CBCS SCHEME

USN | | | | | | | | | | |

BEC306C

Third Semester B.E./B.Tech. Degree Examination, Dec.2024/Jan.2025
Computer Organization and Architecture

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
Q.1	a.	With neat diagram explain connection between the processor and memory.	10	L1	CO1
✓	b.	Write the difference between little endian and big endian memory assignments.	05	L1	CO1
	c.	Write a short note on basic performance equation.	05	L1	CO1
		OR			
Q.2	a.	Describe the concept of branching with an example program of instruction execution.	10	L1	CO1
	b.	Represent the following decimal values as signed 7-bit numbers using sign and magnitude, signed 1's complement and signed 2's complement formats. – 55, +51, 8, – 27, – 39, +43, – 10, 62	05	L2	CO1
	c.	Write a short note on memory operations.	05	L1	CO1
		Module – 2			
Q.3	a.	What is an addressing mode? Explain any four types of addressing modes, with suitable example.	10	L1	CO2
	b.	Write a program to compute the sum of test scores of all the students in the three tests. Store the corresponding sums in memory.	10	L2	CO2
		OR			
Q.4	a.	Explain the Rotate and Shift instructions with an example.	10	L1	CO2
✓	b.	Define subroutine. Explain subroutine linkage using a link register.	05	L1	CO2
	c.	What are assembler directives? Explain any two directives.	05	L1	CO2
		Module – 3			
Q.5	a.	Define I/O interface? Explain I/O interface to connect an input device to the bus with neat diagram.	10	L1	CO3
	b.	What is interrupt? Discuss interrupt I/O method for data transfer.	05	L1	CO3
	c.	Describe two methods of handling multiple devices.	05	L1	CO3
		OR			
Q.6	a.	Explain the use of DMA controllers in a computer system with neat diagram.	10	L1	CO3
	b.	Write a note on Bus Arbitration.	10	L1	CO3
		Module – 4			
Q.7	a.	Explain the organization of 1K×1 memory chip.	10	L1	CO4
	b.	Write a note on : (i) Static memories (ii) Cache memory	10	L1	CO4
		OR			
Q.8	a.	Explain the Magnetic disk principles.	10	L1	CO4
✓	b.	Draw and explain the internal organization of 2M×8 asynchronous DRAM chip.	10	L2	CO4
		Module – 5			
Q.9 ✓	a.	Discuss with neat diagram the single bus organization of data path inside a processor.	10	L1	CO5
	b.	What are the actions required to execute a complete instruction ADD (R_2), R_1	10	L1	CO5
		OR			
Q.10	a.	Draw and explain multiple bus organization of CPU.	10	L1	CO5
	b.	Draw and explain organization of the control unit to allow conditional branching in the microprogram.	10	L1	CO5

* * * * *

Q.1:

a) With a neat diagram, explain the connection between the Processor and memory

Sol:

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.

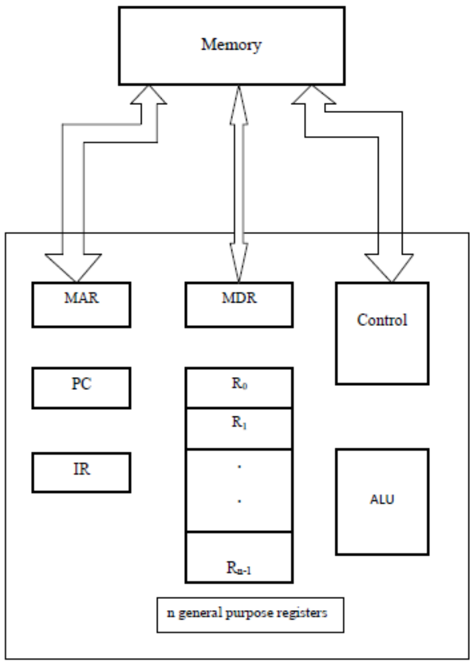


Fig 1.2 Connections between the processor and memory

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.

There are basically three interfacing buses between the processor and Memory namely:

a) Address Bus
b) Data Bus
c) Control Bus

Address bus is used to carry the address information from the Processor to the Memory Chip. It is basically a unidirectional bus that is connected to the Memory Address Register (MAR) in the Processor. In case the process requires to read or write the data into Memory, the address is loaded into the MAR register, which is connected to the Address Bus. Thus any address that is loaded into the MAR register is carried forward to the Memory Chip through the Address Bus.

Control Bus: The processor further intimates the memory chip about the nature of operation associated by using the Control Bus. Processor conveys whether a READ or WRITE operation is being processed to the memory chip using the control bus. Control Bus is bi-directional in nature since, this bus is used to exchange all the control information between the Processor and Memory to facilitate proper handshaking required to execute the memory access by the Processor.

Data Bus: Data Bus carries the data information between the processor and Memory. In case of a read operation being performed by the processor, the memory places the data from the addressed location on to the Data bus, which transfers the same to the Processor's Memory Data Register (MDR). Simiularly, in case if the processor intends to write the data into the memory, data is loaded on to the Data bus from the Memory Data Register (MDR) present in the processor. Hence Data bus is bi-directional in nature.

b) Write the Difference between Little Endian and Big Endian memory assignments

Sol:

<u>Big-Endian and Little-Endian Assignments</u>

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.

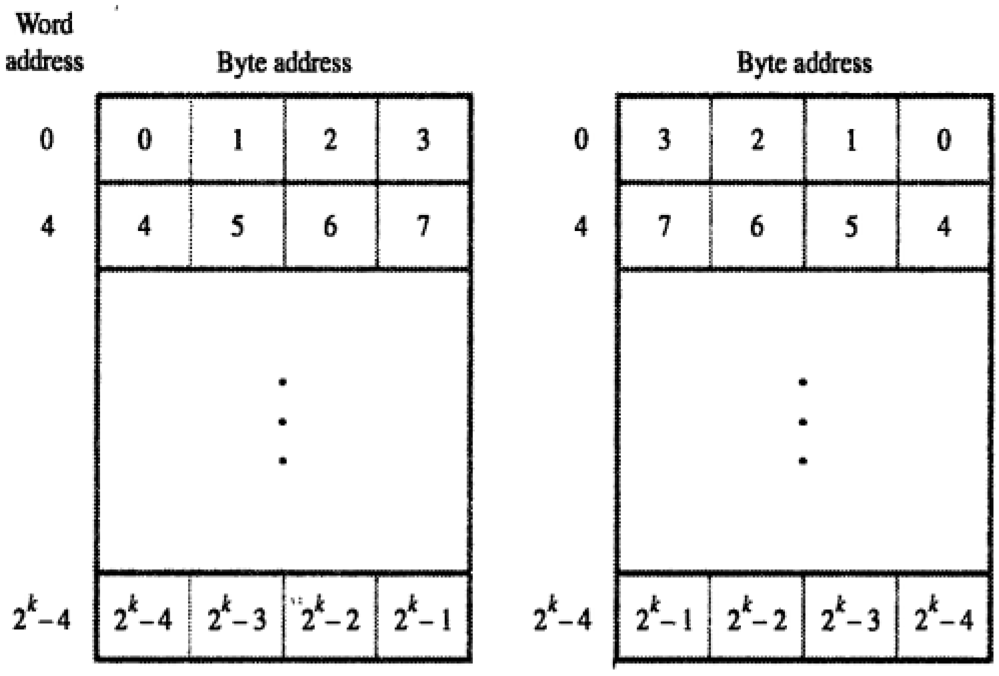


Fig 1.11: a) Big-endian assignment b) Little-endian assignment

c) Short Note on Basic Performance Equation:

Sol:

Basic Performance Equation

Let T be the processor time required to execute a program that has been prepared by some high level language. The compiler generates machine level object program that corresponds to source program. Assume that complete execution of the program requires the execution of N machine language instructions. Suppose that the average number of basic steps needed to execute one machine instruction is S, where each basic step is completed in one clock cycle. If the clock rate is R cycles per second, the program execution time is given by *basic performance equation.*

$$T = \frac{N \times S}{R}$$

To achieve high performance, the value of T must be reduced which can be done by reducing N and S, and increasing R. The value of N is reduced if the source program is compiled in fewer machine instructions. The value of S is reduced if instructions have a smaller number of basic steps to perform or if the execution of instructions are overlapped.

Using a higher-frequency clock increases the value of R which means the time required to complete a basic execution step is reduced.

Q-2:

a) Describe the concept of branching with an example program of instruction execution

Sol:

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.

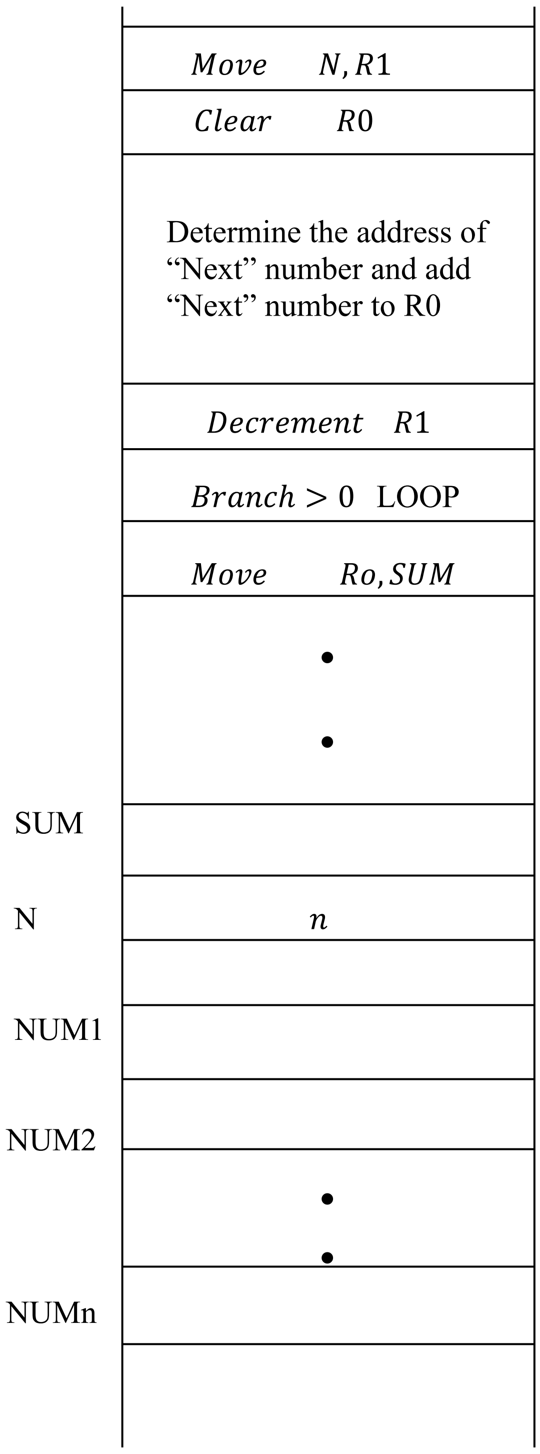


Fig 1.13: Using a loop to add n numbers

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.

b) Represent the following decimal values as signed 7-bit numbers using Sign and magnitude, signed 1’s compliment and signed 2’d complimentformats:
-55, +51, 8, -27, -39, +43, -10, 62

Sol:

Sign and Magnitude Representation

1. -55

- **Sign Bit** = 1 (Negative)
- **Magnitude** = 55 in binary → 0110111
- **Sign and Magnitude Representation** = **10110111**

