B.E Electronics and Communication Engineering **2018-2019**

Semester-V

18BEEC502 **COMPUTER ARCHITECTURE** **3H-3C**

Instruction Hours/week: L: 3 T: 0 P: 0 **Marks:** Internal**:40** External**:60** Total**:100**

End Semester Exam:3 Hours

Course Objectives

- To introduce the basic structure and operation of digital computing systems
- To familiarize the students with arithmetic and logic unit and implementation of fixed point and floating-point arithmetic operations.
- To expose the students to the concept of pipelining.
- To expose the students with different ways of I/O devices and standard

Course Outcomes

At the end of this course students will demonstrate the ability to

- Gain mastery about working principles of computers
- Analyze the performance of computers
- Design ALU
- Gain knowledge on Processing ,memory and other units of a computer.
- Know how computers are designed and built
- Understand issues affecting modern processors (caches, pipelines etc.)

UNIT I ARCHITECTURE OF COMPUTING SYSTEMS

Basic Structure of Computers, Functional units, software, performance issues software, machine instructions and programs, Types of instructions, Instruction sets: Instruction formats, Assembly language, Stacks, Queues, Subroutines. Processor organization, Information representation, number formats.

UNIT II ARITHMETIC UNIT

Multiplication & division, ALU design, Floating Point arithmetic, IEEE 754 floating point formats Control Design, Instruction sequencing, Interpretation

UNIT III PROCESSING UNIT

Hard wired control-Design methods and CPU control unit. Microprogrammed Control - Basic concepts, minimizing microinstruction size, multiplier control unit. Microprogrammed computers - CPU control unit

UNIT IV MEMORY SYSTEM

Memory organization, device characteristics, RAM, ROM, Memory management, Concept of Cache & associative memories, Virtual memory.

UNIT V I/O ORGANIZATION

System organization, Input - Output systems, Interrupt, DMA, Standard I/O interfaces ,Concept of parallel processing, Pipelining, Forms of parallel processing, interconnect network.

Suggested Readings

1. V.Carl Hammacher, “Computer Organisation”, Fifth Edition.
2. A.S.Tanenbum, “Structured Computer Organisation”, PHI, Third edition
3. Y.Chu, "Computer Organization and Microprogramming”, II, Englewood Chiffs, N.J., Prentice Hall.
4. M.M.Mano, “Computer System Architecture”, Edition
5. C.W.Gear, “Computer Organization and Programming”, McGraw Hill, N.V. Edition
6. Hayes J.P, “Computer Architecture and Organization”, PHI, Second edition

UNIT-I ARCHITECTURE OF COMPUTING SYSTEMS

1.1 Computer Organization and Architecture

- Computer Architecture refers to those attributes of a system that have a direct impact on the logical execution of a program. Examples:
  - the instruction set
  - the number of bits used to represent various data types
  - I/O mechanisms
  - memory addressing techniques
- Computer Organization refers to the operational units and their interconnections that realize the architectural specifications. Examples are things that are transparent to the programmer:
  - control signals
  - interfaces between computer and peripherals
  - the memory technology being used
- So, for example, the fact that a multiply instruction is available is a computer architecture issue. How that multiply is implemented is a computer organization issue.

- Architecture is those attributes visible to the programmer
  - Instruction set, number of bits used for data representation, I/O mechanisms, addressing techniques.
  - e.g. Is there a multiply instruction?
- Organization is how features are implemented
  - Control signals, interfaces, memory technology.
  - e.g. Is there a hardware multiply unit or is it done by repeated addition?
- All Intel x86 family share the same basic architecture
- The IBM System/370 family share the same basic architecture
- This gives code compatibility
  - At least backwards
- Organization differs between different versions

1.2 Structure and Function

- Structure is the way in which components relate to each other
- Function is the operation of individual components as part of the structure
- All computer functions are:
  - **Data processing**: Computer must be able to process data which may take a wide variety of forms and the range of processing.
  - **Data storage**: Computer stores data either temporarily or permanently.
  - **Data movement**: Computer must be able to move data between itself and the outside world.
  - **Control**: There must be a control of the above three functions.

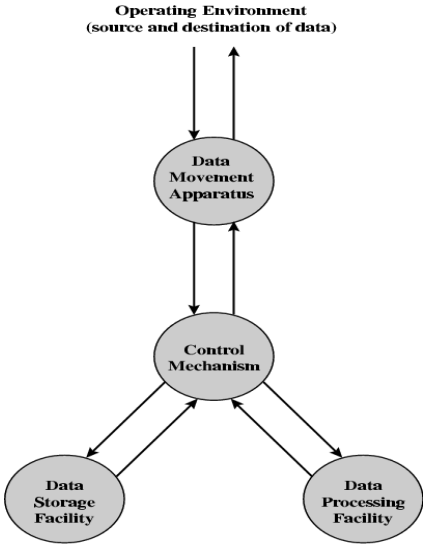


Fig: Functional view of a computer

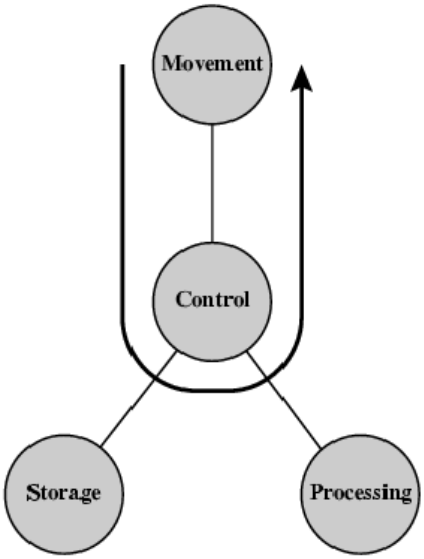


Fig: Data movement operation

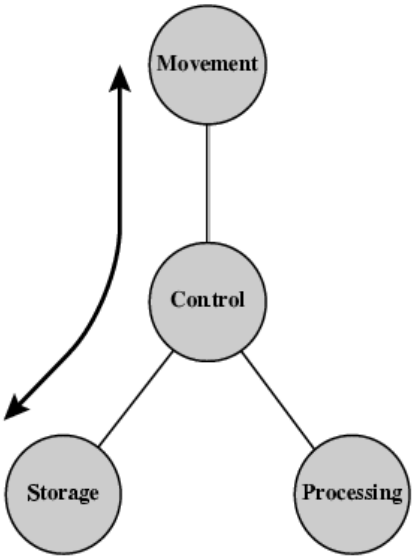


Fig: Storage Operation

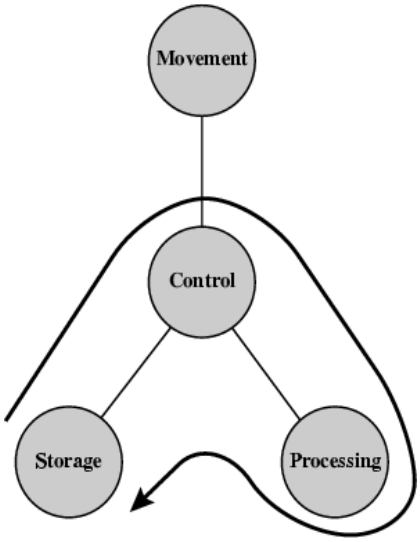


Fig: Processing from / to storage

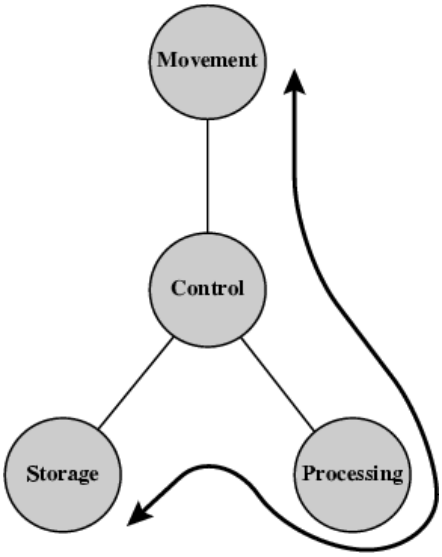


Fig: Processing from storage to i/o

- Four main structural components:
  - Central processing unit (CPU)
  - Main memory
  - I / O
  - System interconnections
- CPU structural components:
  - Control unit
  - Arithmetic and logic unit (ALU)
  - Registers
  - CPU interconnections

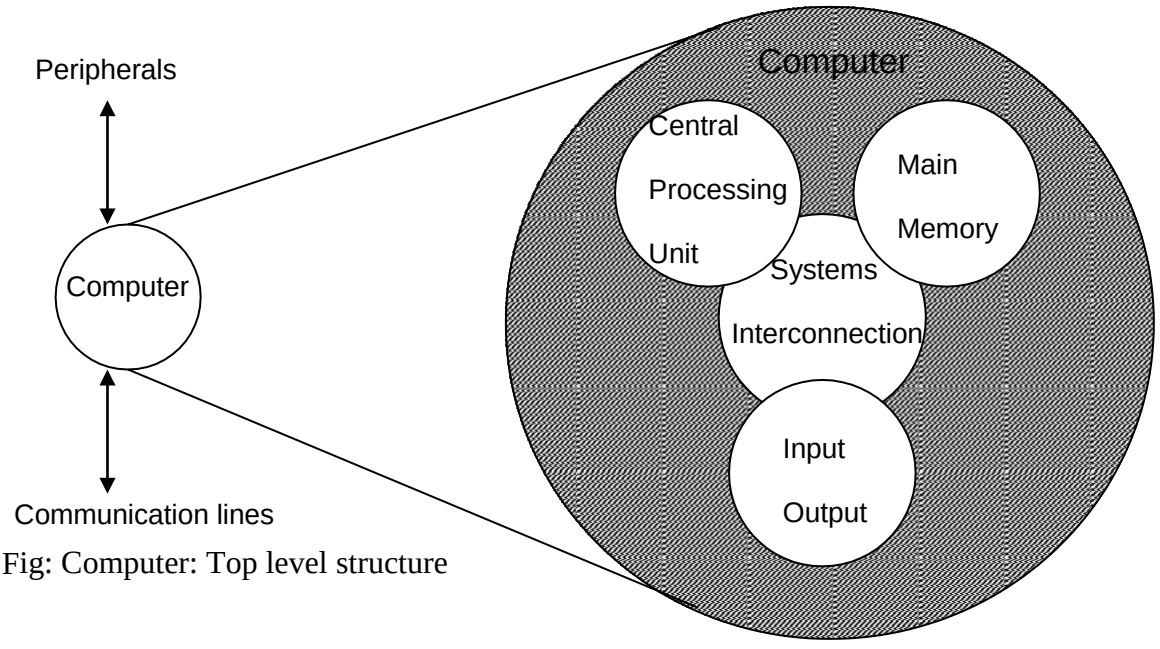


Fig: Computer: Top level structure

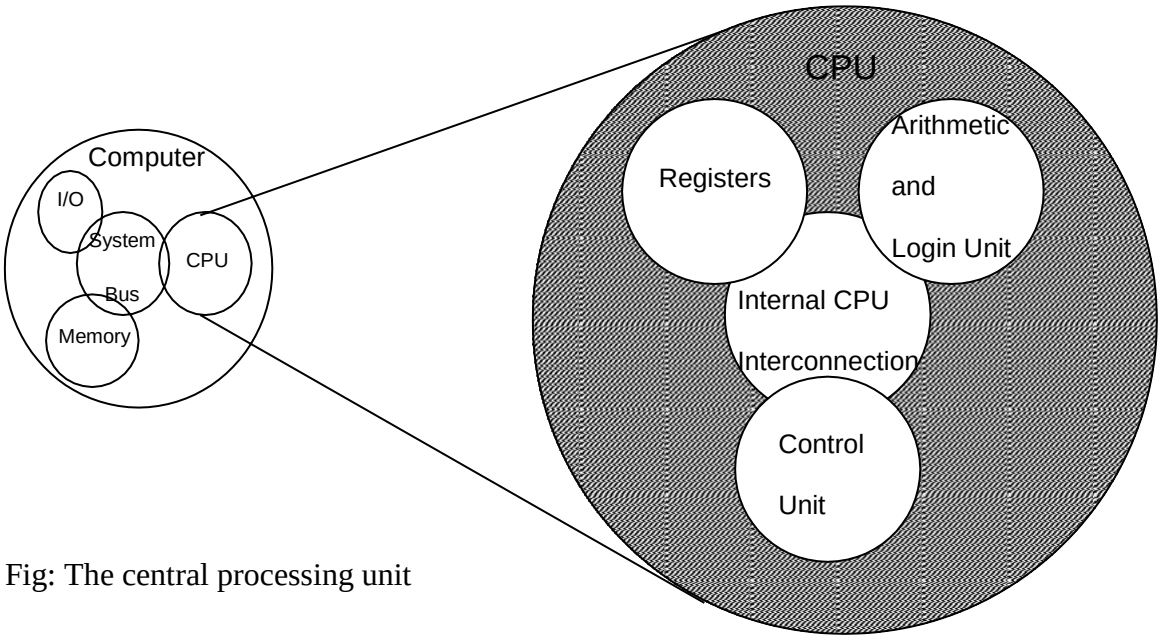


Fig: The central processing unit

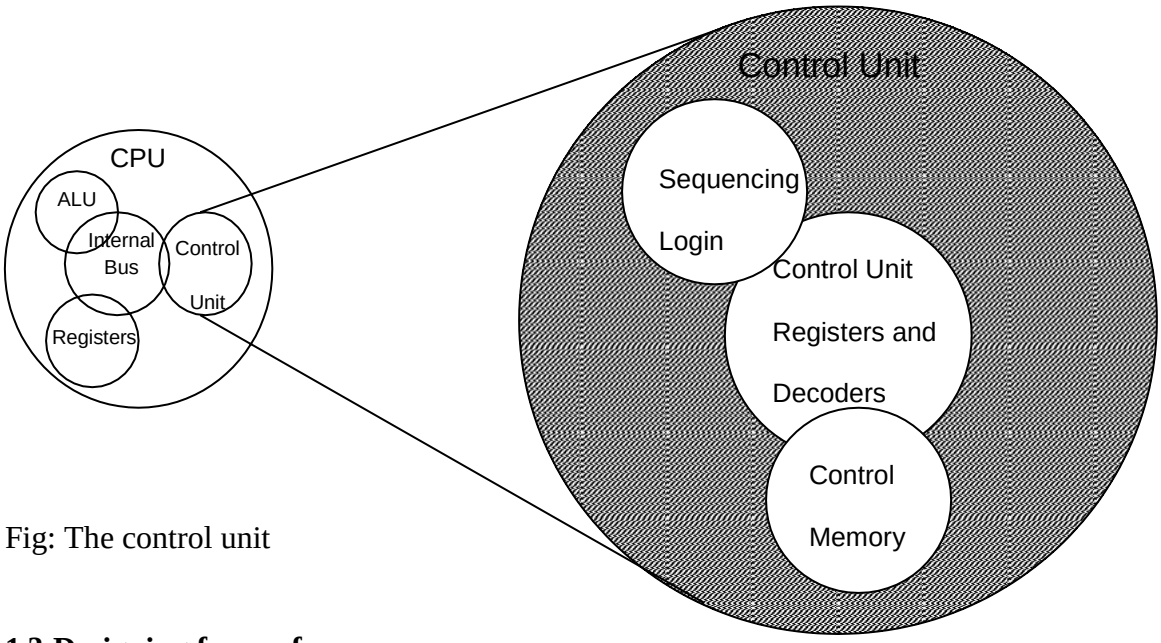


Fig: The control unit

1.3 Designing for performance

Some of the driving factors behind the need to design for performance:

- Microprocessor Speed
  - Pipelining
  - On board cache, on board L1 & L2 cache
  - Branch prediction: The processor looks ahead in the instruction code fetched from memory and predicts which branches, or group of instructions are likely to be processed next.
  - Data flow analysis: The processor analyzes which instructions are dependent on each other's results, or data, to create an optimized schedule of instructions to prevent delay.

- Speculative execution: Using branch prediction and data flow analysis, some processors speculatively execute instructions ahead of their actual appearance in the program execution, holding the results in temporary locations.

- Performance Mismatch
  - Processor speed increased
  - Memory capacity increased
  - Memory speed lags behind processor speed

Below figure depicts the history; while processor speed and memory capacity have grown rapidly, the speed with which data can be transferred between main memory and the processor has lagged badly.

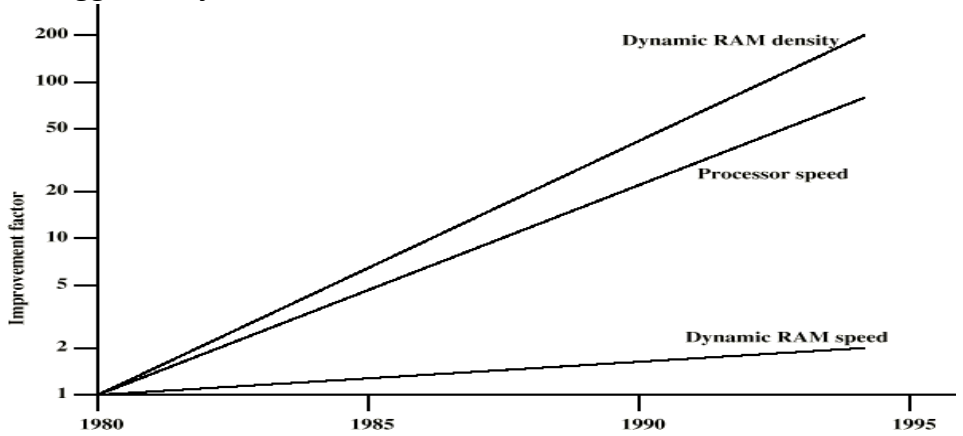


Fig: Evolution of DRAM and processor Characteristics

The effects of these trends are shown vividly in figure below. The amount of main memory needed is going up, but DRAM density is going up faster (number of DRAM per system is going down).

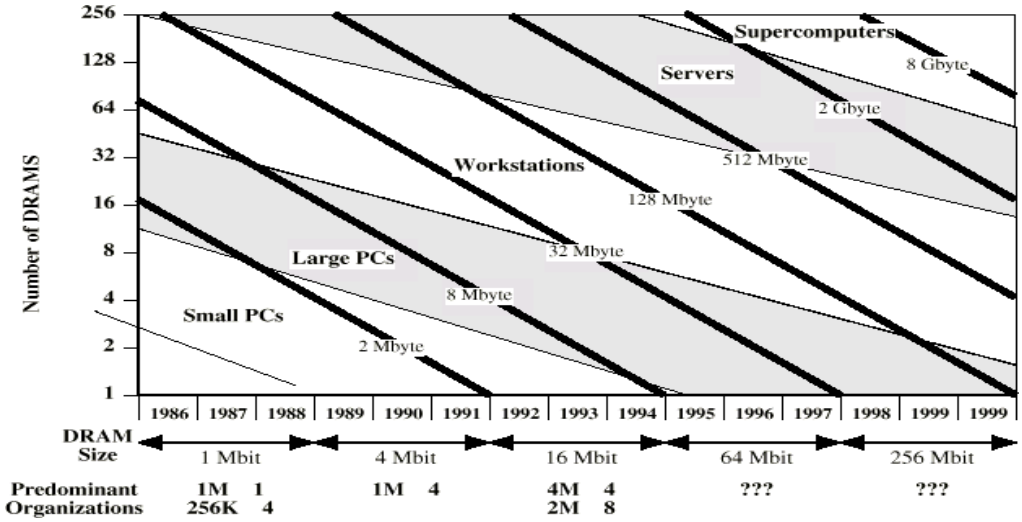


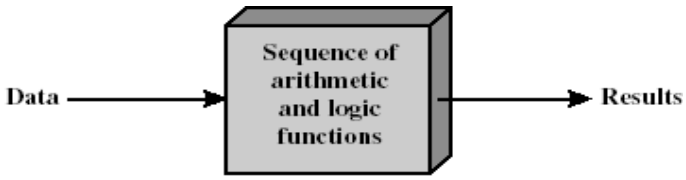
Fig: Trends in DRAM use

Solutions

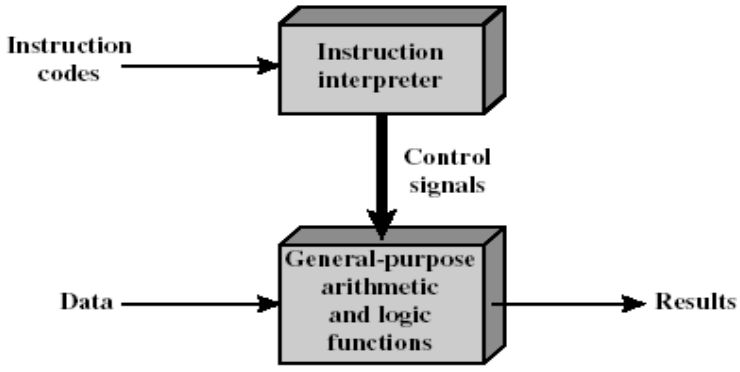
- Increase number of bits retrieved at one time
  - Make DRAM "wider" rather than "deeper" to use wide bus data paths.
- Change DRAM interface
  - Cache
- Reduce frequency of memory access
  - More complex cache and cache on chip
- Increase interconnection bandwidth
  - High speed buses
  - Hierarchy of buses

1.4 Computer Components

- The Control Unit (CU) and the Arithmetic and Logic Unit (ALU) constitute the Central Processing Unit (CPU)
- Data and instructions need to get into the system and results need to get out
  - Input/output (I/O module)
- Temporary storage of code and results is needed
  - Main memory (RAM)
- Program Concept
  - Hardwired systems are inflexible
  - General purpose hardware can do different tasks, given correct control signals
  - Instead of re-wiring, supply a new set of control signals

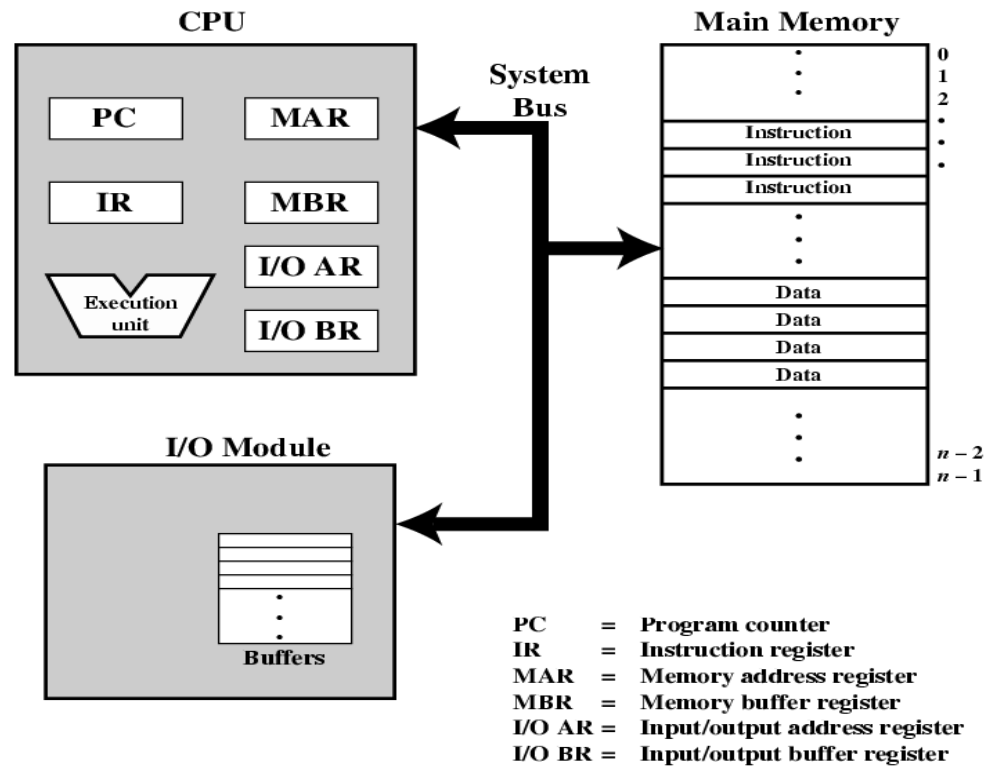


(a) Programming in hardware



(b) Programming in software

Fig: Hardware and Software Approaches



1.5 Computer Function

The basic function performed by a computer is execution of a program, which consists of a set of instructions stored in memory.

- Two steps of Instructions Cycle:
  - Fetch
  - Execute

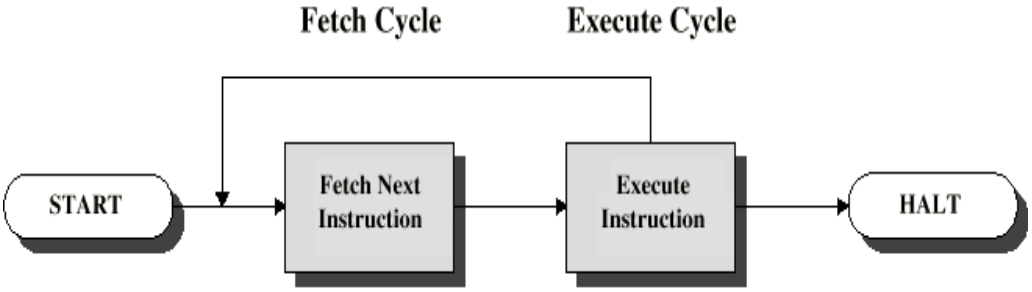


Fig: Basic Instruction Cycle

- Fetch Cycle
  - Program Counter (PC) holds address of next instruction to fetch
  - Processor fetches instruction from memory location pointed to by PC
  - Increment PC
    - Unless told otherwise
  - Instruction loaded into Instruction Register (IR)

- Execute Cycle
  - Processor interprets instruction and performs required actions, such as:
    - Processor - memory
      - data transfer between CPU and main memory
    - Processor - I/O
      - Data transfer between CPU and I/O module
    - Data processing
      - Some arithmetic or logical operation on data
    - Control
      - Alteration of sequence of operations
      - e.g. jump
    - Combination of above

Example of program execution.

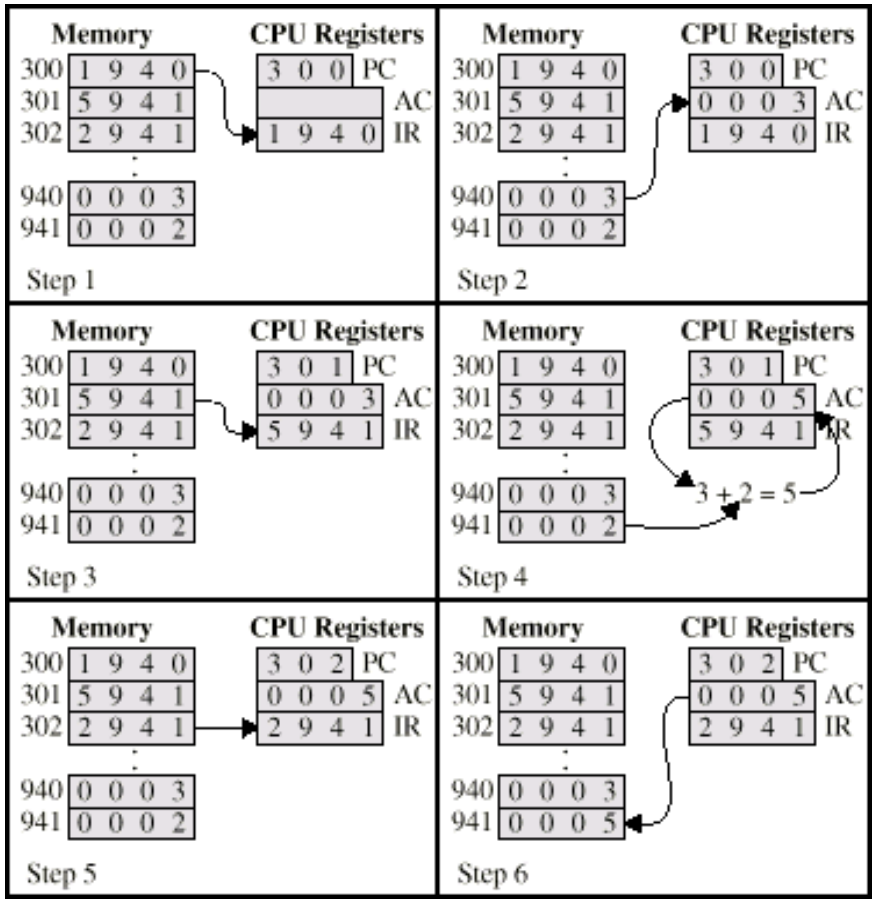


Fig: Example of program execution (consists of memory and registers in hexadecimal)

- The PC contains 300, the address of the first instruction. The instruction (the value 1940 in hex) is loaded into IR and PC is incremented. This process involves the use of MAR and MBR.
- The first hexadecimal digit in IR indicates that the AC is to be loaded. The remaining three hexadecimal digits specify the address (940) from which data are to be loaded.
- The next instruction (5941) is fetched from location 301 and PC is incremented.

- The old contents of AC and the contents of location 941 are added and the result is stored in the AC.
- The next instruction (2941) is fetched from location 302 and the PC is incremented.
- The contents of the AC are stored in location 941.

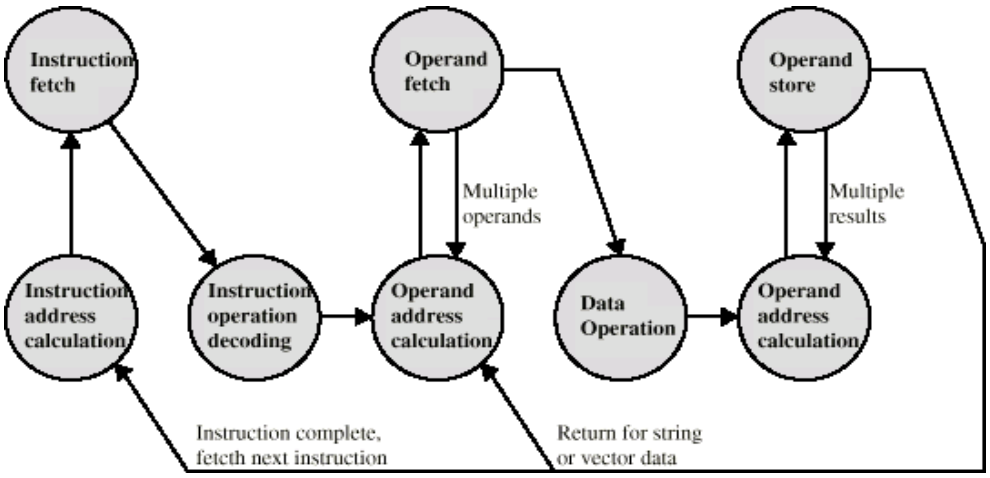
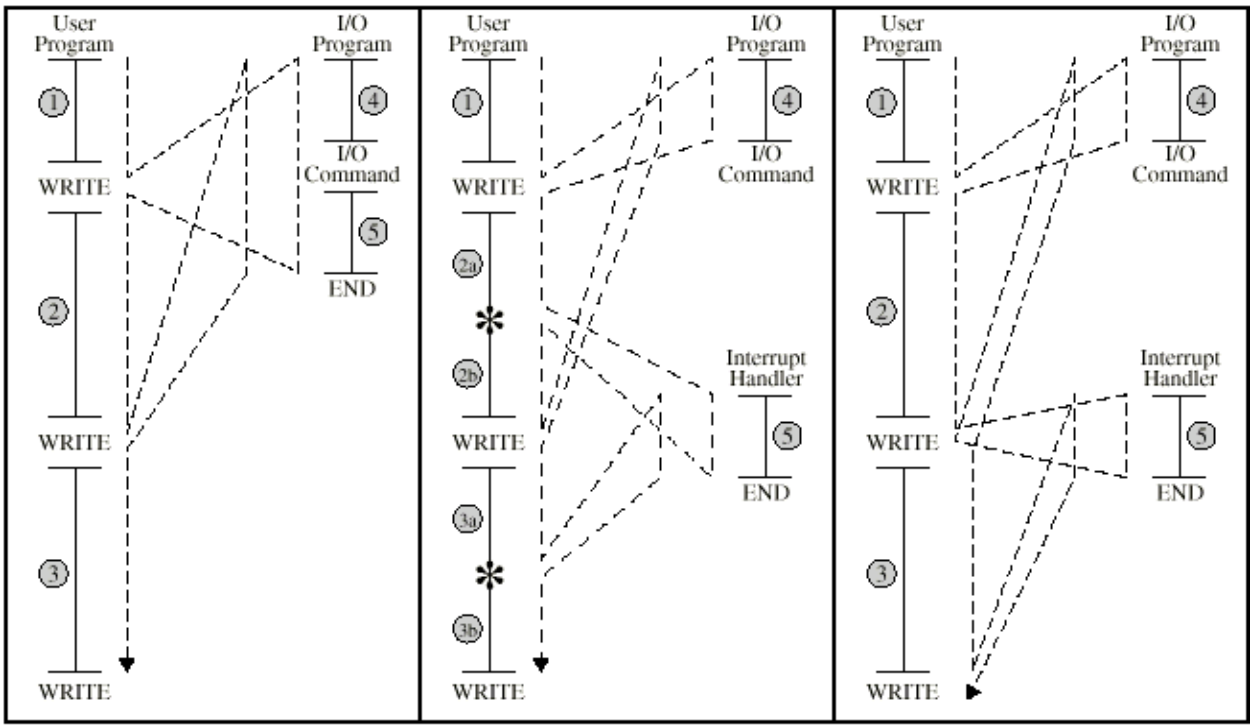


Fig: Instruction cycle state diagram

Interrupts:

- Mechanism by which other modules (e.g. I/O) may interrupt normal sequence of processing
- Program
  - e.g. overflow, division by zero
- Timer
  - Generated by internal processor timer
  - Used in pre-emptive multi-tasking
- I/O
  - from I/O controller
- Hardware failure
  - e.g. memory parity error



- Indicated by an interrupt signal
- If no interrupt, fetch next instruction
- If interrupt pending:
  - Suspend execution of current program
  - Save context
  - Set PC to start address of interrupt handler routine
  - Process interrupt
  - Restore context and continue interrupted program

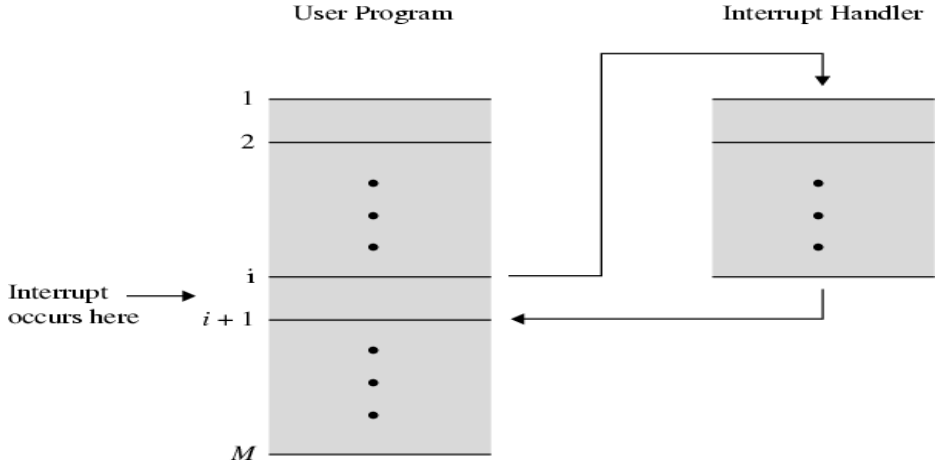


Fig: Transfer of control via interrupts

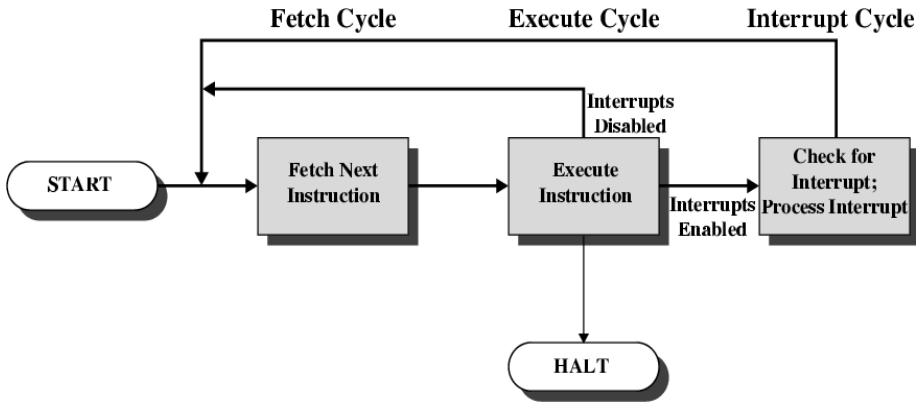


Fig: Instruction Cycle with Interrupts

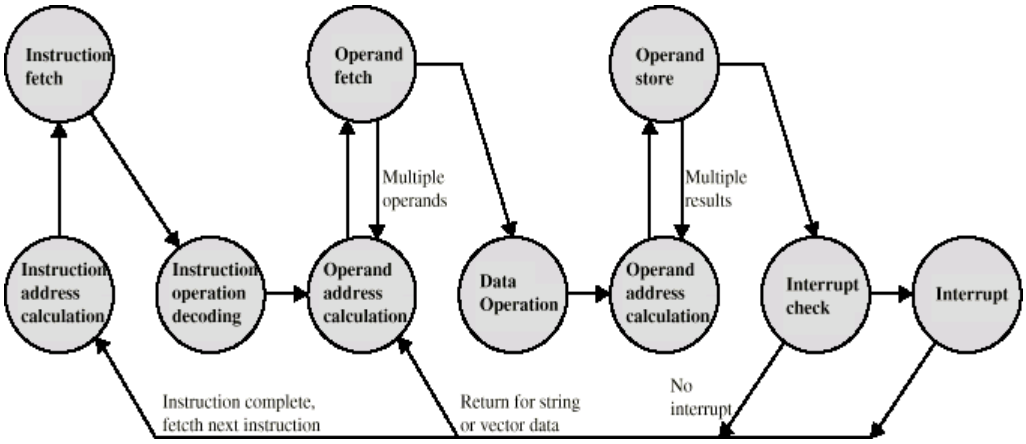


Fig: Instruction cycle state diagram, with interrupts

- Multiple Interrupts
  - Disable interrupts (approach #1)
    - Processor will ignore further interrupts whilst processing one interrupt
    - Interrupts remain pending and are checked after first interrupt has been processed
    - Interrupts handled in sequence as they occur
  - Define priorities (approach #2)
    - Low priority interrupts can be interrupted by higher priority interrupts
    - When higher priority interrupt has been processed, processor returns to previous interrupt

1.6 Interconnection structures

The collection of paths connecting the various modules is called the interconnecting structure.

