Introducing Microprocessors Part 3

The process of fetching, decoding and executing the sequence of instructions which constitutes a program.

By Mike Tooley

Learning Objectives

The general learning objectives for Part 3 are that readers should be able to:

(a) Understand and use a subset of the instruction set of any common 8-bit microprocessor.

(b) Describe, using appropriate diagrams, the microprocessor instruction fetch/execute cycle.

(c) Understand the facilities provided by a monitor program.

The specific objectives for this part are as follows:

2.2 Instruction Sets

2.2.1 Explain what is meant by the terms instruction and instruction set.

2.2.2 Explain the form in which instructions are stored and presented to the microprocessor for execution.

2.2.3 Categorize instructions in the following groups:

data transfer
arithmetic and logical
test, branch and control

2.2.4 Examine the following modes of addressing:

implied
immediate
absolute.

2.2.5 Examine a subset of the instruction set of any common 8-bit microprocessor and identify the types of instruction for data movement (transfer), control, and arithmetic.

2.3 Fetch-Execute Cycle

2.3.1 Explain each stage of the fetch-execute cycle.

2.3.2 Explain the function of the Program Counter, Instruction Register and Instruction Decoder during the fetch-execute cycle.

2.3.3 Draw a timing diagram showing the state of the read, write (or read/write), and bus lines at each stage of the fetch-execute cycle for a representative 8-bit microprocessor.

2.3.4 Draw a timing diagram to show the movement of data during each stage of the fetch-execute cycle.

2.4 Monitor Programs

2.4.1 Use a monitor program.

Instruction and Instruction Sets

The individual commands contained within a microprocessor program are called instructions. Clearly, if a microprocessor is to be capable of performing a variety of operations, a range of different instructions must be available. Some of these will be concerned with moving data from place to place and are aptly known as "data-transfer" instructions. Others are used to perform "arithmetic and logic" functions. A third type of instruction is needed to control the overall flow of the program. Such instructions form part of the "test, branch and control" group.

The microprocessor keeps track of its progress through a series of instructions by regularly updating its Instruction Pointer (or Program Counter). This sixteen bit register effectively points to the address of the next instruction to be fetched in the sequence of execution.

A simple IMP program (expressed in hexadecimal format) might take the form:

```
3E
01  First instruction (two bytes)
06
02  Second instruction (two bytes)
02
80  Third instruction (one byte)
```

The five byte program contains three instructions. The first two take up two bytes each whilst the last instruction only requires a single byte. The hexadecimal representation is compact but not very explicit and readers might be forgiven for wondering what the program actually does. Furthermore, writing anything other than the shortest of programs in hexadecimal format is clearly going to be a rather tedious process.

In practice we make use of a mnemonic shorthand for writing our instructions rather than resorting to hexadecimal code. However, even programming in hexadecimal is one step removed from the binary codes that the microprocessor actually requires (readers may be unaware that the first generation of computer programmers actually wrote their code

in binary!).

As far as the microprocessor is concerned, each instruction comprises an individual binary code (the operation code) which may be followed by one of more further bytes (which constitute and operand). The operand qualifies the instruction in some way and typically may be used to form an address at which data is to be stored or from which data is to be fetched. Clearly, if we are dealing with an operand which it used to denote a 16-bit address, it will require two bytes. IMP knows how many bytes to take as an operand since it is implicit in the operation code which it will have previously decoded.

Assembly language

We have already stated that IMP responds to instructions presented in binary form and that a form of shorthand is used to simplify the task of writing a program. This shorthand is known as "assembly language" and it provides us with a means of expressing our programs in terms of a set of mnemonics.

Assembly language is a low-level language which is (relatively) easy for humans to learn and remember and which can quite easily be translated into the binary code required by a microprocessor. The function of translating mnemonic assembly code into binary code is performed by a utility program known as an "assembler". Some assemblers produce intermediate programs in hexadecimal format which are then translated into binary code for final loading into program memory.

Unfortunately, each microprocessor family has its own dialect of assembly language. This makes it difficult (if not impossible) to transfer programs written in assembly language from one microprocessor to another. High level languages, such as BASIC or PASCAL, are much more "portable" since, with a few changes, they can usually be modified to run on a wide variety of machines.

