Module 8 – Eight bit microprocessor architecture

Overview

Modules 1 to 7 have been concerned with the basic building blocks for a digital computer. We will now use those blocks to construct a computer system.

Microcomputer architecture (concept map)

Objectives

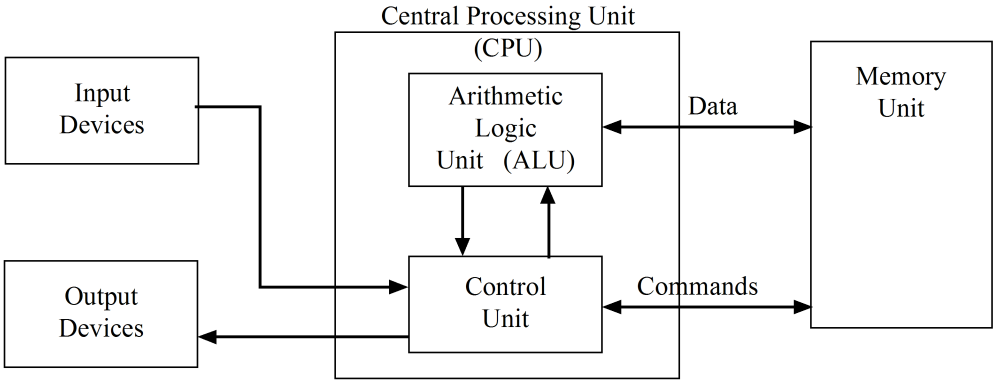
At the completion of his module you will be able to:

- describe the basic modules used in a block diagram of a typical digital computer system
- explain the operation of basic semi-conductor memory systems including RAM, ROM and EPROM
- explain the operation of alternative memory systems including magnetic, optical and bubble memory devices
- state the instructions to perform desired operations in a central processing unit (CPU)
- explain the architecture and basic operation of an elementary CPU.

8.1 Introduction to the microprocessor concept

8.1.1 Introduction

A basic block diagram of a microcomputer system is shown:



Let us broadly consider these blocks.

8.1.2 Input devices

Input devices may be many and are generally familiar to everyone. They include keyboards, keypads, magnetic tapes and disks, analog-to-digital converters, touch screens, light pens, digitisers, character readers, scanners and even voice recognition systems.

8.1.3 Output devices

Output devices may similarly be many. They include printers, visual display units, digital-to-analog converters plasma and LCD screens, plotters, microfiche film and speech systems.

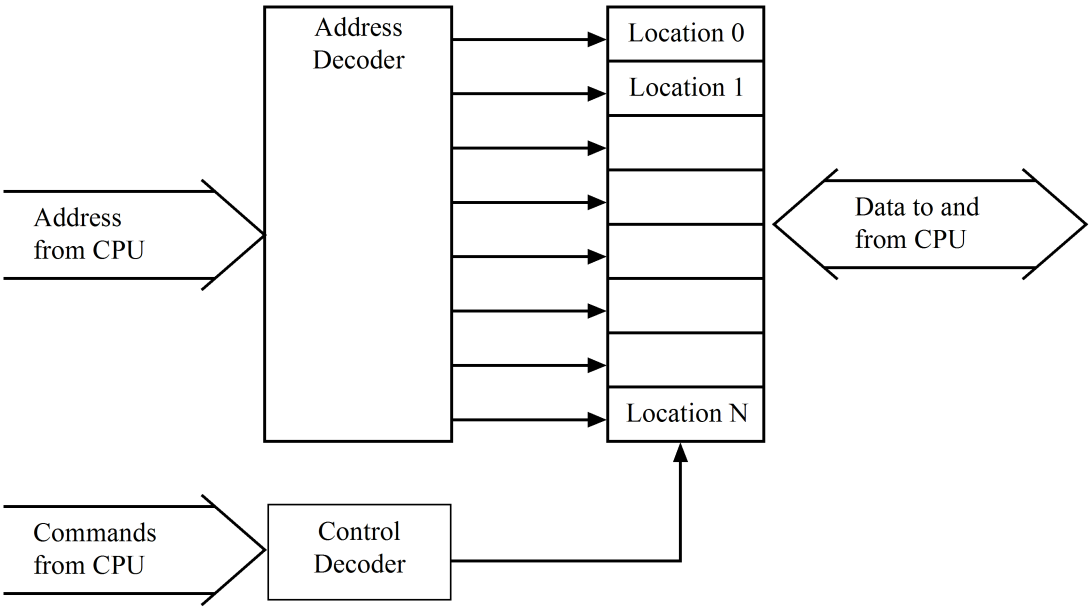
8.1.4 The memory unit

The memory unit is used to store information either temporarily or permanently.