---

2. +51

- **Sign Bit** = 0 (Positive)
- **Magnitude** = 51 in binary → 0110011
- **Sign and Magnitude Representation** = **00110011**

---

3. +8

- **Sign Bit** = 0 (Positive)
- **Magnitude** = 8 in binary → 0001000
- **Sign and Magnitude Representation** = **00001000**

---

4. -27

- **Sign Bit** = 1 (Negative)
- **Magnitude** = 27 in binary → 0011011
- **Sign and Magnitude Representation** = **1011011**

---

5. -39

- **Sign Bit** = 1 (Negative)
- **Magnitude** = 39 in binary → 0100111
- **Sign and Magnitude Representation** = **10100111**

---

6. +43

- **Sign Bit** = 0 (Positive)
- **Magnitude** = 43 in binary → 0101011
- **Sign and Magnitude Representation** = **00101011**

---

7. -10

- **Sign Bit** = 1 (Negative)
- **Magnitude** = 10 in binary → 0001010
- **Sign and Magnitude Representation** = **10001010**

---

8. +62

- **Sign Bit** = 0 (Positive)
- **Magnitude** = 62 in binary → 0111110
- **Sign and Magnitude Representation** = **00111110**

c) Gesg
d)

signed 1’s compliment

1. -55

- **Magnitude: 55 → 0110111**
- **1's Complement: Invert all bits → 1001000**
- **1's Complement Representation = 11001000**

---

2. +51

- **Magnitude: 51 → 0110011**
- **Positive Number (No change) → 0110011**
- **1's Complement Representation = 00110011**

---

3. +8

- **Magnitude: 8 → 0001000**
- **Positive Number (No change) → 0001000**
- **1's Complement Representation = 00001000**

---

4. -27

- **Magnitude: 27 → 0011011**
- **1's Complement: Invert all bits → 1100100**
- **1's Complement Representation = 11100100**

---

5. -39

- **Magnitude: 39 → 0100111**
- **1's Complement: Invert all bits → 1011000**
- **1's Complement Representation = 11011000**

---

6. +43

- **Magnitude: 43 → 0101011**
- **Positive Number (No change) → 0101011**
- **1's Complement Representation = 00101011**

---

7. -10

- **Magnitude: 10 → 0001010**
- **1's Complement: Invert all bits → 1110101**
- **1's Complement Representation = 11110101**

---

8. +62

- **Magnitude: 62 → 0111110**
- **Positive Number (No change) → 0111110**
- **1's Complement Representation = 00111110**

2’s Compliment Representation::

1. -55

- **Magnitude: 55 → 0110111**
- **1's Complement: Invert all bits → 1001000**
- **Add 1 → 1001000 + 1 = 1001001**
- **2's Complement Representation = 1001001**

---

2. +51

- **Magnitude: 51 → 0110011**
- **Positive Number (No change) → 0110011**
- **2's Complement Representation = 0110011**

---

3. +8

- **Magnitude: 8 → 0001000**
- **Positive Number (No change) → 0001000**
- **2's Complement Representation = 0001000**

---

4. -27

- **Magnitude: 27 → 0011011**
- **1's Complement: Invert all bits → 1100100**
- **Add 1 → 1100100 + 1 = 1100101**
- **2's Complement Representation = 1100101**

---

5. -39

- **Magnitude: 39 → 0100111**
- **1's Complement: Invert all bits → 1011000**
- **Add 1 → 1011000 + 1 = 1011001**
- **2's Complement Representation = 1011001**

---

6. +43

- **Magnitude: 43 → 0101011**

- **Positive Number (No change) → 0101011**
- **2's Complement Representation = 0101011**

---

7. -10

- **Magnitude: 10 → 0001010**
- **1's Complement: Invert all bits → 1110101**
- **Add 1 → 1110101 + 1 = 1110110**
- **2's Complement Representation = 1110110**

---

8. +62

- **Magnitude: 62 → 0111110**
- **Positive Number (No change) → 0111110**
- **2's Complement Representation = 0111110**

c. Write a short note on memory Operations

Sol:

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.

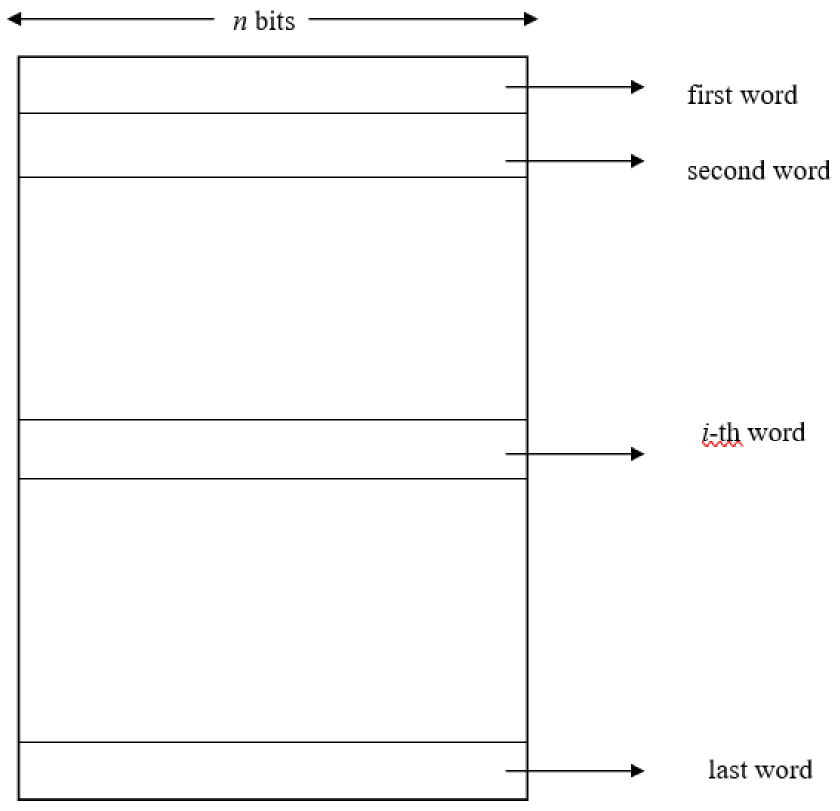


Fig 1.10: Memory words

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.

Memory Operations

Both program instructions and data operands are stored in the memory. To execute an instruction, the processor control circuits must cause the word (or words) containing the instructions to be transferred from the memory to the processor. Operands and results must also be moved between memory and processor. Thus the two basic operations involving memory are needed, namely, *Load* (or Read or Fetch) and *Store* (or Write).

The Load operation transfers a copy of the contents of a specified memory location to the processor. The memory content remains unchanged. The Store operation transfers an item of information from the processor to a specific memory location destroying the contents of that location.

Q3.

a. **What is an addressing mode? Explain any four Addressing modes with suitable examples.**

Addressing Modes

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.

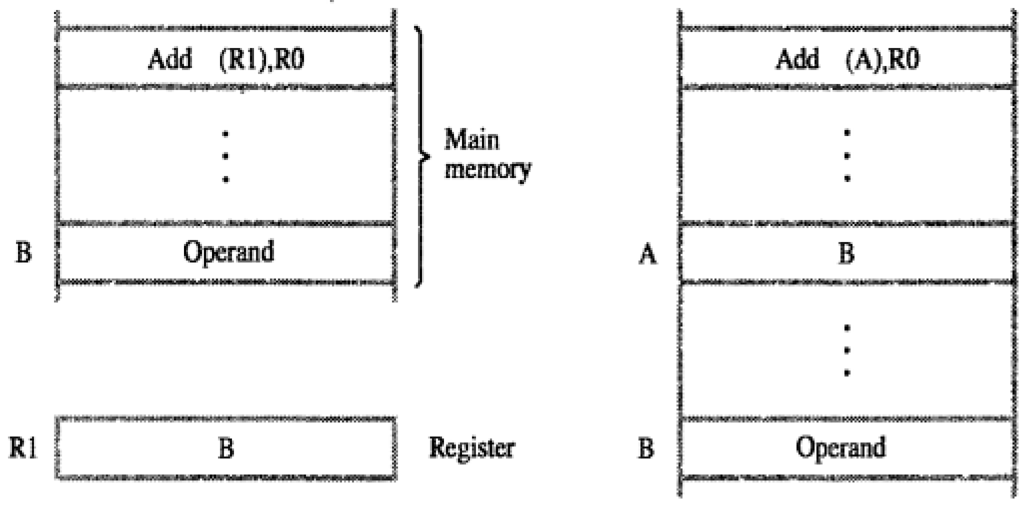
$$\begin{array}{l} Move \;\; B, R1 \\ Add \;\; \#6, R1 \\ Move \;\; R1, A \end{array}$$

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.*

<u>Indexing and Arrays</u>

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.

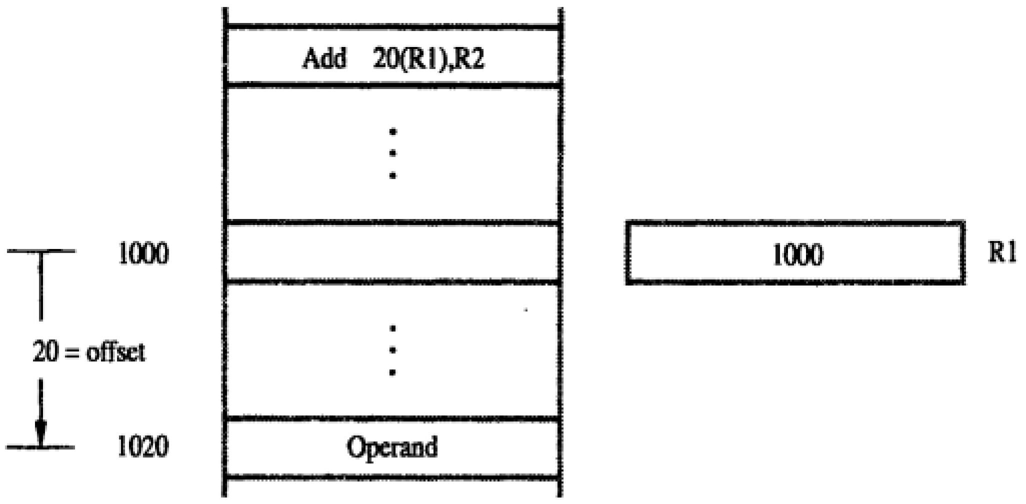


Fig 2.2 a: Offset is given as a constant

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.

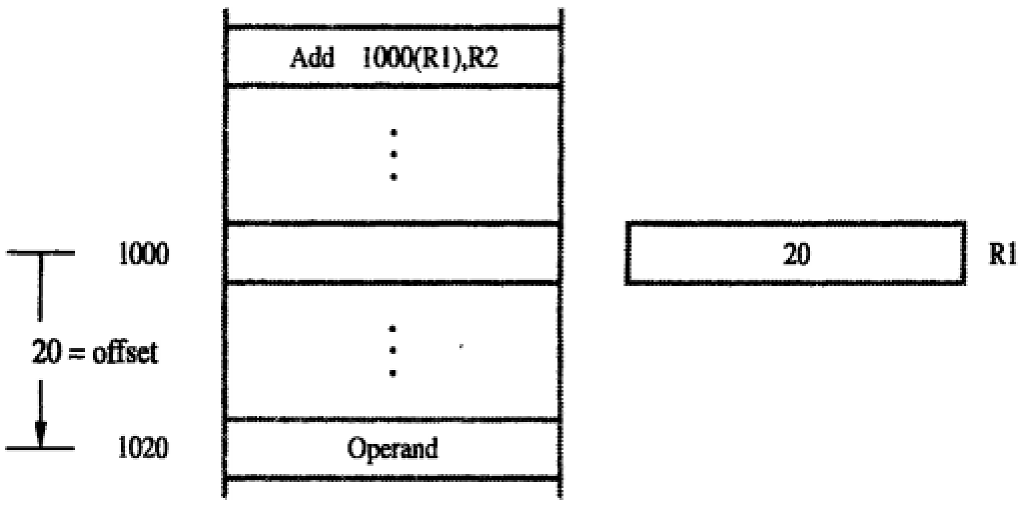


Fig 2.2b: Offset is in the register

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.

