COMPUTER ORGANIZATION AND ARCHITECTURE

VTU End Semester Examination Dec 2019/Jan 2020

Solution

Faculty: Prof. Jyoti M R, Prof. Preethi A, Prof. Krishnateja , Prof. Sunil Kumar

MODULE 1

1 a)

Functional Units

A computer consists of five functionally independent main parts: input, memory, arithmetic and logic, output and control units as shown in fig 1.1.

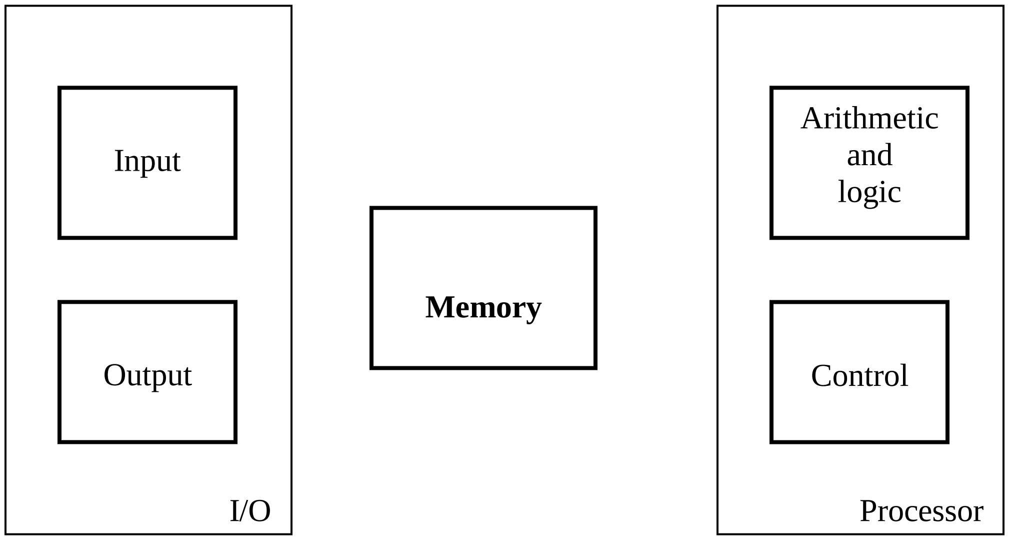


Fig 1.1 Basic functional units of a computer

The input unit accepts coded information from human operators, from electromechanical devices such as keyboards or from other computers over digital communication line. The information received is either stored in the computer memory for later reference or immediately used by the arithmetic and logic circuitry to perform the desired operations. The processing steps are determined by the program stored in the memory. Finally the results are shown on the output unit. All of these actions are co-ordinated by the control unit. We refer to the arithmetic and logic circuits in conjunction to the control circuits as the processor and input and output units are referred to as input-output (I/O) unit.

Instructions or *machine instructions* are explicit commands that

- Govern the transfer of information within a computer as well as between the computers and its I/O devices.
- Specify the arithmetic and logic operations to be performed.

A list of instructions that perform a task is called *program.* The computer is completely controlled by a stored program except for a possible external interruption by an operator or by I/O devices connected to the machine. Data is used to mean any digital information. Each number, character or instruction is encoded as a string of binary digits known as bits each having one of two possible values 0 or 1.

Input Unit

Computers accept the coded information through input units which read the data. The well known input device is Keyboard. When a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted over a cable to either the memory or processor.

Memory Unit The memory unit is used to store program and data. There are two classes of storage known as primary and secondary.

Primary memory is a fast storage that operates at electronic speeds. Programs are stored in the memory while they are executed. The memory contains large number of semiconductor storage cells each capable of storing one bit but instead are processed as groups of fixed size called words. The memory is organized so that a word can be stored or retrieved in one basic operation. A distinct address is associated to each word in the memory. Addresses are numbers that identify successive locations.

Programs must reside in the memory during execution. Instructions and data can be read out or written into the memory under the control of the processor. Memory in which any location can be reached in short and fixed amount of time after specifying its address is called random-access memory (RAM). The small, fast RAM units are called *caches*.

The additional cheaper secondary storage is used when large amount of data and many programs have to be stored particularly for information that is accessed infrequently.

Arithmetic and Logic Unit (ALU)

Any arithmetic and logic operation is initiated by bringing the required operands into the processor where the operation is performed by the ALU. When the operands are brought into the processor they are stored in high speed storage elements called *registers*. Access time to register is faster than access time to the fastest cache unit in the memory hierarchy. The control and the arithmetic logic units are many times faster than any other devices connected to a computer system.

<u>**Output Unit**</u>

The output unit is a counterpart of input unit. Its function is used the processed results to the outside world. The most familiar example of such a device is a printer.

<u>**Control Unit**</u>

The control unit is a well defined physically separate unit that interacts with other parts of the machine. The control unit sends the control signals to other units and senses their states. Timing signals are generated by the control circuits that determine when a given action is to take place. Data transfer between memory and processor is also controlled unit through timing signals. A large set of control lines (wires) carries the signals used for timing and synchronization of events in all units.

1 b)<u>**Basic Operational Concepts**</u>

To perform a given task, an appropriate program consisting of a list of instructions is stored in the memory. Individual instructions are brought from the memory into the processor, which executes the specified operations. Data to be used as operands are also stored in the memory. A typical instruction may be

Add   LOCA, R0

This instruction adds the operand at memory location LOCA to the operand in a register in the processor, R0, and places the sum in the register R0. The original contents of location LOCA are preserved whereas those of R0 are overwritten. First the instruction is fetched from the memory into the processor. Next the operand at LOCA is fetched and added to the contents of R0. Finally the resulting sum is stored in register R0.

Transfers between memory and processor are started by sending the address of the memory location to be accessed to the memory unit and issuing the appropriate control signals. The data is transferred to or from the memory. The memory and processor connection is shown in Fig 1.2.

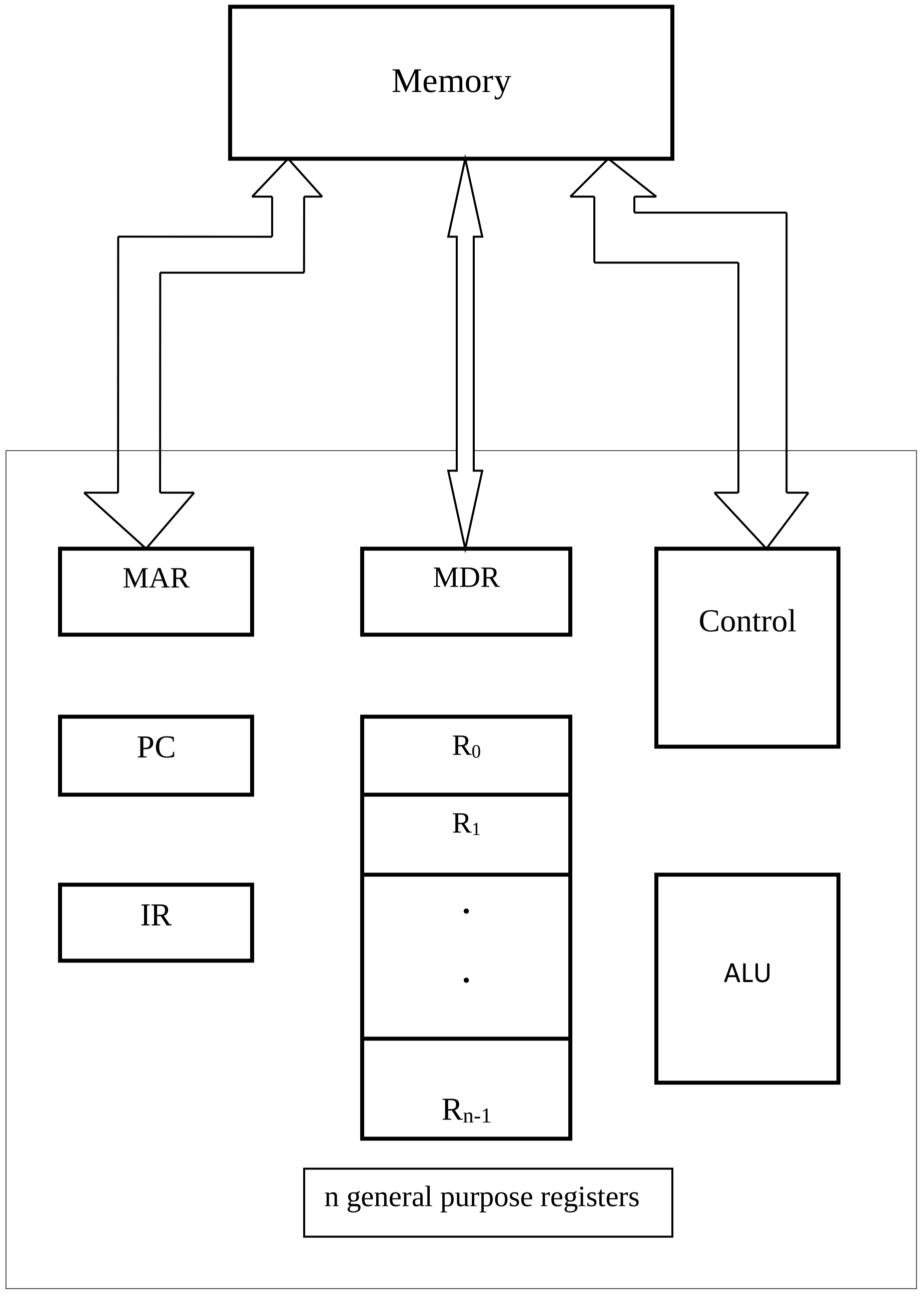

Memory
MAR
MDR
Control
PC
R_0
R_1
IR
ALU
R_{n-1}
n general purpose registers

Fig 1.2 Connections between the processor and memory

The Instruction register (IR) holds the instruction that is currently being executed. Its output is available to control circuits which generate the timing signals that control various processing elements involved in executing the instruction.

The Program Counter (PC) holds the address of the next instruction to be fetched and executed. During the execution of an instruction, the contents of the PC are updated to correspond to the address of the next instruction to be executed. MAR and MDR facilitate communication with the memory.

MAR (Memory Address Register) hold the address of the location to be accessed and MDR (Memory Data Register) contains data written into or read out of the addressed location.

If some devices require urgent servicing then they raise the interrupt signal interrupting the normal execution of the current program. The processor provides the requested service by executing the appropriate interrupt service routine.

1 c)

Bus Structures

The individual parts of a computer need to be connected in an organized way to increase the speed of operation. When a word of data is transferred between units, all its bits are transferred in parallel that is bits are transferred simultaneously over many wires or lines, one bit per line. A group of lines that serves as a connecting path for several devices is called a bus. The simplest way to inter connect functional units is to use a single bus as shown in Fig 1.3.

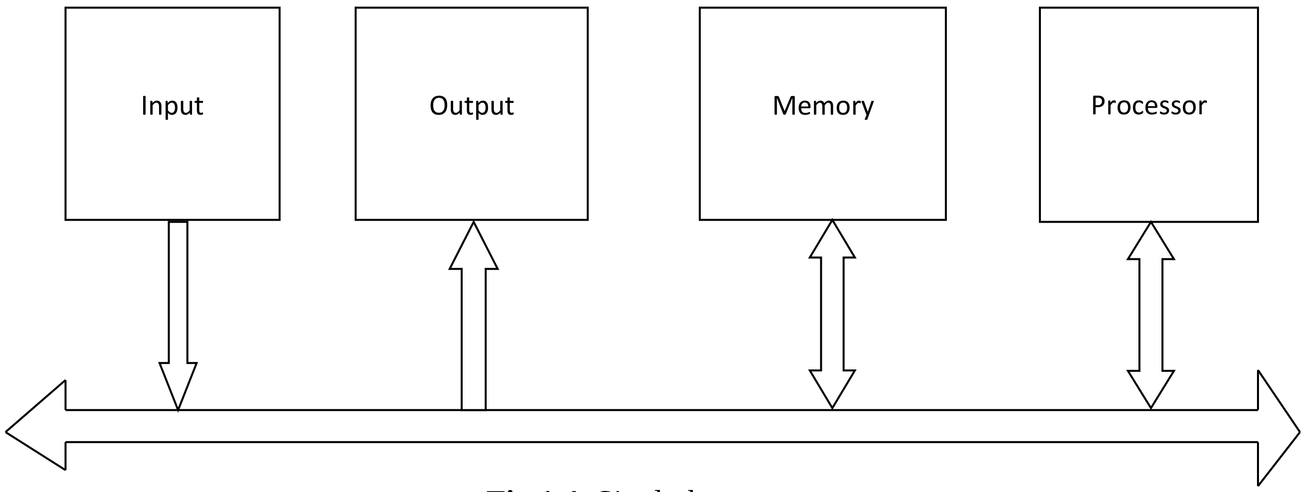


Fig 1.4: Single-bus structure

The main virtue of single-bus structure is its low cost and its flexibility for attaching peripheral devices. Systems that contain multiple buses achieve more concurrency in operation by allowing two or more transfers to be carried out at the same time. This leads to better performance but at increased cost.

