UNIT – I

BASIC STRUCTURES OF COMPUTER: Functional Units, Multiprocessors and Multicomputers, Memory Locations and Addresses, Memory operations, Instructions and Instruction Sequencing, Addressing modes, Assembly Language, Basic Input/Output operations, Stacks and Queues, Subroutines, Shift and rotate Instructions, Byte-Sorting program.

2 MARKS

1. **What are the functional units?**

   A computer consists of five functionally independent main parts. They are

   1. Input
   2. Memory
   3. arithmetic and logic
   4. output and control units

2. **What is meant by input unit?**

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

3. **What is meant by memory unit?**

   The function of the memory unit is to store programs and data. There are two classes of storage, called primary and secondary.

   Primary storage is a fast memory that operators at electronic speeds. Programs must be stored in the memory while they are being executed.

   Memory in which any location can be reached in a short and a fixed amount of timeafter specifying its address is called randam access memory(RAM) the time required to access one word is called memory access time.

4. **What are the operations in ALU?**

   Arithmetic and logic operations – Multiplication, division, comparison of numbers etc.

   When operands are brought into the processor they are stored in high speed storage elements called Registers. Each Register can store one word of data

5. **What is meant by output unit?**

   The output units is the counter part of the input unit. Its function is to sent processed result to the outside world. Output units – Printer(inkjet printer, Laser printer).Graphics display provide both an input function and input function-I/O Unit.

6. **What are the operations in control unit?**

    The operation of a computer can be summarized as follows,

    1. the computer accepts information in the form of programs and data through an input unit and stores it in the memory
    2. information stored in the memory is fetched, under program control, into an arithmetic and logic unit, where it is processed.
    3. Processed information leaves the computer through an output unit.
    4. All activities inside the machines are directed by the control unit.

7. **Define byte addressability?**

    - Three basic information – bit, byte and word
    - A byte Is always 8 bits. But the word length typically ranges from 16 to 64 bits.
    - Memory location assignments refer to successive byte locations in memory
    - Memory is byte-accessible.
    - For 8086, a Word is 16-bits (2 bytes)

8. **What is meant by multiprocessor?**

    Large computer system may contain a number of processor units, in which case they are called multiprocessor systems.

    These systems either execute a number of different application task in parallel, or they execute sub task of a single large in parallel.

9. **What is meant by multicomputers?**

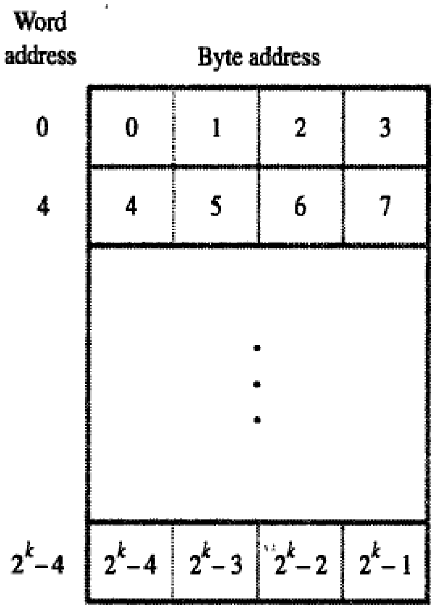
    The high performance of these systems comes with much increased complexity and cost in contrast to multiprocessor systems, it is also possible to use an interconnected group of complete computer to achieve high total computional power.

    When the task they are executing need to communicate data, they do so by exchanging messages over a communication network.

    This property distinguishes them from shared memory multiprocessors, reading to the name message passing multicomputer.

10. What is meant by Big-Endian?

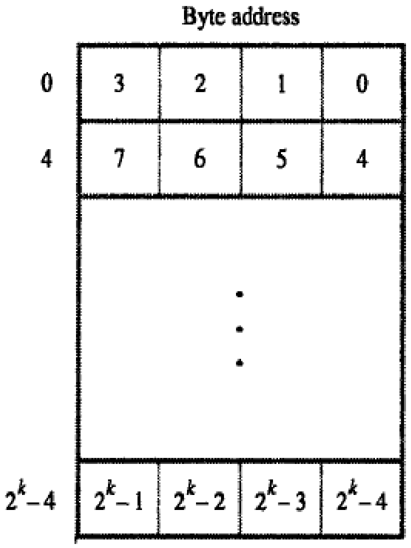
Big-Endian – lower byte addresses are used for the more significant bytes of a word.



(a) Big-endian assignment

11. What is meant by little endian?

Little-Endian – lower byte addresses are used for the less significant bytes of a word



(b) Little-endian assignment

12. How to align the word?

In the case of 32- bit word length, natural word boundaries occur at addresses 0, 4, 8 as shown in the (fig. byte and word addressing). The word locations have aligned addresses. Words are said to be aligned in memory if they begin at a byte address that is a multiple of the number of bytes in a word.

There is no fundamental reason why words cannot begin at an arbitrary byte address. In that case words are said to have unaligned addresses.

13. What are the Ways to indicate the length of the string?

There are two ways to indicate the length of the string.

1. The special control character with the meaning "end of string" can be used as last character in the string.
2. Processor register can contain a number indicating the length of the string in bytes.

14. What are the memory operations?

- To execute an instruction, the processor control circuits must cause the word containing the instruction to be transferred from the memory to the processor
- Two basic operations involving the memory are needed, namely load and store.
- load operation
- store operation

15. Define subroutine?

Subroutine calls are a special type of branch where we return to one instruction below the calling instruction.

Provision must be made to save the return address, since it cannot be written into ROM.

16. What are the basic I/O operations?

- Program controlled IO
- Memory mapped IO

17. Explain stack?

A stack is a list of data elements, usually words or bits, with the accessing restrictions that elements can be added or removed at one end of the list only. This end is called the top of the stack and other end is called the bottom. The structure is sometimes referred to as Push down stack. It uses LIFO principle for two operations-Push and Pop.

18. Explain Queue?

Another useful datastructure that is similar to stack is called the queue. Data are stored and retrieved from a queue on FIFO basics. New data are added at the back and retieved from the front..

19. What is the difference between stack and queue implementation?

There are 2 important differences between how a stack and a queue is implemented.

One end of the is fixed while the other end rises and falls as data are pused and poped. A single pointer is needed to point to top of the stack, at on the other hand both ends of the queue move to higher addresses as data are added at the back and removed from the front.

So two pointers are needed to keep track of the two ends of the queue.

One way to limit the queue to fixed region in a memory is to use a circular buffer.

20. Define Assembly language?

Machine Instructions Are Represented By patterns of 0s and 1s. symbolic names are used to represent patterns. When writing programs for specific computer, such words are normally replaced by acronyms called mnemonics such as MOV,ADD,INC, and BR.R3 is used to refer register 3, and LOC to refer to a location.

A complete set of such symbolic names and rules for their use constitute a programming language, generally referred to as an assembly language.

21. What are the basic operations performed by any computer system?(Nov 2012)

Basic operations of a computer

Inputting

The process of entering the data and instruction in the computer system.2.

Processing

Performing arithmetic operations or logical on data to convert them intouseful information3.

Outputting

The process of producing the useful information or result for the user suchas a printed report or visual display4.

Storing

Storing data and instruction to make them readily available for initial or additional processing whenever required.5.

Controlling

Directing the manner or sequence in which all of the above operation are performed

22. What is syntax of the language?

The set of rules for using the mnemonics in the specification of complete instructions and programs is called the syntax of the language.

23. What are the instructions does the instructions consists of?

A typical assembly language consists of 3 types of instruction statements that are used to define program operations:

- Opcode mnemonics
- Data definitions
- Assembly directives

24. List the various phases of an instruction cycle?(Nov 2012)

Fectch, Decode & Execute.

25. Define Subroutine Nesting.

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

26. Define The Processor Stack.

The stack pointer points to the stack called the processor stack. The call instruction pushes the contents of the PC on to the processor stack and loads the subroutine address into the PC. The return instruction pops the return address from the processor stack into the PC.

27. Define shift and rotate instructions?

There are many applications that require the bits of an operand to be shifted right or left some specified number of bit positions.

The details of how the shifts are performed depend on whether the operand is a signed number or some more general binary coded information. For general operand we use a logical shift , for number we use a arithmetic shift which preserves the sign of the number.

Logical Shift:

Two logical shift instructions are needed , one for shifting left(LShiftL) and another for shifting right (LShiftR).

28. What is meant by subroutine nesting?

Subroutine nesting is to have one subroutine call another. In this case, the return address of the second call is also stored in the link register, destroying its previous contents.

Hence, it is essential to save the contents of the link register in some other location before calling another subroutines. Otherwise, the return address of the first subroutine will be lost.

FUNCTIONAL UNITS

A computer consists of five functionally independent main parts. They are

5. Input
6. Memory
7. arithmetic and logic
8. output and
9. control units

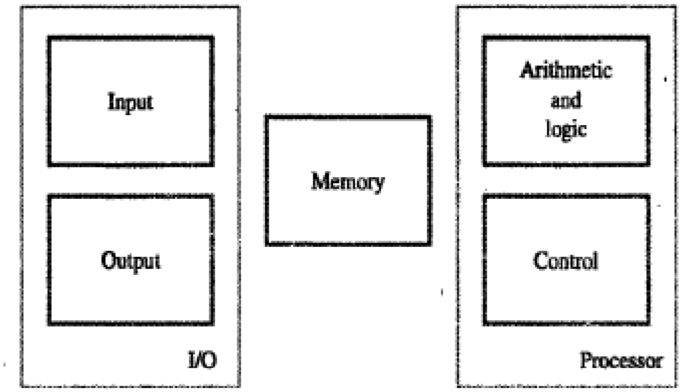


Fig.1.1 Basic functional units of a computer

INPUT UNIT

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

MEMORY UNIT

The function of the memory unit is to store programs and data. There are two classes of storage, called primary and secondary.

Primary storage is a fast memory that operators at electronic speeds. Programs must be stored in the memory while they are being executed. The memory contains the large number of semi-conductor storage cells, each capable of storing one bit of information. These cells are rarely read or written as individual cells but instead are processed in groups of fixed size called words.

To provide easy access to any word in the memory, a distinct address is associated with each word location. Addresses are numbers that identify successive locations. The number of bits in each word is often refer to as the word length of the computer. Typical word length range from 16 to 64 bits.

Memory in which any location can be reached in a short and a fixed amount of timeafter specifying its address is called randam access memory(RAM) the time required to access one word is called memory access time.

The memory of the computer is normally implemented as a memory hierarchy of three or four levels of semi-conductor RAM units with different speeds and sizes. The small, fast RAM units are called Caches. The largest and slowest units is refer to as the main memory.

Primary storage is expensive so in additional secondary storage is used when large amount of data and many programs have to be stored, particularly for information i.e accessed infrequently. Secondary storage is also cheaper. The wide selection of secondary storage device is available, including magnetic disc and tapes and optical disc, CD-ROM.

ARITHMETIC AND LOGIC UNIT

Most computer operations are executed in the arithmetic and logic units(ALU) of the processor.

Arithmetic and logic operations – Multiplication, division, comparison of numbers etc.,

When operands are brought into the processor they are stored in high speed storage elements called Registers. Each Register can store one word of data

OUTPUT UNIT

The output units is the counter part of the input unit. Its function is to sent processed result to the outside world. Output units – Printer(inkjet printer, Laser printer).Graphics display provide both an input function and input function-I/O Unit.

CONTROL UNIT

The control unit is effectively the node center that sends control signals to other units and senses their states. The actual timing signals that coverns the transfer or generated by the control circuit. The operation of a computer can be summarized as follows,

5. the computer accepts information in the form of programs and data through an input unit and stores it in the memory
6. information stored in the memory is fetched, under program control, into an arithmetic and logic unit, where it is processed.
7. Processed information leaves the computer through an output unit.
8. All activities inside the machines are directed by the control unit.

MULTIPROCESSOR AND MULTICOMPUTERS

Large computer system may contain a number of processor units, in which case they are called multiprocessor systems. These systems either execute a number of different application task in parallel, or they execute sub task of a single large in parallel. All processors usually have access to all of the memory in such systems, and the term shared memory multiprocessor systems is often is used to make this clear.

The high performance of these systems comes with much increased complexity and cost in contrast to multiprocessor systems, it is also possible to use an interconnected group of complete computer to achieve high total computionalpower.when the task they are executing need to communicate data, they do so by exchanging messages over a communication network.

This property distinguishes them from shared memory multiprocessors, reading to the name message passing multicomputer.

MEMORY LOCATIONS AND ADDRESSES

Number and character operands, as well as instructions, are stored in the memory of the computer. The memory consists of many millions of storage cells each of which can store a bit of information having the values zero or one. Each group of n bits is refer to as a word of information, and n is called the word length.

Accessing the memory to store or retrieve a single item of information, either a word or a byte, requires distinct names or addresses for each item location. The addresses of successive location in the memory can have values from 0 to 2^k-1.

BYTE ADDRESSBILITY

- Three basic information – bit, byte and word
- A byte Is always 8 bits. But the word length typically ranges from 16 to 64 bits.
- Memory location assignments refer to successive **byte** locations in memory
- Memory is byte-accessible.
- For 8086, a Word is 16-bits (2 bytes)

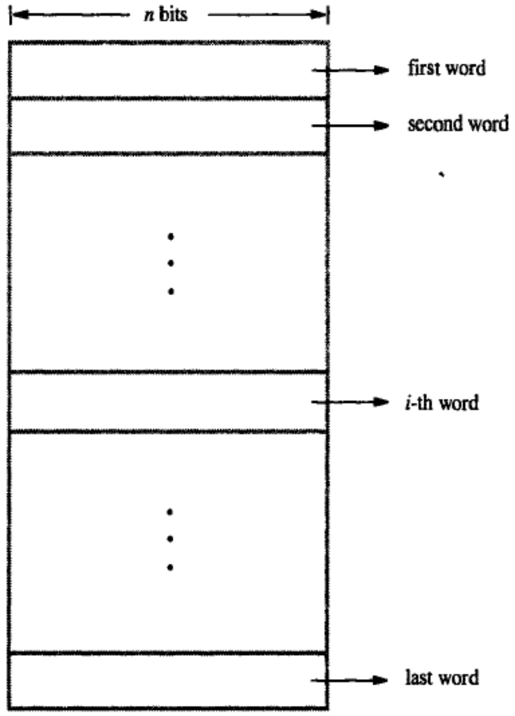
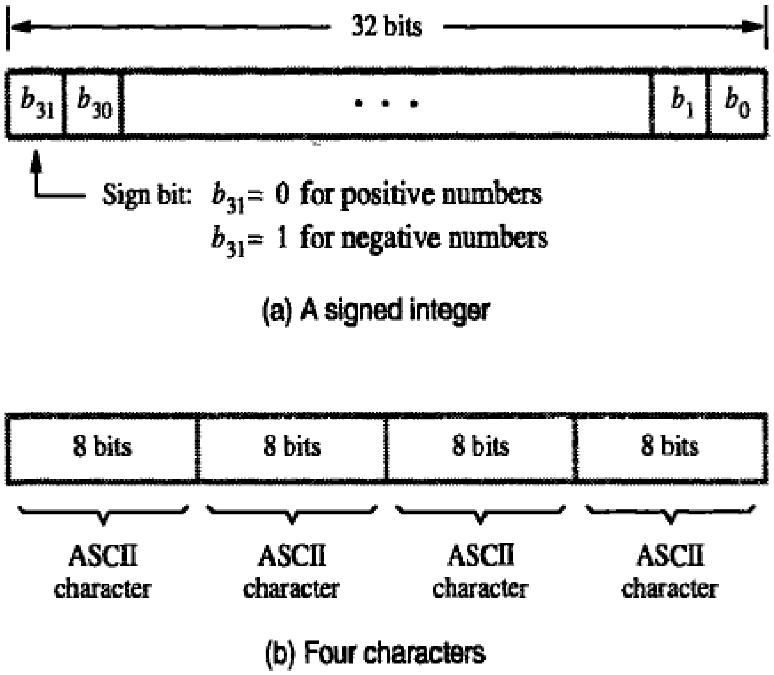


Fig.1.2 Memory words



(a) A signed integer

(b) Four characters

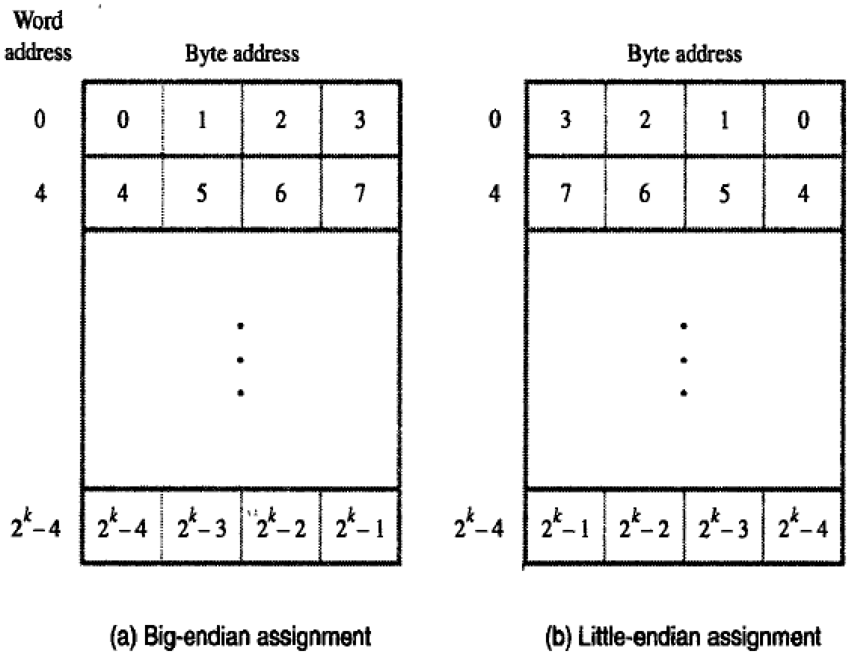
Fig. 1.3 Example of encoded information in 32- bit word

- Byte locations have addresses 0,1, ….

BIG – ENDIAN AND LITTLE – ENDIAN ASSIGNMENTS

There are two ways that byte addresses can be assigned across words.

- Big-Endian – lower byte addresses are used for the more significant bytes of a word
- Little-Endian – lower byte addresses are used for the less significant bytes of a word



(a) Big-endian assignment

(b) Little-endian assignment

Fig.1.4 Byte and word addressing

WORD ALLIGNMENT

In the case of 32- bit word length, natural word boundaries occur at addresses 0, 4, 8,… as shown in the (fig. byte and word addressing). The word locations have aligned addresses. Words are said to be aligned in memory if they begin at a byte address that is a multiple of the number of bytes in a word.

There is no fundamental reason why words cannot begin at an arbitrary byte address. In that case words are said to have unaligned addresses.

ACCESSING NUMBER, CHARACTERS AND CHARACTER STRINGS

A number usually occupies one word. It can be accessed in the memory by specifying its word address. Similarly individual characters can be accessed by a byte address. The beginning of the string is indicate by giving the address of the byte containing its first character. Successive byte locations contain successive characters of the string.

There are two ways to indicate the length of the string.

3. The special control character with the meaning “end of string” can be used as last character in the string.
4. Processor register can contain a number indicating the length of the string in bytes.

MEMORY OPERATIONS

To execute an instruction, the processor control circuits must cause the word containing the instruction to be transferred from the memory to the processor. Operands and results must also be moved between the memory and the processor. Two basic operations involving the memory are needed, namely load and store.

The load operation transfers a copy of the contents of a specific memory location to the processor. The memory contents remain unchanged. To start a load operation, the processor sends the address of the desired location to the memory and requests that its contents be read. The memory reads the data stored at that address and sends them to the processor.

The store operation transfers an item of information from the processor to a specific memory location, destroying the former contents of that location. The processor sends the address of the desired location to the memory, together with the data to be written into that location.

INSTRUCTION AND INSRTUCTION SEQUENCING

The tasks carried out by a computer program consist of a sequence of small steps, such as adding two numbers, testing for a particular condition, reading a character from the keyboard, or sending a character to be displayed on a display screen.

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

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

REGISTER TRANSFER NOTATION

- Identify a location by a symbolic name standing for its hardware binary address (LOC, R0,…)
- Contents of a location are denoted by placing square brackets around the name of the location (R1←[LOC], R3 ←[R1]+[R2])
- Register Transfer Notation (RTN)

ASSEMBLY LANGUAGE NOTATION

- Represent machine instructions and programs.
- Move LOC, R1 = R1←[LOC]
- Add R1, R2, R3 = R3 ←[R1]+[R2]

BASIC INSTRUCTION TYPES

- Three-Address Instructions
  - ADD R1, R2, R3 R1 ← R2 + R3
- Two-Address Instructions
  - ADD R1, R2 R1 ← R1 + R2
- One-Address Instructions
  - ADD M AC ← AC + M[AR]
- Zero-Address Instructions
  - ADD TOS ← TOS + (TOS – 1)
- RISC Instructions
  - Lots of registers. Memory is restricted to Load & Store

Example: Evaluate (A+B) ∗ (C+D)

Three-Address	Two-Address
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	MOV R1, A;R1 ← M[A] ADD R1, B ; R1 ← R1 + 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
One-Address	Zero-Address
LOAD A ; AC ← M[A] ADD B ; AC ← AC + M[B] STORE T ; M[T] ← AC LOAD C ; AC ← M[C] ADD D ; AC ← AC + M[D] MUL T ; AC ← AC ∗ M[T] STORE X ; M[X] ← AC	PUSH A ; TOS ← A PUSH B ; TOS ← B ADD ; TOS ← (A + B) PUSH C ; TOS ← C PUSH D ; TOS ← D ADD ; TOS ← (C + D) MUL ; TOS ← (C+D)∗(A+B) POP X ; M[X] ← TOS
RISC	
LOAD R1, A ; R1 ← M[A] LOAD R2, B ; R2 ← M[B] LOAD R3, C ; R3 ← M[C]	

```
LOAD R4, D   ; R4 ← M[D]
ADD   R1, R1, R2      ; R1 ← R1 + R2
ADD   R3, R3, R4      ; R3 ← R3 + R4
MUL   R1, R1, R3      ; R1 ← R1 ∗ R3
STORE X, R1 ; M[X] ← R1
```

INSTRUCTION EXECUTION AND STRAIGHT-LINE SEQUENCING

The three instruction of the program are in successive word locations starting at location i.

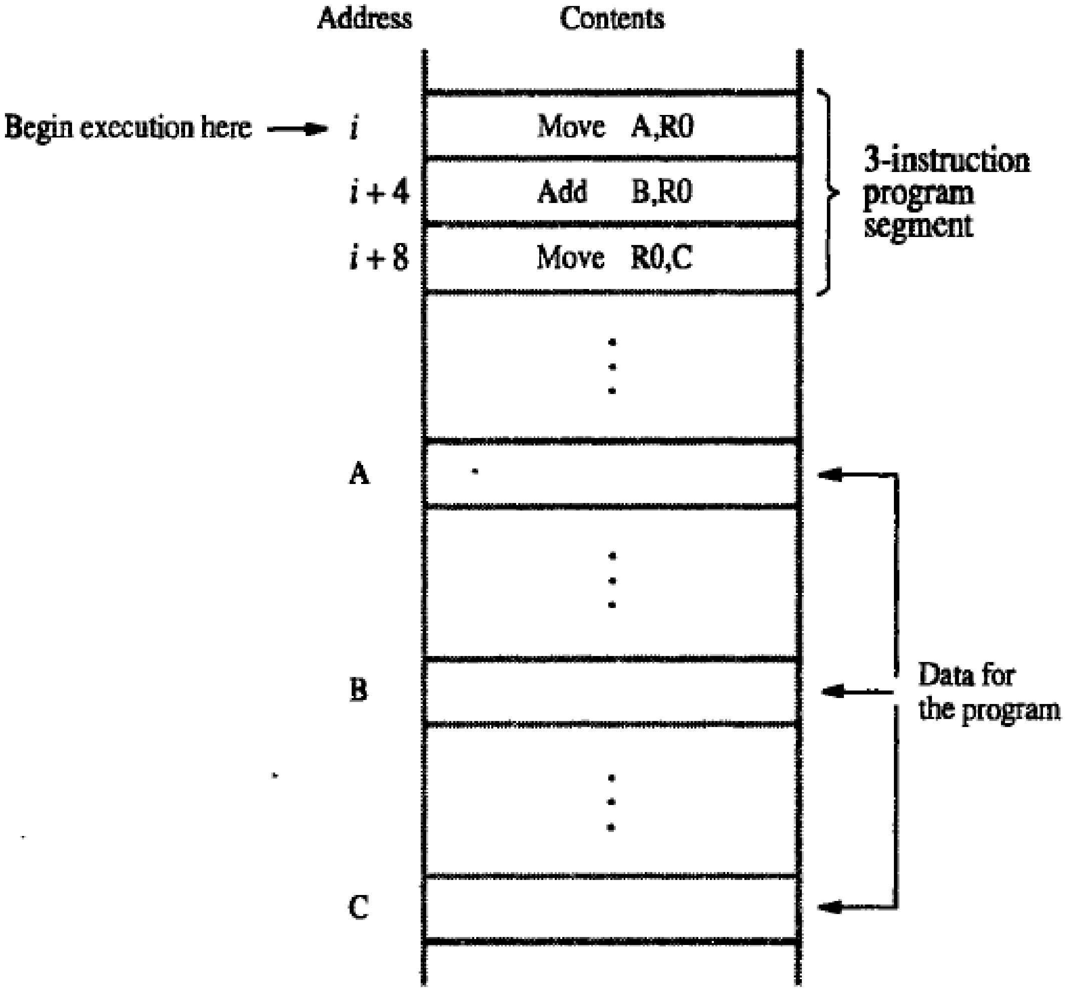


Fig.1.5 A program for C🡐 [a] + [b]

The processor contains a register called the program counter PC, which the address of the instruction to be executed next.

To begin executing a program the address of its first instruction(i) must be placed into the PC.

The processor control circuits use the information in the PC to fetch and execute instruction, one at the time, in order increasing addresses. This is called straight-line sequencing

Executing a given instruction is a two phase procedure. In the first phase, called instruction fetch, the instruction is fetched from the memory location whose address in the PC. This instruction is placed in the instruction Register(IR) in the processor. The second phase called instruction execute, the instruction in IR is examined to determine which operations is to be performed.

The specified operation is performed by the processor. When the execute phase of an instruction Is completed, the PC contains the address of the next instruction, and the new instruction fetch phase can begin.

BRANCHING

- Consider the task of adding a list of n numbers.
- The program outlined in following fig1.6.is a generalization of the program in fig 1.5
- The address of the memory location contain the n numbers are symbolically given as NUM1,NUM2,… NUM n and separate add instruction is used to add each number to the contents of register R0.
- After all the numbers have been added, the result is placed in memory location SUM.

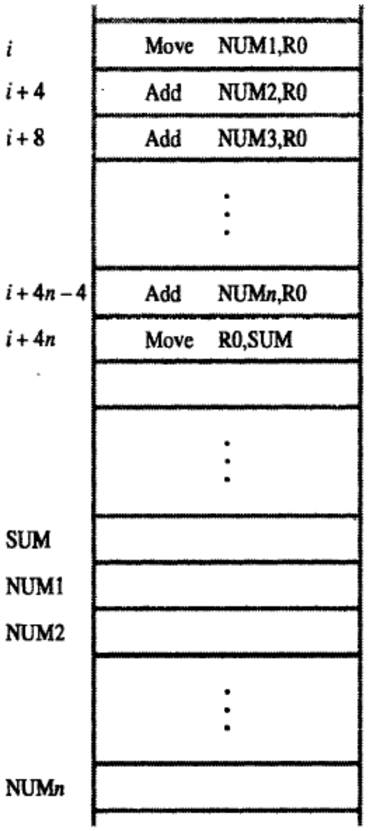


Fig.1.6 A straight line program for adding n numbers

- Instead of using a long list of add instructions, it is possible to place a single ADD instruction in a program loop as shown in Fig. 1.7
- The loop is a straight line sequence of instruction as many times as needed.
- Within the body of the loop, the instruction

  Decrement R1

  Reduces the contents of R1 by 1 each time through the loop.
- We use branch instruction to load a new value into the program counter.
- As a result a processor fetches and executes the instruction at this new address called the branch target.
- A contional branch instruction causes a branch only if a specified condition is satisfied.

  Branch >0 LOOP

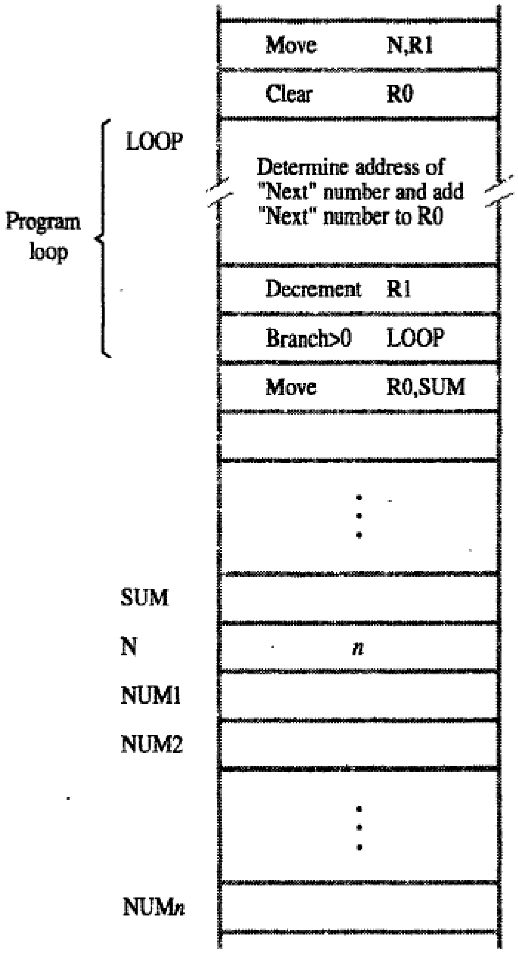


Fig 1.7 Using a loop to add n numbers

CONDITION CODES

The processor keeps track of information about the results of various operations for use by subsequent conditional branch instruction. This is accomplished by recording the required information in individual bits, called condition code flags. These flags are are usually grouped together in a special processor register called the condition code register or status register.

Individual condition code flags are set to 1 or cleared to 0, depending on the outcome of the operation perform. Four commonly used flags are

- N (negative)
- Z (zero)
- V (overflow)
- C (carry)

GENERATING MEMORY ADDRESSES

- How to specify the address of branch target?
- Can we give the memory operand address directly in a single Add instruction in the loop?
- Use a register to hold the address of NUM1; then increment by 4 on each pass through the loop.

ADDRESSING MODES

Programmers used organization called data structure to represent the data used in computation. these include list, linked list, queue, arrays. The different ways in which the location of an operand is specified in an instruction are referred to as Addressing modes

- Implied
  - AC is implied in “ADD M[AR]” in “One-Address” instr.
  - TOS is implied in “ADD” in “Zero-Address” instr.
- Immediate
  - The use of a constant in “MOV R1, 5”, i.e. R1 ← 5
- Register
  - Indicate which register holds the operand

- Register Indirect

  - Indicate the register that holds the number of the register that holds the operand

- MOV R1, (R2)

- Autoincrement / Autodecrement

  - Access & update in 1 instr.

- Direct Address

- Use the given address to access a memory location
- Indirect Address

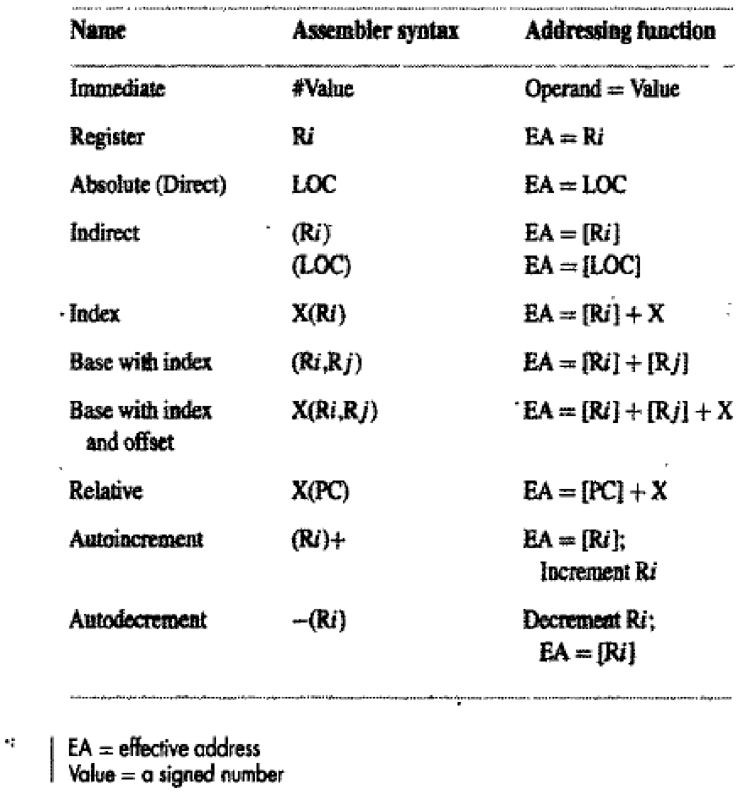
  Indicate the memory location that holds the address of the memory location that holds the data.
- Indexed
  - *EA* = Index Register + Relative Addr

- Base Register

  - *EA* = Base Register + Relative Addr

- Relative Address

  - *EA* = PC + Relative Addr



Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	Ri	EA = Ri
Absolute (Direct)	LOC	EA = LOC

Indirect	(Ri)
	(LOC)
		EA = [Ri]
		EA = [LOC]

Index	X(Ri)	EA = [Ri] + X
Base with index	(Ri,Rj)	EA = [Ri] + [Rj]
Base with index and offset	X(Ri,Rj)	EA = [Ri] + [Rj] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(Ri)+	EA = [Ri]; Increment Ri

Autodecrement	−(Ri)	Decrement Ri;
		EA = [Ri]

EA = effective address
Value = a signed number

INDEXING AND ARRAYS

- Index mode – the effective address of the operand is generated by adding a constant value to the contents of a register.
- Index register
- $X(R_i)$: EA = $X + [R_i]$
- The constant X may be given either as an explicit number or as a symbolic name representing a numerical value.
- If X is shorter than a word, sign-extension is needed
- In general, the Index mode facilitates access to an operand whose location is defined relative to a reference point within the data structure in which the operand appears.
- Several variations:
  (R_i, R_j): EA = $[R_i] + [R_j]$
  $X(R_i, R_j)$: EA = $X + [R_i] + [R_j]$

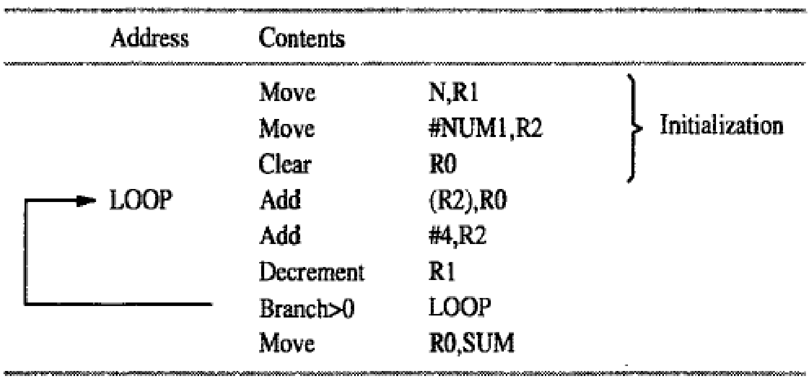
RELATIVE ADDRESSING

- Relative mode – the effective address is determined by the Index mode using the program counter in place of the general-purpose register.
- X(PC) – note that X is a signed number

- Branch>0 LOOP
- This location is computed by specifying it as an offset from the current value of PC.
- Branch target may be either before or after the branch instruction, the offset is given as a singed num.

ADDITIONAL MODES

- Autoincrement mode – the effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list.
- (R_i)+. The increment is 1 for byte-sized operands, 2 for 16-bit operands, and 4 for 32-bit operands.
- Autodecrement mode: -(R_i) – decrement first



Address	Contents		
	Move	N,R1	Initialization
	Move	#NUM1,R2	
	Clear	R0	
LOOP	Add	(R2),R0	
	Add	#4,R2	
	Decrement	R1	
	Branch>0	LOOP	
	Move	R0,SUM	

Fig.1.8 TheAutoincrement addressing mode

ASSEMBLY LANGUAGE

Machine Instructions Are Represented By patterns of 0s and 1s. symbolic names are used to represent patterns. When writing programs for specific computer, such words are normally replaced by acronyms called mnemonics such as MOV,ADD,INC, and BR.R3 is used to refer register 3, and LOC to refer to a location.

A complete set of such symbolic names and rules for their use constitute a programming language, generally referred to as an assembly language.

The set of rules for using the mnemonics in the specification of complete instructions and programs is called the syntax of the language.

An assembler is a type of computer program that interprets software programs written in assembly language into machine language, code and instructions that can be executed by a computer.

The user program in its original alphanumeric text format is called a source program, and the assembled machine language program is called an object program.

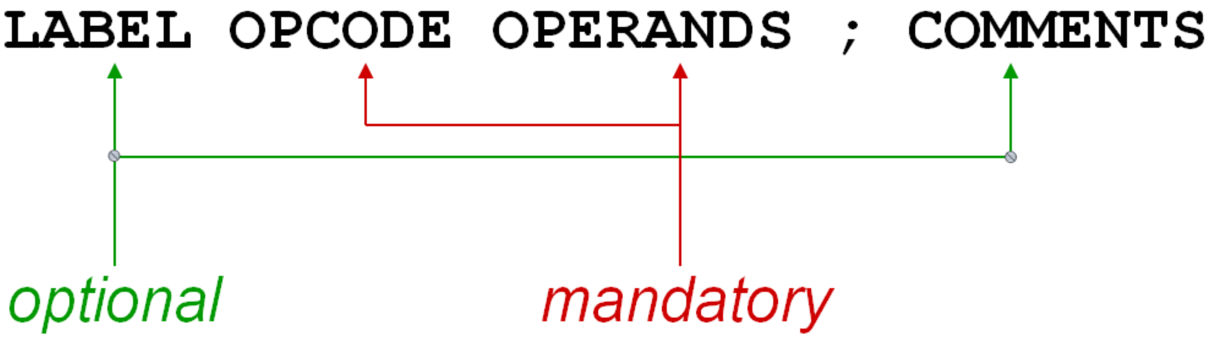
A typical assembly language consists of 3 types of instruction statements that are used to define program operations:

- Opcode mnemonics
- Data definitions
- Assembly directives

ASSEMBLY DIRECTIVES

Pseudo-operations

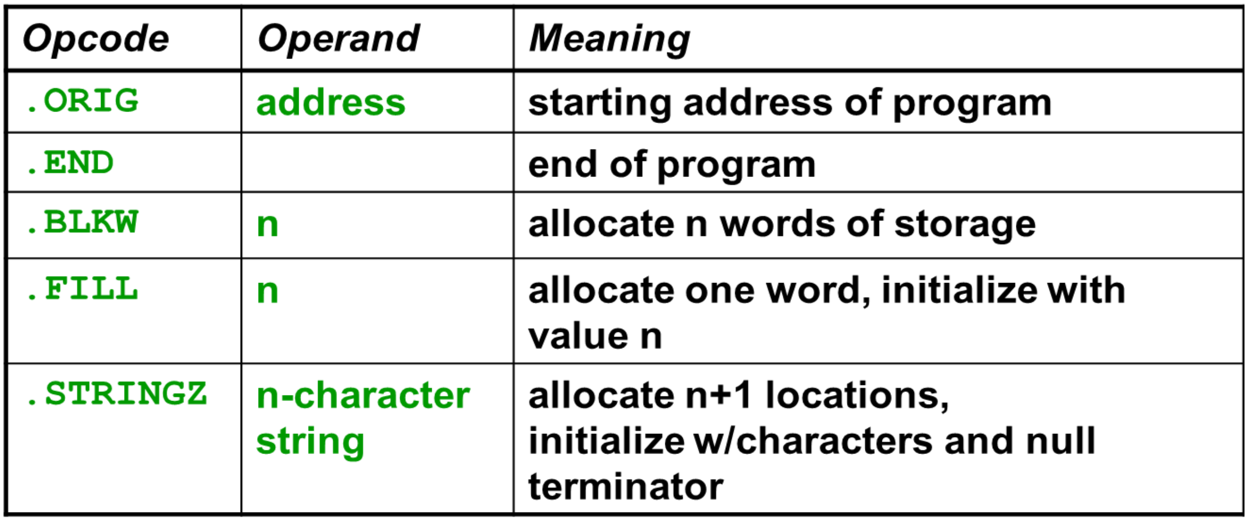
- do not refer to operations executed by program
- used by assembler
- look like instruction, but "opcode" starts with do**t**
- Each line of a program is one of the following:
- an instruction
- an assember directive (or pseudo-op)
- a comment
- Whitespace (between symbols) and case are ignored.
- Comments (beginning with ";") are also ignored.
- An instruction has the following format:



LABEL OPCODE OPERANDS ; COMMENTS

optional

mandatory



Opcode	*Operand*	*Meaning*
.ORIG	address	starting address of program
.END		end of program
.BLKW	n	allocate n words of storage
.FILL	n	allocate one word, initialize with value n
.STRINGZ	n-character string	allocate n+1 locations, initialize w/characters and null terminator

BASIC INPUT/OUTPUT OPERATIONS

- The data on which the instructions operate are not necessarily already stored in memory.
- Data need to be transferred between processor and outside world (disk, keyboard, etc.)
- I/O operations are essential, the way they are performed can have a significant effect on the performance of the computer.

  Program-Controlled I/O

  - Read in character input from a keyboard and produce character output on a display screen.

  Example

  - Rate of data transfer (keyboard, display, processor)
  - Difference in speed between processor and I/O device creates the need for mechanisms to synchronize the transfer of data.

- A solution: on output, the processor sends the first character and then waits for a signal from the display that the character has been received. It then sends the second character. Input is sent from the keyboard in a similar way.

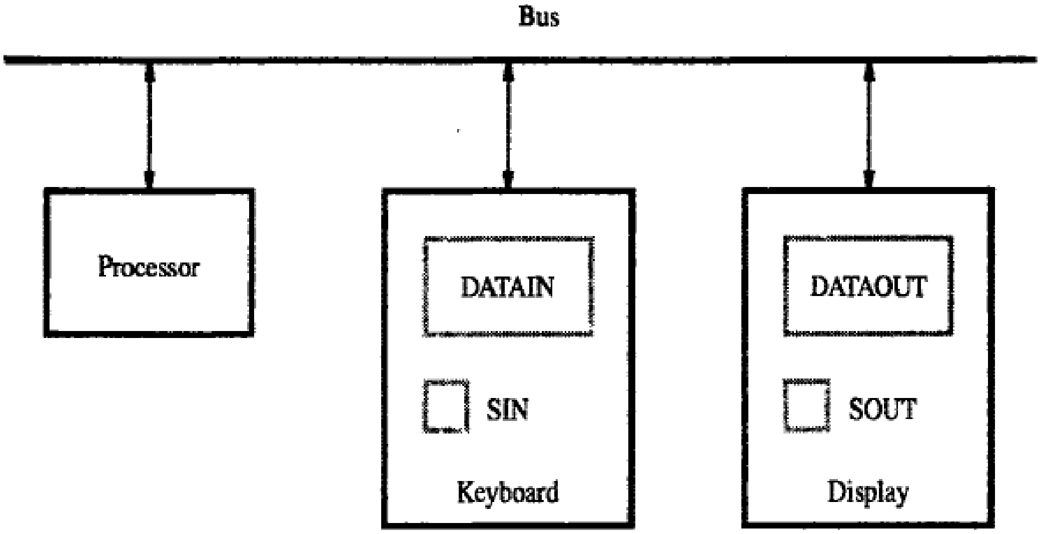


Fig. 1.9 Bus connection for processor, keyboard, and display.

- Machine instructions that can check the state of the status flags and transfer data:

```
READWAIT  Branch to READWAIT if SIN = 0
          Input from DATAIN to R1

WRITEWAIT Branch to WRITEWAIT if SOUT = 0
          Output from R1 to DATAOUT
```

- Memory-Mapped I/O – some memory address values are used to refer to peripheral device buffer registers. No special instructions are needed. Also use device status registers.

```
READWAIT  Testbit  #3, INSTATUS
          Branch=0  READWAIT
MoveByte  DATAIN, R1
```

- Assumption – the initial state of SIN is 0 and the initial state of SOUT is 1.
- Any drawback of this mechanism in terms of efficiency?
- Two wait loops→processor execution time is wasted

STACK

In order to organize the control and information linkage between the main program and the subroutine, the datastructure called the stack is used. A stack is a list of data elements, usually words or bits, with the accessing restrictions that elements can be added or removed at one end of the list only. This end is called the top of the stack and other end is called the bottom. The structure is sometimes referred to as Push down stack. It uses LIFO principle for two operations-Push and Pop.

The processor register is used to keep track of the address of the element of the stack i.e at the top at any given time. This register is called stack pointer(SP).

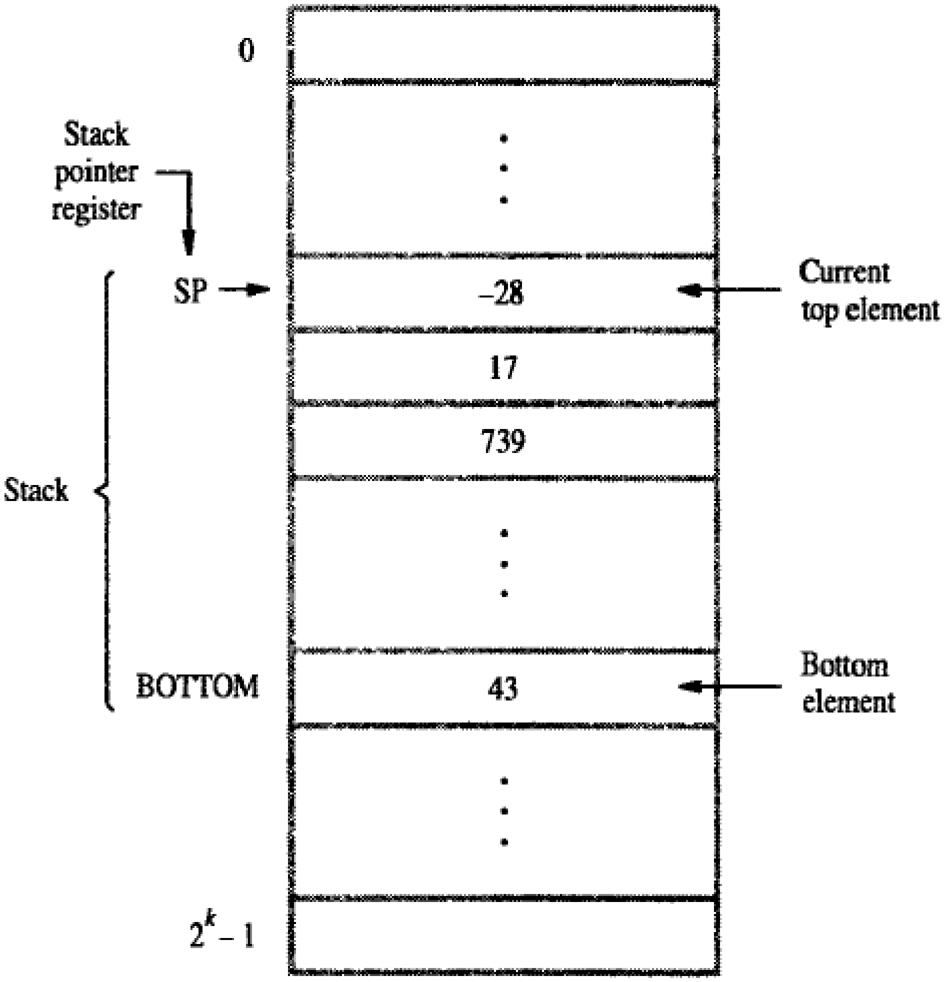
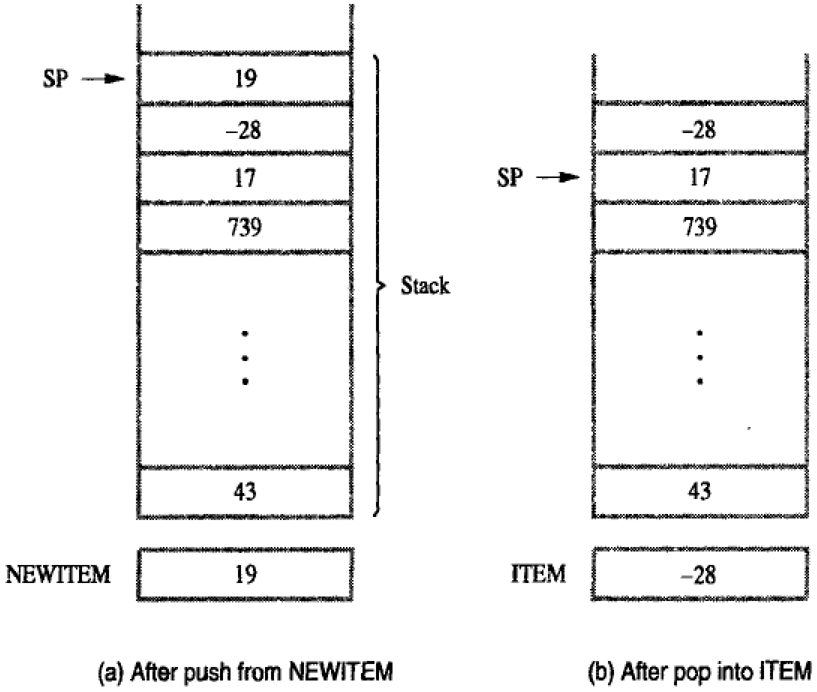


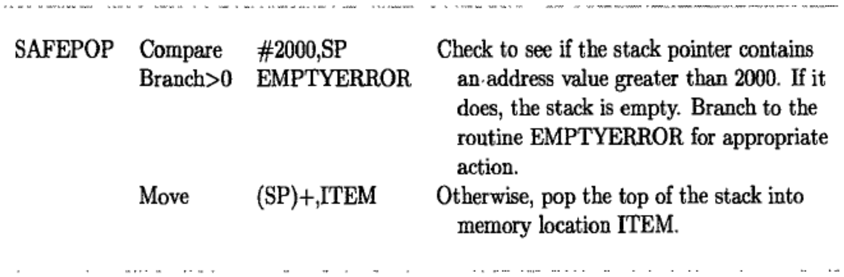
Fig. 1.10 A stack of words in the memory



(a) After push from NEWITEM

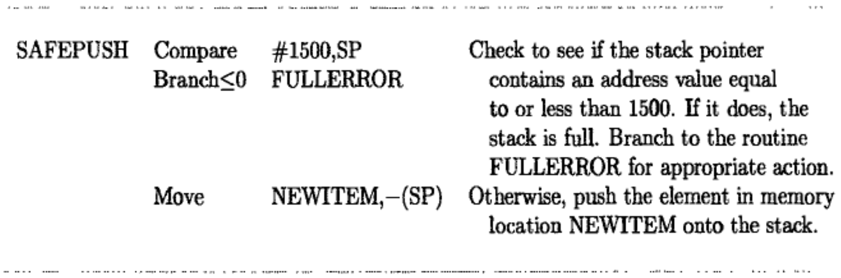
(b) After pop into ITEM

Fig 1.11 Effect of stack operation on the stack in the Fig 1.10



```
SAFEPOP   Compare    #2000,SP      Check to see if the stack pointer contains
          Branch>0   EMPTYERROR      an address value greater than 2000. If it
                                     does, the stack is empty. Branch to the
                                     routine EMPTYERROR for appropriate
                                     action.
          Move       (SP)+,ITEM    Otherwise, pop the top of the stack into
                                     memory location ITEM.
```

(a) Routine for a safe pop operation



```
SAFEPUSH  Compare    #1500,SP        Check to see if the stack pointer
          Branch≤0   FULLERROR         contains an address value equal
                                       to or less than 1500. If it does, the
                                       stack is full. Branch to the routine
                                       FULLERROR for appropriate action.
          Move       NEWITEM,–(SP)   Otherwise, push the element in memory
                                       location NEWITEM onto the stack.
```

(b) Routine for a safe push operation

Fig 1.12 Checking for empty and full errors in POP and PUSH operations.

QUEUE

Another useful datastructure that is similar to stack is called the queue. Data are stored and retrieved from a queue on FIFO basics. New data are added at the back and retieved from the front.

There are 2 important differences between how a stack and a queue is implemented.

One end of the is fixed while the other end rises and falls as data are pused and poped. A single pointer is needed to point to top of the stack, at on the other hand both ends of the queue move to higher addresses as data are added at the back and removed from the front. So two pointers are needed to keep track of the two ends of the queue.

One way to limit the queue to fixed region in a memory is to use a circular buffer.

SUBROUTINES

In a given program it is often necessary to perform a particular sub-task. Many times on different data values ,such a sub-task is usually called as sub-routines.

When a program branches to a sub-routines we say that it is calling the sub-routine. The instruction that performs this branch operation is named a call instruction.

After a sub-routine as been executed a calling program must resume execution, continuing immediately after the instruction that called the sub-routine.The sub-routine is said to return the program that called it byexecuting the return instruction.

Subroutine linkage – The way in which a computer makes it possible to and return from subroutine.

Link register – A register dedicated to the simplest subroutine linkage method is to save the return address in a specific location.

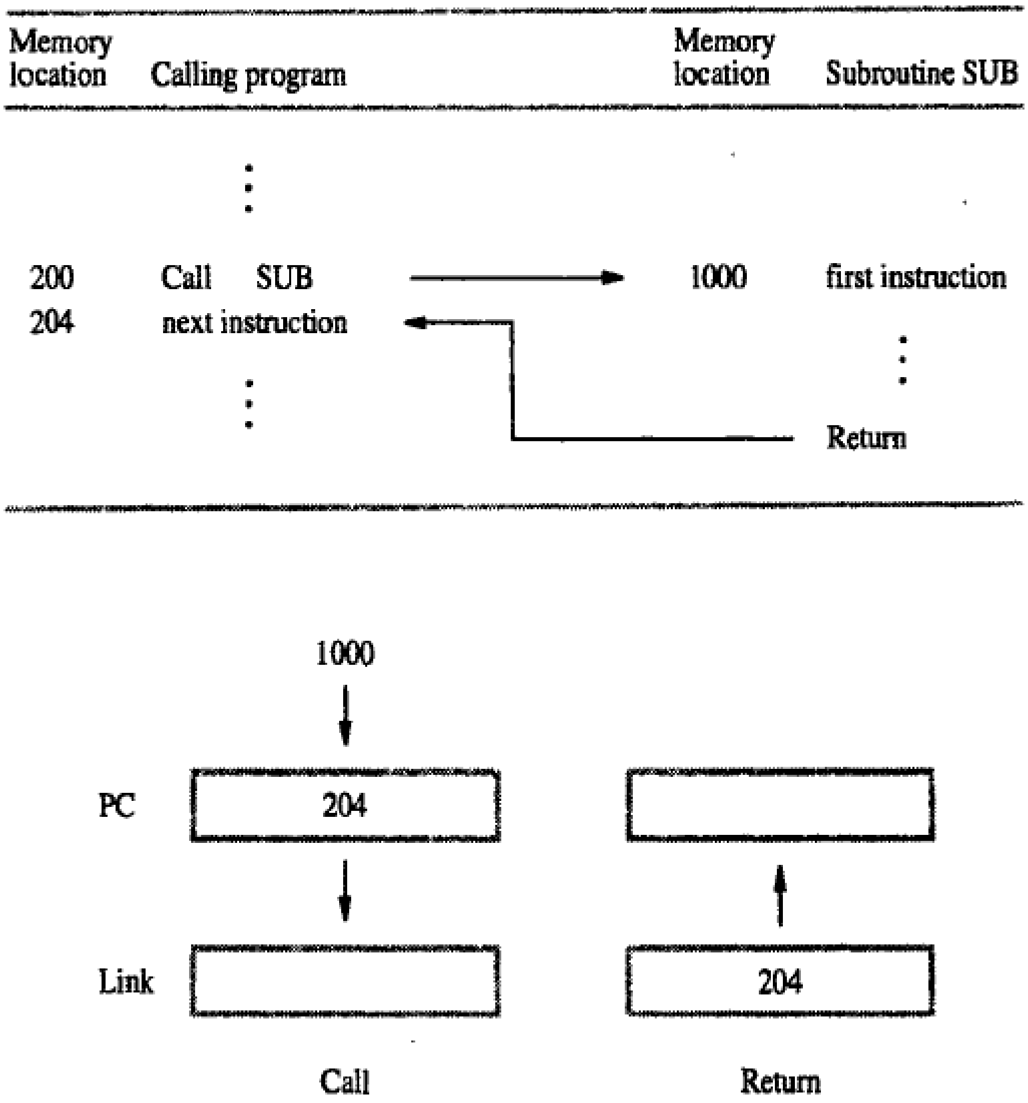


Fig 1.13 Subroutine linkage using A link register

SUBROUTINE NESTING

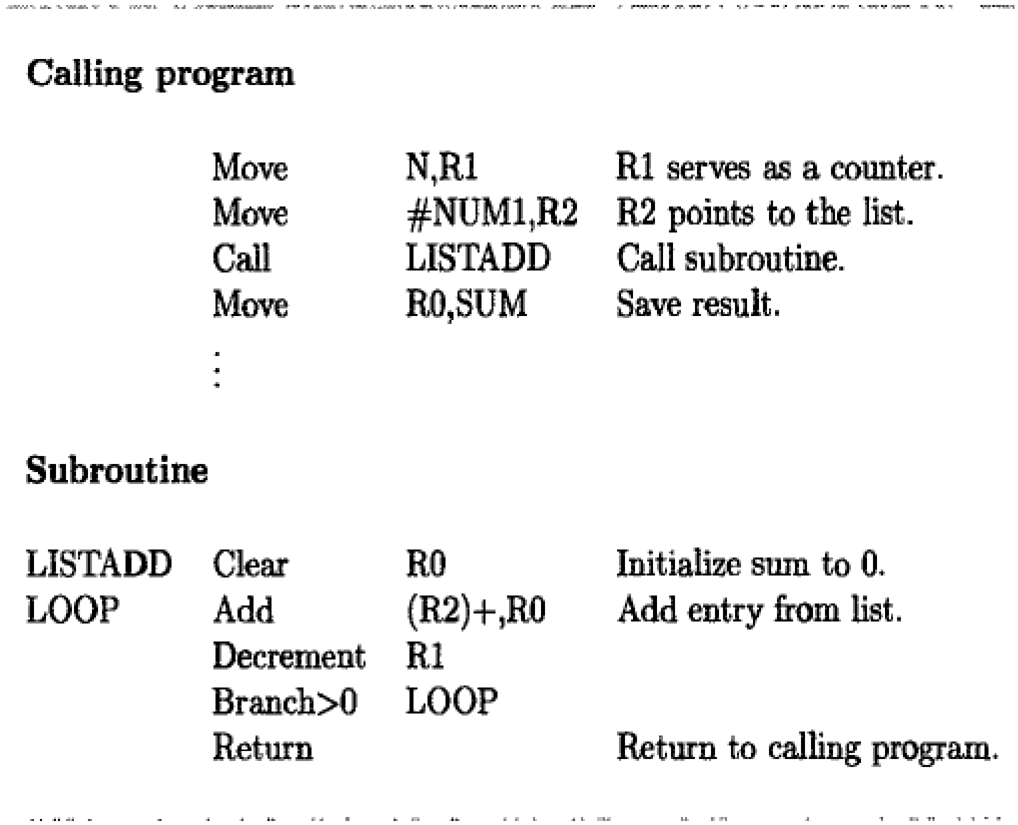
Subroutine nesting is to have one subroutine call another. In this case, the return address of the second call is also stored in the link register, destroying its previous contents. Hence, it is essential to save the contents of the link register in some other location before calling another subroutines.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

THE PROCESSOR STACK

The stack pointer points to the stack called the processor stack. The call instruction pushes the contents of the PC on to the processor stack and loads the subroutine address into the PC. The return instruction pops the return address from the processor stack into the PC.

