC	Program Counter
PCH	Program Counter High
PCL	Program Counter Low
OPER	Operand
#	Immediate Addressing Mode

ADC **Add memory to accumulator with carry** **ADC**

Operation: A + M + C → A, C

N	Z	C	I	D	V
√	√	√	-	-	√

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Immediate	ADC # Oper	69	2	2
Zero Page	ADC Oper	65	2	3
Zero Page, X	ADC Oper,X	75	2	4
Absolute	ADC Oper	6D	3	4
Absolute, X	ADC Oper, X	7D	3	4*
Absolute, Y	ADC Oper, Y	79	3	4*
(Indirect, X)	ADC (Oper, X)	61	2	6
(Indirect), Y	ADC (Oper),Y	71	2	5*

*** Add 1 if page boundary is crossed.**

AND **"AND" memory with accumulator** **AND**

Logical AND to the accumulator
Operation: A ∧ M → A

N	Z	C	I	D	V
√	√	-	-	-	-

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Immediate	AND # Oper	29	2	2
Zero Page	AND Oper	25	2	3
Zero Page, X	AND Oper, X	35	2	4
Absolute	AND Oper	2D	3	4
Absolute, X	AND Oper, X	3D	3	4*
Absolute, Y	AND Oper, Y	39	3	4*
(Indirect, X)	AND (Oper, X)	21	2	6
(Indirect), Y	AND (Oper),Y	31	2	5*

*** Add 1 if page boundary is crossed.**

ASL **ASL Shift Left One Bit (Memory or Accumulator)** **ASL**

Operation: C ← 7 6 5 4 3 2 1 Ø ← Ø

N	Z	C	I	D	V
√	√	√	-	-	-

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Accumulator	ASL A	ØA	1	2
Zero Page	ASL Oper	Ø6	2	5
Zero Page, X	ASL Oper, X	16	2	6
Absolute	ASL Oper	ØE	3	6
Absolute, X	ASL Oper, X	1E	3	7

BCC **BCC Branch on Carry Clear** **BCC**

Operation: Branch on C = Ø

N Z C I D V
- - - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Relative	BCC Oper	9Ø	2	2*

* Add 1 if branch occurs to same page
* Add 2 if branch occurs to different page

BCS **BCS Branch on Carry Set** **BCS**

Operation: Branch on C = 1

N Z C I D V
- - - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Relative	BCS Oper	BØ	2	2*

* Add 1 if branch occurs to same page
* Add 2 if branch occurs to different page

BEQ **BEQ Branch on Result Zero** **BEQ**

Operation: Branch on Z = 1

N Z C I D V
- - - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Relative	BEQ Oper	FØ	2	2*

* Add 1 if branch occurs to same page
* Add 2 if branch occurs to different page

BIT **BIT Test Bits In Memory with Accumulator** **BIT**

Operation: A ∧ M, M7 → N, M6 → V
Bit 6 and 7 are transferred to the status register.
If the result of A ∧ M is zero then Z = 1, otherwise Z = Ø

N Z C I D V
M7 √ - - - M6

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Zero Page	BIT Oper	24	2	3
Absolute	BIT Oper	2C	3	4

BMI **BMI Branch on result minus** **BMI**

Operation: Branch on N = 1

N Z C I D V
- - - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Relative	BMI Oper	3Ø	2	2*

* Add 1 if branch occurs to same page
* Add 2 if branch occurs to different page

BNE **BNE Branch on result not zero** **BNE**

Operation: Branch on Z = 0

N Z C I D V
- - - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Relative	BNE Oper	DØ	2	2*

* Add 1 if branch occurs to same page
* Add 2 if branch occurs to different page

BPL **BPL Branch on result plus** **BPL**

Operation: Branch on N = Ø

N Z C I D V
- - - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Relative	BPL Oper	1Ø	2	2*

* Add 1 if branch occurs to same page
* Add 2 if branch occurs to different page

BRK **BRK Force Break** **BRK**

Operation: Forced Interrupt PC + 2 ↓ P ↓

N Z C I D V
- - - 1 - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	BRK	ØØ	1	7

* A BRK command cannot be masked by setting I.

