RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, BHOPAL

New Scheme Based On AICTE Flexible Curricula

Information Technology, IV-Semester

IT402 - Computer Architecture

Course Objectives

The objective of course is to understand the basic structure and operation of computer system. Students will be able to know the operation of the arithmetic unit including the algorithms & implementation of fixed-point and floating-point addition, subtraction, multiplication & division. To study the different ways of communicating with I/O devices and standard I/O interfaces, hierarchical memory system including cache memories and virtual memory, concept of pipeline.

Unit-I Computer architecture and organization, computer generations, von Neumann model, CPU organization, CPU organization, Register organization, Various CPU register, Register Transfer, Bus and Memory Transfers, Arithmetic, Logic and Shift micro-operations, Arithmetic logic shift unit.

Unit–II The arithmetic and logic unit, Fixed-Point representation: integer representation, sign-magnitude, 1's and 2's complement and range, Integer arithmetic: negation, addition and subtraction, multiplication, division, Floating-Point representation, Floating-Point arithmetic, Hardwired micro-programmed control unit, Control memory, Micro-program sequence.

Unit–III Central Progressing Unit (CPU), Stack Organization, Memory Stack, Reverse Polish Notation. Instruction Formats, Zero, One, Two, Three- Address Instructions, RISC Instructions and CISC Characteristics, Addressing Modes, Modes of Transfer, Priority Interrupt, Daisy Chaining, DMA, Input-Output Processor (IOP).

Unit–IV Computer memory system, Memory hierarchy, main memory: RAM, ROM chip, auxiliary and associative memory, Cache memory: associative mapping, direct mapping, set-associative mapping, write policy, cache performance, Virtual memory: address space, memory space, address mapping, paging and segmentation, TLB, page fault, effective access time, replacement algorithm.

Unit–V Parallel Processing, Pipelining General Consideration, Arithmetic Pipeline, and Instruction Pipeline, Vector Operations, Matrix Multiplication, and Memory Interleaving, Multiprocessors, Characteristics of Multiprocessors.

Course Outcomes

At the end of the course student will be able to :

1. Understand basic structure of computer system, arithmetic operations,
2. Understand the arithmetic operations, Study of hardwired and micro-programmed control units.
3. Develop the concepts of memory management, interleaving and mapping.
4. Analyze the arithmetic and instructional pipelines.

Reference Books:-

1. M. Morris Mano, “Computer System Architecture”, Pearson.
2. Dr. M. Usha, T.S. Srikanth, “Computer System Architecture and Organization”, Wiley India.
3. William Stallings, ”Computer Organization and Architecture”, Pearson.
4. V. Rajaraman, T. Radhakrishnan, “Computer Organization and Architecture”, PHI.

UNIT-1

Computer architecture refers to those parameters of a computer system that are visible to a programmer or those parameters that have a direct impact on the logical execution of a program. Examples of architectural attributes include the instruction set, the number of bits used to represent different data types, I/O mechanisms, and techniques for addressing memory.

Computer Architecture refers to those attributes of a system visible to a programmer or those attributes that have a direct impact on the logical execution of a program.

Examples of architectural attributes include:

a) Instruction set designing
b) Instruction format
c) No of bits used to represent various types of data
d) Different addressing mechanism to access data

Computer organization refers to the operational units and their interconnections that realize the architectural specifications. Examples of organizational attributes include those hardware details transparent to the programmer, such as control signals, interfaces between the computer and peripherals, and the memory technology used.

Ex: Two different models from a same vendor like Intel are brought to analyze. Both the models (laptop and desktop) have same processor like core 2 duo. That means both models understand the same instruction set as we know each processor understands a fixed no of instructions. Henceforth their architecture is same. Due to the placement of various hardware components, one model (laptop) is slim and other is bulky. Hence their organization is different.

Computer Generations

First Generation (1940-1956) Vacuum Tubes

The first computers used vacuum tubes for circuitry and magnetic drums for memory, and were often enormous, taking up entire rooms. They were very expensive to operate and in addition to using a great deal of electricity, the first computers generated a lot of heat, which was often the cause of malfunctions. They relied on machine language, the lowest-level programming language understood by computers, to perform operations.They could only solve one problem at a time, and it could take days or weeks to set-up a new problem. Input was based on punched cards and paper tape, and output was displayed on printouts. The UNIVAC and ENIAC computers are examples of first-generation computing devices.

Advantages

- Vacuum tubes were the only electronic component available during those days.
- Vacuum tube technology made possible to make electronic digital computers.
- These computers could calculate data in millisecond.

Disadvantages

- The computers were very large in size.
- They consumed a large amount of energy.
- They were heated very soon due to thousands of vacuum tubes.
- They were not very reliable.
- Air conditioning was required.
- Constant maintenance was required.
- Non-portable.
- Costly commercial production.

- Very slow speed.
- Limited programming capabilities.
- Used machine language only.
- Used magnetic drums which provide very less data storage.

Second Generation (1956-1963) Transistors

The period of second generation was from 1956-1963. In this generation, transistors were used that were cheaper, consumed less power, were more compact in size, more reliable and faster than the first-generation machines made of vacuum tubes. In this generation, magnetic cores were used as the primary memory and magnetic tape and magnetic disks as secondary storage devices. In this generation, assembly language and high-level programming languages like FORTRAN, COBOL was used. The computers used batch processing and multiprogramming operating system.

Advantages

- Smaller in size as compared to the first-generation computers.
- The 2nd generations Computers were more reliable.
- Used less energy and were not heated.
- Wider commercial use.
- Better portability as compared to the first-generation computers.
- Better speed and could calculate data in microseconds.
- Used faster peripherals like tape drives, magnetic disks, printer etc.
- Used Assembly language instead of Machine language.
- Accuracy was improved.

Disadvantages

- Cooling system was required.
- Constant maintenance was required.
- Commercial production was difficult.
- Only used for specific purposes.
- Costly and not versatile.
- Punch cards were used for input.

Third Generation (1964-1971) Integrated Circuits

The period of third generation was from 1965-1971. The computers of third generation used Integrated Circuits (ICs) in place of transistors. A single IC has many transistors, resistors, and capacitors along with the associated circuitry. The IC was invented by Jack Kilby. This development made computers smaller in size, reliable, and efficient. In this generation remote processing, time-sharing, multiprogramming operating system were used. High-level languages (FORTRAN-II TO IV, COBOL, PASCAL PL/1, BASIC, ALGOL-68 etc.) were used during this generation.

Advantages

- Smaller in size as compared to previous generations.
- Used less energy.
- Produced less heat as compared to the previous two generations of computers.
- Better speed and could calculate data in nanoseconds.
- Used fan for heat discharge to prevent damage.
- Totally general purpose.
- Could be used for high-level languages.
- Good storage.
- Less expensive.

- Better accuracy.
- Commercial production increased.
- Used mouse and keyboard for input.

Disadvantages

- Air conditioning was required.
- Highly sophisticated technology required for the manufacturing of IC chips.

Fourth Generation (1971-Present) Microprocessors

Fourth generation computers became more powerful, compact, reliable, and affordable which gave rise to Personal Computer (PC) revolution. In this generation, time sharing, real time networks, distributed operating system were used. All the high-level languages like C, C++, DBASE etc., were used in this generation.

Advantages

- More powerful and reliable than previous generations.
- Small in size.
- Fast processing power with less power consumption.
- Fan for heat discharging and thus to keep cold.
- No air conditioning required.
- Totally general purpose.
- Commercial production.
- Cheapest among all generations.
- All types of High level languages can be used in this type of computers.

Disadvantages

- The latest technology is required for manufacturing of Microprocessors.

Fifth Generation (Present and Beyond) Artificial Intelligence

The period of fifth generation is 1980-till date. In the fifth generation, VLSI technology became ULSI (Ultra Large-Scale Integration) technology, resulting in the production of microprocessor chips having ten million electronic components.

This generation is based on parallel processing hardware and AI (Artificial Intelligence) software. AI is an emerging branch, which interprets the means and method of making computers think like human beings. All the high-level languages like C and C++, Java, .Net etc. are used in this generation.

AI includes –

- Robotics
- Neural Networks
- Game Playing
- Development of expert systems to make decisions in real-life situations
- *Natural language understanding and generation*

Von Neumann Model

It was developed in 1945 by John Von Neumann. Von Neumann model consist of a CPU, memory and I/O devices. The program is stored in the memory. The CPU fetches an instruction from the memory at a time and executes it.

The Von Neumann architecture is a design model for a stored-program digital computer that uses a processing unit and a single separate storage structure to hold both instructions and data.

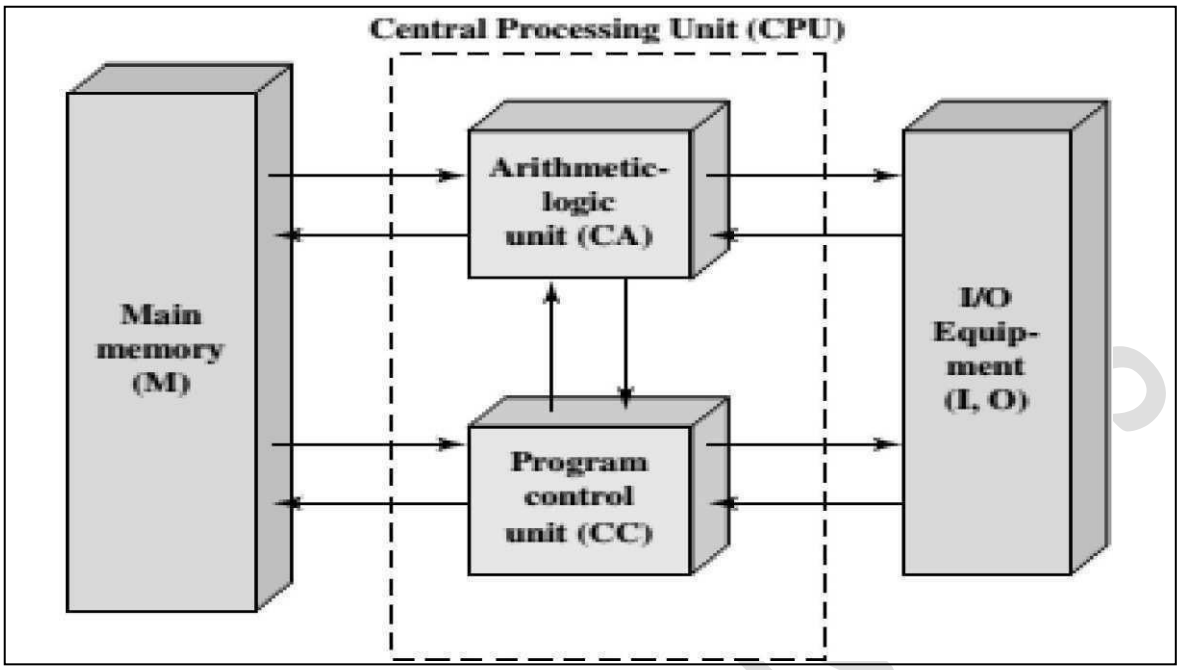


Fig 1.1 Von Neumann Model

A. Central Processor Unit [CPU]:

Central processor unit consists of two basic blocks:

The program control unit has a set of registers and control circuit to generate control signals. The execution unit or data processing unit contains a set of registers for storing data and an Arithmetic and Logic Unit (ALU) for execution of arithmetic and logical operations. In addition, CPU may have some additional registers for temporary storage of data.

B. Input Unit:

With the help of input unit data from outside can be supplied to the computer. Program or data is read into main storage from input device or secondary storage under the control of CPU input instruction. Example of input devices: Keyboard, Mouse, Hard disk, Floppy disk, CD-ROM drive etc.

C. Output Unit:

With the help of output unit computer results can be provided to the user or it can be stored in storage device permanently for future use. Output data from main storage go to output device under the control of CPU output instructions.

Example of output devices: Printer, Monitor, Plotter, Hard Disk, Floppy Disk etc.

D. Memory Unit:

Memory unit is used to store the data and program. CPU can work with the information stored in memory unit. This memory unit is termed as primary memory or main memory module. These are basically semiconductor memories.

There are two types of semiconductor memories -

- Volatile Memory: RAM (Random Access Memory).
- Non-Volatile Memory: ROM (Read only Memory), PROM (Programmable ROM) EPROM (Erasable PROM), EEPROM (Electrically Erasable PROM).

Secondary Memory:

There is another kind of storage device, apart from primary or main memory, which is known as secondary memory. Secondary memories are non-volatile memory and it is used for permanent storage of data and program.

Example of secondary memories:

Hard Disk, Floppy Disk, Magnetic Tape

CPU organization

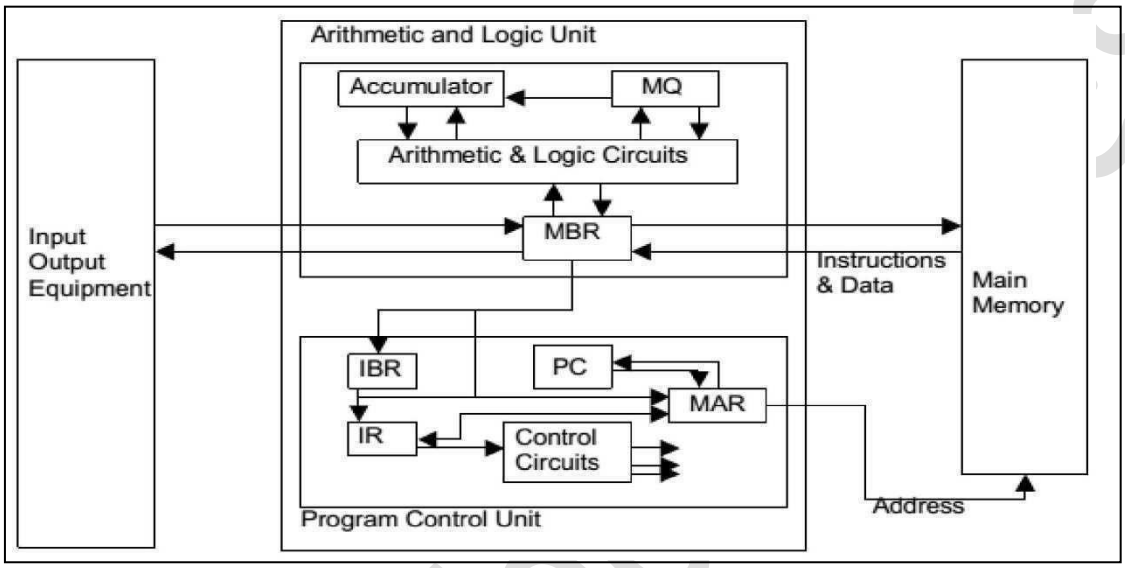


Fig 1.2 CPU Block Diagram

Depending on the internal organization computers can be categorized into one of the three CPU organization

1. Single Accumulator Organization
2. Stack Organization
3. General Register Organization

1. The single accumulator organizations are performed with the implied accumulator register. The instruction format in this type of organization uses one address field.

Example:

ADD X --> AC

2. The stack organized computer can use 1 or 0 address instruction and uses on y two instructions POP and PUSH

Example:

PUSH X --> TOS ss-M[X]

POP --> TOS

3. When there are more than one register, general register organization can be used and the instruction format may contain 2 or 3 address field.

Example:

ADD R1, R2 --> R1

ADD R1, R2, R3 --> R1

Register organization

The CPU is made up of three major parts: Register Set, ALU, and Control Unit as shown in figure below. The register set stores intermediate data used during the execution of the instructions. The arithmetic logic unit (ALU) performs the required microoperations for executing the instructions. The control unit supervises the transfer of information among the registers and instructs the ALU as to which operation to perform.

A bus organization for 7 CPU register is shown in a figure below. All registers are connected to two multiplexers (MUX) that select the registers for bus A and bus B. Registers selected by multiplexers are sent to ALU. Another selector (OPR) connected to ALU selects the operation for the ALU. Output produced by ALU is stored in some register and this destination register for storing the result is activated by the destination decoder (SELD).

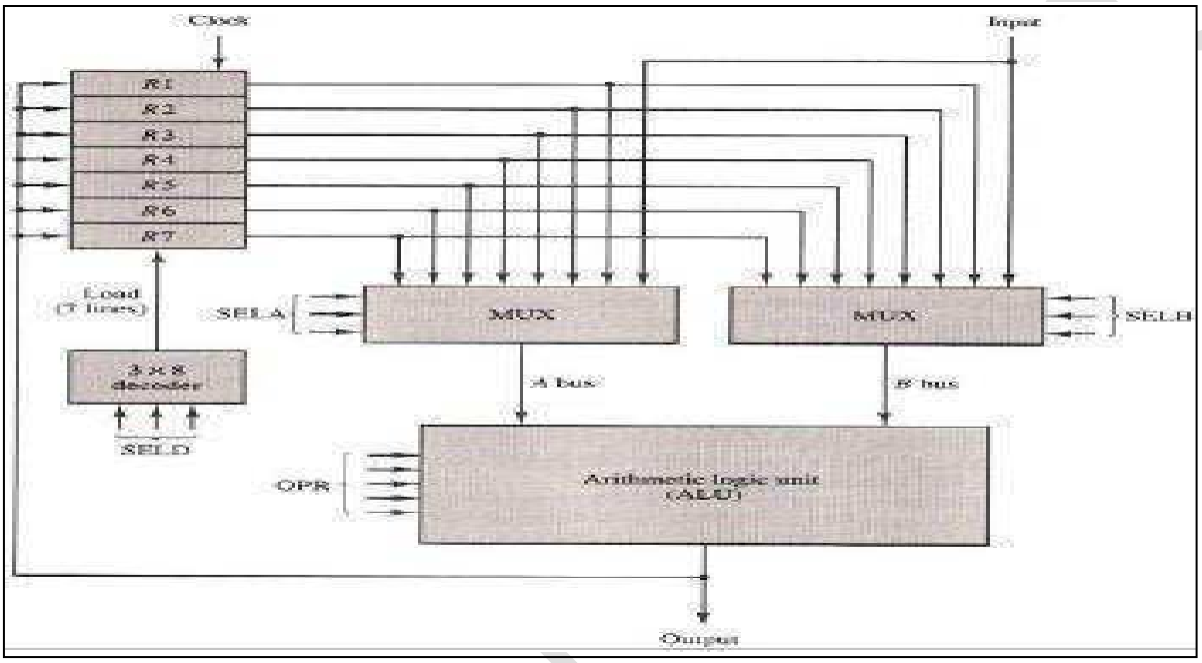


Fig 1.3 General register organization

Example: R1

– MUX selector (SELA): BUS A

– MUX selector (SELB): BUS B

– ALU operation selector (OPR): ALU to ADD

– Decoder destination selector (SELD): R1

Various CPU Registers

1. Memory Address Register (MAR):

This register holds the address of memory where CPU wants to read or write data. When CPU wants to store some data in the memory or reads the data from the memory, it places the address of the required memory location in the MAR.