PARAMETER PASSING

The exchange of information between a calling program and a subroutine is referred to as parameter passing. Parameter passing may be accomplished in several ways passing parameter through processor register is straight forward and efficient. Fig 1.14 shows how the program in fig 1.8 for adding a list of numbers can be implemented as a subroutine, with the parameters passed through registers.



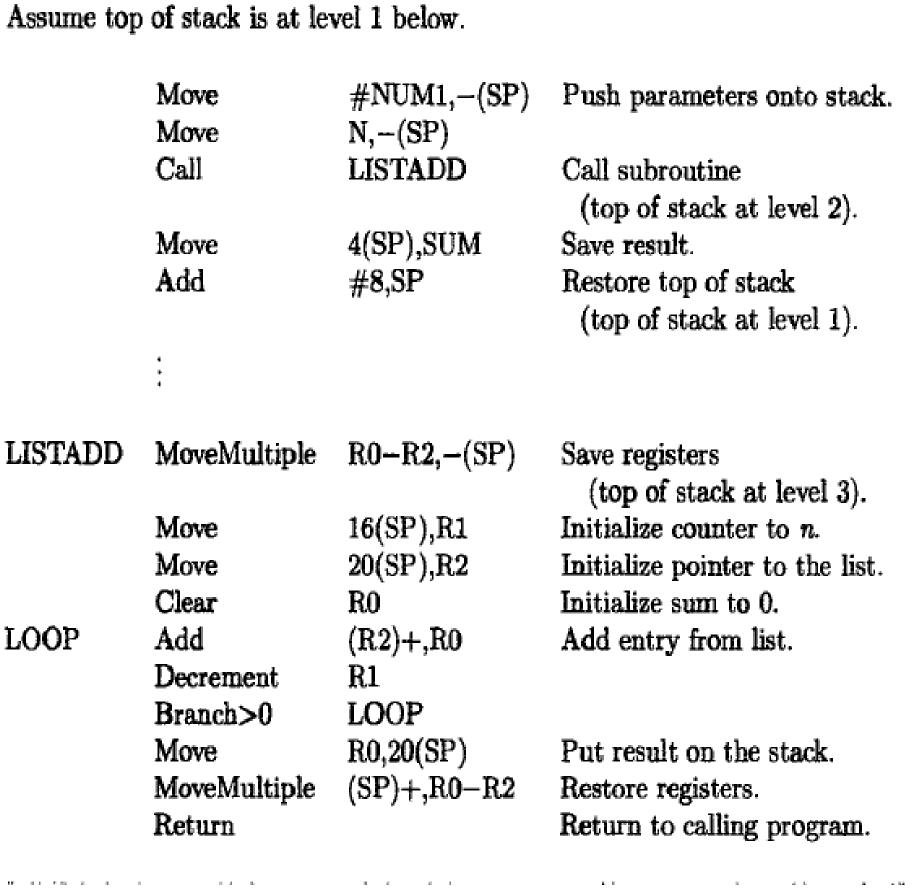
```
Calling program

            Move        N,R1         R1 serves as a counter.
            Move        #NUM1,R2     R2 points to the list.
            Call        LISTADD      Call subroutine.
            Move        R0,SUM       Save result.
            ⋮

Subroutine

LISTADD     Clear       R0           Initialize sum to 0.
LOOP        Add         (R2)+,R0     Add entry from list.
            Decrement   R1
            Branch>0    LOOP
            Return                   Return to calling program.
```

Fig 1.14 program of the Fig.1.8 written as a subroutine; parameters passed through registers

Parameter passing by values and by reference

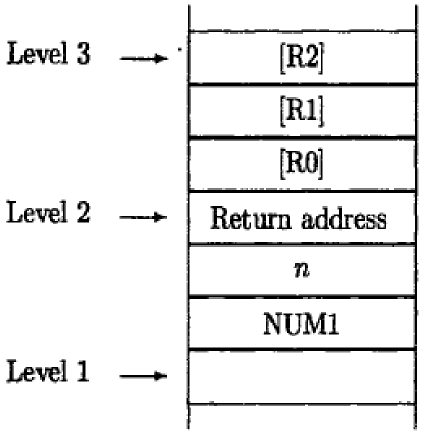
Note the nature of two parameters, NUM1 and n, passed to the subroutines in fig1.14 and fig 1.15The purpose of the subroutine is to add a list of numbers. Instead of passing the actual list entries.the calling program passes the address of the first number in the list. This technique is passing by reference.

The second parameter is passed by value ,i.e the actual number of entries, n, is passed to the subroutine



```
Assume top of stack is at level 1 below.

              Move           #NUM1,-(SP)   Push parameters onto stack.
              Move           N,-(SP)
              Call           LISTADD       Call subroutine
                                             (top of stack at level 2).
              Move           4(SP),SUM     Save result.
              Add            #8,SP         Restore top of stack
                                             (top of stack at level 1).
              :

LISTADD       MoveMultiple   R0-R2,-(SP)   Save registers
                                             (top of stack at level 3).
              Move           16(SP),R1     Initialize counter to n.
              Move           20(SP),R2     Initialize pointer to the list.
              Clear          R0            Initialize sum to 0.
LOOP          Add            (R2)+,R0      Add entry from list.
              Decrement      R1
              Branch>0       LOOP
              Move           R0,20(SP)     Put result on the stack.
              MoveMultiple   (SP)+,R0-R2   Restore registers.
              Return                       Return to calling program.
```

(a) Calling program and subroutine



(b) Top of stack at various times

Fig 1.15 program of fig 1.8 written as a subroutine, parameters passed on the stack.

THE STACK FRAME

During execution of the subroutine, six locations at the top of the stack contain entries that are needed by the subroutines. These locations constitute a private workspace for the subroutine, created at the time the subroutine is entered and freed up when the subroutines returns control to the calling program. Such space is called a stack frame.

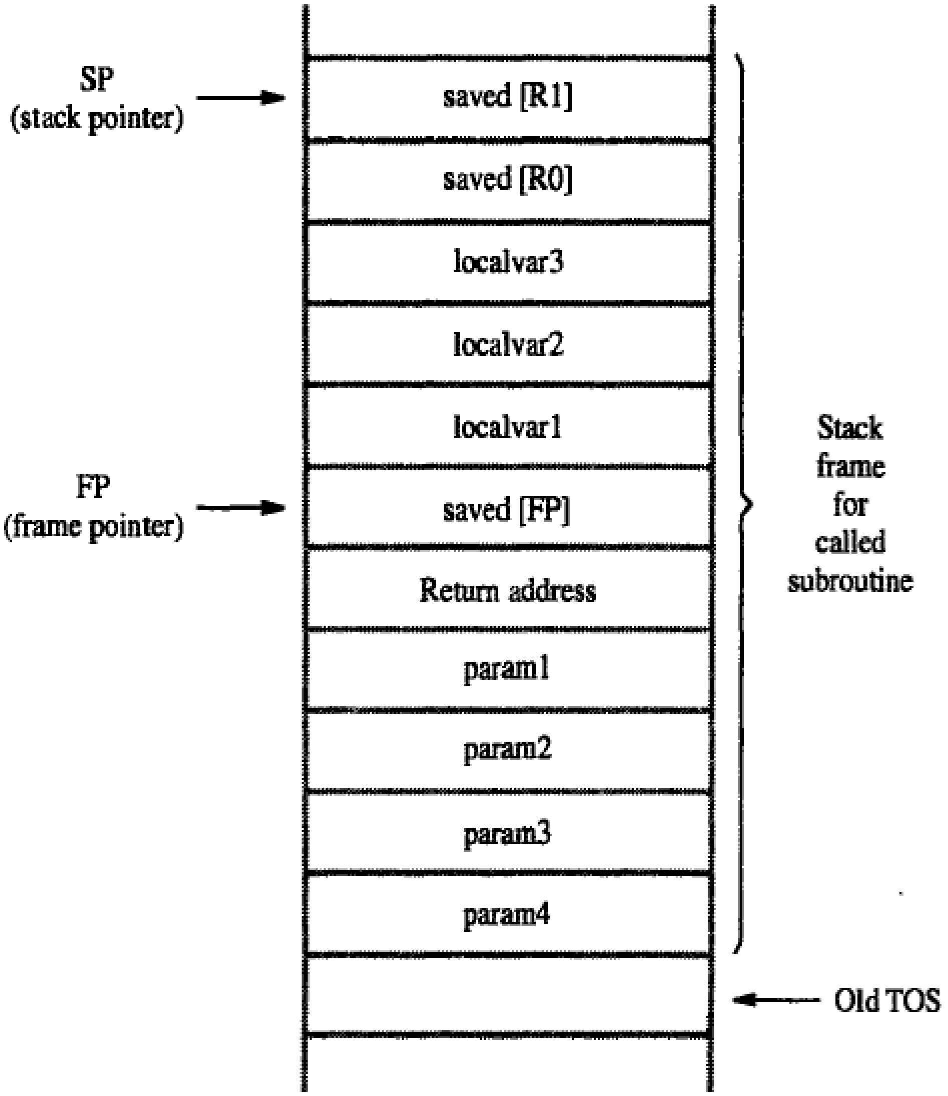
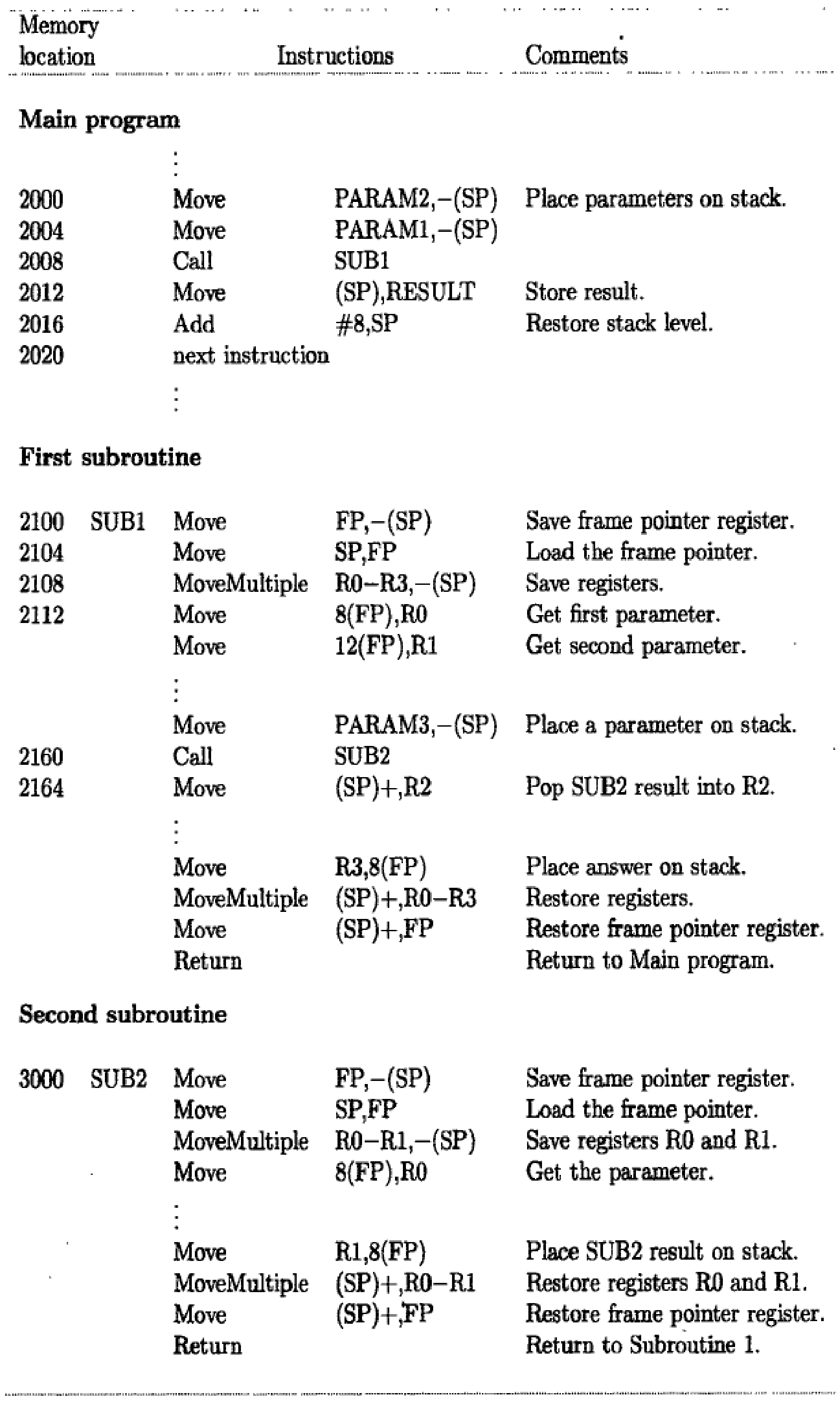


Fig 1.16 A subroutines stack frame example.

Fig 1.16 shows a example of a commonly used layout for information in a stack frame. In addition to a stack pointer(SP), it is useful to have another pointer register called the frame pointer(FP), for convenient access to the parameter passed to the subroutine and to the local memory variable used by the subroutines.



```
Memory
location              Instructions                    Comments

Main program
                  ⋮
2000              Move          PARAM2,−(SP)     Place parameters on stack.
2004              Move          PARAM1,−(SP)
2008              Call          SUB1
2012              Move          (SP),RESULT      Store result.
2016              Add           #8,SP            Restore stack level.
2020              next instruction
                  ⋮

First subroutine

2100    SUB1      Move          FP,−(SP)         Save frame pointer register.
2104              Move          SP,FP            Load the frame pointer.
2108              MoveMultiple  R0−R3,−(SP)      Save registers.
2112              Move          8(FP),R0         Get first parameter.
                  Move          12(FP),R1        Get second parameter.
                  ⋮
                  Move          PARAM3,−(SP)     Place a parameter on stack.
2160              Call          SUB2
2164              Move          (SP)+,R2         Pop SUB2 result into R2.
                  ⋮
                  Move          R3,8(FP)         Place answer on stack.
                  MoveMultiple  (SP)+,R0−R3      Restore registers.
                  Move          (SP)+,FP         Restore frame pointer register.
                  Return                         Return to Main program.

Second subroutine

3000    SUB2      Move          FP,−(SP)         Save frame pointer register.
                  Move          SP,FP            Load the frame pointer.
                  MoveMultiple  R0−R1,−(SP)      Save registers R0 and R1.
                  Move          8(FP),R0         Get the parameter.
                  ⋮
                  Move          R1,8(FP)         Place SUB2 result on stack.
                  MoveMultiple  (SP)+,R0−R1      Restore registers R0 and R1.
                  Move          (SP)+,FP         Restore frame pointer register.
                  Return                         Return to Subroutine 1.
```

Fig 1.17 A nested subroutine

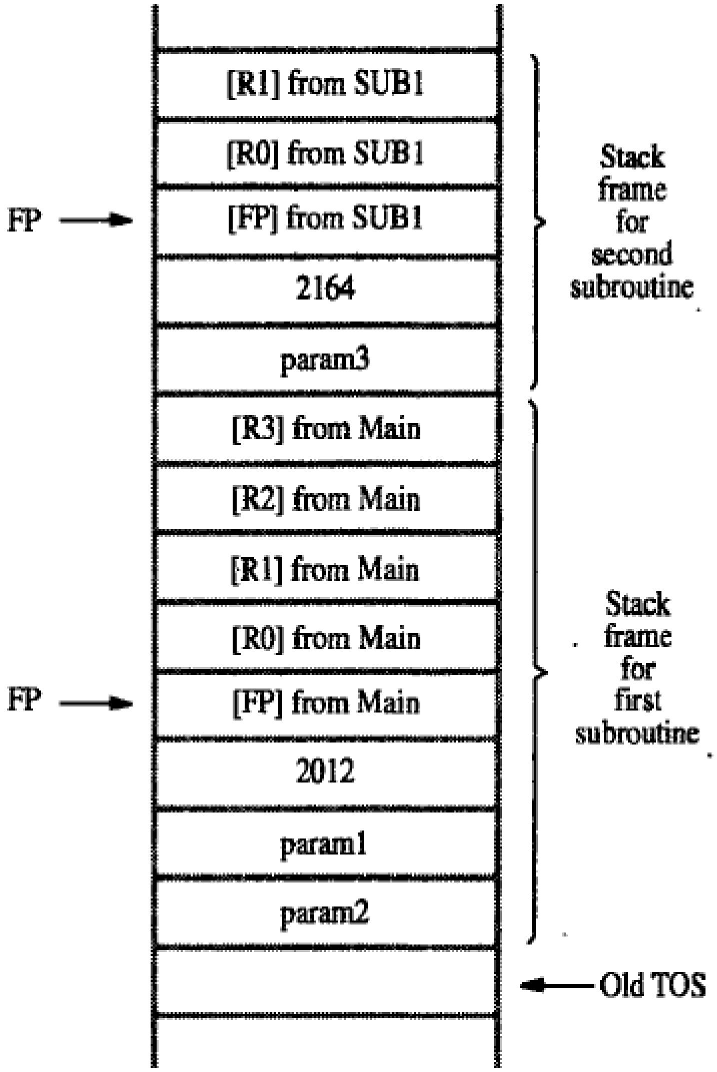


Fig 1.18 Stack frames fig 1.17

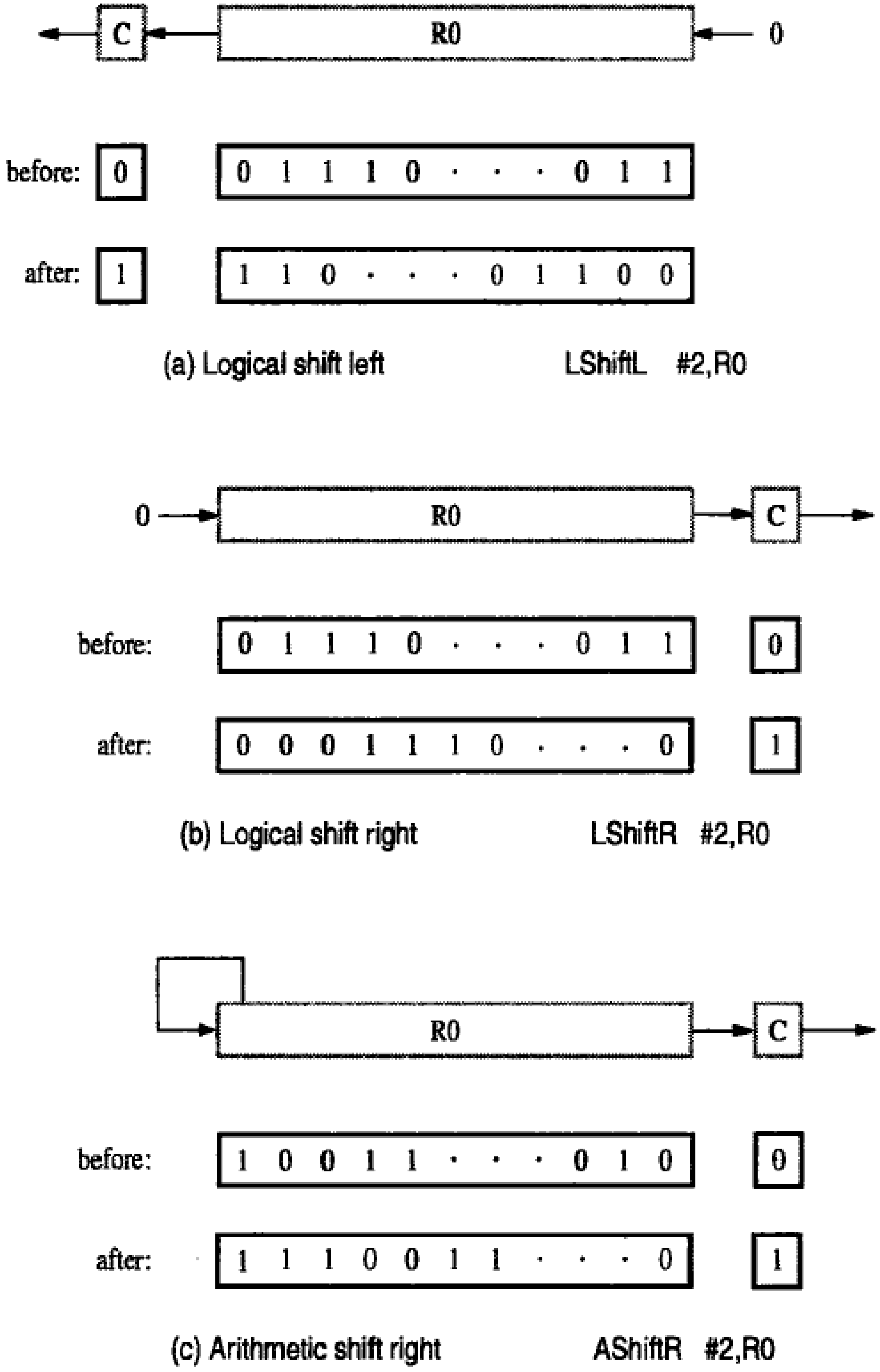
SHIFT AND ROTATE INSTRUCTIONS

There are many applications that require the bits of an operand to be shifted right or left some specified number of bit positions.

The details of how the shifts are performed depend on whether the operand is a signed number or some more general binary coded information. For general operand we use a logical shift , for number we use a arithmetic shift which preserves the sign of the number.

Logical Shift:

Two logical shift instructions are needed , one for shifting left(LShiftL) and another for shifting right (LShiftR).

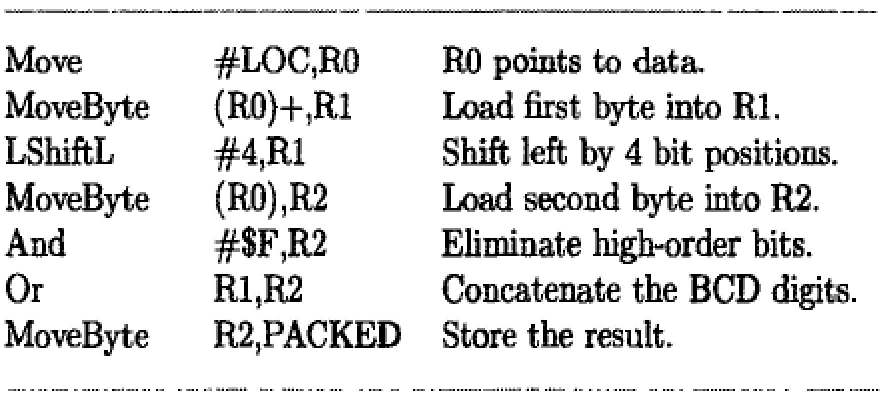


(a) Logical shift left LShiftL #2,R0

(b) Logical shift right LShiftR #2,R0

(c) Arithmetic shift right AShiftR #2,R0

Fig 1.19Logical and arithmetic swift instruction



```
Move        #LOC,R0       R0 points to data.
MoveByte    (R0)+,R1      Load first byte into R1.
LShiftL     #4,R1         Shift left by 4 bit positions.
MoveByte    (R0),R2       Load second byte into R2.
And         #$F,R2        Eliminate high-order bits.
Or          R1,R2         Concatenate the BCD digits.
MoveByte    R2,PACKED     Store the result.
```

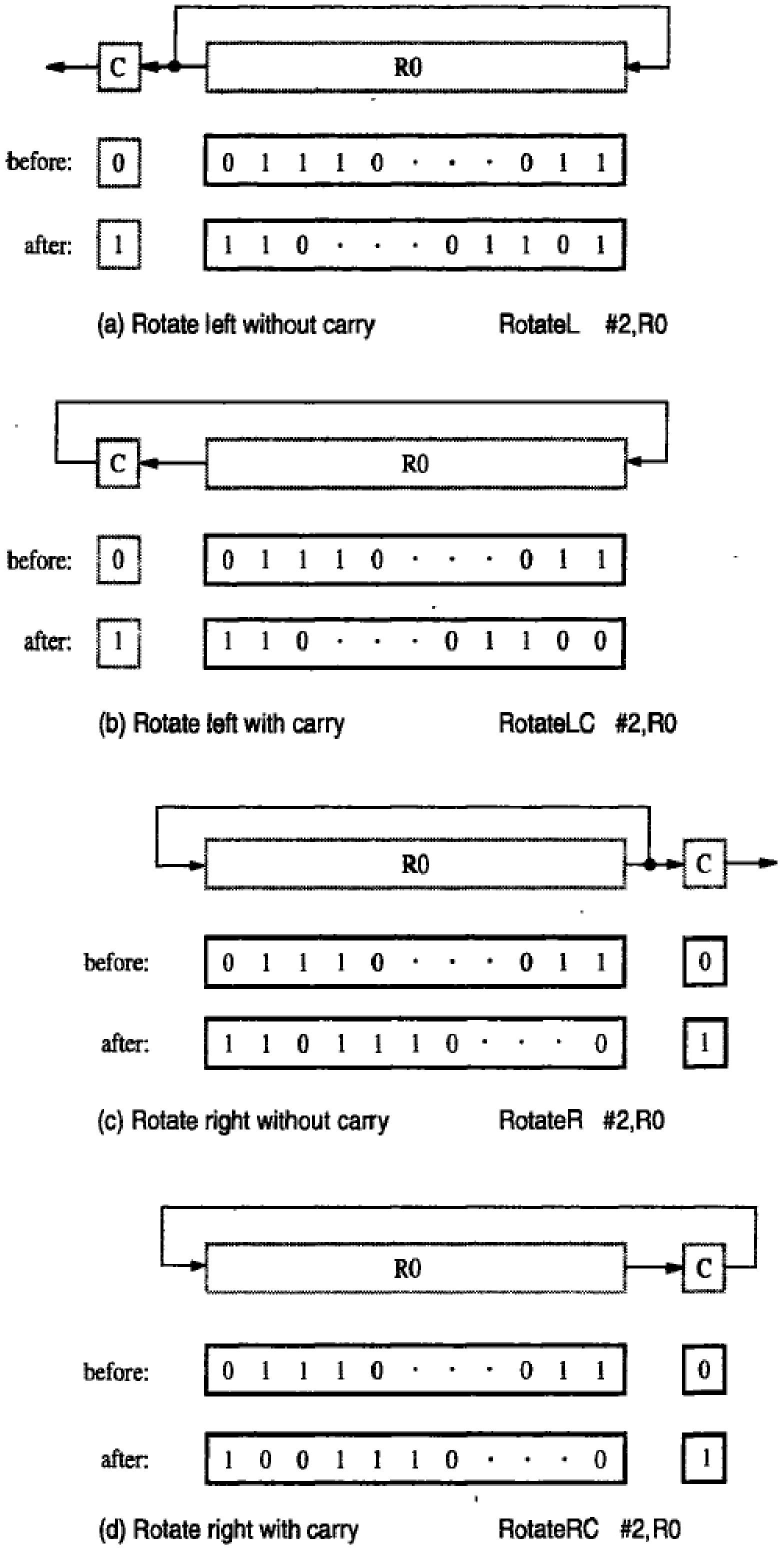
Fig 1.20 A routine that packs two BCD digits

Arithmetic Shifts

- Shifts may occur right and left.
- The requirement on right shifting distinguishes arthimetic shifts from logical in which the fill- in bits is always zero.
- Right shift – AShiftR
- Left shift – AShiftL
- The arithmetic left shift is exactly same as the logical left shift.

Rotate operation

- In the shift operations, the bit shifted out of the operand are lost, except for the last bit shifted out which is retain in the carry flag C.
- To preserve all bits, a set of rotate instructions can be used.
- Two versions of both the left and the right rotate instruction are usually provided.
- The mnemonics Rotate L,Rotate LC, Rotate R, and Rotate Rc, denote the instruction that the rotate operations.



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

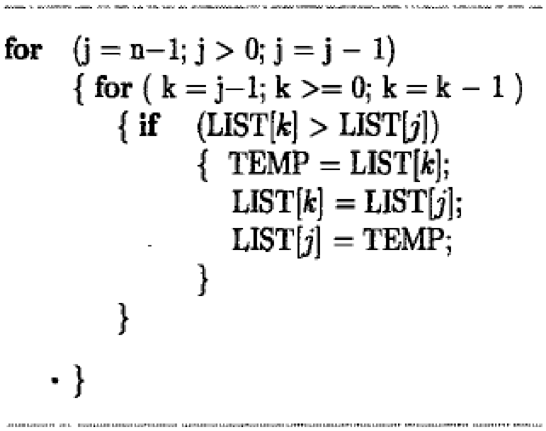
(b) Rotate left with carry RotateLC #2,R0

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

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

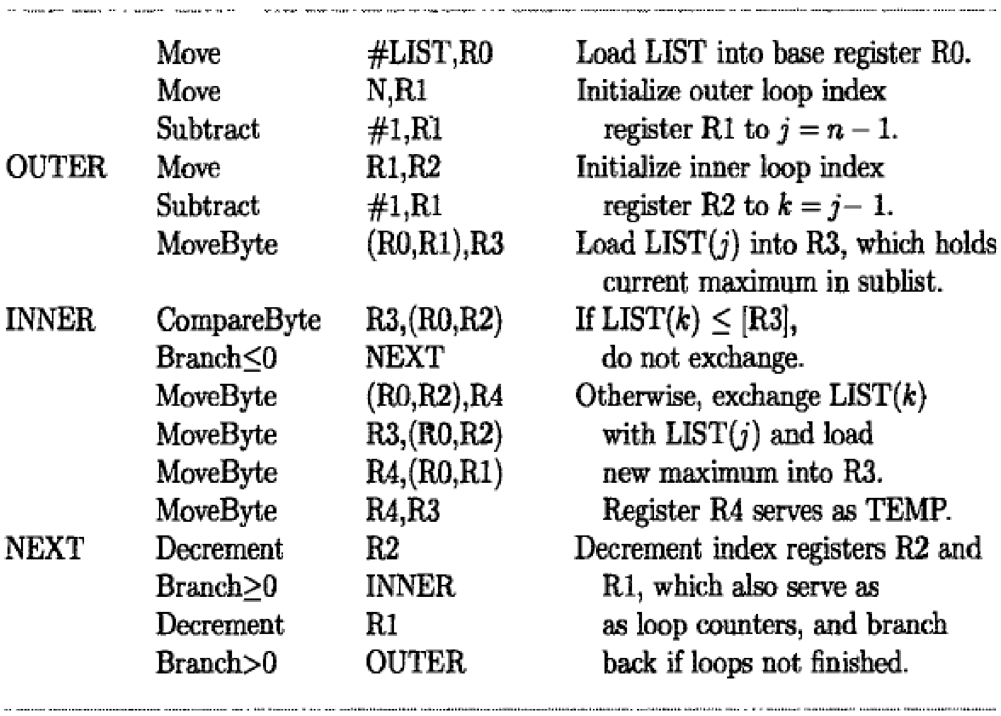
Fig1.21Rotate instructions

BYTE-SORTING PROGRAM



```
for  (j = n−1; j > 0; j = j − 1)
     { for ( k = j−1; k >= 0; k = k − 1 )
          { if   (LIST[k] > LIST[j])
                 { TEMP = LIST[k];
                    LIST[k] = LIST[j];
                    LIST[j] = TEMP;
                 }
          }
   }
```

(a) C-language program for sorting



```
        Move          #LIST,R0      Load LIST into base register R0.
        Move          N,R1          Initialize outer loop index
        Subtract      #1,R1           register R1 to j = n − 1.
OUTER   Move          R1,R2         Initialize inner loop index
        Subtract      #1,R1           register R2 to k = j− 1.
        MoveByte      (R0,R1),R3    Load LIST(j) into R3, which holds
                                      current maximum in sublist.
INNER   CompareByte   R3,(R0,R2)    If LIST(k) ≤ [R3],
        Branch≤0      NEXT            do not exchange.
        MoveByte      (R0,R2),R4    Otherwise, exchange LIST(k)
        MoveByte      R3,(R0,R2)      with LIST(j) and load
        MoveByte      R4,(R0,R1)      new maximum into R3.
        MoveByte      R4,R3           Register R4 serves as TEMP.
NEXT    Decrement     R2            Decrement index registers R2 and
        Branch≥0      INNER           R1, which also serve as
        Decrement     R1              as loop counters, and branch
        Branch>0      OUTER           back if loops not finished.
```

(b) Assembly language program for sorting

Fig 1.22 A byte-sorting program using straight – selection sort.

UNIT II

The IA-32 Pentium Example: Registers and Addressing, IA-32 Instructions, IA-32 Assembly Language, Program Flow Control, Logic and Shift/Rotate Instructions, I/O Operations, Subroutines, Other Instructions, Program Examples.

2 Marks

1. Write about IA-32.

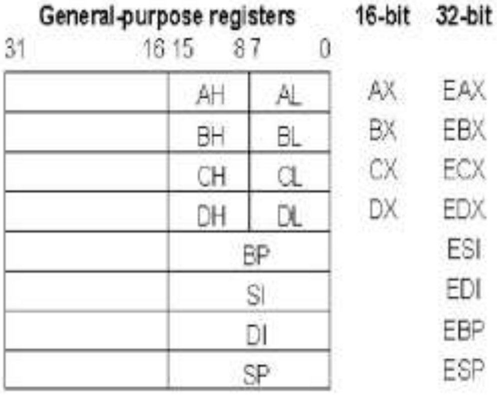
The Intel architecture (IA) processors operate with 32-bit memory address and 32-bit data operands. They are referred to as IA-32 processors, and the most recent Pentium series.

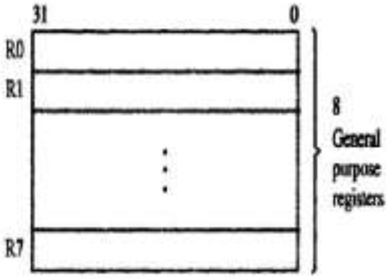
The first IA-32 processor was 80386 then 80486, Pentium, Pentium Pro, Pentium II, Pentium III, Pentium 4 has been implemented. Theses processors has increasing level of performance, achieved through a number of architectural and microelectronic technology improvements. The latest members of the family have specialized instructions for handling multimedia information and for vector data processing.

2. List out the various register of IA- 32 Register structure

- General purpose registers
- floating point registers
- segment registers
- instruction pointer
- status registers

3. Write about General purpose registers

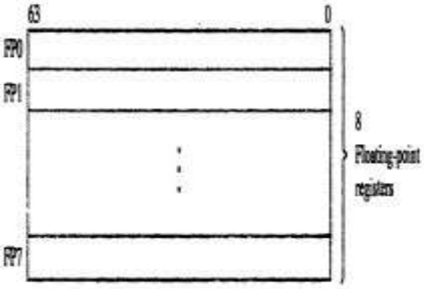




The above diagram shows the general purpose registers. The eight 32-bit registers labeled R0 through R7 are general purpose registers that can be used to bold either data operands or addressing information.

4. Write about floating point registers

There are eight floating point registers for holding double word or quad word (64 bits) floating point data operands. The floating point registers have an extension field to provide a total length of 80 bits, the extra bits are used for increased accuracy while floating point numbers are operated on in the processor.



5. What is Segment registers

There are six segment registers that hold 16-bit segment selectors. A segment selector is a special pointer that identifies a segment in memory. The six segment registers are:

- CS: code segment register
- SS: stack segment register
- DS, ES, FS, GS: data segment registers

Code segment (CS)

The code segment holds the instructions of a program.

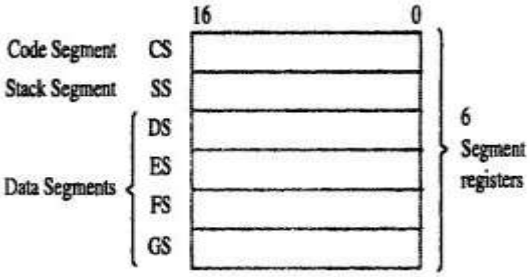
Stack segment (SS)

The stack segment contains the processor stack.

Data segment (DS, RD, FS, and GS)

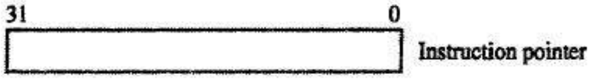
Four data segments are provided for holding data operands. These four data segment registers provide programs with flexible and efficient ways to access data.

The six segment registers below contain selector values that are used in locating these segments in the memory address space.

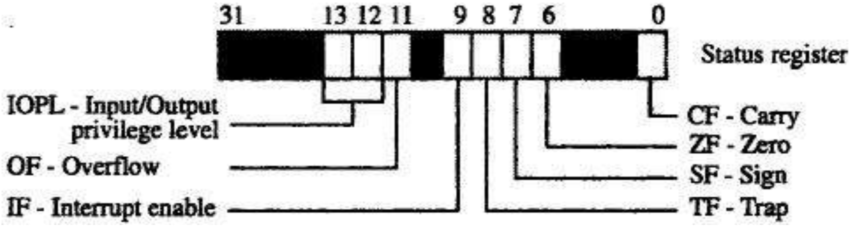


6. Define Instruction pointer

The **EIP** register is the 32-bit instruction pointer. The EIP register (or instruction pointer) can also be called "program counter." It contains the offset in the current code segment for the next instruction to be executed. It is advanced from one instruction boundary to the next in straight-line code or it is moved ahead or backwards by a number of instructions when executing JMP,CALL, RET instructions.



7. Write about status registers



The status register holds the condition code flags (CF,ZF,SF,OF).These flags contain information about the results of arithmetic operations.

8. What is EFLAGS Register

The 32-bit EFLAGS register contains a group of status flags, a control flag, and a group of system flags. When it is used in the conditional control instructions look at the condition code bits (in the EFLAGS register) to make a decision on whether to take the jump or not.

9. Write about ESP register and EBP register

The **ESP** register is the 32-bit stack pointer, used to manage push and pop operations.

The **EBP** register is the 32-bit base pointer. In connection with the ESP, the EBP is used to manage the stack frame, that part of the stack used to communicate with subprograms and store local variables.

10. Write about the Index Registers

The **ESI** and **EDI** registers are used as source and destination addresses for string and array operations.

The ESI "**Extended Source Index**" and EDI "**Extended Destination Index**" facilitate high–speed memory transfers.

11.* List the various addressing modes of IA-32 *(April 2015)

The addressing mode of IA-32 includes:

1. Immediate mode
2. Direct mode
3. Register mode
4. Register indirect mode
5. Base with displacement mode
6. Index with displacement mode
7. Base with index mode
8. Base with index and displacement mode

12. What is immediate mode? Give example

The operand is contained in the instruction. It is a signed 8-bit or 32-bit number, with the length being specified by a bit in the OP code of the instruction. Thus bit is 0 for the short version and 1 for the long version.

Instruction format:

Opcode	Register	Value

Example:

MOV EAX, 25

The above instruction moves the decimal value 25 into the EAX register. A number given in this form using the digits 0 through 9 is assumed to be in decimal notation .the suffix B and H are used to specify binary and hexadecimal numbers, respectively .For example the instruction,

MOV EAX, 3FAOOH

Moves the hex number 3FA00 into EAX.

13. What is direct mode? Give example

The memory address of the operand is given by a 32-bit value in the instruction

Instruction format:

Opcode	Register	Location

Example:

MOV EAX, LOCATION

The above instruction uses the direct addressing mode to move the doubleword at the memory location specified by the address label LOCATION into register EAX.This assumes that the LOCATION has been defined as an address label for a memory location in the data declaration. If LOCATION represents the address 1000 then this instruction moves the doubleword at 1000 into EAX.

In the IA-32 assembly language square brackets can be used to explicitly indicate the direct addressing mode as in the instruction.

MOV EAX, [LOCATION]

14. What is Register mode? Give example

The operand is contained in one of the general purpose register specified in the instruction.

Instruction format:

Opcode	Dest.Register	Src.Register

Example:

MOV EAX, ECX

Both operands use register mode. The contents of register **ECX** are copied to register EAX.

Before execution of the above instruction, the contents of ECX and EAX are:

ECX
50

EAX
00

After execution

ECX
50

EAX
50

15. What is Register indirect mode? Give example

The memory address of the operand is contained in one of the eight general purpose register specified in the instruction.

Instruction format:

Opcode	Register	[Register]

Example:

MOV EAX,[EBX]

The above instruction moves the contents of LOCATION specified by the register EBX into the register EAX.

Before execution of the above instruction, the contents of EAX and EBX are:

EAX
00

EBX
1000

20

1000

After execution

EAX
20

EBX
1000

20

1000

16. What is Base with displacement mode? Give example

An 8-bit or 32-bit signed displacement and one of the eight general purpose register to be used as a base register are specified in the instruction. The effective address of the operand in the sum of the contents of the base register and the displacement.

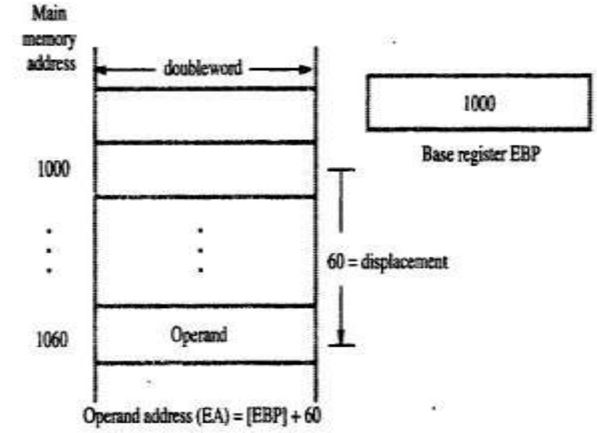
Instruction format:

Opcode	Dest.Register	[Register]+Displacement

Example:

MOV EAX, [EBP + 60]

The second operand uses base displacement mode. The instruction contains a constant. That constant is added to the contents of register EBP to form an effective address. The contents of memory at the effective address are copied into register EAX.



(a) Base with displacement mode, expressed as [EBP + 60]

17. What is Index with displacement mode? Give example

A 32-bit signed displacement, one of the eight general purpose register to be used as an index register, and a scale factor of 1,2,4 or 8 are specified in the instruction. To obtain the effective address of the operand, the contents of the index register are multiplied by the scale factor and then added to the displacement.i.e

Offset = (Index * Scale) + displacement

Instruction format:

Opcode	**Dest.Register**	**[Register] * Scale factor +Displacement**

Example:

MOV AL,[EBP * 4 + 10]

18. What is Base with index mode? Give example

Two of the eight general purpose register and a scale factor of 1,2,4 or 8 are specified in the instruction. The register is used as base and index register and the effective address of the operand is calculated as follows: the contents of the index register are multiplied by the scale factor and added to the contents of the base register. i.e

Offset = Base + (Index * Scale)

Instruction format:

Opcode	**Dest.Register**	**[Register1] + [Register2] * Scale factor**

Example:

MOV EAX, [ESP+ESI*4]

The contents of registers ESI is multiplied with the scale factor 4 and then the content of ESP is added to form an effective address. The contents of memory at the effective address are copied into register EAX

19. What is Base with index and displacement mode? Give example

An 8 bit or 32 –bit signed displacement two of the eight general purpose registers and a scale factor of 1,2,4 or 8 are specified in the instruction. The register is used as base and index register and the effective address of the operand is calculated as follows: the contents of the index register are multiplied by the scale factor and then added to the contents of the base register and the displacement.

An effective address is computed by:

Offset = Base + (Index * Scale) + displacement

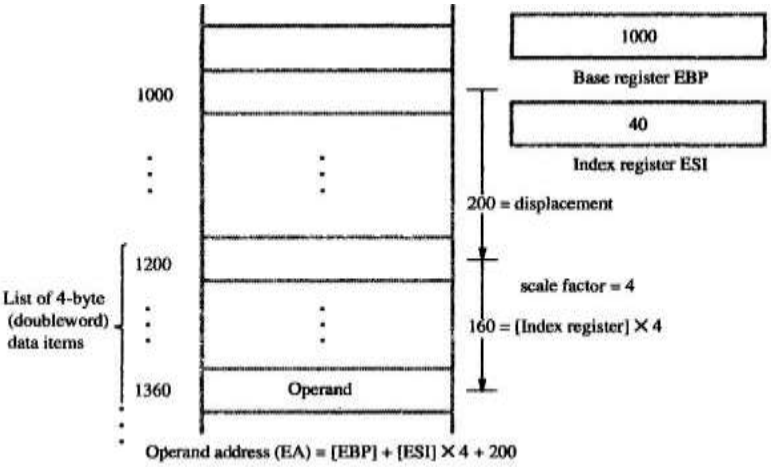
Instruction format:

Opcode	**Dest.Register**	**[Register1] + [Register2] * Scale factor + Displacement**

Example:

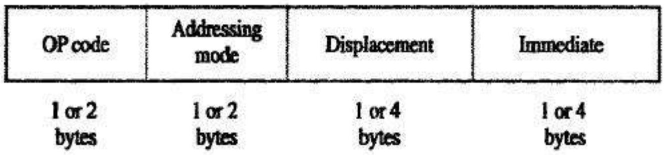
MOV EAX, [EBP+ESI*4 + 200]

The contents of registers ESI is multiplied with the scale factor 4 and then the content of EBP and displacement are added to form an effective address. The contents of memory at the effective address are copied into register EAX


1000
Base register EBP
40
Index register ESI
1000
200 = displacement
1200
scale factor = 4
List of 4-byte (doubleword) data items
160 = [Index register] × 4
1360
Operand
Operand address (EA) = [EBP] + [ESI] × 4 + 200

(b) Base with displacement and index mode, expressed as [EBP + ESI * 4 + 200]

20.Write about IA-32 Instruction



OP code	Addressing mode	Displacement	Immediate
1 or 2 bytes	1 or 2 bytes	1 or 4 bytes	1 or 4 bytes

The instructions are variable in length ,ranging from one byte to 12 bytes ,consisting of up to four fields .the OP-code field consists of one or two bytes, with most instructions requiring only one byte, the addressing mode information is contained in one or two bytes immediately following the OP code. For instructions that involve the use of only one register in generating the effective address of an operand ,only one byte is needed in the addressing mode field. Two bytes are needed for encoding the last two addressing modes. These modes use two registers to generate the effective address of a memory operand.

21. What is One byte instruction?

Registers can be incremented or decremented by instructions, that occupy one byte. Examples are

INC EDI

And

DEC ECX

In which the general purpose register EDI and ECX are specified by 3-bit codes in the single OP-code byte.

22. What is immediate mode encoding?

The OP-code specifies when the immediate addressing mode is used. For example the instruction

MOV EAX, 820

Is encoded into 5 bytes .a one –byte OP code specifies the move operation, the fact that a 32-bit immediate operand is used and the name of the destination register. The OP code byte is directly followed by the 4-byte immediate value of 820.when an 8-bit immediate operand is used, as in the instruction

MOV DL, 5

Only two bytes are needed to encode the instruction

23. In which situation a program flow control changes.

There are two main ways in which the flow of executing instructions varies from straight –line sequencing, calls to subroutines and returns from them break straight line sequencing. Also, branch instructions, either conditional or unconditional, can cause a break. The branch instructions are called jumps.

24. Write about Conditional jump instruction

The conditional Jump instructions test the four condition code flags in the status register. The instruction

JG LABEL

is an example of a conditional Jump instruction. The condition is *greater-than* as indicated by the G suffix in the OP code.

25. Write about Unconditional Jump Instruction.(nov 204)

An unconditional Jump instruction, JMP, causes a branch to the instruction at the target address. In addition to using short (one-byte) or long (four-byte) relative signed offsets to determine the target address, as is done in conditional Jump instructions, the JMP instruction also allows the use of other addressing modes.

26. What is the use of Compare Instructions?

It is often necessary to make conditional jumps in a program based on the results of comparing two numbers. The compare instruction

CMP dst,src

performs the operation

$$dst \leftarrow [src]$$

and sets the condition code flags based on the result obtained. Neither of the operands is changed .the first operand is always compared to the second. For example, the compare instruction by a conditional jump that is based on the “greater than” condition, then the jump will take to the target address if the destination operand is greater than the source operand.

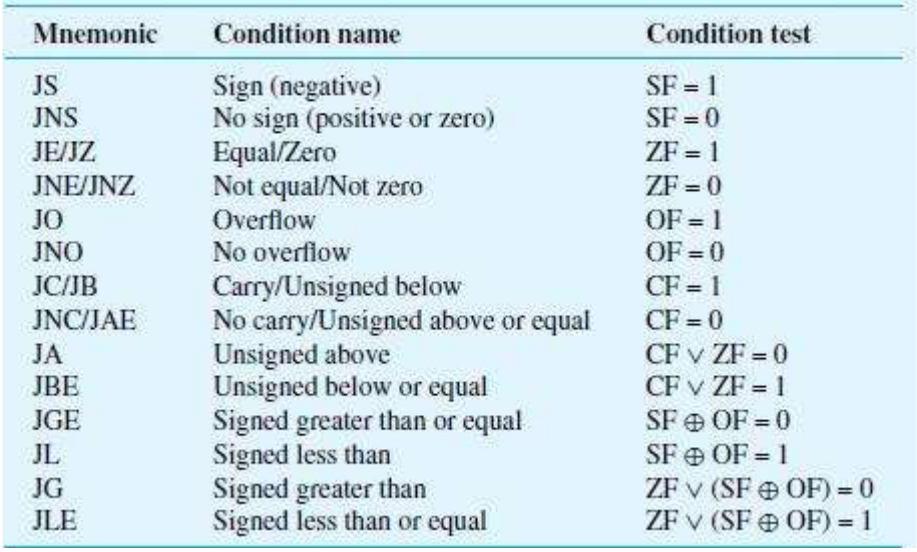
27. What is Logical Shift instruction?

An operand can be shifted right or left, using either logical or arithmetic shifts,by a number of bit positions determined by a specified count. The format of the shift instruction is

OPcode dst , count

Where the destination operand to be shifted is specified by the general addressing modes and the count is given either as an 8-bit immediate value or is contained in the 8-bit register CL.

28. List out the various jump instructions of IA-32



Mnemonic	Condition name	Condition test
JS	Sign (negative)	SF = 1
JNS	No sign (positive or zero)	SF = 0
JE/JZ	Equal/Zero	ZF = 1
JNE/JNZ	Not equal/Not zero	ZF = 0
JO	Overflow	OF = 1
JNO	No overflow	OF = 0
JC/JB	Carry/Unsigned below	CF = 1
JNC/JAE	No carry/Unsigned above or equal	CF = 0
JA	Unsigned above	$CF \vee ZF = 0$
JBE	Unsigned below or equal	$CF \vee ZF = 1$
JGE	Signed greater than or equal	$SF \oplus OF = 0$
JL	Signed less than	$SF \oplus OF = 1$
JG	Signed greater than	$ZF \vee (SF \oplus OF) = 0$
JLE	Signed less than or equal	$ZF \vee (SF \oplus OF) = 1$

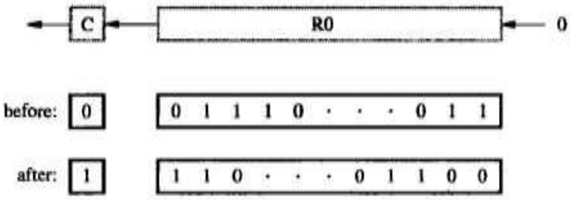
29. List out the various logical shift instruction

There are four shift instructions:

- SHL (Shift left logical)
- SHR (Shift right logical)
- SAL (Shift left arithmetic; operation is identical to SHL)
- SAR (Shift right arithmetic)

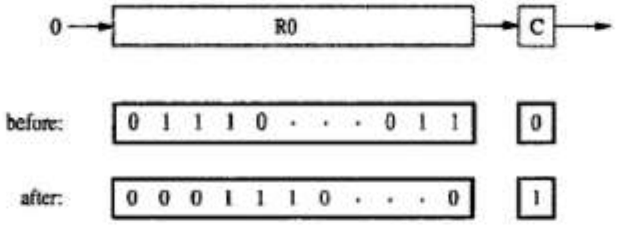
30. What is SHL instruction?

The SHL (shift left) instruction performs a logical left shift on the destination operand, filling the lowest bit with 0.For example, **SHL R0, #2**



31. What is SHR instruction?

The SHR (shift right) instruction performs a logical right shift on the destination operand. The highest bit position is filled with a zero. For example, **SHR R0,#2**



32. What is SAL and SAR

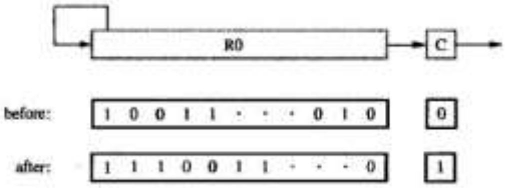
instruction SAL:

The operation of SAL is identical to SHL.

SAR:

SAR (shift arithmetic right) performs a right arithmetic shift on the destination operand. For example,

SAR R0,#2



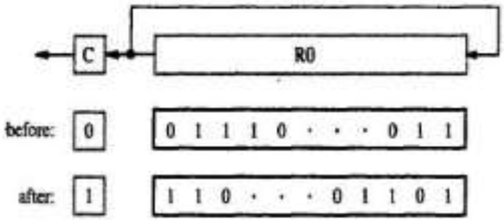
33. List out the various rotate instruction

In addition to the shift instructions, there are also four rotate instructions:

- ROL (Rotate left without the carry flag CF)
- ROR (Rotate right without the carry flag CF)
- RCL (Rotate left including the carry flag CF)
- RCR (Rotate right including the carry flag CF)

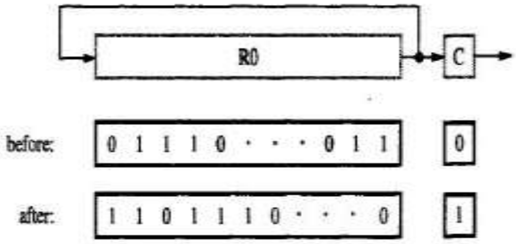
34. What is ROL instruction?

- ROL (rotate) shifts each bit to the left
- The highest bit is copied into both the Carry flag and into the lowest bit
- No bits are lost
- For example , **ROL R0,#2**



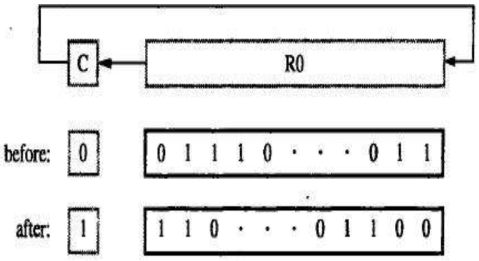
35. What is ROR Instruction?

- ROR (rotate right) shifts each bit to the right
- The lowest bit is copied into both the Carry flag and into the highest bit
- No bits are lost
- For example , **ROR R0,#2**



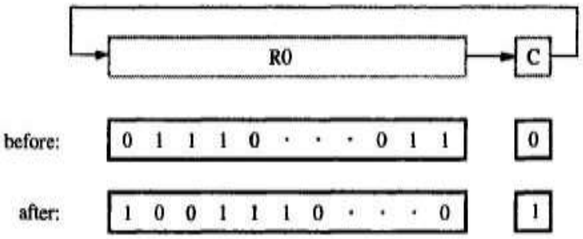
36. What is RCL instruction?

- RCL (rotate carry left) shifts each bit to the left
- Copies the Carry flag to the least significant bit
- Copies the most significant bit to the Carry flag
- For example , **RCL R0,#2**



37. What is RCR instruction?

- RCR (rotate carry right) shifts each bit to the right
- Copies the Carry flag to the most significant bit
- Copies the least significant bit to the Carry flag
- For example , **ROR R0,#2**



38. What is memory mapped I/O?

- Instead of having special methods for accessing the values to be read or written, just get them from memory or put them into memory.
- The device is connected directly to certain main memory locations.

39. Why use memory mapped I/O?

- Makes programming simpler.
- Do not have special commands to access I/O devices.
  - Just use lw and sw.
- Takes some memory locations
  - Very few compared to the size of main memory.

40. What is Isolated I/O?

The IA-32 instructions set also have two instructions, with OP codes IN and OUT, that are used only for I/O purposes. The addresses issued by these instructions are in an address space that is separate from the memory address space used by the other instructions. This arrangement is called isolated I/O .

