BSCS- 511 Computer Organization And Architecture

LECTURE – 01 (CHAPTER 01: INTRODUCTION)

COMPUTER ARCHITECTURE	COMPUTER ORGANIZATION
It refers to those attributes that have a direct impact on the execution of a program.	It refers to the organizational units and their interconnection that realize the architectural specifications.
It refers to those attributes of a system visible to a programmer.	The organizational attributes are those hardware features and details which are transparent to the programmer.
Examples of architectural attributes: the instruction set, the number of bits used to represent various data types, I/O mechanisms and techniques for addressing memory.	Examples of organizational attributes: interfaces between the computer and peripherals, and the memory technology used.
It is an architectural design issue whether a computer will have multiply instruction.	It is an organizational issue whether that instruction will be implemented using a simple multiply unit or by a mechanism that makes repeated use of the add unit of the system.
An architecture may survive for long.	An organization changes more frequently than the architecture.

Evolution:

- Increase in processor size, memory size, I/O speed and capacity
- Decrease in component size

Ways to improve performance:

- By improvement in computer organization
- By heavy use of pipelining, use of parallel execution techniques, speculative execution techniques etc.

COMPUTER STRUCTURE AND FUNCTION:

Structure: The way in which the components are interrelated

Function: The operation of each individual component as part of the structure.

Four main functions:

- Data processing
- Data storage
- Data movement
- Control

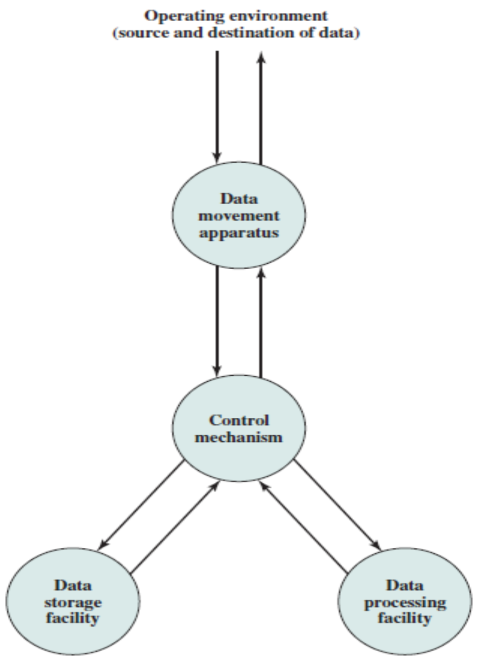


Figure 1.1 A Functional View of the Computer

Four main structural components:

- Central Processing Unit: Controls the operation of the computer and performs its data processing functions.
- Main memory: Stores data.
- I/O: Moves the data between the computer and its external environment.
- System interconnection: Some mechanism that provides for communication among CPU, main memory and I/O.

Major structural components of CPU:

- Control Unit: Controls the operation of the processor and hence the computer.
- Arithmetic and Logic Unit: Performs the computer's data processing functions.
- Registers: Provides storage internal to the CPU.
- CPU interconnection: Some mechanism that provides for communication among the CU, ALU, and registers.

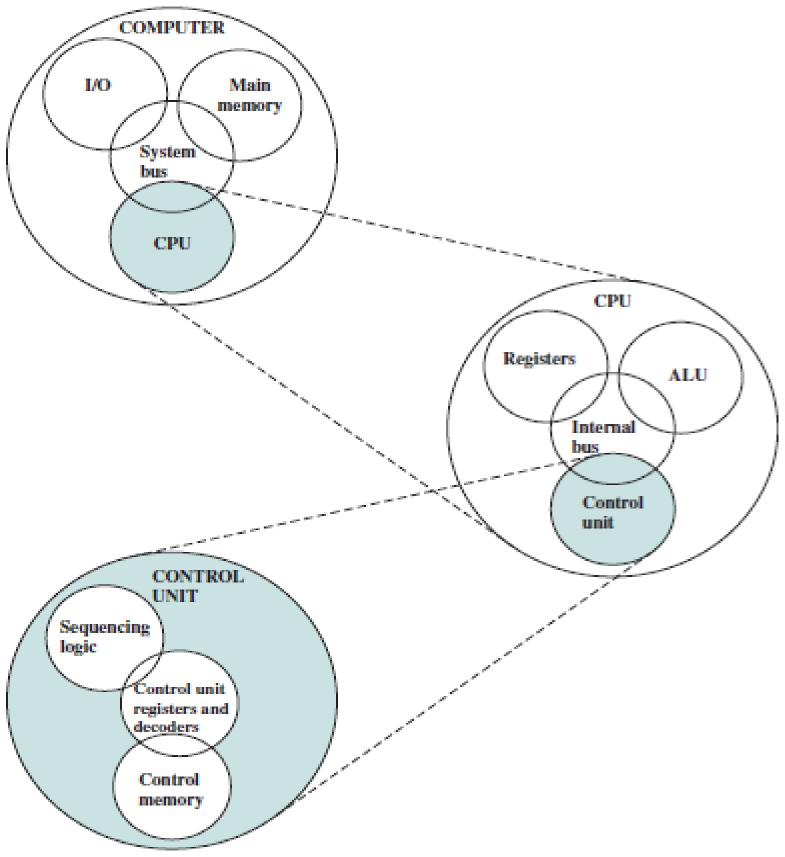


Figure 1.4 The Computer: Top-Level Structure

LECTURE – 02 (CHAPTER 02: COMPUTER EVOLUTION AND PERFORMANCE)

In 1946, von Neumann and his colleagues began the design of a new stored program computer, referred to as the IAS computer.
The IAS computer, although not completed until 1952, is the prototype of all subsequent general-purpose computers.

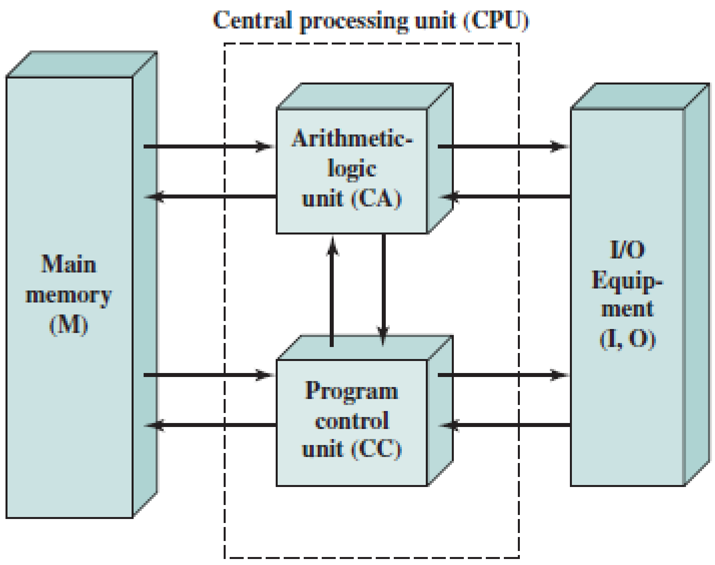

Central processing unit (CPU)
Main memory (M)
Arithmetic-logic unit (CA)
Program control unit (CC)
I/O Equip-ment (I, O)

Figure 2.1 Structure of the IAS Computer

The IAS computer consists of:
- A main memory, which stores both data and instructions in the same form.
- An arithmetic and logic unit capable of operating on binary data.
- A control unit, which interprets the instructions in memory and causes them to be executed.
- I/O equipment operated by the control unit.
- Interconnections.

Key points of the Von Neumann Architecture:
- Data and instructions are stored in a single read write memory.
- The contents of this memory is addressable by location without regard to the type of data it contains.
- Execution occurs in a sequential order (unless explicitly modified) from one instruction to the next.

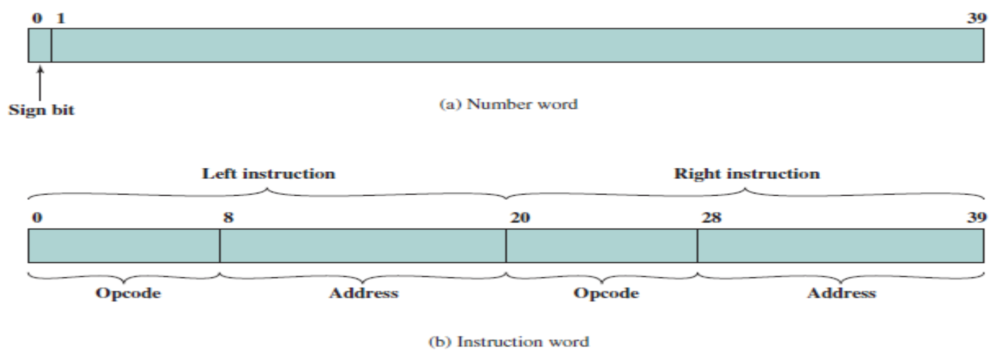


Figure 2.2 IAS Memory Formats

- Memory of IAS consists of 1000 storage locations, called words, of 40 binary digits each.
- Numbers are represented in binary form and each instruction is a binary code.
- Each number is represented by a sign bit and 39 bit value (Fig 2.2a)
- A word may also contain two 20-bit instructions, with each instruction consisting of 8 bit operation code specifying the instruction to be performed and a 12 (total memory locations: 2^{12} = 4096) bit designating one of the words in the memory (ranging from 0 to 999)

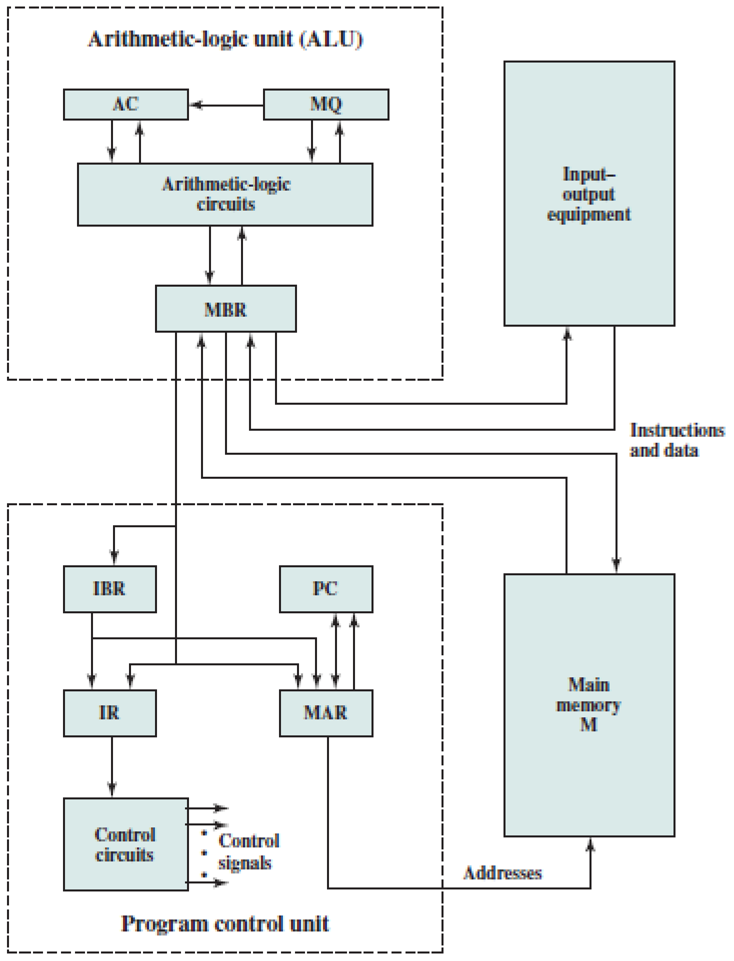


Figure 2.3 Expanded Structure of IAS Computer

Both the control unit and the ALU contain storage locations, called registers.

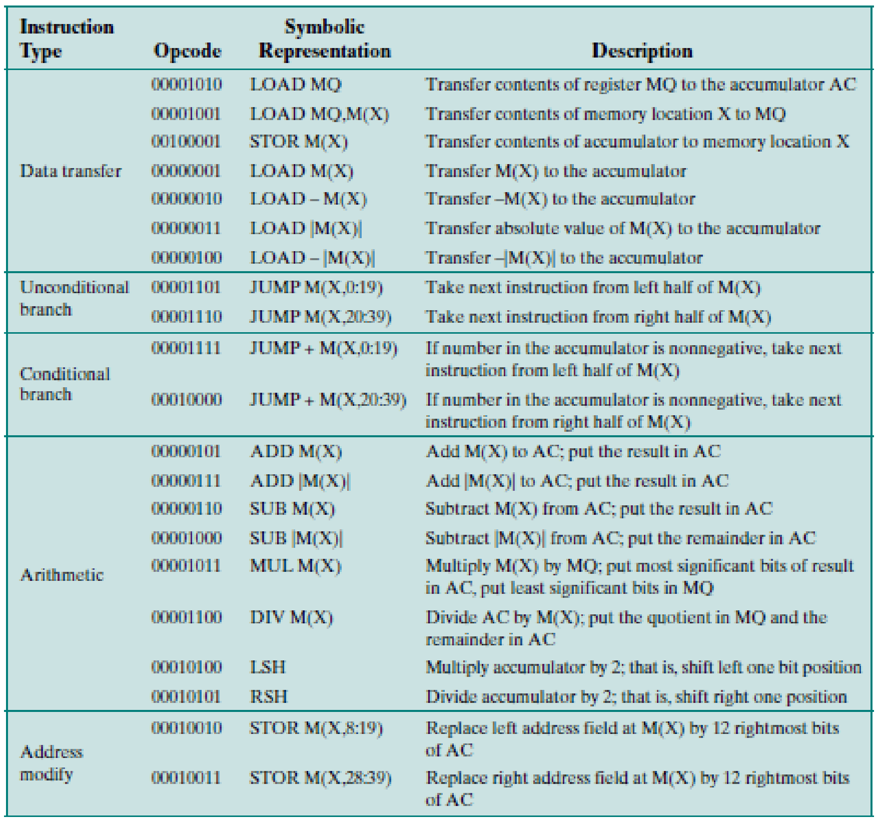
- Memory buffer registers (MBR): Contains a word to be sent to the memory or the I/O unit, or is used to receive a word from the memory or the I/O unit.
- Memory address registers (MAR): Specifies the address in the memory of the word to written from or read into the MBR.
- Instruction register (IR): Contains the 8-bit opcode instruction being executed.
- Instruction buffer register (IBR): Employed to store temporarily the right hand instruction from a word in memory.
- Program counter (PC): Contains the address of the next instruction pair to be fetched from the memory.
- Accumulator (AC) and multiplier quotient (MQ): Employed to hold the temporarily operands and results of ALU operations.

- IAS operates by repetitively performing an instruction cycle.
- Each instruction cycle consists of two sub cycles: fetch cycle and execute cycle.

The IAS computer had a total of 21 instructions. These can be grouped as follows:

- Data transfer: Move data between memory and ALU registers or between two ALU registers.
- Unconditional branch: Normally, the CU executes instruction in sequence from memory. This sequence can be changed by a branch instruction, which facilitates repetitive operations.
- Arithmetic: Operations performed by the ALU.
- Address modify: Permits addresses to be computed in the ALU and then inserted into instructions stored in memory. This allows a program considerable addressing flexibility.

Table 2.1 The IAS Instruction Set



Instruction Type	Opcode	Symbolic Representation	Description
Data transfer	00001010	LOAD MQ	Transfer contents of register MQ to the accumulator AC
	00001001	LOAD MQ,M(X)	Transfer contents of memory location X to MQ
	00100001	STOR M(X)	Transfer contents of accumulator to memory location X
	00000001	LOAD M(X)	Transfer M(X) to the accumulator
	00000010	LOAD – M(X)	Transfer –M(X) to the accumulator
	00000011	LOAD |M(X)|	Transfer absolute value of M(X) to the accumulator
	00000100	LOAD – |M(X)|	Transfer –|M(X)| to the accumulator
Unconditional branch	00001101	JUMP M(X,0:19)	Take next instruction from left half of M(X)
	00001110	JUMP M(X,20:39)	Take next instruction from right half of M(X)
Conditional branch	00001111	JUMP + M(X,0:19)	If number in the accumulator is nonnegative, take next instruction from left half of M(X)
	00010000	JUMP + M(X,20:39)	If number in the accumulator is nonnegative, take next instruction from right half of M(X)
Arithmetic	00000101	ADD M(X)	Add M(X) to AC; put the result in AC
	00000111	ADD |M(X)|	Add |M(X)| to AC; put the result in AC
	00000110	SUB M(X)	Subtract M(X) from AC; put the result in AC
	00001000	SUB |M(X)|	Subtract |M(X)| from AC; put the remainder in AC
	00001011	MUL M(X)	Multiply M(X) by MQ; put most significant bits of result in AC, put least significant bits in MQ
	00001100	DIV M(X)	Divide AC by M(X); put the quotient in MQ and the remainder in AC
	00010100	LSH	Multiply accumulator by 2; that is, shift left one bit position
	00010101	RSH	Divide accumulator by 2; that is, shift right one position
Address modify	00010010	STOR M(X,8:19)	Replace left address field at M(X) by 12 rightmost bits of AC
	00010011	STOR M(X,28:39)	Replace right address field at M(X) by 12 rightmost bits of AC

LECTURE – 03 (CHAPTER 03: TOP LEVEL VIEW OF COMPUTER FUNCTION AND INTERCONNECTION 3.1, 3.2 NOT INCLUDED)

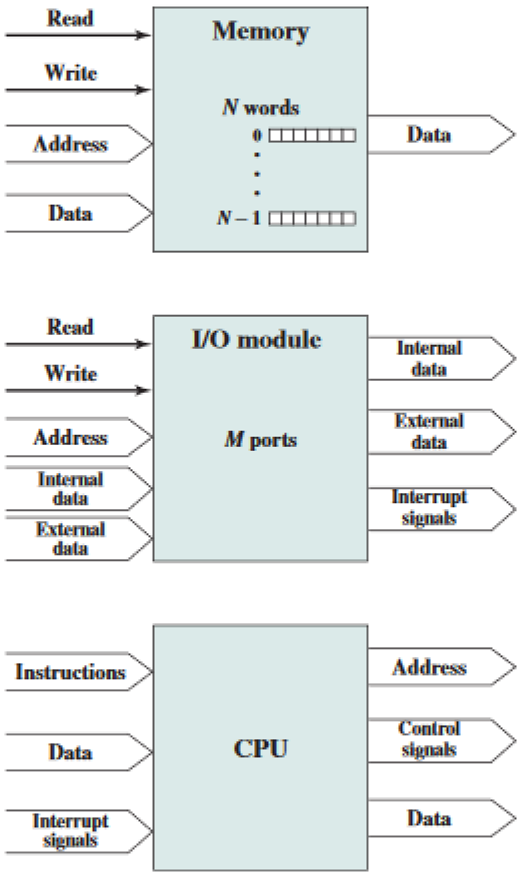


Figure 3.15 Computer Modules

- A computer consists of a set of modules of three basic types (memory, I/O, and processor) that communicate with each other.
- The connection of paths connecting the various modules is called the interconnection structure.
- Design of interconnection structure depends on the nature of exchanges that must be made among modules.