Information stored is usually:

- the operating program
- the data to be used by the program.

A typical memory system is shown in the diagram:



A memory consists of a fixed number of storage areas known as **locations**. Each location has a fixed number of **cells**. Each cell holds one **bit** of information. A common number of cells per location is eight. However this may be 16, 32 or 64 depending upon the application. If we consider one location in an 8 bit memory the usual representation is to draw it as shown:

Location 0 =

1	0	0	1	0	0	1	0

The location has 8 bits and this is referred to as one **byte** of information. An eight bit byte is composed of two 4 bit **nibbles**.

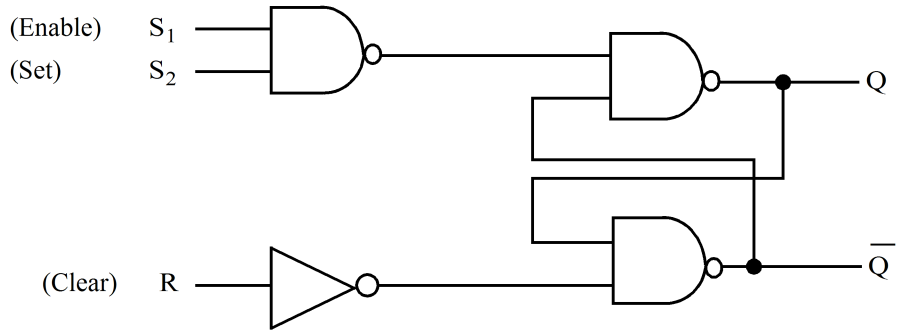
Each location is identified by a unique **address**.

To store information in the memory the central processor first defines the appropriate memory location by specifying a binary number to an address decoder. The processor then sends the required information to the memory block and at the same time, sends a control command to 'write' the information into memory.

8.1.5 Practical memory systems

In many memory systems the basic element used to create a 'cell' is the flip flop. When the flip flop has its Q output set to a logic 1 by the application of a pulse to its set (s) input, it is said to have stored a logic 1 as it remains set after the setting pulse has been removed. Vast arrays of flip flops are used for memories.

This type of memory is called 'Static RAM' (SRAM), where RAM stands for 'random access memory'. As mentioned previously the basic semiconductor element used is the SR bistable and it is used in the configuration as shown below:



To reset this bistable R is made a 1 which stores (puts in memory) a zero, effectively clearing that location.

The input is via a 2 input NAND gate. One input, say S_1, is called the select or **enable** line and when it is made a 1, the FF can be set with a 1 on the S_2 line.

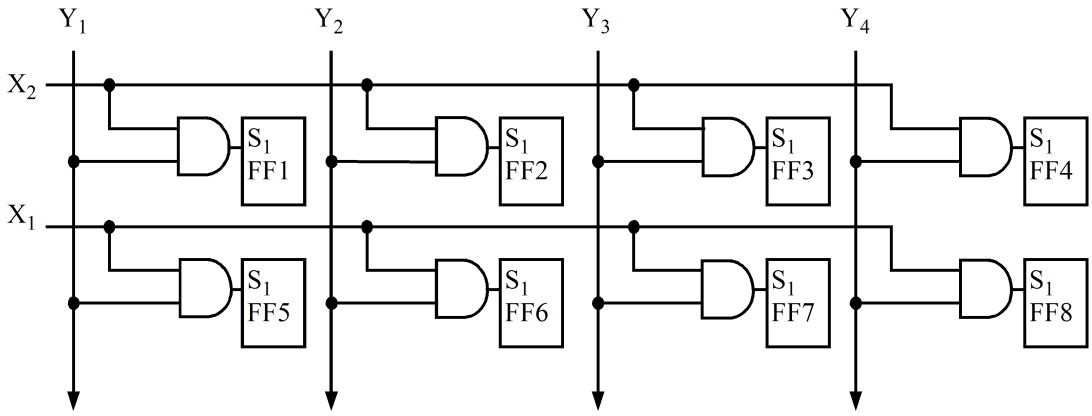
Note with this memory element, the signal at either of the Q or $\overline{Q}$ lines can be measured or sensed as often as desired without affecting or destroying the logic signals.

These semiconductor memories are called 'non-destructive read-out' memories. However if the supply power is removed or lost accidentally, the Q and $\overline{Q}$ signals go to zero and when power is restored, general circuit noise will randomly set Q or $\overline{Q}$ to either a 1 or a 0. We have however, lost the stored information.

This SR element is also called 'volatile memory' for this reason. The problem is sometimes overcome by using a set of batteries permanently connected to the memory element. This is called 'battery-backed' memory.