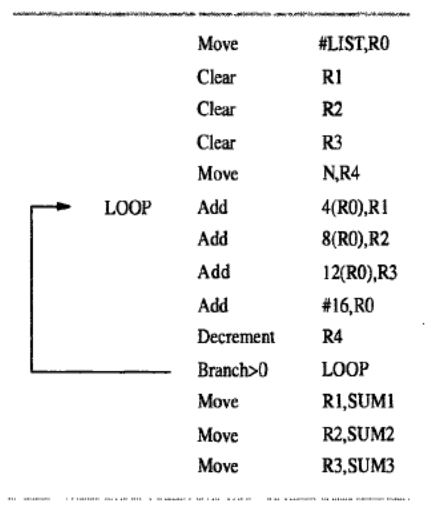
Auto-decrement mode – The content of a register specified in the instruction is first automatically decremented and then used as effective address of the operand. In this mode, operands are accessed in descending address order.

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

b. Write a Program to compute the sum of test scores of all the students in the three tests. Store the corresponding sums in memory.

Sol:

Suppose that we wish to compute the sum of all scores obtained on each of the tests and store these three sums in memory locations SUMI, SUM2, and SUM3.



```
            Move        #LIST,R0
            Clear       R1
            Clear       R2
            Clear       R3
            Move        N,R4
LOOP        Add         4(R0),R1
            Add         8(R0),R2
            Add         12(R0),R3
            Add         #16,R0
            Decrement   R4
            Branch>0    LOOP
            Move        R1,SUM1
            Move        R2,SUM2
            Move        R3,SUM3
```

Q.4:

a. Explain the Rotate and shift instructions with an example.

Sol:

Shift and Rotate Instructions

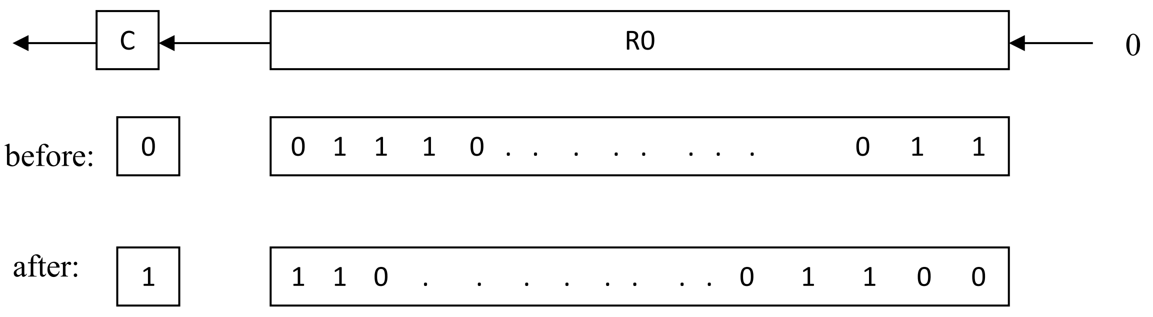
There are applications that require bits of an operand to be shifted to the right or left some specified number of bit positions. For general operands we use a logical shift. For a number we use an arithmetic shift which preserves the sign of the number.

Logical Shifts

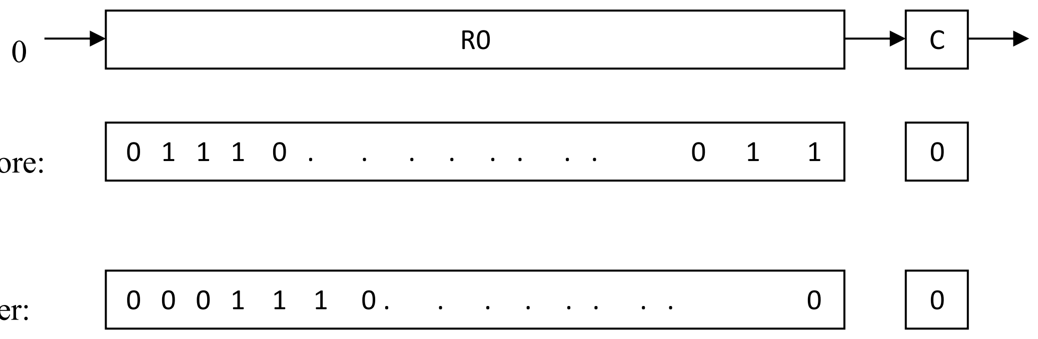
Two logical shift instructions are needed, one for shifting left (LShiftL) and another for shifting right (LShiftR). These instructions shift an operand over a number of bit positions specified in a count operand contained in the instruction. The general form of logical left shift instruction is

```
LShiftL     count, dst
```

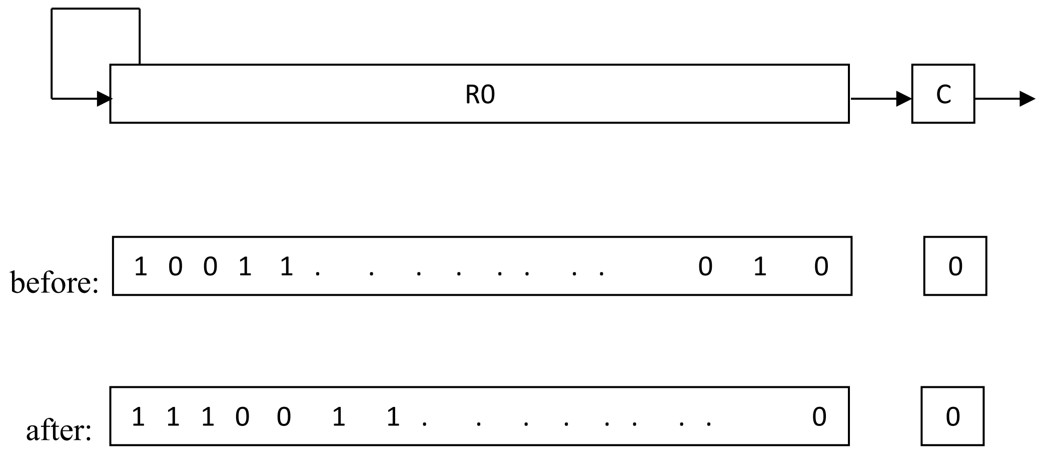
The count operand may be given as an immediate operand or it may be contained in the processor register. Vacated positions are filled with zeros, and the bits shifted out are passed through the Carry flag C, and then dropped. Involving the C flag in shifts is useful in arithmetic operations on large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



(a) Logical shift Left LShiftL #2, R0



(b) Logical shift right LShiftR #2, R0

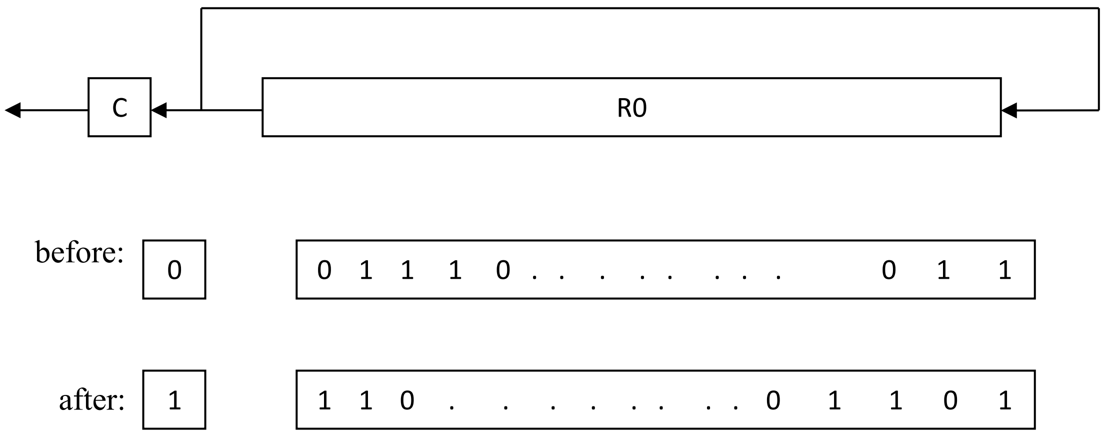


(c) Arithmetic shift right AShiftR #2, R0

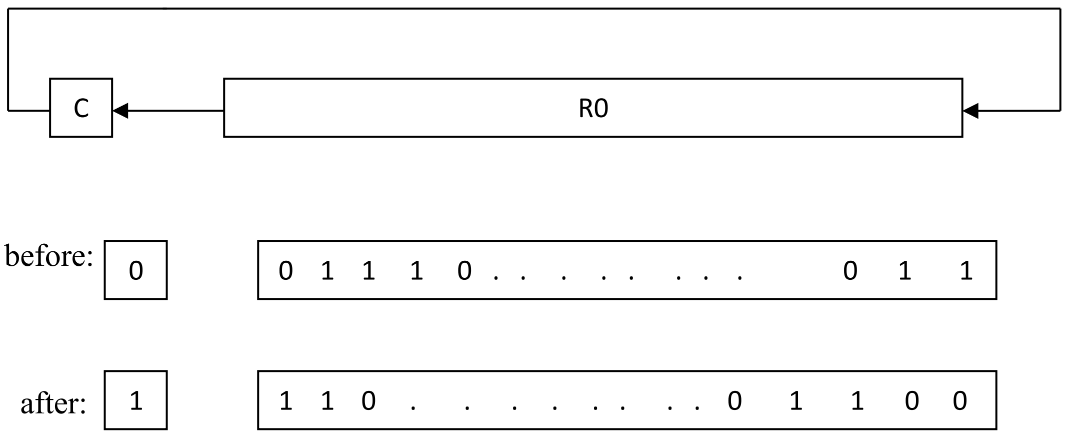
Fig 2.10: Logical and arithmetic shift instructions

Rotate Operations

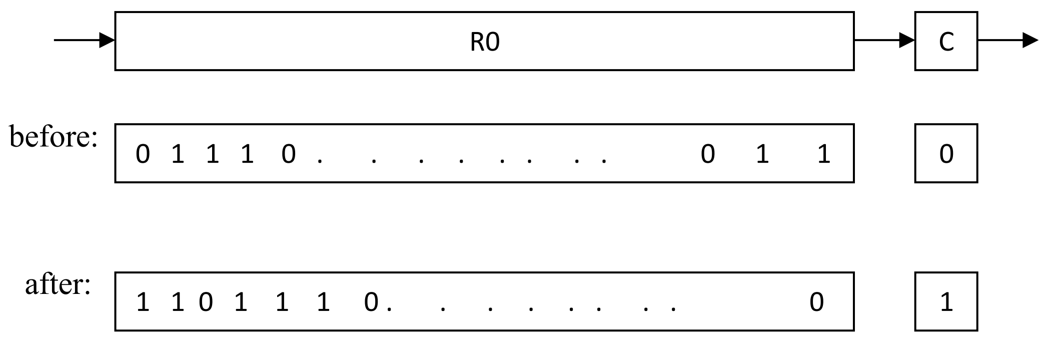
To preserve all the bits, a set of rotate can be used. They move the bits that are shifted out of one end of the operand back in to the other end. Two versions of both left and right rotate instructions are provided. In one version, bits are of the operand are simply rotated. In the other version, the rotation includes the C flag. Figure 2.11 shows the left and right rotate operations with and without C flag being include in the rotation. Note that when C flag is not included in the rotation, it still retains the last bit shifted out of the end of the register.



