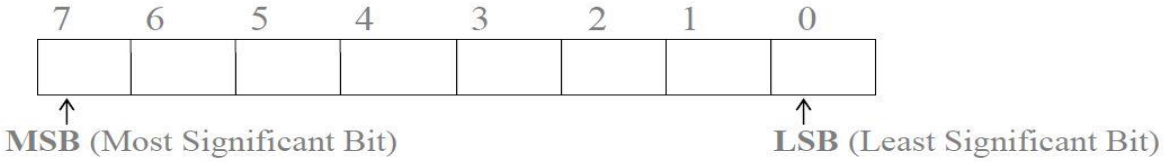
Assembly Language Programming:

Data Representation Basics

- **Bit** – **b**asic **i**nformation uni**t**: (1/0)

- **Byte** – sequence of 8 bits:



- **Word** – a sequence of bits addressed as a **single entity** by the computer

Registers:

General Purpose Register:

AX- accumulator register

BX-base register

CX-counter register

DX-data register

Index register:

SP-stack pointer

BP-base pointer

SI-source index

DI-destination index

Others:

IP/PC-instruction pointer/program counter

Assembly Language Program

- consists of a series of processor instructions, meta-statements, comments, and data
- translated by assembler into machine language instructions (binary code) that can be loaded into memory and executed
- **NASM** - **N**etwide **As**se**m**bler - is assembler and for x86 architecture

Example:

assembly code:

```
MOV AL, 61h      ; load AL with 97 decimal (61 hex)
```

MOV - Move Instruction – copies source to destination

EXAMPLE: MOV AX,BX

Basic Arithmetical Instruction

ADD - add integers

Example:
add AX, BX ;(AX gets a value of AX+BX)

SUB - subtract integers

Example:
sub AX, BX ;(AX gets a value of AX-BX)

ADC - add integers with carry (value of Carry Flag)

Example:
adc AX, BX ;(AX gets a value of AX+BX+CF)

SBB - subtract with borrow (value of Carry Flag)

Example:
sbb AX, BX ;(AX gets a value of AX-BX-CF)

INC - increment integer

Example:
inc AX ;(AX gets a value of AX+1)

DEC - increment integer

Example:
dec byte [buffer] ;([buffer] gets a value of [buffer] -1)

Basic Logical Instructions

NOT – one's complement negation – inverts all the bits

Example:

mov al, 11111110$_b$

not al ;(AL gets a value of 00000001_b)

;($11111110_b + 00000001_b = 11111111_b$)

NEG – two's complement negation – inverts all the bits, and adds 1

Example:

mov al, 11111110$_b$

neg al ;(AL gets a value of not(11111110_b)+1=00000001_b+1=00000010_b)

;($11111110_b + 00000010_b = 100000000_b = 0$)

OR – bitwise or – bit at index i of the destination gets '1' if bit at index i of source or destination are '1'; otherwise '0'

Example:

mov al, 11111100$_b$

mov bl, 00000010$_b$

or AL, BL ;(AL gets a value 11111110_b)

AND– bitwise and – bit at index i of the destination gets '1' if bits at index i of both source and destination are '1'; otherwise '0'

Example:

or AL, BL ;(with same values of AL and BL as in previous example, AL gets a value 11000000)

CMP – Compare Instruction – compares integers

CMP performs a 'mental' subtraction - **affects the flags** as if the subtraction had taken place, but does not store the result of the subtraction.

Examples:

```
mov al, 11111100b
mov bl, 00000010b
cmp al, bl ;(ZF (zero flag) gets a value 0)
```

```
mov al, 11111100b
mov bl, 11111100 b
cmp al, bl ;(ZF (zero flag) gets a value 1)
```

JMP – unconditional jump

jmp label

JMP tells the processor that the next instruction to be executed is located at the label that is given as part of jmp instruction.