The SR circuit is the basis for the semi-conductor memory called a Random Access Memory (RAM), (or read-write memory or scratch-pad memory).

A typical RAM uses several bistables (cells) in a matrix format as shown:



Y_1 to Y_4 represent the data inputs while X_1, X_2 etc. represent the **select** lines. The bistable (FF) outputs are not shown.

Selection of an X and Y line allows a logic 1 to be written into an appropriate FF cell.

This RAM would provide storage for 2 words of 4 bits each and is therefore called a ‘2 by 4’ or 8 bit RAM. In many cases it is desirable to have information permanently stored in a memory. Such a fixed memory is called a Read Only Memory or ROM. ROM’s are used frequently for this purpose. Examples are look-up tables such as trigonometric or logarithmetic tables.

Another good example is **code** conversion such as from electric typewriter code to computer ASCII code etc.

A ROM can be pre-programmed by the manufacturer to suit a customer’s needs. Some ROM’s can be programmed by the customer in the field, only once generally. This type of ROM is usually manufactured containing all logic ones.

To program this ROM, the customer selects the bit location where he wants a **zero** and then applies a specified pulse of current. The current flows through a nickel-chromium metallic link which is vapourised by the magnitude of the current. This is similar to ‘blowing a fuse’. A zero is then permanently stored at this location and it cannot ever be changed. This is called a Programmable Read Only Memory (PROM) and is non-volatile.

8.1.6 Non flip flop memories

Another type of ROM is available which is both field programmable and field erasable. It can be therefore re-used in a new configuration if desired. It is called an Erasable Programmable Read Only Memory or EPROM.

This is a Metal-Oxide-Silicon (MOS) transistor circuit with a floating gate input manufactured as an integrated circuit (IC).

To program a logic 1, a voltage is applied to the appropriate gate and a charge is developed on the device. This charge remains when the voltage is removed as there is no discharge path provided.

The unit can be completely erased of all stored information by exposing it to UV light through a quartz window which is part of the IC case. This removes the charge and the unit can be reprogrammed. This process requires a very intense UV light source for at least 20 minutes to erase the unit. An erased EPROM usually has logic 1's in all cells.

A variation to the EPROM is a ROM which can be electronically erased which of course has the advantage of not requiring the UV source. This is called an Electronically Erasable Programmable Read Only Memory or EEPROM or E^2PROM.

8.2 Alternative storage systems

Semiconductor memories such as those described are normally used by computers to store program instructions and data that the CPU is currently using. Many other programs and files of data in a system may also require storage and alternative storage devices are used for this.

These devices typically use magnetic storage principles and include: magnetic **disks** and **tapes**. However, optical storage device such as CDROMs and DVDs are now more popular for transportable mass storage, while PC cards and USB devices are proving useful for quick data transfer between personal computers.

8.2.1 Magnetic disks

Magnetic disks are used in two forms:

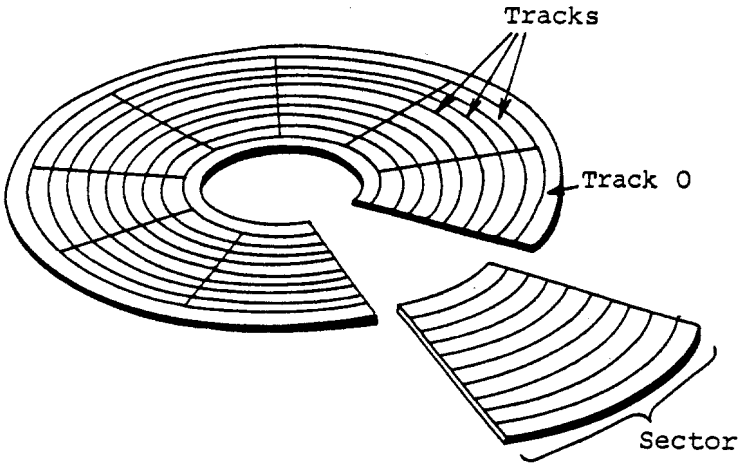
- flexible diskettes (floppy disks) - rarely used today
- hard disks (single disks or multiple disk packs).

- **Flexible diskettes**

Flexible diskettes (floppy disks) are no longer used as a principle auxiliary storage medium for personal computers. Although this type of storage was convenient, reliable, and relatively low in cost, it is rarely used today. However, the principle of operation applies equally to hard drives.