- All the units must be connected
- Different type of connection for different type of unit
  - Memory
  - Input/Output
  - CPU

- Memory Connection
  - Receives and sends data
  - Receives addresses (of locations)
  - Receives control signals
    - Read
    - Write
    - Timing

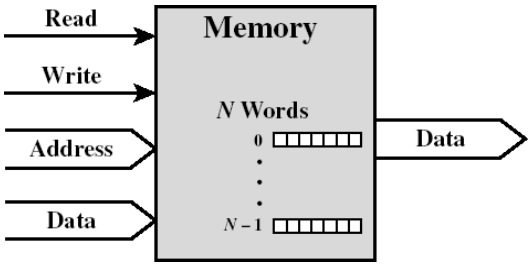


Fig: Memory Module

- I/O Connection
  - Similar to memory from computer's viewpoint
  - Output
    - Receive data from computer
    - Send data to peripheral
  - Input
    - Receive data from peripheral
    - Send data to computer
  - Receive control signals from computer
  - Send control signals to peripherals
    - e.g. spin disk
  - Receive addresses from computer
    - e.g. port number to identify peripheral
  - Send interrupt signals (control)

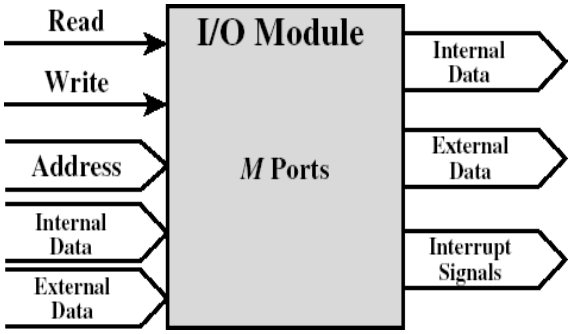


Fig: I/O Module

- CPU Connection
  - Reads instruction and data
  - Writes out data (after processing)
  - Sends control signals to other units
  - Receives (& acts on) interrupts

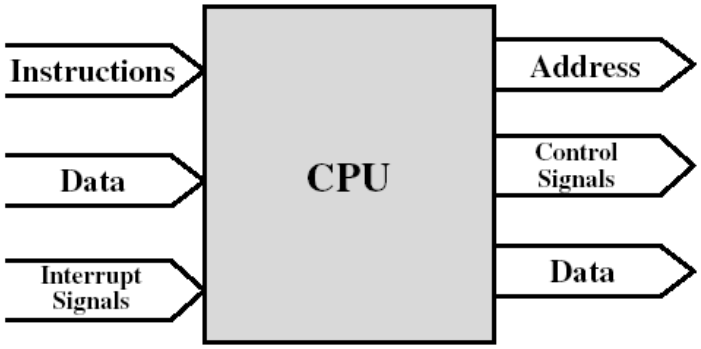


Fig: CPU Module

1.7 Bus interconnection

- A bus is a communication pathway connecting two or more devices
- Usually broadcast (all components see signal)
- Often grouped
  - A number of channels in one bus
  - e.g. 32 bit data bus is 32 separate single bit channels
- Power lines may not be shown
- There are a number of possible interconnection systems
- Single and multiple BUS structures are most common
- e.g. Control/Address/Data bus (PC)
- e.g. Unibus (DEC-PDP)
- Lots of devices on one bus leads to:
  - Propagation delays
  - Long data paths mean that co-ordination of bus use can adversely affect performance
  - If aggregate data transfer approaches bus capacity
- Most systems use multiple buses to overcome these problems

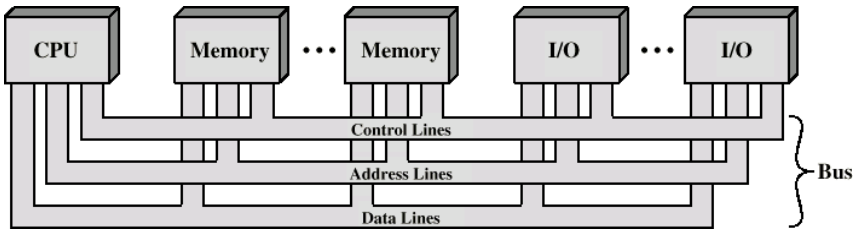


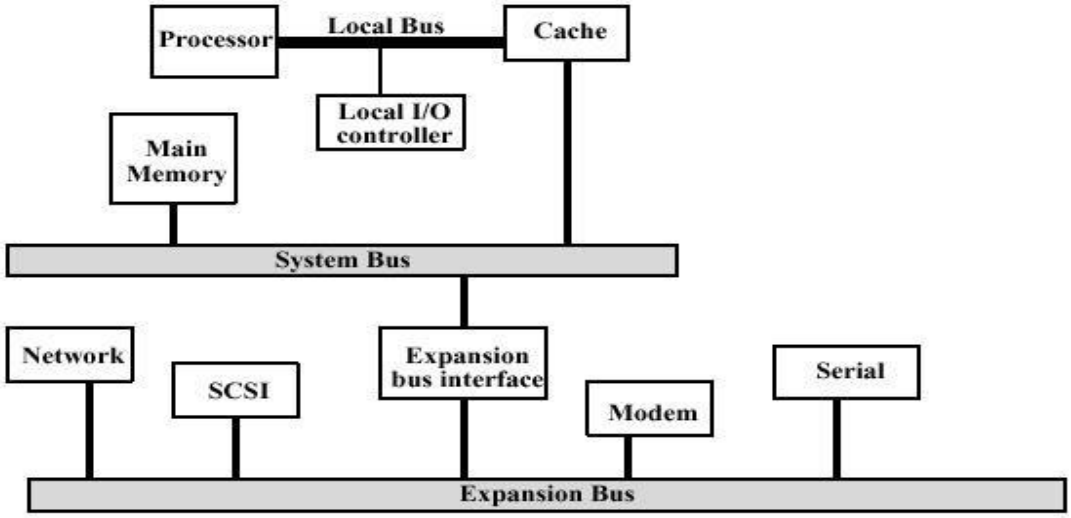
Fig: Bus Interconnection Scheme

- Data Bus
  - Carries data
    - Remember that there is no difference between "data" and "instruction" at this level
  - Width is a key determinant of performance
    - 8, 16, 32, 64 bit
- Address Bus
  - Identify the source or destination of data
  - e.g. CPU needs to read an instruction (data) from a given location in memory
  - Bus width determines maximum memory capacity of system
    - e.g. 8080 has 16 bit address bus giving 64k address space
- Control Bus
  - Control and timing information
    - Memory read
    - Memory write
    - I/O read
    - I/O write
    - Transfer ACK
    - Bus request
    - Bus grant
    - Interrupt request
    - Interrupt ACK
    - Clock
    - Reset

Multiple Bus Hierarchies

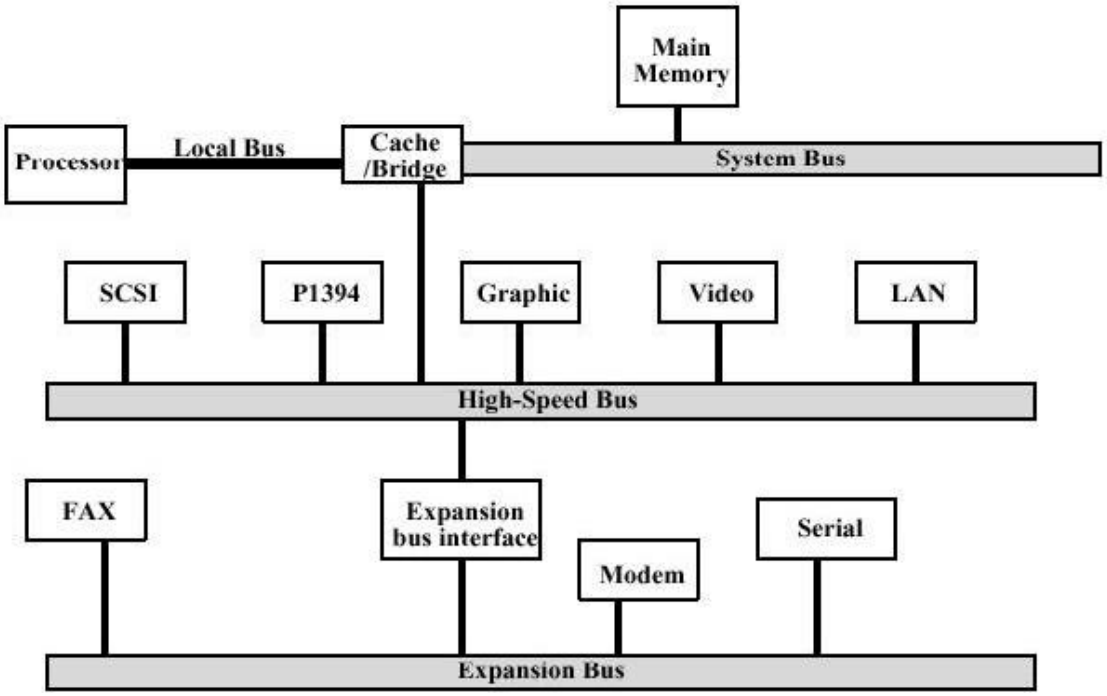
- A great number of devices on a bus will cause performance to suffer
  - Propagation delay - the time it takes for devices to coordinate the use of the bus
  - The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity of the bus (in available transfer cycles/second)
- Traditional Hierarchical Bus Architecture
  - Use of a cache structure insulates CPU from frequent accesses to main memory
  - Main memory can be moved off local bus to a system bus
  - Expansion bus interface
    - buffers data transfers between system bus and I/O controllers on expansion bus
    - insulates memory-to-processor traffic from I/O traffic

Traditional Hierarchical Bus Architecture Example



- High-performance Hierarchical Bus Architecture
    - Traditional hierarchical bus breaks down as higher and higher performance is seen in the I/O devices
    - Incorporates a high-speed bus
        - specifically designed to support high-capacity I/O devices
        - brings high-demand devices into closer integration with the processor and at the same time is independent of the processor
        - Changes in processor architecture do not affect the high-speed bus, and vice versa
    - Sometimes known as a mezzanine architecture

High-performance Hierarchical Bus Architecture Example



Elements of Bus Design

- Bus Types
  - Dedicated
    - Separate data & address lines
  - Multiplexed
    - Shared lines
    - Address valid or data valid control line
    - Advantage - fewer lines
    - Disadvantages
      - More complex control
      - Ultimate performance
- Bus Arbitration
  - More than one module controlling the bus
    - e.g. CPU and DMA controller
  - Only one module may control bus at one time
  - Arbitration may be centralised or distributed
- Centralised Arbitration
  - Single hardware device controlling bus access
    - Bus Controller
    - Arbiter
  - May be part of CPU or separate
- Distributed Arbitration

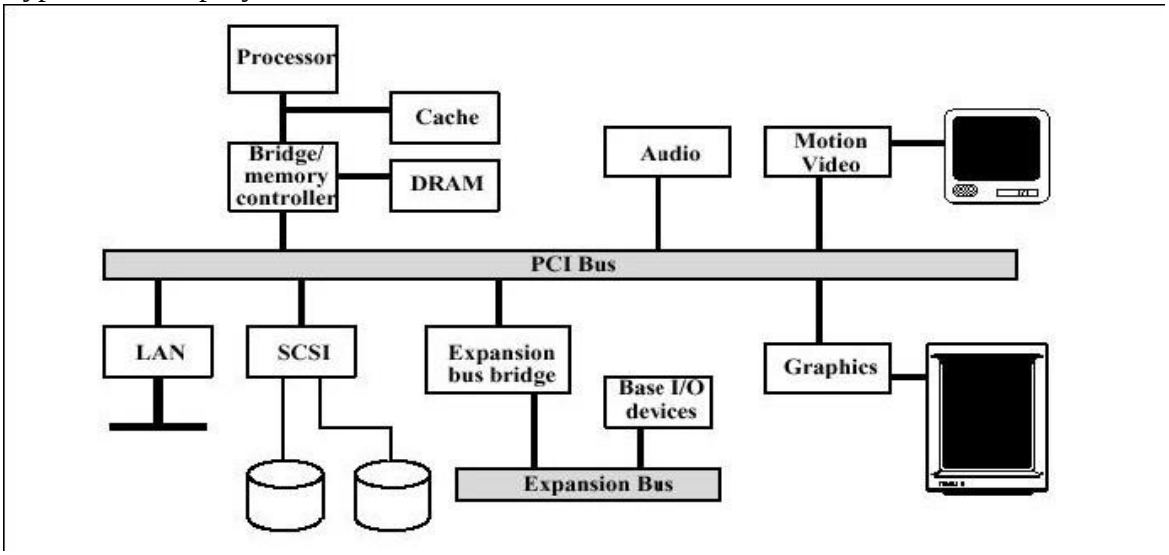
  - Each module may claim the bus
  - Control logic on all modules

- Timing
  - Co-ordination of events on bus
  - Synchronous
    - Events determined by clock signals
    - Control Bus includes clock line
    - A single 1-0 is a bus cycle
    - All devices can read clock line
    - Usually sync on leading edge
    - Usually a single cycle for an event
- Bus Width
  - Address: Width of address bus has an impact on system capacity i.e. wider bus means greater the range of locations that can be transferred.
  - Data: width of data bus has an impact on system performance i.e. wider bus means number of bits transferred at one time.
- Data Transfer Type
  - Read
  - Write
  - Read-modify-write
  - Read-after-write
  - Block

1.8 PCI

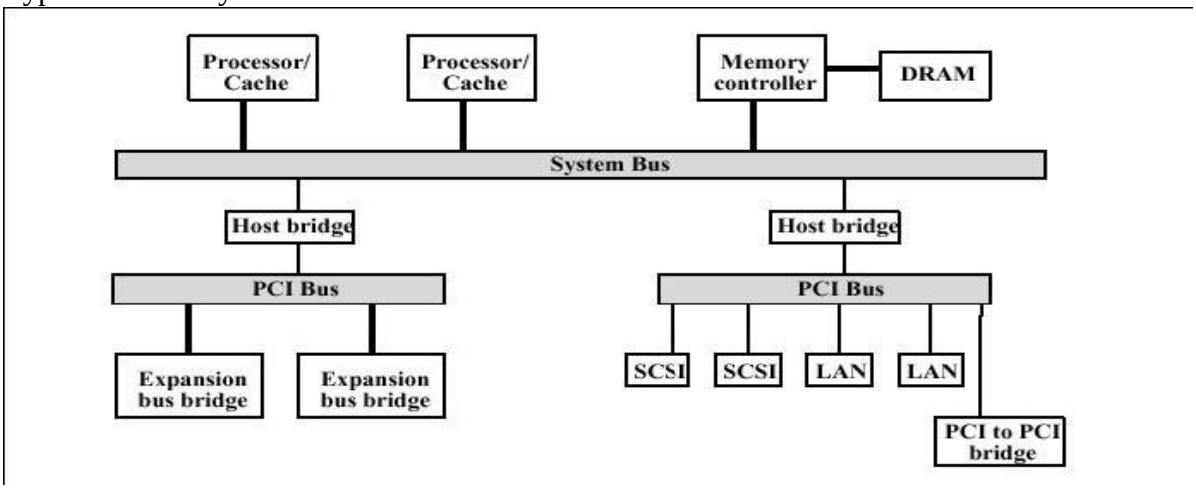
- PCI is a popular high bandwidth, processor independent bus that can function as mezzanine or peripheral bus.
- PCI delivers better system performance for high speed I/O subsystems (graphic display adapters, network interface controllers, disk controllers etc.)
- PCI is designed to support a variety of microprocessor based configurations including both single and multiple processor system.
- It makes use of synchronous timing and centralised arbitration scheme.
- PCI may be configured as a 32 or 64-bit bus.
- Current Standard
  - up to 64 data lines at 33Mhz
  - requires few chips to implement
  - supports other buses attached to PCI bus
  - public domain, initially developed by Intel to support Pentium-based systems
  - supports a variety of microprocessor-based configurations, including multiple processors
  - uses synchronous timing and centralized arbitration

Typical Desktop System



Note: Bridge acts as a data buffer so that the speed of the PCI bus may differ from that of the processor's I/O capability.

Typical Server System



Note: In a multiprocessor system, one or more PCI configurations may be connected by bridges to the processor's system bus.

PCI Bus Lines

- Systems lines
  - Including clock and reset
- Address & Data
  - 32 time mux lines for address/data
  - Interrupt & validate lines
- Interface Control
- Arbitration
  - Not shared
  - Direct connection to PCI bus arbiter

- Error lines
- Interrupt lines
    - Not shared
- Cache support
- 64-bit Bus Extension
    - Additional 32 lines
    - Time multiplexed
    - 2 lines to enable devices to agree to use 64-bit transfer
- JTAG/Boundary Scan
    - For testing procedures

PCI Commands

- Transaction between initiator (master) and target
- Master claims bus
- Determine type of transaction
    - e.g. I/O read/write
- Address phase
- One or more data phases

PCI Enhancements: AGP

- AGP – Advanced Graphics Port
    - Called a port, not a bus because it only connects 2 devices

18BEEC502 COMPUTER ARCHITECTURE 18BEEC502

UNIT-II ARITHMETIC UNIT

Introduction:

Data is manipulated by using the arithmetic instructions in digital computers. Data is manipulated to produce results necessary to give solution for the computation problems. The Addition, subtraction, multiplication and division are the four basic arithmetic operations. If we want then we can derive other operations by using these four operations.

To execute arithmetic operations there is a separate section called arithmetic processing unit in central processing unit. The arithmetic instructions are performed generally on binary or decimal data. Fixed-point numbers are used to represent integers or fractions. We can have signed or unsigned negative numbers. Fixed-point addition is the simplest arithmetic operation.

If we want to solve a problem then we use a sequence of well-defined steps. These steps are collectively called algorithm. To solve various problems we give algorithms.

In order to solve the computational problems, arithmetic instructions are used in digital computers that manipulate data. These instructions perform arithmetic calculations.

And these instructions perform a great activity in processing data in a digital computer. As we already stated that with the four basic arithmetic operations addition, subtraction, multiplication and division, it is possible to derive other arithmetic operations and solve scientific problems by means of numerical analysis methods.

A processor has an arithmetic processor(as a sub part of it) that executes arithmetic operations. The data type, assumed to reside in processor, registers during the execution of an arithmetic instruction. Negative numbers may be in a signed magnitude or signed complement representation. There are three ways of representing negative fixed point - binary numbers signed magnitude, signed 1's complement or signed 2's complement. Most computers use the signed magnitude representation for the mantissa.

Addition and Subtraction :

Addition and Subtraction with Signed –Magnitude Data

We designate the magnitude of the two numbers by A and B. Where the signed numbers are added or subtracted, we find that there are eight different conditions to consider, depending on the sign of the numbers and the operation performed. These conditions are listed in the first column of Table 4.1. The other columns in the table show the actual operation to be performed with the magnitude of the numbers. The last column is needed to present a negative zero. In other words, when two equal numbers are subtracted, the result should be +0 not -0.

The algorithms for addition and subtraction are derived from the table and can be stated as follows (the words parentheses should be used for the subtraction algorithm)

Addition and Subtraction of Signed-Magnitude Number

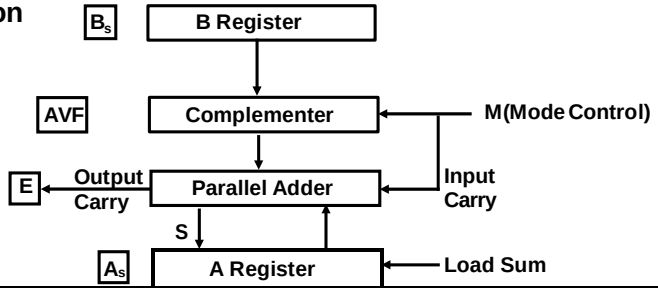
Addition and Subtraction

SIGNED MAGNITUDEADDITION AND SUBTRACTION

Addition: A + B ; A: Augend; B: Addend
Subtraction: A - B: A: Minuend; B: Subtrahend

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

Hardware Implementation

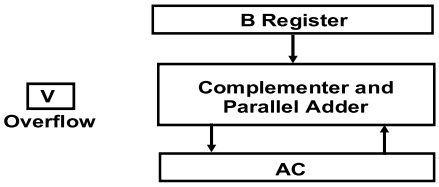


Computer Organization *Prof. H. Yoon*

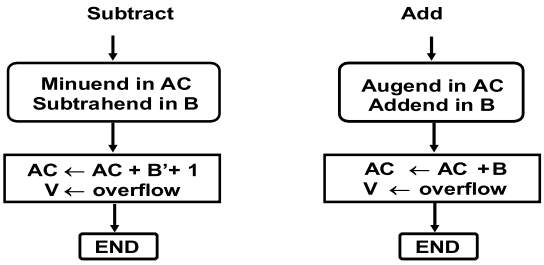
Computer Arithmetic 3 *Addition and Subtraction*

SIGNED 2'S COMPLEMENT ADDITION AND SUBTRACTION

Hardware



Algorithm



Computer Organization *Prof. H. Yoon*

Algorithm:

- The flowchart is shown in Figure 7.1. The two signs A, and B, are compared by an exclusive-OR gate.

  If the output of the gate is 0 the signs are identical; If it is 1, the signs are different.

- For an add operation, identical signs dictate that the magnitudes be added. For a subtract operation, different signs dictate that the magnitudes be added.
- The magnitudes are added with a microoperation EA←A + B, where EA is a register that combines E and A. The carry in E after the addition constitutes an overflow if it is equal to 1. The value of E is transferred into the add-overflow flip-flop AVF.
- The two magnitudes are subtracted if the signs are different for an add operation or identical for a subtract operation. The magnitudes are subtracted by adding A to the 2's complemented B. No overflow can occur if the numbers are subtracted so AVF is cleared to 0.
- 1 in E indicates that A >= B and the number in A is the correct result. If this numbs is zero, the sign A must be made positive to avoid a negative zero.
- 0 in E indicates that A < B. For this case it is necessary to take the 2's complement of the value in A. The operation can be done with one microoperation A←A' +1.
- However, we assume that the A register has circuits for microoperations complement and increment, so the 2's complement is obtained from these two microoperations.
- In other paths of the flowchart, the sign of the result is the same as the sign of A. so no change in A is required. However, when A < B, the sign of the result is the complement of the original sign of A. It is then necessary to complement A, to obtain the correct sign.
- The final result is found in register A and its sign in As. The value in AVF provides an overflow indication. The final value of E is immaterial.
- Figure 7.2 shows a block diagram of the hardware for implementing the addition and subtraction operations.
- It consists of registers A and B and sign flip-flops As and Bs.
- Subtraction is done by adding A to the 2's complement of B.
- The output carry is transferred to flip-flop E , where it can be checked to determine the relative magnitudes of two numbers.
- The add-overflow flip-flop *AVF* holds the overflow bit when A and B are added.
- The A register provides other microoperations that may be needed when we specify the sequence of steps in the algorithm.

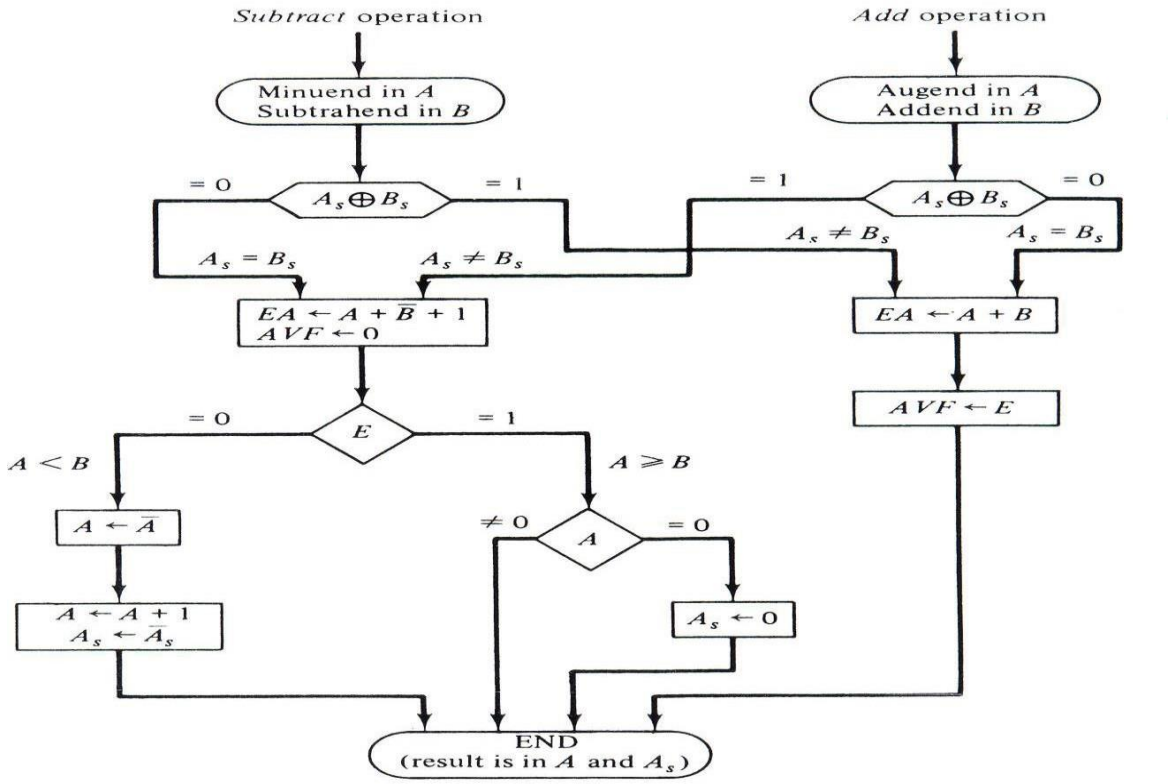
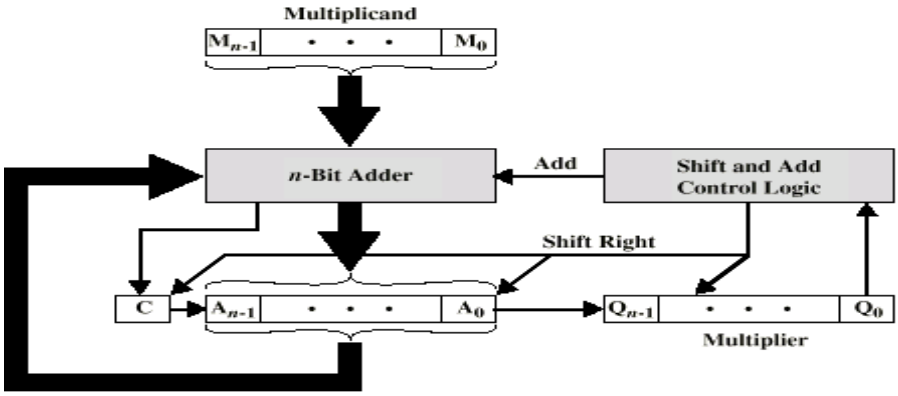


Figure 10-2 Flowchart for add and subtract operations.

Multiplication Algorithm:

In the beginning, the multiplicand is in B and the multiplier in Q. Their corresponding signs are in Bs and Qs respectively. We compare the signs of both A and Q and set to corresponding sign of the product since a double-length product will be stored in registers A and Q. Registers A and E are cleared and the sequence counter SC is set to the number of bits of the multiplier. Since an operand must be stored with its sign, one bit of the word will be occupied by the sign and the magnitude will consist of n-1 bits.

Now, the low order bit of the multiplier in Qn is tested. If it is 1, the multiplicand (B) is added to present partial product (A), 0 otherwise. Register EAQ is then shifted once to the right to form the new partial product. The sequence counter is decremented by 1 and its new value checked. If it is not equal to zero, the process is repeated and a new partial product is formed. When SC = 0 we stops the process.



C	A	Q	M		
0	0000	1101	1011	Initial Values	
0	1011	1101	1011	Add	First Cycle
0	0101	1110	1011	Shift	
0	0010	1111	1011	Shift	Second Cycle
0	1101	1111	1011	Add	Third Cycle
0	0110	1111	1011	Shift	
1	0001	1111	1011	Add	Fourth Cycle
0	1000	1111	1011	Shift	

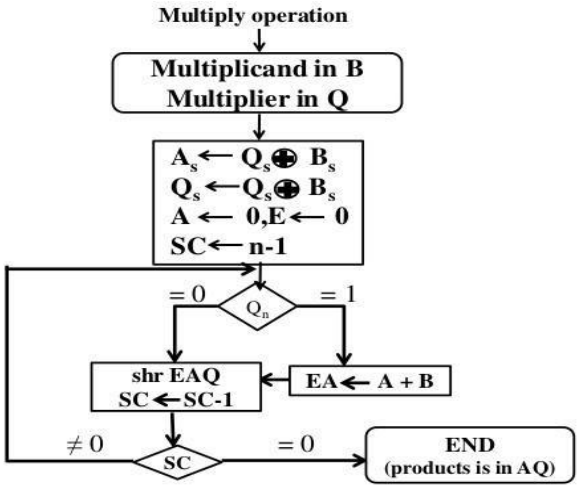


Figure: Flowchart for multiply operation.

Booth's algorithm :

- Booth algorithm gives a procedure for multiplying binary integers in signed- 2's complement representation.
- It operates on the fact that strings of 0's in the multiplier require no addition but just

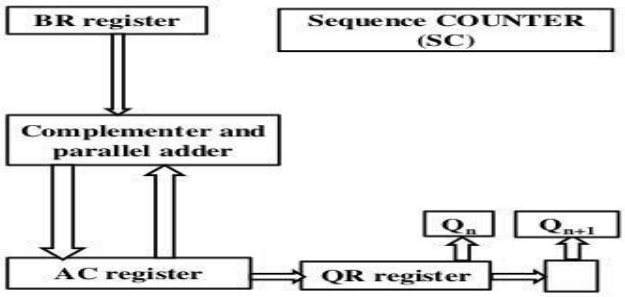
shifting, and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as $2^{k+1} - 2^m$.

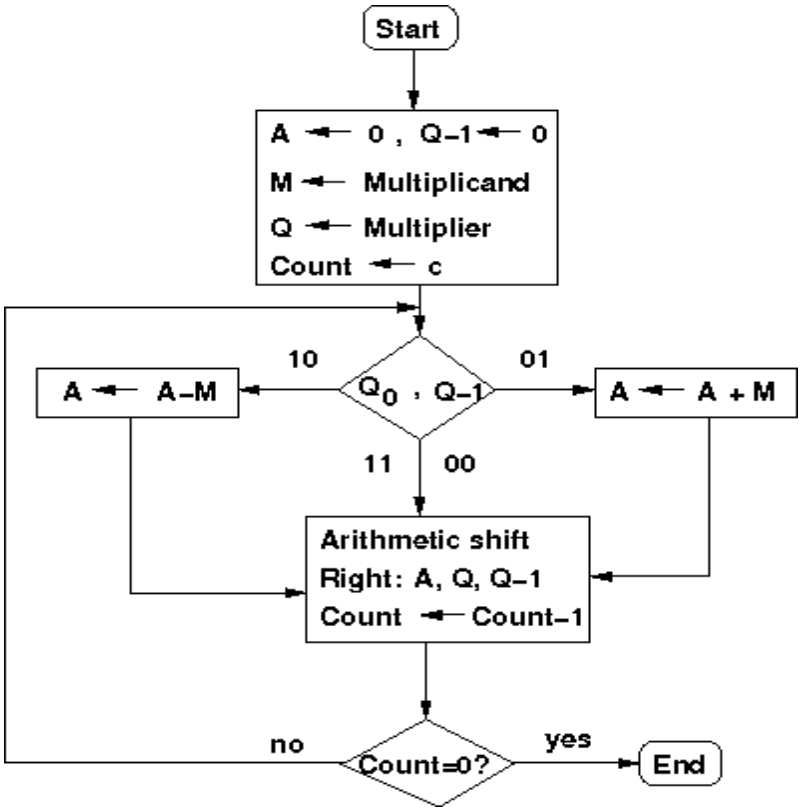
- For example, the binary number 001110 (+14) has a string 1's from 2^3 to 2^1 (k=3, m=1). The number can be represented as $2^{k+1} - 2^m$. = $2^4 - 2^1 = 16 - 2 = 14$. Therefore, the multiplication M X 14, where M is the multiplicand and 14 the multiplier, can be done as M X 2^4 – M X 2^1.

- Thus the product can be obtained by shifting the binary multiplicand M four times to the left and subtracting M shifted left once.

Hardware for Booth Algorithm

- Sign bits are not separated from the rest of the registers
- rename registers A,B, and Q as AC,BR and QR respectively
- Q_n designates the least significant bit of the multiplier in register QR
- Flip-flop Qn+1 is appended to QR to facilitate a double bit inspection of the multiplier





- As in all multiplication schemes, booth algorithm requires examination of the multiplier bits and shifting of partial product.

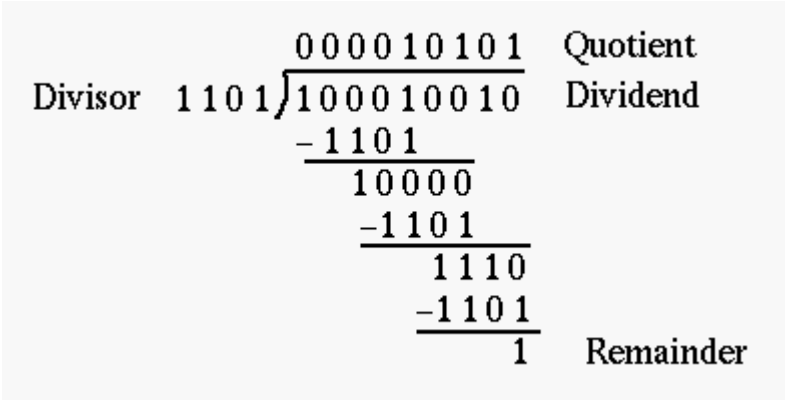
- Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial, or left unchanged according to the following rules:

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

- The algorithm works for positive or negative multipliers in 2's complement representation.
- This is because a negative multiplier ends with a string of 1's and the last operation will be a subtraction of the appropriate weight.
- The two bits of the multiplier in Qn and Qn+1 are inspected.
- If the two bits are equal to 10, it means that the first 1 in a string of 1 's has been encountered. This requires a subtraction of the multiplicand from the partial product in AC.
- If the two bits are equal to 01, it means that the first 0 in a string of 0's has been encountered. This requires the addition of the multiplicand to the partial product in AC.
- When the two bits are equal, the partial product does not change.

Division Algorithms

Division of two fixed-point binary numbers in signed magnitude representation is performed with paper and pencil by a process of successive compare, shift and subtract operations. Binary division is much simpler than decimal division because here the quotient digits are either 0 or 1 and there is no need to estimate how many times the dividend or partial remainder fits into the divisor. The division process is described in Figure



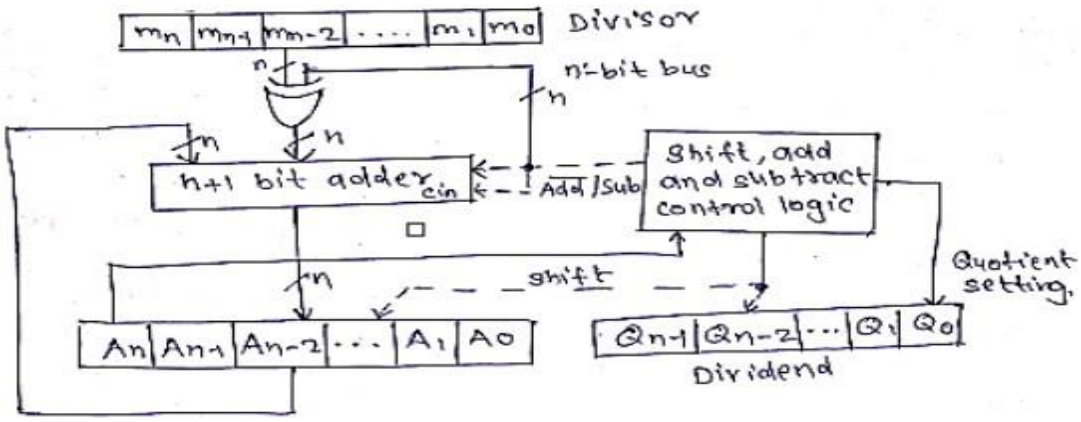
The devisor is compared with the five most significant bits of the dividend. Since the 5-bit number is smaller than B, we again repeat the same process. Now the 6-bit number is greater than B, so we place a 1 for the quotient bit in the sixth position above the dividend. Now we shift the divisor once to the right and subtract it from the dividend. The difference is known as a partial remainder because the division could have stopped here to obtain a quotient of 1 and a remainder equal to the partial

remainder. Comparing a partial remainder with the divisor continues the process. If the partial remainder is greater than or equal to the divisor, the quotient bit is equal to 1. The divisor is then shifted right and subtracted from the partial remainder. If the partial remainder is smaller than the divisor, the quotient bit is 0 and no subtraction is needed. The divisor is shifted once to the right in any case. Obviously the result gives both a quotient and a remainder.

Hardware Implementation for Signed-Magnitude Data

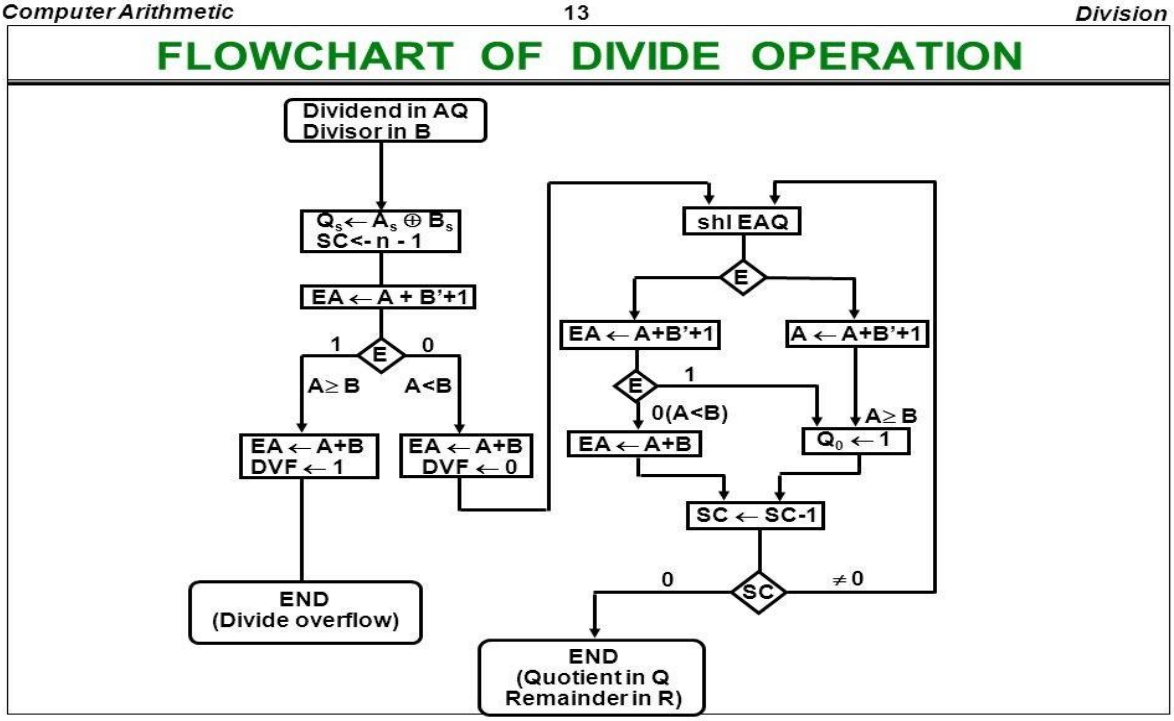
In hardware implementation for signed-magnitude data in a digital computer, it is convenient to change the process slightly. Instead of shifting the divisor to the right, two dividends, or partial remainders, are shifted to the left, thus leaving the two numbers in the required relative position. Subtraction is achieved by adding A to the 2's complement of B. End carry gives the information about the relative magnitudes.