Example:

```
mov eax, 1
inc_again:
        inc eax
        jmp inc_again
        mov ebx, eax
```

this is infinite loop !

this instruction is never reached from this code!

J<Condition> – conditional jump

j<cond> label

- execution is transferred to the target instruction only if the specified condition is satisfied
- usually, the condition being tested is the result of the last arithmetic or logic operation

Example:

```
        read_char:
                …               ; get a character into AL
                cmp al, 'a'     ; compare the character to 'a'
                je a_received   ; if value of al register equals to 'a', jump to a_received
                jmp read_char   ; go back to read another
        a_received:
                …
```

D<Size> – declare initialized data

d<size> initial value

Pseudo-instruction	<size> filed	<size> value
DB	byte	1 byte
DW	word	2 bytes
DD	double word	4 bytes
DQ	quadword	8 bytes
DT	tenbyte	10 bytes
DDQ	double quadword	16 bytes
DO	octoword	16 bytes

Interfacing:

Interface is the path for communication between two components. Interfacing is of two types, memory interfacing and I/O interfacing.

Memory Interfacing

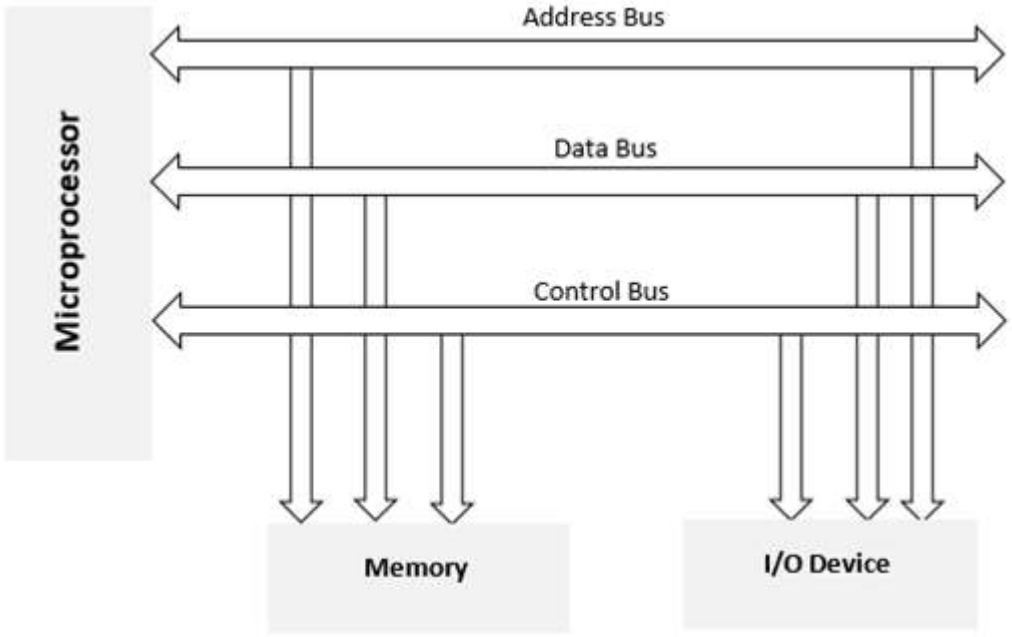
When we are executing any instruction, we need the microprocessor to access the memory for reading instruction codes and the data stored in the memory. For this, both the memory and the microprocessor requires some signals to read from and write to registers.

The interfacing process includes some key factors to match with the memory requirements and microprocessor signals. The interfacing circuit therefore should be designed in such a way that it matches the memory signal requirements with the signals of the microprocessor.

IO Interfacing

There are various communication devices like the keyboard, mouse, printer, etc. So, we need to interface the keyboard and other devices with the microprocessor by using latches and buffers. This type of interfacing is known as I/O interfacing.

Block Diagram of Memory and I/O Interfacing



DMA controller:

What is a DMA Controller?