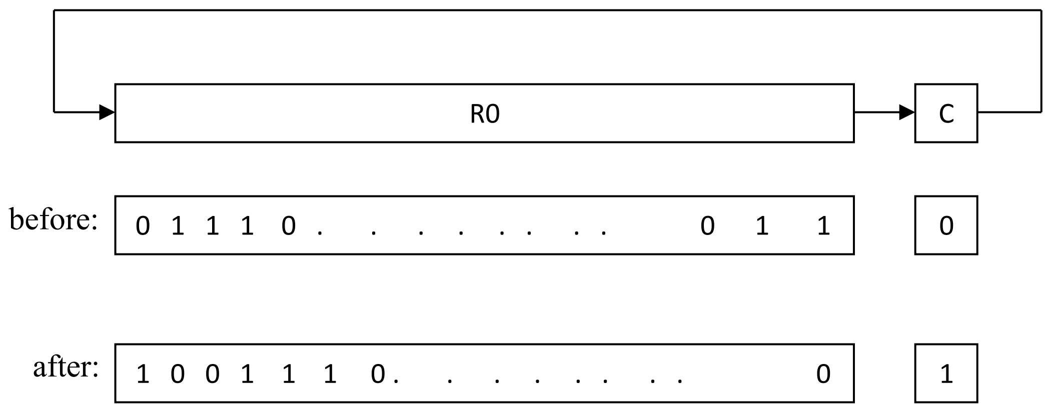
(a) Rotate left without carry RotateL #2, R0



(b) Rotate left with carry RotateLC #2, R0



(c) Rotate right without carry RotateR #2, R0



(d) Rotate right with carry RotateRC #2, R0

Fig 2.11: Rotate instructions

Q.4:

b. Define Subroutine. Explain subroutine linkage using a Link register

Sol:

<u>Subroutines</u>

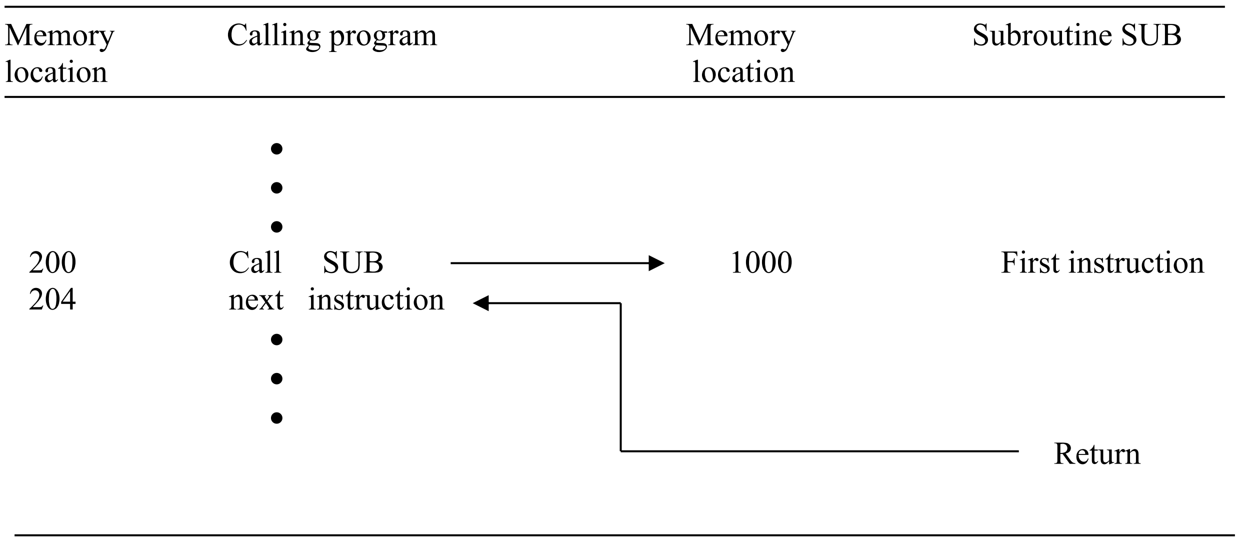
It is often necessary to perform a particular subtask many times on different data values. Such subtask is called *subroutine.* When a program branches to a subroutine we call that it is *calling* a subroutine. The instruction that performs this branch operation is called a Call instruction. The subroutine 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.



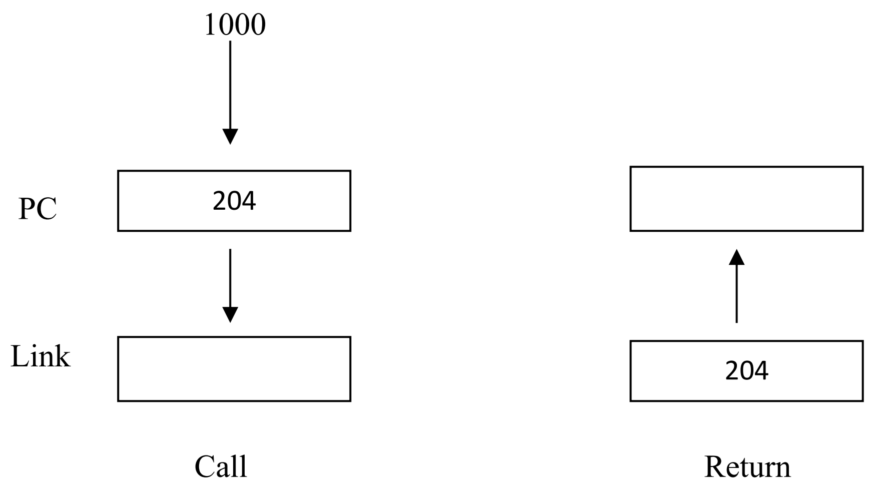


Fig 2.6: Subroutine linkage using a link register

c. What are Assembly Directives? Explain any two directives.
Sol:

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.

5. a) Define I/O interface. Explain I/O interface to connect an input device to the bus with neat diagram.

A simple arrangement to connect I/O devices to a computer is to use single bus arrangement as shown in Fig1. The bus enables the devices connected to it to exchange information. It consists of three set of lines to carry address, data and control signals. Each I/O device is assigned unique set of addresses.

Memory-mapped I/O: When I/O devices and the memory share the same address space, the arrangement is called *memory-mapped I/O.*

With the memory mapped I/O any machine instruction that can access memory can be used to transfer data to or from an I/O device.

Move DATAIN, R0

Reads data from the DATAIN and stores into processor register R0. Similarly

Move R0, DATAOUT

Sends the contents of register R0 to location DATAOUT which is the output data buffer of a display unit or a printer.

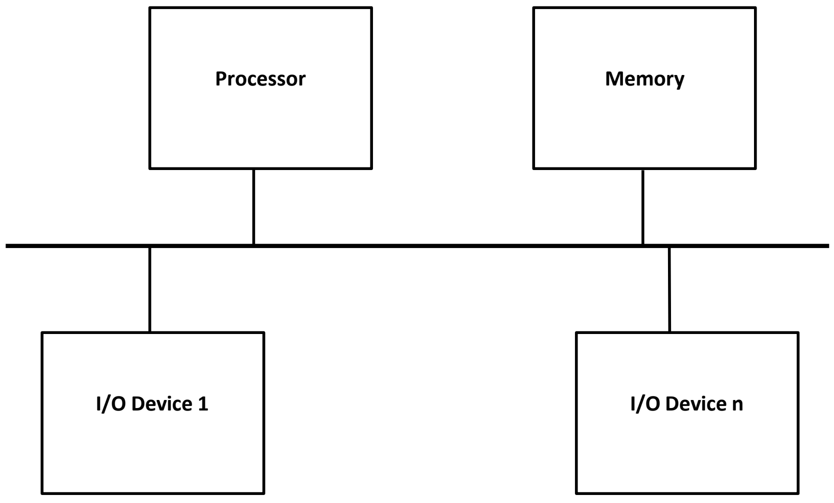


Fig1: Simple bus structure

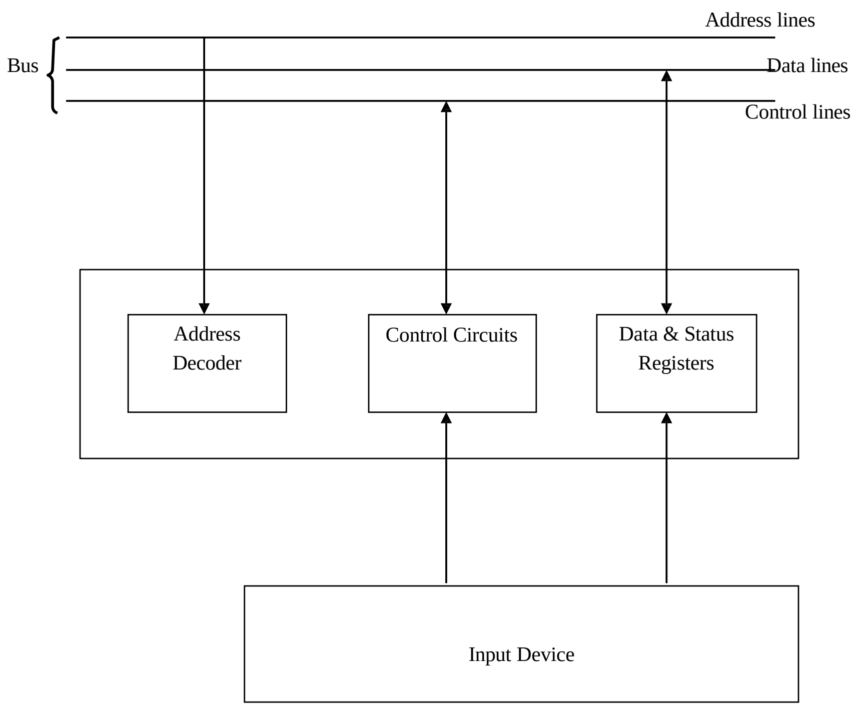


Fig 2: I/O interface of an input device

Fig 2 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.

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

5. b) What is Interrupt? Discuss interrupt I/O method for data transfer

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.