Happily, IMP's assembly language is reasonably conventional. The following instructions (and their hexadecimal equivalents) constitute a small subset of IMP's instruction set. For convenience we have divided these instructions into the three major groups associated with "data transfer", "arithmetic and logic", and "test, branch, and control"

Notes

(a) Mnemonics are used as follows:

LD = LoaD
ADD = ADD
SUB = SUBtract
INC = INCrement
DEC = DECrement
JP = JumP
Z = Zero
NZ = Non-Zero

(b) n and xx represents an immediate data byte (values ranging from 00H to FFH)

(c) nn represents a two byte address (values ranging from 0000H to FFFFH)

(d) ll represents the low byte of an address (values ranging from 00H to FFH)

Readers should note that the general format used for IMP's data transfer instructions involves a destination followed by a source and that these are separated by a comma. As an example, the instruction LD A, B specifies A as the destination and B as the source. It is also important to note that the load instructions do NOT involve the destruction of the source byte; data is effectively copied from source to destination where it replaces whatever was there before the instruc-

Function	Instruction	
	Mnemonic form	Hexadecimal form
Data transfer		
Immediate data to accumulator	LD A,n	3E xx
Immediate data to B register	LD B,n	06 xx
Memory to accumulator	LD A,(nn)	3A ll hh
Accumulator to memory	LD (nn),A	32 ll hh
Accumulator to register B	LD B,A	47
Register B to accumulator	LD A,B	78
Immediate data to HL register pair	LD HL,nn	21
Memory (pointed to by HL register) to accumulator	LD A,(HL)	7E
Accumulator to memory (pointed by HL register)	LD (HL),A	77
Arithmetic and logic		
Add register A to register B	ADD B	80
Subract register B from register A	SUB B	90
Increment register A	INC A	3C
Increment register B	INC B	04
Increment register HL	INC HL	23
Decrement register A	DEC A	3D
Decrement register B	DEC B	05
Decrement register HL	DEC HL	2B
Test, branch and control		
Jump unconditionally to specified IP address	JP nn	C3 ll hh
Jump to specified IP address if zero flag is set	JP Z,nn	CA ll hh
Jump to specified IP address if zero flag is reset	JP NZ,nn	C2 ll hh

tion was executed.

Finally, the meaning of the brackets shown in instructions such as LD A, (HL) are taken to mean "address pointed to by" or " memory location given by". Thus LD, A, (HL) means "load the accumulator with the data found at the address pointed to by the HL register pair". This may sound a little wordy but, in order to avoid confusion, it is important to be quite precise.

Addressing modes

The different ways of locating the data to be used by a microprocessor instruction are referred to as "addressing modes". Three commonly used addressing modes are known as "implied", "immediate", and "absolute".

Addresing mode	Data located . . .	Example
Implied	. . at an address pointed to by other CPU registers	LD A,(HL)
Absolute	. . at an address specified in the instruction	LD A,(nn)
Immediate	. . in the instruction itself	LD A,n

In the "implied" addressing mode another register pair is used to hold the address of the location being accessed. In IMP's case, the instruction LD A, (HL) is an example of this mode. In the "immediate" mode of addressing the data to be used is contained within the instruction itself (i.e the data in question immediately follows the operation code). The instruction LD A, n is an example drawn from IMP's set. In the "absolute" mode of addressing, the address at which the data is located forms part of the instruction. This mode is exemplified by IMP's LD A , (nn) instruction.

A number of other (more complex) addressing modes exist. These, however, are not really appropriate-to an introductory level module and will be left for readers to explore in the event that they continue with studies at a higher level. For the moment, it is merely necessary for readers to be able to recognize and distinguish between the three modes previously mentioned. The following table summarizes these modes of addressing and includes examples from IMP's instruction set:

Problem 3.1

(a) IMP - encounters the hexadecimal values 3E and 00 which appear as successive bytes in a program instruction. What action do they produce?

(b) IMP is performing the assembly language instruction SUB B. If the A register contains the 14H before the instruction was performed and 0AH after the instruction is executed, determine the contents of register B.

(c) A single-byte IMP instruction expressed in binary (MSB first) takes the form 00101011. What action does the instruction perform?

(d) It is necessary to load the HL register pair with 3C02H. What IMP assembly language instruction is required?

(e) What hexadecimal code is used to represent the instruction in (d)?