The term DMA stands for direct memory access. The hardware device used for direct memory access is called the DMA controller. DMA controller is a control unit, part of I/O device's interface circuit, which can transfer blocks of data between I/O devices and main memory with minimal intervention from the processor.

DMA Controller Diagram in Computer Architecture

DMA controller provides an interface between the bus and the input-output devices. Although it transfers data without intervention of processor, it is controlled by the processor. The processor initiates the DMA controller by sending the starting address, Number of words in the data block and direction of transfer of data .i.e. from I/O devices to the memory or from main memory to I/O devices. More than one external device can be connected to the DMA controller.

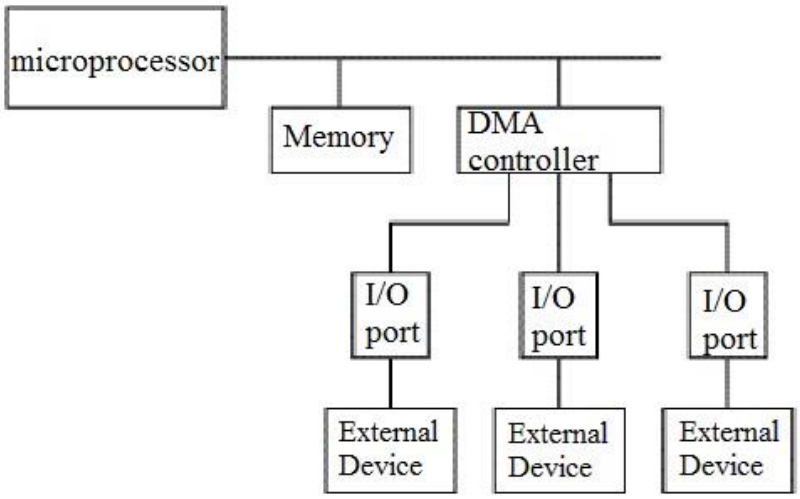
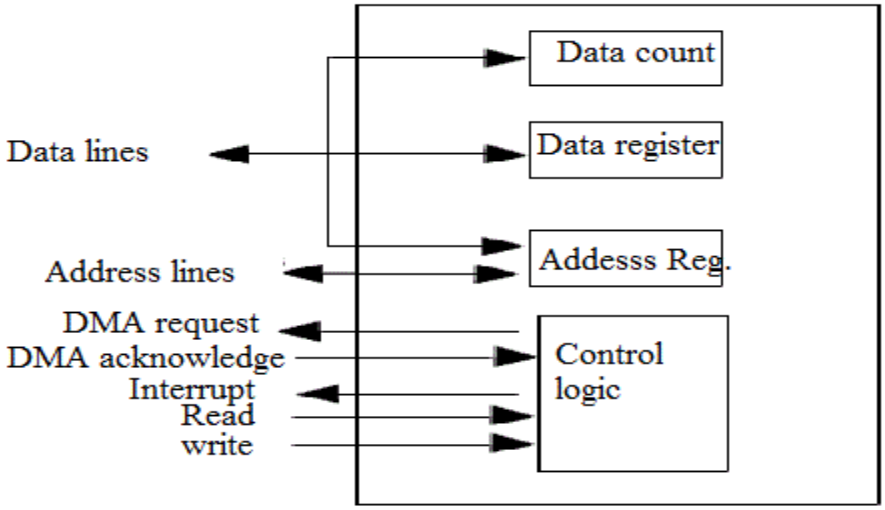


Fig: DMA in Computer Architecture

DMA controller contains an address unit, for generating addresses and selecting I/O device for transfer. It also contains the control unit and data count for keeping counts of the number of blocks transferred and indicating the direction of transfer of data. When the transfer is completed, DMA informs the processor by raising an interrupt.

The typical block diagram of the DMA controller is shown in the figure below.