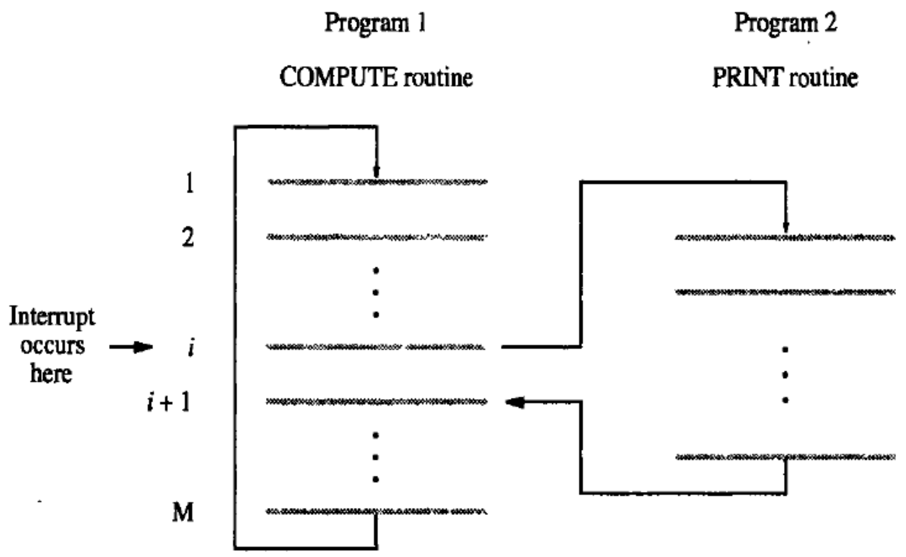


Fig 4: Transfer of control through the use of interrupts

It is possible to overlap printing and computation i.e. 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.

The routine executed in response to an interrupt request is called the **interrupt-service routine**, which is the PRINT routine in the above example. Interrupts bear considerable resemblance to subroutine calls.

i. Assume that an interrupt request arrives during execution of instruction i. (refer Fig. 4).
ii. Then, it loads the program counter with the address of the first instruction of the interrupt-service routine. Let us assume that this address is hardwired in the processor.
iii. After execution of the interrupt-service routine, the processor has to come back to instruction i +1.
iv. Therefore, when an interrupt occurs, the current contents of the PC, which point to instruction i + 1, must be put in temporary storage in a known location.
v. A Return from interrupt instruction at the end of the interrupt-service routine reloads the PC from that temporary storage location, causing execution to resume at instruction i +1.
vi. In many processors, the return address is saved on the processor stack.
Alternatively, it may be saved in a special location, such as a register provided for this purpose

5. C) Describe two methods of handling multiple devices
Handling Multiple devices:

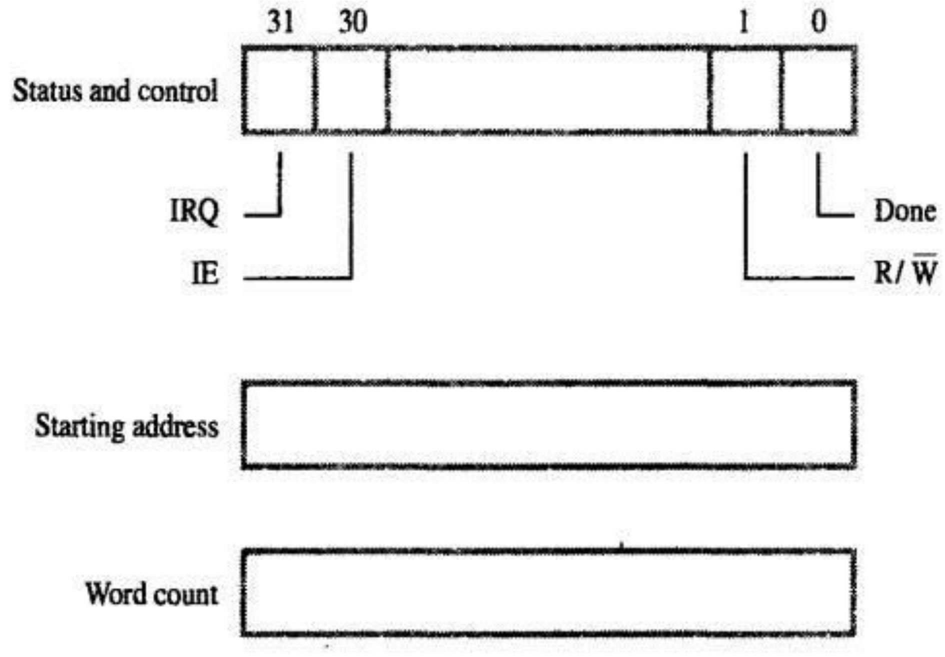
The information needed to determine whether a device is requesting an interrupt is available in its status register. When a device raises an interrupt request, one of the bits of the status register is set to 1 which we call IRQ bit.
e.g. KIRQ, DIRQ are the interrupt request bits for keyboard and display. The simplest way to identify the interrupting device is to have the interrupt-service routine poll all I/O devices connected to the bus. The polling scheme has 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.

Vectored Interrupts:
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 into the PC. This arrangement implies that the interrupt-service routine for a given device must always start at the same location. The interrupt vector may also include a new value for a processor status register.
The interrupted device must wait to put data 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 the INTR signal.

6. **a) Explain the use of DMA controllers in a computer system with neat diagram.**

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.

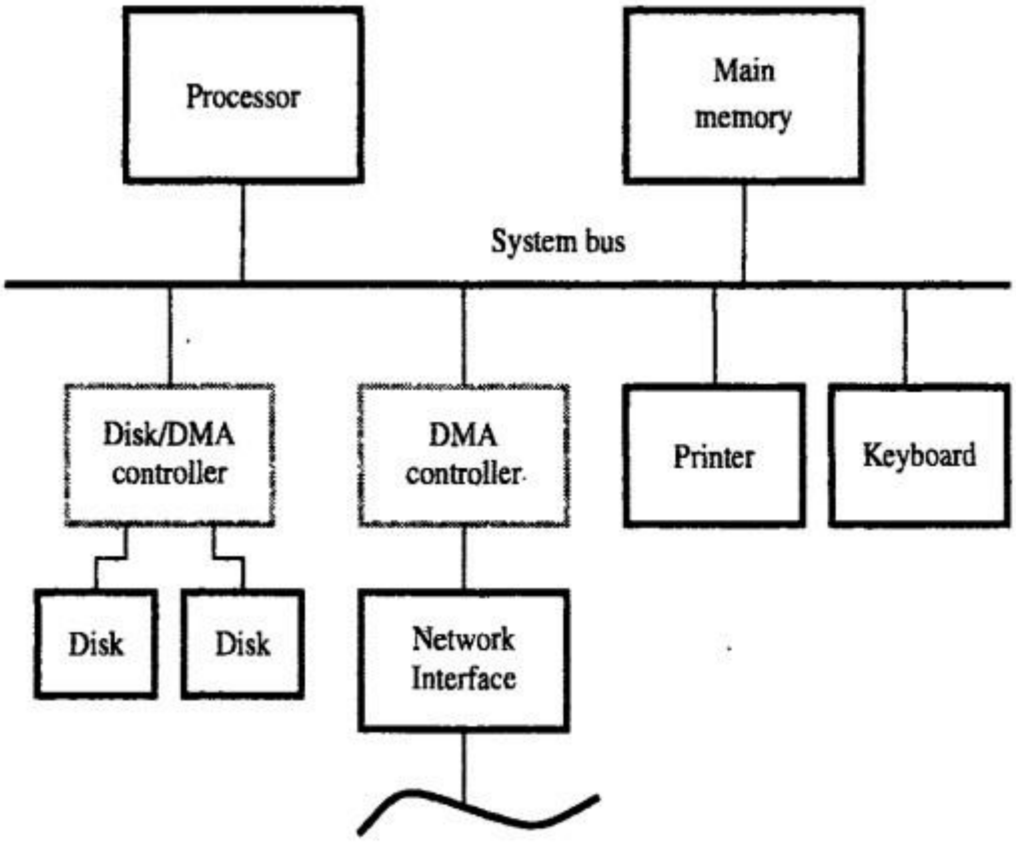
DMA transfers are performed by control circuits that are part of I/O interface called DMA controller. The DMA controller performs functions that would normally be carried out by processor when accessing main memory.



The R/$\overline{W}$ bit determine the direction of transfer. When this bit is set to 1 by a program instruction, the controller performs a 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.

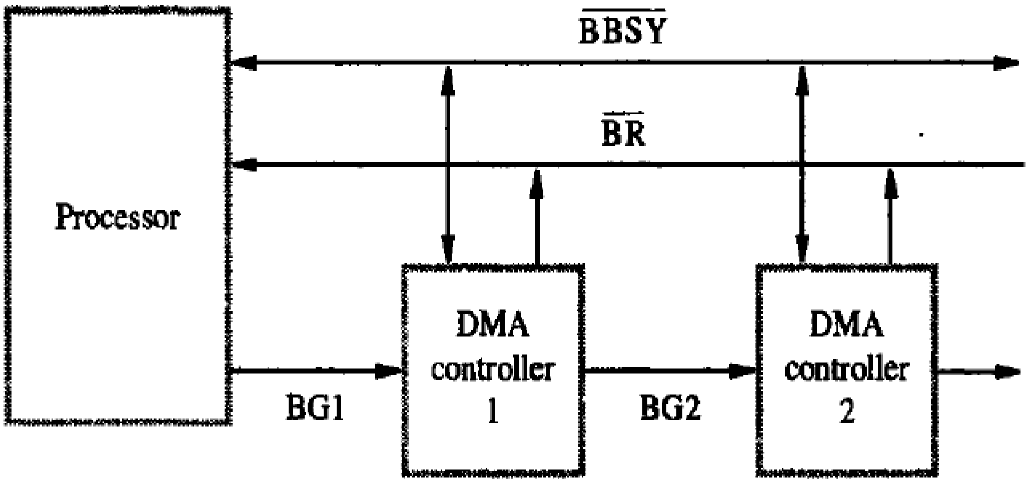


6. b) Write a note on bus arbitration

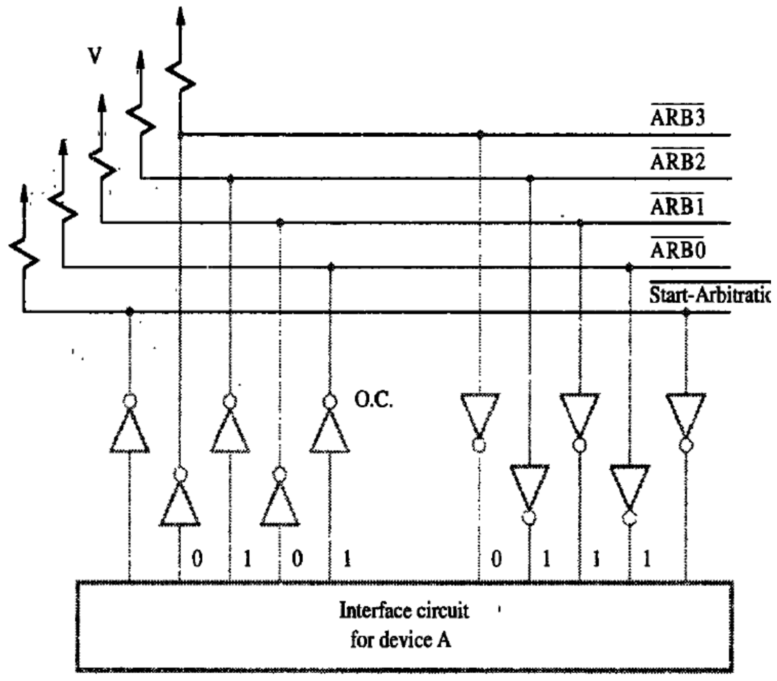
The device that is allowed to initiate data transfers on the bus at any given time is called the bus master. When the current master relinquishes control of the bus, another device can acquire this status. Bus arbitration is the process by 'which the next device to become the bus master is selected and bus mastership is transferred to it. The selection of the bus master must take into account the needs of various devices by establishing a priority system for gaining access to the bus. There are two approaches to bus arbitration: centralized and distributed. In centralized arbitration, a single-bus arbiter performs the required arbitration. In distributed arbitration, all devices participate in the selection of the next bus master.