(f) What addressing mode is used in-the instruction LD A, FFH?

(g) To which group or class of instructions does the instruction LD A, (HL) belong?

The Fetch-Execute Cycle

The operation of a microprocessor is based upon a continuous sequence of events each of which is known as a "fetch-execute" cycle. The fetch-execute cycle involves the following stages:

(a) Fetching the instruction from memory and placing it in the microprocessor's Instruction Register.

(b) Decoding the instruction (using the Instruction Decoder) and determining what subsequent action is required.

(c) If necessary, fetching more data.

(d) Executing the instruction.

This process is illustrated by the flowchart shown in Fig. 3.1.

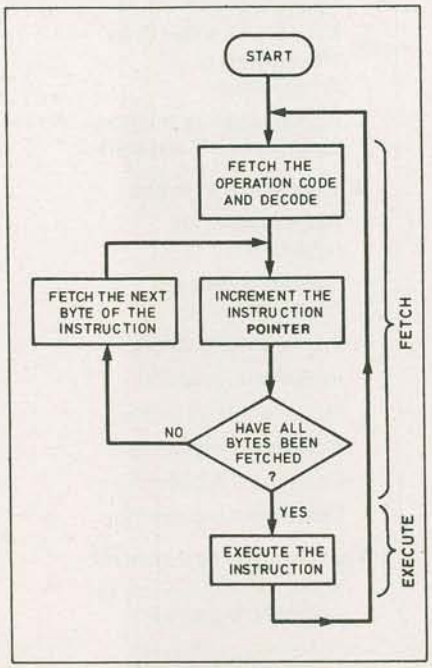

Fig. 3.1. Flowchart for the fetch-execute cycle.

Timing diagrams

Timing diagrams show the relationship between control signals and the data and addresses which appears on the microprocessor buses. Fig 3.2 shows a typical timing diagram which illustrates the sequences of events when IMP performs the instruction LD A, 3F. This fetch-execute sequence occupies just two complete machine cycles. During the first machine cycle, IMP fetches the operation code and decodes it. On the second machine cycle, IMP fetches the data byte (3FH) and copies it into the accumulator.

Read and write operations

When performing memory read or write operations IMP performs different tasks on the first and second half-cycle of the clock. During the first half cycle of the clock signal (i.e. when the clock line is high) IMP Places a valid memory address on the address bus and selects either a read or write operation by taking the R/W line high or low respectively. Data exchanges then take place during the second half of the clock cycle (i.e. when the clock line goes low), the direction of data movement (i.e. to or from IMP) being determined by the previously set condition on the R/W line.

A read cycle (Fig. 3.3) can be used to transfer a byte of data from an address in ROM, RAM, or I/O to one of the IMP's internal registers. A write cycle (Fig. 3.3b), on the other hand, is used to transfer a byte of data from one of the IMP's internal registers to either

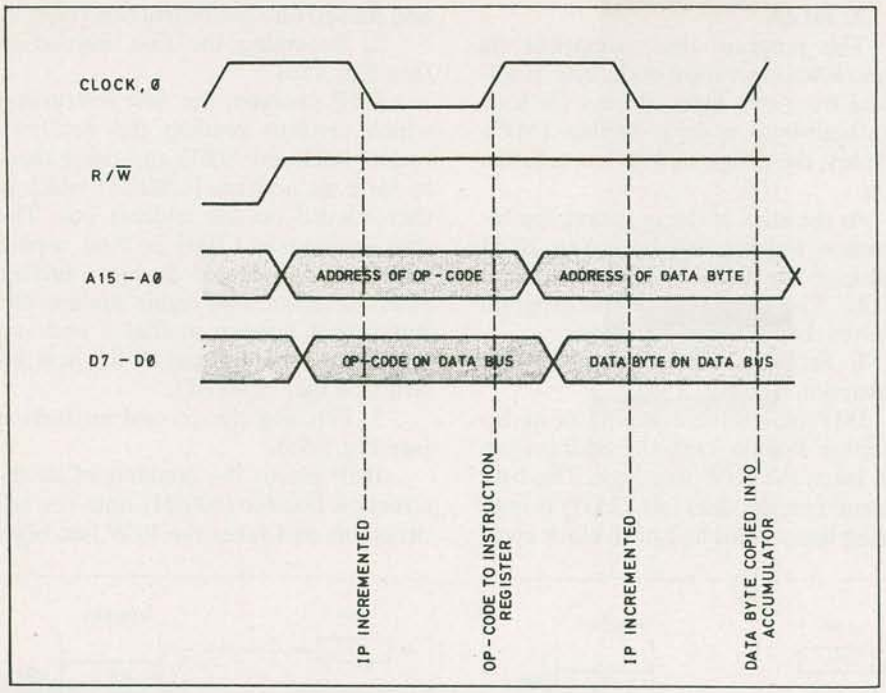

Fig. 3.2. Simplied timing diagram for a fetch-execute cycle.