BVC **BVC Branch on Overflow Clear** **BVC**

Operation: Branch on V = 0

N Z C I D V
- - - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Relative	BVC Oper	5Ø	2	2*

* Add 1 if branch occurs to same page
* Add 2 if branch occurs to different page.

BVS **BVS Branch on Overflow Set** **BVS**

Operation: Branch on V = 1

N Z C I D V
- - - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Relative	BVS Oper	7Ø	2	2*

* Add 1 if branch occurs to same page
* Add 2 if branch occurs to different page.

CLC **CLC Clear Carry Flag** **CLC**

Operation: Ø → C

N Z C I D V
- - Ø - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	CLC	18	1	2

CLD **CLD Clear Decimal Mode** **CLD**

Operation: Ø → D

N Z C I D V
- - - - Ø -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	CLD	D8	1	2

CLI **CLI Clear Interrupt Disable Bit** **CLI**

Operation: Ø → I

N Z C I D V
- - - Ø - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	CLI	58	1	2

CLV **CLV Clear Overflow Flag** **CLV**

Operation: Ø → V

N Z C I D V
- - - - - Ø

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	CLV	B8	1	2

CMP **CMP Compare Memory and Accumulator** **CMP**

Operation: A - M

N Z C I D V
√ √ √ - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Immediate	CMP # Oper	C9	2	2
Zero Page	CMP Oper	C5	2	3
Zero Page, X	CMP Oper,X	D5	2	4
Absolute	CMP Oper	CD	3	4
Absolute, X	CMP Oper, X	DD	3	4*
Absolute, Y	CMP Oper, Y	D9	3	4*
(Indirect, X)	CMP (Oper, X)	C1	2	6
(Indirect), Y	CMP (Oper),Y	D1	2	5*

*** Add 1 if page boundary is crossed.**

CPX **CPX Compare Memory and Index X** **CPX**

Operation: X-M

N Z C I D V
√ √ √ - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Immediate	CPX #Oper	EØ	2	2
Zero Page	CPX Oper	E4	2	3
Absolute	CPX Oper	EC	3	4

CPY **CPY Compare Memory and Index Y** **CPY**

Operation: Y - M

N Z C I D V
√ √ √ - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Immediate	CPY #Oper	CØ	2	2
Zero Page	CPY Oper	C4	2	3
Absolute	CPY Oper	CC	3	4

DEC **DEC Decrement Memory by One** **DEC**

Operation: M - 1 → M

N Z C I D V
√ √ - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Zero Page	DEC Oper	C6	2	5
Zero Page, X	DEC Oper, X	D6	2	6
Absolute	DEC Oper	CE	3	6
Absolute, X	DEC Oper, X	DE	3	7

DEX **DEX Decrement Index X by One** **DEX**

Operation: X - 1 → X

N Z C I D V
√ √ - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	DEX	CA	1	2

DEY **DEY Decrement Index Y by One** **DEY**

Operation: Y - 1 → Y

N Z C I D V
√ √ - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	DEY	88	1	2

EOR **"Exclusive-Or" Memory with Accumulator** **EOR**

Operation: A ∀ M→A

N Z C I D V
√ √ - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Immediate	EOR # Oper	49	2	2
Zero Page	EOR Oper	45	2	3
Zero Page, X	EOR Oper, X	55	2	4
Absolute	EOR Oper	4D	3	4
Absolute, X	EOR Oper, X	5D	3	4*
Absolute, Y	EOR Oper, Y	59	3	4*
(Indirect, X)	EOR (Oper, X)	41	2	6
(Indirect), Y	EOR (Oper), Y	51	2	5*

*** Add 1 if page boundary is crossed.**

INC **INC Increment Memory by One** **INC**

Operation: M + 1 → M

N Z C I D V
√ √ - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Zero Page	INC Oper	E6	2	5
Zero Page, X	INC Oper, X	F6	2	6
Absolute	INC Oper	EE	3	6
Absolute, X	INC Oper, X	FE	3	7

INX **INX Increment Index X by One** **INX**

Operation: X + 1 → X

N Z C I D V
√ √ - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	INX	E8	1	2

INY **INY Increment Index Y by One** **INY**

Operation: Y + 1 → Y

N Z C I D V
√ √ - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	INY	C8	1	2