2. Memory Buffer Register (MBR):

This register holds the contents of data or instruction read from, or written in memory. The contents of instruction placed in this register are transferred to the Instruction Register, while the contents of data are transferred to the accumulator or I/O register.

3. I/O address Register (I/O AR):

I/O Address register is used to specify the address of a particular I/O device.

4. I/O Buffer Register (I/O I3R):
I/O Buffer Register is used for exchanging data between the I/O module and the processor.

5. Program Counter (PC)
Program Counter register is also known as Instruction Pointer Register. This register is used to store the address of the next instruction to be fetched for execution. When the instruction is fetched, the value of IP is incremented. Thus, this register always points or holds the address of next instruction to be fetched.

6. Instruction Register (IR):
Once an instruction is fetched from main memory, it is stored in the Instruction Register. The control unit takes instruction from this register, decodes and executes it by sending signals to the appropriate component of computer to carry out the task.

7. Accumulator Register:
The accumulator register is located inside the ALU, it is used during arithmetic & logical operations of ALU. The control unit stores data values fetched from main memory in the accumulator for arithmetic or logical operation.

8. Stack Control Register:
A stack represents a set of memory blocks; the data is stored in and retrieved from these blocks in an order, i.e. First In and Last Out (FILO). The Stack Control Register is used to manage the stacks in memory. The size of this register is 2 or 4 bytes.

9. Flag Register:
The Flag register is used to indicate occurrence of a certain condition during an operation of the CPU. It is a special purpose register with size one byte or two bytes. Each bit of the flag register constitutes a flag (or alarm), such that the bit value indicates if a specified condition was encountered while executing an instruction.

Register Transfer:
Information transferred from one register to another is designated in symbolic form by means of replacement operator.
R2 ← R1 (It denotes the transfer of the data from register R1 into R2)

- Normally we want the transfer to occur only in predetermined control condition. This can be shown by following if-then statement: if (P=1) then (R2 ← R1)
- Here P is a control signal generated in the control section.

Control Function
A control function is a Boolean variable that is equal to 1 or 0. The control function is shown as:
P: R2 ← R1
The control condition is terminated with a colon. It shows that transfer operation can be executed only if P=1.

Basic symbols for register transfer language

Symbol	Description	Examples
Uppercase letters	Denotes a register	A, R1, MDR
Subscript	An individual cell	A5,B9
Parenthesis	A portion of register	PC(H), MDR(ADR)

Arrow	A transfer	R1 ss R2
Colon	Terminates a control function	T1:
Comma	Multiple operation	T:R1ss R2, R3 ss R4
Square Brackets	Address for memory	MDR ss M[MAR]

Bus & Memory Transfers:

- Memory Read: The read operation for the transfer of a memory unit M from an address register MAR to another data register DR can be illustrated as:
    - Read: DR ←M[MAR]
- Memory Write : The write operation transfer the contents of a data register to a memory word M selected by the address. Assume that the input data are in register R1 and the address in the MAR. The write operation can be stated symbolic as follows:
    - Write: M[MAR] ← R1
- This cause a transfer on information from R1 into the memory word M selected by the address in AR

BUS transfer

Using Multiplexers

Rather than connecting wires between all registers, a common bus is used A bus structure consists of a set of common lines, one for each bit of a register Control signals determine which register is selected by the bus during each transfer Multiplexers can be used to construct a common bus Multiplexers select the source register whose binary information is then placed on the bus the select lines are connected to the selection inputs of the multiplexers and choose the bits of one register

In general, a bus system will multiplex k registers of n bits each to produce an n-line common bus

- This requires n multiplexers – one for each bit
- The size of each multiplexer must be k x 1
- The number of select lines required is log k
- To transfer information from the bus to a register, the bus lines are connected to the inputs of all destination registers and the corresponding load control line must be activated rather than listing each step as

BUS ← C, R1 ← BUS,
R1 ← C, since the bus is implied

Three-state gates

Instead of using multiplexers, three-state gates can be used to construct the bus system
A three-state gate is a digital circuit that exhibits three states

- Two of the states are signals equivalent to logic 1 and 0
- The third state is a high-impedance state – this behaves like an open circuit, which means the output is disconnected and does not have a logic significance

The three-state buffer gate has a normal input and a control input which determines the output state With control 1, the output equals the normal input With control 0, the gate goes to a high-impedance state This enables a large number of three-state gate outputs to be connected with wires to form a common bus line

without endangering loading effects Decoders are used to ensure that no more than one control input is active at any given time This circuit can replace the multiplexer in figure 1.4.
To construct a common bus for four registers of n bits each using three-state buffers, we need n circuits with four buffers in each Only one decoder is necessary to select between the four registers.

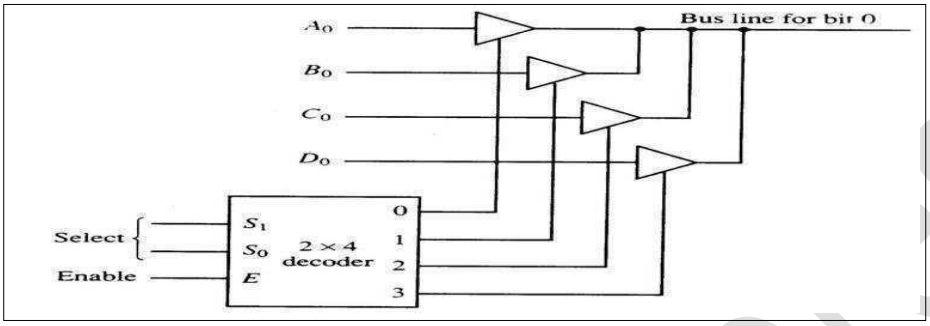


Fig 1.4 Three state bus buffer

Memory Transfer

We designate a memory word by the letter M. It is necessary to specify the address of M when writing memory transfer Operations Designate the address register by AR and the data register by DR.
The read operation can be stated as:

Read: DR ← M [AR]

The write operation can be stated as:

Write: M [AR] ← R1

The address register (AR) is used to select a memory address, and the data register (DR) is used to send and receive data. Both these registers are connected to the internal bus. DR is a bridge between the internal BUS and the memory data BUS.
Memory can also be connected directly to the internal BUS in theory.

M[AR] ← DR

DR ← M[AR]

Hence, accessing memory outside the CPU requires at least two clock cycles. First, we load AR with the desired memory address, and then transfer to or from DR. In most typical computer systems, memory transfers take many clock cycles, known as wait states.

Arithmetic, Logic and Shift micro-operations
Micro-operations perform basic operations on data stored in one or moreregisters, including transferring data between registers or between registers and external buses of the central processing unit (CPU), and performing arithmetic or logical operations on registers.

Types of Micro-operations

- **Register transfer micro-operations**: These types of micro operations are used to transfer from one register to binary information.
- **Arithmetic micro-operations**: These micro-operations are used to perform on numeric data stored in the registers some arithmetic operations.
- **Logic micro-operations:** These micro operations are used to perform bit style operations / manipulations on non-numeric data.

- **Shift micro operations:** As their name suggests they are used to perform shift operations in data store in registers.

Arithmetic Micro-Operations: Some of the basic micro-operations are addition, subtraction, increment and decrement.

Add Micro-Operation
It is defined by the following statement:
R3 → R1 + R2
The above statement instructs the data or contents of register R1 to be added to data or content of register R2 and the sum should be transferred to register R3.

Subtract Micro-Operation
Let us again take an example:
R3 → R1 + R2' + 1
In subtract micro-operation, instead of using minus operator we take 1's compliment and add 1 to the register which gets subtracted, i.e R1 - R2 is equivalent to R3 → R1 + R2' + 1

Increment/Decrement Micro-Operation
Increment and decrement micro-operations are generally performed by adding and subtracting 1 to and from the register respectively.
R1 → R1 + 1
R1 → R1 – 1

Symbolic Designation	Description
R3 ← R1 + R2	Contents of R1+R2 transferred to R3.
R3 ← R1 - R2	Contents of R1-R2 transferred to R3.
R2 ← (R2)'	Compliment the contents of R2.
R2 ← (R2)' + 1	2's compliment the contents of R2.
R3 ← R1 + (R2)' + 1	R1 + the 2's compliment of R2 (subtraction).
R1 ← R1 + 1	Increment the contents of R1 by 1.
R1 ← R1 - 1	Decrement the contents of R1 by 1.

Logic Micro-Operations
These are binary micro-operations performed on the bits stored in the registers. These operations consider each bit separately and treat them as binary variables.

Let us consider the X-OR micro-operation with the contents of two registers R1 and R2.
P: R1 ← R1 X-OR R2
In the above statement we have also included a Control Function.
Assume that each register has 3 bits. Let the content of R1 be 010 and R2 be 100. The X-OR micro-operation will be:

$$010 \rightarrow R1$$

$$\underline{100 \rightarrow R2}$$

$$\underline{110} \rightarrow R1 \text{ after } P=1$$

Shift Micro-Operations

These are used for serial transfer of data. That means we can shift the contents of the register to the left or right. In the shift left operation the serial input transfers a bit to the right most position and in shift right operation the serial input transfers a bit to the left most position.

There are three types of shifts as follows:

a) Logical Shift

It transfers 0 through the serial input. The symbol "shl" is used for logical shift left and "shr" is used for logical shift right.

R1 ← she R1

R1 ← she R1

The register symbol must be same on both sides of arrows.

b) Circular Shift

This circulates or rotates the bits of register around the two ends without any loss of data or contents. In this, the serial output of the shift register is connected to its serial input. "cil" and "cir" is used for circular shift left and right respectively.

c) Arithmetic Shift

This shifts a signed binary number to left or right. An arithmetic shift left multiplies a signed binary number by 2 and shift left divides the number by 2. Arithmetic shift micro-operation leaves the sign bit unchanged because the signed number remains same when it is multiplied or divided by 2.

UNIT-II

The arithmetic and logic unit

An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer. Modern CPUs contain very powerful and complex ALUs.

Fixed-Point representation

Fixed Point Representation

The positive numbers are represented as unsigned numbers but for negative values. For example, the arithmetic uses plus '+' or minus '-'sign to indicate positive or negative numbers. But in binary notation, o is used to indicate positive and 1 is used to indicate negative numbers. In addition to the sign, a number may have a binary or decimal point to represent fractions, integers or mixed integers.

Integer representation

When the number is positive, a '0' is used to represent the positive number or when the number is negative, the sign is represented by 1. And, the rest of the number is represented as by any one of the following method.

- Signed Magnitude representation
- Signed 1's complement representation
- Signed 2's complement representation

Consider, an 8-bit representation of +14

- Signed magnitude representation

+14 ss 00001110
-14 ss 10001110

- Signed 1's complement representation

+14 ss 01110001
-14 ss 11110001

- Signed 2's complement representation

+14 ss 01110010
-14 ss 11110010

Addition and Subtraction

Four basic computer arithmetic operations are addition, subtraction, division and multiplication.

Addition and Subtraction with signed magnitude

Consider two numbers having magnitude A and B. When the signed numbers are added or subtracted, there can be 8 different conditions depending on the sign and the operation performed as shown in the table below:

Operation	Add magnitude	When A > B	When A < B	When A ss 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)

From the table, we can derive an algorithm for addition and subtraction as follows:

Addition (Subtraction) Algorithm:

- When the signs of A & B are identical, add the two magnitudes and attach the sign of A to the result.
- When the sign of A & B are different, compare the magnitude and subtract the smaller number from the large number. Choose the sign of the result to be same as A if A > B, or the complement of the sign of A if A < B. If the two numbers are equal, subtract B from A and make the sign of the result positive.

Hardware Implementation

The hardware consists of two registers A and B to store the magnitudes, and two flip-flops As and Bs to store the corresponding signs. The results can be stored in the register A and As which acts as an accumulator. The subtraction is performed by adding A to the 2's complement of B. The output carry is transferred to the flip-flop E. The overflow may occur during the add operation which is stored in the flip-flop A Ë… F. When m ss 0, the output of E is transferred to the adder without any change along with the input carry of '0". The output of the parallel adder is equal to A + B which is an add operation. When m ss 1, the content of register B is complemented and transferred to parallel adder along with the input carry of 1. Therefore, the output of parallel is equal to A + B' + 1 ss A – B which is a subtract operation.

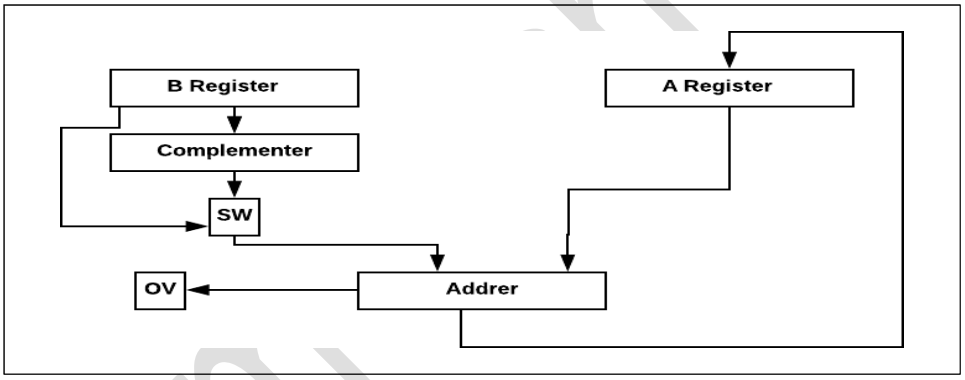


Fig 2.1 Hardware Implementation of addition & subtraction

As and Bs are compared by an exclusive-OR gate. If outputss0, signs are identical, if 1 signs are different.

- For Add operation, identical signs dictate addition of magnitudes and for operation identical signs dictate addition of magnitudes and for *subtraction*, different magnitudes dictate magnitudes be added. Magnitudes are added with a microoperation EA
- Two magnitudes are subtracted if signs are different for add operation and identical for subtract operation. Magnitudes are subtracted with a microoperation EA ss B and number (this number is checked again for 0 to make positive 0 [Asss0]) in A is correct result. E ss 0 indicates A < B, so we take 2's complement of A.

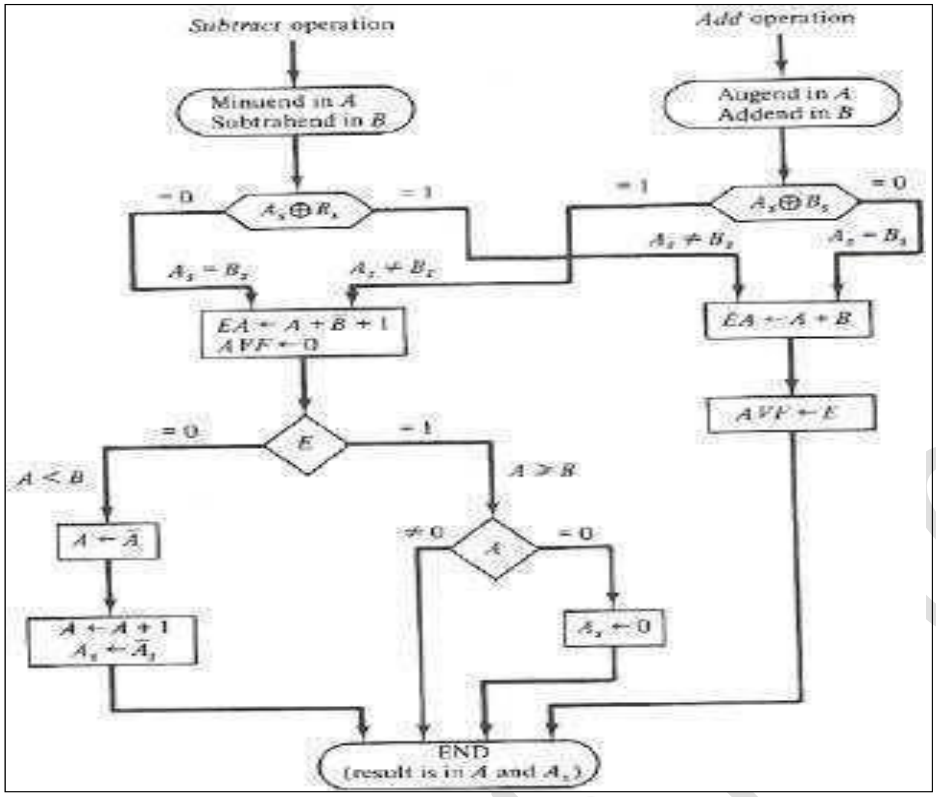


Fig 2.2 Flowchart of addition & subtraction

Multiplication Hardware Implementation and Algorithm

Generally, the multiplication of two final point binary number in signed magnitude representation is performed by a process of successive shift and ADD operation. The process consists of looking at the successive bits of the multiplier (least significant bit first). If the multiplier is 1, then the multiplicand is copied down otherwise, 0's are copied. The numbers copied down in successive lines are shifted one position to the left and finally, all the numbers are added to get the product.
The hardware for the multiplication of signed magnitude data is shown in the figure below.

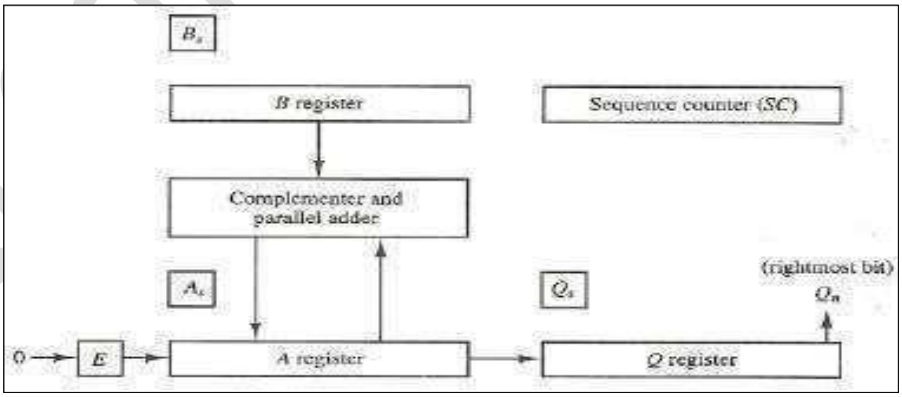


Fig 2.3 Multiplication Hardware Implementation

Initially, the multiplier is stored q register and the multiplicand in the B register. A register is used to store the partial product and the sequence counter (SC) is set to a number equal to the number of bits in the multiplier. The sum of A and B form the partial product and both shifted to the right using a statement "Shr EAQ" as shown in the hardware algorithm. The flip flops As, Bs & Qs store the sign of A, B & Q respectively. A binary '0" inserted into the flip-flop E during the shift right.