41. Write short notes on block transfers

The IA-32 architecture also has two block transfer I/O instructions REPINS and REPOUTS .they transfer a block lf data serially, one item at a time, between the memory and an I/O device. The S suffix in the

OP codes stands for string, and the REP prefix stands for “repeat the item by item transfer until the complex block has been transferred”. The instructions themselves do not specify the parameters needed to describe the transfer. These parameters are specified implicitly by processor registers DX,EDI and ECX as follows:

DX contains a 16-bit I/O device address

EDI contains a 32-bit address for the beginning of a block in memory
ECX contains the number of data items to be transferred.

A suffix B or D in the OP-code mnemonic indicates that the item size is either of byte or doubleword length. Thus REPINSB is a byte –block transfer, and REPINSD is a doubleword –block transfer.

42. How block transfer instruction operates?

The block transfer instruction operates as follows: After each data item is transferred, the index register EDI is incremented by 1 or 4 depending on the size of the data items, and the ECX register is decremented by 1.The transfers are repeated until the contents of the counter register ECX have been decremented to 0.The effect of these single instruction is equivalent to a program loop that uses register ECX as the loop counter

43. Write short notes on Subroutines

In the IA-32 architecture, register ESP is used as the stack pointer. It points to the current top element (TOS) in the processor stack. The stack grows toward lower numbered addresses. The width of the stack is 32 bits, that is, all stack entries are double words.

There are two instructions for pushing and popping individual elements onto and off the stack. The instruction

PUSH src

decrements ESP by 4, and then stores the doubleword at location src into the memory location pointed to by ESP. The instruction

POP dst

reverses this process by retrieving the TOS doubleword from the location pointed to by ESP,storing it at location dst, and then incrementing ESP by 4. These instructions implicitly use ESP as the stack pointer. The source and destination operands are specified using the IA-32 addressing modes.

44. List out the other instructions that are available in IA-32

The other instructions available in IA-32 are:

1. Multiplication instruction:
2. Division instruction:
3. Multimedia Extension (MMX) instructions
4. Vector (SIMD) Floating-Point Operations

45. What is Multiplication instruction?

The signed integer multiplication instruction, IMUL, performs 32-bit multiplication. Depending on the form of the instruction that is used, the destination may be implicit and the 64-bit product may be truncated to 32 bits.

One form of this instruction is

IMUL src

which implicitly uses the EAX register as the multiplicand. The multiplier specified by src can be in a register or in the memory. The full 64-bit product is placed in registers EDX (high-order half) and EAX (low-order half).

46. Write about Division instruction:

The integer divide instruction, IDIV, operates on a 64-bit dividend and a 32-bit divisor to generate a 32-bit quotient and a 32-bit remainder. The format of the instruction is

IDIV src

The source operand is the divisor. The 64-bit dividend is formed by the contents of register EDX (high-order half) and register EAX (low-order half). After performing the division, the quotient is placed in EAX and the remainder is placed in EDX.

47. What is MMX instruction?

The IA-32 instruction set has a number of instructions that operate in parallel on such data packed into 64-bit quadwords.(A quadword contains 8 bytes or four 16-bit words).These instructions are called *multimedia extension* (MMX) instructions.

The operands for MMX instructions can be in the memory, or in the eight floating-point registers. Thus, these registers serve a dual purpose. They can hold either floating-point numbers or MMX operands. When used by MMX instructions, the registers are referred to as MM0 through MM7.

48. Write about Vector (SIMD) Floating-Point Operations

A set of instructions that are used to perform arithmetic operations on small group of floating point numbers is provided.SIMD (single-instruction-multiple-data) instructions are useful for vector and matrix calculations in scientific applications.

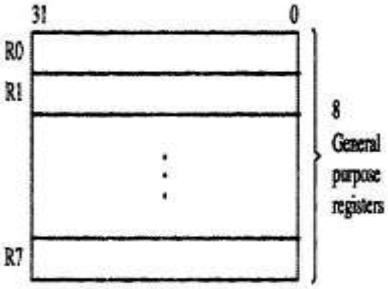
In Intel terminology, these instructions are called streaming SIMD extension (SSE) instructions. They handle packed 128-bit double quadwords, each consisting of four 32-bit floating-point numbers. Eight additional 128-bit registers, XMM0 to XMM7, are available for holding these operands.

IA-32 REGISTERS

IA- 32 Register structure contains the following

- General purpose registers
- floating point registers
- segment registers
- instruction pointer
- status registers

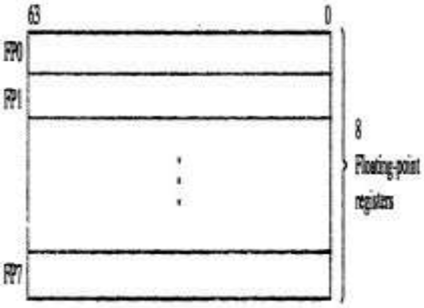
General purpose registers:



The above diagram shows the general purpose registers. The eight 32-bit registers labeled R0 through R7 are general purpose registers that can be used to bold either data operands or addressing information.

Floating point registers:

There are eight floating point registers for holding doubleword or quadword (64 bits) floating point data operands. The floating point registers have an extension field to provide a total length of 80 bits, the extra bits are used for increased accuracy while floating point numbers are operated on in the processor.



Segment registers:

IA-32 architectures are based on a memory model that associates different areas of the memory, called segments, with different usages. There are three segments,

Code segment (CS)

The code segment holds the instructions of a program.

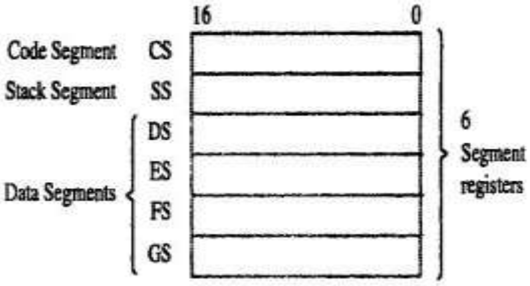
Stack segment (SS)

The stack segment contains the processor stack

Data segment (DS,RD,FS,GS)

Four data segments are provided for holding data operands.

The six segment registers below contain selector values that are used in locating these segments in the memory address space.

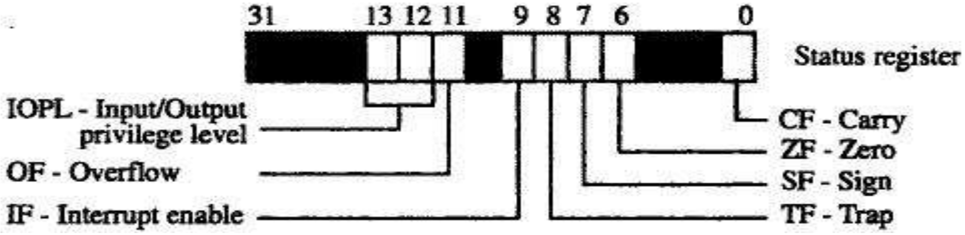


Instruction pointer:

It is a 32-bit register. It serves as the program counter and contains the address of the next instruction to be executed

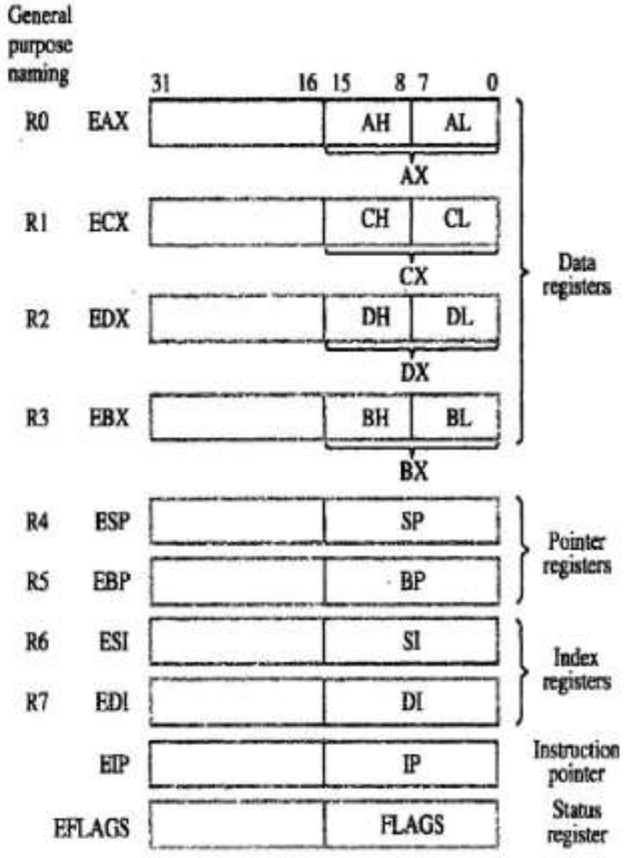
31
0
Instruction pointer

Status registers:



The status register holds the condition code flags (CF, ZF, SF, OF).These flags contain information about the results of arithmetic operations.

Compatibility of the IA-32 register structure:



The eight general purpose registers are grouped into 3 different types

1. Data register
2. Pointer register
3. Index register

The data register used for holding operands ,and the pointer and index registers for holding address and address indices used to determine the effective address of a memory operand.

The above figure shows the compatibility of the IA-32 register structure with earlier Intel processor register structures.

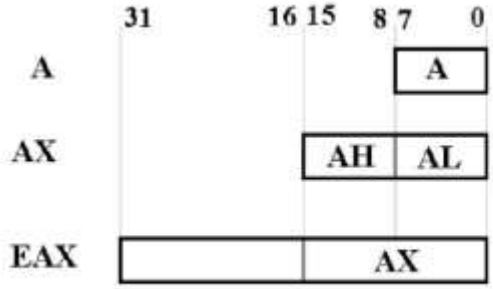
The Intel's original 8-bit processors, the data registers were called A,B,C and D.In Intel 16-bit processors, these registers were labeled AX,BX,CX and DX .the high and low-order bytes in each register are identified by suffixes H and L .For example, the two bytes in register AX are referred to as AH and AL.

In IA-32 processors, the prefix E is used to identify the corresponding "extended "32-bit register :EAX,EBX,ECX and EDX.The E –prefix labeling is also used for the other 32-bit registers shown in the above figure. They are the extended versions of the corresponding 16-bit register used in earlier processors.

The IA-32 processor state can be switched dynamically between 32-bit operation and 16-bit operation during program execution on an instruction by instruction basis by the use of instruction prefix bytes.

EAX register:

This is the general–purpose register used for arithmetic and logical operations. This division is seen also in the EBX, ECX, and EDX registers; the code can reference BX, BH, CX, CL, etc. This register has an implied role in both multiplication and division.



EBX register:

This can be used as a general–purpose register, but was originally designed to be the base register, holding the address of the base of a data structure. The easiest example of such a data structure is a singly dimensioned array.

ECX register:

This can be used as a general–purpose register, but it is often used in its special role as a counter register for loops or bit shifting operations.

EDX register:

This can be used as a general–purpose register. It also plays a special part in executing integer multiplication and division. For multiplication, DX or EDX store the more significant bits of the product. The 16–bit implementation of multiplication uses AX to hold one of the integers to be multiplied and uses the register pair DX:AX to hold the 32-bit product.

The 32–bit implementation of multiplication uses EAX to hold one of the integers to be multiplied and uses the register pair EDX:EAX to hold the 64-bit product.

The Instruction Pointer(EIP)

The **EIP** register is the 32-bit instruction pointer.The EIP register (or instruction pointer) can also be called "program counter." It contains the offset in the current code segment for the next instruction to be executed. It is advanced from one instruction boundary to the next in straight-line code or it is moved ahead or backwards by a number of instructions when executing JMP,CALL, RET instructions.

The Index Registers

The **ESI** and **EDI** registers are used as source and destination addresses for string and array operations.

The ESI "**Extended Source Index**" and EDI "**Extended Destination Index**" facilitate high–speed memory transfers.

The Other Pointers register’s are:

1.ESP register:

The **ESP** register is the 32-bit stack pointer, used to manage push and pop operations.

2.EBP Register:

The **EBP** register is the 32-bit base pointer. In connection with the ESP, the EBP is used to manage the stack frame, that part of the stack used to communicate with subprograms and store local variables.

3.EFLAGS Register:

The **EFLAGS** register holds a collection of at most 32 Boolean flags. The flags are divided into two broad categories: **control flags** and **status flags**.

IA-32 ADDRESSING MODES *(7 Marks April 2015)*

The IA-32 architecture has a large and flexible set of addressing modes. They are designed to access individual data items or data items that are members of an ordered list begins at a specified memory address. There are also several addressing modes that provide mere flexibility in accessing data operands in the memory.

The addressing mode of IA-32 includes:

1. Immediate mode
2. Direct mode
3. Register mode
4. Register indirect mode
5. Base with displacement mode
6. Index with displacement mode
7. Base with index mode
8. Base with index and displacement mode

Immediate mode

The operand is contained in the instruction. It is a signed 8-bit or 32-bit number, with the length being specified by a bit in the OP code of the instruction. Thus bit is 0 for the short version and 1 for the long version.

Instruction format:

Opcode	**Register**	**Value**

Example:

MOV EAX,25

The above instruction moves the decimal value 25 into the EAX register. A number given in this form using the digits 0 through 9 is assumed to be in decimal notation .the suffix B and H are used to specify binary and hexadecimal numbers, respectively .For example the instruction,

MOV EAX,3FAOOH

Moves the hex number 3FA00 into EAX.

Direct mode

The memory address of the operand is given by a 32-bit value in the instruction

Instruction format:

Opcode	**Register**	**Location**

Example:

MOV EAX,LOCATION

The above instruction uses the direct addressing mode to move the doubleword at the memory location specified by the address label LOCATION into register EAX.This assumes that the LOCATION has been defined as an address label for a memory location in the data declaration. If LOCATION represents the address 1000 then this instruction moves the doubleword at 1000 into EAX.

In the IA-32 assembly language square brackets can be used to explicitly indicate the direct addressing mode as in the instruction.

MOV EAX,[LOCATION]

Register mode

The operand is contained in one of the general purpose register specified in the instruction.

Instruction format:

Opcode	Dest.Register	Src.Register

Example:

MOV EAX, ECX

Both operands use register mode. The contents of register **ECX** is copied to register EAX.

Before execution of the above instruction, the contents of ECX and EAX are:

ECX
50

EAX
00

After execution

ECX
50

EAX
50

Register indirect mode

The memory address of the operand is contained in one of the eight general purpose register specified in the instruction.

Instruction format:

Opcode	Register	[Register]

Example:

MOV EAX,[EBX]

The above instruction moves the contents of LOCATION specified by the register EBX into the register EAX.

Before execution of the above instruction, the contents of EAX and EBX are:

EAX
00

EBX
1000

20

1000

After execution

EAX

20

EBX

1000

20

1000

Base with displacement mode

An 8-bit or 32-bit signed displacement and one of the eight general purpose register to be used as a base register are specified in the instruction. The effective address of the operand in the sum of the contents of the base register and the displacement.

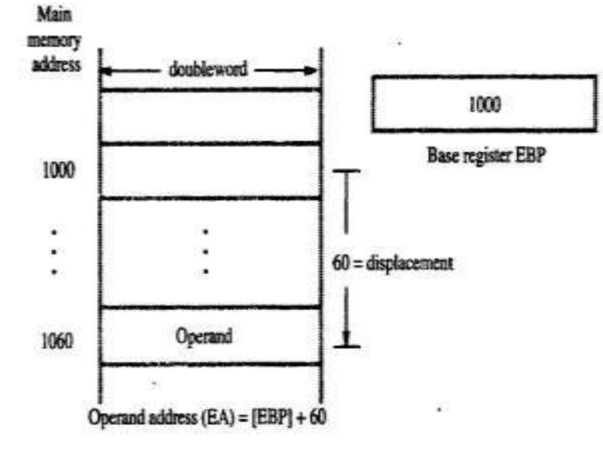
Instruction format:

Opcode	Dest.Register	[Register]+Displacement

Example:

MOV EAX, [EBP + 60]

The second operand uses base displacement mode. The instruction contains a constant. That constant is added to the contents of register EBP to form an effective address. The contents of memory at the effective address are copied into register EAX.



(a) Base with displacement mode, expressed as [EBP + 60]

Index with displacement mode

A 32-bit signed displacement, one of the eight general purpose register to be used as an index register ,and a scale factor of 1,2,4 or 8 are specified in the instruction. To obtain the effective address of the operand, the contents of the index register are multiplied by the scale factor and then added to the displacement.i.e

Offset = (Index * Scale) + displacement

Instruction format:

Opcode	Dest.Register	[Register] * Scale factor +Displacement

Example:

MOV AL,[EBP * 4 + 10]

Base with index mode

Two of the eight general purpose register and a scale factor of 1,2,4 or 8 are specified in the instruction. The registers are used as base and index register and the effective address of the operand is calculated as follows: the contents of the index register are multiplied by the scale factor and added to the contents of the base register.i.e

Offset = Base + (Index * Scale)

Instruction format:

Opcode	Dest.Register	[Register1] + [Register2] * Scale factor

Example:

MOV EAX, [ESP+ESI*4]

The contents of registers ESI is multiplied with the scale factor 4 and then the content of ESP is added to form an effective address. The contents of memory at the effective address are copied into register EAX

Base with index and displacement mode

An 8 bit or 32 –bit signed displacement two of the eight general purpose registers and a scale factor of 1,2,4 or 8 are specified in the instruction. The register is used as base and index register and the effective address of the operand is calculated as follows: the contents of the index register are multiplied by the scale factor and then added to the contents of the base register and the displacement.

An effective address is computed by:

Offset = Base + (Index * Scale) + displacement

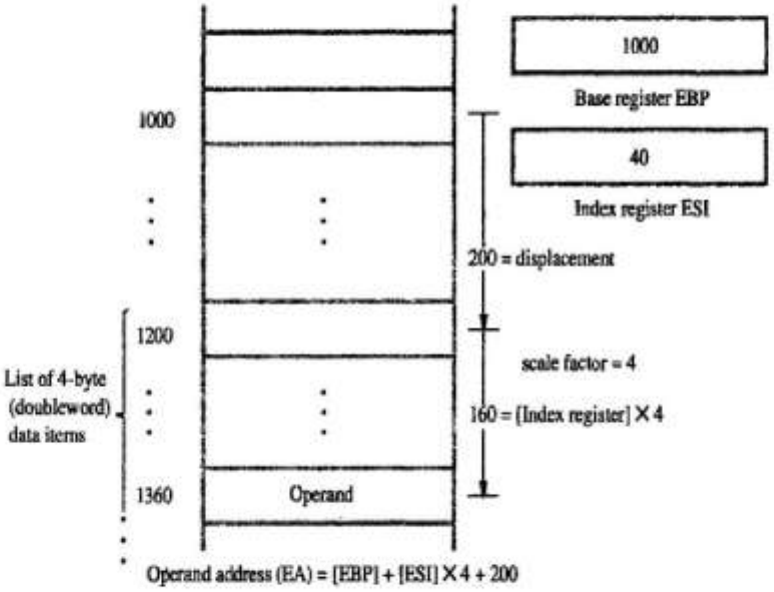
Instruction format:

Opcode	Dest.Register	[Register1] + [Register2] * Scale factor + Displacement

Example:

MOV EAX, [EBP+ESI*4 + 200]

The contents of registers ESI is multiplied with the scale factor 4 and then the content of EBP and displacement are added to form an effective address. The contents of memory at the effective address are copied into register EAX


1000
Base register EBP
1000
40
Index register ESI
200 = displacement
1200
scale factor = 4
List of 4-byte (doubleword) data items
160 = [Index register] X 4
1360
Operand
Operand address (EA) = [EBP] + [ESI] X 4 + 200

(b) Base with displacement and index mode, expressed as [EBP + ESI * 4 + 200]

The below table shows the various addressing modes of IA-32 processor:

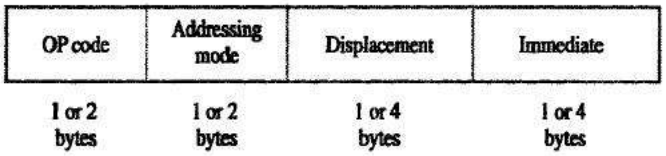
S.No	Name	Assembler syntax	Addressing function
1	Immediate	Value	Operand = Valur
2	Direct	Location	EA = Location
3	Register	Reg	EA = Reg that is Operand=Reg
4	Register indirect	[Reg]	EA = [Reg]
5	Base with displacement	[Reg + Disp]	EA = [Reg] + Disp
6	Index with displacement	[Reg * S + Disp]	EA = [Reg] * S + Disp
7	Base with index	[Reg1 + Reg2 * S]	EA = [Reg1] + [Reg2] * S
8	Base with index and displacement	[Reg1 + Reg2 * S + Disp]	EA = [Reg1] + [Reg2] * S + Disp

Where,

Value	an 8 or 32 – bit signed number
Location	a 32-bit address
Reg,Reg1,Reg2	one of the general purpose registers EAX,EBX,ECX,EDX,ESP,EBP,ESI,EDI,with the exception that ESP cannot be used as an index register.
Disp	an 8 or 32 – bit signed number except that in the index with displacement mode it can only be 32 bits.
S	a scale factor of 1,2,4 or 8

IA-32 INSTRUCTIONS *(11 Marks Nov 2014, 5 Marks April 2015)*

IA-32 Instruction format



OP code	Addressing mode	Displacement	Immediate
1 or 2 bytes	1 or 2 bytes	1 or 4 bytes	1 or 4 bytes

The instructions are variable in length ,ranging from one byte to 12 bytes ,consisting of up to four fields .the OP-code field consists of one or two bytes, with most instructions requiring only one byte, the addressing mode information is contained in one or two bytes immediately following the OP code. For instructions that involve the use of only one register in generating the effective address of an operand ,only one byte is needed in the addressing mode field. Two bytes are needed for encoding the last two addressing modes. These modes use two registers to generate the effective address of a memory operand.

If a displacement value is needed in computing an effective address for a memory operand, it is encoded into either one or four bytes in a field that immediately follows the addressing mode field. If one of the operand is an immediate value, then it is placed in the last field of and instruction and it occupies either one or four bytes.

IA-32 instructions have either one or two operands. In the two –operand case, only one of the operands can be in the memory. The other must be in a process register. In addition to the usual instructions for moving data between the memory and the processor register, and for performing arithmetic operations, the instruction set includes a number of different logical and shift/rotate operations on data. Byte string instructions are included for nonnumeric data processing. Push and Pop operations for manipulating the processor stack are directly supported in the instruction set.

The instruction

ADD dst,src

Performs the operation

$$\text{dst} \leftarrow [\text{dst}]+[\text{src}]$$

And

MOV dst,src

Performs the operation

$$\text{dst} \leftarrow [\text{src}]$$

One byte instructions

Registers can be incremented or decremented by instructions,that occupy one boe byte.examples are

INC EDI

And

DEC ECX

In which the general purpose register EDI and ECX are specified by 3-bit codes in the single OP-code byte.

Immediate mode encoding

The OP-code specifies when the immediate addressing mode is used. For example the instruction

MOV EAX,820

Is encoded into 5 bytes .a one –byte OP code specifies the move operation, the fact that a 32-bit immediate operand is used and the name of the destination register. The OP code byte is directly followed by the 4-byte immediate value of 820.when an 8-bit immediate operand is used, as in the instruction

MOV DL,5

Only two bytes are needed to encode the instruction.

Addressing mode and displacement fields

As a general; rule, one operand of a two-operand instruction must be in a register .the other operand can also be in a register, or it can be in the memory. There are two exceptions where both operands can be in the memory. The first is the case where the source operand is an immediate operand, and the destination operand is in the memory. The second is the case of instruction for Push and Pop operations on the processor stack. The stack is located in the stack segment of memory, and it is possible to push a memory operand onto the stack or to pop an operand from the stack into the memory.

When both operands are in registers, only one addressing mode byte is needed. For example ,the instruction

ADD EAX,EDX

is encoded into two bytes. The first byte contains the OP code and the other byte is an addressing mode byte that specifies the two registers. The instruction

MOV ECX,N

is encoded in 6 bytes: one for the OP code, one for the addressing mode byte that specifies both the Direct mode and the destination register ECX,and four bytes for the address of memory location N.

The instruction

ADD EAX,[EBX+EDI*4]

requires two addressing mode bytes because two registers are used to generate the effective address of the source operand. The scale factor of 4 is also included in the second of these two bytes. Thus, the instruction requires a total of 3 bytes, including the OP-code byte.

In the encoding of two-operand instructions, the specification of the register operand and the memory operand are placed in a fixed order, with the register operand always, being specified first. In order to distinguish between the instructions

MOV EAX,LOCATION

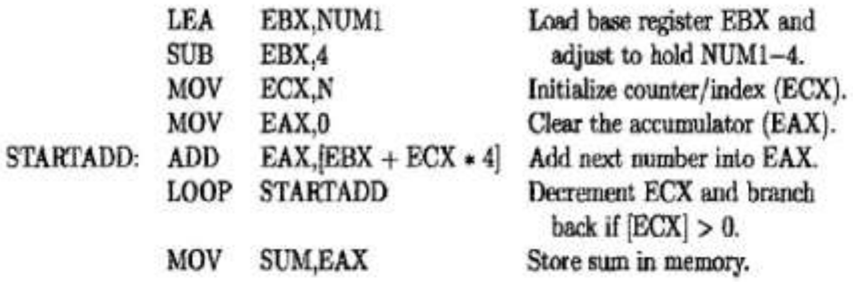
Which loads the contents of memory location LOCATION into register EAX ,and the instruction

MOV LOCATION,EAX

Which stores the contents of EAX into LOCATION ,the OP-code byte contains a bit called the direction bit. This bit indicates which operand is the source.

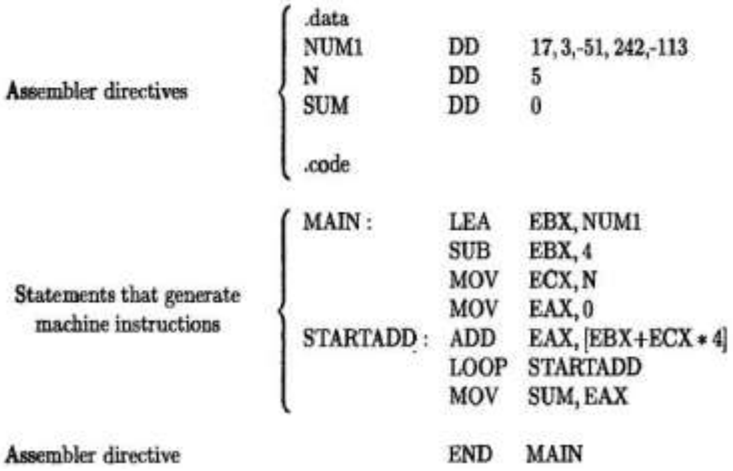
IA-32 ASSEMBLY LANGUAGE *(April 2015 6 Marks, 4 Marks April 2015)*

Basic aspects of the IA-32 assembly language for specifying OP code, addressing modes and instruction address labels are shown in the below program:



```
            LEA    EBX,NUM1              Load base register EBX and
            SUB    EBX,4                   adjust to hold NUM1−4.
            MOV    ECX,N                 Initialize counter/index (ECX).
            MOV    EAX,0                 Clear the accumulator (EAX).
STARTADD:   ADD    EAX,[EBX + ECX * 4]   Add next number into EAX.
            LOOP   STARTADD              Decrement ECX and branch
                                           back if [ECX] > 0.
            MOV    SUM,EAX               Store sum in memory.
```

The assembler directives are needed to define the data area of a program and to define the correspondence between symbolic names for data locations and the actual physical address values. A complete assembly language program is shown below:



```
                         .data
                         NUM1        DD     17,3,-51,242,-113
Assembler directives     N           DD     5
                         SUM         DD     0

                         .code

                         MAIN :      LEA    EBX, NUM1
                                     SUB    EBX, 4
                                     MOV    ECX, N
Statements that generate             MOV    EAX, 0
machine instructions     STARTADD :  ADD    EAX, [EBX+ECX * 4]
                                     LOOP   STARTADD
                                     MOV    SUM, EAX

Assembler directive                  END    MAIN
```

The data and code assembler directives define the beginning og the data and code sections of the program .In the data section ,the DD directives allocate 4-byte doublewords initialized to the decimal values 17,3,-51,242 and -113.the next two doubleword locations ,initialized to 5 and 0 ,are given the address labels N and SUM.

The 3 symbolic names declared in the data section are used in the addressing modes of the instruction in the code section. The MAIN label is used to specify the location where instruction execution is to begin, and this label is used in the END assembler directive that terminates the text file for the program.

IA-32 PROGRAM FLOW CONTROL

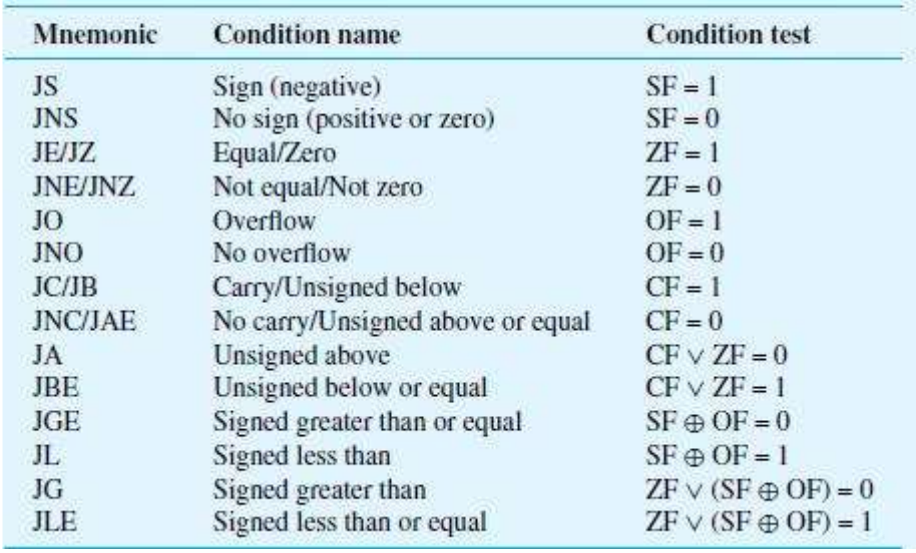
There are two main ways in which the flow of executing instructions varies from straight –line sequencing, calls to subroutines and returns from them break straight line sequencing. Also ,branch instructions, either conditional or unconditional, can cause a break. The branch instructions are called jumps.

Conditional jumps and condition code flags

The conditional Jump instructions test the four condition code flags in the status register. The instruction

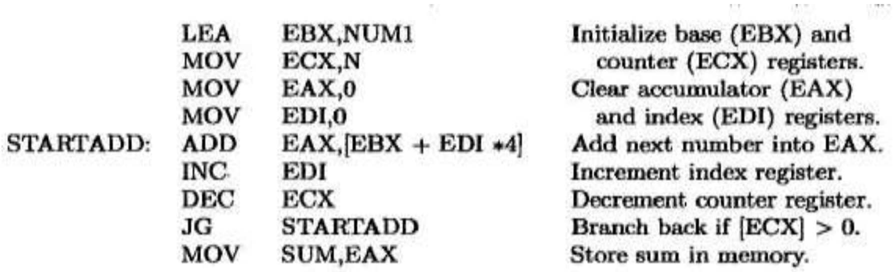
JG LABEL

is an example of a conditional Jump instruction. The condition is *greater-than* as indicated by the G suffix in the OP code. The below table summarizes the conditional Jump instructions and the corresponding combinations of the condition code flags that are tested. The Jump instructions that test the sign flag (SF) are used when the operands of a preceding arithmetic or comparison instruction are signed numbers. For example, the JG instruction tests for the greater-than condition when signed numbers are involved, and it considers the SF flag. When unsigned numbers are involved, the JA (jump-above) instruction tests for the greater than condition without considering the SF flag.



Mnemonic	Condition name	Condition test
JS	Sign (negative)	SF = 1
JNS	No sign (positive or zero)	SF = 0
JE/JZ	Equal/Zero	ZF = 1
JNE/JNZ	Not equal/Not zero	ZF = 0
JO	Overflow	OF = 1
JNO	No overflow	OF = 0
JC/JB	Carry/Unsigned below	CF = 1
JNC/JAE	No carry/Unsigned above or equal	CF = 0
JA	Unsigned above	$CF \vee ZF = 0$
JBE	Unsigned below or equal	$CF \vee ZF = 1$
JGE	Signed greater than or equal	$SF \oplus OF = 0$
JL	Signed less than	$SF \oplus OF = 1$
JG	Signed greater than	$ZF \vee (SF \oplus OF) = 0$
JLE	Signed less than or equal	$ZF \vee (SF \oplus OF) = 1$

For example, program for adding numbers



```
           LEA    EBX,NUM1              Initialize base (EBX) and
           MOV    ECX,N                   counter (ECX) registers.
           MOV    EAX,0                 Clear accumulator (EAX)
           MOV    EDI,0                   and index (EDI) registers.
STARTADD:  ADD    EAX,[EBX + EDI *4]    Add next number into EAX.
           INC    EDI                   Increment index register.
           DEC    ECX                   Decrement counter register.
           JG     STARTADD              Branch back if [ECX] > 0.
           MOV    SUM,EAX               Store sum in memory.
```

the instruction

JG STARTADD

is an example of jump instruction. The condition is “Greater than 0”is tested ,if it is true then the control transfers to STARTADD otherwise control transferred to the next statement which is coming after JG STARTADD.

Unconditional Jump Instruction

An unconditional Jump instruction, JMP, causes a branch to the instruction at the target address. In addition to using short (one-byte) or long (four-byte) relative signed offsets to determine the target address, as is done in conditional Jump instructions, the JMP instruction also allows the use of other addressing modes. This flexibility in generating the target address can be very useful. Consider the Case statement that is found in many high-level languages. It is used to perform one of a number of alternative computations at some point in a program. Each of these alternatives is referred to as a case. Suppose that for each case, a routine is defined to perform the corresponding computation. Suppose also that the 4-byte starting addresses of the routines are stored in a table in the memory, beginning at a location labeled JUMPTABLE. The cases are numbered with indices 0, 1, 2, ... At execution time, the index of the selected case is loaded into index register ESI. A jump to the routine for the selected case is performed by executing the instruction

JMP [JUMPTABLE + ESI * 4]

which uses the Index with displacement addressing mode.

Compare instructions

It is often necessary to make conditional jumps in a program based on the results of comparing two numbers. The compare instruction

CMP dst,src

performs the operation

$$\text{dst} \leftarrow [\text{src}]$$

and sets the condition code flags based on the result obtained. Neither of the operands is changed .the first operand is always compared to the second. For example, the compare instruction by a conditional jump that is based on the “greater than” condition, then the jump will take to the target address if the destination operand is greater than the source operand.

LOGIC AND SHIFT/ROTATE INSTRUCTIONS

Logic operations:

The IA-32 architecture has instructions that perform the logic operations AND, OR, and XOR. The operation is performed bitwise on two operands, and the result is placed in the destination location.

For example, suppose register EAX contains the hexadecimal pattern 0000FFFF and register EBX contains the pattern 02FA62CA. The instruction

AND EBX, EAX

clears the left half of EBX to all zeroes, and leaves the right half unchanged. The result in EBX will be 000062CA.There is also a NOT instruction which generates the logical complement of all bits of the operand, that is, it changes all 1s to 0s and all 0s to 1s.

Shift instruction:

An operand can be shifted right or left ,using either logical or arithmetic shifts,by a number of bit positions determined by a specified count. The format of the shift instruction is

OPcode dst , count

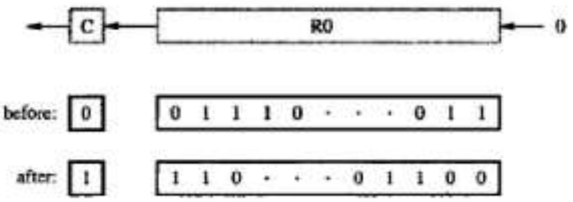
Where the destination operand to be shifted is specified by the general addressing modes and the count is given either as an 8-bit immediate value or is contained in the 8-bit register CL.

There are four shift instructions:

- SHL (Shift left logical)
- SHR (Shift right logical)
- SAL (Shift left arithmetic; operation is identical to SHL)
- SAR (Shift right arithmetic)

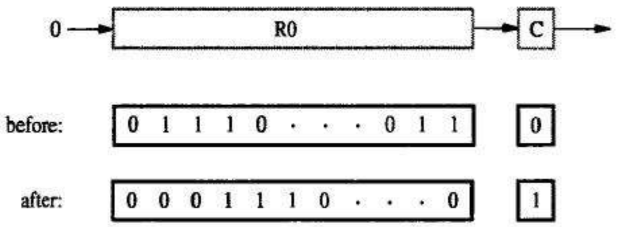
1. SHL (Shift left logical)

The SHL (shift left) instruction performs a logical left shift on the destination operand, filling the lowest bit with 0.For example, **SHL R0,#2**



2. SHR (Shift right logical)

The SHR (shift right) instruction performs a logical right shift on the destination operand. The highest bit position is filled with a zero. For example, **SHR R0,#2**

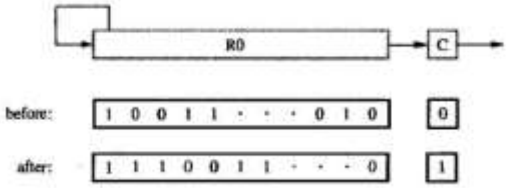


3. SAL (Shift left arithmetic)

The operation of SAL is identical to SHL.

4. SAR (Shift right arithmetic)

SAR (shift arithmetic right) performs a right arithmetic shift on the destination operand. For example, **SAR R0,#2**



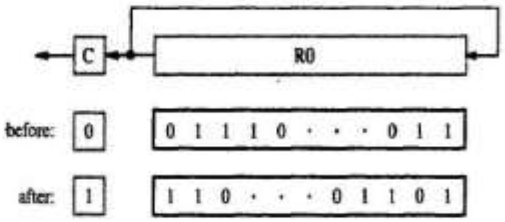
Rotate instruction:

In addition to the shift instructions, there are also four rotate instructions:

- ROL (Rotate left without the carry flag CF)
- ROR (Rotate right without the carry flag CF)
- RCL (Rotate left including the carry flag CF)
- RCR (Rotate right including the carry flag CF)

1. ROL (Rotate left without the carry flag CF)

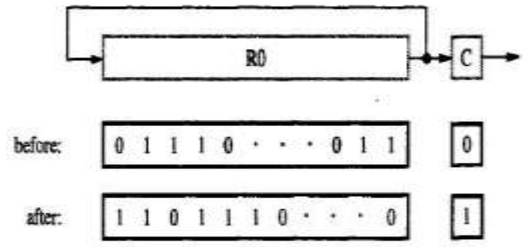
- ROL (rotate) shifts each bit to the left
- The highest bit is copied into both the Carry flag and into the lowest bit
- No bits are lost
- For example , **ROL R0,#2**



2. ROR (Rotate right without the carry flag CF)

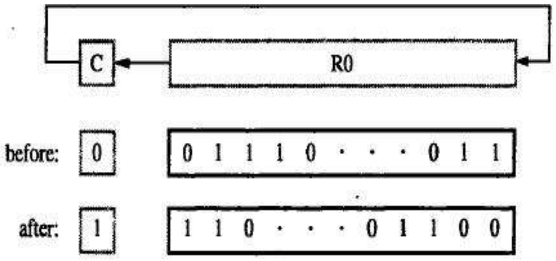
- ROR (rotate right) shifts each bit to the right

- The lowest bit is copied into both the Carry flag and into the highest bit
- No bits are lost
- For example , **ROR R0,#2**



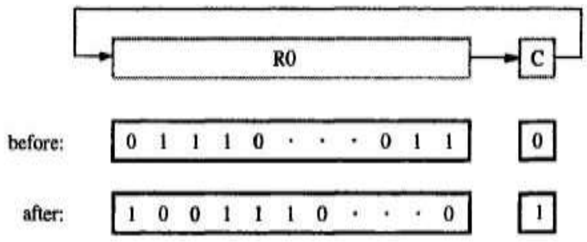
3. RCL (Rotate left including the carry flag CF)

- RCL (rotate carry left) shifts each bit to the left
- Copies the Carry flag to the least significant bit
- Copies the most significant bit to the Carry flag
- For example , **RCL R0,#2**



4. RCR (Rotate right including the carry flag CF)

- RCR (rotate carry right) shifts each bit to the right
- Copies the Carry flag to the most significant bit
- Copies the least significant bit to the Carry flag
- For example , **ROR R0,#2**



The rotate instructions require the count argument to be either an 8-bit immediate value or the 8-bit contents of register CL.

I/O OPERATIONS IN IA-32 (11 Marks Nov 2014)

Memory mapped I/O

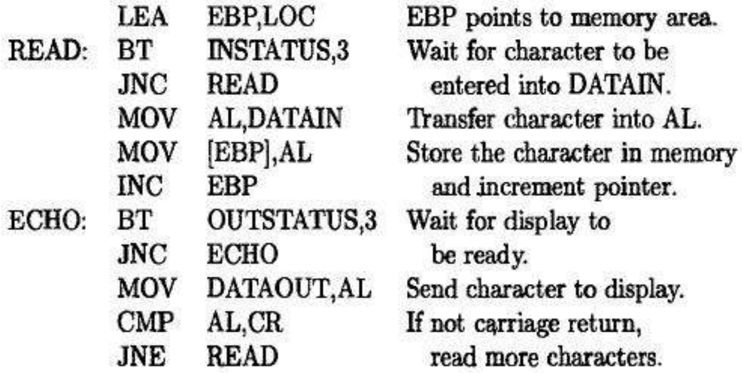
The IA-32 Move instruction can be used to transfer directives to I/O devices ,and to transfer data and status information to and from devices. For example, suppose that keyboard and display devices have their synchronization flags SIN and SOUT stored in bit 3 of device status registers INSTATUS and OUTSTATUS, respectively. Using program controlled I/O; a byte can be read from the keyboard buffer register DATAIN into register AL using the wait loop

```
READWAIT:        BT INSTATUS,3
                 INC READWAIT
                 MOV ALDATAIN
```

The instruction BT is a bit-test instruction .the value in the destination bit position specified by the source operand is loaded into the carry flag CF .the conditional jump JNC causes a jump to READWAIT if CF = 0. Similarly, an output operation to send a byte from register AL to the display buffer register DATAOUT is performed by

```
WRITEWAIT:       BT OUTSTATUS,3
                 INC WRITEWAIT
                 MOV DATAOUT,AL
```

An IA-32 program that reads one line of character from a keyboard, stores them in memory starting at address LOC ,and echo's them back out to the display shown below



```
        LEA   EBP,LOC        EBP points to memory area.
READ:   BT    INSTATUS,3     Wait for character to be
        JNC   READ             entered into DATAIN.
        MOV   AL,DATAIN      Transfer character into AL.
        MOV   [EBP],AL       Store the character in memory
        INC   EBP              and increment pointer.
ECHO:   BT    OUTSTATUS,3    Wait for display to
        JNC   ECHO             be ready.
        MOV   DATAOUT,AL     Send character to display.
        CMP   AL,CR          If not carriage return,
        JNE   READ             read more characters.
```

Isolated I/O

The IA-32 instructions set also have two instructions, with OP codes IN and OUT, that are used only for I/O purposes. The addresses issued by these instructions are in an address space that is separate from the memory address space used by the other instructions. This arrangement is called isolated I/O to distinguish it from memory –mapped I/O in which the addressable locations in I/O devices are in the same address space as memory locations. The same address and data lines on Intel processor chips are used for both address spaces. An output control line is used to indicate which address space is reference by the current instruction.

The 16-bit addresses are used in the byte-addressable I/O address space. There are 8-bit and 32-bit I/O device operand location that hold data ,status, and command information. The first 256 addresses can be specified directly using an 8-bit field in the In and Out instructions. The format for the input instruction using this mode is

IN REG ,DEVADDR

Where the destination REG must be register AL or EAX,denoting an 8-bit or a 32-bit operand transfer respectively. The last field in the instruction contains the 8-bit device address DEVADDR.the corresponding output instruction is

OUT DEVADDR,REG

Since the address space is byte addressable, a keyboard device that can send an 8-bit ASCII character to the processor could have its data buffer register at byte address DEVADDR and its 8-bit starts register at address

DEVADDR + 1

The full 16-bit I/O address spans 64K locations; it can be referenced through the DX register using the input instruction

IN REG,DX

Where, as before,REG must be AL or EAX.The 16-bit device address is contained in the DX register ,which is low order 16 bits of the EDX register, and the width of the data transfer is determined by the size of the REG operand. The corresponding output instruction os

OUT DX,REG

Block transfers

In addition to the instruction IN and OUT that transfer a single item of information between the processor and an I/O device ,the IA-32 architecture also has two block transfer I/O instructions REPINS and REPOUTS .they transfer a block lf data serially, one item at a time, between the memory and an I/O device. The S suffix in the OP codes stands for string, and the REP prefix stands for “repeat the item by item transfer until the complex block has been transferred”. The instructions themselves do not specify the parameters needed to describe the transfer. These parameters are specified implicitly by processor registers DX,EDI and ECX as follows:

DX contains a 16-bit I/O device address

EDI contains a 32-bit address for the beginning of a block in memory
ECX contains the number of data items to be transferred.

A suffix B or D in the OP-code mnemonic indicates that the item size is either of byte or doubleword length. Thus REPINSB is a byte –block transfer, and REPINSD is a doubleword –block transfer.

The block transfer instruction operates as follows :After each data item is transferred ,the index register EDI is incremented by 1 or 4 depending on the size of the data items, and the ECX register is decremented by 1.The transfers are repeated until the contents of the counter register ECX have been decremented to 0.The effect of these single instruction is equivalent to a program loop that uses register ECX as the loop counter.

As an example, suppose that a block lf 128 doublewords is to be transferred from a disk storage device into the memory. Assume that the addressable data buffer register in the disk device contains a doubleword data item, and has the I/O address MEMBLOCK.The counter register ECX has to be initialized to 128.The instruction sequence

```
LEA        EDI,MEMBLOCK
MOV        DX,DISKDATA
MOV        ECX,128
REPINSD
```

can be used to accomplish the transfer. This assumes that MEMBLOCK has been defined as an address label, and DISKDATA has been defined by an EQU assembler directive to represent the 16-bit address of the device data buffer register.

I

In the IA-32 architecture, register ESP is used as the stack pointer. It points to the current top element (TOS) in the processor stack. The stack grows toward lower numbered addresses. The width of the stack is 32 bits, that is, all stack entries are doublewords.

There are two instructions for pushing and popping individual elements onto and off the stack. The instruction

PUSH src

decrements ESP by 4, and then stores the doubleword at location src into the memory location pointed to by ESP. The instruction

POP dst

reverses this process by retrieving the TOS doubleword from the location pointed to by ESP,storing it at location dst, and then incrementing ESP by 4. These instructions implicitly use ESP as the stack pointer. The source and destination operands are specified using the IA-32 addressing modes.

There are also two more instructions that push or pop the contents of multiple registers. The instruction

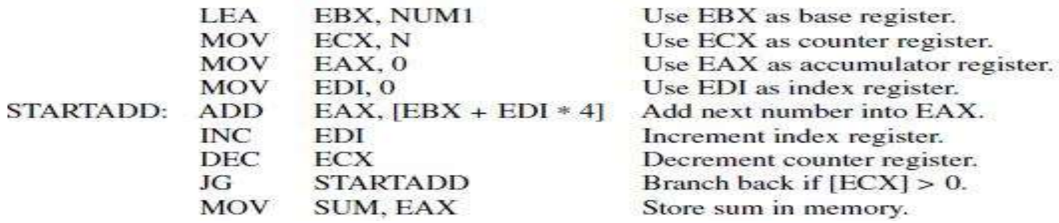
PUSHAD

pushes the contents of all eight general-purpose registers EAX through EDI onto the stack,and the instruction

POPAD

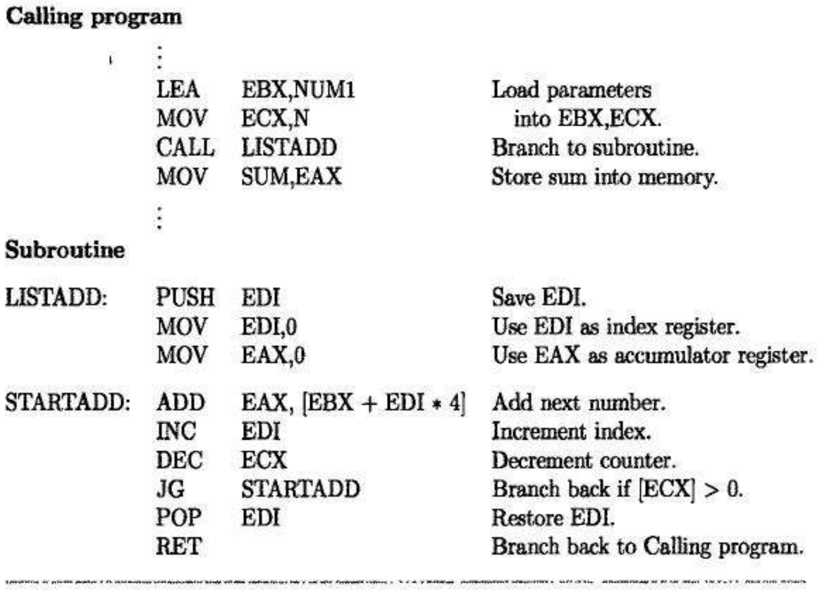
pops them off in the reverse order. When POPAD reaches the old stored value of ESP, it discards those four bytes without loading them into ESP and continues to pop the remaining values into their respective registers. These two instructions are used to efficiently save and restore the contents of all registers as part of implementing subroutines.

For example, The list-addition program shown below



```
           LEA    EBX, NUM1              Use EBX as base register.
           MOV    ECX, N                 Use ECX as counter register.
           MOV    EAX, 0                 Use EAX as accumulator register.
           MOV    EDI, 0                 Use EDI as index register.
STARTADD:  ADD    EAX, [EBX + EDI * 4]   Add next number into EAX.
           INC    EDI                    Increment index register.
           DEC    ECX                    Decrement counter register.
           JG     STARTADD               Branch back if [ECX] > 0.
           MOV    SUM, EAX               Store sum in memory.
```

can be written as a subroutine as shown below:



```
Calling program

            :
            LEA    EBX,NUM1               Load parameters
            MOV    ECX,N                    into EBX,ECX.
            CALL   LISTADD                Branch to subroutine.
            MOV    SUM,EAX                Store sum into memory.
            :
Subroutine

LISTADD:    PUSH   EDI                    Save EDI.
            MOV    EDI,0                  Use EDI as index register.
            MOV    EAX,0                  Use EAX as accumulator register.

STARTADD:   ADD    EAX, [EBX + EDI * 4]   Add next number.
            INC    EDI                    Increment index.
            DEC    ECX                    Decrement counter.
            JG     STARTADD               Branch back if [ECX] > 0.
            POP    EDI                    Restore EDI.
            RET                           Branch back to Calling program.
```

(a) Calling program and subroutine

Parameters are passed through registers. Memory address NUM1 of the first number in the list is loaded into register EBX by the calling program.

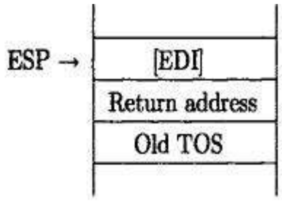
The number of entries in the list, contained in memory location N, is loaded into register ECX. The calling program expects to get the final sum passed back to it in register EAX. Thus, registers EBX,ECX, and EAX are used for passing parameters.

Register EDI is used by the subroutine as an index register in performing the addition, so its contents have to be saved and restored in the subroutine by PUSH and POP instructions.

The subroutine is called by the instruction

CALL LISTADD

which first pushes the return address onto the stack and then jumps to LISTADD. The return address is the address of the MOV instruction that immediately follows the CALL instruction. The subroutine saves the contents of register EDI on the stack.



(b) Stack contents after saving EDI in subroutine

The above figure shows the stack contents at this point. After executing the loop, the saved contents of register EDI are restored. The instruction RET returns execution control to the calling program by popping the TOS element into the Instruction Pointer (register EIP).

OTHER INSTRUCTIONS IN IA-32

The other instructions available in IA-32 are:

1. Multiplication instruction:
2. Division instruction:
3. Multimedia Extension (MMX) instructions
4. Vector (SIMD) Floating-Point Operations

1. Multiplication instruction:

The signed integer multiplication instruction, IMUL, performs 32-bit multiplication. Depending on the form of the instruction that is used, the destination may be implicit and the 64-bit product may be truncated to 32 bits.

One form of this instruction is

IMUL src

which implicitly uses the EAX register as the multiplicand. The multiplier specified by src can be in a register or in the memory. The full 64-bit product is placed in registers EDX (high-order half) and EAX (low-order half).

A second form of this instruction is

IMUL REG, src

The destination operand, REG, must be one of the eight general-purpose registers. The source operand can be in a register or in the memory. The product is truncated to 32 bits before it is placed in the destination register REG.

For both forms, the CF and OF flags are set if there are any 1s (including sign bits) in the high-order half of the 64-bit product. Otherwise, the CF and OF flags are cleared. The other flags are undefined.

2. Division instruction:

The integer divide instruction, IDIV, operates on a 64-bit dividend and a 32-bit divisor to generate a 32-bit quotient and a 32-bit remainder. The format of the instruction is

IDIV src

The source operand is the divisor. The 64-bit dividend is formed by the contents of register EDX (high-order half) and register EAX (low-order half). After performing the division, the quotient is placed in EAX and the remainder is placed in EDX.

All of the condition code flags are undefined. Division by zero causes an exception. If the dividend value is represented by 32 bits, it must first be placed in EAX, and then sign-extended to the required 64-bit operand size in registers EAX and EDX. This is done by the instruction CDQ (convert doubleword to uadword), which has no operands because the source and destination are implicitly registers EAX and EDX, respectively

3. Multimedia Extension (MMX) instructions

A two-dimensional graphic or video image can be represented by a large array of sampled image points, called *pixels*. The color and brightness of each point can be encoded into an 8-bit data item. Processing of such data has two main characteristics.

The first is that manipulations of individual pixels often involve very simple arithmetic or logic operations.

The second is that very high computational performance is needed for some real-time display applications. The same characteristics apply to sampled audio signals or speech processing, where a sequence of signed numbers represents samples of a continuous analog signal taken at periodic intervals.

In such applications, processing efficiency is achieved if the individual data items, which are usually bytes or 16-bit words, are packed into small groups whose elements can be processed in parallel.

The IA-32 instruction set has a number of instructions that operate in parallel on such data packed into 64-bit quadwords.(A quadword contains 8 bytes or four 16-bit words).These instructions are called *multimedia extension* (MMX) instructions.

The operands for MMX instructions can be in the memory, or in the eight floating-point registers. Thus, these registers serve a dual purpose. They can hold either floating-point numbers or MMX operands. When used by MMX instructions, the registers are referred to as MM0 through MM7.

The MOVE instruction is provided for transferring 64-bit quadword operands between the memory and the MMX registers. The instruction

PADDB MMi,src

Adds the corresponding bytes of two 8-byte operands individually and places eight sums in the destination register. The source can be in the memory or in an MMX register but the destination must be an MMX register. The instruction

PADDB MM2, [EBX]

adds eight corresponding bytes of the quadwords in register MM2 and in the memory location pointed to by register EBX. The eight sums are computed in parallel. The results are placed in register MM2.

4. Vector (SIMD) Floating-Point Operations

A set of instructions that are used to perform arithmetic operations on small group of floating point numbers is provided.SIMD (single-instruction-multiple-data) instructions are useful for vector and matrix calculations in scientific applications.

In Intel terminology, these instructions are called *streaming SIMD extension* (SSE) instructions. They handle packed 128-bit double quadwords, each consisting of four 32-bit floating-point numbers. Eight additional 128-bit registers, XMM0 to XMM7, are available for holding these operands.

Add and multiply are two of the basic instructions are provided in this group .they operate on the four corresponding pairs of 32-bit operands in the 128-bit compound source and destination operands and place the four individual results in the 128-bit destination location.

UNIT III - INPUT/OUTPUT ORGANIZATION: Accessing I/O Devices, Interrupts, Interrupt Hardware, Enabling and Disabling Interrupts, Handling Multiple Devices, Controlling Device Requests, Exceptions, Use Of Interrupts in Operating Systems, Pentium Interrupt Structure, Direct Memory Access, Busses, Interface Circuits, Standard I/O

ACCESSING I/O DEVICES

A simple arrangement to connect I/O devices to a computer is to use a single bus structure. It consists of three sets of lines to carry

- Address
- Data
- Control Signals.

When the processor places a particular address on address lines, the devices that recognize this address responds to the command issued on the control lines.

The processor request either a read or write operation and the requested data are transferred over the data lines.

When I/O devices & memory share the same address space, the arrangement is called **memory mapped I/O.**

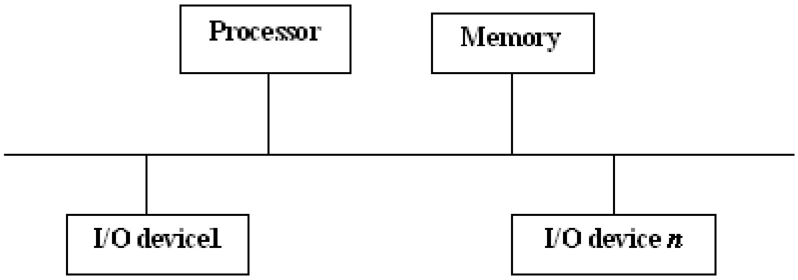
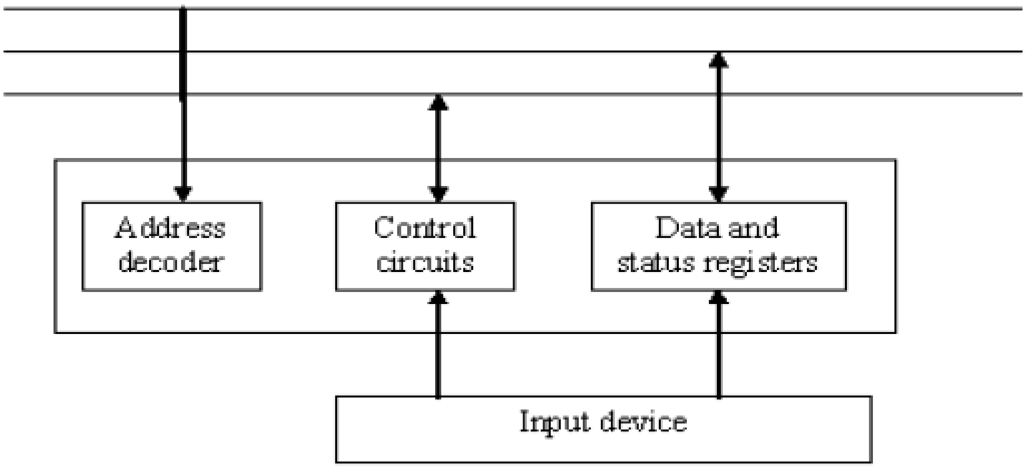


Fig 3.1A single-bus structure

Eg:- Move DATAIN, R_0 → Reads the data from DATAIN then into processor register R_0.
Move R_0, DATAOUT → Send the contents of register Ro to location DATAOUT.
DATAIN → Input buffer associated with keyboard.
DATAOUT → Output data buffer of a display unit / printer.