A common approach is to include buffer registers with the devices to hold the information during transfers.

2 a)

<u>Instructions and Instruction Sequence</u>

A computer must have instructions capable of performing four types of operations:

- Data transfer between memory and the processor registers
- Arithmetic and logic operations on data
- Program sequencing and control
- I/O transfers

The possible locations that may be involved in transfer of information from one location of computer to another are memory locations, processor registers or registers in the I/O subsystem. The names for the addresses of memory locations may be LOC, PLACE, A, VAR2; processor register names may be R0, R5; and I/O register names may be DATAIN, OUTSTATUS. The expression

$$R1 \leftarrow [LOC]$$

The contents of the memory location LOC are transferred in to the processor register $R1$. This type of notation is known as *Register Transfer Notation.* The *assembly language* format is used to represent machine instructions and programs. The assembly language statement

$$Add\ \ R1, R2, R3$$

Adds two numbers contained in the processor registers $R1$ and $R2$ and placing their sum in $R3$ without changing the contents of $R1$ and $R2$.

The two-address instruction is of the form

Operation   Source, Destination

There are machine instructions that specify only one memory operand. When a second operand is needed, it is understood implicitly to be in unique location. A processor register called the *accumulator* is used for this purpose.

Let us consider the one-address instructions

$$Load\ \ A$$

and

$$Store\ A$$

The Load instruction copies the content of memory location A into the accumulator, and the Store instruction copies the contents of accumulator into memory location A. When data is moved to or from a processor register, the *Move* instruction can be used rather than *Load* or *Store* instructions because of the order of source and destination operands determines which operation is intended. Thus

$$Move\quad A, Ri$$

is the same as

$$Load\ \ A, Ri$$

and

$$Move\ \ Ri, A$$

is same as

$$Store\ \ Ri, A$$

A substantial increase in speed is achieved when several operations are performed in succession on data in processor registers without the need to copy the data to or from the memory. It is also possible to use instructions in which location of all operands are defined implicitly. Such instructions are found in machines that store operands in structure called *pushdown stack.* In this case instructions are called *zero-address* instructions.

Instruction Execution and Straight-Line Sequence

We assume computer allows one memory operand per instruction and has a number of processor registers. Fig 1.12 shows a program segment in the memory of a computer. The word length is 32 bits and the memory is byte addressable. Each instruction is 4 bytes long, the second and third instructions start at addresses $i + 4$ and $i + 8$.

The Program Counter (PC) contains the address of the instruction to be executed next. To begin executing a program, the address of its fist instruction must be placed in to PC. Then the processor control circuits use the information in the PC to fetch and execute the instructions, one at a time, in the order of increasing addresses. This is called *straight-line sequencing.*

Executing a given instruction is a two phase-procedure. In the first phase called *instruction fetch*, the instruction is fetched from the memory location whose address is in the PC. This instruction is placed in the *instruction register* (IR) in the processor. At the start of second phase called *instruction execute*, the instruction in the IR is examined to determine which operation is to be determined.

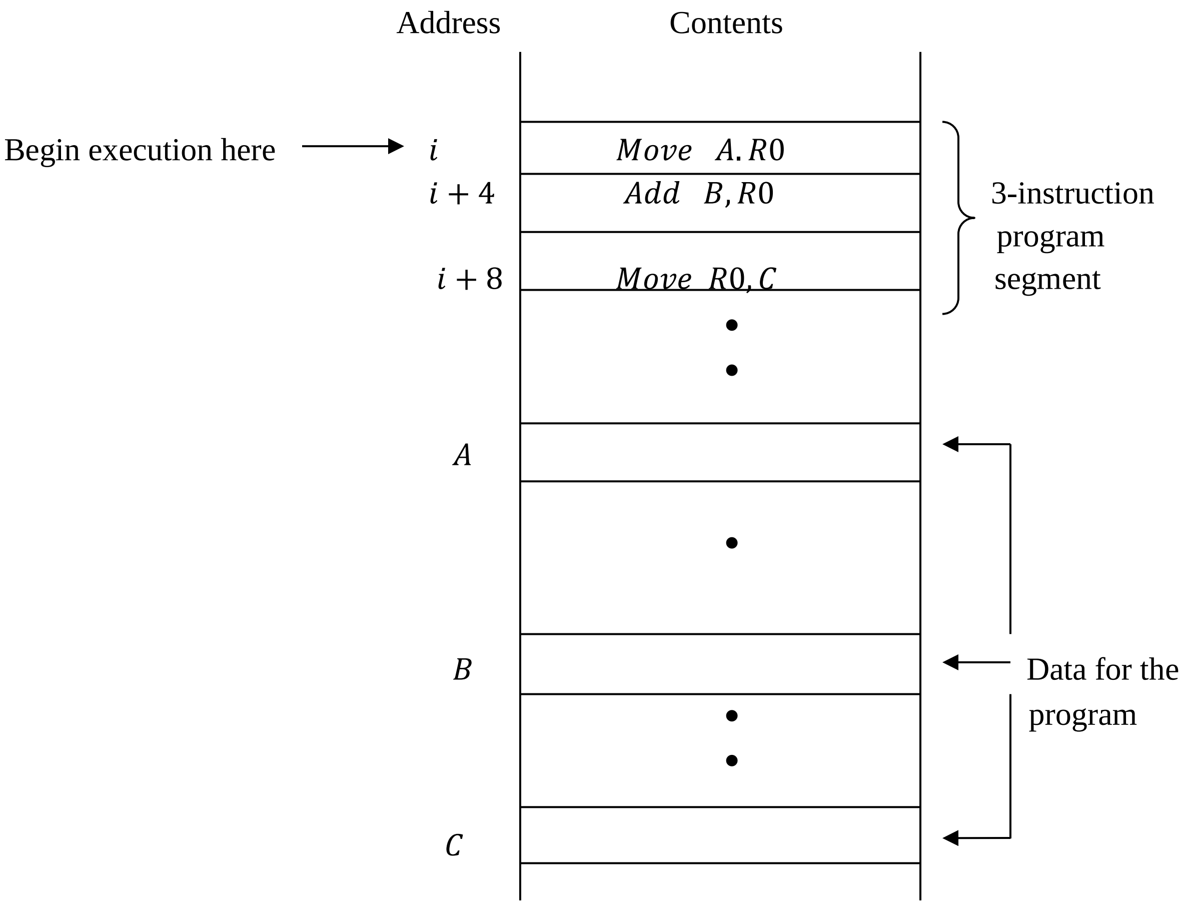


Fig 1.12: A program for $C \leftarrow [A]+[B]$

Branching

Consider a task of adding a list of n numbers. The address of the memory locations containing the n numbers are given as $NUM1, NUM2, \ldots\ldots\ldots NUMn$ and a separate ADD instruction is used to add each number to the contents of the register $R0$. After all numbers have been added, the result is placed in the memory location SUM.

Instead of using a long list of Add instructions, it is possible to place a single Add instruction in a program loop as shown in Fig 1.13. The *loop* is a straight line sequence of instructions executed as many times as needed. It starts at location LOOP and ends at the instruction Branch>0. $R1$ is used as a counter to determine the number of times loop is executed and holds the contents of the memory location N which contains the number of entries in the list n. Then, within the body of loop, the instruction

$$Decreament \quad R1$$

Execution of the loop is repeated as long as the result of the decrement operation is greater than zero.

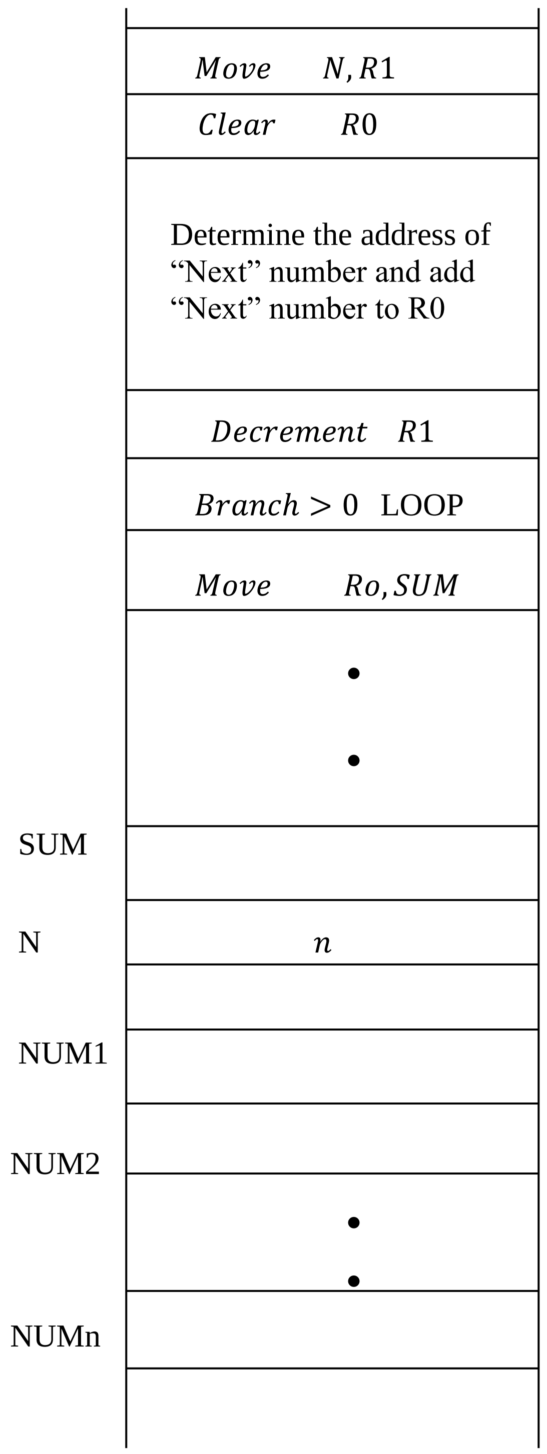


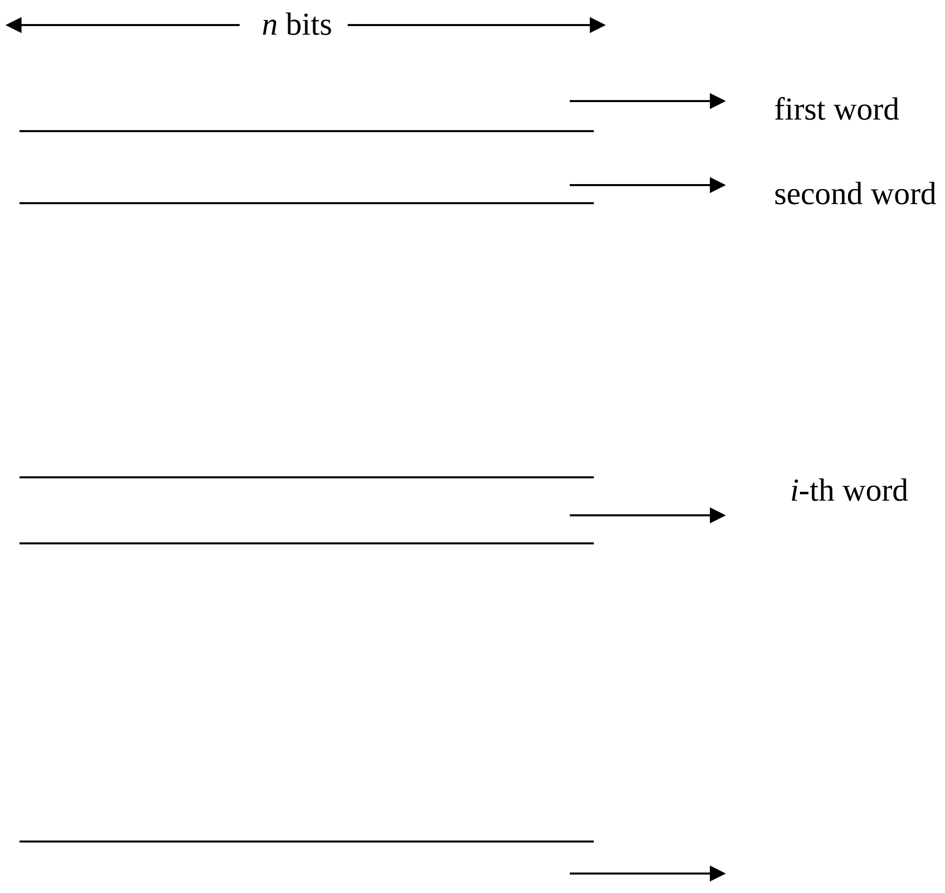
Fig 1.13: Using a loop to add n numbers

A *conditional branch* instruction causes a branch only if a specified condition is satisfied. If the condition is not satisfied, the PC is incremented in a normal way and the next instruction in sequential address order is fetched and executed.