JMP **JMP Jump to New Location** **JMP**

Operation: (PC + 1) → PCL
(PC + 2) → PCH

N Z C I D V
- - - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Absolute	JMP Oper	4C	3	3
Indirect	JMP (Oper)	6C	3	5

JSR **Jump to New Location Saving Return Address** **JSR**

Operation: PC + 2 ↓, (PC + 1) → PCL
(PC + 2) → PCH

N Z C I D V
- - - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Absolute	JSR Oper	20	3	6

LDA **Load Accumulator with Memory** **LDA**

Operation: M → A

N Z C I D V
/ √ - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Immediate	LDA # Oper	A9	2	2
Zero Page	LDA Oper	A5	2	3
Zero Page, X	LDA Oper,X	B5	2	4
Absolute	LDA Oper	AD	3	4
Absolute, X	LDA Oper, X	BD	3	4*
Absolute, Y	LDA Oper, Y	B9	3	4*
(Indirect, X)	LDA (Oper, X)	A1	2	6
(Indirect), Y	LDA (Oper), Y	B1	2	5*

*** Add 1 if page boundary is crossed.**

LDX **LDX Load Index X with Memory** **LDX**

Operation: M → X

N Z C I D V
√ √ - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Immediate	LDX # Oper	A2	2	2
Zero Page	LDX Oper	A6	2	3
Zero Page, Y	LDX Oper, Y	B6	2	4
Absolute	LDX Oper	AE	3	4
Absolute, Y	LDX Oper, Y	BE	3	4*

*** Add 1 if page boundary is crossed.**

LDY **LDY Load Index Y with Memory** **LDY**

Operation: M → Y

N Z C I D V
√ √ - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Immediate	LDY # Oper	AØ	2	2
Zero Page	LDY Oper	A4	2	3
Zero Page, X	LDY Oper, X	B4	2	4
Absolute	LDY Oper	AC	3	4
Absolute, X	LDY Oper, X	BC	3	4*

*** Add 1 if page boundary is crossed.**

LSR **LSR Shift Right One Bit (memory or accumulator)** **LSR**

Operation: Ø → 7 6 5 4 3 2 1 0 → C

N	Z	C	I	D	V
Ø	√	√	-	-	-

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Accumulator	LSR A	4A	1	2
Zero Page	LSR Oper	46	2	5
Zero Page, X	LSR Oper, X	56	2	6
Absolute	LSR Oper	4E	3	6
Absolute, X	LSR Oper, X	5E	3	7

NOP **NOP No Operation** **NOP**

Operation: No Operation (2 cycles)

N	Z	C	I	D	V
-	-	-	-	-	-

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	NOP	EA	1	2

ORA **ORA "OR" Memory with Accumulator** **ORA**

Operation: A V M → A

N	Z	C	I	D	V
√	√	-	-	-	-

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Immediate	ORA # Oper	Ø9	2	2
Zero Page	ORA Oper	Ø5	2	3
Zero Page, X	ORA Oper, X	15	2	4
Absolute	ORA Oper	ØD	3	4
Absolute, X	ORA Oper, X	1D	3	4*
Absolute, Y	ORA Oper, Y	19	3	4*
(Indirect, X)	ORA (Oper, X)	Ø1	2	6
(Indirect), Y	ORA (Oper),Y	11	2	5*

PHA **PHA Push Accumulator on Stack** **PHA**

Operation: A ↓

N	Z	C	I	D	V
-	-	-	-	-	-

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	PHA	48	1	3

PHP **PHP Push Processor Status on Stack** **PHP**

Operation: P ↓

N	Z	C	I	D	V
-	-	-	-	-	-

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	PHP	Ø8	1	3

PLA **PLA Pull Accumulator from Stack** **PLA**

Operation: A ↑

N	Z	C	I	D	V
√	√	-	-	-	-

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	PLA	68	1	4

PLP **PLP Pull Processor Status from Stack** **PLP**

Operation: P ↑

N	Z	C	I	D	V
From Stack					

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	PLP	28	1	4

ROL **ROL Rotate one Bit Left (memory or accumulator)** **ROL**

M or A

Operation: C ← 7 6 5 4 3 2 1 0 ← C

N	Z	C	I	D	V
√	√	√	-	-	-

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Accumulator	ROL A	2A	1	2
Zero Page	ROL Oper	26	2	5
Zero Page, X	ROL Oper, X	36	2	6
Absolute	ROL Oper	2E	3	6
Absolute, X	ROL Oper, X	3E	3	7