Hardware Algorithm

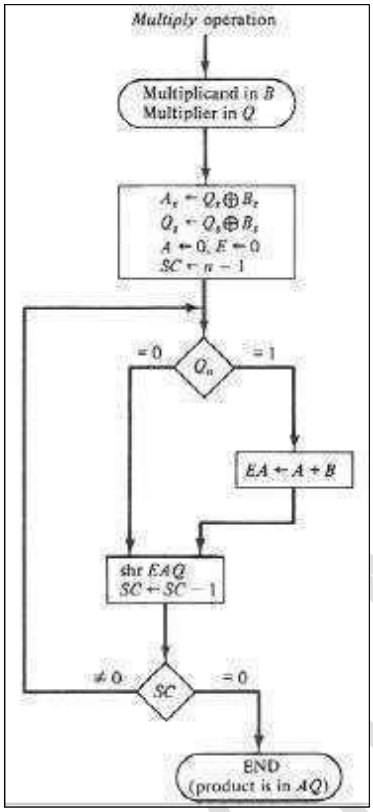


Fig 2.4 Multiplication Flowchart

Example: Multiply 23 by 19 using multiply algorithm.

multiplicand	E	A	Q	SC
Initially,	0	00000	10011	101(5)
Iteration1(Qnss1), add B first partial product shrEAQ,	0	00000 +10111 10111		
	0	01011	11001	100(4)
Iteration2(Qnss1) Add B Second partial product shrEAQ,	1	01011 +10111 00010	11001	
	0	10001	01100	011(3)
Iteration3(Qnss0) shrEAQ,	0	01000	10110	010(2)
Iteration4(Qnss0) shrEAQ,	0	00100	01011	001(1)
Iteration5(Qnss1 Add B Fifth partial product shrEAQ,	0	00100 +10111 11011	01011	
	0	01101	10101	000
Final Production AQ	0110110101			

The final product is in register A & Q. therefore, the product is 0110110101.

Booth Algorithm

The algorithm that is used to multiply binary integers in signed 2's complement form is called booth multiplication algorithm. It works on the principle that the string 0's in the multiplier doesn't need addition but just the shifting and the sting of 1's from bit weight 2^k to 2^m can be treated as $2^{k+1} - 2^m$ (Example, +14 ss 001110 ss 2^3 ss 1 2^1 ss 14). The product can be obtained by shifting the binary multiplication to the left and subtraction the multiplier shifted left once.

According to booth algorithm, the rule for multiplication of binary integers in signed 2's complement form are:

- The multiplicand is subtracted from the partial product of the first least significant bit is 1 in a string of 1's in the multiplicand.
- The multiplicand is added to the partial product if the first least significant bit is 0 (provided that there was a previous 1) in a string of 0's in the multiplier.
- The partial product doesn't change when the multiplier bit is identical to the previous multiplier bit.

This algorithm is used for both the positive and negative numbers in signed 2's complement form. The hardware implementation of this algorithm is in figure below:

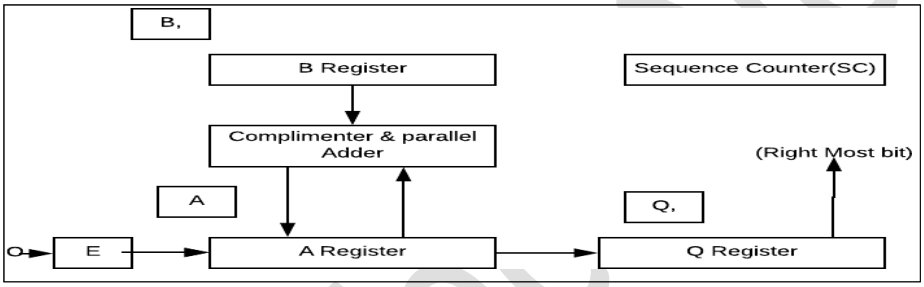


Fig 2.5 Booth hardware implementation

Numerical Example: Booth algorithm

BRss10111(**Multiplicand**)

QRss10011(**Multiplier**)

Array Multiplier

The multiplication algorithm first checks the bits of the multiplier one at time and form partial product. This is a sequential process that requires a sequence of add and shift microoperation. This method is complicated and time consuming. The multiplication of 2 binary numbers can also be done with one microoperation by using combinational circuit that provides the product all at once.

Example.

Consider that the multiplicand bits are b1 and b0 and the multiplier bits are a1 and a0. The partial product is c3c2c1c0. The multiplication two bits a0 and a1 produces a binary 1 if both the bits are 1, otherwise it produces a binary 0. This is identical to the AND operation and can be implemented with the AND gates as shown in figure.

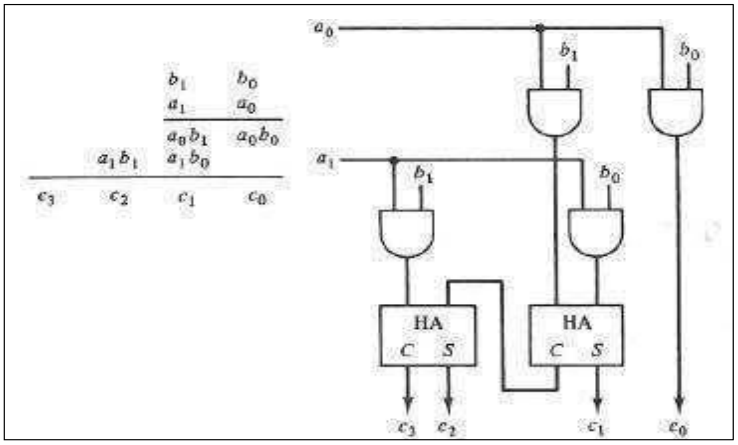


Fig 2.6 2-bit by 2-bit array multiplier

Division Algorithm

The division of two fixed point signed numbers can be done by a process of successive compare shift and subtraction. When it is implemented in digital computers, instead of shifting the divisor to the right, the dividend or the partial remainder is shifted to the left. The subtraction can be obtained by adding the number A to the 2's complement of number B. The information about the relative magnitudes of the information about the relative magnitudes of numbers can be obtained from the end carry,

Hardware Implementation

The hardware implementation for the division signed numbers is shown id the figure.

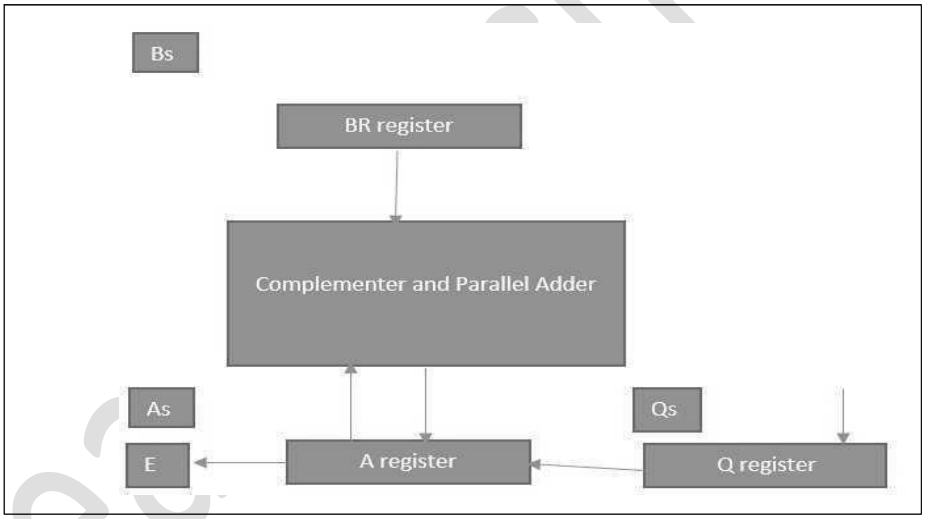


Fig 2.7 Division Algorithm

The divisor is stored in register B and a double length dividend is stored in register A and Q. the dividend is shifted to the left and the divider is subtracted by adding twice complement of the value. If E ss 1, then A >ss B. In this case, a quotient bit 1 is inserted into Qn and the partial remainder is shifted to the left to repeat the process. If E ss 0, then A > B. In this case, the quotient bit Qn remains zero and the value of B is added to restore the partial remainder in A to the previous value. The partial remainder is shifted to the left and approaches continues until the sequence counter reaches to 0. The registers E, A & Q are shifted to the left with 0 inserted into Qn and the previous value of E is lost as shown in the flow chart for division algorithm.

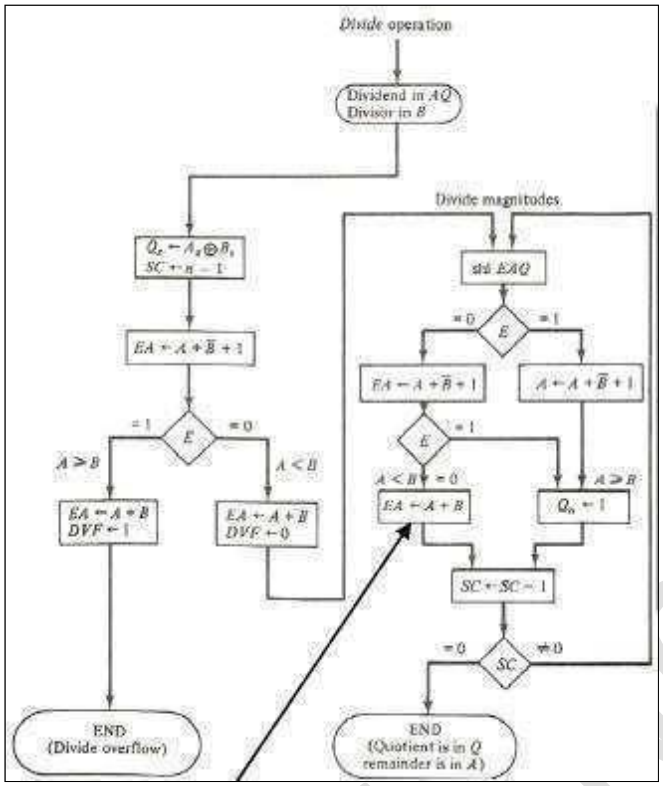


Fig 2.8 Flowchart division algorithm

This algorithm can be explained with the help of an example.
Consider that the divisor is 10001 and the dividend is 01110.

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

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

Fig 2.9 Division Algorithm

Restoring method

Method described above is restoring **method** in which partial remainder is restored by adding the divisor to the negative result. Other methods:

Non-restoring method: In contrast to restoring method, when A -B is negative, B is not added to restore A but instead, negative difference is shifted left and then B is added. How is it possible? Let's argue:

Divide Overflow

The division algorithm may produce a quotient overflow called dividend overflow. The overflow can occur of the number of bits in the quotient are more than the storage capacity of the register. The overflow flip-flop DVF is set to 1 if the overflow occurs.

The division overflow can occur if the value of the half most significant bits of the dividend is equal to or greater than the value of the divisor. Similarly, the overflow can occuessr if the dividend is divided by a 0. The overflow may cause an error in the result or sometimes it may stop the operation. When the overflow stops the operation of the system, then it is called divide stop.

Floating-Point Representation

The floating-point representation of a number has two parts: *mantissa* and *exponent*

Mantissa: represents a signed, fixed-point number. Maybe a fraction or an integer

Exponent: designates the position of the decimal (or binary) point

Example1: decimal number +6132.789 is represented in floating-point as:

Fraction exponent

+0.6132789 +04

Floating-Point arithmetic

e

Floating-point is interpreted to represent a number in the form: m * r . Only the mantissa m and exponent *e* are physically represented in resisters. The radix *r* and the radix point position are always .

Example2: binary number +1001.11 is represented with an 8-bit fraction and 6-bit exponent as,

Fraction exponent

+01001110 000100

or equivalently,

e +4

2 m * ss +(.1001110)2 * 2

Normalization

A floating-point number is said to be *normalized* if the most significant digit of the mantissa is nonzero. For, example, decimal number 350 is normalized but 00035 is not.

Hardwired micro-programmed control unit

To execute an instruction, the control unit of the CPU must generate the required control signal in the proper sequence. There are two approaches used for generating the control signals in proper sequence as Hardwired Control unit and Micro-programmed control unit.

Hardwired Control Unit–

The control hardware can be viewed as a state machine that changes from one state to another in every clock cycle, depending on the contents of the instruction register, the condition codes and the external inputs. The outputs of the state machine are the control signals. The sequence of the operation carried out by this machine is determined by the wiring of the logic elements and hence named as "hardwired".

- Fixed logic circuits that correspond directly to the Boolean expressions are used to generate the control signals.
- Hardwired control is faster than micro-programmed control.
- A controller that uses this approach can operate at high speed.

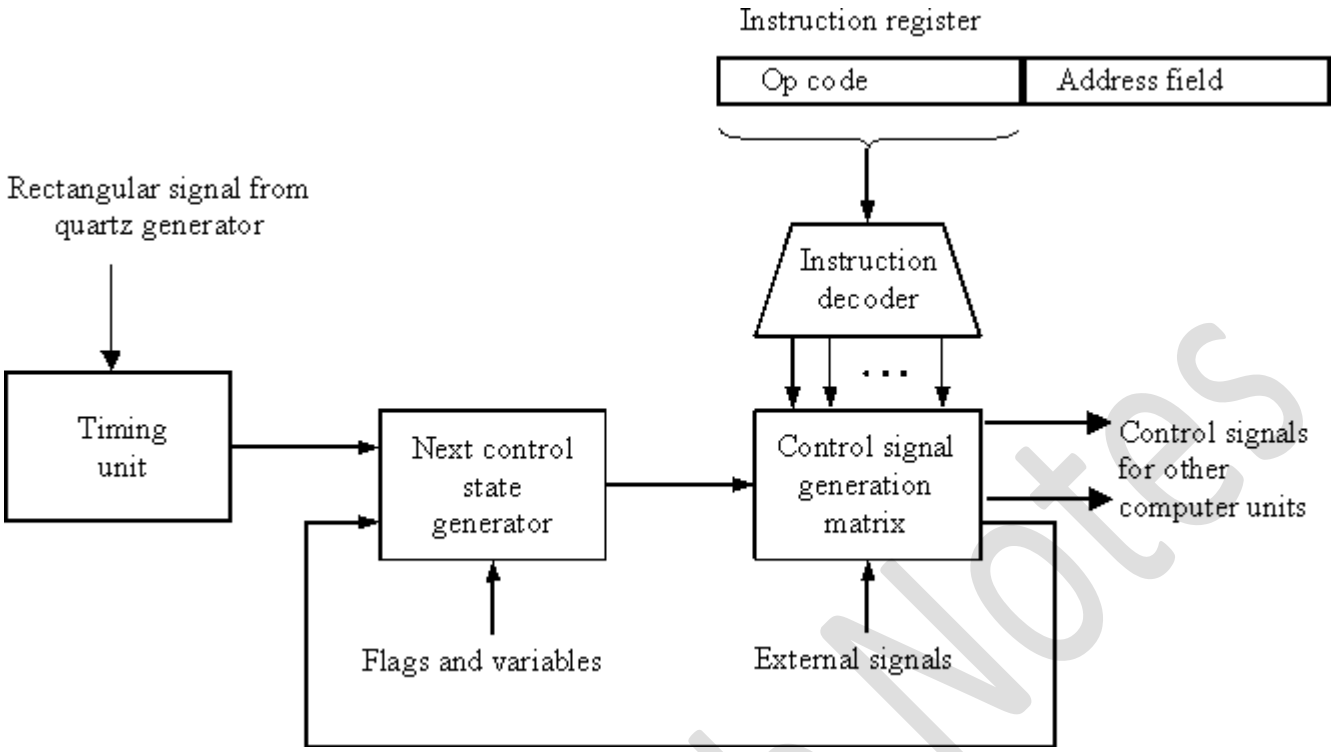


Fig2.10 Hardwired Control Unit

Micro-programmed Control Unit –

- The control signals associated with operations are stored in special memory units inaccessible by the programmer as Control Words.
- Control signals are generated by a program are similar to machine language programs.
- Micro-programmed control unit is slower in speed because of the time it takes to fetch microinstructions from the control memory.

Terminology:

- **Control Word :** A control word is a word whose individual bits represent various
- control signals.
- **Micro-routine :** A sequence of control words corresponding to the control sequence of a machine instruction constitutes the micro-routine for that instruction.
- **Micro-instruction :** Individual control words in this micro-routine are referred to as microinstructions.
- **Micro-program :** A sequence of micro-instructions is called a micro-program, which is stored in a ROM or RAM called a Control Memory (CM).
- **Control Store :** the micro-routines for all instructions in the instruction set of a computer are stored in a special memory called the Control Store.