2 b)

Memory Locations and Addresses

Number and character operands as well as instructions are stored in the memory of a computer. The memory consists of millions of storage *cells*, each of which can store bit of information having a value 0 or 1. The memory is organized so that a group of *n* bits can be stored or retrieved in a single basic operation. Each group of *n* bits is referred to as *word* of information, and *n* is called word length. The memory of a computer can be systematically represented as collection of words as shown in Fig 1.10.



last word

Fig 1.10: Memory words

Accessing the memory to store or retrieve a single item of information, either a byte or word, requires distinct names or *addresses* for each item location. The 2^k addresses constitute the *address space* of the computer, and the memory can have up to 2^k addressable locations. For example, a 24 bit address generates an address space of 2^{24} (16,777,216) locations.

A byte is always 8 bits but the word length typically ranges from 16 to 64 bits. The successive addresses refer to successive byte locations in the memory. The term *byte-addressable memory* is used for this assignment. Byte locations of addresses 0,1,2 ...Thus, if word length of the machine is 32 bits, successive words are located at addresses 0,4,8, ..., with each word consisting of four bytes.

Big-Endian and Little-Endian Assignments

The name *big-endian* is used when the lower byte addresses are used for the most significant bytes (the leftmost bytes) of the word. The *little-endian* is used for the opposite ordering, when the lower byte addresses for the less significant bytes (the rightmost bytes) of the word. In both cases, byte addresses 0,4,8, ..., are taken as the address for the successive words in the memory and are the addresses used when specifying the memory read and write operation for the words. The two ways that the byte addresses can be used across the words as shown in Fig 1.11.

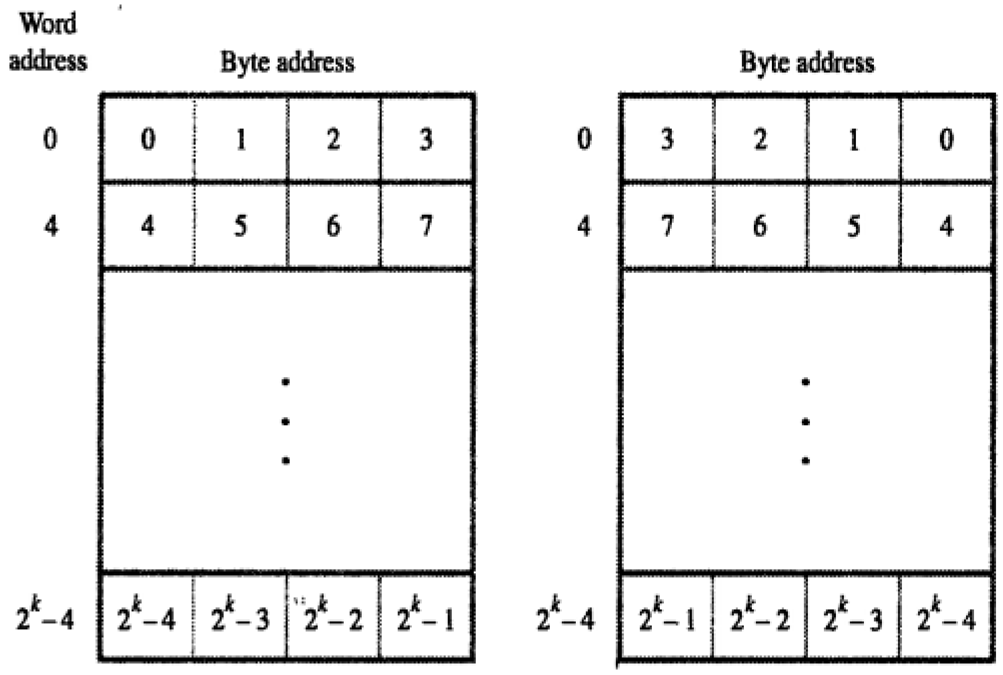


Fig 1.11: a) Big-endian assignment b) Little-endian assignment

The word locations have *aligned* addresses where the word begins at a byte address that is a multiple of number of bytes in a word. If the word length is 16 (2 bytes), aligned words begins at byte addresses $0,2,4$ ....

Individual characters can be accessed by their byte address. The beginning of a string is indicated by giving the address of the byte containing its first character. Successive byte locations contain successive characters of the string. A special character "end of string" can be used as last character in the string, or a separate memory word location or processor register can contain a number indicating the length of the string in bytes.

2 c)

IEEE Standard for Floating-Point Numbers

A computer must be able to represent numbers and operate on them in such a way that the position of the binary point is variable and automatically adjusted as the computation proceeds. In such a case binary point is said to float and the numbers are called *floating-point numbers*. For example, in the familiar decimal scientific notation, the numbers can be written as $6.0247 \times 10^{23}, 6.6254 \times 10^{-27}$. The numbers are said to be given to five *significant digits.* When the decimal point is placed to the right of the first (nonzero) significant digit, the number is said to be *normalized.* The floating-point number is represented by its sign, string of significant digits, commonly called *mantissa,* and an exponent to an implied base for the scale factor.

The standard for representing floating-point numbers in 32 bits is specified by IEEE is given in Fig 1.14. A 24 bit mantissa can represent a 7-digit decimal number and an 8 bit exponent. The sign of the number is given in the first bit, followed by the representation for the exponent (to the base 2) of the scale factor. The signed exponent component E is stored in the form of unsigned integer $E' = E + 127$. This is called excess-127 format.

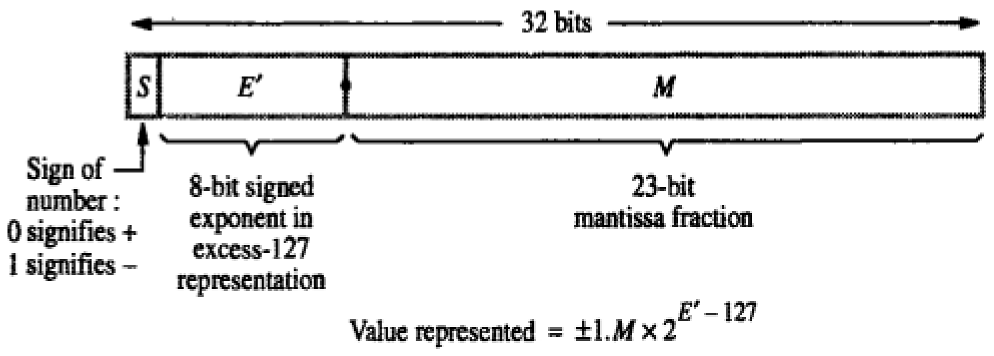

32 bits
S
E′
M
Sign of number : 0 signifies + 1 signifies −
8-bit signed exponent in excess-127 representation
23-bit mantissa fraction
Value represented = ±1.M × 2^(E′ − 127)

Fig 1.14: IEEE standard floating-point format

<u>MODULE 2</u>

3 a)

<u>Addressing Modes</u>

The different ways in which the location of an operand is specified in an instruction is known as *addressing modes*. Variables and constants are the simplest data types. In assembly language, a variable is represented by allocating a register or memory location to hold its value. Thus, the value can be changed as needed using appropriate instructions.

- *Register mode* – The operand is the contents of a processor register; the name of the register is given in the instruction.
- *Absolute mode* – The operand is in a memory location; the address of this location is given explicitly in the instruction.

The instruction $Move \quad LOC, R2$

uses two modes. Processor registers are temporary storage locations where data in a register is accessed using the Register mode. Address and data constants can be represented in assembly language using the Immediate mode addressing where the operand is given explicitly in the instruction. For example, the instruction

$$Move \quad 200_{immediate}, R0$$

Places the value 200 in register $R0$. A common convention is to use # in front of the immediate value to indicate that this value is to be used as an immediate operand. Hence we can write the instruction above in the form

$$Move \quad \#200, R0$$

Constant values are used frequently in high-level language programs. The statements $A = B + 6$ contains the constant 6. Assuming that A and B have been declared as variables and may be accessed using Absolute mode.

$$Move \quad B, R1$$
$$Add \quad \#6, R1$$

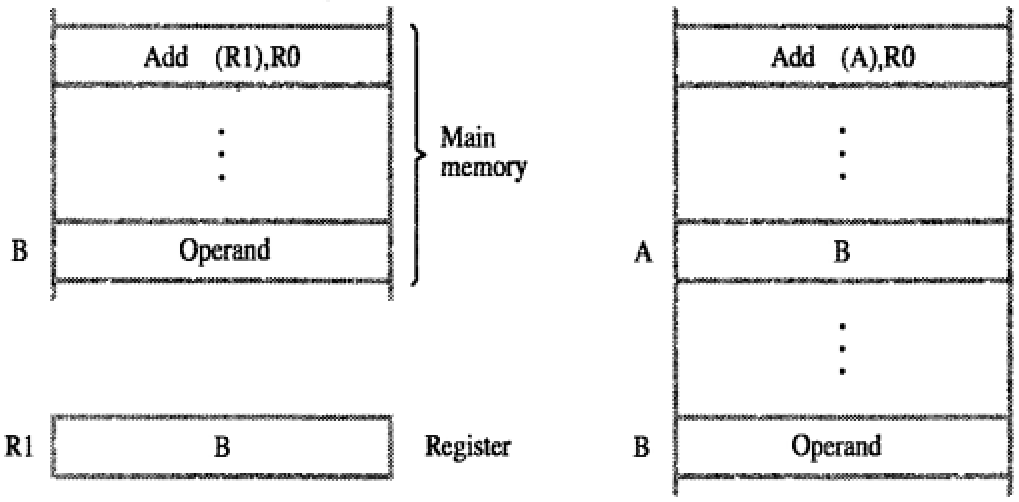
$$Move \quad R1, A$$

Indirection and Pointers

In indirect mode addressing, the instruction does not give the operand or the address explicitly. Instead it provides information from which the memory address of the operand can be determined. This address is referred to as *effective address* (EA) of the operand.

Indirect mode – The effective address of the operand is the contents of a register or memory location whose address appears in the instruction.

To execute the Add instruction in Fig 2.1a, the processor uses the value B, which is in the register $R1$, as the effective address of the operand. It requests a read operation from the memory to read the contents of location B. The value read is the desired operand, which the processor adds to the contents of register $R0$. Indirect addressing through a memory location is also possible as shown in Fig 2.1b. In this case, the processor first reads the contents of memory location A, then request the second read operation using the value B as a address to obtain the operand.



a) Through a general purpose register b) Through a memory location

Fig 2.1: Indirect addressing

The register or the memory location that contains the address of the operand is called a *pointer.*

Indexing and Arrays

This addressing mode provides flexibility for accessing operands and is useful in dealing with lists and arrays.