ROR **ROR Rotate one Bit Right (memory or accumulator)** **ROR**

M or A

Operation: C → 7 6 5 4 3 2 1 0 → C

N	Z	C	I	D	V
√	√	√	-	-	-

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Accumulator	ROR A	6A	1	2
Zero Page	ROR Oper	66	2	5
Zero Page, X	ROR Oper, X	76	2	6
Absolute	ROR Oper	6E	3	6
Absolute, X	ROR Oper, X	7E	3	7

RTI **RTI Return from Interrupt** **RTI**

Operation: P ↑ PC ↑

N	Z	C	I	D	V
From Stack					

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	RTI	4Ø	1	6

RTS **RTS Return from Sub-routine** **RTS**

Operation: PC ↑, PC + 1 → PC

N	Z	C	I	D	V
-	-	-	-	-	-

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	RTS	6Ø	1	6

SBC **SBC Subtract Memory from Accumulator with Borrow** **SBC**

Operation: A - M - C → A
Note: C = Borrow

N	Z	C	I	D	V
√	√	√	-	-	-

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Immediate	SBC # Oper	E9	2	2
Zero Page	SBC Oper	E5	2	3
Zero Page, X	SBC Oper,X	F5	2	4
Absolute	SBC Oper	ED	3	4
Absolute, X	SBC Oper, X	FD	3	4*
Absolute, Y	SBC Oper, Y	F9	3	4*
(Indirect, X)	SBC (Oper, X)	E1	2	6
(Indirect), Y	SBC (Oper),Y	F1	2	5*

*** Add 1 if page boundary is crossed.**

SEC **SEC Set Carry Flag** **SEC**

Operation: 1 → C

N	Z	C	I	D	V
-	-	1	-	-	-

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	SEC	38	1	2

SED **SED Set Decimal Mode** **SED**

Operation: 1 → D

N	Z	C	I	D	V
-	-	-	-	1	-

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	SED	F8	1	2

SEI **SEI Set Interrupt Disable Status** **SEI**

Operation: 1 → I

N	Z	C	I	D	V
-	-	-	1	-	-

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	SEI	78	1	2

STA **STA Store Accumulator in Memory** **STA**

Operation: A → M

N	Z	C	I	D	V
-	-	-	-	-	-

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Zero Page	STA Oper	85	2	3
Zero Page, X	STA Oper, X	95	2	4
Absolute	STA Oper	8D	3	4
Absolute, X	STA Oper, X	9D	3	5
Absolute, Y	STA Oper, Y	99	3	5
(Indirect, X)	STA (Oper, X)	81	2	6
(Indirect), Y	STA (Oper),Y	91	2	6

STX **STX Store Index X in Memory** **STX**

Operation: X → M

N Z C I D V
- - - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Zero Page	STX Oper	86	2	3
Zero Page, Y	STX Oper, Y	96	2	4
Absolute	STX Oper	8E	3	4

STY **STY Store Index Y in Memory** **STY**

Operation: Y → M

N Z C I D V
- - - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Zero Page	STY Oper	84	2	3
Zero Page, X	STY Oper, X	94	2	4
Absolute	STY Oper	8C	3	4

TAX **TAX Transfer Accumulator to Index X** **TAX**

Operation: A → X

N Z C I D V
√ √ - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	TAX	AA	1	2

TAY **TAY Transfer Accumulator to Index Y** **TAY**

Operation: A → Y

N Z C I D V
√ √ - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	TAY	A8	1	2

TSX **TSX Transfer Stack Pointer to Index X** **TSX**

Operation: S → X

N Z C I D V
√ √ - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	TSX	BA	1	2

TXA **TXA Transfer Index X to Accumulator** **TXA**

Operation: X → A

N Z C I D V
√ √ - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	TXA	8A	1	2

TXS **TXS Transfer Index X to Stack Pointer** **TXS**

Operation: X → S

N Z C I D V
- - - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	TXS	9A	1	2