The DMA transfers the data in three modes which include the following.

a) **Burst Mode**: In this mode DMA handover the buses to CPU only after completion of whole data transfer. Meanwhile, if the CPU requires the bus it has to stay ideal and wait for data transfer.

b) **Cycle Stealing Mode**: In this mode, DMA gives control of buses to CPU after transfer of every byte. It continuously issues a request for bus control, makes the transfer of one byte and returns the bus. By this CPU doesn't have to wait for a long time if it needs a bus for higher priority task.

c) **Transparent Mode:** Here, DMA transfers data only when CPU is executing the instruction which does not require the use of buses.

Keyboard controller:

In computing, a keyboard controller is a device that interfaces a keyboard to a computer. Its main function is to inform the computer when a key is pressed or released. When data from the keyboard arrives, the controller raises an interrupt (a *keyboard interrupt*) to allow the CPU to handle the input.

If a keyboard is a separate peripheral system unit (such as in most modern desktop computers), the keyboard controller is not directly attached to the keys, but receives scancodes from a microcontroller embedded in the keyboard via some kind of serial interface. In this case, the controller usually also controls the keyboard's LEDs by sending data back to keyboard through the wire.

A keyboard buffer is a section of computer memory used to hold keystrokes before they are processed.

Keyboard buffers have long been used in command-line processing. As a user enters a command, they see it echoed on their terminal and can edit it before it is processed by the computer.

In time-sharing systems, the location of the buffer depends on whether communications is full-duplex or half-duplex. In full-duplex systems, keystrokes are transmitted one by one. As the main computer receives each keystroke, it ordinarily appends the character which it represents to the end of the keyboard buffer. The exception is control characters, such as "delete" or "backspace" which correct typing mistakes by deleting the character at the end of the buffer.

In half-duplex systems, keystrokes are echoed locally on a computer terminal. The user can see the command line on his terminal and edit it before it is transmitted to the main computer. Thus the buffer is local.

On some early home computers, to minimize the necessary hardware, a CPU interrupt checked the keyboard's switches for key presses multiple times each second, and recorded the key presses in a keyboard buffer for the operating system or application software to read.

On some systems, by pressing too many keys at once, the keyboard buffer overflows and will emit a beep from the computer's internal speaker.

Interrupts

Data transfer between the CPU and the peripherals is initiated by the CPU. But the CPU cannot start the transfer unless the peripheral is ready to communicate with the CPU. When a device is ready to communicate with the CPU, it generates an interrupt signal. A number of input-output devices are attached to the computer and each device is able to generate an interrupt request.

The main job of the interrupt system is to identify the source of the interrupt. There is also a possibility that several devices will request simultaneously for CPU communication. Then, the interrupt system has to decide which device is to be serviced first.

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:

Following are some different types of interrupts:

Hardware Interrupts

When the signal for the processor is from an external device or hardware then this 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.

Microinstruction Sequencing:

A micro-program control unit can be viewed as consisting of two parts:

1. The control memory that stores the microinstructions.
2. Sequencing circuit that controls the generation of the next address.

A micro-program sequencer attached to a control memory inputs certain bits of the microinstruction, from which it determines the next address for control memory. A typical sequencer provides the following address-sequencing capabilities:

1. Increment the present address for control memory.
2. Branches to an address as specified by the address field of the micro instruction.
3. Branches to a given address if a specified status bit is equal to 1.

4. Transfer control to a new address as specified by an external source (Instruction Register).

5. Has a facility for subroutine calls and returns.

Depending on the current microinstruction condition flags, and the contents of the instruction register, a control memory address must be generated for the next micro instruction.

