RGPVNOTES.IN

Program : **B.Tech**

Subject Name: **Computer Architecture**

Subject Code: **IT-402**

Semester: **4th**

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

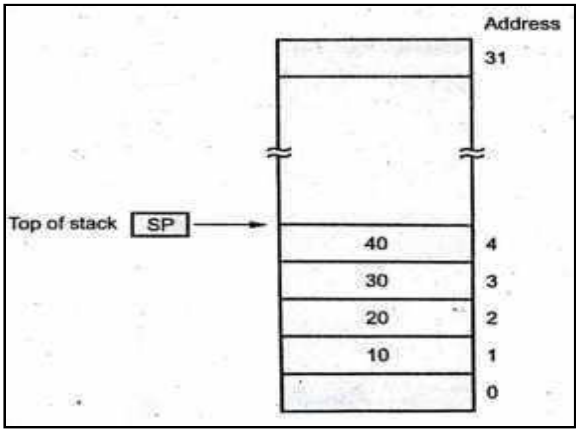


Figure 3.1 word register stack

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 – *

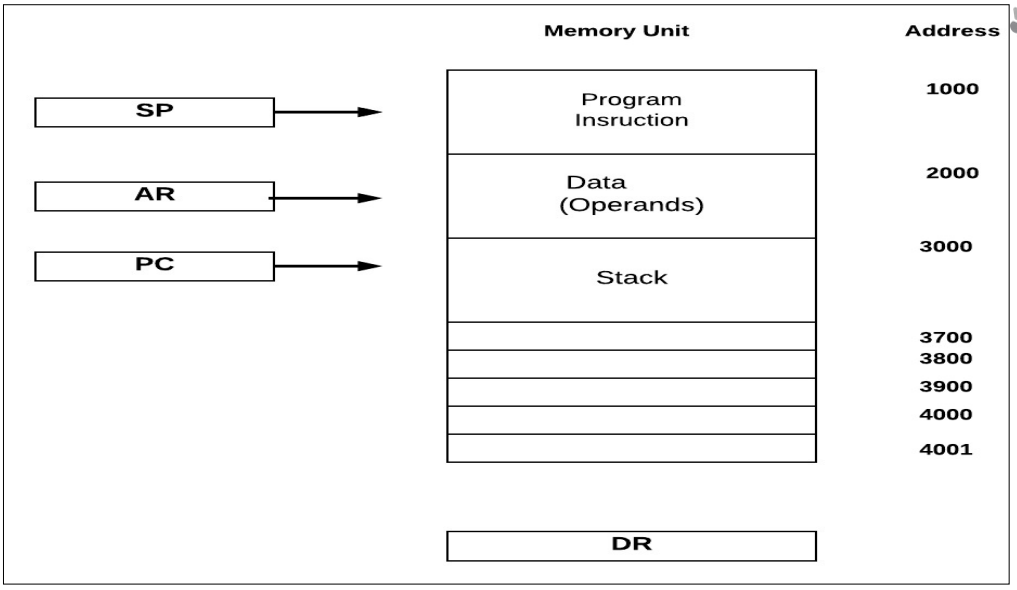


Fig 3.2 Memory Stack

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.

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

ONE-ADDRESS 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.

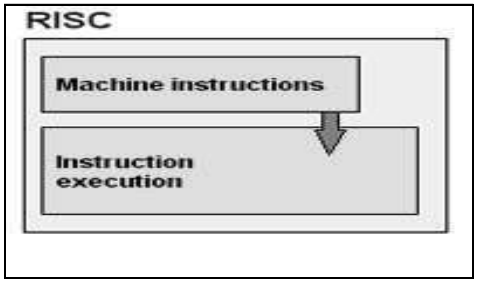


Fig 3.3 RISC architecture

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.

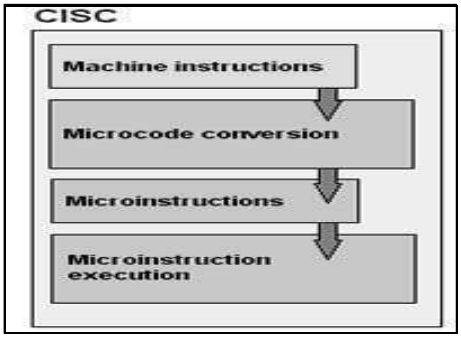


Fig 3.4 CISC architecture

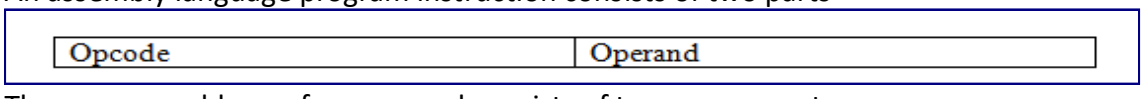
Difference between RISC and CISC

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.