Fig: I/O Interface for an Input Device

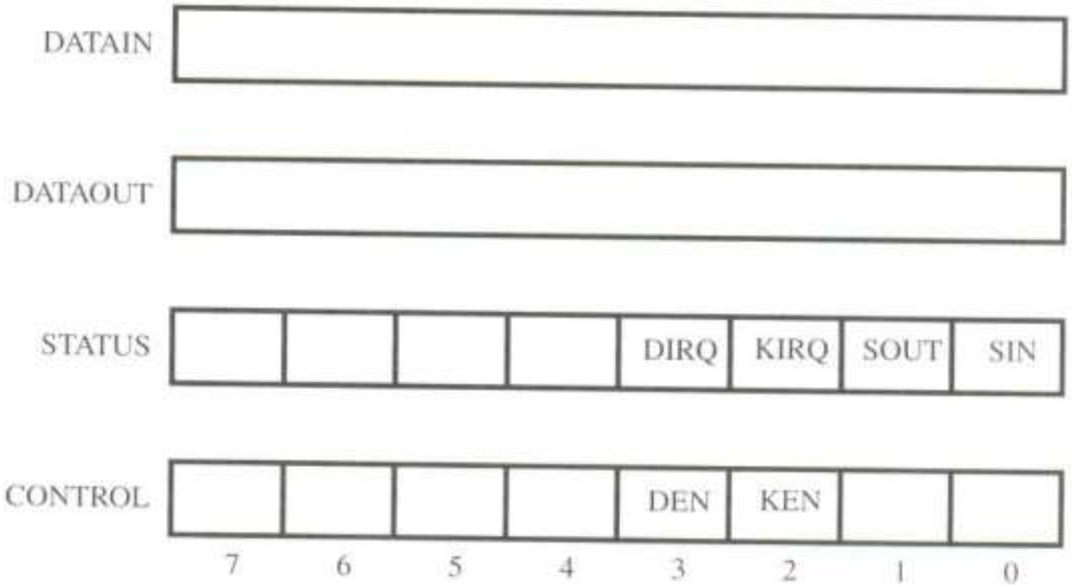


Address Decoder:

- It enables the device to recognize its address when the address appears on address lines.

  Data register → It holds the data being transferred to or from the processor.
  Status register → It contains infn/. Relevant to the operation of the I/O devices.

- The address decoder, data & status registers and the control circuitry required to co-ordinate I/O transfers constitute the device's I/F circuit.
- For an input device, SIN status flag in used SIN = 1, when a character is entered at the keyboard.
- For an output device, SOUT status flag is used SIN = 0, once the char is read by processor.



DIR Q → Interrupt Request for display.
KIR Q →Interrupt Request for keyboard.
KEN → keyboard enable.
DEN → Display Enable.
SIN, SOUT → status flags.

The data from the keyboard are made available in the DATAIN register & the data sent to the display are stored in DATAOUT register

```
WAIT K Move # Line, Ro
Test Bit #0, STATUS
Branch = 0 WAIT K
Move DATAIN, R1
WAIT D Test Bit #1, STATUS
Branch = 0 WAIT D
Move R1, DATAOUT
Move R1, (Ro)+
Compare #OD, R1
Branch = 0 WAIT K
Move #DOA, DATAOUT
Call PROCESS
```

EXPLANATION:

This program, reads a line of characters from the keyboard & stores it in a memory buffer starting at locations LINE. Then it calls the subroutine “PROCESS” to process the input line. As each character is read, it is echoed back to the display. Register Ro is used as a updated using Auto – increment mode so that successive characters are stored in successive memory location. Each character is checked to see if there is carriage return (CR), char, which has the ASCII code 0D(hex). If it is, a line feed character (on) is sent to move the cursor one line down on the display & subroutine PROCESS is called. Otherwise, the program loops back to wait for another character from the keyboard.

PROGRAM CONTROLLED I/O

Here the processor repeatedly checks a status flag to achieve the required synchronization between Processor & I/O device.(ie) the processor polls the device.

There are 2 mechanisms to handle I/o operations. They are,

➔ Interrupt, -

➔ DMA (Synchronization is achieved by having I/O device send special over the bus where is ready for data transfer operation)

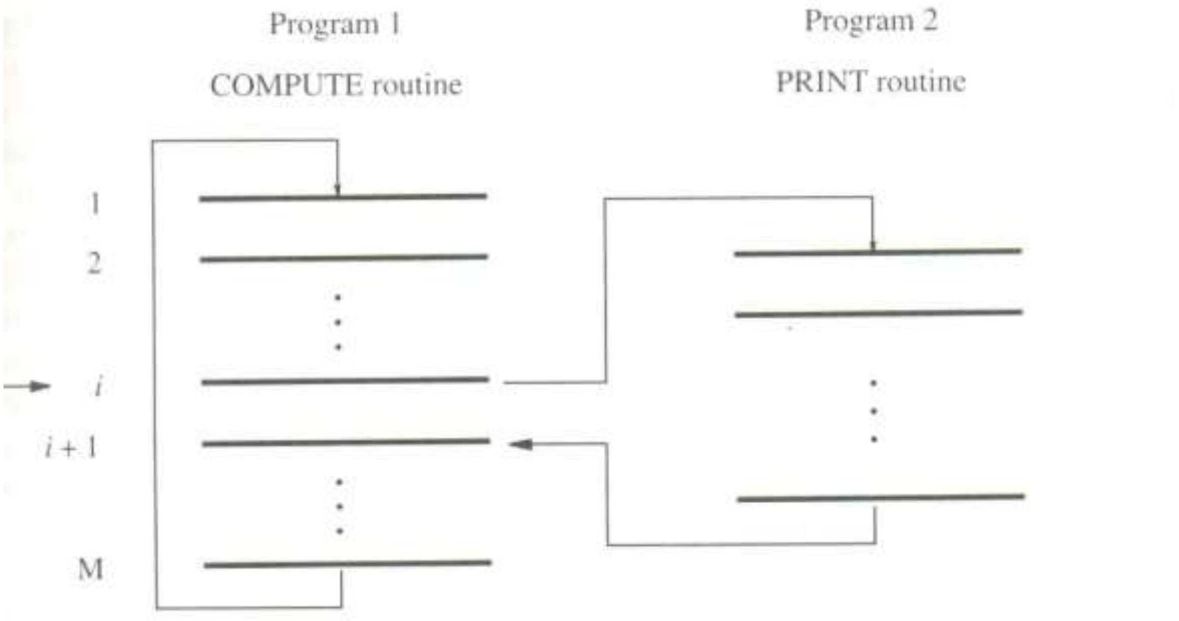
DMA:

1. Synchronization is achieved by having I/O device send special over the bus where is ready for data transfer operation)
2. It is a technique used for high speed I/O device.
3. Here, the input device transfer data directly to or from the memory without continuous involvement by the processor.

INTERRUPTS

- When a program enters a wait loop, it will repeatedly check the device status. During this period, the processor will not perform any function.
- The Interrupt request line will send a hardware signal called the interrupt signal to the processor.
- On receiving this signal, the processor will perform the useful function during the waiting period. The routine executed in response to an interrupt request is called **Interrupt Service Routine.** The interrupt resembles the subroutine calls.

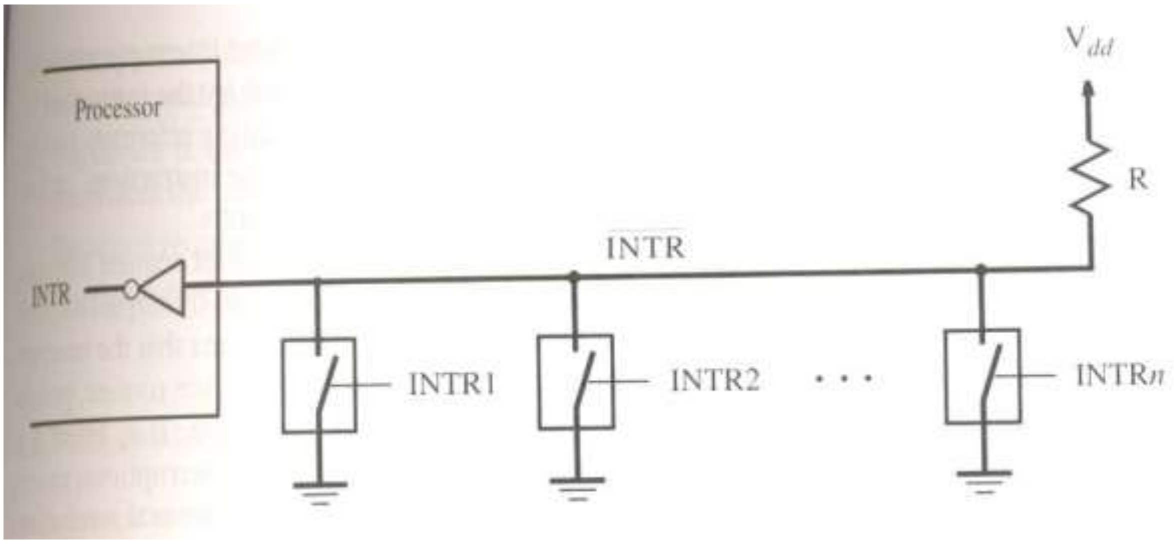
Fig:Transfer of control through the use of interrupts



- The processor first completes the execution of instruction i Then it loads the PC(Program Counter) with the address of the first instruction of the ISR.
- After the execution of ISR, the processor has to come back to instruction i + 1.
- Therefore, when an interrupt occurs, the current contents of PC which point to i +1 is put in temporary storage in a known location.
- A return from interrupt instruction at the end of ISR reloads the PC from that temporary storage location, causing the execution to resume at instruction i+1.
- When the processor is handling the interrupts, it must inform the device that its request has been recognized so that it remove its interrupt requests signal.
- This may be accomplished by a special control signal called the **interrupt acknowledge signal.**
- The task of saving and restoring the information can be done automatically by the processor.
- The processor saves only the contents of **program counter & status register** (ie) it saves only the minimal amount of information to maintain the integrity of the program execution.
- Saving registers also increases the delay between the time an interrupt request is received and the start of the execution of the ISR. This delay is called the **Interrupt Latency.**
- Generally, the long interrupt latency in unacceptable.
- The concept of interrupts is used in Operating System and in Control Applications, where processing of certain routines must be accurately timed relative to external events. This application is also called as **real-time processing**.

Interrupt Hardware:

Fig:An equivalent circuit for an open drain bus used to implement a common interrupt request line



- A single interrupt request line may be used to serve „n‟ devices. All devices are connected to the line via switches to ground.
- To request an interrupt, a device closes its associated switch, the voltage on INTR line drops to 0(zero).
- If all the interrupt request signals (INTR1 to INTRn) are inactive, all switches are open and the voltage on INTR line is equal to Vdd.
- When a device requests an interrupts, the value of INTR is the logical OR of the requests from individual devices.

(ie) **INTR = INTR1+…………+INTRn**

- **INTR →** It is used to name the INTR signal on common line it is active in the low voltage state.
- **Open collector** (bipolar ckt) or **Open drain** (MOS circuits) is used to drive INTR line.
- The Output of the Open collector (or) Open drain control is equal to a switch to the ground that is open when gates input is in „0‟ state and closed when the gates input is in „1‟ state.
- Resistor „R‟ is called a **pull-up resistor** because it pulls the line voltage upto the high voltage state when the switches are open.

ENABLING AND DISABLING INTERRUPTS

- The arrival of an interrupt request from an external device causes the processor to suspend the execution of one program & start the execution of another because the interrupt may alter the sequence of events to be executed.
- INTR is active during the execution of **Interrupt Service Routine**.
- There are 3 mechanisms to solve the problem of infinite loop which occurs due to successive interruptions of active INTR signals.

The following are the typical scenario.

➔ The device raises an interrupt request.
➔ The processor interrupts the program currently being executed.
➔ Interrupts are disabled by changing the control bits is PS (Processor Status register)
➔ The device is informed that its request has been recognized & in response, it deactivates the INTR signal.
➔ The actions are enabled & execution of the interrupted program is resumed.

Edge-triggered:

The processor has a special interrupt request line for which the interrupt handling circuit responds only to the leading edge of the signal. Such a line said to be edge-triggered.

HANDLING MULTIPLE DEVICES

Describe the hardware mechanism for handling multiple interrupt requests. ***(6 Marks April 2015)***

When several devices requests interrupt at the same time, it raises some questions. They are.

- How can the processor recognize the device requesting an interrupt?
- Given that the different devices are likely to require different ISR, how can the processor obtain the starting address of the appropriate routines in each case?
- Should a device be allowed to interrupt the processor while another interrupt is being serviced?
- How should two or more simultaneous interrupt requests be handled?

Polling Scheme:

→If two devices have activated the interrupt request line, the ISR for the selected device (first device) will be completed & then the second request can be serviced.

→The simplest way to identify the interrupting device is to have the ISR polls all the encountered with the IRQ bit set is the device to be serviced

→IRQ (Interrupt Request) -> when a device raises an interrupt requests, the status register IRQ is set to 1.

Merit:

It is easy to implement.

Demerit:

The time spent for interrogating the IRQ bits of all the devices that may not be requesting any service.

Vectored Interrupt: (6 Marks Dec 2014)

- Here the device requesting an interrupt may identify itself to the processor by sending a special code over the bus & then the processor start executing the ISR.

- The code supplied by the processor indicates the starting address of the ISR for the device.
- The code length ranges from 4 to 8 bits.
- The location pointed to by the interrupting device is used to store the staring address to ISR.
- The processor reads this address, called the interrupt vector & loads into PC.
- The interrupt vector also includes a new value for the Processor Status Register.
- When the processor is ready to receive the interrupt vector code, it activate the interrupt acknowledge (INTA) line.

Interrupt Nesting:
Multiple Priority Scheme:

- In multiple level priority scheme, we assign a priority level to the processor that can be changed under program control.
- The priority level of the processor is the priority of the program that is currently being executed.
- The processor accepts interrupts only from devices that have priorities higher than its own.
- At the time the execution of an ISR for some device is started, the priority of the processor is raised to that of the device.
- The action disables interrupts from devices at the same level of priority or lower.

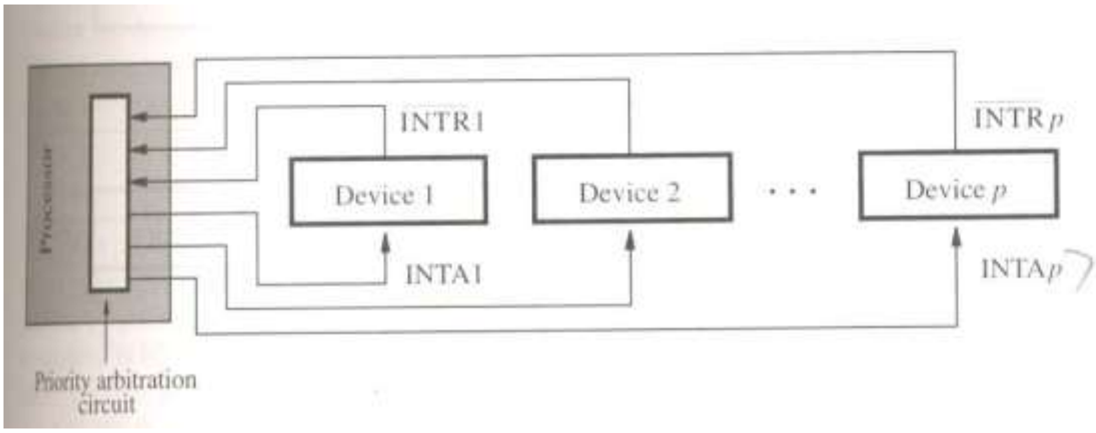
Privileged Instruction:

- The processor priority is usually encoded in a few bits of the Processor Status word. It can also be changed by program instruction & then it is write into PS.
- These instructions are called **privileged instruction.** This can be executed only when the processor is in supervisor mode.
- The processor is in supervisor mode only when executing OS routines.
- It switches to the user mode before beginning to execute application program.

Privileged Exception:

- User program cannot accidently or intentionally change the priority of the processor & disrupts the system operation.
- An attempt to execute a privileged instruction while in user mode, leads to a special type of interrupt called the privileged exception.

Fig: Implementation of Interrupt Priority using individual Interrupt request acknowledge lines

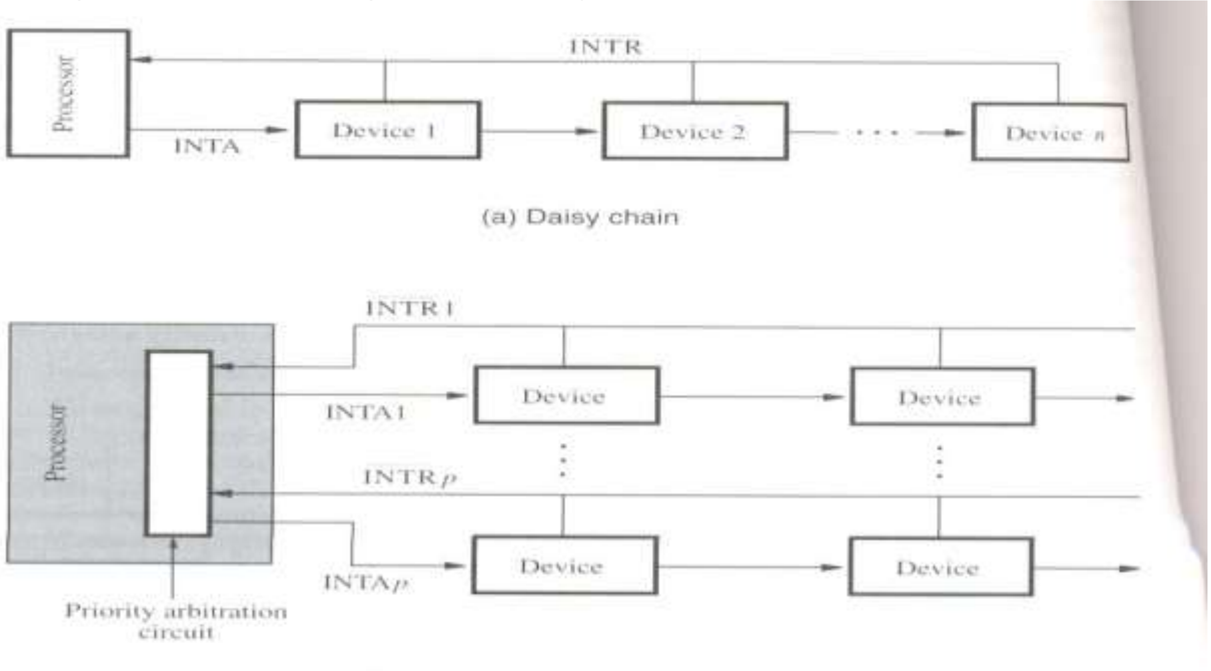


- Each of the interrupt request line is assigned a different priority level.
- Interrupt request received over these lines are sent to a priority arbitration circuit in the processor.
- A request is accepted only if it has a higher priority level than that currently assigned to the processor,

Simultaneous Requests:

Daisy Chain:

The interrupt request line INTR is common to all devices. The interrupt acknowledge line INTA is connected in a daisy chain fashion such that INTA signal propagates serially through the devices. When several devices raise an interrupt request, the INTR is activated & the processor responds by setting INTA line to 1. this signal is received by device.



(a) Daisy chain

(b) Arrangement of priority groups

- Device1 passes the signal on to device2 only if it does not require any service.
- If devices1 has a pending request for interrupt blocks that INTA signal & proceeds to put its identification code on the data lines.
- Therefore, the device that is electrically closest to the processor has the highest priority.

Merits:

It requires fewer wires than the individual connections.

Arrangement of Priority Groups:

Here the devices are organized in groups & each group is connected at a different priority level. Within a group, devices are connected in a daisy chain.

Controlling Device Requests:

KEN → Keyboard Interrupt Enable

DEN → Display Interrupt Enable

KIRQ / DIRQ → Keyboard / Display unit requesting an interrupt.

- There are two mechanism for controlling interrupt requests.
- At the devices end, an interrupt enable bit in a control register determines whether the device is allowed to generate an interrupt requests.
- At the processor end, either an interrupt enable bit in the PS (Processor Status) or a priority structure determines whether a given interrupt requests will be accepted.

Initiating the Interrupt Process:

- ➢ Load the starting address of ISR in location INTVEC (vectored interrupt).
- ➢ Load the address LINE in a memory location PNTR. The ISR will use this location as a pointer to store the i/p characters in the memory.
- ➢ Enable the keyboard interrupts by setting bit 2 in register CONTROL to 1.
- ➢ Enable interrupts in the processor by setting to 1, the IE bit in the processor status register PS.

Exception of ISR:

- ➢ Read the input characters from the keyboard input data register. This will cause the interface circuits to remove its interrupt requests.
- ➢ Store the characters in a memory location pointed to by PNTR & increment PNTR.
- ➢ When the end of line is reached, disable keyboard interrupt & inform program main.
- ➢ Return from interrupt.

EXCEPTIONS

What is an Exception? Explain in detail about the types of Exceptions?*(11 Marks Dec 2014)*

An interrupt is an event that causes the execution of one program to be suspended and the execution of another program to begin.

The Exception is used to refer to any event that causes an interruption.

Kinds of exception:

- Recovery from errors
- Debugging
- Privileged Exception

Recovery From Errors:

- Computers have error-checking code in Main Memory , which allows detection of errors in the stored data.
- If an error occurs, the control hardware detects it informs the processor by raising an interrupt.
- The processor also interrupts the program, if it detects an error or an unusual condition while executing the instance (ie) it suspends the program being executed and starts an execution service routine.
- This routine takes appropriate action to recover from the error.

Debugging:

- System software has a program called debugger, which helps to find errors in a program.
- The debugger uses exceptions to provide two important facilities

They are

- Trace
- Breakpoint

Trace Mode:

- When processor is in trace mode , an exception occurs after execution of every instance using the debugging program as the exception service routine.
- The debugging program examine the contents of registers, memory location etc.
- On return from the debugging program the next instance in the program being debugged is executed
- The trace exception is disabled during the execution of the debugging program.

Break point:

- Here the program being debugged is interrupted only at specific points selected by the user.
- An instance called the Trap (or) software interrupt is usually provided for this purpose.
- While debugging the user may interrupt the program execution after instance „I‟
- When the program is executed and reaches that point it examine the memory and register contents.

Privileged Exception:

- To protect the OS of a computer from being corrupted by user program certain instance can be executed only when the processor is in supervisor mode. These are called privileged exceptions.

- When the processor is in user mode, it will not execute instance (ie) when the processor is in supervisor mode , it will execute instance.

DIRECT MEMORY ACCESS

Explain the advantages of Direct Memory Access and its implementation in a computer.(*11 Marks April 2015,April 2014,Dec 2014,Nov 2012*)

- A special control unit may be provided to allow the transfer of large block of data at high speed directly between the external device and main memory , without continous intervention by the processor. This approach is called **DMA.**
- DMA transfers are performed by a control circuit called the **DMA Controller**.

To initiate the transfer of a block of words , the processor sends,

- Starting address
- Number of words in the block
- Direction of transfer.

- When a block of data is transferred , the DMA controller increment the memory address for successive words and keep track of number of words and it also informs the processor by raising an interrupt signal.
- While DMA control is taking place, the program requested the transfer cannot continue and the processor can be used to execute another program.
- After DMA transfer is completed, the processor returns to the program that requested the transfer.

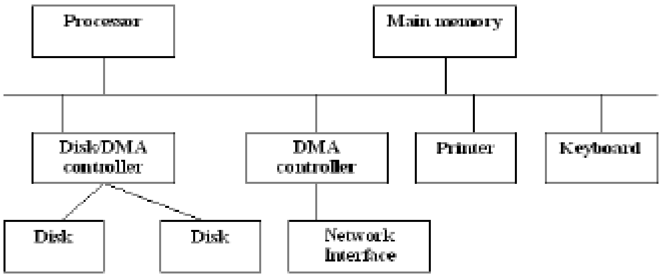
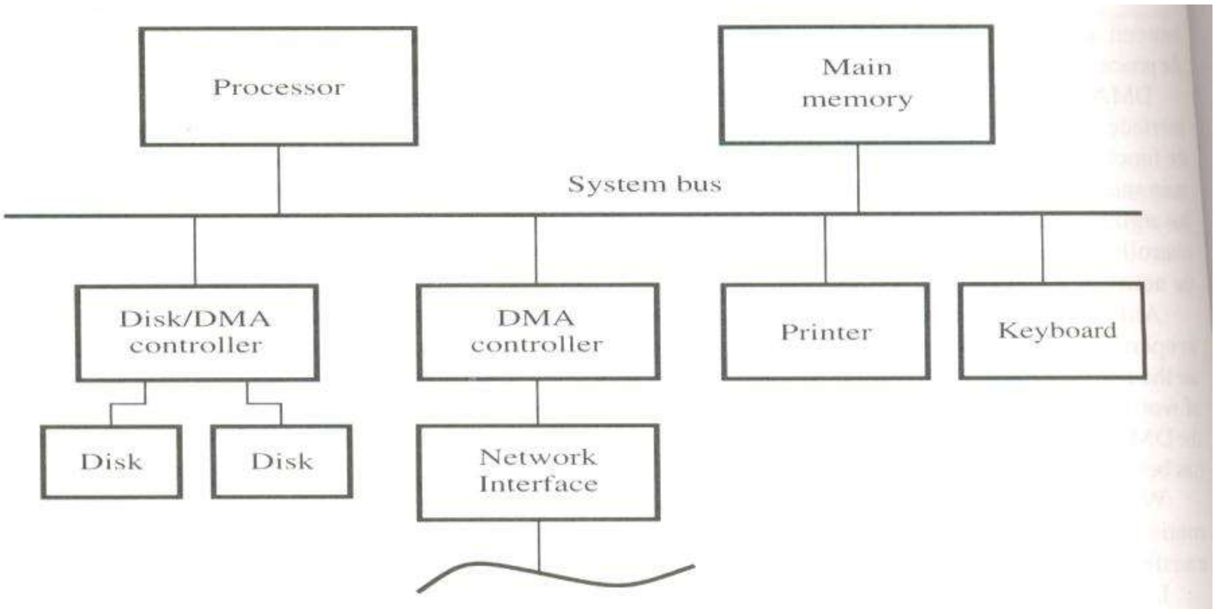


Fig:Registes in a DMA Interface

R/W → Determines the direction of transfer .
When
R/W =1, DMA controller read data from memory to I/O device.
R/W =0, DMA controller perform write operation.
Done Flag=1, the controller has completed transferring a block of data and is ready to receive another command.
IE=1, it causes the controller to raise an interrupt (interrupt Enabled) after it has completed transferring the block of data.
IRQ=1, it indicates that the controller has requested an interrupt.

Fig: Use of DMA controllers in a computer system



- A DMA controller connects a high speed network to the computer bus . The disk controller two disks, also has DMA capability and it provides two DMA channels.
- To start a DMA transfer of a block of data from main memory to one of the disks, the program write s the address and the word count inf. Into the registers of the corresponding channel of the disk controller.
- When DMA transfer is completed, it will be recorded in status and control registers of the DMA channel (ie) **Done bit=IRQ=IE=1**.

Cycle Stealing:

Requests by DMA devices for using the bus are having higher priority than processor requests .

Top priority is given to high speed peripherals such as ,

- Disk
- High speed Network Interface and Graphics display device.

Since the processor originates most memory access cycles, the DMA controller can be said to steal the memory cycles from the processor. This interviewing technique is called **Cycle stealing.**

Burst Mode:

The DMA controller may be given exclusive access to the main memory to transfer a block of data without interruption. This is known as **Burst/Block Mode**

Bus Master:

The device that is allowed to initiate data transfers on the bus at any given time is called the bus master.

Bus Arbitration:

It is the process by which the next device to become the bus master is selected and the bus mastership is transferred to it.

Types:

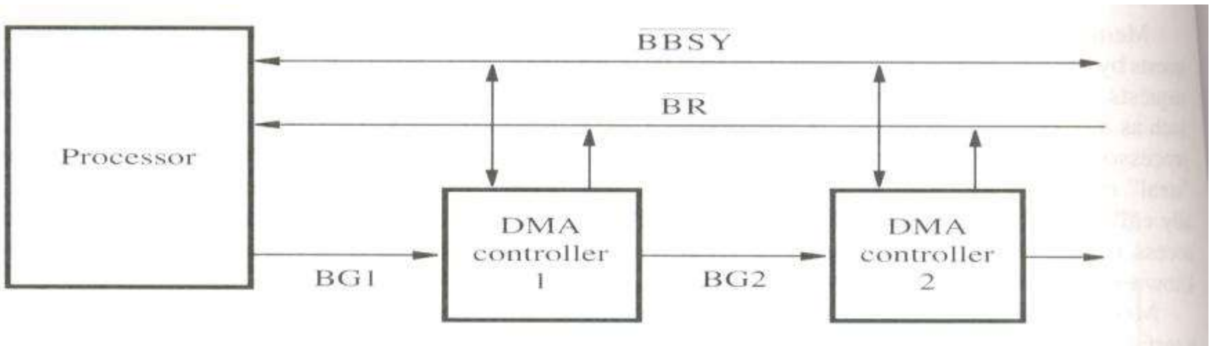
There are 2 approaches to bus arbitration. They are,

- Centralized arbitration (A single bus arbiter performs arbitration)
- Distributed arbitration (all devices participate in the selection of next bus master).

Centralized Arbitration:

- Here the processor is the bus master and it may grants bus mastership to one of its DMA controller.
- A DMA controller indicates that it needs to become the bus master by activating the Bus Request line (BR) which is an open drain line.
- The signal on BR is the logical OR of the bus request from all devices connected to it.
- When BR is activated the processor activates the Bus Grant Signal (BGI) and indicated the DMA controller that they may use the bus when it becomes free.
- This signal is connected to all devices using a daisy chain arrangement.
- If DMA requests the bus, it blocks the propagation of Grant Signal to other devices and it indicates to all devices that it is using the bus by activating open collector line, Bus Busy (BBSY).

Fig:A simple arrangement for bus arbitration using a daisy chain

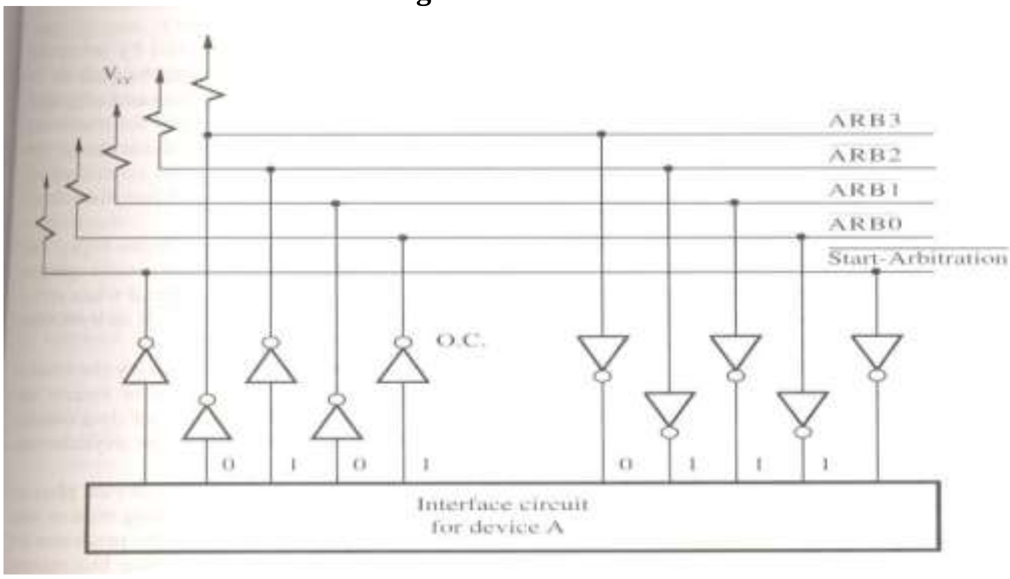


- The timing diagram shows the sequence of events for the devices connected to the processor is shown.
- DMA controller 2 requests and acquires bus mastership and later releases the bus.
- During its tenture as bus master, it may perform one or more data transfer.
- After it releases the bus, the processor resources bus mastership

Distributed Arbitration:

It means that all devices waiting to use the bus have equal responsibility in carrying out the arbitration process.

Fig:A distributed arbitration scheme



- Each device on the bus is assigned a 4 bit id.
- When one or more devices request the bus, they assert the Start-Arbitration signal & place their 4 bit ID number on four open collector lines, ARB0 to ARB3.
- A winner is selected as a result of the interaction among the signals transmitted over these lines.
- The net outcome is that the code on the four lines represents the request that has the highest ID number.
- The drivers are of open collector type. Hence, if the i/p to one driver is equal to 1, the i/p to another driver connected to the same bus line is equal to „0‟(ie. bus the is in low-voltage state).

Eg:

- Assume two devices A & B have their ID 5 (0101), 6(0110) and their code is 0111.
- Each devices compares the pattern on the arbitration line to its own ID starting from MSB.
- If it detects a difference at any bit position, it disables the drivers at that bit position. It does this by placing „0‟ at the i/p of these drivers.
- In our eg. „A‟ detects a difference in line ARB1, hence it disables the drivers on lines ARB1 & ARB0.
- This causes the pattern on the arbitration line to change to 0110 which means that „B‟ has won the contention.

BUSES

What are handshaking signals? Explain the handshake control of data transfer during input and output operation?*(11 Marks Apr 2015)*

A bus protocol is the set of rules that govern the behavior of various devices connected to the bus ie, when to place information in the bus, assert control signals etc.

The bus lines used for transferring data is grouped into 3 types. They are,

- Address line
- Data line
- Control line.

Control signals→Specifies that whether read / write operation has to performed.

- It also carries timing infn/. (ie) they specify the time at which the processor & I/O devices place the data on the bus & receive the data from the bus.
- During data transfer operation, one device plays the role of a „**Master**‟.
- **Master** device initiates the data transfer by issuing read / write command on the bus. Hence it is also called as „**Initiator**‟.
- The device addressed by the master is called as **Slave / Target**.

Types of Buses:

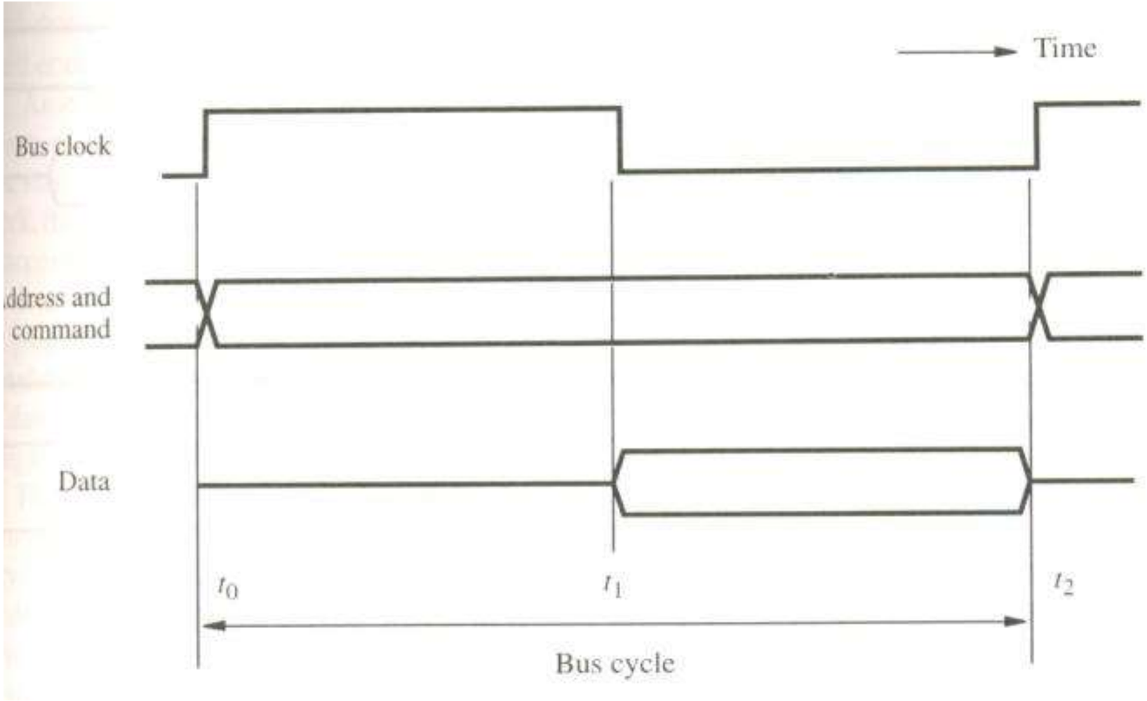
There are 2 types of buses. They are,

- Synchronous Bus
- Asynchronous Bus.

Synchronous Bus:-

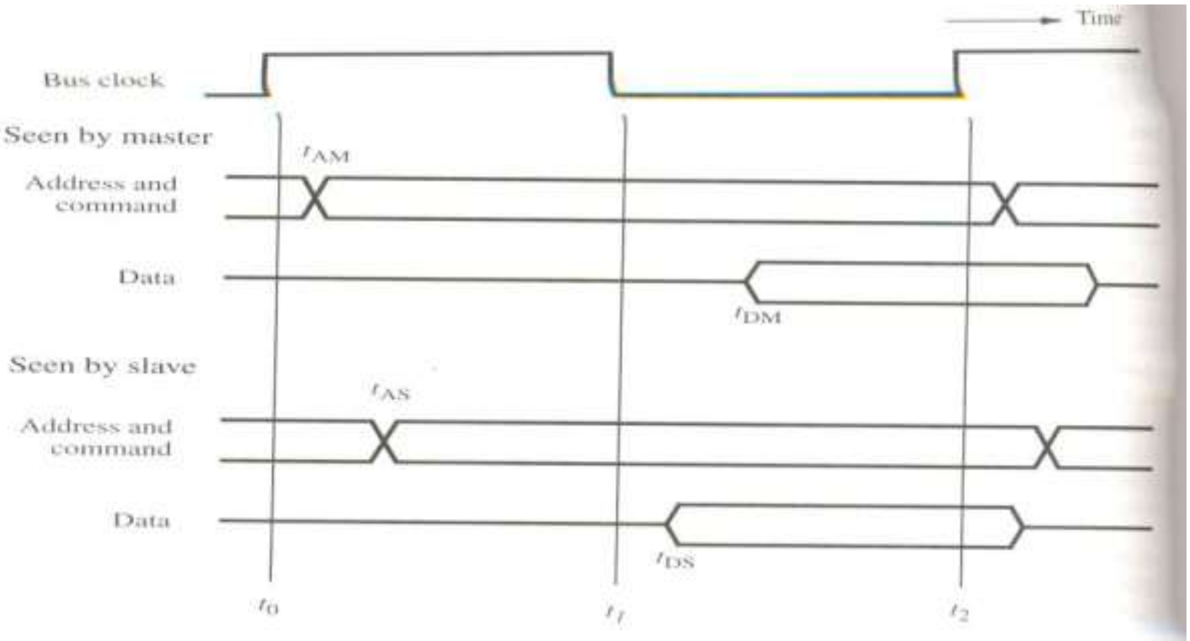
- In synchronous bus, all devices derive timing information from a common clock line.
- Equally spaced pulses on this line define equal time.
- During a „**bus cycle**‟, one data transfer on take place.
- The „**crossing points**‟ indicate the tone at which the patterns change.
- A „**signal line’** in an indeterminate / high impedance state is represented by an intermediate half way between the low to high signal levels.

Fig:Timing of an input transfer of a synchronous bus.



- At time to, the master places the device address on the address lines & sends an appropriate command on the control lines.
- In this case, the command will indicate an input operation & specify the length of the operand to be read.
- The clock pulse width t1 – t0 must be longer than the maximum delay between devices connected to the bus.
- The clock pulse width should be long to allow the devices to decode the address & control signals so that the addressed device can respond at time t_1.
- The slaves take no action or place any data on the bus before t_1.

Fig:A detailed timing diagram for the input transfer



- The picture shows two views of the signal except the clock.
- One view shows the signal seen by the master & the other is seen by the salve.
- The master sends the address & command signals on the rising edge at the beginning of clock period (t_0). These signals do not actually appear on the bus until t_{am}.
- Some times later, at t_{AS} the signals reach the slave. The slave decodes the address & at t1, it sends the requested data.
- At t_2, the master loads the data into its i/p buffer.
- Hence the period t_2, t_{DM} is the setup time for the masters i/p buffer.
- The data must be continued to be valid after t_2, for a period equal to the hold time of that buffers.

Demerits:

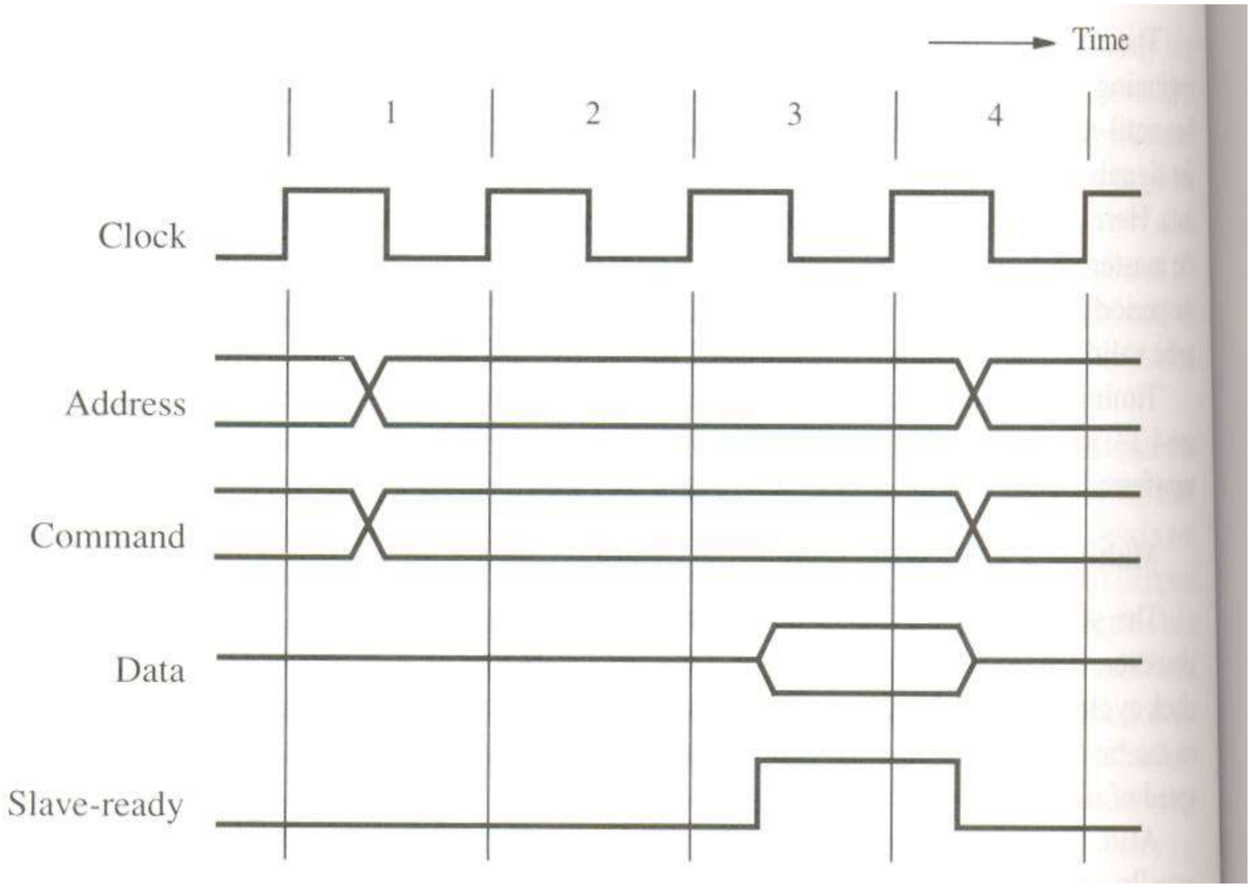
- ➢ The device does not respond.
- ➢ The error will not be detected.

Multiple Cycle Transfer:-

- During, clock cycle1, the master sends address & cmd infn/. On the bus‟ requesting a „read‟ operation.
- The slave receives this information & decodes it.
- At the active edge of the clock (ie) the beginning of clock cycel2, it makes accession to respond immediately.
- The data become ready & are placed in the bus at clock cycle3.
- At the same times, the slave asserts a control signal called „**slave-ready**‟.
- The master which has been waiting for this signal, strobes, the data to its i/p buffer at the end of clock cycle3.
- The bus transfer operation is now complete & the master sends a new address to start a new transfer in clock cycle4.

- The „**slave-ready**‟ signal is an acknowledgement form the slave to the master confirming that valid data has been sent.

Fig:An input transfer using multiple clock cycles



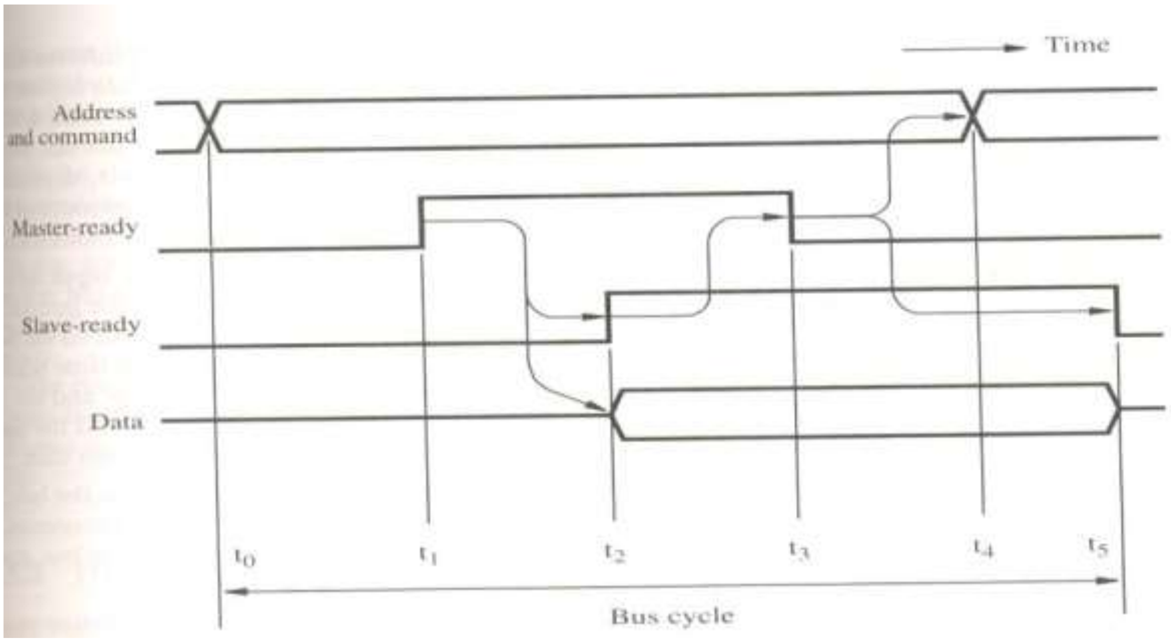
Asynchronous Bus:-

An alternate scheme for controlling data transfer on. The bus is based on the use of „**handshake**‟ between **Master** & the **Slave**. The common clock is replaced by two timing control lines.

They are

→Master–ready
→Slave ready.

Fig:Handshake control of data transfer during an input operation



The handshake protocol proceed as follows :

At **t_0** → The master places the address and command information on the bus and all devices on the bus begin to decode the information

At **t_1** → The master sets the Master ready line to 1 to inform the I/O devices that the address and command information is ready.

- The delay $t_1 – t_0$ is intended to allow for any skew that may occurs on the bus.
- The skew occurs when two signals simultaneously transmitted from one source arrive at the destination at different time.
- Thus to guarantee that the Master ready signal does not arrive at any device a head of the address and command information the delay $t_1 – t_0$ should be larger than the maximum possible bus skew.

At **t_2** → The selected slave having decoded the address and command information performs the required i/p operation by placing the data from its data register on the data lines. At the same time, it sets the “slave – Ready” signal to 1.

At **t_3** → The slave ready signal arrives at the master indicating that the i/p data are available on the bus.

At **t_4** → The master removes the address and command information on the bus. The delay between t_3 and t_4 is again intended to allow for bus skew. Errorneous addressing may take place if the address, as seen by some device on the bus, starts to change while the master – ready signal is still equal to 1.

At **t_5** → When the device interface receives the 1 to 0 tranitions of the Master – ready signal. It removes the data and the slave – ready signal from the bus. This completes the i/p transfer.

- In this diagram, the master place the output data on the data lines and at the same time it transmits the address and command information.
- The selected slave strobes the data to its o/p buffer when it receives the Master-ready signal and it indicates this by setting the slave – ready signal to 1.
- At time t_0 to t_1 and from t_3 to t_4, the Master compensates for bus.
- A change of state is one signal is followed by a change is the other signal. Hence this scheme is called as **Full Handshake**.
- It provides the higher degree of flexibility and reliability.

INTERFACE CIRCUITS

The interface circuits are of two types. They are

- Parallel Port
- Serial Port

Parallel Port:

- The output of the encoder consists of the bits that represent the encoded character and one signal called **vali**d,whi**c**h indicates the key is pressed.
- The information is sent to the interface circuits,which contains a data register,DATAIN and a status flag SIN.
- When a key is pressed, the Valid signal changes from 0 to1,causing the ASCII code to be loaded into DATAIN and SIN set to 1.
- The status flag SIN set to 0 when the processor reads the contents of the DATAIN register.
- The interface circuit is connected to the asynchronous bus on which transfers are controlled using the Handshake signals Master ready and Slave-ready.
- A serial port used to connect the processor to I/O device that requires transmission one bit at a time.
- It is capable of communicating in a bit serial fashion on the device side and in a bit parallel fashion on the bus side.

STANDARD I/O INTERFACE

- A standard I/O Interface is required to fit the I/O device with an Interface circuit.
- The processor bus is the bus defined by the signals on the processor chip itself.
- The devices that require a very high speed connection to the processor such as the main memory, may be connected directly to this bus.
- **The bridge** connects two buses, which translates the signals and protocols of one bus into another.
- The bridge circuit introduces a small delay in data transfer between processor and the devices.

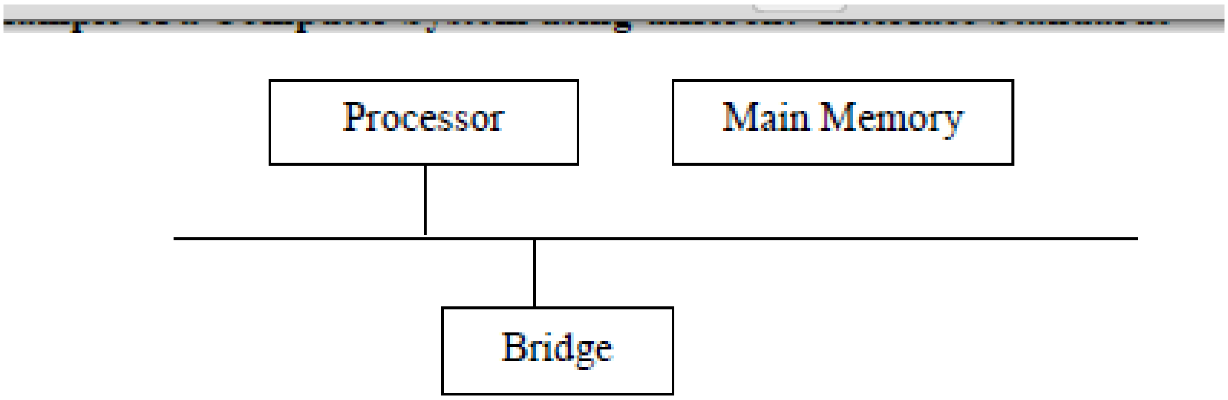
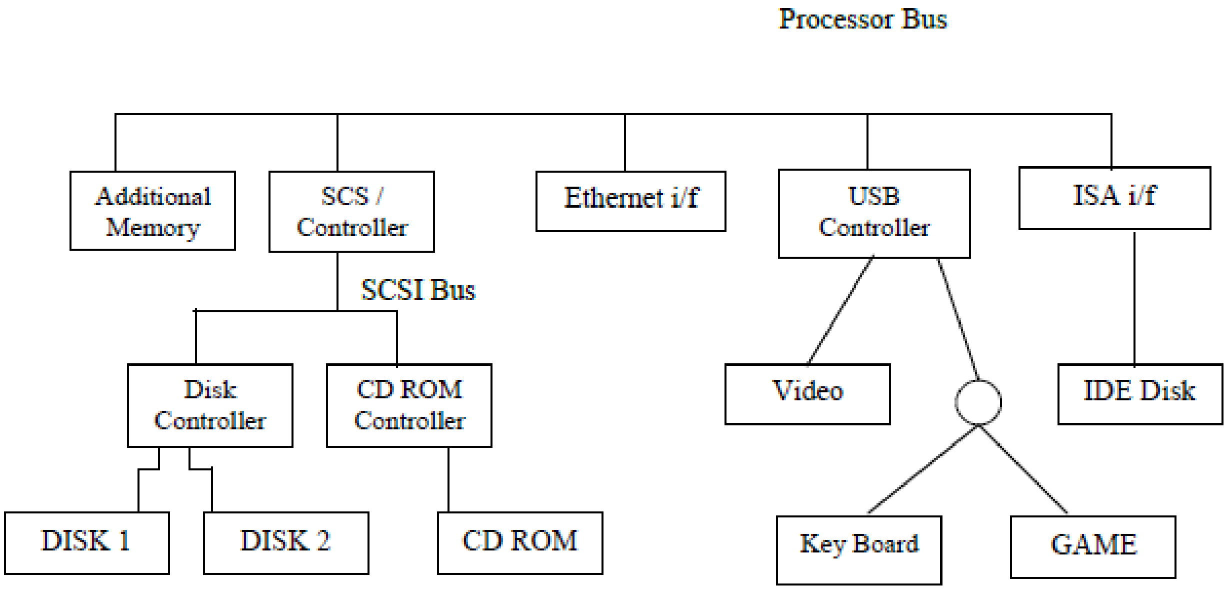


Fig: Example of a Computer System using different Interface Standards



We have 3 Bus standards.They are,

- **PCI (**Peripheral Component Inter Connect)
- **SCSI** (Small Computer System Interface)
- **USB** (Universal Serial Bus)

- **PCI** defines an expansion bus on the motherboard.
- **SCSI** and **USB** are used for connecting additional devices both inside and outside the computer box.
- **SCSI** bus is a high speed parallel bus intended for devices such as disk and video display.

- **USB** uses a serial transmission to suit the needs of equipment ranging from keyboard keyboard to game control to internal connection.
- **IDE (Integrated Device Electronics)** disk is compatible with ISA which shows the connection to an Ethernet.

PCI

- PCI is developed as a low cost bus that is truly processor independent.
- It supports high speed disk, graphics and video devices.
- PCI has plug and play capability for connecting I/O devices.
- To connect new devices, the user simply connects the device interface board to the bus.

UNIT III - INPUT/OUTPUT ORGANIZATION: Accessing I/O Devices, Interrupts, Interrupt Hardware, Enabling and Disabling Interrupts, Handling Multiple Devices, Controlling Device Requests, Exceptions, Use Of Interrupts in Operating Systems, Pentium Interrupt Structure, Direct Memory Access, Busses, Interface Circuits, Standard I/O

Data Tranfer:

- The data are transferred between cache and main memory is the bursts of several words and they are stored in successive memory locations.
- When the processor specifies an address and request a „read‟ operation from memory, the memory responds by sending a sequence of data words starting at that address.
- During write operation, the processor sends the address followed by sequence of data words to be written in successive memory locations.
- PCI supports read and write operation.
- A read / write operation involving a single word is treated as a burst of length one.

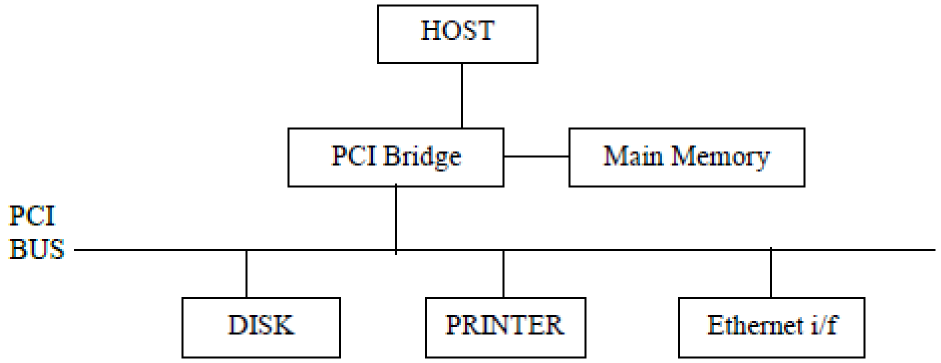
PCI has three address spaces. They are

- ➢ Memory address space
- ➢ I/O address space
- ➢ Configuration address space

I/O address space → It is intended for use with processor
Configuration space → It is intended to give PCI, its plug and play capability.