Index mode – The effective address of the operand is generated by adding a constant value to the contents of a register. This register is referred to as *index register*.

We indicate the Index mode symbolically as $X(Ri)$ where X denotes the constant value contained in the instruction and Ri is the name of the register involved. The effective address of the operand is given by

$$EA = X + [Ri]$$

Fig 2.2 illustrates two ways of using Index mode. In Fig 2.2a, the index register $R1$ contains the address of the memory location and the value X defines an *offset* or *displacement* from this address to the location where the operand is found.

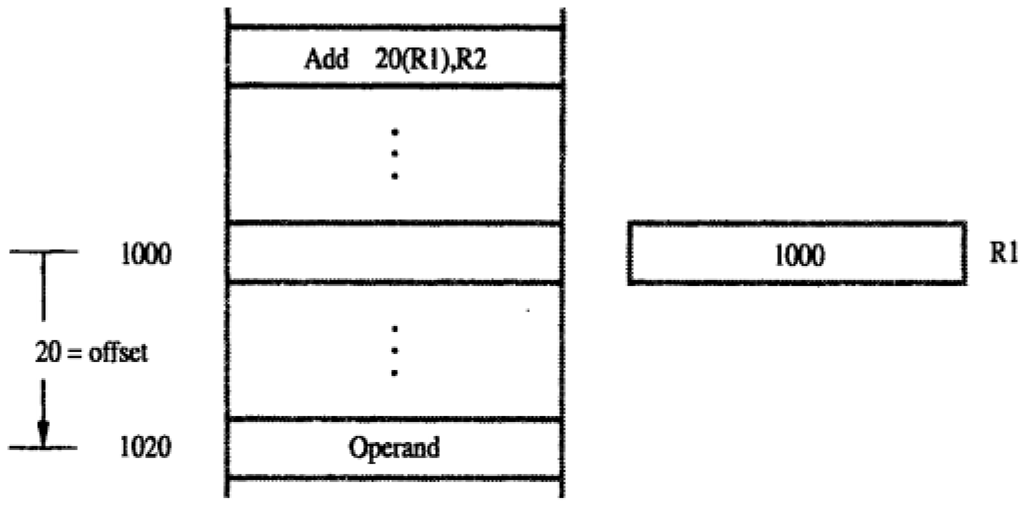


Fig 2.2 a: Offset is given as a constant

An alternate use is illustrated in Fig 2.2b. Here, the constant X corresponds to a memory address and the content of the index register defines the offset to the operand. In either case, the effective address is the sum of two values, one is given explicitly in the instruction and the other is stored in the register.

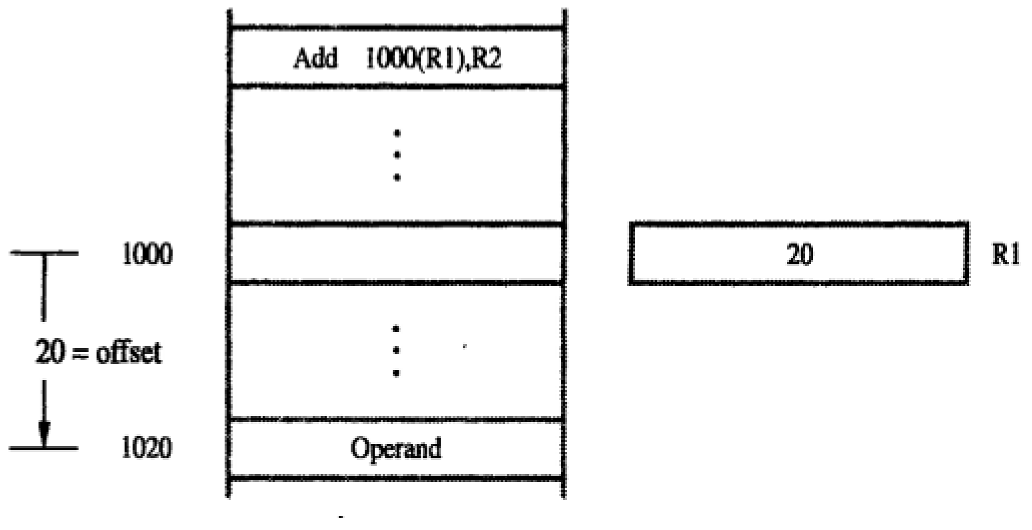


Fig 2.2b: Offset is in the register

Relative Addressing

Here the Program Counter (PC) is used instead of a general purpose register. In *Relative mode,* the effective address is determined by the Index mode using program counter in place of general-purpose register Ri. It's most common use is to specify the target address in branch instructions. An instruction such as

$$Branch > 0 \quad LOOP$$

causes program execution to go to the branch target location identified by the name LOOP if the branch condition is satisfied. This location can be computed by specifying it as an offset from the current value of the program counter. Suppose that Relative mode is used to generate the target branch address LOOP in the Branch instruction of the program

```
LOOP:   Add         (R2),R0
        Add         #4,R2
        Decrement   R1
        Branch > 0  LOOP
```

Assume that the four instructions of the loop body, starting at LOOP are located at memory locations $1000, 1004, 1008$ and 1012. Hence the updated contents of the PC at the time of branch target address is generated will be 1016. To branch to location LOOP(1000), the offset needed is $X = -16$.

Auto-increment mode – The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register is automatically incremented to point to the next item in the list. The Auto-increment mode is written as

$$(Ri)+$$

The increment is 1 for byte-sized operands, 2 for 16 bit operands and 4 for 32 bit operands.

Auto-decrement mode – The content of a register specified in the instruction is first automatically decremented and then used as effective address of the operand. In this mode, operands are accessed in descending address order.

These two modes can be used together to implement an important data structure called stack.

3 b)

```
ORIGIN 1000                        ; SOURCE DATA STARTS FROM ADDRESS1000

SRC DATAWORD  100,120,20,30,.....,60  ; DECLARE WORD ARRAY

ORIGIN 2000                        ; DESTINATION AT 2000

SUM DATAWORD 0

N EQU 100                          ; THE NUMBER OF ELEMENTS IN THE ARRAY

ORIGIN 3000                        ; PROGRAM IS STORED FROM LOCATION 3000

START  MOVE #SRC,R1                ; INITIALISE POINTER

       MOVE N,R2                   ; INITIALIZE COUNTER


CLR R3                             ; CLEAR TARGET REGISTER FOR RESULT

BACK ADD (R1)+,R3

DEC R2

BGTZ BACK

MOVE  R3,SUM                       ; STORE THE RESULT

END START
```

3 c)

Assembler Directives

The assembly language allows the programmer to specify other information needed to translate the source program to object program. Suppose the name SUM is used to represent the value 200. This fact may be conveyed to the assembler program through a statement such as

```
SUM EQU 200
```

This statement does not denote the instruction that will be executed when the object program is run. It informs the assembler that the name SUM should be replaced by the value 200 wherever it appears in the program. Such statements are *assembler directives* (or commands) are used by the assembler when it translates the source program in to a object program.

ORIGIN is a directive that tells the assembler program where in the memory to place the data block.

DATAWORD directive is used to inform the assembler to place the data in the address.

RESERVE directive declares a memory block and does not cause any data to be loaded in these locations.

ORIGIN directive specifies that the instructions of an object program are to be loaded in the memory starting at an address.

END is directive which indicates the end of the source program text. The END directive includes the label START, which is the address of the location at which execution of the program is to begin.

RETURN is an assembler directive that identifies the point at which the execution of the program should be terminated.

The assembly language requires statements in a source program to be written in the form

```
Label Operation Operand(s) Comment
```

Label is an optional name associated with the memory address where the machine language instruction produced from the statement is loaded. The Operation field contains the OP code mnemonic of the assembler directive. The Operand field contains the addressing information for accessing one or more operands depending on the type of instruction.

4 a) Stack operation

Data operated on by a program can be organized in a variety of ways. A *stack* is a list of data elements usually words or bytes with the accessing restriction that elements can be added or removed at one end of the list only. This end is called the top of the stack and the other end is called the bottom. The structure is called *pushdown* stack. *Last-in-first-out* (LIFO) stack is a type of storage mechanism where the last data item placed on the stack is the first one removed when retrieval begins. The terms *push* and *pop* are used to describe placing a new item on the stack and removing top item from the stack. The stack grows in the direction of decreasing memory address.

Fig 2.4 shows a stack of word data items in the memory of a computer. It contains the numerical values, with 43 at the bottom and -28 at the top. A processor register is used to keep track of the address of the element of the stack that is at the top at any given time. This register is called the *stack-pointer* (SP).

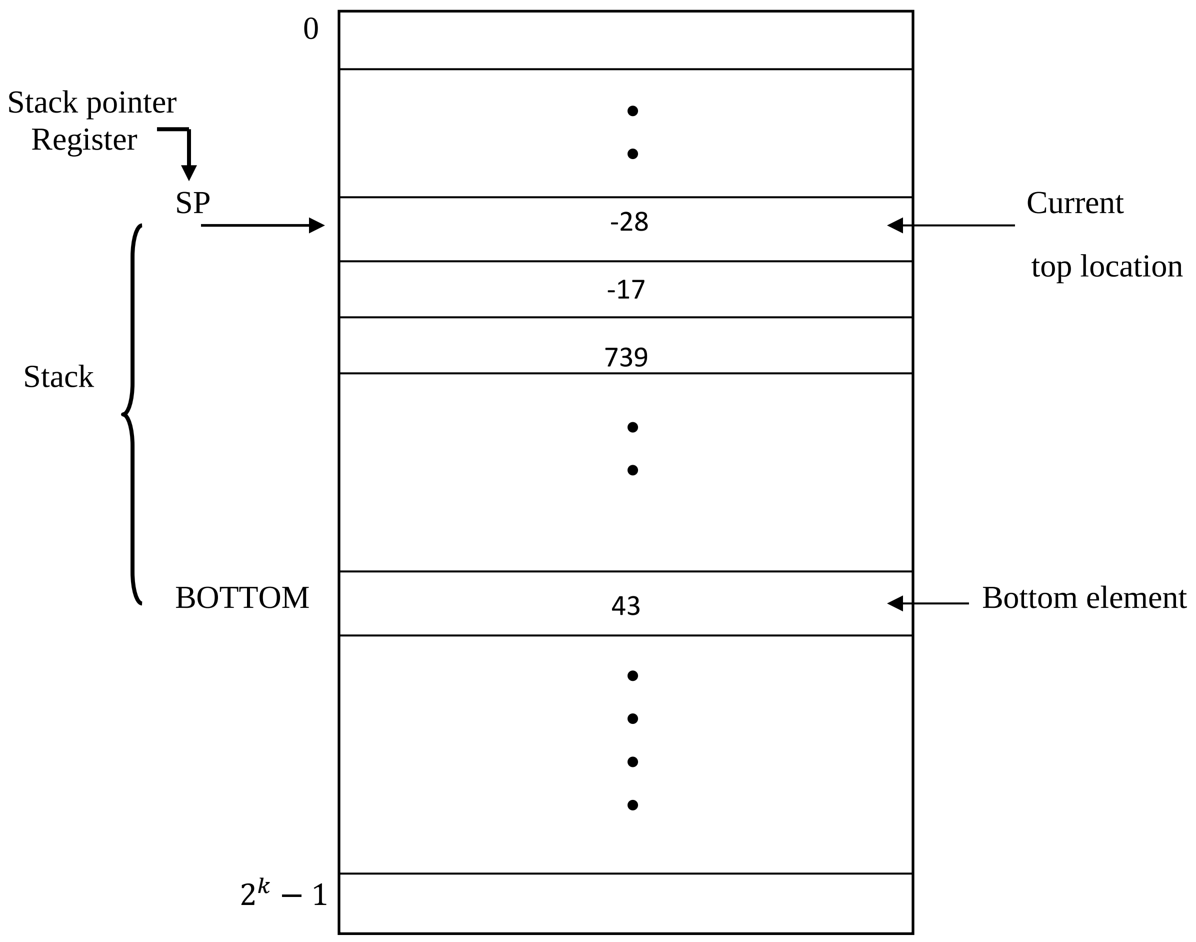


Fig 2.4: A stack of words in the memory

Assuming a byte addressable memory with 32 bit word length the Push operation can be implemented as

```
Subtract   #4, SP
Move        NEWITEM, (SP)
```

These two instructions move word from location NEWITEM onto the top of the stack, decrementing the stack pointer by 4 before the move. The Pop operation can be implemented as

Move (SP), ITEM
Add #4, SP

These two instructions move the top value from the stack into location ITEM and then increment the stack pointer by 4 so that it points to the new top element. Fig 2.5 shows the effect of each of these operations on the stack in Fig 2.4.