The hardware required is identical to that of multiplication. Register EAQ is now shifted to the left with 0 inserted into Qn and the previous value of E is lost. The example is given in Figure 4.10 to clear the proposed division process. The divisor is stored in the B register and the double-length dividend is stored in registers A and Q. The dividend is shifted to the left and the divisor is subtracted by adding its 2's complement value. E



Hardware Implementation for Signed-Magnitude Data

Algorithm:



Example of Binary Division with Digital Hardware

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

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

Floating-point Arithmetic operations :

In many high-level programming languages we have a facility for specifying floating-point numbers. The most common way is by a real declaration statement. High level programming languages must have a provision for handling floating-point arithmetic operations. The operations are generally built in the internal hardware. If no hardware is available, the compiler must be designed with a package of floating-point software subroutine. Although the hardware method is more expensive, it is much more efficient than the software method. Therefore, floating- point hardware is included in most computers and is omitted only in very small ones.

Basic Considerations :

There are two part of a floating-point number in a computer - a mantissa m and an exponent e. The two parts represent a number generated from multiplying m times a radix r raised to the value of e. Thus

$$m \times r^e$$

The mantissa may be a fraction or an integer. The position of the radix point and the value of the radix r are not included in the registers. For example, assume a fraction representation and a radix 10. The decimal number 537.25 is represented in a register with m = 53725 and e = 3 and is interpreted to represent the floating-point number

$$.53725 \times 10^3$$

A floating-point number is said to be normalized if the most significant digit of the mantissa in nonzero. So the mantissa contains the maximum possible number of significant digits. We cannot normalize a zero because it does not have a nonzero digit. It is represented in floating-point by all 0's in the mantissa and exponent.

Floating-point representation increases the range of numbers for a given register. Consider a computer with 48-bit words. Since one bit must be reserved for the sign, the range of fixed-point integer numbers will be + (24^7 – 1), which is approximately + 101^4. The 48 bits can be used to represent a floating-point number with 36 bits for the mantissa and 12 bits for the exponent. Assuming fraction representation for the mantissa and taking the two sign bits into consideration, the range of numbers that can be represented is

$$+ (1 - 2^{-35}) \times 2^{2047}$$

This number is derived from a fraction that contains 35 1's, an exponent of 11 bits (excluding its sign), and because 2^{11}–1 = 2047. The largest number that can be accommodated is approximately 10^{615}. The mantissa that can accommodated is 35 bits (excluding the sign) and if considered as an integer it can store a number as large as (2^{35} –1). This is approximately equal to 10^{10}, which is equivalent to a decimal number of 10 digits.

Computers with shorter word lengths use two or more words to represent a floating-point number. An 8-bit microcomputer uses four words to represent one floating-point number. One word of 8 bits are reserved for the exponent and the 24 bits of the other three words are used in the mantissa.

Arithmetic operations with floating-point numbers are more complicated than with fixed-point numbers. Their execution also takes longer time and requires more complex hardware. Adding or subtracting two numbers requires first an alignment of the radix point since the exponent parts must be made equal before adding or subtracting the mantissas. We do this alignment by shifting one mantissa while its exponent is adjusted until it becomes equal to the other exponent. Consider the sum of the following floating-point numbers:

$$.5372400 \times 10^{2}$$

$$+ .1580000 \times 10^{-1}$$

Floating-point multiplication and division need not do an alignment of the mantissas. Multiplying the two mantissas and adding the exponents can form the product. Dividing the mantissas and subtracting the exponents perform division.

The operations done with the mantissas are the same as in fixed-point numbers, so the two can share the same registers and circuits. The operations performed with the exponents are compared and incremented (for aligning the mantissas), added and subtracted (for multiplication) and division), and decremented (to normalize the result). We can represent the exponent in any one of the three representations - signed-magnitude, signed 2's complement or signed 1's complement.

Biased exponents have the advantage that they contain only positive numbers. Now it becomes simpler to compare their relative magnitude without bothering about their signs. Another advantage is that the smallest possible biased exponent contains all zeros. The floating-point representation of zero is then a zero mantissa and the smallest possible exponent.

Register Configuration

The register configuration for floating-point operations is shown in figure 4.13. As a rule, the same registers and adder used for fixed-point arithmetic are used for processing the mantissas. The difference lies in the way the exponents are handled.

The register organization for floating-point operations is shown in Fig. 4.13. Three registers are there, BR, AC, and QR. Each register is subdivided into two parts. The mantissa part has the same uppercase letter symbols as in fixed-point representation. The exponent part may use corresponding lower-case letter symbol.

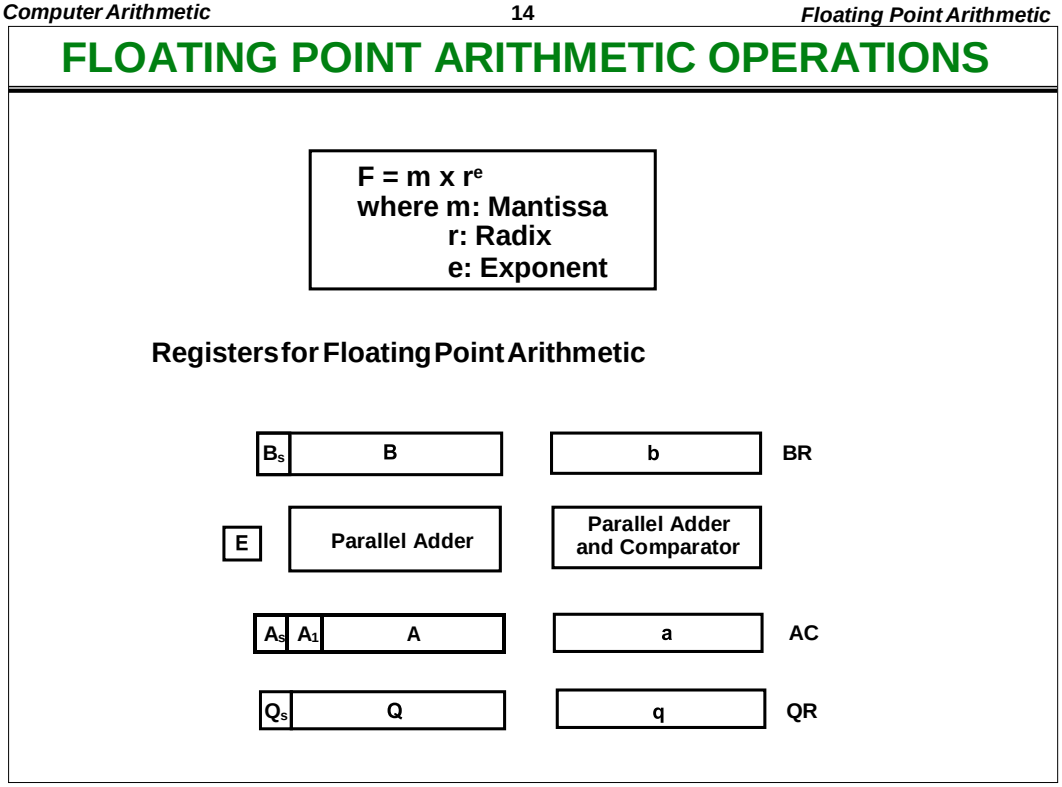


Figure 4.13: Registers for Floating Point arithmetic operations

Assuming that each floating-point number has a mantissa in signed-magnitude representation and a biased exponent. Thus the AC has a mantissa whose sign is in As, and a magnitude that is in A. The diagram shows the most significant bit of A, labeled by A1. The bit in his position must be a 1 to normalize the number. Note that the symbol AC represents the entire register, that is, the concatenation of As, A and a.

In the similar way, register BR is subdivided into Bs, B, and b and QR into Qs, Q and q. A parallel-adder adds the two mantissas and loads the sum into A and the carry into E. A separate parallel adder can be used for the exponents. The exponents do not have a district sign bit because they are biased but are represented as a biased positive quantity. It is assumed that the floating-point number are so large that the chance of an exponent overflow is very remote and so the exponent overflow will be neglected. The exponents are also connected to a magnitude comparator that provides three binary outputs to indicate their relative magnitude.

The number in the mantissa will be taken as a fraction, so they binary point is assumed to reside to the left of the magnitude part. Integer representation for floating point causes certain scaling problems during multiplication and division. To avoid these problems, we adopt a fraction representation.

The numbers in the registers should initially be normalized. After each arithmetic operation, the result will be normalized. Thus all floating-point operands are always normalized.

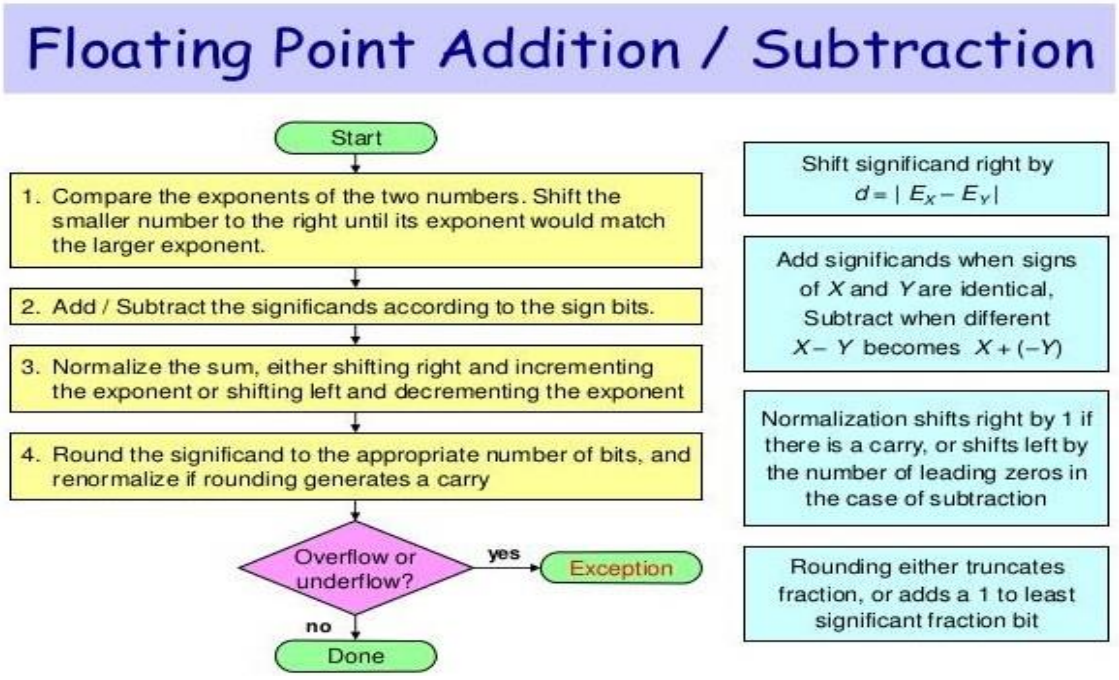
Addition and Subtraction of Floating Point Numbers

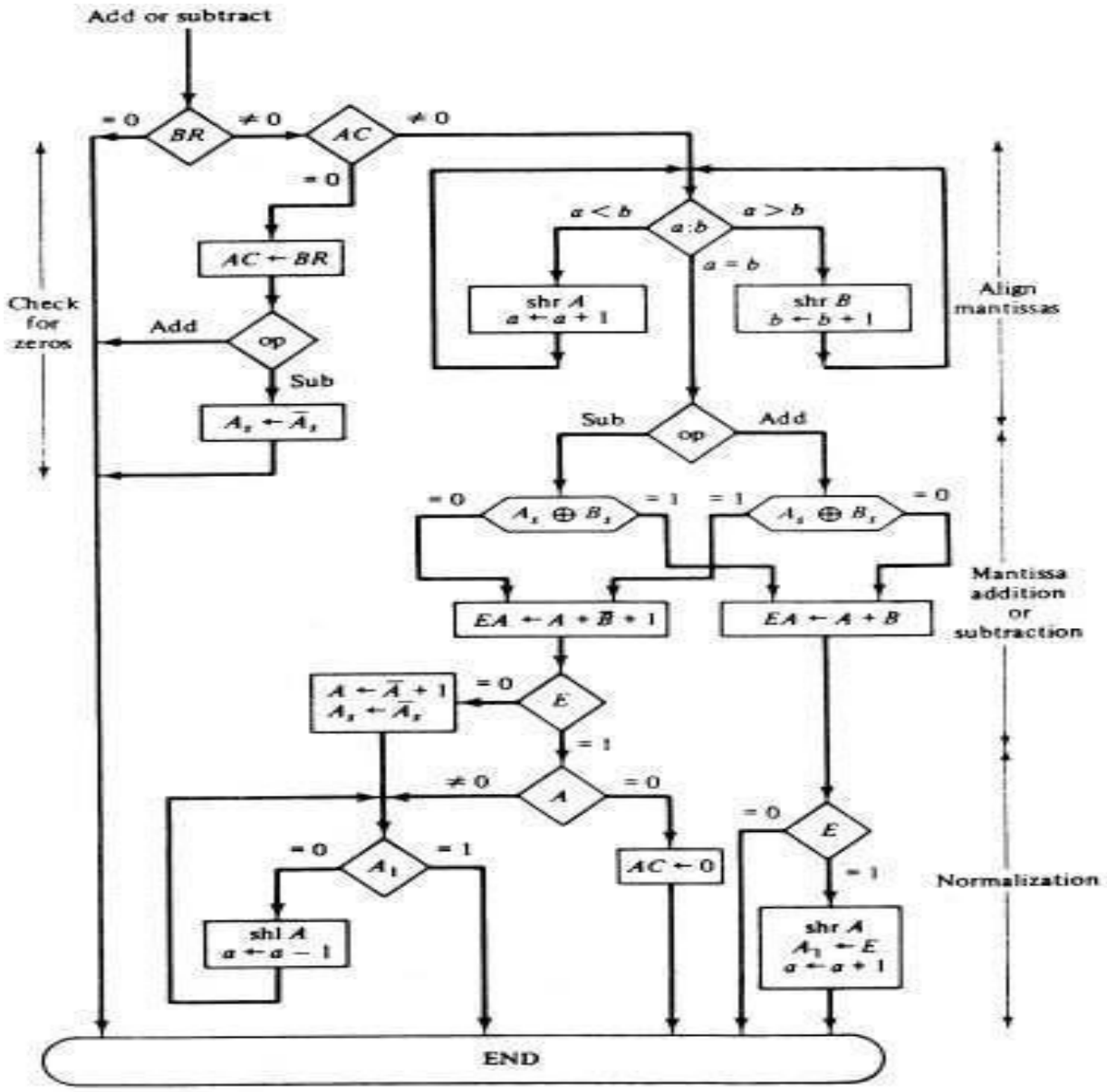
During addition or subtraction, the two floating-point operands are kept in AC and BR. The sum or difference is formed in the AC. The algorithm can be divided into four consecutive parts:

1. Check for zeros.
2. Align the mantissas.
3. Add or subtract the mantissas
4. Normalize the result

A floating-point number cannot be normalized, if it is 0. If this number is used for computation, the result may also be zero. Instead of checking for zeros during the normalization process we check for zeros at the beginning and terminate the process if necessary. The alignment of the mantissas must be carried out prior to their operation. After the mantissas are added or subtracted, the result may be un-normalized. The normalization procedure ensures that the result is normalized before it is transferred to memory.

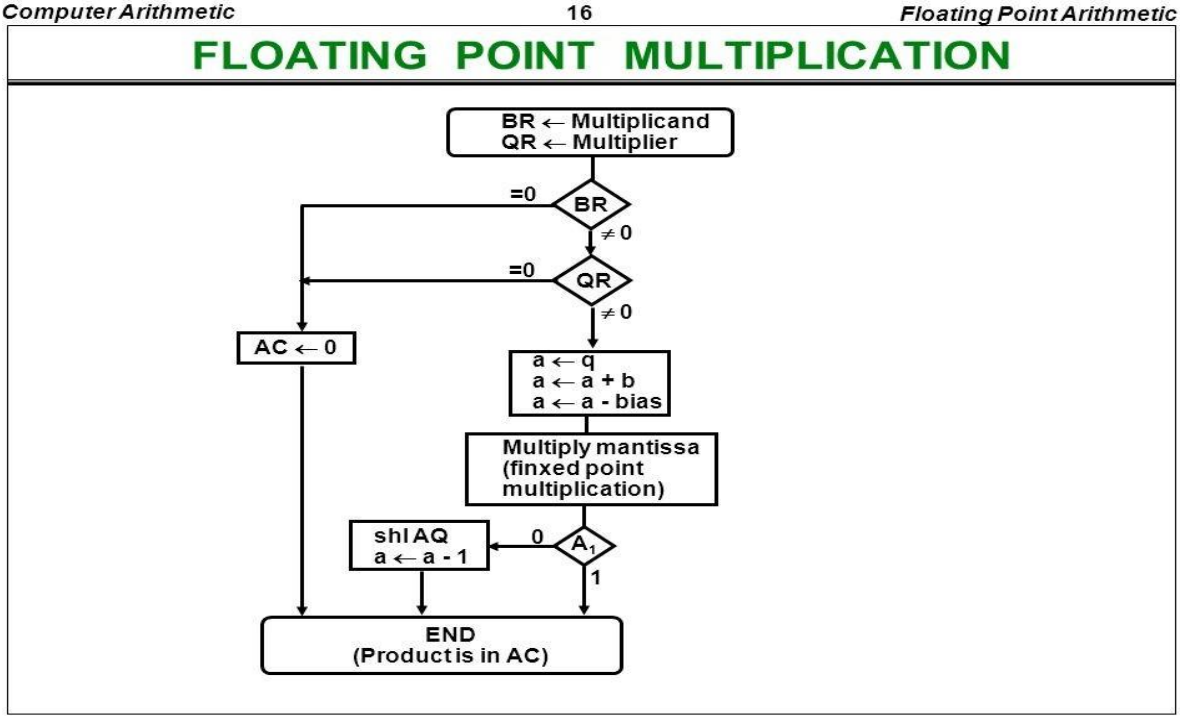
If the magnitudes were subtracted, there may be zero or may have an underflow in the result. If the mantissa is equal to zero the entire floating-point number in the AC is cleared to zero. Otherwise, the mantissa must have at least one bit that is equal to 1. The mantissa has an underflow if the most significant bit in position A1, is 0. In that case, the mantissa is shifted left and the exponent decremented. The bit in A1 is checked again and the process is repeated until A1 = 1. When A1 = 1, the mantissa is normalized and the operation is completed.

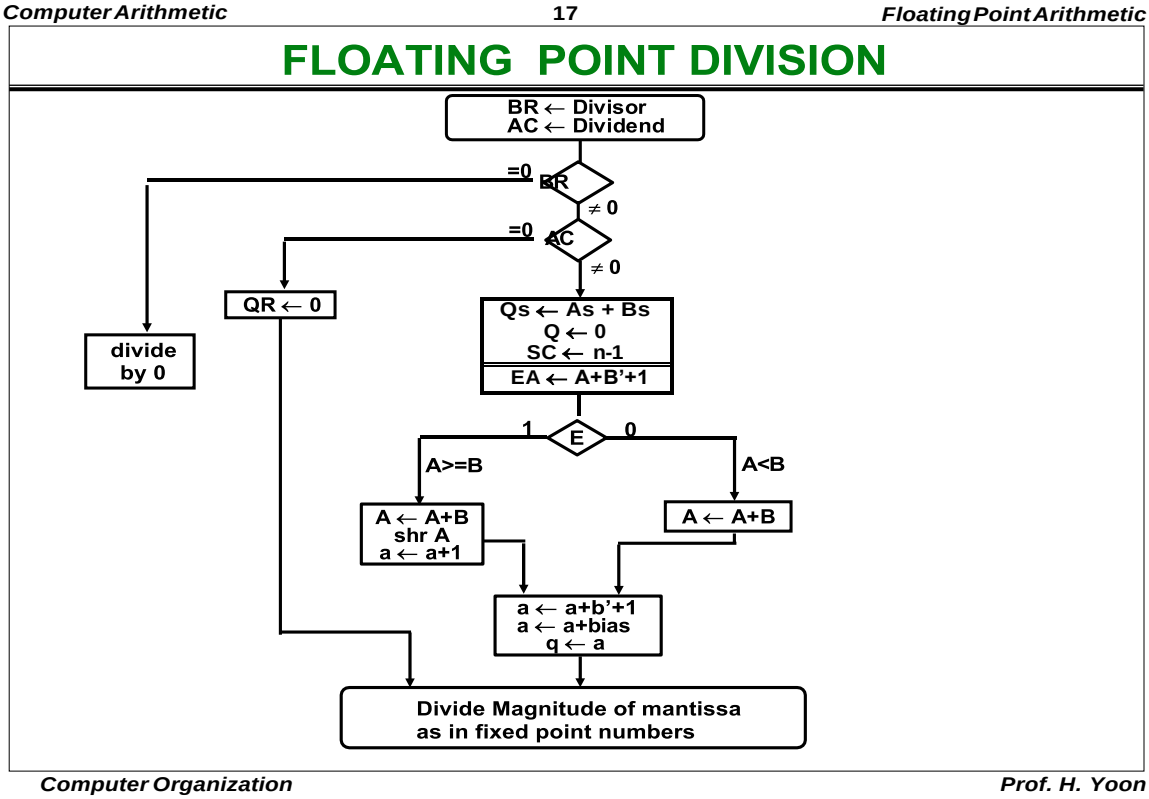




Algorithm for Floating Point Addition and Subtraction

Multiplication:



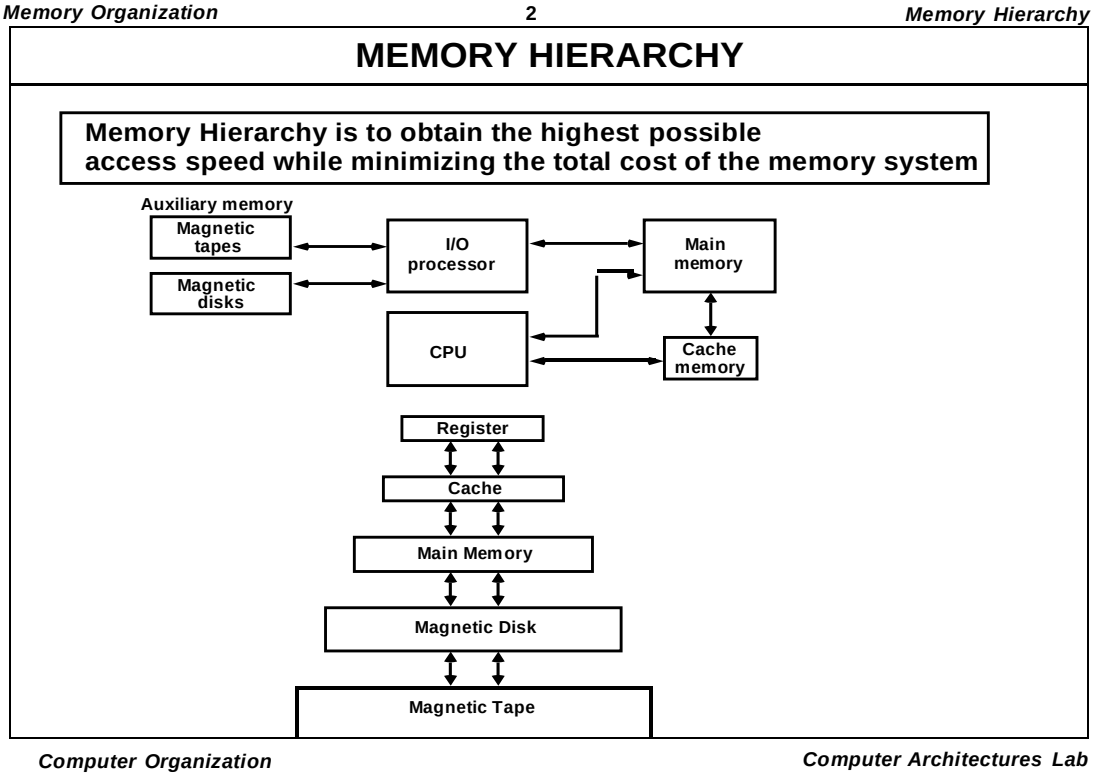


MEMORY ORGANIZATION

: Memory Hierarchy, Main Memory, Auxiliary memory, Associative

Memory, Cache Memory, Virtual Memory

Memory Hierarchy :



memory address map of RAM and ROM.

Main Memory

- The main memory is the central storage unit in a computer system.
- Primary memory holds only those data and instructions on which computer is currently working.
- It has limited capacity and data is lost when power is switched off.
- It is generally made up of semiconductor device.
- These memories are not as fast as registers.
- The data and instruction required to be processed reside in main memory.
- It is divided into two subcategories RAM and ROM.

Memory address map of RAM and ROM

- The designer of a computer system must calculate the amount of memory required for the particular application and assign it to either RAM or ROM.
- The interconnection between memory and processor is then established from knowledge of the size of memory needed and the type of RAM and ROM chips available.
- The addressing of memory can be established by means of a table that specifies the memory address assigned to each chip.
- The table, called a **memory address map**, is a pictorial representation of assigned address space for each chip in the system, shown in table 9.1.

- To demonstrate with a particular example, assume that a computer system needs 512 bytes of RAM and 512 bytes of ROM.
- The RAM and ROM chips to be used are specified in figure 9.1 and figure 9.2.

Memory address map of RAM and ROM

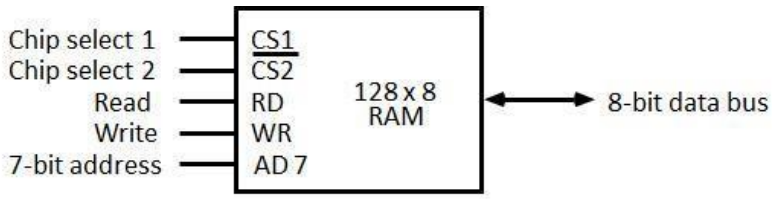


Figure 9.1: Typical RAM chip

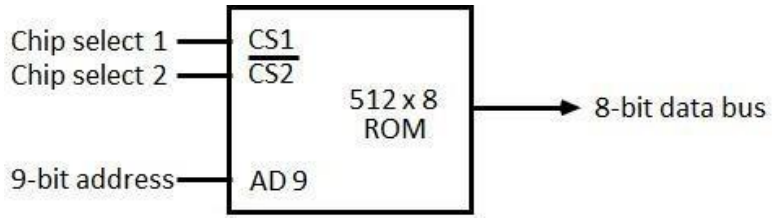


Figure 9.2: Typical ROM chip

Component	Hexa address	Address bus 10 9 8 7 6 5 4 3 2 1
RAM 1	0000 - 007F	0 0 0 x x x x x x x
RAM 2	0080 - 00FF	0 0 1 x x x x x x x
RAM 3	0100 - 017F	0 1 0 x x x x x x x
RAM 4	0180 - 01FF	0 1 1 x x x x x x x
ROM	0200 - 03FF	1 x x x x x x x x x

- The component column specifies whether a RAM or a ROM chip is used.
- The hexadecimal address column assigns a range of hexadecimal equivalent addresses for each chip.
- The address bus lines are listed in the third column.
- Although there are 16 lines in the address bus, the table shows only 10 lines because the other 6 are not used in this example and are assumed to be zero.
- The small x's under the address bus lines designate those lines that must be connected to the address inputs in each chip.
- The RAM chips have 128 bytes and need seven address lines. The ROM chip has 512 bytes and needs 9 address lines.
- The x's are always assigned to the low-order bus lines: lines 1 through 7 for the RAM and lines 1 through 9 for the ROM.

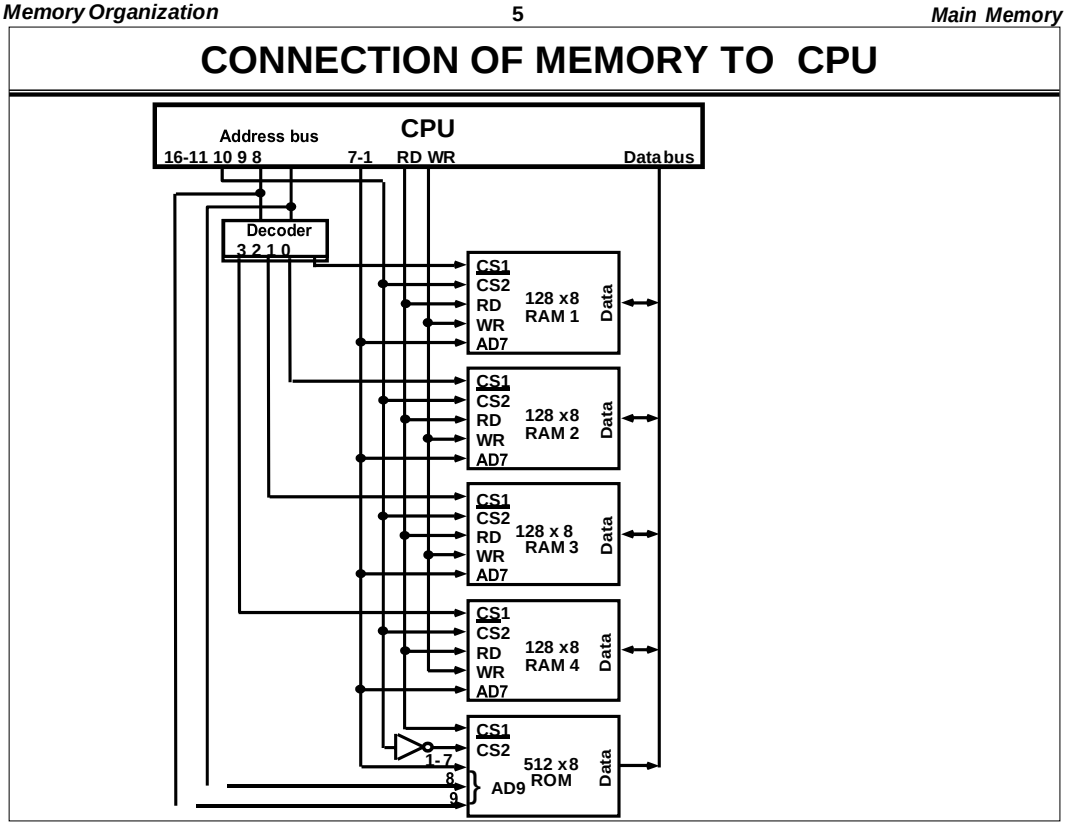
- It is now necessary to distinguish between four RAM chips by assigning to each a different address. For this particular example we choose bus lines 8 and 9 to represent four distinct binary combinations.
- The table clearly shows that the nine low-order bus lines constitute a memory space for RAM equal to $2^9 = 512$ bytes.
- The distinction between a RAM and ROM address is done with another bus line. Here we choose line 10 for this purpose.

When line 10 is 0, the CPU selects a RAM, and when this line is equal to 1, it selects the ROM

Memory connections to CPU :

- RAM and ROM chips are connected to a CPU through the data and address buses

- The low-order lines in the address bus select the byte within the chips and other lines in the address bus select a particular chip through its chip select inputs.



Auxiliary Memory :

- **Magnetic Tape:** Magnetic tapes are used for large computers like mainframe computers where large volume of data is stored for a longer time. In PC also you can use tapes in the form of cassettes. The cost of storing data in tapes is inexpensive. Tapes consist of magnetic materials that store data permanently. It can be 12.5 mm to 25 mm wide plastic film-type and 500 meter to 1200 meter long which is coated with magnetic material. The deck is connected to the central processor and information is fed into or read from the tape through the processor. It's similar to cassette tape recorder.

Magnetic tape is an information storage medium consisting of a magnetisable coating on a thin plastic strip. Nearly all recording tape is of this type, whether used for video with a video cassette recorder, audio storage (reel-to-reel tape, compact audio cassette, digital audio tape (DAT), digital linear tape (DLT) and other formats including 8-track cartridges) or general purpose digital data storage using a computer (specialized tape formats, as well as the above-mentioned compact audio cassette, used with home computers of the 1980s, and DAT, used for backup in workstation installations of the 1990s).

- Magneto-optical and optical tape storage products have been developed using many of the same concepts as magnetic storage, but have achieved little commercial success.

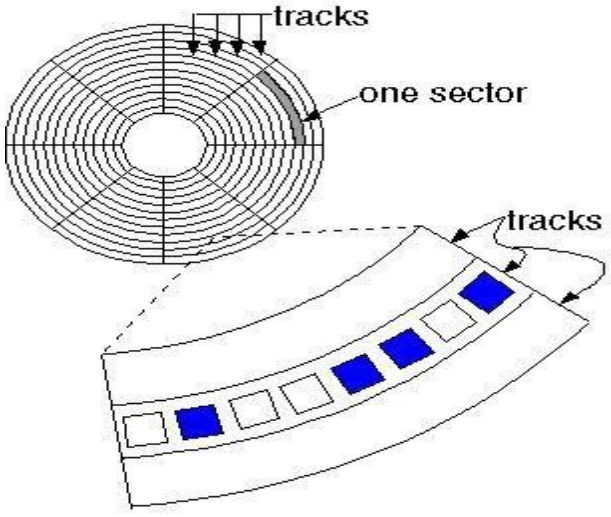
- **Magnetic Disk:** You might have seen the gramophone record, which is circular like a disk and coated with magnetic material. Magnetic disks used in computer are made on the same principle. It rotates with very high speed inside the computer drive. Data is stored on both the surface of the disk. Magnetic disks are most popular for direct access storage device. Each disk consists of a number of invisible concentric circles called tracks. Information is recorded on tracks of a disk surface in the form of tiny magnetic spots. The presence of a magnetic spot represents one bit and its absence represents zero bit. The information stored in a disk can be read many times without affecting the stored data. So the reading operation is non-destructive. But if you want to write a new data, then the existing data is erased from the disk and new data is recorded. For Example-Floppy Disk.

The primary computer storage device. Like tape, it is magnetically recorded and can be re-recorded over and over. Disks are rotating platters with a mechanical arm that moves a read/write head between the outer and inner edges of the platter's surface. It can take as long as one second to find a location on a floppy disk to as little as a couple of milliseconds on a fast hard disk. See hard disk for more details.

The disk surface is divided into concentric tracks (circles within circles). The thinner the tracks, the more storage. The data bits are recorded as tiny magnetic spots on the tracks. The smaller the spot, the more bits per inch and the greater the storage.

Sectors

Tracks are further divided into sectors, which hold a block of data that is read or written at one time; for example, READ SECTOR 782, WRITE SECTOR 5448. In order to update the disk, one or more sectors are read into the computer, changed and written back to disk. The operating system figures out how to fit data into these fixed spaces. Modern disks have more sectors in the outer tracks than the inner ones because the outer radius of the platter is greater than the inner radius



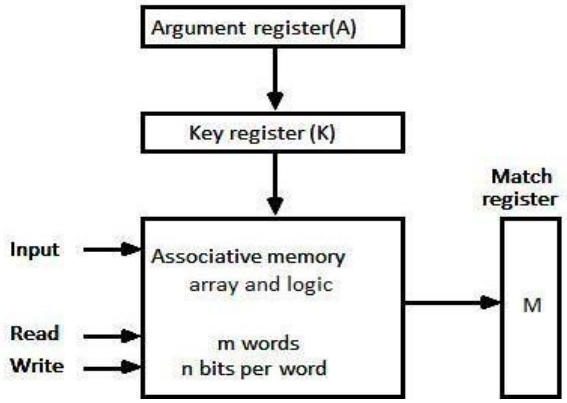
Block diagram of Magnetic Disk

Optical Disk: With every new application and software there is greater demand for memory capacity. It is the necessity to store large volume of data that has led to the development of optical disk storage medium. Optical disks can be divided into the following categories:

1. **Compact Disk/ Read Only Memory (CD-ROM**
2. **Write Once, Read Many (WORM)**
3. **Erasable Optical Disk**

Associative Memory :Content Addressable Memory (CAM).

- The time required to find an item stored in memory can be reduced considerably if stored data can be identified for access by the content of the data itself rather than by an address.
- A memory unit accessed by content is called an associative memory or content addressable memory (CAM).
- This type of memory is accessed simultaneously and in parallel on the basis of data content rather than by specific address or location.
- The block diagram of an associative memory is shown in figure 9.3.



- It consists of a memory array and logic form words with n bits per word.
- The argument register A and key register K each have n bits, one for each bit of a word.
- The match register M has m bits, one for each memory word.
- Each word in memory is compared in parallel with the content of the argument register.
- The words that match the bits of the argument register set a corresponding bit in the match register.
- After the matching process, those bits in the match register that have been set indicate the fact that their corresponding words have been matched.
- Reading is accomplished by a sequential access to memory for those words whose corresponding bits in the match register have been set.

Hardware Organization

- The key register provides a mask for choosing a particular field or key in the argument word.
- The entire argument is compared with each memory word if the key register contains all 1's.

- Otherwise, only those bits in the argument that have 1st in their corresponding position of the key register are compared.
- Thus the key provides a mask or identifying piece of information which specifies how the reference to memory is made.
- To illustrate with a numerical example, suppose that the argument register A and the key register K have the bit configuration shown below.
- Only the three leftmost bits of A are compared with memory words because K has 1's in these position.

A	101 111100	
K	111 000000	
Word1	100 111100	no match
Word2	101 000001	match

- Word 2 matches the unmasked argument field because the three leftmost bits of the argument and the word are equal.

