RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, BHOPAL

New Scheme Based On AICTE Flexible Curricula

Computer Science and Engineering, IV-Semester

CS404 Computer Org. & Architecture

Objectives: Students to be familiarize the basic principles of computer architecture, Design and Multi Processing, Types of data transfer, Concept of semi conductor memories which is useful for research work in field Computer System.

Basic Structure of Computer: Structure of Desktop Computers, CPU: General Register Organization-Memory Register, Instruction Register, Control Word, Stack Organization, Instruction Format, ALU, I/O System, bus,CPU and Memory Program Counter, Bus Structure, Register Transfer Language-Bus and Memory Transfer, addressing modes. Control Unit Organization: Basic Concept of Instruction, Instruction Types, Micro Instruction Formats, Fetch and Execution cycle, Hardwired control unit, Micro-programmed Control unit microprogram sequencer Control Memory, Sequencing and Execution of Micro Instruction.

Computer Arithmetic: Addition and Subtraction, Tools Compliment Representation, Signed Addition and Subtraction, Multiplication and division, Booths Algorithm, Division Operation, Floating Point Arithmetic Operation. design of Arithmetic unit

I/O Organization:I/O Interface –PCI Bus, SCSI Bus, USB, Data Transfer: Serial, Parallel, Synchronous, Asynchronous Modes of Data Transfer, Direct Memory Access(DMA), I/O Processor.

Memory Organization: Main memory-RAM, ROM, Secondary Memory –Magnetic Tape, Disk, Optical Storage, Cache Memory: Cache Structure and Design, Mapping Scheme, Replacement Algorithm, Improving Cache Performance, Virtual Memory, memory management hardware

Multiprocessors: Characteristics of Multiprocessor, Structure of Multiprocessor-Inter-processor Arbitration, Inter-Processor Communication and Synchronization. Memory in Multiprocessor System, Concept of Pipelining, Vector Processing, Array Processing, RISC And CISC, Study of Multicore Processor –Intel, AMD.

Reference Books:

1. Morris Mano , “Computer System Organization ”PHI
2. Alan Clements: “Computer Organization and Architecture”, Cengage Learning
3. Subrata Ghosal: “Computer Architecture and Organization”, Pearson
4. William stalling ,“Computer Architecture and Organization” PHI
5. M. Usha, T.S. Shrikant: “Computer System Architecture and Organization”, Willey India
6. Chaudhuri, P.Pal: “Computer Organization and Design”, PHI
7. Sarangi: “Computer Organization and Architecture”,Mc-Graw Hills

Computer Org.& Architecture (List of Practicals)

1. Study of Multiplexer and Demultiplexer
2. Study of Half Adder and Subtractor
3. Study of Full Adder and Subtractor
4. WAP to add two 8 bit numbers and store the result at memory location 2000
5. WAP to multiply two 8 bit numbers stored at memory location 2000 and 2001 and stores the result at memory location 2000 and 2001.
6. WAP to add two 16-bit numbers. Store the result at memory address starting from 2000.
7. WAP which tests if any bit is '0' in a data byte specified at an address 2000. If it is so, 00 would be stored at address 2001 and if not so then FF should be storedat the same address.
8. Assume that 3 bytes of data are stored at consecutive memory addresses of the data memory starting at 2000. Write a program which loads register C with (2000), i.e. with data contained at memory address2000, D with (2001), E with (2002) and A with (2001).
9. Sixteen bytes of data are specified at consecutive data-memory locations starting at 2000. Write a program which increments the value of all sixteen bytes by 01.
10. WAP to add t 10 bytes stored at memory location starting from 3000. Store the result at memory location 300A

UNIT – 1

Basic Structure of Computer:- Structure of Desktop Computers, CPU: General Register Organization- Memory Register, Instruction Register, Control Word, Stack Organization, Instruction Format, ALU, I/O System, bus, CPU and Memory Program Counter, Bus Structure, Register Transfer Language- Bus and Memory Transfer, addressing modes.

STRUCTURE OF DESKTOP COMPUTERS

The desktop computers are the computers which are usually found on a home or office desk. They consist of processing unit, storage unit, visual display and audio as output units, and keyboard and mouse as input units. Usually storage unit of such computer consists of hard disks, CD-ROMs, and diskettes. Desktop computers are basically digital computers. They consist of **fi**ve functionally independent units: input, memory, arithmetic and logic, output and control units. Memory unit is also known as storage unit, and arithmetic and logic unit (ALU) and control unit are combine as processing unit. Fig. shows these five functional

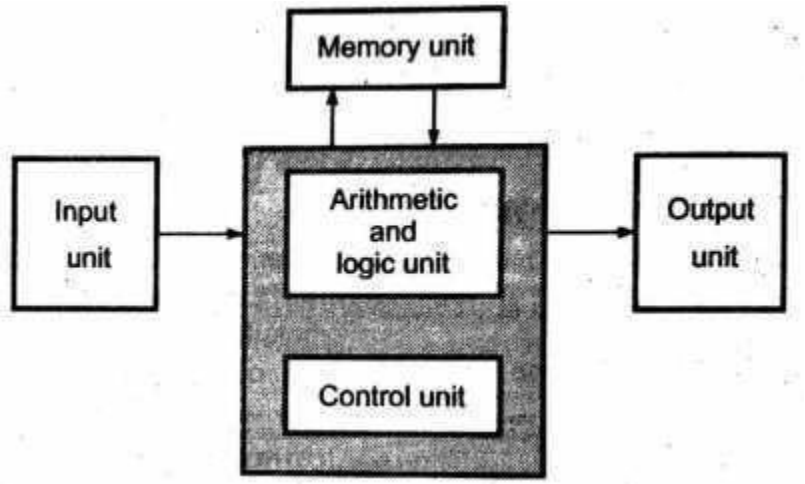


Figure 2. units of a computer.

Input Unit - The input unit accepts the digital information from user with the help of input devices such as keyboard, mouse, microphone etc. The information received from the input unit is either stored in the memory for later use or immediately used by the arithmetic and logic unit to perform the desired operations. The most commonly used input devices are keyboard and mouse, the keyboard is used for entering text and numeric information. On the other hand, mouse is used to position the screen cursor and thereby-enter the information by selecting option. Apart from keyboard and mouse there are many other input devices are available, which include joysticks, trackball, digitizers and scanners etc.

Memory Unit - The memory unit is used to store programs and data. Usually, two types of memory devices are used to form a memory unit: primary storage memory device and secondary storage memory device. The primary memory, commonly called main memory is a fast memory used for the storage of programs and active data. The main memory is a semiconductor memory. It consists of a large number of semiconductor storage cells, each capable of storing one bit of information. The main memory consists of only randomly accessed memories. These memories are fast but they are small in capacities and expensive. Therefore, the computer uses the secondary storage memories such as magnetic tapes, magnetic disks for the storage of large amount of data.

Arithmetic and Logic Unit - The Arithmetic and Logic Unit (ALU) is responsible for performing arithmetic operations such as add, subtract, division and multiplication and logical operations such

as ANDing, ORing, Inverting etc. To perform these operations, operands from the main memory are brought into the high speed storage elements called registers of the processor. Each register can store one word of data and they are used to store frequently used operands. The access times to registers are typically 5 to 10 times faster than access times to memory. After performing operation, the result is either stored in the register or memory location.

Output Unit - The output unit sends the processed results to the user using output devices such as video monitor, printer, plotter, etc. The video monitors display the output on the CRT screen whereas printers and plotters give the hard- copy output.

Control Unit - The control unit co—ordinates and controls the- activities amongst the functional. The basic function, of control unit is to fetch the instructions stored in the main memory, identify .the operations and the devices involved in it and accordingly generate control signals to execute the desired operations. The control unit uses control signals or timing signals to determine when a given action is to take place. It controls input and output operations, data transfers between the processor, memory and input/ output devices using timing signals.

CPU (CENTRAL PROCESSING UNIT)

The CPU is the brain of the Computer system. It works as an administrator of a system. All the operations within the system are supervised and controlled by CPU. It interprets and co-ordinates the instructions. The data and instructions are temporarily stored in its memory unit. After performing Operation, the result of operation can be stored in this memory unit.

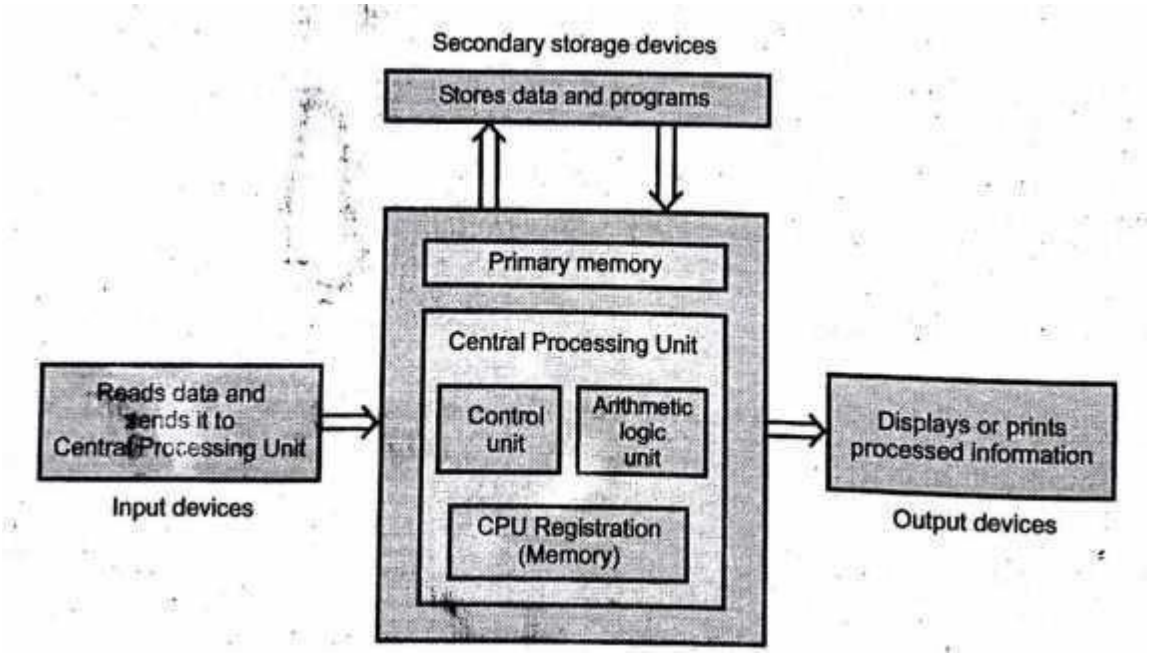


Figure 2 CPU and its interaction with other units

The results of operation are sent towards output unit for the user. Thus, CPU controls all internal and external devices, performs arithmetic and logical operations, controls the memory usage and control the sequence of operations. For performing all these operations, the CPU has three subunits.

- Arithmetic and Logic Unit (ALU)
- Control Unit
- Memory (CPU registers)

Arithmetic and Logic Unit (ALU) - ALU is a subunit of CPU. It performs arithmetic operations like addition, subtraction and logic operations like OR, AND, invert, exclusive-OR on binary words.

The data stored in memory unit is transferred to ALU. The ALU performs the operation, that is, the data is processed and the result is stored in internal memory unit of CPU. The result of final operation is transferred from memory unit to an output unit. Arithmetic and logic operations performed by ALU sets flags to represent certain conditions such as equal to condition, zero condition, greater than condition and so on.

Control Unit - The control unit controls all the operations which internally take place within the CPU and also the operations of CPU related to input/output devices. The control unit directs the overall functioning of a computer system. This unit also checks the correctness of sequence of operations. It fetches instructions in a program from the primary storage unit, interprets them and generates control signals to ensure correct execution of the program. The control signals generated by the control unit direct the overall functioning of the other units of the computer.

CPU Registers - Register is a group of flip-flops which can be used to store a word. It is a high speed temporary storage space for holding data, addresses and instructions during processing the instruction. Registers are not referenced by their addresses; they are directly accessed. To perform execution of instruction, the processor contains a number of registers used for temporary storage of data and some special function registers.

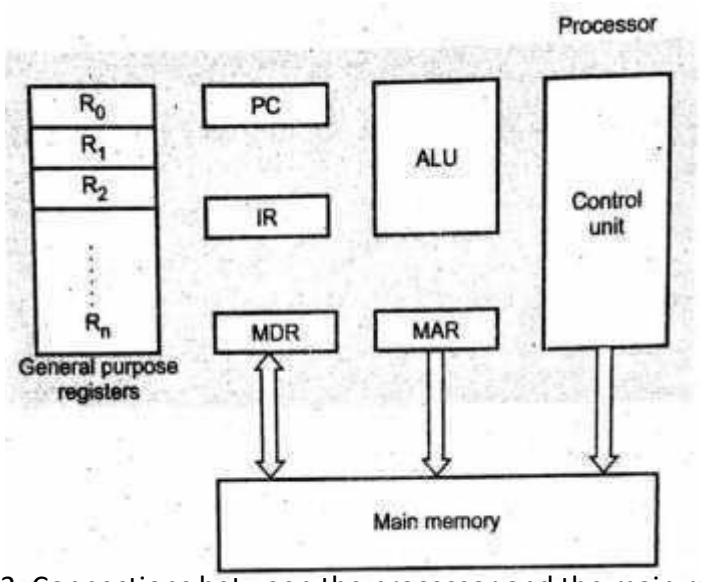


Figure 3. Connections between the processor and the main memory

The special function registers include Program Counter (PC), Instruction Register (IR), Memory Address Register (MAR) and Memory Data Register (MDR).
Program Counter (PC):- A program is a series of instructions stored in the memory. These instructions tell the CPU exactly how to get the desired result. It is important that these instructions must be executed in a proper order to get. the correct result. The sequence of instruction execution is monitored by the program counter. It keeps track of which instruction is being executed and what the next instruction.

Instruction Register (IR):- It is used to hold the instruction that is currently being executed. The contents of IR are available to the control unit, which generate the timing signals that control the various processing elements involved in executing the instruction.

Memory Address Register [MAR] and Memory Data Register (MDR): - These registers are used to handle the data transfer between the main memory and the processor. The MAR holds the address of the main memory to or from which data is to be transferred. The MDR sometimes also called MBR (Memory Buffer Register) contains the data to be written into or read from the addressed word of the main memory.

General purpose registers - These are used to hold the operands for arithmetic and logic operations and/or used to store the result of the operation. Since the access time of these registers is lowest, these are used to store frequently used data. Arithmetic logic unit uses CPU registers to accept data for processing. After processing data ALU Stores result again in the CPU register. A'LU also stores the status of the result in. the CPU register. This result includes information whether result is positive/negative, zero, having even/odd parity, any carry/borrow resulted during arithmetic operation and so on. This register is commonly known as flag register or program status register. Supervisory control unit of a CPU uses flag register to execute conditional branch/jump instructions.

GENERAL REGISTER ORGANIZATION- MEMORY REGISTER, INSTRUCTION REGISTER

Bus organization

Fig. 4 shows the general bus organization for seven CPU registers. It shows that how registers are selected and how data flow between register and ALU take place. Decoder is used to select a particular register. The output of each register is connected to two multiplexers (MUX) to form the two buses A and B. The selection lines in each multiplexer select the input data for the particular bus. The A and B buses form the two inputs of an Arithmetic Logic Unit (ALU). The operation select lines decide the micro operation to be performed by ALU. The result of the micro- operation is available at the output bus. The output bus connected to the inputs of all registers, thus by selecting a destination register it is possible to store the result in it.

The basic function performed by a computer is the execution of a program. The program, which is to be executed, is a set of instructions, which are stored in memory. The central processing unit (CPU) executes the instructions of the program to complete a task. The instruction execution takes place in the CPU registers. Let us, first discuss few typical registers, some of which are commonly available in of machines. These registers are:-

- Memory Address Register (MAR):- Connected to the address lines of the system bus. It specifies the address of memory location from which data or instruction is to be accessed (for read operation) or to which the data is to be stored (for write operation).
- Memory Data Register (MDR):- Connected to the data lines of the system bus. It specifies which data is to be accessed (for read operation) or to which data is to be stored (for write operation).
- Program Counter (PC):- Holds address of next instruction to be fetched, after the execution of an on-going instruction.
- Instruction Register (IR):- Here the instructions are loaded before their execution or holds last instruction fetched.
- Accumulator (ACC):- It holds the result generated by ALU.

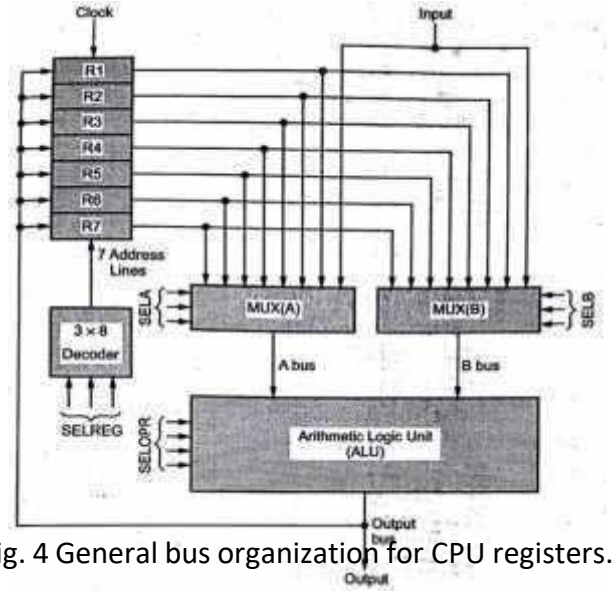


Fig. 4 General bus organization for CPU registers.

We designate computer registers by capital letters to denote the function of the register. For example, the register that holds an address for the memory unit is usually called a memory address register, represented by MAR. Other examples are PC (for program counter), IR (for instruction register) and R1 (for processor register). We show the individual flip-flops in an n-bit register by giving numbers them in sequence from 0 through n - 1, starting from 0 in the right most position and increasing the numbers toward the left. A 16-bit register is divided into two halves.

Low byte (Bits 0 through 7) is assigned the symbol L
High byte (Bits 8 through 15) is assigned the symbol H.

The name of a 16-bit register is PC. The symbol PC (L) represents the low order byte and PC (H) designates the high order byte. The statement

Table 2.1: List of Registers for the Basic Computer

Register Symbol	Number of bits	Register name	Function
DR	16	Data Register	Holds memory operand
AR	12	Address Register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

R2 R1 refers the transfer of the content of register R1 into register R2.

It should be noted that the content of the source register R1 does not change after the transfer. In real applications, the transfer occurs only under a predetermined control condition. This can be shown by means of an “if-then” statement:

If P=1 then R2 R1

where P is a control signal generated in the control section of the system. For convenience we separate the control variables from the register transfer operation by specifying a control function. A control function is a Boolean variable that is equal to 1 or 0. The control function is written as follows:

P: R2 R1

A read operation implies transfer of information to the outside environment from a memory word, whereas storage of information into the memory is defined as write operation. Symbolizing a memory word by the letter M, it is selected by the memory address during the transfer which is a specification for transfer operations. The address is specified by enclosing it in square brackets following the letter M. For example, the read operation for the transfer of a memory unit M from an address register AR to another data register DR can be illustrated as:

Read: DR CM[AR]

The write operation transfer the contents of a data register to a memory word M selected by the address. Assume that the input data are in register R1 and the address in the AR. The write operation can be stated symbolic as follows:

Write: M[AR] C R1

This cause a transfer on information from R1 into the memory word M selected by the address in AR

CONCEPT OF CONTROL WORD

The combined value of a binary selection inputs specifies the control word. That Fig. 5 shows the control Word format. It consists of four **fields** SELA, SELB and SELREG contain three bits each and SELOPR field contains four bits. Thus the total bits in the control word are 13-bits.

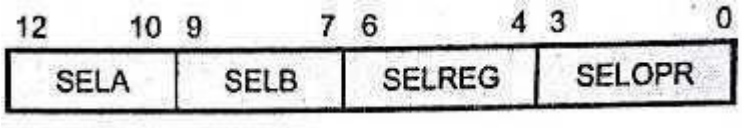


Figure 5 Format of control word

The three bits of SELA select a source registers of the A input of the ALU. The three bits of SELB select a source registers of the B input of the ALU. The three bits of SELREG select a destination register using the decoder. The four bits of SELOPR select the operation to be performed by ALU.

STACK ORGANIZATION

Data operated on by a program can be stored in the CPU registers and in the computer memory. It can be organized properly for easy access. For example, the data items of similar data type can be organized in the term of arrays and can be accessed using array pointers. Like array data structure, mast of the computers supports an important data structure known as a stack.

A stack is a list of data elements, usually words or bytes with the accessing restriction that the 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 of the stack. As only one end is used for adding and removing data elements, the element stored last is the first element retrieved. Thus stack structure is also known as Last-In-First-Out (LIFO) list. The terms Push and Pop are used to describe placing a new data element on the stack and removing the top data element from the stack, respectively.

The stack in the digital computer is a part of register unit or memory unit with a register that holds the address for the stack. The part of register array or memory used for stack is called stack area and the register used to hold the address of stack is called stack pointer. The value in the stack pointer always points at the top data element in the stack.

1. Register Stack

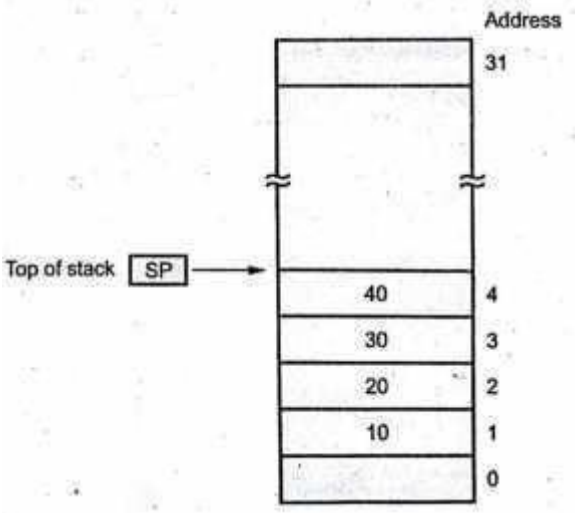


Figure 6. 32 word register stack

A stack can be placed in a portion of a memory unit or it can be organized as a collection of a finite number of CPU registers. The Figure 6 shows the organization of a 32-word register stack. The stack pointer holds the address of the register that is currently the top of stack. As shown in the Fig. 6 four data elements 10, 20, 30 and 40 are placed in the stack. The data element 40 is on the top of stack therefore, the content of SP is now 4.

To remove the top data element the stack is popped by reading the register at address 4 and decrementing the content of SP. The data element 30 is now on top of the stack since SP holds address 3. To insert a new data element, stack is pushed by incrementing SP and writing a data element in the incremented location in the stack. If SP reaches 31 and one more data element is pushed then SP is incremented to 0 and data element is stored in the register 0. Now there is no more empty registers in the stack and stack is said to be FULL. The Fig. 6 shows the operation of stack when specified sequence of instructions are executed.

2. Memory Stack

The operation of memory stack is exactly similar to the register stack. It is implemented using computer memory instead of CPU register array. The number of registers in the CPU is limited and it restricts the size of stack in the stack computer. But when stack is implemented using memory its size is extended upto the memory addressing capacity of the CPU. But computer memory is also shared by program and data, as shown in the Fig. 7 Hence a part of computer memory is used for the stack. The memory stack has an advantage of large size but the operation on it is slower than that of register stack. This is because register stack is internal to the CPU and does not heed any memory access.

In the stack organized computers instruction's operands are stored at the top of the stack, so data processing instructions do not need to contain addresses as they do in a conventional Von Neumann computer. The add operation A + B is specified for a stack computer by the following sequence of three instructions.

```
PUSH    A
PUSH    B
ADD
```

The first PUSH instruction loads A into top of stack (TOS). Execution of PUSH B causes A's location to become TOS + 1 and places B in the new TOS immediately above A. To execute ADD, the top two words of the stack are popped into the ALU where they are added, and the sum is pushed back into the stack. As shown by program sequence in the stack computer operator is specified after the operands i.e. AB + instead of A + B. This method of representing arithmetic expression is known as Reverse Polish notation. The stack computer evaluates arithmetic and other expressions using a reverse polish notation format.

In most conventional computers stack is used to implement subroutine call and return instructions. In some situations, it is better to execute part of a program that is not in sequence with the main program. For example, there may be a part of a program that must be repeated many times during the execution of the entire program. Rather than writing repeated part of the program again and again, the programmer' can write that part only once. This part is written separately. The part of the program which is written separately is called subroutine. When the subroutine is called by CALL instruction the current content of PC are pushed on the stack and after execution of subroutine program the content of PC are popped back into the PC to execute the next instruction after the CALL.

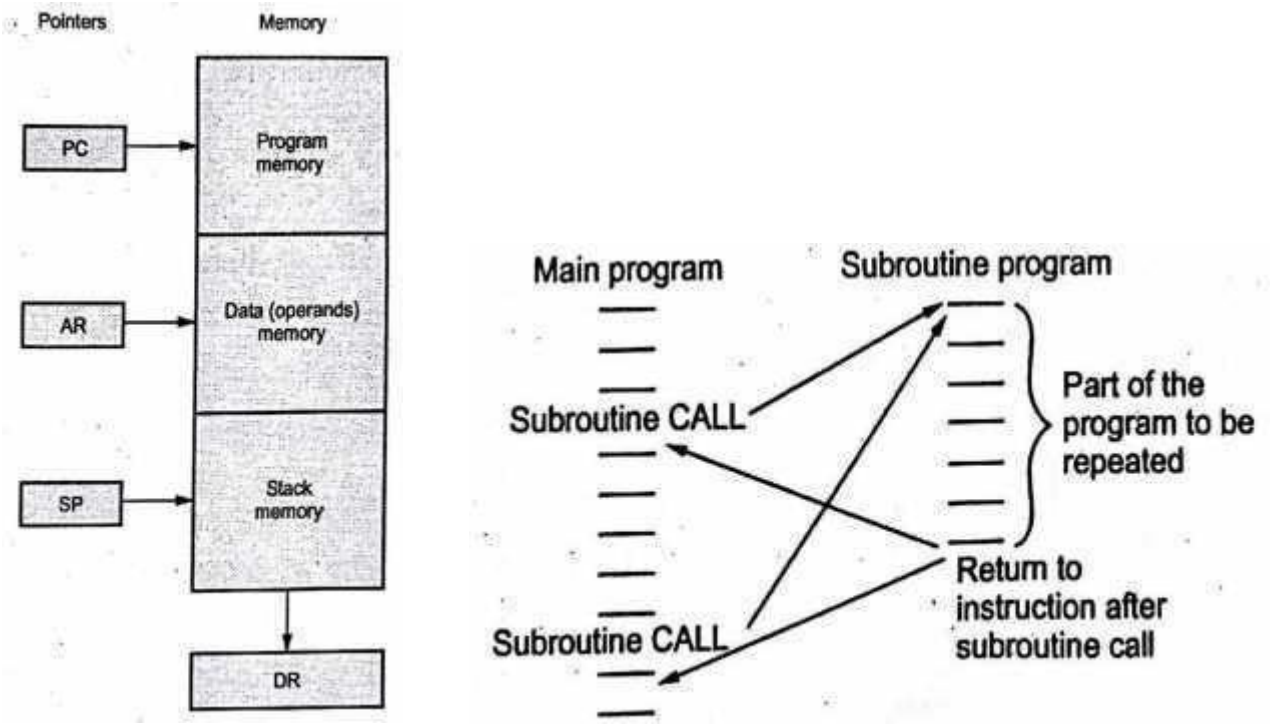


Figure 7 a) Sharing of computer memory by programs, data& stack area b) Execution of subroutine programs

Notations

Basically, there are three methods of representing arithmetic expressions. These are

1. A+B : Infix notation.
2. + AB : Prefix or polish notation.
3. AB + : Postfix or reverse polish notation

INSTRUCTION FORMAT

The internal construction of the CPU, including the processor registers available and their logical capabilities. Computer has a variety of instruction code formats, it is the control unit within the CPU that interprets each instruction code and provides the necessary control functions needed to process the instruction. The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register. The bits of the instruction are divided into groups called fields. These information fields of instructions are called elements of instruction & most common fields found in instruction formats are:

Operation code: - The operation code field in the instruction specifies the operation to be performed. The operation is specified by binary code hence the name operation code or simply opcode.

Source / Destination operand: - The source/destination operand field directly specifies the source/destination operand for the instruction. Source operand address: - The operation specified by the instruction may require one or more operands. The source operand may be in the CPU register or in the memory. Many times the instruction specifies the address of the source operand so that operand(s) can be accessed and operated by the CPU according to the instruction.

Destination operand address: - The operation executed by the CPU may produce result. Most of the time the results are stored in one of the operand. Such operand is known as destination operand. The instruction which produce result specifies the destination operand address.

Next instruction address: - The next instruction address tells the CPU from where to fetch the next instruction after completion of execution of current instruction. For JUMP and BRANCH instructions the address of the next instruction is specified within the instruction. However, for other instructions, the next instruction to be fetched immediately follows the current instruction.

At times other special fields are also employed, sometimes employed as for example a field that gives the number of shifts in a shift-type instruction. The operation code field of an instruction is referred to a group of bits that define various processor operations, such as add, subtract, complement, and shift. A variety of alternatives for choosing the operands from the given address is specified by the bits that define the mode field of the instruction code. Execution of operations done by some data stored in memory or processor registers through specification received by computer instructions.

A memory address specifies operands residing in memory whereas those residing in processor registers are specified by a register address. A register address is a binary number of k bits that defines one of 2^k registers in the CPU. Thus a CPU with 16 processor registers R_0 through R_{15} will have a register address field of four bits. The internal organization of the registers of a computer determines the number of address fields in the instruction formats. Most computers fall into one of three types of CPU organizations:

1. Single accumulator organization.

An example of an accumulator-type organization is the basic computer. All operations are performed with an implied accumulator register. The instruction format in this type of computer uses one address field. For example, the instruction that specifies an arithmetic addition is defined by an assembly language instruction as:

ADD X

Where X is the address of the operand. The ADD instruction in this case results in the operation AC ? AC + M[X]. AC is the accumulator register and M[X] symbolizes the memory word located at address X.

2. General register organization.

The instruction format in this type of computer needs three register address fields. Thus, the instruction for an arithmetic addition may be written in an assembly language as:

ADD Rl, R2, R3

To denote the operation Rl ? R2 + R3. The number of address fields in the instruction can be reduced from three to two if the destination register is the same and one of the source registers. Thus, the instruction:

ADD Rl, R2

Would denote the operational R1 ? R1 + R2. Only register addresses for R1 and R2 need to be specified in this instruction.

3. Stack organization.

Computers with multiple processor registers use the move instruction with a mnemonic MOV to symbolize a transfer instruction. Thus the instruction:

MOV R1, R2

denotes the transfer R1 ? R2 (or R2 ? R1, depending on the particular computer). Thus, transfer-type instructions need two address fields to specify the source and the destination.

Usually only two or three address fields are employed general register-type computers in their instruction format. Each address field may specify a processor register or a memory word. An instruction symbolized by:

ADD R1, X

would specify the operation R1 R1+M[X]. It has two address fields, one for register R1 and the other for the memory address X.

Computers with stack organization would have PUSH and POP instructions which require an address field. Thus the instruction:

PUSH X

Will push the word at address X to the top of the stack. The stack pointer is updated automatically. Operation-type instructions do not need an address field in stack-organized computers. This is because the operation is performed on the two items that are on top of the stack. The instruction:

ADD

In a stack computer consists of an operation code only with no address field. This operation has the effect of popping the two top numbers from the stack, adding the numbers, and pushing the sum into the stack. Since all operands are implied to be in the stack, the need for specifying operands with the address field does not arise. Most of the computers comprise one of the three types of organizations. we will evaluate the arithmetic statement

X = (A + B)*(C + D)

Three Address Instructions

The three address instruction can be represented symbolically as

ADD A, B, C

Where A, B, C are the variables. These variable names are assigned to distinct locations in the memory. In this instruction operands A and B are called source operands and operand C is called destination operand and ADD is the operation to be performed on the operands. Thus the general instruction format for three address instruction is

Operation Source 1, Source 2, Destination

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand. The program in assembly language that evaluates X = (A + B) * (C + D) is shown below, together with comments that explain the register transfer operation of each instruction.

ADD R1, A, B R1 M[A] + M[B] ADD R2, C, D R2 M[C] + M[D] MUL X, R1, R2 M[X] R1 * R2

It is assumed that the computer has two processor registers, Rl and R2. The symbol M[A] denotes the operand at memory address symbolized by A.

The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

The number of bits required to represent such instruction include :

1. Bits required to specify the three memory addresses of the three operands. If n-bits are required to specify one memory address, 3n bits are required to specify three memory addresses.
2. Bits required to specify the operation.

Two Address Instructions

The two address instruction can be represented symbolically as

ADD A, B

This instruction adds the contents of variables A and B and stores the sum in variable B destroying its previous contents. Here, operand A is source operand however operand B serves as both source and destination operand. The general instruction format for two address instruction is

Operation Source, Destination

Two-address instructions are the most common in commercial computers. Here again each address field can specify either a processor register or memory word. The program to evaluate X = (A + B)* (C + D) is as follows:

MOV R1, A R1 M[A] ADD R1, B
R1 RI+M[B] MOV R2, C R2 M[C]
ADD R2, D R2 R2+M[D] MUL R1,
R2 R1 R1*R2
MOV X, R1 M[X] R1

The MOV instruction moves or transfers the operands to and from memory and processor registers. The first symbol listed in an instruction is assumed to be both a source and the destination where the result of the operation transferred.

To represent this instruction less number of bits are required as compared to three address instruction. The number of bits required to represent two address instructions include:

1. Bits required specifying the two memory addresses of the two operands, i. e. 2n bits.
2. Bits required specifying the operation.

One Address Instruction

The one address instruction can be represented symbolically as

ADD B

This instruction adds the contents of variable A into the processor register called accumulator and stores the sum back into the accumulator destroying the previous contents of the accumulator. In this instruction the second operand is assumed implicitly in a unique location accumulator. The general instruction format for one address instruction is

Operation Source
One-address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication and division there is a need for a second register. However, here we will neglect the

second register and assume that the AC contains the result of all operations. The program to evaluate X=(A+B)*(C+D) is:

LOAD A AC ?M[A] ADD B
AC?AC+M[B] STORE T M[T] ? AC
LOAD C AC ? M[C] ADD D, A
AC?AC+M[D] MUL T AC ?AC *
M[T] STORE X M[X] ?AC

All operations are done between the AC register and a memory operand. T is the address of a temporary memory location required for storing the intermediate result

Few more examples of one address instructions are:

LOAD A: This instruction copies the contents of memory location A into the M accumulator.
STORE B: This instruction copies the contents of accumulator into memory location B.

Zero Address Instructions

In these instructions, the locations of all operands are defined implicitly. Such instructions are found in machines that store operands m a structure called a pushdown stack. A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack. The following program shows how X = (A + B) * (C + D) will be written for a stack organized computer. (TOS stands for top of stack.)

PUSH A TOS?A PUSH B
TOS?B ADD T TOS?(A+B)
PUSH C TOS?C PUSH D
TOS?D ADD T TOS?(C+D)
MUL T TOS?(C+D)*(A+B)
POP T M[X]?TOS

To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse Polish notation. The name “zero-address” is given to this type of computer because of the absence of an address field in the computational instructions.

The instruction with only one address will require less number of bits to represent it, and instruction with three addresses will require more number of bits to represent it. Therefore, to access entire instruction from the memory, the instruction with three addresses requires more memory accesses while instruction with one address requires less memory accesses. The speed of instruction execution is mainly depend on how much memory accesses it requires for the execution. If memory accesses are more time is required to execute the instruction. Therefore, the execution time for three address instructions is more than the execution time for one address instructions.

To have a less execution time we have to use instructions with minimum memory accesses. For this instead of referring the operands from memory it is advised to refer operands from processor registers.