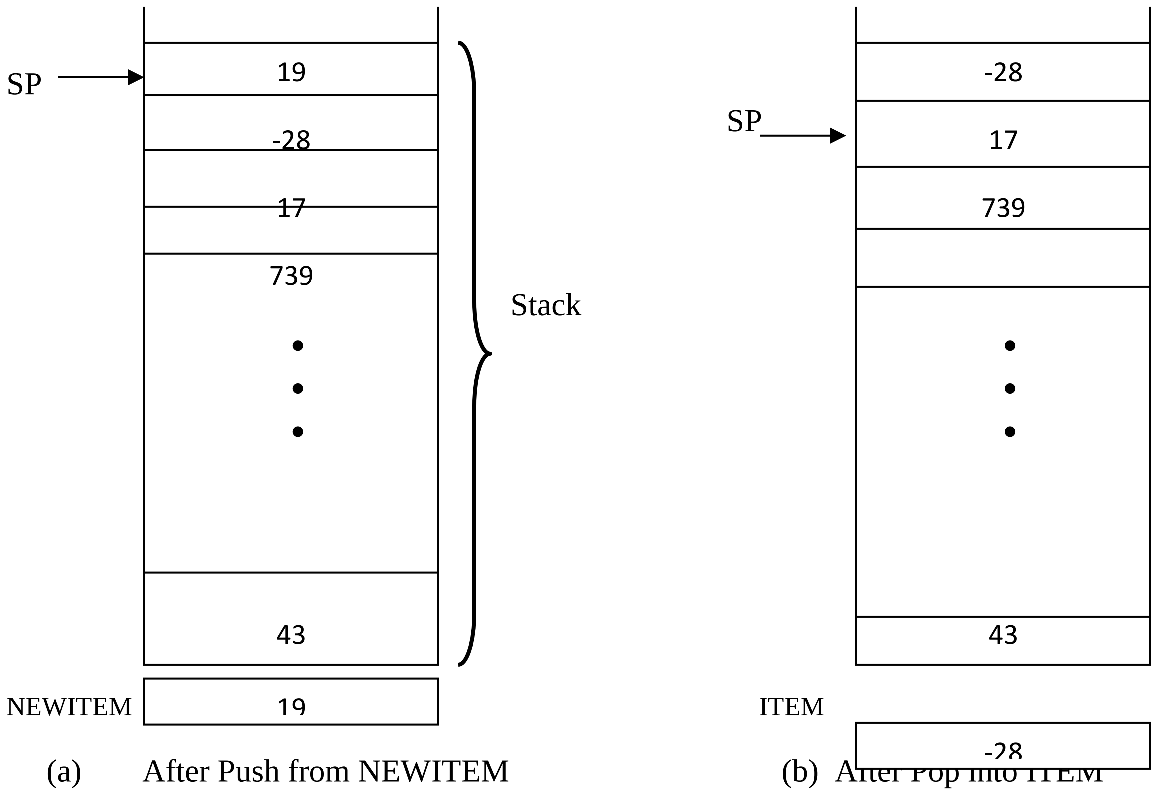


Fig 2.5: Effect of stack operations on the stack

If the processor has the Autoincrement and Autodecrement addressing modes, then the push operation can be performed by the single instruction

Move NEWITEM, -(SP)

and the pop operation can be performed by

Move (SP) +, ITEM

The Compare instruction

Compare src, dst

performs the operation

$$[dst] – [src]$$

and sets the condition flags according to the result. It does not change the value of either operand.

4 b) Subroutine

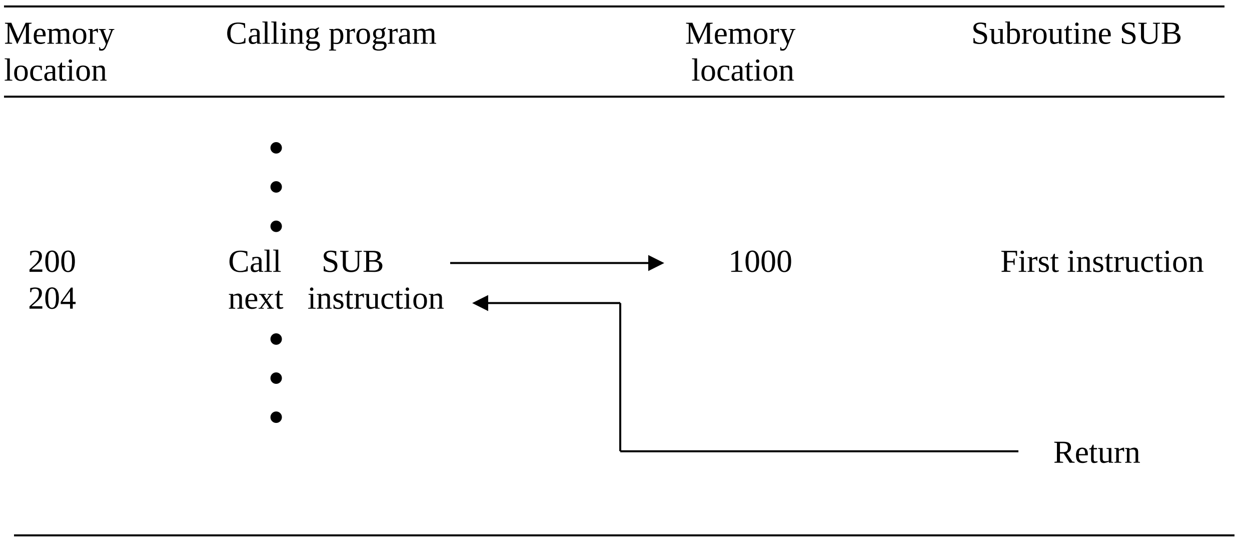
It is often necessary to perform a particular subtask many times on different data values. Such subtask is called *subroutine.* When a program branches to a subroutine we call that it is *calling* a subroutine. The instruction that performs this branch operation is called a Call instruction. The subroutine is said to *return* to program that called it by executing a Return instruction. The location where the calling program resumes execution is the location pointed by the updated PC while the Call instruction being executed. Hence the contents of the PC must be saved by the Call instruction to enable correct return to the calling program. This way in which the computer makes it possible to call and return from subroutines is referred to as *subroutine linkage method.*

The Call instruction is a special branch instruction that performs the following operations:

1. Store the contents of PC in the link register.
2. Branch to the target address specified by the instruction.

The Return instruction is a special branch instruction that performs the operation:
Branch to the address contained in the link register.

Fig 2.6 illustrates this procedure.



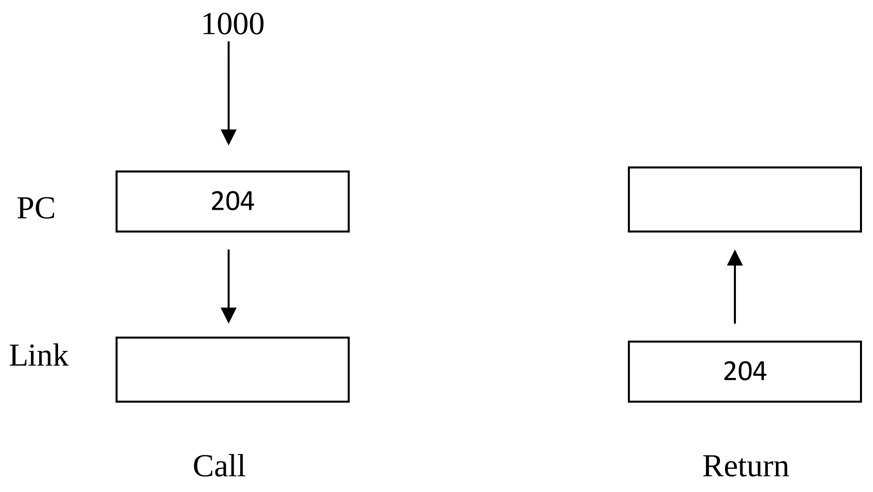


Fig 2.6: Subroutine linkage using a link register

Subroutine Nesting and the Processor Stack

In *subroutine nesting,* we have one subroutine call another. In this case the return address of the second call is also stored in the link register destroying its previous contents. Hence it is essential to save the contents of the link register in some other location before calling another subroutine. Otherwise return address of the first subroutine will be lost.

Subroutine nesting can be called to any depth. Eventually the last subroutine called completes its computations and returns to the subroutine that called it. The return address needed for this first return is the last one generated in the nested call sequence. That is, return addresses are generated and used in last-in-first-out-order. The return addresses associated with the subroutine calls should be pushed on to the stack. The stack pointer points to a stack called *processor stack.* The Call instruction pushes the contents of PC onto processor stack and loads the subroutine address into the PC. The Return instruction pops the return address from the processor stack into the PC.

4 C) Rotate and Shift operations

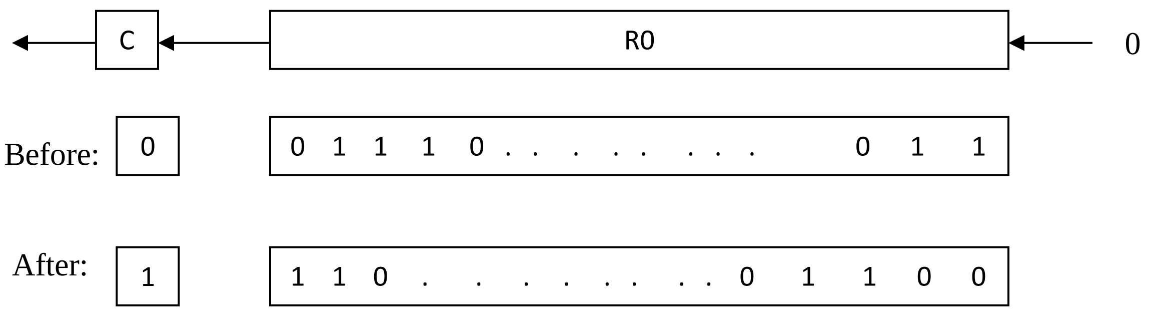
There are applications that require bits of an operand to be shifted to the right or left some specified number of bit positions. For general operands we use a logical shift. For a number we use an arithmetic shift which preserves the sign of the number.

Logical Shifts

Two logical shift instructions are needed, one for shifting left (LShiftL) and another for shifting right (LShiftR). These instructions shift an operand over a number of bit positions specified in a count operand contained in the instruction. The general form of logical left shift instruction is

LShiftL count, dst

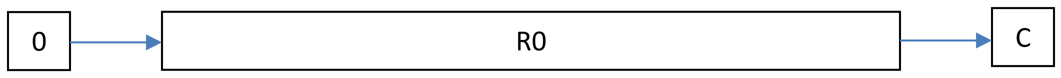
The count operand may be given as an immediate operand or it may be contained in the processor register. Vacated positions are filled with zeros, and the bits shifted out are passed through the Carry flag C, and then dropped. Involving the C flag in shifts is useful in arithmetic operations on large numbers that occupy more than one word. Fig 2.10 illustrates all the shift operations.



(a) Logical shift Left LShiftL #2, R0

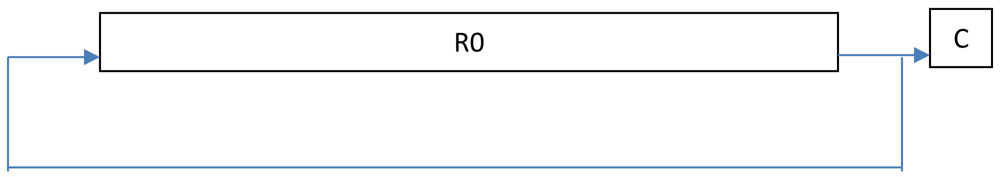
LShiftR count, dst

The count operand may be given as an immediate operand or it may be contained in the processor register. Vacated positions are filled with zeros, and the bits shifted out are passed through the Carry flag C, and then dropped. Involving the C flag in shifts is useful in arithmetic operations on large numbers that occupy more than one word.