TYA **TYA Transfer Index Y to Accumulator** **TYA**

Operation: Y → A

N Z C I D V
√ √ - - - -

Addressing Mode	Assembly Language Form	Op Code	No of Bytes	No of Cycles
Implied	TYA	98	1	2

	Accumulator	Immediate	Zero Page	Zero Page,X	Zero Page,Y	Absolute	Absolute,X	Absolute,Y	Implied	Relative	(Indirect,X)	(Indirect,Y)	Absolute Indirect
ADC		69	65	75		6D	7D	79			61	71	
AND		29	25	35		2D	3D	39			21	31	
ASL	0A		06	16		0E	1E						
BCC										90			
BCS										B0			
BEQ										F0			
BIT			24			2C							
BMI										30			
BNE										D0			
BPL										10			
BRK									00				
BVC										50			
BVS										70			
CLC									18				
CLD									D8				
CLI									58				
CLV									B8				
CMP		C9	C5	D5		CD	DD	D9			C1	D1	
CPX		E0	E4			EC							
CPY		C0	C4			CC							
DEC			C6	D6		CE	DE						
DEX									CA				
DEY									88				
EOR		49	45	55		4D	5D	59			41	51	
INC			E6	F6		EE	FE						
INX									E8				
INY									C8				
JMP						4C							6C

	Accumulator	Immediate	Zero Page	Zero Page,X	Zero Page,Y	Absolute	Absolute,X	Absolute,Y	Implied	Relative	(Indirect,X)	(Indirect,Y)	Absolute Indirect
JSR						20							
LDA		A9	A5	B5		AD	BD	B9			A1	B1	
LDX		A2	A6		B6	AE		BE					
LDY		A0	A4	B4		AC	BC						
LSR	4A		46	56		4E	5E						
NOP									EA				
ORA		09	05	15		0D	1D	19			01	11	
PHA									48				
PHP									08				
PLA									68				
PLP									28				
ROL	2A		26	36		2E	3E						
ROR	6A		66	76		6E	7E						
RTI									40				
RTS									60				
SBC		E9	E5	F5		ED	FD	F9			E1	F1	
SEC									38				
SED									F8				
SEI									78				
STA			85	95		8D	9D	99			81	91	
STX			86		96	8E							
STY			84	94		8C							
TAX									AA				
TAY									A8				
TYA									98				
TSX									BA				
TXA									8A				
TXS									9A				

Numerical Listing

00 - BRK
01 - ORA - (Indirect,X)
02 - Future Expansion
03 - Future Expansion
04 - Future Expansion
05 - ORA - Zero Page
06 - ASL - Zero Page
07 - Future Expansion
08 - PHP
09 - ORA - Immediate
0A - ASL - Accumulator
0B - Future Expansion
0C - Future Expansion
0D - ORA - Absolute
0E - ASL - Absolute
0F - Future Expansion
10 - BPL
11 - ORA - (Indirect),Y
12 - Future Expansion
13 - Future Expansion
14 - Future Expansion
15 - ORA - Zero Page,X
16 - ASL - Zero Page,X
17 - Future Expansion
18 - CLC
19 - ORA Absolute,Y
1A - Future Expansion
1B - Future Expansion
1C - Future Expansion
1D - ORA - Absolute,X
1E - ASL - Absolute,X
1F - Future Expansion
20 - JSR
21 - AND - (Indirect,X)
22 - Future Expansion
23 - Future Expansion
24 - BIT - Zero Page
25 - AND - Zero Page
26 - ROL - Zero Page
27 - Future Expansion
28 - PLP
29 - AND - Immediate
2A - ROL - Accumulator
2B - Future Expansion
2C - BIT - Absolute
2D - AND - Absolute
2E - ROL - Absolute
2F - Future Expansion
30 - BMI
31 - AND - (Indirect,X)
32 - Future Expansion
33 - Future Expansion
34 - Future Expansion
35 - AND - Zero Page,X
36 - ROL - Zero Page,X
37 - Future Expansion
38 - SEC
39 - AND - Absolute,Y
3A - Future Expansion
3B - Future Expansion
3C - Future Expansion
3D - AND - Absolute,X
3E - ROL - Absolute,X
3F - Future Expansion
40 - RTI
41 - EOR - (Indirect,X)
42 - Future Expansion
43 - Future Expansion
44 - Future Expansion
45 - EOR - Zero Page
46 - LSR - Zero Page
47 - Future Expansion
48 - PHA
49 - EOR - Immediate
4A - LSR - Accumulator
4B - Future Expansion
4C - JMP - Absolute
4D - EOR - Absolute
4E - LSR - Absolute
4F - Future Expansion
50 - BVC
51 - EOR - (Indirect),X
52 - Future Expansion
53 - Future Expansion
54 - Future Expansion
55 - EOR - Zero Page,X
56 - LSR - Zero Page,X
57 - Future Expansion
58 - CLI
59 - EOR - Absolute,Y
5A - Future Expansion
5B - Future Expansion
5C - Future Expansion
5D - EOR - Absolute,X
5E - LSR - Absolute,X
5F - Future Expansion