Types of exchanges that are needed for each type of module:

- **Memory module:**
  - Each memory module consists of N words of equal length.
  - Each word is assigned a unique numerical address (0, 1, …, N-1)
  - The nature of operation is indicated by the read and write control signals.
  - The location of the operation is specified by an address.
  - Data is to be written or read.
- **I/O module:**
  - From internal point of view, the functionality of I/O module is similar to memory module.
  - Nature of operation → read and write

    - Address → port address identifying the input and output devices.
    - Data → to and from external devices
    - Data → to and from microprocessor
    - Interrupt → to the processor
- **Processor:**
    - Reads in instruction and data
    - Writes out data after processing
    - Uses control signals to control the overall operation of the system.
    - Receives interrupt from I/O ports.

BUS INTERCONNECTION:

- A bus is a communication pathway connecting one or more devices by a shared transmission medium.
- In bus interconnection, only one device at a time can successfully transmit, others can only receive.
- Typically, bus consists of multiple lines. Each line capable of transmitting a single bit.
- Several lines of a bus can be used to transmit binary digits in parallel.
- Computer system contains number of different buses to connect components at various levels of computer system hierarchy.
- A bus that connects major computer components is called a system bus.
- The most common computer interconnection structures are based on the use of one or more system buses.

Bus Structure:

- A typical bus consists of about 50 to 100 separate lines.
- Each line is assigned a particular meaning or function.
- The lines can be classified into three functional groups:
    - **Data lines:** provide a path for moving data among system modules. (no. of lines = width of data bus) The number of lines determine how many bits can be transferred at a time.
    - **Address lines:** used to designate the source or destination of the data on the data bus. The width of the address bus determines the maximum possible memory capacity of the system.
    - **Control lines:** used to control the access to and the use of the data and address lines. Control signals transmit both timing and command information among system modules.
      Example control signals: memory read, memory write, I/O read, I/O write, transfer ACK, bus request, bus grant, interrupt request, interrupt ACK, clock, reset.

Operation of the bus:
If one module wishes to send data to another
- Obtain the use of the bus.
- Transfer data via the bus.

If one module wishers to request data from another
- Obtain the use of the bus
- Transfer a request to the other module over the appropriate control and address lines.

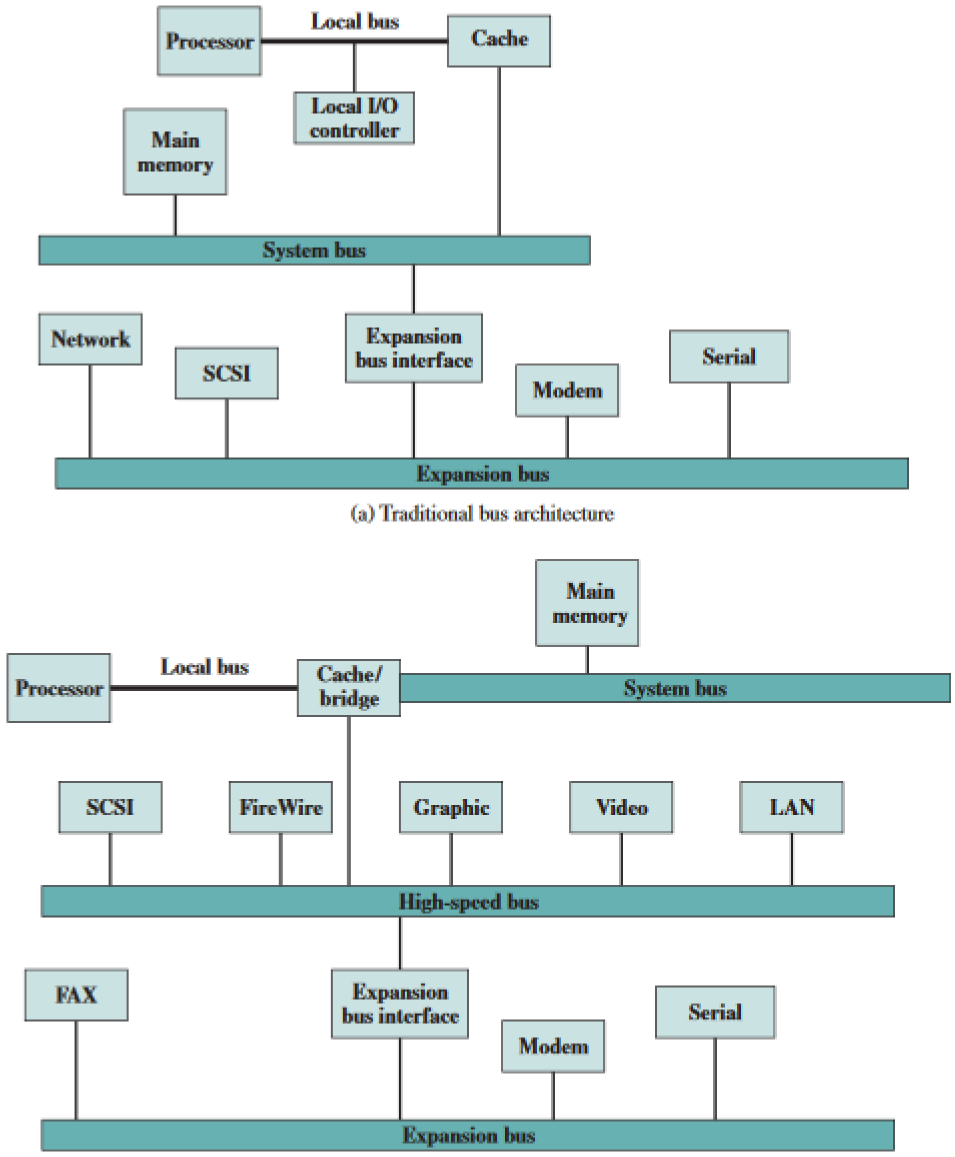


Figure 3.17 Example Bus Configurations

LECTURE – 04

ELEMENTS OF BUS DESIGN

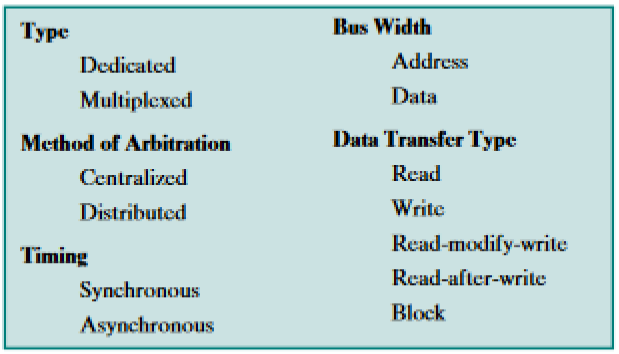


Table 3.2 Elements of Bus Design

Type	**Bus Width**
Dedicated	Address
Multiplexed	Data
Method of Arbitration	**Data Transfer Type**
Centralized	Read
Distributed	Write
Timing	Read-modify-write
Synchronous	Read-after-write
Asynchronous	Block

Bus Types:

- Dedicated: it is permanently assigned to either one function or to a physical subset of computer components.
- Multiplexed: method for using same lines for multiple purposes is called time multiplexing. It uses fewer lines which results in less cost. It has complex circuitry within each module. Events that share the same lines can't take place in parallel.

Method of Arbitration:

- Centralized Arbitration: a single hardware device, called a bus controller or arbiter, is responsible for allocating time on the bus.
- Distributed Arbitration: Each modules contains access control logic and the modules act together to share the bus.

Timing:

- Synchronous Timing: the occurrence of events on bus is determined by a clock.
- Asynchronous Timing: the occurrence of one event on a bus follows and depends on the occurrence of a previous event.

POINT-TO-POINT INTERCONNECT

Compared to the shared bus structure, the ptp interconnect has lower latency, higher data rate, and better scalability.

Intel's QuickPath Interconnect (QPI): 2008

- **Multiple Direct Connections:** multiple components within the system have direct connections to other components. No need for arbitration.
- **Layered Protocol Architecture:** QPI processor-layer interconnect use a layered protocol architecture, rather than the simple use of control signals.
- **Packetized Data Transfer:** Data are not sent as a raw bit stream, rather sent as a sequence of packets, each of which includes control header and error control codes.

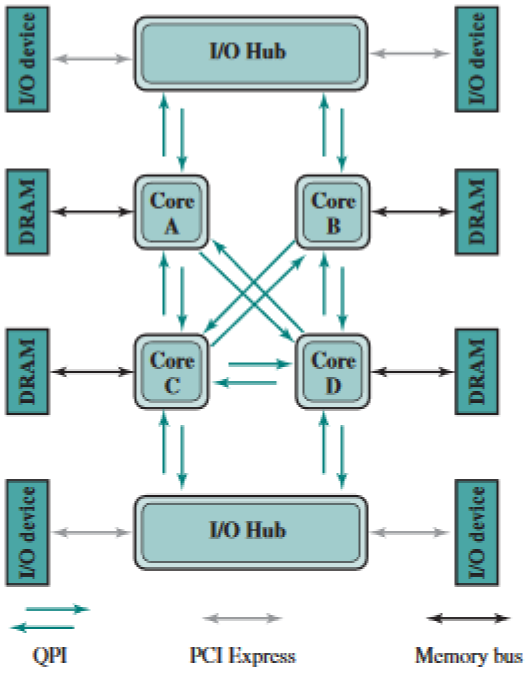


Figure 3.20 Multicore Configuration Using QPI

QPI is a four layered protocol architecture, including the following layers: (bottom to top)

- **Physical:** consists of wires carrying the signals. The unit of transfer at the Physical layer is **20** bits, which is called Phit (physical unit)
- **Link:** responsible for reliable transmission and flow control. The unit of transfer at the Link layer is 80 bit, called Flit (flow control unit)
- **Routing:** provides a framework for directing packets through the fabric.
- **Protocol:** high level set of rules for exchanging packets of data between devices.

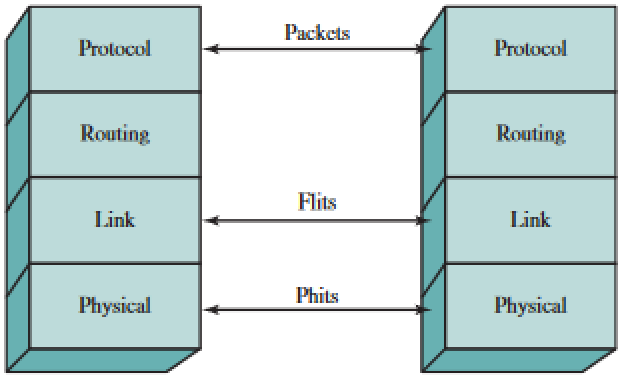


Figure 3.21 QPI Layers

QPI PORT

- 84 individual links.
- Each path consists of pair of wires that transmits bits at a time.
- Pair is called lane.
- 20 data lanes in each direction.
- 20 bits in parallel in each direction.

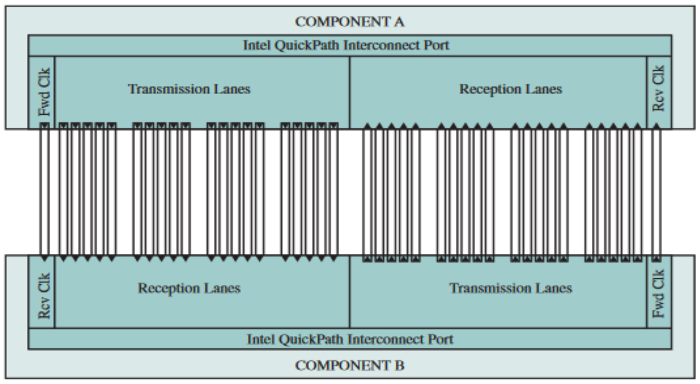


Figure 3.22 Physical Interface of the Intel QPI Interconnect

PCI EXPRESS: (2003)

Peripheral Component Interconnect is a high-bandwidth, processor-independent bus that can function as a peripheral or mezzanine bus.

PCIe, Peripheral Component Interconnect Express, as with QPI, is a ptp interconnect scheme intended to replace bus-based schemes such as PCI.

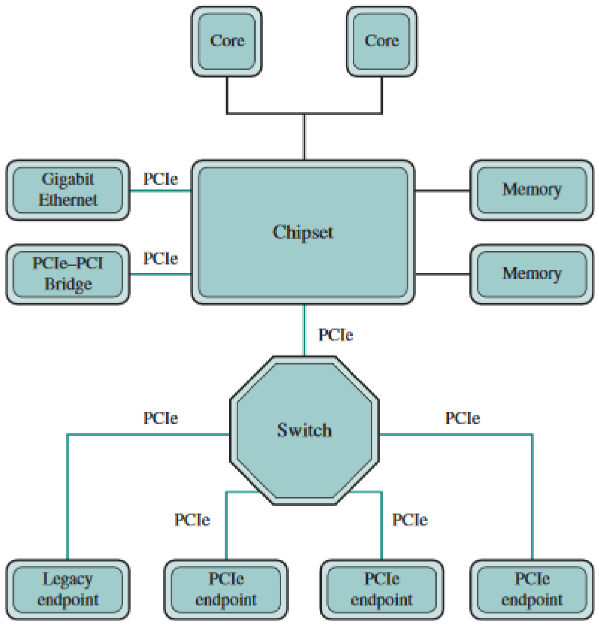
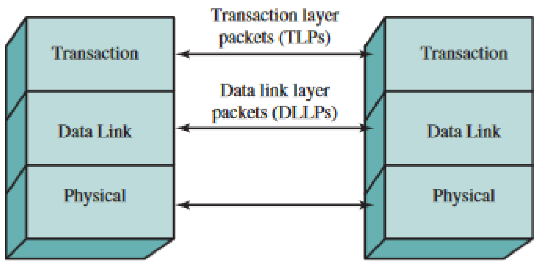


Figure 3.24 Typical Configuration Using PCIe


Transaction layer packets (TLPs)
Data link layer packets (DLLPs)
Transaction
Data Link
Physical
Transaction
Data Link
Physical
Figure 3.25 PCIe Protocol Layers

Chipset, also called host bridge and root complex device:

- Connects the processor and memory subsystems to the PCIe switch fabric comprising one or more PCIe and PCIe switch devices.
- Acts as a buffering device, to deal with the difference in data rates b/w I/O controllers, and memory and processor components.
- Translates b/w PCIe transaction formats and the processor and the memory signal and control requirements.

What types of transfers must a computer's interconnection structure (e.g., bus) support?

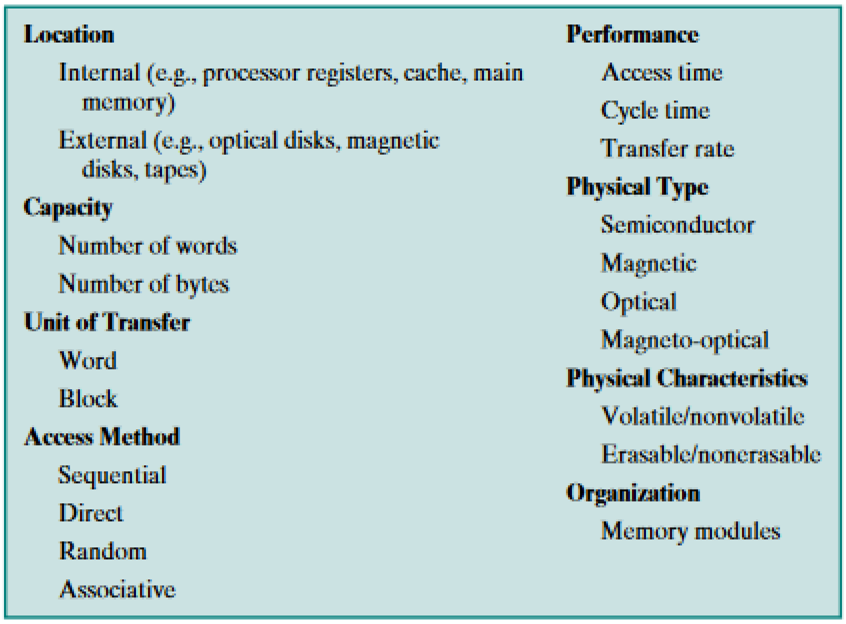
Memory to processor: The processor reads an instruction or a unit of data from memory.
Processor to memory: The processor writes a unit of data to memory.
I/O to processor: The processor reads data from an I/O device via an I/O module.
Processor to I/O: The processor sends data to the I/O device.
I/O to or from memory: For these two cases, an I/O module is allowed to exchange data directly with memory, without going through the processor, using direct memory access (DMA).

What is the benefit of using a multiple-bus architecture compared to a single-bus architecture?

With multiple buses, there are fewer devices per bus. This (1) reduces propagation delay, because each bus can be shorter, and (2) reduces bottleneck effects.

LECTURE – 05 (CHAPTER 04: CACHE MEMORY)

Table 4.1 Key Characteristics of Computer Memory Systems



Location
- Internal (e.g., processor registers, cache, main memory)
- External (e.g., optical disks, magnetic disks, tapes)

Capacity
- Number of words
- Number of bytes

Unit of Transfer
- Word
- Block

Access Method
- Sequential
- Direct
- Random
- Associative

Performance
- Access time
- Cycle time
- Transfer rate

Physical Type
- Semiconductor
- Magnetic
- Optical
- Magneto-optical

Physical Characteristics
- Volatile/nonvolatile
- Erasable/nonerasable

Organization
- Memory modules

CHARACTERISTICS OF COMPUTER MEMORY SYSTEMS

Location:

- INTERNAL: processor register, cache, and main memory.
- EXTERNAL: optical disks, magnetic disks, and tapes.

Capacity:

- Typically expressed in terms of bytes or words.

Unit of Transfer:

- For internal memory, the unit of transfer is equal to the number of electrical lines into and out of the memory module. It may be equal to word length or larger.
- Number of bits read out or written into the memory at a time.
- For external memory, unit of transfer is usually referred as block.

Access Method:

- SEQUENTIAL ACCESS: Memory is organized into units of data, called record. Access must be made in a specific linear sequence. Time to access an arbitrary record is highly variable, depending on the shared read/write mechanism used to move the records from its current location to the desire location. (tape units)

- DIRECT ACCESS: In direct access, individual blocks or records have a unique address based on physical location. Access is accomplished by direct access to reach a block plus sequential searching, counting, or waiting to reach the final location. (disk units)

- RANDOM ACCESS: Each addressable location in memory has a unique addressing mechanism. Access time is constant. Any location can be selected at random and directly addressed and accessed. (Main memory, and some cache systems)

- ASSOCIATIVE: A storage device in which location is identified by what is in it, rather than by its position is known as associative access method. It is a random access type that enable a comparison of desired information with the stored information. Each location has its own addressing mechanism and retrieval time is constant independent of location or prior access patterns. (Cache memories)

Performance:

- ACCESS TIME: (latency)
  - For RAM, the time from the instant that an address is presented to the memory to the instant that data have been stored.
  - For non-RAM, the time it takes to position the read-write mechanism at the desired location. Time required to read the record will depend on the length of the record thus not included in access time.
    - Access time depends on the following factors: (in non-RAM)
      - Location of the information required.
      - Current location of the storage system relative to the desired information.