ALU

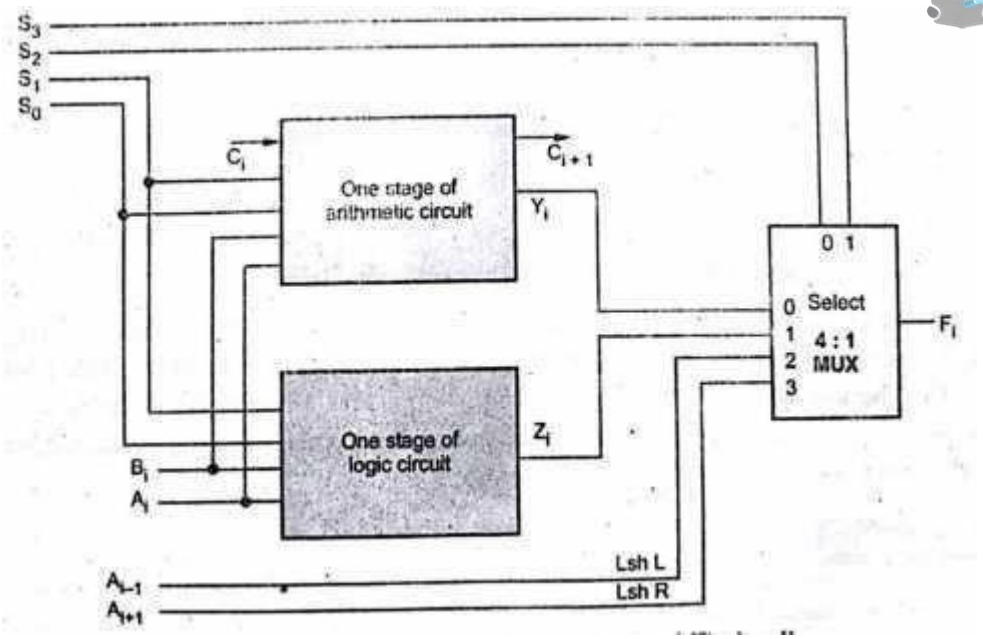


Figure 8. One stage of arithmetic logic shift circuit

The design of arithmetic and logic circuits for a simple ALU. By combining both Circuits with the help of multiplexer we can get the arithmetic and logic circuit, as shown in the Fig. 8. The Fig. 8 shows one stage of an arithmetic logic shift unit. The subscript i designates a typical stage. Inputs Ai and Bi are applied to both the arithmetic and logic units. Inputs S_1 and S_0 are used to select a particular micro- operation. A 4:1 multiplexer at the output chooses between an arithmetic output in Y_i and a logic output in Z_i. The data in the multiplexer are selected with inputs S_3 and S_2. The other two data inputs to the multiplexer receive inputs A_{i-1} for shift left operation and A_{i+1} for the right shift operation.

I/O SYSTEM BUS

The central processing unit memory unit and I/O unit are the hardware components/modules of the computer. They work together with communicating each other and have paths for connecting the modules together. The collection of paths connecting the various modules is called the interconnection structure. A group of wires called bus is used to provide necessary signals for communication between modules. A bus is a shared transmission medium, it must only be used by one device at a time and when used to connect major computer components (CPU, memory, I/O) is ‘ called a system bus.

The system bus is separated into three functional groups: data bus, address bus and control bus

Data lines (data bus) - Move data between system modules. The data bus lines are bidirectional CPU can read data on these lines from mentor or from a port, as well as send data out on these lines to a memory location or to a port. Width is a key factor. It determines number of bytes that can be transferred in one cycle and hence the overall system performance.

Address lines (address bus) - Designate source or destination of data on the data bus. It is an unidirectional bus. Width determines the maximum possible memory capacity of the system. It also used to address I/O ports. Typically: High-order bits select a particular module. Lower order bits select a memory location or I/0 port within the module.

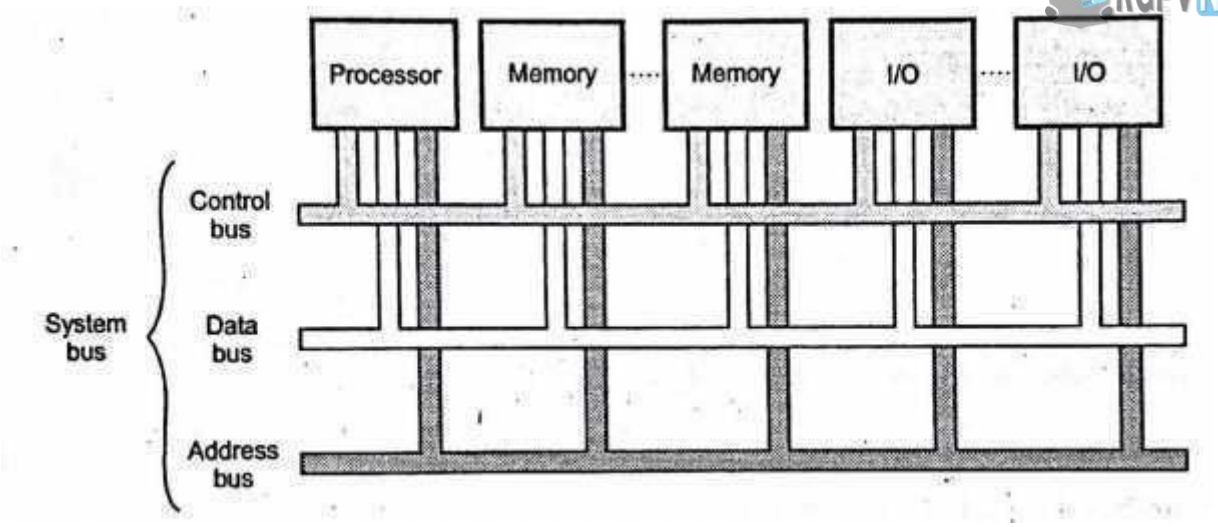


Figure 9. Bus Interconnection scheme

Control lines (Control bus) - Control access to use the data and address lines. Typical control lines include:

1. Memory Read and Memory write
2. I/O Read and [/0 Write
3. Transfer ACK
4. Bus Request and Bus Grant
5. Interrupt Request and Interrupt ACK
6. Clock & Reset

If one module wishes to send data to another, it must:

1. Obtain use of the bus
2. Transfer data via the bus

If one module wishes to request data from another, it must _:

1. Obtain use of the bus
2. Transfer a request to the other module over control and address lines
3. Wait for second module to send data

Connecting I/O Devices to CPU and Memory

Fig 10. Shows the bus interconnection scheme. It shows that how I/0 devices are connected to CPU and memory. I/0 devices are interfaced to CPU through I/O interface or I/O module. The I/O interface consists of circuit, which connect an I/O device to a CPU bus. On one side of the interface we have the bus signals for address, data and control. On the other side we have a data path with its associated controls to transfer data between the interface and the I/O device. Usually I/O interface or I/O module is capable of interfacing more than one external device.

Since data, address and control buses are connected in parallel to CPU, memory and I/O the I/O module is allowed to exchange data directly with memory without going through the processor, using Direct Memory-Access (DMA). The bus interconnection scheme shown in Fig. 10 supports following types of data transfers:

1. Memory to processor - Memory read operation
2. Process to memory – Memory write operation
3. Processor to I/O – I/O write operation
4. I/O to processor - I/O read operation
5. I/O to or from memory- DMA operation

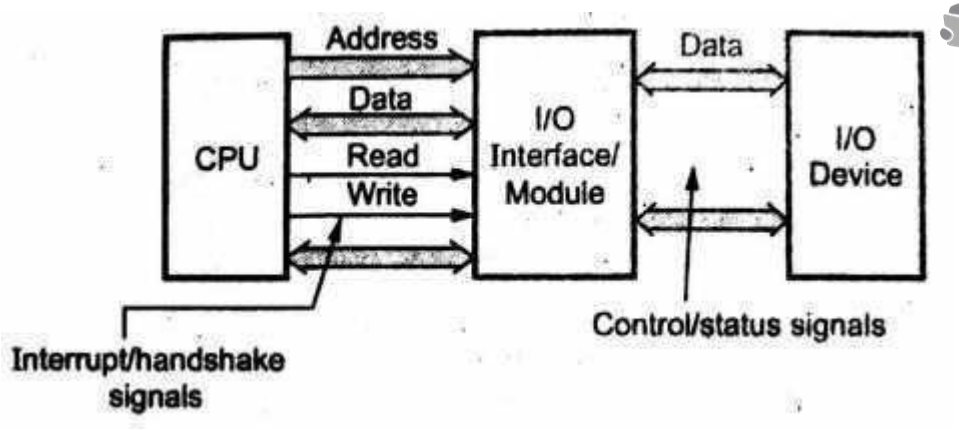


Figure 10. Connecting I/O device to the CPU

BUS STRUCTURE

A more efficient scheme for transferring information between registers in a multiple- register configuration is a common bus system. A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time. Control signals determine which register is selected by the bus during each particular register transfer. A common bus system can be constructed using multiplexers. These multiplexers select the source register whose binary information is then placed on the bus. The system bus is a cable which carries data communication between the major components of the computer, including the microprocessor. Not all of the communication that uses the bus involves the CPU, although naturally the examples used in this tutorial will centre on such instances. The system bus consists of three different groups of wiring, called the data bus, control bus and address bus. These all have separate responsibilities and characteristics, which can be outlined as follows:

Control Bus

The control bus carries the signals relating to the control and co-ordination of the various activities across the computer, which can be sent from the control unit within the CPU. Different architectures result in differing number of lines of wire within the control bus, as each line is used to perform a specific task. For instance, different, specific lines are used for each of read, write and reset requests.

Data Bus

This is used for the exchange of data between the processor, memory and peripherals, and is bi-directional so that it allows data flow in both directions along the wires. Again, the number of wires used in the data bus (sometimes known as the 'width') can differ. Each wire is used for the transfer of signals corresponding to a single bit of binary data. As such, a greater width allows greater amounts of data to be transferred at the same time.

Address Bus

The address bus contains the connections between the microprocessor and memory that carry the signals relating to the addresses which the CPU is processing at that time, such as the locations that the CPU is reading from or writing to. The width of the address bus corresponds to the maximum addressing capacity of the bus, or the largest address within memory that the bus can work with. The addresses are transferred in binary format, with each line of the address bus carrying a single binary digit. Therefore the maximum address capacity is equal to two to the power of the number of lines present (2^lines).

Bus structure is divided in two types

1. Single Bus Structure

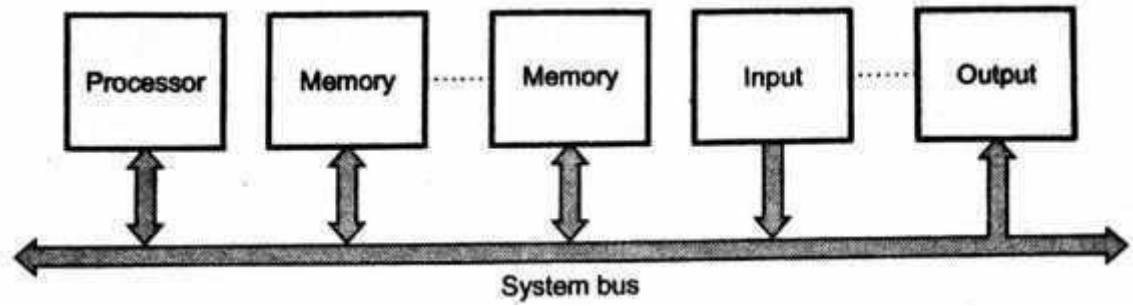


Figure 11. Single bus structure

Another way to represent the same bus connection scheme is shown in Fig. 11. Here address bus, data bus and control bus are shown by single bus called system bus. Hence such interconnection bus structure is called single bus structure. In a Single bus structure all units are connected to common bus called system bus. With single bus only two units can communicate with each other at a time. The bus control lines are used to arbitrate multiple requests for use of the bus. The main advantage of single bus structure is its low cost and its flexibility for attaching peripheral devices.

2. Multiple Bus Structure

Multiple architecture is divided in to two type. A great number of devices on a bus will cause performance to suffer due to

1. Propagation delay - the time it takes for devices to coordinate the use of bus.
2. The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity of the bus.

Now-a-days the data transfer rates for video controllers and network interfaces are growing rapidly. The need of high speed shared bus is impractical to satisfy with a single bus. Thus, most computer systems use the multiple buses. These buses have the hierarchical structure as shown in the Fig. 12

2.1 Traditional hierarchical bus architecture - The traditional bus connection uses three buses : local bus, system bus and expanded bus. Use of a cache structure insulates CPU from frequent accesses to main memory. Main memory can be moved off local bus to a system bus. Expansion bus interface is used to buffers data transfers between system bus and an I/O controller on expansion bus insulates memory-to-processor traffic from I/O traffic. Traditional hierarchical bus breaks down as higher and higher performance is seen in the I/O devices.

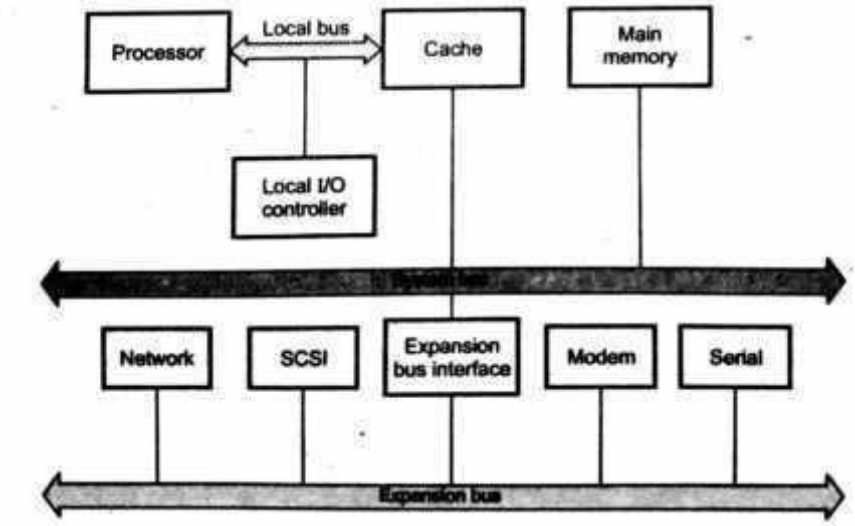


Figure 12 Traditional bus configuration

2.2 High-performance hierarchical bus I architecture - Specifically designed to support high—capacity I/O devices. It uses high-speed bus along with the three buses used in the traditional bus connection. Here cache controller is connected to high-speed bus. This bus supports connection to high-speed LANs, such as Fiber Distributed Data Interface (FDDI), video and graphics workstation controllers, as well as interface controllers to local peripheral buses including SCSI and P1394.Changes in processor architecture do not affect the high speed bus and vice versa

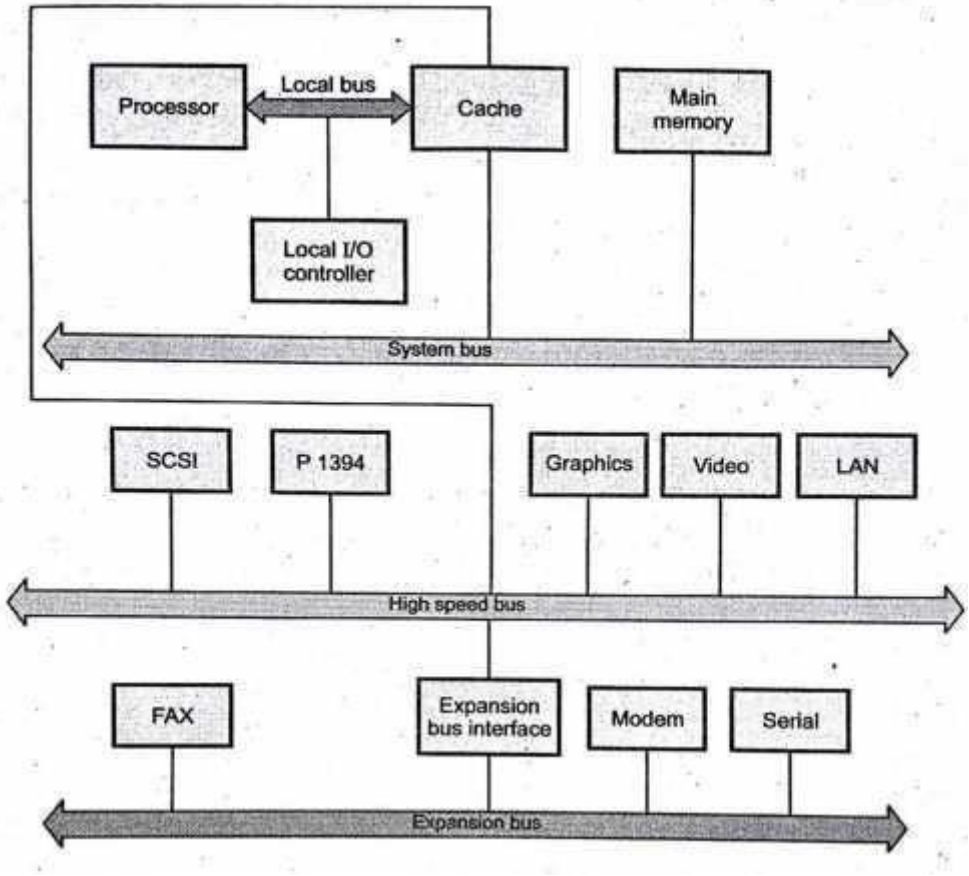


Figure 13 High-speed bus configuration

REGISTER TRANSFER LANGUAGE

The digital system can be constructed with flip flops and gates using sequential logic design. However, it is difficult redesign large digital systems using this approach. To overcome this difficulty, large digital systems are designed using modular approach. In the modular approach, system is partitioned into subsystems such that each subsystem performs some functional task. Such modules are constructed from registers, counters, decoders, multiplexers, arithmetic and logic elements and control Logic. The modules are then interconnected with common data and control paths to form a digital system.

The features of register transfer logic

1. Uses registers as a primitive component in the digital system instead of flip-flops and gates.
2. The information flow and processing tasks among the data stored in the registers is described in a concise and precise manner.
3. Uses a set of expressions and statements which resemble the statements used in programming languages.

4. The presentation of digital functions in register transfer logic is very user friendly.

Basic Components of Register Transfer Logic

The register transfer logic method uses four basic components to describe digital systems. These are as follows:

1. Registers and their functions: The total number of registers in the system and their functions includes all its counterparts such as shift registers, counters and memory unit. Counter is register, which increments the information stored in it by
   1. A memory unit is a set of storage registers where information can be stored.

2. Information: The information is stored in the registers may be binary numbers, BCD numbers, alphanumeric characters, control information or any other binary coded information.

3. Operations: The operations performed on the information stored in the registers.
   The operations performed on the data stored in registers are called micro-operations and it is performed in one clock cycle. The operation may be arithmetic operation or logical, operation. A statement that requires a sequence of micro-operations for its implementation is called a macro-operation.

4. Control function: The control functions that activate operations and central the sequence of operations. They consists of timing signals that sequence the operations one at a time the control function is basically a binary variable, when it is logic 1 it initiates the operation otherwise it inhibits the operation.-

Definition of Register Transfer Language

A micro operation is an elementary operation performed on the information stored in one or more registers. The result of the operation may replace the previous binary information of a register or may by transfer to another register. The symbolic notation used to describe the micro operation transfer among registers is called a *register transfer language*. The term "register transfer" implies the availability of hardware logic circuits that can perform stated micro operation and transfer the results to the operation to the same or another register. Examples of micro-operation are lead, store, clear, shift, and count and so on. Sequences of micro- operations are performed to perform one complete operation. For example, to add two numbers following micro-operation sequence has to be performed.

1. Load first number in register 1
2. Load second number in register 2.
3. Perform add micro-operation.
4. Store the result in the destination register.

As shown above it is possible to specify the sequence of micro-operation in a word, but it involves lengthy descriptive explanation. Hence, it is preferred to use symbolic notations to describe the micro-operations. These symbolic notations are also called a register transfer language or computer hardware description language.

The micro-operations used in the digital system can be classified as:

1. Register transfer micro-operations: The micro-operations that transfer information from one register to another. 2. Arithmetic micro-operations: The micro-operations that perform arithmetic operations on numeric data stored in registers.
3. Logic micro-operation: The micro-operations that perform bit manipulation operations on non numeric data stored in registers.

4. Shift micro-operations: The micro-operations that perform shift operations on data stored in registers.

1. Arithmetic Micro-operations

These micro-operations perform some basic arithmetic operations on the numeric data stored in the registers. These basic operations may be addition, subtraction, incrementing a number, decrementing a number and arithmetic shift operation. An **'add'** micro-operation can be specified as: R3 ? R1 + R2

It implies: add the contents of registers R1 and R2 and store the sum in register R3. The add

operation mentioned above requires three registers along with the addition circuit in the ALU. Subtraction is implemented through complement and addition operation as:

R3 ? R1 − R2 is implemented as
R3 ? R1 + (2's complement of R2)
R3 ? R1 + (1's complement of R2 + 1)
R3 ? R1 + R2 + 1

An increment operation can be symbolized as: R1 ? R1 + 1

While a decrement operation can be symbolized as: R1 ? R1 − 1

We can implement increment and decrement operations by using a combinational circuit or binary up/down counters. In most of the computers multiplication and division are implemented using add/subtract and shift micro-operations. If a digital system has implemented division and multiplication by means of combinational circuits then we can call these as the micro-operations for that system. An arithmetic circuit is normally implemented using parallel adder circuits. Each of the multiplexers (MUX) of the given circuit has two select inputs. This 4-bit circuit takes input of two 4-bit data values and a carry-in-bit and outputs the four resultant data bits and a carry-out-bit. With the different input values we can obtain various micro-operations.

Equivalent micro-operation	Micro- operation name
R ? R1 + R2	Add
R ?R1 + R2 +1	Add with carry
R ?R1 + R2	Subtract with borrow
R ?R1 + 2's	Subtract
R ?R1	Transfer
R ?R1 + 1	Increment
R ?R1 − 1	Decrement

2. Logic Micro-operations

These operations are performed on the binary data stored in the register. For a logic micro-operation each bit of a register is treated as a separate variable. For example, if R1 and R2 are 8 bits registers and

R1 contains 10010011 and
R2 contains 01010101

R1 AND R2 00010001

Some of the common logic micro-operations are AND, OR, NOT or complements Exclusive OR, NOR, NAND. We can have four possible combinations of input of two variables. These are 00, 01, 10 and 11. Now, for all these 4 input combination we can have 2^4 = 16 output combinations of a function. This implies that for two variables we can have 16 logical operations.

Logic Micro Operations

SELECTIVE SET

The selective-set operation sets to 1 the bits in register A where there are corresponding 1's in register B. it does not affect bit positions that have 0's in B. the following numerical example clarifies this operation:-

```
1010        A before
1100        B (logic operand)
----------------------------
1110        A after
```

It is clear that the OR micro-operation can be used to selectively set the bits of a register.

SELECTIVE COMPLEMENT

The selective-complement operation complements bits in register A where there are corresponding 1's in register B. it does not affect bit positions that have 0's in B. The following numerical example clarifies this
operation:-

```
1010        A before
1100        B (logic operand)
----------------------------
0110        A after
```


Hence the exclusive OR micro-operation can be used to selectively complement bits of a register.

SELECTIVE CLEAR

The selective-clear operation clears to 0 the bits in register A only where there are corresponding 1's in register B. For example:-

```
1010        A before
1100        B (logic operand)
----------------------------
0010        A after
```

Hence the logic micro-operation corresponding to this is: A B ^ B

MASK OPERATION

The mask operation is similar to the selective-clear operation except that the bits of A are cleared only where there are corresponding 0's in B. the mask operation is an AND micro operation, for example:-

```
1010        A before
1100        B (logic operand)
----------------------------
1000        A after masking
```

The two right most bits of A are cleared because the corresponding bits of B are 0's.
The two right most bits are left unchanged due to the corresponding bits of B (i.e.
1). The mask operation is more convenient to use than the selective clear because most computers provide an AND instruction, and few provide an introduction that executes the micro operation is an AND micro operation.

INSERT OPERATION

The insert operation inserts a new value into a group of bits. This is done by first masking the bits and then ORing them with the required value. For example, suppose that an A register contains eight bits, 0110 1010. To replace the four unwanted bits:-

```
0110 1010        A before
0000 1111        B (mask)
---------------------------
0000 1010        A after masking
```

and then insert the new value by replace the four leftmost bits by the value 1001

```
0000 1010        A before
1001 0000        B (insert)
---------------------------
1001 1010        A after insertion
```

The mask operation is an AND micro operation and the insert operation is an OR micro operation.

CLEAR OPERATION

The clear operation compares the words in A and B and produces an all 0's result if the two numbers are equal. This operation is achieved by an exclusive-OR micro operation as has own by the following example:

```
1010          A
1010          B
-----------------
0000     ←    A⊕     A  B
```

When A and B are equal, the two corresponding bits are either both 0 or both 1. In either case the exclusive-OR operation produces a 0. The all-0's result is then checked to determine if the tow numbers were equal.

3. Shift Micro operations

Shift micro operation can be used for serial transfer of data. They are used generally with the arithmetic, logic, and other data-processing operations. The contents of a register can be shifted to the left or the right. During a shift-right operation the serial input transfers a bit into the leftmost position. The serial input transfers a bit into the rightmost position during a shift-left operation. There are three types of shifts, logical, circular and arithmetic.

TABLE 4-7 Shift Microoperations

Symbolic designation	Description
$R \leftarrow \text{shl } R$	Shift-left register R
$R \leftarrow \text{shr } R$	Shift-right register R
$R \leftarrow \text{cil } R$	Circular shift-left register R
$R \leftarrow \text{cir } R$	Circular shift-right register R
$R \leftarrow \text{ashl } R$	Arithmetic shift-left R
$R \leftarrow \text{ashr } R$	Arithmetic shift-right R

Logical shift

A logical shift operation is one that transfers 0 through the serial input. We use the symbols shl and shr for logical shift left and shift right micro operations, e.g.

R1 C shl R1
R2 C shr R2

are the two micro operations that specify a 1-bit shift left of the content of register R1 and a 1- bit shift right of the content of register R2. In logical shift 1change in 0 and 0 change in 1.

Circular shift

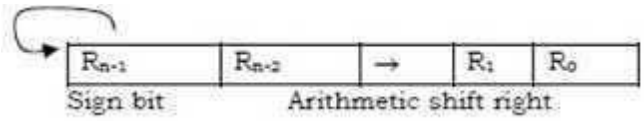
The circular shift is also known as rotate operation. It circulates the bits of the register around the two ends and there is no loss of information. This is accomplished by connecting the serial output of the shift register to its serial input. We use the symbols cil and cir for the circular shift left and circular shift right. In circular shift while shifting value not change only position change. E.g. suppose Q1 register contains 01101101 then

Q1 register bit 01101101 *Q1 register bit*

*0*1101101
After cir operation, it contains *1*0110110 And after cil operation it will contain 1101101*0*.

Arithmetic Shift

An arithmetic shift micro operation shifts a signed binary number to the left or right. The effect of an arithmetic shift left operation is to multiply the binary number by 2. Similarly an arithmetic shift right divides the number by 2. Because the sign of the number must remain the same, arithmetic shift-right must leave the sign bit unchanged, when it is multiplied or divided by 2. The left most bit in a register holds the sign bit, and the remaining bits hold the number. The sign bit is 0 for positive and 1 for negative. Negative numbers are in 2's complement form. Following figure shows a typical register of n bits.



Bit R_{n-1} in the left most position holds the sign bit. R_{n-2} is the most significant bit of the number and R0 is the least significant bit. The arithmetic shift-right leaves the sign bit unchanged and shifts the number (including the sign bits) to the right. Thus R_{n-1} remains the same; R_{n-2} receives the bit from R_{n-1}, and so on for other bits in the register. In arithmetic shift value change 1 from 0 and 0

from 1, 1 indicate negative and 0 indicate positive, if negative change in positive we say overflow otherwise vice versa.

Register transfer

The symbolic notation used to describe the micro-operation belongs to transfers among registers forms a syntax/ statement of a register transfer language. For example, the statement denotes a transfer of the content of register R1 into register R2. After transfer the contents of register R2 are replaced by the content of register R1, however the content of register R1 remain unchanged. The term register transfer implies the availability of hardware logic circuit that can perform a stated micro-operation and transfer the result of operation to the same or another register. Therefore just like various programming languages we have register transfer language to specify the micro-operations. Each statement in the register transfer language specify the unique micro-operation.

A register transfer language also allows to specify control conditions in the statement. To specify control conditions it uses control variable along with the colon at the leftmost side of the statement as shown below.

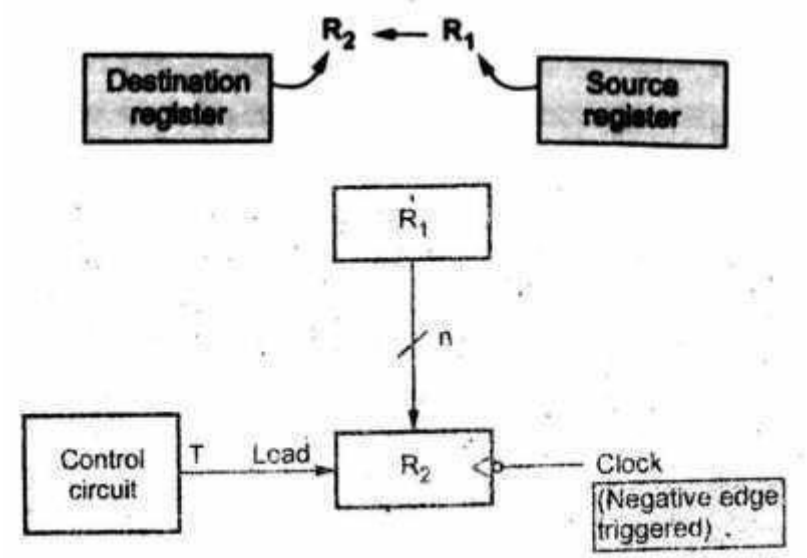


Figure 14 block diagram for representing R2<— R1 micro-operation

Here, T is used as a control variable it is basically a Boolean variable having value equal to 1 or 0. This statement indicates that the content of R1 are transferred to R2 only when T=1; otherwise transfer operation is not performed. In a computer, micro-operations are synchronized with the help of clock signal. The micro- operation is initiated either at the rising edge or at the falling edge of the clock signal depending on the circuit.

It is important to note that the clock is not included as a variable in the register transfer statements. It is assumed that all transfers occur during a clock edge transition either positive or negative. If hardware permits two micro-operations can be executed at the same time. For example, statement exchanges the contents of two registers, they perform R2 R1, R3 R4 micro operations simultaneously they are separated by coma in a statement.

BUS AND MEMORY TRANSFER

Bus Transfer

A digital computer has many registers and it is necessary to provide data path between them to transfer information from one register to another. If separate lines are used between each register,

there will be excess number of wires and controlling of those wires make circuit complex. Therefore, in multiple-register configuration a common bus system is used to transfer information between two registers. A common bus consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time control signals are used to determine which is the source register and which is a destination register for that particular transfer. The common bus scheme can be implemented in two ways

1. Using multiplexers.
2. Using tri-state bus buffers.

Implementation of common bus using multiplexers

The Fig. 15 shows the implementation of common bus system for four registers using multiplexers. Each register has four bits, numbered 0 through 3 and they are routed through multiplexers to the common bus. Here, four multiplexers are used to select four bits of the source register. Each multiplexers has four input lines, two select lines and one output line. The four input lines of multiplexer 0 (MUX 0) are connected to the bit 0 outputs of four registers such that bit 0 of register is connected to input 0, bit 0 of register 1 is connected to input 1, bit 0 of register 2 is connected to input 2 and bit 0 of register 3 is connected to input 3. Similarly, inputs for MUX 1 are connected to bit 1 outputs, inputs for MUX 2 are connected to bit 2, and inputs for MUX 3 are connected to bit 3 outputs of registers
0 through 3. To avoid the complexity of the diagram, only input connections for
MUX 3 are physically shown. To show other connections labels are used. Two same labels have connection between them.

The two selection lines S_1 and S_0 are connected to the selection inputs of all four multiplexers. These lines choose the four bits of one register and transfer them into the four-line common bus through OUT lines. When $S_1S_0 = 00$, the input 0 of all four multiplexers are selected and applied to the outputs to transfer them on the common bus. As a result a common bus receives the contents of register 0. Since the outputs of register 0 are connected to the input 0 of the multiplexers. Similarly, the content of register 1 are transferred on the common bus when S_1S_1. The Table below shows the register selection according to the status of S_1 and S_0 lines.

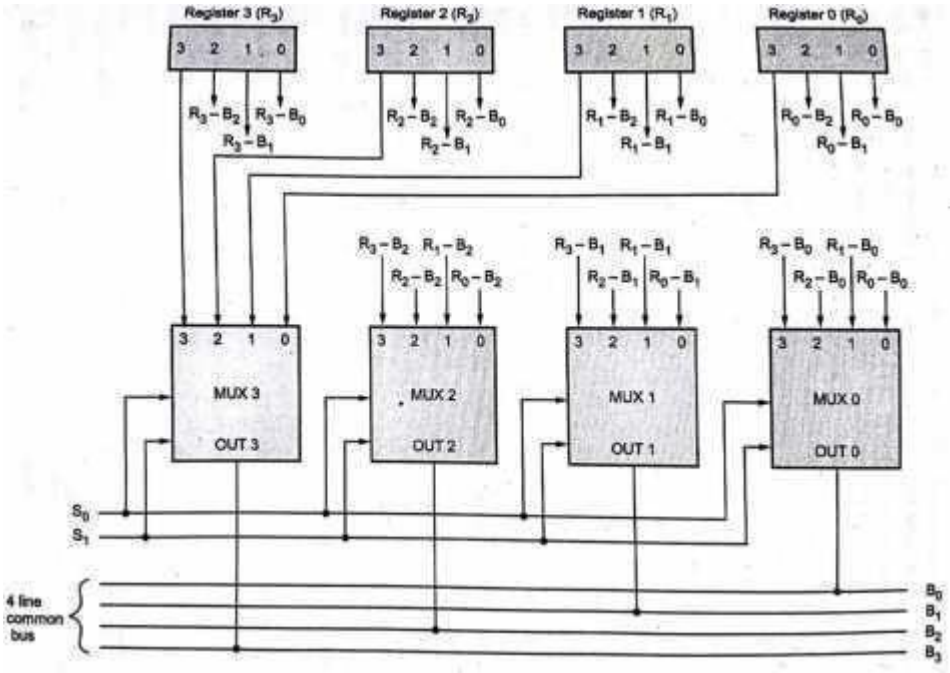


Figure 16. Computer bus System using multiplexer

S_1	S_0	Selected register
0	0	Register 0
0	1	Register 1
1	0	Register 2
1	1	Register 3

Table 1 Selection table

In general, common bus system will have n multiplexers and n line common bus where n represents the number of bits in each register and k number of inputs to each multiplexer where it represents the total number of registers. Therefore, the size of each multiplexer will be k x 1. The data transfer from a bus to any destination register can be accomplished by connecting the bus lines to the inputs. of all destination registers and activating the load control input of the particular (selected) destination register.

Implementing common bus using tri-state buffer

A common bus system can be implemented using tri-state or three state gates instead of multiplexers. A tri-state gate is a digital circuit that can have three output states: HIGH, LOW and high impedance. The high impedance state behaves like an open circuit, which means that the output is disconnected.

Bus is a common connection between number of registers and other units. Therefore, the data transfer through tri-state gates may require larger sinking and sourcing of current. The tri-state gate which provides more sinking and sourcing capacities is called tri-state buffer. The Fig. 17 shows the logic symbol for tri-state

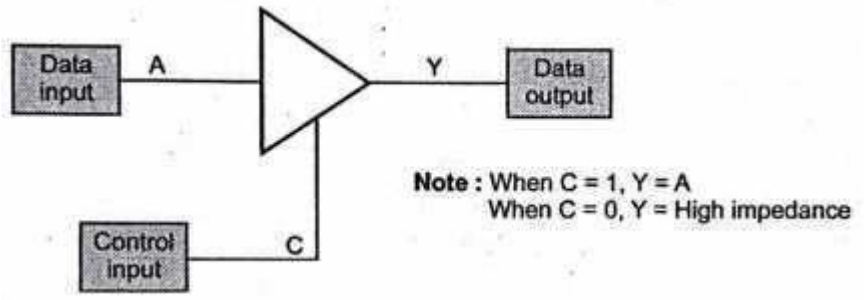
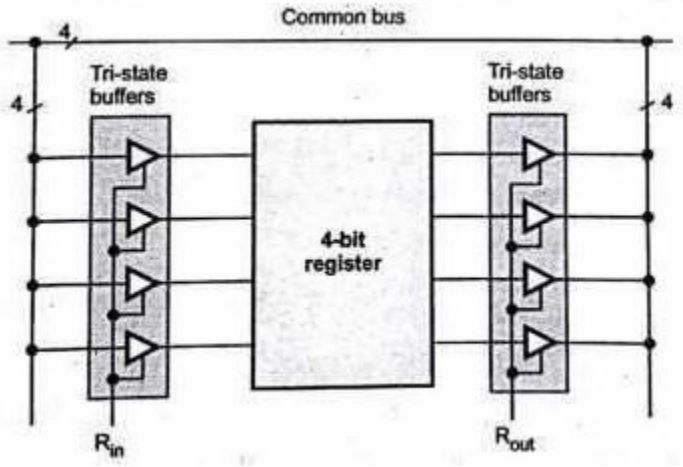


Figure 17 (a) Logic symbol of tri-state buffer



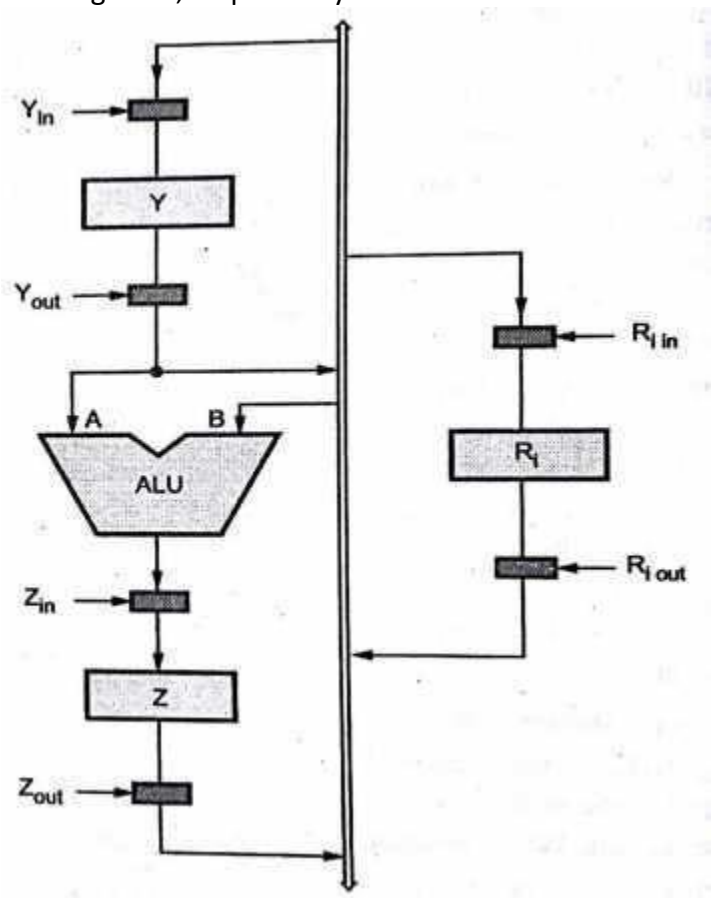
(b) working of tri state buffer with register

As shown in the Fig.17 the control input decides the state of the output. When control input is 1, the data is available at the output, i.e. Y = A. On the other hand, when control input is 0, the output gets to '0' high-impedance state.