Centralized Arbitration

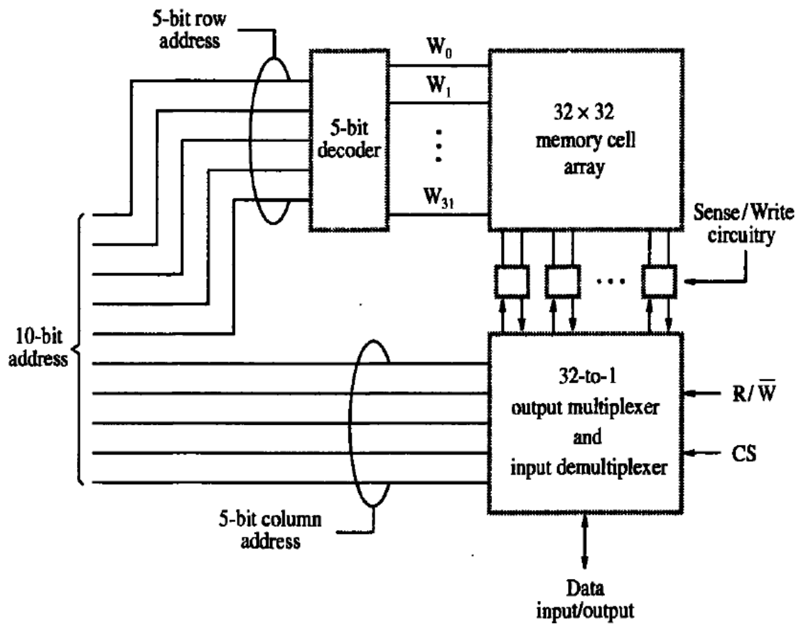
The bus arbiter may be the processor or a separate unit connected to the bus. In this case, the processor is normally the bus master unless it grants bus mastership to one of the controllers. A DMA controller indicates that it needs to become the bus master by activating the Bus-Request line, BR. The signal on the Bus-Request line is the logical OR of the bus requests from all the devices connected to it. When Bus-Request is activated, the processor activates the Bus-Grant signal, BGI, indicating to the DMA controllers that they may use the bus when it becomes free. This signal is connected to all DMA controllers using a daisy-chain arrangement. Thus, if DMA controller 1 is requesting the bus, it blocks the propagation of the grant signal to other devices. Otherwise, it passes the grant downstream by asserting BG2.



Distributed arbitration means that all devices waiting to use the bus have equal responsibility in carrying out the arbitration process, without using a central arbiter. Each device on the bus is assigned a 4-bit identification number. When one or more devices request the bus, they assert the Start-Arbitration signal and place their 4-bit ID numbers on four open-collector lines, ARBO through ARB3. A winner is selected as a result of the interaction among the signals transmitted over these lines by all contenders. The net outcome is that the code on the four lines represents the request that has the highest ID number.

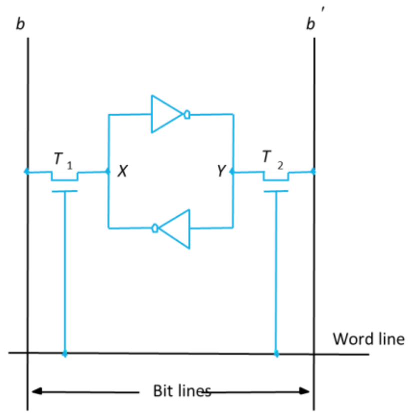


7. a) Explain the organization of 1K X 1 memory chip



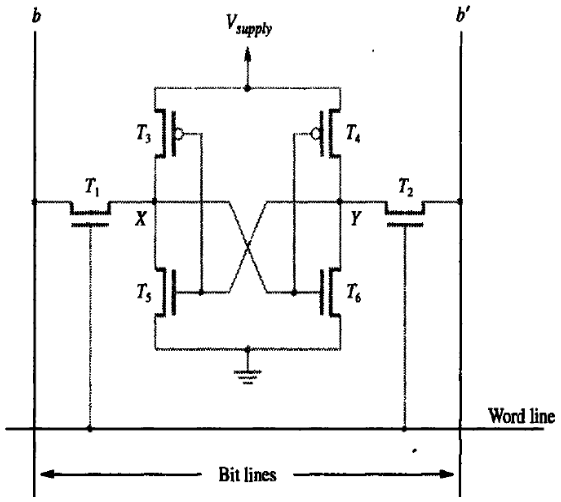
- 10-bit address with 1 bit data line
- 10-bit address is divided into two groups of 5 bits each to form the row and column addresses for the cell array
- A row address selects arow of 32 cells, all of which are accessed in parallel.
- However, according to the column address, only one of these cells is connected to the external data line by the output multiplexer and input demultiplexer.

7. b) Write a note on: (i) Static memories (ii) Cache memory



- Memories that consist of circuits capable of retaining their state as long as power is applied are known as static memories.
- Two inverters are cross connected to implement a basic flip-flop latch.
- The latch is connected to two bit lines by transistors T1 and T2
- These transistors act as switches that can be opened or closed under control of the word line.
- When word line is at ground level, the transistors are turned off and the latch retains its state
- Consider that the cell is in state 1 if the logic value at point X is 1 and at point Y is 0.
- This state is maintained as long as the signal on the word line is at ground level.

- **Read operation:** In order to read state of SRAM cell, the word line is activated to close switches T1 and T2.
- If the cell is in state 1, the signal on bit line b is high and the signal on bit line b' is low.
- Sense/Write circuits at the bottom monitor the state of b and b’.
- **Write operation:** The state of the cell is set by placing the appropriate value on bit line b and its complement on b’,
- When the word line is activated, it forces the cell into the corresponding state.
- The required signals on the bit lines are generated by the Sense/Write circuit.
- SRAMs are said to be *volatile* memories
- **Application of SRAM:**
- Static RAMs has access time of a few nanoseconds
- SRAMs are used in applications where speed is of critical concern.

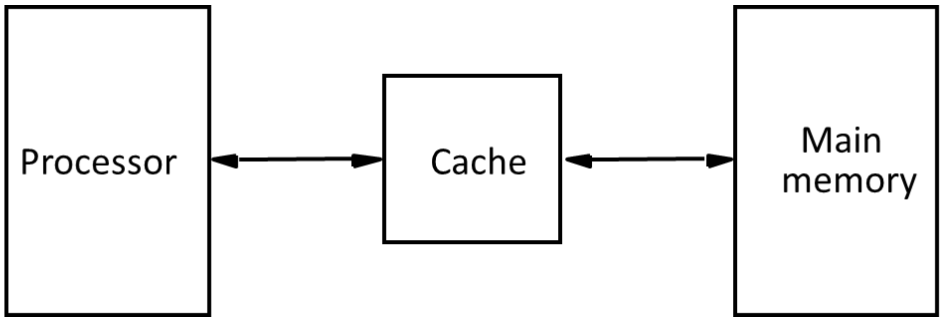


- Transistor pairs (T3, T5) and (T4, T6) form the inverters in the latch
- In state 1, the voltage at point X is maintained high by having transistors T3 and T6 on, while T4 and Ts are off
- If T1 andT2 are turned on (closed), bit lines b and b' will have high and low signals, respectively.
- The power supply voltage, Vsupply is 5V in older CMOS SRAMs or 3.3 V in new low-voltage versions.
- A major advantage of CMOS SRAMs is their very low power consumption

Cache Memory

- Cache memory is an architectural arrangement which makes the main memory appear faster to the processor than it really is.
- Cache memory is based on the property of computer programs known as “locality of reference”.
- Analysis of programs indicates that many instructions in localized areas of a program are executed repeatedly during some period of time, while the others are accessed relatively less frequently.
- These instructions may be the ones in a loop, nested loop or few procedures calling each other repeatedly. This is called “locality of reference”.
- **Temporal locality of reference:**
- Recently executed instruction is likely to be executed again very soon.
- **Spatial locality of reference**:

- Instructions with addresses close to a recently executed instruction are likely to be executed soon.



At any given time, only some blocks in the main memory are held in the cache. Which blocks in the main memory are in the cache is determined by a “mapping function”.
When the cache is full, and a block of words needs to be transferred from the main memory, some block of words in the cache must be replaced. This is determined by a “replacement algorithm”.
Existence of a cache is transparent to the processor. The processor issues Read and Write requests in the same manner. If the data is in the cache it is called a Read or Write hit.

Read hit:
The data is obtained from the cache, main memory is not involved.

Write hit:
Cache has a replica of the contents of the main memory. Contents of the cache and the main memory may be updated simultaneously. This is the write-through protocol. Alternately, update the contents of the cache, and mark it as updated by setting a bit known as the dirty bit or modified bit. The contents of the main memory are updated when this block is replaced. This is write-back or copy-back protocol.
If the data is not present in the cache, then a Read miss or Write miss occurs.

Read miss:
Block of words containing this requested word is transferred from the memory. After the block is transferred, the desired word is forwarded to the processor. The desired word may also be forwarded to the processor as soon as it is transferred without waiting for the entire block to be transferred. This is called load-through or early-restart.

Write-miss:
Write-through protocol is used, then the contents of the main memory are updated directly. If write-back protocol is used, the block containing the addressed word is first brought into the cache. The desired word is overwritten with new information.

8. a. Explain the Magnetic Disc Principles. [10, L1, CO4]

Magnetic Hard Disks

● The storage medium in a magnetic-disk system consists of one or more disks mounted on a common spindle.

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

● The disks are placed in a rotary drive so that the magnetized surfaces move in close proximity to read/write heads, as shown in Figure 5.29a.

● The disks rotate at a uniform speed.

● Each head consists of a magnetic yoke and a magnetizing coil, as indicated in Figure 5.29b.

Magnetic Hard Disks..