Rotate Instructions

RotateR count, dst



5. a. Show the possible register configurations of I/O interface, explain program controlled input/output.

Sol:

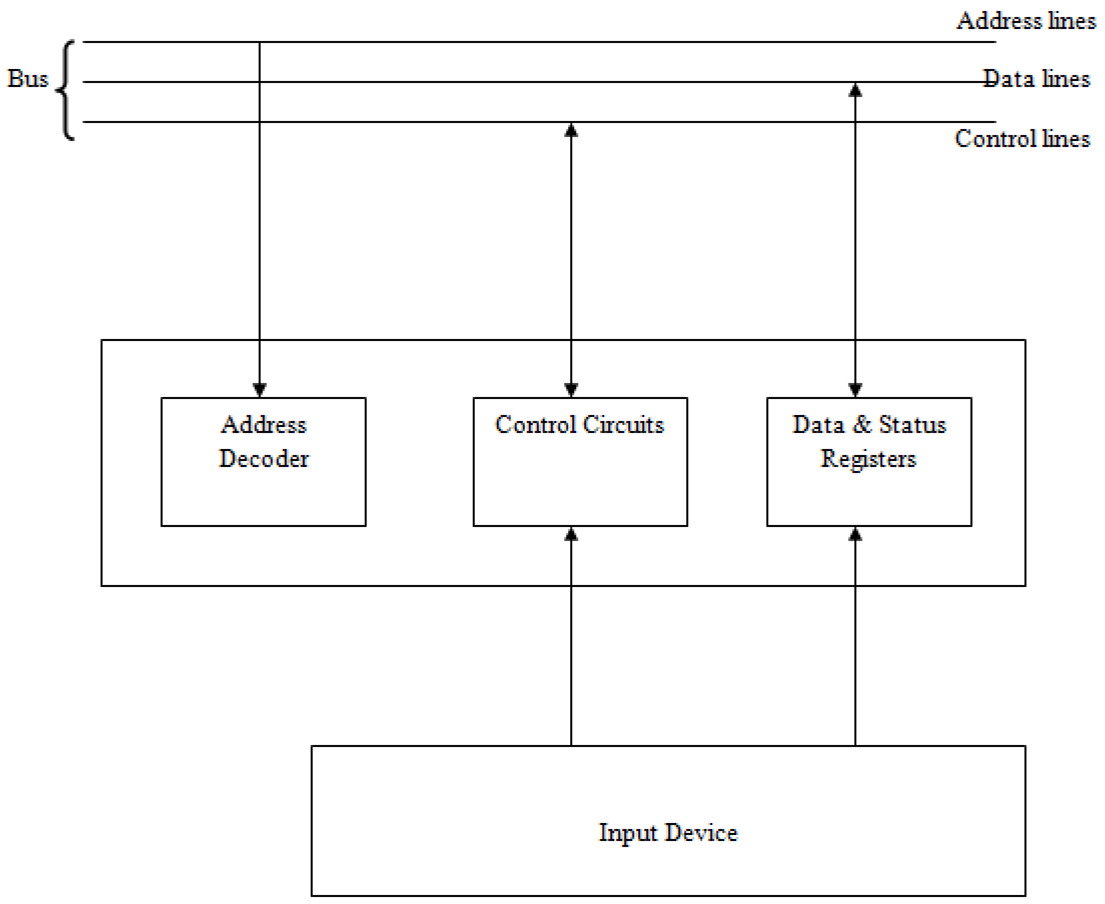


Fig 5: I/O interface of an input device

Fig 5 illustrates the hardware required to connect the I/O device to the bus. The address decoder enables the device to recognize its address when its address appears on the address lines. The data register holds the data being transferred to or from the processor. The status register contains the information relevant to the operation of I/O device. Both status and data registers are connected to the data bus and assigned unique addresses.

The address decoder, data & status registers and the control circuitry required to coordinate I/O transfers constitutes the device interface circuit.

5. b. What is an interrupt? With an example illustrate the concept of interrupt.

Sol: The other tasks can be performed by the processor while waiting for the I/O device to become ready. When the I/O device becomes ready, it sends a hardware signal called interrupt to the processor. Using the interrupts waiting periods can be eliminated.

Consider a task that requires some computations to be performed and the results to be printed on a line printer. This is followed by more computations and output and so on. Let the program consists of two routines COMPUTE and PRINT. Assume COMPUTES produces a set of ‘n’ lines of output to be printed by PRINT routine.

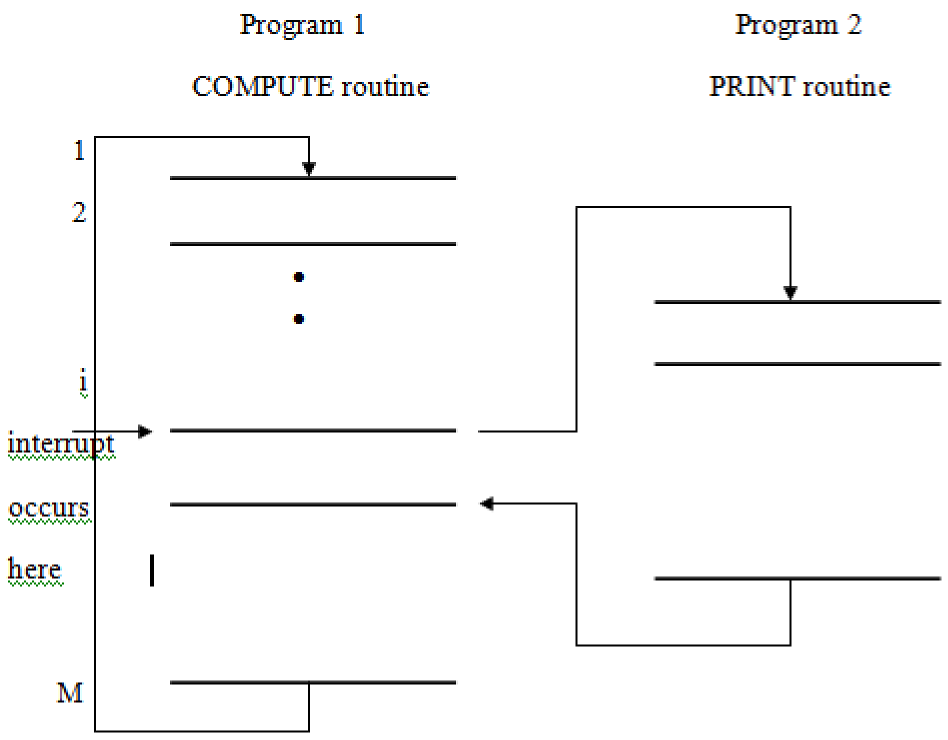


Fig 5 b: Transfer of control through the use of interrupts

It is possible to overlap printing and computation ie to execute COMPUTE routine while printing is in progress, a faster overlap speed of execution will result. Whenever printer becomes ready, it alerts the processor by sending a interrupt request signal. In response the processor interrupts the COMPUTE routine and transfers the control to the PRINT routine. This process continues until all ‘n’ lines are printed and PRINT routine ends.

If COMPUTE takes longer to generate ‘n’ lines than the time required to print them, then the processor will be performing useful computations all the time. Saving registers also increases the delay between the time the interrupt request is received and the start of execution of interrupt service routine. This delay is called interrupt latency.

6a. Explain in detail where a number of devices capable of initiating an interrupt are connected to processor? How to resolve them?

Sol: The information needed to determine whether a device is requesting an interrupt is available in its status register. When a device raises an interrupt request, one of the bits of the status register is set to 1 which we call IRQ bit.

KIRQ, DIRQ are the interrupt request bits for keyboard and display. The polling scheme ha the disadvantage that the time spent interrogating the IRQ bits of all the devices that may not be requesting any service. An alternate approach is to use vectored interrupts.

Interrupt Nesting:

I/O devices should be organized in a priority structure. An interrupt request from a high priority should be accepted while the processor is serving another request from the lower priority device.

We can assign priority level to the processor that can be changed under program control. The priority level of the processor is the priority of the program that is currently being executed. The processor accepts interrupts from devices that have priorities higher than its own.

The processor is in supervisory mode when it is executing the OS routines. It switches to User mode before beginning to execute application programs. The privileged instructions can be executed only while the processor is running in the supervisory mode. A multiple priority scheme can be implemented by using separate interrupt request and interrupt acknowledge lines from each device.

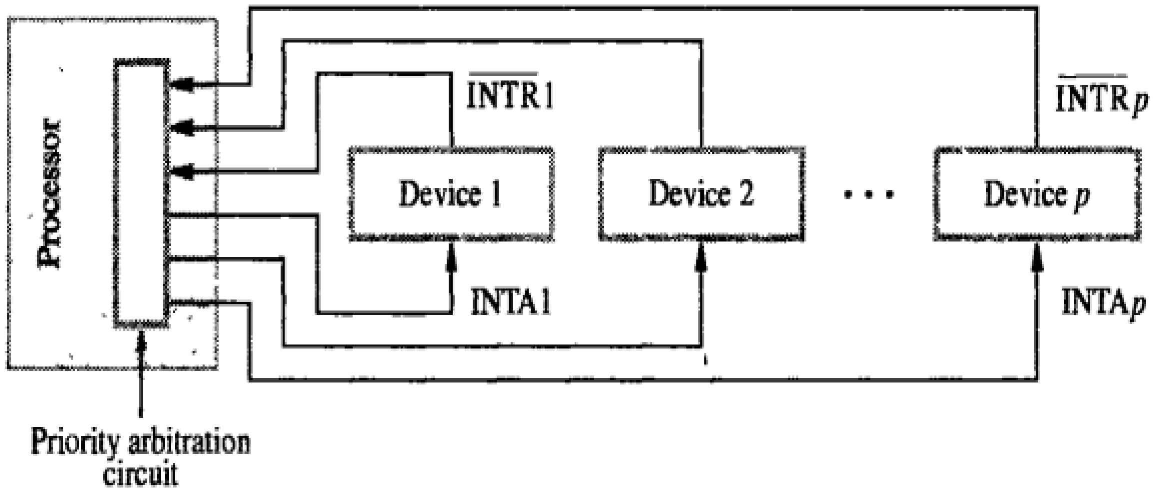

Processor
$\overline{\text{INTR}}1$
$\overline{\text{INTR}}p$
Device 1
Device 2
Device p
INTA1
INTAp
Priority arbitration circuit

Fig 6 a: Implementation of interrupt priority using individual interrupt request & acknowledge lines

6b. Explain the registers involved in DMA interface to illustrate DMA.

Sol: To transfer large blocks of data at high speeds, an alternate approach is used. A special control unit may be provided to allow transfer of block of data directly between external device and main memory without intervention by processor. This approach is called direct memory access or DMA.

DMA transfers are performed by control circuits that are part of I/O interface called DMA controller. The DMA controller performs functions that would normally be carried out by processor when accessing main memory.