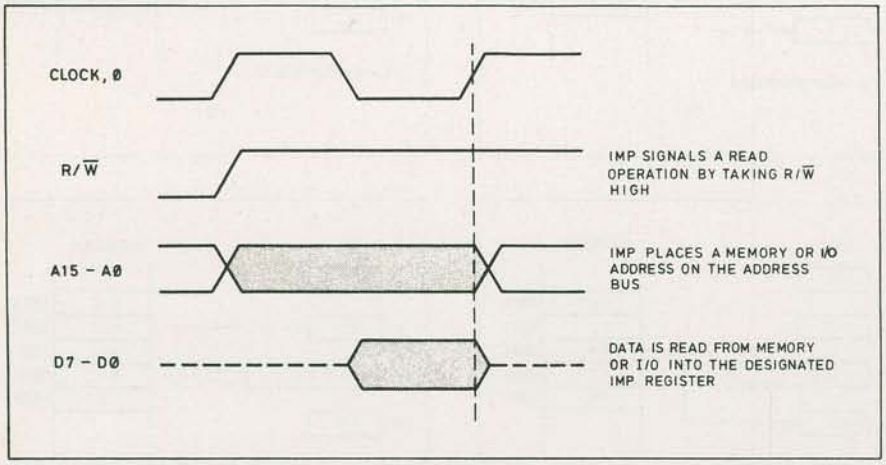

Fig. 3.3a. Simplified timing diagram for a read cycle.

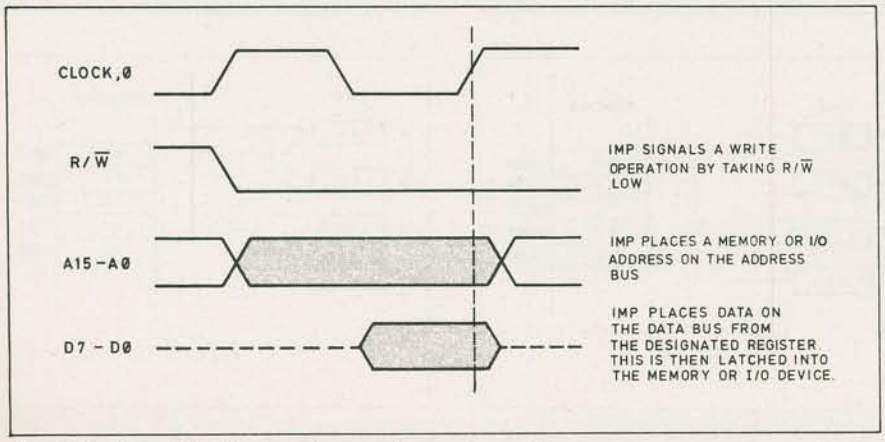

Fig. 3.3b. Simplified timing diagram for a write cycle.

RAM or I/O. Note that, whilst it is possible to undertake a write operation to an address in ROM there is little point in doing so as this would, by definition, have no effect on the contents of the address location in question.

Example 1

Now, let's consider a simple example. Suppose that we wish to add together two bytes of data stored in RAM as part of IMP's program. This task will involve three instructions. The first will load the first operand (in this case a byte of immediate data) into the accumulator (A). The second will load the second byte of data into the B register. Finally, the third instruction will add the contents of the A and B registers and deposit the result back into the accumulator.