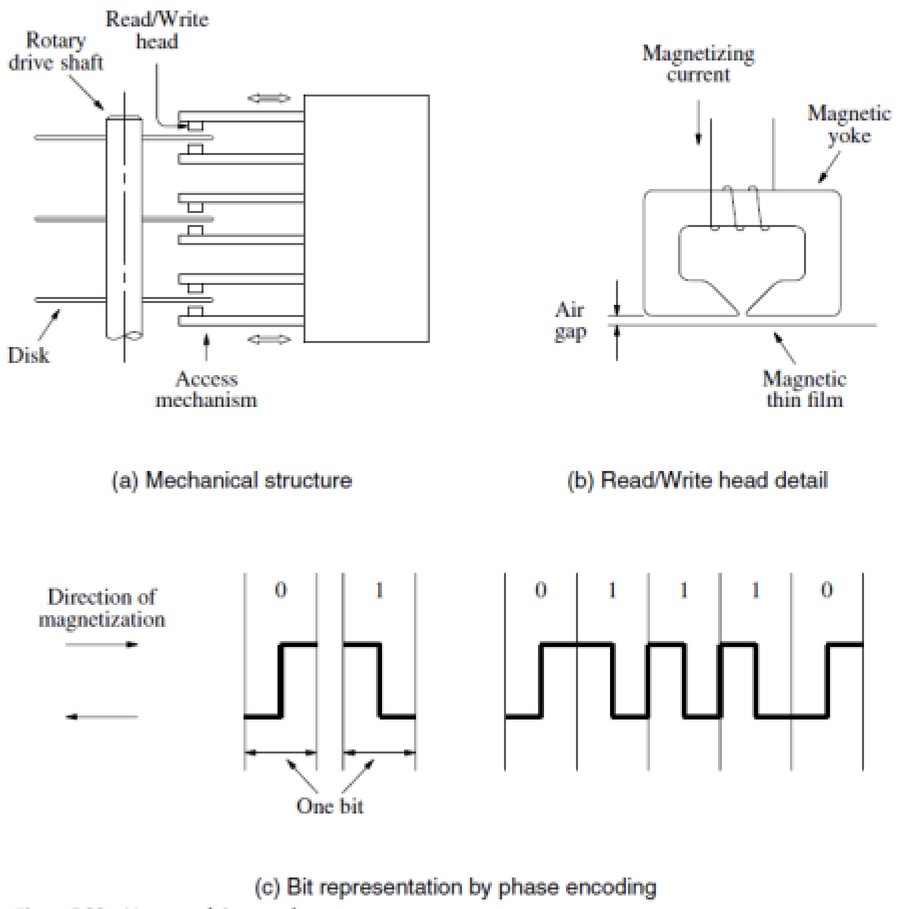


Figure 5.29 Magnetic disk principles.

Digital information can be stored on the magnetic film by applying current pulses of suitable polarity to

the magnetizing coil.

- This causes the magnetization of the film in the area immediately underneath the head to switch to a direction parallel to the applied field.
- The same head can be used for reading the stored information.

● In this case, changes in the magnetic field in the vicinity of the head caused by the movement of the film relative to the yoke induce a voltage in the coil, which now serves as a sense coil.

The polarity of this voltage is monitored by the control circuitry to determine the state of magnetization of the film.

● Only changes in the magnetic field under the head can be sensed during the Read operation.

● Therefore, if the binary states 0 and 1 are represented by two opposite states of magnetization, a voltage is induced in the head only at 0-to-1 and at 1-to-0 transitions in the bit stream.

● A long string of 0s or 1s causes an induced voltage only at the beginning and end of the string. To determine the number of consecutive 0s or 1s stored, a clock must provide information for synchronization.

● In some early designs, a clock was stored on a separate track, where a change in magnetization is

forced for each bit period.

● Using the clock signal as a reference, the data stored on other tracks can be read correctly. The modern approach is to combine the clocking information with the data (self-clocking schemes).

● One simple scheme, depicted in Figure 5.29c, is known as phase encoding or Manchester encoding.

● In this scheme, changes in magnetization occur for each data bit, as shown in the figure.

● A change in magnetization is guaranteed at the midpoint of each bit period, thus providing the clocking information.

● The drawback of Manchester encoding is its poor bit-storage density.

● The space required to represent each bit must be large enough to accommodate two changes in magnetization.

8. b. Draw and explain the internal organization of 2M*8 asynchronous DRAM chip. [10, L2, CO4]

Asynchronous DRAMs..

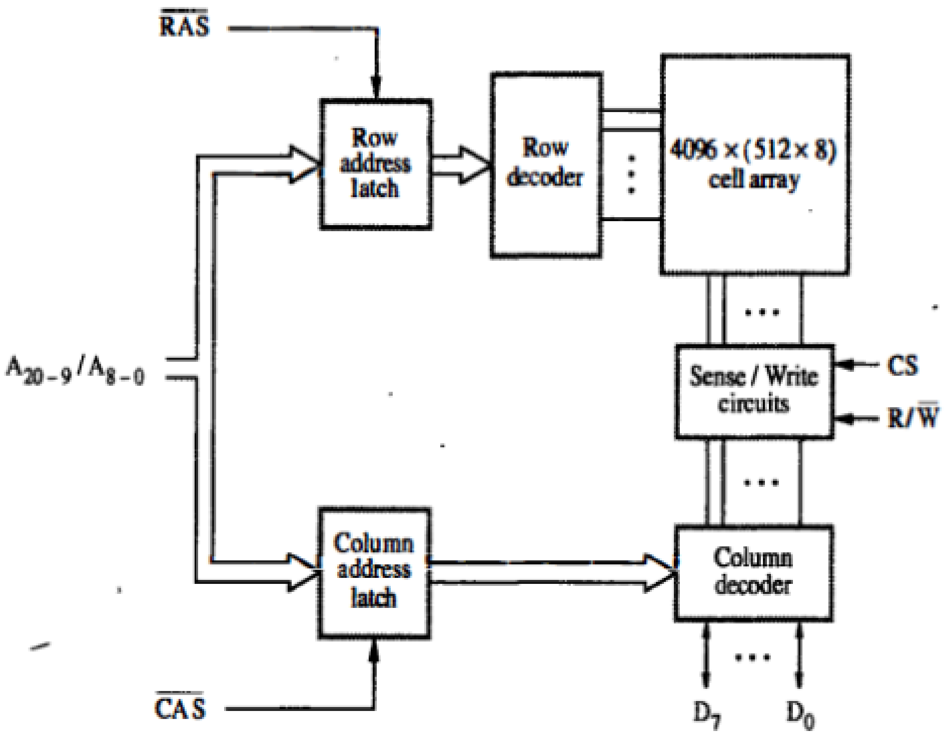


Figure 5.7 Internal organization of a 2M × 8 dynamic memory chip.

The high-order 12 bits constitute the row address and the low-order 9 bits of the address constitute column address of a byte.

● To reduce the number of pins needed for external connections, the row and column addresses are multiplexed on 12 pins.

● During a Read or a Write operation, the row address is applied first.

● It is loaded into the row address latch in response to a signal pulse on the Row Address Strobe (RAS) input of the chip.

Then a Read operation is initiated, in which 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 and loaded into the column address latch under control of the Column Address Strobe (CAS) signal.

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

● If the R/W control signal indicates a Read operation, the output values of the selected circuits are transferred to the data lines, D7-0.

● For a Write operation, the information on the D7-0 lines is transferred to the selected circuits.

● This information is then used to overwrite the contents of the selected cells

in the corresponding 8 columns.

In commercial DRAM chips, the RAS and CAS control signals are active low.

● To indicate this fact, these signals are shown on diagrams as RAS and CAS.

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

● A refresh circuit usually performs this function automatically.

The timing of the memory device is controlled asynchronously.

● A specialized memory controller circuit provides the necessary control signals, RAS and 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 DRAMs.

9.a. Discuss with a neat diagram the single bus organization of data path inside the processor.

Processor Organization:

● Figure 7.1 shows an organization in which the ALU and all the registers are interconnected via a single common bus.

● This bus is internal to the processor.

● The data and address lines of the external memory bus are connected to the internal processor bus via the memory data register, MDR, and the memory address register, MAR, respectively.Register MDR has two inputs and two outputs.

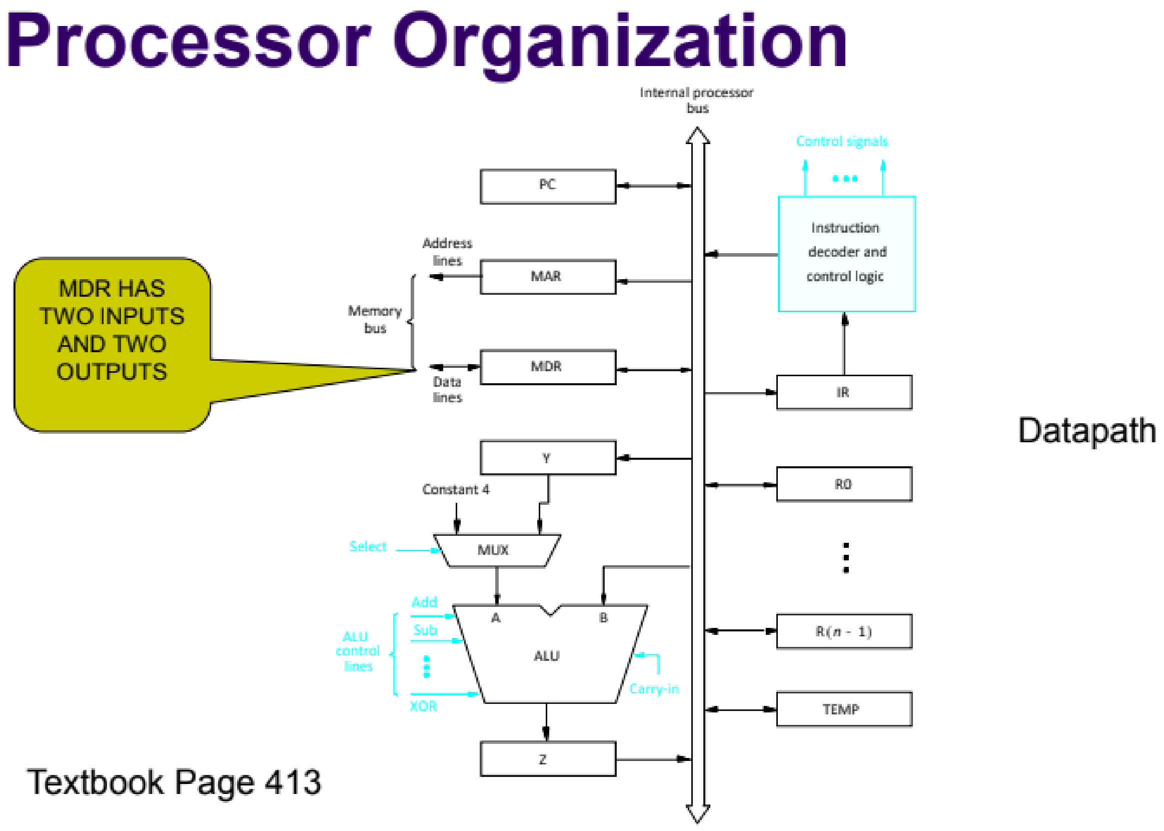


Figure 7.1 Single-bus organization of the datapath inside a processor.

- Data may be loaded into MDR either from the memory bus or from the internal processor bus.

● The data stored in MDR may be placed on either bus.

● The input of MAR is connected to the internal bus, and its output is connected to the external bus.

● The control lines of the memory bus are connected to the instruction decoder and control logic block.

● This unit is responsible for issuing the signals that control the operation of all the units inside the processor and for interacting with the memory bus.

The number and use of the processor registers R0 through R(n - 1) vary considerably from one processor to another.

● Registers may be provided for general-purpose use by the programmer.

● Some may be dedicated as special-purpose registers, such as index registers or stack pointers.

● The registers, Y, Z, and TEMP are used by the processor for temporary storage during execution of some instructions.

● These registers are never used for storing data generated by one instruction for later use by another instruction.

The multiplexer MUX selects either the output of register Y or a constant value 4 to be provided as input A of the ALU.

● The constant 4 is used to increment the contents of the program counter.

● We will refer to the two possible values of the MUX control input Select as Select4 and SelectY for selecting the constant 4 or register Y, respectively.

As instruction execution progresses, data are transferred from one register to another, often passing through the ALU to perform some arithmetic or logic operation.

● The instruction decoder and control logic unit is responsible for implementing the actions specified by the instruction loaded in the IR register.