The Fig. 17 (b) shows how the data transfer takes place between register and common bus. Here, 4-bit register is shown with tri-state buffers at input and output Sides. Each data line requires two tri-state buffers one at the input side and one at the output side. Therefore, for 8-bit register we require eight tri-state buffers at the input side and eight tri-state buffers at the output side.

As shown in the Fig. 17 all tri-state buffers at the input side are control by a common control signal R_{in}. When R_{in} is active the n-bit data from the common bus is loaded into the register. Similarly, at the output, a common signal R_{out} controls all output tri—state buffers. When R_{out} is active the 4-bit data from register is placed on the common bus.

The Fig. 18 shows the bus structure for the data transfer between various registers and the common bus. As shown in the Fig. 18 each register has input and output tri-state buffers and these tri-state buffers are controlled by corresponding control signals. Control signals R_{in} and R_{out} controls the input and output gating of register R. When R_{in} is set to 1, the data available on the common data bus is loaded into register $R_{i\ in}$. Similarly, when $R_{i\ out}$ is set to 1, the contents of register Ri are placed on the common data bus. The signals $R_{i\ in}$ and $R_{i\ out}$ are commonly known as input enable and output enable signals of registers, respectively.



Flg.18 Input and output tri-state buffers for the register

We can transfer the data from r1 register to R2 register by

1. Activate the output enable signal of R1, $R1_{out}$ = 1. This places the contents of R1 on the common bus.
2. Activate the input enable signal of R2, $R2_{out}$ = 1. This loads data from the common bus into the register R2.

All operations and data transfers within the processor take place in synchronization with the clock signal. The control signals which controls a particular transfer are activated either at the rising edge or at the falling edge of the clock cycle.

Memory Transfer

A memory is a collection of storage cells. Each cell store l-bit of information. The memory stores binary information in groups of bits called words. To access information from a particular word from main memory each word in the main memory has distinct address. This allows to access any word from the main memory by specifying corresponding address. The transfer of information from a memory word to the outside environment is called a read operation. The transfer of new information to be stored into the memory is called a write operation.

The number of words in the memory decides the size of the memory and the number of address bits. For example, 8-bit address can access upto 28 = 256 different words. The number of information bits can be read or written at a time is decided by the word length (numbers of bits in one word) of the memory. The Fig. 19 shows the typical connection between processor and memory. As shown in the Fig. 1.11.7 the control lines from the processor decides the memory operation.

In case of read operation Read signal is activated. It is used to enable the active low output enable signal of the memory. In case of write operation Write signal is activated to indicate the write operation. The data transfer between the memory and processor takes place through the use of two processor registers, usually called MAR (Memory Address Register) or simply AR (Address Register) and MDR (Memory Data Register) or simply DR (Data Register).

If MAR is k-bit long and MDR is n-bit long, it is possible to access upto 2^k memory locations and during one memory cycle, it is possible to transfer n bit data. In read operation, the address of the memory word is specified by address register, AR and the data word read from the memory is loaded into the data register, DR. This read operation can be represented as,

Read : DR ?M [AR]

Letter M represents memory word whose address is specified by address register, AR. The control signal Read indicates that the operation is performed only when Read signal is active. Once the data is available in the DR register it can be transferred to any other register within the CPU.

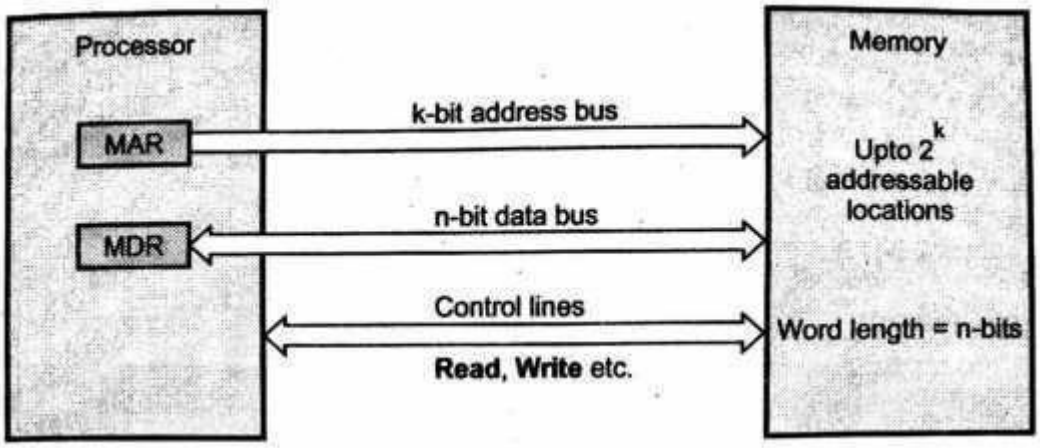


Figure 19. Connection between memory and processor

It is important to note that for above operation the activation of Read signal-is not necessary; it is necessary only when data is read from memory unit. The write operation transfers the contents of a

data register to a memory word M selected by the address in the address register, AR. The write operation can be represented as,

Write : M [AR] ← DR

The control signal Write indicates that the operation is performed only when Write signal is active. When it is necessary to transfer data from any other CPU register to the memory. we have to transfer the data from that register to the data register, DR before write operation.

The Fig. 20 shows the communication between memory unit and multiple registers. Register A0 through A3 are the address registers. The MUX 1 selects one of the address source for memory. The MUX 2 selects one of data source for write operation of memory. The decoder selects the destination register to read data from memory.

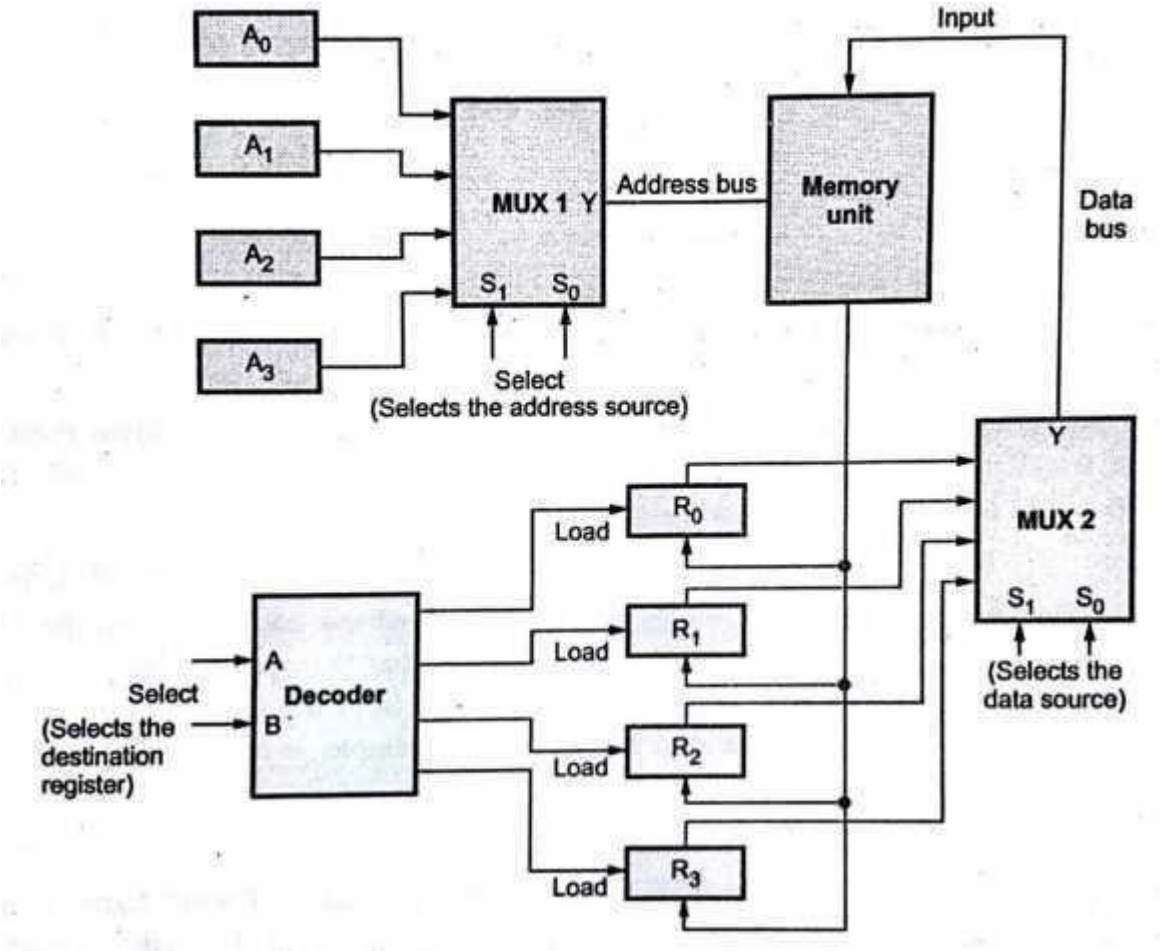


Figure 20 Communication between memory unit and multiple register

ADDRESSING MODES

The address of the memory can be specified directly within the instruction. For example, MOV [2000H], R1. In this instruction the memory address is fix, it can not be dynamically changed in the program itself. There are some situations where we need to change the memory address dynamically. For example program of array, to add the data and get the total sum of all array elements. In this we have to repeat the add instruction number of times equal to the array elements and each time we have to add a number from a new successive memory location. Every time the address of memory is different. So to change the address of memory each time when we enter the loop address variable is used. Such addressing is called indirect addressing.

The operation to be performed is specified by the operation field of the instruction. The execution of the operation is performed on some data stored in computer registers or memory words and the

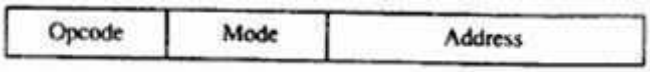
way the operands are chosen during program. Selection of operands during program execution depends on the addressing mode of the instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referred. The purpose of using address mode techniques by the computer is to accommodate one or both of the following provisions:

1. To give programming flexibility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.
2. To reduce the number of bits in the addressing field of the instruction.

Part of the programming flexibility for each processor is the number and different kind of ways the programmer can refer to data stored in the memory or I/O device. The different ways that a processor can access data are referred to as addressing schemes or addressing modes. Usage of addressing modes lends programming versatility to the user and helps to write program data mode affection in terms of instruction and implementation. The basic operation cycle of the computer must be understood to have a thorough knowledge of addressing modes. The instruction cycle is executive in the control unit of the computer and is divided into three principal phases:

1. Fetch the instruction from memory.
2. Decode the instruction.
3. Execute the instruction.

The program counter or PC register in the computer hold the instruction in the program stored memory. Each time when instruction is fetched from memory the PC is incremented, for it holds the address of the instruction to be executed next. Decoding involves determination of the operation to be performed, the addressing mode of the instruction, and the returns of the operands. The computer then executes the instruction and returns to step 1 to fetch the next instruction in sequence. Figure show the distinct addressing mode field of in instruction format.



Instruction format with mode field

The operation code (opcode) specifies the operation to be performed. The mode field issue to locate the operands needed for the operation. An address field in an instruction may or may not be present.

1. Implied Mode: This mode specify the operands implicitly in the definition of the instruction. For example, the instruction ‘‘complement accumulator’’ is an implied mode instruction because the operand in the accumulator register is implied in the definition of the instruction. In fact, all register references instructions that use an accumulator are implied mode instructions. Zero-address introductions are implied mode instructions.

2. Immediate Mode: The operand is specified in the instruction itself in this mode i.e. the immediate mode instruction has an operand field rather than an address field. The actual operand to be used in conjunction with the operation specified in the instruction is contained in the operand field.
   Example : MOVE A, #20

3. Register Mode: In this mode the operands are in registers that reside within the CPU. The register required is chosen from a register field in the instruction. Example: MOV R1, R2

4. Register Indirect Mode: In this mode the instruction specifies a register in the CPU that contains the address of the operand and not the operand itself. Usage of register indirect mode instruction necessitates the placing of memory address of the operand in the processor register with a previous instruction. EA=R

   Example: MOVE A, (R0)

5. Auto increment or Auto decrement Mode: After execution of every instruction from the data in memory it is necessary to increment or decrement the register. This is done by using the increment or decrement instruction. Given upon its sheer necessity some computers use special mode that increments or decrements the content of the registers automatically.

   Example: MOVE (R2), + R0
   MOVE (R2), - R0

6. Direct Address Mode: In this mode the operand resides in memory and its address is given directly by the address field of the instruction such that the affective address is equal to the address part of the instruction. Example: MOVE A, 2000

7. Indirect Address Mode: Unlike direct address mode, in this mode give the address field gives the address where the effective address is stored in memory. The instruction from memory is fetched through control to read is address part to access memory again to read the effective address. A few addressing modes require that the address field of the instruction be added to the content of a specific register in the CPU. The effective address in these modes is obtained from the following equation:

   Effective address = address part of instruction + context of CPU register

   The CPU Register used in the computation may be the program counter, Index Register or a base Register.

8. Relative Address Mode: This mode is applied often with branch type instruction where the branch address position is relative to the address of the instruction word itself. As such in the mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address whose position in memory is relative to the address of the next instruction. Since the relative address can be specified with the smaller number of bits then those required the design the entire memory address, it results in a shorter address field in the instruction format. EA = PC + Address part of instruction

9. Indexed Addressing Mode: In this mode the effective address is obtained by adding the content of an index register to the address part of the instruction. The index register is a special CPU register that contains an index value and can be incremented after its value is used to access the memory. EA= offset + R

   Example: MOVE 20 [R1], R2

10. Base Register Addressing Mode: In this mode the effective address is obtained by adding the content of a base register to the part of the instruction like that of the indexed addressing mode though the register here is a base register and not a index register.

    The difference between the two modes is based on their usage rather than their computation. An index register is assumed to hold an index number that is relative to the address part of the instruction. A base register is assumed to hold a base address and the address field of the instruction, and gives a displacement relative to this base address. The base register addressing mode is handy for relocation of programs in memory to another as required in multi programming systems. The address values of instruction must reflect this change of position with a base register, the displacement values of instructions do not have to change. Only the value of the base register requires updating to reflect the beginning of a new memory segment.

Mode	Assembly Convention	Register Transfer
Direct Address	LD ADR	AC<-M[ADR]
Indirect Address	LD @ADR	AC<-M[M[ADR]]
Relative Address	LD $ADR	AC<-M[PC+ADR]
Immediate Operand	LD #NBR	AC<-NBR
Index Addressing	LD ADR(X)	AC<-M[ADR+XR]
Register	LD R1	AC<-R1
Register Indirect	LD(R1)	AC<-M[R1]
Auto increment	LD (R1)	AC<-M[R1],R1<-R1+1

UNIT – 2

Control Unit Organization - Basic Concept of Instruction, Instruction Types, Micro Instruction Formats, Fetch and Execution cycle, Hardwired control unit, Micro-programmed Control unit - micro program sequencer Control Memory, Sequencing and Execution of Micro Instruction.

BASIC CONCEPT OF INSTRUCTION

The user of a computer can control the process by means of a program. A program is a set of instruction that specify the operations, operands and the sequence by which processing has to occur. The data processing task may be altered by specifying a new program with different instructions or specifying the same instructions with different data. A computer instruction is a binary code that specifies a sequence of micro-operations for the computer. Instruction codes together with da ta are stored in memory. The computer reads each instruction from memory and places it in a control register. The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of micro-operations. Every computer has its own unique instruction set. The ability to store and execute instructions, the stored program concept, is the most important property of a general purpose computer. The operation of the central processing - unit and the computer system is determined by the instructions executed by central processing unit. These instructions are known as machine instructions or computer instructions. Machine instructions are in the form of binary codes. A particular sequence of these binary codes used to perform particular task is known as machine language program. Each instruction of the CPU has specific information fields, which are required to execute it. These information fields of instructions are called elements of instruction.

An instruction code of a group of bits that instruct the computer to perform a specific operation. It is usually divided into parts, each having its own particular interpretation. The most basic part of an instruction code is its operation part. The operation code of an instruction is a group p of bits that define such operations as add, subtract, multiply, shift, and complement. The number of bits required f or the operation code of an instruction depends on the total number of operations available in the computer. The operation code must consist of at least n bits f or a given 2"(or less) distinct operations.

An operation is part of an instruction stored in computer memory. It is a binary code that tells the computer to perform a specific operation. The control unit receives the instruction from memory and interprets the operation code bits. It then issues a sequence of control signals to initiate micro-operations in internal computer registers. For every operation code, the control issues a sequence of micro-operations needed f or the hard- ware implementation of the specified operation. For this reason, an operation code is sometime s called a macro-operation because it specifies a set of micro- operations.

The operation part of an instruction code specifies the operation to be performed. This operation must be performed on some data stored in processor registers or in memory. An instruction code must therefore specify not only the operation but also the registers or the memory words where the operands are to be found, as well as the register or memory word where the result is to be stored. Memory words can be specified in instruction codes by their address. Processor register s can be specified by assigning to the instruction another binary code of k bits that specifies one of 2 k registers. There are many variations f or arranging the binary code of instructions, and each computer has its own particular instruction code format. Instruction code formats are conceived by computer designers who specify the architecture of the computer

INSTRUCTION TYPES & FORMATS

A basic computer has **three types** of instructions:

1. Memory reference instructions
2. Register reference instructions
3. Input-Output instructions

Memory Reference Instructions - The Fig.1 shows the instruction format for memory reference instructions. Each memory reference instruction has 16-bits. Out of 16-bits:

l-Bit (I) specifies addressing mode : Direct or indirect
3-Bits (Opcode) specify the opcode and
12-Bits (Address) specify the address.

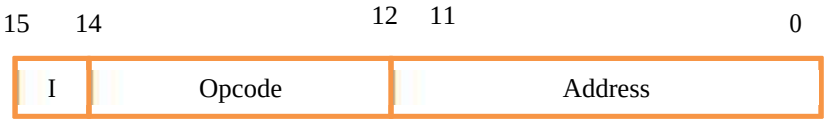


Figure. 1 Memory reference Instruction format

I = 0 : Direct addressing mode
I = 1 : Indirect addressing mode
Opcode : It can have value from 000 through 110 since there are 7 memory reference instructions

Example:

Mnemonic	**Description**	**Instruction Code in HEX**	
AND	Logically ANDs the contents of Specified memory location and AC AC ?AC + M {AR}	**I=0** 0xxx	**I=1** 8xxx
STA	Store the contents of AC in the specified memory location M [AR]?AC	1xxx	9xxx

Register Reference Instructions - A register reference instruction specifies an operation or a test to be performed with AC register. These instructions do not need to access memory and hence 12-bits are used to specify an operation or a test to be performed.

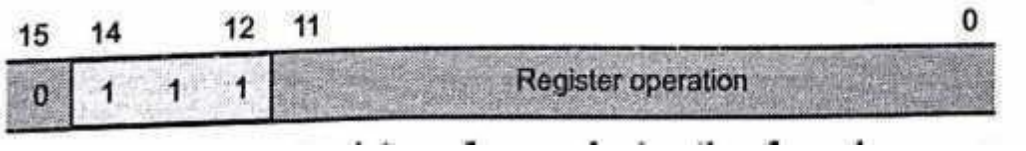


Figure 2. Register reference instruction format

Example:

Mnemonic	**Description**
CME	Complement E bit

INC	Increment AC register
SPA	Skip next instruction if contents of AC register are +ive

Input-Output Instructions - Like register reference instructions, input-output instructions do not need memory reference. The opcode and I bit for these instructions are 111 and 1, respectively. The remaining 12-bits specify the type of input-output operation or test to be performed.

Figure 3. Input-output instruction format

Example:

Mnemonic	**Description**
INP	Load a character in AC register from input port
OUT	Send a character to outpot port from AC register

FETCH AND EXECUTION CYCLE

Instruction Cycle

The most basic unit of computer processing in the simplest form, consists of two parts.

1. ***Opcode*** (operation code) − a portion of a machine language instruction that specifies the operation to be performed.
2. ***Operands*** − a part of a machine language instruction that specifies the data to be operated on

The simplest model of instruction processing can be a two step process. The CPU **reads (fetches)** instructions (codes) from the memory one at a time, and **executes**. Instruction fetch involves reading of an instruction from a memory location to the CPU register. The execution of this instruction may involve several operations depending on the nature of the instruction. Instructions are processed by the control unit in a systematic, step-by-step manner. The sequence of steps in which instructions are loaded from memory and executed is called instruction cycle. Each step in the sequence is referred to as a phase. Fundamentally, there are 6 phases.

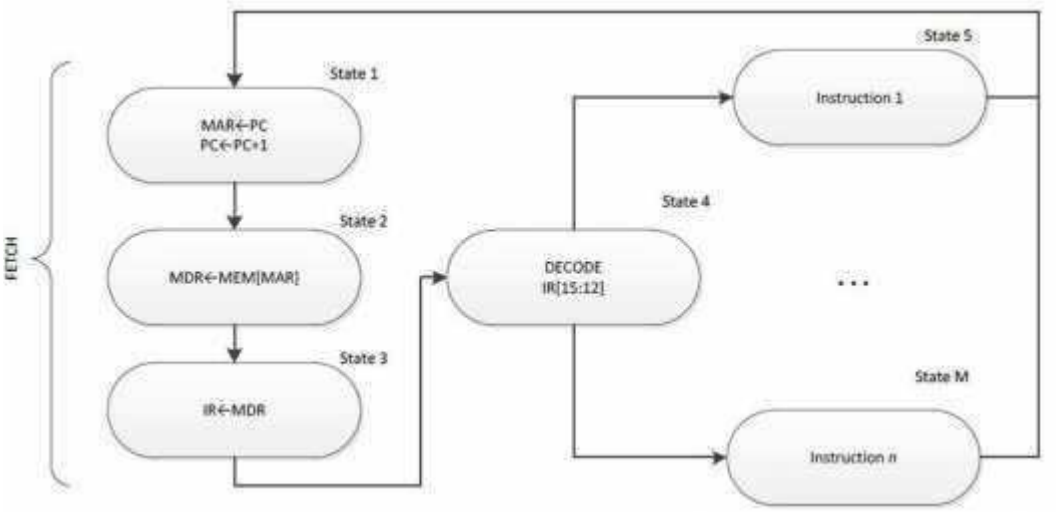
1. **FETCH (instruction) -** This phase obtains the next instruction from memory and stores it in the IR. The address of the next instruction to be executed is stored in the PC register. Proceeds in the following manner

MAR C PC (memory address register is loaded with the content of PC). PC C PC + 1
(value stored in the PC is incremented by one)
MDR C MEM [MAD] (interrogate memory, resulting in the instruction being placed in the MDR).
IR C MDR (load the instruction from MDR to the instruction register).

For now, we will say that each of these steps proceeds in one *machine cycle.* Note that the instruction to be executed is now stored in IR and the address of the *next* instruction to be executed is stored in PC

2. **DECODE -** In this phase the instruction stored in PC is examined in order to decide what portion of the micro architecture needs to be involved in the execution of the instruction. For example, for a 4-bit opcode, this can be implemented as a 4-to-16 decoder. This decoder will examine bits 12-15 stored in the IR and will activate the appropriate circuitry necessary to carry out the instruction

3. **EVALUATE ADDRESS -** This phase Compute the address of the memory location that is needed to process the instruction. Some instructions do not need this phase, e.g., instructions that work directly with the registers and do not require any operands to be loaded or stored form memory.



4. **FETCH OPERANDS -** In this phase, the source operands needed to carry out the instruction are obtained from memory. For some instructions, this phase equals to loading values form the register file. For others, this phase involves loading operands from memory

5. **EXECUTE -** In this phase instruction is carried out. Some instructions may not require this phase, e.g., data movement instructions for which all the work is actually done in the FETCH OPERANDS phase 6. **STORE RESULTS -** In this phase the result is written to its designated destination.

After the 6 phases of the instruction cycle are done, the control unit begins the next instruction cycle, starting with the new FETCH (instruction) phase. Since the PC was previous incremented by one, it contains the pointer to the next instruction to be fetched and executed

An instruction cycle basic involves three sub cycles.

1. **Fetch**
2. **Decode**
3. **Execute**

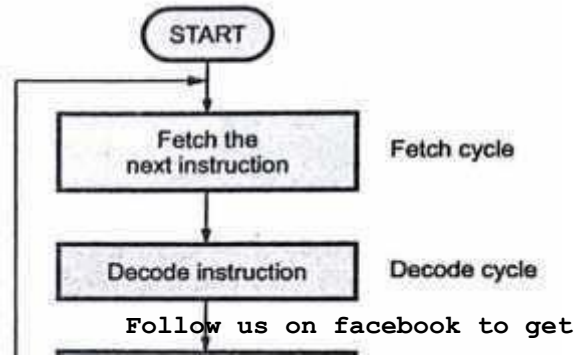


Figure 4. shows the basic instruction cycle.

The fetch phase reads the next instruction from memory into the CPU. The decode phase interprets the opcode by decoding it. The execute phase performs the indicated operation.

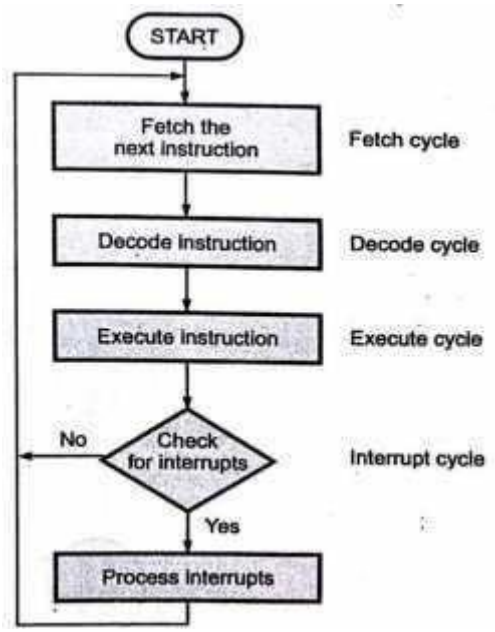


Figure 5. Basic instruction cycle with interrupt cycle

Actually, CPU checks for valid interrupt request after each instruction cycle. If any valid interrupt request is present, CPU saves the current process state, and services the interrupt. Servicing the interrupt means executing interrupt service routine. After completing it, CPU starts the new Instruction cycle from where it has been interrupted. Fig. 5 shows this instruction cycle with interrupt cycle.

The indirect cycle: If the operands on Which the instruction works are present Within the CPU-registers, a memory access is not required. But if the execution of an instruction involves one or more operands in memory, each requires a memory access. If indirect addressing is used then additional memory accesses are required. For fetching the indirect addresses, one or more instruction sub cycles are required. After fetching the instruction, it is decoded and if any indirect addressing is involved, the required operands are fetched using indirect addressing. Also, after performing the operation on the operands according to the opcode, a similar process may be needed to store the result in memory. Following execution, interrupt processing may be required before fetching the next instruction. The same process can be viewed as shown in Fig 6.

Fetch cycle - Initially, the program counter PC is loaded with the address of the first instruction in the program. To provide decoded timing signal To, the Sequence Counter (SC) is cleared to 0. After

each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T0, T1, T2 and so on.

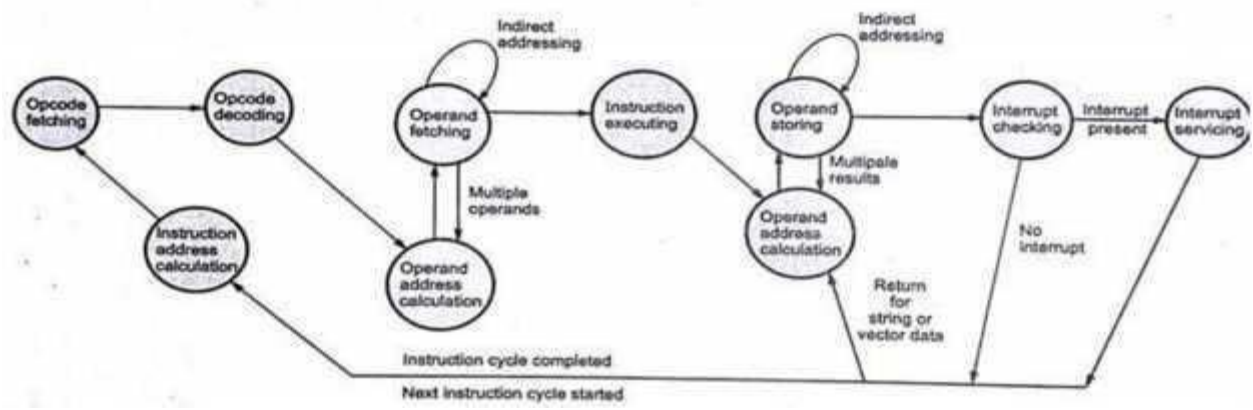


Figure 6. Instruction cycle state diagram

Register transfer statements for fetch cycle are:

T0 : AR PC
T1: IR M[AR], PC PC+1

At To, the address from PC is transferred from PC to AR. The instruction read from the memory is then placed in the Instruction Register (IR) during T1. At the same time, PC is incremented by one.

The Fig. 6 shows the implementation of the first two register transfer statements in the common bus system.

In T0: 1. The contents of PC are placed onto the common bus by enabling its EN input by setting S2 S1 S0 =010
2. The contents of the common bus are transferred to AR by enabling its LD input.

In T1: 1. Read input of the memory is enabled by setting S2 S1 S0 = 1 1 1. This places the contents of memory onto the bus.
2. The contents of common bus are transferred to IR by enabling its LD input.
3. PC is incremented by enabling the INR input of PC.

Decode cycle - At T2, decoding s instruction is done. Register transfer statement for decode cycle is:

T2 : D0……D7 Decode IR (12 - 14), AR IR (0 - 11), I IR (15)

In T2 : 1. Bits 12 - 14 from IR are decoded using 3 : 8 decoder.
2. Bits 0-11 from IR are loaded into the AR.
3. Bit 15 of IR is loaded into the addressing mode (I).

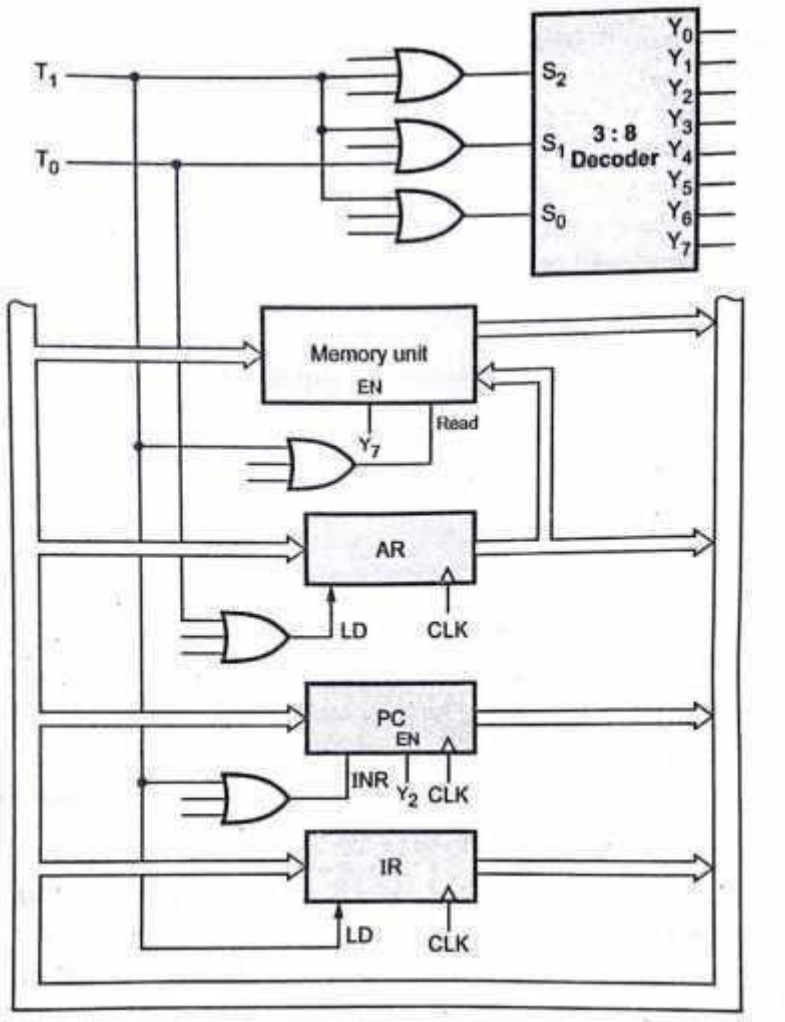


Figure 7. implementation of register transfer instruction for fetch cycle

Determination of type of Instruction

The Fig. 8 shows how the control circuitry determines the type of instruction after the decoding. As shown in the Fig. 8, if decoder output D7 = 0, it is memory reference instruction; otherwise, it is as register reference or I/O instruction. According to D7 and I bits, different instructions are executed listed in Table 1.

Table 1

D_7	I	Instruction executed
0	0	Memory reference instruction with a direct address
0	1	Memory reference instruction with an indirect address
1	0	Register reference instruction
1	1	I/O instruction

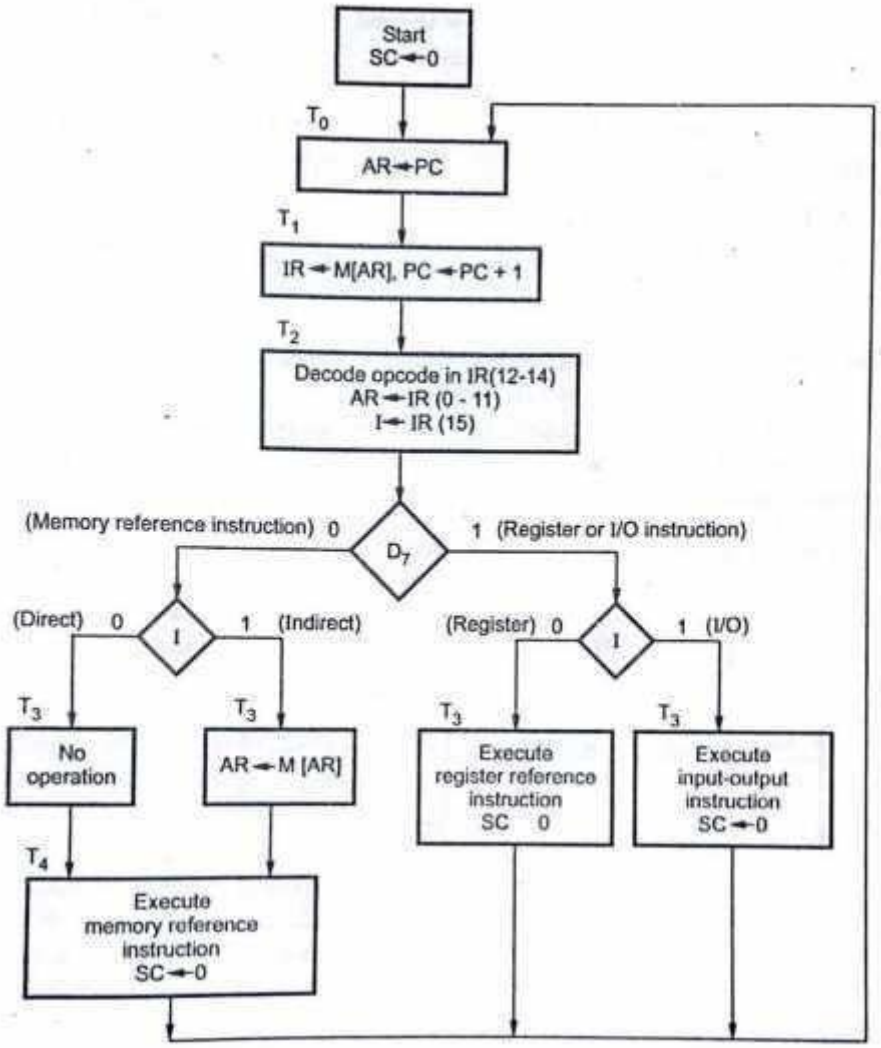


Figure 8. Flowchart for Instruction cycle

HARDWIRED CONTROL UNIT

The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and micro-operations for the accumulator. There are **two major types** of control organization: **Hardwired Control** and **Micro-programmed Control**. In the hardwired organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits. It has the **advantage** that it can be optimized to produce a **fast mode of operation**. In the micro-programmed organization, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of micro-operations. A hardwired control, as the name implies, re- quires changes in the wiring among the various components if the design has to be modified or changed. In the micro-programmed control, any required changes or modifications can be done by updating the micro-program in control memory. A hardwired control for the basic computer is presented in this section.