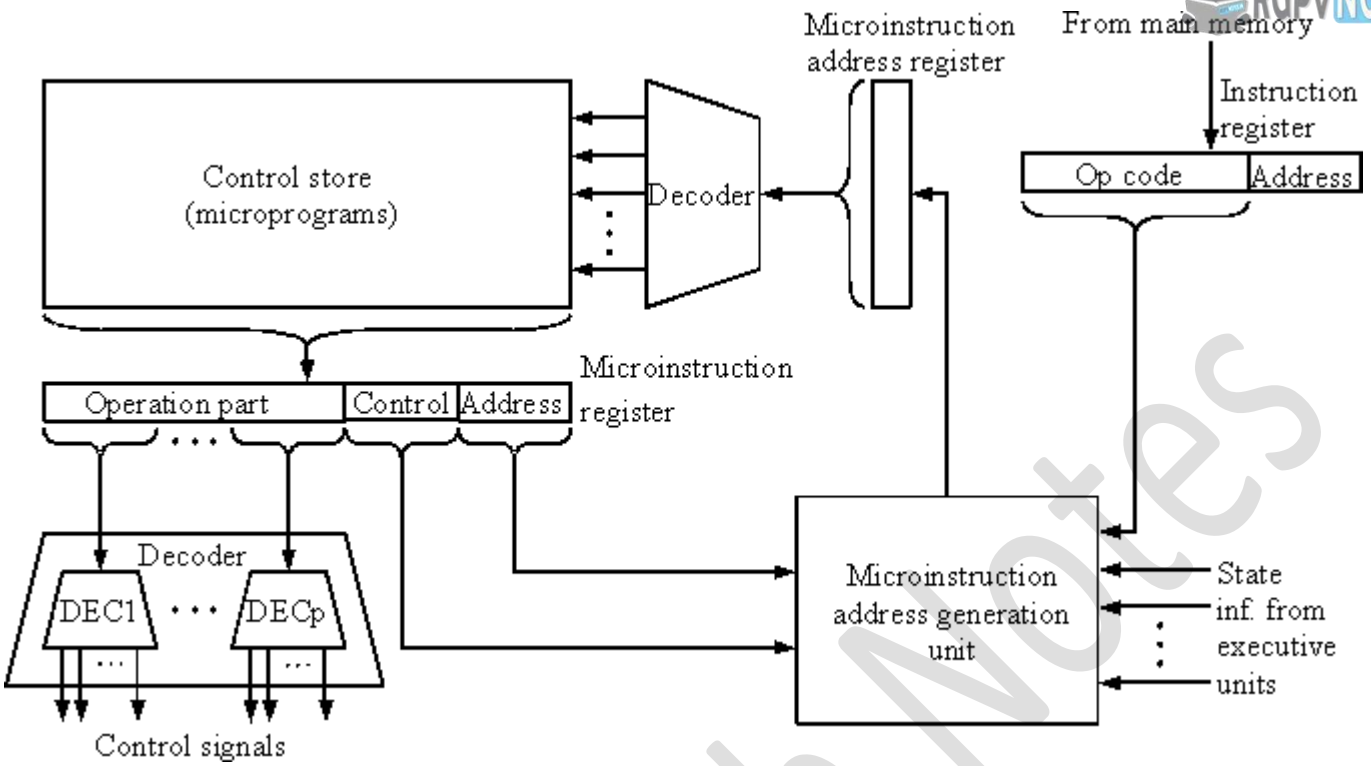


Fig 2.11 Microprogramed Control Unit

Control Memory

- The control unit in a digital computer initiates sequences of micro operations
- The complexity of the digital system is derived form the number of sequences that are performed
- When the control signals are generated by hardware, it is hardwired
- In a bus-oriented system, the control signals that specify micro operations are groups of
- bits that select the paths in multiplexers, decoders, and ALUs
- The control unit initiates a series of sequential steps of micro operations
- The control variables can be represented by a string of 1's and 0's called a control word
- A micro programmed control unit is a control unit whose binary control variables are stored in memory
- The control unit initiates a series of sequential steps of micro operations
- The control variables can be represented by a string of 1's and 0's called a control word
- A micro programmed control unit is a control unit whose binary control variables are stored in memory
- A sequence of microinstructions constitutes a micro program
- The control memory can be a read-only memory
- Dynamic microprogramming permits a micro program to be loaded and uses a writable control memory
- A computer with a micro programmed control unit will have two separate memories: a main memory and a control memory
- The micro program consists of microinstructions that specify various internal control signals for execution of register micro operations
- These microinstructions generate the micro operations to:

1. fetch the instruction from main memory
2. evaluate the effective address
3. execute the operation

- Return control to the fetch phase for the next instruction.
- The control memory address register specifies the address of the microinstruction
- The control data register holds the microinstruction read from memory

- The microinstruction contains a control word that specifies one or more
- Micro operations for the data processor

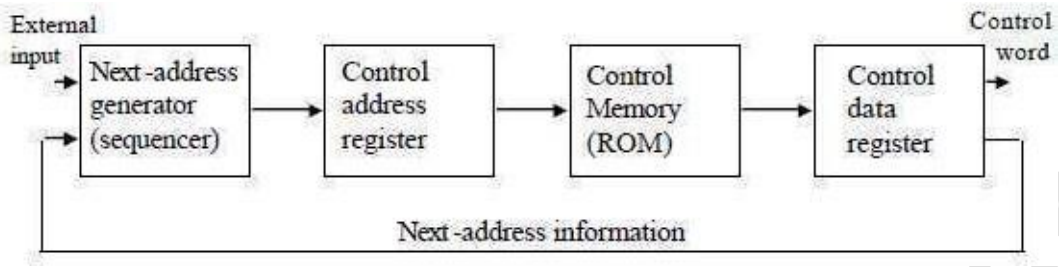


Fig 2.12 Control Memory

Micro Program sequencer

In computer architecture and engineering, a sequencer or micro sequencer generates the addresses used to step through the micro program of a control store. It is used as a part of the control unit of a CPU or as a stand-alone generator for address ranges. Usually the addresses are generated by some combination of a counter, a field from a microinstruction, and some subset of the instruction register. A counter is used for the typical case, that the next microinstruction is the one to execute. A field from the microinstruction is used for jumps, or other logic. Since CPUs implement an instruction set, it's very useful to be able to decode the instruction's bits directly into the sequencer, to select a set of microinstructions to perform a CPU's instructions. Most modern CPUs are considerably more complex than this description suggests. They tend to have multiple cooperating micro machines with specialized logic to detect and handle interference between the micro machines.

A micro program sequencer for micro programmed control unit develops micro program consecutive addresses, branches to subroutines with address saving and possible return to micro program, as well as interrupting micro program forcings with address saving of the interrupted micro programs.

In order to allow the double saving of micro program and subroutine addresses in case of concurrent interruptions and branches, the sequencer is provided with two address generation loops each including a register. The two loops have a common portion to which they accede through a multiplexer (23). The first loop (23, 25, 22, 21, 30, 31) is further coupled to a saving register stack (20). While the first loop executes the saving of a micro program address and the latching of a branch address received from the second loop, the second loop (23, 25, 24, 39, 17, 18, 42, 19, 27, 29) executes a first updating and-, related latching of interrupting micro program address. During the following cycle, by command of the first microinstruction of the interrupting micro program, the second loop performs a first updating and related latch of the interrupting micro program address and the first loop saves into the register stack (20) the branch address and performs a second updating and related latching of the interrupting micro program address.

UNIT-III Central Progressing Unit (CPU)

STACK ORGANIZATION

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

Register Stack

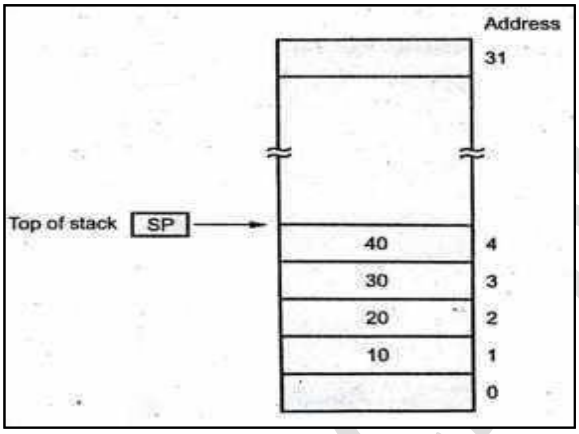


Figure 3.1 word register stack

A stack can be placed in a portion of a memory unit or it can be organized as a collection of a finite number of CPU registers. The Figure 3.1 shows the organization of a 32-word register stack. The stack pointer holds the address of the register that is currently the top of stack. As shown in the Figure 3.1, four data elements 10, 20, 30 and 40 are placed in the stack. The data element 40 is on the top of stack therefore, the content of SP is now 4.

Memory Stack

A portion of memory can be used as a stack with a processor register as a SP. Figure 3.2 shows a portion of memory partitioned into 3 parts: program, data and stack.

PC: used during fetch phase to read an instruction.

AR:used during execute phase to read an operand.

SP:used to push or pop items into or from the stack.

Here, initial value of SP is 4001 and stack grows with *decreasing addresses*. First item is stored at 4000, second at 3999 and last address that can be used is 3000. No provisions are available for stack limit checks.

Reverse Polish Notation

Reverse Polish Notation is a way of expressing arithmetic expressions that avoids the use of brackets to define priorities for evaluation of operators. Computer uses stack to evaluate expression.

For ex:

Given Infix Notation (3 + 5) * (7 – 2) is converted to postfix, stored in stack and then evaluated using stack In this notation the above expression would be 3 5 + 7 2 – *

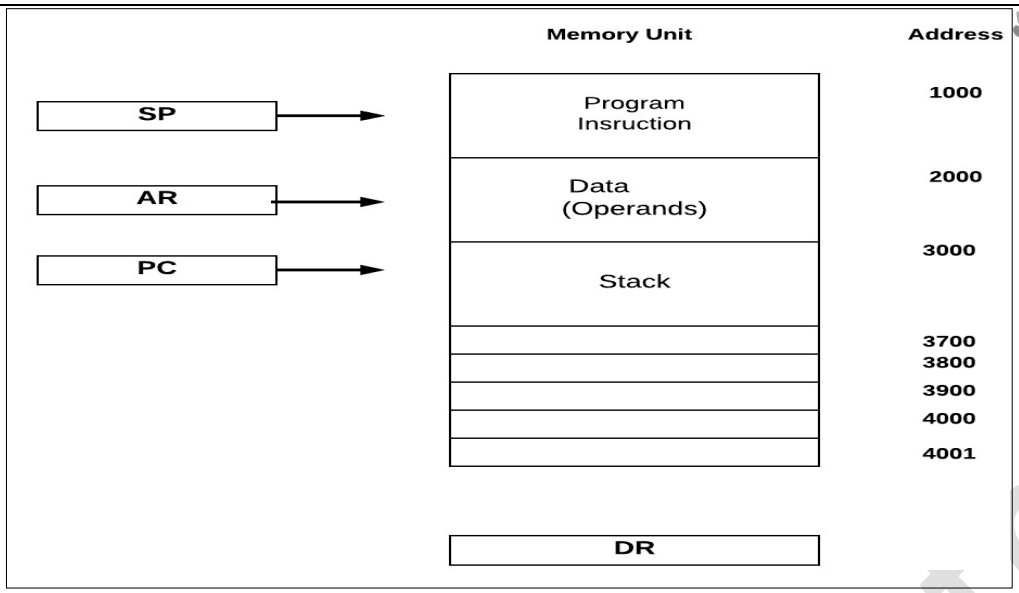


Fig 3.2 Memory Stack

Reading from left to right, this is interpreted as follows:

- Push 3 onto the stack.
- Push 5 onto the stack. Reading from the bottom, the stack now contains (3, 5).
- Apply the + operation: take the top two numbers off the stack, add them together, and put the result back on the stack. The stack now contains just the number 8.
- Push 7 onto the stack.
- Push 2 onto the stack. It now contains (8, 7, 2).
- Apply the – operation: take the top two numbers off the stack, subtract the top one from the one below, and put the result back on the stack. The stack now contains (8, 5).
- Apply the * operation: take the top two numbers off the stack, multiply them together, and put the result back on the stack. The stack now contains just the number 40.

Instruction Format

The instruction format consists of three fields. They are mode field, opcode field and the address field. The address field can also be divided into one, two, or three subfields. The number of address field in the instruction format depends on the internal organization or its register.
The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register. The bits of the instruction are divided into groups called fields.

15 14	12 11	0
Mode Field	**Opcode Field**	**Address Field**

Operation code: - The operation code field in the instruction specifies the operation to be performed. The operation is specified by binary code hence the name operation code or simply opcode.
Source / Destination operand: - The source/destination operand field directly specifies the source/destination operand for the instruction.
Source operand address: - The operation specified by the instruction may require one or more operands. The source operand may be in the CPU register or in the memory.
Destination operand address: - The operation executed by the CPU may produce result. Most of the time the results are stored in one of the operand. Such operand is known as destination

operand. s.

Next instruction address: - The next instruction address tells the CPU from where to fetch the next instruction after completion of execution of current instruction.

Address Instructions

In these instructions, the locations of all operands are defined implicitly. Such instructions are found in machines that store operands in a structure called a pushdown stack. A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack.

The following program shows how X = (A + B) * (C + D) will be written for a stack organized computer. (TOS stands for top of stack.)

THREE-ADDRESS INSTRUCTIONS: Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand. The program in assembly language that evaluates X = (A + B) ∗ (C + D) is shown below,

ADD R1, A, B	‘1 ← M [A] + M [B]
ADD R2, C, D	‘2 ← M [C] + M [D]
MUL X, R1, R2	M [X] ← ‘1 ∗ R2

TWO-ADDRESS INSTRUCTIONS: 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. The program to evaluate X = (A + B) ∗ (C + D) is as follows:

MOV R1, A	‘1 ← M [A]
ADD R1,B	‘1 ← ‘1 + M [B]
MOV R2, C	‘2 ← M [C]
ADD R2, D	‘2 ← ‘2 + M [D]
MUL R1,	‘2 ‘1 ← ‘1∗R2
MOV X, R1	M [X] ← ‘1

ONE-ADDRESS INSTRUCTIONS: One-address instructions use an implied accumulator (AC) register for all data manipulation. The program to evaluate X = (A + B) ∗ (C + D) is as follows:

LOAD A	AC ← M [A]
ADD B	AC ← A [C] + 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

ZERO-ADDRESS INSTRUCTIONS: A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack.

The following program shows how X = (A + B) ∗ (C + D) will be written for a stack organized computer. 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

The name “zero-address” is given to this type of computer because of the absence of an address field in the computational instructions.

RISC Architecture

The term ‘‘RISC stands for ‘‘Reduced Instruction Set Computer’’. It is a CPU design plan based on simple orders and acts fast. This is small or reduced set of instructions. Here, every instruction is expected to attain very small jobs. In this machine, the instruction sets are modest and simple, which help in comprising more complex commands. Each instruction is about the similar length; these are wound together to get compound tasks done in a single operation. Most commands are completed in one machine cycle. This pipelining is a crucial technique used to speed up RISC machines.

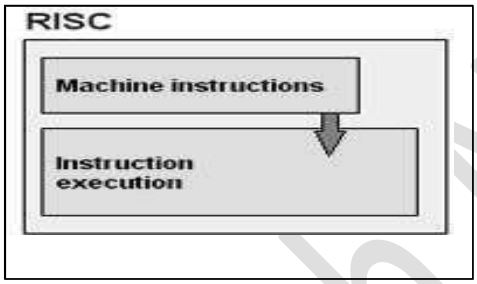


Fig 3.3 RISC architecture

The features of RISC include the following

- The demand of decoding is less
- Few data types in hardware
- General purpose register Identical
- Uniform instruction set
- Simple addressing nodes

CISC Architecture

The term CISC stands for ‘‘Complex Instruction Set Computer’’. It is a CPU design plan based on single commands, which are skilled in executing multi-step operations. CISC computers have small programs. It has a huge number of compound instructions, which takes a long time to perform. Here, a single set of instruction is protected in several steps; each instruction set has additional than 300 separate instructions. Maximum instructions are finished in two to ten machine cycles. In CISC, instruction pipelining is not easily implemented.

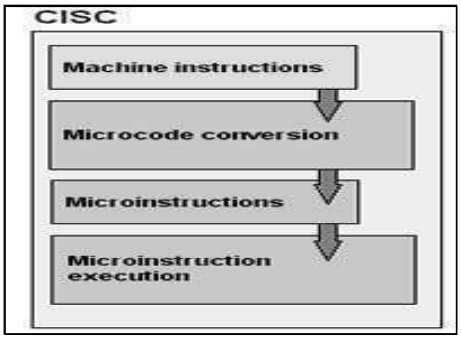


Fig 3.4 CISC architecture

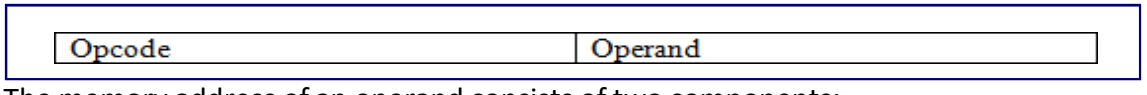
Difference between RISC and CISC

RISC	CISC
1. RISC stands for Reduced Instruction Set Computer.	1. CISC stands for Complex Instruction Set Computer.
2. RISC processors have simple instructions taking about one clock cycle. The average clock cycle per instruction (CPI) is 1.5	2. CSIC processor has complex instructions that take up multiple clocks for execution. The average clock cycle per instruction (CPI) is in the range of 2 and 15.
3. Performance is optimized with more focus on software	3. Performance is optimized with more focus on hardware.
4. It has no memory unit and uses a separate hardware to implement instructions.	4. It has a memory unit to implement complex instructions.
5. It has a hard-wired unit of programming.	5. It has a microprogramming unit.
6. The instruction set is reduced i.e. it has only a few instructions in the instruction set. Many of these instructions are very primitive.	**6.** The instruction set has a variety of different instructions that can be used for complex operations.
7. The instruction set has a variety of different instructions that can be used for complex operations.	**7.** CISC has many different addressing modes and can thus be used to represent higher-level programming language statements more efficiently.
8. Complex addressing modes are synthesized using the software.	8. CISC already supports complex addressing modes
9. Multiple register sets are present	9. Only has a single register set
10. RISC processors are highly pipelined	10. They are normally not pipelined or less pipelined
11. The complexity of RISC lies with the compiler that executes the program	11. The complexity lies in the microprogram
12. Execution time is very less	12. Execution time is very high
13. Code expansion can be a problem	13. Code expansion is not a problem
14. Decoding of instructions is simple.	14. Decoding of instructions is complex
15. It does not require external memory for calculations	15. It requires external memory for calculations
16. The most common RISC microprocessors are Alpha, ARC, ARM, AVR, MIPS, PA-RISC, PIC, Power Architecture, and SPARC.	16. Examples of CISC processors are the System/360, VAX, PDP-11, Motorola 68000 family, AMD and Intel x86 CPUs.
17. RISC architecture is used in high-end applications such as video processing, telecommunications and image processing.	17. CISC architecture is used in low-end applications such as security systems, home automation, etc.

Addressing Modes

Addressing Modes– The term addressing modes refers to the way in which the operand of an instruction is specified. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually executed.

An assembly language program instruction consists of two parts



The memory address of an operand consists of two components:

- **Starting address** of memory segment.

- **Effective address or Offset**: An offset is determined by adding any combination of three address elements: **displacement, base and index.**
- **Displacement:** It is an 8 bit or 16-bit immediate value given in the instruction.
    - **Base**: Contents of base register, BX or BP.
    - **Index**: Content of index register SI or DI.

There are different ways in which an operand may be specified in an instruction.