● The decoder generates the control signals needed to select the registers involved and direct the transfer of data.

● The registers, the ALU, and the interconnecting bus are collectively referred to as the datapath.

● Transfer a word of data from one processor register to another or to the ALU.

● Perform an arithmetic or a logic operation and store the result in a processor register.

● Fetch the contents of a given memory location and load them into a processor

register.

● Store a word of data from a processor register into a given memory location.

9.b. What are actions required to execute a complete instruction ADD (R3), R1. [10,L1, CO5]

Execution of a Complete Instruction

● **Consider the instruction Add (R3), R1**

● **Executing this instruction requires the following actions:**

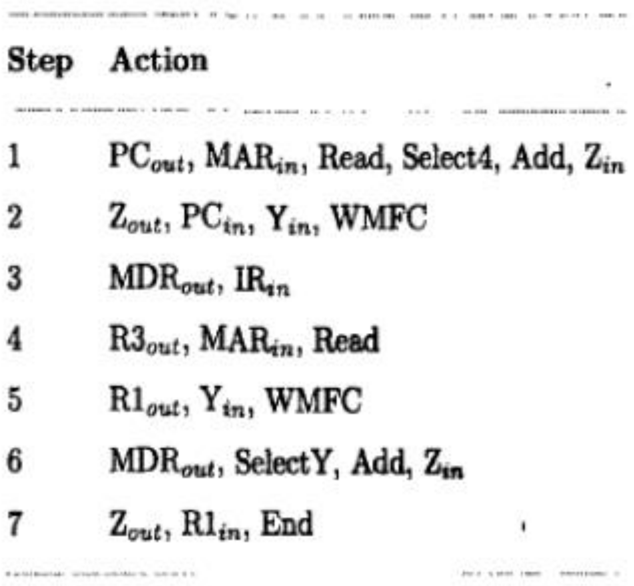
1. Fetch the instruction

2. Fetch the first operand (the contents of the memory location pointed to by R3)

3. Perform the addition

4. Load the result into R1

Add (R3), R1



Step	Action
1	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in}, Y_{in}, WMFC
3	MDR_{out}, IR_{in}
4	$R3_{out}$, MAR_{in}, Read
5	$R1_{out}$, Y_{in}, WMFC
6	MDR_{out}, SelectY, Add, Z_{in}
7	Z_{out}, $R1_{in}$, End

Figure 7.6 Control sequence for execution of the instruction Add (R3),R1.

● Figure 7.6 gives the sequence of control steps required to perform these operations for the single-bus architecture of Figure 7.1.

● Steps 1 through 3 constitute the instruction fetch phase,

● This is the same for all instructions.

● The instruction decoding circuit interprets the contents of the IR at the beginning of step 4.

● This enables the control circuitry to activate the control signals for steps 4 through 7, which constitute the execution phase.

10.a. Draw and Explain Multiple bus organization of a CPU. [10,L1, CO5]

Multiple-Bus Organization

● To reduce the number of steps needed, most commercial processors provide multiple internal paths that enable several transfers to take place in parallel.

● Figure 7.8 depicts a three-bus structure used to connect the registers and the ALU of a processor.

Multiple-Bus Organization..

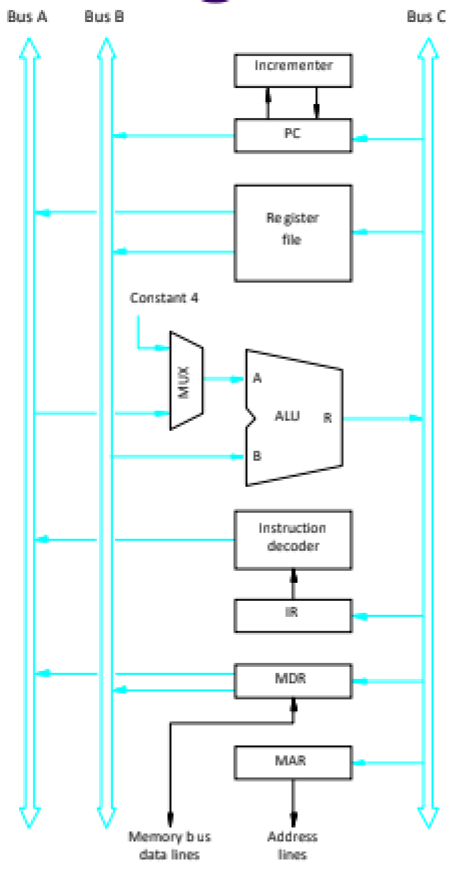


Figure 7.8 Three-bus organization of the datapath.

● All general-purpose registers are combined into a single block called the register file.

● Implemented in the form of an array of memory cells.

● The register file in Figure 7.8 is said to have three ports.

● There are two outputs, allowing the contents of two different registers to be accessed simultaneously and have their contents placed on buses A and B.

- The third port allows the data on bus C to be loaded into a third register during the same clock cycle.
- Buses a and B are used to transfer the source operands to the a and B inputs of the ALU, where an arithmetic or logic operation may be performed.
- The result is transferred to the destination over bus C.
- If needed, the ALU may simply pass one of its two input operands unmodified to bus C.
- We will call the ALU control signals for such an operation R=A or R=B.
- The Incrementer unit is used to increment the PC by 4.
- Using the Incrementer eliminates the need to add 4 to the PC using the main ALU.
- The source for the constant 4 at the ALU input multiplexer is still useful.
- It can be used to increment other addresses, such as the memory addresses in LoadMultiple and StoreMultiple instructions.

10.b. Draw and Explain organization of the control unit to allow conditional branching in the microprogram. [10, L1,CO5]

Control Unit Organization

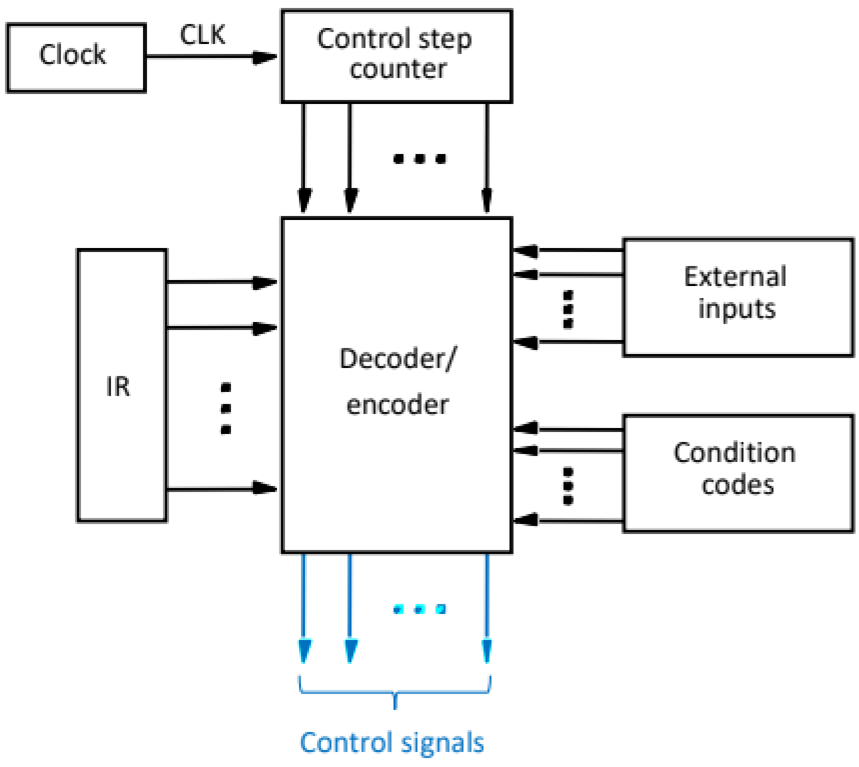


Figure 7.10. Control unit organization.

- **The required control signals are determined by the following information:**
- **Contents of the control step counter**
- **Contents of the instruction register**
- **Contents of the condition code flags**
- **External input signals, such as MFC and interrupt requests**
- **The decoder/encoder block in Figure 7.10 is a combinational circuit that generates the required control outputs, depending on the state of all its inputs.**

● By separating the decoding and encoding functions, we obtain the more detailed block diagram in Figure 7.11.

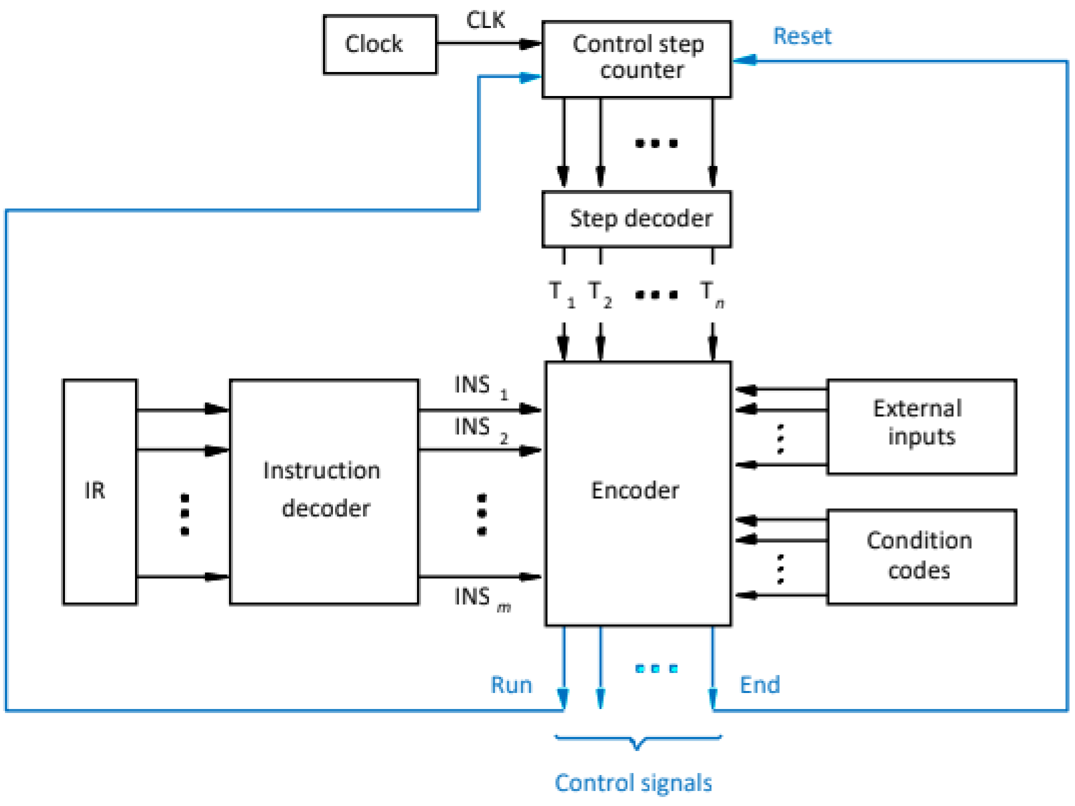


Figure 7.11. Separation of the decoding and encoding functions.

● The step decoder provides a separate signal line for each step, or time slot, in the control sequence.

● Similarly, the output of the instruction decoder consists of a separate line for each machine instruction.

● For any instruction loaded in the IR, one of the output lines INS1

through INSm is set to 1, and all other lines are set to 0.

● The input signals to the encoder block are combined to generate the individual control signals Yin, PCout, Add, End, and so on.