60 - RTS
61 - ADC - (Indirect,X)
62 - Future Expansion
63 - Future Expansion
64 - Future Expansion
65 - ADC - Zero Page
66 - ROR - Zero Page
67 - Future Expansion
68 - PLA
69 - ADC - Immediate
6A - ROR - Accumulator
6B - Future Expansion
6C - JMP - Indirect
6D - ADC - Absolute
6E - ROR - Absolute
6F - Future Expansion
70 - BVS
71 - ADC - (Indirect),Y
72 - Future Expansion
73 - Future Expansion
74 - Future Expansion
75 - ADC - Zero Page,X
76 - ROR - Zero Page,X
77 - Future Expansion
78 - SEI
79 - ADC - Absolute,Y
7A - Future Expansion
7B - Future Expansion
7C - Future Expansion
7D - ADC - Absolute,X
7E - ROR - Absolute,X
7F - Future Expansion
80 - Future Expansion
81 - STA - (Indirect,X)
82 - Future Expansion
83 - Future Expansion
84 - STY - Zero Page
85 - STA - Zero Page
86 - STX - Zero Page
87 - Future Expansion
88 - DEY
89 - Future Expansion
8A - TXA
8B - Future Expansion
8C - STY - Absolute
8D - STA - Absolute
8E - STX - Absolute
8F - Future Expansion
90 - BCC
91 - STA - (Indirect,Y)
92 - Future Expansion
93 - Future Expansion
94 - STY - Zero Page,X
95 - STA - Zero Page,X
96 - STX - Zero Page,Y
97 - Future Expansion
98 - TYA
99 - STA - Absolute,Y
9A - TXS
9B - Future Expansion
9C - Future Expansion
9D - STA - Absolute,X
9E - Future Expansion
9F - Future Expansion
A0 - LDY - Immediate
A1 - LDA - (Indirect,X)
A2 - LDX - Immediate
A3 - Future Expansion
A4 - LDY - Zero Page
A5 - LDA - Zero Page
A6 - LDX - Zero Page
A7 - Future Expansion
A8 - TAY
A9 - LDA - Immediate
AA - TAX
AB - Future Expansion
AC - LDY - Absolute
AD - LDA - Absolute
AE - LDX - Absolute
AF - Future Expansion
B0 - BCS
B1 - LDA - (Indirect),Y
B2 - Future Expansion
B3 - Future Expansion
B4 - LDY - Zero Page,X
B5 - LDA - Zero Page,X
B6 - LDX - Zero Page,Y
B7 - Future Expansion
B8 - CLV
B9 - LDA - Absolute,Y
BA - TSX
BB - Future Expansion
BC - LDY - Absolute,X
BD - LDA - Absolute,X
BE - LDX - Absolute,Y
BF - EF - Future Expansion