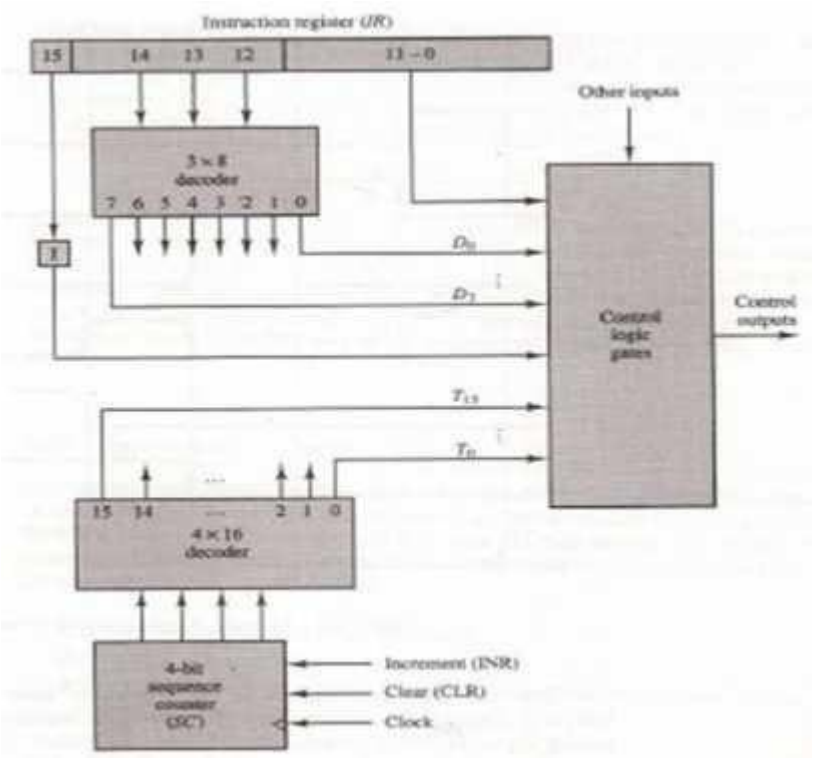


Figure 9. Hardwired Control Unit of Basic Computer

In the hardwired control, the **control units use fixed logic circuit** to interpret instructions and generate control signals from them. The fixed logic circuits use contents of the control step counter, contents of the instruction register, contents of the condition code flag and the external input signals such as MFC and interrupt requests to generate control Signals.

The block diagram of the control unit is shown in Figure 9. It consists of two decoders, a sequence counter, and a number of control logic gates. An instruction read from memory is placed in the instruction register **(IR).** The position of this register in the common bus system is indicated in Fig. The instruction register is shown again in above Fig. where it is divided into three parts: the 1 bit, the operation code, and bits **0** through **11**. The operation code in bits 12 through 14 are decoded with a 3 x 8 decoder. The eight outputs of the decoder are designated by the symbols **D0** through **D7**. The subscripted decimal number is equivalent to the binary value of the corresponding operation code. Bit 15 of the instruction is transferred to a flip-flop designated by the symbol **1**. Bits **0** through 11 are applied to the control logic gates. The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals **To** through **T15**. The internal logic of the control gates can be derived when we consider the design of the computer in detail. The sequence counter SC can be incremented or cleared synchronously. Most of the time, the counter is incremented to provide the sequence of timing signals out of the 4 x 16 decoder. Once in a while, the counter is cleared to 0, causing the next active timing signal to be **To**.

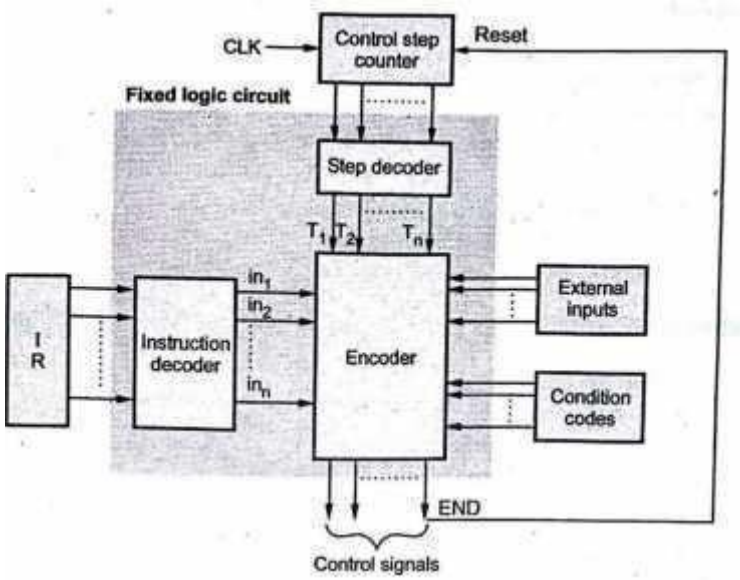


Figure 10. Detail block diagram for hardwired control unit

Figure 10 shows the typical hardwired, control unit. Here, the fixed logic circuit block includes combinational circuit (decoder and encoder) that generates the required control outputs, depending on the state of all its inputs. By separating the decoding and encoding functions, we have shown more detail block diagram for hardwired control unit as shown in the Fig. 10. The instruction decoder decodes the instruction loaded in the IR. If IR is an 8-bit register then instruction decoder generates 2^8 i.e.

256 lines; one for each. According to code in file IR, only one line amongst output lines of decoder goes high i.e. set to l and all other lines are set to 0. The Step decoder provides a separate signal line for each step, or time slot, in a control sequence. The encoder 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.

Advantages of Hardwired Control Unit

1. Hardwired control unit is fast because control signals are generated by combinational circuits.
2. The delay in generation of control signals depends upon the number of gates.
3. It has greater chip area efficiency since its uses less area on-chip.

Disadvantages of Hardwired Control Unit

1. More the control signals required by CPU; more complex will be the design of control unit.
2. Modifications in control signal are very difficult. That means it requires rearranging of wires in the hardware circuit:
3. It is difficult to correct mistake in original design or adding new feature in existing design of control unit.

Design Methods of Hardwired Control Unit - There are four simplified and systematic methods for the design 0f hardwired controllers.

1. **State-table Method**: It is the standard algorithmic approach to sequential circuit design.
2. **Delay-element Method**: It is a heuristic method based on the use of clocked delay elements for control signal timing.
3. **Sequence-counter Method**: It uses counters for timing purposes.

4. **PLA Method**: It uses programmable logic array.

CONTROL MEMORY

Every instruction in a processor is implemented by a sequence of one or more, sets of, concurrent micro-operations. Each micro-operation is associated with a specific set of control lines which, when activated, causes that micro-operation to take place. In the hardwired control, the control unit uses fixed logic circuits to interpret instructions and generate control signals from them. Micro-programming is an elegant and systematic method for controlling the .micro—operation sequences. Since the number of instructions and control lines is often in the hundreds, the complexity of hardwired control unit is very high Thus, it is costly and difficult to design. Furthermore, the hardwired control unit is relatively inflexible because it is difficult to change the design, if one wishes to correct design error or modify the instruction set.

An advance development known as **Dynamic Micro-programming** permits a micro- program to be loaded initially from an auxiliary memory such as a magnetic disk. Control units that use dynamic micro-programming use a writable control memory. This type of memory can be used for writing (to change the micro-program) but is used mostly for reading. A memory that is part of a control unit is called a **Control Memory.**

Micro-programming is a method of control unit design in which the control signal selection and sequencing information is stored in a ROM or RAM **called a control memory CM**. The control signals to be activated at any time are specified by a microinstruction, which is fetched from CM in much similar way an instruction is fetched from main memory. Each micro-instruction also explicitly or implicitly Specifies, the next microinstruction to be used, thereby providing the necessary information for sequencing. A sequence of one or more micro-operations designed to control specific operation, such as addition, multiplication is **called a micro program.** The micro-programs for all instructions are stored in the control memory.

A **control variable is a binary digit or bit (0 or 1)** controls the function that specifies a micro-operation. When it is binary 1 state, the corresponding micro- operation is executed while in the opposite binary state, the state of the registers in the system remains unchanged. In a bus-organized system, the control signals that specify the Moro-operations are groups of bits that select the paths in multiplexers, decoders and ALUs. The address where the microinstructions are stored in control memory is generated by micro-program sequencer/micro-program controller in the **micro-programmed control unit.** Thus, the control unit initiates a series of sequential steps of micro-operations. As per the operation, certain micro-operations are to be initiated, while others remain idle at any given time. Grouping the control variables at any given time form a ‘string of 1's and 0's, **called a control word**. The control words are stored in the control memory to perform various operations on the components of the system. The control unit whose binary control variables are stored in memory **is called a micro-programmed control unit**. Each word in control memory contains within it a microinstruction. Each microinstruction specifies one or more micro-operations for the system. A sequence of microinstructions constitutes a micro-program. If the operations that are to be perforated by a control unit are fixed, a Read-Only Memory (ROM) can be used as a control memory.

MICRO-PROGRAMMED CONTROL UNIT

A computer that uses a micro-programmed control unit usually has two separate memories - **a main memory and a control memory**. The **main memory** is available to the user for storing their programs. The contents of main memory may change when the data are manipulated and every

time the program is changed. The user's program in main memory consists of machine instructions and data, whereas, the **control memory** holds a fixed micro-program that cannot be altered by the occasional user. The micro-program consists of micro-instructions that specify various internal control signals for execution of register micro-operations. Each machine instruction initiates a series of microinstructions in control memory. These microinstructions generate the micro-operations to fetch the instruction from main memory; to evaluate the effective address, to execute the operation specified by the instruction, and to return control to the fetch phase in order to repeat the cycle for the next instruction.

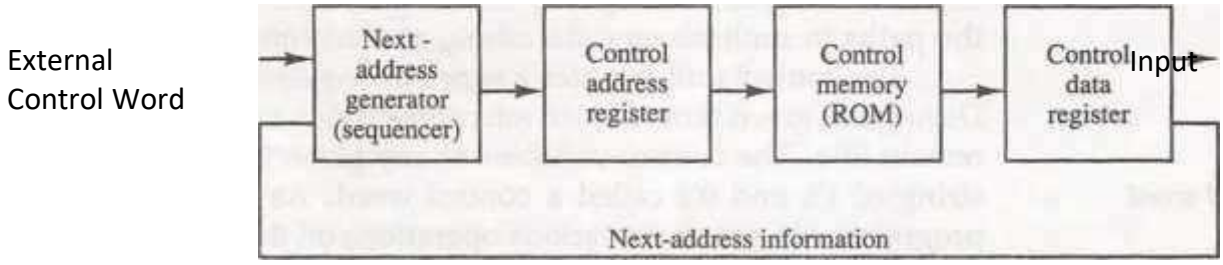


Figure 11. Micro-programmed Control Organization

The general configuration of a micro-programmed control unit is demonstrated in the block diagram above. The **control memory** is assumed to be a ROM, within which all control information is permanently stored. The **control memory address register** specifies the address of the microinstruction, and the **control data register** holds the microinstruction read from memory. The microinstruction contains a control word that specifies one or more micro-operations for the data processor. Once these operations are executed, the control must determine the next address. The location of the next microinstruction may be the following place:

1. One next in sequence, or
2. It may be located somewhere else in the control memory
3. The next address may also be a function of external input conditions.

For this reason it is necessary to use some bits of the present microinstruction to control the generation of the address of the next microinstruction. While the micro- operations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction. Thus a microinstruction contains bits for initiating micro- operations in the data processor part and bits that determine the address sequence for the control memory.

The next address generator is sometimes called a **Micro-program *Sequencer,*** as it determines the address sequence that is read from control memory. The address of the next microinstruction can be specified in several ways, depending on the sequencer inputs. Typical functions of a micro-program sequencer are incrementing the control address register by one, loading into the control address register an address from control memory, transferring an external address, or loading an initial address to start the control operations. The control data register holds the present microinstruction while the next address is computed and read from memory. The data register is sometimes called a ***Pipeline Register.*** It allows the execution of the micro-operations specified by the control word simultaneously with the generation of the next microinstruction. This configuration requires a two-phase clock, with one clock applied to the address register and the other to the data register.

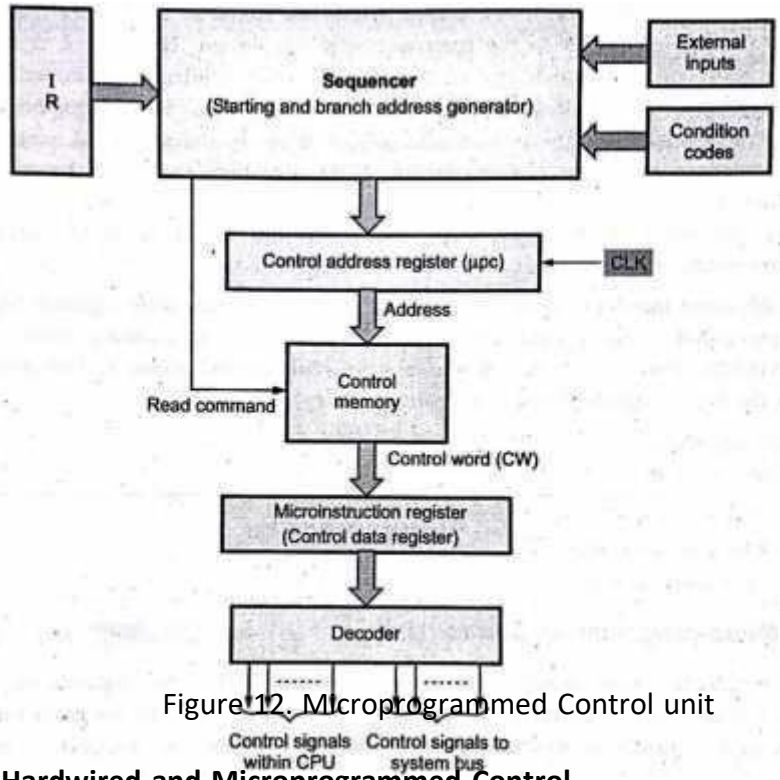


Figure 12. Microprogrammed Control unit

Comparison between Hardwired and Microprogrammed Control

Attribute	Hardwired control	Microprogrammed control
Speed	Fast	Slow
Control functions	Implemented in hardware	Implemented in software
Flexibility	Not flexible, to accommodate new system specifications or new instructions.	More flexible, to accommodate new system specification or new instructions redesign is required.
Ability to handle large/complex instruction sets	Somewhat difficult	Easier
Ability to support operating systems and diagnostic features	Very difficult (unless anticipated during design)	Easy
Design process	Somewhat complicated	Orderly and systematic
Applications	Mostly RISC microprocessors	Mainframes, some microprocessors
Instruction set size	Usually under 100 instructions	Usually over 100 instructions
ROM size	-	2 K to 10 K by 20-400 bit microinstructions
Chip area efficiency	Uses least area	Uses more area

Advantages of microprogrammed control

1. It simplifies the design of control unit, Thus it is both, cheaper and less error prone to implement.
2. Control functions are implemented in software rather than hardware.
3. The design process is orderly and systematic.
4. More flexible, can be changed to accommodate new system specifications or to correct the design errors quickly and cheaply.
5. Complex function Such as floating point arithmetic can be realized efficiently

Disadvantages of micro programmed control

1. A micro programmed control unit is somewhat slower than the hardwired control unit, because time is required to access the microinstructions from CM.
2. The flexibility is achieved at some extra hardware cost due to the control memory and its access circuitry.

Besides these disadvantages, the microprogramming is the dominant technique for implementing control units. However, the most computers based on the Reduced Instruction Set Computer (RISC) architecture concept use hardwired control.

ADDRESS SEQUENCEING

A simple way to structure microinstructions is to assign one bit position to each control signal required in the CPU. However, this scheme has one serious drawback assigning individual bits to each control signal results in long microinstructions, because the number of required signals is usually large. Moreover, only a few bits are used in any given instruction. The solution of this problem is to group the control signals. Grouping technique is used to reduce the number of bits in the microinstruction.

Microinstructions are stored in control memory in groups, with each group specifying a ***Routine.*** Each computer instruction has its own micro-program routine. It is stored in control memory to generate the micro-operations that execute the instruction. The hardware that controls the address sequencing of the control memory must be capable of sequencing the microinstructions within a routine and be able to branch from one routine to another. An initial address is loaded into the **control address register** when power is turned on in the computer. This address is usually the address of the first microinstruction/ that activates the instruction fetch routine. The fetch routine may be sequenced by incrementing the control address register through the rest of its microinstructions. At the end of the fetch routine, the instruction is in the instruction register of the computer.

The control memory next must go through the routine that determines the effective address of the operand. A machine instruction may have bits that specify various addressing modes, such as indirect address and index registers. The effective address computation routine in control memory can be reached through a branch microinstruction. When the effective address computation routine is completed, the address of the operand is available in the memory address register. The next step is to generate the micro-operations that execute the instruction fetched from memory. The microoperation steps to be generated in processor registers depend on the operation code part of the instruction. Each instruction has its own micro-program routine stored in a given location of control memory. The transformation from the instruction code bits to an address in control memory where the routine is located is referred to as a ***mapping*** **process**.

A mapping procedure is a rule that transforms the instruction code into a control memory address. Once the required routine is reached, the microinstructions that execute the instruction may be sequenced by incrementing the control address register, but sometimes the sequence of micro-operations will depend on values of certain status bits in processor registers. After completion of instruction execution, control must return to the fetch routine. This is done by executing an unconditional branch microinstruction to the first address of the fetch routine. In summary, the address sequencing capabilities required in a control memory are: 1. Incrementing of the control addresses register.

2. Unconditional branch or conditional branch, depending on status bit conditions.
3. A mapping process from the bits of the instruction to an address for control memory.
4. A facility for subroutine call and return.

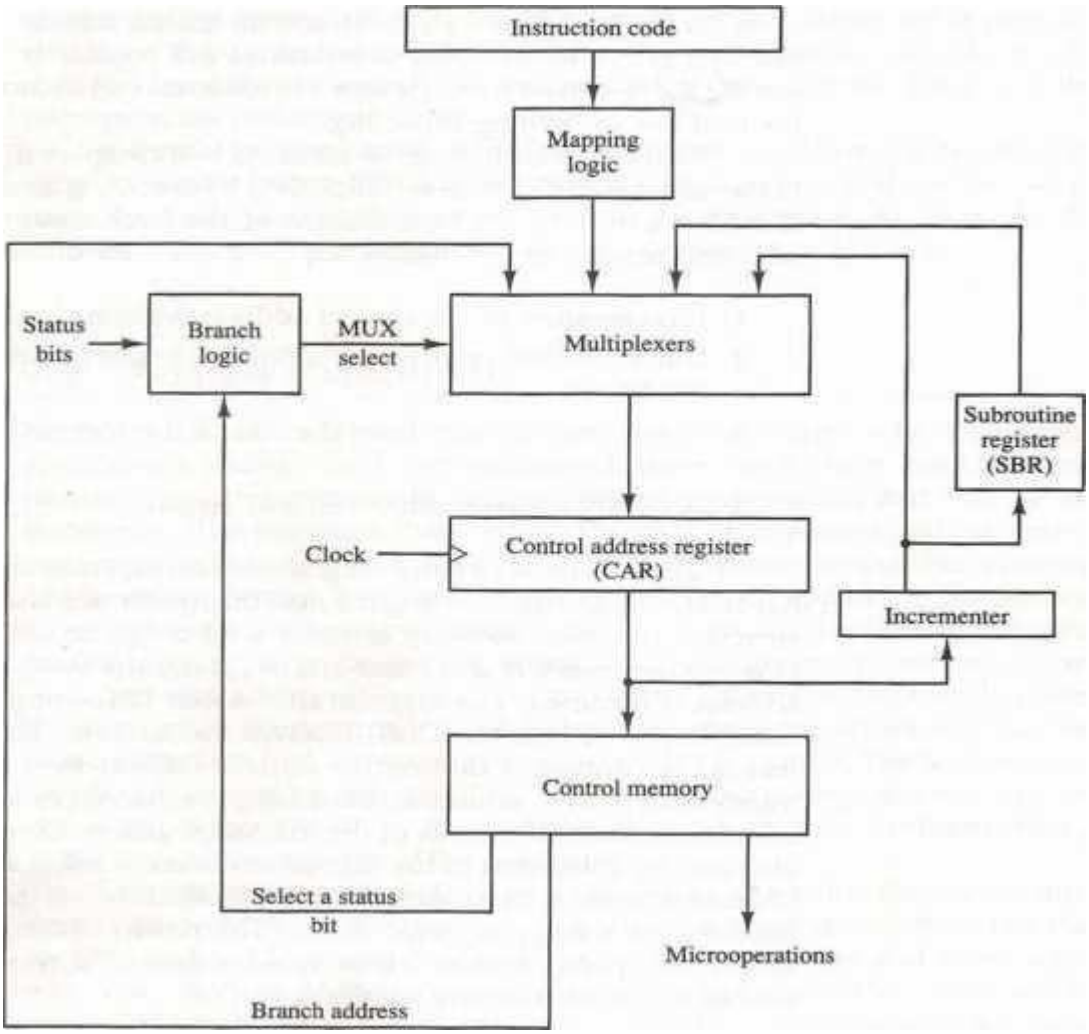


Figure 13. Selection of address for control memory

Figure above shows a block diagram of a control memory and the associated hardware needed for selecting the next microinstruction address. The diagram shows four different paths from which the control address register (CAR) receives the address. The incrementer increments the content of the control address register by one, to select the next microinstruction in sequence. Branching is achieved by specifying the branch address in one of the fields of the microinstruction. Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition. An external address is transferred into control memory via a mapping logic

circuit. The return address for a subroutine is stored in a special register whose value is then used when the micro-program wishes to return from the subroutine. **Conditional Branching**

The branch logic of above Figure provides **decision-making capabilities** in the control unit.

The **status conditions are special bits** in the system that provide parameter information such as the carry-out of an adder, the sign bit of a number, the mode bits of an instruction, and input or output status conditions. Information in these bits can be tested and actions initiated based on their condition: whether their value is 1 or 0. The status bits, together with the field in the microinstruction that specifies a branch address, control the conditional branch decisions generated in the branch logic.

The branch logic hardware may be implemented in a variety of ways. The simplest way is to test the specified condition and branch to the indicated address if the condition is met; otherwise, the address register is incremented. This can be implemented with a multiplexer. An unconditional branch microinstruction can be implemented by loading the branch address from control memory into the control address register. This can be accomplished by fixing the value of one status bit at the input of the multiplexer, so it is always equal to 1. A reference to this bit by the status bit select lines from control memory causes the branch address to be loaded into the control address register unconditionally.

Special bits - Conditional branching is achieved by using part of the microinstruction to select a specific status bit in order to determine its condition. **Special bits are used to check Conditions** such as the sign bit of a number, carry- out of an adder; the mode bits in an instruction and input or output status conditions The branch logic checks the status of these bits **(1 or 0)** together with the field in the microinstruction that specifies a branch address and control the conditional branch decisions.

Branch logic - The branch logic hardware checks the status of bits reserved in the microinstruction to take branching decision on the occurrences of specified conditions. One way to implement branch logic hardware is given below. Suppose 8 different parameters are to be checked in a system; It requires 8 status bits. A multiplexer can be used to implement branch logic hardware. In this case, three bits in the microinstruction are used to specify any one of eight status bit conditions. These three bits are used as selection input variables for the multiplexer. If the selected status bit is in the 1 state, the output of the multiplexer goes 1; otherwise it is 0. The 1 output of the multiplexer generates a control signal that transfers the branch address from the microinstruction into the control address register. A 0 output of the multiplexer increments the address register.

When conditional branch microinstruction is executed, the microprogram follows one of two possible paths. The value of status bit decides the selection of path. When an unconditional branch microinstruction is executed, the branch address is loaded from control memory into the control address register. This can be implemented by fixing the value of one status bit at the input of the multiplexer, so it. is always equal to 1.

Mapping of Instruction

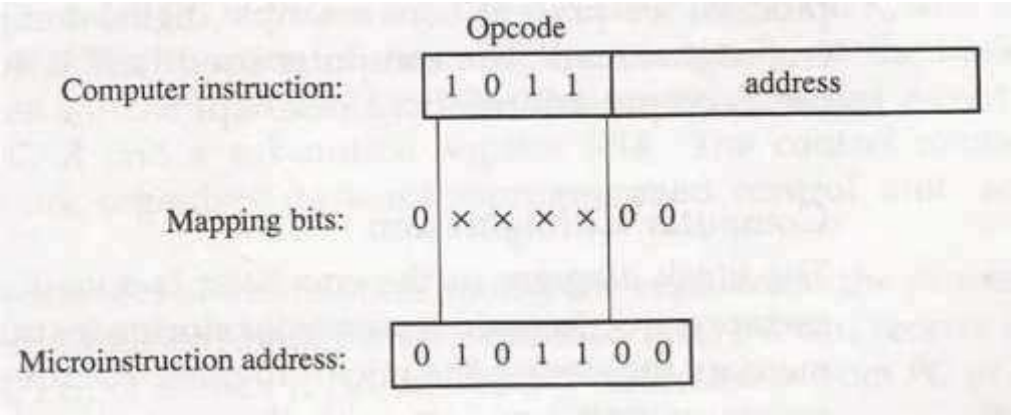


Figure 14. Mapping from instruction code to microinstruction address.

We have seen that the transformation from the instruction code bits to an address in control memory where the microprogram routine is located is **referred to as a mapping process**. A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a micro-program routine for an instruction is located. The status bits for this type of branch are the bits in the operation code part of the instruction.

For example, a computer with a simple instruction format as shown in above has an operation code of four bits which can specify up to 16 distinct instructions. Assume further that the control memory has 128 words, requiring an address of seven bits. For each operation code there exists a micro-program routine in control memory that executes the instruction. One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory is shown in Fig. This mapping consists of

1. Placing a 0 in the most significant bit of the address.
2. Transferring the four operation code bits.
3. Clearing the two least significant bits of the control address register.

If the routine needs more than four microinstructions, it can use **addresses 1000000** through **1111111**. If it uses fewer than four microinstructions, the unused memory locations would be available for other routines.

One can extend this concept to a more general mapping rule by using a ROM to specify the mapping function. In this configuration, the bits of the instruction specify the address of a mapping ROM. The contents of the mapping ROM give the bits for the control address register. In this way the micro-program routine that executes the instruction can be placed in any desired location in control memory. The mapping concept provides flexibility for adding instructions for control memory as the need arises.

Subroutines

When a certain group of instructions is repeatedly required in a program, then instead of writing it repeatedly, it can be stored separately and can be called in a main program whenever required. This group of instructions is called subroutine. This is the way to use memory efficiently.

In microprogrammed control unit, many microprograms contain identical sections of code. In this case, subroutines can be used for common sections of microcode. For example, generation of the effective address of the operand for an instruction is the task commonly required for the execution of memory reference instructions. The subroutine can be written to perform this task and can be called within many routines to execute the effective address computation.

Subroutine Register

Subroutines can be used in microprograms to use control memory efficiently. When the microprogram (main routine) needs subroutine, the address of subroutine is to be loaded into the control address register. This transfers program control from main routine to the subroutine. However, after execution of subroutine, the control should be transferred back to the main routine. So, before transferring control from main routine to the subroutine, it is necessary to store return address. This may be accomplished by storing the incremented output from the control address register (return address) into a subroutine register and then branching to the beginning of the subroutine. Upon completion of subroutine execution, the return address is restored into the control address register from subroutine register. Thus, the subroutine register stores the return address during a subroutine call and restores it during a subroutine return. The stack (registers organized in LIFO fashion) can be used for the execution of subroutines.

SEQUENCING AND EXECUTION OF MICRO INSTRUCTION.

Microinstruction Format

Figure 15 shows the microinstruction format for the control memory. As shown in Fig 15 the microinstruction includes four fields.

1. F1, F2, F3: These are micro-operation fields. Each field is of three bits. They specify micro-operations for the Computer.
2. CD: This two-bit field selects status bit conditions for branching operation. The condition includes zero value in AC, sign bit of AC equal to 1 or 0, etc.
3. BR: This 2-bit field specifies the type of branch to be used. Branch types include unconditional branch, branch if zero, and branch if negative and so on.
4. AD: This is an address field which contains a branch address. This field is of seven bits since control memory has 128 words. (128 = 27).

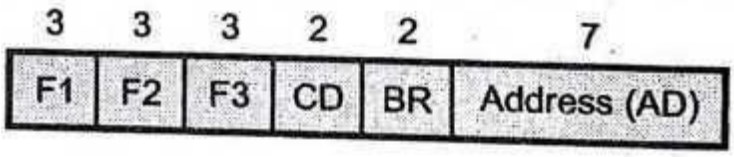


Figure 15 Microinstruction format

The micro-operations are subdivided into three fields as F1, F2, F3, each is of 3-bits. The three bits in each field are encoded to specify seven distinct micro-operations.

Thus there are 21 micro-operations. A microinstruction includes three micro- operations, one from each field. So, no more than three micro-operations can be selected for a microinstruction. If the microinstruction needs micro-operations less than three, one or more of the micro—operation fields will be filled by a binary code
000 for no operation.

F1 3-bit	Associated micro-operation	Symbol	Description
000	—	NOP	No operation
001	AC ← AC+ DR	ADD	Add DR and AC and store result in AC
010	AC ← 0	CLRAC	Clear AC
011	AC ← AC+ 1	INCAC	Increment AC
100	AC← DR	DRTAC	Copy contents of AC in DR
101	AR ← DR(0 – 10)	DRTAR	Copy the contents of DR in AR
110	AR ← PC	PCTAR	Copy the contents of DC in AR
111	M[AR] ← DR	WRITE	Copy the contents o fDR into memory location addressed by AR

F2 3-bit	Associated micro-operation	Symbol	Description
000	—	NOP	No operation.
001	AC ← AC – DR	SUB	Subtract the contents of DR from the contents of AC and store the result in AC.
010	AC ← AC ∨ DR	OR	Logically OR the contents of DR with the contents of AC and store the result in AC.
011	AC ← AC ∧ DR	AND	Logically AND the contents of DR with the contents of AC and store the result in AC.
100	DR ← M[AR]	READ	Read the contents of memory location addressed by AR in DR
101	DR ← AC	ACTDR	Copy the contents of AC in DR
110	DR ← DR + 1	INCDR	Increment the contents of DR
111	DR(0 – 10) ← PC	PCTDR	Copy the contents of PC in DR.

F3 3-bits	Associated micro-operation	Symbol	Description
000	—	NOP	No operation
001	AC ← AC ⊕ DR	XOR	Logically XOR the contents of DR and AC and store the result in AC.
010	AC ← $\overline{AC}$	COM	Complement the contents of AC.
011	AC ← shl AC	SHL	Left shift the contents of AC by 1-bit.
100	AC ← shr AC	SHR	Right shift the contents of AC by 1-bit.
101	PC ← PC + 1	INCPC	Increment contents of PC.
110	PC ← AC	ARTPC	Copy the contents of AC into PC.
111	—	—	—

Table 1 Micro-instructions with their binary code, micro-operation and symbol

From above table, we can observe

1. Each micro-operation is defined with a register transfer statement.
2. A symbol is assigned to each macro-operation. This is useful in writing symbolic program.
3. All transfer-type micro-operations symbols use five letters. The first two letters indicate the source register, the third letter is always T and the last two letters indicate the destination register.

Condition Field

The two-bits in the condition (CD) field are encoded to specify four status bit conditions as listed m Table 2.

CD 2-bit	Symbol	Status bit condition	Description
00	U	(1) always	Represents unconditional branch.
01	I	15^{th} bit of DR : DR(15)	Indirect address bit.
10	S	15^{th} bit of AC : AC(15)	Represents sign bit of AC.
11	Z	AC = 0	Indicates zero value in AC.

Table 2 Condition field with binary code, condition and symbol

From Table 2. we can observe,

1. The first condition is always a 1, so that a reference to 'CD = 00' (symbol U) will always find the condition to be true. When this condition is used with the branch field, it provides an unconditional branch operation.
2. The indirect bit I is available from bit 15 of DR after reading an instruction from memor
   y.
3. The bit 15 (sign-bit) of an accumulator provides the next status bit.
4. When the content of an accumulator becomes zero, the binary variable 2 becomes, equal to 1. The symbols U, I, S and Z are used for the four status bits while writing microprograms in symbolic form.

Branch Field and Address Field

The two bit branch field is used with 7- bit address field to obtain the address of the next microinstruction. The two-bits in the branch .field are encoded to specify four branches as shown in Table 3.

BR 2-bits	Symbol	Micro-operation
00	JMP	Condition = 1 CAR ← AD Condition = 0 CAR ← CAR + 1
01	CALL	Condition = 1 CAR ← AD, SBR ← CAR + 1 Condition = 0 CAR ← CAR + 1
10	RET	CAR ← SBR
11	MAP	CAR (2-5) ← DR (11 - 14), CAR (0, 1, 6) ← 0

Table 3 Branch field with binary code. symbol and function

From Table 3, we can observe,

1. When BR = 00, the control performs a jump operation.
2. When BR = 01, the control performs a call to subroutine operation.

MICRO-PROGRAM SEQUENCER

The subunit of the microprogrammed control unit which presents an address to the control memory is **called microprogram sequencer**. The next-address logic of the sequencer determines the specific address source to be loaded into the control address register.

The Fig. 16 shows the block diagram of commercial microprogram sequencer. It consists of a multiplexer that selects an address from four sources and routes it into a control address register. The output from CAR provides the address for the control memory. The contents of CAR are incremented and applied to the multiplexer and to the stack register file. The register selected in the stack is determined by the stack pointer. Inputs 12, 11, I0 and T derived from the CD and BR fields of microinstruction specify the operation for the sequencer. They s p e c i f y the input source to the multiplexer also generate push and pop signals required for stack operation.

The stack pointer is a , three-bit register and it gives the address of stack register file consists of (2^3 = 8) eight registers Initially, the stack pointer is cleared and is said to point at address 0 in the stack. Using push operation it is possible to write data into the stack at the address specified by the stack pointer. After data is written, stack pointer is incremented by one to get ready for the next push operations.

In pop operation stack pointer is decremented by one and then the contents of the register specified by the new value' of stack pointer are read. With this mechanism it is possible to implement subroutine calls. During subroutine call the incremented address (the address of the next instruction) is stored in the stack. This address also called return address is transferred back into CAR with a subroutine return operation.