1. **Implied Mode:** In implied addressing mode, the instruction itself specifies the data to be operated. the implied addressing mode is also called implicit addressing mode, because there is no need to explicitly specify an effective address for either the source or the destination.
   For example -''complement accumulator''
2. **Immediate Mode:** In this mode the operand is specified in the instruction itself. The actual operand to be used in conjunction with the operation specified in the instruction is contained in the operand field.
   Example: MOVE A, #20
3. **Register Mode:** In this mode the operands are in registers that reside within the CPU. The register required is chosen from a register field in the instruction.
   Example: MOV R1, R2
4. **Register Indirect Mode:** In this mode the instruction specifies a register that contains the address of the operand and not the operand itself.
   Effective Address=R
   Example: MOVE A, (R0)
5. **Auto increment or Auto Decrement Mode:** After execution of every instruction from the data in memory it is necessary to increment or decrement the register. This is done by using the increment or decrement instruction.
   Example: MOVE R2), + R0
   MOVE (R2), - R0
6. **Direct Address Mode:** In this mode the operand resides in memory and its address is given directly by the address field of the instruction such that the effective address is equal to the address part of the instruction. Example: MOVE A, 2000
7. **Indirect Address Mode:** The effective address of the operand is the contents of a register or main memory location, location whose address appears in the instruction. Indirection is noted by placing the name of the register or the memory address given in the instruction in parentheses.
   Effective address = address part of instruction + context of CPU register
8. **Relative Address Mode:** In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.
   EA = PC + Address part of instruction
9. **Indexed Addressing Mode:** In this mode the effective address is obtained by adding the content of an index register to the address part of the instruction.
   EA= offset + R
   Example: MOVE 20 [R1], R2
8. **Base Register Addressing Mode:** In this mode the effective address is obtained by adding the content of a base register to the part of the instruction. A base register is assumed to hold a base address and the address field of the instruction, and gives a displacement relative to this base address. The base register addressing mode is handy for relocation of programs in memory to another as required in multi programming systems
   Example: ADD AX, [BX+SI]

Table: Eight Addressing Modes for load Instruction

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

Instruction Cycle

An instruction cycle, also known as **fetch-decode-execute cycle** is the basic operational process of a computer. This process is repeated continuously by CPU from boot up to shut down of computer.
Following are the steps that occur during an instruction cycle:

1. Fetch the Instruction
The instruction is fetched from memory address that is stored in PC(Program Counter) and stored in the instruction register IR. At the end of the fetch operation, PC is incremented by 1 and it then points to the next instruction to be executed.

2. Decode the Instruction
The instruction in the IR is executed by the decoder.
Read the Effective Address
If the instruction has an indirect address, the effective address is read from the memory. Otherwise operands are directly read in case of immediate operand instruction.

3. Execute the Instruction
The Control Unit passes the information in the form of control signals to the functional unit of CPU. The result generated is stored in main memory or sent to an output device.

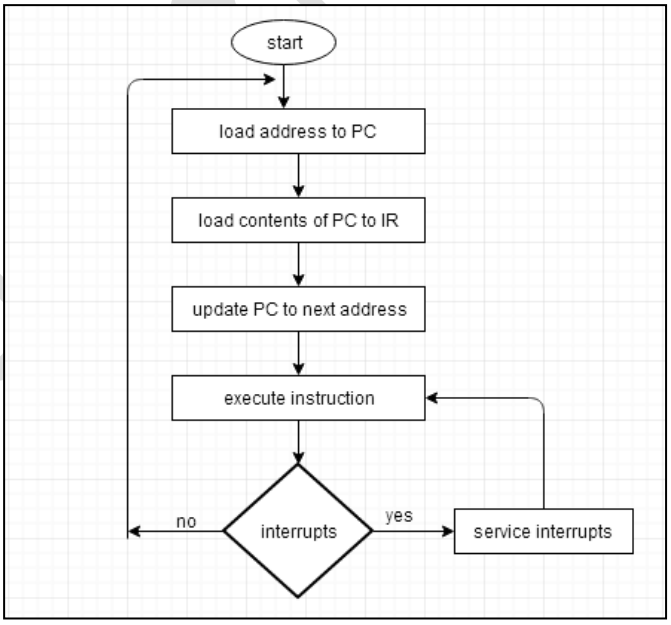


Fig 3.5 Instruction Cycle

The cycle is then repeated by fetching the next instruction. Thus, in this way the instruction cycle is repeated continuously.

Modes of Data Transfer

Mode of Transfer:

The binary information that is received from an external device is usually stored in the memory unit. The information that is transferred from the CPU to the external device is originated from the memory unit. CPU merely processes the information but the source and target is always the memory unit. Data transfer between CPU and the I/O devices may be done in different modes.

Data transfer to and from the peripherals may be done in any of the three possible ways

1. Programmed I/O.
2. Interrupt- initiated I/O.
3. Direct memory access(DMA).

1. **Programmed I/O:** It is due to the result of the I/O instructions that are written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually the transfer is from a CPU register and memory. In this case it requires constant monitoring by the CPU of the peripheral devices.

   Example of Programmed I/O: In this case, the I/O device does not have direct access to the memory unit. A transfer from I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from device to the CPU and store instruction to transfer the data from CPU to memory. In programmed I/O, the CPU stays in the program loop until the I/O unit indicates that it is ready for data transfer. This is a time-consuming process since it needlessly keeps the CPU busy. This situation can be avoided by using an interrupt facility. This is discussed below.
2. **Interrupt- initiated I/O:** Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer. By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device. In the meantime, the CPU can proceed for any other program execution. The interface meanwhile keeps monitoring the device. Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer. Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.
3. **Direct Memory Access**: The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU. Thus, we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access. During DMA the CPU is idle and it has no control over the memory buses. The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit.

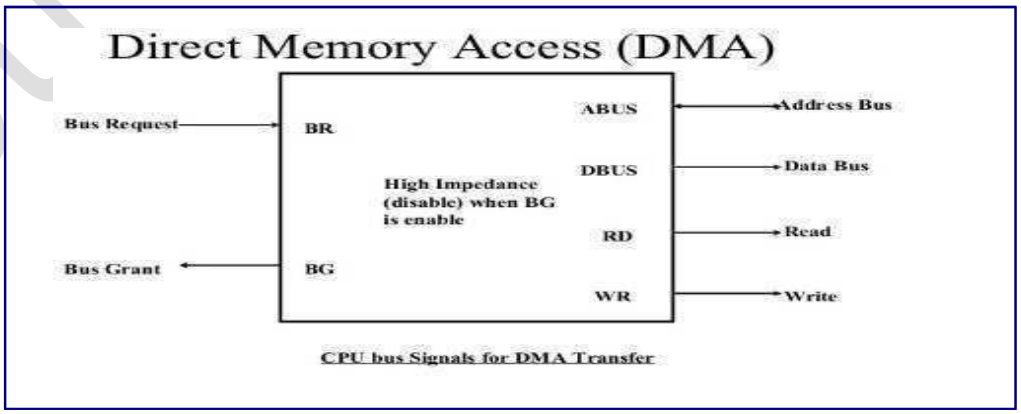


Fig 3.6 DMA

Priority Interrupt

A priority interrupt is a system which decides the priority at which various devices, which generates the interrupt signal at the same time, will be serviced by the CPU. The system has authority to decide which

conditions are allowed to interrupt the CPU, while some other interrupt is being serviced. Generally, devices with high speed transfer such as magnetic disks are given high priority and slow devices such as keyboards are given low priority.

When two or more devices interrupt the computer simultaneously, the computer services the device with the higher priority first.

Types of Interrupts:

Hardware Interrupts

When the signal for the processor is from an external device or hardware then these interrupts is known as **hardware interrupt**. Let us consider an example: when we press any key on our keyboard to do some action, then this pressing of the key will generate an interrupt signal for the processor to perform certain action. Such an interrupt can be of two types:

- **Maskable Interrupt**
  The hardware interrupts which can be delayed when a much high priority interrupt has occurred at the same time.
- **Non Maskable Interrupt**
  The hardware interrupts which cannot be delayed and should be processed by the processor immediately.

Software Interrupts

The interrupt that is caused by any internal system of the computer system is known as a **software interrupt**. It can also be of two types:

- **Normal Interrupt**
  The interrupts that are caused by software instructions are called **normal software interrupts**.
- **Exception**
  Unplanned interrupts which are produced during the execution of some program are called **exceptions**, such as division by zero.

Daisy Chaining Priority

In daisy chaining system all the devices are connected in a serial form. This way of deciding the interrupt priority consists of serial connection of all the devices which generates an interrupt signal. The device with the highest priority is placed at the first position followed by lower priority devices and the device which has lowest priority among all is placed at the last in the chain.

. The interrupt line request is common to all devices. If any device has interrupt signal in low level state then interrupt line goes to low level state and enables the interrupt input in the CPU. When there is no interrupt the interrupt line stays in high level state. The CPU respond to the interrupt by enabling the interrupt acknowledge line. This signal is received by the device 1 at its PI input. The acknowledge signal passes to next device through PO output only if device 1 is not requesting an interrupt.

The following figure 3.7 shows the block diagram for daisy chaining priority system.

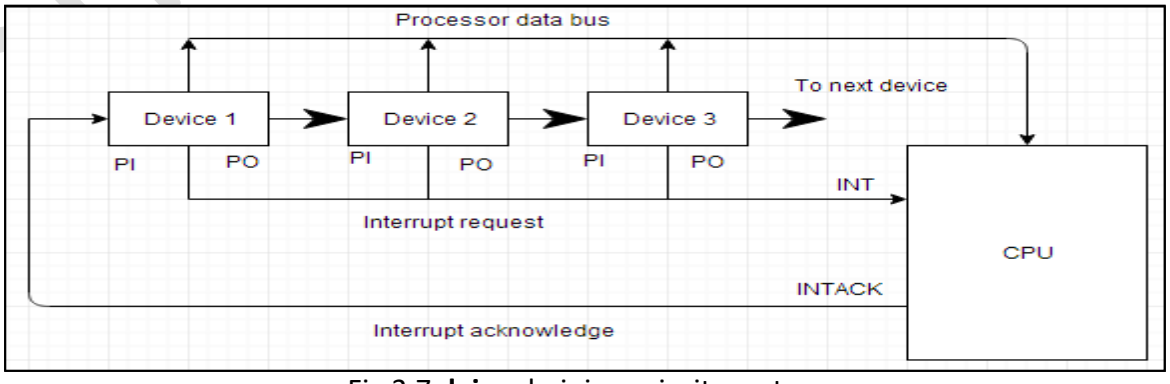


Fig 3.7 **daisy** chaining priority system

Input/output Processor

An input-output processor (IOP) is a processor with direct memory access capability. In this, the computer system is divided into a memory unit and number of processors. Each IOP controls and

manage the input-output tasks. The IOP is similar to CPU except that it handles only the details of I/O processing. The IOP can fetch and execute its own instructions. These IOP instructions are designed to manage I/O transfers only.

Block Diagram Of IOP

Below fig 3.8 is a block diagram of a computer along with various I/O Processors. The memory unit occupies the central position and can communicate with each processor. The CPU processes the data required for solving the computational tasks. The IOP provides a path for transfer of data between peripherals and memory. The CPU assigns the task of initiating the I/O program. The IOP operates independent from CPU and transfer data between peripherals and memory.

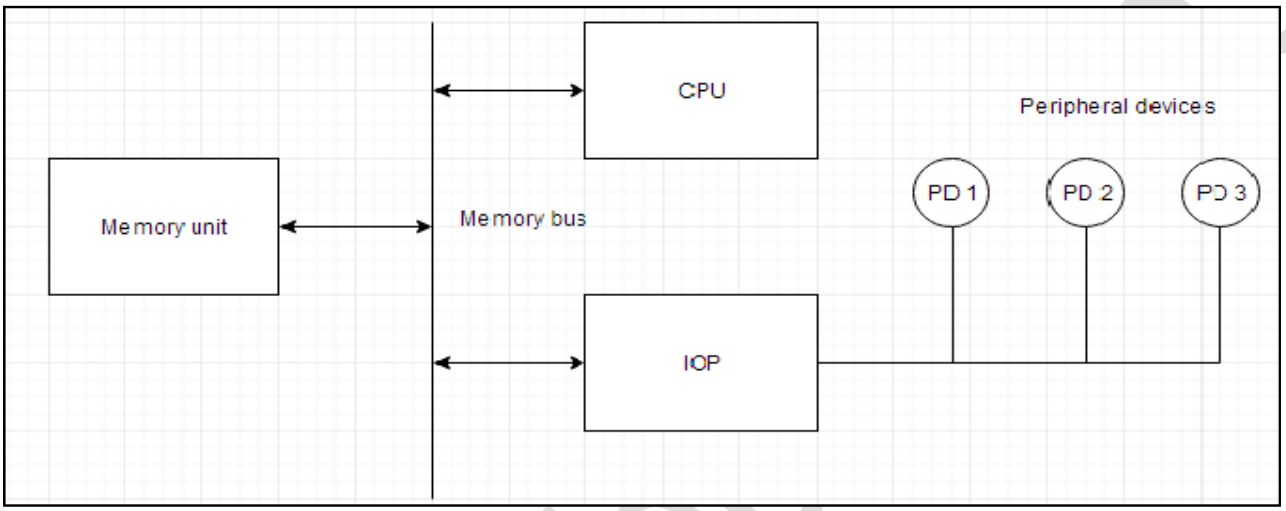


Fig 3.8 Block Diagram Of IOP

The communication between the IOP and the devices is similar to the program control method of transfer. And the communication with the memory is similar to the direct memory access method.

In large scale computers, each processor is independent of other processors and any processor can initiate the operation. The CPU can act as master and the IOP act as slave processor. The CPU assigns the task of initiating operations but it is the IOP, who executes the instructions, and not the CPU. CPU instructions provide operations to start an I/O transfer. The IOP asks for CPU through interrupt. Instructions that are read from memory by an IOP are also called commands to distinguish them from instructions that are read by CPU. Commands are prepared by programmers and are stored in memory. Command words make the program for IOP. CPU informs the IOP where to find the commands in memory.

UNIT-IV Parallel Processing

Parallel Processing

Instead of processing each instruction sequentially, a parallel processing system provides concurrent data processing to increase the execution time. In this the system may have two or more ALU's and should be able to execute two or more instructions at the same time. The purpose of parallel processing is to speed up the computer processing capability and increase its throughput.

Parallel processing can be viewed from various levels of complexity. At the lowest level, we distinguish between parallel and serial operations by the type of registers used. At the higher level of complexity, parallel processing can be achieved by using multiple functional units that perform many operations simultaneously.

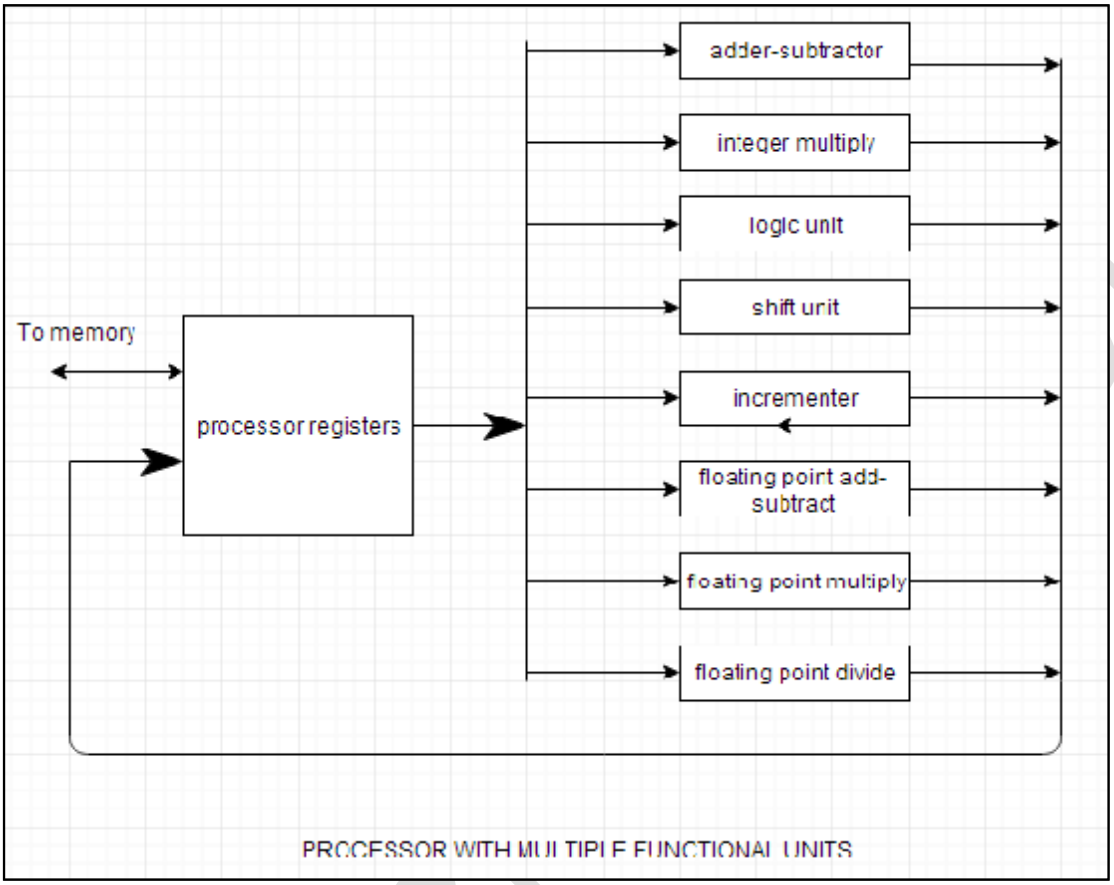


Fig 5.1 processor with multiple functional units

Data Transfer Modes of a Computer System

According to the data transfer mode, computer can be divided into 4 major groups:

SISD (Single Instruction Stream, Single Data Stream)

It represents the organization of a single computer containing a control unit, processor unit and a memory unit. Instructions are executed sequentially. It can be achieved by pipelining or multiple functional units.

SIMD (Single Instruction Stream, Multiple Data Stream)

It represents an organization that includes multiple processing units under the control of a common control unit. All processors receive the same instruction from control unit but operate on different parts of the data.

They are highly specialized computers. They are basically used for numerical problems that are expressed in the form of vector or matrix. But they are not suitable for other types of computations

MISD (Multiple Instruction Stream, Single Data Stream)

It consists of a single computer containing multiple processors connected with multiple control units and a common memory unit. It is capable of processing several instructions over single data stream simultaneously. MISD structure is only of theoretical interest since no practical system has been constructed using this organization.

MIMD (Multiple Instruction Stream, Multiple Data Stream

It represents the organization which is capable of processing several programs at same time. It is the

organization of a single computer containing multiple processors connected with multiple control units and a shared memory unit. The shared memory unit contains multiple modules to communicate with all processors simultaneously. Multiprocessors and multicomputer are the examples of MIMD. It fulfils the demand of large scale computations.