- MEMORY CYCLE TIME: (concerned with system bus) Applied only to RAM systems, and consists of the access time plus any additional time required before a second access can commence.

  Cycle time = access time + transient time

- TRANSFER RATE: The rate at which data can be transferred into or out of the memory unit.

  For RAM = 1/cycle time

  For non-RAM

  $$T_N = T_A + n/R$$

  T_N = average time to read or write n bits.
  T_A = average access time.
  n = number of bits.
  R = Transfer rate, in bps

Physical Types:

- Semiconductor memory, magnetic surface memory, used for disk and tape, and optical and magneto-optical.
- Access time and physical characteristics depend upon physical type.

Physical Characteristics:

- Volatile memory: information is lost when electrical power is switched off.
- Nonvolatile memory: information once recorded remains without deterioration until deliberately changed.

LECTURE – 05 (CHAPTER 05: CACHE MEMORY)

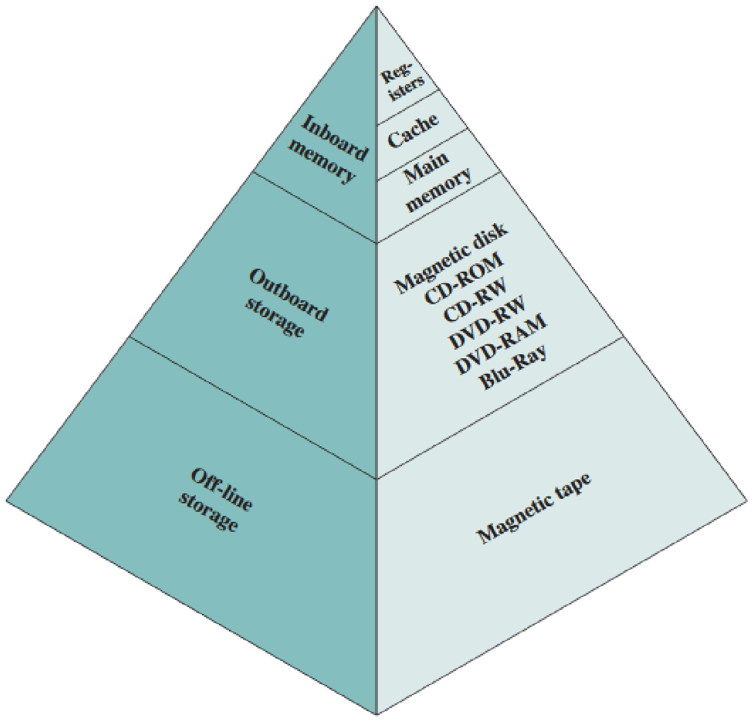


Figure 4.1 The Memory Hierarchy

The relation among the three key characteristics of computer memory is as follows:
- Faster access time, greater cost per bit.
- Greater capacity, smaller cost per bit.
- Greater capacity, slower access time.

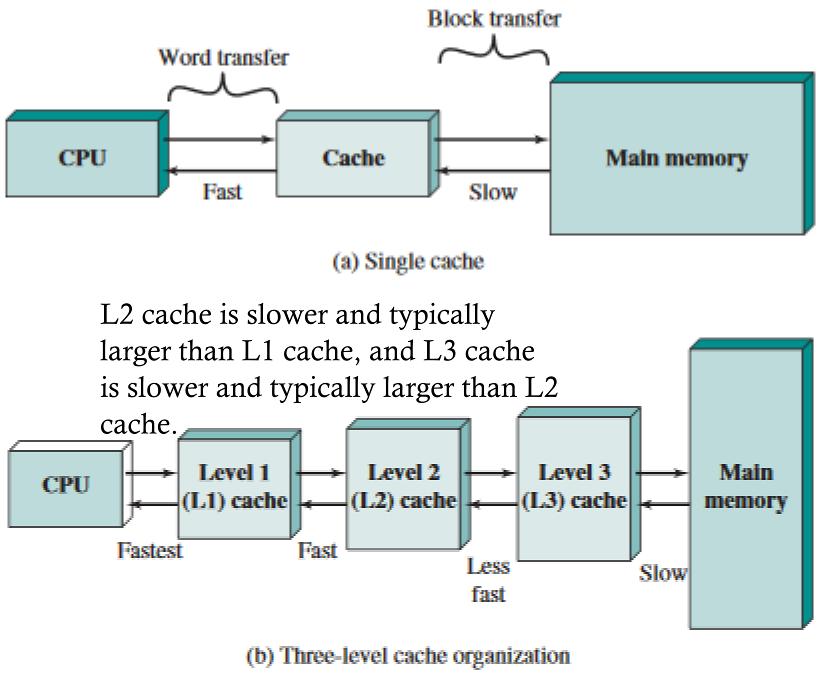
As one goes down in the hierarchy, the following occurs:
- Decreasing cost per bit
- Increasing capacity
- Increasing access time
- Decreasing frequency of access of the memory by the processor

The basis for the validity of condition (d) is a principle known as **locality of reference**.

LECTURE – 06 (CHAPTER 05: CACHE MEMORY)

CACHE MEMORY PRINCIPLES

Cache memory is designed to combine the memory access time of expensive, high-speed memory combined with large memory size of less expensive, lower-speed memory.



L2 cache is slower and typically larger than L1 cache, and L3 cache is slower and typically larger than L2 cache.

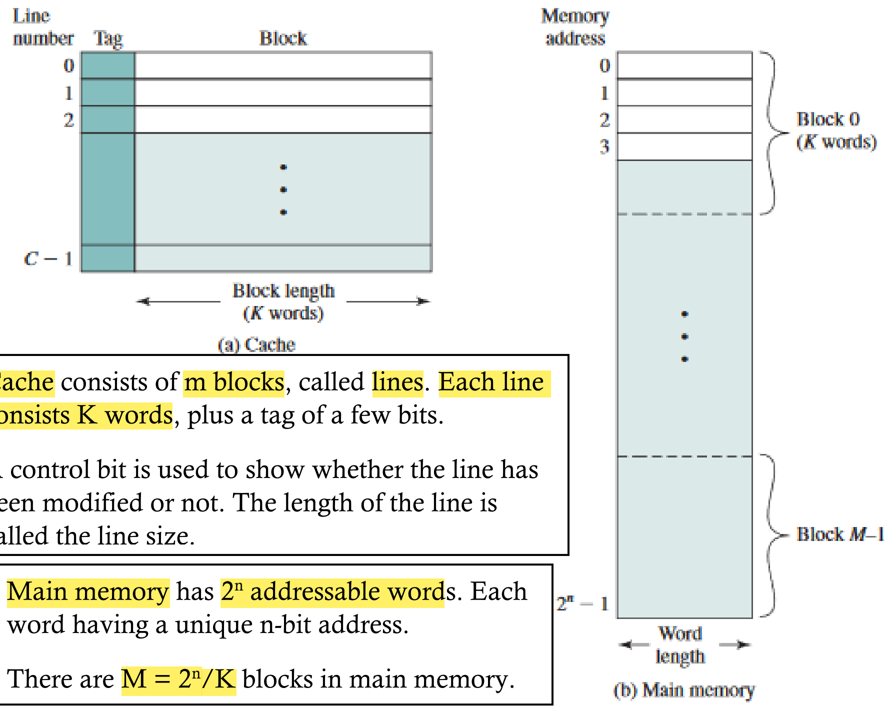
Figure 4.3 Cache and Main Memory

Clearly explain how the working of cache memory is related to some characteristics of large program.

- Study of large program reveal that most of the execution time is spent in the execution of a few routines (sub-sections). When the execution is localized within these routines, a number of instructions are executed repeatedly, this property of programs is known as LOCALITY OF REFERENCE
- Thus while some localized areas of the program are executed repeatedly, the other areas are executed less frequently.
- To reduce the execution time, these most repeated segments may be replaced with a fast memory known as Cache (Buffer) memory.
- The memory control circuitry is designed to take advantage of the property of locality of reference.
- If a word in a block of memory is read, that block is transferred to one of the slots of the cache.

Operation of Cache:

- When the processor attempts to read a word on the memory, a check is made to determine if the word is present in the cache.
- If so, the word is delivered to the processor.
- If not, a block of memory, consisting of some fixed number of words, is read into the cache.
- Then the word is delivered from the cache to the processor.
- Cache includes tags to identify which block of main memory is in each cache slot.



Cache consists of m blocks, called lines. Each line consists K words, plus a tag of a few bits.

A control bit is used to show whether the line has been modified or not. The length of the line is called the line size.

Main memory has 2^n addressable words. Each word having a unique n-bit address.

There are $M = 2^n/K$ blocks in main memory.

Figure 4.4 Cache/Main Memory Structure

Read Address = RA

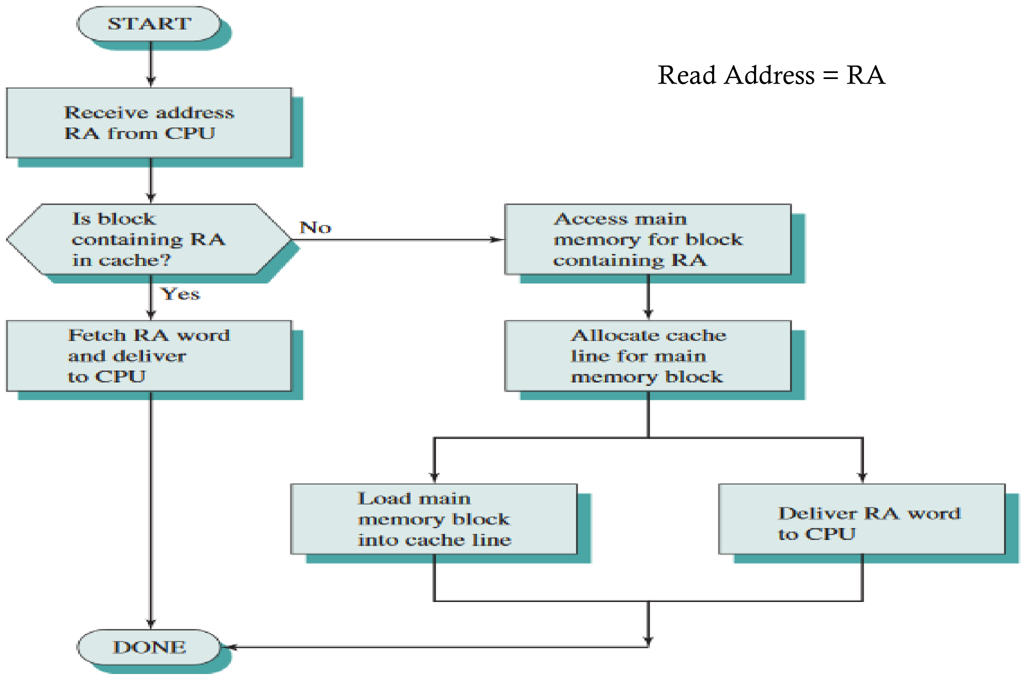

START
Receive address RA from CPU
Is block containing RA in cache?
No
Yes
Access main memory for block containing RA
Fetch RA word and deliver to CPU
Allocate cache line for main memory block
Load main memory block into cache line
Deliver RA word to CPU
DONE

Figure 4.5 Cache Read Operation

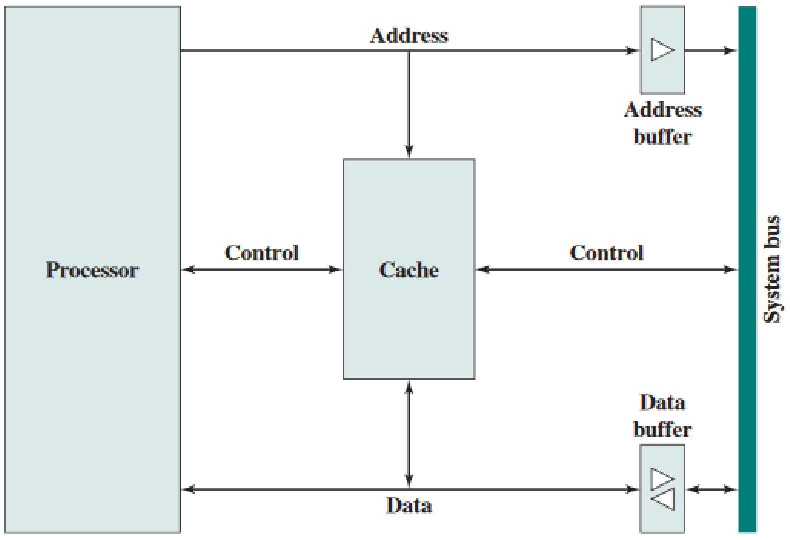


Figure 4.6 Typical Cache Organization

In the cache organization, cache connects to the processor via control, data and address lines.

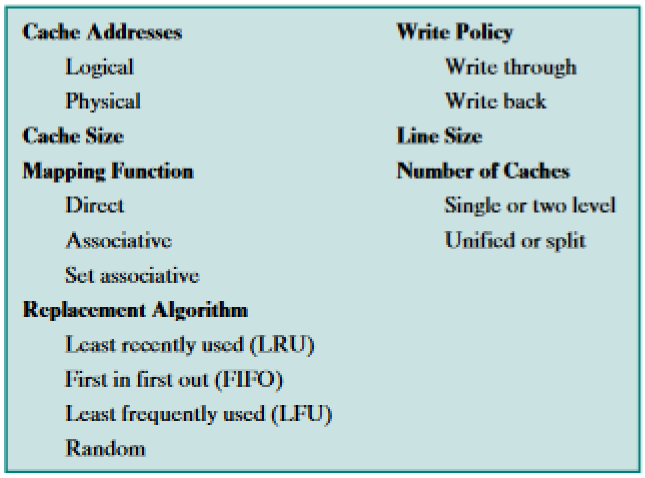
The data and address lines also attach to data and address buffers, which attach to a system bus from which main memory is reached.

During a cache hit, the data and address buffers are disabled and communication is only between cache and the processor.

During a cache miss, the desired address is loaded onto the system bus and the data are returned through the data buffer to both the cache and processor.

ELEMENTS OF CACHE DESIGN:

Table 4.2 Elements of Cache Design



Cache Addresses	**Write Policy**
Logical	Write through
Physical	Write back
Cache Size	**Line Size**
Mapping Function	**Number of Caches**
Direct	Single or two level
Associative	Unified or split
Set associative	
Replacement Algorithm	
Least recently used (LRU)	
First in first out (FIFO)	
Least frequently used (LFU)	
Random	

Cache Size:

- Size of cache should be small enough so that the average cost per bit is close to that of main memory and large enough so that the average access time is close to that of a cache alone.
- The larger the cache, the larger the number of gates involved in addressing the cache, so large cache tend to be slower than the smaller ones.
- Number of lines is considerably less than the number of main memory block, m << M

Mapping Function:

- Because there are fewer cache line than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines.
- A means is needed for determining which main memory block currently occupies a cache line.
- The choice of mapping function dictates how the cache is organized
- Three techniques can be used:

Direct: It is the simplest technique which maps each block of main memory into only one possible cache line. The mapping is expressed as

i = j modulo m

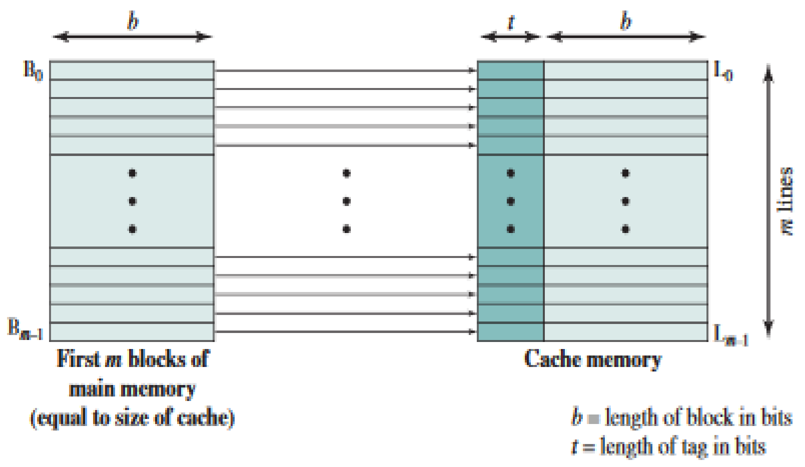
i = cache line number

j = main memory block number

m = number of line in cache

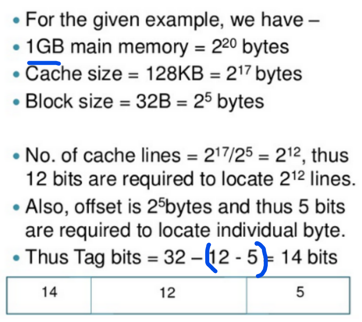
Mapping is implemented using main memory address.