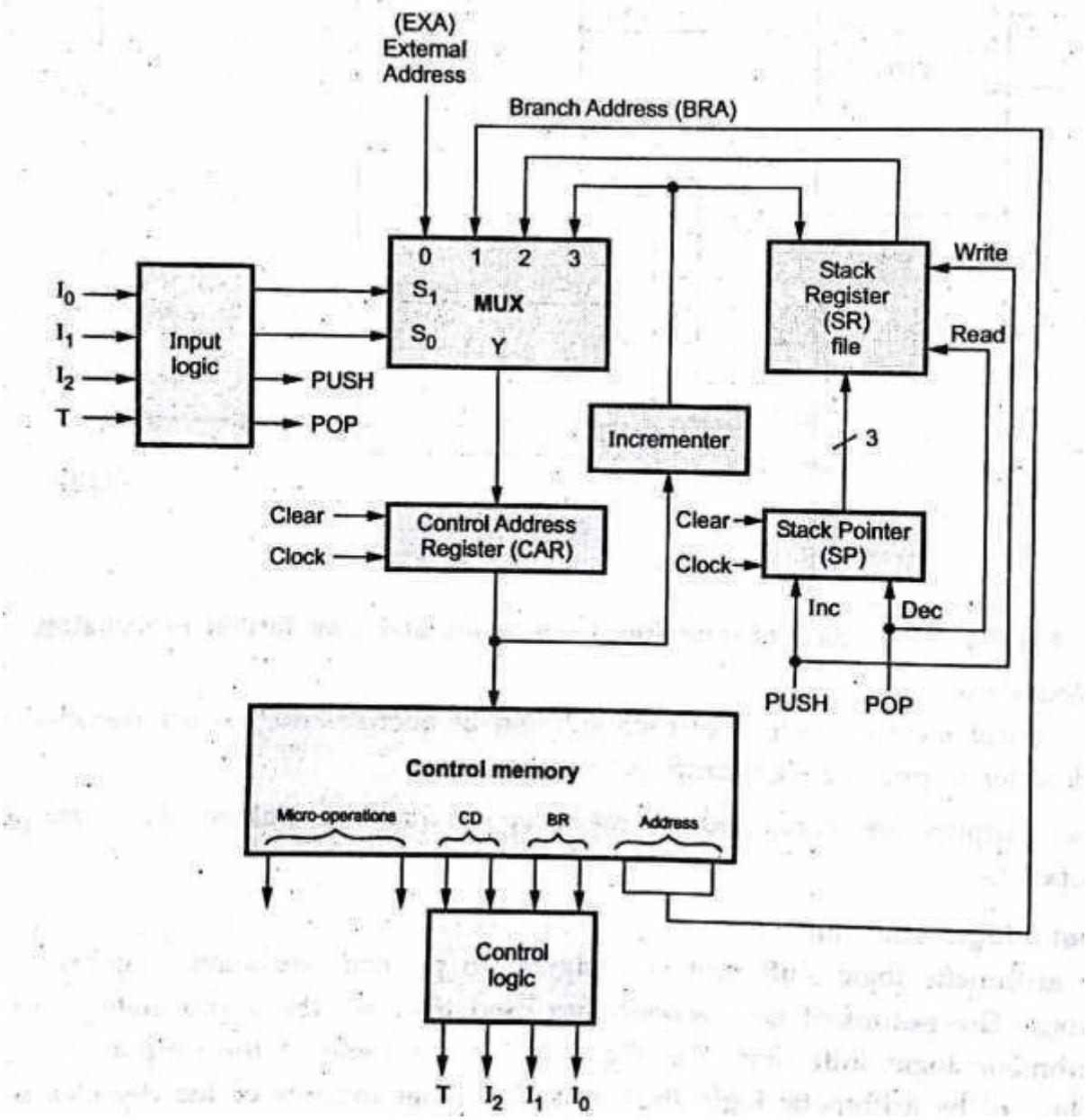


Figure 16 Typical microprogram sequencer organization

The Table 4 gives the function table for microprogram sequencer. When **S1S0 = 00**, an external address is transferred to CAR. The transfer from address field of microinstruction occurs when **S1S0 = 01** and T = 1. When **S1S0 = 10**, stack register contents are transferred to CAR and when **S1S0 = 11**, incremented contents of CAR are transferred to the CAR.

A call to subroutine is executed by activating push signal when **S1S0 = 01**. This causes a push stack operation and a branch to the address specified by microinstruction. The return from subroutine is executed by activating pop signal when **S1S0 = 10.** This cause a pop-stack operation and a branch to the address stored on top of the stack

I_2	I_1	I_0	T	S_1	S_0	Operation	Description
X	0	0	X	0	0	CAR ← EXA	Transfer external address.
1	0	1	1	0	1	CAR ← BRA, SR ← CAR + 1	Branch to subroutine and save the next instruction address in stack (Push operation).
0	0	1	1	0	1	CAR ← BRA	Transfer branch address.
X	1	0	X	1	0	CAR ← SR	Transfer from stack register.
0	1	1	0	1	1	CAR ← CAR + 1	Increment address.

Table 4 Function table of micro program sequencer

UNIT – 3

Syllabus :Computer Arithmetic: Addition and Subtraction, Two's Compliment Representation, Signed Addition and Subtraction, Multiplication and division, Booths Algorithm, Division Operation, Floating Point Arithmetic Operation, Design of Arithmetic unit.

INTRODUCTION

- Based on the number system two basic data types are implemented in the computer system: fixed point numbers and floating point numbers.
- Representing numbers in such data types is commonly known as fixed point representation and floating point representation, respectively.
- In binary number system, a number can be represented as an integer or a fraction.
- Depending on the design, the hardware can interpret number as an integer or fraction.
- The radix point is never explicitly specified It is implicited in the design and the hardware interprets it accordingly.
- In integer number radix point is fixed and assumed to be to the right of the right most digits.
- As radix point is fixed, the number system is referred to as fixed point number system.
- With fixed point number system we can represent positive or negative integer numbers.
- Floating point number system allows the representation of numbers having both integer part and fractional part.

ADDITION AND SUBTRACTION OF SIGNED NUMBERS

We can relate addition and subtraction operations of numbers by the following Relationship:

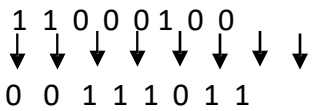
$$(\pm A) - (+B) = (\pm A) + (-B) \text{ and } (\pm A) - (-B) = (\pm A) + (+B)$$

Therefore, we can change subtraction operation to an addition operation by changing the sign of the subtrahend.

1's Complement Representation

The 1's complement of a binary number is the number that results when we change all 1's to zeros and the zeros to ones.
Find 1's complement of $(11000100)_2$.

Solution -



2's Complement Representation

The 2's complement is the binary number that results when we add 1 to the 1's Complement. It is given as 2's complement = **1's complement + 1**
The 2's complement form' is used to represent negative numbers.
Find 2's complement of $(11000100)_2$

Solution – 1		1	0	0	0	1	0	0	
		0	1	1	1	0	1	1	1's compliment
	0	0	1	1	1	0	1	1	1's compliment
+								1	2's compliment
		0	0	1	1	1	1	0	0

Subtraction of Binary Numbers using 2's Complement Method

In a 2's complement subtraction, negative number is represented in the 2's complement form and actual addition is performed to get the desired result. For example, operation A-B is performed using following steps:

1. Take 2's complement of B.
2. Result A + 2's complement of B.
3. If carry is generated then the result is positive and in the true form. In this case, carry is ignored.
4. If carry is not generated then the result is negative and in the 2's complements form.

Addition / Subtraction Logic Unit

Figure 1 shows hardware to implement integer addition and subtraction. It consists of n-bit adder, 2's complement circuit, overflow detector logic circuit and AVF (overflow flag). Number a and number b are the two inputs for n-bit adder. For subtraction, the subtrahend (number from B register) is converted into its 2's complement form by making Add/ Subtract control signal to the logic one. When Add/Subtract control signal is one, all bits of number b are complemented and carry zero (C0) is set to one. Therefore n-bit adder gives result as R = a + b(bar) + 1, where b (bar) +1 represents 2's complement of number b.

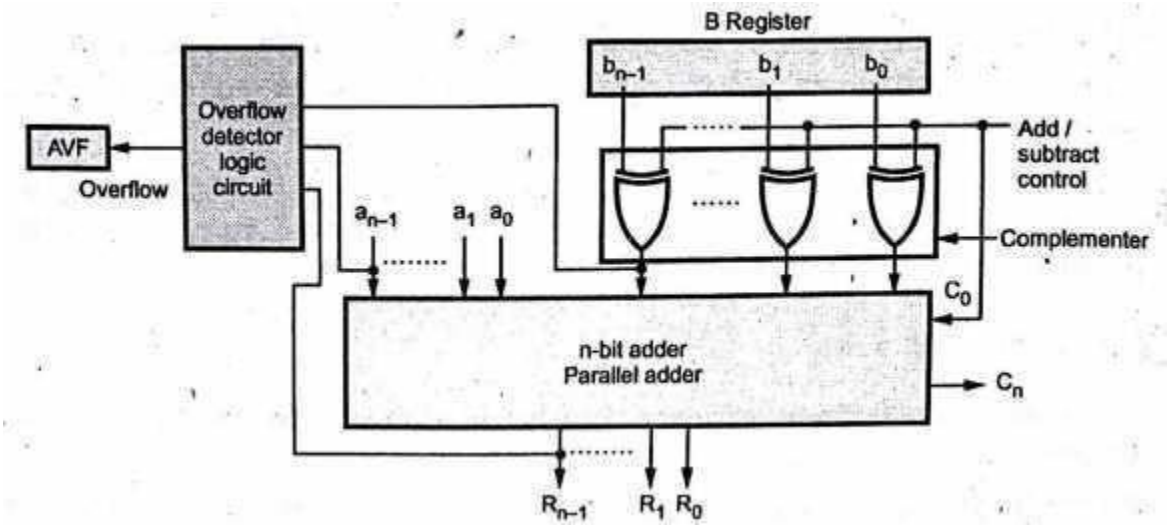


Figure 1: Hardware for Integer addition and subtraction

Overflow In Integer Arithmetic

Assuming numbers are in 4 bit for addition and subtraction.

Case 1: Both numbers positive

1 1 1	carry bit
0 1 1 1	(+ 7)
0 0 1 1	(+ 3)
1 0 1 0	Result: 2's complement of 6

Result is -6 ; it is wrong due to overflow.

Case 2: Both numbers negative

1	Carry
1 0 1 1	2's complement of 5, i.e. (- 5)
1 1 0 0	2's complement of 4, i.e. (- 4)
1 0 1 1 1	(+ 7)

- Result is + 7 ; it is wrong due to overflow,
- When adding signed a, numbers, a carry bit beyond the end of the word does not serve as the overflow indicator.
- If we add then numbers +7 and + 3 in a 4-bit adder, the output is 1010, which is the code of -6, a wrong result.
- In this case, carry bit from the MSB position is 0. Similarly, if we add -5 and -4, we get output = + 7, another error.
- In this case carry bit from the MSB position is 1.
- One thing we can surely say that, the addition of numbers with different signs cannot cause overflow, because the absolute value of the sum is always smaller than the absolute value of one of the two operands.

From above discussion we can conclude following points:

1. Overflow can occur only when adding two numbers that have the same sign.
2. The carry bit from the MSB position is not a sufficient indicator of overflow when adding signed numbers.
3. When both operands a and b have the same sign, an overflow occurs when the sign of result does not agree with the signs of a and b. The logical expression to detect overflow can be given as

Overflow = $a_{n-1}\, b_{n-1}\, R_{n-1} + a_{n-1}\, b_{n-1} R_{n-1}$

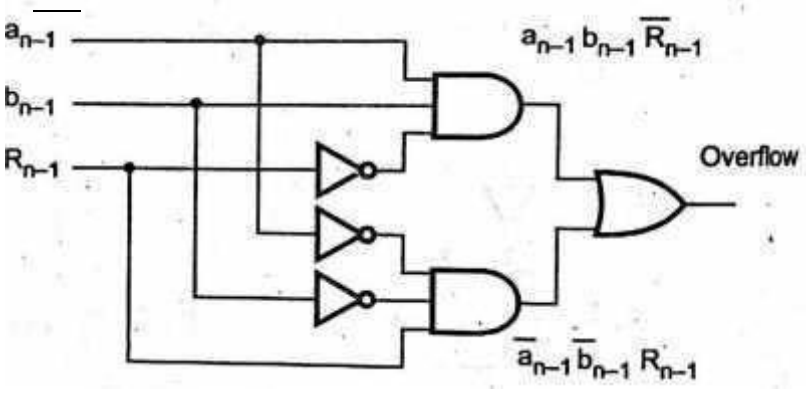


Figure 2: Overflow detector logic circuit

ADDITION AND SUBTRACTION ALGO & FLOWCHART

There are three ways of representing negative fixed-point binary numbers:

1. Signed magnitude
2. Signed- 1's complement
3. Signed-2's complement.

Most computers use the signed-2's complement representation when performing arithmetic operation's with integers. For floating-point operations, most computers use the signed-magnitude representation for the mantissa.

Addition and Subtraction with Signed Magnitude Data

The representation of numbers in signed-magnitude is familiar because it is used in everyday arithmetic calculations. We designate the magnitude of the two numbers by A and B, When the signed numbers are added or subtracted, we find that there are eight different conditions to consider, depending on the sign of the numbers and the operation performed. These conditions are listed in the first column of Table 1. The other columns in the table show the actual operation to be performed with the magnitude of the numbers. The last column is needed to prevent a negative zero. In other words, when two equal numbers are subtracted, the result should be +0 not -0.

Table 1: Addition and Subtraction of Signed Magnitude numbers

Operation	Add Magnitudes	Subtract Magnitudes		
		When A>B	When A<B	When A=B
(+A)+(+B)	+(A+B)			
(+A)+(-B)		+(A-B)	-(B-A)	+(A-B)
(-A)+(+B)		-(A-B)	+(B-A)	+(A-B)
(-A)+(-B)	-(A+B)			
(+A)-(+B)		+(A-B)	-(B-A)	+(A-B)
(+A)-(-B)	+(A+B)			
(-A)-(+B)	-(A+B)			
(-A)-(-B)		-(A-B)	+(B-A)	+(A-B)

Hardware Implementation:

To implement the two arithmetic operations with hardware, it is first necessary that the two numbers be stored in registers. Let A and B be two registers that hold the magnitudes of the numbers, and A_S and B_S be two flip flops that hold the corresponding signs. The result of the operation may be transferred to a third register.

Consider now the hardware implementation of the algorithms above.

- First, a parallel adder is needed to perform the micro operation A + B.
- Second, a comparator circuit is needed to establish if A>B, A=B, or A< B.
- Third, two parallel subtractor circuits are needed to perform the micro operations A-B and B-A.

The sign relationship can be determined from an exclusive - OR gate with A_S and B_S as inputs.

Subtraction can be accomplished by means of complement and add. Second, the result of a comparison can be determined from the end carry after the subtraction. Careful investigation of the alternatives reveals that the use of 2's complement for subtraction and comparison is an efficient procedure that requires only an adder and a complementor.

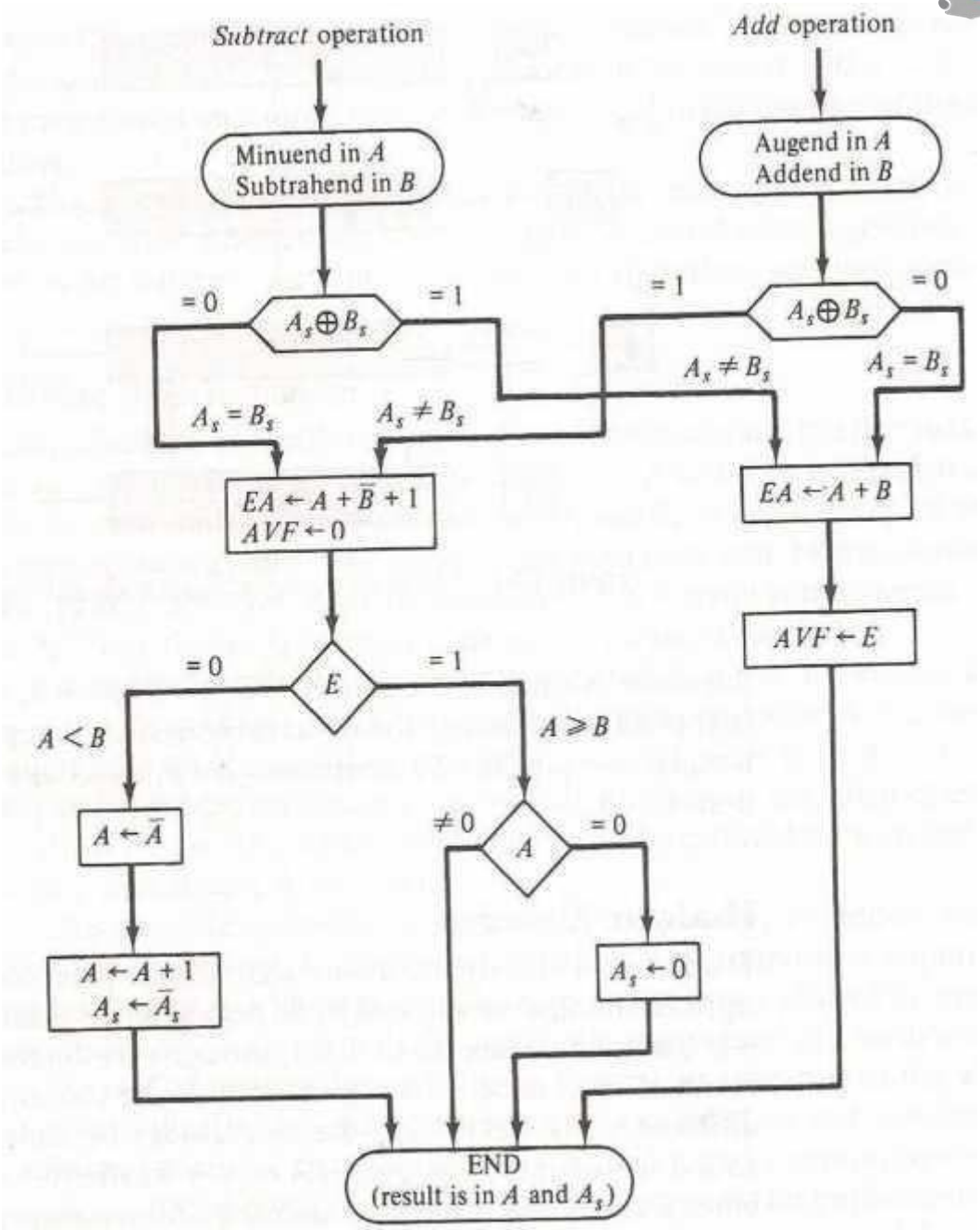


Figure 3: Flowchart of Add & Subtract Operation

The flowchart for the hardware algorithm is presented in Figure 3. The two signs A_S and B_S are compared by an exclusive-OR gate. If the output of the gate is 0, the signs are identical; if it is 1 then signs are different.

For an add operation, identical signs indicate that the magnitudes be added. For a subtract operation, different signs indicate that the magnitudes be added. The magnitudes are added with a micro operation EA<- A+B, where EA is a register that combines E and A. The carry in E after the addition constitutes an overflow if it is equal to1. The value of E is transferred into the add-overflow flip-flop AVF.

The two magnitudes are subtracted if the signs are different for an add operation or identical for a subtract operation. The magnitudes are subtracted by adding A to the 2's complement of B. No overflow can occur if the numbers are subtracted so AVF is cleared to 0. A 1in E indicates that A>=B and the number in A is the correct result. If this number is zero, the sign As must be made

positive to avoid a negative zero. A 0 in E indicates that A< B. For this case it is necessary to take the 2's complement of the value in A. This operation can be done with one micro operation A<-Ã + 1. Here, we assume that the A register has circuit for micro operations complement and increment, so the 2's complement is obtained from these two micro operations.

In other paths of the flowchart, the sign of the result is the same as the sign of A, so no change in AS is required. However, when A<B , the sign of the result is the complement of the original sign of A. It is then necessary to complement AS to obtain the correct sign. The final result is found in register A and its sign in AS.

Addition and Subtraction with Signed-2's Complement Data

The signed 2's complement representation of numbers together with arithmetic algorithms for addition and subtraction. The left most bit of a binary number represents the sign bit: 0 for positive and 1 for negative. If the sign bit is 1 the entire number is represented in 2's complement form. Thus +33 are represented as 00100001 and -33 as 11011111. Note that 11011111 is the 2's complement of 00100001 and vice versa.

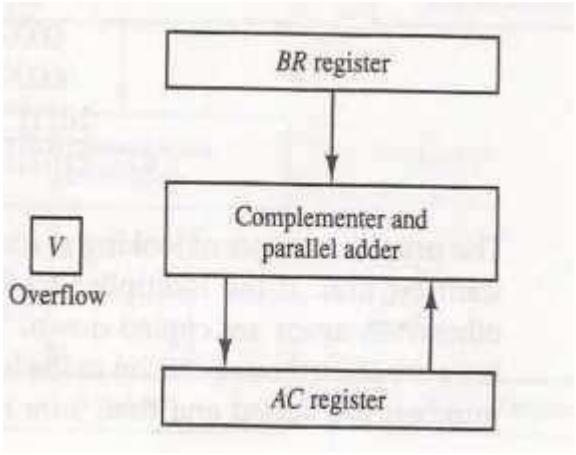


Figure 4: Hardware for signed 2'scomplement addition and subtraction

The addition of two numbers in signed 2's complement form consists of adding the numbers with the sign bits treated the same as the other bits of the number. A carry out of the sign bit position is discarded.

The subtraction consists of first taking the 2's complement of the subtrahend and then adding it to the minuend. When two numbers of n digits each are added and the sum occupies n+ 1 digit, we say that an overflow occurred. An overflow can be detected by inspecting the last two carries out of the addition.

The algorithm for adding and subtracting two binary numbers in signed- 2's complement representation is shown in the flowchart of Figure 5. The sum is obtained by adding the contents of AC and BR (including their sign bits).

The overflow bit V is set to 1 if the Exclusive-OR of the last two carries is l, and it is cleared to 0 otherwise. The subtraction operation is accomplished by adding the content of AC to the 2's complement of BR. Taking the 2's complement of BR has the effect of changing a positive number to negative, and vice versa.

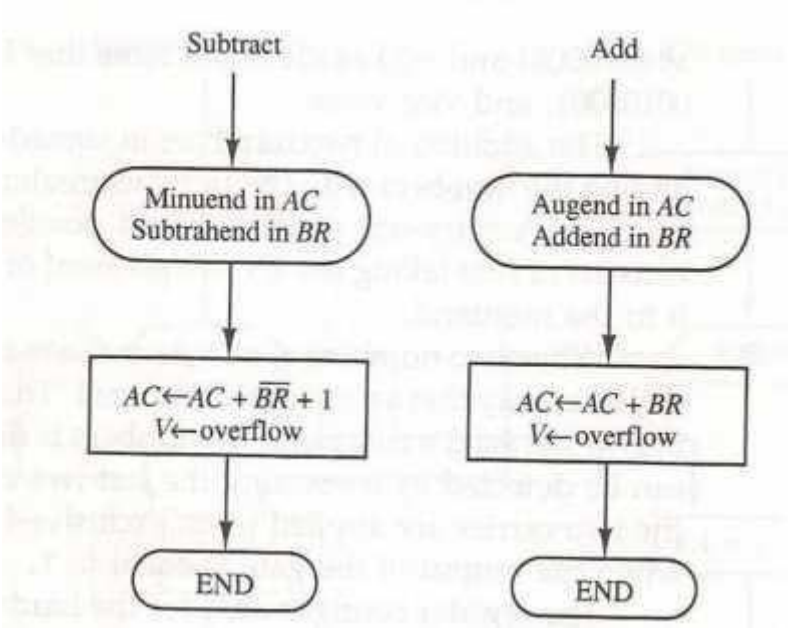


Figure 5: Algorithm for adding and subtracting numbers in signed-2's complement form representation

An overflow must be checked during this operation because the two numbers added could have same sign. The programmer must realize that if an overflow occurs, there will be an erroneous result in the AC register. Comparing this algorithm with its signed-magnitude counterpart, we note that it is much simpler to add and subtract numbers if negative numbers are maintained in signed 2's complement representation. For this reason most computers adopt this representation over the more familiar signed magnitude.

MULTIPLICATION ALGO. & FLOWCHART

Multiplication of two fixed-point binary numbers in signed magnitude representation is done with paper and pencil by a process of successive shift and adds operations. This process is best illustrated with a numerical example.

```
23      10111    Multiplicand
19    × 10011    Multiplier
        10111
       10111
      00000   +
     00000
    10111
437 110110101    Product
```

The process consists of looking at successive bits of the multiplier, LSB first. If the multiplier bit is a 1 multiplicand is copied down otherwise, zeros are copied down. The numbers copied down in successive lines are shifted one position to the left from the previous number. Finally, the numbers are added and their sum forms the product. The sign of the product is determined from the sign of the multiplicand and multiplier. If they are alike, the sign of the product is positive. If they are unlike, the sign of the product is negative.

As shown in Figure 6: initially, the multiplicand is in B and the multiplier in Q. Their corresponding signs are in B_S and Q_S respectively. The signs are compared, and both A and Q are set to correspond to the sign of the product since a double length product will be stored in registers A and Q. Registers A and E are cleared and the sequence counter SC is set to a number equal to the number of bits of the multiplier.

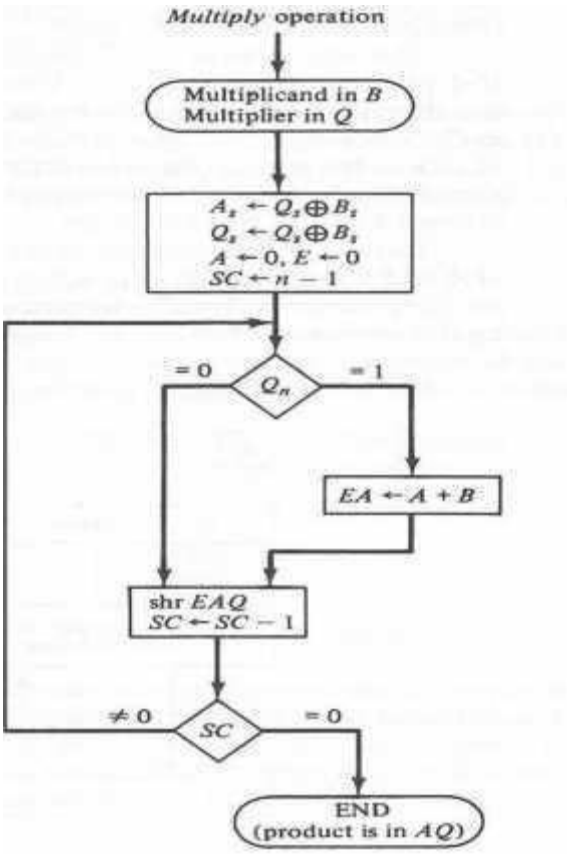


Figure 6: Flowchart of the hardware multiply algorithm.

We are assuming here that operands are transferred to registers from a memory unit that has words of n bits. Since operand must be stored with its sign, one bit of the word will be occupied by the sign and the magnitude will consist of n-1 bits.

After the initialization, the low order bit of the multiplier in Q is tested. If it is 1 the multiplicand in B is added to the present partial product in A. If it is 0, nothing is done. Register EAQ is then shifted once to the right to from the new partial product. The sequence counter is decremented by 1 and its new value checked. If it is not equal to zero, the process is repeated and a new partial product is formed. The process stops when SC= 0. Note that the partial product formed in A is shifted into Q one bit at a time and eventually replaces the multiplier. The final product is available in both A and Q, with A holding the most significant bits and Q holding the least significant bits.

Table 2: Binary Multiplication example

Multiplicand B = 10111	E	A	Q	SC
Multiplier in Q	0	00000	10011	101
Q_n = 1; add B		10111		
First partial product	0	10111		
Shift right EAQ	0	01011	11001	100
Q_n = 1; add B		10111		
Second partial product	1	00010		
Shift right EAQ	0	10001	01100	011
Q_n = 0; shift right EAQ	0	01000	10110	010
Q_n = 0; shift right EAQ	0	00100	01011	001
Q_n = 1; add B		10111		
Fifth partial product	0	11011		
Shift right EAQ	0	01101	10101	000
Final product in AQ = 0110110101				

Booth Multiplication Algorithm

Booth algorithm gives a procedure for multiplying binary integers in signed 2'scomplement representation.

It operates on the fact that strings of 0's in the multiplier require no addition but just shifting and a string of 1's in the multiplier from bit weight 2K to weight 2M can be treated as 2K+ l-2M.

Booth algorithm requires examination of the multiplier bits and shifting of the partial product. Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial product, or left unchanged according to the following rules:

1. The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier.
2. The multiplicand is added to the partial product upon encountering the first 0 (provided that there was a previous 1) in a string of 0' s in the multiplier.
3. The partial product does not change when the multiplier bit is identical to the previous multiplier bit.

Table 3: Example of booth Multiplication

$Q_n Q_{n+1}$	$BR = 10111$ $\overline{BR} + 1 = 01001$	AC	QR	Q_{n+1}	SC
	Initial	00000	10011	0	101
1 0	Subtract BR	01001			
		01001			
	ashr	00100	11001	1	100
1 1	ashr	00010	01100	1	011
0 1	Add BR	10111			
		11001			
	ashr	11100	10110	0	010
0 0	ashr	11110	01011	0	001
1 0	Subtract BR	01001			
		00111			
	ashr	00011	10101	1	000

The hardware implementation of Booth algorithm requires the register configuration shown in Figure 7. This is similar to Figure 6, above (multiply) except that the sign bits are not separated from the rest of the register. To show this difference, we rename registers A, B, and Q as AC, BR, and QR, If the two bits are equal to 10, it means that the first 1 in a string of 1's has been encountered. This requires a subtraction of the multiplicand from the partial product in AC. If the two bits are equal to 01 it means that the first 0 in a string of 0 's has been encountered. This requires the addition of the multiplicand to the partial product in AC . When the two bits are equal, the partial product does not change. An overflow cannot occur because the addition and subtraction of the multiplicand follow each other. As a consequence, the two numbers that are added always have opposite sign, a condition that excludes an overflow. The next step is to shift right the partial product and the multiplier (including bit Qn +1). This is an arithmetic shift right (ashr) operation which shifts AC and QR to the right and leaves the sign bit in AC unchanged. The sequence counter is decremented and the computational loop is repeated n times.

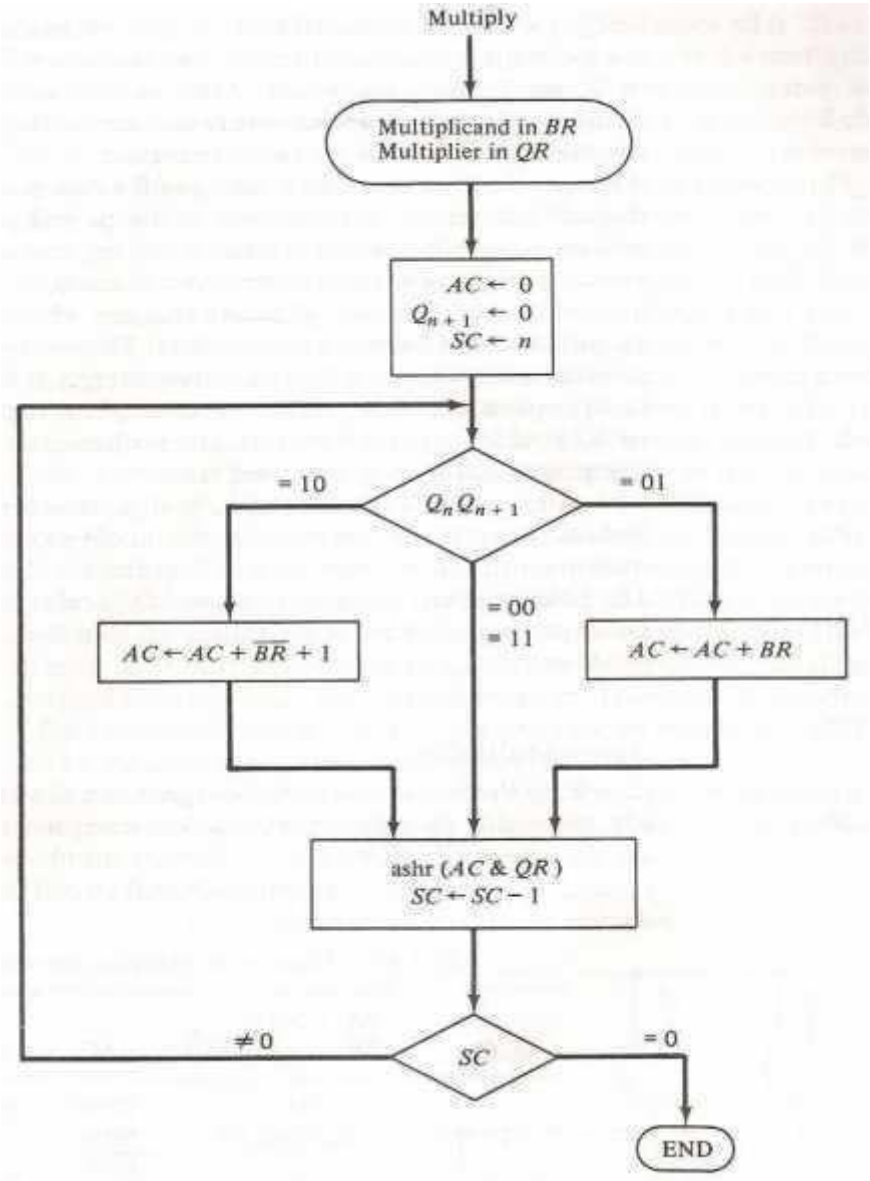


Figure 7: Flowchart for Booth Algorithm

DIVISION ALGO. & FLOWCHART

When the division is implemented in a digital computer, it is convenient to change the process slightly. Instead of shifting the divisor to the right, the dividend, or partial remainder, is shifted to the left, thus leaving the two numbers in the required relative position. Subtraction may be achieved by adding A to the 2's complement of B. The information about the relative magnitudes is then available from the end carry.

The hardware for implementing the division operation is identical to that required for multiplication and consists of the component shown above multiplication. In Figure 8 register EAQ is now shifted to the left with 0 inserted into Q_n and the previous value of E lost

The divisor is stored in the B register and the double length dividend is stored in registers A and Q .The dividend is shifted to the left and the divisor is subtracted by adding its 2's complement value. The information about the relative magnitude is available in E. If E=1 it signifies that A>=B. A quotient bit 1 is inserted into Q_n and the partial remainder is shifted to the left to repeat the process. If E= 0, it signifies that A< B so the quotient in Q_n.

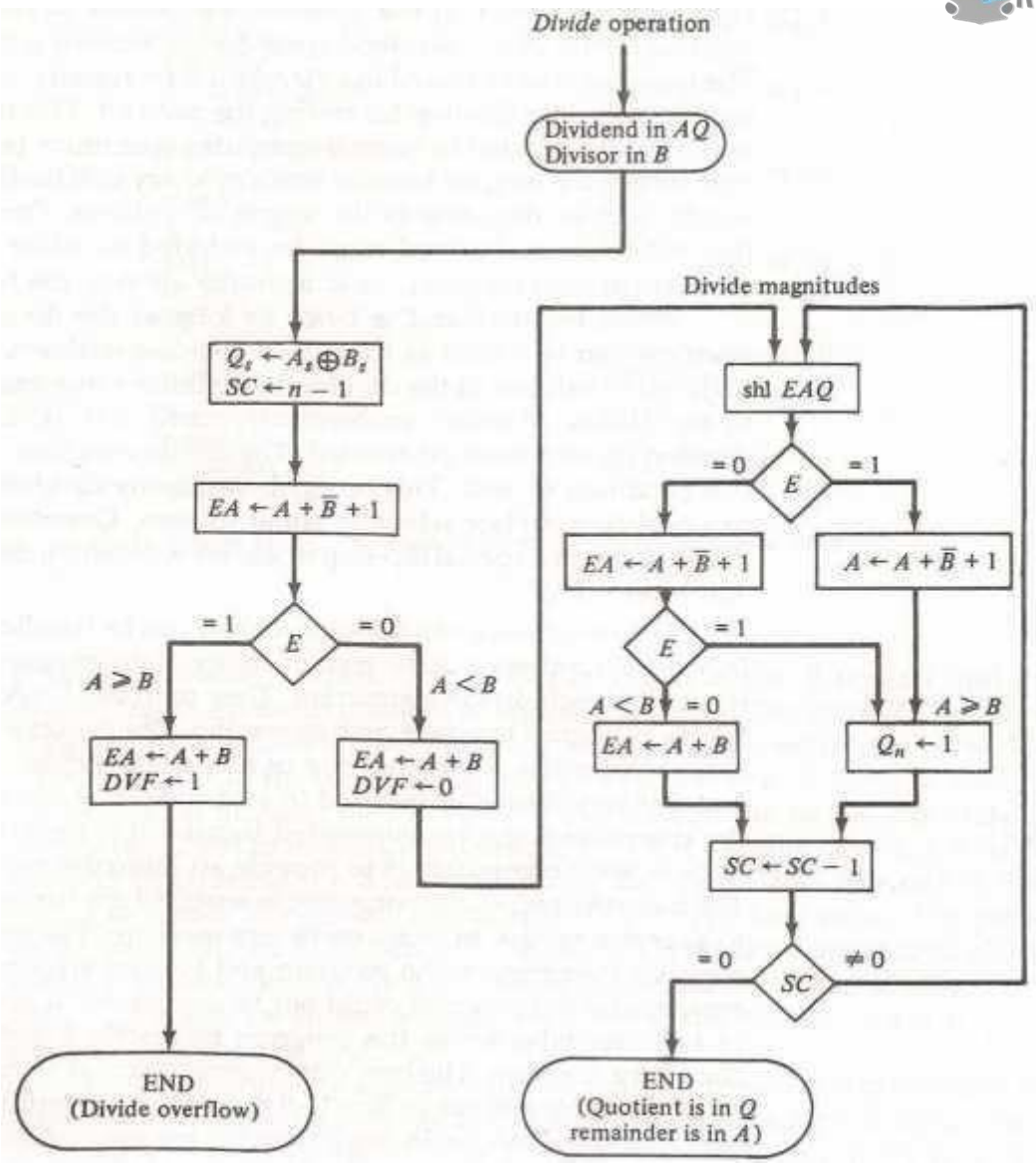


Figure 8: Flowchart for Division Algorithm

The hardware divide algorithm is shown in the flowchart of Figure 8.

- The dividend is in A and Q and the divisor in B.
- The sign of the result is transferred into QS to be part of the quotient.
- A constant is set into the sequence counter SC to specify the number of bits in the quotient.
- As in multiplication, we assume that operands are transferred to registers from a memory unit that has words of n bits.
- Since an operand must be stored with its sign, one bit of the word will be occupied by the sign and the magnitude will consist of n-1 bits.
- A divide overflow condition is tested by subtracting the divisor in B from half of the bits of the dividend stored in A.
- If A > B, the divide overflow flip-flop DVF is set and the operation is terminated prematurely.
- If A <B, no divide overflow occurs so the value of the dividend is restored by adding B to A.