We shall assume that the program starts at a hexadecimal address of 1000. Written in assembly language mnemonics, the program looks like this:

```
LD A, 01
LD B, 02
ADD A, B
```

The hexadecimal representation of the program is as follows:

```
3E 01   First instruction
06 02   Second instruction
80      Third instruction
```

Readers will probably have spotted that this program is identical to that which we introduced earlier. Note how each of the two load instructions is followed by the respective data to be loaded. Within IMP's memory, the program will thus take the form:

At the start of the program, the Instruction Pointer will be set to 1000H whilst, at the end, it will have reached 1005H. Execution of the program involves the following steps:

1. Fetching and decoding the first instruction (see Fig. 3.4a).

IMP places the contents of its Instruction Pointer (1000H) onto the address bus and takes the R/W line high. The byte returned on the data bus (3EH) is read during the second half of the clock cycle and passed into the instruction register.

2. Executing the first instruction (see Fig. 3.4b).

IMP executes the first instruction which involves copying the next byte (i.e. that which follows the operation code, 3EH) into the accumulator, IMP also updates the Instruction Pointer so

that it points to the address of the next instruction byte at 1002H.

3. Fetching and decoding the second instruction (see Fig. 3.4c)

IMP places the contents of its Instruction Pointer (1002H) onto the address bus and takes the R/W line high. The byte returned on the data bus (06H) is read during the second half of the clock cycle and passed into the instruction register.

4. Executing the second instruction (see Fig. 3.4d).

IMP executes the second instruction which involves copying the next byte (i.e. that which follows the operation code, 06H) into the B register. IMP also updates the Instruction Pointer so that it points to the address of the next instruction byte at 1004H.

5. Fetching the third instruction (see Fig. 3.4e)

IMP places the contents of its Instruction Pointer (1004H) onto the address bus and takes the R/W line high. The byte returned on the data bus (80H) is read during the second half of the clock cycle and passed into the instruction register.

6. Executing the third instruction (see Fig. 3.4f).

IMP executes the third instruction which involves passing the contents of the A and B registers into the ALU and adding the two bytes together. The result is then passed back into the accumulator (replacing the byte that was originally present). Also note that the byte present in the B register has remained unchanged. IMP also updates the Instruction Pointer so that it points to the address of the next instruction byte at 1005H.

Example 2

Now, as a further example, suppose that we wish to copy a byte of data from an address in ROM (G04EH) to an address in RAM (2AB0H). This task would obviously involve two instructions; a read operation followed by a write operation. We shall again assume that the program again starts at a hexadecimal address of 100H. The program would be written in assembly language as follows:

LD A, (C04EH)
LD (2AB0), A

The hexadecimal machine code corresponding to these two instructions is given below:

3A 4E C0
32 B0 2A

The program thus comprises six bytes. Each operation code byte is followed by a two byte address (in low-byte/high-byte order). Within IMP's memory, the program will thus take the form:

At the start of the program, the Instruction Pointer will be set to 100H whilst, at the end, it will have reached 1006H. The execution of the program involves the following four steps:

1. Fetching and decoding th first instruction (see Fig. 3.5a).

IMP places the contents of its Instruction Pointer onto the address bus and takes the R/W line high. The byte returned on the data bus (3AH) is read during the second half of th clock cycle and passed into the instruction register.

2. Executing the first instruction (see Fig. 3.5b).

IMP executes the first instruction which involves reading the next two bytes (4EH and C0H) and using them to form an address (C04EH) which is then placed on the address bus. The data present at C04H is then copied into the accumulator during a further read operation. IMP again updates the Instruction Pointer so that it ends up pointing to the address of the next instruction byte at 1003H.

3. Fetching the second instruction (see Fig. 3.5c).

IMP places the contents of its Instruction Pointer (1003H) onto the address bus and takes the R/W line high.