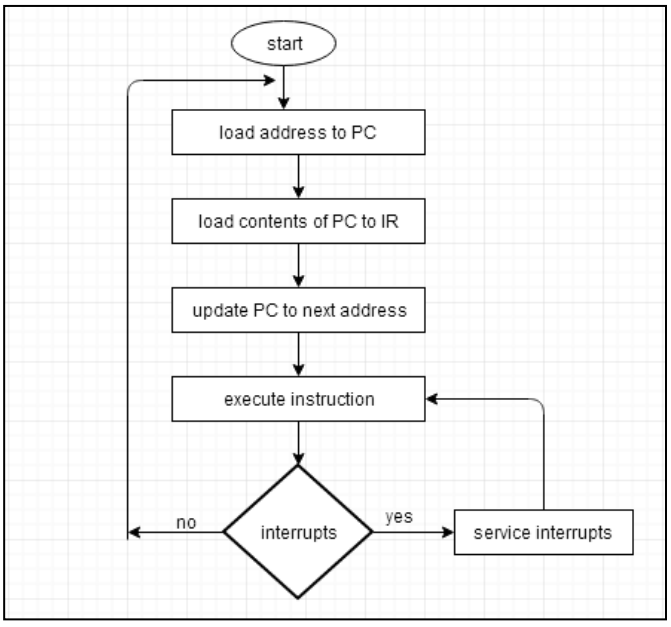


Fig 3.5 Instruction Cycle

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.

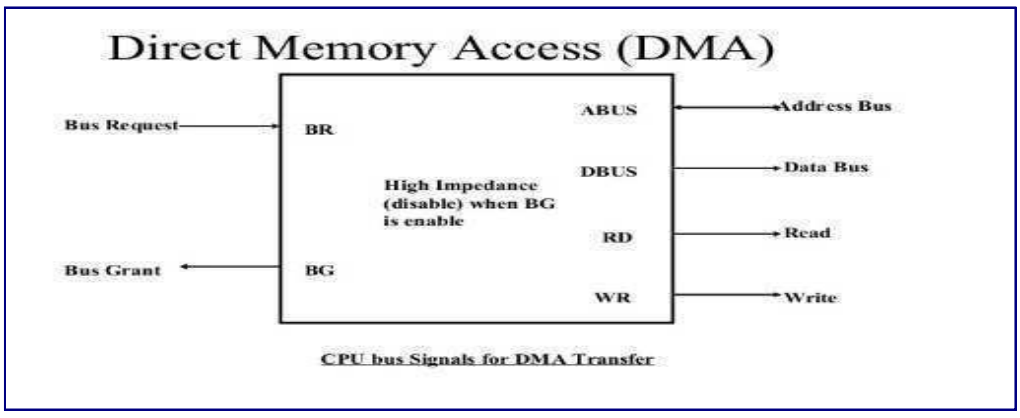


Fig 3.6 DMA

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.

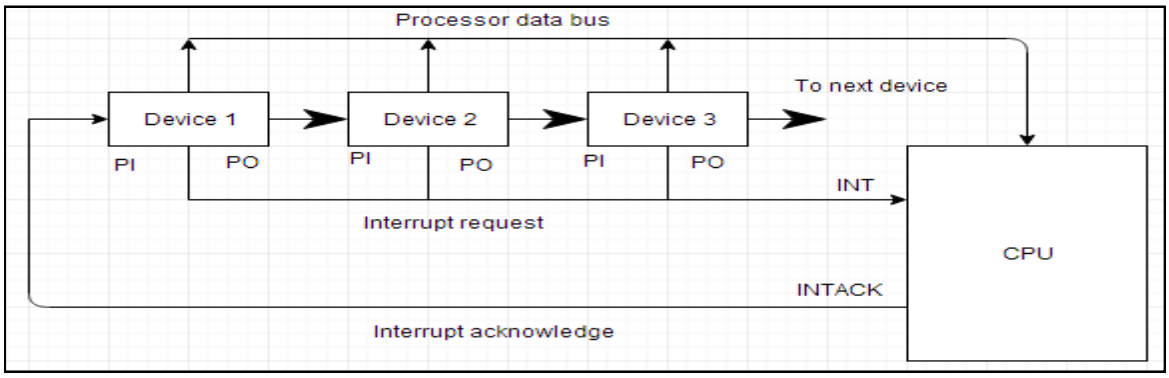


Fig 3.7 **daisy** chaining priority system

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.