Cache line	Main memory blocks assigned
0	$0, m, 2m, \ldots, 2^s - m$
1	$1, m+1, 2m+1, \ldots, 2^s - m + 1$
⋮	⋮
$m-1$	$m-1, 2m-1, 3m-1, \ldots, 2^s - 1$


b
t
b
B_0
B_{m-1}
L_0
L_{m-1}
m lines
First m blocks of main memory (equal to size of cache)
Cache memory
b = length of block in bits
t = length of tag in bits

(a) Direct mapping

✓



- For the given example, we have –
- 1GB main memory = 2^{20} bytes
- Cache size = 128KB = 2^{17} bytes
- Block size = 32B = 2^5 bytes

- No. of cache lines = $2^{17}/2^5 = 2^{12}$, thus 12 bits are required to locate 2^{12} lines.
- Also, offset is 2^5bytes and thus 5 bits are required to locate individual byte.
- Thus Tag bits = 32 – (12 - 5) = 14 bits

14	12	5

5-r

Summary:

- Address length = (s + w) bits
- Number of addressable units = $2^{(s + w)}$ words or bytes
- Block size = line size = 2^w words or bytes
- Number of block in main memory = $2^{(s + w)} / 2^w = 2^s$
- Number of lines in cache = m = 2^r
- Size of cache = $2^{(r + w)}$
- Size of tag = (s – r) bits

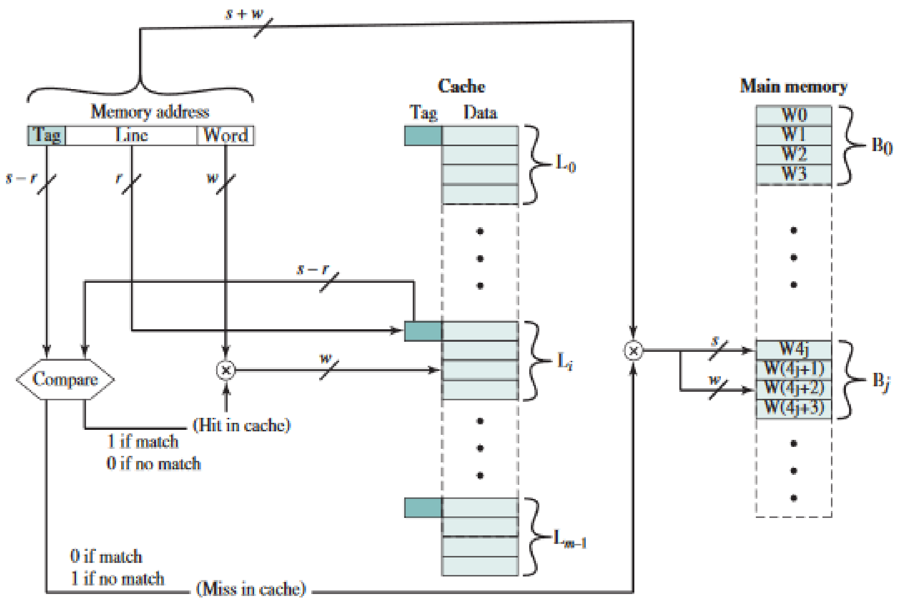
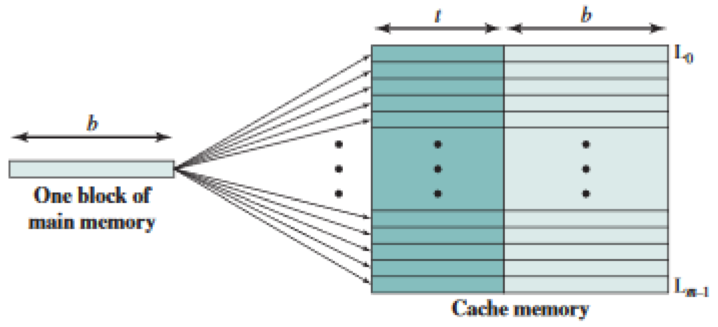


Figure 4.9 Direct-Mapping Cache Organization

- The use of a portion of the address as line number provides a unique mapping of each block of main memory into the cache.
- When a block is actually read into its assigned line, it is necessary to tag the data to distinguish it from other block that can fit in the same line. The most significant bits s-r serve this purpose.
- Direct mapping is simple and inexpensive to implement.
- DISADVANTAGE: if a program reference words repeatedly from two different blocks that map into the same line, then the block will be continuously swapped and the hit ratio will be low, this continuous swapping is called thrashing
  - Victims Cache: a small cache of 4 to 16 lines resided between a direct mapped L1 cache and the next level of memory is proposed to hold discarded (thrashed) data.

Associative:

- Each memory block can be loaded into any line of the cache.
- The cache control logic interprets a memory address simply as a tag and a word field.
- The tag field uniquely identifies a block of main memory.
- To determine whether a block is in the cache, it simultaneously examines every line's tag for a match.



(b) Associative mapping

Figure 4.8 Mapping from Main Memory to Cache: Direct and Associative

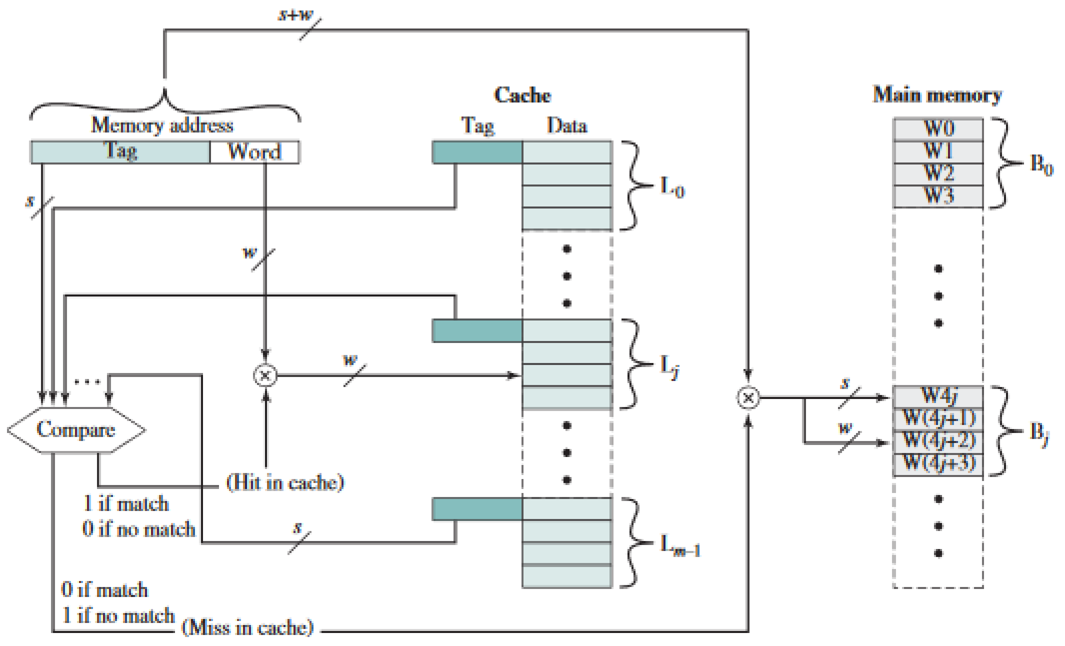
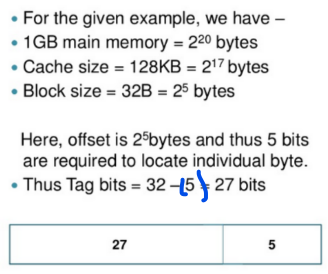


Figure 4.11 Fully Associative Cache Organization

Summary:

- Address length = (s + w) bits
- Number of addressable units = $2^{(s + w)}$ words or bytes
- Block size = line size = 2^w words or bytes
- Number of block in main memory = $2^{(s + w)} / 2^w = 2^s$
- Number of lines in cache = m = undetermined from address
- Size of tag = s bits



- For the given example, we have –
- 1GB main memory = 2^{20} bytes
- Cache size = 128KB = 2^{17} bytes
- Block size = 32B = 2^5 bytes

Here, offset is 2^5bytes and thus 5 bits are required to locate individual byte.

- Thus Tag bits = 32 – (5) = 27 bits

27	5

The disadvantage of associative method is the complex circuitry requited to examine the tag of all cache line in parallel.

Set-Associative:

- The cache consists of a number of sets, each consisting a number of lines,

$$m = v * k$$
$$i = j \text{ module } v$$

where,

i = cache set number
j = main memory block number
m = number of line in cache
v = number of sets
k = number of line in each set

This is also referred as k-way set associative mapping.

Block Bj can be mapped into any of the lines of set j.
Cache control logic interprets a memory address as three fields:

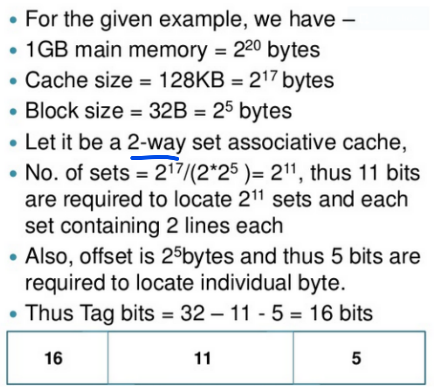
- Tag
- Set
- Word

s−r
s
s−d

Summary:

- Address length = (s + w) bits
- Number of addressable units = $2^{(s + w)}$ words or bytes
- Block size = line size = 2^w words or bytesNumber of block in main memory = $2^{(s + w)} / 2^w = 2^s$
- Number of lines in set = k
- Number of sets = v = 2^d
- Number of lines in cache = m = kv = k * 2^d
- Size of cache = k * 2^{d+w} words or bytes
- Size of tag = (s – d) bits

s bits (of the tag and set field) specify one of the 2^s blocks of main memory.
With k – way set associative mapping, the tag in memory address is smaller than the associative method and is only compared to the k tags within a single set.



- For the given example, we have –
- 1GB main memory = 2^{20} bytes
- Cache size = 128KB = 2^{17} bytes
- Block size = 32B = 2^5 bytes
- Let it be a 2-way set associative cache,
- No. of sets = $2^{17}/(2*2^5) = 2^{11}$, thus 11 bits are required to locate 2^{11} sets and each set containing 2 lines each
- Also, offset is 2^5bytes and thus 5 bits are required to locate individual byte.
- Thus Tag bits = 32 – 11 - 5 = 16 bits

16	11	5

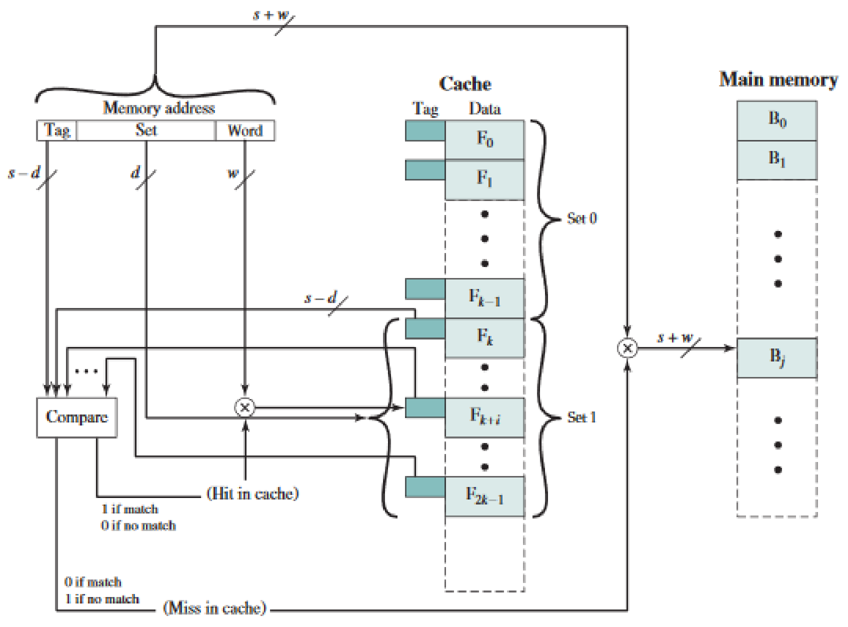


Figure 4.14 *K*-Way Set Associative Cache Organization

Replacement Algorithm:

- When Cache is full, a decision is to be made about which block should be removed to make room for the other block from the main memory. Such a decision can potentially affect the system performance. The Locality of Reference provide a clue to reasonable strategy.
- As sub-program stay for reasonable periods of time, it can be assume that a block which has recently been referenced will also be referenced in near future.
- Thus the block that has stayed for long without being referenced should be removed. Such a block is known as LEAST RECENTLY USED Block and the algorithm to determine such a block is known as LRU Algorithm.
- A counter is used to keep track of LRU Block in Cache. For a 4 slots cache 2 bit counter is used, for 8 slots cache 3 bits are required.

For a Read Operation:
In case of HIT (Content in Cache):

- Counter for the block referenced is set to '0'
- All other counters with value originally lower than referenced one are incremented by 1, while all others remain unchanged.

In case of MISS (Content not in Cache)
If Cache is Not FULL:

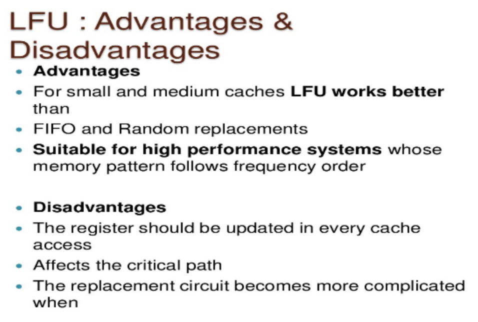
- The counter associated with the new block loaded form Main Memory is reset to '0'.
- Values of all other counters are incremented by 1.

If Cache is FULL:

- The block with the largest count is removed.
- New block is put in its place.
- Its counter is reset to '0'
- The remaining counters are incremented by 1

Other Algorithms are Least Frequently Used (LFU), First in First out (FIFO) and Random (selecting line to be replaced at random).

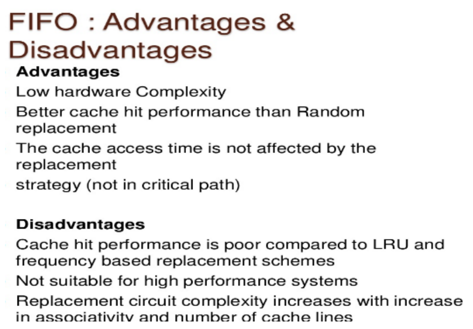
Based on LOR chances are there that the block referenced most frequently is referenced next and the block which is less frequently used may not be referenced again so it can be removed (Least Frequently Used). It is also implemented by use of a counter. This algorithm is also relatively good. While tests have shown that Random technique is best while FIFO has shown poor performance.



LFU : Advantages & Disadvantages

- **Advantages**
- For small and medium caches **LFU works better** than
- FIFO and Random replacements
- **Suitable for high performance systems** whose memory pattern follows frequency order

- **Disadvantages**
- The register should be updated in every cache access
- Affects the critical path
- The replacement circuit becomes more complicated when



FIFO : Advantages & Disadvantages

Advantages
Low hardware Complexity
Better cache hit performance than Random replacement
The cache access time is not affected by the replacement
strategy (not in critical path)

Disadvantages
Cache hit performance is poor compared to LRU and frequency based replacement schemes
Not suitable for high performance systems
Replacement circuit complexity increases with increase in associativity and number of cache lines

Write Policy:
For a write operation to a word in cache:
WRITE THROUGH POLICY:

- Main memory and cache are updated simultaneously.
- Main memory is always updated (valid)
- Disadvantage:
    - Generates substantial memory traffic
    - May create a bottleneck

WRITE BACK POLICY:

- Updates only the cache and mark the location with a flag called dirty bit or use bit.
- When the block is replaced, it is written back to the main memory if and only if the dirty bit is set.
- Disadvantage:
    - Complex circuitry
    - Creates bottleneck

In a bus organization with more than one device having their separate caches but shared memory the task of keeping memory valid becomes more complex.

A system that keep all caches and the main memory valid is said to maintain cache coherency.

Techniques for cache coherency include the following:

- Bus watching with write through: Each cache controller monitors the address lines for each controller for write operation.
- Hardware transparency: additional hardware is used to ensure all updates in main memory via caches are reflected in all caches.
- Non Cacheable memory: Only a portion of main memory is shared by more than one processor and this is designated as non-cacheable.

Line Size:

- As the block size increases from small to large, hit ratio increases because of principle of locality.
- Principle of locality: data in the vicinity of a referenced word are likely to be referenced in the near future.
- Two specific effects come into play:
    - Larger number of blocks reduce the number of blocks in the cache. Small number of block result in data being overwritten shortly after they are fetched.
    - As a block becomes larger, each additional word is farther from the requested word and therefore likely to be needed in the future.
- The relation between line size and hit ratio is complex and no optimum value has been found.
- 8 to 64 bytes of line size seems optimum in normal computers.
- In High Processing Computers, 64 to 128 bytes cache size are frequently used.

Number of Caches:
Level of caches
Unified vs split cache

MULTILEVEL CACHE:

- If there is no L2 (Off chip) cache and the processor makes an access request for a memory location not in L1 cache, then the processor must access the DRAM or ROM across the bus, resulting in poor performance.
- If an L2 (SRAM) cache is used with speed matching bus speed, then data can be accessed using a zero-wait state
- Using multilevel cache complicate all of the design issues related to cache, including size, replacement algorithm and write policy.
- Results show that L2 cache with at least double the size of L1 cache improve hit ratio.

UNIFIED VS SPLIT CACHE:

Benefits of Unified Cache:
- Higher hit rate than split cache.
- Only one cache needs to be designed and implemented.

Benefits of Split Cache:
- It is used for parallel instruction execution for prefetching of predicted future instructions.
- It eliminates contention for the cache between the instructions fetch/decode unit and the execution unit.

LECTURE – 07 (Arithmetic and Logic Unit)

- The ALU is that part of the computer that actually performs arithmetic and logical operations on data.
- ALU is based on the use of simple digital logic devices that can store binary digits and perform simple Boolean logic operations.
- Operands for arithmetic and logic operations are presented to the ALU in registers, and the results of an operation are stored in registers.
- These registers are temporary storage locations within the processor that are connected by signal paths to the ALU.

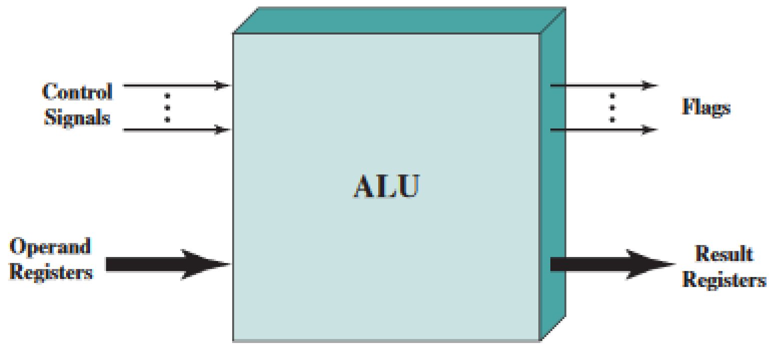


Figure 10.1 ALU Inputs and Outputs

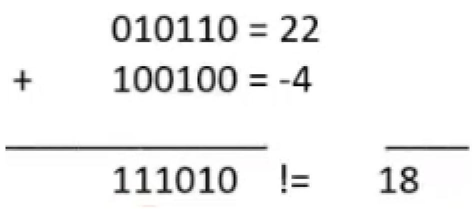
- The ALU may also set flags as the result of an operation.
- The flag values are also stored in registers within the processor.