- PCI Bridge provides a separate physical connection to main memory.
- The master maintains the address information on the bus until data transfer is completed.
- At any time, only one device acts as **bus master**.
- A master is called „initiator‟ in PCI which is either processor or **DMA.**
- The addressed device that responds to read and write commands is called a **target.**
- A complete transfer operation on the bus, involving an address and bust of data is called a „**transaction’.**

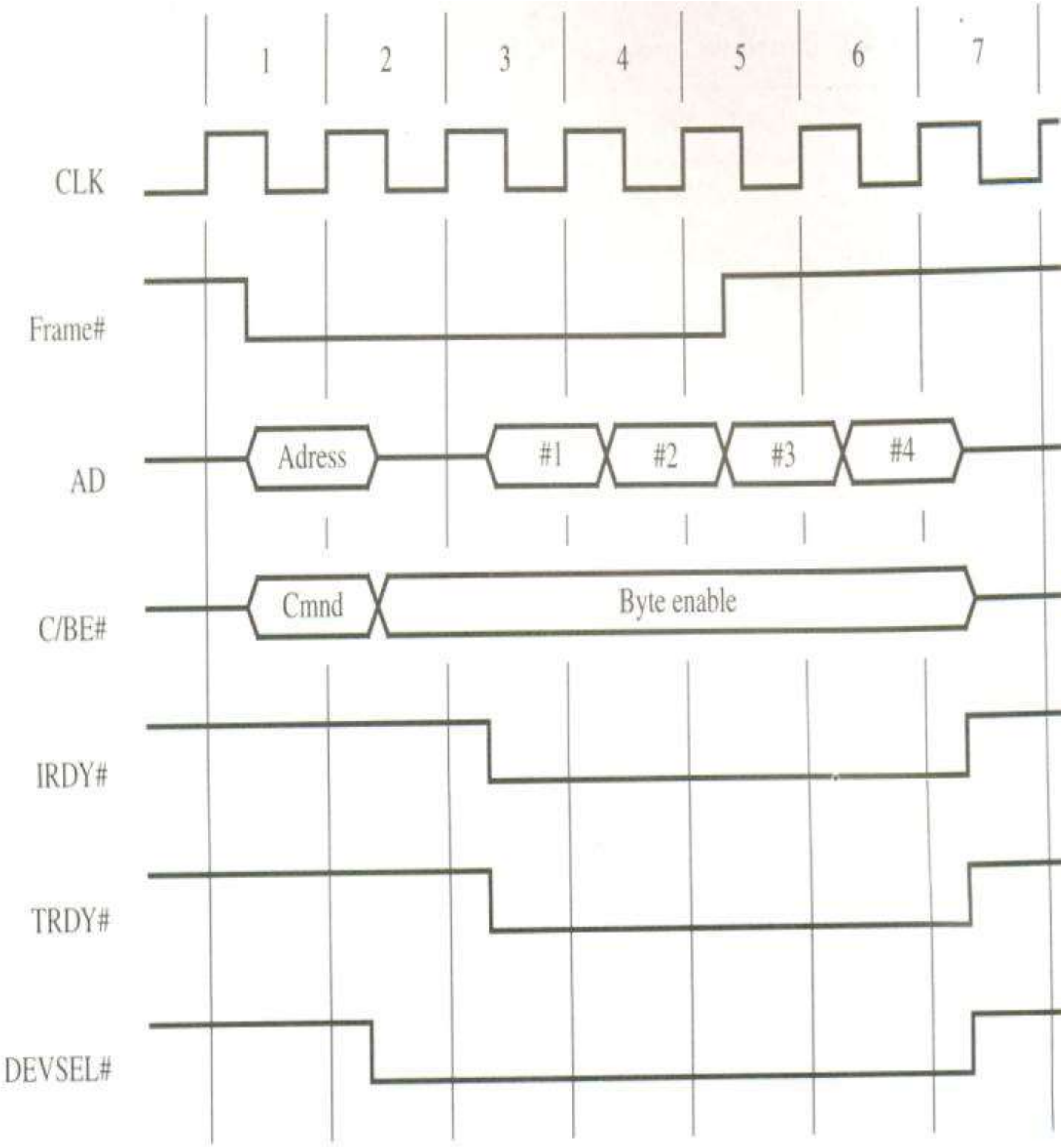
Fig:Use of a PCI bus in a Computer system



Data Transfer Signals on PCI Bus:
Name Function
CLK → 33 MHZ / 66 MHZ clock
FRAME # → Sent by the indicator to indicate the duration of transaction
AD → 32 address / data line

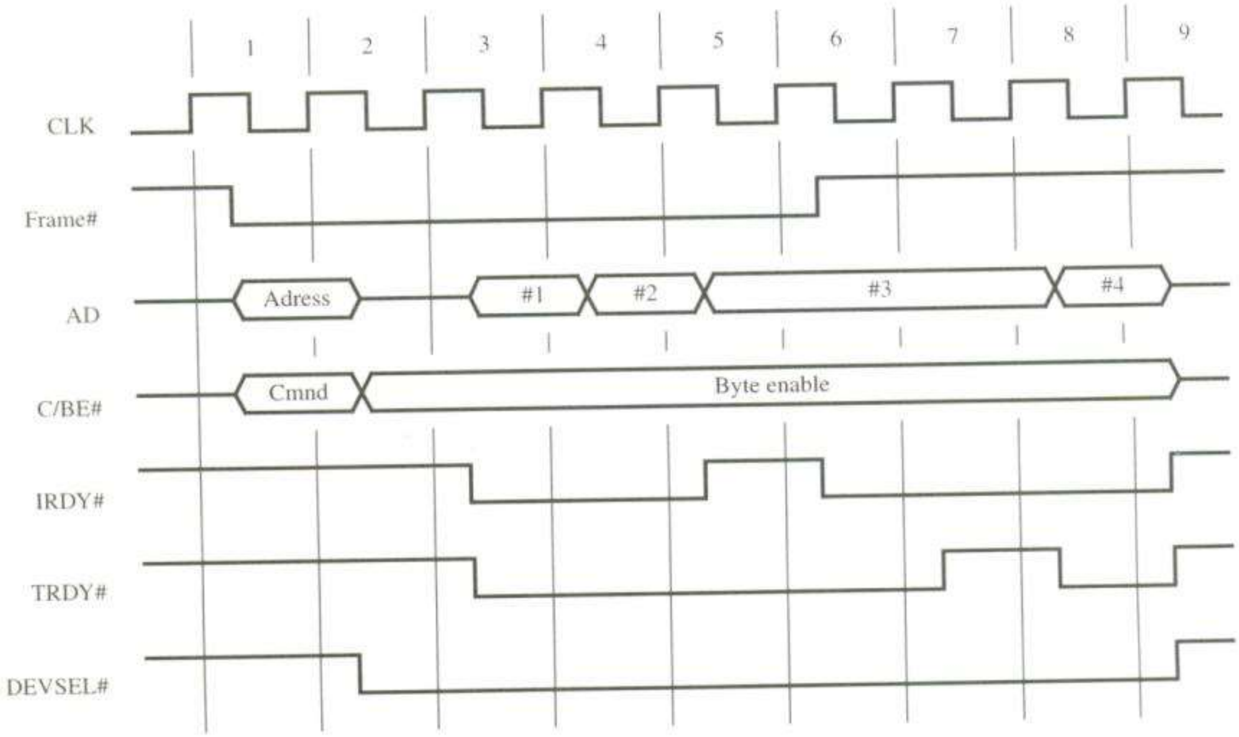
C/BE # → 4 command / byte Enable Lines
IRDY, TRDYA→ Initiator Ready, Target Ready Signals
DEVSEL # → A response from the device indicating that it has recognized its address and is ready for data transfer transaction.
IDSEL # → Initialization Device Select
Individual word transfers are called „**phases’**.

Fig :Read operation an PCI Bus



- In Clock cycle1, the processor asserts FRAME # to indicate the beginning of a transaction ; it sends the address on AD lines and command on C/BE # Lines.
- Clock cycle2 is used to turn the AD Bus lines around ; the processor ; The processor removes the address and disconnects its drives from AD lines.
- The selected target enable its drivers on AD lines and fetches the requested data to be placed on the bus.
- It asserts DEVSEL # and maintains it in asserted state until the end of the transaction.
- During clock cycle 3, the initiator asserts IRDY #, to indicate that it is ready to receive data.
- If the target has data ready to send then it asserts TRDY #. In our eg, the target sends 3 more words of data in clock cycle 4 to 6.
- The indicator uses FRAME # to indicate the duration of the burst, since it read 4 words, the initiator negates FRAME # during clock cycle 5.
- After sending the 4th word, the target disconnects its drivers and negates DEVSEL # during clockcycle 7.

Fig: A read operation showing the role of IRDY# / TRY#



- C/BE # is used to send a bus command in clock cycle and it is used for different purpose during the rest of the transaction.
- It indicates the pause in the middle of the transaction.
- The first and words are transferred and the target sends the 3rd word in cycle 5.
- But the indicator is not able to receive it. Hence it negates IRDY#.
- In response the target maintains 3rd data on AD line until IRDY is asserted again.
- In cycle 6, the indicator asserts IRDY. But the target is not ready to transfer the fourth word immediately, hence it negates TRDY in cycle 7. Hence it sends the 4th word and asserts TRDY# at cycle 8.

-

Device Configuration:

The PCI has a configuration ROM memory that stores information about that device. The configuration ROM‟s of all devices are accessible in the configuration address space. The initialization s/w read these ROM‟s whenever the S/M is powered up or reset In each case, it determines whether the device is a printer, keyboard, Ethernet interface or disk controller. Devices are assigned address during initialization process and each device has an w/p signal called IDSEL # (Initialization device select) which has 21 address lines (AD) (AD to AD31). During configuration operation, the address is applied to AD i/p of the device and the corresponding AD line is set to and all other lines are set to 0.

AD11 - AD31 →**Upper address line**
A00 - A10 →**Lower address line** → Specify the type of the operation and to
access the content of device configuration ROM.

- The configuration software scans all 21 locations.
- PCI bus has interrupt request lines.
- Each device may requests an address in the I/O space or memory space

Electrical Characteristics:

- The connectors can be plugged only in compatible motherboards PCI bus can operate with either 5 – 33V power supply.
- The motherboard can operate with signaling system.

SCSI Bus:- (Small Computer System Interface)

- SCSI refers to the standard bus which is defined by ANSI (American National Standard Institute).
- SCSI bus the several options. It may be,

Narrow bus → It has 8 data lines & transfers 1 byte at a time.
Wide bus → It has 16 data lines & transfer 2 byte at a time.
Single-Ended Transmission → Each signal uses separate wire.
HVD (High Voltage Differential) → It was 5v (TTL cells)
LVD (Low Voltage Differential) → It uses 3.3v

- Because of these various options, SCSI connector may have 50, 68 or 80 pins.
- The data transfer rate ranges from 5MB/s to 160MB/s 320Mb/s, 640MB/s.

The transfer rate depends on,

- ➢ Length of the cable
- ➢ Number of devices connected.

- To achieve high transfer rat, the bus length should be 1.6m for SE signaling and 12m for LVD signaling.
- The SCSI bus us connected to the processor bus through the SCSI controller.
- The data are stored on a disk in blocks called sectors.
- Each sector contains several hundreds of bytes. These data will not be stored in contiguous memory location.

- SCSI protocol is designed to retrieve the data in the first sector or any other selected sectors.
- Using SCSI protocol, the burst of data are transferred at high speed.

The controller connected to SCSI bus is of 2 types. They are,

- Initiator
- Target

Intiator:

- It has the ability to select a particular target & to send commands specifying the operation to be performed.
- They are the controllers on the processor side.

Target:

- The disk controller operates as a target.
- It carries out the commands it receive from the initiator. The initiator establishes a logical connection with the intended target.

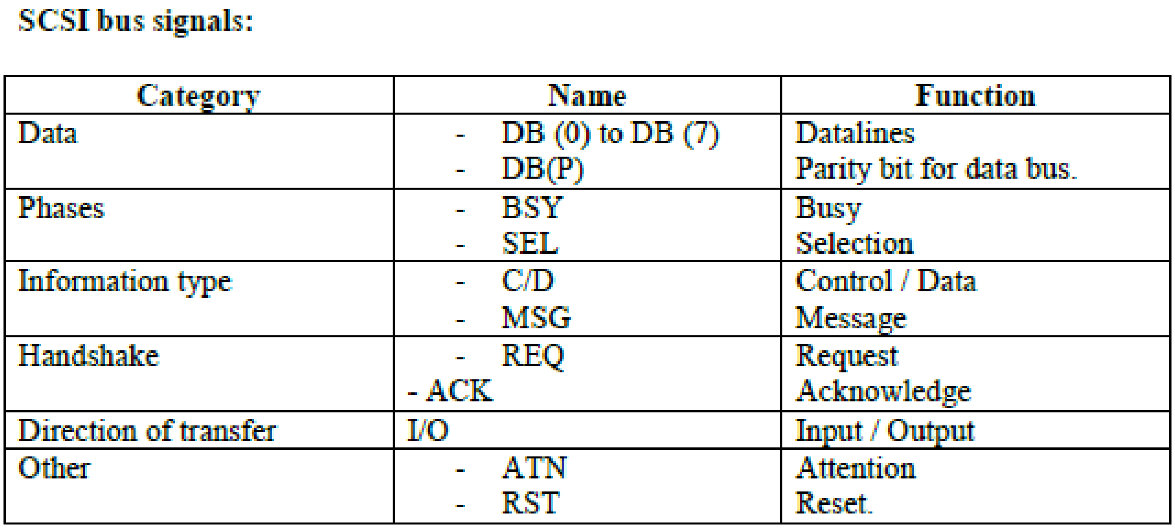
Steps:

Consider the disk read operation, it has the following sequence of events.

- The SCSI controller acting as initiator, contends process, it selects the target controller & hands over control of the bus to it.
- The target starts an output operation, in response to this the initiator sends a command specifying the required read operation.
- The target that it needs to perform a disk seek operation, sends a message to the initiator indicating that it will temporarily suspends the connection between them.
- Then it releases the bus.
- The target controller sends a command to disk drive to move the read head to the first sector involved in the requested read in a data buffer. When it is ready to begin transferring data to initiator, the target requests control of the bus. After it wins arbitration, it reselects the initiator controller, thus restoring the suspended connection.
- The target transfers the controls of the data buffer to the initiator & then suspends the connection again. Data are transferred either 8 (or) 16 bits in parallel depending on the width of the bus.
- The target controller sends a command to the disk drive to perform another seek operation. Then it transfers the contents of second disk sector to the initiator. At the end of this transfer, the logical connection b/w the two controller is terminated.
- As the initiator controller receives the data, if stores them into main memory using DMA approach.
- The SCSI controller sends an interrupt to the processor to inform it that the requested operation has been completed.

Bus Signals:-

- The bus has no address lines.
- Instead, it has data lines to identify the bus controllers involved in the selection / reselection / arbitration process.
- For narrow bus, there are 8 possible controllers numbered from 0 to 7.
- For a wide bus, there are 16 controllers.
- Once a connection is established b/w two controllers, these is no further need for addressing & the datalines are used to carry the data.



SCSI bus signals:

Category	Name	Function
Data	- DB (0) to DB (7) - DB(P)	Datalines Parity bit for data bus.
Phases	- BSY - SEL	Busy Selection
Information type	- C/D - MSG	Control / Data Message
Handshake	- REQ - ACK	Request Acknowledge
Direction of transfer	I/O	Input / Output
Other	- ATN - RST	Attention Reset.

- All signal names are proceeded by minus sign.
- This indicates that the signals are active or that the dataline is equal to 1, when they are in the low voltage state.

Phases in SCSI Bus:-

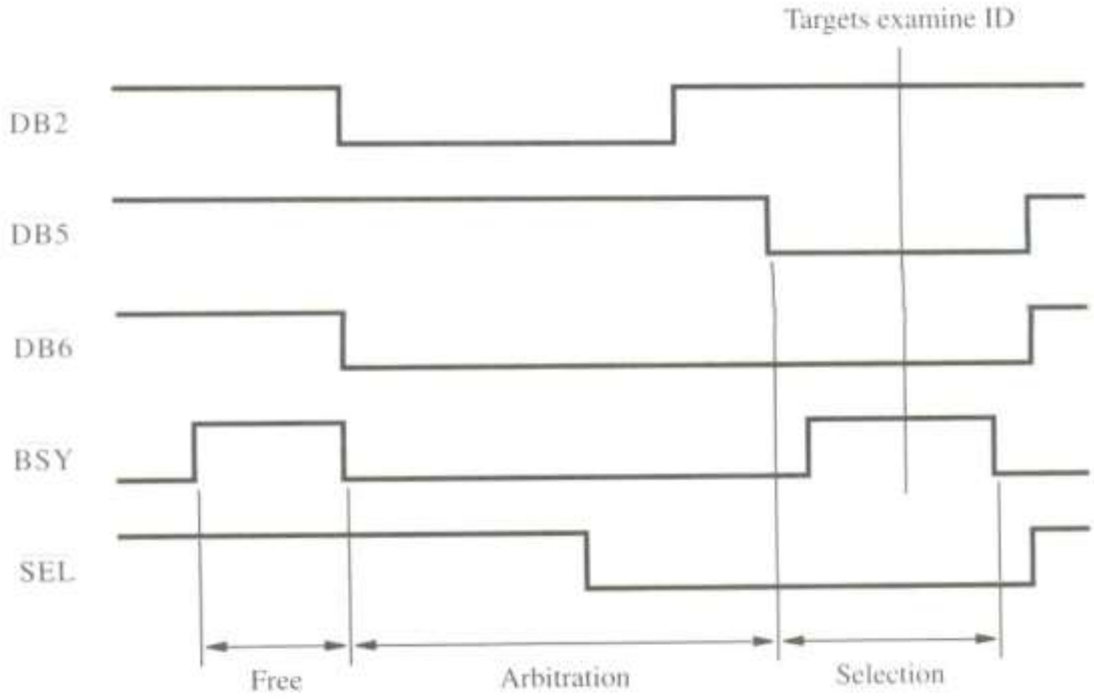
- The phases in SCSI bus operation are,

➢ Arbitration
➢ Selection
➢ Information transfer
➢ Reselection

Arbitration:-

- When the –BSY signal is in inactive state, the bus will he free & any controller can request the use of the bus.
- Since each controller may generate requests at the same time, SCSI uses distributed arbitration scheme.
- Each controller on the bus is assigned a fixed priority with controller 7 having the highest priority.
- When –BSY becomes active, all controllers that are requesting the bus examines the data lines & determine whether the highest priority device is requesting the bus at the same time.
- The controller using the highest numbered line realizes that it has won the arbitration process.
- At that time, all other controllers disconnect from the bus & wait for –BSY to become inactive again.

Fig:Arbitration and selection on the SCSI bus.Device 6 wins arbitration and select device 2



Selection:

- Here Device wons arbitration and it asserts –BSY and –DB6 signals.
- The Select Target Controller responds by asserting –BSY.
- This informs that the connection that it requested is established.

Reselection:

- The connection between the two controllers has been reestablished, with the target in control the bus as required for data transfer to proceed.

USB – Universal Serial Bus
Explain the protocols of ?USB with a neat diagram.*(5 Marks Nov 2015)*

- USB supports 3 speed of operation. They are,

➢ Low speed (1.5Mb/s)
➢ Full speed (12mb/s)
➢ High speed (480mb/s)

The USB has been designed to meet the key objectives. They are,

❖ It provide a simple, low cost & easy to use interconnection s/m that overcomes the difficulties due to the limited number of I/O ports available on a computer.
❖ It accommodate a wide range of data transfer characteristics for I/O devices including telephone & Internet connections.
❖ Enhance user convenience through **'Plug & Play'** mode of operation.

Port Limitation:-

- Normally the system has a few limited ports.

- To add new ports, the user must open the computer box to gain access to the internal expansion bus & install a new interface card.
- The user may also need to know to configure the device & the s/w.
-

Merits of USB:-
→USB helps to add many devices to a computer system at any time without opening the computer box.

Device Characteristics:-

- The kinds of devices that may be connected to a cptr cover a wide range of functionality.
- The speed, volume & timing constrains associated with data transfer to & from devices varies significantly.

Eg:1 Keyboard → Since the event of pressing a key is not synchronized to any other event in a computer system, the data generated by keyboard are called asynchronous.
The data generated from keyboard depends upon the speed of the human operator which is about 100bytes/sec.

Eg:2 Microphone attached in a cptr s/m internally / externally
The sound picked up by the microphone produces an analog electric signal, which must be converted into digital form before it can be handled by the cptr.

- This is accomplished by sampling the analog signal periodically.
- The sampling process yields a continuous stream of digitized samples that arrive at regular intervals, synchronized with the sampling clock. Such a stream is called isochronous (ie) successive events are separated by equal period of time.
- If the sampling rate in „S‟ samples/sec then the maximum frequency captured by sampling process is s/2.
- A standard rate for digital sound is 44.1 KHz.

Requirements for sampled Voice:-

- It is important to maintain precise time (delay) in the sampling & replay process.
- A high degree of jitter (Variability in sampling time) is unacceptable.

Eg-3:Data transfer for Image & Video:-

- The transfer of images & video require higher bandwidth.
- The bandwidth is the total data transfer capacity of a communication channel.
- To maintain high picture quality, The image should be represented by about 160kb, & it is transmitted 30 times per second for a total bandwidth if 44MB/s.

Plug & Play:-

- The main objective of USB is that it provides a plug & play capability.
- The plug & play feature enhances the connection of new device at any time, while the system is operation.

The system should,

➢ Detect the existence of the new device automatically.
➢ Identify the appropriate device driver s/w.
➢ Establish the appropriate addresses.
➢ Establish the logical connection for communication.

USB Architecture:-

- USB has a serial bus format which satisfies the low-cost & flexibility requirements.
- Clock & data information are encoded together & transmitted as a single signal.
- There are no limitations on clock frequency or distance arising form data skew, & hence it is possible to provide a high data transfer bandwidth by using a high clock frequency.
- To accommodate a large no/. of devices that can be added / removed at any time, the USB has the tree structure.

USB Tree Structure

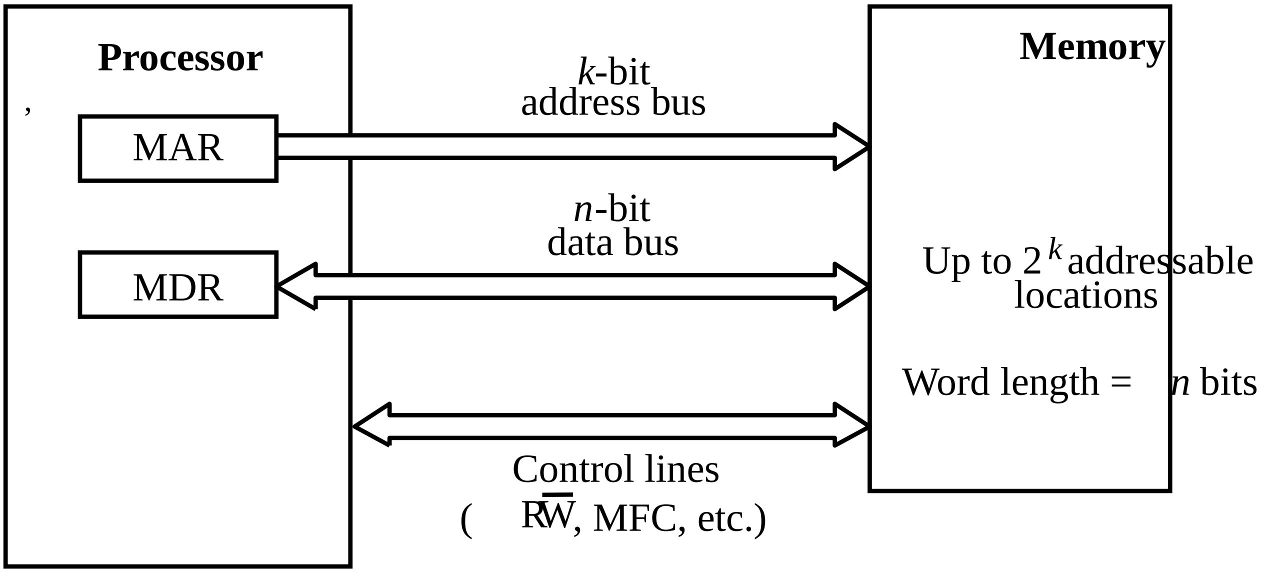
- Each node of the tree has a device called „**hub**‟, which acts as an intermediate control point b/w host & I/O devices.
- At the root of the tree, the „root hub‟ connects the entire tree to the host computer.
- The leaves of the tree are the I/O devices being served.

UNIT IV - THE MEMORY SYSTEM: Some Basic Concepts, Semiconductor RAM Memories, Read-Only Memories, Speed, Size, and Cost, Cache Memories, Performance Considerations, Virtual memories, Memory Management requirements, Secondary Storage.

THE MEMORY SYSTEM

SOME BASIC CONCEPTS

- Maximum size of the Main Memory
- byte-addressable
- CPU-Main Memory Connection



- Measures for the speed of a memory:
  - memory access time.
  - memory cycle time.
- An important design issue is to provide a computer system with as large and fast a memory as possible, within a given cost target.
- Several techniques to increase the effective size and speed of the memory:
  - Cache memory (to increase the effective speed).
  - Virtual memory (to increase the effective size).

SEMICONDUCTOR RAM MEMORIES

- Each memory cell can hold one bit of information.
- Memory cells are organized in the form of an array.
- One row is one memory word.
- All cells of a row are connected to a common line, known as the "word line".
- Word line is connected to the address decoder.
- Sense/write circuits are connected to the data input/output lines of the memory chip.

UNIT IV - THE MEMORY SYSTEM: Some Basic Concepts, Semiconductor RAM Memories, Read-Only Memories, Speed, Size, and Cost, Cache Memories, Performance Considerations, Virtual memories, Memory Management requirements, Secondary Storage.

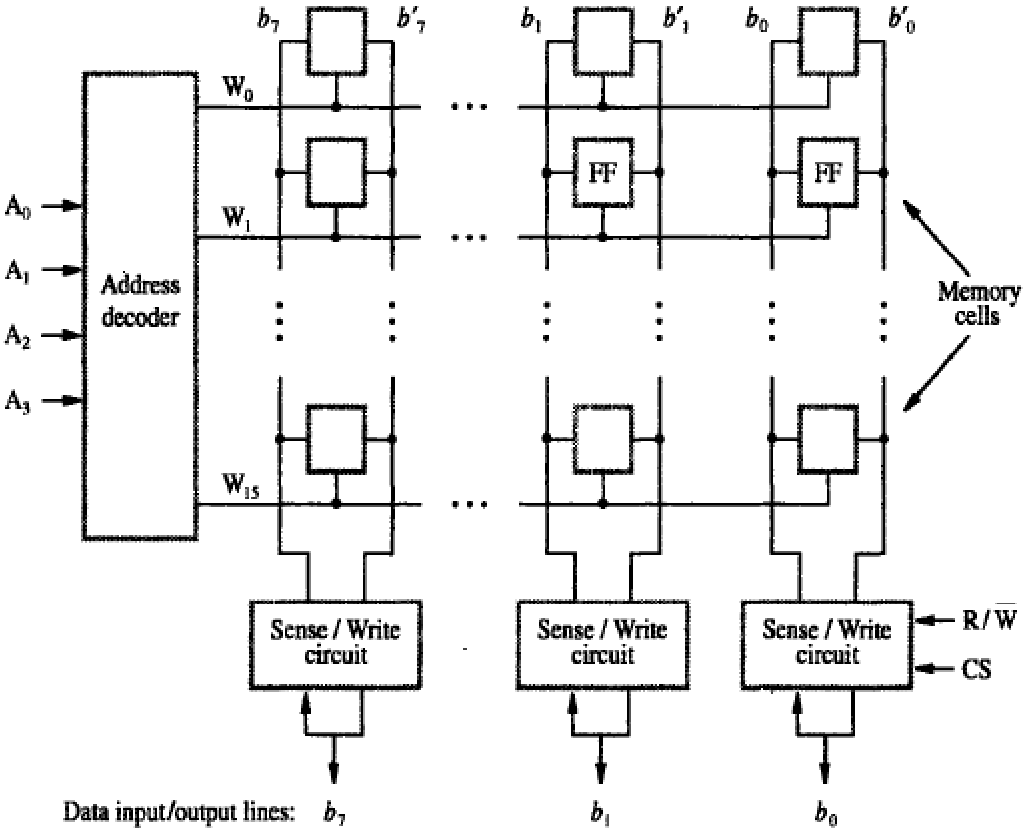


Fig 4.1 organization of bit cells in a memory chips

SRAM cells

- Two transistor inverters are cross connected to implement a basic flip-flop.
- The cell is connected to one word line and two bits lines by transistors T1 and T2
- When word line is at ground level, the transistors are turned off and the latch retains its state
- Read operation: In order to read state of SRAM cell, the word line is activated to close switches T1 and T2. Sense/Write circuits at the bottom monitor the state of b and b’

UNIT IV - THE MEMORY SYSTEM: Some Basic Concepts, Semiconductor RAM Memories, Read-Only Memories, Speed, Size, and Cost, Cache Memories, Performance Considerations, Virtual memories, Memory Management requirements, Secondary Storage.

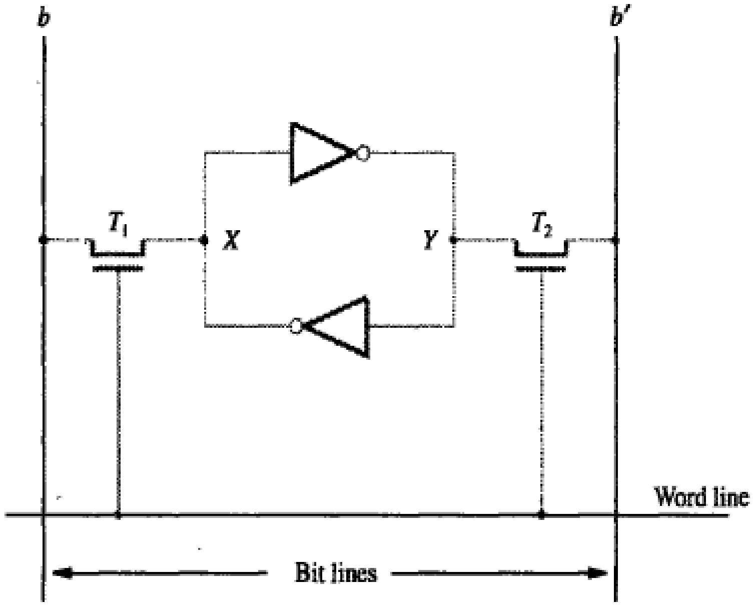


Fig 4.2 A Static RAM cell

Asynchronous DRAMs

- Static RAMs (SRAMs):
    - Consist of circuits that are capable of retaining their state as long as the power is applied.
    - Volatile memories, because their contents are lost when power is interrupted.
    - Access times of static RAMs are in the range of few nanoseconds.
    - However, the cost is usually high.
- Dynamic RAMs (DRAMs):
    - Do not retain their state indefinitely.
    - Contents must be periodically refreshed.
    - Contents may be refreshed while accessing them for reading.

UNIT IV - THE MEMORY SYSTEM: Some Basic Concepts, Semiconductor RAM Memories, Read-Only Memories, Speed, Size, and Cost, Cache Memories, Performance Considerations, Virtual memories, Memory Management requirements, Secondary Storage.

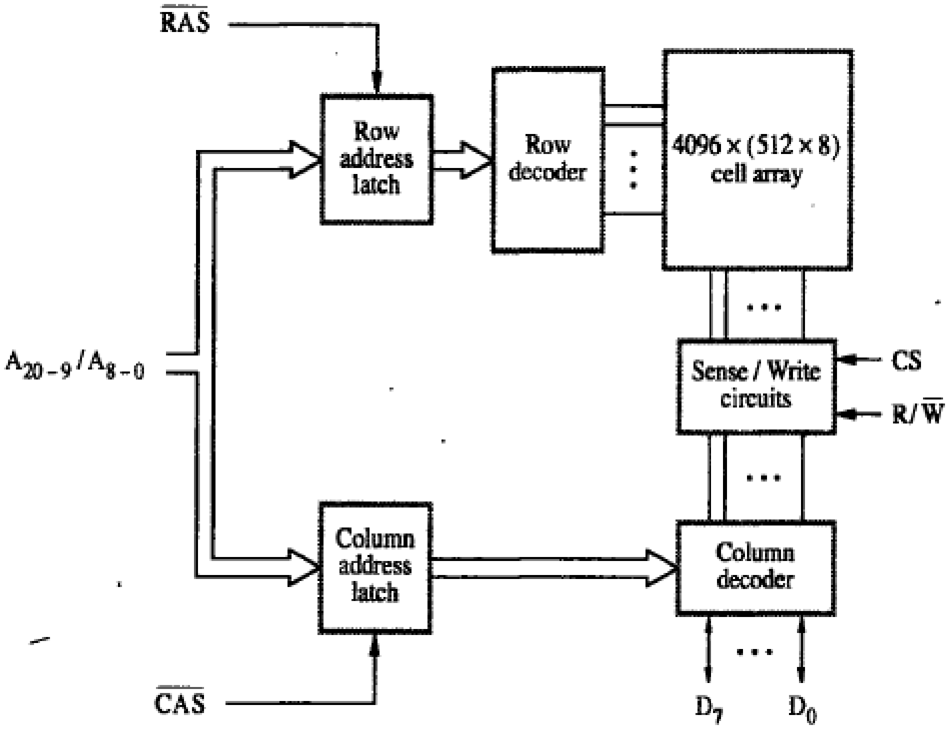


Fig 4.3 Internal organization of a 2M X 8 dynamic memory chip

- Each row can store 512 bytes. 12 bits to select a row, and 9 bits to select a group in a row. Total of 21 bits.
- First apply the row address, RAS signal latches the row address. Then apply the column address, CAS signal latches the address.
- Timing of the memory unit is controlled by a specialized unit which generates RAS and CAS.
- This is asynchronous DRAM

Fast Page Mode

- column addresses can be applied to select and place different bytes on the Suppose if we want to access the consecutive bytes in the selected row.
- This can be done without having to reselect the row.
  - Add a latch at the output of the sense circuits in each row.
  - All the latches are loaded when the row is selected.
  - Different data lines.
- Consecutive sequence of column addresses can be applied under the control signal CAS, without reselecting the row.

- Allows a block of data to be transferred at a much faster rate than random accesses.
- A small collection/group of bytes is usually referred to as a block.
- This transfer capability is referred to as the fast page mode feature.

Synchronous DRAMs

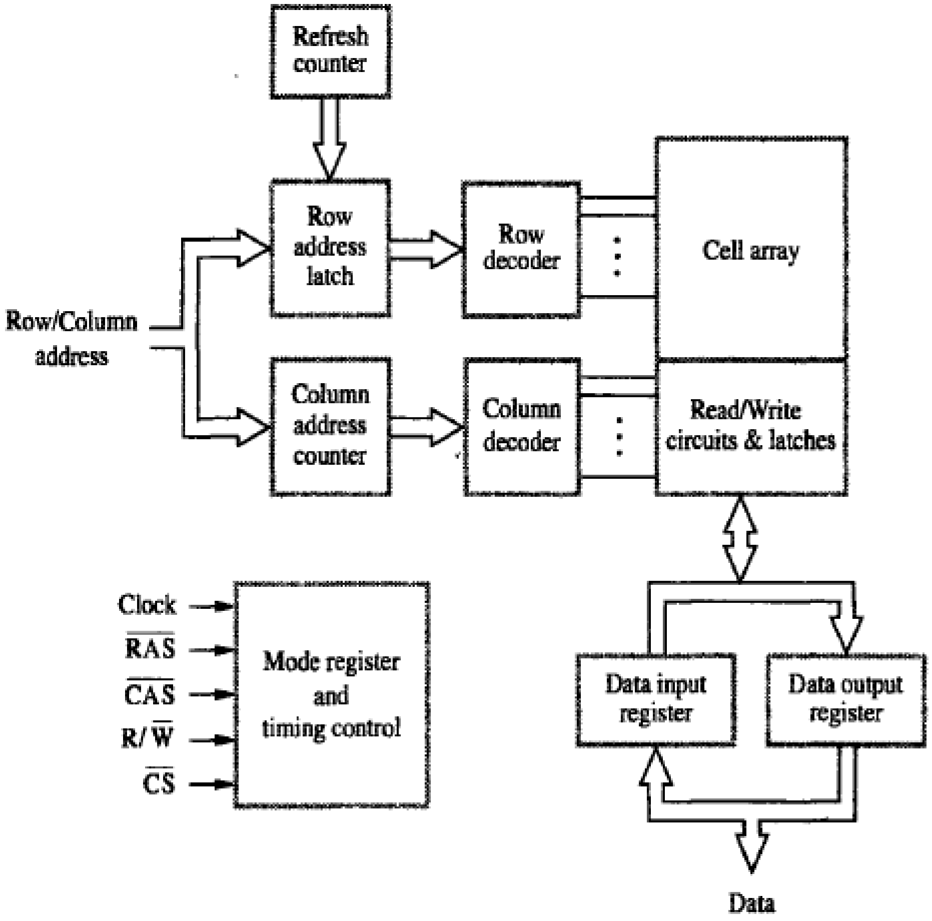


Fig 4.4 Synchronous DRAM.

- Operation is directly synchronized with processor clock signal.
- The outputs of the sense circuits are connected to a latch.
- During a Read operation, the contents of the cells in a row are loaded onto the latches.
- During a refresh operation, the contents of the cells are refreshed without changing the contents of the latches.
- Data held in the latches correspond to the selected columns are transferred to the output.
- For a burst mode of operation, successive columns are selected using column address counter and clock.

- CAS signal need not be generated externally. A new data is placed during raising edge of the clock

Latency, Bandwidth, and DDRSDRAMs

- Memory latency is the time it takes to transfer a word of data to or from memory
- Memory bandwidth is the number of bits or bytes that can be transferred in one second.
- DDRSDRAMs
  - Cell array is organized in two banks

Static memories

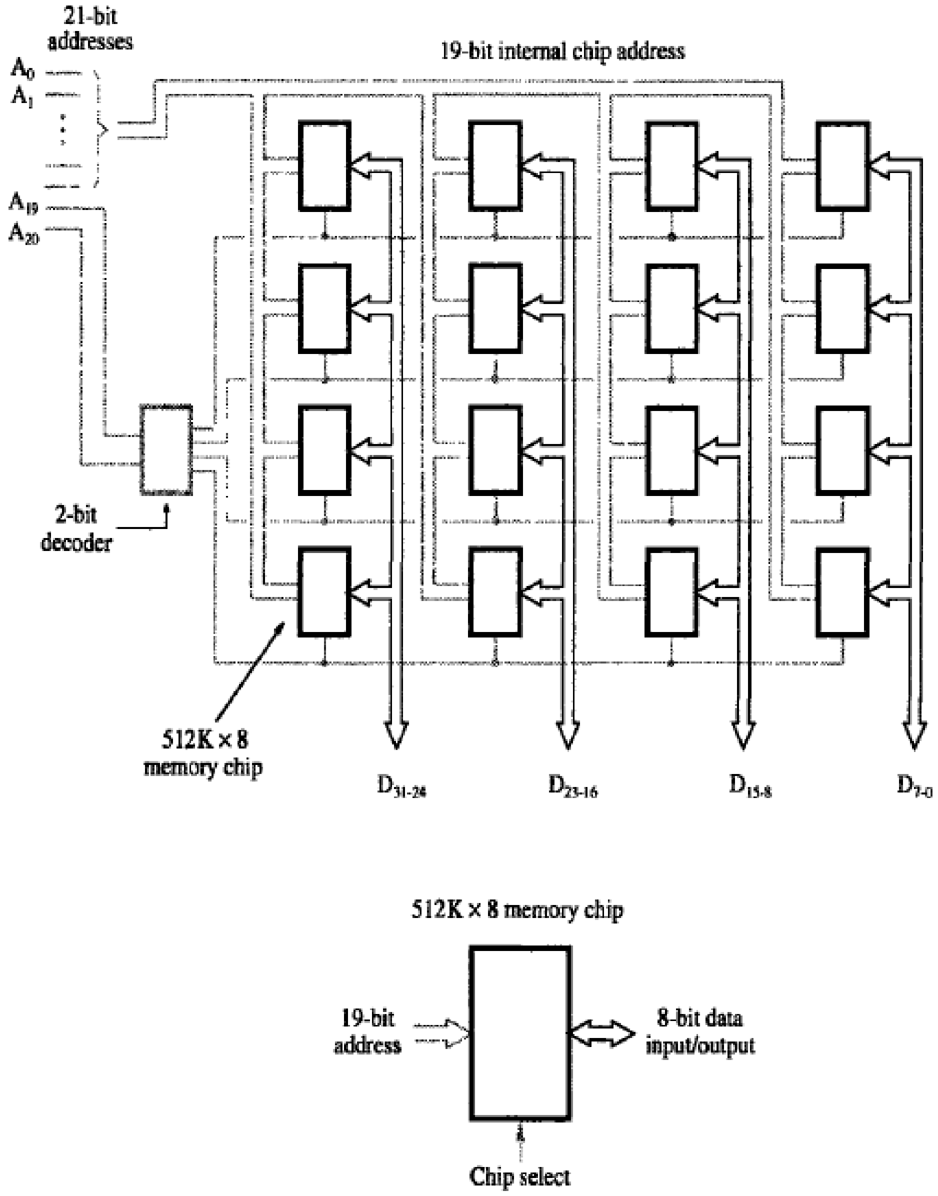


Fig 4.5 static memory chip

- Implement a memory unit of 2M words of 32 bits each.
- Use 512x8 static memory chips.
- Each column consists of 4 chips.
- Each chip implements one byte position.
- A chip is selected by setting its chip select control line to 1.
- Selected chip places its data on the data output line, outputs of other chips are in high impedance state.
- 21 bits to address a 32-bit word.
- High order 2 bits are needed to select the row, by activating the four Chip Select signals.
- 19 bits are used to access specific byte locations inside the selectedchip.

Dynamic memories

- Large dynamic memory systems can be implemented using DRAM chips in a similar way to static memory systems.
- Placing large memory systems directly on the motherboard will occupy a large amount of space.
    - Also, this arrangement is inflexible since the memory system cannot be expanded easily.
- Packaging considerations have led to the development of larger memory units known as SIMMs (Single In-line Memory Modules) and DIMMs (Dual In-line Memory Modules).
- Memory modules are an assembly of memory chips on a small board that plugs vertically onto a single socket on the motherboard.
    - Occupy less space on the motherboard.
    - Allows for easy expansion by replacement.

Memory controller

- Recall that in a dynamic memory chip, to reduce the number of pins, multiplexed addresses are used.
- Address is divided into two parts:
    - High-order address bits select a row in the array.
    - They are provided first, and latched using RAS signal.
    - Low-order address bits select a column in the row.
    - They are provided later, and latched using CAS signal.
- However, a processor issues all address bits at the same time.
- In order to achieve the multiplexing, memory controller circuit is inserted between the processor and memory.

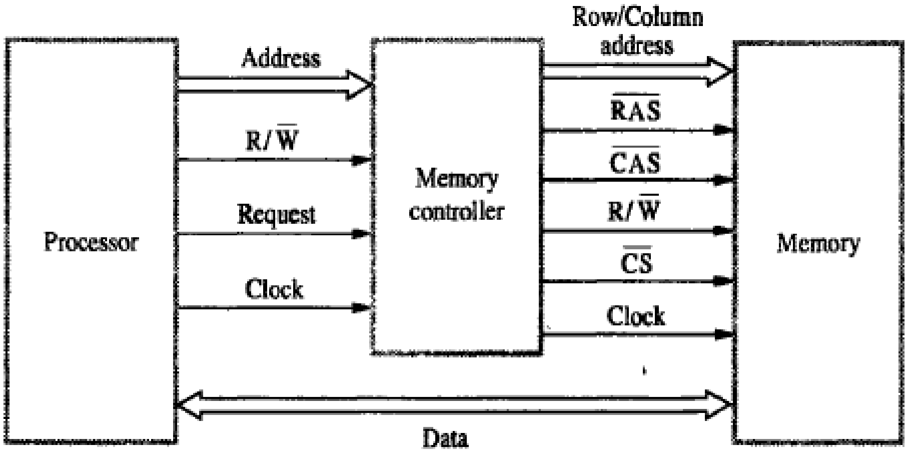


Fig 4.6 Use of Memory controller

READ-ONLY MEMORIES

Describe about the types of ROM. (11 Marks Dec 2015)

- SRAM and SDRAM chips are volatile:
  - Lose the contents when the power is turned off.
- Many applications need memory devices to retain contents after the power is turned off.
  - For example, computer is turned on, the operating system must be loaded from the disk into the memory.
  - Store instructions which would load the OS from the disk.
  - Need to store these instructions so that they will not be lost after the power is turned off.
  - We need to store the instructions into a non-volatile memory.
- Non-volatile memory is read in the same manner as volatile memory.
  - Separate writing process is needed to place information in this memory.
  - Normal operation involves only reading of data, this type of memory is called Read-Only memory (ROM).
- Read-Only Memory:
  - Data are written into a ROM when it is manufactured.
- Programmable Read-Only Memory (PROM):
  - Allow the data to be loaded by a user.
  - Process of inserting the data is irreversible.
  - Storing information specific to a user in a ROM is expensive.
  - Providing programming capability to a user may be better.
- Erasable Programmable Read-Only Memory (EPROM):
  - Stored data to be erased and new data to be loaded.

  - Flexibility, useful during the development phase of digital systems.
  - Erasable, reprogrammable ROM.

Erasure requires exposing the ROM to UV light

- Electrically Erasable Programmable Read-Only Memory (EEPROM):
  - To erase the contents of EPROMs, they have to be exposed to ultraviolet light.
  - Physically removed from the circuit.
  - EEPROMs the contents can be stored and erased electrically.
- Flash memory:
  - Has similar approach to EEPROM.
  - Read the contents of a single cell, but write the contents of an entire block of cells.
  - Flash devices have greater density.
    - Higher capacity and low storage cost per bit.
  - Power consumption of flash memory is very low, making it attractive for use in equipment that is battery-driven.
  - Single flash chips are not sufficiently large, so larger memory modules are implemented using flash cards and flash drives.

SPEED, SIZE, AND COST

Explain the memory hierarchy with respect to speed, size and cost.(11 Marks Nov 2015)

- A big challenge in the design of a computer system is to provide a sufficiently large memory, with a reasonable speed at an affordable cost.
- Static RAM:
  - Very fast, but expensive, because a basic SRAM cell has a complex circuit making it impossible to pack a large number of cells onto a single chip.
- Dynamic RAM:
  - Simpler basic cell circuit, hence are much less expensive, but significantly slower than SRAMs.
- Magnetic disks:
  - Storage provided by DRAMs is higher than SRAMs, but is still less than what is necessary.
  - Secondary storage such as magnetic disks provide a large amount of storage, but is much slower than DRAMs.

UNIT IV - THE MEMORY SYSTEM: Some Basic Concepts, Semiconductor RAM Memories, Read-Only Memories, Speed, Size, and Cost, Cache Memories, Performance Considerations, Virtual memories, Memory Management requirements, Secondary Storage.

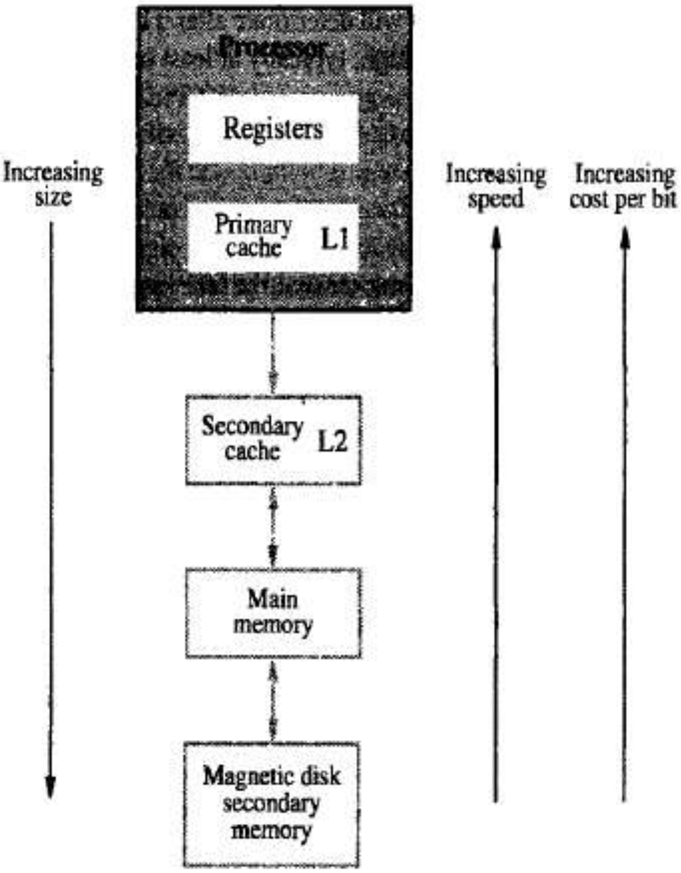


Fig 4.6 Memory Hierarchy

- Fastest access is to the data held in processor registers. Registers are at the top of the memory hierarchy.
- Relatively small amount of memory that can be implemented on the processor chip. This is processor cache.
- Two levels of cache. Level 1 (L1) cache is on the processor chip. Level 2 (L2) cache is in between main memory and processor.
- Next level is main memory, implemented as SIMMs. Much larger, but much slower than cache memory.
- Next level is magnetic disks. Huge amount of inexepensive storage.
- Speed of memory access is critical, the idea is to bring instructions and data that will be used in the near future as close to the processor as possible.

UNIT IV - THE MEMORY SYSTEM: Some Basic Concepts, Semiconductor RAM Memories, Read-Only Memories, Speed, Size, and Cost, Cache Memories, Performance Considerations, Virtual memories, Memory Management requirements, Secondary Storage.

CACHE MEMORIES

Explain the various mapping techniques associated with cache memories with example.(11 Marks April 2015,Dec 2014)

- Processor is much faster than the main memory.
    - As a result, the processor has to spend much of its time waiting while instructions and data are being fetched from the main memory.
    - Major obstacle towards achieving good performance.
- Speed of the main memory cannot be increased beyond a certain point.
- Cache memory is an architectural arrangement which makes the main memory appear faster to the processor than it really is.
- Cache memory is based on the property of computer programs known as "locality of reference".
- Analysis of programs indicates that many instructions in localized areas of a program are executed repeatedly during some period of time, while the others are accessed relatively less frequently.
    - These instructions may be the ones in a loop, nested loop or few procedures calling each other repeatedly.
    - This is called "locality of reference".
- Temporal locality of reference:
    - Recently executed instruction is likely to be executed again very soon.
- Spatial locality of reference:
    - Instructions with addresses close to a recently instruction are likely to be executed soon.

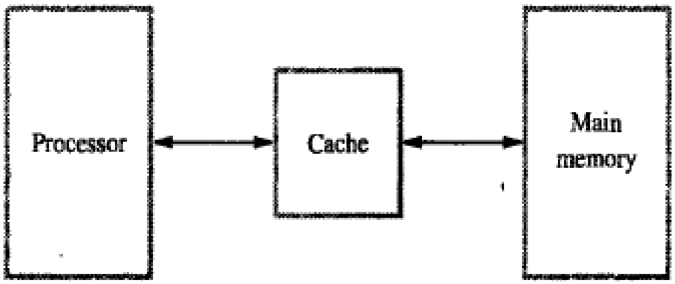


Fig 4.7 Use of a cache memory

- Processor issues a Read request, a block of words is transferred from the main memory to the cache, one word at a time.
- Subsequent references to the data in this block of words are found in the cache.

- At any given time, only some blocks in the main memory are held in the cache. Which blocks in the main memory are in the cache is determined by a “mapping function”.
- When the cache is full, and a block of words needs to be transferred from the main memory, some block of words in the cache must be replaced. This is determined by a “replacement algorithm”.

Cache hit

- Existence of a cache is transparent to the processor. The processor issues Read and Write requests in the same manner.
- If the data is in the cache it is called a Read or Write hit.
- Read hit:
    - The data is obtained from the cache.
- Write hit:
    - Cache has a replica of the contents of the main memory.
    - Contents of the cache and the main memory may be updated simultaneously. This is the write-through protocol.

Update the contents of the cache, and mark it as updated by setting a bit known as the dirty bit or modified bit. The contents of the main memory are updated when this block is replaced. This is write-back or copy-back protocol

Cache miss

- If the data is not present in the cache, then a Read miss or Write miss occurs.
- Read miss:
    - Block of words containing this requested word is transferred from the memory.
    - After the block is transferred, the desired word is forwarded to the processor.
    - The desired word may also be forwarded to the processor as soon as it is transferred without waiting for the entire block to be transferred. This is called load-through or early-restart.
- Write-miss:
    - Write-through protocol is used, then the contents of the main memory are updated directly.

  - If write-back protocol is used, the block containing the addressed word is first brought into the cache. The desired word is overwritten with new information.

Cache Coherence Problem

- A bit called as “valid bit” is provided for each block.
- If the block contains valid data, then the bit is set to 1, else it is 0.
- Valid bits are set to 0, when the power is just turned on.
- When a block is loaded into the cache for the first time, the valid bit is set to 1.
- Data transfers between main memory and disk occur directly bypassing the cache.
- When the data on a disk changes, the main memory block is also updated.
- However, if the data is also resident in the cache, then the valid bit is set to 0.
- What happens if the data in the disk and main memory changes and the write-back protocol is being used?
- In this case, the data in the cache may also have changed and is indicated by the dirty bit.
- The copies of the data in the cache, and the main memory are different. This is called the cache coherence problem.
- One option is to force a write-back before the main memory is updated from the disk.

Mapping functions

- Mapping functions determine how memory blocks are placed in the cache.
- A simple processor example:
  - Cache consisting of 128 blocks of 16 words each.
  - Total size of cache is 2048 (2K) words.
  - Main memory is addressable by a 16-bit address.
  - Main memory has 64K words.
  - Main memory has 4K blocks of 16 words each.
- Three mapping functions:
  - Direct mapping
  - Associative mapping
  - Set-associative mapping.

Direct mapping

- Block j of the main memory maps to j modulo 128 of the cache.0 maps to 0, 129 maps to l.

- More than one memory block is mapped onto the same position in the cache.

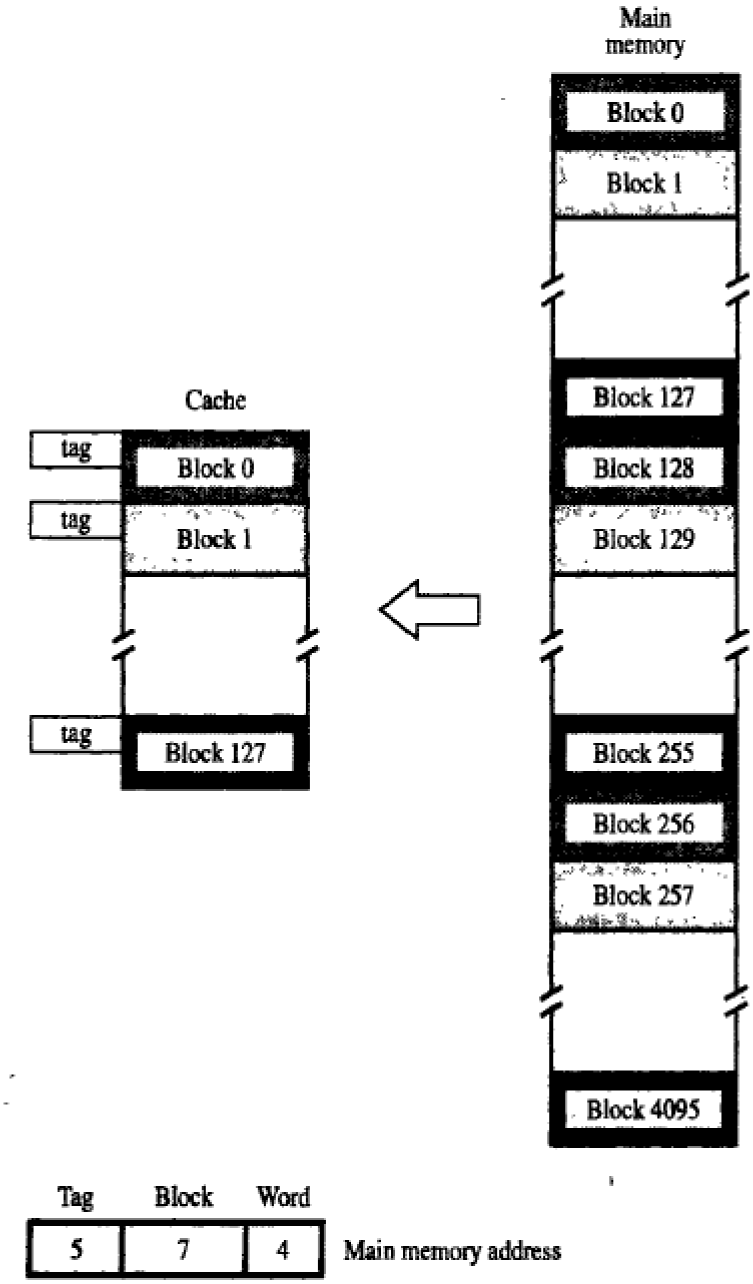


Fig 4.8 Direct mapped cache

- May lead to contention for cache blocks even if the cache is not full.
- Resolve the contention by allowing new block to replace the old block, leading to a trivial replacement algorithm.
- Memory address is divided into three fields:

- Low order 4 bits determine one of the 16 words in a block.

- When a new block is brought into the cache,the the next 7 bits determine which cache block this new block is placed in.

- High order 5 bits determine which of the possible 32 blocks is currently present in the cache. These are tag bits.