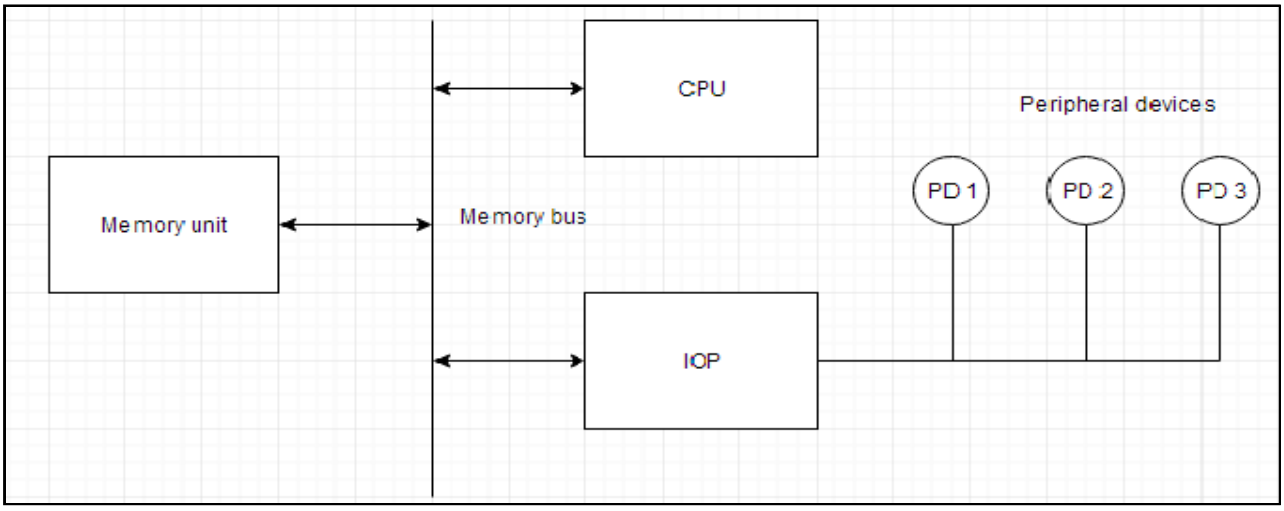


Fig 3.8 Block Diagram Of IOP

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.