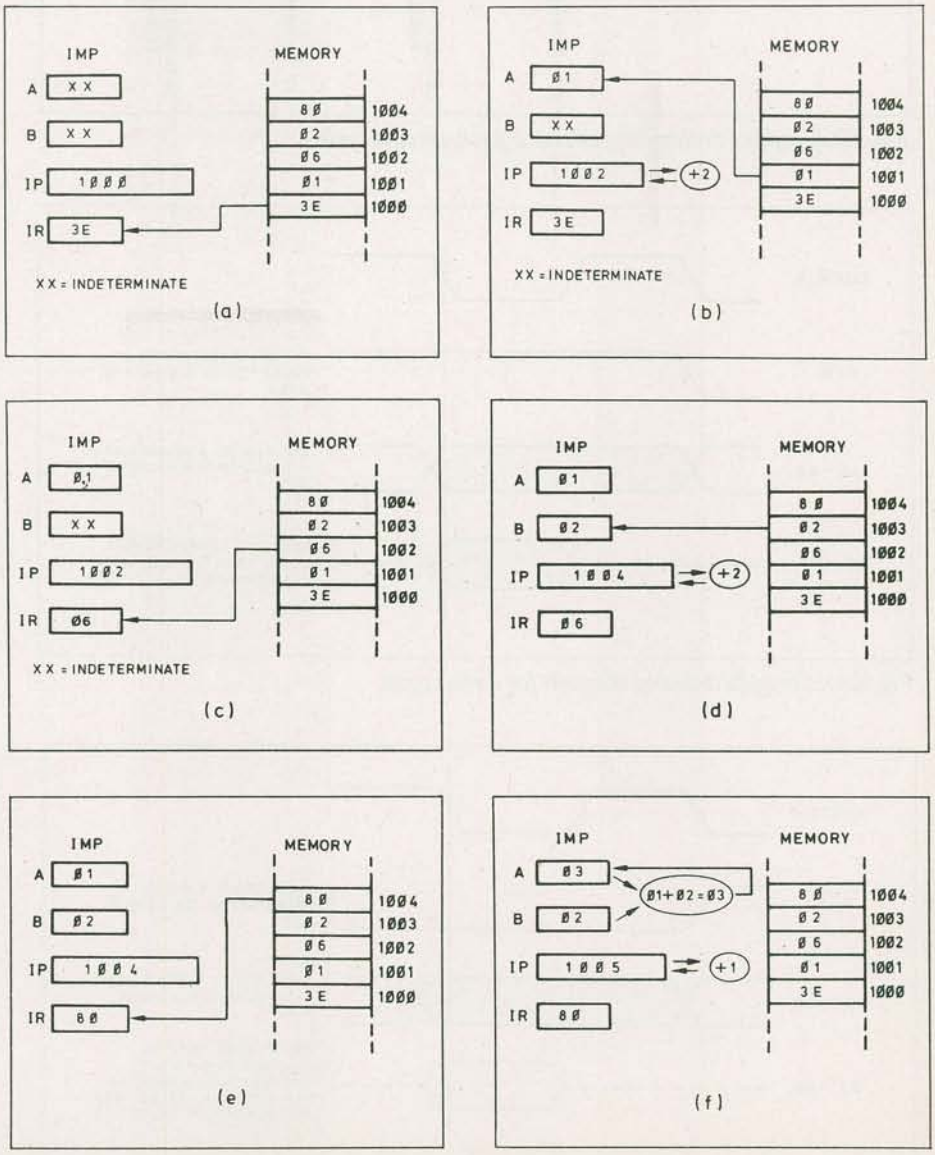

Fig. 3.4. Flow of data between IMP and memory in Example 1.

Address (hex)	Byte (hex)	Function
1000	3A	Operation code for LD A,(nn)
1001	4E	Low byte of address operand
1002	C0	High byte of address operand
1003	32	Operation code for LD (nn),A
1004	B0	Low byte of address operand
1005	2A	High byte of address operand