Pipelining

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

Pipeline system is like the modern-day assembly line setup in factories. For example, in a car manufacturing industry, huge assembly lines are setup and at each point, there are robotic arms to perform a certain task, and then the car moves on ahead to the next arm.

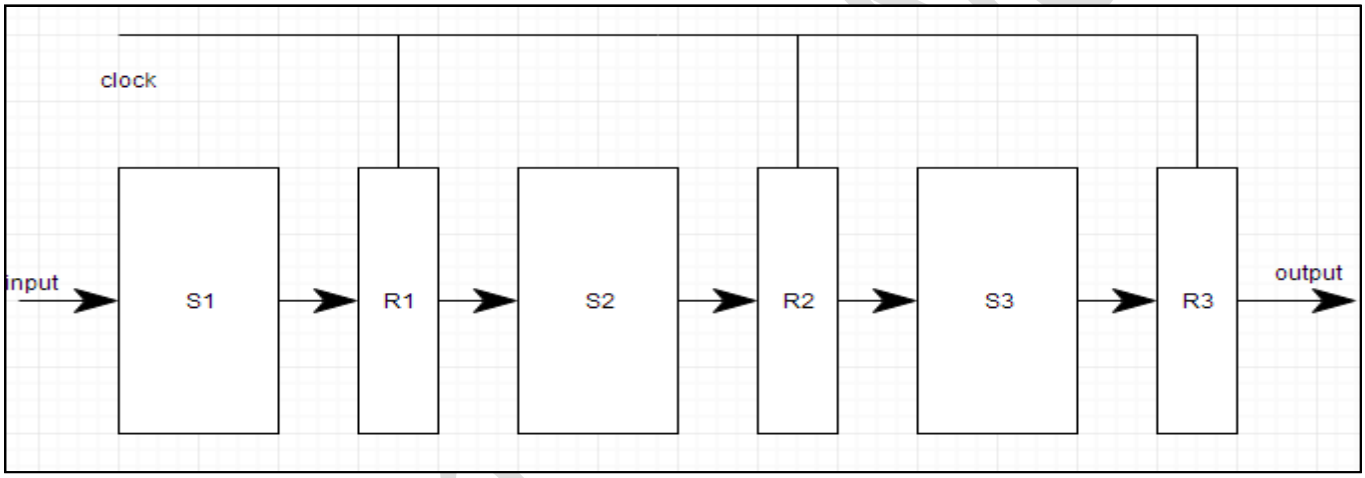


Fig 5.2 pipelining

Types of Pipeline: It is divided into 2 categories:

Arithmetic Pipeline

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

X = A*2^a

Y = B*2^b

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

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

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

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

Instruction Pipeline

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

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

Pipeline Conflicts

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

Timing Variations

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

Data Hazards

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

Branching

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

Interrupts

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

Data Dependency

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

Advantages of Pipelining

1. The cycle time of the processor is reduced.
2. It increases the throughput of the system
3. It makes the system reliable.

Disadvantages of Pipelining

1. The design of pipelined processor is complex and costly to manufacture.
2. The instruction latency is more.

Vector(Array) Processing

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

Scalar CPUs can manipulate one or two data items at a time, which is not very efficient. Also, simple instructions like ADD A to B, and store into C are not practically efficient. Addresses are used to point to the memory location where the data to be operated will be found, which leads to added overhead of data lookup. So, until the data is found, the CPU would be sitting ideal, which is a big performance issue. Hence, the concept of **Instruction Pipeline** comes into picture, in which the instruction passes through several sub-units in turn. These sub-units perform various independent functions, for example: the first one decodes the instruction, the second sub-unit fetches the data and the third sub-unit performs the math itself. Therefore, while the data is fetched for one instruction, CPU does not sit idle, it rather works on decoding the next instruction set, ending up working like an assembly line.

Vector processor, not only use Instruction pipeline, but it also pipelines the data, working on multiple data at the same time. A normal scalar processor instruction would be ADD A, B, which leads to addition of two operands, but what if we can instruct the processor to ADD a group of numbers (from 0 to n

memory location) to another group of numbers (let's say, n to k memory location). This can be achieved by vector processors. In vector processor a single instruction, can ask for multiple data operations, which saves time, as instruction is decoded once, and then it keeps on operating on different data items.

Applications of Vector Processors

The following are some areas where vector processing is used:

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

Memory interleaving

It is a technique for increasing memory speed. It is a process that makes the system more efficient, fast and reliable. For example: In the above example of 4 memory banks, data with virtual address 0, 1, 2 and 3 can be accessed simultaneously as they reside in separate memory banks, hence we do not have to wait for completion of a data fetch, to begin with the next. An interleaved memory with n banks is said to be **n-way interleaved**. In an interleaved memory system, there are still **two banks of DRAM** but logically the system seems one bank of memory that is twice as large.

In the interleaved bank representation below with 2 memory banks, the first long word of bank 0 is flowed by that of bank 1, which is followed by the second-long word of bank 0, which is followed by the second-long word of bank 1 and so on.

Types:

There are two methods for interleaving a memory:

- 2-Way Interleaved: Two memory blocks are accessed at same time for writing and reading operations.
- 4-Way Interleaved: Four memory blocks are accessed at the same time.

Multiprocessor system

A multiprocessor system is an interconnection of two or more CPU, with memory and input-output equipment. As defined earlier, multiprocessors can be put under MIMD category. The term multiprocessor is sometimes confused with the term multi computers. Though both support concurrent operations, there is an important difference between a system with multiple computers and a system with multiple processors. In a multi computers system, there are multiple computers, with their own operating systems, which communicate with each other, if needed, through communication links. A multiprocessor system, on the other hand, is controlled by a single operating system, which coordinate the activities of the various processors, either through shared memory or inter processor messages.

The advantages of multiprocessor systems are:

·Increased reliability because of redundancy in processors

·Increased throughput because of execution of multiple jobs in parallel portions of the same job in parallel

Characteristics of Multiprocessors

A multiprocessor system is an interconnection of two or more CPUs with memory and input-output equipment.

- The "processor" may be either a central processing unit (CPU) or an input-output processor (IOP).
- Multiprocessors are *multiple instruction streams, multiple data stream* (MIMD) systems
- Multiprocessing can enhance performance by decomposing a program into parallel executable tasks.
- The user can explicitly declare that certain tasks of the program to be executed in parallel.
- This must be done prior to loading the program by specifying the parallel executable segments.

- Other is to provide a compiler with multiprocessor software that can automatically detect parallelism in a user's program.
- A multiprocessor system with *common shared memory* is classified as a *shared-memory* or *tightly coupled multiprocessor*.
- Each processor element with its own *private local memory* is classified as a *distributed-memory* or *loosely coupled system.*
- when the interaction between tasks is minimal it is most efficient
- Multiprocessing improves the reliability of the system
- In a multiprocessor organization, multiple independent jobs can be made to operate in parallel.
- Also partition of a single job into multiple parallel tasks.
- The similarity and distinction between multiprocessor and multicomputer
- Both support concurrent operations Distinction
- The network consists of several autonomous computers, communication with each other may or may not take place.
- A multiprocessor system is controlled by one operating system which provides interaction between processors and all the components of the system

UNIT-IV Parallel Processing

Parallel Processing

Instead of processing each instruction sequentially, a parallel processing system provides concurrent data processing to increase the execution time. In this the system may have two or more ALU's and should be able to execute two or more instructions at the same time. The purpose of parallel processing is to speed up the computer processing capability and increase its throughput.

Parallel processing can be viewed from various levels of complexity. At the lowest level, we distinguish between parallel and serial operations by the type of registers used. At the higher level of complexity, parallel processing can be achieved by using multiple functional units that perform many operations simultaneously.

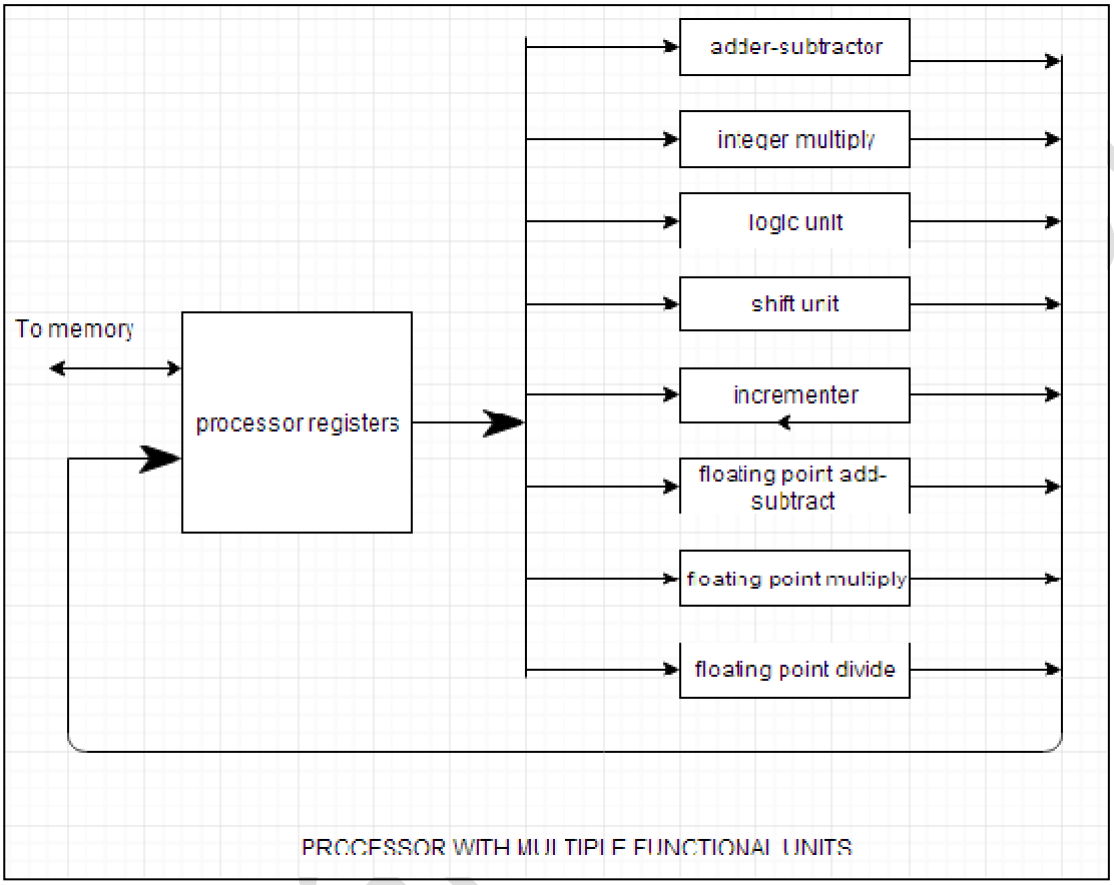


Fig 5.1 processor with multiple functional units

Data Transfer Modes of a Computer System

According to the data transfer mode, computer can be divided into 4 major groups:

SISD (Single Instruction Stream, Single Data Stream)

It represents the organization of a single computer containing a control unit, processor unit and a memory unit. Instructions are executed sequentially. It can be achieved by pipelining or multiple functional units.

SIMD (Single Instruction Stream, Multiple Data Stream)

It represents an organization that includes multiple processing units under the control of a common control unit. All processors receive the same instruction from control unit but operate on different parts of the data.

They are highly specialized computers. They are basically used for numerical problems that are expressed in the form of vector or matrix. But they are not suitable for other types of computations

MISD (Multiple Instruction Stream, Single Data Stream)

It consists of a single computer containing multiple processors connected with multiple control units and a common memory unit. It is capable of processing several instructions over single data stream simultaneously. MISD structure is only of theoretical interest since no practical system has been

constructed using this organization.

MIMD (Multiple Instruction Stream, Multiple Data Stream

It represents the organization which is capable of processing several programs at same time. It is the organization of a single computer containing multiple processors connected with multiple control units and a shared memory unit. The shared memory unit contains multiple modules to communicate with all processors simultaneously. Multiprocessors and multicomputer are the examples of MIMD. It fulfils the demand of large scale computations.

Pipelining

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

Pipeline system is like the modern-day assembly line setup in factories. For example, in a car manufacturing industry, huge assembly lines are setup and at each point, there are robotic arms to perform a certain task, and then the car moves on ahead to the next arm.

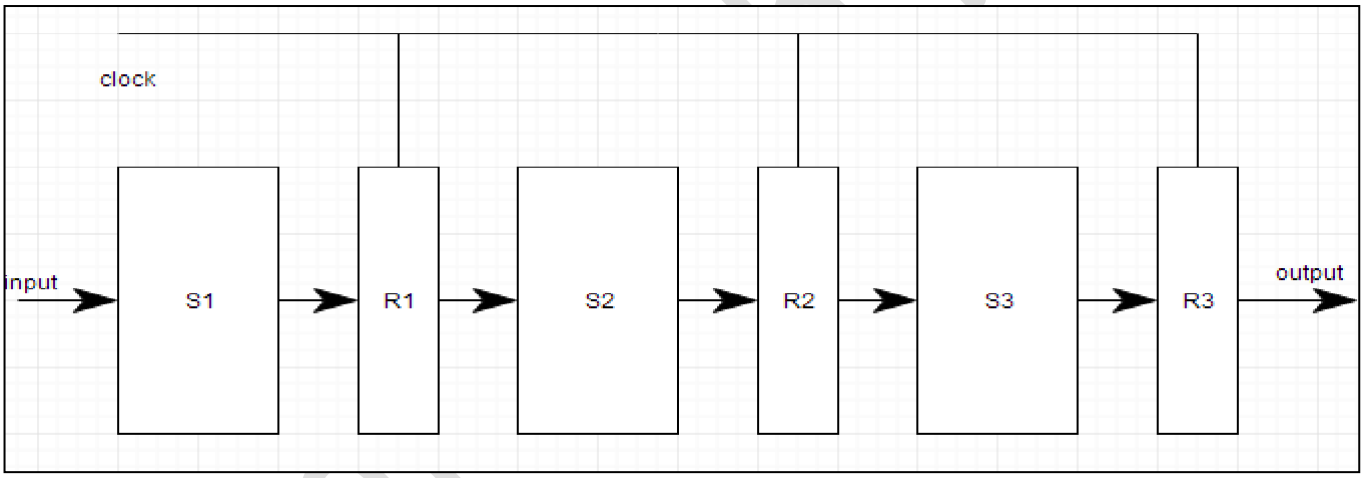


Fig 5.2 pipelining

Types of Pipeline: It is divided into 2 categories:

Arithmetic Pipeline

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

X = A*2^a

Y = B*2^b

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

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

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

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

Instruction Pipeline

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

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

Pipeline Conflicts

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

Timing Variations

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

Data Hazards

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

Branching

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

Interrupts

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

Data Dependency

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

Advantages of Pipelining

1. The cycle time of the processor is reduced.
2. It increases the throughput of the system
3. It makes the system reliable.

Disadvantages of Pipelining

1. The design of pipelined processor is complex and costly to manufacture.
2. The instruction latency is more.

Vector(Array) Processing

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

Scalar CPUs can manipulate one or two data items at a time, which is not very efficient. Also, simple instructions like ADD A to B, and store into C are not practically efficient. Addresses are used to point to the memory location where the data to be operated will be found, which leads to added overhead of data lookup. So, until the data is found, the CPU would be sitting ideal, which is a big performance issue. Hence, the concept of **Instruction Pipeline** comes into picture, in which the instruction passes through several sub-units in turn. These sub-units perform various independent functions, for example: the first one decodes the instruction, the second sub-unit fetches the data and the third sub-unit performs the math itself. Therefore, while the data is fetched for one instruction, CPU does not sit idle, it rather works on decoding the next instruction set, ending up working like an assembly line.

Vector processor, not only use Instruction pipeline, but it also pipelines the data, working on multiple data at the same time. A normal scalar processor instruction would be ADD A, B, which leads to addition of two operands, but what if we can instruct the processor to ADD a group of numbers (from 0 to n memory location) to another group of numbers (let's say, n to k memory location). This can be achieved by vector processors. In vector processor a single instruction, can ask for multiple data operations, which saves time, as instruction is decoded once, and then it keeps on operating on different data items.

Applications of Vector Processors

The following are some areas where vector processing is used:

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

Memory interleaving

It is a technique for increasing memory speed. It is a process that makes the system more efficient, fast and reliable. For example: In the above example of 4 memory banks, data with virtual address 0, 1, 2 and 3 can be accessed simultaneously as they reside in separate memory banks, hence we do not have to wait for completion of a data fetch, to begin with the next. An interleaved memory with n banks is said to be **n-way interleaved**. In an interleaved memory system, there are still **two banks of DRAM** but logically the system seems one bank of memory that is twice as large.

In the interleaved bank representation below with 2 memory banks, the first long word of bank 0 is flowed by that of bank 1, which is followed by the second-long word of bank 0, which is followed by the second-long word of bank 1 and so on.

Types:

There are two methods for interleaving a memory:

- 2-Way Interleaved: Two memory blocks are accessed at same time for writing and reading operations.
- 4-Way Interleaved: Four memory blocks are accessed at the same time.

Multiprocessor system

A multiprocessor system is an interconnection of two or more CPU, with memory and input-output equipment. As defined earlier, multiprocessors can be put under MIMD category. The term multiprocessor is sometimes confused with the term multi computers. Though both support concurrent operations, there is an important difference between a system with multiple computers and a system with multiple processors. In a multi computers system, there are multiple computers, with their own operating systems, which communicate with each other, if needed, through communication links. A multiprocessor system, on the other hand, is controlled by a single operating system, which coordinate the activities of the various processors, either through shared memory or inter processor messages.

The advantages of multiprocessor systems are:

·Increased reliability because of redundancy in processors

·Increased throughput because of execution of multiple jobs in parallel portions of the same job in parallel

Characteristics of Multiprocessors

A multiprocessor system is an interconnection of two or more CPUs with memory and input-output equipment.