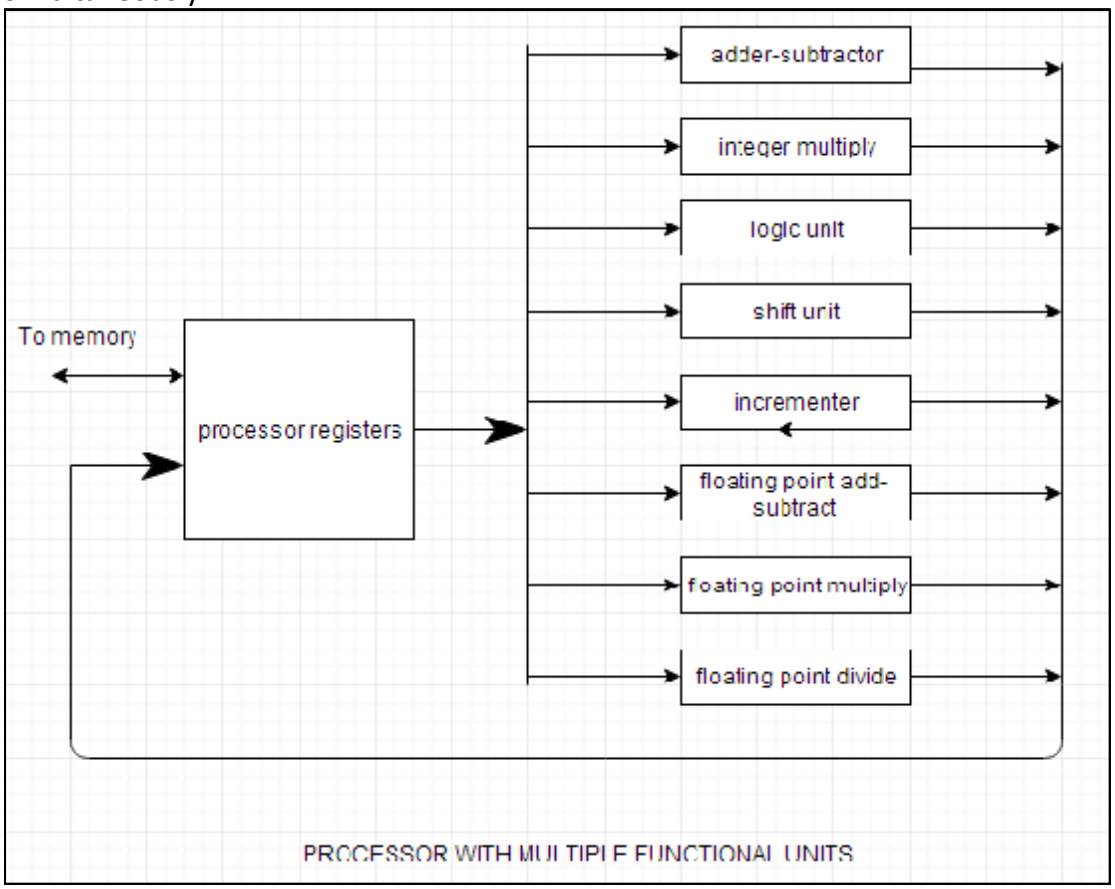


Fig 5.1 processor with multiple functional units

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.

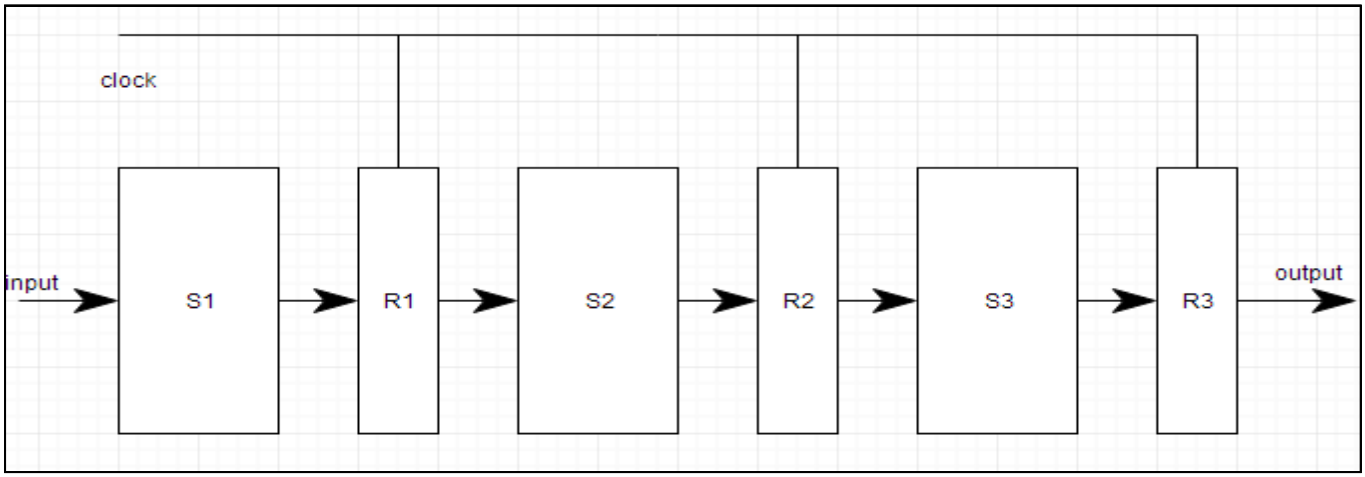


Fig 5.2 pipelining

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

RGPVNOTES.IN

We hope you find these notes useful.

You can get previous year question papers at https://qp.rgpvnotes.in .

If you have any queries or you want to submit your study notes please write us at rgpvnotes.in@gmail.com

LIKE & FOLLOW US ON FACEBOOK
facebook.com/rgpvnotes.in