INTEGER: (all whole numbers)
If an n-bit sequence of binary digits is interpreted as an unsigned integer A:

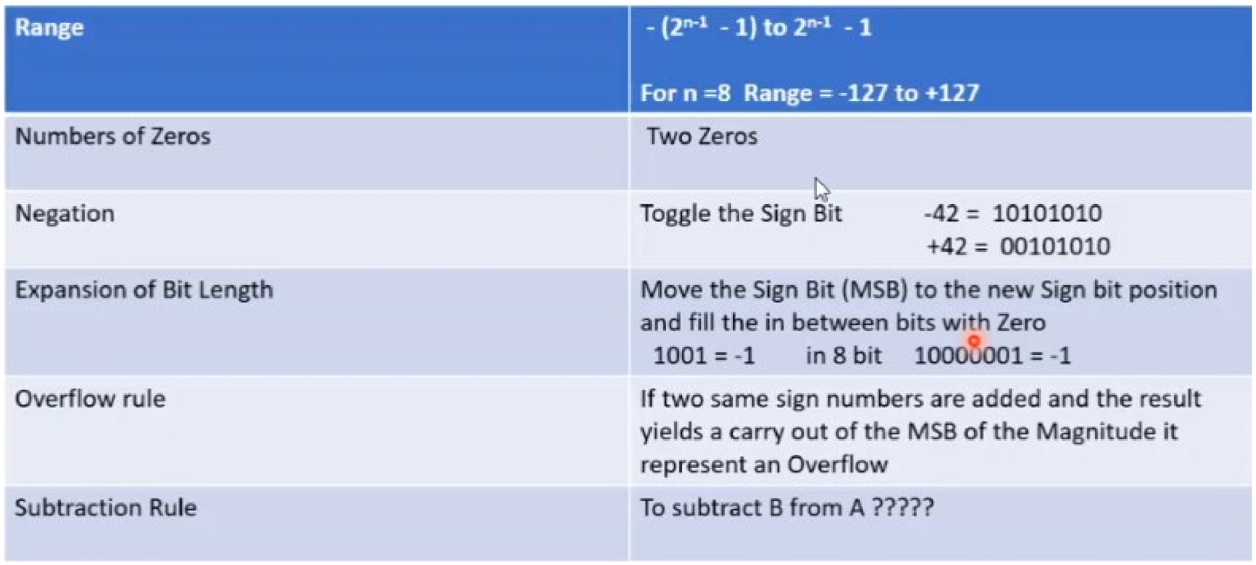
$$A = \sum_{i=0}^{n-1} 2^i a_i$$

Sign Magnitude Representation:
- Treat MSB (most significant bit), left most, in the word as sign bit.
- If the sign bit is 0, the number is positive. If the sign bit is 1, the number is negative.
- In sign magnitude the right most bit, n-1 bits, in a word of n bit holds the magnitude of the integer.

Example:
1101 = -5
00110101 = +53
Drawbacks:
- Dual representation of ‘0’
  - +0 = 00000000
  - -0 = 10000000
- Addition and subtraction require consideration of both sign and magnitude of the number.



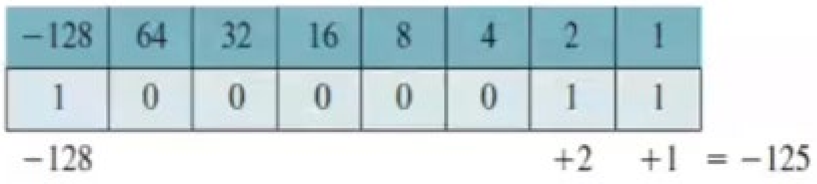
```
          010110 = 22
+         100100 = -4
____________________      _____
          111010   !=      18
```



Range	$-(2^{n-1} - 1)$ to $2^{n-1} - 1$ For n =8 Range = -127 to +127
Numbers of Zeros	Two Zeros
Negation	Toggle the Sign Bit -42 = 10101010 +42 = 00101010
Expansion of Bit Length	Move the Sign Bit (MSB) to the new Sign bit position and fill the in between bits with Zero 1001 = -1 in 8 bit 10000001 = -1
Overflow rule	If two same sign numbers are added and the result yields a carry out of the MSB of the Magnitude it represent an Overflow
Subtraction Rule	To subtract B from A ?????

Two's Complement Representation:

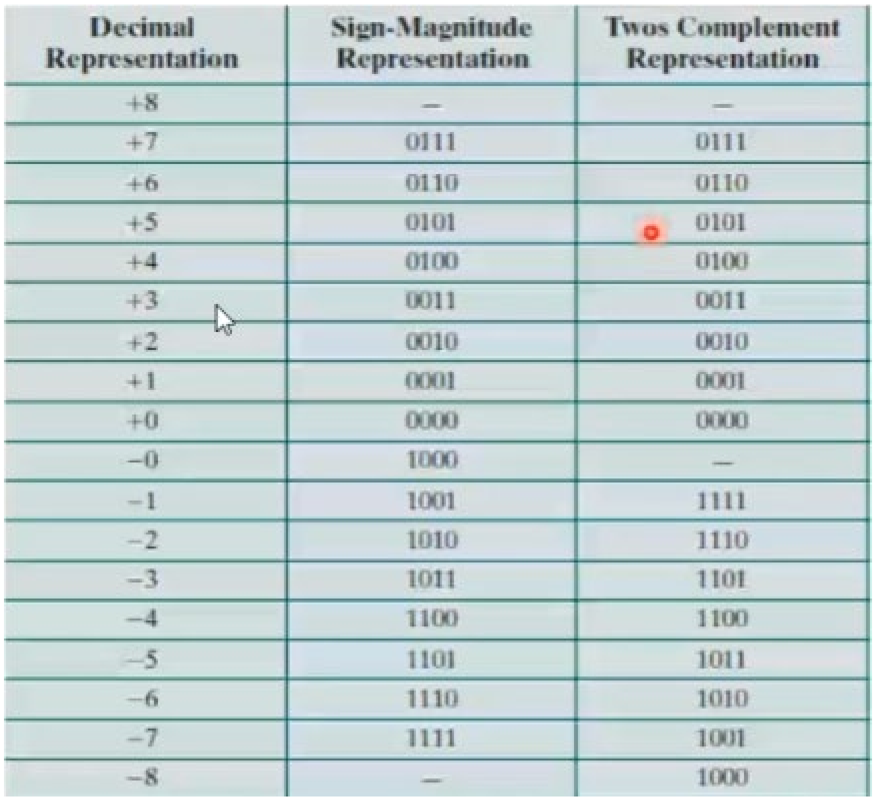
- The MSB represents the sign bit, as well as weight.



−128	64	32	16	8	4	2	1	
1	0	0	0	0	0	1	1	
−128						+2	+1	= −125

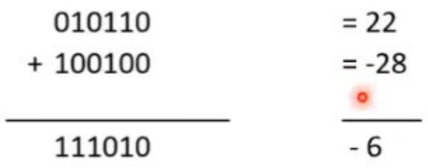
Convert binary 10000011 to decimal

To get -5, we should apply two's complement operation on the binary representation of +5.

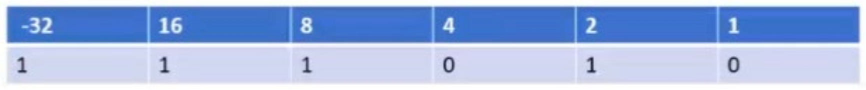


Decimal Representation	Sign-Magnitude Representation	Twos Complement Representation
+8	—	—
+7	0111	0111
+6	0110	0110
+5	0101	0101
+4	0100	0100
+3	0011	0011
+2	0010	0010
+1	0001	0001
+0	0000	0000
−0	1000	—
−1	1001	1111
−2	1010	1110
−3	1011	1101
−4	1100	1100
−5	1101	1011
−6	1110	1010
−7	1111	1001
−8	—	1000

- +8, -8 is out of range of the register, that's why it's not mentioned. The register is 4 bit and can 2^4 values stored in it.

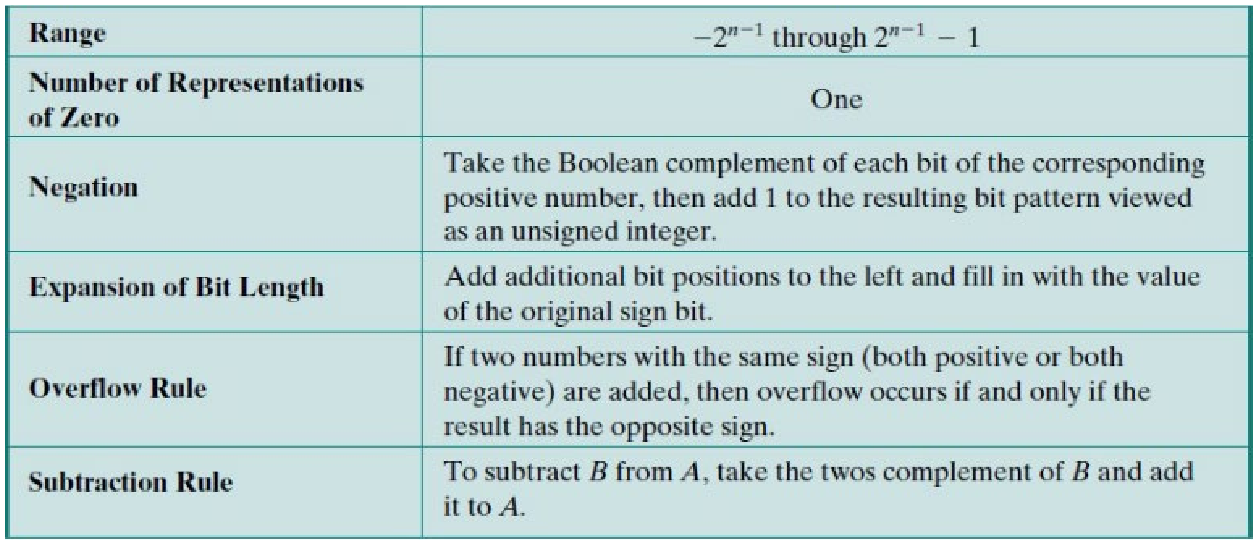


```
  010110        = 22
+ 100100        = -28
________        _____
  111010        - 6
```



-32	16	8	4	2	1
1	1	1	0	1	0

-32+16+8+2 = -6

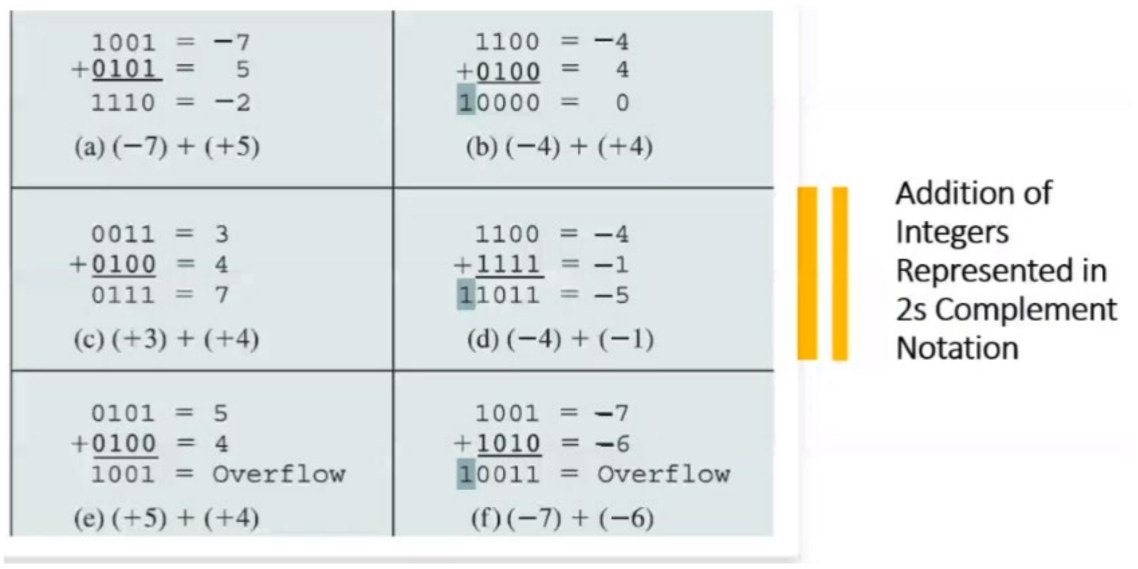


Range	-2^{n-1} through $2^{n-1} - 1$
Number of Representations of Zero	One
Negation	Take the Boolean complement of each bit of the corresponding positive number, then add 1 to the resulting bit pattern viewed as an unsigned integer.
Expansion of Bit Length	Add additional bit positions to the left and fill in with the value of the original sign bit.
Overflow Rule	If two numbers with the same sign (both positive or both negative) are added, then overflow occurs if and only if the result has the opposite sign.
Subtraction Rule	To subtract B from A, take the twos complement of B and add it to A.

Negation Rules:
- Take the complement of each bit of integer (including the sign bit)
- Treating the result an unsigned integer add 1.

Special cases:
- Negation of 0 is 0, there is a carry out of the MSB position which is ignored.
- If we take negation of a bit pattern of 1 followed by all zeroes, we get back the same result.

Addition:
- Addition proceeds as if the two numbers are unsigned integers.
- Ignore the carry if it occurs.
- If result is larger than that can be held in the word size, it is said to be overflow condition. ALU must signal this fact so that no attempt is made to use the result.
- OVERFLOW RULE: if two numbers are added, and they are either positive or negative, then overflow occurs if and only if the result has the opposite sign. Overflow can occur whether or not there is a carry.

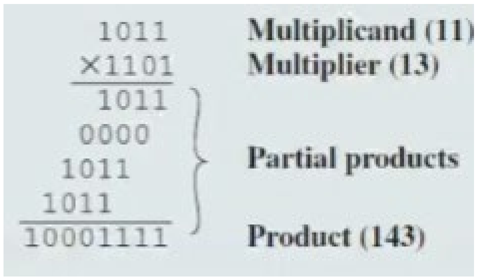


```
  1001 = -7
 +0101 =  5
  1110 = -2
(a) (−7) + (+5)

  1100 = -4
 +0100 =  4
 10000 =  0
(b) (−4) + (+4)

  0011 = 3
 +0100 = 4
  0111 = 7
(c) (+3) + (+4)

  1100 = -4
 +1111 = -1
 11011 = -5
(d) (−4) + (−1)

  0101 = 5
 +0100 = 4
  1001 = Overflow
(e) (+5) + (+4)

  1001 = -7
 +1010 = -6
 10011 = Overflow
(f) (−7) + (−6)
```

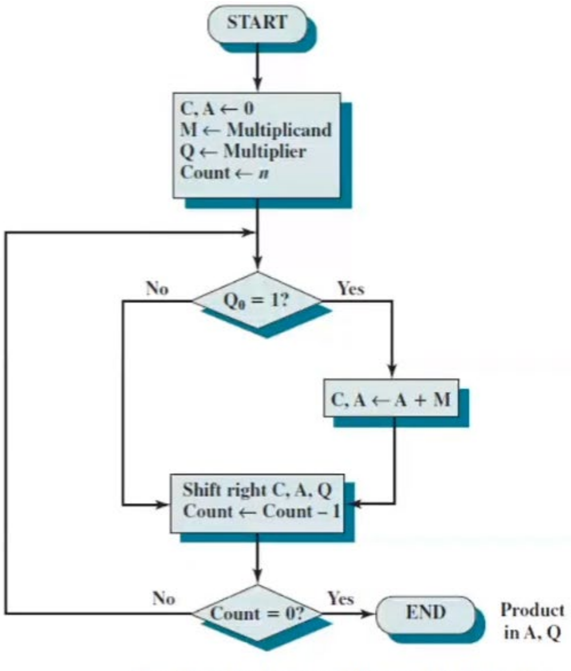
Addition of Integers Represented in 2s Complement Notation

Subtraction:
To subtract A from B, take the two's complement of B and add it in A.

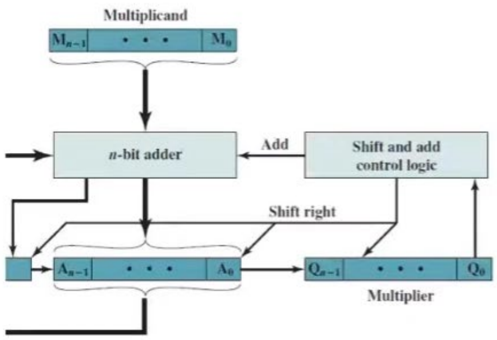
Multiplication:
Unsigned Integers:

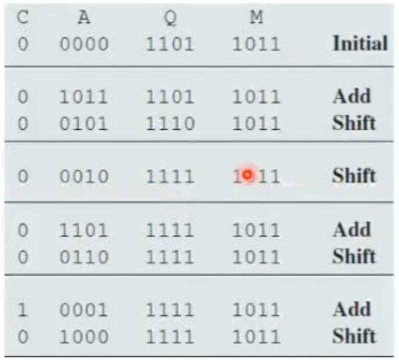


```
    1011     Multiplicand (11)
   ×1101     Multiplier (13)
    1011  )
   0000   |
  1011    }  Partial products
 1011     |
10001111  )  Product (143)
```



Flowchart for Unsigned Binary Multiplication

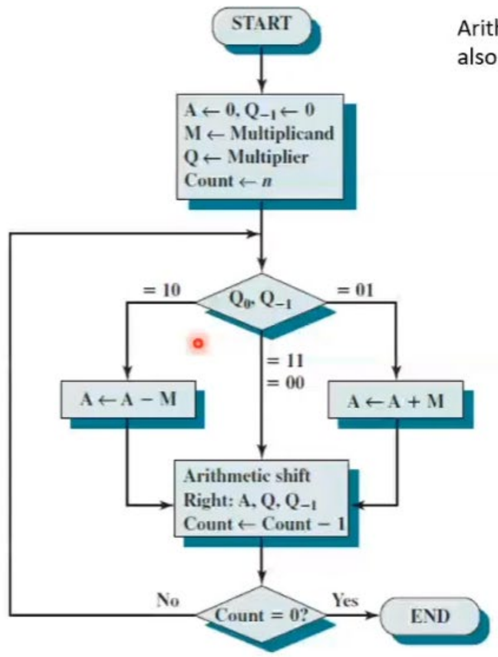




C	A	Q	M	
0	0000	1101	1011	Initial
0	1011	1101	1011	Add
0	0101	1110	1011	Shift
0	0010	1111	1011	Shift
0	1101	1111	1011	Add
0	0110	1111	1011	Shift
1	0001	1111	1011	Add
0	1000	1111	1011	Shift

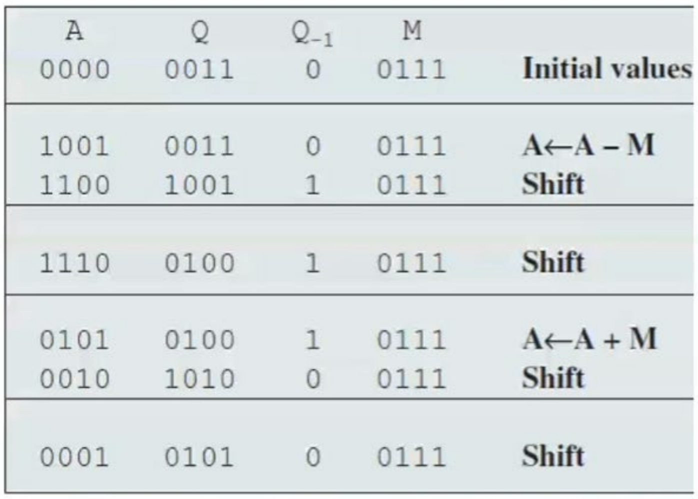
- If Q right most bit is 1
  - **A+M**
  - Shift right
- If Q right most bit is 0
  - Shift right

Two's complement Multiplication:



Booth's Algorithm for Twos Complement Multiplication

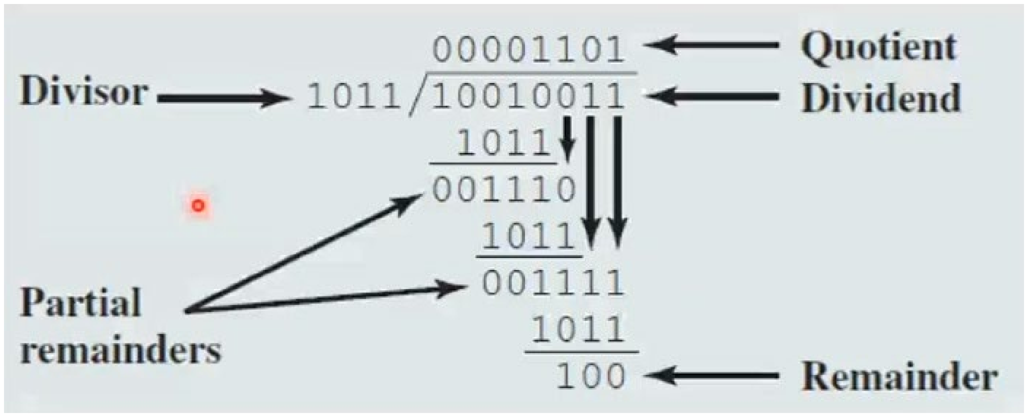
Arithmetic Shift Right preserves the sign bit by not only shifting A_{n-1} to A_{n-2} but also keeping it there.