C0 - CPY - Immediate
C1 - CMP - (Indirect,X)
C2 - Future Expansion
C3 - Future Expansion
C4 - CPY - Zero Page
C5 - CMP - Zero Page
C6 - DEC - Zero Page
C7 - Future Expansion
C8 - INY
C9 - CMP - Immediate
CA - DEX
CB - Future Expansion
CC - CPY - Absolute
CD - CMP - Absolute
CE - DEC - Absolute
CF - Future Expansion
D0 - BNE
D1 - CMP - (Indirect),Y
D2 - Future Expansion
D3 - Future Expansion
D4 - Future Expansion
D5 - CMP - Zero Page,X
D6 - DEC - Zero Page,X
D7 - Future Expansion
D8 - CLD
D9 - CMP- Absolute,Y
DA - Future Expansion
DB - Future Expansion
DC - Future Expansion
DD - CMP - Absolute,X
DE - DEC - Absolute,X
DF - Future Expansion
E0 - CPX - Immediate
E1 - SBC - (Indirect,X)
E2 - Future Expansion
E3 - Future Expansion
E4 - CPX - Zero Page
E5 - SBC - Zero Page
E6 - INC - Zero Page
E7 - Future Expansion
E8 - INX
E9 - SBC - Immediate
EA - NOP
EB - Future Expansion
EC - CPX - Absolute
ED - SBC - Absolute
EE - INC - Absolute
EF - Future Expansion
F0 - BEQ
F1 - SBC - (Indirect),Y
F2 - Future Expansion
F3 - Future Expansion
F4 - Future Expansion
F5 - SBC - Zero Page,X
F6 - INC - Zero Page,X
F7 - Future Expansion
F8 - SED
F9 - SBC - Absolute,Y
FA - Future Expansion
FB - Future Expansion
FC - Future Expansion
FD - SBC - Absolute,X
FE - INC - Absolute,Y
FF - Future Expansion

Alphabetical List

ADC Add Memory to Accumulator with Carry
AND 'AND' Memory with Accumulator
ASL Shift Left One Bit (Memory to Accumulator)
BCC Branch on Carry Clear
BCS Branch on Carry Set
BEQ Branch on Result Zero
BIT Test Bits in Memory with Accumulator
BMI Branch on Result Minus
BNE Branch on Result not Zero
BPL Branch on Result Plus
BRK Force Break
BVC Branch on Overflow Clear
BVS Branch on Overflow Set
CLC Clear Carry Flag
CLD Clear Decimal Mode
CLI Clear Interrupt Disable Bit
CLV Clear Overflow Flag
CMP Compare Memory and Accumulator
CPX Compare Memory and Index X
CPY Compare Memory and Index Y
DEC Decrement Memory by One
DEX Decrement Index X by One
DEY Decrement Index Y by One
EOR 'Exclusive Or' Memory with Accumulator
INC Increment Memory by One
INX Increment Index X by One
INY Increment Index Y by One
JMP Jump to New Location
JSR Jump to New Location Saving Return Address
LDA Load Accumulator with Memory
LDX Load Index X with Memory
LDY Load Index Y with Memory
LSR Shift Right One Bit (Memory or Accumulator)
NOP No Operation
ORA 'OR' Memory with Accumulator
PHA Push Accumulator to Stack
PHP Push Processor Status to Stack
PLA Pull Accumulator from Stack
PLP Pull Processor Status from Stack
ROL Rotate One Bit Left (Memory or Accumulator)
ROR Rotate One Bit Right (Memory or Accumulator)
RTI Return from Interrupt
RTS Return from Subroutine
SBC Subtract Memory from Accumulator with Borrow
SEC Set Carry Flag
SED Set Decimal Mode
SEI Set Interrupt Disable Status

STA Store Accumulator in Memory
STX Store Index X in Memory
STY Store Index Y in Memory
TAX Transfer Accumulator to Index X
TAY Transfer Accumulator to Index Y
TSX Transfer Stack Pointer to Index X
TXA Transfer Index X to Accumulator
TXS Transfer Index X to Stack Pointer
TYA Transfer Index X to Accumulator

Instruction Addressing Modes and Related Execution Times (in clock cycles)

	Accumulator	Immediate	Zero Page	Zero Page,X	Zero Page,Y	Absolute	Absolute,X	Absolute,Y	Implied	Relative	(Indirect,X)	(Indirect,Y)	Absolute Indirect
ADC		2	3	4		4	4*	4*			6	5*	
AND		2	3	4		4	4*	4*		6	6	5*	
ASL	2		5	6		6	7						
BCC										2**			
BCS										2**			
BEQ										2**			
BIT			3			4							
BMI										2**			
BNE										2**			
BPL										2**			
BRK													
BVC										2**			
BVS										2**			
CLC									2				
CLD									2				
CLI									2				
CLV									2				
CMP		2	3	4		4	4*	4*			6	5*	
CPX		2	3			4							
CPY		2	3			4							
DEC			5	6		6	7						
DEX									2				
DEY									2				
EOR		2	3	4		4	4*	4*			6	5*	
INC			5	6		6	7						
INX									2				
INY									2				
JMP						3							5