Divisor $B = 10001$, $\overline{B} + 1 = 01111$

	E	A	Q	SC
Dividend:		01110	00000	5
shl EAQ	0	11100	00000	
add $\overline{B} + 1$		01111		
$E = 1$	1	01011		
Set $Q_n = 1$	1	01011	00001	4
shl EAQ	0	10110	00010	
Add $\overline{B} + 1$		01111		
$E = 1$	1	00101		
Set $Q_n = 1$	1	00101	00011	3
shl EAQ	0	01010	00110	
Add $\overline{B} + 1$		01111		
$E = 0$; leave $Q_n = 0$	0	11001	00110	
Add B		10001		
Restore remainder	1	01010		2
shl EAQ	0	10100	01100	
Add $\overline{B} + 1$		01111		
$E = 1$	1	00011		
Set $Q_n = 1$	1	00011	01101	1
shl EAQ	0	00110	11010	
Add $\overline{B} + 1$		01111		
$E = 0$; leave $Q_n = 0$	0	10101	11010	
Add B		10001		
Restore remainder	1	00110	11010	0
Neglect E				
Remainder in A:		00110		
Quotient in Q:			11010	

Figure 9: Division Algorithm Example

FLOATING POINT REPRESENTATION

To accommodate very large integers and very small fractions, 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 is automatically adjusted as computation proceeds.

In this case, the binary point is said to float, and the numbers are called floating- point numbers. The floating point representation has three fields: sign, significant digits and exponent.

To represent the number 111101.1000110 in floating point format, first binary point is shifted to right of the first bit and the number is multiplied by the correct scaling factor to get the same value. The number is said to be in the normalized form and is given as shown in below format.

5 <---------- Exponent

111101.1000110 ---1. 11101100110 x 2

Significant digits Scaling Factor

Number represented in normalized form

It is important to note that the base in the scaling factor is fixed 2. The string of the significant digits is commonly known as mantissa. In the above example

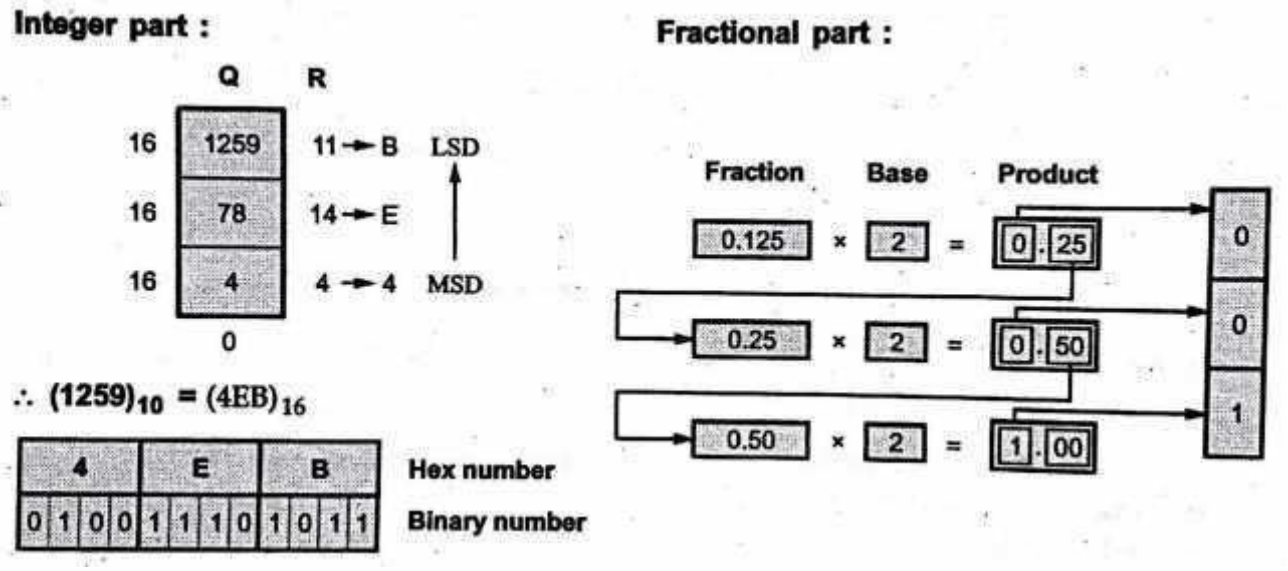
Sign = 0 Mantissa = 11101100110 Exponent = 5

In floating point numbers, bias value is added to the true exponent. This solves the problem of representation of negative exponent.

Example: Represent 1259.12510 in single and double precision format.

Solution:

Step 1: Convert decimal number in binary format



$(1259)_{10} = (1001110\ 1011)_2$ and $(0.125)_{10} = (0.001)_2$

Binary number = 10011101011 + 0.001 = 10011101011.001

Step 2: Normalize the number

$10011101011.001 = 1.0011101011001 \times 2^{10}$

Step 3: Single precision representation

For a given number S = 0, E = 10 and M = 0011101011001

Bias for single precision format is - 127

E' = E + 127 = 10 + 127 = 13710 = 100010012

Number in single precision format is given as

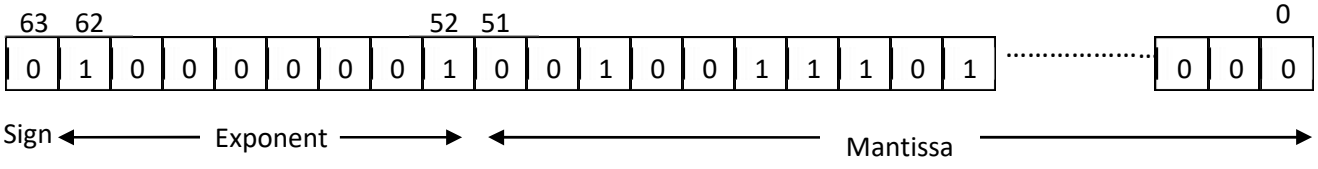
Step 4: Double precision representation. For a given number

S = 0, E = 10, and M = 0011101011001

Bias for double precision format is = 1023

E' = E+1023 = 10+1023 = 103310 = 100000010012

Number in double precision format is given as



Design of Arithmetic Unit:

ALU is responsible to perform the operation in the computer.

The basic operations are implemented in hardware level. ALU is having collection of two types of operations:

Arithmetic Operations (ADD, SUB, MUL, DIV)

Logical Operations (AND, OR, NOT, EX-OR)

Consider an ALU having 4 arithmetic operations and 4 logical operations.

To identify any one of these four logical operations or four arithmetic operations, two control lines are needed. Also to identify the any one of these two groups- arithmetic or logical, another control line is needed. So, with the help of three control lines, any one of these eight operations can be identified.

Consider an ALU is having four arithmetic operations. Addition, subtraction, multiplication and division. Also consider that the ALU is having four logical operations: OR, AND, NOT & EX-OR.

We need three control lines to identify any one of these operations. The input combination of these control lines are shown in Table 4:

Control line C_2 is used to identify the group: logical or arithmetic, i.e. C_2=0: arithmetic operation, C_2=1: logical operation.

Control lines $C_0$and C_1 are used to identify any one of the four operations in a group. One possible combination is given here.

Table 4: Operation Identification in ALU

C_1	C_0	Arithmetic $C_2 = 0$	Logical $C_2 = 1$
0	0	Addition	OR
0	1	Subtraction	AND
1	0	Multiplication	NOT
1	1	Division	**EX-OR**

Figure 10 shows the block diagram of ALU in which the ALU has got two input registers named as A and B and one output storage register, named as C. It performs the operation as:

$$C = A \text{ op } B$$

The input data are stored in A and B, and according to the operation specified in the control lines, the ALU perform the operation and put the result in register C.

As for example, if the contents of controls lines are, 000, then the decoder enables the addition operation and it activates the adder circuit and the addition operation is performed on the data that are available in storage register A and B. After the completion of the operation, the result is stored in register C.

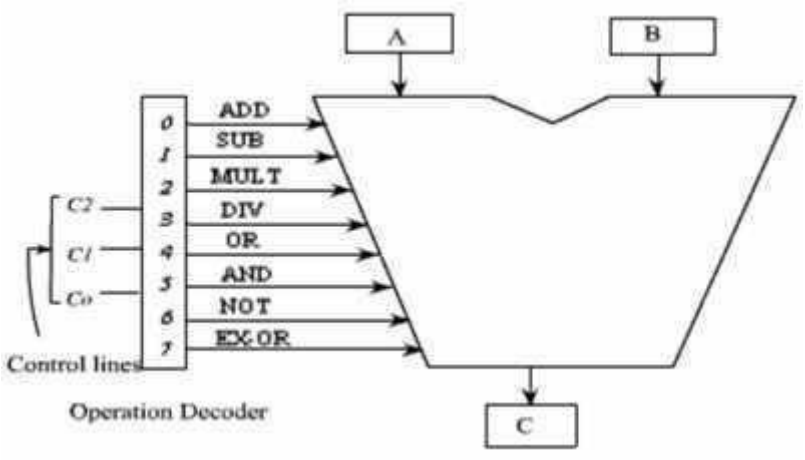


Figure 10: Block Diagram of ALU Design

UNIT – **4**

Syllabus: - I/O Organization: I/O Interface − PCI Bus, SCSI Bus, USB, Data Transfer: Serial, Parallel, Synchronous, Asynchronous Modes of Data Transfer, Direct Memory Access (DMA), I/O Processor.

Input Output Organization:

The input-output subsystems of computer referred to as I/O provides an efficient mode of communication between the central system and the outside world. The most familiar means of entering information into a computer is through a typewriter-like keyboard that allows a person to enter alphanumeric information directly.

Input and output devices attached to the computer are also called peripherals. Among the common peripherals are keyboards, display units, printers. Peripherals that provide auxiliary storage for the system are magnetic disks and tapes. Peripherals are electromechanical and electromagnetic devices of some complexity.

ASCII Alphanumeric Characters:

Input and output devices that communicate with people and the computer are usually involved in the transfer of alphanumeric information to and from the devices and the computer. ASCII (American Standard Code for Information Interchange). It uses 7 bits to code 128 characters as shown in table 1. The seven bits of the code are designated by b1 through b7, with b7 being the most significant bit. The letter A, for example is represented in ASCII as 1000001(column 100, row 001). The ASCII code contains 94 characters that can be printed and 34 nonprinting characters used for various control functions. The printing characters consists of 26 uppercase letters A through Z, 26 lowercase letters, 10 numerals 0 through 9 and 32 special printable characters such as %,* and $.

Table 1: American Standard Code for Information Interchange (ASCII)

$b_4b_3b_2b_1$	$b_7b_6b_5$							
	000	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0000	NUL	DLE	SP	0	@	P	.	P
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	“	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	U
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	‘	7	G	W	g	W
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	-	o	DEL

Input Output Interface:

I/O interface provides a method for transferring information between internal storage and external storage and external I/O devices. Peripherals connected to a computer need special communication links for interfacing them with the CPU. The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral.

The major differences are:
Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory, which are electronic devices. Therefore a conversion of signal values may be required.
The data transfer rate of peripherals is usually slower than the transfer rate of the CPU, and consequently, a synchronization mechanism may be needed.
Data codes and formats in peripherals differ from the word format in the CPU and memory.
The operation modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.
To resolve these differences, computer systems include special hardware components between the CPU and peripherals to supervise and synchronize all input and output transfers. These components are called interface units because they interface between the processor bus and the peripheral device.

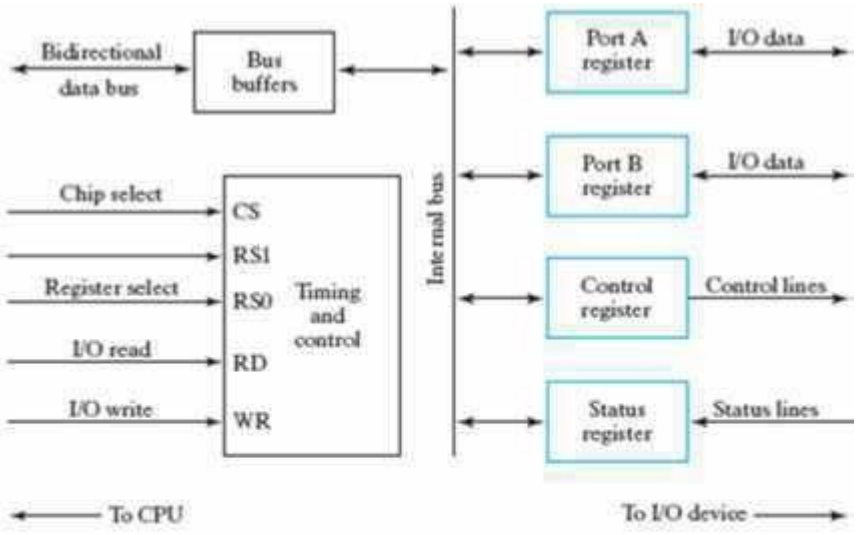
Example of I/O Interface:

Figure 1 shows the block diagram of I/O interface unit. It consists of 2 data registers called ports, a control register, a status register, bus buffers and timing and control circuits.

- The interface communicates with the CPU through data bus.
- The chip select (CS) and register select (RS) inputs determine the address assigned to the interface.
- The I/O read and write are two control lines that specify an input or output respectively.
- The four registers communicate directly with the I/O device attached to the interface.
- The I/O data to and from the device can be transferred into either port A or port B. T

The interface may operate with an output device or with an input device or with a device that requires both input and output.

In this example, address bus selects the interface unit through CS and RS1 & RS0 and a particular interface is selected by the circuit (decoder) enabling CS, RS1 and RS0 select one of 4 registers.
The interface registers communicate with the CPU through the bidirectional data bus. The address bus selects the interface unit through the chip select the two register select inputs. A circuit must be provided externally to detect the address assigned to the interface registers. This circuit enables the chip select input when the interface is selected by the address bus. The two register select inputs RS1 and RS0 are usually connected to the two least significant lines of the address bus. These two inputs select one of the four registers in the interface as specified in the table. The content of the selected register is transfer into the CPU via the data bus when the I/O read signal is enabled. The CPU transfers binary information into the selected register via the data bus when the I/O write input is enabled.



CS	RS1	RS0	Register selected
0	X	X	None: data bus in high-impedance state
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

Figure 1: Example of I/O Interface Unit

PCI Bus - A Peripheral Component Interconnect Bus (PCI bus) connects the CPU and expansion boards such as modem cards, network cards and sound cards. These expansion boards are normally plugged into expansion slots on the motherboard.

Features:

It is designed to economically meet the I/O requirements of modern systems. It requires very few chips to implement and support other buses attached to the PCI bus.

It bypasses the standard I/O bus, uses the system bus to increase the bus clock speed and take full advantage of the CPU's data path.

It has an ability to function with a 64 bit data bus.

- It has high bandwidth. The information is transferred across the PCI bus at 33MHz, at the full data width of the CPU.

PCI Configuration

Figure 2(a) shows a typical use of PCI in a single processor system. Notice that the processor bus is separate and independent of the PCI bus. The processor connects to the PCI bus through an integrated circuit called a PCI bridge. The memory controller and PCI Bridge provides tight coupling with the processor and delivers data at high speeds. Figure 2(b) shows a typical use of PCI in a multiprocessor system. As shown in the Figure 2(b) in multiprocessor systems one or more PCI configurations may be connected by bridges to the processor's system bus. Again, the use of bridges keeps the PCI independent of the processor speed yet provides the ability to receive and deliver data rapidly.

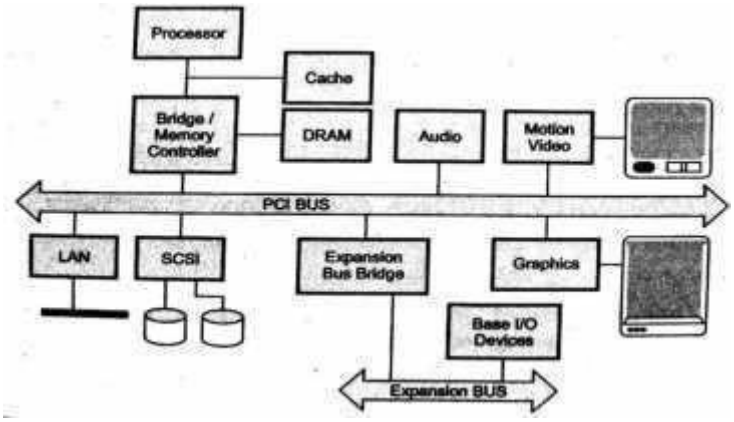


Figure 2(a): Conceptual design of PCI bus for single processor system

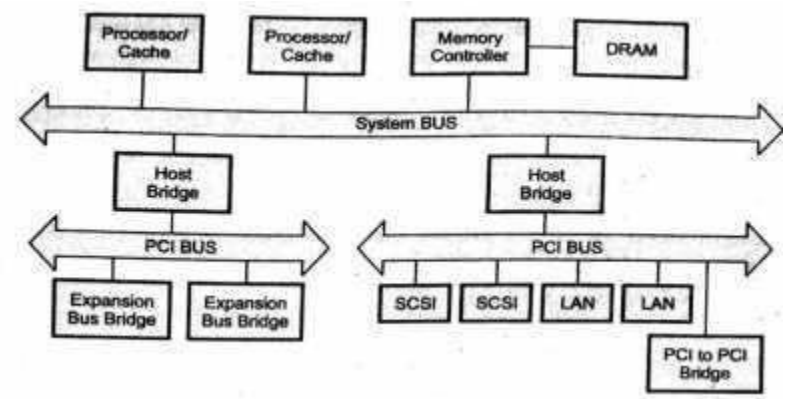


Figure 2 (b) Conceptual diagram of PCI bus for multiprocessor system

Data Transfer

Every data transfer on the PCI bus is a single transaction consisting of one address phase and one or more data phases. All the events during read cycle are synchronized with the falling edge of the clock cycle.

SCSI Bus

The acronym SCSI stands for small computer system interface. It was adopted as a standard by the American National Standard Institute (ANSI) in 1986. This bus connects I/O devices such as hard disk units and printers to personal computers. SCSI was originally designed to transfer data a byte at a time at rates up to 5 MB/s. The Figure 3 shows the SCSI I/O bus.

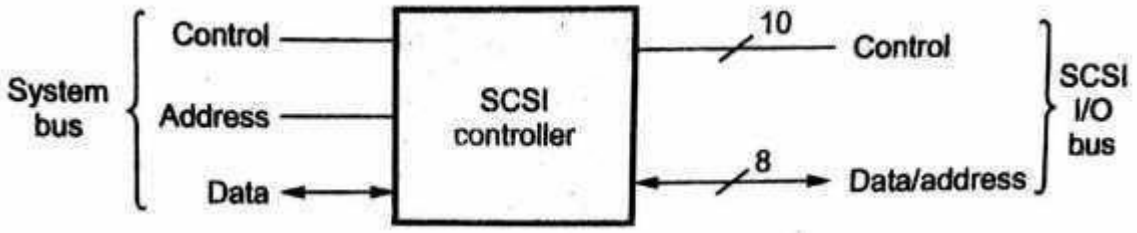


Figure 3: SCSI I/O bus

SCSI is smaller and simpler bus and its data sub bus is only 8-bits wide. Its data bus is also used to transfer addresses. Ten additional lines provide all the necessary control functions. Recent extensions to the original SCSI standard have wider data buses (16 and 32 bits), more control features and higher data transfer rates.

SCSI Standard

SCSI-1 defines the basics of the first SCSI buses, including cable length signaling characteristics, commands and transfer modes.

- Devices corresponding to the SCSI-1 standard use only a narrow (8- bit) bus, With a 5 MB/s maximum transfer rate.
- Only single-ended transmission was supported, with passive termination.
- There were also difficulties associated with the standard gaining universal acceptance due to the fact that many manufacturers implemented different subsets of its features.
- Devices that adhere to the SCSI-l standard in most cases can be used with host adapters and other devices that use the higher transfer rates of the more advanced SCSI-2 protocols, but they will still function at their original slow speed.

SCSI-2 is an extensive enhancement over SCSI-1. In addition, the standard defines the following significant new features as additions to the original SCSI-1 specification:

1. **Fast SCSI:** This higher-speed transfer protocol doubles the speed of the bus to 10 MHz
2. **Wide SCSI:** The width of the original SCSI bus was increased to 16 (or even 32) bits. This permits more data throughput at a given signaling speed.
3. **More Devices per Bus**: On buses that are running with Wide SCSI, 16 devices are supported.

SCSI-3 - SCSI-3 specification defines the mechanical, electrical and protocol layers of the interface. Data transfers of 8 b its at 20Mbps over a 50 pin connector and 16 bits at 40Mbps over a 68 pin connector. The number of devices on the bus increased to 16 (for Fast-10), Fast-20 allows 8 devices maximum with a number of other combinations

Universal Serial Bus (USB)

USB gives fast and flexible interface for connecting all kinds of peripherals. USB is playing a key role in fast growing consumer areas like digital imaging, PC telephony, and multimedia; games, etc. The presence of USB in most new PCs and its plug-n- play capability, means that PCs and peripherals (such as CD ROM drives, tape and floppy drives, scanners, printers, video devices, digital cameras, digital speakers, telephones, modems, key boards, mice, digital joysticks and others) will automatically configure and work together, With high degree of reliability, in this exciting new application areas.

USB Features

- **Simple cables**
- **One interface for many devices**
- **Automatic configuration**
- **No user setting**

Data Transfer: Serial, Parallel

The transfer of data between two units may be done in two ways: parallel or serial. In **serial data transmission**, each bit in the message is sent in sequence one at a time. This method requires the use of one pair of conductors or one conductor and a common ground. Serial transmission is slower but is less expensive since it requires only one pair of conductors.

In **parallel data transmission**, each bit of the message has its own path and the total message is transmitted at the same time. This means that an n bit message must be transmitted through n separate conductor paths. Parallel transmission is faster but requires many wires. It is used for short distances and where speed is important.

Serial transmission can be synchronous or asynchronous.

In **synchronous transmission**, the two units share a common clock frequency and bits are transmitted continuously at the rate dictated by the clock pulses. In long distant serial transmission, each unit is driven by a separate clock of the same frequency. Synchronization signals are transmitted periodically between the two units to keep their clocks in step with each other.

In **asynchronous transmission**, binary information is sent only when it is available and the line remains idle when there is no information to be transmitted. This is in contrast to synchronous transmission, where bits must be transmitted continuously to keep the clock frequency in both units synchronized with each other .

Asynchronous serial transmission:

With this technique, each character consists of three parts: a start bit, the character bits, and stop bits. The convention is that the transmitter rests at the state when no characters are transmitted. The first bit, called the start bit, is always a 0 and is used to indicate the beginning of a character. The last bit called the stop bit is always a 1. A transmitted character can be detected by the receiver from knowledge of the transmission rules:

1. When a character is not being sent, the line is kept in the 1 state.
2. The initiation of a character transmission is detected from the start bit, which is always 0.
3. The character bits always follow the start bit.

After the last bit of the character is transmitted, a stop bit is detected when the line returns to the 1 state for at least one bit time. Using these rules, the receiver can detect the start bit when the line goes from 1 to 0. A clock in the receiver examines the line at proper bit times. The receiver knows the transfer rate of the bits and the number of character bits to accept. After the character bits are transmitted, one or two stop bits are sent. The stop bits are always in the 1 state and frame the end of the character to signify the idle or wait state.

Synchronous Modes of Data Transfer

In a synchronous data transfer, all devices derive timing information from a common clock signal. The Figure 4 shows the timing diagram for synchronous input/output transfer At time t_o, the processor places the device address on the address lines of System bus and sets the control lines to perform the input operation. During time t_0- t_1, addressed device gets, the address and it recognizes that an input operation is requested. At time t_1, address device places its data on the data bus. At the end of bus cycle that is at time t_2 the processor reads the data lines and loads the data from data bus into its input buffer.

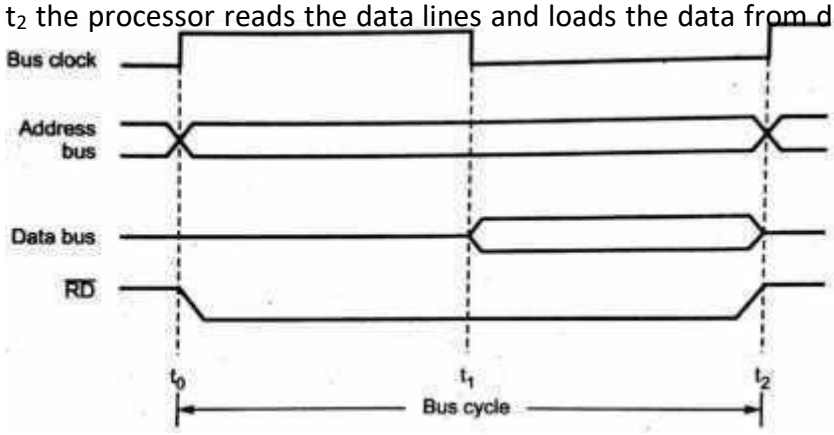


Figure 4: Timing diagram for synchronous input transfer

The timing diagrams shown in Figure 5 do not consider the propagation delay so they represent ideal timings for data transfer that takes place on lines. In practice the propagation delay, delays the timings at which signals actually change state. This is illustrated in Figure 6. The Figure 6 shows the signals as seen by i the master and the other as seen by the slave.

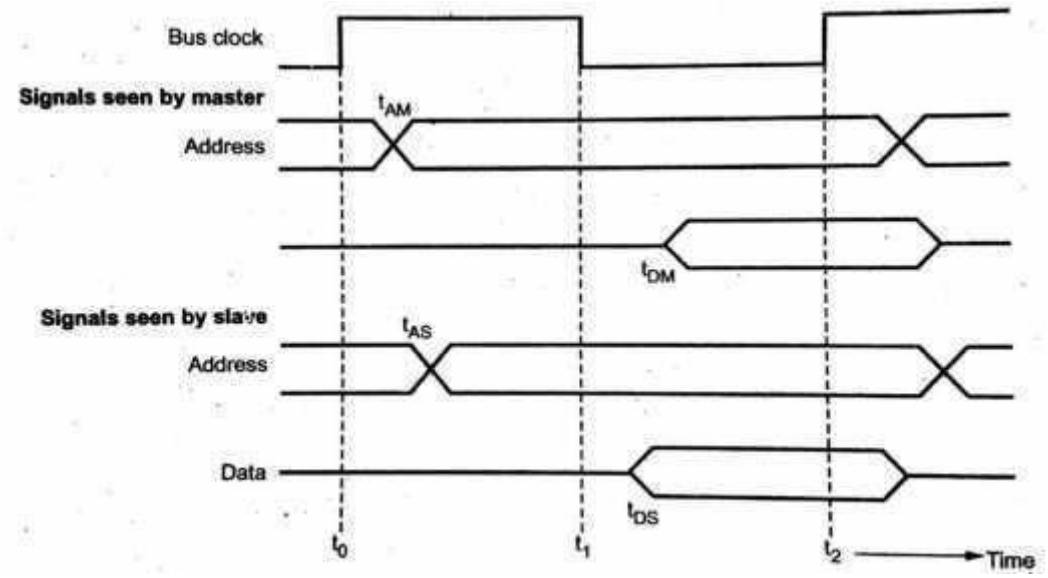


Figure 5: Timing diagram for synchronous output

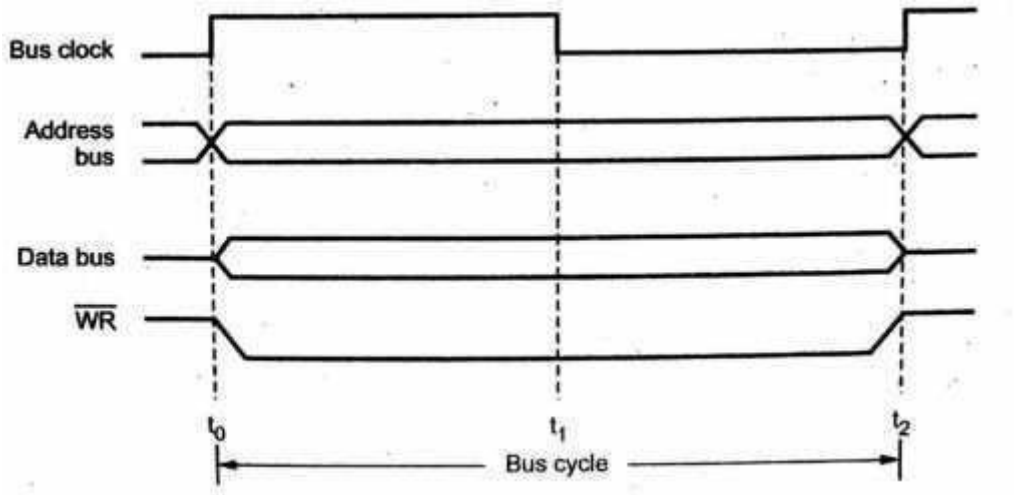


Figure 6: Timing diagram for synchronous input transfer considering propagation delay;

The master sends address signals on the rising edge at the beginning of t_0 but due to the delay in the bus driver circuit address signals appear on the address bus after time delay of t_{AM}. After some more delay, at t_{As}, the address signals reach the slave. The slave decodes the address and at t_1 sends the requested data. Here again, the data signals do not appear on the data bus until t_{DS}. Data signals reach to the master at time t_{DM}. At t_2, the master reads and loads the data into its input buffer; The time t_2 - t_{DM} is known as data setup time for master's input buffer. The data must continue to be valid after t_2 for a period equal to the hold time of the master's input buffer.

Limitations:- In this mode the transfer has to be completed within one clock cycle, the clock period, t_z—t_o, must be selected to accommodate the longest delays on the bus and the slowest device interface. This forces all devices to operate at the speed of the slowest device.

Serial Communication:

In parallel communication number of lines required to transfer data depend on the number of bits to be transferred. For example, to transfer a byte of data, 8 lines are required and all 8 bits are transferred

simultaneously. Thus for transmitting data over a long distance, using parallel communication is impractical due to the increase in cost of cabling.

Table 2 : Difference between Serial and Parallel Data Transfer

S. No.	Serial Data Transfer	Parallel Data Transfer
1	It transfer data one bit at a time	It can transmit more than one data bit at a time.
2	Lower data transfer rate	Faster data transfer rate
3	Needs less number of wires to connect devices in the system.	It needs more number of wires to connect devices in the system.
4	Well suited for long distances, because fewer wires are used as compared to a parallel bus.	The inter connection penalty increases as distances increase. Thus not suitable for long distance communication.

Asynchronous Serial Communication

Figure 8 shows the transmission format for asynchronous serial transmission. Asynchronous formats are character oriented. In this, the bits of a character or data word are sent at a constant rate, but characters can come at any rate (asynchronously) as long as they do not overlap. When no characters are being sent, a line stays high at logic 1 called mark, logic 0 is called space. The beginning of a character is indicated by a start bit which is always low. This is used to synchronize the transmitter and receiver. After the start bit, the data bits are sent with least significant bit first, followed by one or more stop bits (active high). The stop bits indicate the end of character.

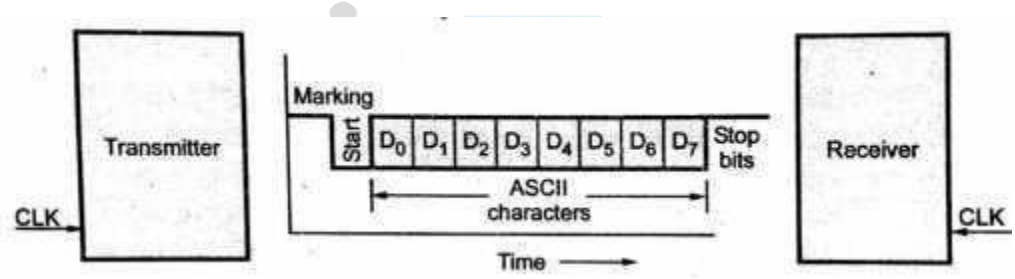


Figure 7: Transmission format for asynchronous transmission

The data rate can be expressed as bits/sec. or characters/sec. The term bits/sec is also called the baud rate. The asynchronous format is generally used in low-speed transmission (less than 20 Kbits/ sec).

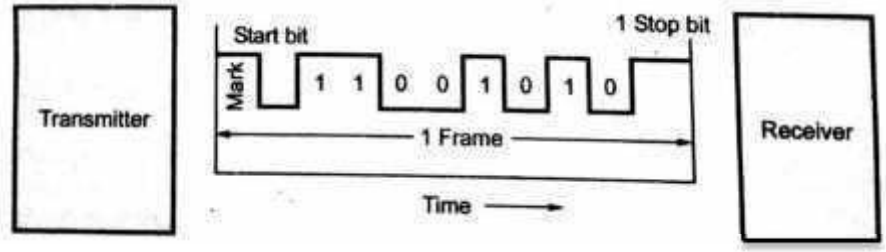


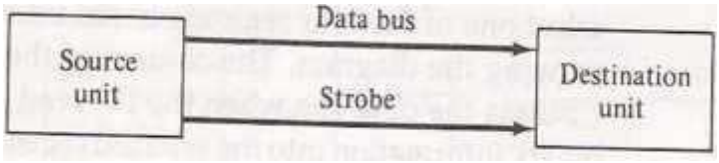
Figure 8 : Asynchronous com with m byte CAH

Asynchronous Modes of Data Transfer

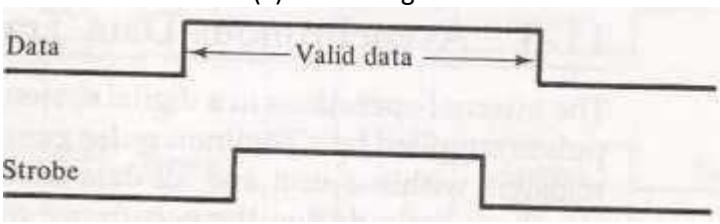
Asynchronous data transfer between two independent units requires that control signals be transmitted between the communicating units to indicate the time at which data is being transmitted. One way of achieving this is by means of a strobe.

Pulse supplied by one of the units to indicate to the other unit when the transfer has to occur.
The unit receiving the data item responds with another control signal to acknowledge receipt of the data.
This type of agreement between two independent units is referred to as handshaking.
The strobe pulse method and the handshaking method of asynchronous data transfer are not restricted to I/O transfers.

Strobe Control - The strobe control method of asynchronous data transfer employ a single control line to time each transfer. The strobe may be activated by either the source or the destination unit. Figure (a) shows a source initiated transfer. The, data bus carries the binary information from source unit to the destination unit. Typically, the bus has multiple lines to transfer an entire byte or word. The strobe is a single line that informs the destination unit when a valid data word is available in the bus.



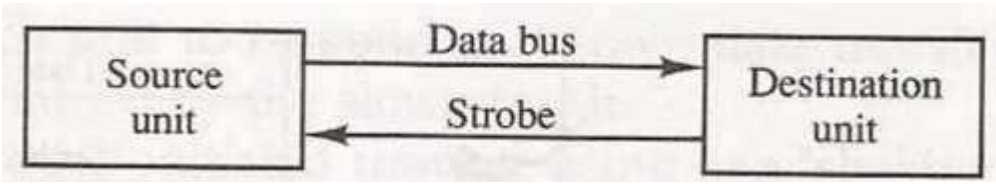
(a) Block diagram



(b) Timing diagram

Figure 9: Source initiated strobe for data transfer.

Figure 10 shows a data transfer initiated by the destination unit. In this case the destination unit activates the strobe pulse, informing the source to provide the data. The source unit responds by placing the requested binary information on the data bus. The data must be valid and remain in the bus long enough for the destination unit to accept it.



(a) Block diagram

Figure 10: Destination initiated strobe for data transfer

Handshaking - The disadvantage of the strobe method is that the source unit that initiates the transfer has no way of knowing whether the destination unit has actually received the data item that was placed in the bus. Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has actually placed the data on the bus. The handshake method solves this problem by introducing a second control signal that provides a reply to the unit that initiates the transfer. One control line is in the same direction as the data flow in the bus from the source to the destination. It is used by the source unit to inform the destination unit whether there are valid data in the bus .The other control line is in the other direction from the destination to the source. It is used by the destination unit to inform the source whether it can accept data. The sequence of control during the transfer depends on the unit that initiates the transfer.

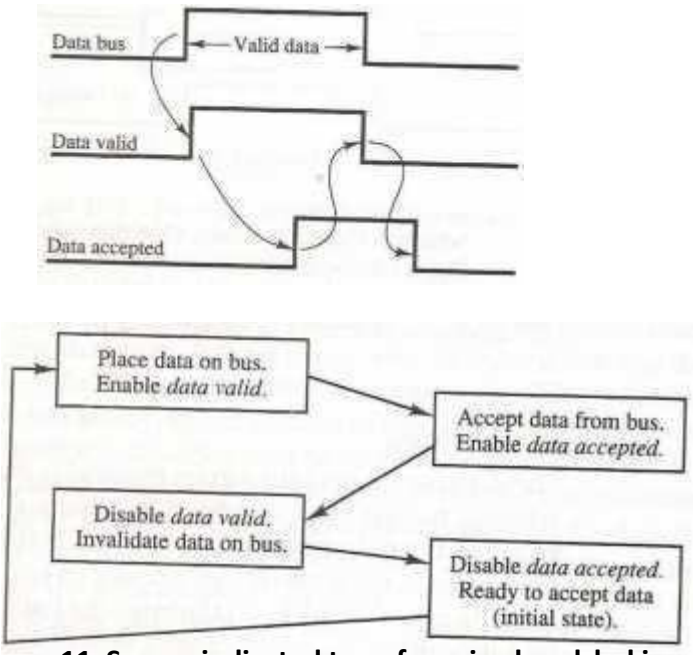


Figure 11: Source indicated transfer using handshaking.