- The "processor" may be either a central processing unit (CPU) or an input-output processor (IOP).
- Multiprocessors are *multiple instruction streams, multiple data stream* (MIMD) systems

- Multiprocessing can enhance performance by decomposing a program into parallel executable tasks.
- The user can explicitly declare that certain tasks of the program to be executed in parallel.
- This must be done prior to loading the program by specifying the parallel executable segments.
- Other is to provide a compiler with multiprocessor software that can automatically detect parallelism in a user's program.
- A multiprocessor system with *common shared memory* is classified as a *shared-memory* or *tightly coupled multiprocessor*.
- Each processor element with its own *private local memory* is classified as a *distributed-memory* or *loosely coupled system.*
- when the interaction between tasks is minimal it is most efficient
- Multiprocessing improves the reliability of the system
- In a multiprocessor organization, multiple independent jobs can be made to operate in parallel.
- Also partition of a single job into multiple parallel tasks.
- The similarity and distinction between multiprocessor and multicomputer
- Both support concurrent operations Distinction
- The network consists of several autonomous computers, communication with each other may or may not take place.
- A multiprocessor system is controlled by one operating system which provides interaction between processors and all the components of the system

What is Parallelism?

- Parallel processing is a term used to denote simultaneous computation in CPU for the purpose of measuring its computation speeds
- Parallel Processing was introduced because the sequential process of executing instructions took a lot of time

Parallel Processing

"Parallel processing is running different part of an activity simultaneously to achieve result quickly."

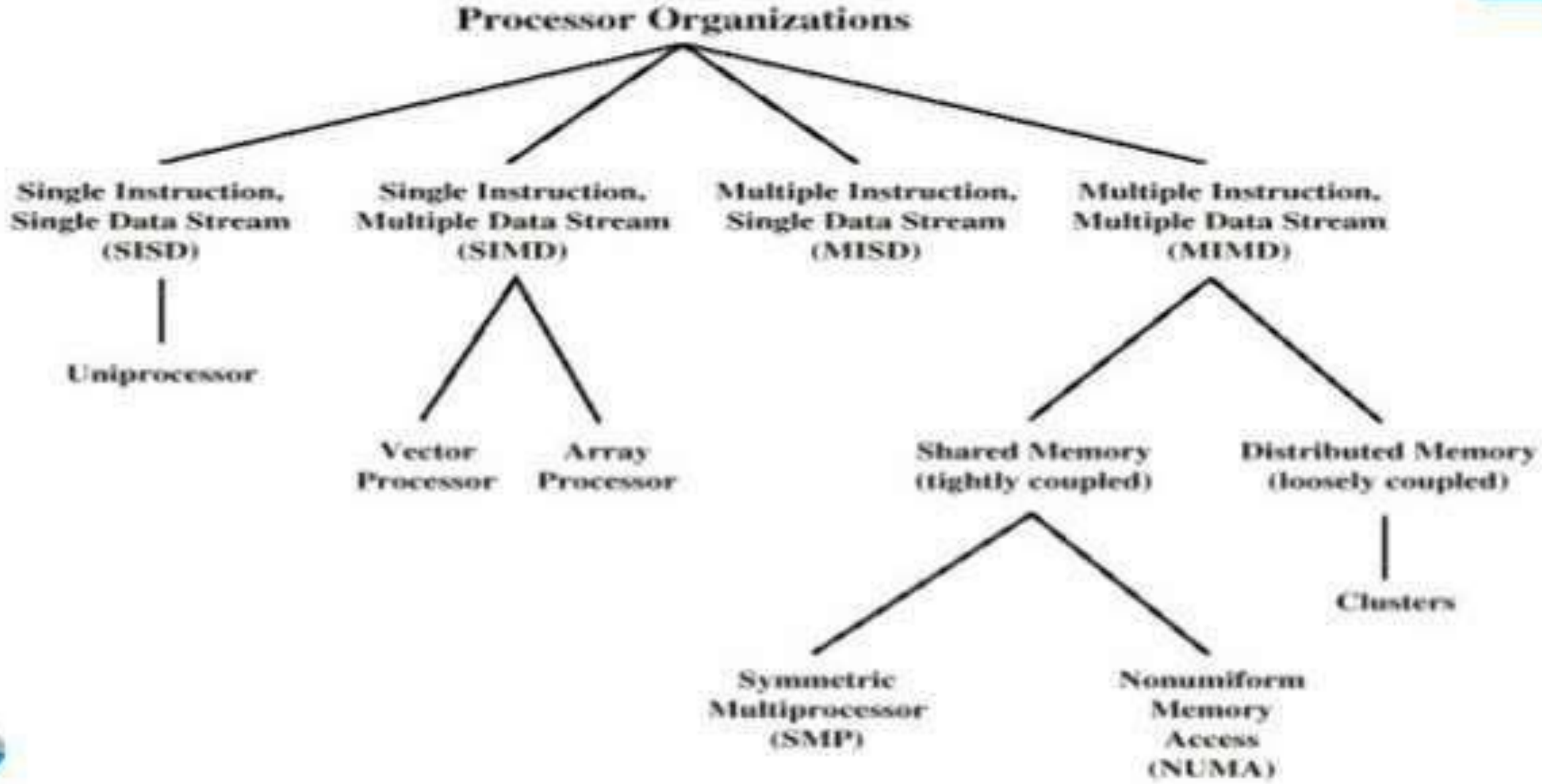
The activity is to be broken down into sub-activities which can be executed independently and parallely to produce result quickly than sequencial execution.

Applications are easier to write by following sequential execution model where instruction execute one after another in the order specified by the programmer.

Frequency scaling of microprocessors is used to produce result quickly. If fastest microprocessor is unable to meet the requirement or the computer is prohibitively costly, parallel processing is an alternative solution. **Multi-Core microprocessors have become common even in PCs because semi-conductor industry is finding it difficult to scale frequency** of chip at same rate as in past. Therefore, parallel processing would become important in PCs also.

The presentation is about computer architecture for parallel processing and it and takes bottom up approach (hardware to software).

Classification Parallel Processor Architectures

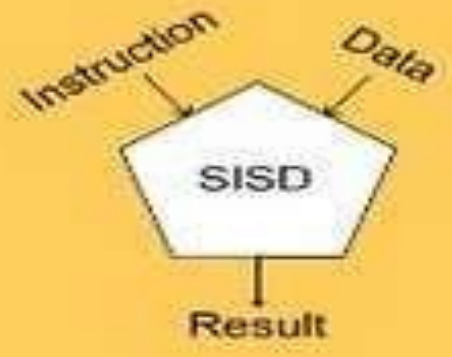


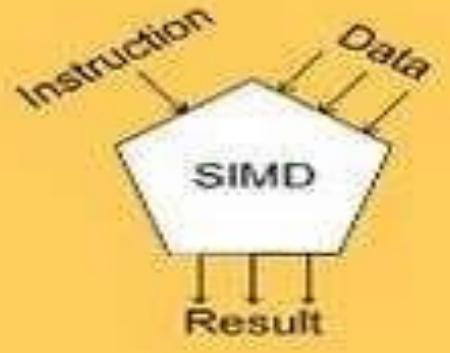
Michael Flynn's Classification

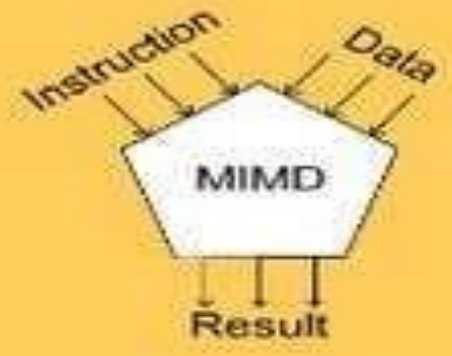
Michael Flynn's Taxonomy: Michael J Flynn's classification of architecture is based on simultaneous support of single or multiple instruction steams operating on single or multiple data sets.

Four cases get created:

- Single Instruction Single Data
- Multiple Instruction Single Data
- Single Instruction Multiple Data
- Multiple Instruction Multiple Data







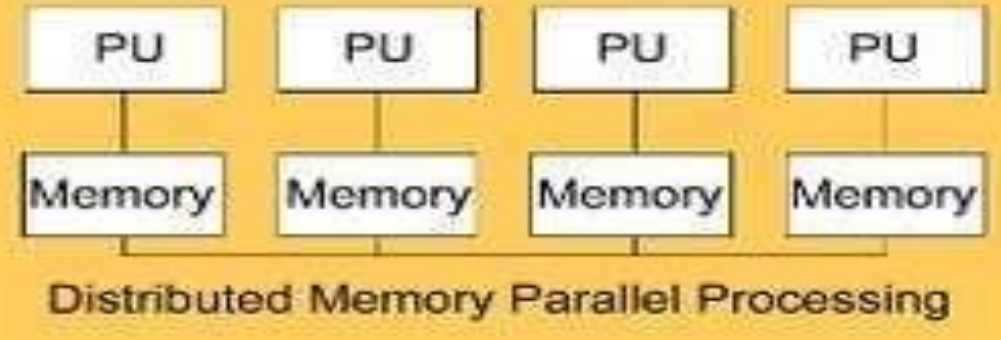
SISD represents a sequential computing. Other three categories (MISD, SIMD and MIMD) represent parallel computing. Practical machines are based on two of them- SIMD and MIMD.

Memory Model

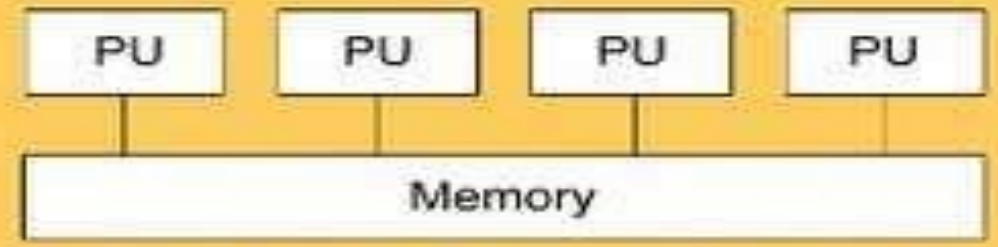
Shared vs Distributed Memory Systems

In **Shared Memory** parallel processing architecure, all processing units see the entire memory region.

In **Distributed Memory** architecure, each processing unit has its own private memory. Memory of other processing units are not visible.



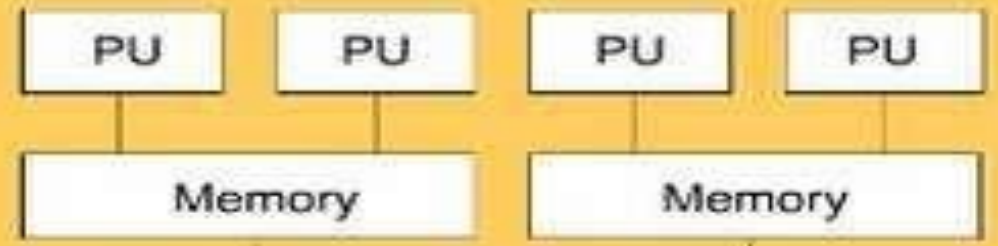
Distributed Memory Parallel Processing



Shared Memory Parallel Processing

Distributed Shared Memory Systems

This architecture is mix of both shared and distributed systems. A node consists of a memory shared by multiple processing units. Multiple such node make the system.



Distributed Shared Memory Parallel Processing

Machines for Parallel Processing

1. **Single instruction multiple data machines (SIMD)** on Shared memory.
   SIMD is always performed on Shared Memory
   Example: Vector processing in CPU

2. **Multi-CPU Machines.**
   A MIMD (multiple instruction multiple data) on Shared memory.
   Example: Multi-Core Microprocessor based machine, SMP, NUMA machines.

3. **Multiples Machines Networked Together.**
   A MIMD on Distributed memory.
   Example: Grid computing.

4. **Hybrid Systems.**
   Real life systems are too often hybrid of different concepts.
   Example: Massive Parallel Processing on Distributed Shared Memory.

The applications are very much hardware architecture dependent. Communication between sub-jobs, synchronisation and race condition handling varies depending on architecture.

Machines for Parallel Processing

Note:

1. SIMD is always performed on Shared Memory.

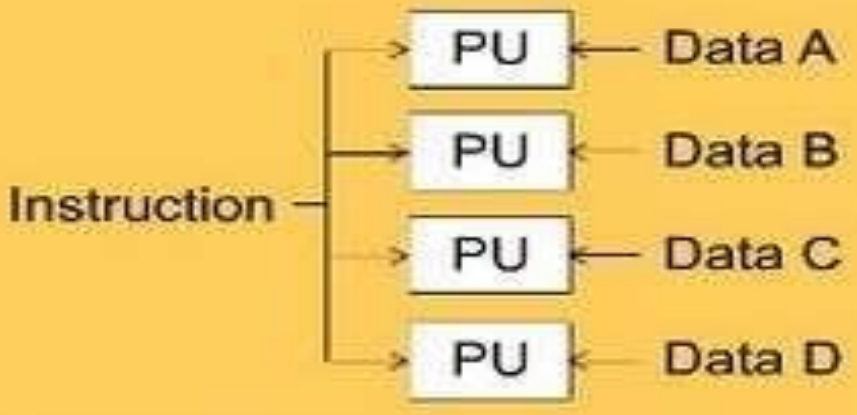
2. Graphics processing unit, DSP etc. may harness many of parallel processing concepts but they are not considered parallel computing because they do not perform general purpose computing.

*3. Parallel processing can be performed at hardware level without modifying application. Example: superscale architecture. These techniques, which are also knkwn as **'Implicit Parallelism'**, are build in hardware and not in scope of this presentation.*

Even after implicit parallelism by hardware, large scope remains for parallelization of the application explicitly. Presentation focuses on different type of hardwares mentioned in previous slide and how to develop application for them.

Single Instruction Multiple Data

An instruction is executed on multiple data sets simultaneously. SIMD is implemented in microprocessor. SIMD is also known as vector processing.



Single Instruction Multiple Data

Super computers of 70s/80s were based of SIMD. For example famous Cray machines. In SIMD architecture, matrix operation, loops operations can be very easily parallelized as long as data in a loop is independent of other loops.

Single Instruction Multiple Data

Latest Trend

In 90s, supercomputers moved away from SIMD to MIMD when off the self computer become very fast and cheap.

SIMD is now very common on small scale in general purpose CPUs for example MMI (multi-media extension) in Pentium microprocessor. Cell processor developed by IBM, used in PlayStation3, is heavily based on SIMD.

NEC's Earth Simulator is also based on their SX-6 SIMD architecture. Earth Simulator became faster supercomputer during 2002-04. Current trend in supercomputers is to use general purpose CPUs,earth simulator is an exception.

Multi-CPU Machine

In multi-cpu machine, an application can use multiple thread of execution simultaneously. Multiple computing units of the machine runs asynchronously and independently. It is an MIMD (Multiple Instruction Multiple Data) machine.

Resource of the machine (memory, files etc.) are shared and visible to all computing units of the machine.

Symmetry - Ideally, all resource (memory etc.) should be equidistance from the computing units(CU) and computing units identical. Such machines are known as symmetrical multi-processing (SMP).

When number of computing units on a machine grow to large number, maintaining symmetry is not always feasible. Common points such as memory bus etc. becomes bottle-neck. Non-symmetrical machines such as NUMA achitecture solves this problem.

If computing units are different(non-symmetrical), application development is radically different. For example IBM cell processor. Such machines are not in scope of the presentation.

NUMA Architecture

Non-Uniform Memory Architecture

In NUMA architecture, memory access time is dependent on memory location relative to processor. Each processor has local memory which can be accessed very fast. Memory access to local memory of other processor takes more time. In **ccNUMA**, cache coherency is maintained by hardware. Modification of a memory data by on processor is updated to all other processor which contain that memory in cache.

An application developed for SMP will also run on ccNUMA but performance will be bad if access to non-local memory is high. Sub-jobs can be made local and non-local memory aware to enhance performance, which is an additional level of complexity in application development.

Multi-Core Microprocessor

Multi-Core Microprocessor

A single silicon piece can contain two or more computation units, commonly known as core. Cache coherency is easy to maintain since cores are on same piece of silicon. These cores usually share cache memory and memory bus. Such processors are known as multi-core CPUs or multi-core microprocessors.

A multi-cpu machine can have two or more physically isolated single core microprocessors, a single multi-core microprocessor or combination of both.

Multiple Machines Networked Together

Compared to SIMD and Multi-CPU machines, variation in Networked machines are very large. Three important dimension of Networked machines are:

1. **Speed of the Network.**
   Examples: Connected through dedicate high speed network InfiniBand etc., Machine on WAN or Internet etc.

2. **Level Heterogeneity.**
   Examples: 32/64bit, Linux 2.4/2.6. Solaris /Windows Vista. Memory and hard disk size etc.

3. **Availability of Computational Resource.**
   Examples: All available together (dedicated machines or concept of reservation). Free machines shared by uses on Internet etc.

Application developed for one type of Networked machine is very un-likely to work efficiently on other type even if it runs. No widely accepted standard development philosophy exists for networked machines. However, a lot of libraries are available which can be used to fulfill specific need.

Message Passing

Message Passing

This is a solution for Distributed Memory system in which programmer explicitly divide data and work across the processing units, as well as manage the communications and syncronization among them with the help of standard interface. MPI and PVM is two such popular APIs.

Message Passing Interface(MPI) & Parallel Virtual Machine(PVM) are language-independent communications protocol, syncronization mechanism and load sharing/balancing system used for parallel programming on heterogeneous machines connected by network.

Multiple Machines Networked Together

Few Examples of Networked Machines for parallel processing:

UNIX/Windows/Mac machines on LAN or WAN can be used for parallel processing using standard features rsh, NFS and libraries MPI, PVM etc.

Grid Computing utilizes free computing resource available in machines on internet which belongs to different individuals and organizations. Since computers belong to different individuals/organizations, they may not always be trustworthy and availability is not predictable (owner of machine will add it to grid when not in use say in night). These two are difficulty of grid computing but the cost per unit of computation would be lowest since it is based on resource which will get wasted anyway.

Vector Processing

- There is a class of computational problems that are beyond the capabilities of the conventional computer.
- These are characterized by the fact that they require vast number of computation and it take a conventional computer days or even weeks to complete.
- Computers with vector processing are able to handle such instruction and they have application in following fields:

- Long range weather forecasting
- Petroleum exploration
- Seismic data analysis
- Medical diagnosis
- Aerodynamics and space simulation
- Artificial Intelligence and expert system
- Mapping the human genome
- Image Processing

Vector Operation

- A vector V of length n is represented as row vector by

$$V = [V_1 \quad V_2 \quad V_3 \quad \cdots \quad V_n]$$

- The element V_i of vector V is written as V(I) and the index I refers to a memory address or register where the number is stored.