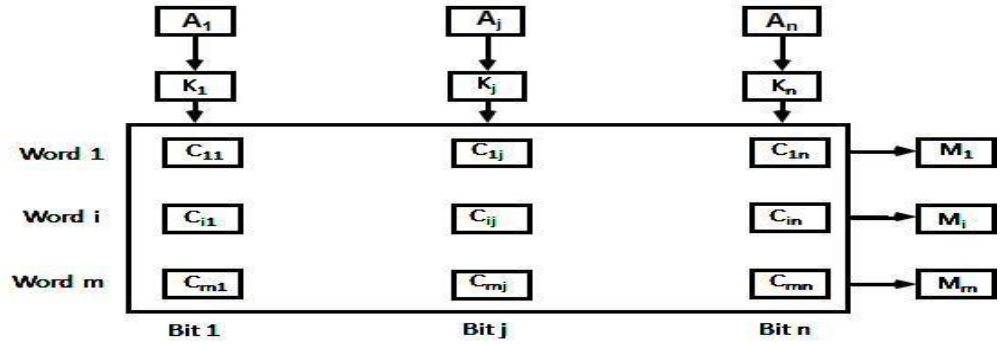


Figure 9.4: Associative memory of *m* word, n cells per word.

- The relation between the memory array and external registers in an associative memory is shown in figure 9.4.
- The cells in the array are marked by the letter C with two subscripts.
- The first subscript gives the word number and the second specifies the bit position in the word.
- Thus cell Cij is the cell for bit j in words i.
- A bit Aj in the argument register is compared with all the bits in column j of the array provided that Kj =1.
- This is done for all columns j = 1, 2... n.
- If a match occurs between all the unmasked bits of the argument and the bits in word i, the corresponding bit Mi in the match register is set to 1.
- If one or more unmasked bits of the argument and the word do not match, Mi is cleared to 0.

Cache Memory :

- Cache is a fast small capacity memory that should hold those information which are most likely to be accessed.

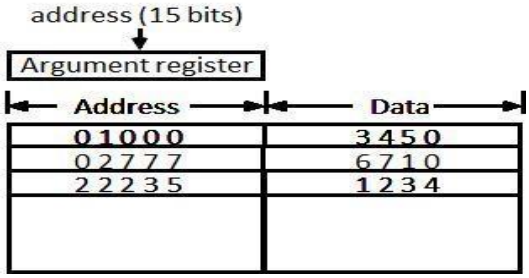
- The basic operation of the cache is, when the CPU needs to access memory, the cache is examined.
- If the word is found in the cache, it is read from the fast memory. If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word.
- The transformation of data from main memory to cache memory is referred to as a **mapping process**.

Associative mapping

- Consider the main memory can store 32K words of 12 bits each.
- The cache is capable of storing 512 of these words at any given time.
- For every word stored in cache, there is a duplicate copy in main memory.
- The CPU communicates with both memories.
- It first sends a 15-bit address to cache. If there is a hit, the CPU accepts the 12-bit data from cache, if there is miss, the CPU reads the word from main memory and the word is then transferred to cache.

Figure 9.5: Associative mapping cache

(all numbers in octal)



- The associative memory stores both the address and content (data) of the memory word.
- This permits any location in cache to store any word from main memory.
- The figure 9.5 shows three words presently stored in the cache. The address value of 15 bits is shown as a five-digit octal number and its corresponding 12-bit word is shown as a four-digit octal number.
- A CPU address of 15 bits is placed in the argument register and the associative memory is searched for a matching address.
- If the address is found the corresponding 12-bit data is read and sent to CPU.
- If no match occurs, the main memory is accessed for the word.
- The address data pairs then transferred to the associative cache memory.
- If the cache is full, an address data pair must be displaced to make room for a pair that is needed and not presently in the cache.
- This constitutes a first-in first-one (FIFO) replacement policy.

direct mapping in organization of cache memory:

- The CPU address of 15 bits is divided into two fields.
- The nine least significant bits constitute the index field and the remaining six bits from the tag field.
- The figure 9.6 shows that main memory needs an address that includes both the tag and the index.

Figure 9.6: Addressing relationships between main and cache memories

- The number of bits in the index field is equal to the number of address bits required to access the cache memory.
- The internal organization of the words in the cache memory is as shown in figure 9.7.

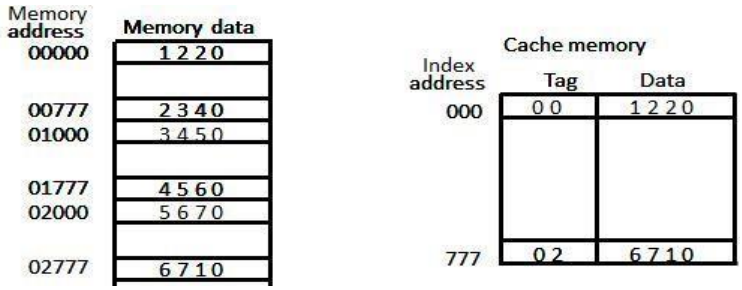


Figure 9.7: Direct mapping cache organization

- Each word in cache consists of the data word and its associated tag.
- When a new word is first brought into the cache, the tag bits are stored alongside the data bits.
- When the CPU generates a memory request the index field is used for the address to access the cache.
- The tag field of the CPU address is compared with the tag in the word read from the cache.
- If the two tags match, there is a hit and the desired data word is in cache.
- If there is no match, there is a miss and the required word is read from main memory.
- It is then stored in the cache together with the new tag, replacing the previous value.
- The word at address zero is presently stored in the cache (index = 000, tag = 00, data = 1220). Suppose that the CPU now wants to access the word at address 02000.

  The index address is 000, so it is used to access the cache. The two tags are then compared.

  The cache tag is 00 but the address tag is 02, which does not produce a match.

  Therefore, the main memory is accessed and the data word 5670 is transferred to the CPU.

  The cache word at index address 000 is then replaced with a tag of 02 and data of 5670.

  The **disadvantage** of direct mapping is that two words with the same index in their address but with different tag values cannot reside in cache memory at the same time.

The comparison logic is done by an associative search of the tags in the set similar to an associative memory search: thus the name "set-associative".

When a miss occurs in a set-associative cache and the set is full, it is necessary to replace one of the tag-data items with a new value.

The most common replacement algorithms used are: random replacement, first-in first-out (FIFO), and least recently used (LRU).

Write-through and Write-back cache write method.

Write Through

- The simplest and most commonly used procedure is to update main memory with every memory write operation.
- The cache memory being updated in parallel if it contains the word at the specified address. This is called the *write-through* method.
- This method has the advantage that main memory always contains the same data as the cache.
- This characteristic is important in systems with direct memory access transfers.
- It ensures that the data residing in main memory are valid at all times so that an I/O device communicating through DMA would receive the most recent updated data.

Write-Back (Copy-Back)

- The second procedure is called the write-back method.
- In this method only the cache location is updated during a write operation.
- The location is then marked by a flag so that later when the word is removed from the cache it is copied into main memory.
- The reason for the write-back method is that during the time a word resides in the cache, it may be updated several times.
- However, as long as the word remains in the cache, it does not matter whether the copy in main memory is out of date, since requests from the word are filled from the cache.
- It is only when the word is displaced from the cache that an accurate copy need be rewritten into main memory.

Virtual Memory

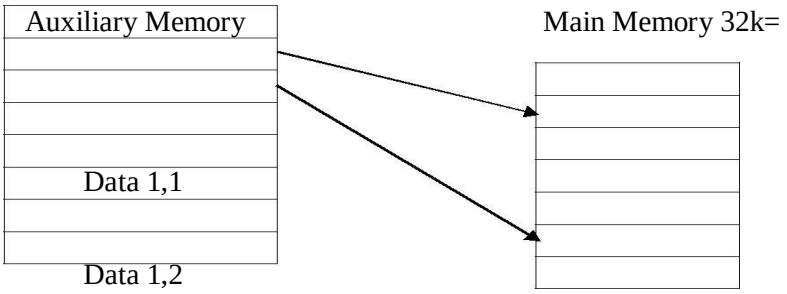
- Virtual memory is used to give programmers the illusion that they have a very large memory at their disposal, even though the computer actually has a relatively small main memory.
- A virtual memory system provides a mechanism for translating program-generated addresses into correct main memory locations.

Address space

An address used by a programmer will be called a virtual address, and the set of such addresses is known as address space.

Memory space

An address in main memory is called a location or physical address. The set of such locations is called the memory space.



Address space $1024k=2^{10}$

- As an illustration, consider a computer with a main-memory capacity of 32K words (K = 1024). Fifteen bits are needed to specify a physical address in memory since $32K = 2^{15}$.
- Suppose that the computer has available auxiliary memory for storing $2^{20} = 1024K$ words.
- Thus auxiliary memory has a capacity for storing information equivalent to the capacity of 32 main memories.
- Denoting the address space by N and the memory space by M, we then have for this example N = 1024K and M = 32K.
- In a multiprogramming computer system, programs and data are transferred to and from auxiliary memory and main memory based on demands imposed by the CPU.
- Suppose that program 1 is currently being executed in the CPU. Program 1 and a portion of its associated data are moved from auxiliary memory into main memory as shown in figure 9.9.
- Portions of programs and data need not be in contiguous locations in memory since information is being moved in and out, and empty spaces may be available in scattered locations in memory.
- In our example, the address field of an instruction code will consist of 20 bits but physical memory addresses must be specified with only 15 bits.
- Thus CPU will reference instructions and data with a 20-bit address, but the information at this address must be taken from physical memory because access to auxiliary storage for individual words will be too long.

Address mapping using pages.

- AThe table implementation of the address mapping is simplified if the information in the address space and the memory space are each divided into groups of fixed size.
- The physical memory is broken down into groups of equal size called blocks, which may range from 64 to 4096 words each.
- The term page refers to groups of address space of the same size.
- Consider a computer with an address space of 8K and a memory space of 4K.
- If we split each into groups of 1K words we obtain eight pages and four blocks as shown in figure 9.9
- At any given time, up to four pages of address space may reside in main memory in any one of the four blocks.

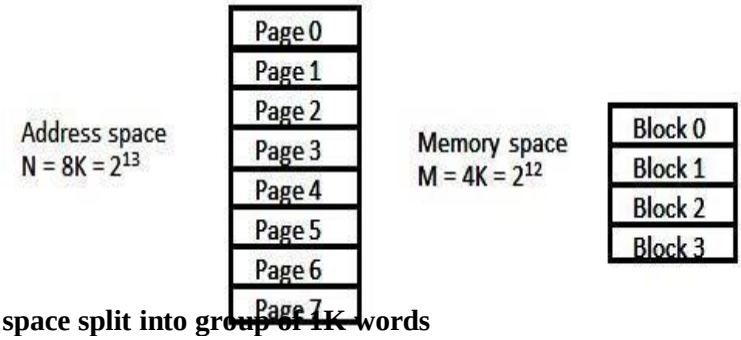


Figure 9.10 Address and Memory space split into group of 1K words

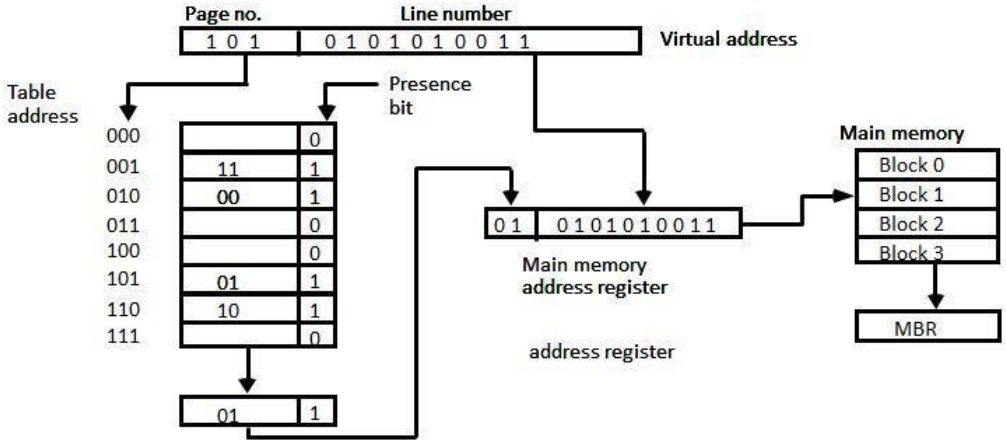


Figure 9.11: Memory table in paged system

- The organization of the memory mapping table in a paged system is shown in figure 9.10.
- The memory-page table consists of eight words, one for each page.
- The address in the page table denotes the page number and the content of the word give the block number where that page is stored in main memory.
- The table shows that pages 1, 2, 5, and 6 are now available in main memory in blocks 3, 0, 1, and 2, respectively.
- A presence bit in each location indicates whether the page has been transferred from auxiliary memory into main memory.
- A 0 in the presence bit indicates that this page is not available in main memory.

The CPU references a word in memory with a virtual address of 13 bits.

The three high-order bits of the virtual address specify a page number and also an address for the memory-page table.

The content of the word in the memory page table at the page number address is read out into the memory table buffer register.

If the presence bit is a 1, the block number thus read is transferred to the two high-order bits of the main memory address register.

The line number from the virtual address is transferred into the 10 low-order bits of the memory address register.

A read signal to main memory transfers the content of the word to the main memory buffer register ready to be used by the CPU.

If the presence bit in the word read from the page table is 0, it signifies that the content of the word referenced by the virtual address does not reside in main memory.

Segment

A segment is a set of logically related instructions or data elements associated with a given name.

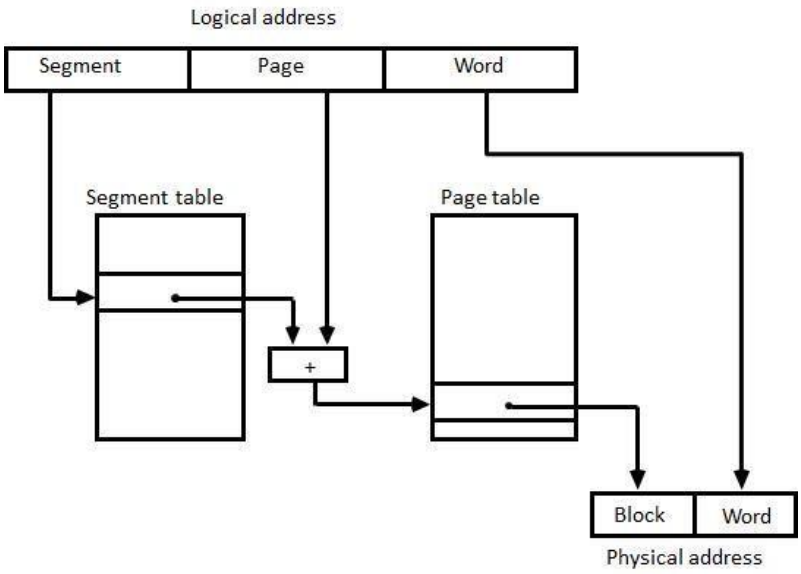
Logical address

The address generated by segmented program is called a logical address.

Segmented page mapping

- The length of each segment is allowed to grow and contract according to the needs of the program being executed. Consider logical address shown in figure 9.12.

Figure 9.12: Logical to physical address mapping



- The logical address is partitioned into three fields.
- The segment field specifies a segment number.
- The page field specifies the page within the segment and word field gives specific word within the page.
- A page field of k bits can specify up to 2k pages.
- A segment number may be associated with just one page or with as many as 2k pages.
- Thus the length of a segment would vary according to the number of pages that are assigned to it.
- The mapping of the logical address into a physical address is done by means of two tables, as shown in figure 9.12.
- The segment number of the logical address specifies the address for the segment table.
- The entry in the segment table is a pointer address for a page table base.
- The page table base is added to the page number given in the logical address.
- The sum produces a pointer address to an entry in the page table.

- The concatenation of the block field with the word field produces the final physical mapped address.
- The two mapping tables may be stored in two separate small memories or in main memory.
- In either case, memory reference from the CPU will require three accesses to memory: one from the segment table, one from the page table and the third from main memory.
- This would slow the system significantly when compared to a conventional system that requires only one reference to memory.

REFERENCE :

1. COMPUTER SYSTEM ARCHITECTURE, MORRIS M. MANO, 3RD EDITION, PRENTICE HALL INDIA.

UNIT-III PROCESSING UNIT

The part of the computer that performs the bulk of data processing operations is called the Central Processing Unit (CPU) and is the central component of a digital computer. Its purpose is to interpret instruction cycles received from memory and perform arithmetic, logic and control operations with data stored in internal register, memory words and I/O interface units. A CPU is usually divided into two parts namely processor unit (Register Unit and Arithmetic Logic Unit) and control unit.

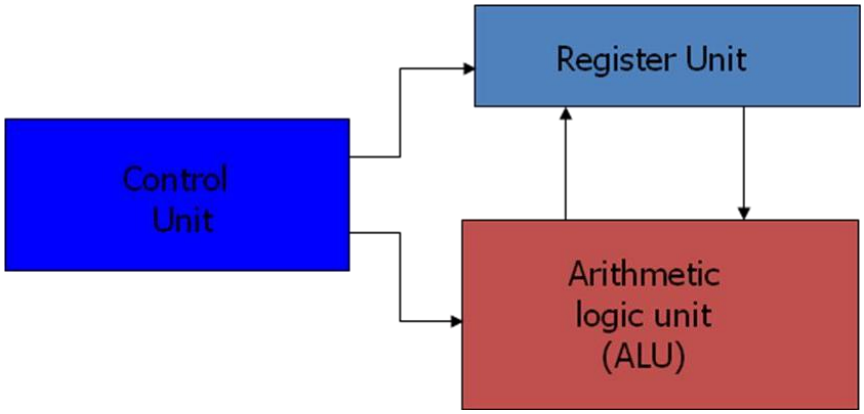


Fig: Components of CPU

Processor Unit:

The processor unit consists of arithmetic unit, logic unit, a number of registers and internal buses that provides data path for transfer of information between register and arithmetic logic unit. The block diagram of processor unit is shown in figure below where all registers are connected through common buses. The registers communicate each other not only for direct data transfer but also while performing various micro-operations.

Here two sets of multiplexers select register which perform input data for ALU. A decoder selects destination register by enabling its load input. The function select in ALU determines the particular operation that to be performed.

For an example to perform the operation: $R_3 \leftarrow R_1 + R_2$

1. MUX A selector (SELA): to place the content of R_1 into bus A.
2. MUX B selector (SELB): to place the content of R_2 into bus B.
3. ALU operation selector (OPR): to provide arithmetic addition A + B.
4. Decoder destination selector (SELD): to transfer the content of the output bus into R_3.

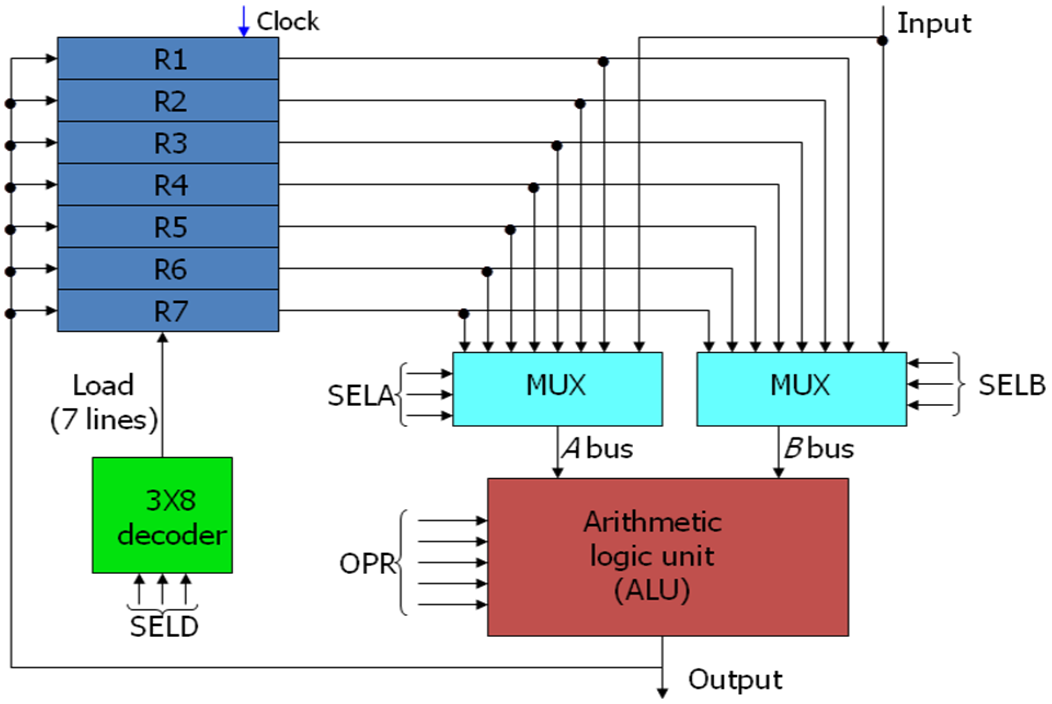


Fig: Processor Unit

Control unit:

The control unit is the heart of CPU. It consists of a program counter, instruction register, timing and control logic. The control logic may be either hardwired or micro-programmed. If it is a hardwired, register decodes and a set of gates are connected to provide the logic that determines the action required to execute various instructions. A micro-programmed control unit uses a control memory to store micro instructions and a sequence to determine the order by which the instructions are read from control memory.

The control unit decides what the instructions mean and directs the necessary data to be moved from memory to ALU. Control unit must communicate with both ALU and main memory and coordinates all activities of processor unit, peripheral devices and storage devices. It can be characterized on the basis of design and implementation by:

- Defining basic elements of the processor
- Describing the micro-operation that processor performs
- Determining the function that the control unit must perform to cause the micro-operations to be performed.

Control unit must have inputs that allow determining the state of system and outputs that allow controlling the behavior of system.

The input to control unit are:

- Flag: flags are headed to determine the status of processor and outcome of previous ALU operation.

- Clock: All micro-operations are performed within each clock pulse. This clock pulse is also called as processor cycle time or clock cycle time.
- Instruction Register: The op-code of instruction determines which micro-operation to perform during execution cycle.
- Control signal from control bus: The control bus portion of system bus provides interrupt, acknowledgement signals to control unit.

The outputs from control unit are:

- Control signal within processor: These signals causes data transfer between registers, activate ALU functions.
- Control signal to control bus: These are signals to memory and I/O module. All these control signals are applied directly as binary inputs to individual logic gate.

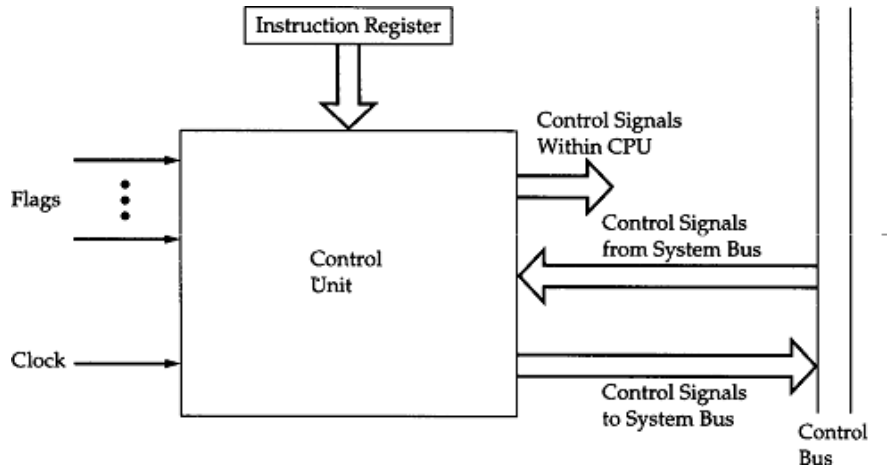


Fig: Control Unit

2.1 CPU Structure and Function

Processor Organization

- Things a CPU must do:
  - Fetch Instructions
  - Interpret Instructions
  - Fetch Data
  - Process Data
  - Write Data

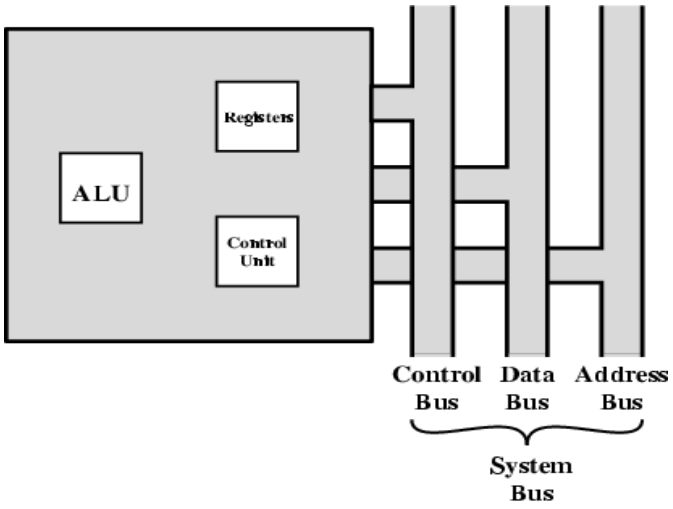


Fig: The CPU with the System Bus

- A small amount of internal memory, called the registers, is needed by the CPU to fulfill these requirements

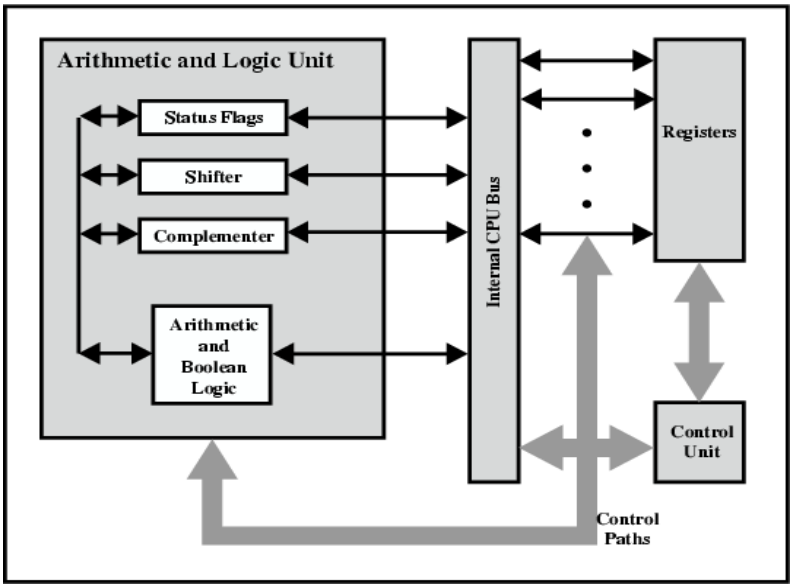


Fig: Internal Structure of the CPU

- Components of the CPU
  - Arithmetic and Logic Unit (ALU): does the actual computation or processing of data
  - Control Unit (CU): controls the movement of data and instructions into and out of the CPU and controls the operation of the ALU.

Register Organization

- Registers are at top of the memory hierarchy. They serve two functions:
  1. User-Visible Registers - enable the machine- or assembly-language programmer to minimize main-memory references by optimizing use of registers
  2. Control and Status Registers - used by the control unit to control the operation

of the CPU and by privileged, OS programs to control the execution of programs

User-Visible Registers

Categories of Use

- General Purpose registers - for variety of functions
- Data registers - hold data
- Address registers - hold address information
- Segment pointers - hold base address of the segment in use
- Index registers - used for indexed addressing and may be auto indexed
- Stack Pointer - a dedicated register that points to top of a stack. Push, pop, and other stack instructions need not contain an explicit stack operand.
- Condition Codes (flags)

Design Issues

- Completely general-purpose registers or specialized use?
  - Specialized registers save bits in instructions because their use can be implicit
  - General-purpose registers are more flexible
  - Trend is toward use of specialized registers
- Number of registers provided?
  - More registers require more operand specifier bits in instructions
  - 8 to 32 registers appears optimum (RISC systems use hundreds, but are a completely different approach)
- Register Length?
  - Address registers must be long enough to hold the largest address
  - Data registers should be able to hold values of most data types
  - Some machines allow two contiguous registers for double-length values
- Automatic or manual save of condition codes?
  - Condition restore is usually automatic upon call return
  - Saving condition code registers may be automatic upon call instruction, or may be manual

Control and Status Registers

- Essential to instruction execution
  - Program Counter (PC)
  - Instruction Register (IR)
  - Memory Address Register (MAR) - usually connected directly to address lines of bus
  - Memory Buffer Register (MBR) - usually connected directly to data lines of bus
- Program Status Word (PSW) - also essential, common fields or flags contained include:
  - Sign - sign bit of last arithmetic operation
  - Zero - set when result of last arithmetic operation is 0
  - Carry - set if last op resulted in a carry into or borrow out of a high-order bit
  - Equal - set if a logical compare result is equality
  - Overflow - set when last arithmetic operation caused overflow
  - Interrupt Enable/Disable - used to enable or disable interrupts
  - Supervisor - indicates if privileged ops can be used

- Other optional registers
  - Pointer to a block of memory containing additional status info (like process control blocks)
  - An interrupt vector
  - A system stack pointer
  - A page table pointer
  - I/O registers
- Design issues
  - Operating system support in CPU
  - How to divide allocation of control information between CPU registers and first part of main memory (usual tradeoffs apply)

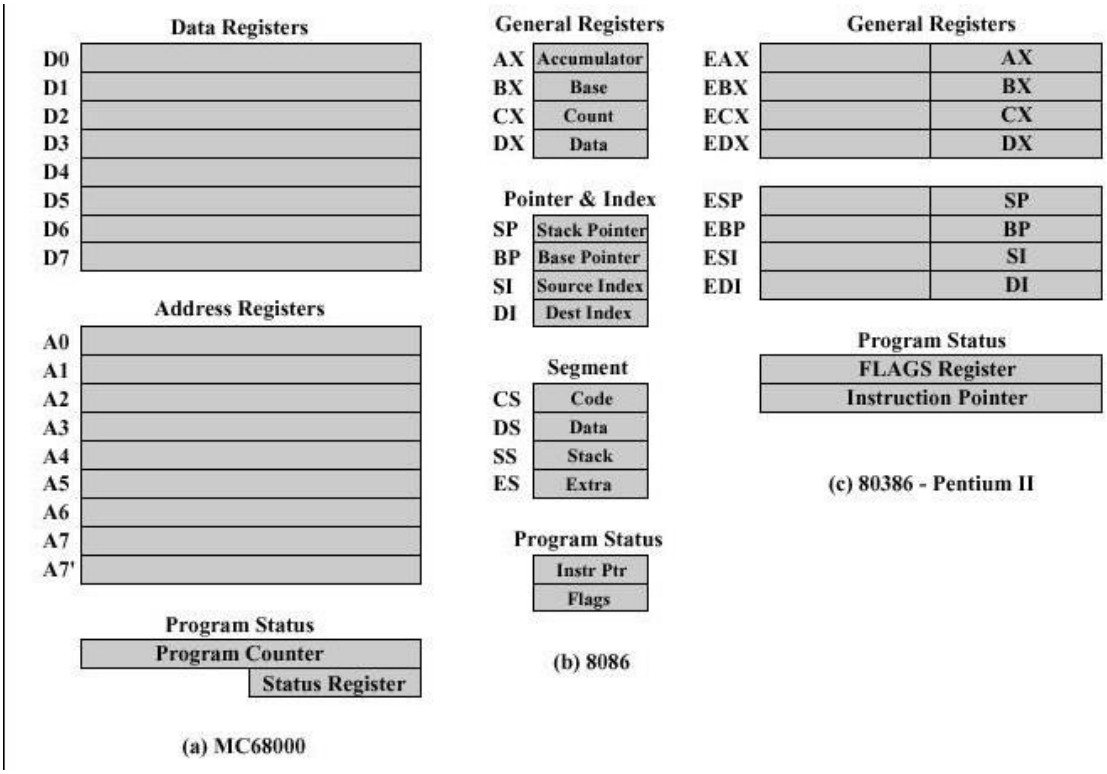


Fig: Example Microprocessor Register Organization

The Instruction Cycle

Basic instruction cycle contains the following sub-cycles.

- Fetch - read next instruction from memory into CPU
- Execute - Interpret the opcode and perform the indicated operation
- Interrupt - if interrupts are enabled and one has occurred, save the current process state and service the interrupt

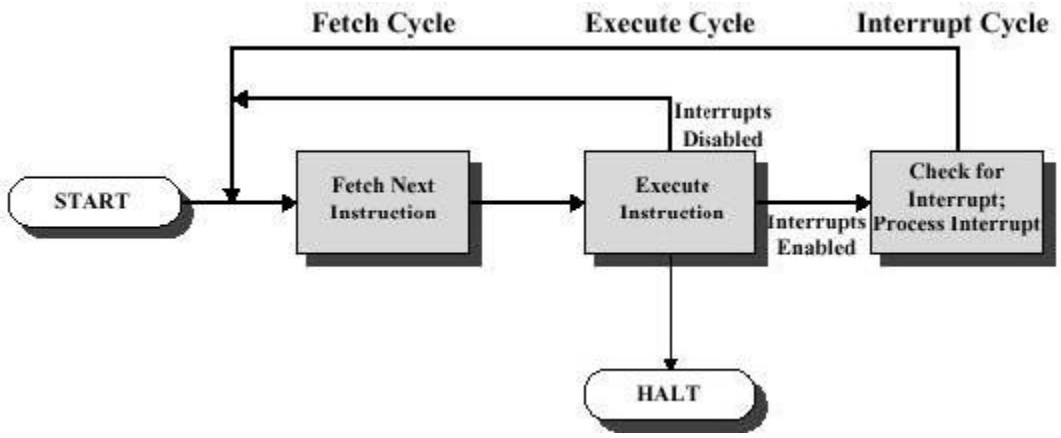


Fig: Instruction Cycles

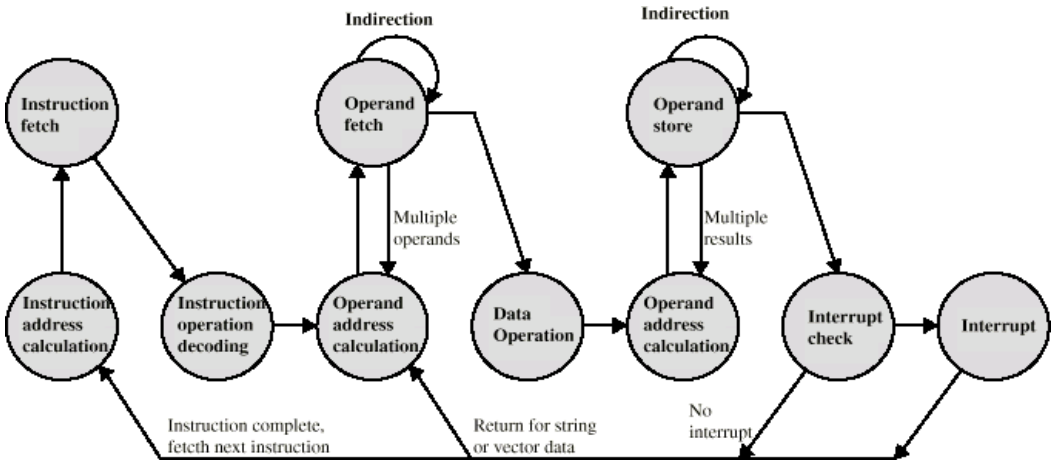


Fig: Instruction Cycle State Diagram

The Indirect Cycle

- Think of as another instruction sub-cycle
- May require just another fetch (based upon last fetch)
- Might also require arithmetic, like indexing

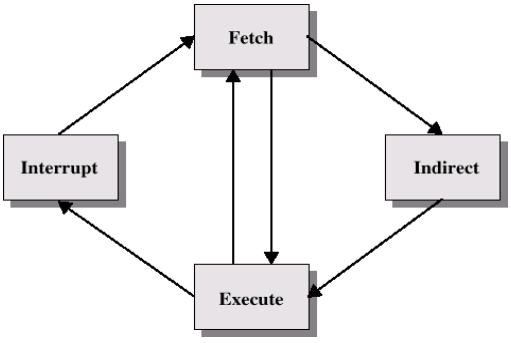


Fig: Instruction Cycle with Indirect

Data Flow

- Exact sequence depends on CPU design
- We can indicate sequence in general terms, assuming CPU employs:
    - a memory address register (MAR)
    - a memory buffer register (MBR)
    - a program counter (PC)
    - an instruction register (IR)

Fetch cycle data flow

- PC contains address of next instruction to be fetched
- This address is moved to MAR and placed on address bus
- Control unit requests a memory read
- Result is
    - placed on data bus
    - result copied to MBR
    - then moved to IR
- Meanwhile, PC is incremented

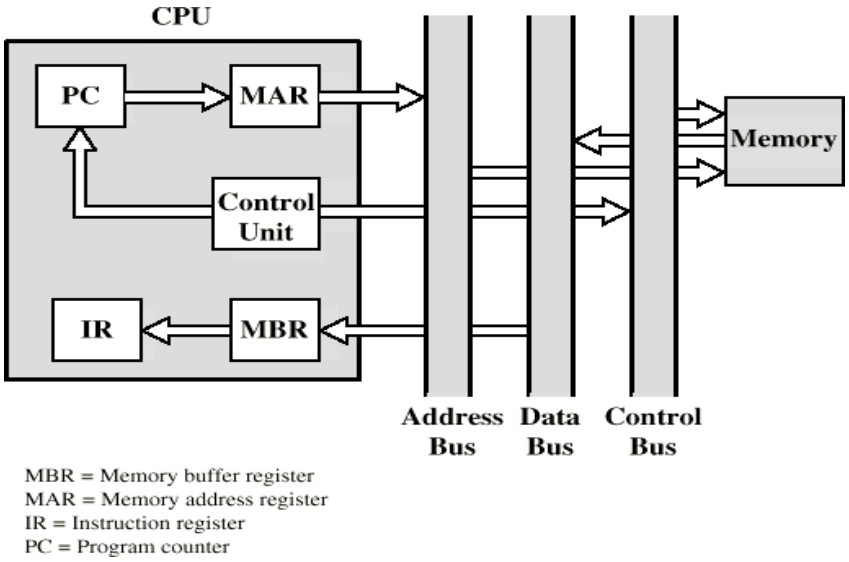


Fig: Data flow, Fetch Cycle

t1: MAR ← (PC)
t2: MBR ← Memory
PC ← PC + 1
t3: IR(Address) ← (MBR(Address))

Indirect cycle data flow

- Decodes the instruction
- After fetch, control unit examines IR to see if indirect addressing is being used. If so:
- Rightmost n bits of MBR (the memory reference) are transferred to MAR
- Control unit requests a memory read, to get the desired operand address into the MBR

t1: MAR ← (IR(Address))
t2: MBR ← Memory
t3: IR(Address) ← (MBR(Address))

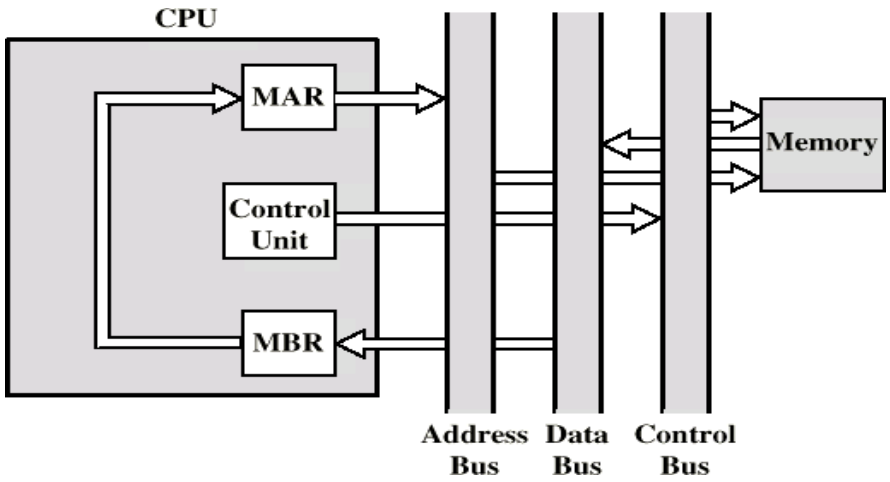


Fig: Data Flow, Indirect Cycle

Execute cycle data flow

- Not simple and predictable, like other cycles
- Takes many forms, since form depends on which of the various machine instructions is in the IR
- May involve
    - transferring data among registers
    - read or write from memory or I/O
    - invocation of the ALU

For example: ADD R_1, X
t1: MAR ← (IR(Address))
t2: MBR ← Memory
t3: R_1 ← (R_1) + (MBR)

Interrupt cycle data flow

- Current contents of PC must be saved (for resume after interrupt), so PC is transferred to MBR to be written to memory
- Save location's address (such as a stack ptr) is loaded into MAR from the control unit
- PC is loaded with address of interrupt routine (so next instruction cycle will begin by fetching appropriate instruction)

t1: MBR ← (PC)
t2: MAR ← save_address
PC ← Routine_address
t3: Memory ← (MBR)

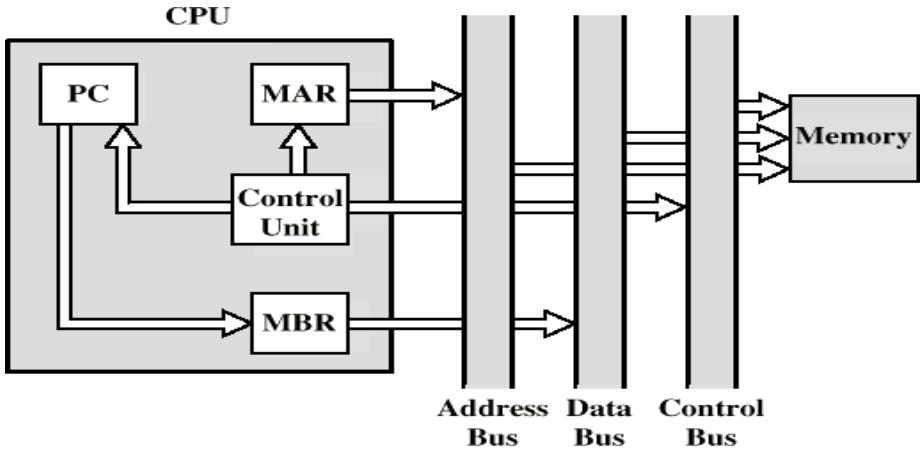
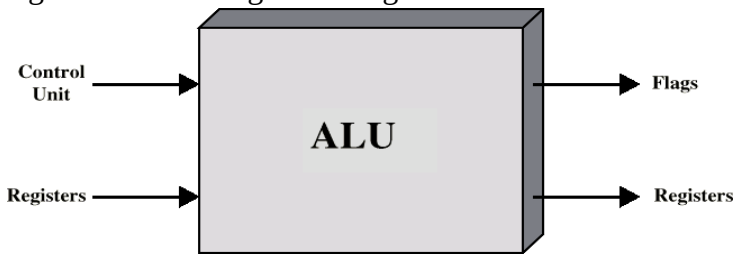


Fig: Data Flow, Interrupt Cycle

2.2 Arithmetic and Logic Unit

ALU is the combinational circuit of that part of computer that actually performs arithmetic and logical operations on data. All of the other elements of computer system- control unit, registers, memory, I/O are their mainly to bring data into the ALU for it to process and then to take the result back out. An ALU & indeed all electronic components in computer are based on the use of simple digital logic device that can store binary digit and perform simple Boolean logic function. Figure indicates in general in general term how ALU is interconnected with rest of the processor.