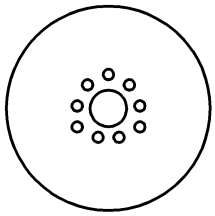
A floppy disk consists of a circular piece of thin mylar plastic (the actual disk), which is coated with an oxide material similar to that used on magnetic tape. On a 3½ inch disk, the

circular piece of plastic is enclosed in a rigid plastic cover and a piece of metal called the shutter covers the reading and recording area. When the 3½ inch disk is inserted into a disk drive, the drive slides the shutter to the side to expose the disk surface. The term 'formatting' is used when preparing a floppy disk for use as a storage medium. Most floppy disks are purchased already formatted. The **formatting** process includes defining the tracks and sectors on its surface as shown. A **track** is a narrow recording band forming a full circuit around the disk. Each track on the disk is divided into sectors. A sector is a section of a track. When data is read from a disk a minimum of one full sector is read. When data is stored on a disk, at least one sector is written. The number of tracks and sectors that are placed on a disk when it is formatted varies based on the capacity of the disk. The 3½ inch disks are mostly formatted with 80 tracks and 18 sectors on each side. When 80 tracks are recorded on a diskette, the tracks are numbered from 0 to 79.

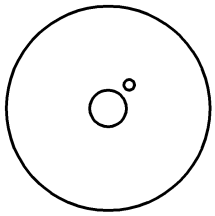


Floppy disk format

Disks are classified as either **hard**-sectored or **soft**-sectored. A hard-sectored disk has a hole in the disk near its centre at the beginning of each sector. These holes provide timing information to the disk drive unit. A soft-sectored disk has a single timing hole only which indicates the start of any track.



Hard-sectored disk



Soft-sectored disk

The storage capacity of these disks depends upon the number of sectors, number of tracks and number of sides used. Older disk drives only allow recording on one side of the disk while more recently both sides are used. Typically a high density disk with 80 tracks on each side, eighteen sectors per track can store 512 bytes per sector. This gives a total capacity of (18× 512 × 80 × 2) = 1 475 560 bytes per disk. The term 'cylinder' is often used with double-sided disks meaning all tracks of the same number. For example track 0 on side 1 and track 0 on side 2 of the disk is called cylinder 0. Floppy disks normally rotate at 300 rpm and the read/write head accesses the disk through a slotted hole in the disk's protective cover. Data is stored using the ASCII code by writing logic ones as magnetic areas on the surface. The

read/write head moves laterally across the disk to access the various tracks. The time required to access data is called the access time. It has 4 main components:

- **Seek time**, the time it takes to position the read/write head over the proper track.
- **Latency**, the time it takes for the sector containing the data to rotate under the read/write head.
- **Settling time**, the time required for the read/write head to be placed in contact with the disk.
- **Data transfer rate**, the time required to transfer the data from the disk to main memory.

The access time for floppy disks is about 150 milliseconds.

- **Hard disks**

Hard disks are constructed of magnetically coated platters permanently mounted-inside the computer in a sealed case and are not accessible like floppy disks. Data is stored in a similar fashion to floppy disks however. The disk unit may have one or more platters attached to a single spindle rotating at high speed, typically 5400 to 7200 rpm. One or more read/write heads floats on a cushion of air just above the disk and do not even touch the disk surface. The storage capacity depends upon the number of platters and is usually in the tens of gigabytes of data. Typical access times are between 5 and 12 msec.

8.2.2 Magnetic Tapes

One of the first storage media used with mainframe computers was tape and it is still used today for mass storage. Magnetic tape consists of a thin band of plastic. The tape is coated on one side with a material that can be magnetised to record the bit patterns that represent data. The most common types of magnetic tape devices are reel-to-reel and cartridge, although reel-to-reel devices have been replaced almost completely by cartridge devices. Reel-to-reel tape is usually ½ inch wide and cartridge tape is ¼ inch wide.

Data is recorded on magnetic tape as small magnetised areas similar to our disks. Reel to reel tapes are formatted into a series of horizontal rows called channels. Both EBCDIC and ASCII codes are typically used giving nine channels horizontally along the tape, 7 for the code, 1 for parity and 1 for timing. Tape density is the number of bytes that can be stored per inch of tape. Common densities are 800, 3200, 6250 bytes per inch. Data is stored serially in blocks on a tape with ‘inter record gaps’ between them.

Typically tapes are 2400 feet long on reels and speeds up to 200 inches per second are used in the transport mechanisms.

Access time varies depending upon the physical distance to travel from one position to another. Large storage capacities are typical as shown by the following example.

A magnetic tape with nine tracks of data has a recording density of 200 bits per inch and is 2400 feet long. The tape speed is 75 in/s.

a. How much data could be stored on the tape?
b. What is the average access time?
c. What is the maximum access time?

d. What is the data rate in bits per second?