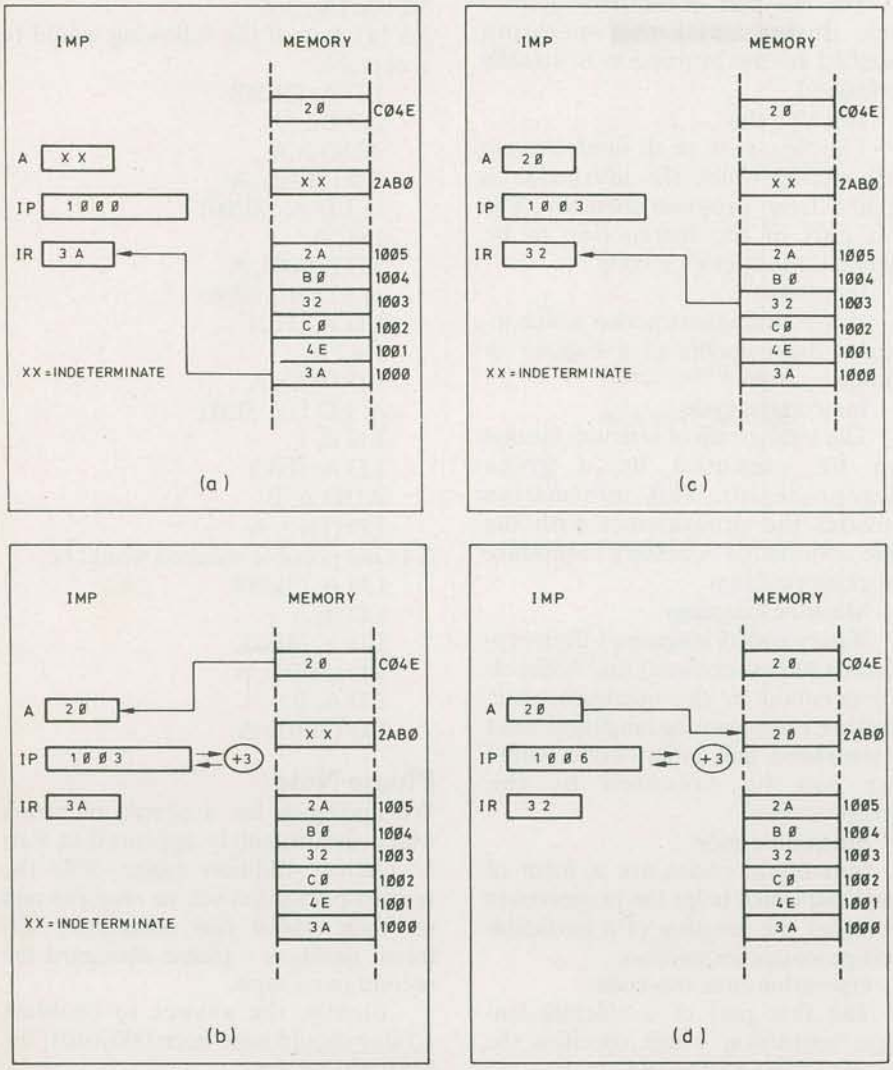

Fig. 3.5. Flow of data between IMP and memory in Example 2.

MACHINE CYCLE	M1	M2	M3	M4
TYPE OF CYCLE	MEMORY READ (OP-CODE FETCH)	MEMORY READ	MEMORY READ	MEMORY WRITE (EXECUTE)
ADDRESS BUS	IP (LOCATION OF OP-CODE BYTE)	IP+1 (LOCATION OF SECOND INSTRUCTION BYTE)	IP+2 (LOCATION OF THIRD INSTRUCTION BYTE)	ADDRESS ASSEMBLED FROM DATA READ DURING M2&M3
DATA BUS				

Fig. 3.6. Table for Problem 3.2.

The byte returned on the data bus (32H) is read during the second half of the clock cycle and passed into the instruction register.

4. Executing the second instruction (see Fig. 3.5d).

IMP executes the second instruction which involves reading the next two bytes (B0H and 2AH) and using them to form an address (2AB0H) which is then placed on the address bus. The data present in the accumulator is then written to address 2AB0H during a final write operation. IMP also updates its Instruction Pointer so that it ends up pointing to the address of the next instruction byte at 1006H. Readers should now be getting a feel for the way in which IMP operates. In particular, the following should be noted:

(a) instructions may comprise one, two, three (or more) bytes.

(b) instructions comprise an operation code which may be followed by a further byte or bytes which constitutes an operand.

(c) instructions may involve further read and/or write operations, not just fetching (i.e. reading) the instruction itself.

Problem 3.2

Fig. 3.6 shows the sequence of operations which occur during the fetch-execute cycle associated with the instruction LD (2E00), A. Given that the accumulator contains 7FH immediately before the instruction is executed, complete the table showing the byte present on the data bus at each stage of the fetch-execute cycle.

Problem 3.3.

Write simple assembly language programs (using only the given subset of IMP's instruction set) which will:

(a) add 1 to the data stored in memory location 3E00H,

(b) exchange the data bytes present at memory locations 3E00H and 3E01H.