Instruction Addressing Modes and Related Execution Times (in clock cycles)

	Accumulator	Immediate	Zero Page	Zero Page,X	Zero Page,Y	Absolute	Absolute,X	Absolute,Y	Implied	Relative	(Indirect,X)	(Indirect,Y)	Absolute Indirect
JSR						6							
LDA		2	3	4		4	4*	4*			6	5*	
LDX		2	3		4	4		4*					
LDY		2	3	4		4	4*						
LSR	2		5	6		6	7						
NOP									2				
ORA		2	3	4		4	4*	4*			6	5*	
PHA									3				
PHP									3				
PLA									4				
PLP									4				
ROL	2		5	6		6	7						
ROR	2		5	6		6	7						
RTI									6				
RTS									6				
SBC		2	3	4		4	4*	4*			6	5*	
SEC									2				
SED									2				
SEI									2				
STA			3	4		4	5	6			6	6	
STX			3		4	4							
STY			3	4		4							
TAX									2				
TAY									2				
TSX									2				
TXA									2				
TXS									2				
TYA									2				

*Add one cycle if indexing across page boundary.
**Add one cycle if branch is taken. Add one additional if branching operation crosses page boundary.

Appendix B Conversion Table Etc

Hexadecimal Conversion Table

*	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

*Hexadecimal values

Forward Relative Branch

*	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

*Forward Relative Branch Values

Backward Relative Branch

*	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113
9	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97
A	96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81
B	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65
C	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49
D	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
E	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
F	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

*Backward Relative Branch Value

EMMA Program Sheet No:

Programmer: **Program Title:**

Hexadecimal				Symbolic			Comments
Addr	1	2	3	Label	MNEM	Operand	

L.J. Technical Systems

L.J. Technical Systems Ltd.
Francis Way
Bowthorpe Industrial Estate
Norwich, NR5 9JA. England.
Telephone: (0603) 748001
Telex: 975504
Fax: (0603) 746 340

L.J. Technical Systems Inc.
19 Power Drive
Hauppauge
N.Y. 11788. USA.
Telephone: 1800 237 348
In N.Y. 516 234 2100
Fax: 516 234 2656

Distributor

L.J. Technical Systems

L.J. Technical Systems Ltd.
Francis Way
Bowthorpe Industrial Estate
Norwich, NR5 9JA. England.
Telephone: (0603) 748001
Telex: 975504
Fax: (0603) 746 340

L.J. Technical Systems Inc.
19 Power Drive
Hauppauge
N.Y. 11788. USA.
Telephone: 1800 237 348
In N.Y. 516 234 2100
Fax: 516 234 2656

Designed, Typeset and Produced by LJ Technical Systems Publicity Department © 1987.

Distributor

L.J. Technical Systems

L.J. Technical Systems Ltd.
Francis Way
Bowthorpe Industrial Estate
Norwich, NR5 9JA. England.
Telephone: (0603) 748001
Telex: 975504
Fax: (0603) 746 340

L.J. Technical Systems Inc.
19 Power Drive
Hauppauge
N.Y. 11788. USA.
Telephone: 1800 237 348
In N.Y. 516 234 2100
Fax: 516 234 2656

Designed, Typeset and Produced by LJ Technical Systems Publicity Department © 1987.

Distributor

L.J. Technical Systems

L.J. Technical Systems Ltd.
Francis Way
Bowthorpe Industrial Estate
Norwich, NR5 9JA. England.
Telephone: (0603) 748001
Telex: 975504
Fax: (0603) 746 340

L.J. Technical Systems Inc.
19 Power Drive
Hauppauge
N.Y. 11788. USA.
Telephone: 1800 237 348
In N.Y. 516 234 2100
Fax: 516 234 2656

Designed, Typeset and Produced by LJ Technical Systems Publicity Department © 1987.

Distributor