Solution

a. Total length = 2400 × 12 = 28 800 in. Per track: 200 bits per in. × 28 800 in. = 5.76×10^6 bits. The tape can store 5.76×10^6 9-bit words if 100 percent utilisation of the tape is assumed.

b. Average access time is the time to reach half the tape length:

$$T = \frac{28\,800}{2 \quad 75} = 193\,\text{s} \quad 3.25\,\text{min}$$

c. Maximum access time is time to reach the total tape:

T = 385 s 6.5 min

d.

$$\text{Data rate} = \frac{5.76 \quad 10^6\ \text{words}}{385\,\text{s}} \quad 15\,000\,\text{words per s}$$

$$15\,000 = 9 \quad 135\,000\,\text{bit per}$$

Cartridge tapes are formatted differently. Data is recorded on a single track along the length of the tape. When the end is reached, the tape reverses direction and data is recorded in another track in the opposite direction. Up to seven tracks may be formatted on to a ¼ inch wide tape. Tape densities range from 6 000bpi to more than 60 000bpi

8.2.3 Optical storage

To store large amounts of data an optical disk system can be used. A laser is used to burn microscopic holes into the surface of a plastic disk. A lower power laser can then be used to detect the presence or absence of these holes by reflection from the surface. The presence or absence of holes represents logic 1 or 0 respectively.

A full-size 12 inch optical disk can store several billion characters of information. Smaller disks, about five inches in diameter, can store over 800 million characters or approximately 600 times the data that can be stored on a 3½ inch floppy disk. The smaller optical disks are called **CDROM**, for compact disk read-only memory. They use the same laser technology that is used for the CDs disks that have become popular for recorded music.

8.3 The Central Processing Unit (CPU)

8.3.1 Introduction

The simplified central processing unit (CPU) contains the control unit and the arithmetic/logic unit. These two components combine using the program and data stored in memory to perform the processing operations.

The **control unit** operates by using the following four operations: fetching, decoding, executing, and storing. **Fetching** means obtaining the next program instruction from memory. **Decoding** is translating the program instruction into the commands that the computer can process. **Executing** refers to the actual processing of the computer commands, and **storing** takes place when the result of the instruction is written to memory.

The central processing unit (CPU) performs and controls all the operations. It also contains the Arithmetic Logic Unit (ALU) which can ADD, SUBTRACT, AND, OR and EXOR. The CPU has a number of temporary storage areas known as **registers**. Registers are used to supply memory addresses, receive data and store data for intermediate operations. One register in the system is the most important and is used for all operations. It is usually known as the **accumulator** (ACC).

The CPU handles binary information of a specific number of bits. 8, 16, 32 and 64 are common and the number of bits is called the **word size**. Generally the larger the word size the more complex, accurate and costly is the computer. When a CPU is constructed, it has designed into it the capacity to perform a fixed set of operations. These operations are activated by a set of instructions provided by the manufacturer.

Typical instructions are: **add**, **subtract**, **load**, **store** and **branch**. These are usually abbreviated to three letter representations called **mnemonics**. For the above instructions, the mnemonics are ADD, SUB, LDA, STA and BRA.

8.3.2 The instruction set

To solve a problem a technique is usually decided upon by the programmer and he defines it by means of a **flowchart**. Each step in the flowchart can be executed by means of an instruction obtained from the total list of instructions provided by the manufacturer. This list is known as an **instruction set**. The completed program is stored in the memory and when started, instructions are fetched in sequence, decoded and executed automatically. This type of operation is known as the **stored program concept**.

Each instruction in the set has a unique binary code describing it. This is known as its **machine code**, **source code** or **operation code** (OPCODE).

A set of codes to perform a set task is known as a **machine language program**.

An example of some typical instructions in a set is now shown. Consider the first name in the instruction set. The operation required by the programmer is ADD, the mnemonic is ADD, the code is 9B hexadecimal and the description for the action is shown in the last column.

Table 8.1: Instruction set

Name	Mnemonic	OPCODE	Description
ADD	ADD	10011011 or $9B_{16}$	Add the contents of the location whose address is given in the 'operand address field' to the accumulator. The result remaining in the accumulator.
SUBTRACT	SUB	10010000 or 90_{16}	Subtract the contents of the location whose address is given in the 'operand address field' from the accumulator. The result remaining in the accumulator.
Load the accumulator	LDA	10010110 or 96_{16}	Load the contents of the location whose address is given in the operand address field into the accumulator.
Store the accumulator	STA	10010111 or 97_{16}	Store the contents of the accumulator into the address given in the 'operand address field'.
Branch	BRA	00110000 or 30_{16}	Branch to the address given in the operand address field.
Branch if minus	BMI	00110001 or 31_{16}	Branch to the address given in the operand address field **if** the status of the previous result was negative. If **not** negative **do not** Branch.
HALT	HLT	00111111 or $3F_{16}$	Stop all operations.