Data are presented to ALU in register and the result of operation is stored in register. These registers are temporarily storage location within the processor that are connected by signal path to the ALU. The ALU may also set flags as the result of an operation. The flags values are also stored in registers within the processor. The control unit provides signals that control the operation of ALU and the movement of data into an out of ALU.

The design of ALU has three stages.

1. **Design the arithmetic section**
   The basic component of arithmetic circuit is a parallel adder which is constructed with a number of full adder circuits connected in cascade. By controlling the data inputs to the parallel adder, it is possible to obtain different types of arithmetic operations. Below figure shows the arithmetic circuit and its functional table.

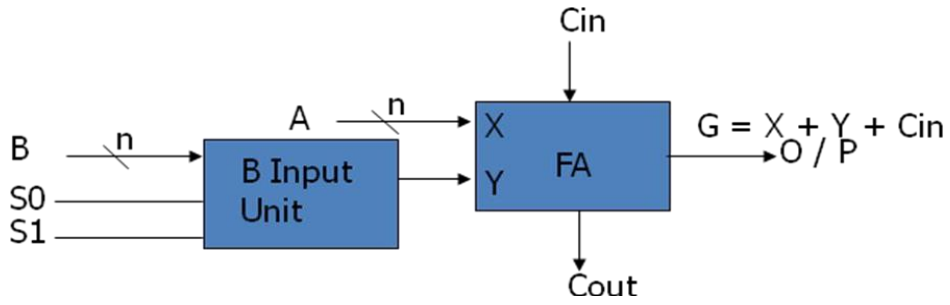


Fig: Block diagram of Arithmetic Unit

Functional table for arithmetic unit:

Select		Input	Output		Microoperation	
S_1	S_0	Y	Cin = 0	Cin = 1	Cin = 0	Cin = 1
0	0	0	A	A+1	Transfer A	Increment A
0	1	B	A+B	A+B+1	Addition	Addition with carry
1	0	B'	A+B'	A+B'+1	Subtraction with borrow	Subtraction
1	1	-1	A-1	A	Decrement A	Transfer A

2. Design the logical section

The basic components of logical circuit are AND, OR, XOR and NOT gate circuits connected accordingly. Below figure shows a circuit that generates four basic logic micro-operations. It consists of four gates and a multiplexer. Each of four logic operations is generated through a gate that performs the required logic. The two selection input S1 and S0 choose one of the data inputs of the multiplexer and directs its value to the output. Functional table lists the logic operations.

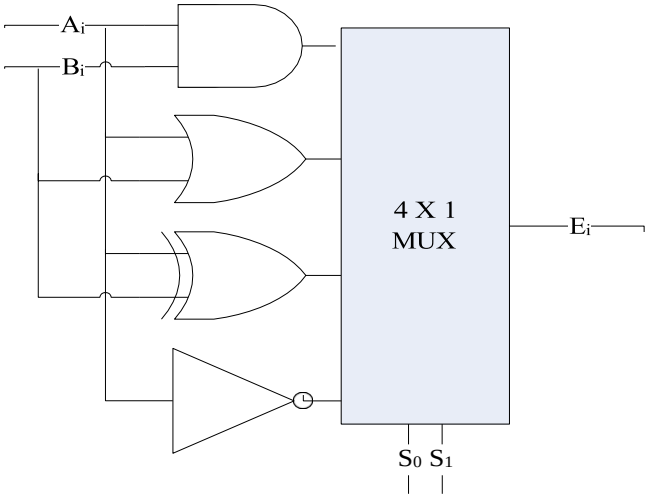


Fig: Block diagram of Logic Unit

Functional table for logic unit:

S1	S0	output	Microoperation
0	0	Ai && Bi	AND
0	1	Ai || Bi	OR
1	0	Ai XOR Bi	XOR
1	1	Ai'	NOT

3. Combine these 2 sections to form the ALU

Below figure shows a combined circuit of ALU where n data input from A are combined with n data input from B to generate the result of an operation at the G output line. ALU has a number of selection lines used to determine the operation to be performed. The selection lines are decoded with the ALU so that selection lines can specify distinct operations. The mode select S_2 differentiate between arithmetic and logical operations. The two functions select S_1 and S_0 specify the particular arithmetic and logic operations to be performed. With three selection lines, it is possible to specify arithmetic operation with S_2 at 0 and logical operation with S_2 at 1.

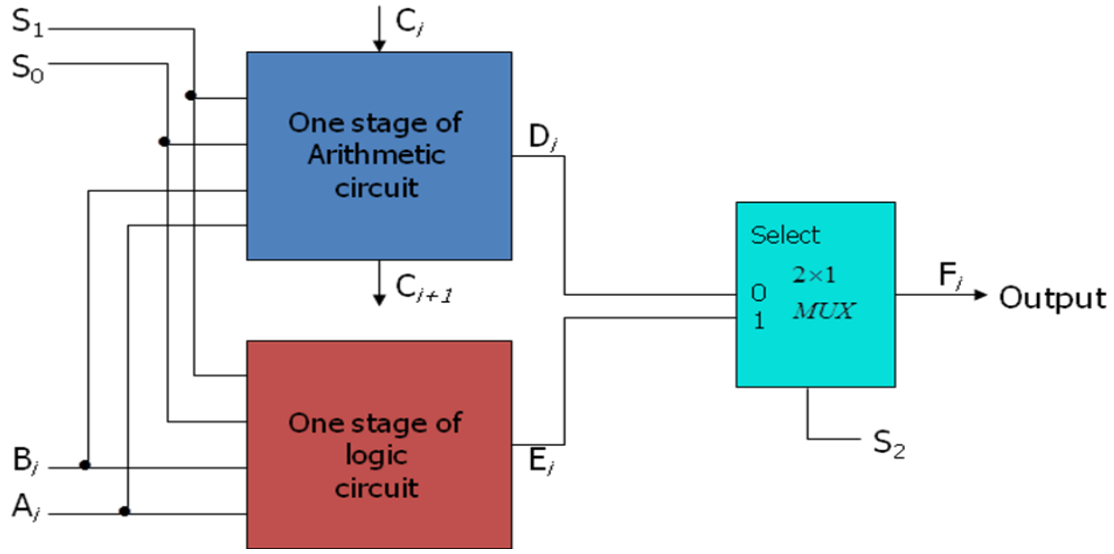
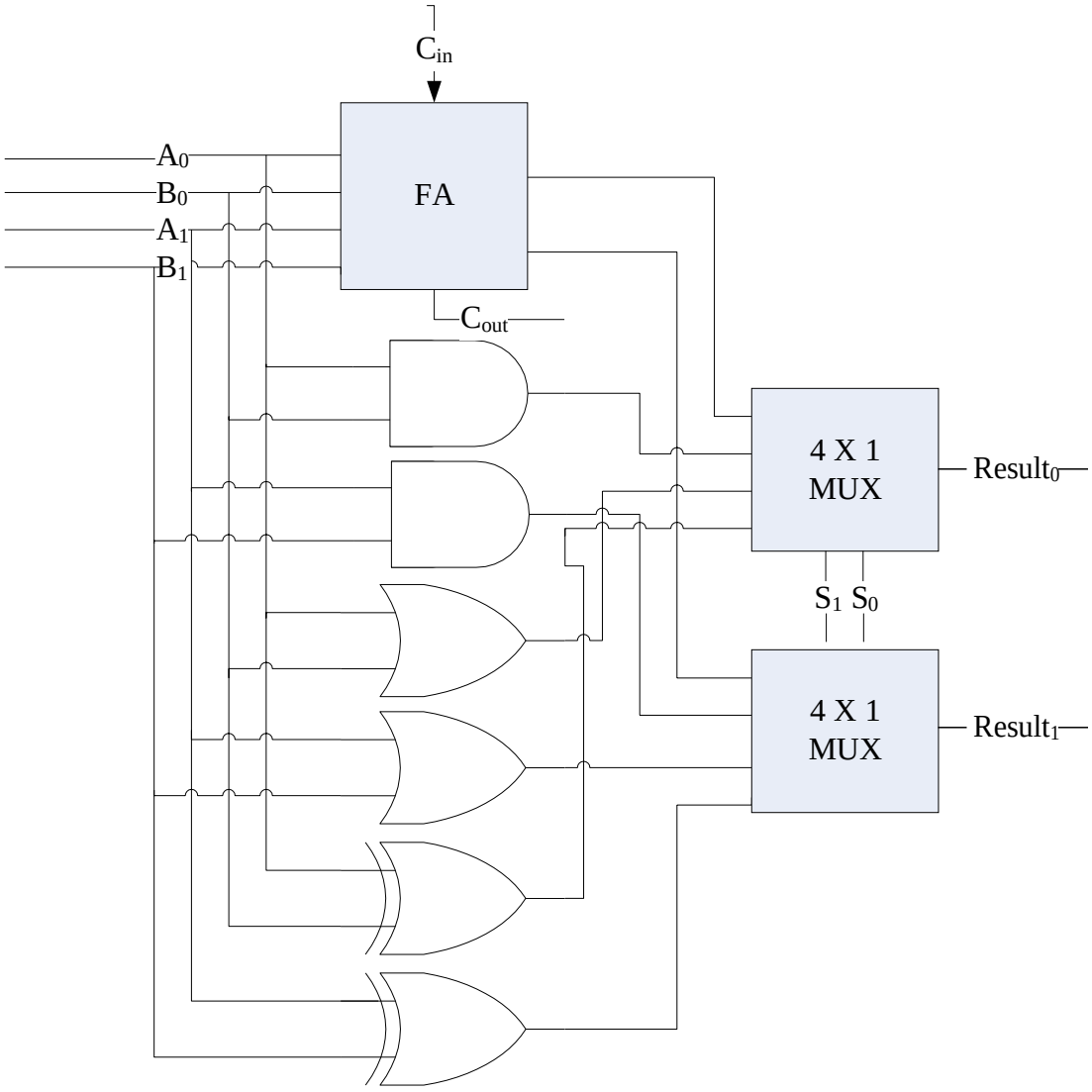


Fig: Block diagram of ALU

Example: Design a 2-bit ALU that can perform addition, AND, OR, & XOR.



2.3 Instruction Formats

The computer can be used to perform a specific task, only by specifying the necessary steps to complete the task. The collection of such ordered steps forms a 'program' of a computer. These ordered steps are the instructions. Computer instructions are stored in central memory locations and are executed sequentially one at a time. The control reads an instruction from a specific address in memory and executes it. It then continues by reading the next instruction in sequence and executes it until the completion of the program.

A computer usually has a variety of Instruction Code Formats. It is the function of the control unit within the CPU to interpret each instruction code and provide the necessary control functions needed to process the instruction. An n bit instruction that k bits in the address field and m bits in the operation code field come addressed 2^k location directly and specify 2^m different operation.

- The bits of the instruction are divided into groups called fields.
- The most common fields in instruction formats are:
  - An **Operation code** field that specifies the operation to be performed.
  - An **Address field** that designates a memory address or a processor register.
  - A **Mode field** that specifies the way the operand or the effective address is determined.

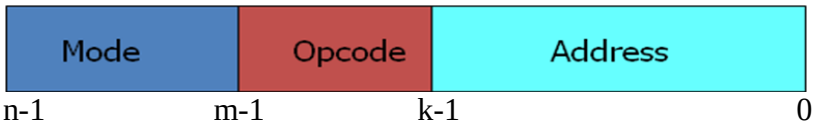


Fig: Instruction format with mode field

The operation code field (Opcode) of an instruction is a group of bits that define various processor operations such as add, subtract, complement, shift etcetera. The bits that define the mode field of an instruction code specify a variety of alternatives for choosing the operands from the given address. Operation specified by an instruction is executed on some data stored in the processor register or in the memory location. Operands residing in memory are specified by their memory address. Operands residing in processor register are specified with a register address.

Types of Instruction

- Computers may have instructions of several different lengths containing varying number of addresses.
- The number of address fields in the instruction format of a computer depends on the internal organization of its registers.
- Most computers fall into one of 3 types of CPU organizations:

Single accumulator organization:- All the operations are performed with an accumulator register. The instruction format in this type of computer uses one address field. For example: ADD X, where X is the address of the operands .

General register organization:- The instruction format in this type of computer needs three register address fields. For example: ADD R1,R2,R3

Stack organization:- The instruction in a stack computer consists of an operation code with no address field. This operation has the effect of popping the 2 top numbers from the stack, operating the numbers and pushing the sum into the stack. For example: ADD

Computers may have instructions of several different lengths containing varying number of addresses. Following are the types of instructions.

1. **Three address Instruction**
   With this type of instruction, each instruction specifies two operand location and a result location. A temporary location T is used to store some intermediate result so as not to alter any of the operand location. The three address instruction format requires a very complex design to hold the three address references.
   Format: Op X, Y, Z; X ← Y Op Z

Example: ADD X, Y, Z; X ← Y + Z

- ADVANTAGE: It results in short programs when evaluating arithmetic expressions.
- DISADVANTAGE: The instructions requires too many bits to specify 3 addresses.

2. **Two address instruction**
   Two-address instructions are the most common in commercial computers. Here again each address field can specify either a processor register, or a memory word. One address must do double duty as both operand and result. The two address instruction format reduces the space requirement. To avoid altering the value of an operand, a MOV instruction is used to move one of the values to a result or temporary location T, before performing the operation.
   Format: Op X, Y; X ← X Op Y
   Example: SUB X, Y; X ← X - Y

3. **One address Instruction**
   It was generally used in earlier machine with the implied address been a CPU register known as accumulator. The accumulator contains one of the operand and is used to store the result. One-address instruction uses an implied accumulator (Ac) register for all data manipulation. All operations are done between the AC register and a memory operand. We use LOAD and STORE instruction for transfer to and from memory and Ac register.
   Format: Op X; Ac ← Ac Op X
   Example: MUL X; Ac ← Ac * X

4. **Zero address Instruction**
   It does not use address field for the instruction like ADD, SUB, MUL, DIV etc. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack. The name "Zero" address is given because of the absence of an address field in the computational instruction.
   Format: Op; TOS ← TOS Op (TOS – 1)
   Example: DIV; TOS ← TOS DIV (TOS – 1)

Example: To illustrate the influence of the number of address on computer programs, we will evaluate the arithmetic statement X=(A+B)*(C+D) using Zero, one, two, or three address instructions.

1. Three-Address Instructions:
   ADD R1, A, B; R1 ← M[A] + M[B]
   ADD R2, C, D; R2 ← M[C] + M[D]
   MUL X, R1,R2; M[X] ← R1 * R2

It is assumed that the computer has two processor registers R1 and R2. The symbol M[A] denotes the operand at memory address symbolized by A.

2. Two-Address Instructions:
   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

3. One-Address Instruction:

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

Here, T is the temporary memory location required for storing the intermediate result.

4. Zero-Address Instructions:

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

2.4 Addressing Modes

- Specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.
- Computers use addressing mode techniques for the purpose of accommodating the following purposes:-
  - To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data and various other purposes.
  - To reduce the number of bits in the addressing field of the instructions.
    - Other computers use a single binary for operation & Address mode.
    - The mode field is used to locate the operand.
    - Address field may designate a memory address or a processor register.
    - There are 2 modes that need no address field at all (Implied & immediate modes).

Effective address (EA):

- The effective address is defined to be the memory address obtained from the computation dictated by the given addressing mode.
- The effective address is the address of the operand in a computational-type instruction.

The most well known addressing mode are:

- Implied Addressing Mode.
- Immediate Addressing Mode
- Register Addressing Mode
- Register Indirect Addressing Mode
- Auto-increment or Auto-decrement Addressing Mode
- Direct Addressing Mode
- Indirect Addressing Mode
- Displacement Address Addressing Mode
- Relative Addressing Mode
- Index Addressing Mode
- Stack Addressing Mode

Implied Addressing Mode:

- In this mode the operands are specified implicitly in the definition of the instruction.
  For example:- CMA - "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction. In fact, all register reference instructions that use an accumulator are implied-mode instructions.

Instruction

Opcode

Advantage: no memory reference. Disadvantage: limited operand

Immediate Addressing mode:

- In this mode the operand is specified in the instruction itself. In other words, an immediate-mode instruction has an operand field rather than an address field.
- This instruction has an operand field rather than an address field. The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction.
- These instructions are useful for initializing register to a constant value;
  For example MVI B, 50H

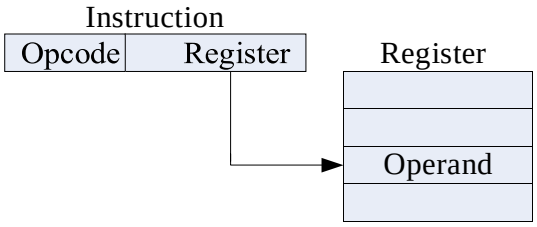
Instruction

Opcode	Operand

It was mentioned previously that the address field of an instruction may specify either a memory word or a processor register. When the address field specifies a processor register, the instruction is said to be in register-mode.
Advantage: no memory reference. Disadvantage: limited operand

Register direct addressing mode:

- In this mode, the operands are in registers that reside within the CPU.
- The particular register is selected from the register field in the instruction.
  For example MOV A, B



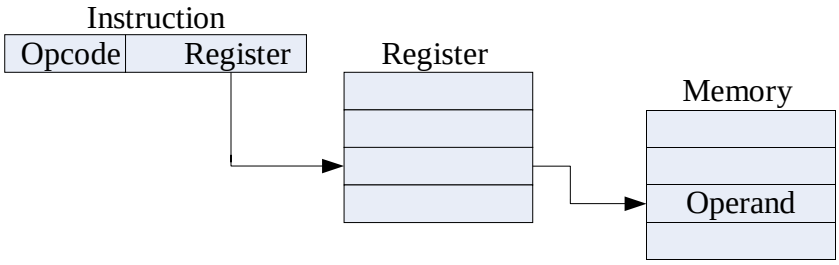
Effective Address (EA) = R

Advantage: no memory reference. Disadvantage: limited address space

Register indirect addressing mode:

- In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in the memory.
- In other words, the selected register contains the address of the operand rather than the operand itself.
- Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction.

  For example LDAX B



Effective Address (EA) = (R)

Advantage: Large address space.

The address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

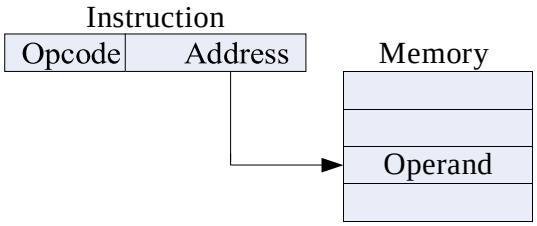
Disadvantage: Extra memory reference

Auto increment or Auto decrement Addressing Mode:

- This is similar to register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory.
- When the address stored in the registers refers to a table of data in memory, it is necessary to increment or decrement the registers after every access to the table.
- This can be achieved by using the increment or decrement instruction. In some computers it is automatically accessed.
- The address field of an instruction is used by the control unit in the CPU to obtain the operands from memory.
- Sometimes the value given in the address field is the address of the operand, but sometimes it is the address from which the address has to be calculated.

Direct Addressing Mode

- In this mode the effective address is equal to the address part of the instruction. The operand resides in memory and its address is given directly by the address field of the instruction.
  For example LDA 4000H

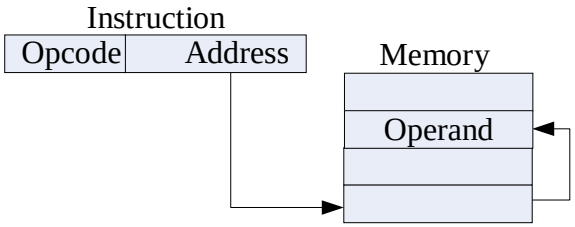


Effective Address (EA) = A

Advantage: Simple. Disadvantage: limited address field

Indirect Addressing Mode

- In this mode the address field of the instruction gives the address where the effective address is stored in memory.
- Control unit fetches the instruction from the memory and uses its address part to access memory again to read the effective address.

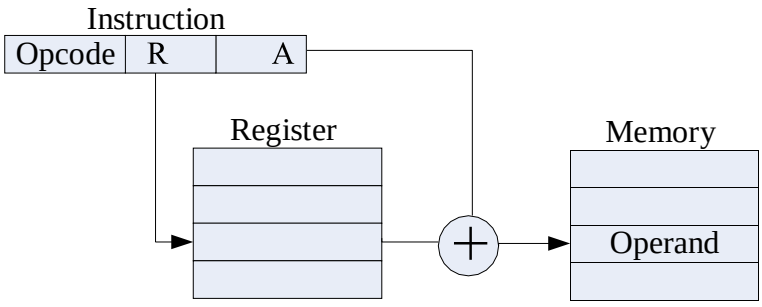


Effective Address (EA) = (A)

Advantage: Flexibility. Disadvantage: Complexity

Displacement Addressing Mode

- A very powerful mode of addressing combines the capabilities of direct addressing and register indirect addressing.
- The address field of instruction is added to the content of specific register in the CPU.



Effective Address (EA) = A + (R)

Advantage: Flexibility. Disadvantage: Complexity

Relative Addressing Mode

- In this mode the content of the program counter (PC) is added to the address part of the instruction in order to obtain the effective address.
- The address part of the instruction is usually a signed number (either a +ve or a –ve number).
- When the number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction.
  Effective Address (EA) = PC + A

Indexed Addressing Mode

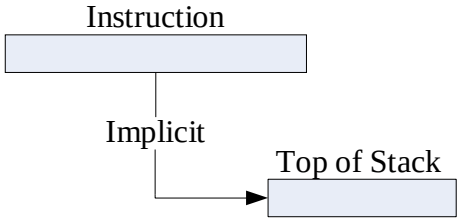
- In this mode the content of an index register (XR) is added to the address part of the instruction to obtain the effective address.
- The index register is a special CPU register that contains an index value.
- Note: If an index-type instruction does not include an address field in its format, the instruction is automatically converted to the register indirect mode of operation.
  Effective Address (EA) = XR + A

Base Register Addressing Mode

- In this mode the content of a base register (BR) is added to the address part of the instruction to obtain the effective address.
- This is similar to the indexed addressing mode except that the register is now called a base register instead of the index register.
- The base register addressing mode is used in computers to facilitate the relocation of programs in memory i.e. when programs and data are moved from one segment of memory to another.
  Effective Address (EA) = BR + A

Stack Addressing Mode

- The stack is the linear array of locations. It is some times referred to as push down list or last in First out (LIFO) queue. The stack pointer is maintained in register.



Effective Address (EA) = TOS

Let us try to evaluate the addressing modes with as example.

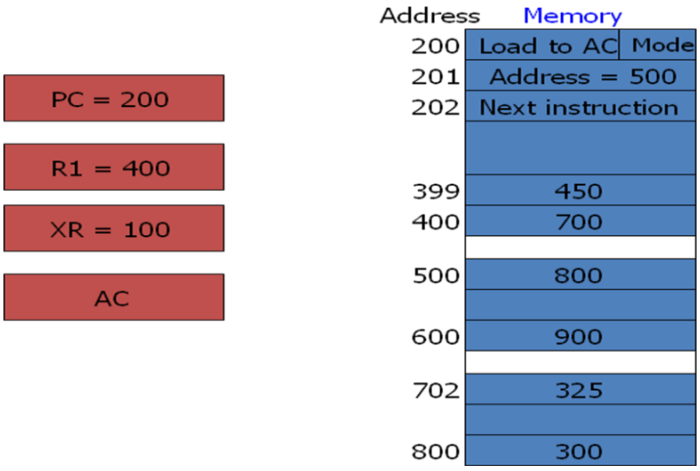


Fig: Numerical Example for Addressing Modes

Addressing Mode	Effective Address	Content of AC
Direct address	500	800
Immediate operand	201	500
Indirect address	800	300
Relative address	702	325
Indexed address	600	900
Register	—	400
Register indirect	400	700
Autoincrement	400	700
Autodecrement	399	450

Fig: Tabular list of Numerical Example

2.5 Data Transfer and Manipulation

Data transfer instructions cause transfer of data from one location to another without changing the binary information. The most common transfer are between the

- Memory and Processor registers
- Processor registers and input output devices
- Processor registers themselves

Typical Data Transfer Instructions

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Data manipulation Instructions

Data manipulation instructions perform operations on data and provide the computational capabilities for the computer. These instructions perform arithmetic, logic and shift operations.

Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

Logical and Bit Manipulation Instructions

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

Program Control Instructions

The program control instructions provide decision making capabilities and change the path taken by the program when executed in computer. These instructions specify conditions for altering the content of the program counter. The change in value of program counter as a result of execution of program control instruction causes a break in sequence of instruction execution. Some typical program control instructions are:

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by ANDing)	TST

Subroutine call and Return

A subroutine call instruction consists of an operation code together with an address that specifies the beginning of the subroutine. The instruction is executed by performing two tasks:

- The address of the next instruction available in the program counter (the return address) is stored in a temporary location (stack) so the subroutine knows where to return.
- Control is transferred to the beginning of the subroutine.

The last instruction of every subroutine, commonly called return from subroutine; transfer the return address from the temporary location into the program counter. This results in a transfer of program control to the instruction where address was originally stored in the temporary location.

Interrupt

The interrupt procedure is, in principle, quite similar to a subroutine call except for three variations:

- The interrupt is usually initiated by an external or internal signal rather than from execution of an instruction.
- The address of the interrupt service program is determined by the hardware rather than from the address field of an instruction.
- An interrupt procedure usually stores all the information necessary to define the state of the CPU rather than storing only the program counter.

2.6 RISC and CISC

- Important aspect of computer – design of the instruction set for processor.
- Instruction set – determines the way that machine language programs are constructed.
- Early computers – simple and small instruction set, need to minimize the hardware used.
- Advent of IC – cheaper digital software, instructions intended to increase both in number of complexity.
- Many computers – more than 100 or 200 instructions, variety of data types and large number of addressing modes.

Complex Instruction Set Computers (CISC)

- The trend into computer hardware complexity was influenced by various factors:
  - Upgrading existing models to provide more customer applications
  - Adding instructions that facilitate the translation from high-level language into machine language programs
  - Striving to develop machines that move functions from software implementation into hardware implementation
- A computer with a large number of instructions is classified as a *complex instruction set computer* (CISC).
- One reason for the trend to provide a complex instruction set is the desire to simplify the compilation and improve the overall computer performance.

- The essential goal of CISC architecture is to attempt to provide a single machine instruction for each statement that is written in a high-level language.
- Examples of CISC architecture are the DEC VAX computer and the IBM 370 computer. Other are 8085, 8086, 80x86 etc.

The major characteristics of CISC architecture

- A large number of instructions– typically from 100 to 250 instructions
- Some instructions that perform specialized tasks and are used infrequently
- A large variety of addressing modes—typically from 5 to 20 different modes
- Variable-length instruction formats
- Instructions that manipulate operands in memory
- Reduced speed due to memory read/write operations
- Use of microprogram – special program in control memory of a computer to perform the timing and sequencing of the microoperations – fetch, decode, execute etc.
- Major complexity in the design of microprogram
- No large number of registers – single register set of general purpose and low cost

Reduced Instruction Set Computers (RISC)

A computer uses fewer instructions with simple constructs so they can be executed much faster within the CPU without having to use memory as often. It is classified as a *reduced instruction set computer* (RISC).

- RISC concept – an attempt to reduce the execution cycle by simplifying the instruction set
- Small set of instructions – mostly register to register operations and simple load/store operations for memory access
- Each operand – brought into register using load instruction, computations are done among data in registers and results transferred to memory using store instruction
- Simplify instruction set and encourages the optimization of register manipulation
- May include immediate operands, relative mode etc.

The major characteristics of RISC architecture

- Relatively few instructions
- Relatively few addressing modes
- Memory access limited to load and store instructions
- All operations done within the registers of the CPU
- Fixed-length, easily decoded instruction format
- Single-cycle instruction execution
- Hardwired rather than microprogrammed control

Other characteristics attributed to RISC architecture

- A relatively large number of registers in the processor unit
- Use of overlapped register windows to speed-up procedure call and return
- Efficient instruction pipeline – fetch, decode and execute overlap
- Compiler support for efficient translation of high-level language programs into machine language programs
- Studies that show improved performance for RISC architecture do not differentiate between the effects of the reduced instruction set and the effects of a large register file.
- A large number of registers in the processing unit are sometimes associated with RISC processors.
- RISC processors often achieve 2 to 4 times the performance of CISC processors.
- RISC uses much less chip space; extra functions like memory management unit or floating point arithmetic unit can also be placed on same chip. Smaller chips allow a semiconductor mfg. to place more parts on a single silicon wafer, which can lower the per chip cost dramatically.
- RISC processors are simpler than corresponding CISC processors, they can be designed more quickly.

Comparison between RISC and CISC Architectures

S.N.	RISC	CISC
1	Simple instructions taking one cycle	Complex instructions taking multiple cycles
2	Only load and store memory references	Any instructions may reference memory
3	Heavily pipelined	Not/less pipelined
4	Multiple register sets	Single register set
5	Complexity is in compiler	Complexity is in micro-programming
6	Instructions executed by hardware	Instructions interpreted by micro-programming
7	Fixed format instructions	Variable format instructions
8	Few instructions and modes	Large instructions and modes

Overlapped register windows

- Some computers provide multiple-register banks, and each procedure is allocated its own bank of registers. This eliminates the need for saving and restoring register values.
- Some computers use the memory stack to store the parameters that are needed by the procedure, but this required a memory access every time the stack is accessed.
- A characteristic of some RISC processors is their use of overlapped register windows to provide the passing of parameters and avoid the need for saving and restoring register values.
- The concept of overlapped register windows is illustrated in below figure.
- In general, the organization of register windows will have the following relationships:
  Number of global registers = G
  Number of local registers in each window = L
  Number of registers common to two windows = C
  Number of windows = W
- The number of registers available for each window is calculated as followed:
  Window size = L + 2C + G
- The total number of registers needed in the processor is
  Register file = (L + C)W + G

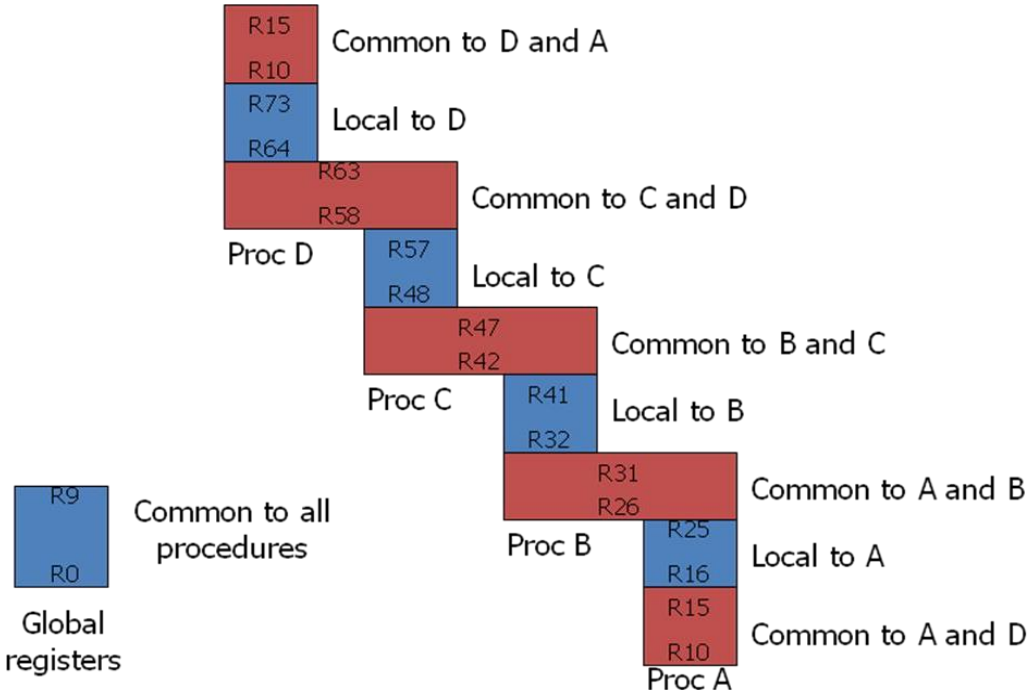


Fig: Overlapped Register Window

- A total of 74 registers
- Global Registers = 10 → common to all procedures
- 64 registers → divided into 4 windows A, B, C & D
- Each register window = 10 registers → local
- Two sets of 16 registers → common to adjacent procedures

Berkeley RISC I

- The Berkeley RISC I is a 32-bit integrated circuit CPU.
  - It supports 32-bit address and either 8-, 16-, or 32-bit data.
  - It has a 32-bit instruction format and a total of 31 instructions.
  - There are three basic addressing modes: Register addressing, immediate operand, and relative to PC addressing for branch instructions.
  - It has a register file of 138 registers; 10 global register and 8 windows of 32 registers in each
  - The 32 registers in each window have an organization similar to overlapped register window.

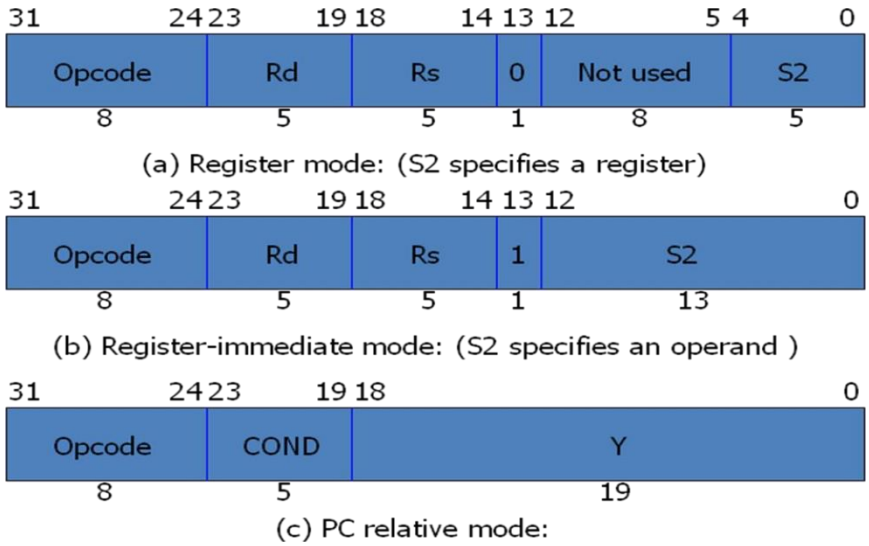
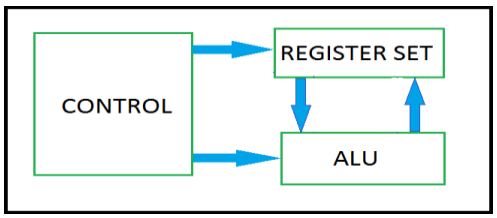


Fig: Instruction Format of Berkeley RISC I

- Above figure shows the 32-bit instruction formats used for *register-to-register instructions* and memory access instructions.
- Seven of the bits in the operation code specify an operation, and the eighth bit indicates whether to update the status bits after an ALU operation.
- For *register-to-register instructions* :
  - The 5-bit R_d field select one of the 32 registers as a destination for the result of the operation
  - The operation is performed with the data specified in fields Rs and S_2.
  - Thus the instruction has a three-address format, but the second source may be either a register or an immediate operand.
- For memory access instructions:
  - Rs to specify a 32-bit address in a register
  - S_2 to specify an offset
  - Register R_0 contains all 0's, so it can be used in any field to specify a zero quantity
- The third instruction format combines the last three fields to form a 19-bit relative address Y and is used primarily with the jump and call instructions.
  - The *COND field* replaces the Rd field for jump instructions and is used to specify one of *16 possible branch conditions*.