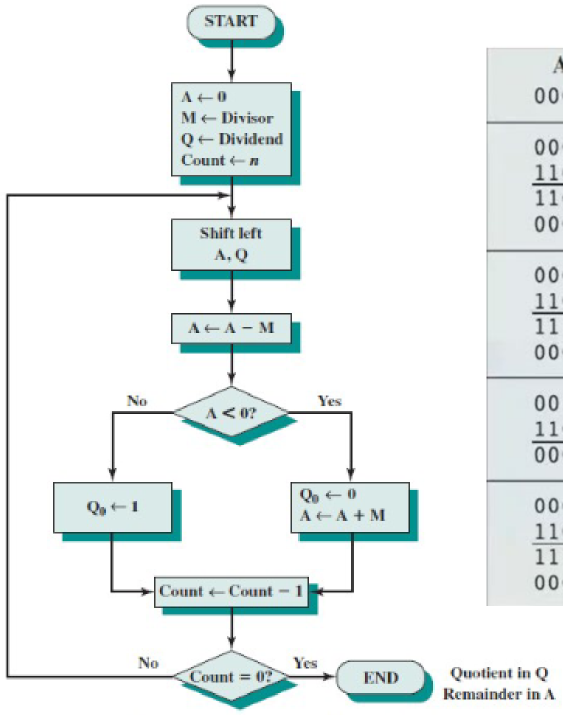
A	Q	Q_{-1}	M	
0000	0011	0	0111	Initial values
1001	0011	0	0111	A←A – M
1100	1001	1	0111	Shift
1110	0100	1	0111	Shift
0101	0100	1	0111	A←A + M
0010	1010	0	0111	Shift
0001	0101	0	0111	Shift

Example of Booth's Algorithm (7 × 3)

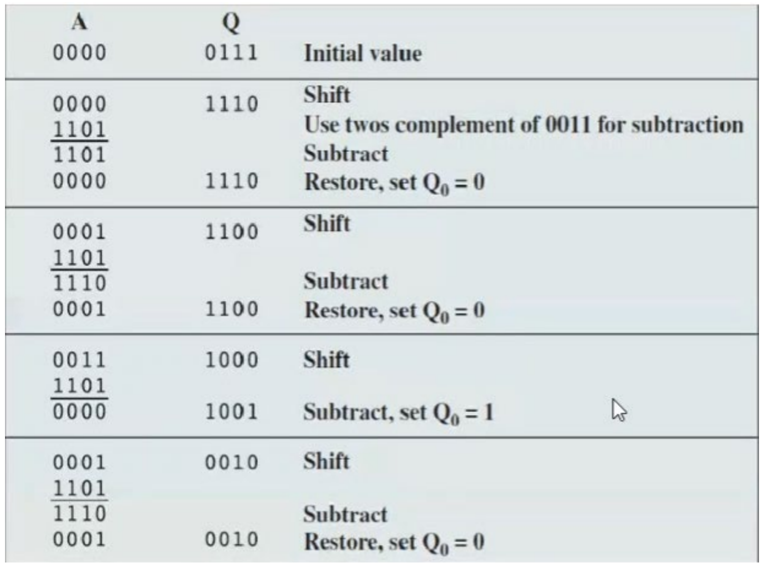
Two's Complement Division:



Restoring Division Algorithm is used for unsigned binary division



Flowchart for Unsigned Binary Division



A	Q	
0000	0111	Initial value
0000	1110	Shift
1101		Use twos complement of 0011 for subtraction
1101		Subtract
0000	1110	Restore, set $Q_0 = 0$
0001	1100	Shift
1101		
1110		Subtract
0001	1100	Restore, set $Q_0 = 0$
0011	1000	Shift
1101		
0000	1001	Subtract, set $Q_0 = 1$
0001	0010	Shift
1101		
1110		Subtract
0001	0010	Restore, set $Q_0 = 0$

- The algorithm assumes that the divisor V and the dividend D are both positive:
  - |V| < |D|
  - |V| = |D| then Q = 1 and R = 0
  - |V| > |D| then Q = 0 and R = D
- To do Twos Complement Division we need to convert the operands into unsigned values and at the end to account for the signs do complementation where needed:
  - Sign of Remainder Sign (R) = Sign (D)
  - Sign of Quotient Sign (Q) = Sign (D) x Sign (V)
- For example: -17/5: R=2, Q = 3
  - We save the negative sign and divide 17 by 5
  - The result is then changed according to the signs
  - R = 2 will be negated, two's complement of 2. 0010 → 1110
  - Also take two's complement of Q if the signs of both D and V are opposite.

FLOATING POINT NUMBERS: (all except integers)

The term floating point is derived from the fact that there is no fixed number of digits before and after the radix point; that is, the decimal point can float.

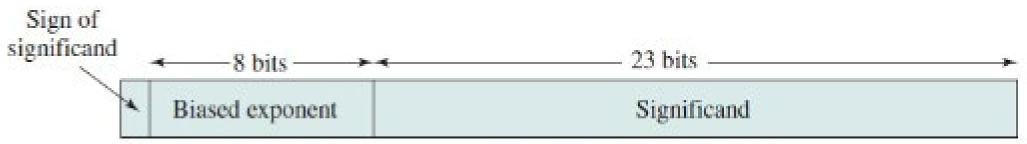
In general, floating point representations are slower and less accurate than fixed point representation, but they can handle a larger range of numbers.

- Floating point numbers can be represented in scientific notation.
- We can represent a number in the following form:

$$\pm S \times B^{\pm E}$$

Sign = plus or minus
Significand S
Exponent E

- If radix is moved to the left, add in the exponent value
- If radix is moved to the right, subtract in the exponent value.



(a) Format

$$1.1010001 \times 2^{10100} = 0\ 10010011\ 10100010000000000000000 = 1.6328125 \times 2^{20}$$
$$-1.1010001 \times 2^{10100} = 1\ 10010011\ 10100010000000000000000 = -1.6328125 \times 2^{20}$$
$$1.1010001 \times 2^{-10100} = 0\ 01101011\ 10100010000000000000000 = 1.6328125 \times 2^{-20}$$
$$-1.1010001 \times 2^{-10100} = 1\ 01101011\ 10100010000000000000000 = -1.6328125 \times 2^{-20}$$

(b) Examples

Typical 32-Bit Floating-Point Format

Base B is fixed and need not to be stored repeatedly.

The exponent is stored in biased representation.
A fixed representation, called the bias, is subtracted from the field to get the true exponent value.
Bias = $2^{(k-1)} - 1$, k is number of bits in binary exponent.

For example, if the exponent field is of 8 bits then $2^{(8-1)} - 1 = 127$. Now every value saved in this exponent field will be saved with additional 127, like 00000000 + 127 = 01111111.

If the value saved in exponent field is 00000000, then the actual value will be -127.

In 8 bit bias representation, a range of -127 to +127 can be represented.

Advantage of bias representation is that non negative floating point number can be treated as integer for comparison purpose.

Floating point numbers are stored after normalization. Normalization is moving the radix point to the right of the most significant 1.

1 is not saved as it is the same for all.

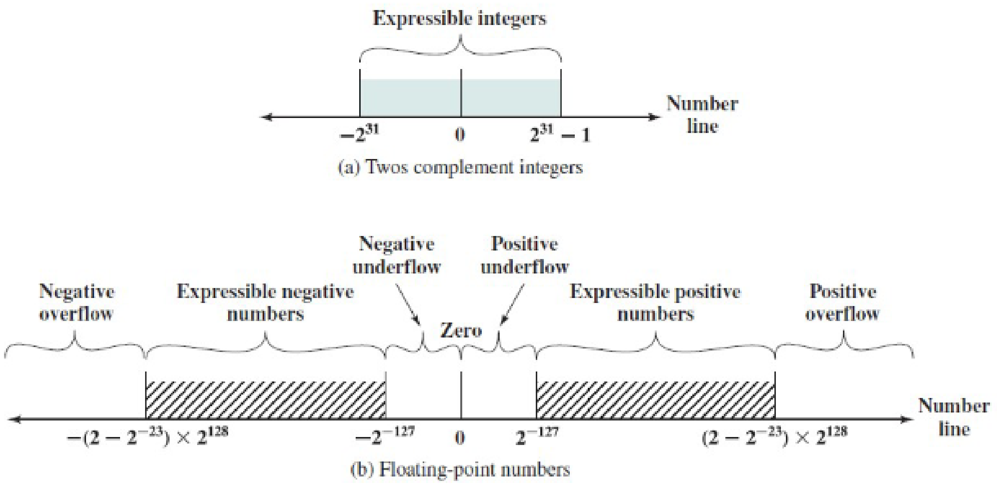
SUMMARY:

- The sign is stored in the first bit of the word.
- The first bit of the true significand is always 1 and need not to be stored in the significand field.
- The value of bias (127 in case of bit exponent) is added to the true exponent to be stored in the exponent field.
- The base is 2 and need not to be stored.

In a 32-bit word length, of 23 bit significand, 8 bit for biased exponent and 1 bit for sign, 2^{32} different numbers can be represented in the range:

Negative numbers: $-(2 - 2^{-23}) \times 2^{128}$ and -2^{-127}

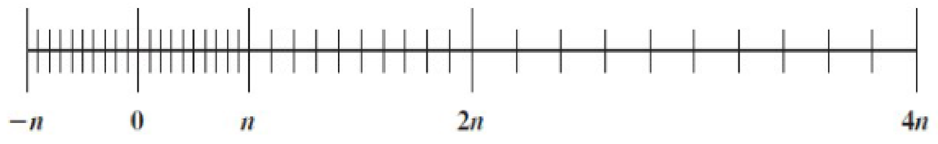
Positive Numbers = $+2^{-127}$ and $+(2 - 2^{-23}) \times 2^{128}$


Expressible integers
Number line
-2^{31}
0
$2^{31}-1$
(a) Twos complement integers
Negative overflow
Expressible negative numbers
Negative underflow
Zero
Positive underflow
Expressible positive numbers
Positive overflow
Number line
$-(2-2^{-23}) \times 2^{128}$
-2^{-127}
0
2^{-127}
$(2-2^{-23}) \times 2^{128}$
(b) Floating-point numbers

Expressible Numbers in Typical 32-Bit Formats

Floating point number have a large range because their numbers are at different distances from each other.

Range and Precision:



Density of Floating-Point Numbers

As the magnitude increases, the distance between them also increases.

- A large exponent yields a large range.
- Precision increases by increasing bits for significand.
- A large exponent base gives a greater range at the expense of less precision.

Floating Point Representation in IEEE 754 Format:

- An exponent of zero together with a fraction of zero represents positive or negative zero (depending on sign bit)

  0 0000 0000 0000 0000 0000 0000 0000 000
  1 0000 0000 0000 0000 0000 0000 0000 000

- An exponent of all ones together with fraction of zero represent positive and negative infinity.

  0 1111 1111 0000 0000 0000 0000 0000 000 = +infinity
  1 1111 1111 0000 0000 0000 0000 0000 000 = -infinity

In floating point overflow should be treated as an error or represented as infinity.

- An exponent of zero together with a fraction of non-zero fraction represent as a subnormal number. (bit to the left of radix point is 0 and true exponent is -126)

  0 0000 0000 0000 0000 0000 0000 0000 0010

- An exponent of all one together with a non-zero fraction is the value NaN (not a number), used to represent exception conditions.

Floating Point Arithmetic:
For addition and subtraction:
Both operand must have the same exponent value
Alignment is the process of making the exponent values same.

For Multiplication and Division:
In multiplication, exponent values of both operands is added and simple multiplication is performed on the significand part.

In division, exponent values of both operands is subtracted and simple division is performed on the significand part.

Let $X = X_S \times B^{X_E}$ & $Y = Y_S \times B^{Y_E}$

$$X \times Y = (X_S \times Y_S) \times B^{X_E + Y_E - 127}$$
$$X / Y = (X_S / Y_S) \times B^{X_E - Y_E + 127}$$
$$X + Y = (X_S \times B^{X_E - Y_E} + Y_S) \times B^{Y_E}$$
$$X - Y = (X_S \times B^{X_E - Y_E} - Y_S) \times B^{Y_E}$$

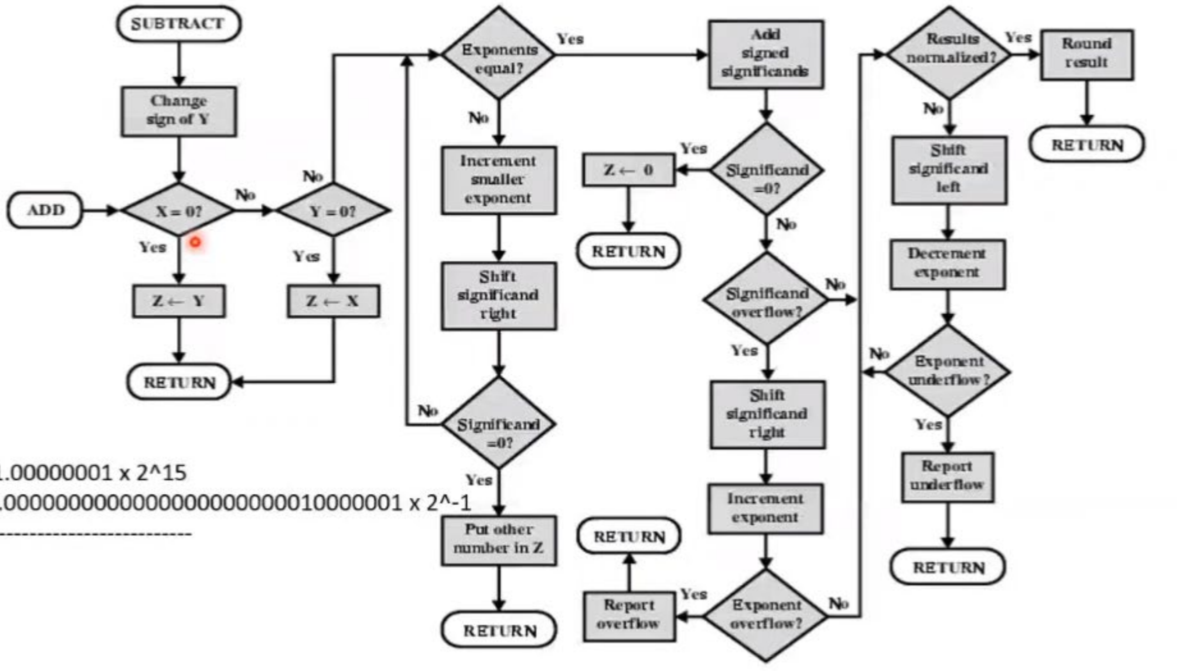
True value X = 1.000001 x 2 ^20
True Y= 1.1010101 x 2 ^20
Biased X = 1.000001 x 2 ^20+127
Biased Y= 1.1010101 x 2 ^20+127
Baised Result X x Y = (..........) x 2 ^ 147+147
Biased Result = (...........) x 2 ^ 294
True Result = (..........) x 2 ^ 167

This is double biased, that's why 127 is subtracted from it in multiplication and 127 will be added in division.

BSCS- 511 Computer Organization And Architecture

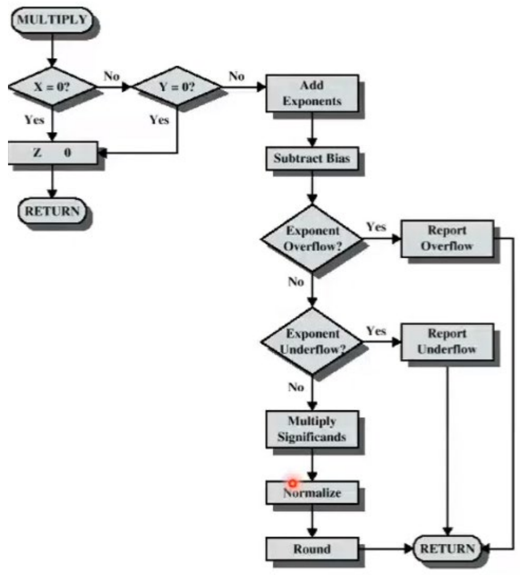
Four basic phases of the algorithm for addition and subtraction:

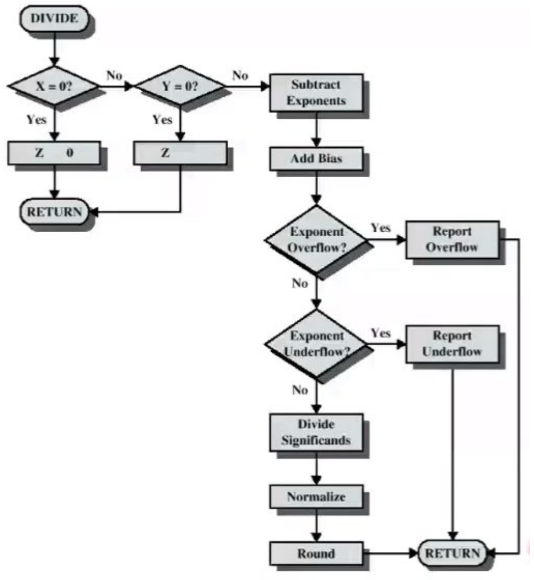
1. Check for zeroes.
2. Align the significands (small power is made equivalent to the bigger power by moving the radix point)
3. Add or subtract the significands.
4. Normalize the result.