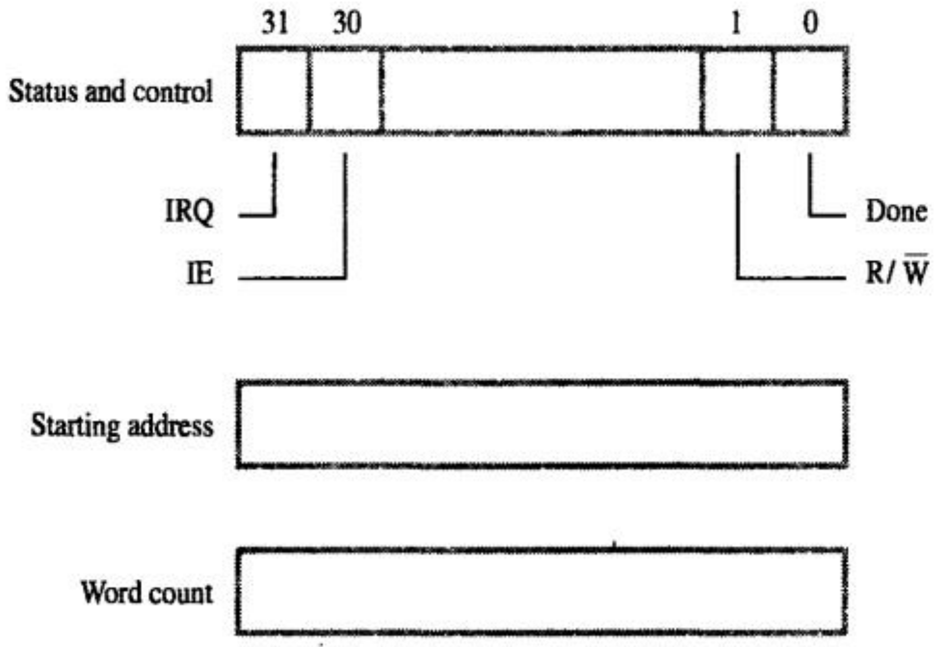


Fig 6b: Registers in DMA interface

The R/$\overline{W}$ bit determine the direction of transfer. When this bit is set to 1 by a program instruction, the controller performs read operation that is it transfers data from memory to I/O device. When transfer is complete, it sets done flag to 1. When IE is1, it causes the controller to raise an interrupt after it has completed transferring block of data. Finally IRQ bit is set to 1 when it has requested interrupt.

Requests from DMA devices are given high priority than processor requests. Among different DMA devices high priority is given to high speed peripherals such as disks, high speed network interface or graphic display device.

The processor originates most memory cycles, the DMA controller is said to steal memory cycles from processor. This technique is called cycle stealing. DMA controller is given access to main memory to transfer a block of data without interruption. This is called as block or burst mode.

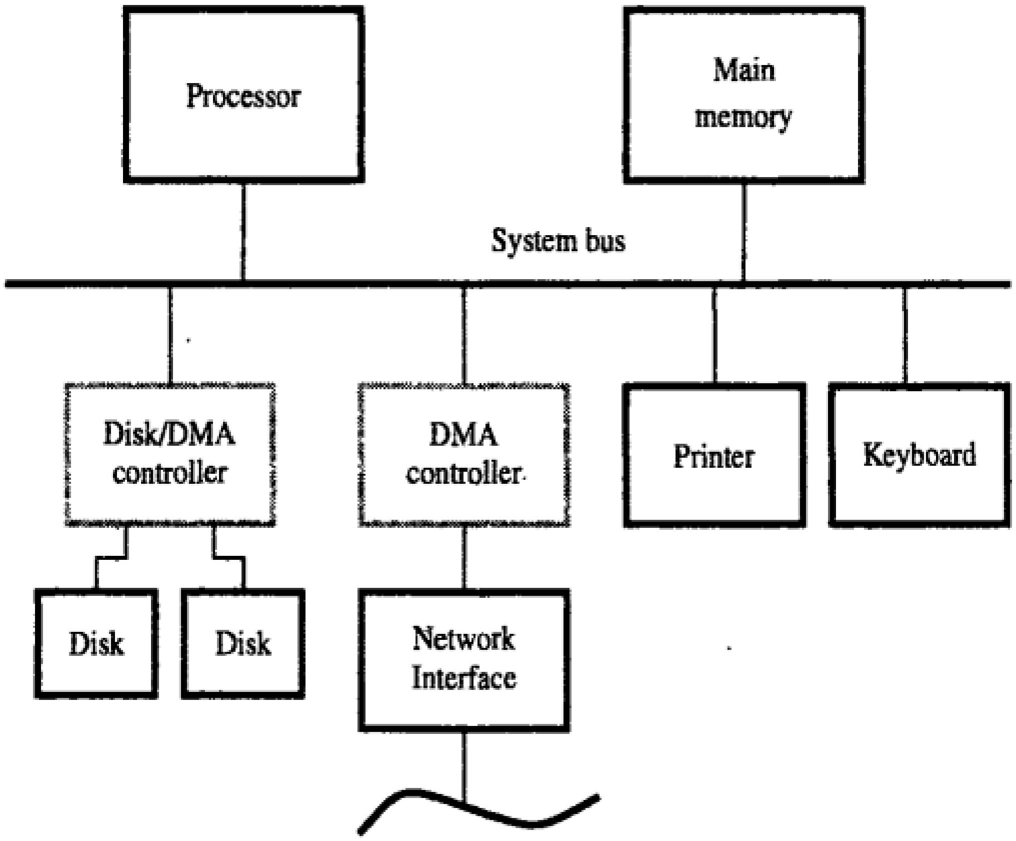


Fig 6b: Use of DMA controllers in computer system

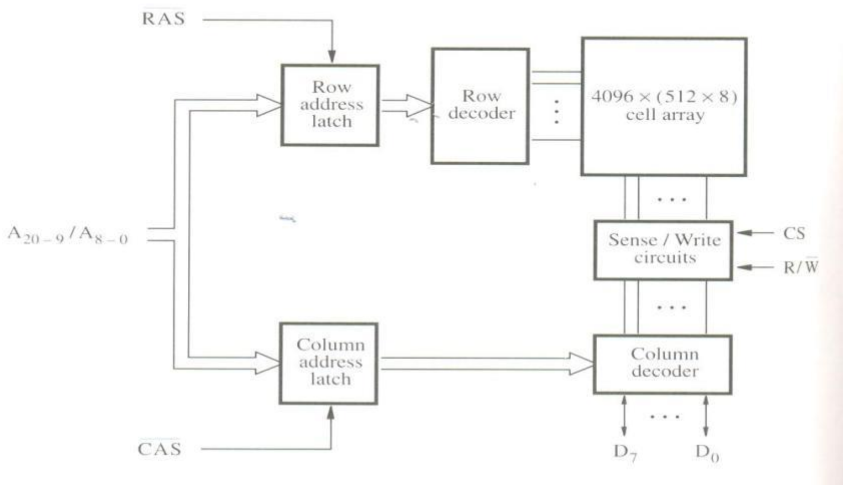
6c. Explain the concept of vectored interrupt.

Sol: A device requesting an interrupt can identify itself by sending special code to the processor over the bus. The code supplied by the device represents the starting address of the interrupt service routine. The code length is 4 to 8 bits. The processor reads this address called the interrupt vector and stores it in to the PC. The interrupt vector may also include a new value for a processor status register.

The interrupted device must wait to put on the bus only when the processor is ready to receive it. When the processor is ready to receive the vector interrupt code, it activates the interrupt acknowledge line INTA. The I/O device responds by sending its interrupt vector code and turning off INTR signal.

Module 4

7(a) Internal organization of a 2M X 8 dynamic Memory chip.



DESCRIPTION:

-
- The 4 bit cells in each row are divided into 512 groups of 8.

  21 bit address is needed to access a byte in the memory(12 bit → To select a row,9 bit → Specify the group of 8 bits in the selected row).

  A_{8-0} → Row address of a byte.

  A_{20-9} → Column address of a byte.

- During Read/ Write operation ,the row address is applied first. It is loaded into the row address latch in response to a signal pulse on **Row Address Strobe(RAS)** input of the chip.

- When a Read operation is initiated, all cells on the selected row are read and refreshed.

- Shortly after the row address is loaded,the column address is applied to the address pins & loaded into **Column Address Strobe(CAS).**

- The information in this latch is decoded and the appropriate group of 8 Sense/Write circuits are selected.

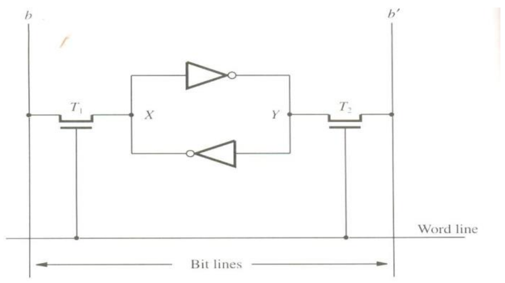
- R/W =1(read operation) → The output values of the selected circuits are transferred to the data lines D0 - D7.

- R/W =0(write operation) → The information on D0 - D7 are transferred to the selected circuits.

(b) **Static Memories**:

Memories that consists of circuits capable of retaining their state as long as power is applied are known as **static memory.**

Fig:Static RAM cell



-
-
-

Two inverters are cross connected to form a batch

The batch is connected to two bit lines by transistors T_1 and T_2.

These transistors act as switches that can be opened / closed under the control of the word line.

- When the wordline is at ground level, the transistors are turned off and the latch retain its state.

Read Operation:

-

In order to read the state of the SRAM cell, the word line is activated to close switches T_1 and T_2.

- If the cell is in state 1, the signal on bit line b is high and the signal on the bit line b is low.Thus b and b are complement of each other.

- Sense / write circuit at the end of the bit line monitors the state of b and b‟ and set the output accordingly.

Write Operation:

- The state of the cell is set by placing the appropriate value on bit line b and its complement on b and then activating the word line. This forces the cell into the corresponding state.

- The required signal on the bit lines are generated by Sense / Write circuit.

8(b) **READ ONLY MEMORY:**

- Both SRAM and DRAM chips are volatile,which means that they lose the stored information if power is turned off.

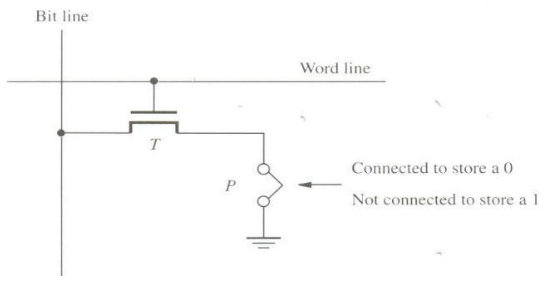
- Many application requires Non-volatile memory (which retain the stored information if power is turned off).

-
- Eg:Operating System software has to be loaded from disk to memory which requires the program that boots the Operating System ie. It requires non-volatile memory.

- Non-volatile memory is used in embedded system.

  Since the normal operation involves only reading of stored data ,a memory of this type is called ROM.

Fig:ROM cell



At Logic value '0' → Transistor(T) is connected to the ground point(P).

Transistor switch is closed & voltage on bitline nearly drops to zero.

At Logic value '1' → Transistor switch is open.

The bitline remains at high voltage.

-
- To read the state of the cell,the word line is activated.

A Sense circuit at the end of the bitline generates the proper output value.

Types of ROM: Different types of non-volatile memory are,

- PROM
- EPROM
- EEPROM
- Flash Memory

PROM:-Programmable ROM:

PROM allows the data to be loaded by the user.

Programmability is achieved by inserting a „fuse‟ at point P in a ROM cell. Before it is programmed, the memory contains all 0‟s

The user can insert 1‟s at the required location by burning out the fuse at these locations using high-current pulse.This process is irreversible.

EPROM:-Erasable reprogrammable ROM:

EPROM allows the stored data to be erased and new data to be loaded.

- In an EPROM cell, a connection to ground is always made at „P" and a special transistor is used, which has the ability to function either as a normal transistor or as a disabled transistor that is always turned „off".
- This transistor can be programmed to behave as a permanently open switch, by injecting charge into it that becomes trapped inside.
- Erasure requires dissipating the charges trapped in the transistor of memory cells. This can be done by exposing the chip to ultra-violet light, so that EPROM chips are mounted in packages that have transparent windows.

Merits:

- It provides flexibility during the development phase of digital system.
- It is capable of retaining the stored information for a long time.

Demerits:

- The chip must be physically removed from the circuit for reprogramming and its entire contents are erased by UV light.

EEPROM:-Electrically Erasable ROM:

Merits:

- It can be both programmed and erased electrically.
- It allows the erasing of all cell contents selectively.

Demerits:

- It requires different voltage for erasing ,writing and reading the stored data.

Flash Memory:

-

- In EEPROM, it is possible to read & write the contents of a single cell.

- In Flash device, it is possible to read the contents of a single cell but it is only possible to write the entire contents of a block.

- Prior to writing,the previous contents of the block are erased.

- Eg.In MP3 player,the flash memory stores the data that represents sound.