N.B. The **mnemonic** is the accepted abbreviation representing the instruction.

For example, when the computer received the hexadecimal number 9B, it knows that means ADD two numbers together and put the result in the accumulator register. The use of these instructions in a program will be shown later in the module.

8.3.3 Instruction content

To help introduce the architecture of a CPU a simplified conceptual development of the instruction content will be considered. This is based on what information is required in an instruction to allow the CPU to perform the desired operation.

Consider the basic computer operation ‘ADD’.

i.e. A + B = C where A, B and C are the address of the memory locations, the contents of which are to be used in the addition operation.

Based on the understanding that the computer must be told **‘Exactly what to do’**, the following can be deduced:

- The computer must be told to ADD.
- The addition must be performed on two numbers (operands) and the address of each must be specified.
- The address of the result must be included.

and

- Having completed this instruction, the CPU must be told **‘What to do next’**, i.e., the address of the next instruction.

This can be summarised in an instruction form as

OPCODE FIELD (i.e. ‘ADD’)	Address of 1st operand (i.e. A)	Address of 2nd operand (i.e. B)	Address of the result (i.e. C)	Address of next instruction

Given that this instruction must be a coded binary number, i.e., in machine code, the impractical nature of this instruction can be highlighted as follows:

For a very simple computer with only 8 instructions and a memory of only 32 locations the length of each instruction becomes:

- OPCODE requires ‘3 bits’. (8 instructions, one of which is ADD).
- each address must be 5 bits since the memory size is 32 locations.

3 bits	5 bits	5 bits	5 bits	5 bits

This instruction will therefore be 23 bits long. For a memory of realistic size, i.e., 4K, each address would be 12 bits giving an unworkable instruction length of 51 bits.

N.B. 1K refers to 1000 memory locations. In fact, 1K of memory requires a 10 bit address and the actual number of locations is 1024, i.e. $2^{10} = 1024$. For a 12 bit address the number of locations which can be addressed is 2^{12} or 4096. (i.e. 4K; locations 0 to 4095)

8.3.4 Reduction of instruction word length

The information provided in the above instruction is essential and no matter what modifications are made, the same information must be somehow imparted to the CPU. However, it is **too lengthy**. A reduction in the instruction length is accomplished by a compromise between

- introducing extra hardware in the CPU. This reduces flexibility and limits the operation of each instruction.

and

- adding further instructions to compensate. This increases the program length.

The reduction is accomplished as follows:

- Most programs are largely sequential in nature causing instructions to be fetched in sequence from sequential locations in memory. Only occasionally will a BRANCH or JUMP to a different section of program be required, e.g. in a program loop.

  The conclusion is to remove the need for the 'Next Instruction Address' and in its place, put as part of the CPU a hardware binary counter. This counter can be incremented during the execution of each instruction in readiness to provide the next instruction address in sequence.

  This counter is called the **'Program Counter'** or PC. In all digital computers:

  The address of the instruction to be fetched is provided by the 'program counter'

  e.g., To start a program (fetch the first instruction), the address of the memory location, containing the first instruction, must be placed in the PC.

  To alter the sequence of a program, i.e. as in a loop, special instructions must be added to load the PC with a new instruction address.

  Such instructions are:

  BRANCH
  or instructions
  JUMP

- The instruction length can be further reduced by making the address of one of the OPERANDS also the address of the result.

  i.e., this is equivalent to the statement

  A = A + B

This removes the need for the ‘Result address’.

This can be taken further by allocating a small section of memory (e.g. locations 0, 1, 2 and 3) to be reserved as **working registers**. All operations must then be performed between one of these ‘working registers’ and an OPERAND from memory.

i.e., ‘Reg 0’ + A and put result in ‘Reg 0’.

This further saves on instruction length since in place of the ‘Address of the second OPERAND’ there now only needs to be ‘2 bits’ to specify one of 4 working registers.

The instruction becomes:

OPCODE Field	Register Field	Operand Address Field

In general, however, the ‘working registers’ are separated away from the memory and become part of the CPU. This makes it easier and quicker for the CPU since it does not have to continually supply an address in memory for registers 0 to 3. The number of ‘working registers’ in a CPU will vary from one make of CPU to another.

In cases where the number of ‘working registers’ is limited to **one** or **two**, they are commonly referred to as ‘Accumulator’ ALU operations such as ADD, SUB, AND, OR etc., are always performed with one operand as the ‘accumulator’ and the result being left in the accumulator.

i.e., (accumulator) + operand → result in (accumulator).