2.7 64 – bit Processor

- The brain of the PC is processor or CPU.
- It performs the system's calculating and processing operations.
- The term N-bits means that its ALU, internal registers and most of its instructions are designed to work with N-bit binary words.
- The major components of CPU are:



- 64-bit processors have 64-bit ALUs, 64-bit registers, and 64-bit buses.
- A 64-bit register can address up up to 2^{64} bytes of logical address.
- 64-bit processors have been with us since 1992.
- Eg: 64-bit AMD processor.

Internal Architecture

- The internal logic design of microprocessor which determines how and when various operations are performed.
- The various function performed by the microprocessor can be classified as:
  - Microprocessor initiated operations
  - Internal operations
  - Peripheral operations
- Microprocessor initiated operations mainly deal with memory and I/O read and write operations.
- Internal operations determines how and what operations can be performed with the data.The operations include:
  1. storing
  2. performing arithmetic and logical operations
  3. test for conditions
  4. store in the stack
- External initiated operations are initiated by the external devices to perform special operations like reset, interrupt, ready, etc.
- The block diagram of 64-bit microprocessor is shown below.
- The major parts of the block diagram are:

- General register unit
- Control and decoding unit
- Bus unit
- Cache memory unit
- Floating point register unit
- Issue ports

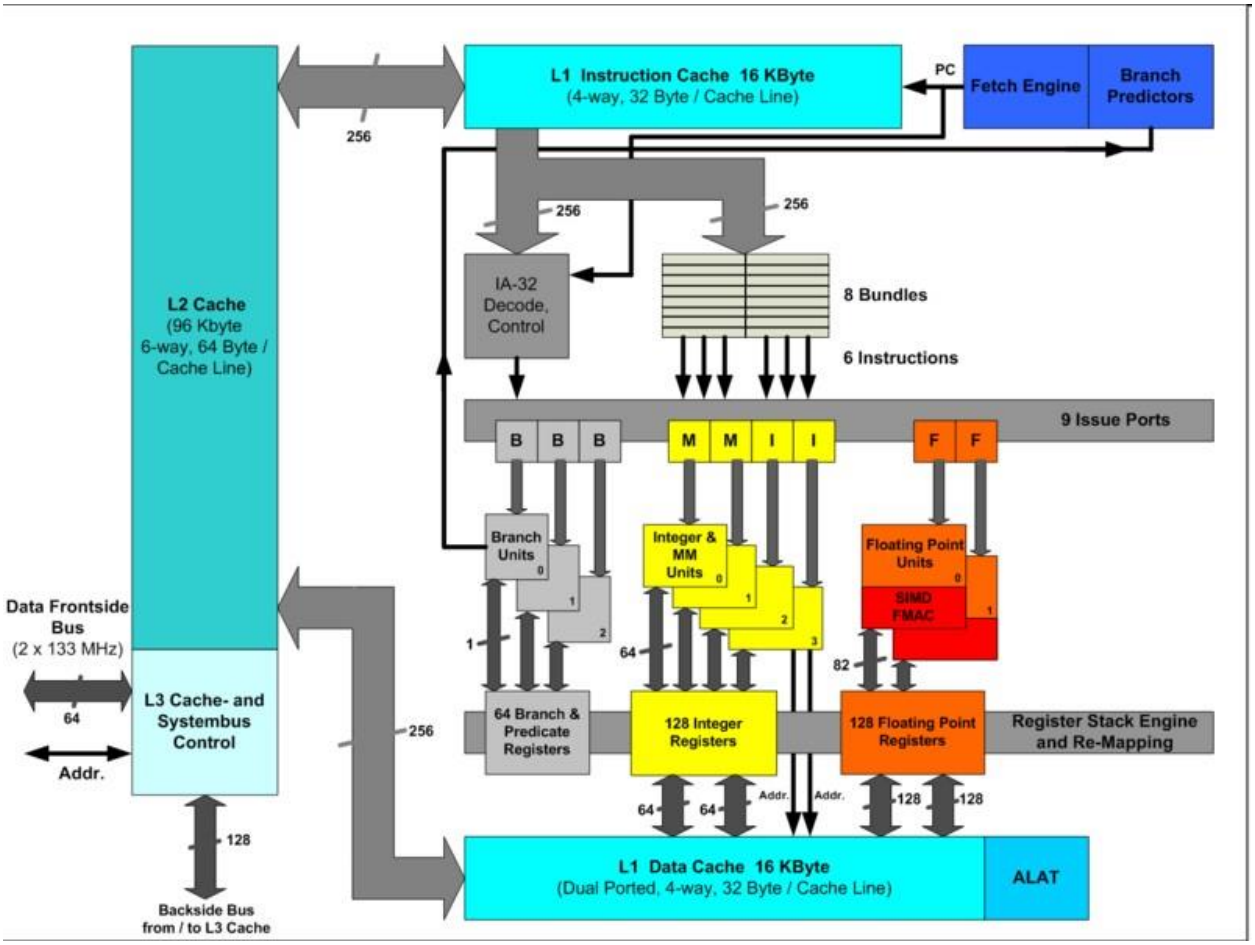


Fig: Block diagram of 64-bit internal architecture

Architecture Elements

- Addressing Modes
- General Purpose Registers
- Non-modal and modal Instructions
- New Instructions in Support of 64-bit
- New immediate Instructions

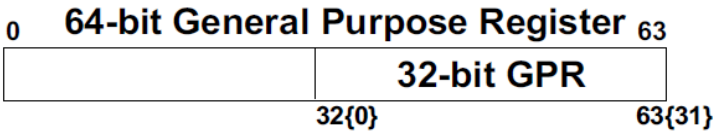
Addressing modes

- This addressing mode determines the working environment. i.e 24,32 or 64 bit mode
- PSW bits 31 and 32 designate addressing mode (out of 64 bit).
    - Addressing modes bits:00=24 bit-mode

01=32 bit-mode
11=64 bit-mode

General purposes register (GPR)

- The register is treated as 64-bits for:
  - Address generation in 64-bit mode.



- The register is treated as 32-bits for:
  - Address generation in 24/32-bit mode.

New instructions in 64-bit:

- Load Reversed - LRV, LRVR
- Multiply Logical - ML, MLR
- Divide Logical - DL, DLR
- Add Logical w/ Carry - ALC
- Subtract Logical w/ Borrow - SLB
- Store Reversed - STRV
- Rotate Left Single Logical – RLL

New immediate Instructions

- Load Logical Immediate
- Insert Logical Immediate
- AND Immediate
- OR Immediate
- Test Under Mask (High/Low)

Comparison of 64-bit with 32-bit

- Contains 32-bit data lines whereas 64-bit contains 64 data lines.
- Can address max 2^32(4 GB) of data whereas 64 bit can address 2^64(18 billion GB).
- Speed and execution is both fast in 64-bit processors.
- 64-bit processors can drive 32-bit applications even faster, by handling more data per clock cycle than a 32-bit processor.
- The table shows the basic difference between two:

Table 1. Resources required to load, add, and store two 64-bit integers

Operation	Resources on 32-bit processor	Resources on 64-bit processor	Effective improvement with 64-bit
Load two 64-bit integers	• Requires four (4) 32-bit registers to hold data • Requires 4 load instructions	• Requires two (2) 64-bit registers to hold data • Requires 2 load instructions	Reduced number of instructions to load data by one half and fewer registers consumed by one half
Add two 64-bit integers	• Requires 2 addition instructions; an add with carry and an extended to include the carry	• Requires one addition instruction	Reduced number of instructions by one half and reduced interlocking among instructions and carry status
Store two 64-bit integers	• Requires four (4) 32-bit registers to hold data • Requires 4 store instructions to save data	• Requires two (2) 64-bit registers to hold data • Requires 2 store instructions to save data	Reduced number of instructions to store data by one half and registers consumed by one half
Total resources	10 instructions issued and 4 registers plus carry field	5 instructions issued and 2 registers used	One half the instructions, less than one half the resources consumed

Advantages and disadvantages:

advantages	Disadvantages
➢ Previous processors can have max 4 Gb of physical memory but 64-bit can handle more. ➢ More general purpose registers than in older processors. ➢ Significant increase in speed due to wider data bus and processing is fast.	➢ Compatibility difficulty with existing software as they are mostly developed to the 32-bit processors. ➢ 64-bit OS must have 64-bit drivers, for working efficiently. ➢ They are costly.

UNIT-IV MEMORY SYSTEM

Microcomputer Memory

Memory is an essential component of the microcomputer system.
It stores binary instructions and datum for the microcomputer.
The memory is the place where the computer holds current programs and data that are in use.
None technology is optimal in satisfying the memory requirements for a computer system.
Computer memory exhibits perhaps the widest range of type, technology, organization, performance and cost of any feature of a computer system.
The memory unit that communicates directly with the CPU is called main memory.
Devices that provide backup storage are called auxiliary memory or secondary memory.

Characteristics of memory systems

The memory system can be characterised with their Location, Capacity, Unit of transfer, Access method, Performance, Physical type, Physical characteristics, Organisation.

Location

Processor memory: The memory like registers is included within the processor and termed as processor memory.
Internal memory: It is often termed as main memory and resides within the CPU.
External memory: It consists of peripheral storage devices such as disk and magnetic tape that are accessible to processor via i/o controllers.

Capacity

Word size: Capacity is expressed in terms of words or bytes.
The natural unit of organisation
Number of words: Common word lengths are 8, 16, 32 bits etc.
or Bytes

Unit of Transfer

Internal: For internal memory, the unit of transfer is equal to the number of data lines into and out of the memory module.
External: For external memory, they are transferred in block which is larger than a word.
Addressable unit
Smallest location which can be uniquely addressed
Word internally
Cluster on Magnetic disks

Access Method

Sequential access: In this access, it must start with beginning and read through a specific linear sequence. This means access time of data unit depends on position of records (unit of data) and previous location.

e.g. tape

Direct Access: Individual blocks of records have unique address based on location. Access is accomplished by jumping (direct access) to general vicinity plus a sequential search to reach the final location.

e.g. disk

Random access: The time to access a given location is independent of the sequence of prior accesses and is constant. Thus any location can be selected out randomly and directly addressed and accessed.

e.g. RAM

Associative access: This is random access type of memory that enables one to make a comparison of desired bit locations within a word for a specified match, and to do this for all words simultaneously.

e.g. cache

Performance

Access time: For random access memory, access time is the time it takes to perform a read or write operation i.e. time taken to address a memory plus to read / write from addressed memory location. Whereas for non-random access, it is the time needed to position read / write mechanism at desired location.

Time between presenting the address and getting the valid data

Memory Cycle time: It is the total time that is required to store next memory access operation from the previous memory access operation.

Memory cycle time = access time plus transient time (any additional time required before a second access can commence).

Time may be required for the memory to "recover" before next access

Cycle time is access + recovery

Transfer Rate: This is the rate at which data can be transferred in and out of a memory unit.

Rate at which data can be moved

For random access, R = 1 / cycle time

For non-random access, Tn = Ta + N / R; where Tn – average time to read or write N bits, Ta – average access time, N – number of bits, R – Transfer rate in bits per second (bps).

Physical Types

Semiconductor

RAM

Magnetic

Disk & Tape

Optical

CD & DVD

Others

Bubble
Hologram

Physical Characteristics

Decay: Information decays mean data loss.
Volatility: Information decays when electrical power is switched off.
Erasable: Erasable means permission to erase.
Power consumption: how much power consumes?

Organization

Physical arrangement of bits into words
Not always obvious

- e.g. interleaved

The Memory Hierarchy

Capacity, cost and speed of different types of memory play a vital role while designing a memory system for computers.

If the memory has larger capacity, more application will get space to run smoothly.

It's better to have fastest memory as far as possible to achieve a greater performance. Moreover for the practical system, the cost should be reasonable.

There is a tradeoff between these three characteristics cost, capacity and access time. One cannot achieve all these quantities in same memory module because

If capacity increases, access time increases (slower) and due to which cost per bit decreases.

If access time decreases (faster), capacity decreases and due to which cost per bit increases.

The designer tries to increase capacity because cost per bit decreases and the more application program can be accommodated. But at the same time, access time increases and hence decreases the performance.

So the best idea will be to use memory hierarchy.

Memory Hierarchy is to obtain the highest possible access speed while minimizing the total cost of the memory system.

Not all accumulated information is needed by the CPU at the same time.

Therefore, it is more economical to use low-cost storage devices to serve as a backup for storing the information that is not currently used by CPU

The memory unit that directly communicate with CPU is called the *main memory*

Devices that provide backup storage are called *auxiliary memory*

The memory hierarchy system consists of all storage devices employed in a computer system from the slow by high-capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory

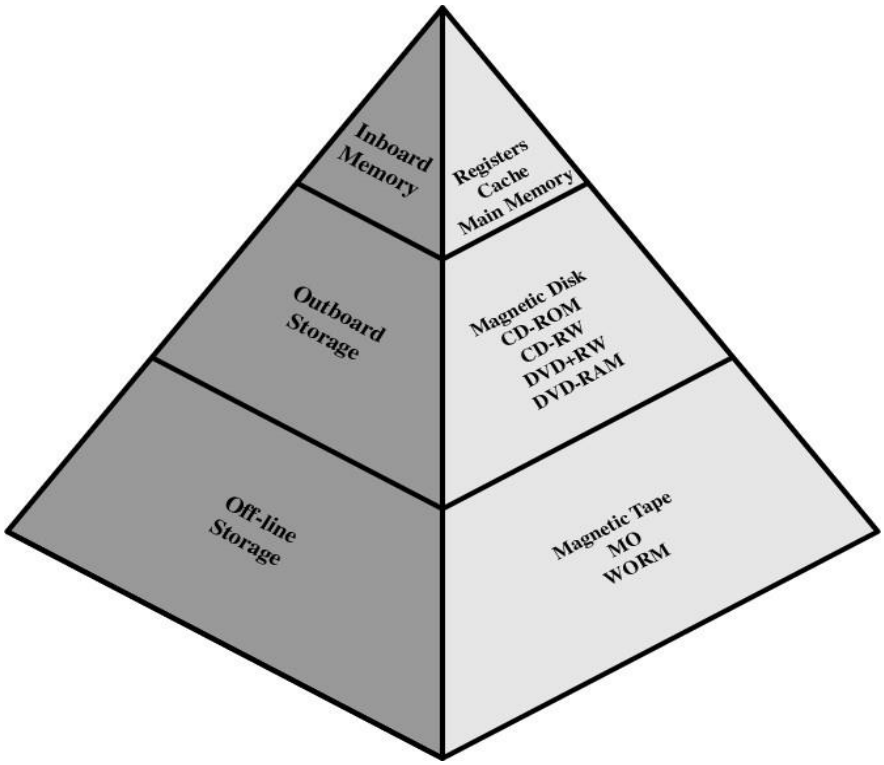
The main memory occupies a central position by being able to communicate directly with the CPU and with auxiliary memory devices through an I/O processor

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

CPU logic is usually faster than main memory access time, with the result that processing speed is limited primarily by the speed of main memory

The cache is used for storing segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations

The memory hierarchy system consists of all storage devices employed in a computer system from slow but high capacity auxiliary memory to a relatively faster cache memory accessible to high speed processing logic. The figure below illustrates memory hierarchy.



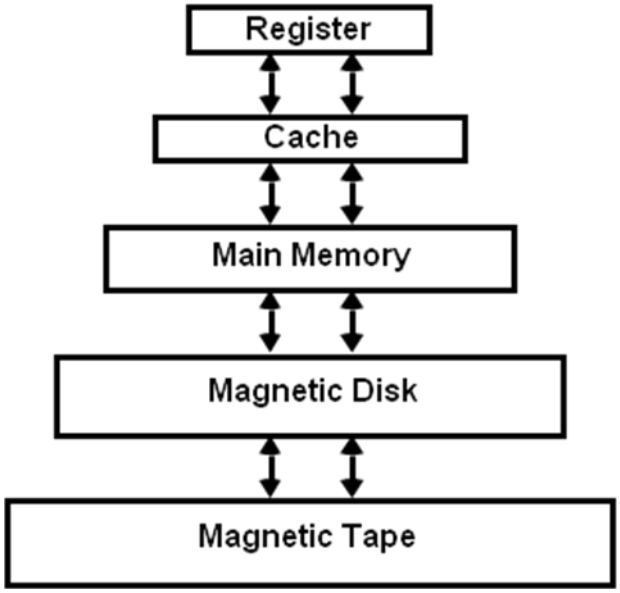


Fig: Memory Hierarchy

As we go down in the hierarchy

Cost per bit decreases
Capacity of memory increases
Access time increases
Frequency of access of memory by processor also decreases.

Hierarchy List

Registers
L1 Cache
L2 Cache
Main memory
Disk cache
Disk
Optical
Tape

Internal and External memory

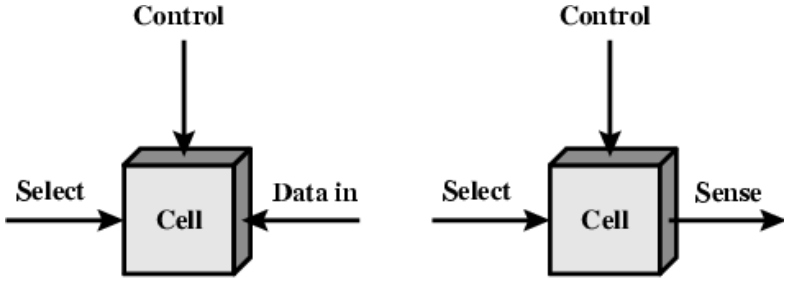
Internal or Main Memory

The main memory is the central unit of the computer system. It is relatively large and fast memory to store programs and data during the computer operation. These memories employ semiconductor integrated circuits. The basic element of the semiconductor memory is the memory cell.

The memory cell has three functional terminals which carries the electrical signal.

The select terminal: It selects the cell.
The data in terminal: It is used to input data as 0 or 1 and data out or sense terminal is used for the output of the cell's state.
The control terminal: It controls the function i.e. it indicates read and write.

Most of the main memory in a general purpose computer is made up of RAM



(a) Write (b) Read

integrated circuits chips, but a portion of the memory may be constructed with ROM chips

RAM– Random Access memory

Memory cells can be accessed for information transfer from any desired random location.

The process of locating a word in memory is the same and requires of locating a word in memory is the same and requires an equal amount of time no matter where the cells are located physically in memory thus named 'Random access'.

Integrated RAM are available in two possible operating modes, S*tatic and Dynamic*

Static RAM (SRAM)

The static RAM consists of flip flop that stores binary information and this stored information remains valid as long as power is applied to the unit.

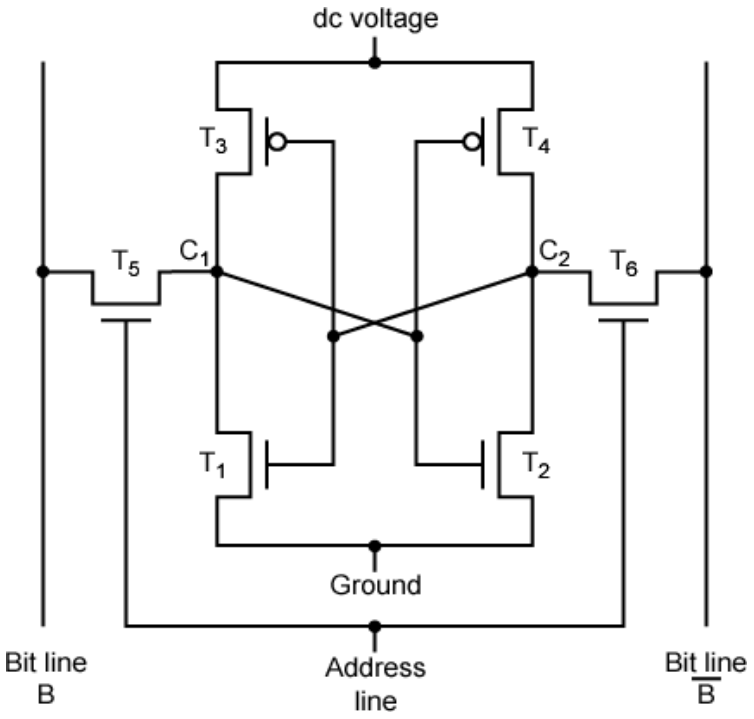


Fig: SRAM structure

Four transistors T1, T2, T3 and t4 are cross connected in an arrangement that produces a stable logical state.

In logic state 1, point C1 is high and point C2 is low. In this state, T1 & T4 are off and T2 & T3 are on.

In logic state 0, point C1 is low and C2 is high. In this state, T1 & T4 are on and T2 & T3 are off.

The address line controls the two transistors T5 & T6. When a signal is applied to this line, the two transistors are switched on allowing for read and write operation.

For a write operation, the desired bit value is applied to line B while it's complement is applied to line B complement. This forces the four transistors T1, T2, T3 & T4 into a proper state.

For the read operation, the bit value is read from line B.

Dynamic RAM (DRAM)

The dynamic RAM stores the binary information in the form of electrical charges and capacitor is used for this purpose.

Since charge stored in capacitor discharges with time, capacitor must be periodically recharged and which is also called refreshing memory.

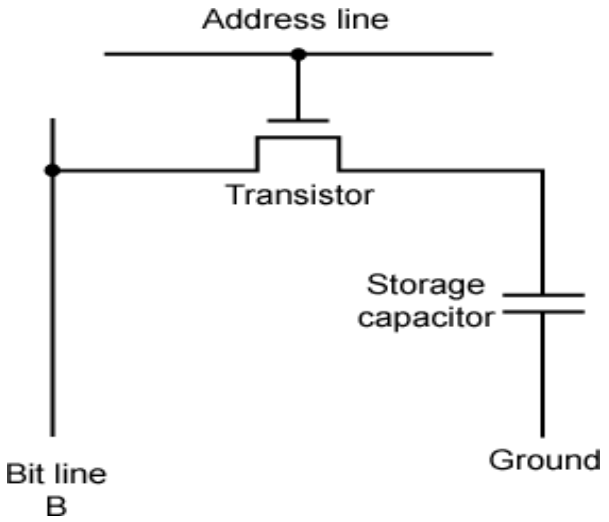


Fig: DRAM structure

The address line is activated when the bit value from this cell is to be read or written.

The transistor acts as switch that is closed i.e. allowed current to flow, if voltage is applied to the address line; and opened i.e. no current to flow, if no voltage is present in the address line.

For DRAM writing

The address line is activated which causes the transistor to conduct.

The sense amplifier senses the content of the data bus line for this cell.

If the bus line is low, then amplifier will ground the bit line of cell and any charge in capacitor is addressed out.

If data bus is high, then a +5V is applied on bit line and voltage will flow through transistor and charge the capacitor.

For DRAM reading

Address line is activated which causes the transistor to conduct.

If there is charge stored in capacitor, then current will flow through transistor and raise the voltage in bit line. The amplifier will store the voltage and place a 1 on data out line.

If there is no charge stored in capacitor, then no current will flow through transistor and voltage bit line will not be raised. The amplifier senses that there is no charge and places a 0 on data out line.

SRAM versus DRAM

Both volatile

- Power needed to preserve data

Static RAM

Uses flip flop to store information
Needs more space
Faster, digital device
Expensive, big in size
Don't require refreshing circuit
Used in cache memory

Dynamic RAM

Uses capacitor to store information
More dense i.e. more cells can be accommodated per unit area
Slower, analog device
Less expensive, small in size
Needs refreshing circuit
Used in main memory, larger memory units

ROM– Read Only memory

Read only memory (ROM) contains a permanent pattern of data that cannot be changed.

A ROM is non-volatile that is no power source is required to maintain the bit values in memory.

While it is possible to read a ROM, it is not possible to write new data into it.

The data or program is permanently presented in main memory and never be loaded from a secondary storage device with the advantage of ROM.

A ROM is created like any other integrated circuit chip, with the data actually wired into the chip as part of the fabrication process.

It presents two problems

The data insertion step includes a relatively large fixed cost, whether one or thousands of copies of a particular ROM are fabricated.

There is no room for error. If one bit is wrong, the whole batch of ROM must be thrown out.

Types of ROM

Programmable ROM (PROM)

- It is non-volatile and may be written into only once. The writing process is performed electrically and may be performed by a supplier or customer at a time later than the original chip fabrication.

Erasable Programmable ROM (EPROM)

- It is read and written electrically. However, before a write operation, all the storage cells must be erased to the same initial state by exposure of the packaged chip to ultraviolet radiation (UV ray). Erasure is performed by shining an intense ultraviolet light through a window that is designed into the memory chip. EPROM is optically managed and more expensive than PROM, but it has the advantage of the multiple update capability.

Electrically Erasable programmable ROM (EEPROM)

- This is a read mostly memory that can be written into at any time without erasing prior contents, only the byte or byte addresses are updated. The write operation takes considerably longer than the read operation, on the order of several hundred microseconds per byte. The EEPROM combines the advantage of non-volatility with the flexibility of being updatable in place, using ordinary bus control, addresses and data lines. EEPROM is more expensive than EPROM and also is less dense, supporting fewer bits per chip.

Flash Memory

- Flash memory is also the semiconductor memory and because of the speed with which it can be reprogrammed, it is termed as flash. It is interpreted between EPROM and EEPROM in both cost and functionality. Like EEPROM, flash memory uses an electrical erasing technology. An entire flash memory can be erased in one or a few seconds, which is much faster than EPROM. In addition, it is possible to erase just blocks of memory rather than an entire chip. However, flash memory doesn't provide byte level erasure, a section of memory cells are erased in an action or 'flash'.

External Memory

The devices that provide backup storage are called external memory or auxiliary memory. It includes serial access type such as magnetic tapes and random access type such as magnetic disks.

Magnetic Tape

A magnetic tape is the strip of plastic coated with a magnetic recording medium. Data can be recorded and read as a sequence of character through read / write head. It can be stopped, started to move forward or in reverse or can be rewound. Data on tapes are structured as number of parallel tracks running length wise. Earlier tape system typically used nine tracks. This made it possible to store data one byte at a time with additional parity bit as 9th track. The recording of data in this form is referred to as parallel recording.

Magnetic Disk

A magnetic disk is a circular plate constructed with metal or plastic coated with magnetic material often both side of disk are used and several disk stacked on one spindle which Read/write head available on each surface. All disks rotate together at high speed. Bits are stored in magnetize surface in spots along concentric circles called tracks. The tracks are commonly divided into sections called sectors. After the read/write head are positioned in specified track the system has to wait until the rotating disk reaches the specified sector under read/write head. Information transfer is very fast once the beginning of sector has been reached. Disk that are permanently attached to the unit assembly and cannot be used by occasional user are called hard disk drive with removal disk is called floppy disk.

Optical Disk

The huge commercial success of CD enabled the development of low cost optical disk storage technology that has revolutionized computer data storage. The disk is form from resin such as polycarbonate. Digitally recorded information is imprinted as series of microscopic pits on the surface of poly carbonate. This is done with the finely focused high intensity leaser. The pitted surface is then coated with reflecting surface usually aluminum or gold. The shiny surface is protected against dust and scratches by the top coat of acrylic.

Information is retrieved from CD by low power laser. The intensity of reflected light of laser changes as it encounters a pit. Specifically if the laser beam falls on pit which has somewhat rough surface the light scatters and low intensity is reflected back to the surface. The areas between pits are called lands. A land is a smooth surface which reflects back at higher intensity. The change between pits and land is detected by photo sensor and converted into digital signal. The sensor tests the surface at regular interval.

DVD-Technology

Multi-layer
Very high capacity (4.7G per layer)
Full length movie on single disk
Using MPEG compression
Finally standardized (honest!)
Movies carry regional coding
Players only play correct region films

DVD-Writable

Loads of trouble with standards
First generation DVD drives may not read first generation DVD-W disks
First generation DVD drives may not read CD-RW disks

Cache memory principles

Principles

Intended to give memory speed approaching that of fastest memories available but with large size, at close to price of slower memories

Cache is checked first for all memory references.

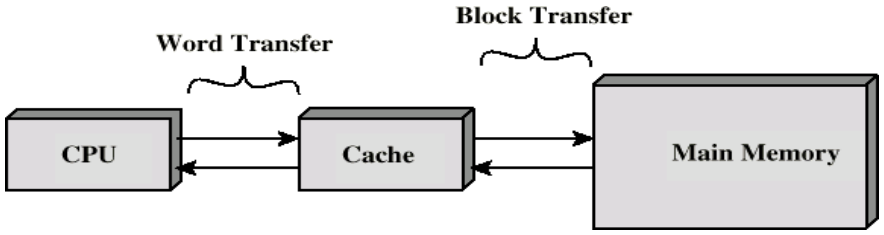
If not found, the entire block in which that reference resides in main memory is stored in a cache slot, called a line

Each line includes a tag (usually a portion of the main memory address) which identifies which particular block is being stored

Locality of reference implies that future references will likely come from this block of memory, so that cache line will probably be utilized repeatedly.

The proportion of memory references, which are found already stored in cache, is called the hit ratio.

Cache memory is intended to give memory speed approaching that of the fastest memories available, and at the same time provide a large memory size at the price of less expensive types of semiconductor memories. There is a relatively large and slow main memory together with a smaller, faster cache memory contains a copy of portions of main memory.
When the processor attempts to read a word of memory, a check is made to determine if the word is in the cache. If so, the word is delivered to the processor. If not, a block of main memory, consisting of fixed number of words is read into the cache and then the word is delivered to the processor.
The locality of reference property states that over a short interval of time, address generated by a typical program refers to a few localized area of memory repeatedly. So if programs and data which are accessed frequently are placed in a fast memory, the average access time can be reduced. This type of small, fast memory is called cache memory which is placed in between the CPU and the main memory.



When the CPU needs to access memory, cache is examined. If the word is found in cache, it is read from the cache and if the word is not found in cache, main memory is accessed to read word. A block of word containing the one just accessed is then transferred from main memory to cache memory.

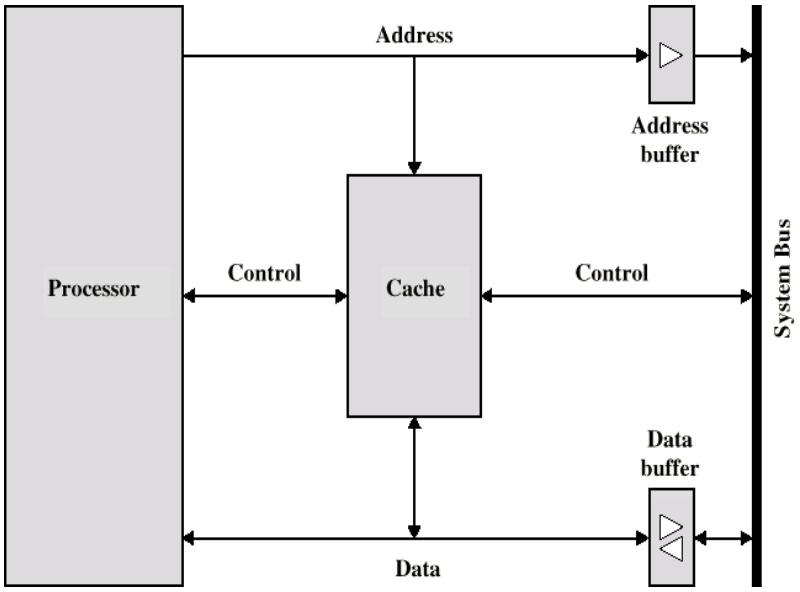


Fig: Typical Cache organization

Cache connects to the processor via data control and address line. The data and address lines also attached to data and address buffer which attached to a system bus from which main memory is reached.

When a cache hit occurs, the data and address buffers are disabled and the communication is only between processor and cache with no system bus traffic. When a cache miss occurs, the desired word is first read into the cache and then transferred from cache to processor. For later case, the cache is physically interposed between the processor and main memory for all data, address and control lines.

Cache Operation Overview

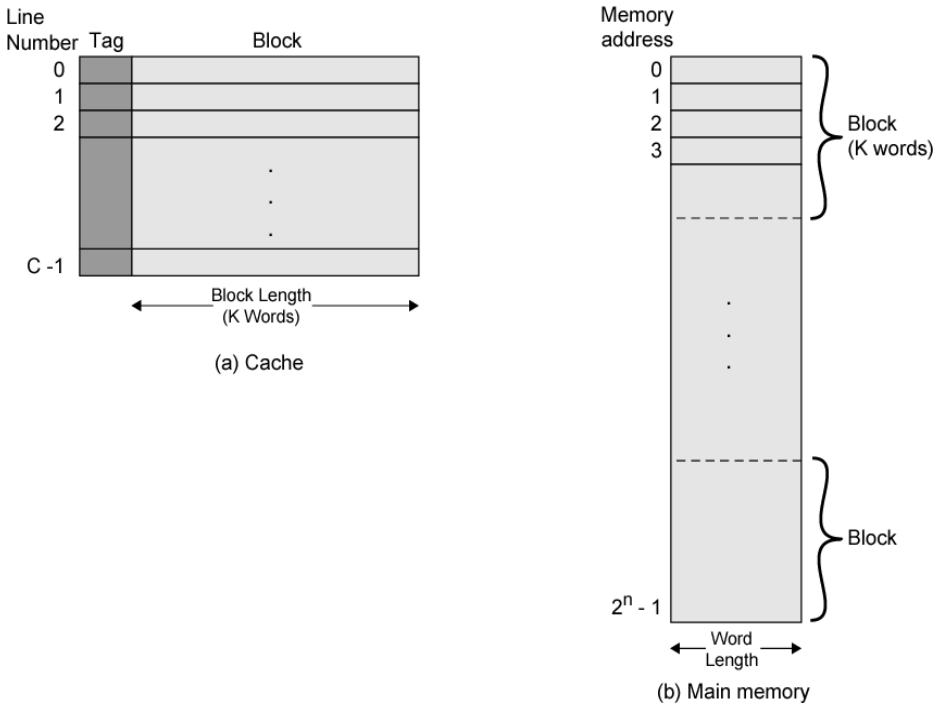


Fig: Cache memory / Main memory structure

CPU generates the receive address (RA) of a word to be moved (read).
Check a block containing RA is in cache.
If present, get from cache (fast) and return.
If not present, access and read required block from main memory to cache.
Allocate cache line for this new found block.
Load bock for cache and deliver word to CPU
Cache includes tags to identify which block of main memory is in each cache slot

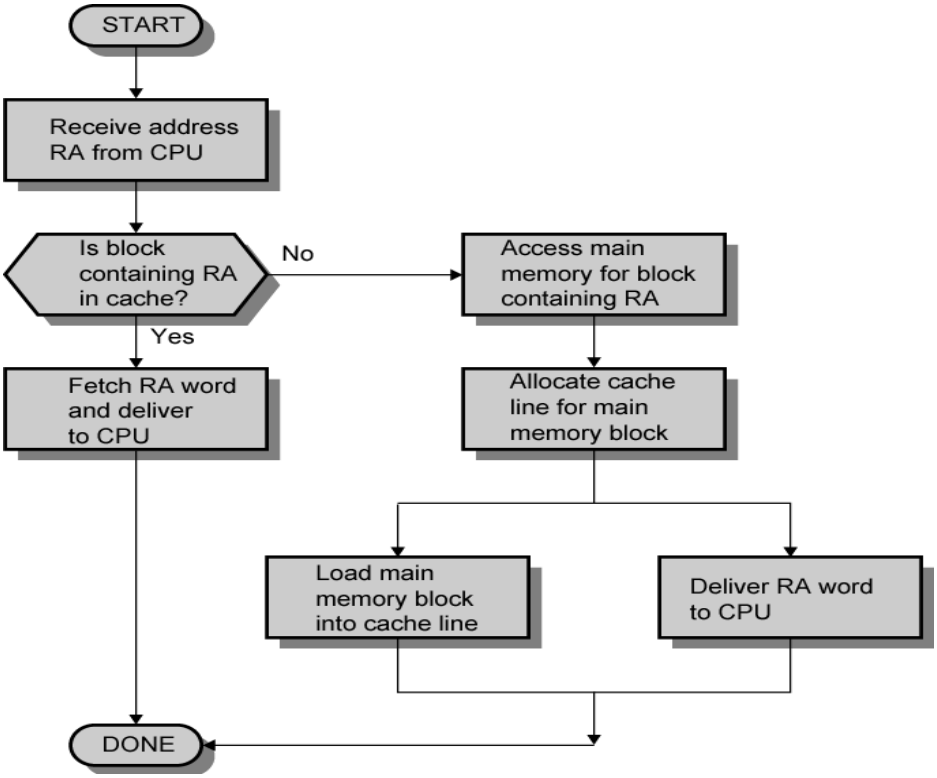


Fig: Flowchart for cache read operation

Locality of Reference

The reference to memory at any given interval of time tends to be confined within a few localized area of memory. This property is called locality of reference. This is possible because the program loops and subroutine calls are encountered frequently. When program loop is executed, the CPU will execute same portion of program repeatedly. Similarly, when a subroutine is called, the CPU fetched starting address of subroutine and executes the subroutine program. Thus loops and subroutine localize reference to memory.

This principle states that memory references tend to cluster over a long period of time, the clusters in use changes but over a short period of time, the processor is primarily working with fixed clusters of memory references.

Spatial Locality

It refers to the tendency of execution to involve a number of memory locations that are clustered.

It reflects tendency of a program to access data locations sequentially, such as when processing a table of data.

Temporal Locality

It refers to the tendency for a processor to access memory locations that have been used frequently. For e.g. Iteration loops executes same set of instructions repeatedly.

Elements of Cache design

Cache size

Size of the cache to be small enough so that the overall average cost per bit is close to that of main memory alone and large enough so that the overall average access time is close to that of the cache alone.

The larger the cache, the larger the number of gates involved in addressing the cache.