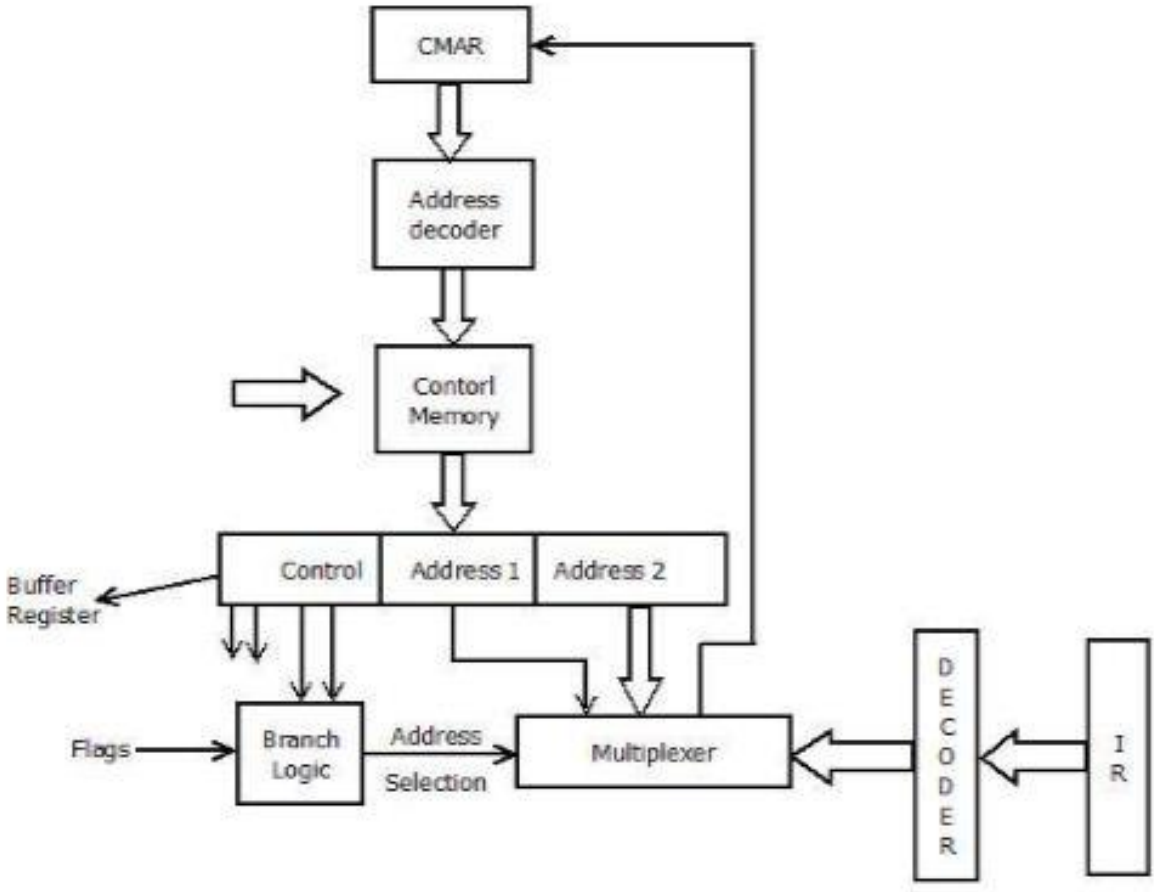
There are three general techniques based on the format of the address information in the microinstruction:

1. Two Address Field.

2. Single Address Field.

3. Variable Format

Two Address Field:



The simplest approach is to provide two address field in each microinstruction and multiplexer is provided to select:

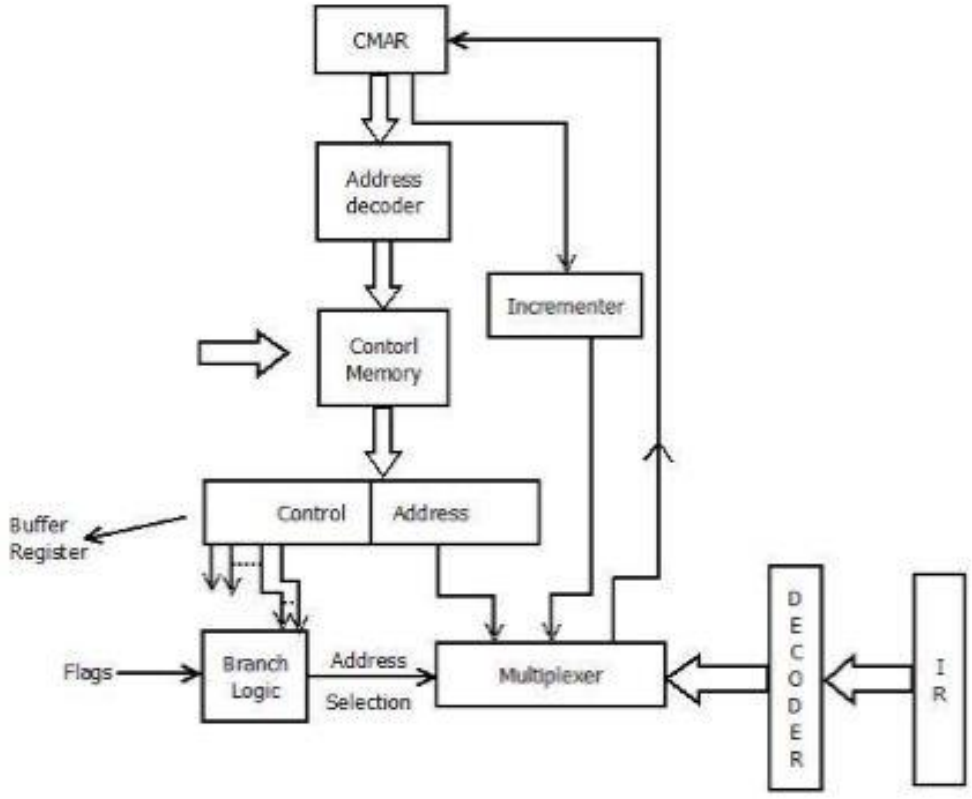
- Address from the second address field.
- Starting address based on the OPcode field in the current instruction.

The address selection signals are provided by a branch logic module whose input consists of control unit flags plus bits from the control partition of the micro instruction.

Single Address Field:

Two-address approach is simple but it requires more bits in the microinstruction. With a simpler approach, we can have a single address field in the micro instruction with the following options for the next address.

- Address Field.
- Based on OPcode in instruction register.
- Next Sequential Address.



The address selection signals determine which option is selected. This approach reduces the number of address field to one. In most cases (in case of sequential execution) the address field will not be used. Thus the microinstruction encoding does not efficiently utilize the entire microinstruction.

Variable Format:

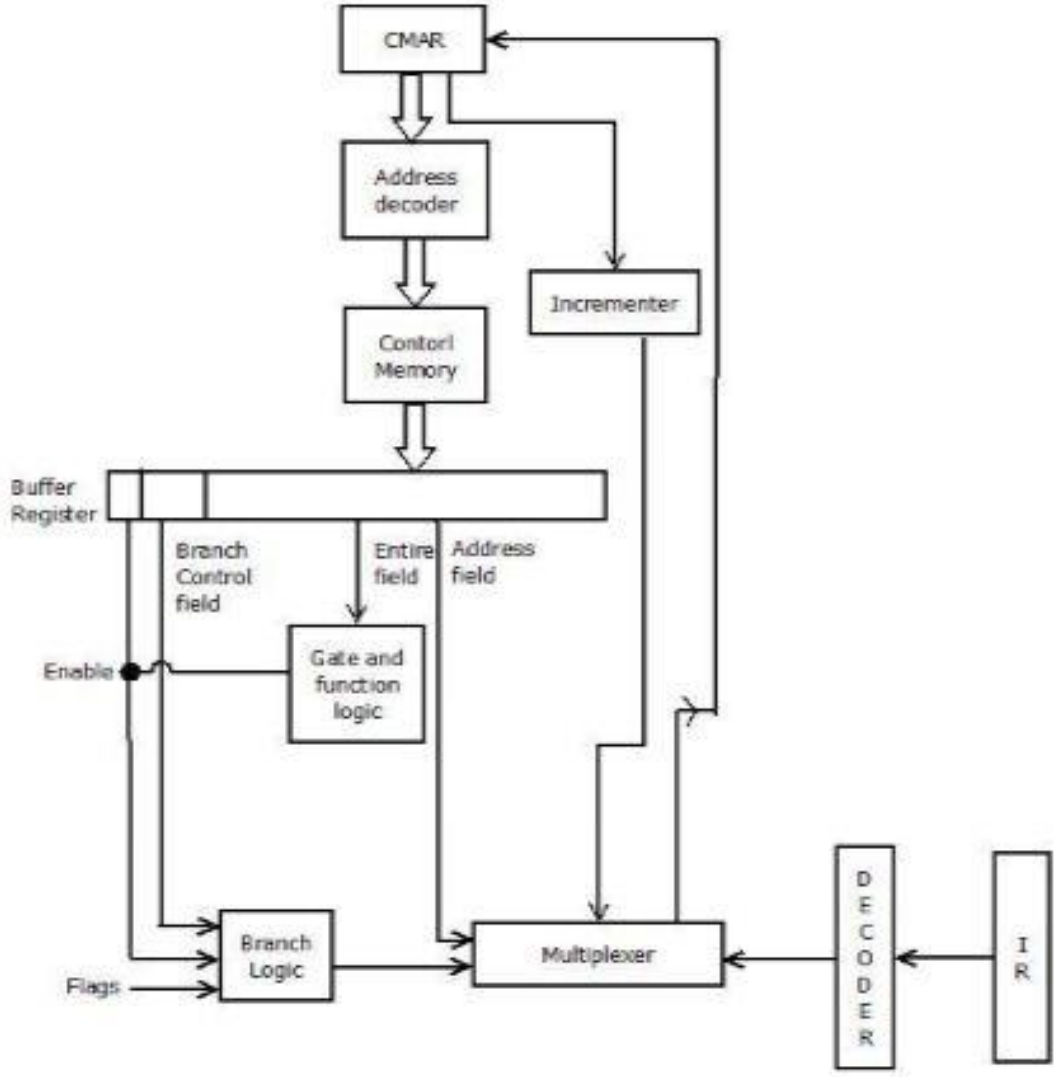


Fig. 3.41: Branch control logic, variable format

In this approach, there are two entirely different microinstruction formats. One bit designates which format is being used. In this first format, the remaining bits are used to activate control signals. In the second format, some bits drive the branch logic module, and the remaining bits provide the address. With the first format, the next address is either the next sequential address or an address derived from the instruction register. With the second format, either a conditional or unconditional branch is specified.

Bit-Slice Processor

A microprocessor designed to allow variable word sizes using 2, 4, or 8 bit processor units on a single chip. These devices can be paralleled to yield an 8, 12, 16, 24, or 32 bit processor when assembled with other components. Occasionally used in printers.

It is a technique for constructing a processor from modules, each of which processes one bit-field or "slice" of an operand. Bit slice processors usually consist of an ALU of 1, 2, 4 or 8 bits and control lines (including carry or overflow signals usually internal to the CPU). For example, two 4-bit ALUs could be arranged side by side, with control lines between them, to form an 8-bit ALU. A sequencer executes a program to provide data and control signals.

Bit-slice microprocessors are the Legos of microprocessors. In bit-slice microprocessors the MPU is split apart into CU and ALU chips. At first blush, this may seem to go against the rationale for using a MPU. Splitting these functions apart into multiple increases complexity and expense. After all the MCU was built to shrink a computer and bit-slice seems to be expanding it. Actually it is and does, but there are some interesting advantages to the bit-slice approach.

The bit-slice microprocessor design has three very significant advantages.