This has limited the original concept of ‘ADD’ as illustrated in the beginning of this section, however, extra instructions are available to offset this limitation.

LDA	C	–	load the contents of ‘C’ into the accumulator.
STA	D	–	store the accumulator contents into address D.

The operation A + B = C now becomes:

LDA	B	–	put B in the accumulator.
ADD	A	–	(accumulator + A) → accumulator.
STA	C	–	store accumulator into C.

The accumulator becomes an extremely important register which is the central element in a large number of operations. The major uses of the accumulator are:

- All ALU functions are performed with the accumulator initially providing one of the operands and later storing the result of the operation.
- It is the register which is used to transfer data between the CPU and I/O devices, and into and out of memory.

  (**N.B.** I/O is the abbreviation for **input/output**.)

Some CPU architectures will have:

- one accumulator only

or

- several working registers all of which can be used as accumulators as outlined above

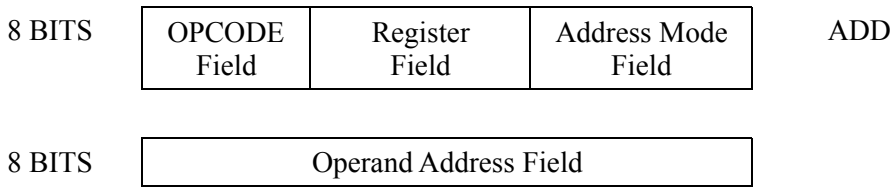
or

- several working registers but only selected ones can act as accumulators. The remainder being used for temporary storage only and not for operations involving the ALU.

The final result of all these actions is that the instruction will have an added component called the 'Address Mode Field' which we will consider later. For a memory size of 64K (16 address lines) the instruction length is still considerable.

The trend is to still limit the size of a memory word, e.g. to 8 bits, and to store the instruction over more than 1 location in memory.

e.g.



8.3.5 Structure of an elementary CPU

Refer to figure 8.1

The function of some of the elements of Figure 8.1 have already been considered. Studying the function of each block in turn, consider:

- **The program counter** – always supplies the address of the next instruction.
- **The status register** – holds the status bits set at the end of execution of the previous instruction.

  i.e. C if a carry was generated.

  V if a 2's complement overflow occurred.

  Z if the result was zero.

  N if the result was negative.

  (This will be explained in a later module.)
- **The Arithmetic Logic Unit** (ALU) – is that circuitry in the CPU which performs the arithmetic and logic functions. These commonly are ADD, ADD with carry, SUB, SUB with BORROW, AND, OR, EXCLUSIVE-OR, INVERT (OR COMPLEMENT).

  The ALU has two data inputs, one from the accumulator, the other is the operand from memory which is taken from the data bus via the data register. The output of the ALU is stored back into the accumulator.

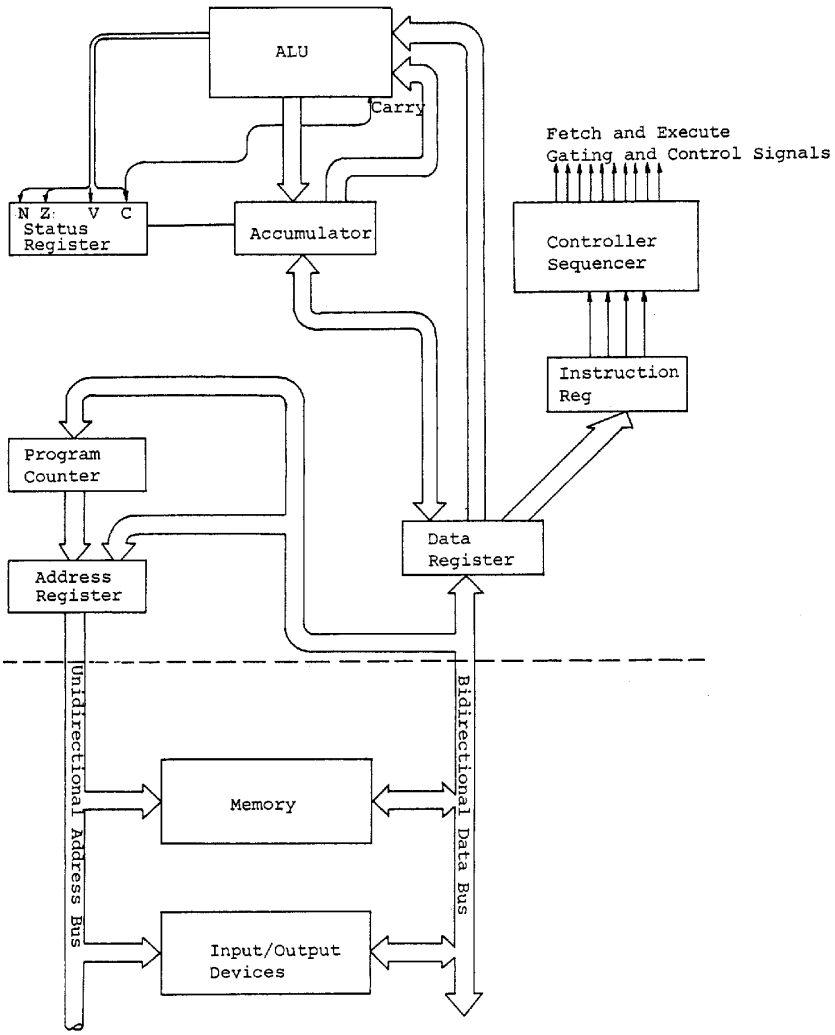
  Which particular ALU function is performed is determined by control lines (not shown on figure 8.1) from the **controller/sequencer**.
- **The accumulator** – This is a general purpose working register whose primary function is for use in conjunction with the ALU.