- Simple to implement but not very flexible.

Associative mapping

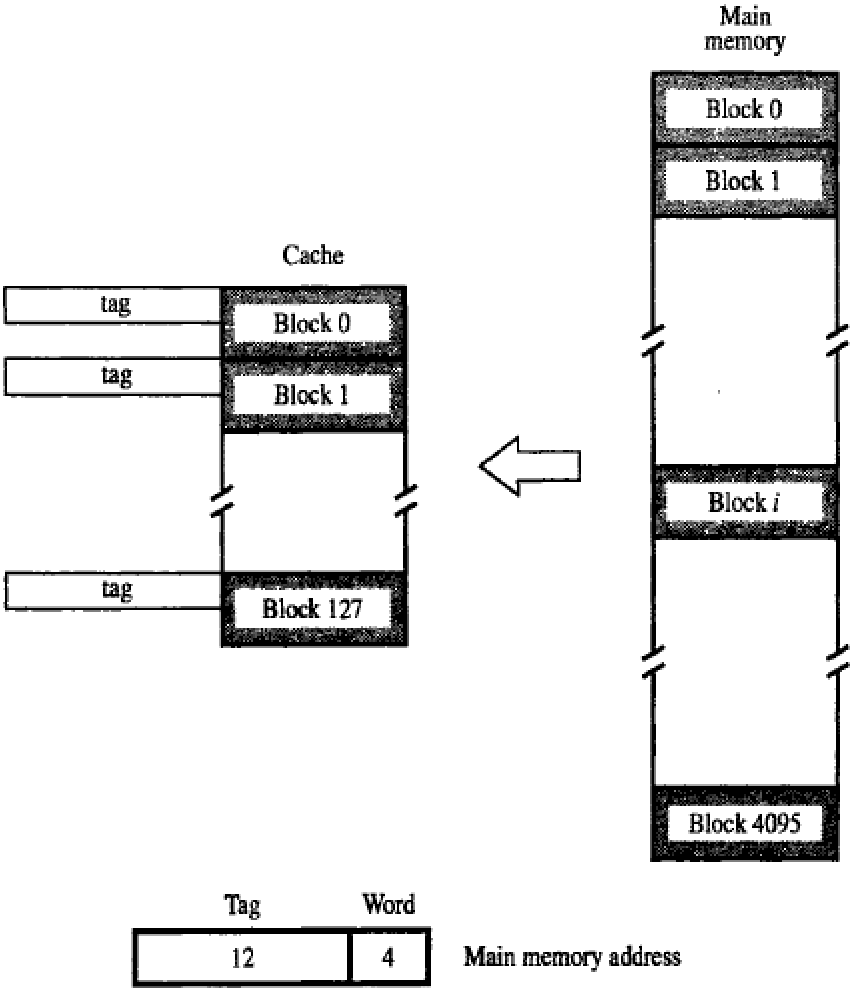


Fig 4.9 Associative- mapped cache

- Main memory block can be placed into any cache position.
- Memory address is divided into two fields:

- Low order 4 bits identify the word within a block.

- High order 12 bits or tag bits identify a memory block when it is resident in the cache.

- Flexible, and uses cache space efficiently.
- Replacement algorithms can be used to replace an existing block in the cache when the cache is full.
- Cost is higher than direct-mapped cache because of the need to search all 128 patterns to determine whether a given block is in the cache.

Set-Associative mapping

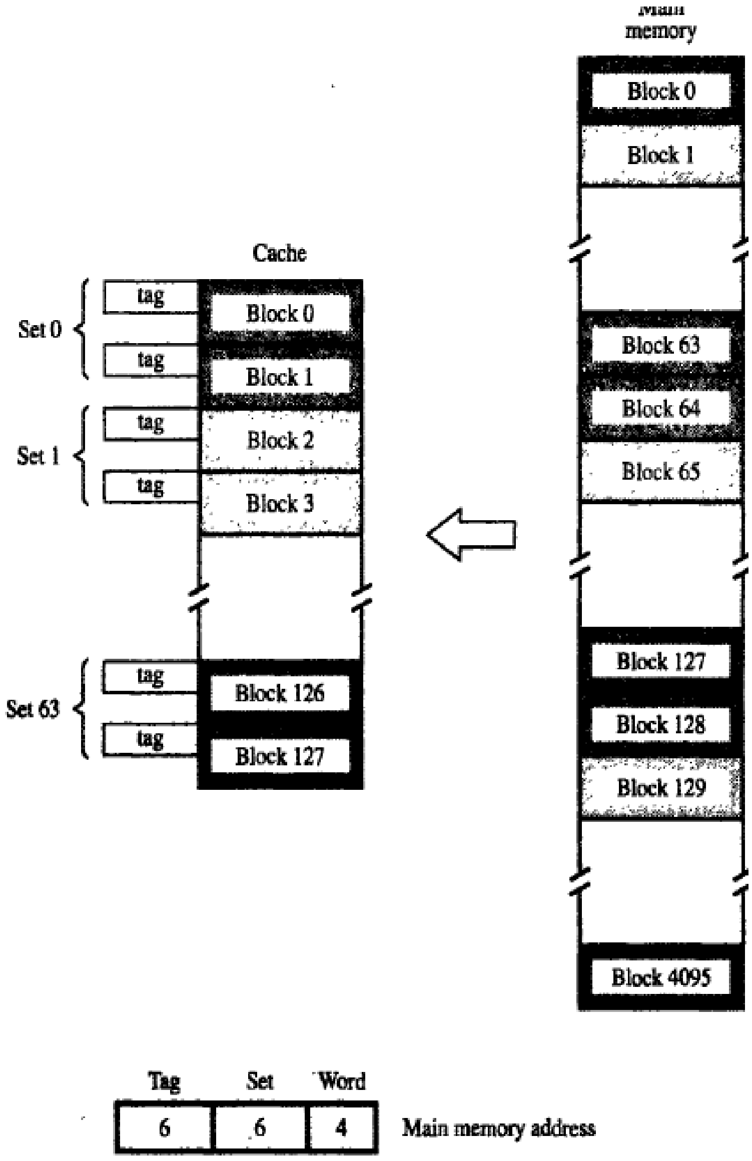


Fig 4.10 Set-associative mapped cache with two blocks per set

- Blocks of cache are grouped into sets.
- Mapping function allows a block of the main memory to reside in any block of a specific set.
- Divide the cache into 64 sets, with two blocks per set.
- Memory block 0, 64, 128 etc. map to block 0, and they can occupy either of the two positions.
- Memory address is divided into three fields:
    - 6 bit field determines the set number.
    - High order 6 bit fields are compared to the tag
- fields of the two blocks in a set.
- Set-associative mapping combination of direct and associative mapping.
- Number of blocks per set is a design parameter.
    - One extreme is to have all the blocks in one set, requiring no set bits (fully associative mapping).
    - Other extreme is to have one block per set, is the same as direct mapping.

PERFORMANCE CONSIDERATIONS

- A key design objective of a computer system is to achieve the best possible performance at the lowest possible cost.
    - Price/performance ratio is a common measure of success.
- Performance of a processor depends on:
    - How fast machine instructions can be brought into the processor for execution.
    - How fast the instructions can be executed.

Interleaving

- Divides the memory system into a number of memory modules. Each module has its own address buffer register (ABR) and data buffer register (DBR).
- Arranges addressing so that successive words in the address space are placed in different modules.
- When requests for memory access involve consecutive addresses, the access will be to different modules.
- Since parallel access to these modules is possible, the average rate of fetching words from the Main Memory can be increased.

UNIT IV - THE MEMORY SYSTEM: Some Basic Concepts, Semiconductor RAM Memories, Read-Only Memories, Speed, Size, and Cost, Cache Memories, Performance Considerations, Virtual memories, Memory Management requirements, Secondary Storage.

Methods of address layouts

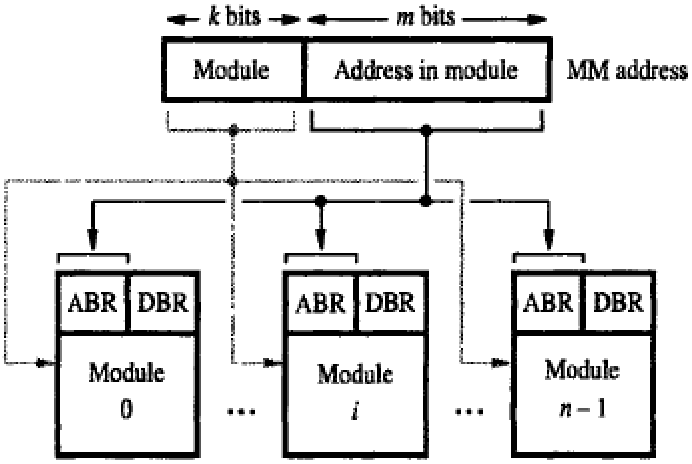


Fig 4.11 Consecutive words in a module

- Consecutive words are placed in a module.
- High-order k bits of a memory address determine the module.
- Low-order m bits of a memory address determine the word within a module.
- When a block of words is transferred from main memory to cache, only one module is busy at a time.

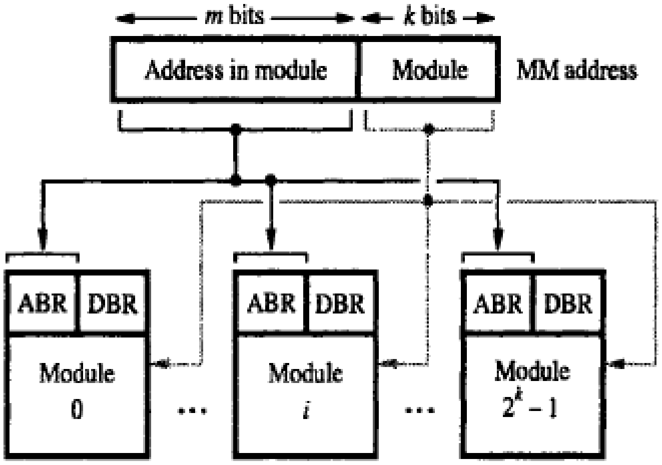


Fig 4.12 consecutive words in a consecutive modules

- Consecutive words are located in consecutive modules.
- Consecutive addresses can be located in consecutive modules.
- While transferring a block of data, several memory modules can be kept busy at the same time.

Hit Rate and Miss Penalty

- Hit rate
- Miss penalty
- Hit rate can be improved by increasing block size, while keeping cache size constant
- Block sizes that are neither very small nor very large give best results.
- Miss penalty can be reduced if load-through approach is used when loading new blocks into cache.

Caches on the processor chip

- In high performance processors 2 levels of caches are normally used.
- Avg access time in a system with 2 levels of caches is

$T_{ave} = h1c1+(1-h1)h2c2+(1-h1)(1-h2)M$

Other Performance Enhancements

Write buffer

- Write-through:
- Each write operation involves writing to the main memory.
- If the processor has to wait for the write operation to be complete, it slows down the processor.
- Processor does not depend on the results of the write operation.
- Write buffer can be included for temporary storage of write requests.
- Processor places each write request into the buffer and continues execution.
- If a subsequent Read request references data which is still in the write buffer, then this data is referenced in the write buffer.
- Write-back:
- Block is written back to the main memory when it is replaced.
- If the processor waits for this write to complete, before reading the new block, it is slowed down.
- Fast write buffer can hold the block to be written, and the new block can be read first.

Prefetching

- New data are brought into the processor when they are first needed.
- Processor has to wait before the data transfer is complete.
- Prefetch the data into the cache before they are actually needed, or a before a Read miss occurs.
- Prefetching can be accomplished through software by including a special instruction in the machine language of the processor.
    - Inclusion of prefetch instructions increases the length of the programs.

- Prefetching can also be accomplished using hardware:
    - Circuitry that attempts to discover patterns in memory references and then prefetches according to this pattern.

<u>Lockup-Free Cache</u>

- Prefetching scheme does not work if it stops other accesses to the cache until the prefetch is completed.
- A cache of this type is said to be "locked" while it services a miss.
- Cache structure which supports multiple outstanding misses is called a lockup free cache.
- Since only one miss can be serviced at a time, a lockup free cache must include circuits that keep track of all the outstanding misses.
- Special registers may hold the necessary
  information about these misses.

VIRTUAL MEMORIES

Explain how the virtual address is converted into real address in a paged virtual memory system.(4 Marks April 2015,Nov 2015)

- An important challenge in the design of a computer system is to provide a large, fast memory system at an affordable cost.
- Architectural solutions to increase the effective speed and size of the memory system.
- Cache memories were developed to increase the effective speed of the memory system.
- <u>Virtual memory</u> is an architectural solution to increase the effective size of the memory system.
- Recall that the addressable memory space depends on the number of address bits in a computer.
    - For example, if a computer issues 32-bit addresses, the addressable memory space is 4G bytes.
- Physical main memory in a computer is generally not as large as the entire possible addressable space.
    - Physical memory typically ranges from a few hundred megabytes to 1G bytes.
- Large programs that cannot fit completely into the main memory have their parts stored on secondary storage devices such as magnetic disks.
    - Pieces of programs must be transferred to the main memory from secondary storage before they can be executed.
- When a new piece of a program is to be transferred to the main memory, and the main memory is full, then some other piece in the main memory must be replaced.
    - Recall this is very similar to what we studied in case of cache memories.
- Operating system automatically transfers data between the main memory and secondary storage.
    - Application programmer need not be concerned with this transfer.

  - Also, application programmer does not need to be aware of the limitations imposed by the available physical memory.

- Techniques that automatically move program and data between main memory and secondary storage when they are required for execution are called virtual-memory techniques.
- Programs and processors reference an instruction or data independent of the size of the main memory.
- Processor issues binary addresses for instructions and data.
  - These binary addresses are called logical or virtual addresses.
- Virtual addresses are translated into physical addresses by a combination of hardware and software subsystems.
  - If virtual address refers to a part of the program that is currently in the main memory, it is accessed immediately.
  - If the address refers to a part of the program that is not currently in the main memory, it is first transferred to the main memory before it can be used.

Virtual memory organization

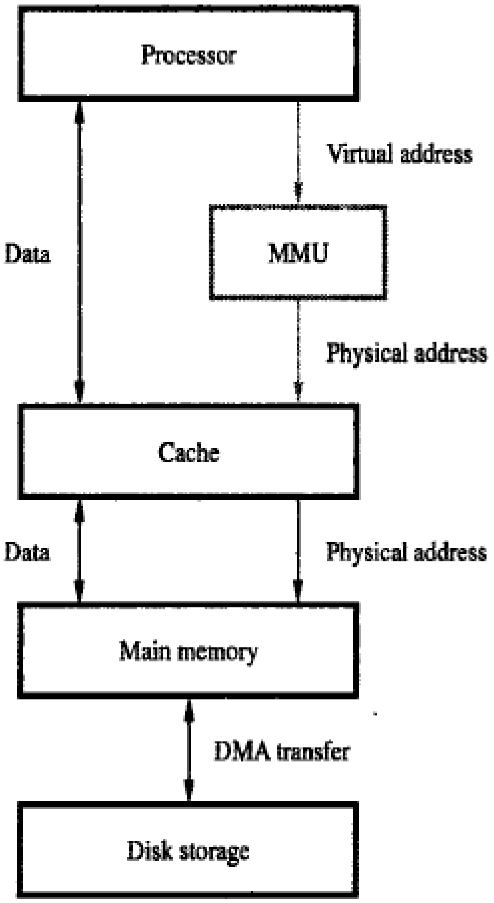


Fig 4.13 virtual memory organization

- Memory management unit (MMU) translates virtual addresses into physical addresses.
- If the desired data or instructions are in the main memory they are fetched as described previously.
- If the desired data or instructions are not in the main memory, they must be transferred from secondary storage to the main memory.
- MMU causes the operating system to bring the data from the secondary storage into the main memory.

Address translation

- Assume that program and data are composed of fixed-length units called pages.
- A page consists of a block of words that occupy contiguous locations in the main memory.
- Page is a basic unit of information that is transferred between secondary storage and main memory.
- Size of a page commonly ranges from 2K to 16K bytes.
  - Pages should not be too small, because the access time of a secondary storage device is much larger than the main memory.
  - Pages should not be too large, else a large portion of the page may not be used, and it will occupy valuable space in the main memory.
- Concepts of virtual memory are similar to the concepts of cache memory.
- Cache memory:
  - Introduced to bridge the speed gap between the processor and the main memory.
  - Implemented in hardware.
- Virtual memory:
  - Introduced to bridge the speed gap between the main memory and secondary storage.
  - Implemented in part by software.
- Each virtual or logical address generated by a processor is interpreted as a virtual page number (high-order bits) plus an offset (low-order bits) that specifies the location of a particular byte within that page.
- Information about the main memory location of each page is kept in the page table.
  - Main memory address where the page is stored.
  - Current status of the page.
- Area of the main memory that can hold a page is called as page frame.
- Starting address of the page table is kept in a page table base register.
- Virtual page number generated by the processor is added to the contents of the page table base register.
  - This provides the address of the corresponding entry in the page table.

- The contents of this location in the page table give the starting address of the page if the page is currently in the main memory.

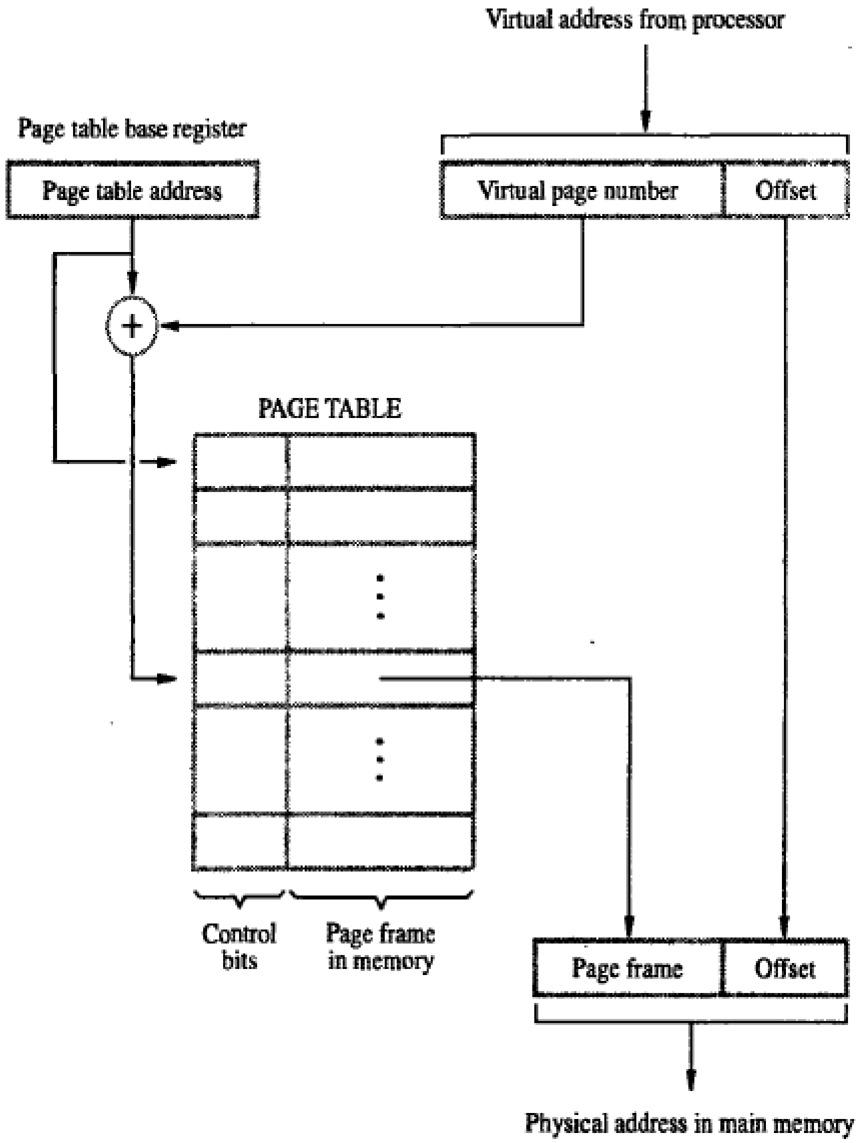


Fig 4.14 virtual memory address translation

- Page table entry for a page also includes some control bits which describe the status of the page while it is in the main memory.

- One bit indicates the validity of the page.
    - Indicates whether the page is actually loaded into the main memory.
    - Allows the operating system to invalidate the page without actually removing it.

- One bit indicates whether the page has been modified during its residency in the main memory.
    - This bit determines whether the page should be written back to the disk when it is removed from the main memory.
    - Similar to the dirty or modified bit in case of cache memory.
- Other control bits for various other types of restrictions that may be imposed.
    - For example, a program may only have read permission for a page, but not write or modify permissions.
- Where should the page table be located?
- Recall that the page table is used by the MMU for every read and write access to the memory.
    - Ideal location for the page table is within the MMU.
- Page table is quite large.
- MMU is implemented as part of the processor chip.
- Impossible to include a complete page table on the chip.
- Page table is kept in the main memory.
- A copy of a small portion of the page table can be accommodated within the MMU.
    - Portion consists of page table entries that correspond to the most recently accessed pages.
- A small cache called as Translation Lookaside Buffer (TLB) is included in the MMU.
    - TLB holds page table entries of the most recently accessed pages.
- Recall that cache memory holds most recently accessed blocks from the main memory.
    - Operation of the TLB and page table in the main memory is similar to the operation of the cache and main memory.
- Page table entry for a page includes:
    - Address of the page frame where the page resides in the main memory.
    - Some control bits.

- In addition to the above for each page, TLB must hold the virtual page number for each page.

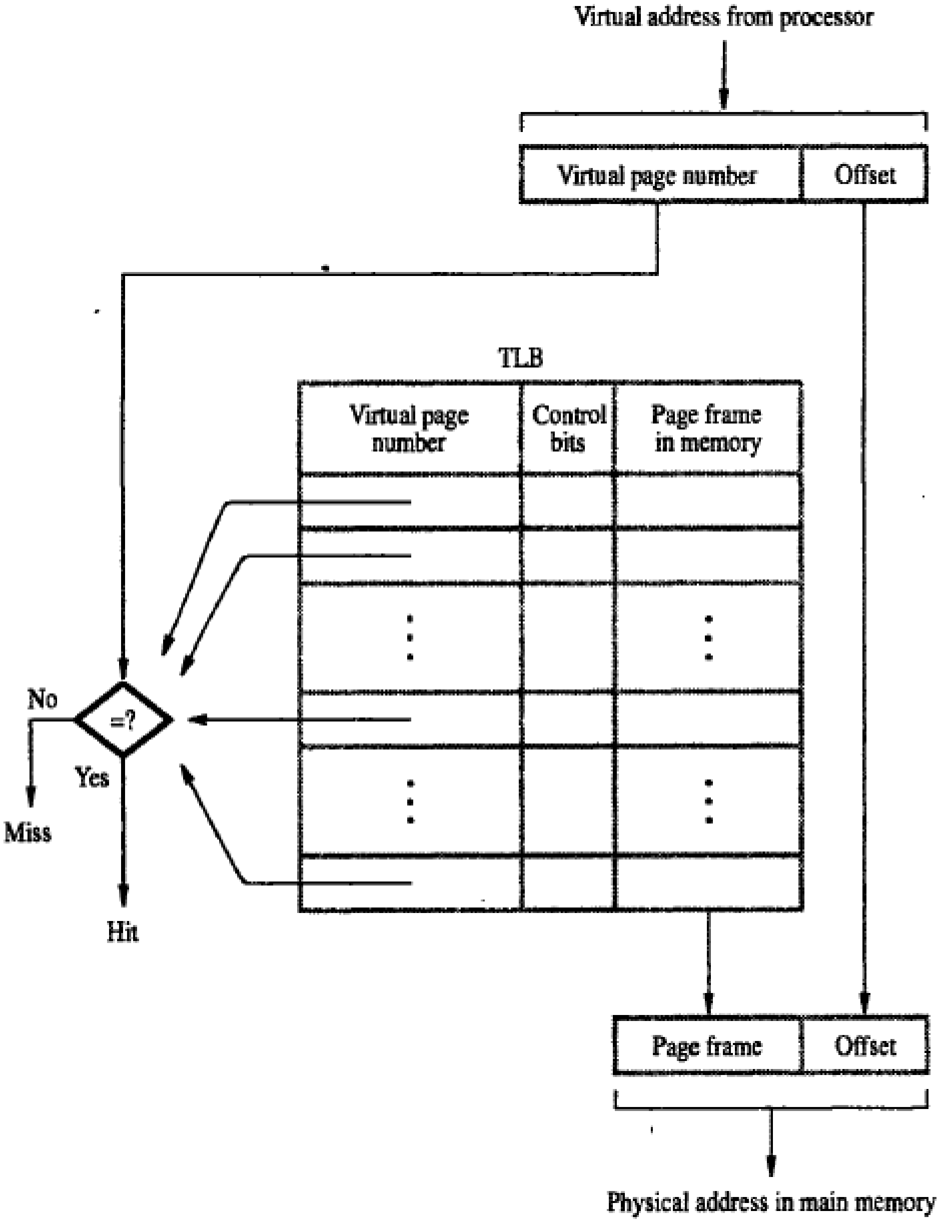


Fig 4.15 Use of an associative – mapped TLB

<u>Associative-mapped TLB</u>

- High-order bits of the virtual address generated by the processor select the virtual page.
- These bits are compared to the virtual page numbers in the TLB.
- If there is a match, a hit occurs and the corresponding address of the page frame is read.

- If there is no match, a miss occurs and the page table within the main memory must be consulted.
- Set-associative mapped TLBs are found in commercial processors
- How to keep the entries of the TLB coherent with the contents of the page table in the main memory?
- Operating system may change the contents of the page table in the main memory.
    - Simultaneously it must also invalidate the corresponding entries in the TLB.
- A control bit is provided in the TLB to invalidate an entry.
- If an entry is invalidated, then the TLB gets the information for that entry from the page table.
    - Follows the same process that it would follow if the entry is not found in the TLB or if a "miss" occurs.
- What happens if a program generates an access to a page that is not in the main memory?
- In this case, a page fault is said to occur.
    - Whole page must be brought into the main memory from the disk, before the execution can proceed.
- Upon detecting a page fault by the MMU, following actions occur:
    - MMU asks the operating system to intervene by raising an exception.
    - Processing of the active task which caused the page fault is interrupted.
    - Control is transferred to the operating system.
    - Operating system copies the requested page from secondary storage to the main memory.
    - Once the page is copied, control is returned to the task which was interrupted.
- Servicing of a page fault requires transferring the requested page from secondary storage to the main memory.
- This transfer may incur a long delay.
- While the page is being transferred the operating system may:

  - Suspend the execution of the task that caused the page fault.
  - Begin execution of another task whose pages are in the main memory.
- Enables efficient use of the processor.
- How to ensure that the interrupted task can continue correctly when it resumes execution?
- There are two possibilities:
  - Execution of the interrupted task must continue from the point where it was interrupted.
  - The instruction must be restarted.
- Which specific option is followed depends on the design of the processor.
- When a new page is to be brought into the main memory from secondary storage, the main memory may be full.
  - Some page from the main memory must be replaced with this new page.
- How to choose which page to replace?
  - This is similar to the replacement that occurs when the cache is full.
  - The principle of locality of reference (?) can also be applied here.
  - A replacement strategy similar to LRU can be applied.
- Since the size of the main memory is relatively larger compared to cache, a relatively large amount of programs and data can be held in the main memory.
  - Minimizes the frequency of transfers between secondary storage and main memory.
- A page may be modified during its residency in the main memory.
- When should the page be written back to the secondary storage?
- Recall that we encountered a similar problem in the context of cache and main memory:
  - Write-through protocol(?)
  - Write-back protocol(?)

- Write-through protocol cannot be used, since it will incur a long delay each time a small amount of data is written to the disk.

MEMORY MANAGEMENT REQUIREMENTS

- Operating system is concerned with transferring programs and data between secondary storage and main memory.
- Operating system needs memory routines in addition to the other routines.
- Operating system routines are assembled into a virtual address space called system space.
- System space is separate from the space in which user application programs reside.
    - This is user space.
- Virtual address space is divided into one

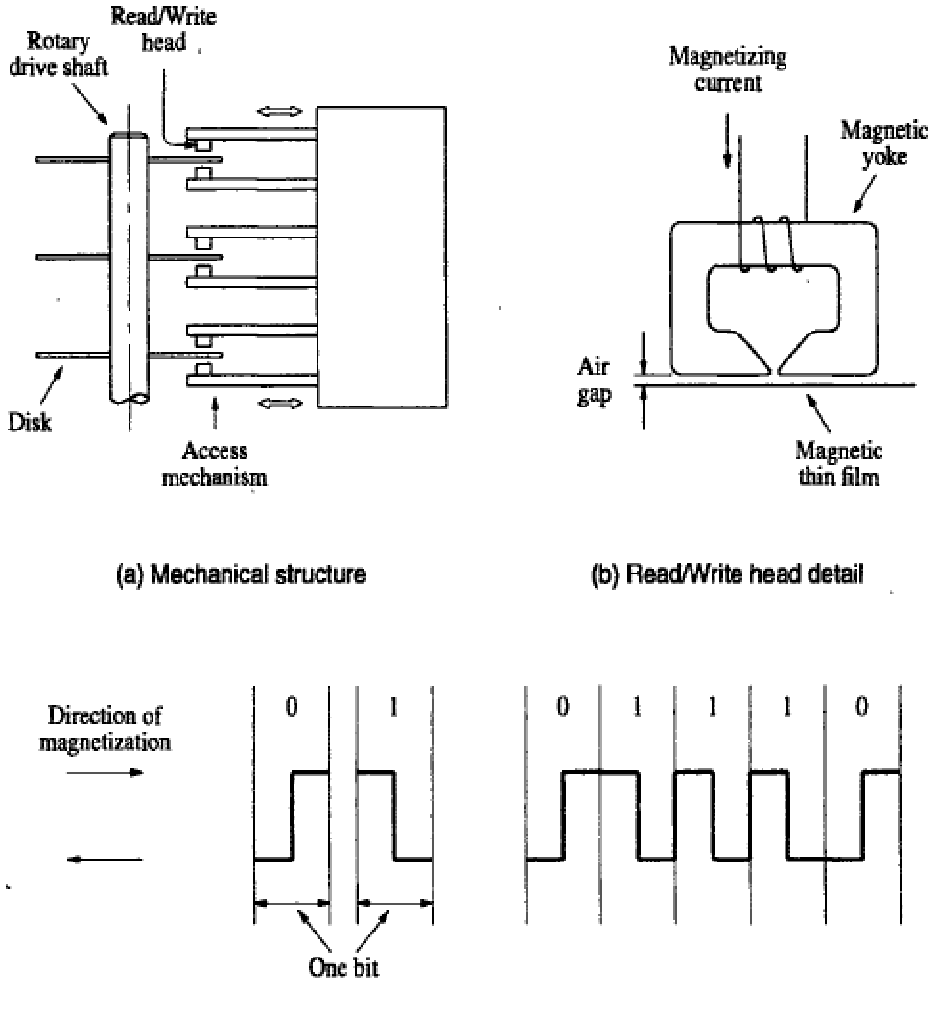
  system space + several user spaces.
- Recall that the Memory Management Unit (MMU) translates logical or virtual addresses into physical addresses.
- MMU uses the contents of the page table base register to determine the address of the page table to be used in the translation.
    - Changing the contents of the page table base register can enable us to use a different page table, and switch from one space to another.
- At any given time, the page table base register can point to one page table.
    - Thus, only one page table can be used in the translation process at a given time.
    - Pages belonging to only one space are accessible at any

  given time.
- When multiple, independent user programs coexist in the main memory, how to ensure that one program does not modify/destroy the contents of the other?
- Processor usually has two states of operation:
    - Supervisor state.
    - User state.

- Supervisor state:
    - Operating system routines are executed.
- User state:
    - User programs are executed.
    - Certain privileged instructions cannot be executed in user state.
    - These privileged instructions include the ones which change page table base register.
    - Prevents one user from accessing the space of other users.

SECONDARY STORAGE

Explain the organization and accessing of data on a disk.(7 Marks April 2015)



(a) Mechanical structure

(b) Read/Write head detail

(c) Bit representation by phase encoding

Fig 4.15 magnetic disk principles

- Disk
- Disk drive
- Disk controller

Organization of Data on a Disk

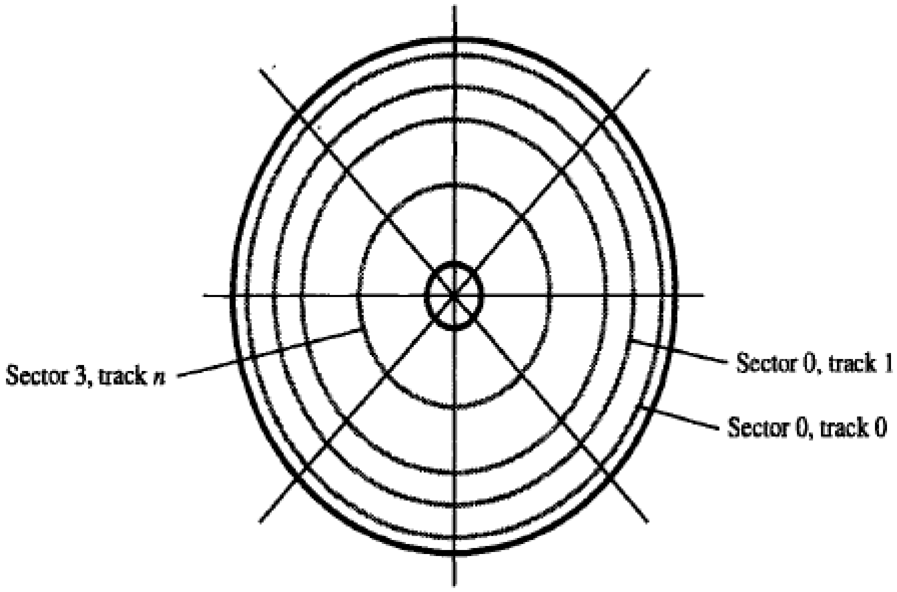


Fig 4.16 organization of one surface of a disk

Access Data on a Disk

- Sector header
- Following the data, there is an error-correction code (ECC).
- Formatting process
- Difference between inner tracks and outer tracks
- Access time – seek time / rotational delay (latency time)
- Data buffer/cache

Disk Controller

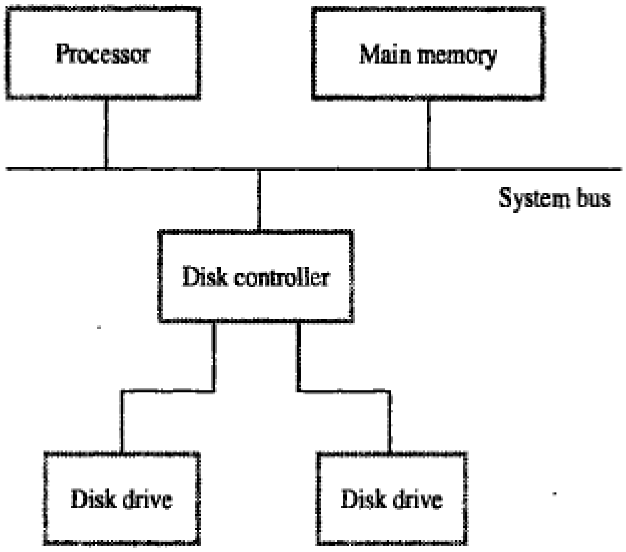


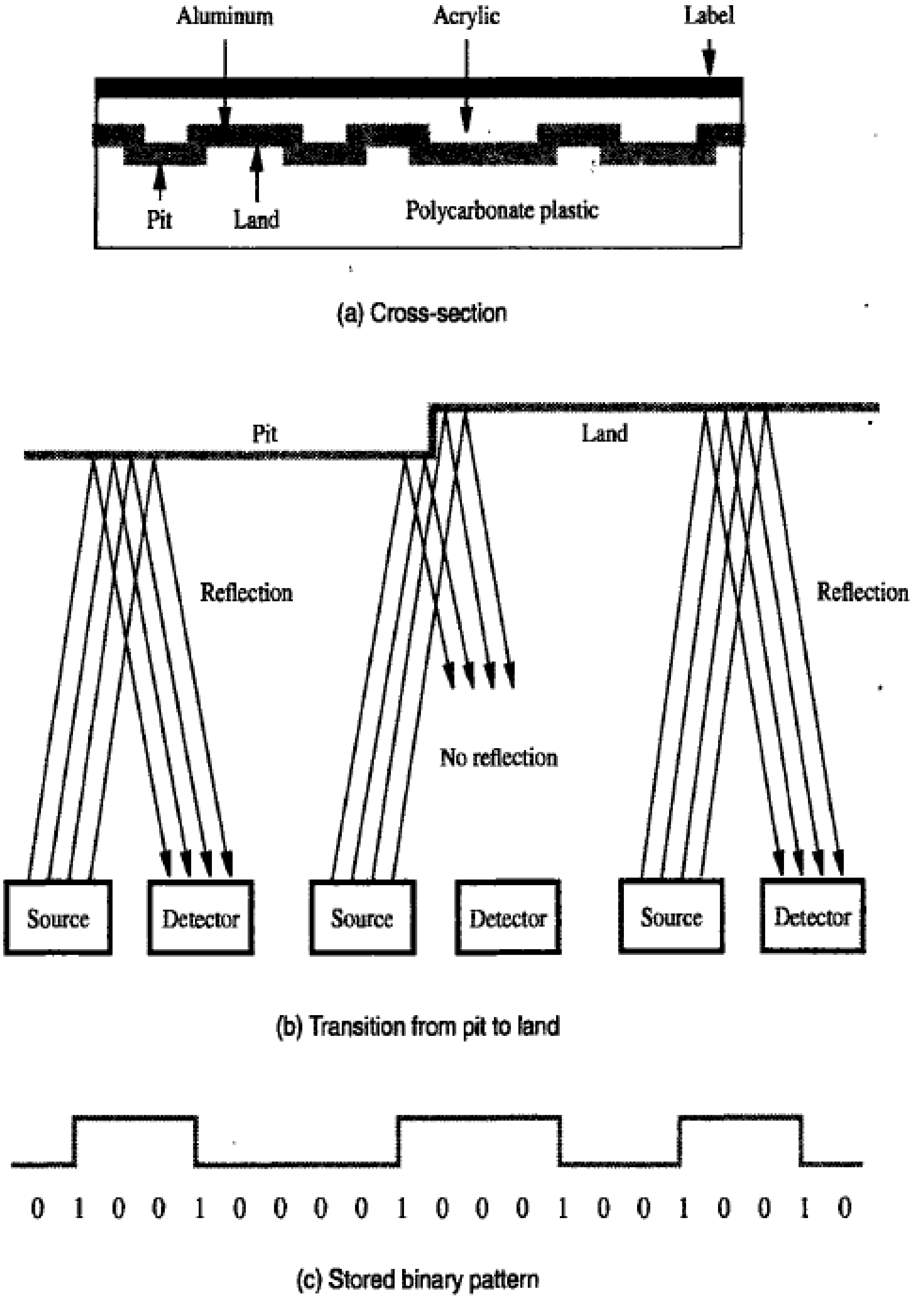
Fig 4.17 disk connected to the system bus.

- Seek
- Read
- Write
- Error checking

RAID Disk Arrays

- Redundant Array of Inexpensive Disks
- Using multiple disks makes it cheaper for huge storage, and also possible to improve the reliability of the overall system.
- RAID0 – data striping
- RAID1 – identical copies of data on two disks
- RAID2, 3, 4 – increased reliability
- RAID5 – parity-based error-recovery

Optical Disks



(a) Cross-section

(b) Transition from pit to land

(c) Stored binary pattern

Fig 4.18 Optical disk

- CD-ROM
- CD-Recordable (CD-R)

- CD-ReWritable (CD-RW)
- DVD
- DVD-RAM

Magnetic Tape Systems

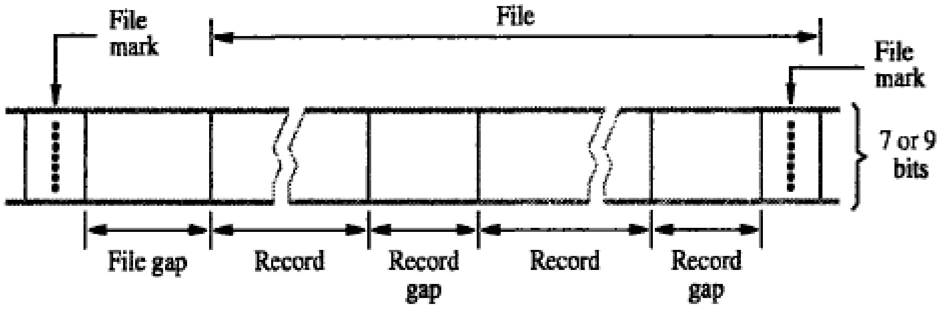
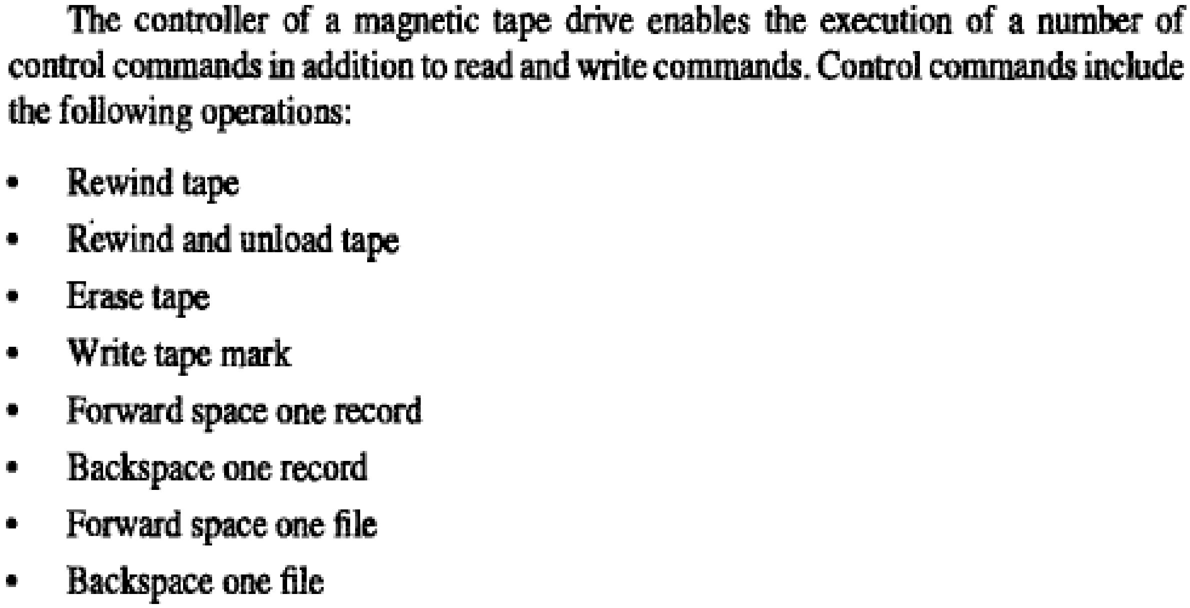


Fig 4.19 organization of data on magnetic tape



The controller of a magnetic tape drive enables the execution of a number of control commands in addition to read and write commands. Control commands include the following operations:

- Rewind tape
- Rewind and unload tape
- Erase tape
- Write tape mark
- Forward space one record
- Backspace one record
- Forward space one file
- Backspace one file

UNIT – V

BASIC PROCESSING UNIT: Some Fundamental Concepts, Execution of a Complete Instruction, Multiple-Bus Organization, Hardwired Control, Micro programmed Control,

PIPELINING: Basic Concepts, Data Hazards, Instruction Hazards, Influence on Instructions Sets, Data path and Control Considerations, Superscalar Operations, Performance Considerations

2 MARKS

1. What do you mean by micro-operation?

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

2. Define Processor.

It executes machine instructions and coordinates the activities of other units. It is also called as instruction set processor or central processing unit (CPU).

3. What is Data path?

The data registers, ALU and the interconnecting bus are referred to as data path.

4. What is meant by program counter?

It is a processor register mainly used for execution. It stores the address of the next instruction to be executed. After fetching an instruction the content of the PC are updated to point to the next instruction in the sequence.

5. Define IR?

IR is an instruction register. To execute an instruction the processor fetches the contents of the memory location pointed by the PC. The contents of this location are interpreted as an instruction to be executed. They are loaded into the IR.

6. What is micro program? (Apr 13)

A sequence of one or more micro operations designed to control specific operation, such as addition, multiplication is called a micro program.

7. What do you mean by hardwired control unit?

In the hardwired control, the control units use fixed logic circuits to interpret instructions and generate control signals from them.

8. Define microinstruction?

It 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 micro instructions, because the number of required signal is usually large. Moreover, only few bits are used in any given instruction .The solution of this problem is to group the control signals.

9. List the two techniques used for grouping of control signals

1. Control signals: IN and OUT signals
2. Gating signals: Read, write, clear A, set carry in, continue operation   etc.

10. Write down the steps to execute an instruction.

- Fetch the contents of the memory location pointed by the PC and store that content into instruction registers. IR← ╔ PC ╗
- Increment the PC value by 4 to point out the next instruction in the program. PC←[PC]+4
- Carry out the actions specified by the instruction in the IR.

11. Define fetch step.

To perform the execution of the instruction we have to fetch the content from the memory and store that content into the processor register IR. This is known as fetching phase.

12. What is meant by execution phase?

Carry out the actions specified by the instruction in the instruction in the instruction register is known as execution instruction

13. Define MAR, MDR?

MAR means memory address register and MDR means memory data registers. These two are the processor registers that can be used in memory read and write operations.

14. Define register transfer and list out the signals used to do it.

As instruction execution involves a sequence of steps in which data are transferred from one register to another register. Two control signals are used to place the contents of the registers on the bus or to load the data on the bus into the registers. The signals are Ri in, Ri out.

15. Write down the control sequence for Move (R1), R2.

The control sequence is:

R1 out, MAR in Read

MDRoutE, WMFC

MDRout, R2 in,

16. Write down the steps to transfer the content of register R1 to register R4.

- Enable the output of register R1 by setting R1 out to 1. This places the contents of R1 on the processor bus.

- Enable the input of register R4 by setting R4 into 1. This loads data from processor bus into register R4.

17. Define multiphase clocking.

In some processor data transfers may use both the rising and falling edges of the clock. Two or more clock signals are needed to guarantee proper transfer of data. This is known as multiphase clocking.

18. Define MFC signal.

To accommodate the validity in response time, the processor waits until it receives an indication that the requested Read operation has been completed. A control signal MFC (Memory Function Complete) is used for this purpose.

19. Write down the steps to execute Add (R3), R1 instruction.

Fetch the instruction

Fetch the first operand

Perform the addition

Load the result into R1.

20. Define register file.

In multi bus architecture all the general purpose registers are called combined into a single clock called as register file.

21. Define interrupt?

CPU supervises the other system components via special control lines. Whenever the CPU receives the signals from the IO device (i.e.) interrupt signals, it suspends the current execution of the program and performs the interrupt request. After process the interrupt request, CPU transfers from supervisor mode to user mode.

22. Define instruction cycle. (Nov 12)

The sequence of operations involved in processing an instruction is called as an instruction cycle. It is divided into two phases: 1.fetch cycle 2. Execution cycle. The instruction is obtained from main memory during the fetch cycle. The execution cycle includes decoding the instruction, fetching any required operands, and performing the operation specified by the instructions opcode.

23. Define Hardwired control?

The circuit is design with the useful goals of minimizing the number of components used and maximizing the speed of operation. Once the unit is constructed, the only way implement changes in control unit behaviors are by redesigning the entire unit. Such a circuit is called hardwired control design.

24. What is the difference between hardwired control and micro' programmed control memory?

Hardwired Control: Implementation of hardwired is using sequential circuits and flip flops. If any change is to be done then the whole design is to be modified.

Micro Program Control:_Micro program is based on microinstruction. If the change is *to* be design then part of program is *to* be modified.

25. What is the difference between horizontal microinstructions and vertical microinstructions

Horizontal Micro Instruction: Ability to express a high degree of parallelism. The length of format is long. Little encoding of control information.

Vertical Micro Instruction: The length of the format is short Limited ability *to* express parallel micro operations. Considerable encoding of control information

26. Define multi-cycle?

ALU processes each m bit slice in K consecutive clock cycles is termed as multi-cycle.

27. Explain load-store architecture?

The program fragment that uses only the "load and store instruction to access memory is called load and store architecture. It is common to allow other instruction to specify operands in memory.

28. How do you measure the speed of a pipeline?

Pipeline speedup s(m) = T(1)1T(m) ,

M-Stage ...

T(m)- The execution time for same target workload on an m-stage pipeline

T(1) - The execute on time for same target workload on a non pipelined processor

29. How do you calculate the performance of the pipeline?

Pipeline's performance *I* cost ratio PCR = *f/k*

f - Clock frequency k - Hardware cost

30. Define Hit ratio.

The performance of cache memory is frequently measured in terms of a quantity called hit ratio. Let N1 and N2 denote the number of references to M1 and M2respectively in the block address stream.The block hit ratio H is defined by H=N1/N1+N2

31. What is the difference between macro and microinstructions?

Macro Instruction: Assign symbolic name to sequence of instructions I

Micro Instruction: Specify low-level micro operations.

32. Explain coprocessor function?

Coprocessor is a separate instruction set processor (ie) closely coupled to the CPU and whose instruction and registers direct extensions of the CPU'S.

33. What is control word?

It is a word whose individual bits represent the various control signals. Control sequence of an instruction defines a unique combination of 1’s and 0’s in the control word.

A sequence of CW’s corresponding to the control sequence of a machine Instruction constitutes the micro routine for that instruction.

34. Define control store.

The micro routines for all instructions in the instructions set of a computer are stored in a special memory called the control store. To read the control words sequentially from the control store, a micro program counter is used.

35. List out the situations that not increment the micro Pc value.

- When a new instruction is loaded into the IR, the micro PC is loaded with the starting address of the micro routine for that instruction.
- When a branch instruction is encountered and the branch condition is satisfied the micro Pc is loaded with the micro Pc is loaded with the branch target address.
- When an End instruction is encountered micro Pc is loaded with the address of the first CW in the micro routine for the instruction fetch cycle.

36. What is the drawback present in micro instruction s representation and how can we eliminate it?

Assigning individual bits to each control signal results in long micro instruction s because the number of required signals is usually large. Moreover only a few bits are set 1. So the available bit space is poorly used. We can overcome this draw back by grouping the relevant control signals.

37. Define vertical organization.

Highly encoded scheme groups more number of instruction s into a single group. So minimum number of groups is enough to represent instruction set. This is known as vertical organization.

38. What is meant by horizontal organization?

Minimally encoded scheme groups minimum number of instruction s into single group. So we need more group to represent the instruction set. This is known as vertical organization.

39. Define bit OR ing technique.

By using this technique we can modify the branch address. It use an Or gate to change the least significant bit of the specified instruction’s address to1, if the addressing mode is used.

40. Why it is need of pre fetch instruction?

One drawback of micro programmed control is the slower operation because of the time it takes to fetch instruction s from the control store. Faster operation is achieved if the next instruction is pre fetched while three current one is being executed.

41. Define emulation.

Programs written in the machine language of M2 can be run on computer M1 that is M1 emulate M2. Emulation allows us to replace absolute equipments.

42. What is meant by micro programmed control?

In some processor the control signals are generated by a program similar to machine language programs. This is known as micro programmed control.

43. Comparison between Hardwired and Micro programmed control control

Attribute	Hardwired control	Micro programmed control
Speed	Fast	Slow
Ability to handle large	Somewhat difficult	Easier
Design process	Somewhat complicated	Orderly and systematic
Applications	Mostly RISC microprocessors	Mainframe ,some microprocessors

44. Write the register transfer sequence for storing a word in memory.

Writing word into a memory location, the derived address is loaded into MAR. Then the data can be written are loaded into MDR and a write command is issued.

Hence executing the instruction MOV R2, (R1) requires the **following sequence**

R1out, MAR in

R2out MDR in, write

MDR out E, WMFC.

The processor remains in step3 until the memory operation is completed and as MFC response is received.

45. What is a micro program sequencer?

If each machine instruction is implemented by a microinstruction using a bit for each control word, a micro-program counter is sufficient to control sequencing.

Advantage:

Writing micro program is fairly simple because standard software techniques can be used.

Disadvantage:

Two major disadvantages exist.

Large number of microinstructions and large control store

Execution time is larger.

Consider a more complicated example of a complex machine instruction

ADD src,Rdst which adds the source operand to the contents of the destination register and result will be stored in the destination register.

Assume that source operand can be specified in the following addressing modes

register 2) autoincrement 3)autodecrement 4)indexed as well as the indirect forms of these four modes

46. What are the sequences of operations involved in processing an instruction constitutes an instruction cycle?

The sequence of operations involved in processing an instruction constitutes an instruction cycle, which can be subdivided into 3 major phases:

1. Fetch cycle

2. Decode cycle

3. Execute cycle

47. What are advantage and disadvantage of hardwired control and Micro programmed control?

Advantages of Micro programmed control

It simplifies the design of control unit. Thus it is both, cheaper and less error prone implement.

Control functions are implemented in software rather than hardware.

Disadvantages

A micro programmed control unit is somewhat slower than the hardwired control unit, because time is required to access the microinstructions from CM.

48. What is the address sequencing capabilities required in control memory?

Each microinstruction should explicitly or implicitly specify the next micro instruction to be used; such address sequencing capabilities are required in the control memory

49. In what ways Width and Height of the control memory can be reduced?

To reduce the number of pins, the dynamic memory chips use multiplexed address inputs. The address is divided into two parts.

High-order address bits-select a row in the cell array

Low-order address bits-select a column in the cell array

A typical processor issues all bits of an address at the same time

50. List the advantages of Multi-bus organization.

Compared to single-bus architecture, the using of multiple-bus architecture have a great advantage in speed and of course, will affect performance also. Instead of using single-bus architecture,

It is more convenient to use multiple-bus architecture. Using multiple-bus architecture will make each device to connect to own bus, which means that each device will have its own bus.

51. What are the inputs for Hardwired control?

Step	Action	Comments
1	PCout,MARin,read,select 4,add,Zin	Load pc in MAR Issue read request to memory. select 4 to Y.do the add operation result stored in Z [PC<-PC+4->word size)]
2	Zout,Pcin,Vin,WMFC	Load pc with next address(INR).wait until memory responds
3	MDRout,IRin	Load 1nstr from MDR to IR
4	R3out,MARin,read	Read the first operand pointed to by R2(from memory)
5	R1out,Yin,WMFC	Enable Riout to transfer the second operand to yin
6	MDRout,select Y, Add, in	Add the operand Y&R1and store the result in z.
7	Zout,R1in,End	Transfer the result back to resulted R

52. Under what situations the micro program counter is not incremented after new instruction is fetched from micro program memory?

There is a situation arises when the control unit is required to check the status of the condition codes or external inputs to choose between alternative course of action.

53. What are the relative merits of horizontal and vertical micro instruction format?

Table shows the comparison between horizontal and vertical organization.

S.No	Horizontal	Vertical
1.	Long formats	Short formats
2.	Ability to express a high degree of parallelism	Limited ability to express parallel micro operations
3.	Little encoding of the control information	Considerable encoding of the control information
4.	Useful when higher operating speed is desired	Slower operating speeds

54. Define Pipelining.

Pipelining increases instruction throughput by performing multiple operations at the same time (in parallel), but does not reduce instruction latency (the time to complete a single instruction from start to finish) as it still must go through all steps.

55. Explain the role of cache memory in Pipelining?

- Each pipeline stage is expected to complete in one clock cycle.
- The clock period should be long enough to let the slowest pipeline stage to complete.
- Faster stages can only wait for the slowest one to complete.
- Since main memory is very slow compared to the execution, if each instruction needs to be fetched from main memory, pipeline is almost useless.
- Fortunately, we have cache.

56. What is hazard?

Any condition that causes the pipeline to stall is called a hazard.

57. What is data hazard?

A data hazard is any condition in which either the source or destination operands of an instruction are not available at the time expected in the pipeline.

58. What is instruction hazard?

The pipeline may be stalled because of a delay in the availability of an instruction which results in cache miss. These hazards are called control hazards or instruction hazards.

59. What is structural hazard?

Structural hazard arises in a situation when two instructions require use of a given hardware resource at the same time.

60. What is branch penalty?

The time lost as result of a branch instruction is often referred to as branch penalty. The branch penalty is one clock cycle. For a longer pipeline, the branch penalty may be higher.

61. What is meant by dispatch unit?

Dispatch unit is a separate unit, which takes instructions from the front of the queue and sends them to the execution unit. The dispatch unit also performs the decoding function.

62. Define the terms branch delay slot and delayed branching.

The location following the branch instruction is called a branch delay slot. There maybe more than one branch delay slot, depending on the time it takes to execute a branch instruction.

A technique called delayed branching can minimize the penalty incurred as a result of conditional branch instructions. The instructions in delay slots are always fetched. If there are no useful instructions, these are filled with NOP instructions. That is branching takes place one instruction later than where the branch instruction appears in the instruction sequence in the memory, hence “delayed branch”.

63. What is speculative execution?

Speculative execution means that instructions are executed before the processor is certain that they are in the correct execution sequence. It must be noted that no processor registers or memory locations are updated until it is confirmed that these instructions should indeed be executed.

64. Define static and dynamic branch prediction.

A decision on which way to predict the result of the branch may be made in hardware by observing whether the target address of the branch is lower than or higher than the address of the branch instruction.

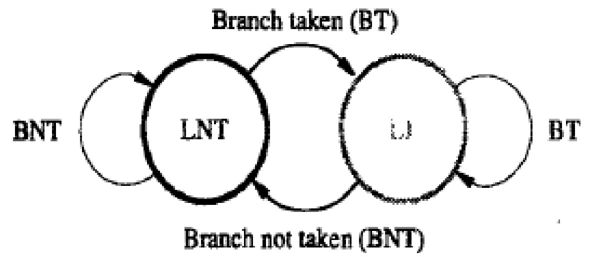
The branch prediction decision is always the same every time a given instruction is executed is called static branch prediction. The approach in which the branch prediction decision may change depending on execution history is called dynamic branch prediction.