X= 1.00000001 x 2^15
Y=0.0000000000000000000010000001 x 2^-1

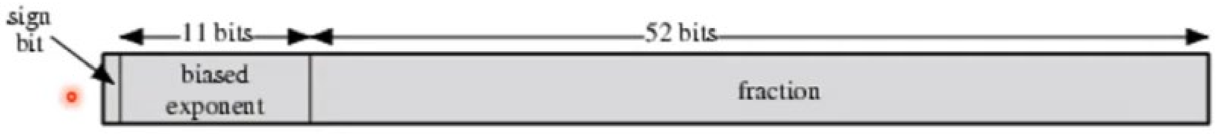
GUARD BITS: Registers in ALU contain additional bits for holding implied bit and is used to pad out the right end of the significand with 0s.





Special conditions in floating point arithmetic:

- Exponent overflow → positive or negative infinity. (greater or equal than 128)
- Exponent underflow → 0 (subnormal)
- Significand underflow → in the process of aligning the significand digits may flow off the right end of the significand. (significand part becomes 0, then we use the guard bits)
- Significand overflow → addition of 2 same sign significand may result in a carry out of MSB. This can be fixed by realignment.



(b) Double format

64 bit pattern

LECTURE – 08 (Instruction Set Handout)

MACHINE INSTRUCTION CHARACTERISTICS
The operation of the processor is determined by the instructions it execute, referred to as machine instructions or computer instructions. The collection of different instruction executed by a processor is referred to as processor's instruction set.

Elements of Machine Instruction:

Each instruction must contain information required by the processor for execution.

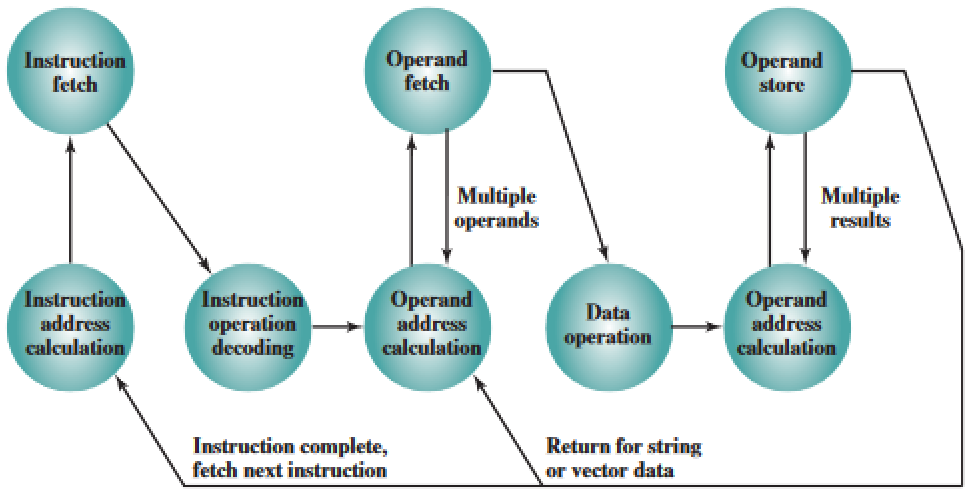

Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data operation
Operand address calculation
Instruction complete, fetch next instruction
Return for string or vector data

Figure 12.1 Instruction Cycle State Diagram

The elements are as follows:
Operation Code: it specifies the operation to be performed. The operation is specified by a binary code, called the operation code or opcode.
Source Operand Reference: the operation may involve one or more source operands, that is, operands that are inputs for the operation.
Result Operand Reference: the operation may produce a result.
Next Instruction Reference: this tells the processor where to fetch the next instruction after the execution of this instruction is complete.

Source or result operand reference can be in the following areas:

- Main or Virtual memory.
- CPU Registers
- I/O Devices
- Immediate

From Designer's POV:
Machine instruction set provide the fundamental requirement of the processor.
Implementing the processor is implementing the machine instruction set.

From Program's POV:
Become aware of registers and memory structure, types of data directly supported by machine, and functioning of ALU.

In most cases, the next instruction immediately follows the current instruction, if a program counter is used, there is no need for explicit reference to the next instruction.

Instruction Representation:
Instruction is represented by a sequence of bits.
It is divided in fields, corresponding to the elements of instruction.

OPCODE	Source Operand References 1, 2	Result Operand References 3	Next Instruction References 4

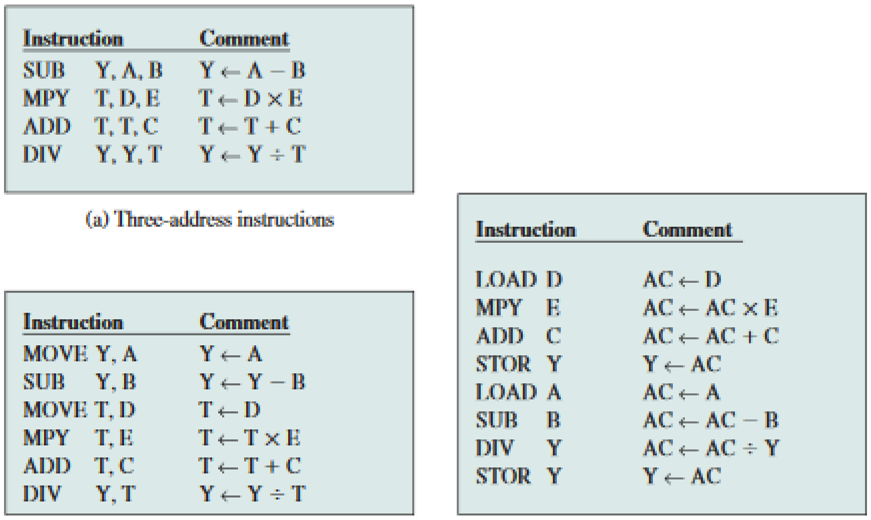
- Most instruction sets use more than one format.
- Processor extracts data from various instruction field once it is in the Instruction Register.
- In common practice, instructions are represented by symbolic representation called mnemonics. Like, ADD, SUB, MUL, DIV, STOR, LOAD, etc.
- A machine language instruction express operations in a basic form involving movement of data to or from registers. Like, ADD x, y.

Instruction Types:
Machine instruction set must be efficient to express any of the instructions from a high level language.

- Data Processing: Arithmetic and logic instructions.
- Data Storage: Movement of data into or out of the registers or the memory locations.
- Data Movement: I/O instructions.
- Control: Test and Branch instructions

- Arithmetic instructions provide computational capabilities for processing numeric data.
- Logic instructions operate on the bits of a word as bits rather than as numbers, it processes any kind of data.
- Memory instruction are used for moving data between the memory and the registers.
- I/O instructions are needed to transfer programs and data into the memory and the results of computation back out to the user.
- Test instructions are used to test the value of a data word or the status of a computation.
- Branch instructions are used to branch a different set of instructions depending on the decision made.

Number of Addresses:
- The processor architecture is typically described in terms of number of addresses contained in each instruction. It is not common today.
- Present day processor don't need all of the four addresses, explicitly.
- One, two or three address instruction may be used, the address of the next instruction is provided implicitly by the program counter.
- Most CPU designs involve a variety of instruction format.



Instruction		Comment
SUB	Y, A, B	$Y \leftarrow A - B$
MPY	T, D, E	$T \leftarrow D \times E$
ADD	T, T, C	$T \leftarrow T + C$
DIV	Y, Y, T	$Y \leftarrow Y \div T$

(a) Three-address instructions

Instruction		Comment
MOVE	Y, A	$Y \leftarrow A$
SUB	Y, B	$Y \leftarrow Y - B$
MOVE	T, D	$T \leftarrow D$
MPY	T, E	$T \leftarrow T \times E$
ADD	T, C	$T \leftarrow T + C$
DIV	Y, T	$Y \leftarrow Y \div T$

(b) Two-address instructions

Instruction		Comment
LOAD	D	$AC \leftarrow D$
MPY	E	$AC \leftarrow AC \times E$
ADD	C	$AC \leftarrow AC + C$
STOR	Y	$Y \leftarrow AC$
LOAD	A	$AC \leftarrow A$
SUB	B	$AC \leftarrow AC - B$
DIV	Y	$AC \leftarrow AC \div Y$
STOR	Y	$Y \leftarrow AC$

(c) One-address instructions

Figure 12.3 Programs to Execute $Y = \dfrac{A - B}{C + (D \times E)}$

What does less addresses mean?

- More basic instructions and less complex design of the CPU.
- Instruction word is shorter.
- More instructions mean longer program and longer execution time.
- Program design is also complex.

Most computers employ a mixture of instruction formats in terms of number of explicit addresses.
Number of addresses per instruction is a basic design decision.
Zero-address instructions are applicable to a special memory organization called a stack.

Instruction Set Design:
The most analyzed and most interesting aspect of the computer design is the instruction set design. The design of the instruction set is very complex since it affects many aspects of the computer system.

The instruction set defines many of the functions performed by the processor and has a significant effect on the implementation of the processor. The instruction set is the programmer's means of controlling the processor. Therefore, programmer's requirement must be considered in designing the instruction set.

The most fundamental design issues that still remain in dispute are:
OPERATION REPERTOIRE: how many and which operation to provide and how complex should each operation be.
DATA TYPES: the various types of data on which operations are performed.
INSTRUCTION FORMAT: instruction length, number of addresses, size of various fields and so on.
REGISTERS: number of instruction registers that can be referred by the instruction and their use.
ADDRESSING: the mode or modes by which the address of an operand is specified.

The instruction format is part of the instruction set design and is also complex as it includes:

- Specification of total number of bits (instruction length)
- Number of addresses in the instruction.
- Specification of size of various fields in the instruction word.

Types of Operand:
Machine instructions operate on data. The most general categories of data are:

- Address (they are treated as data → unsigned integers)
- Numbers (limited in terms of magnitude and precision)
- Characters
- Logical Data

NUMBERS:
Three types of numerical data is common in computers

- Binary integer or binary fixed point
- Binary floating point.
- Decimal

CHARACTERS:
The most commonly used character code is International Reference Alphabet (IRA), referred to as American Standard Code for Information Interchange (ASCII). Each code is a unique 7 bit address.

LOGICAL DATA:
Each word or other addressable unit is treated as a single unit of data.

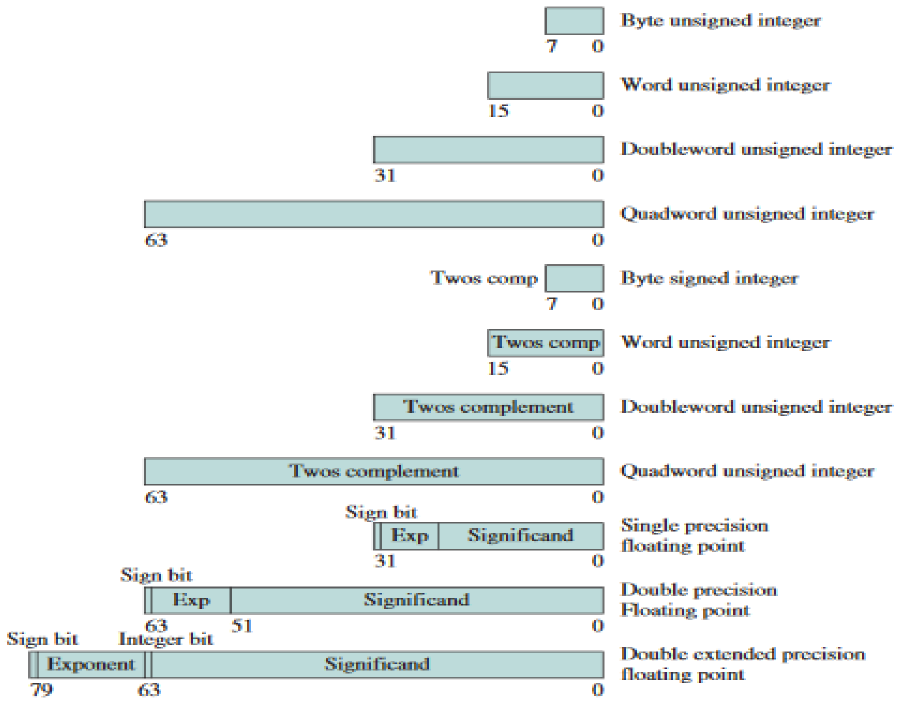


Figure 12.4 x86 Numeric Data Formats

Arm data Types:

- ARM processors support data types of 8 (bytes), 16 (halfword), and 32 (words) bits in length.
- Halfword access should be halfword aligned and word accesses should be word aligned.
- For nonaligned access attempt, the architecture support three alternatives:
    - Default case:
        - Treated as truncated.
        - Bit [1:0] treated as zero for word.
        - Bit [0] treated as zero for halfword.
        - Load single word instructions rotate right word aligned data transferred by non word-aligned address one, two or three bytes.
    - Alignment checking: when the appropriate control bit is set, a data abort signal indicates an alignment fault for attempting unaligned access.
    - Unaligned access: the processor uses one or more memory accesses to generate the required transfer of adjacent bytes transparently to the programmer.
- Unsigned integer interpretation is supported for all types.
- All three data types can also be used for twos complement signed integer.
- Majority of ARM processor implementation do not provide floating point hardware, which saves power and area.
- A floating point arithmetic is implemented in software, if required.
- ARM supports an optional floating point coprocessor that supports single and double precision floating point data types defined in IEEE 754.

Types of Operations:
Number of different opcodes varies from machine to machine.
General type of operations found on all machines are:
- DATA TRANSFER: move, store, load, exchange, clear, set, push, pop.
- ARITHMETIC: add, subtract, multiply, divide, absolute, negate, increment, decrement.
- LOGICAL: AND, OR, NOT, Exclusive-OR, test, compare, shift, rotate, set control variables.
- CONVERSION: translate, convert (e.g. binary to decimal) .
- INPUT/OUTPUT: input, output, test I/O, start I/O
- SYSTEM CONTROL
- TRANFER OF CONTROL: jump, jump conditional, jump to subroutine, return, execute, skip, skip conditional, halt, wait, no operation.

DATA TRANSFER:
A data transfer operation must specify:
- Location of source and destination operands
- Length of data to be transferred.
- Addressing mode for each operand.

Data transfer operation occurs in the following steps:
- Calculate memory address, based on the address mode.
- Determine whether the addressed item is in the cache.
- If not, issue a command to the memory module.

ARITHMETIC:
- Provided for:
    - Signed integer numbers.
    - Floating point
    - Packed decimal numbers.
- Single operand instructions:
    - Absolute: |A|
    - Negate: -A
    - Increment: A++ (add 1 to the operand)
    - Decrement: A-- (subtract one from the operand)

LOGICAL:

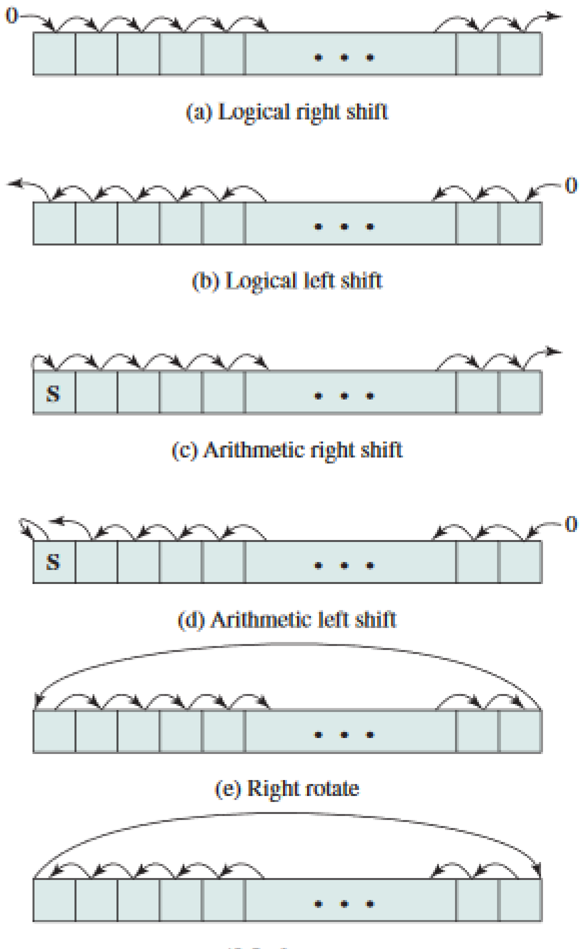


Figure 12.6 Shift and Rotate Operations

Table 12.7 Examples of Shift and Rotate Operations

Input	Operation	Result
10100110	Logical right shift (3 bits)	00010100
10100110	Logical left shift (3 bits)	00110000
10100110	Arithmetic right shift (3 bits)	11110100
10100110	Arithmetic left shift (3 bits)	10110000
10100110	Right rotate (3 bits)	11010100
10100110	Left rotate (3 bits)	00110101

INPUT/OUTPUT:

- There are a variety of approaches taken to execute I/O operations, including:
  - Isolated programmed I/O
  - Memory-mapped programmed I/O (using data movement instructions)
  - DMA (separate controller)
  - Use of an I/O processor.

SYSTEM CONTROL:

- Only execute when the processor is in privileged state (kernel mode).
- Instructions for the use of OS.
- Example
  - A system control instruction may read or alter a control register.
  - To read or modify a storage protection key.

TRANSFER OF CONTROL:

- Branch: (Jump)
    - Conditional: branch to x if result is positive.
    - Unconditional

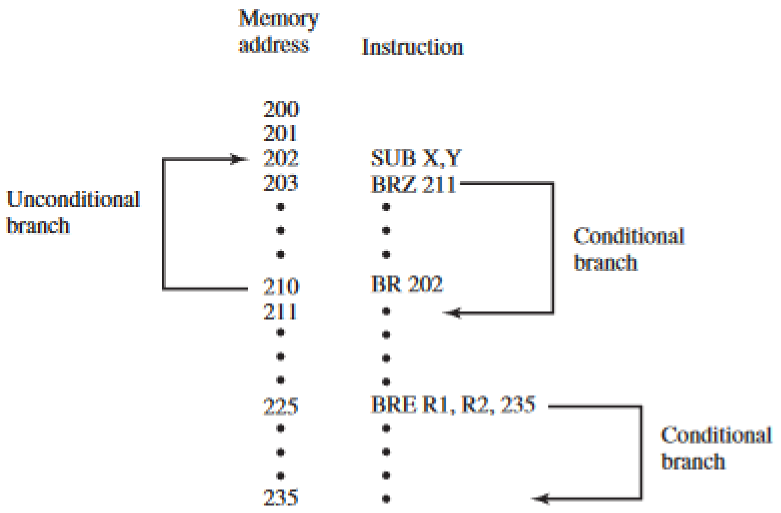


Figure 12.7 Branch Instructions

- Skip:
    - Increment and skip if 0
- Procedure Call: (Subroutine Call)
    - A procedure is a self-contained computer program that is incorporated into a larger program.