I/O Interfacing Techniques

I/O devices can be interfaced to a computer system I/O in two Ways, which are filled interfacing techniques, 1. Memory mapped I/O 2. I/O mapped I/O

Memory mapped I/O **–** In this technique, the total memory address space is partitioned and part of this space is devoted to I/O addressing as shown in Fig. 19. When this technique is used, a memory reference instruction that causes data to be fetched from or stored at address specified, automatically becomes an I/O instruction if that address is made the address of an I/O port.

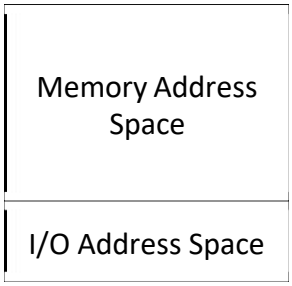


Figure 12: Address space

Advantage - The usual memory related instructions are used for I/O related operations. The special I/O instructions are not required.

Disadvantage - The memory address space is reduced.

I/O mapped I/O - If we do not want to reduce the memory address space, we allot a different I/O address space, apart from total memory space which is called I/O mapped I/O technique as shown in Figure 17.

Advantage - The advantage is that the full memory address space is available.

Disadvantage - The memory related instructions do not work, Therefore, processor can only use this mode if it has special instructions for No related operations such as I/O read, I/O write.

Table 3: Memory Mapped I/O, I/O Mapped I/O Comparison

S. No.	Memory Mapped I/O	I/O Mapped I/O
1	Memory and I/O share the entire address range of processor	Processor provides separate address range for memory and I/O devices.
2	Usually, processor provides more address lines for accessing memory. Therefore more decoding is required control signals.	Usually, processor provides less address I/O. Therefore less decoding is required.
3	Memory control signals are used to control read and write I/O operations.	I/O control signals are used to control read and write I/O operations.

Direct Memory Access

In software control data transfer, processor executes a series of instructions to carry out data transfer. For each instruction execution fetch, decode and execute phases are required. Figure 21 gives the flowchart to transfer data from memory to I/O device. Thus to carry out these tasks processor requires considerable time. So this method of data transfer is not suitable for large data transfers such as data transfer from magnetic disk or optical disk to memory.

Drawbacks in programmed I/O and interrupt driven I/O

- The I/O transfer rate is limited by the speed with which the CPU can test and service a device.
- The time that the CPU spends testing I/O device status and executing a number of instructions for I/O data transfers can often be better spent on other processing tasks.
- To overcome above drawbacks an alternative technique, hardware controlled data transfer can be used.

DMA Block Diagram

For performing the DMA operation, the basic blocks required in a DMA channel controller are shown in Figure 13. DMA controller communicates with the CPU via the data bus and control lines. The registers in DMA are selected by the CPU through the address bus by enabling the DS (DMA select) and RS (Register select) inputs. The RD (Read) and WR (write) inputs are bidirectional.

When the BG (bus grant) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write the DMA registers RD and WR signals are input signals for DMA.
When BG=1, the CPU has relinquished the buses and the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR signals (RD and WR are now output signals for DMA). DMA consists of data count, data register, address register and control-logic.

Data counter register stores the number which gives the number data transfer to be done in one DMA cycle. It is automatically decremented after each word transfer. Data register acts as buffer whereas address register initially holds the starting address of the device. Actually, it stores the address of the next word to be transferred. It is automatically incremented or decremented after each word transfer. After each transfer, data counter is tested for zero. When the data count reaches zero, the DMA transfer halts. The DMA controller is normally provided with an interrupts capability in Which case it sends an interrupt to processor to signal the end of the I/O data transfer.

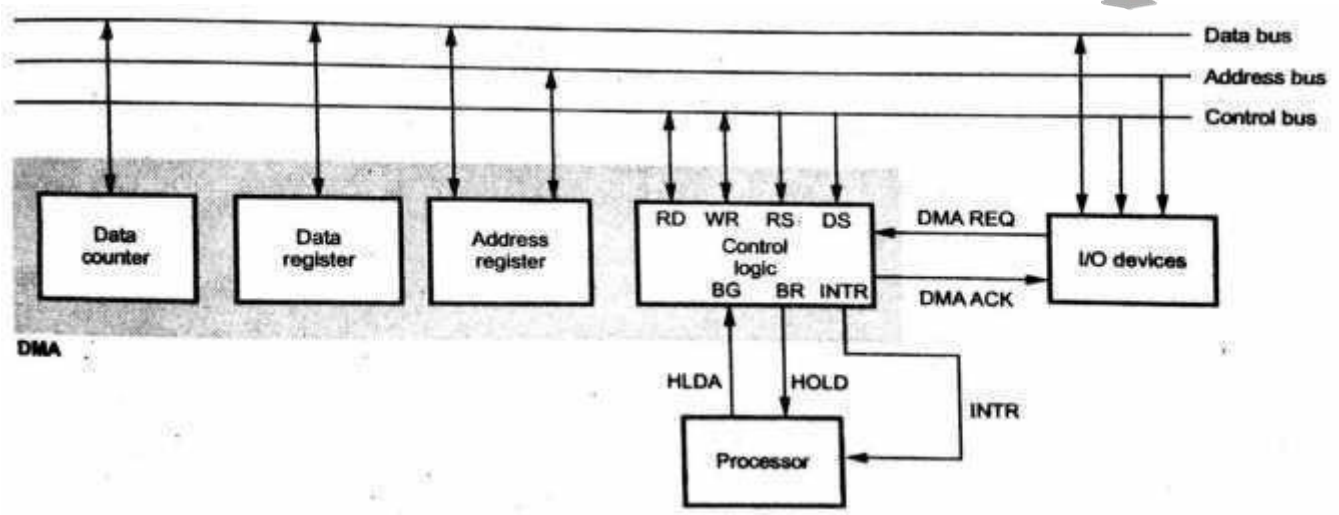


Figure 13: Typical DMA block Diagram

Data Transfer Modes

DMA controller transfers data in one of the following three modes:
1. Single transfer mode (cycle stealing)
2. Block transfer mode
3. Demand or burst transfer mode

Single transfer mode (Cycle stealing mode) - In this mode device can make only one transfer (byte or word). After each transfer DMAC gives the control of all buses to the processor. Due to this processor can have access to the buses on a regular basis. It allows the DMAC to time share the, buses with the processor, hence this mode is most commonly used.

Block transfer mode – In this mode device can make number of transfers as programmed in the word count register. After each transfer word count is decremented by 1 and the address is decremented or incremented by 1. The DMA transfer is continued until the word count "role over" from zero to FFFFH, a Terminal Count (TC) or an external END of Process (EOP) is encountered. Block transfer mode is used when the DMAC needs to transfer a block of data.

Demand transfer mode - In this mode the device is programmed to continue making transfers until a TC or external W is encountered or until DREQ goes inactive.

I/O Processor:

The I/O processor (IOP) has an ability to execute I/O instructions and it can have complete control over me /0 operations. The I/0 instructions are stored in main memory. When I/0 transfer is required, the CPU initiates an I/0 transfer by instructing the I/0 channel to execute an I/O program stored in the main memory. The I/0 program specifies the device or devices, the area of memory storage, priority and actions to be taken for certain error conditions.

Block Diagram of IOP: Figure 14 shows the block diagram of computer system with an I/O processor. The CPU and I/O processor work independently and communicate with each other using centrally located memory and DMA. The CPU does the processing of needed in the solution-of computational tasks and IOP does the data transfer between various peripheral devices and the memory unit.

CPU and IOP Communication - The communication between CPU and IOP may be different for different processor and IOP configurations. However, in most of cases the memory based control blocks are used to store the information about the task to be performed. The processor uses these blocks to leave information in it for the other processor. The memory control block is linked, i.e. the address of the next memory based control blocks is available in the previous memory based control block.

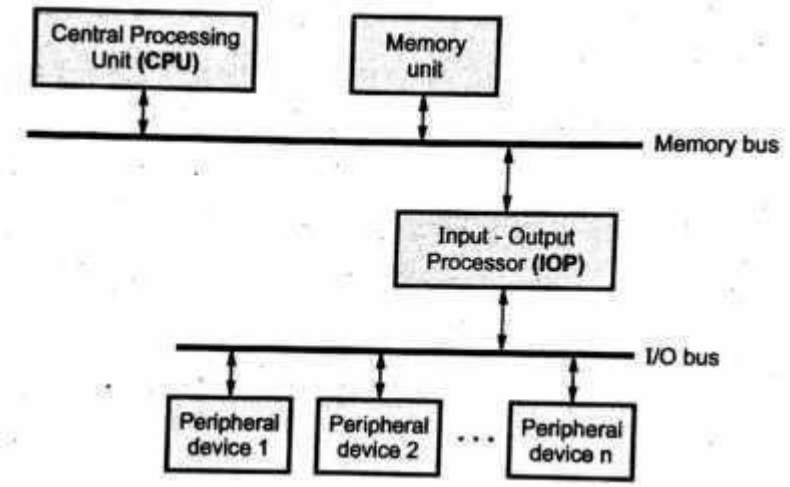


Figure 14: Block diagram of computer with I/O processor

Unit -5

Syllabus: Memory Organization: Main memory- RAM, ROM, Secondary Memory – Magnetic Tape, Disk, Optical Storage, Cache Memory: Cache Structure and Design, Mapping Scheme, Replacement Algorithm, Improving Cache Performance, Virtual Memory, memory management hardware.

Memory Organization:

The memory unit is an essential component in any digital computer since it is needed for storing programs and data. The memory unit that communicates directly with CPU is called the main memory. Devices that provide backup storage are called auxiliary memory. The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. They are used for storing system programs, large data files, and other backup information. Only programs and data currently needed by the processor reside in main memory. All the other information is stored in auxiliary memory and transferred to main memory when needed.

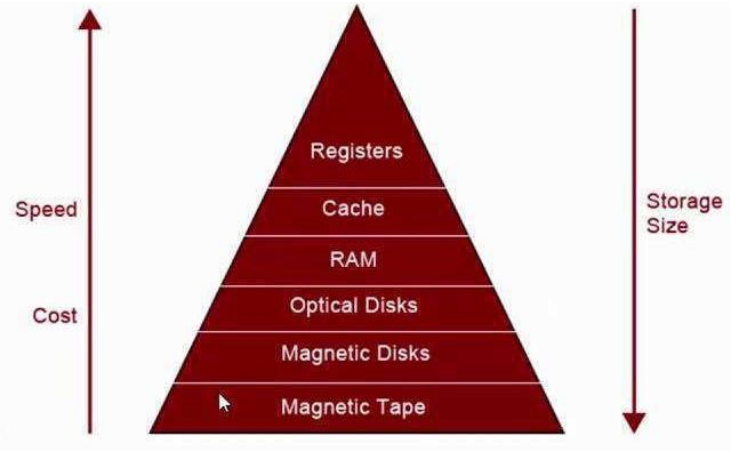


Figure 1: Memory hierarchy

Figure 1 illustrates the components in a typical memory hierarchy.

- At the bottom of the hierarchy are the relatively slow magnetic tapes used to store removable files.
- Next are the magnetic disks used as backup storage.
- The main memory occupies a central position by being able to communicate directly with the CPU and with auxiliary memory devices through an I/O processor.
- A special very-high-speed memory called a cache memory is sometimes used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate.
- The cache memory is employed in computer systems to compensate for the speed differential between main memory access time and processor logic.

MAIN MEMORY:

- The main memory is the central storage unit in a computer system.
- It is a relatively fast memory used to store programs and data during the computer operation.
- The principal technology used for the main memory is based on semiconductor integrated circuits.
- Integrated circuit RAM chips are available in two possible operating modes, static and dynamic.
- The static RAM consists essentially of internal flip-flops that store the binary information. The stored information remains valid as long as power is applied to the unit.
- The dynamic RAM stores the binary information in the form of electric charges that are applied to capacitors.
- The capacitors are provided inside the chip by MOS transistors.

- The stored charge on the capacitors tends to discharge with time and the capacitors must be periodically recharged by refreshing the dynamic memory.
- ROM portion of main memory is needed for storing an initial program called a bootstrap loader. The bootstrap loader is a program whose function is to start the computer software operating when power is turned on.

RAM CHIPS:

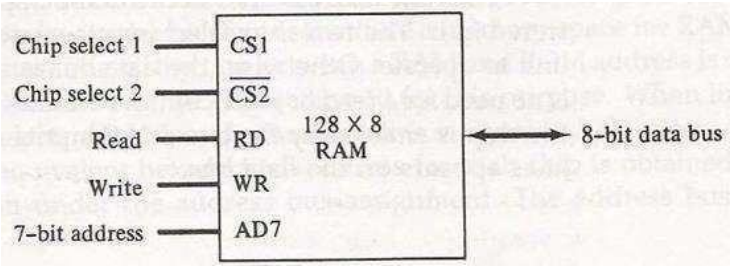


Figure 2:Block diagram of a RAM chip

The block diagram of a RAM chip is shown in Figure 2. The capacity of the memory is 128 words of eight bits (one byte) per word. This requires a 7-bit address and an 8- bit bidirectional data bus.

The read and write inputs specify the memory operation and the two chips select (CS) control inputs are enabling the chip only when it is selected by the microprocessor. The availability of more than one control input to select the chip facilitates the decoding of the address lines when multiplechips are used in the microcomputer. The read and write inputs are sometimes combined into one line labeled R/W. When the chip is selected, the two binary states in this line specify the two operations of read or write. The function table listed in Figure 3 specifies the operation of the RAM chip. When the WR input is enabled, the memory stores a byte from the data bus into a location specified by the address input lines. When the RDinput is enabled, the content of the selected byte is placed into the data bus. The RD and WR signals control the memory operation as well as the bus buffers associated with the bidirectional data bus.

CS1	$\overline{CS2}$	RD	WR	Memory function	State of data bus
0	0	×	×	Inhibit	High-impedance
0	1	×	×	Inhibit	High-impedance
1	0	0	0	Inhibit	High-impedance
1	0	0	1	Write	Input data to RAM
1	0	1	×	Read	Output data from RAM
1	1	×	×	Inhibit	High-impedance

Figure 3: Function Table of RAM

ROM CHIPS:

ROM chip is organized externally in a similar manner. However, since a ROM can only read, the data bus can only be in an output mode. The block diagram of a ROM chip is shown in Figure 4. For the same-size chip, it is possible to have more bits of ROM than of RAM, because the internal binary cells in ROM occupy less space than in RAM. For this reason, the diagram specifies a 512-byte ROM, while the RAM has only 128 bytes. The nine address lines in the ROM chip specify any one of the 512 bytes stored in it. The two chip select inputs must be CS1 = 1 and CS2 = 0 for the unit to operate. Otherwise, the data bus is in a high-impedance state. There is no need for a read or write control because the unit can only read. Thus when the chip is enabled by the two select inputs, the byte selected by the address lines appears on the data bus.

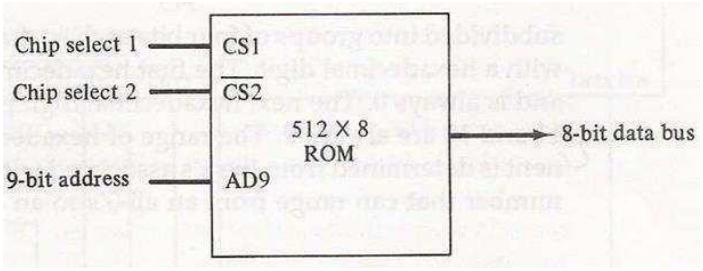


Figure 4: Rom Chip

Secondary Memory –The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. Other components used, but not as frequently, are magnetic drums, magnetic bubble memory, and optical disks.

The recording surface rotates at uniform speed and is not startedor stopped during access operations. Bits are recorded as magnetic spots on the surface as it passes a stationary mechanism called a write head. Stored bits are detected by a change in magnetic field produced by a recorded spot on the surface as it passes through a read head. The amount of surface available for recording in a disk is greater than in a drum of equal physical size. Therefore, more information can be stored on a disk than on a drum of comparable size. For this reason, disks have replaced drums in more recent computers.

Magnetic Disks:

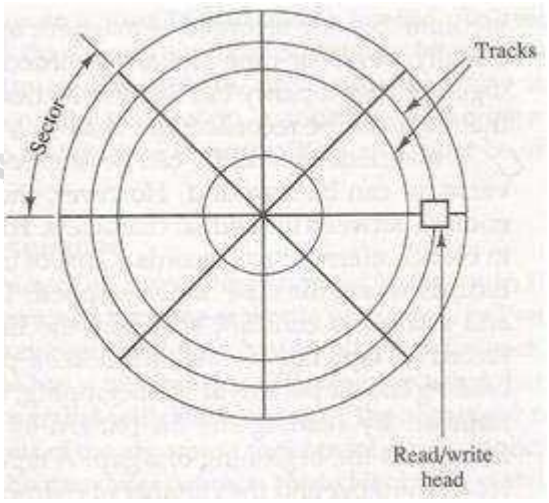


Figure 5: Magnetic Disk

- A magnetic disk is a circular plate constructed of metal or plastic coated with magnetized material.
- Often both sides of the disk are used and several disks may be stacked on one spindle with read/write heads available on each surface.
- All disks rotate together at high speed and are not stopped or started for access purposes.
- Bits are stored in the magnetized surface in spots along concentric circles called tracks.
- The tracks are commonly dividedinto sections called sectors.
- In most systems, the minimum quantity of information which can be transferred is a sector.
- The subdivision of one disk surface into tracks and sectors is shown in Figure5.
- There are two sizes commonly used, with diameters of 5.25 and 3.5 inches. The 3.5-inch disks are smaller and can store more data than can the 5.25-inch disks.

Magnetic Tape:

- A magnetic tape transport consists of the electrical, mechanical, and electronic components to provide the parts and control mechanism for a magnetic-tape unit.
- The tape itself is a strip of plastic coated with a magnetic recording medium.

- Bits are recorded as magnetic spots on the tape along several tracks.
- Usually, seven or nine bits are recorded simultaneously to form a character together with a parity bit.
- Read/write heads are mounted one in each track so that data can be recorded and read as a sequence of characters.
- Magnetic tape units can be stopped, started to move forward or in reverse, or can be rewound.
- Gaps of unrecorded tape are inserted between records where the tape can be stopped.
- The tape starts moving while in a gap and attains its constant speed by the time it reaches the next record.
- Each record on tape has an identification bit pattern at the beginning and end.
- A tape unit is addressed by specifying the record number and the number of characters in the record. Records may be of fixed or variable length.

Optical Storage:

Optical storage devices use optical technology to save and retrieve data on discs, like a Blu-ray, CD, DVD. The device uses a laser light to read information on the disc and to "write" new information to the disc for future retrieval.

CACHE MEMORY: CACHE STRUCTURE AND DESIGN

A special very-high-speed memory called a cache memory is sometimes used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate.

The basic operation of the cache is as follows:

- When the CPU needs to access memory, the cache is examined. If the word is found in the cache, it is called cache hit. If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word and it is called cache miss.
- Some data are transferred to cache so that future references to memory find the required words in the fast cache memory.
- The performance of cache memory is frequently measured in terms of a quantitycalled *hit ratio.*
- The ratio of the number of hits divided by the total CPU references to memory (hits plus misses) is the hit ratio.

The transformation of data from main memory to cache memory is referred to as a *mapping* process.

Three types of mapping procedures are of practical interest when considering the organization of cache memory:

- Associative mapping
- Direct mapping
- Set- associative mapping

To help in the discussion of these three mapping procedures we will use a specific example of a memory organization as shown in Figure 6. The main memory can store 32K words of 12 bits each. The cache is capable of storing 512 of these words at any given time. For every word stored in cache, there is a duplicate copy in main memory. The CPU communicates with both memories. It first sends a 15-bit address to cache. If there is a hit, the CPU accepts the 12-bit data from cache. If there is a miss, the CPU reads the word from main memory and the word is then transferred to cache.

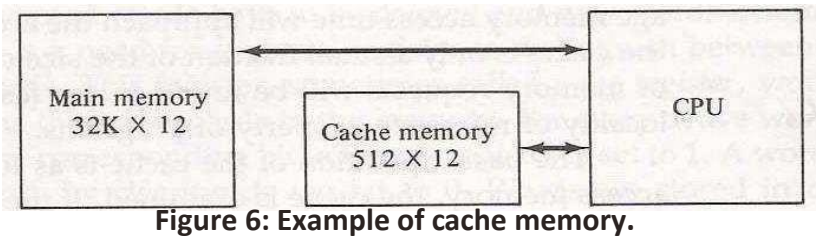


Figure 6: Example of cache memory.

MAPPING SCHEME:

1. Associative Mapping:

- The fastest and most flexible cache organization uses an associative memory.
- The associative memory stores both the address and content (data) of the memory word.
- This permits any location in cache to store any word from main memory.

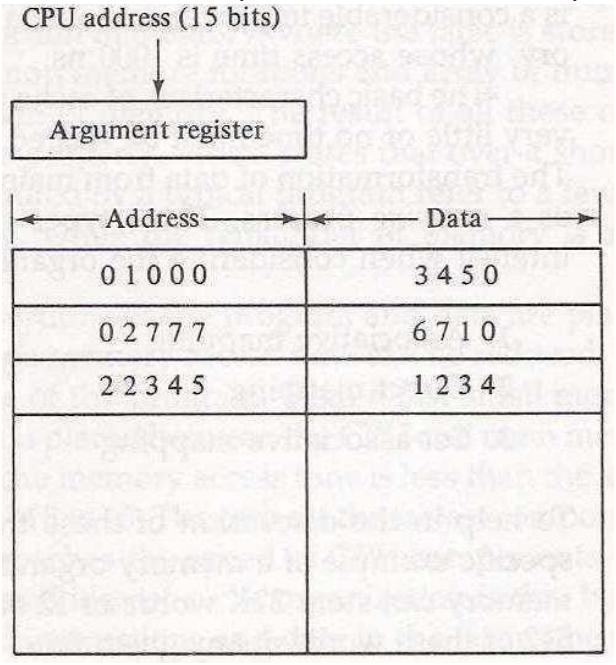


Figure 7: Associative mapping cache (all numbers in octal).

- The diagram shows three words presently stored in the cache.
- The address value of 15 bits is shown as a five-digit octal number and its corresponding 12-bit word is shown as a four-digit octal number.
- A CPU address of 15 bits is placed in the argument register and the associative memory is searched for a matching address.
- If the address is found, the corresponding 12 -bit data is read and sent to the CPU. If no match occurs, the main memory is accessed for the word.
- The address data pair is then transferred to the associative cache memory.
- If the cache is full, an address data pair must be displaced to make room for a pair that is needed and not present in the cache.
- The decision as to what pair is replaced is determined from the replacement algorithm that the designer chooses for the cache.

2. Direct Mapping:

- Associative memories are expensive compared to random-access memories because of the added logic associated with each cell.
- The CPU address of 15 bits is divided into two fields. The nine least significant bits constitute the indexfield and the remaining six bits forms the tagfield.
- The disadvantage of direct mapping is that the hit ratio can drop considerably if two or more words whose addresses have the same index but different tags are accessed repeatedly.

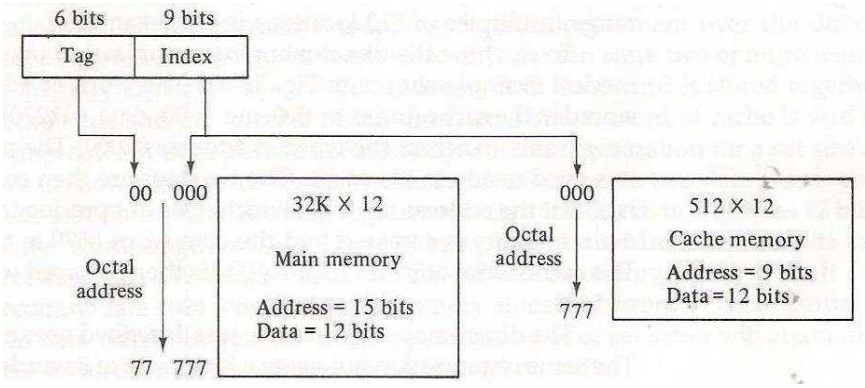
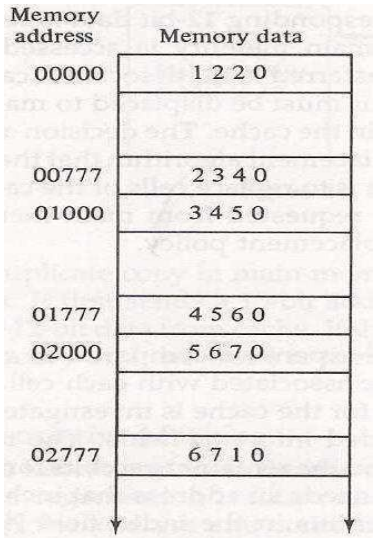
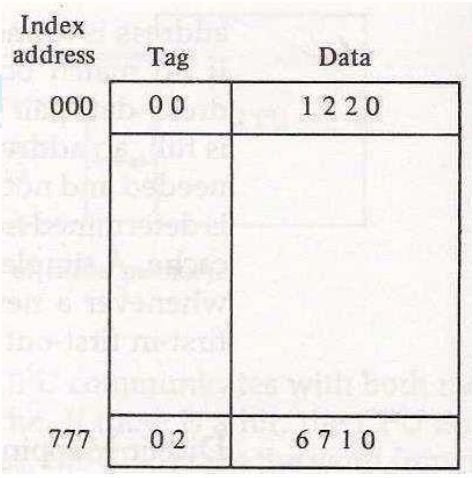


Figure 8: Addressing relationships between main and cache memories.

Consider a numerical example.

- The word at address zero is presently stored in the cache (index = 000, tag = 00, data = 1220).
- Suppose that the CPU now wants to access the word at address 02000.
- The index address is 000, so it is used to access the cache. The two tags are then compared.
- The cache tag is 00 but the address tag is 02, which does not produce a match.
- Therefore, the main memory is accessed and the data word 5670 is transferred to the CPU.
- The cache word at index address 000 is then replaced with a tag of 02 and data of 5670.



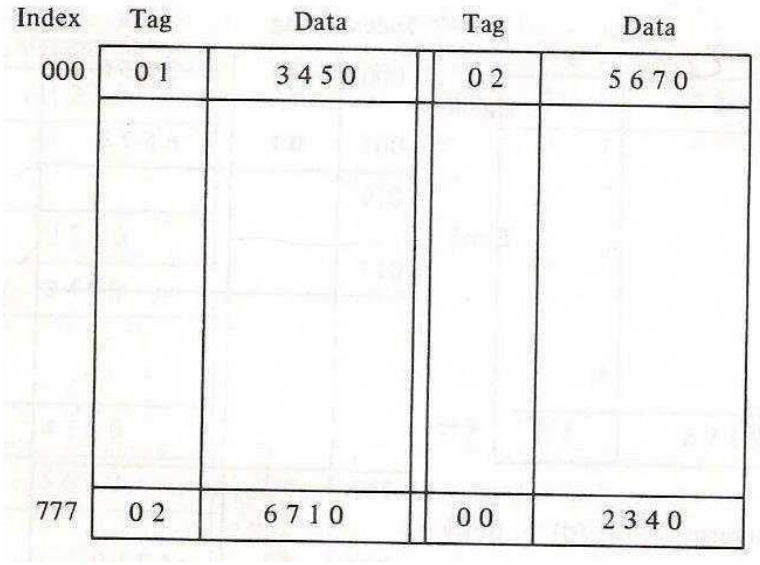


(a) Main memory **(b) Cache memory**

Figure 9: Direct mapping cache organization.

3. Set-Associative Mapping:

- It was mentioned previously that the disadvantage of direct mapping is that, two words with the same index but different tag values cannot reside in cache memory at the same time.
- A third type of cache organization, called set associative mapping, is an improvement over the direct-mapping organization, in that each word of cache can store two or more words of memory under the same index address.
- Each data word is stored together with its tag and the number of tag data items in one word of cache is said to form a set.



Index	Tag	Data	Tag	Data
000	01	3450	02	5670
777	02	6710	00	2340

Figure 10: Two-way set -associative mapping cache.

The octal numbers listed in Figure 10 are with reference to the main memorycontents.

- The words stored at addresses 01000 and 02000 of main memory are stored in cache memory at index address 000.
- Similarly, the words at addresses 02777 and 00777 are stored in cache at index address 777.
- When the CPU generates a memory request, the index value of the address is used to access the cache.
- The tag field of the CPU address is then compared with both tags in the cache to determine if a match occurs.
- The comparison logic is done by an associative search of the tags in the set similar to an associative memory search: thus the name "set-associative."

REPLACEMENT ALGORITHM:

Avirtual memory system is a combination of hardware and software techniques. The memory management software system handles all the software operations for the efficient utilization of memory space.

It must decide:

- Which page in main memory ought to be removed to make room for a new page.
- When a new page is to be transferred from auxiliary memory to main memory.
- Where the page is to be placed in main memory.

The hardware mapping mechanism and the memory management software together constitute the architecture of a virtual memory. When a program starts execution, one or more pages are transferred into main memory and the page table is set to indicate their position. The program is executed from main memory until it attempts to reference a page that is still in auxiliary memory. This condition is called*page fault.* When page fault occurs, the execution of the present program is suspended until the required page is brought into main memory.

Three of the most common replacement algorithms used is the **First-In-First – Out (FIFO), Least Recently Used(LRU)** and the **Optimal Page Replacement Algorithms.**

FIFO:

- The FIFO algorithm selects for replacement the page that has been in memory the longest time.
- Each time a page is loaded into memory, its identification number is pushed into a FIFO stack.

- FIFO will be full whenever memory has no more empty blocks.
- When a new page must be loaded, the page least recently brought in is removed.
- The FIFO replacement policy has the advantage of being easy to implement. It has the disadvantage that under certain circumstances pages are removed and loaded from memory too frequently.

LRU:

- The LRU policy is more difficult to implement but has been more attractive on the assumption that the least recently used page is a better candidate for removal than the least recently loaded page as in FIFO.
- The LRU algorithm can be implemented by associating a counter with every page that isin main memory.
- When a page is referenced, its associated counter is set to zero. At fixed intervals of time, the counters associated with all pages presently in memory are incremented by 1.
- The least recently used page is the page with the highest count. The counters are often called *aging registers,* as their count indicates their age, that is, how long ago their associated pages have been referenced.

Optimal Page Replacement Algorithms

- The optimal page algorithm simply says that the page with the highest label should be removed.
- If one page will not be used for 8 million instructions and another page will not be used for 6 million instructions, removing the former pushes the page fault that will fetch it back as far into the future as possible.

IMPROVING CACHE PERFORMANCE:

There are three ways to improve cache performance:

1. Reduce the miss rate.
2. Reduce the miss penalty.
3. Reduce the time to hit in the cache.

VIRTUAL MEMORY

- In a memory hierarchy system, programs and data are first stored in auxiliary memory.
- Portions of a program or data are brought into main memory as they are needed by the CPU.
- *Virtual memory* is a concept used in some large computer systems that permit the user to construct programs as though a large memory space were available, equal to the totality of auxiliary memory.
- Each address that is referenced by the CPU goes through an address mapping from the so-called virtual address to a physical address in main memory.
- Virtual memory is used to give programmers the illusion that they have a very large memory at their disposal, even though the computer actually has a relatively small main memory.

Address Space and Memory Space

- An address used by a programmer will be called a virtual address, and the set of such addresses is the address space.
- An address in main memory is called a location or physical address. The set of such locations is called the memory space.
- Thus the address space is the set of addresses generated by programs as they reference instructions and data, the memory space consists of the actual main memory locations directly addressable for processing.
- In most computers the address and memory spaces are identical.
- The address space is allowed to be larger than the memory space in computers with virtual memory.

- In a multi-program computer system, programs and data are transferred to and from auxiliary memory and main memory based on demands imposed by the CPU.

Suppose that program 1 is currently being executed in the CPU. Program 1 and a portion of its associated data are moved from auxiliary memory into main memory as shown in Figure 11. Portions of programs and data neednot be in contiguous locations in memory since information is being moved in and out, and empty spaces may be available in scattered locations in memory.

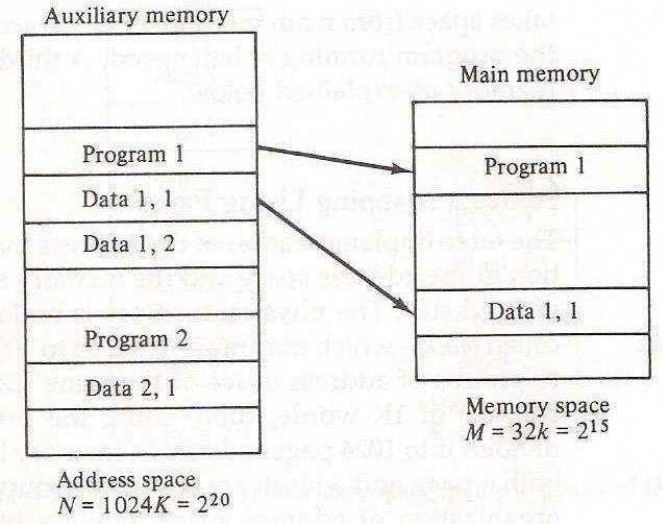


Figure 11: Relation between address and memory space in a virtual memory system.

Address Mapping Using Pages:

- The table implementation of the address mapping is simplified if the information in the address space and the memory space are each divided into groups of fixed size.
- The physical memory is broken down into groups of equal size called *blocks,* which may range from 64 to 4096 words each. The term pagerefers to groups of address space of the same size.

For example, if a page or block consists of 1K words, then, address space is divided into 1024 pages and main memory is divided into 32 blocks. Although both a page and a block are split into groups of 1K words, a page refers to the organization of address space, while a block refers to the organization of memory space. The programs are also considered to be split into pages. Portions of programs are moved from auxiliary memory to main memory in records equal to the size of a page. The term "page frame" is sometimes used to denote a block. Consider a computer with an address space of 8K and a memory space of 4K.

- The mapping from address space to memory space is facilitated if each virtual address is considered to be represented by two numbers: a page number address and a line within the page.
- In a computer with 2 words per page, *p* bits are used to specify a line address and the remaining high-order bits of the virtual address specify the page number.

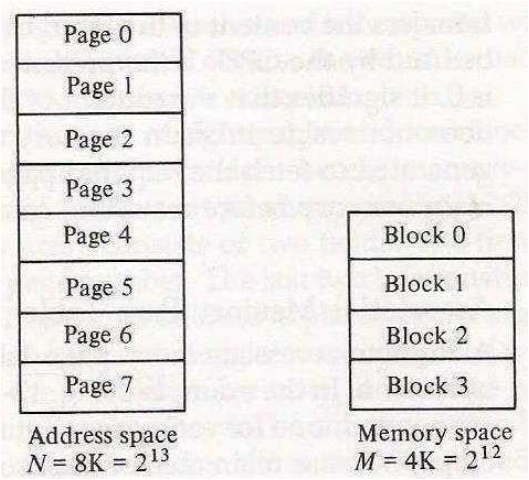


Figure 12:Address space and memory space split into groups of 1K words.

MEMORY MANAGEMENT HARDWARE:

- A memory management system is a collection of hardware and software procedures for managing the various programs residing in memory.
- The memory management software is part of an overall operating system available in many computers.

The basic components of a memory management unit are:

- A facility for dynamic storage relocation that maps logical memory references into physical memory addresses
- A provision for sharing common programs stored in memory by different users.
- Protection of information against unauthorized access between users and preventing users from changing operating system functions

The dynamic storage relocation hardware is a mapping process similar to the paging system. The fixed page size used in the virtual memory system causes certain difficulties with respect to program size and the logical structure of programs.

- It is more convenient to divide programs and data into logical parts called segments.
- A segment is a set of logically related instructions or data elements associated with a given name.
- Segments may be generated by the programmer or by the operating system.
- Examples of segments are a subroutine, an array of data, a table of symbols, or a user's program.
- The address generated by a segmented program is called a logical address.
- This is similar to a virtual address except that logical address space is associated with variable-length segments rather than fixed-length pages.

Segmented-Page Mapping

- The property of logical space is that it uses variable-length segments.
- The length of each segment is allowed to grow and contractaccording to the needs of the program being executed.
- One way of specifying the length of a segment is by associating with it a number of equal-size pages.