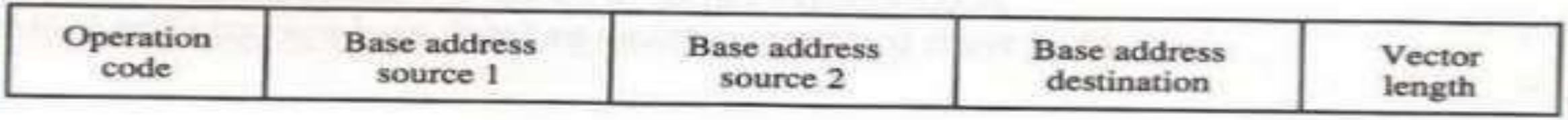
- Let us consider the program in assembly language that two vectors A and B of length 100 and put the result in vector C.

```
        Initialize I = 0
20      Read A(I)
        Read B(I)
        Store C(I) = A(I) + B(I)
        Increment I = I + 1
        If I ≤ 100 go to 20
        Continue
```

- A computer capable of vector processing eliminates the overhead associated with the time it takes to fetch and execute the instructions in the program loop.
- It allows operations to be specified with a single vector instruction of the form:

$$C(1:100) = A(1:100) + B(1:100)$$

Figure Instruction format for vector processor.



Matrix Multiplication

- Let us consider the multiplication of two 3*3 matrix A and B.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

The product matrix C is a 3×3 matrix whose elements are related to the elements of A and B by the inner product:

$$c_{ij} = \sum_{k=1}^{3} a_{ik} \times b_{kj}$$

For example, the number in the first row and first column of matrix C is calculated by letting $i = 1, j = 1$, to obtain

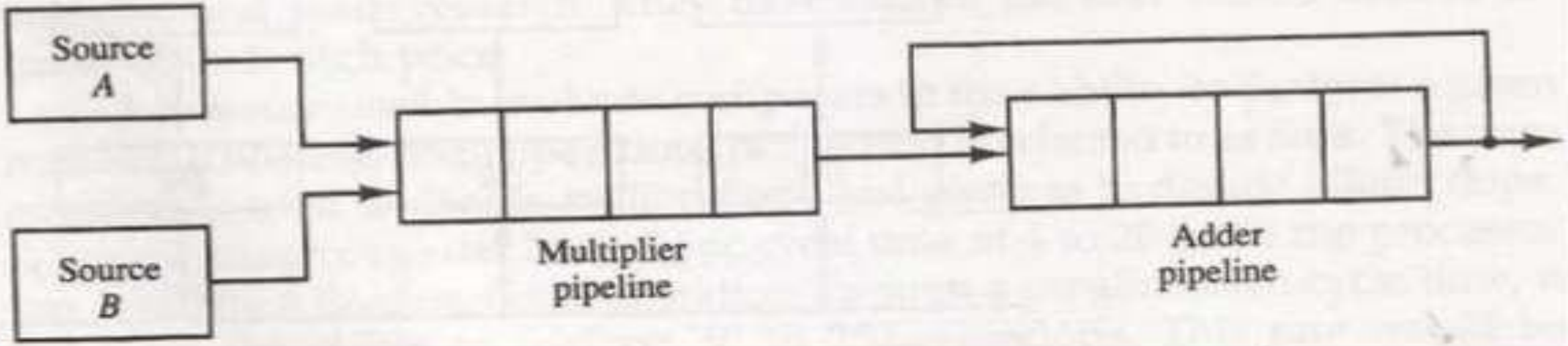
$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}$$

- This requires three multiplication and(after initializing c_{11} to 0) three addition.
- Total number of addition or multiplication required is 3*9.
- In general inner product consists of the sum of k product terms of the form:

$$C = A_1 B_1 + A_2 B_2 + A_3 B_3 + A_4 B_4 + \cdots + A_k B_k$$

- In typical application value of k may be 100 or even 1000.
- The inner product calculation on a pipeline vector processor is shown below.
- Floating point adder and multiplier are assumed to have four segments each.

Figure Pipeline for calculating an inner product.

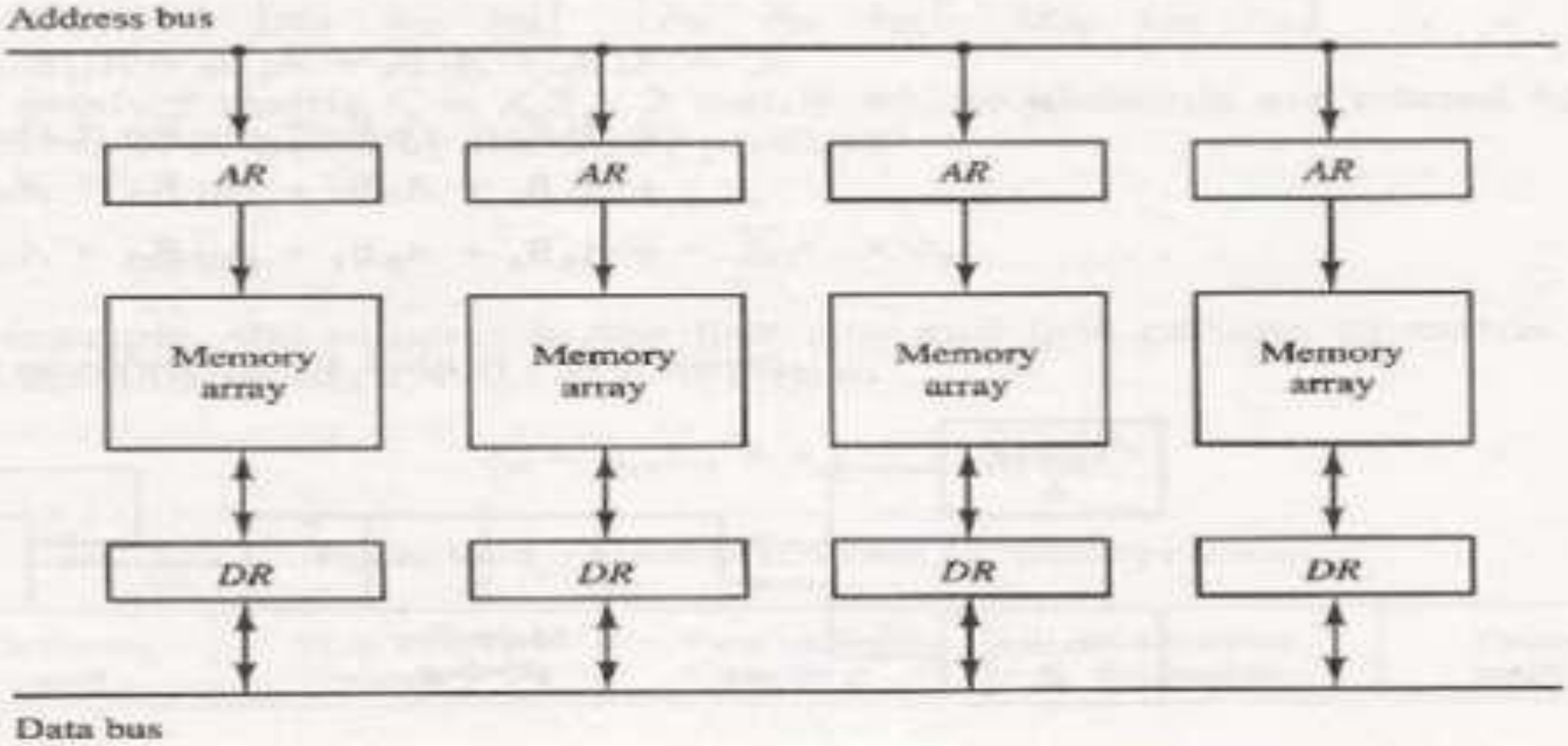


$$C = A_1 B_1 + A_5 B_5 + A_9 B_9 + A_{13} B_{13} + \cdots$$
$$+ A_2 B_2 + A_6 B_6 + A_{10} B_{10} + A_{14} B_{14} + \cdots$$
$$+ A_3 B_3 + A_7 B_7 + A_{11} B_{11} + A_{15} B_{15} + \cdots$$
$$+ A_4 B_4 + A_8 B_8 + A_{12} B_{12} + A_{16} B_{16} + \cdots$$

- The four partial sum are added to form the final sum

Memory Interleaving

Figure Multiple module memory organization.



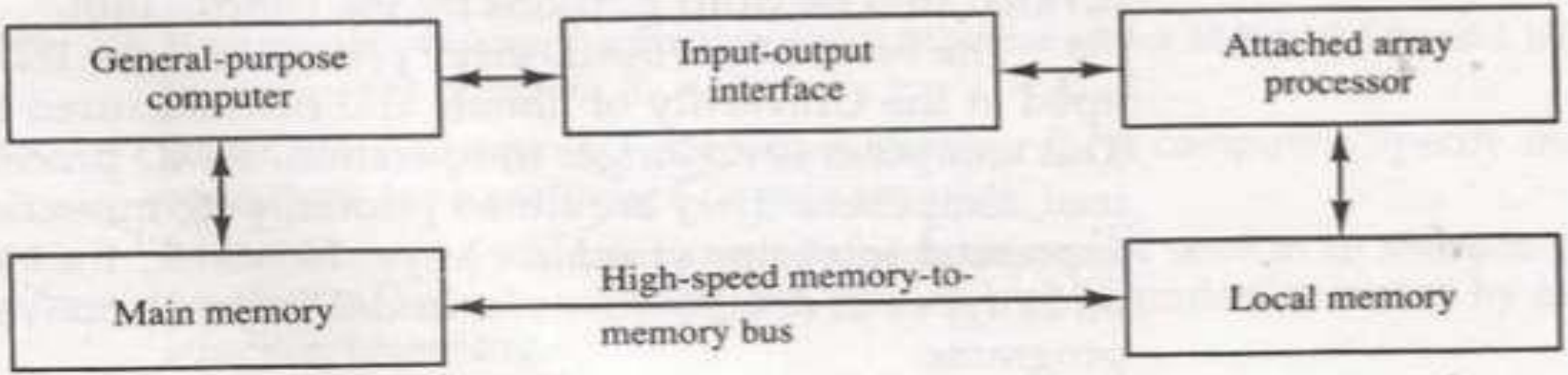
Array Processor

- An array processor is a processor that performs the computations on large arrays of data.
- There are two different types of array processor:
  - Attached Array Processor
  - SIMD Array Processor

Attached Array Processor

- It is designed as a peripheral for a conventional host computer.
- Its purpose is to enhance the performance of the computer by providing vector processing.
- It achieves high performance by means of parallel processing with multiple functional units.

Figure Attached array processor with host computer.



SIMD Array Processor

- It is processor which consists of multiple processing unit operating in parallel.
- The processing units are synchronized to perform the same task under control of common control unit.
- Each processor elements(PE) includes an ALU , a floating point arithmetic unit and working register.

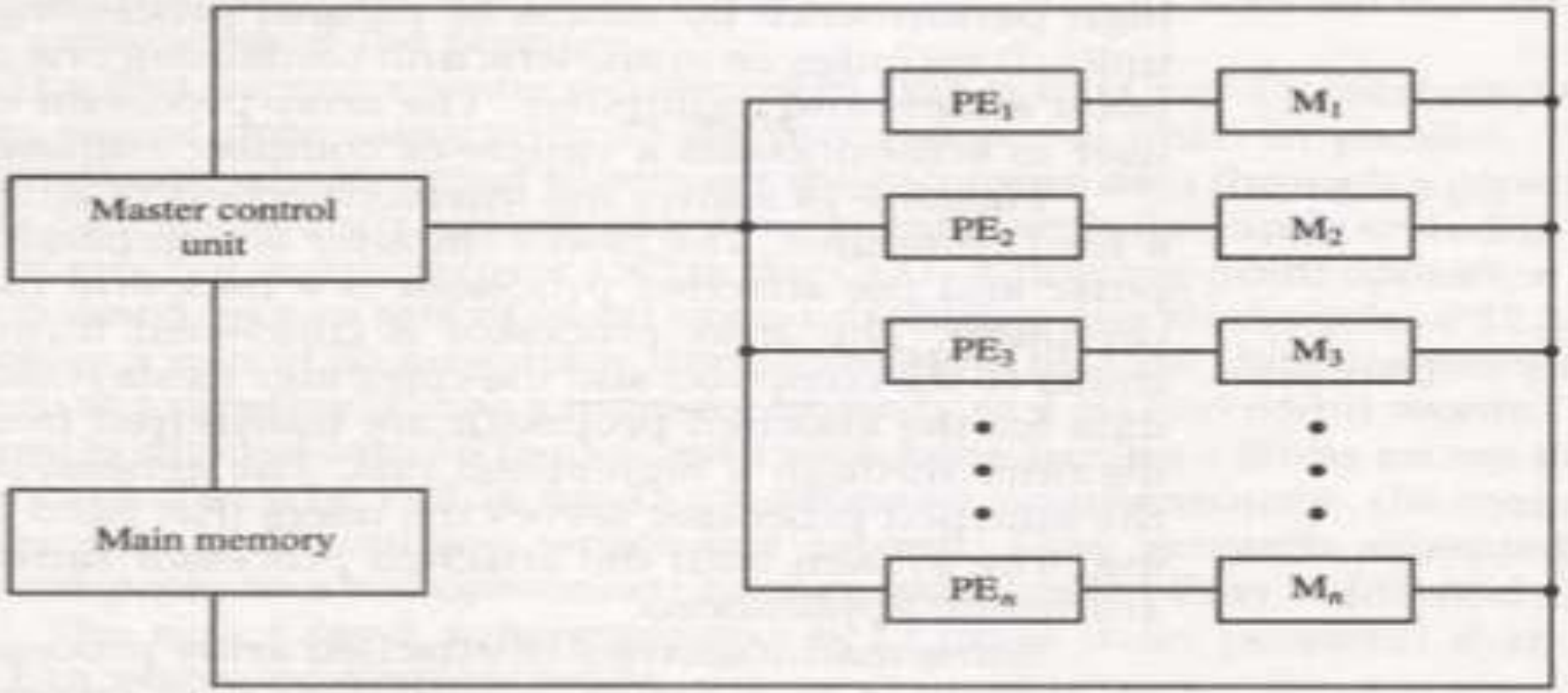


Figure SIMD array processor organization.

PIPELINING

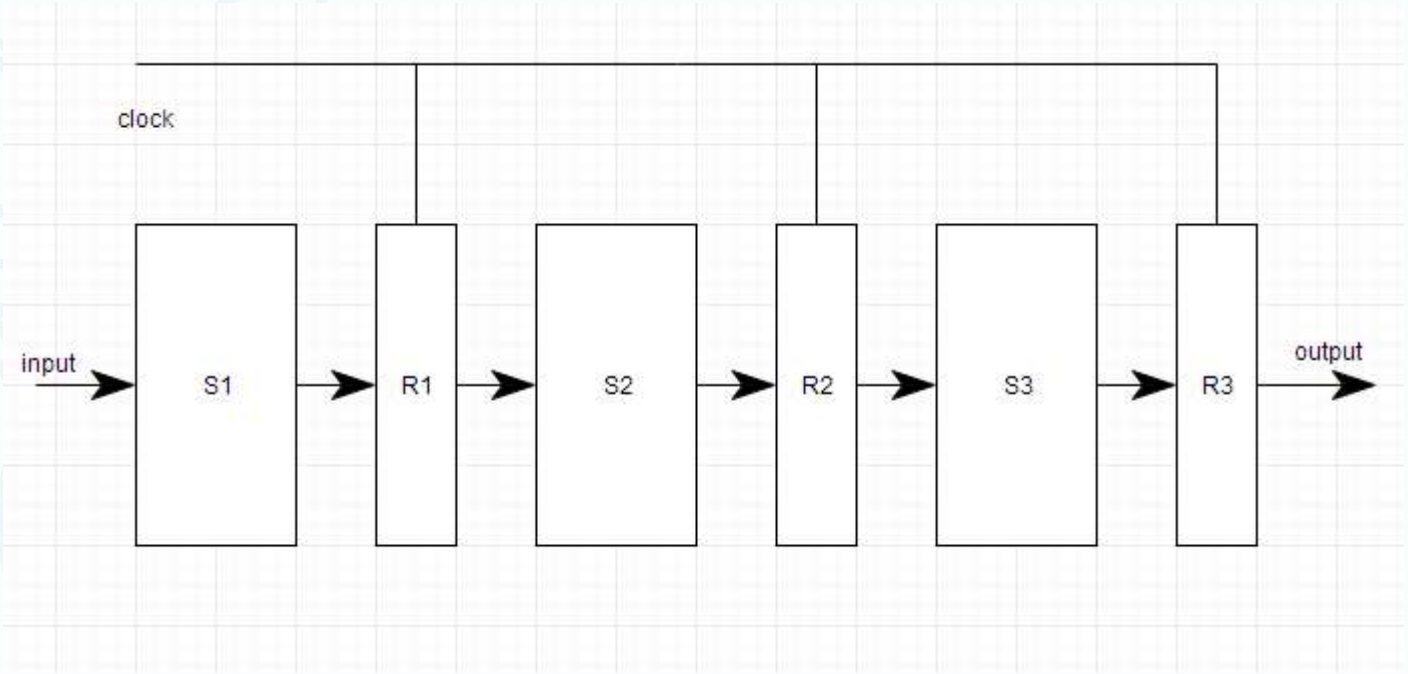
INTRODUCTION

- What is Pipelining?
- Types of Pipeline
- Pipeline Conflicts
- Advantages of Pipelining
- Disadvantages of Pipelining

What is Pipelining?

- Pipelining is the process of accumulating instruction from the processor through a pipeline.
- It allows storing and executing instructions in an orderly process. It is also known as **pipeline processing**.
- Pipelining is a technique where multiple instructions are overlapped during execution.
- Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure.
- Instructions enter from one end and exit from another end.

- In pipeline system, each segment consists of an input register followed by a combinational circuit.
- The register is used to hold data and combinational circuit performs operations on it.
- The output of combinational circuit is applied to the input register of the next segment.



Types of Pipeline

1. Arithmetic Pipeline
2. Instruction Pipeline

Instruction Pipeline

- In this a stream of instructions can be executed by overlapping *fetch*, *decode* and *execute* phases of an instruction cycle.
- This type of technique is used to increase the throughput of the computer system.
- An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline.
- Thus we can execute multiple instructions simultaneously.
- The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.

Arithmetic Pipeline

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

  $X = A*2^a$

  $Y = B*2^b$
- Here A and B are mantissas (significant digit of floating point numbers), while a and b are exponents.

Pipeline Conflicts

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

1. Timing Variations
2. Data Hazards
3. Branching
4. Interrupts
5. Data Dependency

Advantages of Pipelining

1.The cycle time of the processor is reduced.
2.It increases the throughput of the system
3.It makes the system reliable.

Disadvantages of Pipelining

1.The design of pipelined processor is complex and costly to manufacture.
2.The instruction latency is more.