Monitor Programs

Monitor programs provide us with a variety of useful facilities which can not only aid our understanding of the operation of a microprocessor but also allow us to enter, test and debug simple programs. A typical monitor program comprises about 2K of code and provides the user with the means to:

(a) display the contents of a given block of memory in hexadecimal and ASCII (see note 1) format

(b) modify or edit hexadecimal bytes in memory

(c) display the contents of the CPU registers

(d) modify the contents of the CPU registers

(e) disassemble a given block of memory into assembly language mnemonics

(f) insert breakpoints (see note 2) into a program

(g) execute a program from a given start address until a breakpoint is encountered

(h) trace the execution of a program with a continuous display of the CPU registers and memory contents as each instruction is executed.

Notes:

1. ASCII stands for "American Standard Code for Information Interchange". The ASCII code is commonly used for representing alphanumeric characters (i.e. letters, numbers and punctuation) within a microprocessor system. Each character is represented by a single byte (i.e. 8 bits). Since the standard ASCII code uses seven bits, the leading (i.e. most significant) bits, the leading (i.e. most significant) bit is either ignored or used to distinguish special graphic characters or tokenized keywords.

2. A breakpoint is a code (usually a single byte) inserted into a program during testing or debugging which, when encountered during the course of a program, suspends execution and returns control to the monitor program. This facility allows the user to examine the state of the system when a certain point is reached in the program.

Glossary for Part Three

Address modes

The various methods of specifying an address as part of an instruction.

Assembly program

A program which translates assembly language statements into the binary code machine code which is directly executable by the microprocessor.

Assembly language

Assembly language is a machine-oriented low-level programming language as distinct from human-oriented high-level languages. An assembly language program is normally written as a series of statements using mnemonics. It is then translated into machine code by an assembler program.

Decrement

Programming instruction which decreases the contents of a register or storage location.

Execute (cycle)

The last part of the fetch-execute cycle during which the operation specified by the instruction is actually performed.

Fetch (cycle)

The first part of th fetch-execute cycle during which the instruction is fetched from program memory. The first part of the instruction to be fetched is the operation code.

Increment

Programming instruction which increases the contents of a register or storage location.

Instruction cycle

The total group of instructions that can be executed by a given microprocessor. This information provides the programmer with the basic information necessary to produce a working program.

Machine language

Binary coded language (often represented in hexadecimal) that is directly understood by the microprocessor. All other programming languages must be translated into binary code before they can be executed by the microprocessor.

Mnemonic code

Mnemonic codes are a form of shorthand which helps the programmer remember the function of a particular microprocessor instruction.

Operation code (op-code)

The first part of a machine-language instruction which specifies the operation to be performed.

Answers to Problems

3.1 (a) load the accumulator with immediate data of 00H

(b) 0AH

(c) DEC HL

(d) LD HL, 3C02

(e) 21 02 3C

(f) immediate

(g) data transfer

3.2 See Fig. 3.7

3.3 (a) Any of the following would be acceptable:

LD A, (3E00)
LD B.1
ADD A,B
LD (3E00), A
or LD A, (3E00)
INC A
LD (3E00), A
or LD HL, 3E00
LD A, (HL)
INC A
LD (HL), A
or LD HL, 3E00
LD B, 1
LD A, (HL)
ADD A, B
LD (HL), A

3.4 One possible solution would be:

LD A, (3E00)
LD B, A
LD A, (3E01)
LD (3E00), A
LD A, B
LD (3E01) A

Please Note

We apologise for a couple of errors which unfortunately appeared in Part 1. Under Addition (page 597) the second paragraph set an example and we then added two completely different numbers - please disregard the second paragraph.

Finally, the answer to Problem 1.14(a) should have been 00000101; the MSB was wrong.

MACHINE CYCLE	M1	M2	M3	M4
TYPE OF CYCLE	MEMORY READ (OP-CODE FETCH)	MEMORY READ	MEMORY READ	MEMORY WRITE (EXECUTE)
ADDRESS BUS	IP (LOCATION OF OP-CODE BYTE)	IP+1 (LOCATION OF SECOND INSTRUCTION BYTE)	IP+2 (LOCATION OF THIRD INSTRUCTION BYTE)	ADDRESS ASSEMBLED FROM DATA READ DURING M2 & M3
DATA BUS	32 (OP-CODE)	ØØ (LOW BYTE OF ADDRESS OPERAND)	2E (HIGH BYTE OF ADDRESS OPERAND)	7F (BYTE FROM ACCUMULATOR)

Fig. 3.7. Answer to Problem 3.2.