65. What is a 2-state algorithm?

The branch prediction algorithms are to reduce the probability of making a wrong decision, to avoid fetching instructions that eventually have to be discarded. The algorithm may be described by the two-state machine. The two states are:

LT: Branch is likely to be taken

LNT: Branch is likely not to be taken



66. Represent a 4-state machine algorithm.

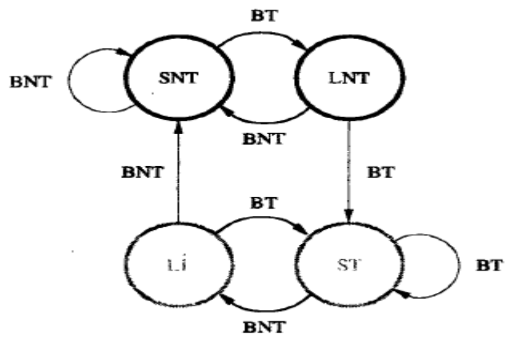
The four states are:

ST: Strongly likely to be taken

LT: Likely to be taken

LNT: Likely not to be taken

SNT Strongly likely not to be taken



67. What is misprediction?

The state information used in dynamic branch prediction may be recorded in a look-up table. It is possible for two branch instructions to share the same table entry. This leads to branch being mispredicted, but it does not cause an error in execution. Misprediction only introduces a small delay in execution time.

68. State the advantage and disadvantage of complex addressing modes.

Complex addressing modes involve several accesses to memory that do not necessarily lead to faster execution.

The main advantage is that they reduce the number of instructions needed to perform a given task and thereby reduce the program space needed in the main memory.

The disadvantage is that their long execution times cause the pipeline to stall, thus reducing its effectiveness. They require more complex hardware to decode and execute them.

69. State the features of addressing modes used in modern processors.

- Access to an operand does not require more than one access to the memory.

- Only load and store instructions access memory operands.
- The addressing modes used do not have side effects.

The addressing modes that have these features are register, register indirect and index.

70. List the way condition codes are to be handled.

1. To provide flexibility in reordering instructions, the condition-code flags should be affected by as few instructions as possible.

2. The compiler should be able to specify in which instructions of a program the condition codes are affected and in which they are not.

71. What are superscalar processors?

A processor can be equipped with multiple processing units to handle several instructions in parallel in each processing stage. Hence, several instructions can start execution in the same clock cycle, and the processor is said to use multiple-issue. Such processors are capable of achieving an instruction execution throughput of more than one instruction per cycle. They are known as superscalar processors.

72. What is deadlock?

A deadlock is a situation that can arise when two units, A and B, use a shared resource. Suppose that unit B cannot complete its task until unit A completes its task. At the same time, unit B has been assigned a resource that unit A needs. If this happens, neither unit can complete its task.

73. How to prevent a deadlock that arises during instruction execution?

If instructions are dispatched out of order, a deadlock can arise. To prevent deadlocks, the dispatcher must take many factors into account. Issuing instructions out of order increases the complexity of the dispatch unit. Hence, most processor use only in-order dispatching.

74. How to indicate the performance? What tool is used?

A useful performance indicator is the Instruction throughput, which is the number of instructions executed per second. For sequential execution, the throughput P_s is given by,

$$P_s = R/S$$

Any time a pipeline is stalled, the instruction throughput is reduced. The performance is highly influenced by factors such as branch and cache miss penalties.

11 MARKS

1. Explain about Single bus organization of the Data path inside a processor:

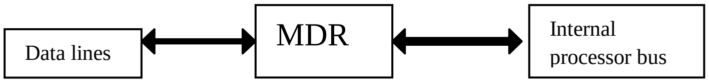
To execute an instruction, the processor has to perform the following steps:

- Fetch the contents of the memory location pointed to by the pc. The contents of this location are interpreted as an instruction to be executed.
- Hence, they are loaded in to the IR. Symbolically, this can be written as
  - IR <- [[PC]]
  - Assuming that the memory is byte addressable increment the contents of the pc by 4 that is
  - PC <-[PC] +4

Carry out the actions specified by the instruction in the IR. The first two steps are termed as fetch step and the 3rd step is known as execution phase

The fig shows an organization of the data path inside a processor with a single bus. The bus is internal to the processor that connects the processor to the memory and IO devices

- The data and address lines of the external memory are connected to the internal processor bus via the memory data register (MDR) and the memory address register (MAR). The registers MDR has two inputs and two outputs.



Data may be loaded into MDR either from the memory bus or from the internal process bus. The data stored in MDR may be placed on either bus.

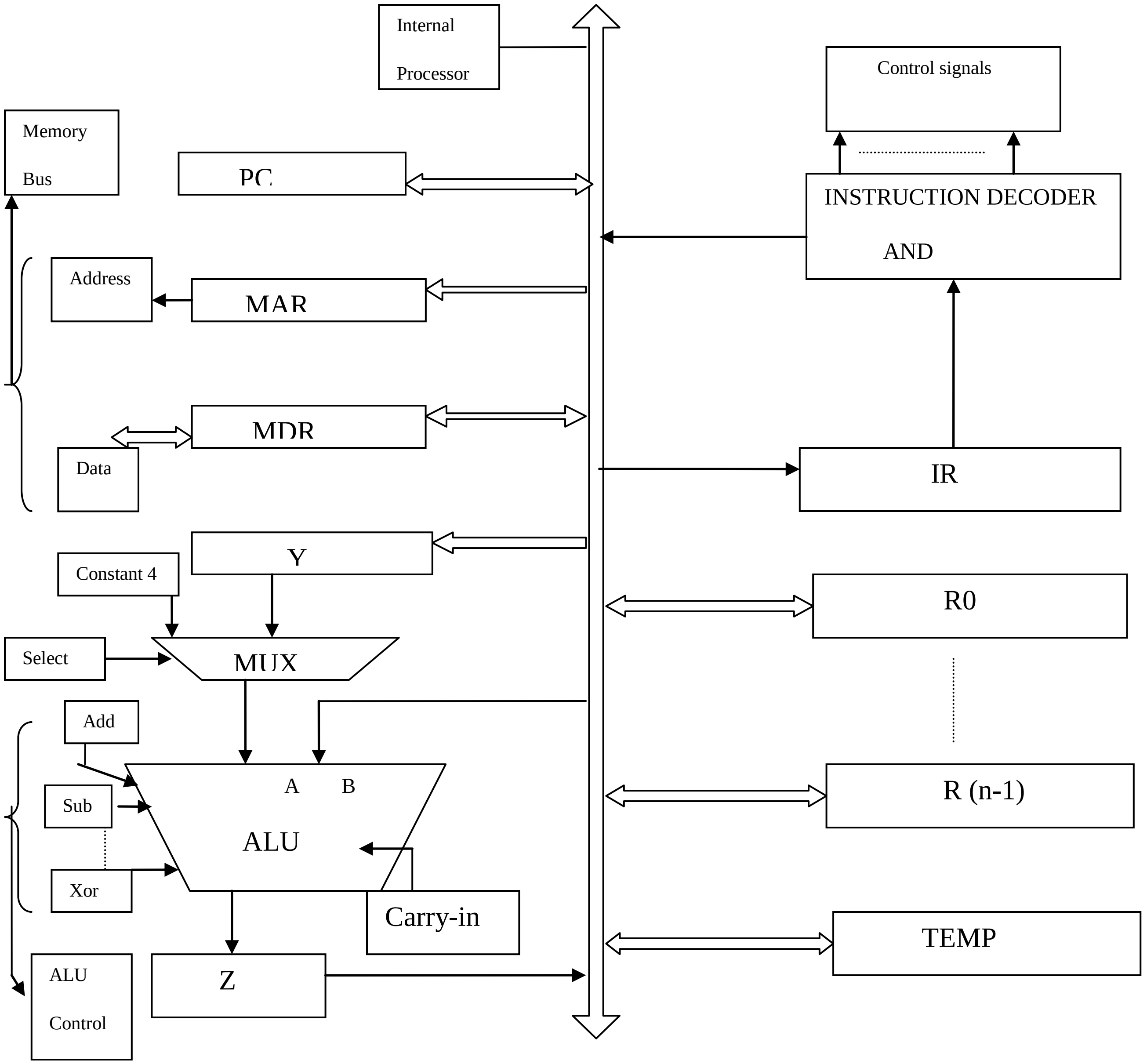
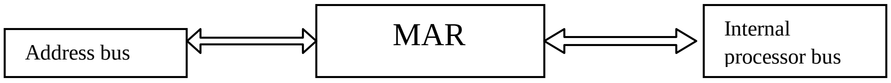


Fig: Single-Bus Organization Of The Data Path Inside A Processor

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



- The control lines of the memory bus are connected to the instruction decoder and controls logic block. This unit is responsible for issuing the signals that control the operation of all units inside the processor and for interacting with the memory bus.
- The registers R0 through R (n-1) in the general purpose register the number of register and the use of processor may vary from one processor to another.

- Some register are dedicated as special purpose register such as index register, stack pointers, accumulator etc. for example in the diagram y, z, temp are used by the programmer for any instruction.
- The multiplexer MUX selects either the output of register Y or a constant value to be provided as input A of the AW.The constant 4 is used to increment the contents of the program counter.
- As instruction execution progresses, data are transferred from one register to other of the ten passing through the ALU to perform some arithmetic or logical operation.
- The instruction decoder and control logic unit is responsible for implementing the actions specified by the instruction loaded in the IR register.
- The registers, ALU and inter connecting bus are collectively referred to as the data path.
- The sequence of execution of instruction is as follows:
    - ✓ Transfer a word of data from one processor register to another or to the ALU.
    - ✓ Perform arithmetic or a logic operation and store the result in a processor register
    - ✓ Fetch the contents of a given memory location and load them into a processor register
    - ✓ Store a word of data from a processor register into a given memory location.

Some of the operations performed while executing an instruction are:

1. Register transfer
2. Arithmetic or logic operation
3. Fetching a word from memory
4. Storing a word in memory

Register transfer:

- ➢ Instruction execution will be faster if the operands are stored in registers. During execution of an instruction data may be transferred from one register to another.
- ➢ The registers are connected to the bus via switches controlled by the signals Riin and Riout, when Riin is set to 1, the data on the bus are loaded into Ri are placed on the bus.

The data transfer from R1 to R2 is done as follows:

- Enable R1out by setting it to 1. The content of R1 is now available on the processor internal bus.
- Enable R2in by setting H to 1. This loads data from the processor bus in to R2.

- ➢ Depending on the operations to be done, the control signals associated with the registers are asserted at the start of the clock cycle, as the flip-flops that the resistors are edge triggered.

- When edge triggered flip-flops are not used, two or more clock signals may be needed to guarantee proper transfer of data. This is called multiphase clocking.

The instruction MOV R1, R2 is done as follows.

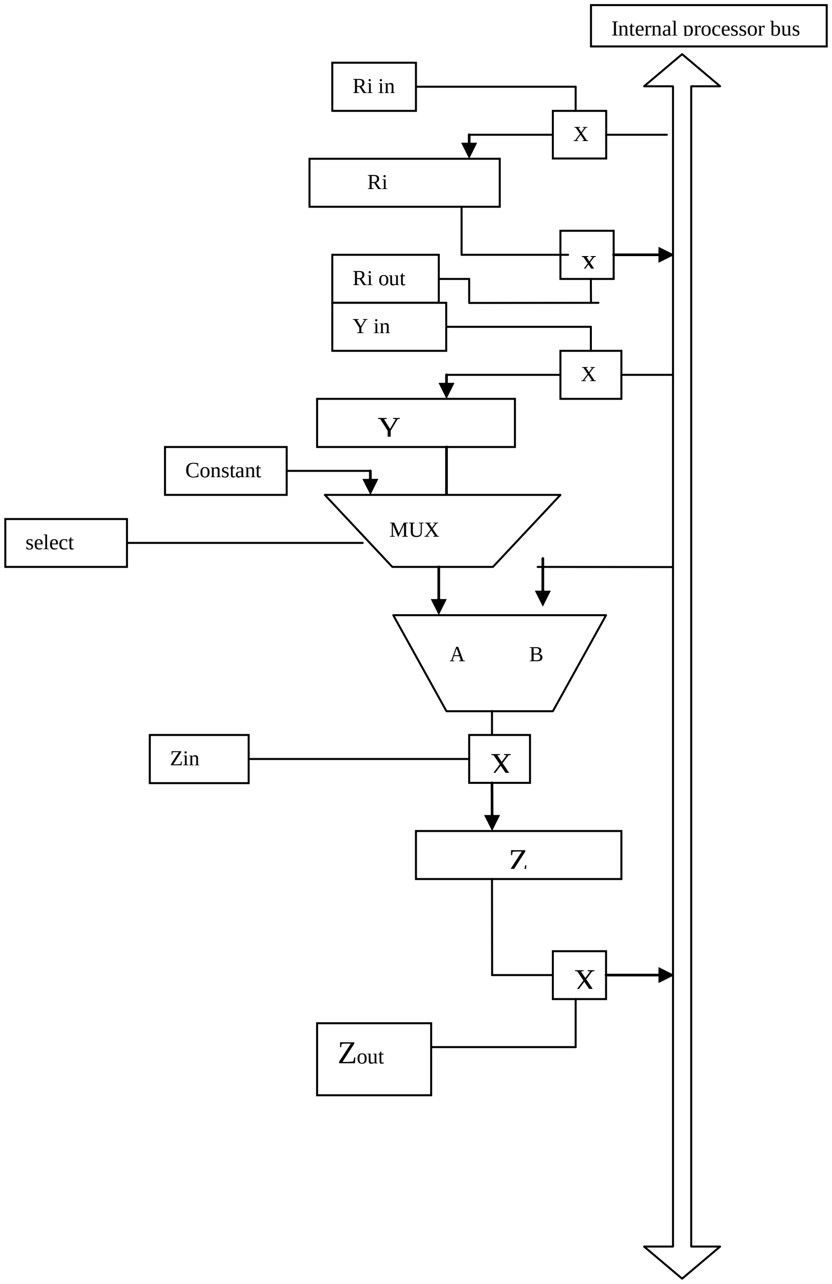


Fig: Register Transfer

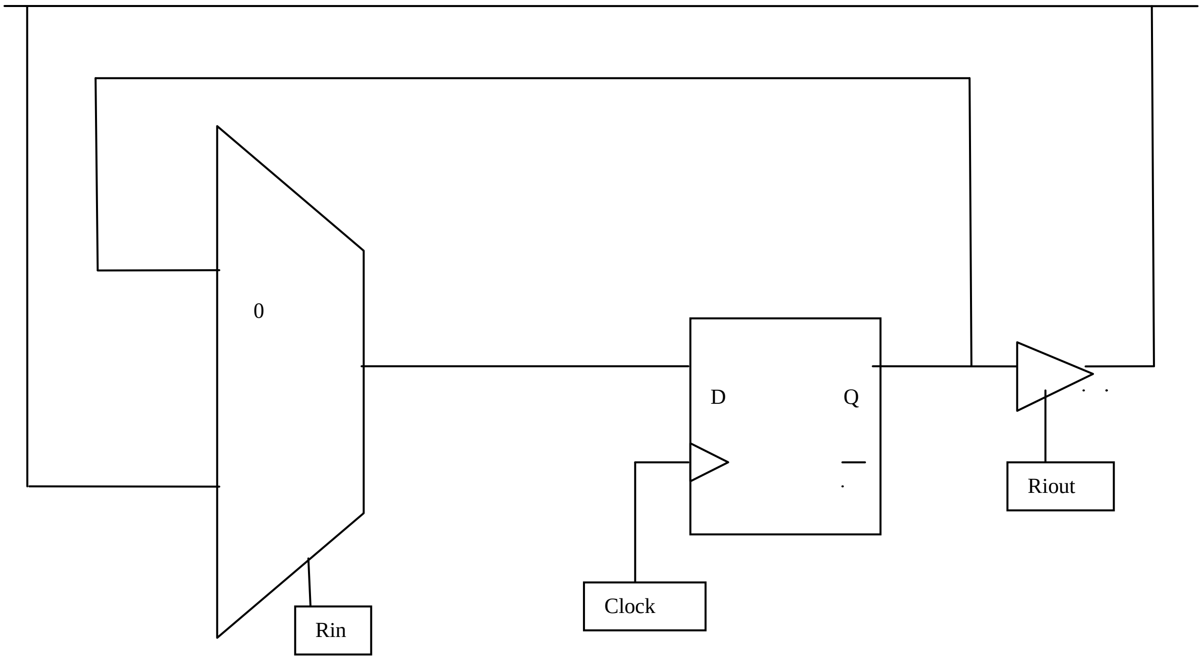


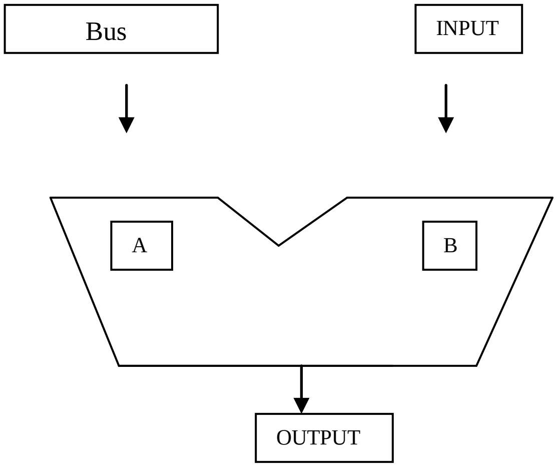
Fig: Input And Output Gating For One Register Bit

Performing an arithmetic or logic operation:

The arithmetic and logic unit does the computations. During computations, the data may be taken from the registers or from the memory via the bus. For e.g.: as shown in fig the ALU takes two inputs, one from the MUX, which selects either the constant 4 or one of the registers and other input from the bus. The result is then stored in a temporary register z which is then transferred to either any one of the general purpose registers R0 to Rn, or to the memory via the bus. Hence the sequence of operation to add R1 and R2 and then store the result in register R3 is

R3=R1+R2 is

1. *R1 out, Y in*
2. *R2 out, select Y, add, Z in [Selects Y is control signal to the MUX to select the register content through Y register]*
3. *Z out, R3 in. [At any point of time.]*



Fetching a word from memory:

X	Y	Z	
0	0	0	ADD
0	0	1	SUB
0	1	0	DIVIDE
0	1	1	MULTIPLY
1	0	0	AND
1	0	1	OR
1	1	0	NOT
1	1	1	XOR

- ✓ To fetch a word from memory, the processor has to specify the address of the memory location where this information is stored and request a read operation.
- ✓ The processor transfers the required address to MAR, where o/p is connected to the address lines of the memory bus.
- ✓ The requested data from the memory is stored in registers MDR, form where they are transferred to other registers in the processor.

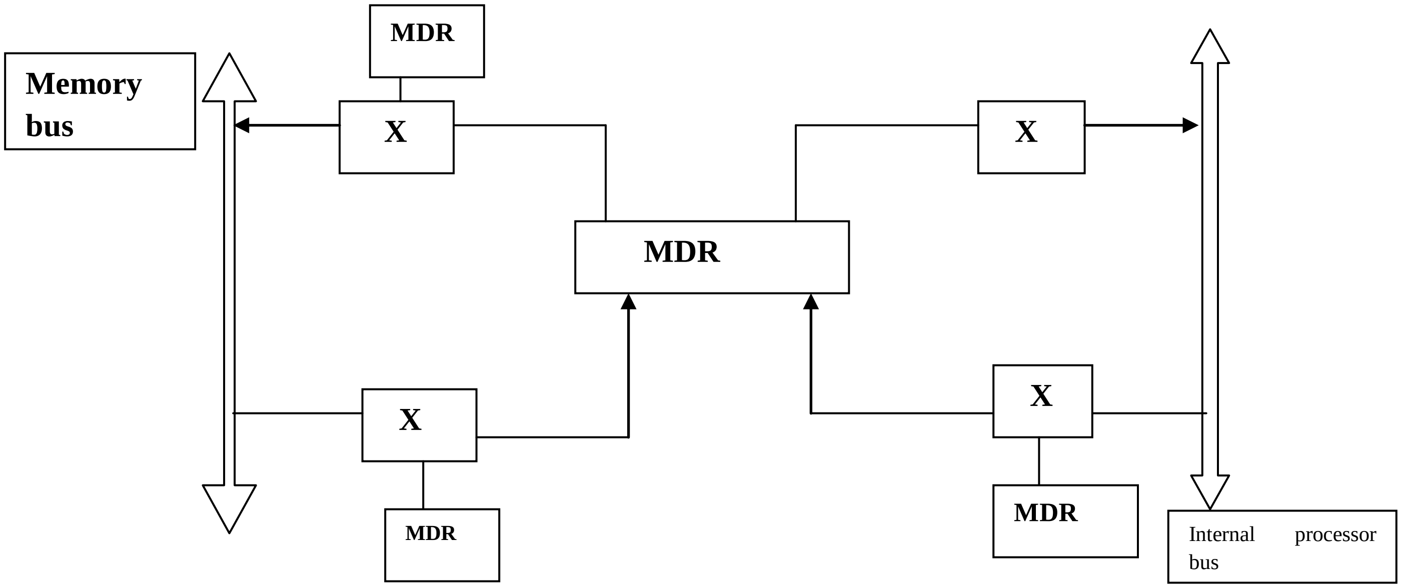


Fig: Connection And Control Signals For Register MDR

✓ It has four control signals:

- MDR in and MDR outE control the connection to the internal bus
- MDR inE and MDR outE control the connection to the external bus
- During memory read and write operation, the timing of internal. Processor operations must be coordinated with the response of the addressed device on the memory bus.
- The processor completes one internal data transfer in one clock cycles.

Example:

To perform read operation, consider the instruction move R1, R2. This instruction execution is given as follows.

1. MAR<-[R1]
2. Start a read operation on the memory bus.
3. Wait for the control signal called memory function completed (MFC)
4. Load MDR from the memory bus.

5. R2<-MDR.

The control signals is being activated as follows for the above instruction

1. R1 Out, MAR, read.
2. MDRin E, WMFC (wait for memory function complete)
3. MDRout, R2 in

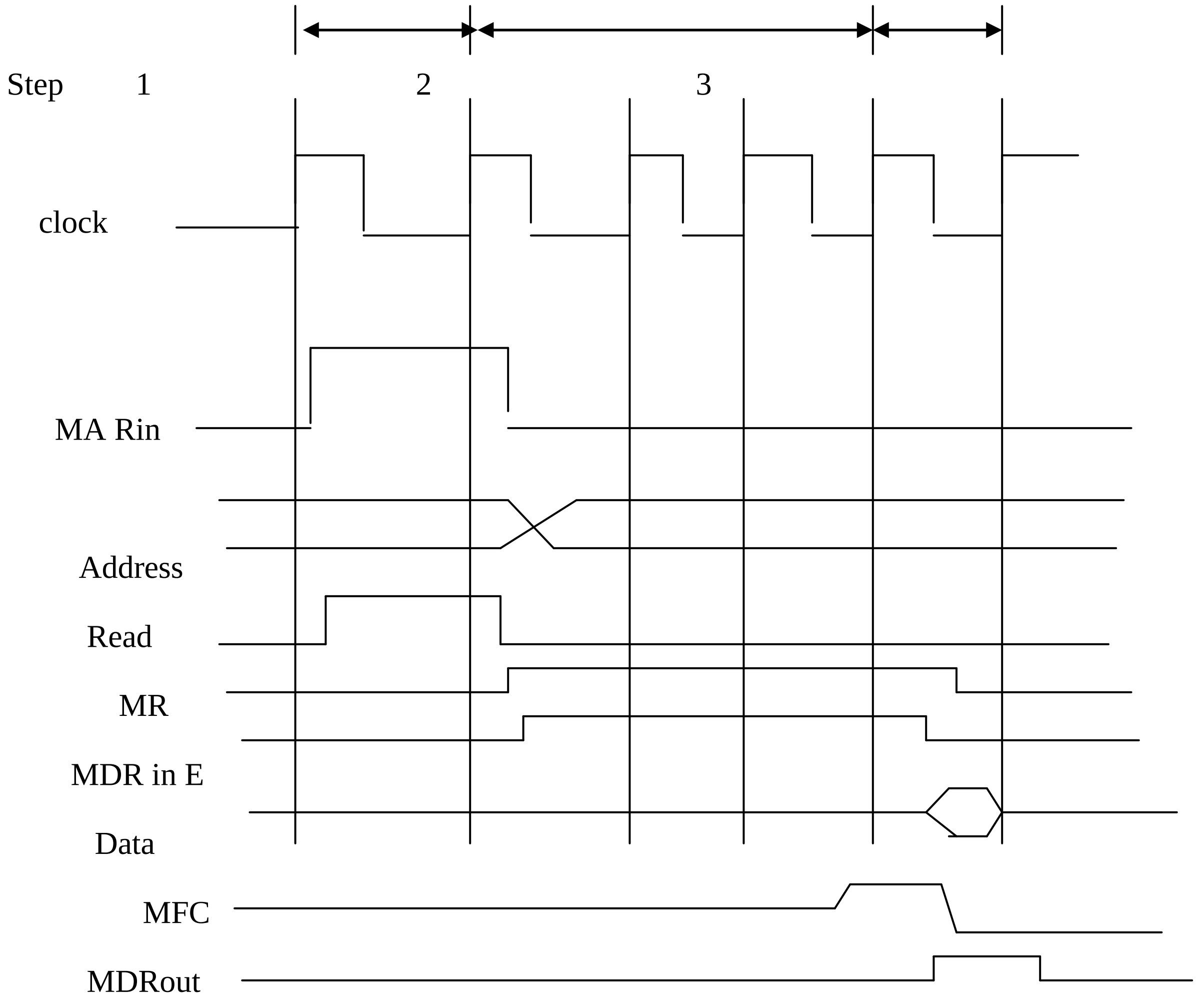


Fig: Timing Of A Memory Read Operation

Storing a word in memory:

Writing word into a memory location, the derived address is loaded into MAR. Then the data can be written are loaded into MDR and a write command is issued.

Hence executing the instruction MOV R2, (R1) requires the following sequence

1. R1out, MAR in
2. R2out MDR in, write
3. MDR out E, NMFC.

The processor remains in step3 until the memory operation is completed and as MFC response is received.

2. List and explain the steps involved in the execution of a complete execution.

Consider the instruction which adds the content of memory location pointed to by R3 to Register R1.

Actions:

The following are the actions while executing an instruction

i) Fetch the instruction

ii) Fetch the first operand

iii) Perform the addition

iv) Load the result into R1

The control sequence for execution of the above instruction:

Step	action	Comments
1	PCout,MARin,read,select 4,add,Zin	Load pc in MAR Issue read request to memory. select 4 to Y.do the add operation result stored in Z [PC<-PC+4->word size)]
2	Zout,Pcin,Vin,WMFC	Load pc with next address(INR).wait until memory responds
3	MDRout,IRin	Load 1nstr from MDR to IR
4	R3out,MARin,read	Read the first operand pointed to by R2(from memory)
5	R1out,Yin,WMFC	Enable Riout to transfer the second operand to yin
6	MDRout,select Y, Add, in	Add the operand Y&R1and store the result in z.
7	Zout,R1in,End	Transfer the result back

		to resulted R

STEP	ACTION	COMMENTS
1	PC out, MAR in,read,select 4,add, Zin	Increment PC
2	Z out,PC in, Y in, WMFC	Update PC and wait for memory
3	MDR out, IR in	Instruction fetch in IR
4	Offset-in-IR out add, Z in	Target address calculation pc+offset given in branch instruction
5	Zout, PC in, end	Load pc with new address(target address)

The steps 1 to 3 constitute the instruction fetch phase:

- The contents of PC is loaded intoMAR and read request is sent.
- Select signal is used to select the constant 4
- The value is added to the operand and the result is stored in register Z.
- The updated value is moved from register Z back into the PC in step 2.
- The word fetched from the memory is loaded into the IR.
- The steps 4 to 7 constitute the instruction decoding phase:
- The contents of R3 are transferred to the MAR in step4 and memory read operation is initiated.
- When read operation is completed, the memory operand is available in register MDR and the addition operation is performed in step 6.
- The sum is stored in register Z and transferred to R1 in step 7.
- The End signal causes a new instruction fetch cycle to begin by returning to step 1.

Branch instruction:

A branch instruction replaces the contents of the PC with the branch target address. This address is usually obtained by adding an offset X, which is given in branch instruction.

There are two types of branch

1. Unconditional branch
2. Conditional branch

Control sequence for an unconditional branch instruction:

- The steps 1 to 3 constitute the fetch phase and it ends when the instructions is loaded into the IR in step3.
- Since the value of updated PC is already available in register Y, the offset X is gated onto the bus in step 4.
- The result which is the branch target address is loaded into the PC in step5.
- For the conditional branch:
- The ststus of the condition codes must be checked before loading a new value into thw PC.
- The step 4 in the above control sequence must be replaced with Offset-field.
- Thus if N=0 the processor returns to step 1 immediately after step 4.
- If N=1 step 5 is performed to execute branch instruction.

3. Explain multiple bus organization in detail. OR Explain the execution of a three operand instruction using multiple bus organization

Multiple bus organization

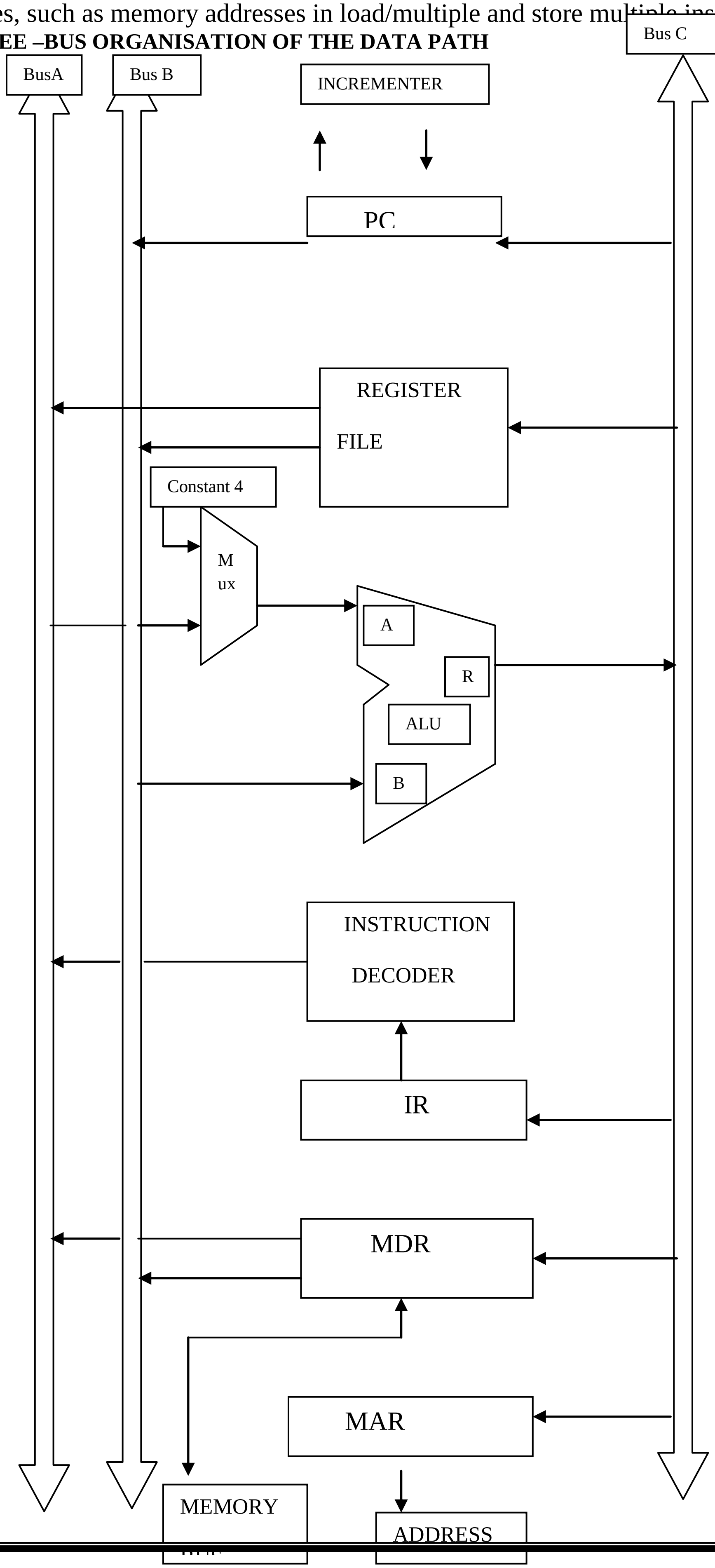
- It shows the three bus structure used to connect the registers and the ALU of a processor. Using 3-bus structure we can reduce the number of steps needed.
- All the general purpose registers are combined into a single block called the register file. The register file is said to have 3 ports.
- There are 2 ports allowing the contents of 2 different registers to be accessed simultaneously and have their contents placed on the buses A and B .The third port allows the data on bus C to the loaded into a third register during the same clock cycle.

FEATURES:

- Buses A and B are used to transfer the source operand to the A and B input of the ALU ,Where an arithmetic or logic operation may be performed. The result is transferred to the destination over bus C. If needed, the ALU may simply pass one of its 2 input operands unmodified to bus C. We will call the ALU control signals for such an operation R = A or R = B.
- A second feature is the introduction of the incriminator unit, Which is used to increment the PC by A. Using the incriminator eliminates the need to add 4 to the PC using the main ALU as was done in the figure.

The source of the constant 4 at the ALU input multiplier is still useful. It can be used to increment other addresses, such as memory addresses in load/multiple and store multiple instruction.

Fig:THREE –BUS ORGANISATION OF THE DATA PATH



Consider the 3 operand instruction

ADD R4,R5,R6.

The control sequence for executing this instruction is

STEP	ACTION
1	PC_{out}, R = B, MAR_{in}, Read, IncPC
2	WMF_{c}
3	MDR_{out} B,R = B,IR_{in}
4	$R4_{out}$,R5 $_{out}$ B,Select A,ADD,$R6_{in}$,END

STEP 1: The contents of PC are passes through the ALU , using R = B control signal, and loaded into the MAR to start a memory read operation. At the same time the PC is incremented by 4.The value is loaded into MAR is the original contents of the PC. The incremented value is loaded into the PC at the end of the clock cycle and will not effect the contents of the MAR.

STEP 2:The processor waits for MFC and loads the data received into MDR, then transfers them into IR.

STEP 3: Finally the execution phase of the instructs requires only one control step to complete. By introducing more paths for data transfer a significant reduction in the number of clock cycles needed to execute an instruction is achieved.

4. Explain the various design methods of hardwired control unit.

Or Describe the control unit organization with a separate Encoder and Decoder functions in a hardwired control

Hardwired control:

To execute instruction, the processor must have some means of generating the control signals needed in proper sequence. The 2 categories are

(i) Hardwired control

(ii) Micro programmed control

Hardwired control:

- A Hardwired control unit consists of combinational circuits to generate various control signals. It shows an overall block diagram of the hardwired control unit

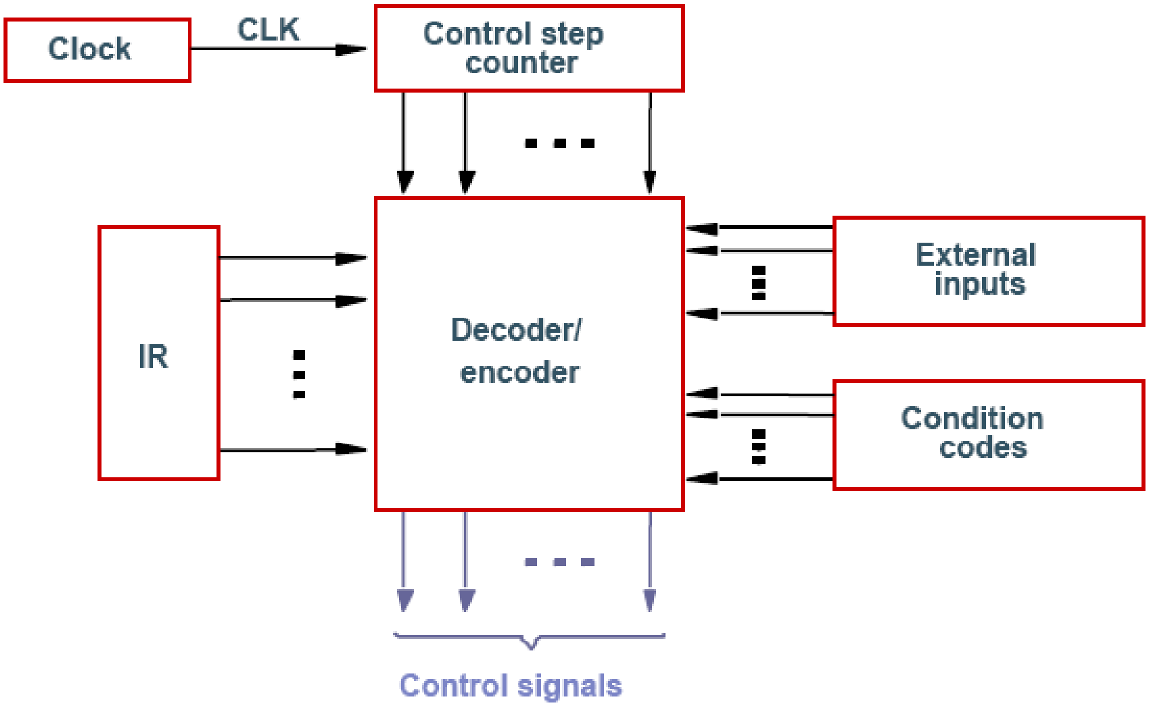


Fig. Control Unit Organization

- The opcode is the main input to the control unit. It is an input to the opcode decoder.
- The decoder/encoder block is a combinational circuit that generates the required control output, depending on the state of its inputs. Consider the sequence of control signals for the instruction ADD(R3), R1

1. PC_{out},MAR_{in},Read,Select 4,Add,Z_{in}

2. Z_{out},PC_{in},Y_{in},WMF_c

3. MDR_{out},IR_{in}

4. $R3_{out}$,MAR_{in},Read

5. $R1_{out}$,Y_{in},WMF_c

6. MDR_{out},Select Y,Add,Z_{in}

7. Zo_{ut},$R1_{in}$,End

Each step in the sequence is completed in one clock period. A counter may be used to keep track of the control steps, are shown in the figure. Each state or count of this counter corresponds to one control step. The required control signals are determined by the following instruction.

1. Contents of the control step counter

2. Contents of the instruction register

3. Contents of the condition code flags

4. External input signals such as MFC and interrupt requests.

The figure shows the separation of the decoding and encoding functions.

- The step decoder function provides a separate signal line for each step, or time slot, in the control sequence.
- Similarly, the output of the instruction decoder consists of a separate line for each machine instruction. For any instruction loaded in the IR, one of the outlines INS, through INS_{in}, is set to 1, and all other lines are set to 0..
- The input signals to the encoder block in figure are combined to generate the individual control signals Y_{in}, PC_{out}, Add, End and so on.

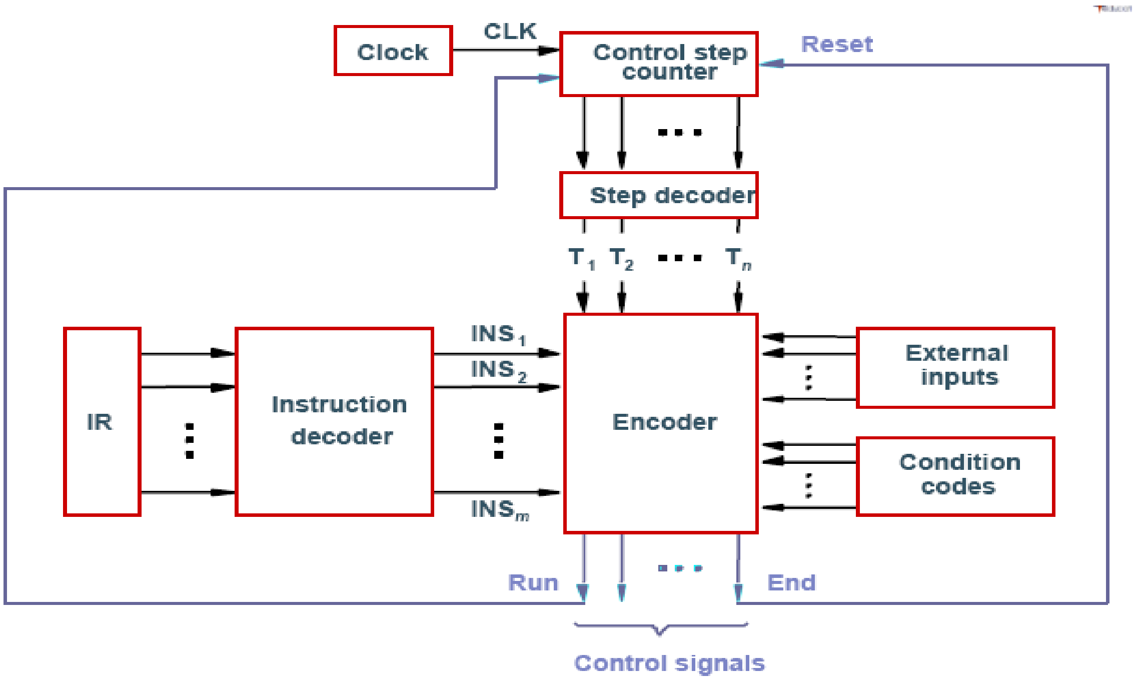
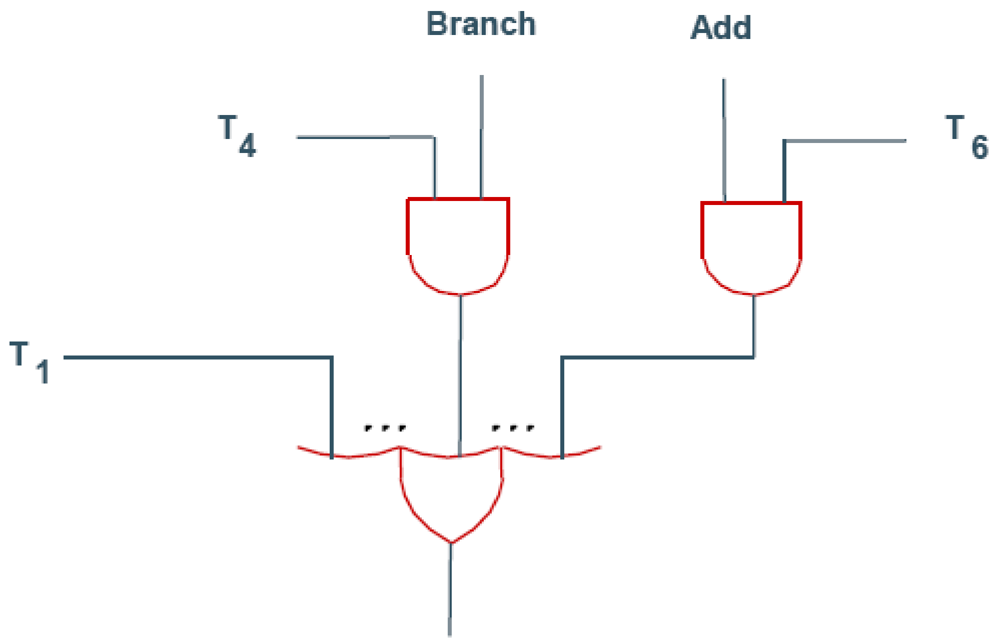


Fig. Separation of decoding and encoding functions

Let us see how the encoder generates signal for single bus processor organization shown in figure



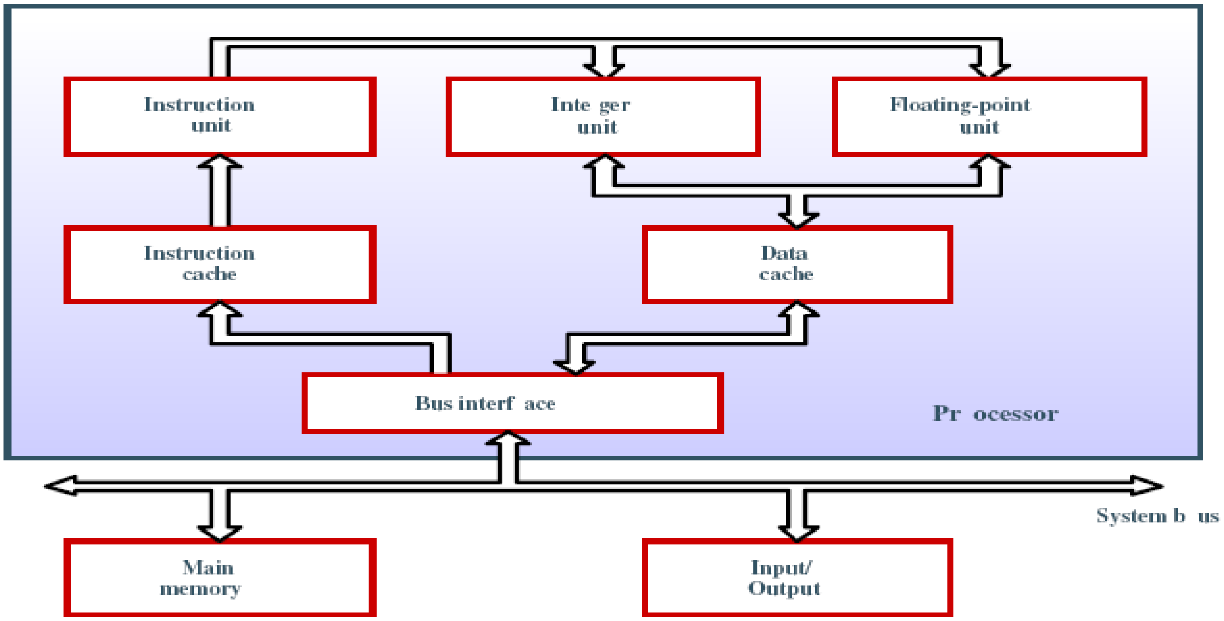
- Z_{in},The encoder circuit implements the following logic functions to generate Z_{in}.The logic function for Z_{in} is derived from the control sequence in figure.

The end signal starts a new instruction fetch cycle by resetting the control step counter to its starting value.

- Another control signal is called RUN is set to 1,RUN causes the counter to be implemented by one at the end of every clock cycle. When RUN is equal to 0, the counter stops counting. This is needed whenever the WMF_c signal is issued, to cause the processor to wait for the reply from the memory.
- The sequence of operation carried out by machine is determined by the wiring of logic elements, hence the name hard-wired.
- The controller that uses this approach can operate at high speed. However, it has little flexibility, and the complexity of the instruction set it can implement is limited.

5. Draw and explain the block diagram of a complete processor

A Complete Processor



- The processor consists of instruction unit, integer unit, floating point unit, instruction cache, data cache, bus interface unit, main memory module and input/output module.

- The instruction unit fetches instruction from the instruction cache or from the main memory. The complete processor provides 2 processing units floating point and integer unit.

- These 2 units gets data from data cache.The processor provides bus interface unit to control the interface of processor to system bus, main memory module and input/output modules.

Advantages of hardwired control unit:

- A hardwired control unit works fast.

- The combinational circuits generates the control signals based on the input signal status. As soon as the required input signal combination takes place, immediately the output is generated by a combinational circuit.

Disadvantages of hardwired control unit:

- If the CPU has the large number of control points, the control unit design becomes very complex. It is tedious to design the pulse distributor circuitry since several gates input and output have to be kept tracks of during designing.

- The design does not give any flexibility. If any modification is required, it is extremely difficult to make the correction. Design modification becomes necessary under the following situations:

1. There is a design mistake in the original gates.

2. A new feature is to be added to an existing design.

3. A new hardware component of higher speed is available, which will improve the performance.

6. Explain micro programmed control unit. What are the advantages and disadvantage of it? OR With a neat diagram explain the internal organization of a processor. (Apr 11)

Micro programmed Control:

Microprogramming is a modern concept used to designing a control unit. It can be used for designing control logic for any digital system. Some common applications of input/output controllers such as disk controller, peripheral devices such as printer and hard disk drive.

- The microinstructions are executed when the corresponding control signals are made active. Each instruction needs a specific set of micro operation in an order .
- Micro programmed control, in which control signals are generated by a program similar to machine language programs. First, we introduce some common terms.

Control memory: The control signals needed for an instruction and the time sequence can be stored in the memory.

Control Word: CW is a word whose individual bits represent various control signals. Each of the control steps in the control sequence of an instruction defines a unique

combination of 1s and 0s in the CW. The CW corresponding to the 7 steps of fig.

Micro instruction	PC in	PC out	MAR in	read	MDR out	IR in	Y In	Select	add	Z in	Z out	R1 out	R1 in	R3 out	WM FC	end
1	0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0
2	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
4	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0
5	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0
6	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1

ig: Example –Micro Instruction

- If the bit is 1, the corresponding control signal is activated. If the bit is 0, the control signal is not activated.
- When a control memory word is fetched from control memory some bits may be 1 and others 0.The control signals corresponding to '1' bits are generated. Thus, multiple micro operations are executed simultaneously.

Control Store:

A special memory in which the microinstructions for all instructions in the instruction set of a computer are stored in a control memory. It is also called as a control memory

Micro routine: A sequence of control words corresponding to the control sequence of a machine instruction constitutes the micro routine.

Micro instruction: Each control memory word is known as a microinstruction.

Basic organization of micro programmers control unit is shown fig.

- To read the control words sequentially from the control memory, a microprogram counter(MPC) is used. Every time a new instruction is loaded in the IR, the output of the block labelled ;starting address generator' is loaded into the MPC. 24
- The MPC is automatically incremented by the clock, causing successive microinstructions to be read from the control memory. Hence the control signals are delivered to various parts to the processor in the control sequence.

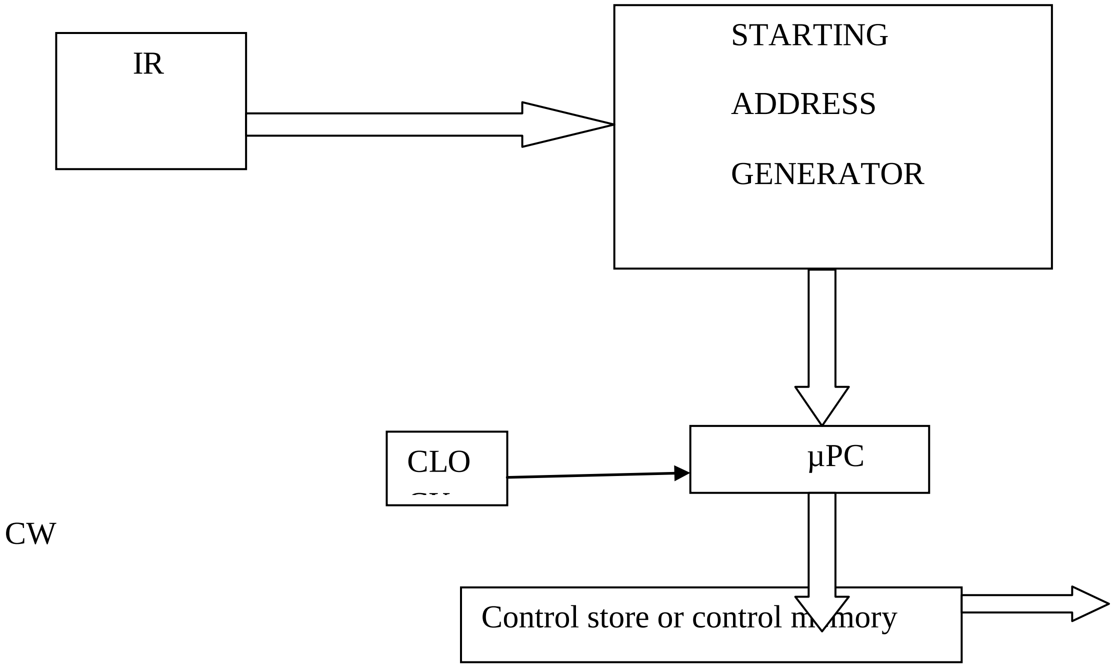


Fig: Basic Organization Of A Micro programmed Control Unit

- There is a situation arises when the control unit is required to check the status of the condition codes or external inputs to choose between alternative course of action. In micro programmed control the alternative approach is to use conditional branch microinstructions.

- In addition to the branch address, these microinstruction specify which of the external inputs, condition codes, or possibly bits of the instruction register should be checked as a condition for branching to take place.

- The instruction Branch < 0 may be implemented by a micro routine . After loading this instruction into IR, a branch microinstruction transfers control to the corresponding micro routine, which is assumed to start at location 25 in the control memory. This address is the starting address generator block in fig.

- The microinstruction at location 25 tests the N bit of the condition codes. If the bit is equal to 0, a branch takes place to location 0 to fetch a new machine instruction.

- Otherwise, the microinstruction at location 26 is executed to put the branch target address into register Z1. The microinstruction in location 27 loads this address into the PC.

- To support micro program branching, the organization of the control, unit should be modified as shown in fig.

The starting address generator block of a fig becomes the starting and branch address generator. This blocks a new address into the MPC when a microinstruction instructs it to do so.

To allow implementation of a conditional branch, inputs to this block consists of the external inputs and condition codes as well as the contents of the instruction register. In this control unit , the MPC is incremented every time a new instruction is fetched from the micro program memory, except in the following situations:

1) When a new instruction is loaded into the IR, the MPC is loaded with the starting address of the micro routine for that instruction.

2) When a branch microinstruction is encounted and the branch condition is satisfied, the MPC is loaded with the branch address

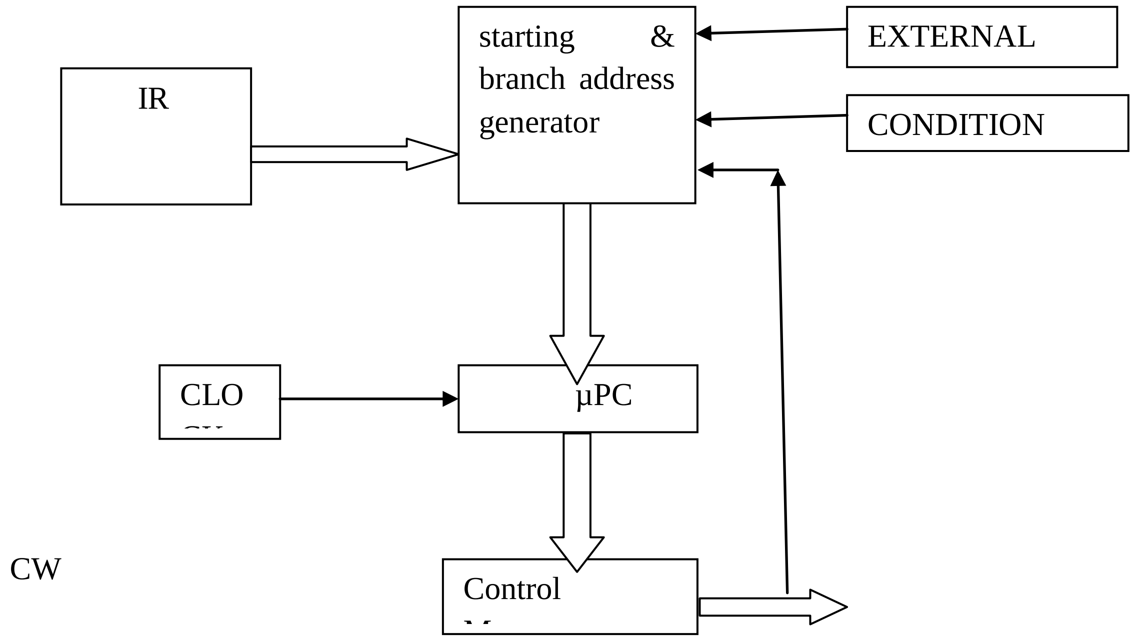


Fig:A Organization Of Control Unit To Allow Conditional Branching In The Micro program

ADDRESS	MICROINSTRUCTION
0	PC_{out}, MAR_{in}, Read, Select 4, Add, Z_{in}
1	Z_{out}, PC_{in}, Y_{in}, WMF_C
2	MDR_{out},IR_{in}
3	Branch to starting address of appropriate instruction
……	
25	If N = 0, then branch to microinstruction 0
26	Offset-field of IR_{out}, Select Y,Add,Z_{in}
27	Z_{out}, PC_{in},End

Fig: Micro routine For The Instruction Branch < 0

MICROINSTRUCTIONS

- A simple way to structure microinstruction 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 required signals is usually large.

- Moreover only a few bits are used in any given instruction. The solution of the problem is solved by grouping the control signals.

Grouping of control signals:

- Control signals can be grouped so that all mutually exclusive signals are placed in the same group. Thus, at most one micro operations per group is specified in any microinstruction. Then it is possible to use a binary coding scheme to represent the signals with a group.

Example:

4 bits suffice to represent 16 available functions in the ALU. Register output control signals can be placed in a group consisting of PC_{out}, $R0_{out}$, $R1_{out}$, $R2_{out}$, $R3_{out}$ and $TEMP_{out}$, Z_{out}, MDR_{out}.

Any one of these can be selected by a unique 4-bit code. Further natural groups can be made for the remaining signals.

Gating signals: IN and OUT signals
Control signals: Read, Write, Clear A, Set carry in WMFC, END etc.

ALU Signals:

- Add, Sub, etc. There are in all 39 signals and hence each microinstruction will have 39 bits. It is not all necessary to use all 39 bits for every microinstruction because by grouping of control signals we minimize number of bits for microinstructions.

Ways to reduce number of bits in microinstruction:

1)Most signals are not needed simultaneously

2) Many signals are mutually exclusive(e.g) only one function of ALU can be activated at a time.

3) A source for data transfers must be unique which means that it should not be possible to get the contents of 2 different registers on to the bus at the same time.

4) Read and Write signals to the memory cannot be activated simultaneously.

The below figure shows an example of a partial format for the microinstructions, in which each group occupies a field large enough to contain the required codes. Most fields much include one inactive code for the case in which no action is required.