Large caches tend to be slightly slower than small ones – even when built with the same integrated circuit technology and put in the same place on chip and circuit board.

The available chip and board also limits cache size.

Mapping function

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

Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines.

There are three different types of mapping functions in common use and are direct, associative and set associative. All the three include following elements in each example.

The cache can hold 64 Kbytes

Data is transferred between main memory and the cache in blocks of 4 bytes each. This means that the cache is organized as 16Kbytes = 2^{14} lines of 4 bytes each.

The main memory consists of 16 Mbytes with each byte directly addressable by a 24 bit address (2^{24} = 16Mbytes). Thus, for mapping purposes, we can consider main memory to consist of 4Mbytes blocks of 4 bytes each.

Direct Mapping

It is the simplex technique, maps each block of main memory into only one possible cache line i.e. a given main memory block can be placed in one and only one place on cache.

i = j modulo m

Where I = cache line number; j = main memory block number; m = number of lines in the cache

The mapping function is easily implemented using the address. For purposes of cache access, each main memory address can be viewed as consisting of three fields.

The least significant w bits identify a unique word or byte within a block of main memory.

The remaining s bits specify one of the 2^s blocks of main memory.

The cache logic interprets these s bits as a tag of (s-r) bits most significant position and a line field of r bits. The latter field identifies one of the m = 2^r lines of the cache.

Tag s-r	Line or Slot r	Word w
8	14	2

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 blocks in main memory = $2^{s+w}/2^w = 2^s$
Number of lines in cache = m = 2^r
Size of tag = (s – r) bits

24 bit address
2 bit word identifier (4 byte block)
22 bit block identifier
8 bit tag (=22-14), 14 bit slot or line
No two blocks in the same line have the same Tag field
Check contents of cache by finding line and checking Tag

Cache line	**Main Memory blocks held**
0	0, m, 2m, 3m…2s-m
1	1,m+1, 2m+1…2s-m+1
m-1	m-1, 2m-1,3m-1…2s-1

Cache Line	**0**	**1**	**2**	**3**	**4**
Main Memory Block	0	1	2	3	4
	5	6	7	8	9
	10	11	12	13	14
	15	16	17	18	19
	20	21	22	23	24

Note that

all locations in a single block of memory have the same higher order bits (call them the block number), so the lower order bits can be used to find a particular word in the block.
within those higher-order bits, their lower-order bits obey the modulo mapping given above (assuming that the number of cache lines is a power of 2), so they can be used to get the cache line for that block
the remaining bits of the block number become a tag, stored with each cache line, and used to distinguish one block from another that could fit into that same cache

line.

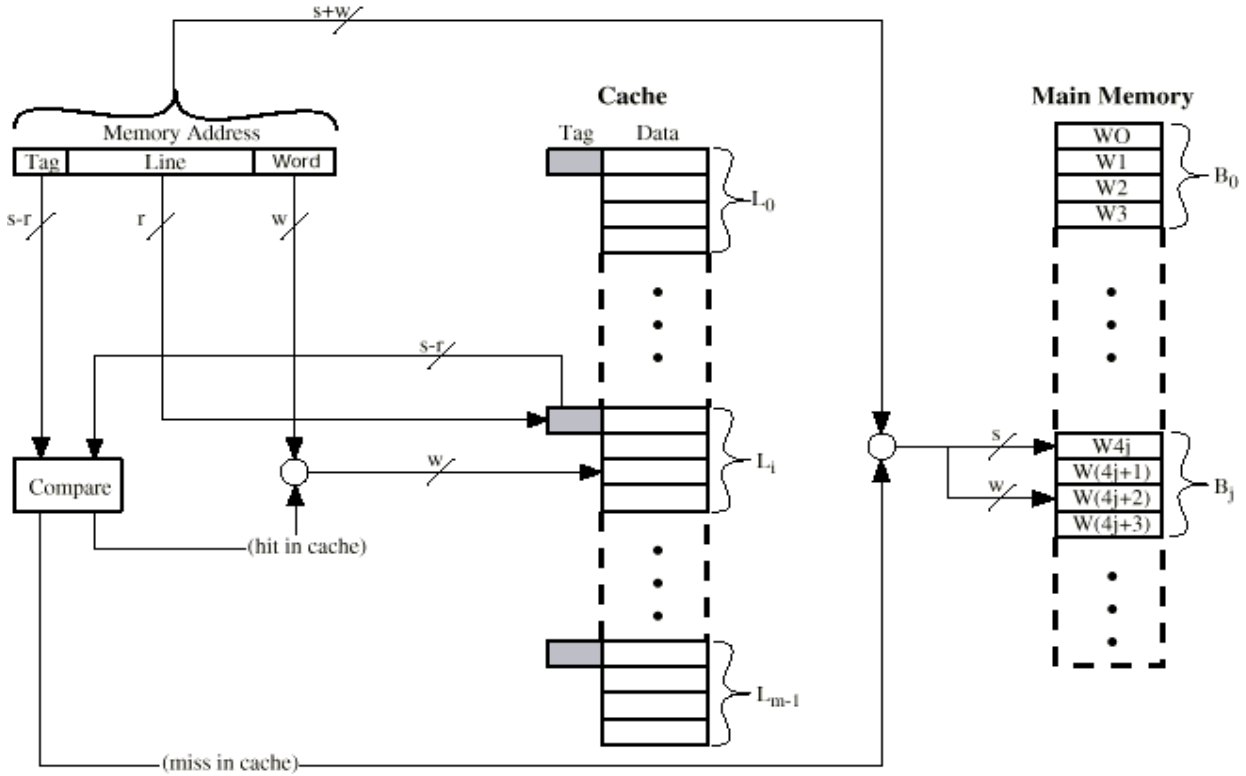
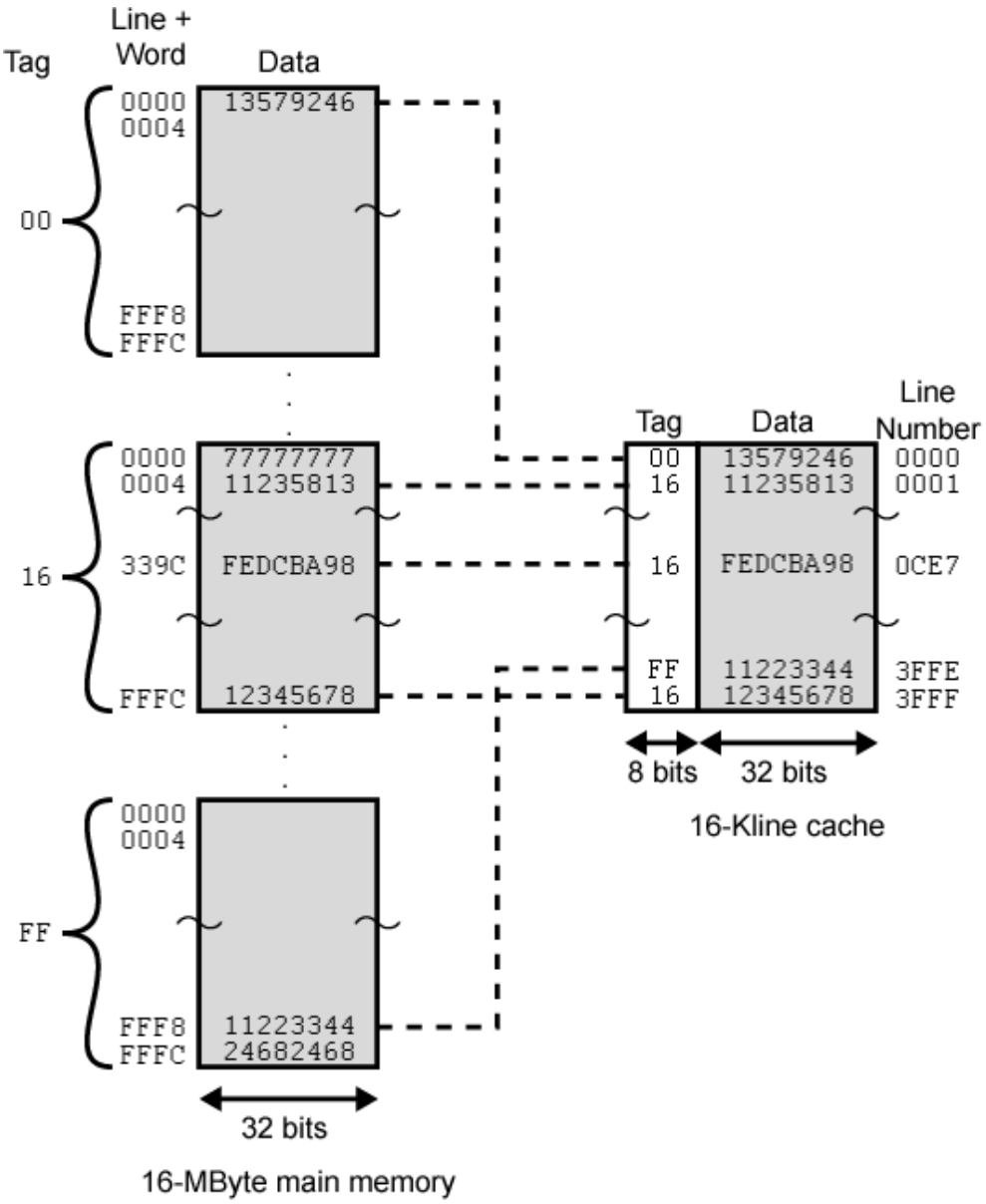


Fig: Direct mapping structure



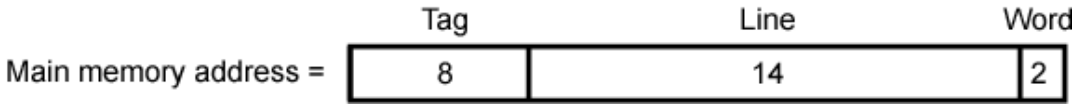


Fig: Direct mapping example

Pros and Cons

Simple
Inexpensive
Fixed location for given block

- If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high

Associated Mapping

It overcomes the disadvantage of direct mapping by permitting each main memory block to be loaded into any line of cache.

Cache control logic interprets a memory address simply as a tag and a word field

Tag uniquely identifies block of memory

Cache control logic must simultaneously examine every line's tag for a match which requires fully associative memory

very complex circuitry, complexity increases exponentially with size

Cache searching gets expensive

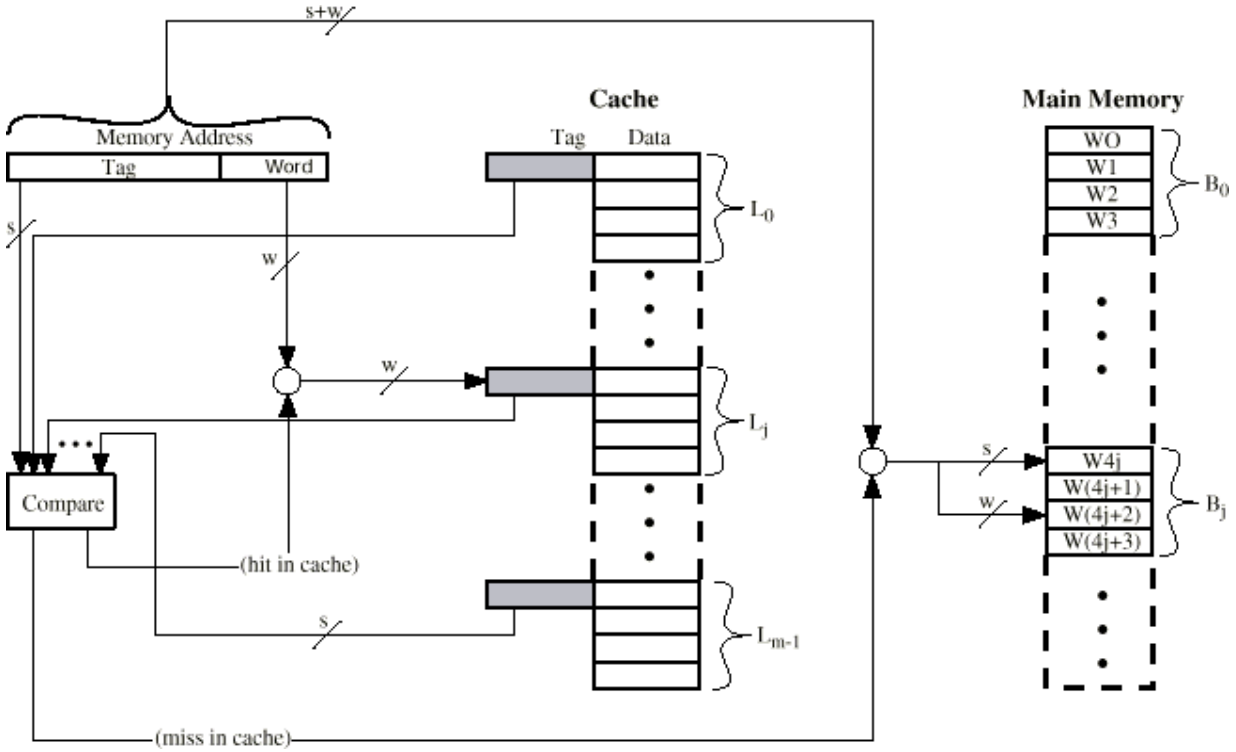


Fig: Associative structure

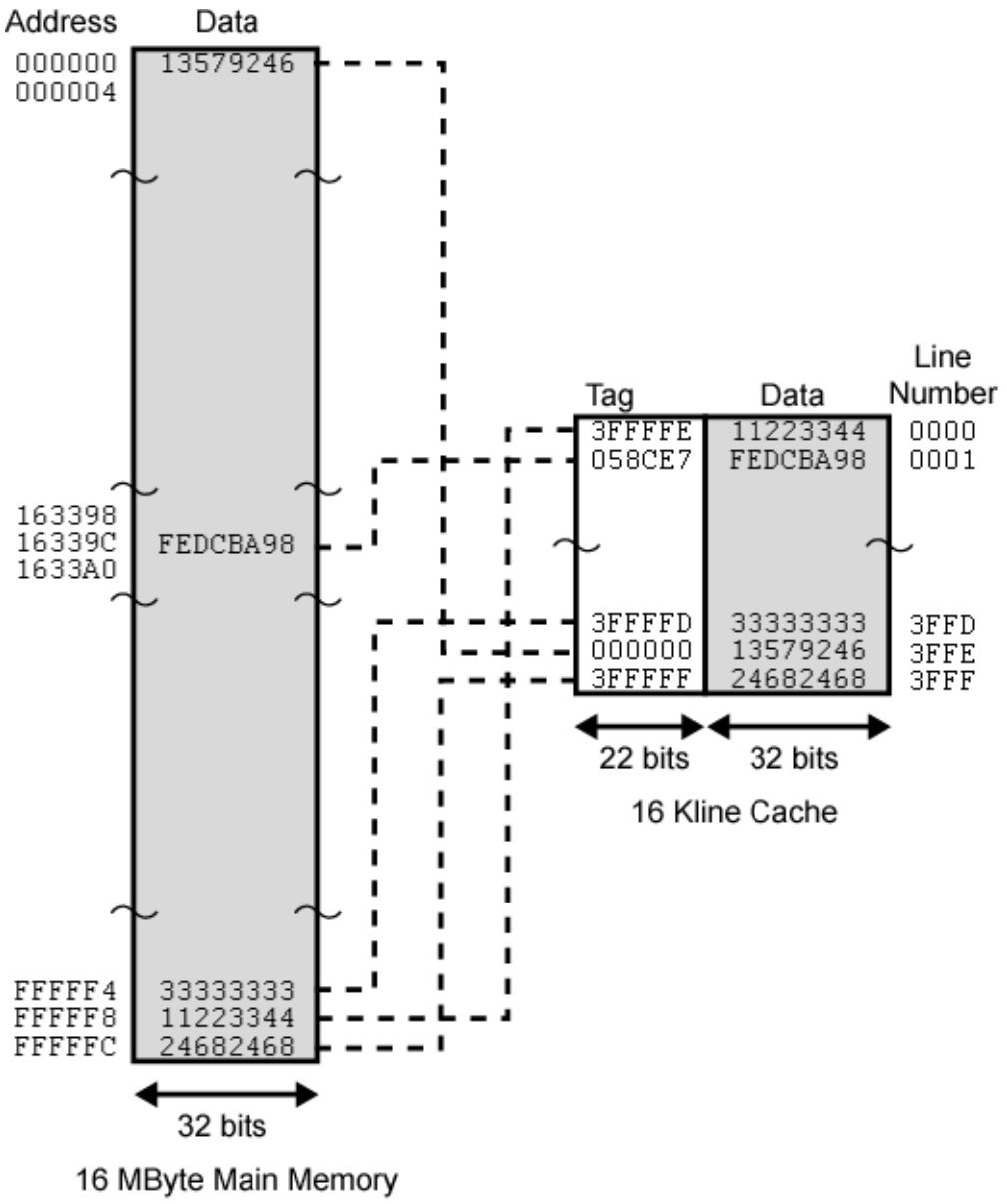
Address length = (s + w) bits

Number of addressable units = 2^{s+w} words or bytes

Block size = line size = 2^w words or bytes

Number of blocks in main memory = $2^{s+w}/2^w = 2^s$

Number of lines in cache = undetermined, Size of tag = s bits



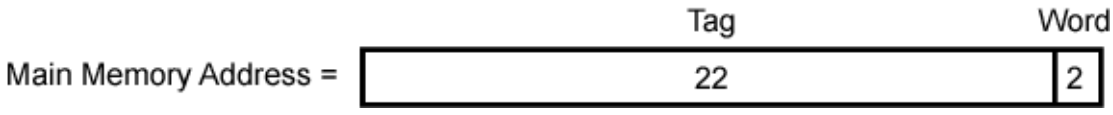


Fig: Associative mapping example

22 bit tag stored with each 32 bit block of data
Compare tag field with tag entry in cache to check for hit
Least significant 2 bits of address identify which 16 bit word is required from 32 bit data block
e.g.

Address	Tag	Data	Cache line
FFFFFC	FFFFFC	24682468	3FFF

Set Associated Mapping

It is a compromise between direct and associative mappings that exhibits the strength and reduces the disadvantages

Cache is divided into v sets, each of which has k lines; number of cache lines = vk

$M = v \times k$

$I = j \text{ modulo } v$

Where, i = cache set number; j = main memory block number; m = number of lines in the cache

So a given block will map directly to a particular set, but can occupy any line in that set (associative mapping is used within the set)

Cache control logic interprets a memory address simply as three fields tag, set and word. The d set bits specify one of $v = 2^d$ sets. Thus s bits of tag and set fields specify one of the 2^s block of main memory.

The most common set associative mapping is 2 lines per set, and is called two-way set associative. It significantly improves hit ratio over direct mapping, and the associative hardware is not too expensive.

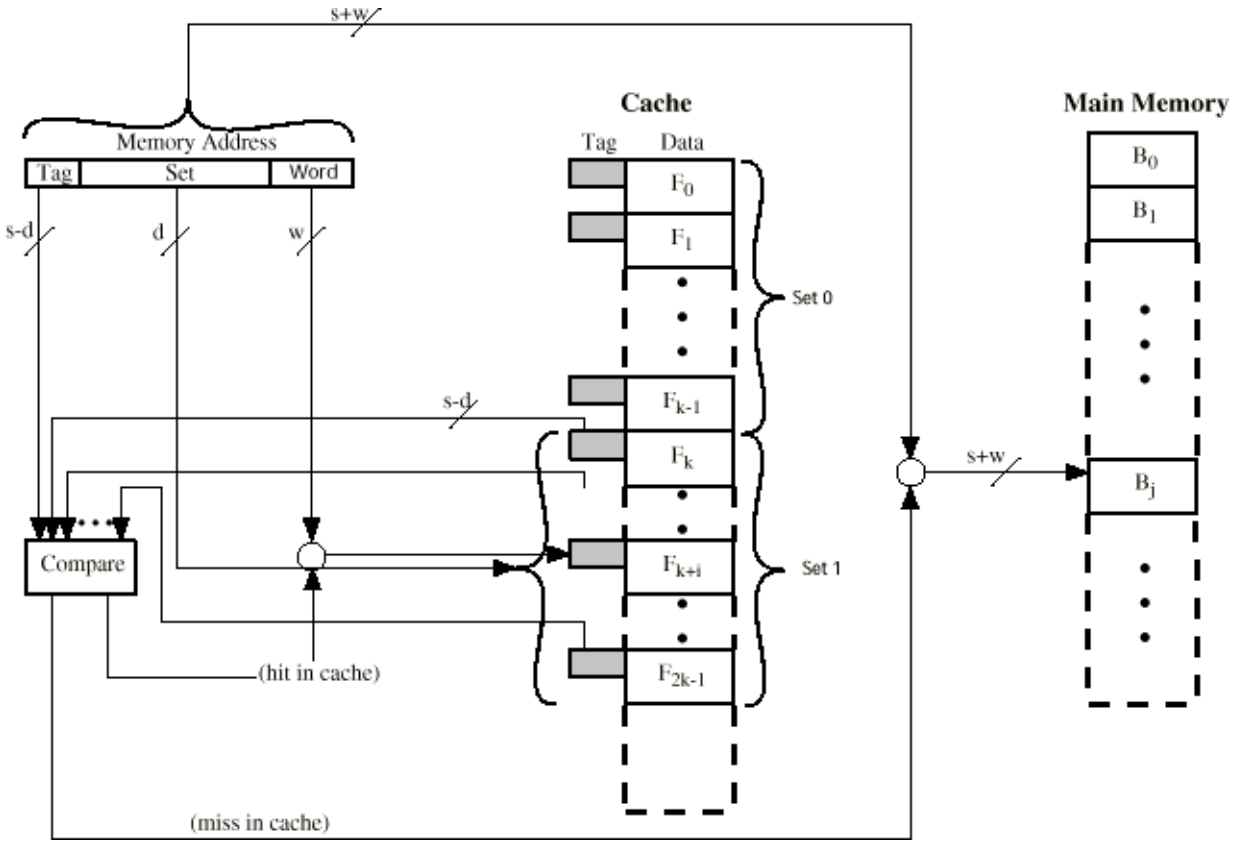
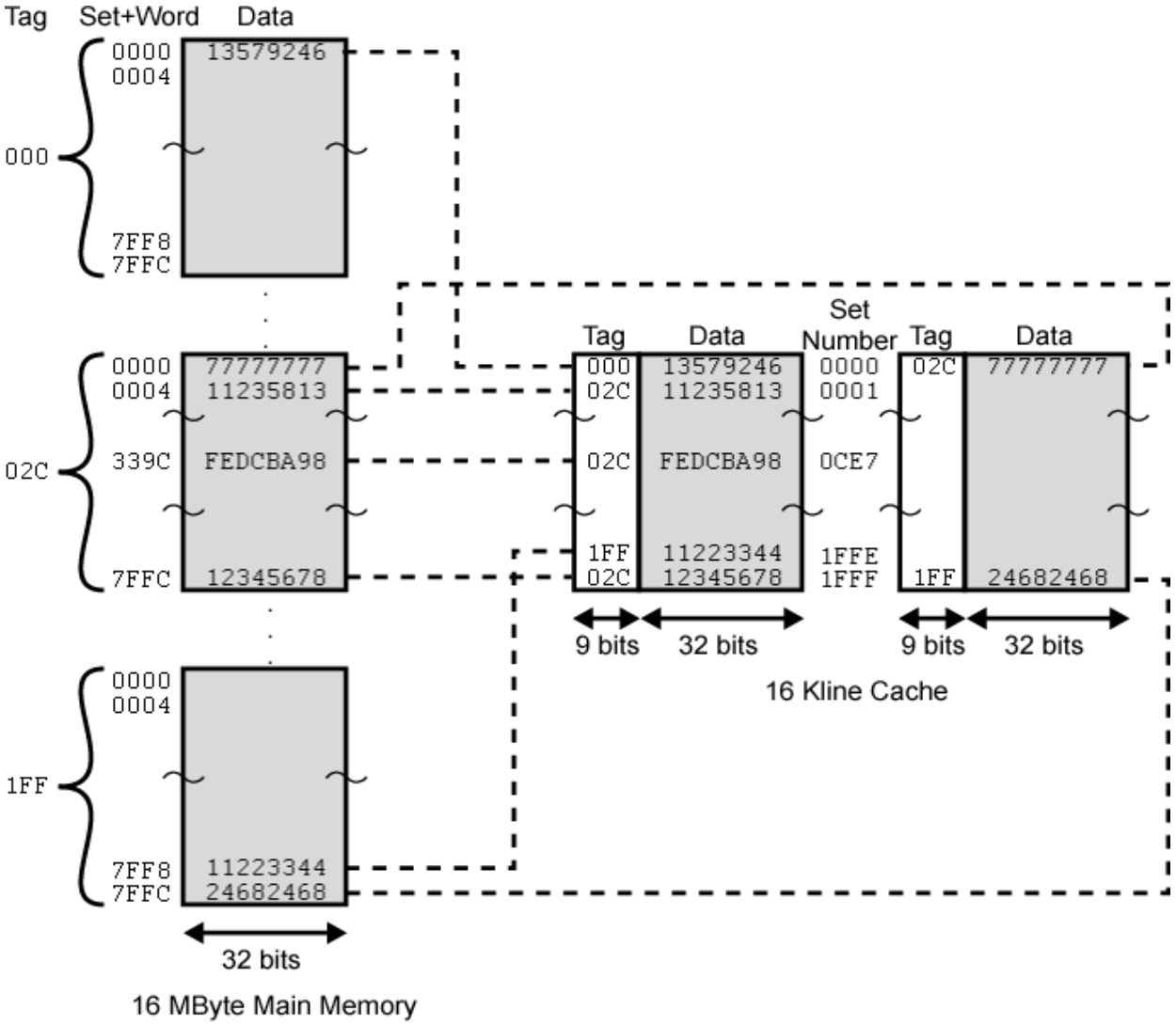


Fig: Set associative mapping structure

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 blocks in main memory = 2^d
Number of lines in set = k
Number of sets = v = 2^d
Number of lines in cache = kv = k * 2^d
Size of tag = (s – d) bits



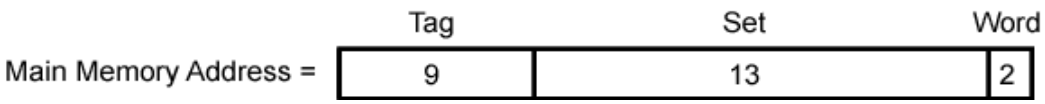


Fig: Set associative mapping example

13 bit set number
Block number in main memory is modulo 2^{13}

000000, 00A000, 00B000, 00C000 … map to same set
Use set field to determine cache set to look in
Compare tag field to see if we have a hit

e.g Address

	Tag	Data	Set number
1FF 7FFC	1FF	12345678	1FFF
001 7FFC	001	11223344	1FFF

Replacement algorithm

When all lines are occupied, bringing in a new block requires that an existing line be overwritten.

Direct mapping

No choice possible with direct mapping
Each block only maps to one line
Replace that line

Associative and Set Associative mapping

Algorithms must be implemented in hardware for speed

Least Recently used (LRU)

replace that block in the set which has been in cache longest with no reference to it

Implementation: with 2-way set associative, have a USE bit for each line in a set. When a block is read into cache, use the line whose USE bit is set to 0, then set its USE bit to one and the other line's USE bit to 0.

Probably the most effective method

First in first out (FIFO)

replace that block in the set which has been in the cache longest

Implementation: use a round-robin or circular buffer technique (keep up with which slot's "turn" is next

Least-frequently-used (LFU)

replace that block in the set which has experienced the fewest references or hits

Implementation: associate a counter with each slot and increment when used

Random

replace a random block in the set

Interesting because it is only slightly inferior to algorithms based on usage

Write policy

When a line is to be replaced, must update the original copy of the line in main memory if any addressable unit in the line has been changed

If a block has been altered in cache, it is necessary to write it back out to main memory before replacing it with another block (writes are about 15% of memory references)

Must not overwrite a cache block unless main memory is up to date

I/O modules may be able to read/write directly to memory

Multiple CPU's may be attached to the same bus, each with their own cache

Write Through

All write operations are made to main memory as well as to cache, so main memory is always valid

Other CPU's monitor traffic to main memory to update their caches when needed

This generates substantial memory traffic and may create a bottleneck

Anytime a word in cache is changed, it is also changed in main memory

Both copies always agree

Generates lots of memory writes to main memory

Multiple CPUs can monitor main memory traffic to keep local (to CPU) cache up to date

Lots of traffic

Slows down writes

Remember bogus write through caches!

Write back

When an update occurs, an UPDATE bit associated with that slot is set, so when the block is replaced it is written back first

During a write, only change the contents of the cache

Update main memory only when the cache line is to be replaced

Causes "cache coherency" problems -- different values for the contents of an address are in the cache and the main memory

Complex circuitry to avoid this problem

Accesses by I/O modules must occur through the cache

Multiple caches still can become invalidated, unless some cache coherency system is used. Such systems include:

Bus Watching with Write Through - other caches monitor memory writes by other caches (using write through) and invalidates their own cache line if a match

Hardware Transparency - additional hardware links multiple caches so that writes to one cache are made to the others

Non-cacheable Memory - only a portion of main memory is shared by more than one processor, and it is non-cacheable

Number of caches L1 and L2 Cache

On-chip cache (L1 Cache)

It is the cache memory on the same chip as the processor, the on-chip cache. It reduces the processor's external bus activity and therefore speeds up execution times and increases overall system performance.

Requires no bus operation for cache hits

Short data paths and same speed as other CPU transactions

Off-chip cache (L2 Cache)

It is the external cache which is beyond the processor. If there is no L2 cache and processor makes an access request for memory location not in the L1 cache, then processor must access DRAM or ROM memory across the bus. Due to this typically slow bus speed and slow memory access time, this results in poor performance. On the other hand, if an L2 SRAM cache is used, then frequently the missing information can be quickly retrieved.

It can be much larger

It can be used with a local bus to buffer the CPU cache-misses from the system bus

Unified and Split Cache

Unified Cache

Single cache contains both instructions and data. Cache is flexible and can balance “allocation” of space to instructions or data to best fit the execution of the program.

Has a higher hit rate than split cache, because it automatically balances load between data and instructions (if an execution pattern involves more instruction fetches than data fetches, the cache will fill up with more instructions than data)

Only one cache need be designed and implemented

Split Cache

Cache splits into two parts first for instruction and second for data. Can outperform unified cache in systems that support parallel execution and pipelining (reduces cache contention)

Trend is toward split cache because of superscalar CPU’s

Better for pipelining, pre-fetching, and other parallel instruction execution designs

Eliminates cache contention between instruction processor and the execution unit (which uses data)

UNIT-V I/O ORGANIZATION

Peripheral devices

- In addition to the processor and a set of memory modules, the third key element of a computer system is a set of input-output subsystem referred to as I/O, provides an efficient mode of communication between the central system and the outside environment.
- Programs and data must be entered into computer memory for processing and results obtained from computations must be recorded or displayed for the user.
- Devices that are under the direct control of the computer are said to be connected on-line. These devices are designed to read information into or out of the memory unit upon command from CPU.
- Input or output devices attached to the computer are also called peripherals.
- Among the most common peripherals are keyboards, display units, and printers.
- Perhaps those provide auxiliary storage for the systems are magnetic disks and tapes.
- Peripherals are electromechanical and electromagnetic devices of some complexity.
- We can broadly classify peripheral devices into three categories:
    - **Human Readable**: Communicating with the computer users, e.g. video display terminal, printers etc.
    - **Machine Readable**: Communicating with equipments, e.g. magnetic disk, magnetic tape, sensor, actuators used in robotics etc.
    - **Communication**: Communicating with remote devices means exchanging data with that, e.g. modem, NIC (network interface Card) etc.

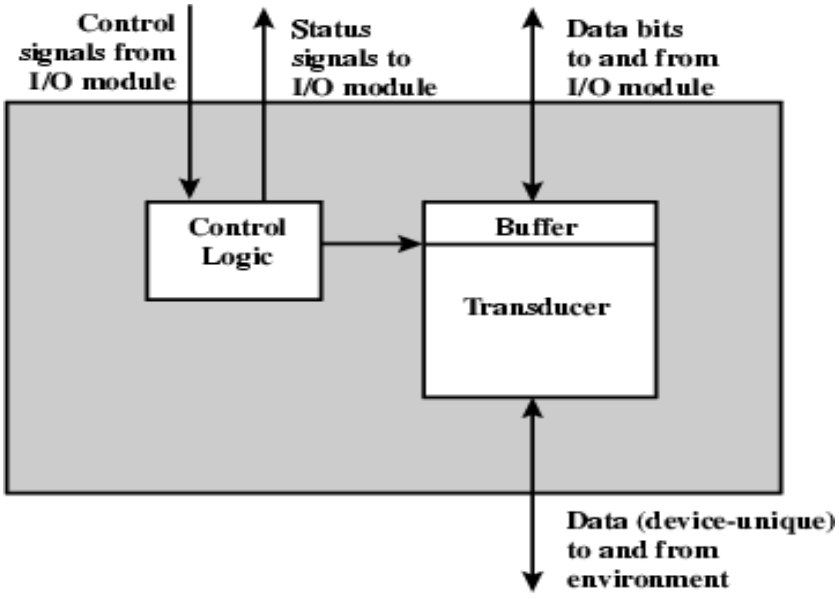


Fig: Block diagram of Peripheral device

- Control signals determine the function that the device will perform such as send data to I/O module, accept data from I/O module.
- Status signals indicate the state of the device i.e. device is ready or not.
- Data bits are actual data transformation.

- Control logic associated with the device controls the device's operation in response to direction from the I/O module.
- The transducer converts data from electrical to other forms of energy during output and from other forms to electrical during input.
- Buffer is associated with the transducer to temporarily hold data being transferred between the I/O module and external devices i.e. peripheral environment.

Input Device

- Keyboard
- Optical input devices
  - Card Reader
  - Paper Tape Reader
  - Optical Character Recognition (OCR)
  - Optical Bar code reader (OBR)
  - Digitizer
  - Optical Mark Reader
- Magnetic Input Devices
  - Magnetic Stripe Reader
  - Magnetic Ink Character Recognition (MICR)
- Screen Input Devices
  - Touch Screen
  - Light Pen
  - Mouse
- Analog Input Devices

Output Device

- Card Puncher, Paper Tape Puncher
- Monitor (CRT, LCD, LED)
- Printer (Impact, Ink Jet, Laser, Dot Matrix)
- Plotter
- Analog
- Voice

7.2 I/O modules

- I/O modules interface to the system bus or central switch (CPU and Memory), interfaces and controls to one or more peripheral devices. I/O operations are accomplished through a wide assortment of external devices that provide a means of exchanging data between external environment and computer by a link to an I/O module. The link is used to exchange control status and data between I/O module and the external devices.

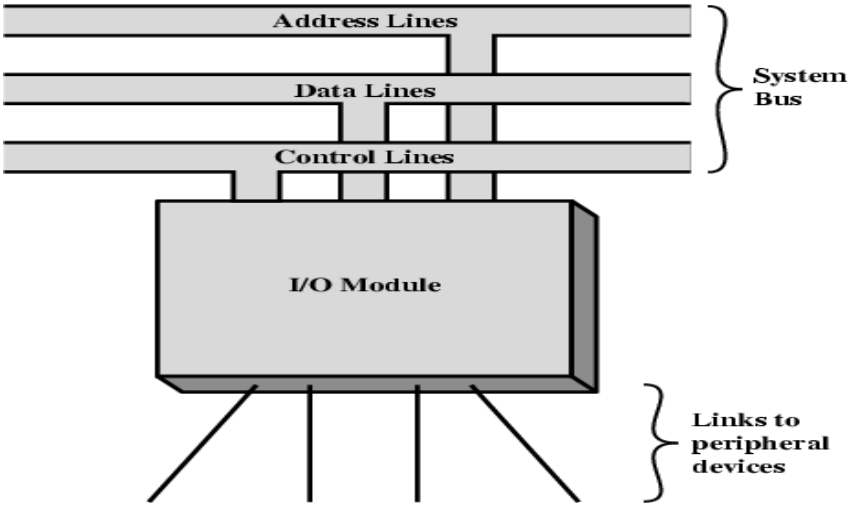


Fig: Model of I/O module

- Peripherals are not directly connected to the system bus instead an I/O module is used which contains logic for performing a communication between the peripherals and the system bus. The reasons due to which peripherals do not directly connected to the system bus are:
    - There are a wide variety of peripherals with various methods of operation. It would be impractical to incorporate the necessary logic within the processor to control a range of devices.
    - The data transfer rate of peripherals is often much slower than that of the memory or processor. Thus, it is impractical to use high speed system bus to communicate directly with a peripheral and vice versa.
    - Peripherals often use different data format and word length than the computer to which they are connected.
- Thus an I/O module is required which performs two major functions.
    - Interface to the processor and memory via the system bus
    - Interface to one or more peripherals by tailored data links

I/O Module Functions

- The I/O module is a special hardware component interface between the CPU and peripherals to supervise and synchronize all I/O transformation The detailed functions of I/O modules are;

  Control & Timing: I/O module includes control and timing to coordinate the flow of traffic between internal resources and external devices. The control of the transfer of data from external devices to processor consists following steps:
    - The processor interrogates the I/O module to check status of the attached device.
    - The I/O module returns the device status.
    - If the device is operational and ready to transmit, the processor requests the transfer of data by means of a command to I/O module.
    - The I/O module obtains the unit of data from the external device.
    - The data are transferred from the I/O module to the processor.

  Processor Communication: I/O module communicates with the processor which involves:

- Command decoding: I/O module accepts commands from the processor.
- Data: Data are exchanged between the processor and I/O module over the bus.
- Status reporting: Peripherals are too slow and it is important to know the status of I/O module.
- Address recognition: I/O module must recognize one unique address for each peripheral it controls.

Device Communication: It involves commands, status information and data.

Data Buffering: I/O module must be able to operate at both device and memory speeds. If the I/O device operates at a rate higher than the memory access rate, then the I/O module performs data buffering. If I/O devices rate slower than memory, it buffers data so as not to tie up the memory in slower transfer operation.

Error Detection: I/O module is responsible for error detection such as mechanical and electrical malfunction reported by device e.g. paper jam, bad ink track & unintentional changes to the bit pattern and transmission error.