  it holds one of the operands in an ALU operation and it holds the result following the ALU operation.

In general no matter what a program endeavours to do with data, it must first be put into the accumulator.

Note – It is possible to 'Read' the operand from the 'accumulator' and 'write' the result back into the 'accumulator' in one operation. This is possible because of the special features of the master-slave flip/flop.

Figure 8.1:



- **Data register** – This is a temporary storage register whose function is to act as a buffer between the Data Bus and the CPU. The name given to the Data Register varies depending on the text, e.g. **data buffer** or **data buffer register**. In some texts it isn't included at all.
- **Address register** – A temporary storage register which holds the address of the memory location, or I/O device, being used in the present operation. Its purpose is to act as a buffer and to latch (hold the value) the address supplied by the CPU. This is often referred to as the 'Memory Address Register'.
- **Instruction decoder/controller sequencer** – Often referred to as the 'Control Unit' which decodes the instruction, decides what operation is to be performed and supplies all gating and control signals to execute the instruction. Since each instruction is different, a separate control sequence is initiated by each instruction. It is the function of the control

unit to ‘gate’ the contents of the ‘PC’ into the Address Register, to send out ‘**Memory** Read’ etc., to execute the instruction.

The word size and the address range (i.e. number of address lines) will vary depending on the CPU, e.g. minicomputer or microprocessor. In the following sections it will be assumed that the word size is 8 bits, the address bus is 8 bits, i.e. it can address memory from $0 – 255_{10}$, and each memory location is also 8 bits.

We will now consider an example of using the processor to solve a simple problem. This example is designed to illustrate the ‘concepts’ of machine code programming only. In module 11 we will write a program for this problem in the international format used by all programmers.

8.4 Program example

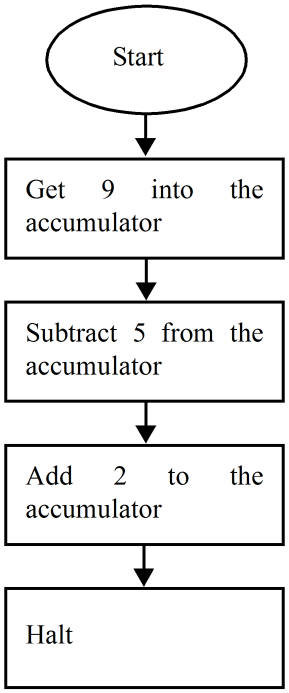
Write a program using the stored program concept to execute the following numerical example:

9 – 5 + 2

It is normal programming practice to draw a flowchart indication the technique used to solve the problem. We also know that all operations are performed in the accumulator register.

NOTE: This example is **not** designed to run on the 68HC11 microcontroller.

8.4.1 Flowcharts



We will also suppose the memory starts at location zero. i.e. 00 Hexadecimal and goes to location FF. Using the table 8.1 instruction set we set out a program in the following columns known as ‘fields’.

8.4.2 Program development

Location	Contents	Mnemonic	Comments
00	09	–	Number 9 stored in location 00.
01	05	–	Number 5 stored in location 01.
02	02	–	Number 2 stored in location 02.
03	96	LDA	Load accumulator from location 00.
04	00		
05	90	SUB	Subtract the contents of location 01 from the accumulator.
06	01		
07	9B	ADD	Add the contents of location 02 to the accumulator.
08	02		
09	3F	HLT	Stop all operations.

To run the program, the hexadecimal contents column would be typed into the computer and the program would be started at location 03. The computer would halt at location 09 and the result or answer to this problem would rest in the accumulator.