- Single flash chips cannot provide sufficient storage capacity for embedded system application.

- There are 2 methods for implementing larger memory modules consisting of number of chips.They are,
  - Flash Cards
  - Flash Drives.

RAS and CAS are active low so that they cause the latching of address when they change from high to low. This is because they are indicated by RAS & CAS.

To ensure that the contents of a DRAM „s are maintained, each row of cells must be accessed periodically.

Refresh operation usually perform this function automatically.

A specialized memory controller circuit provides the necessary control signals RAS & CAS, that govern the timing.

The processor must take into account the delay in the response of the memory.

Such memories are referred to as **Asynchronous DRAM's.**

Fast Page Mode:

Transferring the bytes in sequential order is achieved by applying the consecutive sequence of column address under the control of successive CAS signals.

This scheme allows transferring a block of data at a faster rate. The block of transfer capability is called as **Fast Page Mode.**

8© SECONDARY STORAGE:

- The Semi-conductor memories donot provide all the storage capability.
- The Secondary storage devices provide larger storage requirements.
- Some of the Secondary Storage devices are,
  - Magnetic Disk
  - Optical Disk
  - Magnetic Tapes.

Magnetic Disk:

Magnetic Disk system consists o one or more disk mounted on a common spindle.

A thin magnetic film is deposited on each disk, usually on both sides.

The disk are placed in a rotary drive so that the magnetized surfaces move in close proximity to read /write heads.

Each head consists of **magnetic yoke & magnetizing coil**.

Digital information can be stored on the magnetic film by applying the current pulse of suitable polarity to the magnetizing coil.

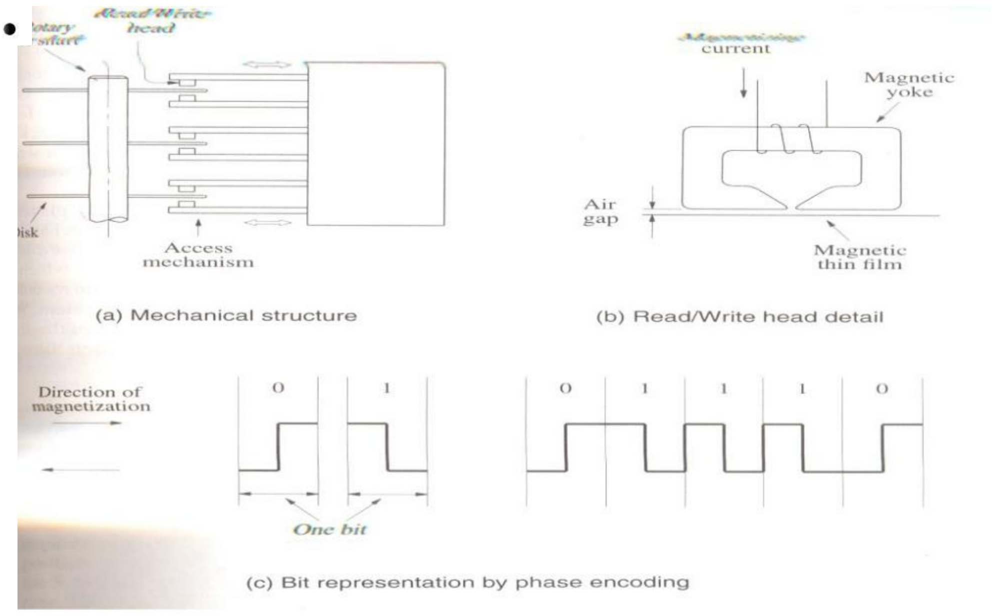
Only changes in the magnetic field under the head can be sensed during the Read operation. Therefore if the binary states 0 & 1 are represented by two opposite states of magnetization, a voltage is induced in the head only at 0-1 and at 1-0 transition in the bit stream.

- A consecutive (long string) of 0"s & 1"s are determined by using the clock which is mainly used for synchronization.
- Phase Encoding or Manchester Encoding is the technique to combine the clocking information with data.
- The Manchester Encoding describes that how the self-clocking scheme is implemented.

The Read/Write heads must be maintained at a very small distance from the moving disk surfaces in order to achieve high bit densities.

-

When the disk are moving at their steady state, the air pressure develops between the disk surfaces & the head & it forces the head away from the surface.

- 

-

The flexible spring connection between head and its arm mounting permits the head to fly at the desired distance away from the surface

Disk Controller

The disk controller acts as interface between disk drive and system bus.

The disk controller uses DMA scheme to transfer data between disk and main memory. When the OS initiates the transfer by issuing Read/Write request, the controllers register will load the following information. They are,

Main memory address(address of first main memory location of the block of words involved in the transfer)

Disk address(The location of the sector containing the beginning of the desired block of words)

(number of words in the block to be transferred).

<u>Module 5</u>

9(b) Explain Hardwired control unit organisation with the help of necessary diagrams

The control units use fixed logic circuits to interpret instructions and generate control signals from them.

The fixed logic circuit block includes combinational circuit that generates the required control outputs for decoding and encoding functions.

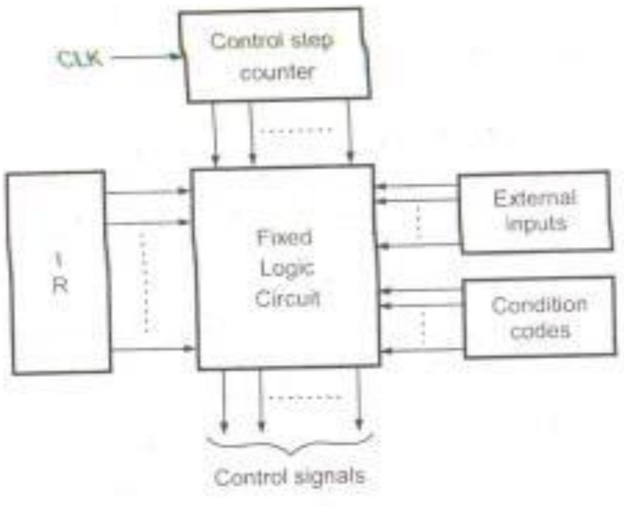


Fig. 3.10 Typical hardwired control unit

decoder decodes the instruction loaded in the

Instruction decoder

It decodes the instruction loaded in the IR.

If IR is an 8 bit register then instruction decoder generates 2^8(256 lines); one for each instruction.

According to code in the IR, only one line amongst all output lines of decoder goes high (set to 1 and all other lines are set to 0).

Step decoder It provides a separate signal line for each step, or time slot, in a control sequence.

Encoder

It gets in the input from instruction decoder, step decoder, external inputs and condition codes.

It uses all these inputs to generate the individual control signals.

After execution of each instruction end signal is generated this resets control step counter and make it ready for generation of control step for next instruction.

The encoder circuit implements the following logic function to generate Y_{in}

$Y_{in} = T_1 + T_5 . Add + T . BRANCH+...$

The Y_{in} signal is asserted during time interval T_1 for all instructions, during T_5 for an ADD instruction, during T_4 for an unconditional branch instruction, and so on.

As another example, the logic function to generate Z_{out} signal can given by

$Z_{out} = T_2 + T_7 . ADD + T_6 . BRANCH +....$

The Z_{out} signal is asserted during time interval T_2 of all instructions, during T_7 for an ADD instruction, during T_6 for an unconditional branch instruction, and so on.

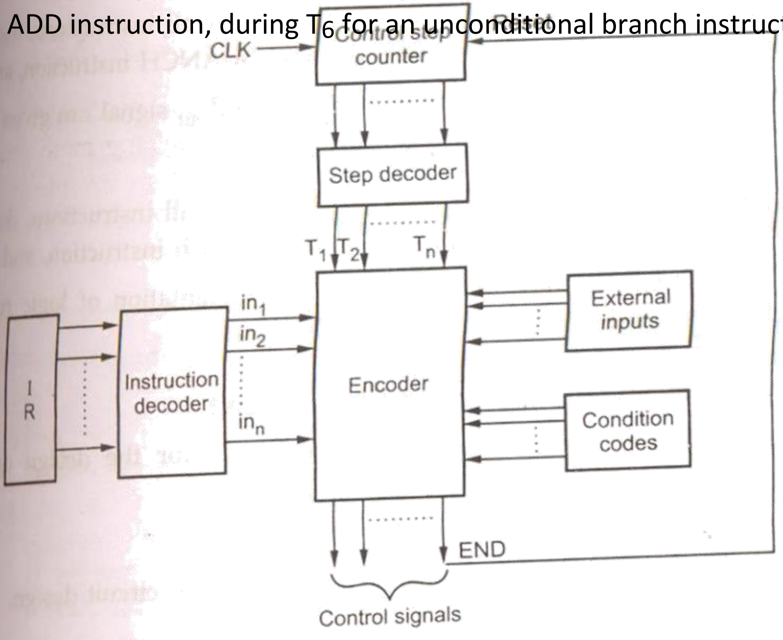
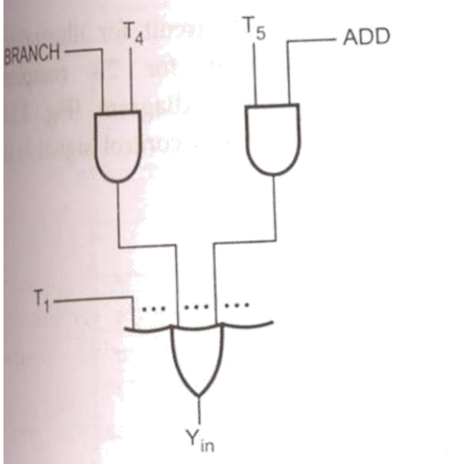


Fig. 3.11 Detail block diagram for hardwired control unit

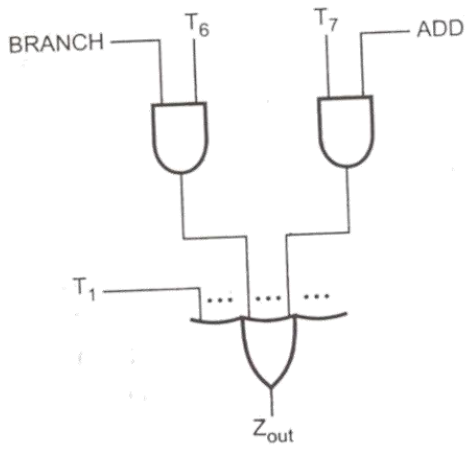
Let us see how the encoder generates signal for single bus processor organisation wn in Fig. 3.12 Y_{in}. The encoder circuit implements the following logic function to erate Y_{in}.

$$Y_{in} = T_1 + T_5 \cdot ADD + T_4 \cdot BRANCH + \ldots$$



Generation of the Y_{in} control signal

Fig. 3.12



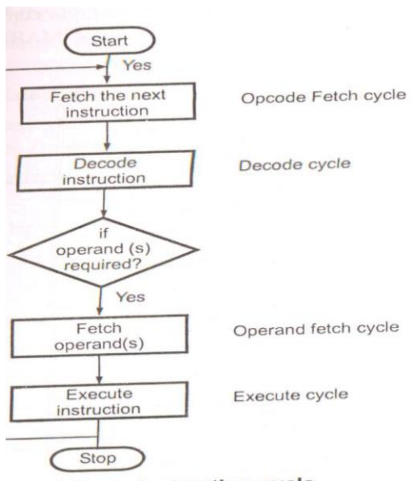
Generation of the Z_{out} control signal

Fig. 3.13

•

10(b) The primary function of a processor unit is to execute sequence of instructions stored in a memory, which is external to the processor unit.The sequence of operations involved in processing an instruction constitutes an instruction cycle,which can be subdivided into 3 major phases:

1. Fetch cycle
2. Decode cycle
3. Execute cycle



g. 3.1 Basic instruction cycle

•

To perform fetch, decode and execute cycles the processor unit has to perform set of operations called micro-operations.

Single bus organization of processor unit shows how the building blocks of processor unit are organised and how they are interconnected.

They can be organised in a variety of ways, in which the arithmetic and logic unit and all processor registers are connected through a single common bus.

It also shows the external memory bus connected to memory address(MAR) and data register(MDR).