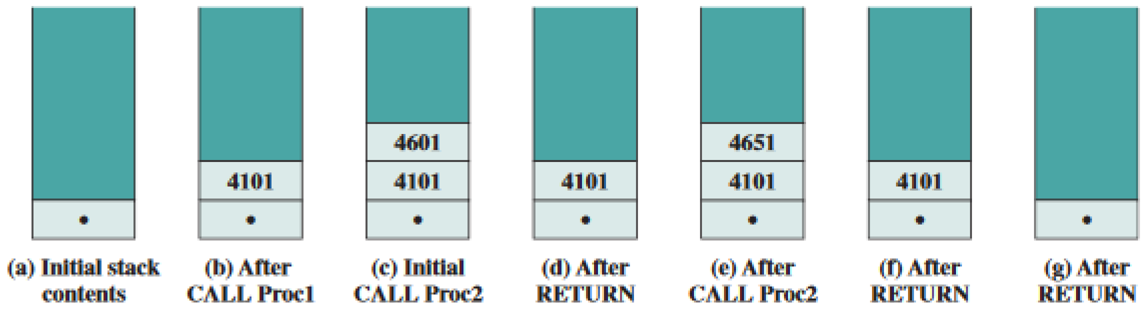


Figure 12.9 Use of Stack to Implement Nested Subroutines of Figure 12.8

ADDRESSING MODES:
Most common addressing modes:
Immediate: the operand value is present in the instruction.

Operand = A

Direct: the address field contains the effective address of the operand

EA = A

Indirect: the address field refer to the address of a word in memory, which in turn contains the full-length address of the operand.

EA = (A)

Register: the address field refers to a register, in which the operand value is contained.

EA = R

Register Indirect: the address field refers to the address of a register, which in turn contains the full length address of the operand.

EA = (R)

Displacement: combines the capabilities of direct and register indirect addressing.

EA = A + (R)

Stack: a form of implied addressing. Machine instructions need not include a memory reference but implicitly operate on the top of the stack.

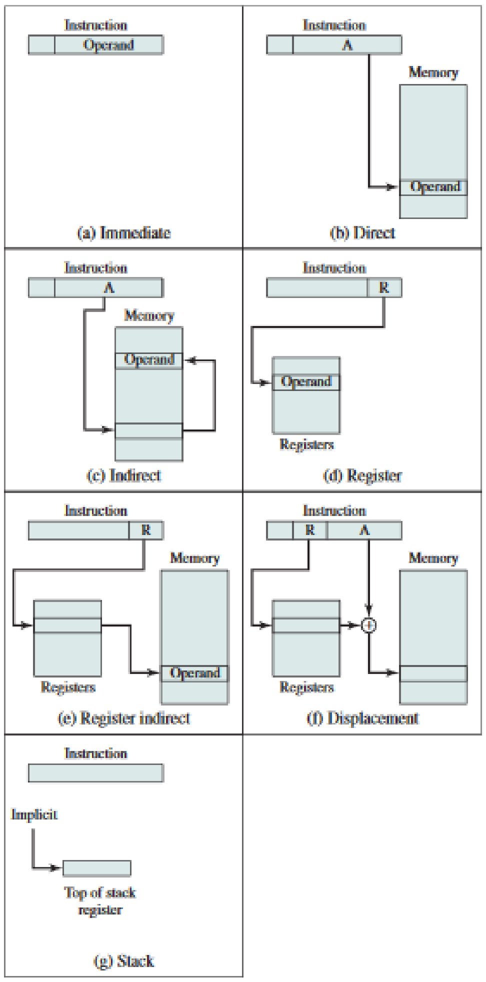


Figure 13.1 Addressing Modes

Table 13.1 Basic Addressing Modes

Mode	Algorithm	Principal Advantage	Principal Disadvantage
Immediate	Operand = A	No memory reference	Limited operand magnitude
Direct	EA = A	Simple	Limited address space
Indirect	EA = (A)	Large address space	Multiple memory references
Register	EA = R	No memory reference	Limited address space
Register indirect	EA = (R)	Large address space	Extra memory reference
Displacement	EA = A + (R)	Flexibility	Complexity
Stack	EA = top of stack	No memory reference	Limited applicability

LECTURE – 09 (INSTRUCTION CYCLE)

The processor must be capable of doing the following things:

- **Fetch Instruction:** reads instruction from the memory.
- **Interpret Instruction:** instruction is decoded to determine which action is required.
- **Fetch Data:** execution of instruction may require reading of data from memory or an I/O module.
- **Process Data:** execution of instruction may require performing some arithmetic or logical operation on data.
- **Write Data:** the results of an execution may require writing data to memory or I/O module.

System performance depends upon:

- Smaller clock cycle, better system performance.
- The lesser the memory access the, the better the performance.

INSTRUCTION CYCLE:

The instruction cycle includes the following stages:

- **Fetch:** read the instruction from the memory into the processor.
- **Execute:** interpret the opcode and perform the indicated operation.
- **Interrupt:** if interrupts are enabled and an interrupt has occurred, save the current process state and perform the indicated operation.

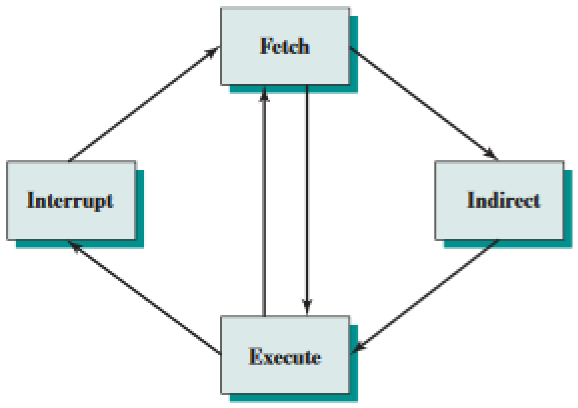


Figure 14.4 The Instruction Cycle

The Indirect Cycle: This is an additional stage. The execution of an instruction may involve one or more operands in memory, each of which requires a memory access.
If indirect addressing is used, then additional memory accesses are required. We can think of fetching of indirect addresses as one more instruction stage.
The main line of activity consists of alternating instruction fetch and instruction execution activities. After an instruction is fetched, it is examined to determine if any indirect addressing is involved. If so, the required operands are fetched using indirect addressing.

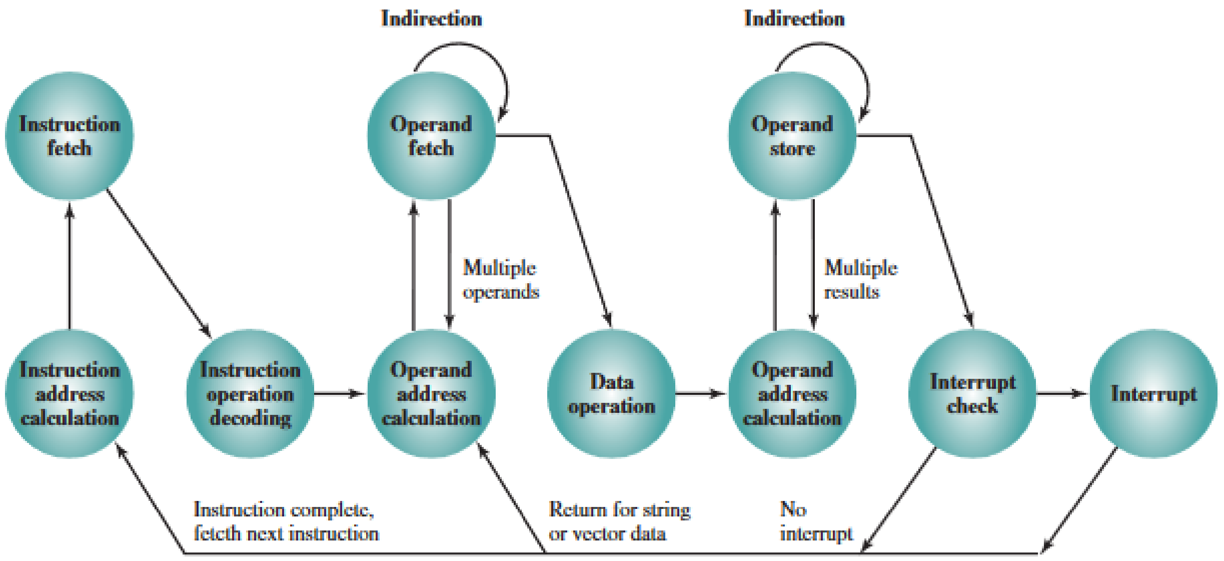

Indirection
Indirection
Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data operation
Operand address calculation
Interrupt check
Interrupt
Instruction complete, fetch next instruction
Return for string or vector data
No interrupt

Figure 14.5 Instruction Cycle State Diagram

Data Flow:

- It depend on the processor design.
- For instruction fetch:
  - PC contains address of next instruction
  - Address is moved to MAR.
  - Address is then placed on the address bus.
  - Control unit requests memory read signal.
  - Result is placed on data bus, then copied to MBR and then to IR.
  - Meanwhile PC is incremented by 1.

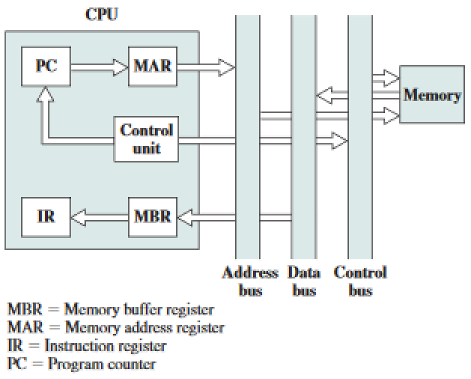

CPU
PC
MAR
Control unit
IR
MBR
Memory
Address bus
Data bus
Control bus
MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

Figure 14.6 Data Flow, Fetch Cycle

- For data fetch:
  - IR is examined.
  - If indirect addressing, then indirect cycle is performed,
    - Right most N bits of MBR transferred to MAR.
    - Control unit request memory read signal.
    - Result (address of operand) is moved to MBR.

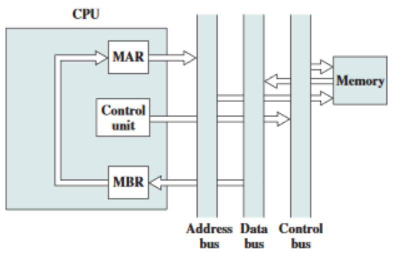


Figure 14.7 Data Flow, Indirect Cycle

- For interrupt:
  - Simple
  - Predictable
  - Current contents of the PC must be saved so that the processor can resume normal activity after interrupt.
  - Contents of PC copied to MBR.
  - Special memory location (stack pointer) is loaded into MAR from the CU.
  - Content of MBR is written onto memory.
  - PC is loaded with the address of the interrupt handling routine.

- Next instruction cycle begins by fetching the appropriate instruction.

INSTRUCTION PIPELINING:

In instruction pipelining, instruction execution cycle is perceived as being divided into a number of stages, where new instruction are accepted at one end before previously accepted instruction completes itself and appears as outputs at the other end.

An instruction can be divided into two stages: fetch instruction and execute instruction. There is time during the execution of instruction when main memory is not being accessed. This time can be used to fetch the next instruction in parallel with the execution of the current one.

The pipeline has two independent stages:

- The first stage fetches instruction and buffers it.
- When the second stage is free, the first stage passes it the buffered instruction.
- While the second stage is executing the instruction, the first stage takes advantage of any unused memory cycles to fetch and buffer the next instruction. This is called instruction pre-fetch or fetch overlap.

Pipelining requires registers to store data between stages.

This process will speed up instruction execution. If the fetch and execute stages were of equal duration, the instruction cycle time would be halved. However, this doubling of execution rate is unlikely for two reasons:

1. The execution time will generally be longer than the fetch time. Execution will involve reading and storing operands and the performance of some operation. Thus, the fetch stage may have to wait for some time before it can empty its buffer.
2. A conditional branch instruction makes the address of the next instruction to be fetched unknown. Thus, the fetch stage must wait until it receives the next instruction address from the execute stage. The execute stage may then have to wait while the next instruction is fetched.

Guessing can reduce the time loss from the 2nd reason.

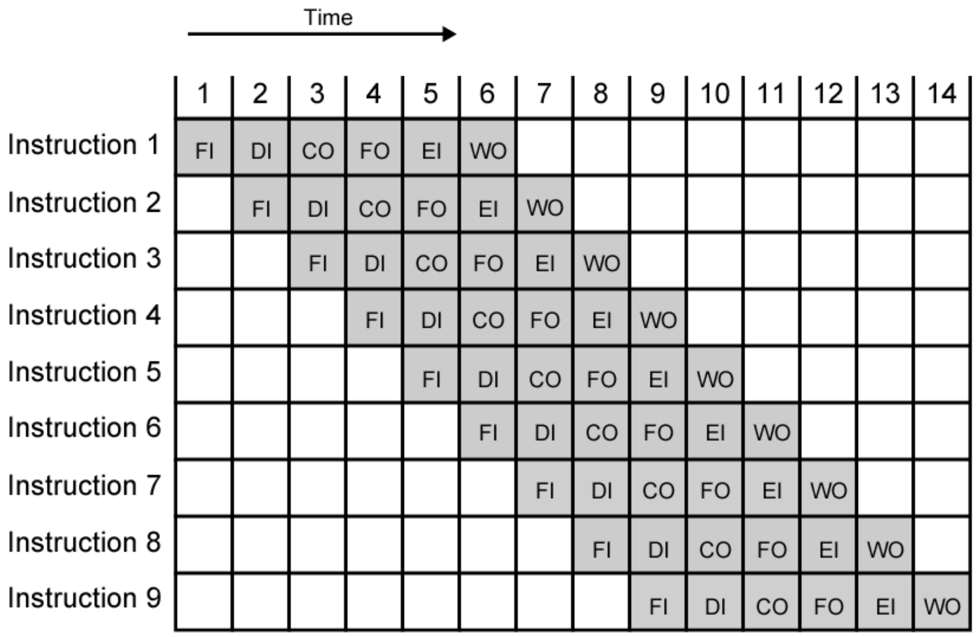
A simple rule is the following: When a conditional branch instruction is passed on from the fetch to the execute stage, the fetch stage fetches the next instruction in memory after the branch instruction. Then, if the branch is not taken, no time is lost.

If the branch is taken, the fetched instruction must be discarded and a new instruction fetched.

To gain further speedup, the pipeline must have more stages. A suggestive decomposition of the instruction processing:

- Fetch instruction (FI): Read the next expected instruction into a buffer.
- Decode instruction (DI): Determine the opcode and the operand specifiers.
- Calculate operands (CO): Calculate the effective address of each source operand. This may involve displacement, register indirect, indirect, or other forms of address calculation.
- Fetch operands (FO): Fetch each operand from memory. Operands in registers need not be fetched.
- Execute instruction (EI): Perform the indicated operation and store the result, if any, in the specified destination operand location.
- Write operand (WO): Store the result in memory.

With this decomposition, the various stages will be of more nearly equal duration. Using this assumption, a six-stage pipeline can reduce the execution time for 9 instructions from 54 time units to 14 time units.
Using this assumption, we can see that a six-stage pipeline can reduce the execution time for 9 instructions from 54 time units to 14 time units.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

Assumptions:

1) We assumes that each instruction goes through all six stages of the pipeline. This will not always be the case.
For example, a load instruction does not need the Write Operand WO stage. However, to simplify the pipeline hardware, the timing is set up assuming that each instruction requires all six stages.

2) We assumes that all of the stages can be performed in parallel and there are no memory conflicts. For example, the FI, FO, and WO stages involve a memory access. We assume that all these accesses can occur simultaneously. Most memory systems will not permit that. However, the desired value may be in cache, or the FO or WO stage may be null. Thus, much of the time, memory conflicts will not slow down the pipeline.

Several other factors serve to limit the performance enhancement.

If all the stages are not of equal duration, there will be some waiting involved at various pipeline stages.

In case of a conditional branch instruction, several instruction fetches can be invalidated.

An interrupt can also invalidate several instruction fetches.

Other problems arise that did not appear in our simple two-stage organization. The CO stage may depend on the contents of a register that could be altered by a previous instruction that is still in the pipeline. Other such register and memory conflicts could occur. The system must contain logic to account for this type of conflict.

It appears that the greater the number of stages in the pipeline, the faster the execution rate.

Instruction pipelining is a powerful technique for enhancing performance but requires careful design to achieve optimum results with reasonable complexity.

PIPELINE HAZARD

A pipeline hazard occurs when the pipeline, or some portion of the pipeline, must stall because conditions do not permit continued execution. Such a pipeline stall is also referred to as a pipeline bubble. There are three types of hazards: Resource, Data, and Control.

Resource Hazards:
A resource hazard occurs when two (or more) instructions that are already in the pipeline need the same resource.
The result is that the instructions must be executed in serial rather than parallel for a portion of the pipeline.
A resource hazard is sometime referred to as a structural hazard.

Assume a simplified five-stage pipeline, in which each stage takes one clock cycle. Now assume that main memory has a single port and that all instruction fetches and data reads and writes must be performed one at a time. Further, ignore the cache. In this case, an operand read to or write from memory cannot be performed in parallel with an instruction fetch. Therefore, the fetch instruction stage of the pipeline must idle for one cycle before beginning the instruction fetch for instruction I3.

Another example of a resource conflict is a situation in which multiple instructions are ready to enter the execute instruction phase and there is a single ALU.

One solutions to such resource hazards is to increase available resources, such as having multiple ports into main memory and multiple ALU units.

Data Hazards:
A data hazard occurs when there is a conflict in the access of an operand location.

In general terms, we can state the hazard in this form:
- Two instructions in a program are to be executed in sequence and both access a particular memory or register operand. If the two instructions are executed in strict sequence, no problem occurs. However, if the instructions are executed in a pipeline, then it is possible for the operand value to be updated in such a way as to produce a different result than would occur with strict sequential execution. In other words, the program produces an incorrect result because of the use of pipelining.

There are three types of data hazards;
- Read after write (RAW), or true dependency: An instruction modifies a register or memory location and a succeeding instruction reads the data in that memory or register location. A hazard occurs if the read takes place before the write operation is complete.
- Write after read (WAR), or anti-dependency: An instruction reads a register or memory location and a succeeding instruction writes to the location. A hazard occurs if the write operation completes before the read operation takes place.

- Write after write (WAW), or output dependency: Two instructions both write to the same location. A hazard occurs if the write operations take place in the reverse order of the intended sequence.

Control Hazards:
A control hazard, also known as a branch hazard, occurs when the pipeline makes the wrong decision on a branch prediction and therefore brings instructions into the pipeline that must subsequently be discarded.