Fig:An example of a partial format for field encoded microinstruction

F1	F2	F3		

F1(4 bits)	F2(3 bits)	F3(3 bits)	F4(4 bits)	F5(2 bits)
0000: No transfer	0000:No transfer	0000:add transfer	00:Noaction	00:Noaction
0001:PC_{out}	001:PC_{in}	001:MAR_{in}	0000:ADD	01: Read

0010:MDR_{out}	010:IR_{in}	010 MDR_{in}	0001:SUB	10:Write
0011:Z_{out}	011:Z_{in}	011:TEMPin	.	16 ALU Functions
0100:$R0_{out}$	100:$R0_{in}$			100:Yin .
0101: $R1_{out}$	101: $R1_{in}$		1111 :XOR	
0110: $R2_{out}$	110: $R2_{in}$			
0111: $R3_{out}$	111: $R3_{in}$			
1010:$TEMP_{out}$				
1011:$Offset_{out}$				

F6	F7

F6(1bit)	F7(1bit)	F8(1bit)
0:select Y	0:No action	0:Continue
1:select 4	1:WMF_C	1:End

- ➢ For example, the all zero pattern in F1 indicates that none of the registers that may be specified in this field should have its contents placed on the bus. An inactive code is not needed in all fields.
- ➢ For example, F4 contains 4 bits that specify one of the 16 operations performed in the ALU. Since no space code is included, the ALU is active during the execution of every microinstruction.
- ➢ However its activity is monitored by the rest of the machine through registerZ,which is loaded only when the Z_{in} signal is activated.
- ➢ Grouping control signals into fields requires a little more hardware because decoding circuits must be used to decode the bit patterns of each field into individual control signals.

Techniques of grouping of control signals:

The grouping of control signals can be done either by using technique called vertical organization or by using technique called horizontal organization.

- Highly encoded scheme that use impact codes to specify only a small number of control function in each microinstruction are referred to as a vertical organization.
- On the other hand, minimally encoded scheme, in which resources can be controlled with a single instruction, is called a horizontal organization.

Microinstruction sequencing:

Micro program sequencing is done by micro program sequencer. There are 2 important factors that must be considered while designing the micro program sequencer:

- The size of the microinstruction and
- The address generation time.

The size of the microinstruction should be minimum so that the size of control memory required to store microinstructions is also less. This reduces the cost of control memory with less address generation time, microinstructions can be executed in less terms,, resulting better throughput.

During execution of a micro program the address of the next microinstruction to be executed has 3 resources:

- Determined by instruction register
- Next sequential address
- Branch

- ➢ One of these 3 address sources first occurs once per instruction cycle. The second source is most commonly used. However if we store separate microinstructions for each machine instruction, there will be large number of microinstructions.
- ➢ As a result, large space in control memory is required to store these microinstructions. We want to organise the micro program such that they share as many microinstruction S POSSIBLE.
- ➢ This requires many branch microinstructions, both unconditional and conditional to transfer control among the various microinstructions. Thus it is important to design compact, time efficient techniques for microinstruction branching.

Consider instruction ADD, Ser, Rdst. The instruction adds the source operand to the contents of register Rdst and places the sum in Rdst, the destination register.

- ➢ It shows a flowchart of a micro program for ADD, Ser, Rdst instruction.
- ➢ Each box in the chart corresponds to a microinstruction that controls the transfers and operation indicated within the box. The microinstructions isolated at the address indicated by the actual number above the upper right corner of the box. Each octal digit represents 3 bits.

Consider the point labelled α in the fig. At this point it is necessary to choose between actions required by direct and indirect addressing modes.

- ➢ If the indirect mode is specified in the instruction, then the microinstruction is location 170 is performed to fetch the operand from the memory. If the direct mode is specified, This fetch must be bypassed by branching immediately to location 171.
- ➢ The remaining micro operations required to complete execution of instruction are same and hence shared by the instructions having operand with different addressing modes.

We can say branching allows sharing of microinstructions for different micro programs and it reduces the size of control memory.

Advantages of microprogramming:

- The design of micro program control unit is less complex.
- The micro program is flexible.
- The given CPUs instruction set can be easily modified by changing the micro programs without affecting the data path.
- The debugging and maintenance of a micro programmers CPU is easy.

Disadvantages of microprogramming:

- A micro programmed CPU is slow
- For a small CPU with very limited hardware resources, a micro program CU is expensive as compared to a hardwired control unit.

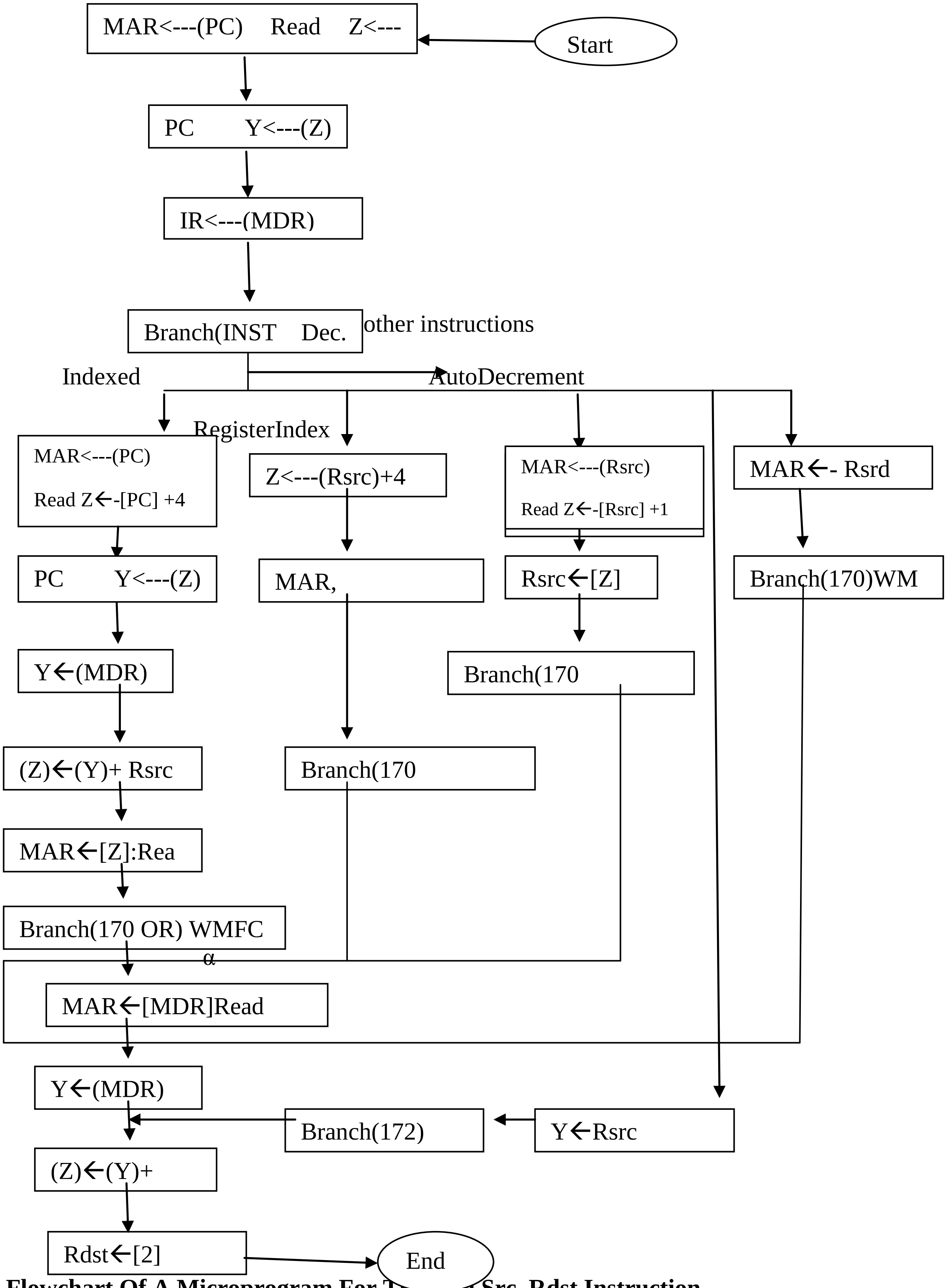


Fig: Flowchart Of A Microprogram For The Add Src, Rdst Instruction

7. Compare hardwired control unit and micro programmed control unit

S.No.	Attribute	Hardwired Control	Micro programmed Control
1.	Speed	Fast	Slow
2.	Flexibility	Implementation hardware	Implemented in software
3.	Ability to handle large/complex instructions	Somewhat difficult	Easier
4.	Design Process	Somewhat complication	Orderly and system active
5.	Ability to support operating system and diagnostic feature	Very difficult	Easy
6.	Applications	Mostly RISC	Mainframes, some microprocessor
7.	Chip area efficiency	Use least area	Use most area

8. Explain how control signals are generated using micro programmed control or

Draw the necessary diagrams and explain the control signal generation using micro programmed control. Or

How the functional field micro instruction is generated? Explain

Microinstructions:

- A simple way to structure microinstruction 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 required signals is usually large.
- Moreover only a few bits are used in any given instruction. The solution of the problem is solved by grouping the control signals.

Grouping of control signals:

- Control signals can be grouped so that all mutually exclusive signals are placed in the same group. Thus, at most one micro operations per group is specified in any microinstruction.
- Then it is possible to use a binary coding scheme to represent the signals with a group.

Example:

4 bits suffice to represent 16 available functions in the ALU. Register output control signals can be placed in a group consisting of PC_{out}, $R0_{out}$, $R1_{out}$, $R2_{out}$, $R3_{out}$ and $TEMP_{out}$, Z_{out}, MDR_{out}.

Any one of these can be selected by a unique 4-bit code. Further natural groups can be made for the remaining signals.

Gating signals: IN and OUT signals

Control signals: Read, Write, Clear A, Set carry in WMFC, END etc.

ALU Signals:

- Add, Sub, etc. There are in all 39 signals and hence each microinstruction will have 39 bits. It is not all necessary to use all 39 bits for every microinstruction because by grouping of control signals we minimize number of bits for microinstructions.

Ways to reduce number of bits in microinstruction:

1)Most signals are not needed simultaneously

2) Many signals are mutually exclusive(e.g) only one function of ALU can be activated at a time.

3) A source for data transfers must be unique which means that it should not be possible to get the contents of 2 different registers on to the bus at the same time.

4) Read and Write signals to the memory cannot be activated simultaneously.

The below figure shows an example of a partial format for the microinstructions, in which each group occupies a field large enough to contain the required codes.

- For example, the all zero pattern in F1 indicates that none of the registers that may be specified in this field should have its contents placed on the bus. An inactive code is not needed in all fields.
- For example, F4 contains 4 bits that specify one of the 16 operations performed in the ALU. Since no space code is included, the ALU is active during the execution of every microinstruction.
- However its activity is monitored by the rest of the machine through register Z, which is loaded only when the Z_{in} signal is activated.

Most fields much include one inactive code for the case in which no action is required.

F1	F2	F3		

F1(4 bits)	F2(3 bits)	F3(3 bits)	F4(4 bits)	F5(2 bits)
0000: No transfer	0000:No transfer	0000:add transfer	00:Noaction	00:Noaction
0001:PC_{out}	001:PC_{in}	001:MAR_{in}	0000:ADD	01: Read

0010:MDR$_{out}$	010:IR$_{in}$	010 MDR$_{in}$	0001:SUB	10:Write
0011:Z$_{out}$	011:Z$_{in}$	011:TEMPin	.	16 ALU Functions
0100:R0$_{out}$	100:R0$_{in}$	100:Yin	.	
0101: R1$_{out}$	101: R1$_{in}$		1111 :XOR	
0110: R2$_{out}$	110: R2$_{in}$			
0111: R3$_{out}$	111: R3$_{in}$			
1010:TEMP$_{out}$				
1011:Offset$_{out}$				

F6	F7

F6(1bit)	F7(1bit)	F8(1bit)
0:select Y	0:No action	0:Continue
1:select 4	1:WMF$_{C}$	1:End

Fig: An example of a partial format for field encoded microinstruction

.

- Grouping control signals into fields requires a little more hardware because decoding circuits must be used to decode the bit patterns of each field into individual control signals.

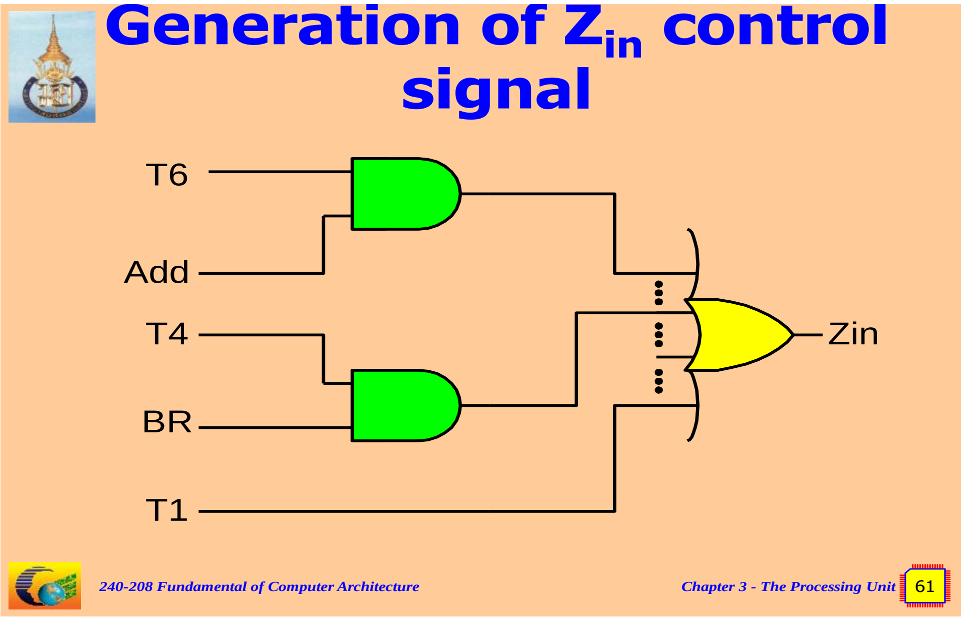
Techniques of grouping of control signals:

The grouping of control signals can be done either by using technique called vertical organisation or by using technique called horizontal organisation.

- Highly encoded scheme that use impact codes to specify only a small number of control function in each microinstruction are referred to as a vertical organization.
- On the other hand, minimally encoded scheme, in which resources can be controlled with a single instruction, is called a horizontal organization.

9. Generate thye logic circuit for the followin g functions and explain

(i). $Z_{in} = T_{1} + T_{6}$. ADD + T_{4}.BR + ...

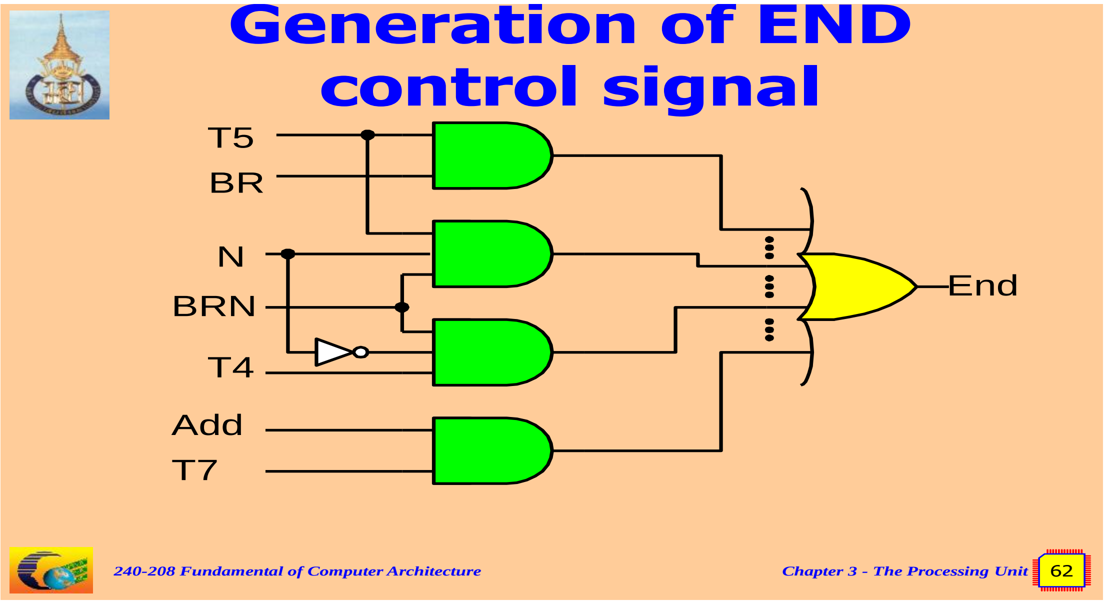


Generation of Z_{in} control signal

240-208 Fundamental of Computer Architecture

Chapter 3 - The Processing Unit 61

(ii).END = T_7. ADD + T_5 .BR+($T_{5.}$ N+ $T_{4.}$N).BRN+.............



Generation of END control signal

240-208 Fundamental of Computer Architecture

Chapter 3 - The Processing Unit 62

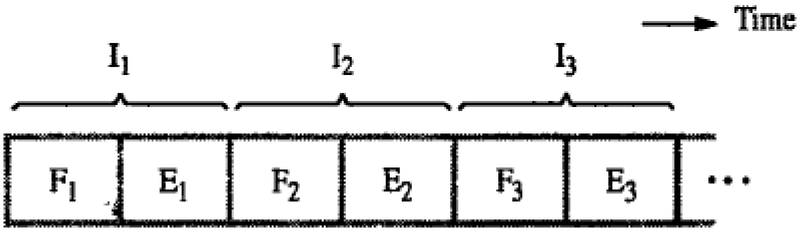
10. Explain the concepts of Pipelining.

Pipelining

It is a particularly effective way of organizing concurrent activity in a computer system.

Sequential execution

The processor executes a program by fetching and executing instructions, one after the other. Execution of a program consists of a sequence of fetch and execute steps.

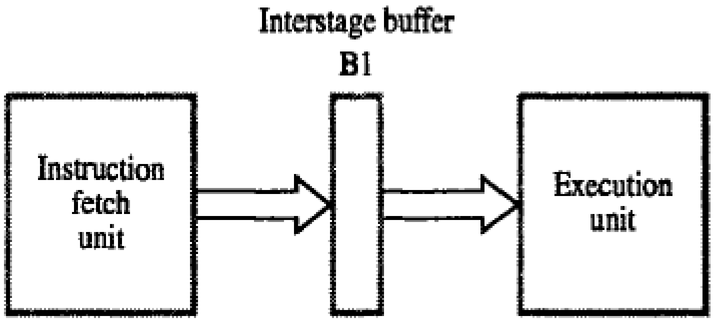


Hardware organization

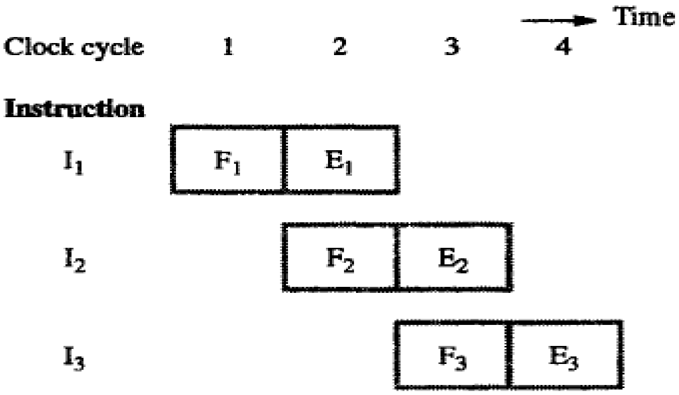
An inter-stage storage buffer, B1, is needed to hold the information being passed from one stage to the next.

New information is loaded into this buffer at the end of each clock cycle.

F Fetch: read the instruction from the memory
D Decode: decode the instruction and fetch the source operand(s)
E Execute: perform the operation specified by the instruction
W Write: store the result in the destination location



Pipelined execution



- In the first clock cycle, the fetch unit fetches an instruction I1 (step F1) and stores it in buffer B1 at the end of the clock cycle.
- In the second clock cycle the instruction fetch unit proceeds with the fetch operation for instruction I2 (step F2).

- Meanwhile, the execution unit performs the operation specified by instruction I1,which is available to it in buffer B1 (step E1).
- By the end of the second clock cycle, the execution of instruction I1 is completed and instruction I2 is available.
- Instruction I2 is stored in B1, replacing I1, which is no longer needed.
- Step E2 is performed by the execution unit during the third clock cycle, while instruction I3 is being fetched by the fetch unit.
- Four instructions are in progress at any given time. So it needs four distinct hardware units.
- These units must be capable of performing their tasks simultaneously and without interfering with one another.
- Information is passed from one unit to the next through a storage buffer.
- During clock cycle 4, the information in the buffers is as follows:
- Buffer B1 holds instruction I3, which was fetched in cycle 3 and is being decoded by the instruction-decoding unit.
- Buffer B2 holds both the source operands for instruction I2 and the specifications of the operation to be performed. This is the information produced by the decoding hardware in cycle 3.
    - The buffer also holds the information needed for the write step of instruction I2(step W2).
    - Even though it is not needed by stage E, this information must be passed on to stage W in the following clock cycle to enable that stage to perform the required write operation.
- Buffer B3 holds the results produced by the execution unit and the destination information for instruction I1.

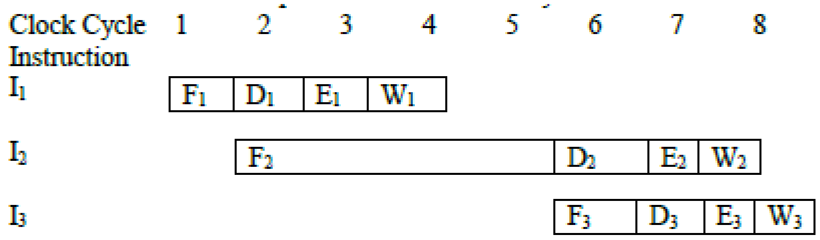
11. Write short notes on Pipeline performance

Pipeline performance

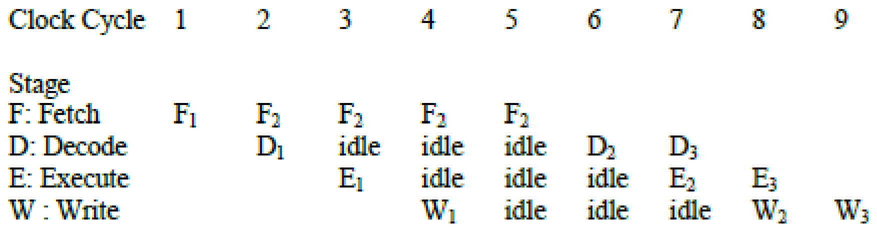
- Pipelining is proportional to the number of pipeline stages.
- For variety of reasons, one of the pipeline stages may not be able to complete its processing task for a given instruction in the time allotted.
- For eg, stage E in the four-stage pipeline is responsible for arithmetic and logic operations and one clock cycle is assigned for this task.
- Although this may be sufficient for most operations some operations such as divide may require more time to complete.
- Instruction I2 requires 3 cycles to complete from cycle 4 through cycle 6.
- Thus in cycles 5 and 6 the write stage must be told to do nothing, because it has no data to work with.
- Meanwhile, the information in buffer B2 must remain intact until the execute stage has completed its operation.
- This means that stage 2 and in turn stage1 are blocked from accepting new instructions because the information in B1 cannot be overwritten.
- Thus steps D4 and F5 must be postponed.
- The pipeline may also be stalled because of a delay in the availability of an instruction.

- For example, this may be a result of a miss in the cache, requiring the instruction to be fetched from the main memory.
- Such hazards are often called control hazards or instruction hazards.

Instruction execution steps in successive clock cycles:



Function performed by each process stage in successive clock cycles



Clock Cycle	1	2	3	4	5	6	7	8	9
Stage									
F: Fetch	F_1	F_2	F_2	F_2	F_2				
D: Decode		D_1	idle	idle	idle	D_2	D_3		
E: Execute			E_1	idle	idle	idle	E_2	E_3	
W : Write				W_1	idle	idle	idle	W_2	W_3

- Instruction I1 is fetched from the cache in cycle1,and its execution proceeds normally.
- Thus in cycles 5 and 6, the write stage must be told to do nothing, because it has no data to work with.
- Meanwhile, the information in buffer B2 must remain intact until the execute stage has completed its operation.
- This means that stage 2 and in turn stage1 are blocked from accepting now instructions because the information in B1 cannot be overwritten.
- Thus steps D4 and F4 must be postponed.
- Pipelined operation is said to have been stacked for two clock cycles.
- Normal pipelined operation resumes in cycle 7.

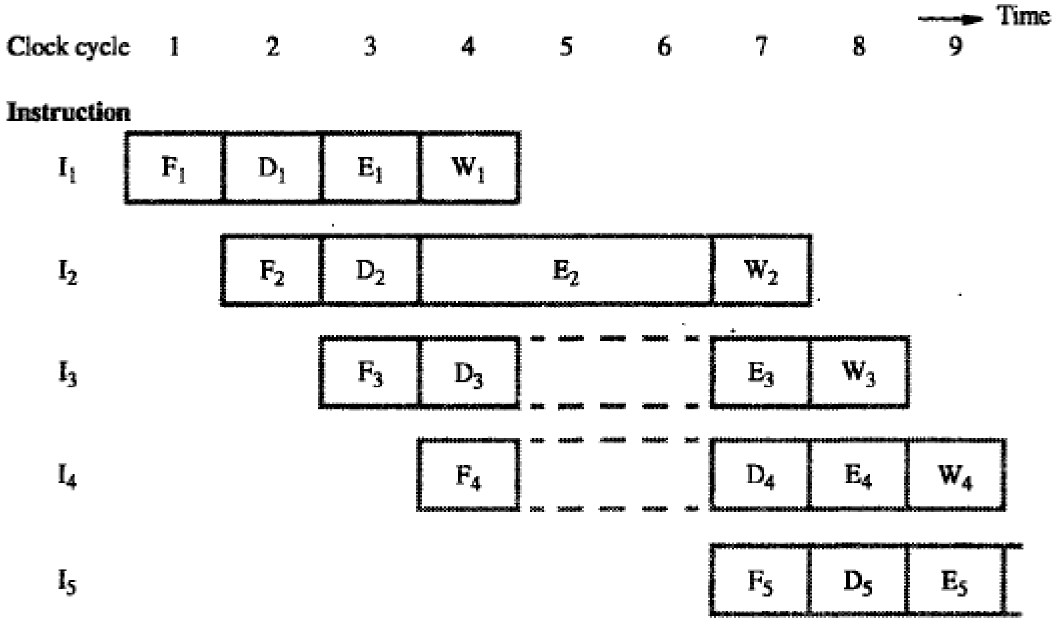
12. Define hazard. What are types of hazards?

Hazard

Any condition that causes the pipeline to stall is called a hazard. Consider the following example where execution takes more than a cycle. Here the pipelined operation is said to have been stalled for two clock cycles.

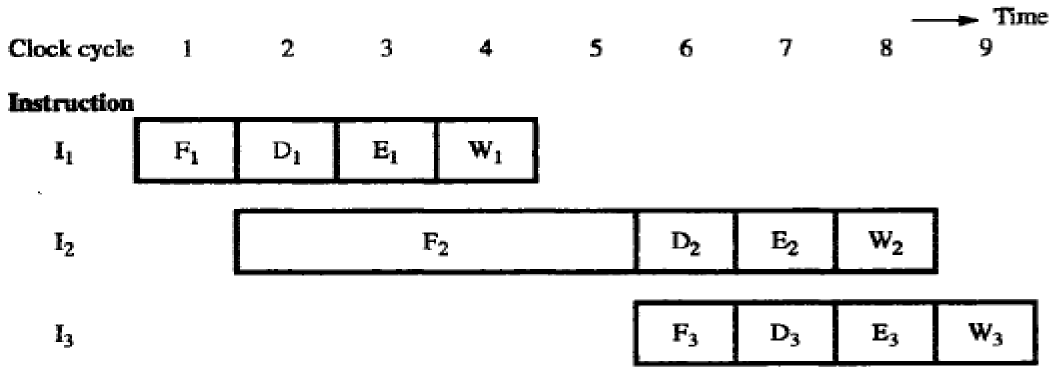
Data Hazard

A data hazard is any condition in which either the source or the destination operands of an instruction are not available at the time expected in the pipeline. As a result, some operation has to be delayed, and the pipeline stalls.



Control Hazards

The pipeline may also be stalled because of a delay in the availability of an instruction. For example, this may a result of a miss in the cache, requiring the instruction to be fetched from the main memory. Such hazards are often called control hazards or instruction hazards.



(a) Instruction execution steps in successive clock cycles

Clock cycle	1	2	3	4	5	6	7	8	9
Stage									
F: Fetch	F_1	F_2	F_2	F_2	F_2	F_3			
D: Decode		D_1	idle	idle	idle	D_2	D_3		
E: Execute			E_1	idle	idle	idle	E_2	E_3	
W: Write				W_1	idle	idle	idle	W_2	W_3

(b) Function performed by each processor stage in successive clock cycles

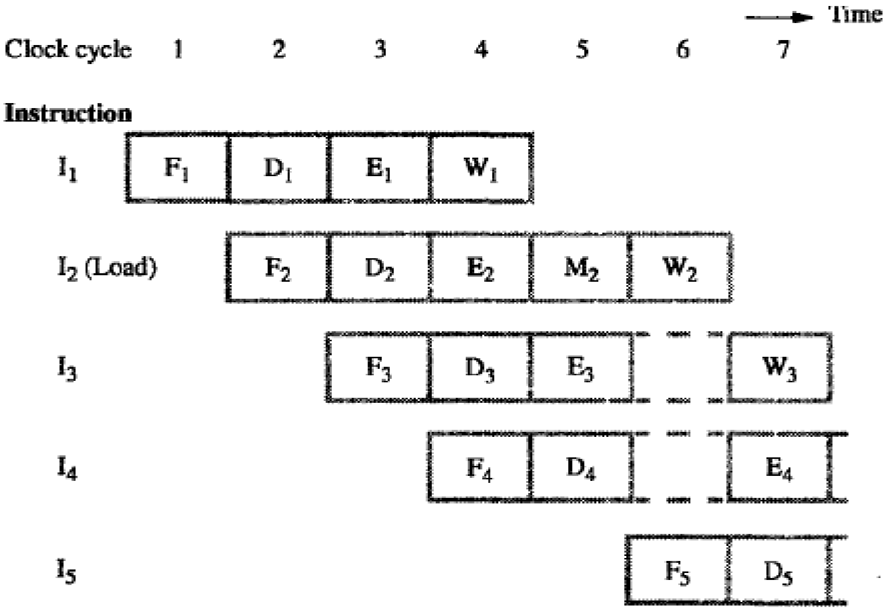
Pipeline stall caused by a cache miss in F2.

Structural Hazards

A third type of hazard that may be encountered in pipelined operation is known as a structural hazard. This is the situation when two instructions require the use of a given hardware resource at the same time. The most common case in which this hazard may arise is in access to memory. Many processors use separate instruction and data caches to avoid this delay.

Example of the structural hazard is shown below.

Load X(R1),R2



13. Explain Data Hazards in detail.

A data hazard is a situation in which the pipeline is stalled because the data to be operated on are delayed for some reason. Consider the program that contains to instructions I1 followed by I2. When this program is executed in a pipeline, the execution of I2 can begin before the execution of I1 is completed. This means that the results generated by I1 may not be available for use by I2.

- We must ensure that the results obtained when instructions are executed in a pipelined processor are identical to those obtained when the same instructions are executed sequentially.
- Hazard occurs

  $A \leftarrow 3 + A$

  $B \leftarrow 4 \times A$
- No hazard

  $A \leftarrow 5 \times C$

  $B \leftarrow 20 + C$
- When two operations depend on each other, they must be executed sequentially in the correct order.

- Another example:

  Mul R2, R3, R4

  Add R5, R4, R6

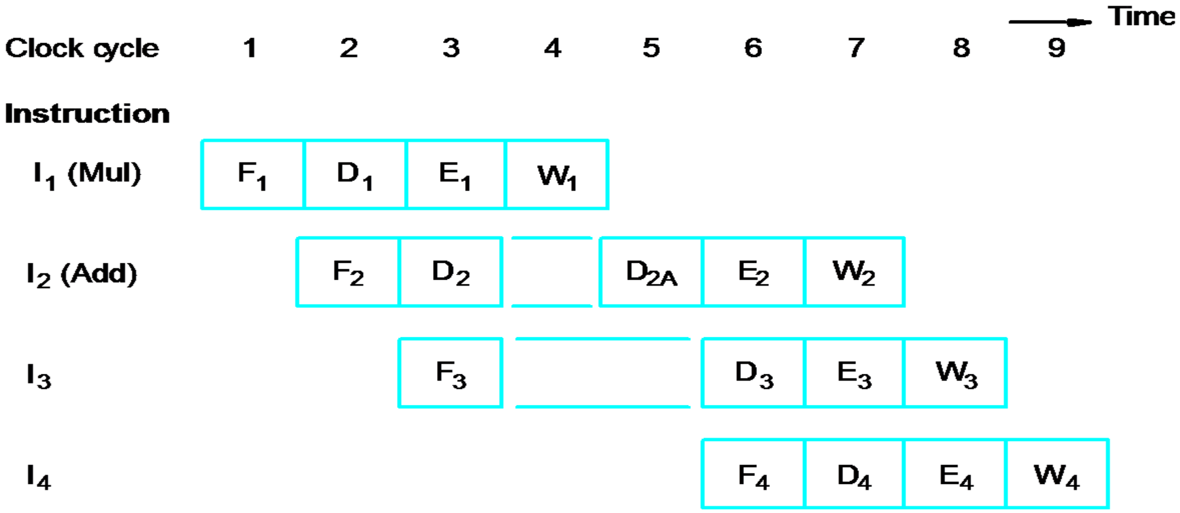


Fig. Pipeline stalled by data dependency between D2 and W1

Operand Forwarding

- Instead of from the register file, the second instruction can get data directly from the output of ALU after the previous instruction is completed.
- A special arrangement needs to be made to “forward” the output of ALU to the input of ALU.

Fig. Operand forwarding in a pipelined processor

Handling Data Hazards in Software

- Let the compiler detect and handle the hazard:

```
I1: Mul  R2, R3, R4
    NOP
    NOP
I2: Add  R5, R4, R6
```

- The compiler can reorder the instructions to perform some useful work during the NOP slots.

Side Effects

- The previous example is explicit and easily detected.
- Sometimes an instruction changes the contents of a register other than the one named as the destination.

- When a location other than one explicitly named in an instruction as a destination operand is affected, the instruction is said to have a side effect. (Example?)
- Example: conditional code flags:

```
Add  R1, R3

AddWithCarry  R2, R4
```

- Instructions designed for execution on pipelined hardware should have few side effects.

14. Explain Instruction Hazards in detail. (Apr 12)

The purpose of the instruction fetch unit is to supply the execution units with a steady stream of instructions. Whenever this stream is interrupted, the pipeline stalls, as cache miss. A branch instruction may also cause the pipeline to stall.

Unconditional Branches

Fig. An idle cycle caused by a branch instruction

Branch Timing

Fig. Branch timing

Instruction Queue and Prefetching

Either a cache miss or a branch instruction stalls the pipeline for one or more clock cycles. To reduce the effect of these interruptions, many processors employ sophisticated fetch units that can fetch instructions before they are needed and put them in a queue.

A separate unit, called the dispatch unit, takes instructions from the front of the queue and sends them to the execution unit.

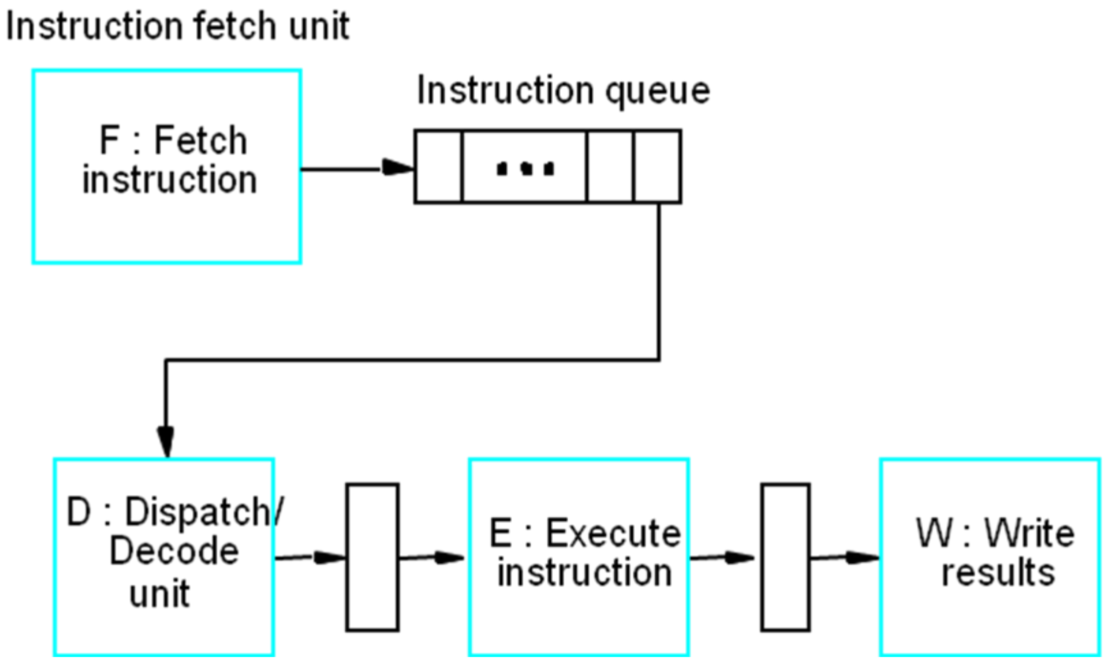


Fig. Use of instruction queue in hardware organization

Conditional Braches

- A conditional branch instruction introduces the added hazard caused by the dependency of the branch condition on the result of a preceding instruction.
- The decision to branch cannot be made until the execution of that instruction has been completed.
- Branch instructions represent about 20% of the dynamic instruction count of most programs.

Delayed Branch

- The instructions in the delay slots are always fetched. Therefore, we would like to arrange for them to be fully executed whether or not the branch is taken.
- The objective is to place useful instructions in these slots.
- The effectiveness of the delayed branch approach depends on how often it is possible to reorder instructions.

```
LOOP      Shift_left     R1
          Decrement      R2
          Branch=0       LOOP
NEXT      Add            R1,R3
```

(a) Original program loop

```
LOOP      Decrement      R2
          Branch=0       LOOP
          Shift_left     R1
NEXT      Add            R1,R3
```

Fig. Reordering of instructions for a delayed branch

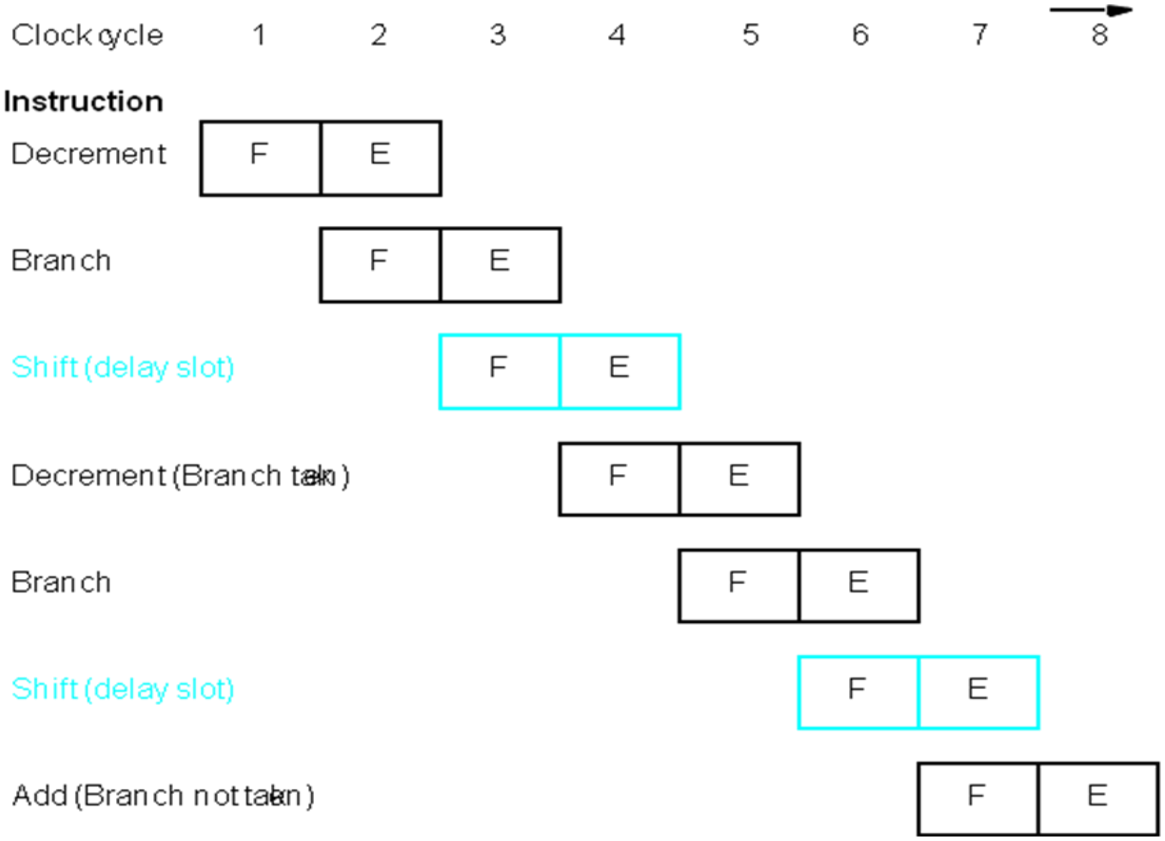


Fig. Execution timing showing the delay slot being filled during the last two passes through the loop

Branch Prediction

Another technique for reducing the branch penalty associated with conditional branches is to attempt to predict whether or not a particular branch will be taken. Until the branch condition is evaluated, instruction execution along the predicted path must be done on a speculative basis.

- Speculative execution means that instructions are executed before the processor is certain that they are in the correct execution sequence.
- Need to be careful so that no processor registers or memory locations are updated until it is confirmed that these instructions should indeed be executed.

Incorrectly Predicted Branch

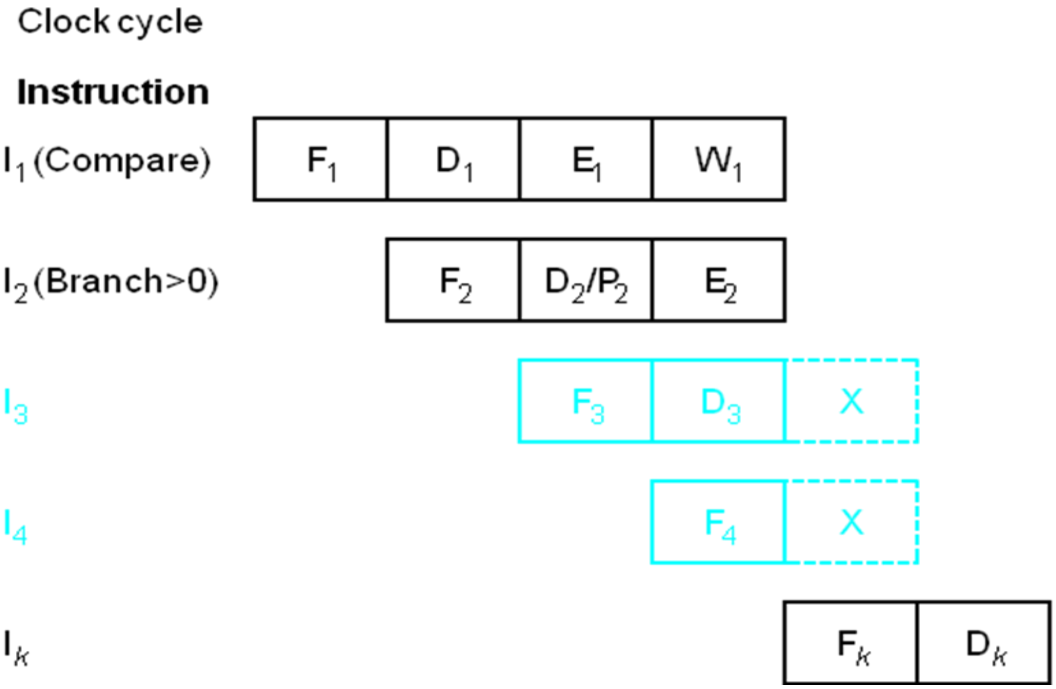


Fig. Timing when a branch decision has been incorrectly predicted as not taken.

- Better performance can be achieved if we arrange for some branch instructions to be predicted as taken and others as not taken.
- Use hardware to observe whether the target address is lower or higher than that of the branch instruction.
- Let compiler include a branch prediction bit.
- So far the branch prediction decision is always the same every time a given instruction is executed – static branch prediction.

15. Write in detail about the influence of Pipelining on Instruction Sets

Influence on Instruction Sets

Some instructions are much better suited to pipeline execution than others. Two key aspects of machine instructions are;

- Addressing modes

- Conditional code flags

Addressing Modes

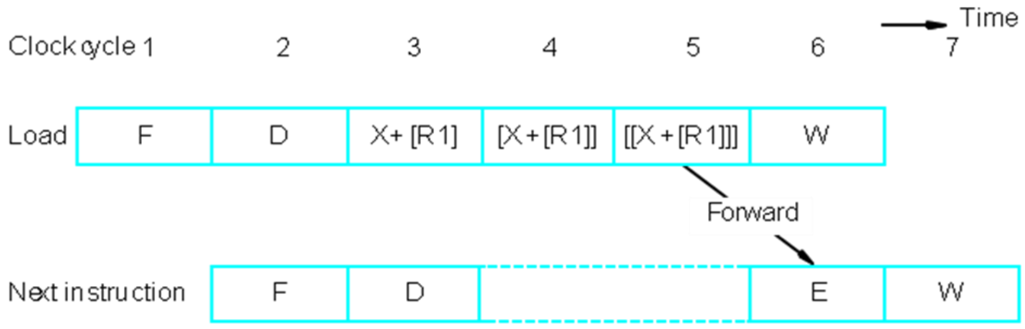
Addressing modes should provide the means for accessing a variety of data structures simply and efficiently. Useful addressing modes include index, indirect, auto-increment, and auto-decrement. In choosing the addressing modes to be implemented in a pipelined processor, we must consider the effect of each addressing mode on instruction flow in the pipeline. Two important considerations are the side effects of modes such as auto-increment and auto-decrement and the extent to which complex addressing modes cause the pipeline to stall. Another important factor is whether a given mode is likely to be used by compilers.

Load X(R1), R2

Fig. Effect of a load instruction on pipeline timing

Complex Addressing Mode

Load (X(R1)), R2



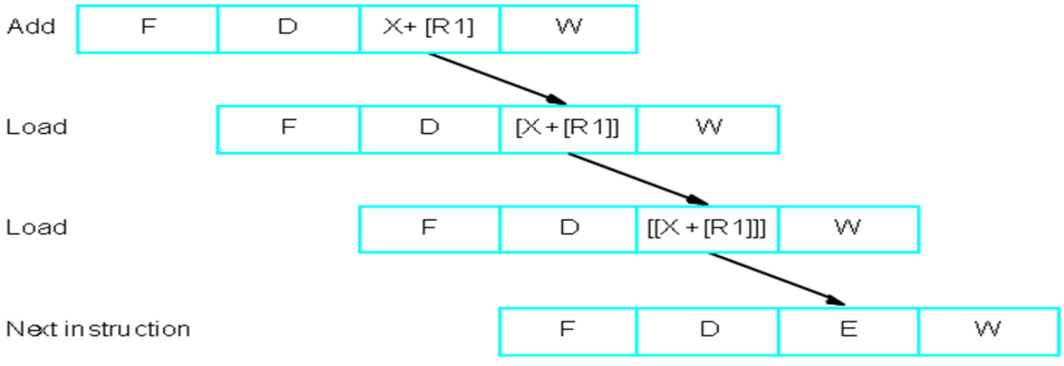
(a) Complex addressing mode

Simple Addressing Mode

Add #X, R1, R2

Load (R2), R2

Load (R2), R2



(b) Simple addressing mode

- In a pipelined processor, complex addressing modes do not necessarily lead to faster execution.

Advantage: reducing the number of instructions / program space

Disadvantage: cause pipeline to stall / more hardware to decode / not convenient for compiler to work with

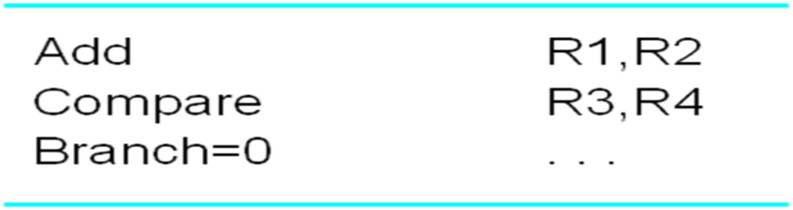
Conclusion: complex addressing modes are not suitable for pipelined execution.

Good addressing modes should have:

- Access to an operand does not require more than one access to the memory
- Only load and store instruction access memory operands

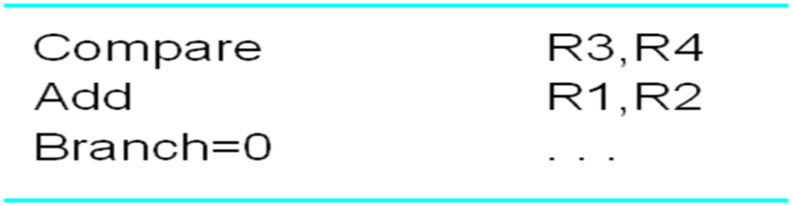
16. What are condition codes? Explain.

In many processors, the condition code flags are stored in the processor status register. They are either set or cleared by many instructions, so that they can be tested by subsequent conditional branch instructions to change the flow of program execution. An optimizing compiler for a pipelined processor attempts to reorder instructions to avoid stalling the pipeline when branches or data dependencies between successive instructions occur. In doing so, the compiler must ensure that reordering does not cause a change in the outcome of a computation. The dependency introduced by the condition-code flags reduces the flexibility available for the compiler to reorder instructions.



```
Add                      R1,R2
Compare                  R3,R4
Branch=0                 . . .
```

(a) A program fragment



```
Compare                  R3,R4
Add                      R1,R2
Branch=0                 . . .
```

(b) Instructions reordered

- Two conclusion:

➢ To provide flexibility in reordering instructions, the condition-code flags should be affected by as few instruction as possible.

➢ The compiler should be able to specify in which instructions of a program the condition codes are affected and in which they are not.

17. What are the considerations needed for using pipelined design?

Original design

Fig. Three bus organization of the data path

Pipelined Design

Separate instruction and data caches

- PC is connected to IMAR

- DMAR

- Separate MDR

- Buffers for ALU

- Instruction queue

- Instruction decoder output

- Reading an instruction from the instruction cache

- Incrementing the PC

- Decoding an instruction

- Reading from or writing into the data cache

- Reading the contents of up to two regs
- Writing into one register in the reg file
- Performing an ALU operation

Fig. Data path modified for pipelined execution with inter-stage buffers at input and output of ALU

18. Explain in detail about Superscalar operation.

Superscalar Operations

Pipelining makes instructions to execute concurrently. Several instructions are present in the pipeline at the same time, but they are in different stages of their execution. While one instruction is performing an ALU operation, another instruction is being decoded and yet another is being fetched from the memory. In the absence of hazards, one instruction enters the pipeline and one instruction completes execution in each clock cycle. This means that the maximum throughput of a pipelined processor is one instruction per clock cycle.

A more aggressive approach is to equip the processor with multiple processing units to handle several instructions in parallel in each processing stage. With this arrangement, several instructions start execution in the same clock cycle, and the processor is said to use multiple-issue. Such processors are capable of achieving an instruction execution throughput of more than one instruction per cycle. They are known as superscalar processors.

Fig. A processor with two execution units

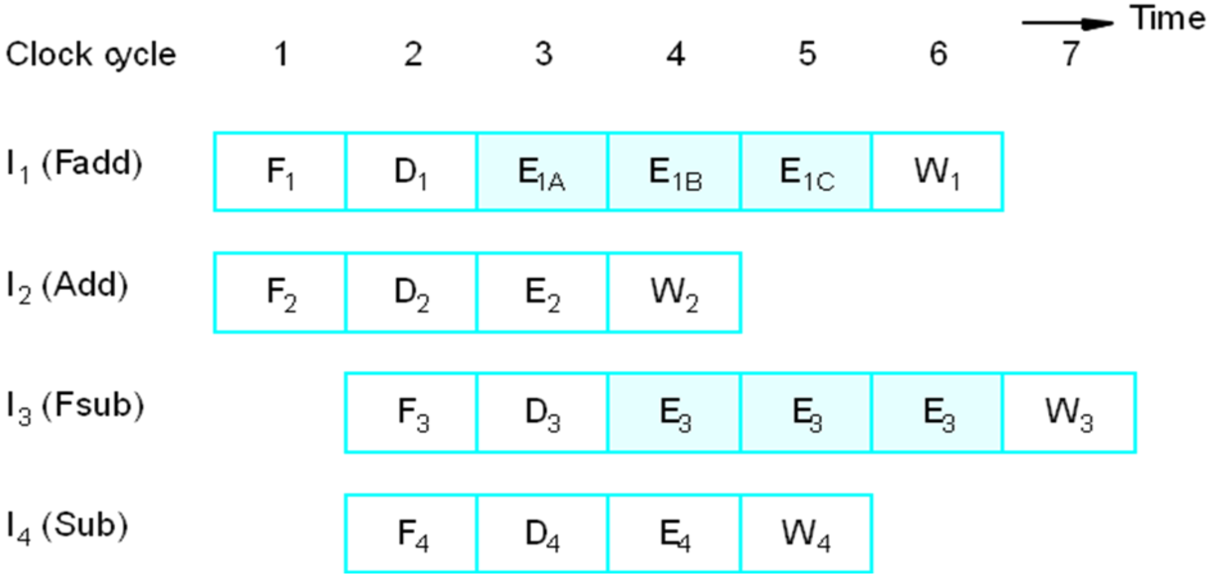
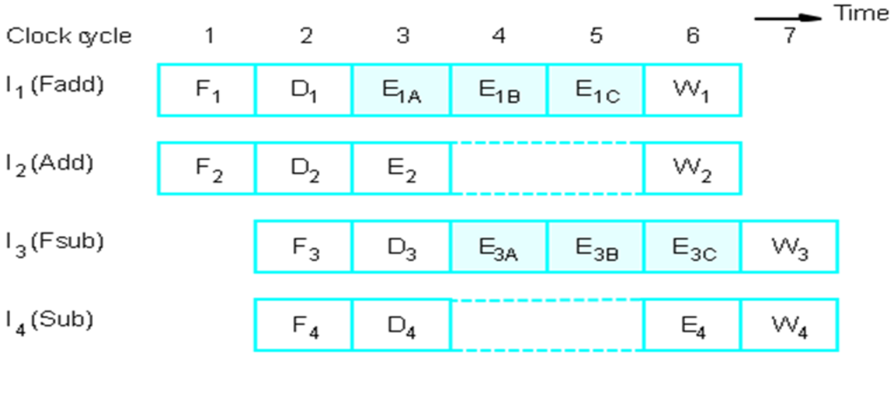


Fig.An example of instruction execution flow in the processor assuming no hazards are encountered.

Out Of Order Execution

Exceptions may be caused by a bus error during an operand fetch or by an illegal operation, such as an attempt to divide by zero. the program counter points to the instruction have been executed to completion. If such a situation is permitted, the processor is said to have imprecise exceptions.

If an exception occurs during an instruction, all subsequent instructions that may have been partially executed are discarded. This is called a precise exception.



(a) Delayed write

Execution Completion

- It is desirable to used out-of-order execution, so that an execution unit is freed to execute other instructions as soon as possible.
- At the same time, instructions must be completed in program order to allow precise exceptions.

The results are written into temporary registers. The content of these registers are later transferred to the permanent registers in correct program order. This step is often called the commitment step because the effect of the instruction cannot be reversed after that point. If an instruction causes an exception, the results of any subsequent instruction that has been executed would still be in temporary registers and can safely be discarded.

When out-of-order execution is allowed, a special control unit is needed to guarantee in-order commitment. This is called the commitment unit. It uses a queue called the reorder buffer to determine which instruction should be committed next.

An instruction is said to have retired only when it is at the head of the queue, all the instructions that were dispatched before it must also have been retired. Hence, instructions may complete execution out of order, but they are retired in program order.

19. List out the performance considerations using pipeline.

- The execution time T of a program that has a dynamic instruction count N is given by:

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

where S is the average number of clock cycles it takes to fetch and execute one instruction, and R is the clock rate.

- Instruction throughput is defined as the number of instructions executed per second.

$$P_s = \frac{R}{S}$$

- An *n*-stage pipeline has the potential to increase the throughput by *n* times.
- However, the only real measure of performance is the total execution time of a program.
- Higher instruction throughput will not necessarily lead to higher performance.

Number of Pipeline Stages

- Since an *n*-stage pipeline has the potential to increase the throughput by *n* times, how about we use a 10,000-stage pipeline?
- As the number of stages increase, the probability of the pipeline being stalled increases.
- The inherent delay in the basic operations increases.
- Hardware considerations (area, power, complexity,)

PONDICHERRY UNIVERSITY QUESTIONS

2 MARKS

1. What is micro program? (Apr 13) (Pg. No. 1) (Qn. No. 6)

2. Define instruction cycle. (Nov 12) (Pg. No. 3) (Qn. No. 22)

11 MARKS

1. Explain micro programmed control unit. What are the advantages and disadvantage of it? OR

With a neat diagram explain the internal organization of a processor. (Apr 11) (Pg. No. 30) (Qn. No. 6)

2. Explain Instruction Hazards in detail. (Apr 12) (Pg. No. 50) (Qn. No. 14)