I/O Module Structure

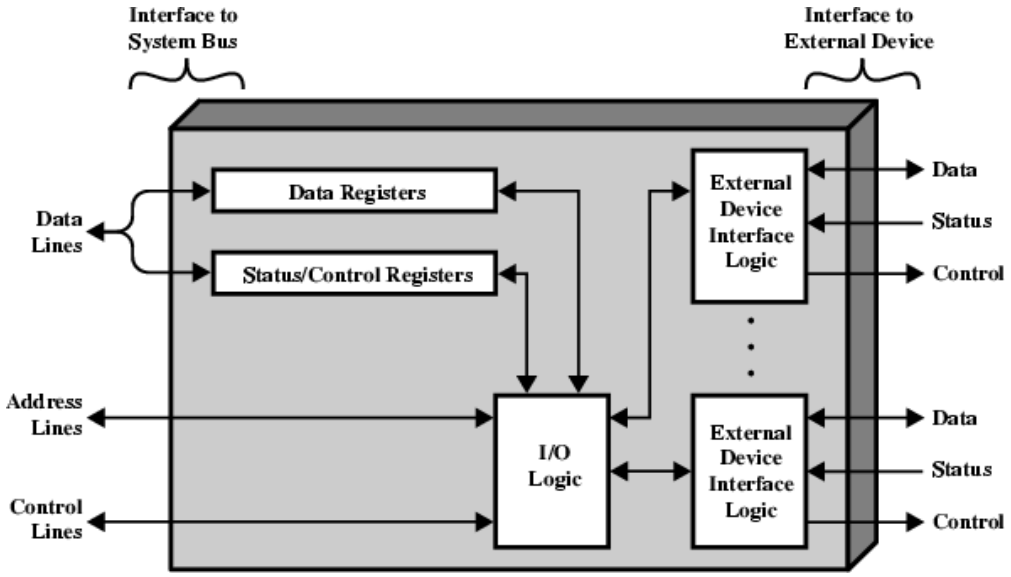


Fig: Block diagram of I/O Module

- The I/O bus from the processor is attached to all peripheral interfaces
- To communicate with the particular devices, the processor places a device address on the address bus.
- Each interface contains an address decoder that monitors the address line. When the interface detects the particular device address, it activates the path between the data line and devices that it controls.
- At the same time that the address is made available in the address line, the processor provides a function code in the control way includes control command, output data and input data.

I/O Module Decisions

- Hide or reveal device properties to CPU
- Support multiple or single device

- Control device functions or leave for CPU
- Also O/S decisions
    - e.g. Unix treats everything it can as a file

7.3 Input-Output interface

- Input-Output interface provides a method for transferring information between internal storage (such as memory and CPU registers) and external I/O devices.
- Peripherals connected to a computer need special communication links for interfacing them with the central processing unit.
- The communication link resolves the following *differences* between the computer and peripheral devices.
    - Devices and signals
      Peripherals - Electromechanical Devices
      CPU or Memory - Electronic Device
    - Data Transfer Rate
      Peripherals - Usually slower
      CPU or Memory - Usually faster than peripherals
      Some kinds of Synchronization mechanism may be needed
    - Unit of Information
      Peripherals - Byte
      CPU or Memory - Word
    - Operating Modes
      Peripherals - Autonomous, Asynchronous
      CPU or Memory – Synchronous
- To resolve these differences, computer systems include special hardware components (Interfaces) between the CPU and peripherals to supervise and synchronize all input and output interfaces.

I/O Bus and Interface Modules

- The I/O bus consists of data lines, address lines and control lines.

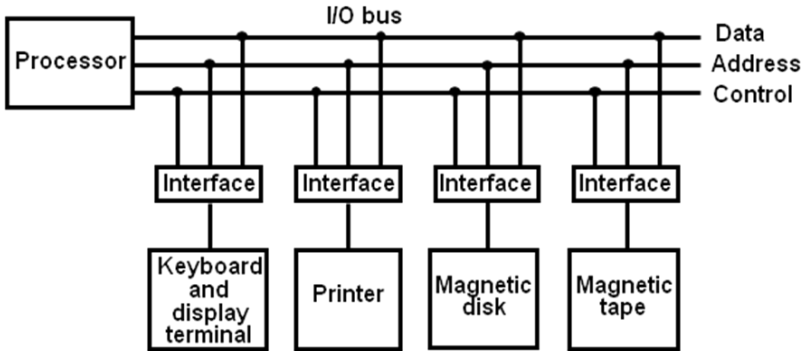


Fig: Connection of I/O bus to input-output devices

- Interface performs the following:
    - Decodes the device address (device code)
    - Decodes the commands (operation)
    - Provides signals for the peripheral controller

  - Synchronizes the data flow and supervises the transfer rate between peripheral and CPU or Memory
- I/O commands that the interface may receive:
  - Control command: issued to activate the peripheral and to inform it what to do.
  - Status command: used to test various status conditions in the interface and the peripheral.
  - Output data: causes the interface to respond by transferring data from the bus into one of its registers.
  - Input data: is the opposite of the data output.

I/O versus Memory Bus

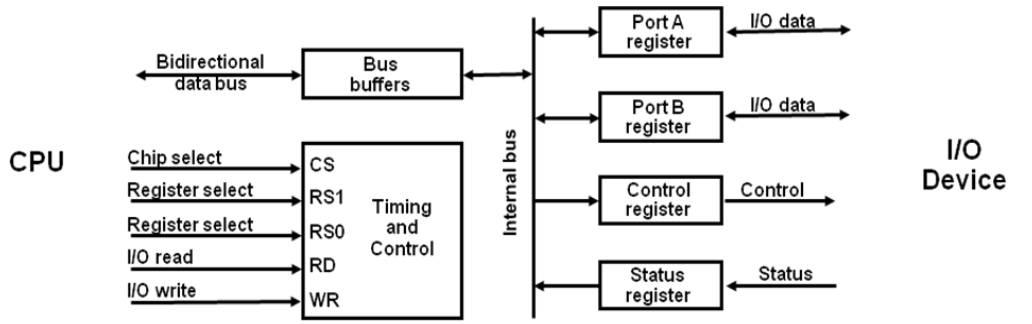
- Computer buses can be used to communicate with memory and I/O in three ways:
  - Use two separate buses, one for memory and other for I/O. In this method, all data, address and control lines would be separate for memory and I/O.
  - Use one common bus for both memory and I/O but have separate control lines. There is a separate read and write lines; I/O read and I/O write for I/O and memory read and memory write for memory.
  - Use a common bus for memory and I/O with common control line. This I/O configuration is called memory mapped.

Isolated I/O versus Memory Mapped I/O

- **Isolated I/O**
  - Separate I/O read/write control lines in addition to memory read/write control lines
  - Separate (isolated) memory and I/O address spaces
  - Distinct input and output instructions

- **Memory-mapped I/O**
  - A single set of read/write control lines (no distinction between memory and I/O transfer)
  - Memory and I/O addresses share the common address space which reduces memory address range available
  - No specific input or output instruction so the same memory reference instructions can be used for I/O transfers
  - Considerable flexibility in handling I/O operations

Example of I/O Interface

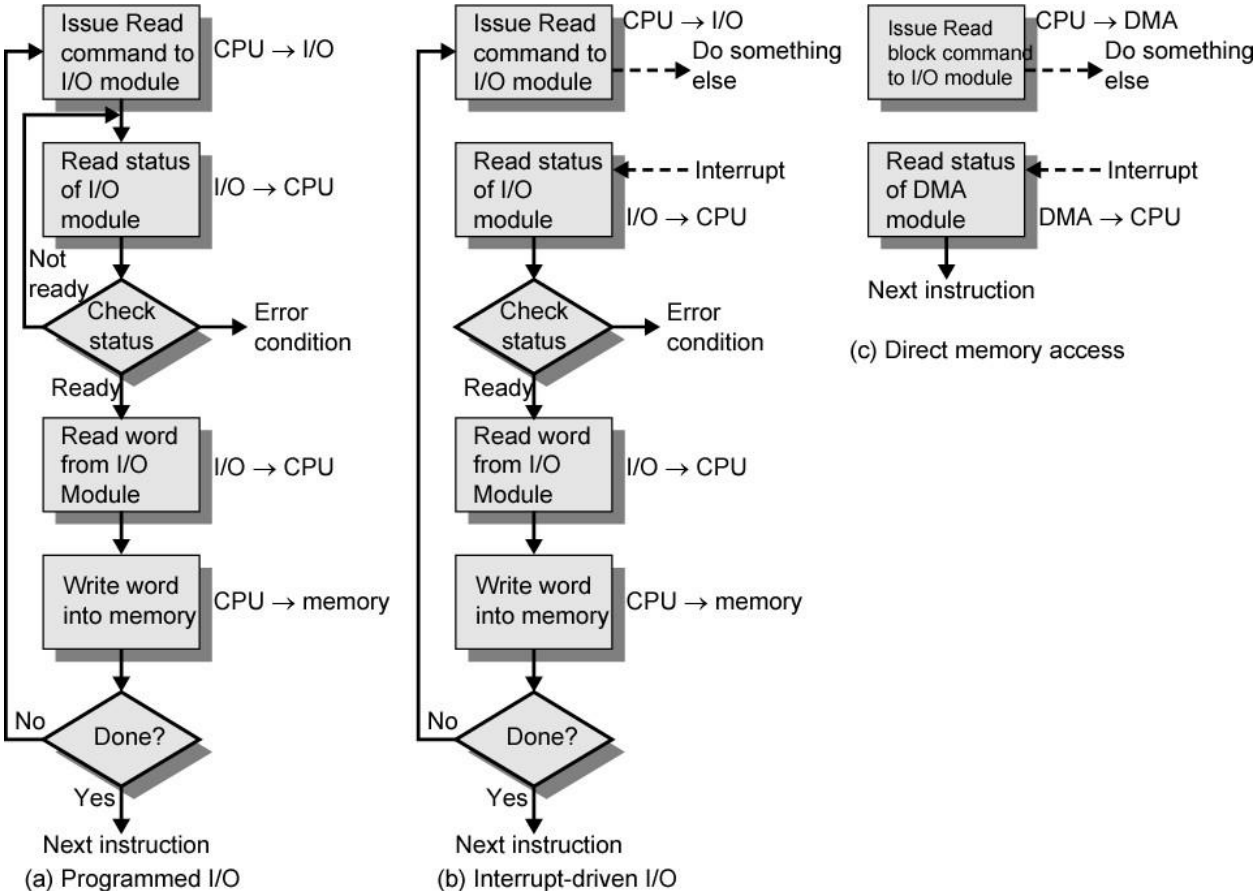


CS	RS1	RS0	Register selected
0	x	x	None - data bus in high-impedence
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

- Information in each port can be assigned a meaning depending on the mode of operation of the I/O device
  - Port A = Data; Port B = Command; Port C = Status
- CPU initializes (loads) each port by transferring a byte to the Control Register
  - Allows CPU can define the mode of operation of each port
  - *Programmable Port*: By changing the bits in the control register, it is possible to change the interface characteristics

7.4 Modes of transfer

- Data Transfer between the central computer and I/O devices may be handled in a variety of modes.
- Some modes use CPU as an intermediate path, others transfer the data directly to and from the memory unit.
- Data transfer to and from peripherals may be handled in one of three possible modes.
  - Programmed I/O
  - Interrupt Driven I/O
  - Direct Memory Access (DMA)



7.4.1 Programmed I/O

- Programmed I/O operations are the result of I/O instructions written in the computer program.
- In programmed I/O, each data transfer in initiated by the instructions in the CPU and hence the CPU is in the continuous monitoring of the interface.
- Input instruction is used to transfer data from I/O device to CPU, store instruction is used to transfer data from CPU to memory and output instruction is used to transfer data from CPU to I/O device.
- This technique is generally used in very slow speed computer and is not a efficient method if the speed of the CPU and I/O is different.

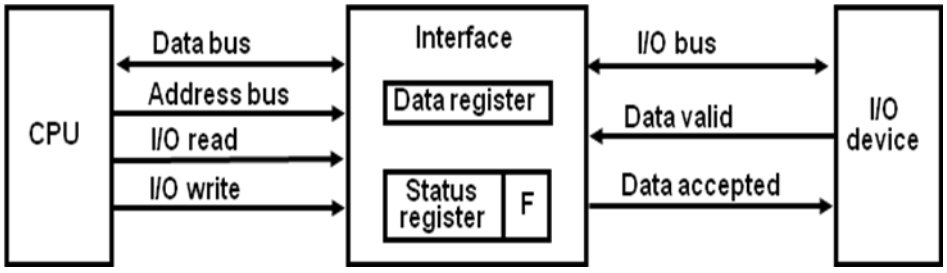


Fig: Data transfer from I/O device to CPU

- I/O device places the data on the I/O bus and enables its data valid signal
- The interface accepts the data in the data register and sets the F bit of status register and also enables the data accepted signal.
- Data valid line is disables by I/O device.
- CPU is in a continuous monitoring of the interface in which it checks the F bit of the status register.
    - If it is set i.e. 1, then the CPU reads the data from data register and sets F bit to zero
    - If it is reset i.e. 0, then the CPU remains monitoring the interface.
- Interface disables the data accepted signal and the system goes to initial state where next item of data is placed on the data bus.

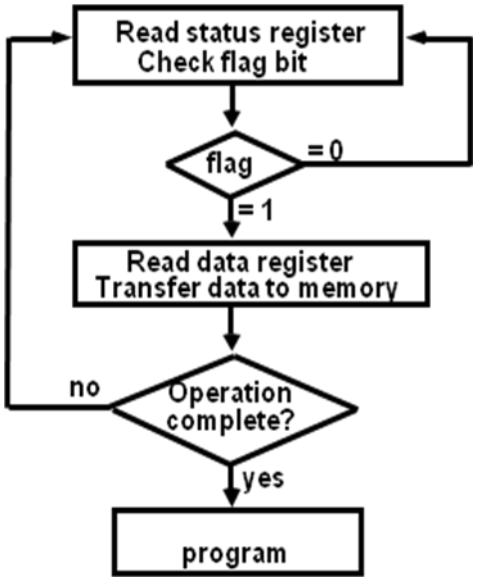


Fig: Flowchart for CPU program to input data

Characteristics:

- Continuous CPU involvement
- CPU slowed down to I/O speed
- Simple
- Least hardware

Polling, or polled operation, in computer science, refers to actively sampling the status of an external device by a client program as a synchronous activity. Polling is most often used in terms of input/output (I/O), and is also referred to as **polled I/O or software driven I/O.**

7.4.2 Interrupt-driven I/O

- Polling takes valuable CPU time
- Open communication only when some data has to be passed -> *Interrupt*.
- I/O interface, instead of the CPU, monitors the I/O device
- When the interface determines that the I/O device is ready for data transfer, it generates an *Interrupt Request* to the CPU
- Upon detecting an interrupt, CPU stops momentarily the task it is doing, branches to the service routine to process the data transfer, and then returns to the task it was performing

The problem with programmed I/O is that the processor has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data. The processor, while waiting, must repeatedly interrogate the status of the I/O module. As a result, the level of the performance of the entire system is severely degraded. An alternative is for the processor to issue an I/O command to a module and then go on to do some other useful work. The I/O module will then interrupt the processor to request service when it is ready to exchange data with processor. The processor then executes the data transfer, and then resumes its former processing. The interrupt can be initiated either by software or by hardware.

Interrupt Driven I/O basic operation

- CPU issues read command
- I/O module gets data from peripheral whilst CPU does other work
- I/O module interrupts CPU
- CPU requests data
- I/O module transfers data

Interrupt Processing from CPU viewpoint

- Issue read command
- Do other work
- Check for interrupt at end of each instruction cycle
- If interrupted:-
    - Save context (registers)
    - Process interrupt
    - Fetch data & store

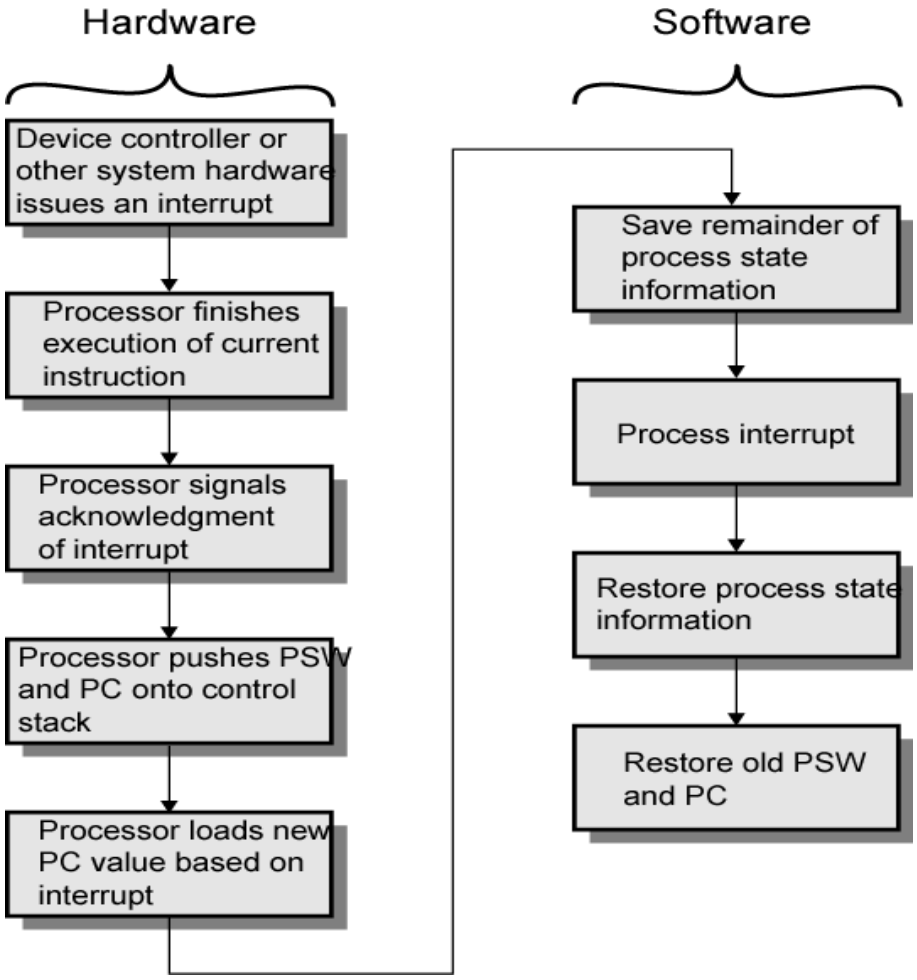


Fig: Simple Interrupt Processing

Priority Interrupt

- Determines which interrupt is to be served first when two or more requests are made simultaneously
- Also determines which interrupts are permitted to interrupt the computer while another is being serviced
- Higher priority interrupts can make requests while servicing a lower priority interrupt

Priority Interrupt by Software (Polling)

- Priority is established by the order of polling the devices (interrupt sources), that is identify the highest-priority source by software means
- One common branch address is used for all interrupts
- Program polls the interrupt sources in sequence
- The highest-priority source is tested first
- Flexible since it is established by software
- Low cost since it needs a very little hardware
- Very slow

Priority Interrupt by Hardware

- Require a priority interrupt manager which accepts all the interrupt requests to determine the highest priority request
- Fast since identification of the highest priority interrupt request is identified by the hardware
- Fast since each interrupt source has its own interrupt vector to access directly to its own service routine

1. Daisy Chain Priority (Serial)

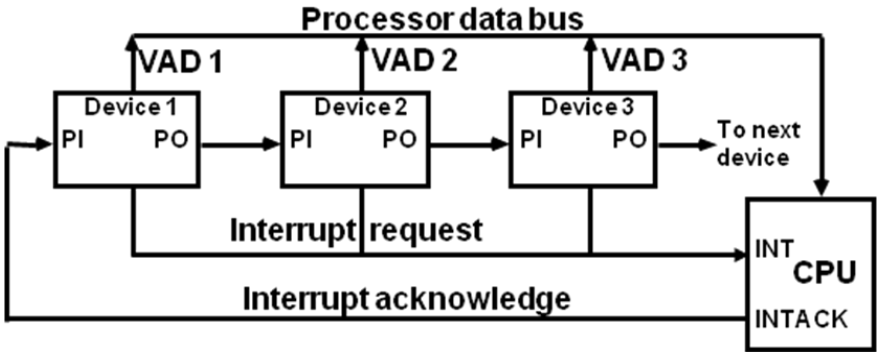
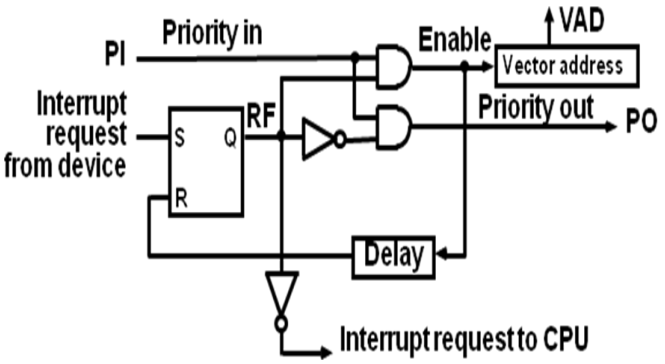


Fig: Daisy Chain priority Interrupt

- Interrupt Request from any device
- CPU responds by INTACK
- Any device receives signal(INTACK) at PI puts the VAD on the bus
- Among interrupt requesting devices the only device which is physically closest to CPU gets INTACK and it blocks INTACK to propagate to the next device



PI	RF	PO	Enable
0	0	0	0
0	1	0	0
1	0	1	0
1	1	1	1

Fig: One stage of Daisy chain priority arrangement

2. Parallel Priority

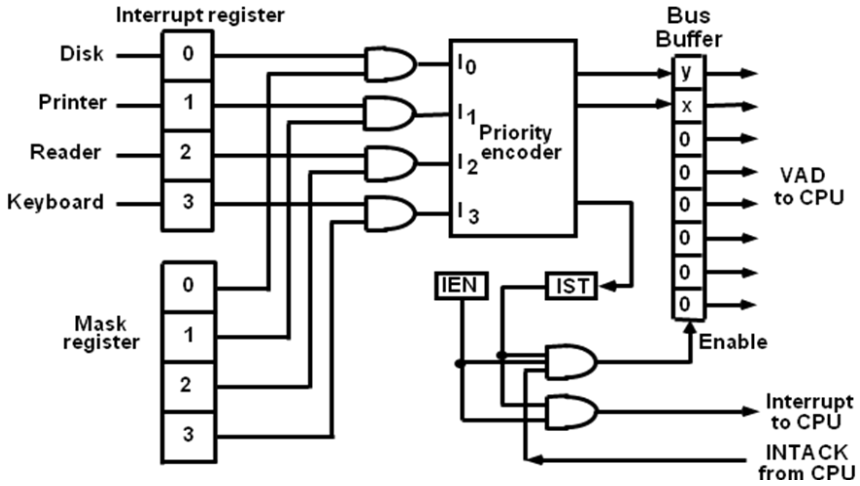


Fig: Parallel priority interrupts hardware

- IEN: Set or Clear by instructions ION or IOF
- IST: Represents an unmasked interrupt has occurred. INTACK enables tristate Bus Buffer to load VAD generated by the Priority Logic
- Interrupt Register:
  - Each bit is associated with an Interrupt Request from different Interrupt Source - different priority level
  - Each bit can be cleared by a program instruction
- Mask Register:
  - Mask Register is associated with Interrupt Register
  - Each bit can be set or cleared by an Instruction

Priority Encoder

- Determines the highest priority interrupt when more than one interrupts take place

Inputs				Outputs			Boolean functions
I_0	I_1	I_2	I_3	x	y	IST	
1	d	d	d	0	0	1	
0	1	d	d	0	1	1	
0	0	1	d	1	0	1	$x = I_0' I_1'$
0	0	0	1	1	1	1	$y = I_0' I_1 + I_0' I_2'$
0	0	0	0	d	d	0	$(IST) = I_0 + I_1 + I_2 + I_3$

Fig: Priority Encoder Truth Table

Interrupt Cycle

At the end of each Instruction cycle

- CPU checks IEN and IST
- If IEN and IST = 1, CPU -> Interrupt Cycle
    - SP ← SP – 1; Decrement stack pointer
    - M[SP] ← PC; Push PC into stack
    - INTACK ← 1; Enable interrupt acknowledge
    - PC ← VAD; Transfer vector address to PC
    - IEN ← 0; Disable further interrupts
    - Go To Fetch to execute the first instruction in the interrupt service routine

7.4.3 Direct Memory access

- Large blocks of data transferred at a high speed to or from high speed devices, magnetic drums, disks, tapes, etc.
- DMA controller Interface that provides I/O transfer of data directly to and from the memory and the I/O device
- CPU initializes the DMA controller by sending a memory address and the number of words to be transferred
- Actual transfer of data is done directly between the device and memory through DMA controller -> Freeing CPU for other tasks

The transfer of data between the peripheral and memory without the interaction of CPU and letting the peripheral device manage the memory bus directly is termed as Direct Memory Access (DMA).

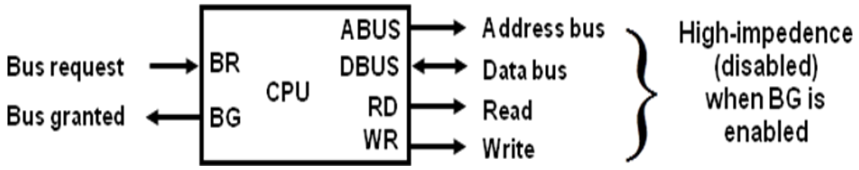


Fig: CPU bus signal for DMA transfer

The two control signals Bus Request and Bus Grant are used to fascinate the DMA transfer. The bus request input is used by the DMA controller to request the CPU for the control of the buses. When BR signal is high, the CPU terminates the execution of the current instructions and then places the address, data, read and write lines to the high impedance state and sends the bus grant signal. The DMA controller now takes the control of the buses and transfers the data directly between memory and I/O without processor interaction. When the transfer is completed, the bus request signal is made low by DMA. In response to which CPU disables the bus grant and again CPU takes the control of address, data, read and write lines.

The transfer of data between the memory and I/O of course facilitates in two ways which are DMA Burst and Cycle Stealing.

DMA Burst: The block of data consisting a number of memory words is transferred at a time.

Cycle Stealing: DMA transfers one data word at a time after which it must return control of the buses to the CPU.

- CPU is usually much faster than I/O (DMA), thus CPU uses the most of the memory cycles
- DMA Controller steals the memory cycles from CPU
- For those stolen cycles, CPU remains idle
- For those slow CPU, DMA Controller may steal most of the memory cycles which may cause CPU remain idle long time

DMA Controller

The DMA controller communicates with the CPU through the data bus and control lines. DMA select signal is used for selecting the controller, the register select is for selecting the register. When the bus grant signal is zero, the CPU communicates through the data bus to read or write into the DMA register. When bus grant is one, the DMA controller takes the control of buses and transfers the data between the memory and I/O.

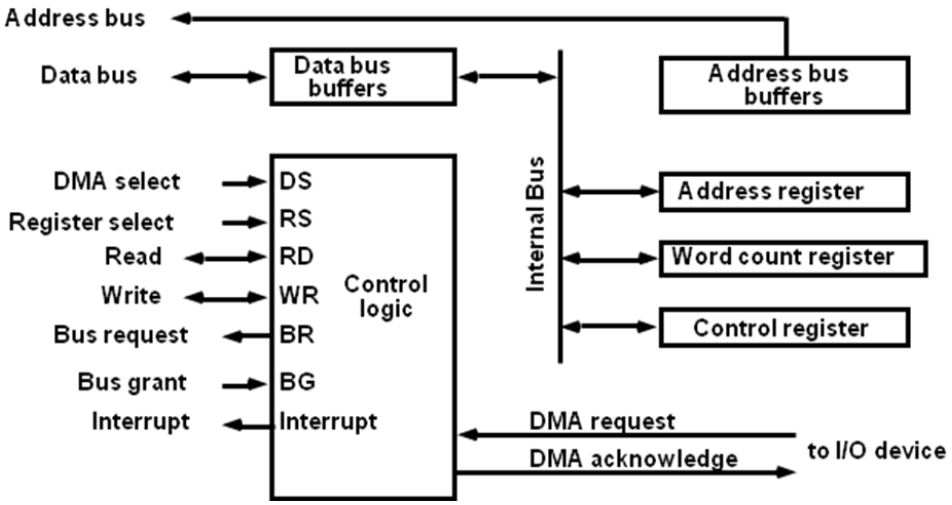


Fig: Block diagram of DMA controller

The address register specifies the desired location of the memory which is incremented after each word is transferred to the memory. The word count register holds the number of words to be transferred which is decremented after each transfer until it is zero. When it is zero, it indicates the end of transfer. After which the bus grant signal from CPU is made low and CPU returns to its normal operation. The control register specifies the mode of transfer which is Read or Write.

DMA Transfer

- DMA request signal is given from I/O device to DMA controller.

- DMA sends the bus request signal to CPU in response to which CPU disables its current instructions and initialize the DMA by sending the following information.
    - The starting address of the memory block where the data are available (for read) and where data to be stored (for write)
    - The word count which is the number of words in the memory block
    - Control to specify the mode of transfer
    - Sends a bust grant as 1 so that DMA controller can take the control of the buses
    - DMA sends the DMA acknowledge signal in response to which peripheral device puts the words in the data bus (for write) or receives a word from the data bus (for read).

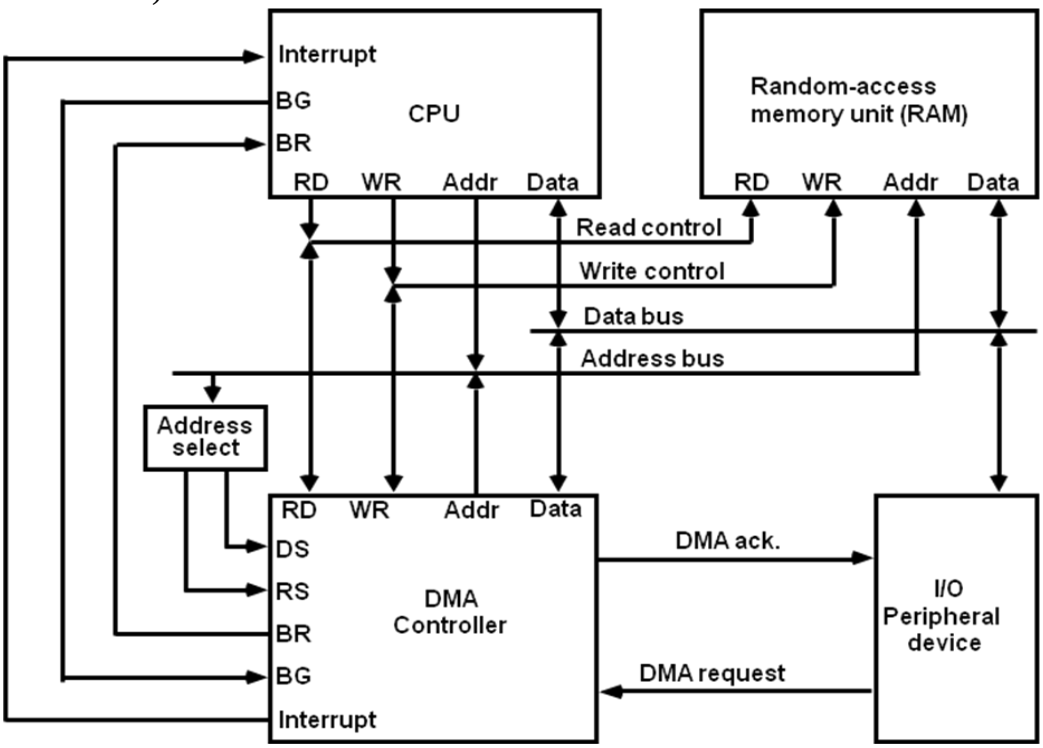


Fig: DMA transfer in a computer system

DMA Operation

- CPU tells DMA controller:-
    - Read/Write
    - Device address
    - Starting address of memory block for data
    - Amount of data to be transferred
- CPU carries on with other work
- DMA controller deals with transfer
- DMA controller sends interrupt when finished

7.5 I/O Processors

- Processor with direct memory access capability that communicates with I/O devices
- Channel accesses memory by cycle stealing

- Channel can execute a Channel Program
- Stored in the main memory
- Consists of Channel Command Word(CCW)
- Each CCW specifies the parameters needed by the channel to control the I/O devices and perform data transfer operations
- CPU initiates the channel by executing a channel I/O class instruction and once initiated, channel operates independently of the CPU

A computer may incorporate one or more external processors and assign them the task of communicating directly with the I/O devices so that no each interface need to communicate with the CPU. An I/O processor (IOP) is a processor with direct memory access capability that communicates with I/O devices. IOP instructions are specifically designed to facilitate I/O transfer. The IOP can perform other processing tasks such as arithmetic logic, branching and code translation.

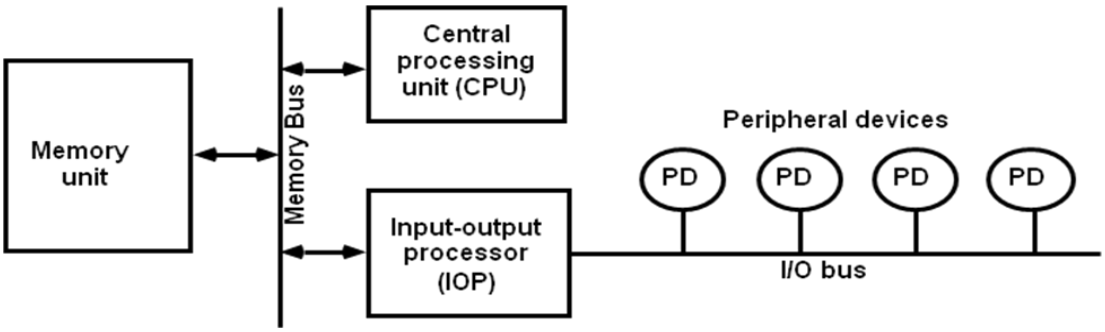


Fig: Block diagram of a computer with I/O Processor

The memory unit occupies a central position and can communicate with each processor by means of direct memory access. The CPU is responsible for processing data needed in the solution of computational tasks. The IOP provides a path for transferring data between various peripheral devices and memory unit.

In most computer systems, the CPU is the master while the IOP is a slave processor. The CPU initiates the IOP and after which the IOP operates independent of CPU and transfer data between the peripheral and memory. For example, the IOP receives 5 bytes from an input device at the device rate and bit capacity. After which the IOP packs them into one block of 40 bits and transfer them to memory. Similarly the O/P word transfer from memory to IOP is directed from the IOP to the O/P device at the device rate and bit capacity.

CPU – IOP Communication

The memory unit acts as a message center where each processor leaves information for the other. The operation of typical IOP is appreciated with the example by which the CPU and IOP communication.

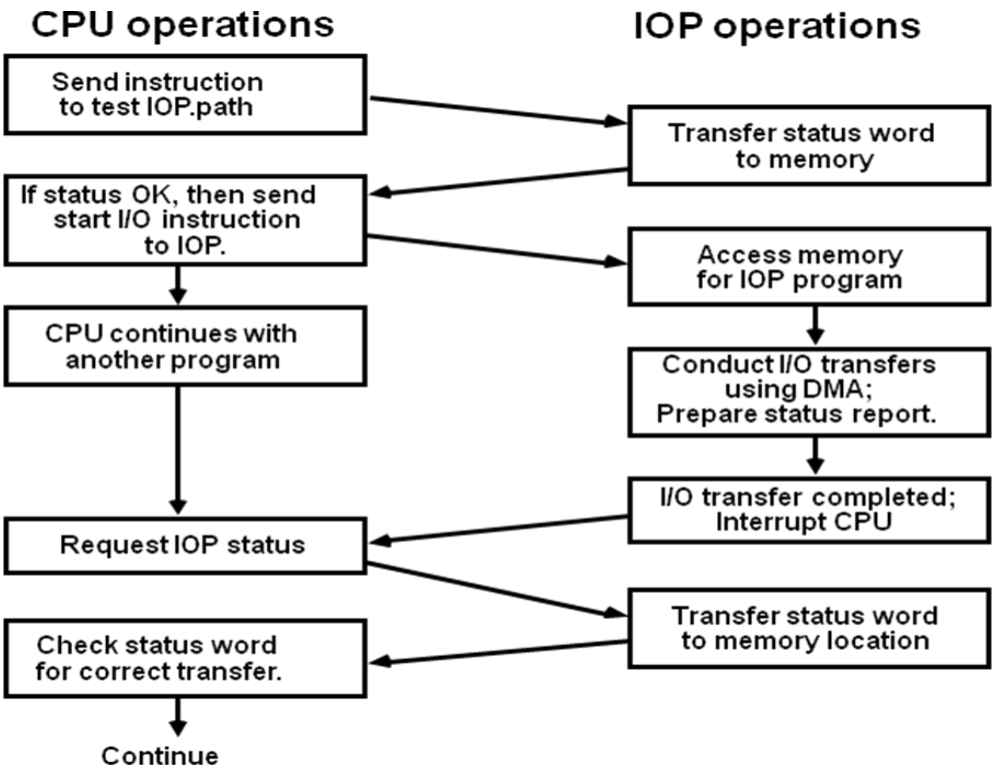


Fig: CPU – IOP communication

- The CPU sends an instruction to test the IOP path.
- The IOP responds by inserting a status word in memory for the CPU to check.
- The bits of the status word indicate the condition of the IOP and I/O device, such as IOP overload condition, device busy with another transfer or device ready for I/O transfer.
- The CPU refers to the status word in in memory to decide what to do next.
- If all right up to this, the CPU sends the instruction to start I/O transfer.
- The CPU now continues with another program while IOP is busy with I/O program.
- When IOP terminates the execution, it sends an interrupt request to CPU.
- CPU responds by issuing an instruction to read the status from the IOP.
- IOP responds by placing the contents to its status report into specified memory location.
- Status word indicates whether the transfer has been completed or with error.

7.6 Data Communication Processor

- Distributes and collects data from many remote terminals connected through telephone and other communication lines.
- Transmission:
  - Synchronous
  - Asynchronous
- Transmission Error:
  - Parity
  - Checksum
  - Cyclic Redundancy Check

    - Longitudinal Redundancy Check
- Transmission Modes:
    - Simples
    - Half Duplex
    - Full Duplex
- Data Link & Protocol

A data communication (command) processor is an I/O processor that distributes and collects data from remote terminals connected through telephone and other communication lines. In processor communication, processor communicates with the I/O device through a common bus i.e. data and control with sharing by each peripherals. In data communication, processor communicates with each terminal through a single pair of wires.

The way that remote terminals are connected to a data communication processor is via telephone lines or other public or private communication facilities. The data communication may be either through synchronous transmission or through asynchronous transmission. One of the functions of data communication processor is check for transmission errors. An error can be detected by checking the parity in each character received. The other ways are checksum, longitudinal redundancy check (LRC) and cyclic redundancy check (CRC).

Data can be transmitted between two points through three different modes. First is simplex where data can be transmitted in only one direction such as TV broadcasting. Second is half duplex where data can be transmitted in both directions at a time such as walkie-talkie. The third is full duplex where data can be transmitted in both directions simultaneously such as telephone.

The communication lines, modems and other equipment used in the transmission of information between two or more stations is called data link. The orderly transfer of information in a data link is accomplished by means of a protocol.