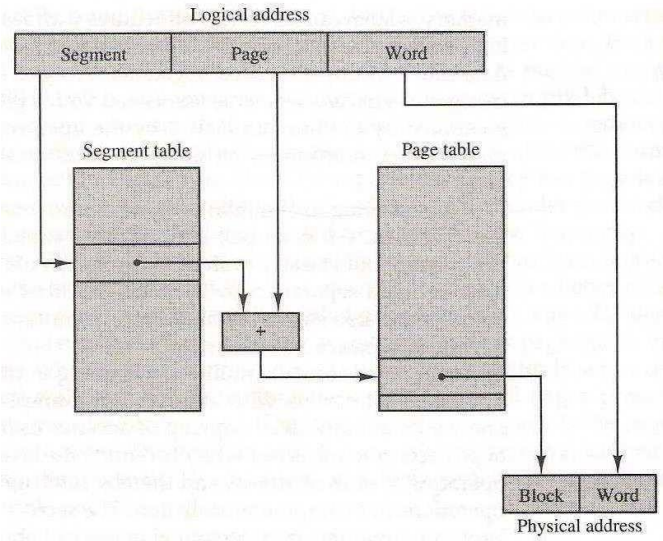


Figure 13: Logical to physical address mapping

- The logical address is partitioned into three fields: the segment field specifies a segment number, the page field specifies the page within the segment and the word field gives the specific word within the page.
- A page field of k bits can specify up to 2^k pages.
- A segment number may be associated with just one page or with as many as2^k pages.
- Thus the length of a segment would vary according to the number of pages that are assigned to it.
- The mapping of the logical address into a physical address is done by means of two tables, as shown in figure 13.

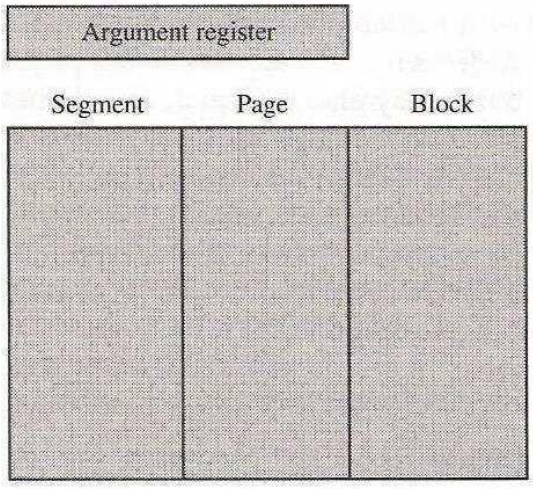


Figure 14: Associate memory translation look aside buffer TLB

The mapping of the logical address into a physical address is done by means of two tables, as shown in figure 13. The segment number of the logical address specifies the address for the segment table.

- The entry in the segment table is a pointer address for a page table base.
- The page table base is added to the page number given in the logical address.
- The sum produces a pointer address to an entry in the page table.
- The value, found in the page table pro vides the block number in physical memory.
- Theconcatenation of the block field with the word field produces the final physical mapped address.

The two mapping tables may be stored in two separate small memories or in main memory. In either case, a memory reference from the CPU will require three accesses to main memory: one from the segment table, one from the page table, and the third front main memory.

Unit 6
Multiprocessors

Characteristics of Multiprocessor, Structure of Multiprocessor- Inter-processor Arbitration, Inter-Processor Communication and Synchronization. Memory in Multiprocessor System, Concept of Pipelining, Vector Processing, Array Processing, RISC and CISC, Study of Multicore Processor – Intel, AMD.

Multiprocessor:

Multiprocessor system is the use of two or more central processing units (CPUs) within a single computer system. The term also refers to the ability of a system to support more than one processor or the ability to allocate tasks between them. There are many variations on this basic theme, and the definition of multiprocessing can vary with context, mostly as a function of how CPUs are defined (multiple cores on one die, multiple dies in one package, multiple packages in one system unit, etc.).

According to some on-line dictionaries, a multiprocessor is a computer system having two or more processing units (multiple processors) each sharing main memory and peripherals, in order to simultaneously process programs.

Characteristics of Multiprocessor:

- A Multiprocessor system implies the existence of multiple CPUs, although usually there will be one or more IOPs as well.
- A Multiprocessor system is controlled by one operating system that provides interaction between processors and all the components of the system cooperate in the solution of a problem.
- Microprocessors take very little physical space and are very inexpensive brings about the feasibility of interconnecting a large number of microprocessors into one computer system.

Advantages of Multiprocessor Systems are:

- Increased Reliability because of redundancy in processors.
- Increased throughput because of execution.
- Multiple jobs in parallel.
- Portions of the same job in parallel.

A single job can be divided into independent tasks, either manually by the programmer, or by the compiler, which finds the portions of the program that are data independent, and can be executed in parallel. The multiprocessors are further classified into two groups depending on the way their memory is organized. The processors with shared memory are called tightly coupled or shared memory processors. The information in these processors is shared through the common memory. Each of the processors can also have their local memory too. The other class of multiprocessors is loosely coupled or distributed memory multi-processors. In this, each processor has their own private memory, and they share information with each other through interconnection switching scheme or message passing.

The principal characteristic of a multiprocessor is its ability to share a set of main memory and some I/O devices. This sharing is possible through some physical connections between them called the interconnection structures.

Structure of Multiprocessor:

Time Shared Common Bus:

A system common bus multiprocessor system consists of a number of processors connected through path to a memory unit.

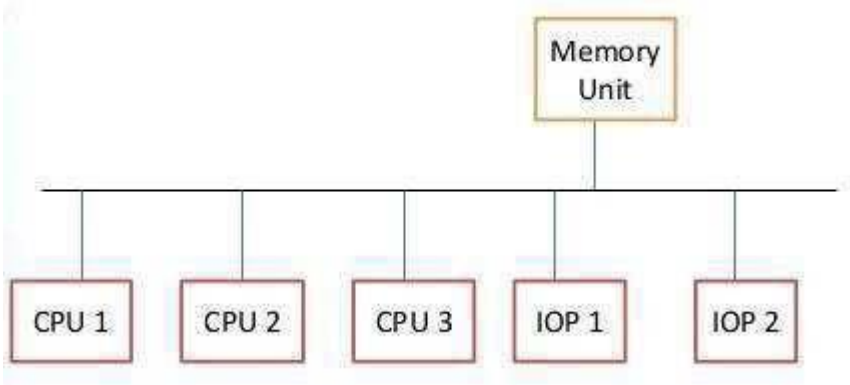


Figure 1: Time Share Common Bus

Hierarchical Bus Systems:

A hierarchical bus system consists of a hierarchy of buses connecting various systems and sub-systems/components in a computer. Each bus is made up of a number of signal, control, and power lines. Different buses like local buses, backplane buses and I/O buses are used to perform different interconnection functions.

Local buses are the buses implemented on the printed-circuit boards. A backplane bus is a printed circuit on which many connectors are used to plug in functional boards. Buses which connect input/output devices to a computer system are known as I/O buses.

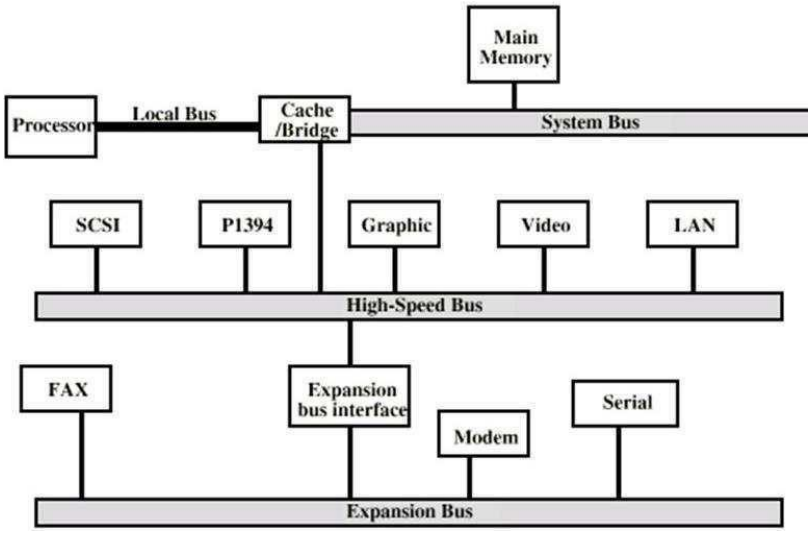


Figure 2: Hierarchical Bus System

Crossbar switch and Multiport Memory:

Switched networks give dynamic interconnections among the inputs and outputs. Small or medium size systems mostly use crossbar networks. Multistage networks can be expanded to the larger systems, if the increased latency problem can be solved.

Both crossbar switch and multiport memory organization is a single-stage network. Though a single stage network is cheaper to build, but multiple passes may be needed to establish certain connections. A multistage network has more than one stage of switch boxes. These networks should be able to connect any input to any output.

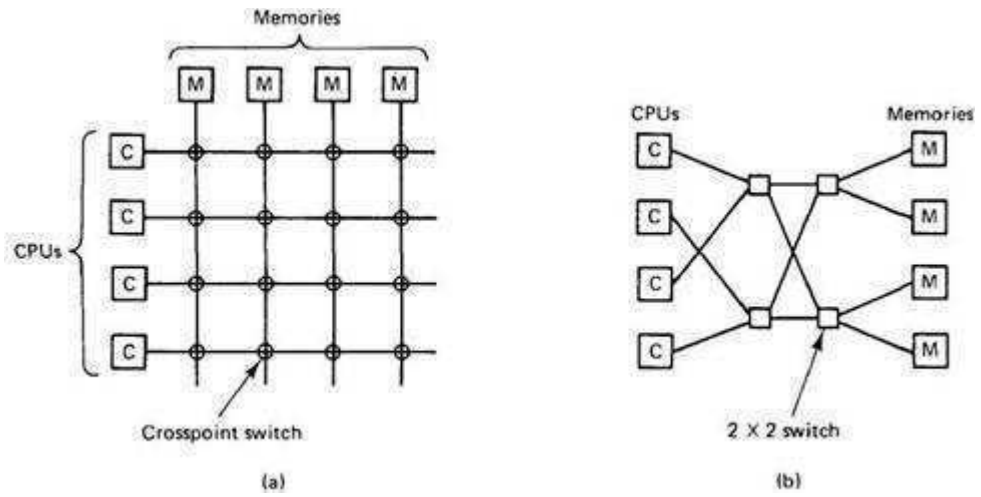


Figure 3: Cross Bar Switch

Multistage and Combining Networks:

Multistage networks or multistage interconnection networks are a class of high-speed computer networks which is mainly composed of processing elements on one end of the network and memory elements on the other end, connected by switching elements.

These networks are applied to build larger multiprocessor systems. This includes Omega Network, Butterfly Network and many more.

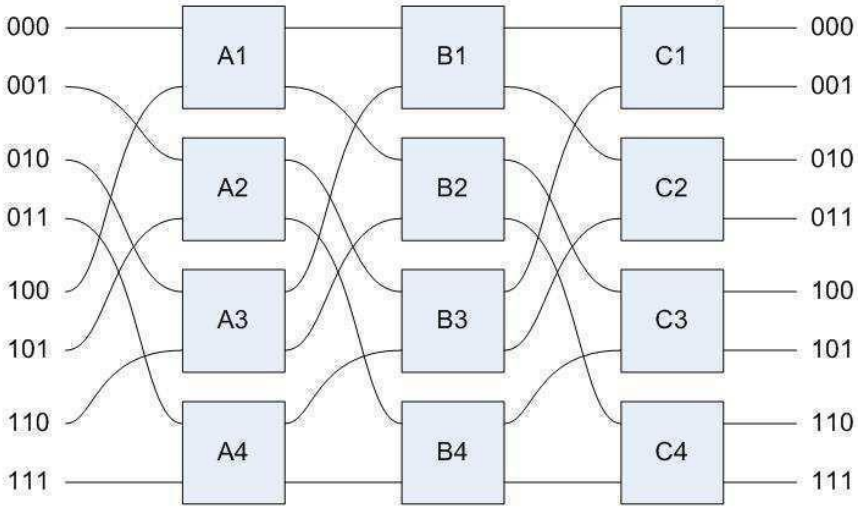


Figure 4: Multistage and Combining Networks

Multi-computers:

Multi-computers are distributed memory MIMD architectures. The following diagram shows a conceptual Multicomputer.

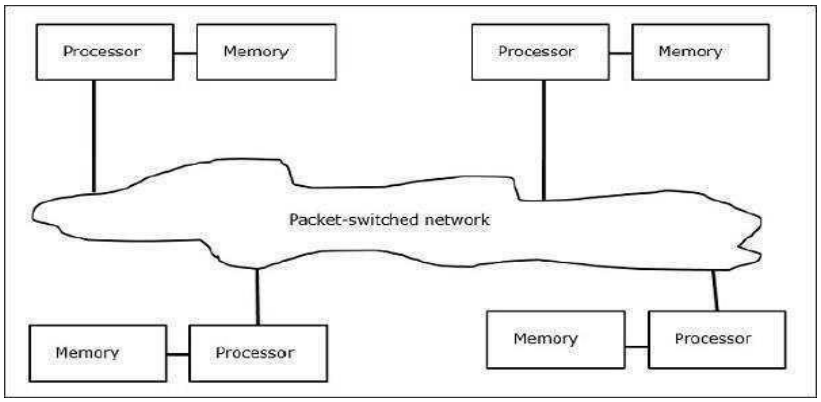


Figure 5: Multi-Computers Interconnection

Multi-computers are message-passing machines which apply packet switching method to exchange data. Here, each processor has a private memory, but no global address space as a processor can access only its own local memory. So, communication is not transparent: here programmers have to explicitly put communication primitives in their code.

Having no globally accessible memory is a drawback of multi-computers. This can be solved by using the following two schemes −

Virtual Shared Memory (VSM)

Shared Virtual Memory (SVM)

In these schemes, the application programmer assumes a big shared memory which is globally addressable. If required, the memory references made by applications are translated into the message-passing paradigm.

Virtual Shared Memory (VSM)

VSM is a hardware implementation. So, the virtual memory system of the Operating System is transparently implemented on top of VSM. So, the operating system thinks it is running on a machine with a shared memory.

Shared Virtual Memory (SVM)

SVM is a software implementation at the Operating System level with hardware support from the Memory Management Unit (MMU) of the processor. Here, the unit of sharing is Operating System memory pages.

If a processor addresses a particular memory location, the MMU determines whether the memory page associated with the memory access is in the local memory or not. If the page is not in the memory, in a normal computer system it is swapped in from the disk by the Operating System. But, in SVM, the Operating System fetches the page from the remote node which owns that particular page.

Inter-processor Arbitration:

Computer system needs buses to facilitate the transfer of information between its various components. For example, even in a uni-processor system, if the CPU has to access a memory location, it sends the address of the memory location on the address bus. This address activates a memory chip. The CPU then sends a red signal through the control bus, in the response of which the memory puts the data on the address bus. This address activates a memory chip. The CPU then sends a read signal through the control bus, in the response of which the memory puts the data on the data bus. Similarly, in a multiprocessor system, if any processor has to read a memory location from the shared areas, it follows the similar routine.

There are buses that transfer data between the CPUs and memory. These are called memory buses. An I/O bus is used to transfer data to and from input and output devices. A bus that connects major components in a multiprocessor system, such as CPUs, I/Os, and memory is called system bus. A processor, in a multiprocessor system, requests the access of a component through the system bus. In case there is no processor accessing the bus at that time, it is given then control of the bus immediately. If there is a second processor utilizing the bus, then this processor has to wait for the bus to be freed. If at any time, there is request for the services of the bus by more than one processor, then the arbitration is performed to resolve the conflict. A bus controller is placed between the local bus and the system bus to handle this.

Inter-Processor Communication and Synchronization:

In computer science, inter-process communication or inter process communication (IPC) refers specifically to the mechanisms an operating system provides to allow the processes to manage shared data. Typically, applications can use IPC, categorized as clients and servers, where the client requests data and the server responds to client requests. Many applications are both clients and servers, as commonly seen in distributed computing. Methods for doing IPC are divided into categories which vary based on software

requirements, such as performance and modularity requirements, and system circumstances, such as network bandwidth and latency.

In order to cooperate concurrently executing processes must communicate and synchronize. Inter process communication is based on the use of shared variables (variables that can be referenced by more than one process) or message passing.

Process Synchronization:

Process Synchronization means sharing system resources by processes in such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes. Process Synchronization was introduced to handle problems that arose while multiple process executions. Synchronization is often necessary when processes communicate. To make this concept clearer, consider the batch operating system again. A shared buffer is used for communication between the leader process and the executor process. These processes must be synchronized so that, for example, the executor process never attempts to read data from the input if the buffer is empty.

Depending on the solution, an IPC mechanism may provide synchronization or leave it up to processes and threads to communicate amongst themselves (e.g. via shared memory).

While synchronization will include some information (e.g. whether or not the lock is enabled, a count of processes waiting, etc.) it is not primarily an information-passing communication mechanism per se.

Examples of synchronization primitives are:

- Semaphore
- Spinlock
- Barrier
- Mutual exclusion:

Memory in Multiprocessor System:

In computer hardware, shared memory refers to a (typically large) block of random access memory (RAM) that can be accessed by several different central processing units (CPUs) in a multiprocessor computer system.

Shared-Memory Multicomputer:

Three most common shared memory multiprocessors models are −

1. Uniform Memory Access (UMA):

In this model, all the processors share the physical memory uniformly. All the processors have equal access time to all the memory words. Each processor may have a private cache memory. Same rule is followed for peripheral devices.

When all the processors have equal access to all the peripheral devices, the system is called a symmetric multiprocessor. When only one or a few processors can access the peripheral devices, the system is called an asymmetric multiprocessor.

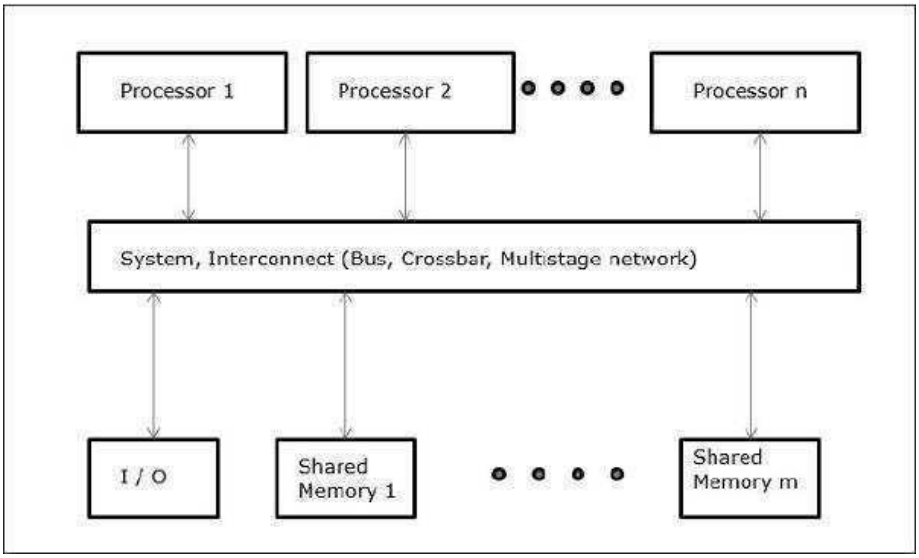


Figure 6: Uniform Memory Access

2. Non-uniform Memory Access (NUMA)

In NUMA multiprocessor model, the access time varies with the location of the memory word. Here, the shared memory is physically distributed among all the processors, called local memories. The collection of all local memories forms a global address space which can be accessed by all the processors.

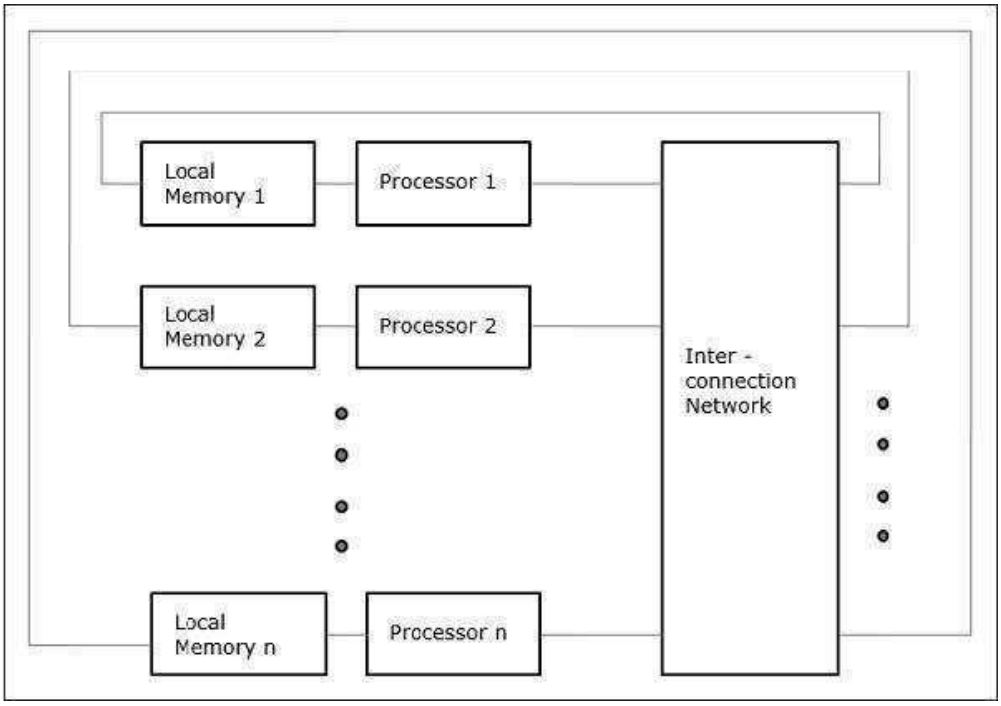


Figure 7: Non-uniform Memory Access

3. Cache Only Memory Architecture (COMA):

The COMA model is a special case of the NUMA model. Here, all the distributed main memories are converted to cache memories.

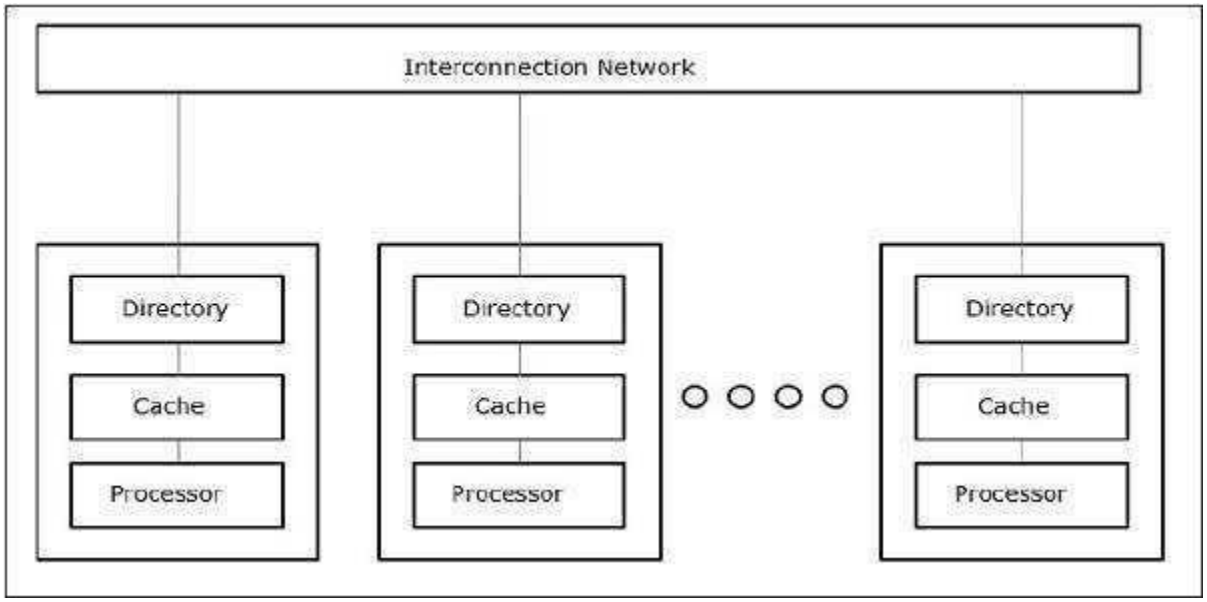


Figure 8: Cache only Memory Architecture

Concept of Pipeline:

- Pipelining is the process of accumulating instruction from the processor through a pipeline. It allows storing and executing instructions in an orderly process. It is also known as pipeline processing.
- Pipelining is a technique where multiple instructions are overlapped during execution. Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end.
- Pipelining increases the overall instruction throughput.
- In pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and combinational circuit performs operations on it. The output of combinational circuit is applied to the input register of the next segment.

Types of Pipeline:

It is divided into 2 categories:

1. Arithmetic Pipeline:

Arithmetic pipelines are usually found in most of the computers. They are used for floating point operations, multiplication of fixed point numbers etc. For example: The input to the Floating Point Adder pipeline is:

X = A*2^a

Y = B*2^b

Here A and B are mantissas (significant digit of floating point numbers), while a and b are exponents.

The floating point addition and subtraction is done in 4 parts:

1. Compare the exponents.
2. Align the mantissas.
3. Add or subtract mantissas
4. Produce the result.

Registers are used for storing the intermediate results between the above operations.

2. Instruction Pipeline:

In this a stream of instructions can be executed by overlapping fetch, decode and execute phases of an instruction cycle. This type of technique is used to increase the throughput of the computer system.

An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline. Thus we can execute multiple instructions simultaneously. The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.

Pipeline Conflicts:

There are some factors that cause the pipeline to deviate its normal performance. Some of these factors are given below:

1. **Timing Variations:**

All stages cannot take same amount of time. This problem generally occurs in instruction processing where different instructions have different operand requirements and thus different processing time.

2. **Data Hazards:**

When several instructions are in partial execution, and if they reference same data then the problem arises. We must ensure that next instruction does not attempt to access data before the current instruction, because this will lead to incorrect results.

3. **Branching:**

In order to fetch and execute the next instruction, we must know what that instruction is. If the present instruction is a conditional branch, and its result will lead us to the next instruction, then the next instruction may not be known until the current one is processed.

4. Interrupts: Interrupts set unwanted instruction into the instruction stream. Interrupts effect the execution of instruction.

5. **Data Dependency:**

It arises when an instruction depends upon the result of a previous instruction but this result is not yet available.

Advantages of Pipelining

1. The cycle time of the processor is reduced.
2. It increases the throughput of the system
3. It makes the system reliable.
4. Disadvantages of Pipelining
5. The design of pipelined processor is complex and costly to manufacture.
6. The instruction latency is more.

Vector Processing:

There is a class of computational problems that are beyond the capabilities of a conventional computer. These problems require vast number of computations on multiple data items that will take a conventional computer (with scalar processor) days or even weeks to complete. Such complex instructions, which operate on multiple data at the same time, requires a better way of instruction execution, which was achieved by Vector processors.

Scalar CPUs can manipulate one or two data items at a time, which is not very efficient. Also, simple instructions like ADD A to B, and store into C are not practically efficient. Addresses are used to point to the memory location where the data to be operated will be found, which leads to added overhead of data lookup. So until the data is found, the CPU would be sitting ideal, which is a big performance issue.

Hence, the concept of **Instruction Pipeline** comes into picture, in which the instruction passes through several sub-units in turn. These sub-units perform various independent functions, for example: the first one decodes the instruction, the second sub-unit fetches the data and the third sub-unit performs the math itself. Therefore, while the data is fetched for one instruction, CPU does not sit idle; it rather works on decoding the next instruction set, ending up working like an assembly line.

Vector processor, not only use Instruction pipeline, but it also pipelines the data, working on multiple data at the same time.

In vector processor a single instruction, can ask for multiple data operations, which saves time, as instruction is decoded once, and then it keeps on operating on different data items.

Applications of Vector Processors:

Computer with vector processing capabilities are in demand in specialized applications. The following are some areas where vector processing is used:

1. Petroleum exploration.
2. Medical diagnosis.
3. Data analysis.
4. Weather forecasting.
5. Aerodynamics and space flight simulations.
6. Image processing.
7. Artificial intelligence.

Array Processors:

Array processors are also known as multiprocessors or vector processors. They perform computations on large arrays of data. Thus, they are used to improve the performance of the computer.

Why use the Array Processor:

- An array processor increases the overall instruction processing speed.
- As most of the Array processors operate asynchronously from the host CPU, hence it improves the overall capacity of the system.
- Array Processors has its own local memory, hence providing extra memory for systems with low memory.

There are basically two types of array processors:

Attached Array Processors:

An attached array processor is a processor which is attached to a general purpose computer and its purpose is to enhance and improve the performance of that computer in numerical computational tasks. It achieves high performance by means of parallel processing with multiple functional units.

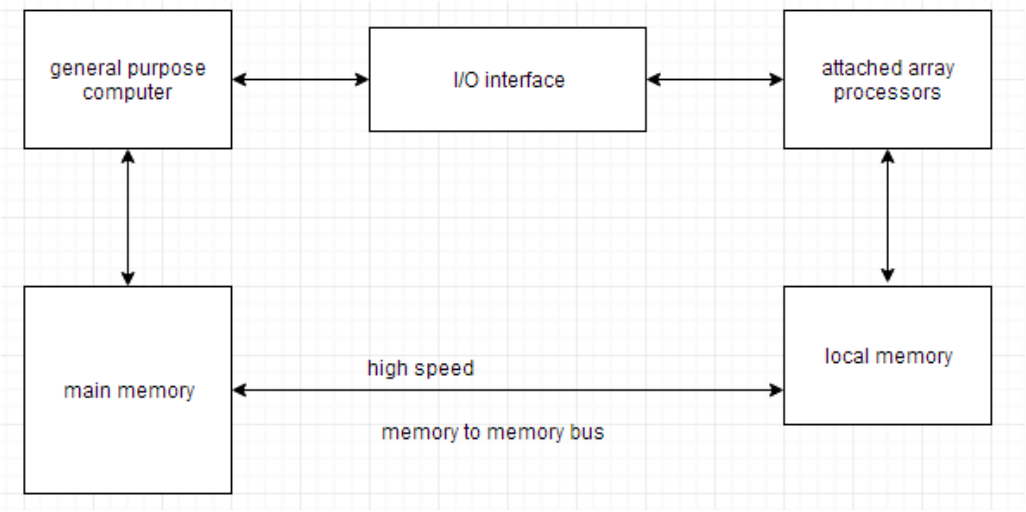


Figure 9: Attached Array Processors

SIMD Array Processors:

SIMD is the organization of a single computer containing multiple processors operating in parallel. The processing units are made to operate under the control of a common control unit, thus providing a single instruction stream and multiple data streams.

A general block diagram of an array processor is shown below. It contains a set of identical processing elements (PE's), each of which is having a local memory M. Each processor element includes an **ALU**and **registers**. The master control unit controls all the operations of the processor elements. It also decodes the instructions and determines how the instruction is to be executed.

The main memory is used for storing the program. The control unit is responsible for fetching the instructions. Vector instructions are send to all PE's simultaneously and results are returned to the memory.

The best known SIMD array processor is the **ILLIAC IV** computer developed by the **Burroughs corps**. SIMD processors are highly specialized computers. They are only suitable for numerical problems that can be expressed in vector or matrix form and they are not suitable for other types of computations.

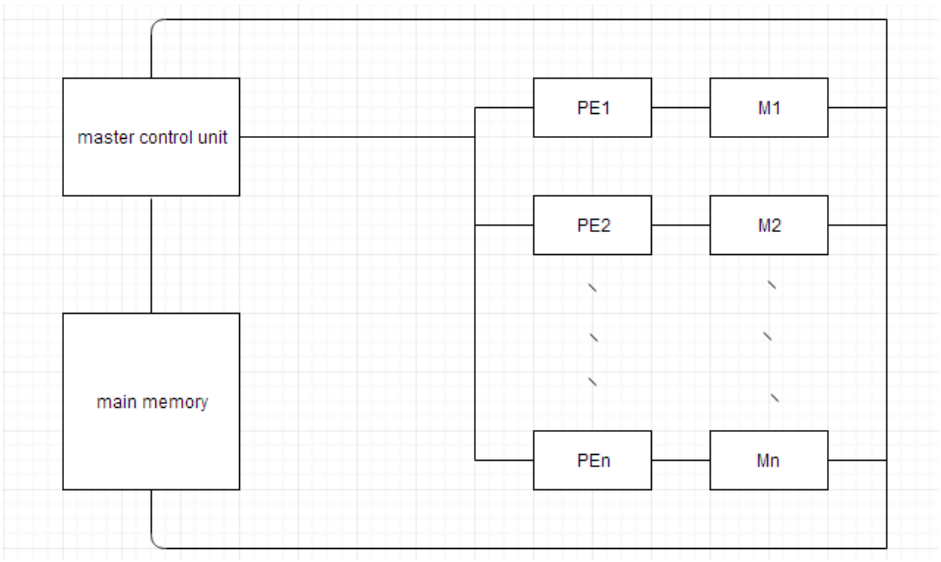


Figure 10: SIMD Array Processors

Type of Multicore Processors:
Ideally, a dual core processor is nearly twice as powerful as a single core processor. In practice, performance gains are said to be about fifty percent: a dual core processor is likely to be about one-and-a-half times as powerful as a single core processor.

Multi-core processing is a growing industry trend as single-core processors rapidly reach the physical limits of possible complexity and speed. Most current systems are multi-core. Systems with a large number of processor cores -- tens or hundreds -- are sometimes referred to as many-core or massively multi-core systems.

Inter-Process Arbitration:
Computer system needs buses to facilitate the transfer of information between its various components. For example, even in a uniprocessor system, if the CPU has to access a memory location, it sends the address of the memory location on the address bus. This address activates a memory chip. The CPU then sends a red signal through the control bus, in the response of which the memory puts the data on the address bus. This address activates a memory chip. The CPU then sends a read signal through the control bus, in the response of which the memory puts the data on the data bus. Similarly, in a multiprocessor system, if any processor has to read a memory location from the shared areas, it follows the similar

routine.

There are buses that transfer data between the CPUs and memory. These are called memory buses. An I/O bus is used to transfer data to and from input and output devices. A bus that connects major components in a multiprocessor system, such as CPUs, I/O's, and memory is called system bus. A processor, in a multiprocessor system, requests the access of a component through the system bus. In case there is no processor accessing the bus at that time, it is given then control of the bus immediately. If there is a second processor utilizing the bus, then this processor has to wait for the bus to be freed. If at any time, there is request for the services of the bus by more than one processor, then the arbitration is performed to resolve the conflict. A bus controller is placed between the local bus and the system bus to handle this.

RISC Processor:

- It is known as Reduced Instruction Set Computer. It is a type of microprocessor that has a limited number of instructions.
- They can execute their instructions very fast because instructions are very small and simple.
- RISC chips require fewer transistors which make them cheaper to design and produce.
- In RISC, the instruction set contains simple and basic instructions from which more complex instruction can be produced.
- Most instructions complete in one cycle, which allows the processor to handle many instructions at the same time.
- In this instructions are register based and data transfer takes place from register to register.

CISC Processor:

- It is known as Complex Instruction Set Computer.
- It was first developed by Intel.
- It contains large number of complex instructions.
- In this instructions are not register based.
- Instructions cannot be completed in one machine cycle.
- Data transfer is from memory to memory.
- Micro programmed control unit is found in CISC.
- Also they have variable instruction formats.

Table 1: Comparison between RISC and CISC Processor

Architectural Characteristics	Complex Instruction Set Computer(CISC)	Reduced Instruction Set Computer(RISC)
Instruction size and format	Large set of instructions with variable formats (16-64 bits per instruction).	Small set of instructions with fixed format (32 bit).
Data transfer	Memory to memory.	Register to register.
CPU control	Most micro coded using control memory (ROM) but modern CISC use hardwired control.	Mostly hardwired without control memory.
Instruction type	Not register based instructions.	Register based instructions.
Memory access	More memory access.	Less memory access.
Clocks	Includes multi-clocks.	Includes single clock.
Instruction Nature	Instructions are complex.	Instructions are reduced and simple.

Study of Multicore Processor – Intel, AMD:

A multi-core processor is a single computing component with two or more independent actual processing units (called "cores"), which are the units that read and execute program instructions. The instructions are ordinary CPU instructions such as add, move data, and branch, but the multiple cores can run multiple instructions at the same time, increasing overall speed for programs amenable to parallel computing

- Intel and AMD have the same design philosophy but different approaches in their micro architecture and implementation.
- AMD technology uses more cores than Intel, but Intel uses Hyper-threading technology to augment the multi-core technology.
- AMD uses Hyper-Transport technology to connect one processor to another and Non-Uniform Memory Access to (NUMA) to access memory. Intel on the other hand usesQuick Path Interconnect technology to connect processor to one another and Memory controller Hub for memory access.
- AMD supports virtualization using Rapid Virtualization Indexing and Direct Connect architecture. While Intel virtualization technology is Virtual Machine Monitor.
- AMD ranked higher in virtualization support than Intel. Moreover, the Quick Path Interconnect in Intel Proliant server have self-healing links and clock failover, hence their technology focuses more on data security while AMD Proliant servers focuses more on power management.

Note: For More Details you may refer:

https://en.wikipedia.org/wiki/Advanced_Micro_Devices

https://www.slideshare.net/anaghvj/intel-core-i7-processors