1. ALU's can be attached together in horizontal configurations to create computers that can handle very large chunks of data at a time. An example of a bit-slice processor is the AMD 2901. With the AMD 2901 bit-slice, the 2901 is the ALU, the AMD 2910 is the CU. The 2901 was a peer of the Intel 8080, but the 8080 could only handle 8-bits of data at time. The 2901 was a 4-bit ALU, but 4 2901's could be linked together to create a computer that could handle 16-bits at a time, 8 could be put together to create a 32-bit computer and so on. Whereas the 8080 would have to use multiple cycles to process 16 or 32 bits, the appropriate 2901 configuration could handle it in a single cycle giving such a computer significantly more power than the 8080.

2. Advantage of the Bit-slice design is the fact that the two chip design allowed the chips to use bipolar chip technology (example: the Intel 3002). Bipolar is very fast, but consumes lots of power and dissipates lots of heat. Because of the heat dissipation problem, bipolar chips could not be as dense (in number of transistors per area) as the PMOS or NMOS chips. It was not possible to build single-chip CPU's using bipolar technology. So, in addition to the wider data paths the bit-slice devices could achieve, they were inherently faster due the bipolar technology that was employed to build the chips.

3. The bit-slices had the ability to allow users to create their own instruction sets for their applications. Instruction sets could be created to emulate, or enhance, existing processors such as the 6502 or 8080, or to create a unique instruction set specially adapted to maximize performance of a specific application. The combination of the speed and flexibility of bit-slice devices made the very popular and created a cult-like following, especially for the AMD 2901.

RISC & CISC machines:

Reduced Set Instruction Set Architecture (RISC) – The main idea behind is to make hardware simpler by using an instruction set composed of a few basic steps for loading, evaluating and storing operations just like an addition command will be composed of loading data, evaluating and storing.

Complex Instruction Set Architecture (CISC) – The main idea is to make hardware complex as a single instruction will do all loading, evaluating and storing operations just like a multiplication command will do stuff like loading data, evaluating and storing it.

Both approaches try to increase the CPU performance

- **RISC:** Reduce the cycles per instruction at the cost of the number of instructions per program.
- **CISC:** The CISC approach attempts to minimize the number of instructions per program but at the cost of increase in number of cycles per instruction.

$$CPU\ Time = \frac{Seconds}{Program} = \frac{Instructions}{Program} X \frac{Cycles}{Instructions} X \frac{Seconds}{Cycle}$$

Earlier when programming was done using assembly language, a need was felt to make instruction do more task because programming in assembly was tedious and error prone due to which CISC architecture evolved but with uprise of high level language dependency on assembly reduced RISC architecture prevailed.

Characteristic of RISC –

1. Simpler instruction, hence simple instruction decoding.
2. Instruction come under size of one word.
3. Instruction take single clock cycle to get executed.
4. More number of general purpose register.
5. Simple Addressing Modes.
6. Less Data types.

7. Pipeling can be achieved.

Characteristic of CISC –

1. Complex instruction, hence complex instruction decoding.
2. Instruction are larger than one word size.
3. Instruction may take more than single clock cycle to get executed.
4. Less number of general purpose register as operation get performed in memory itself.
5. Complex Addressing Modes.
6. More Data types.

Example – Suppose we have to add two 8-bit number:

- **CISC approach:** There will be a single command or instruction for this like ADD which will perform the task.
- **RISC approach:** Here programmer will write first load command to load data in registers then it will use suitable operator and then it will store result in desired location.

So, add operation is divided into parts i.e. load, operate, store due to which RISC programs are longer and require more memory to get stored but require less transistors due to less complex command.

Difference –

RISC	CISC
Focus on software	Focus on hardware
Transistors are used for more registers	Transistors are used for storing complex Instructions
Code size is large	Code size is small
A instruction execute in single clock cycle	Instruction take more than one clock cycle
A instruction fit in one word	Instruction are larger than size of one word

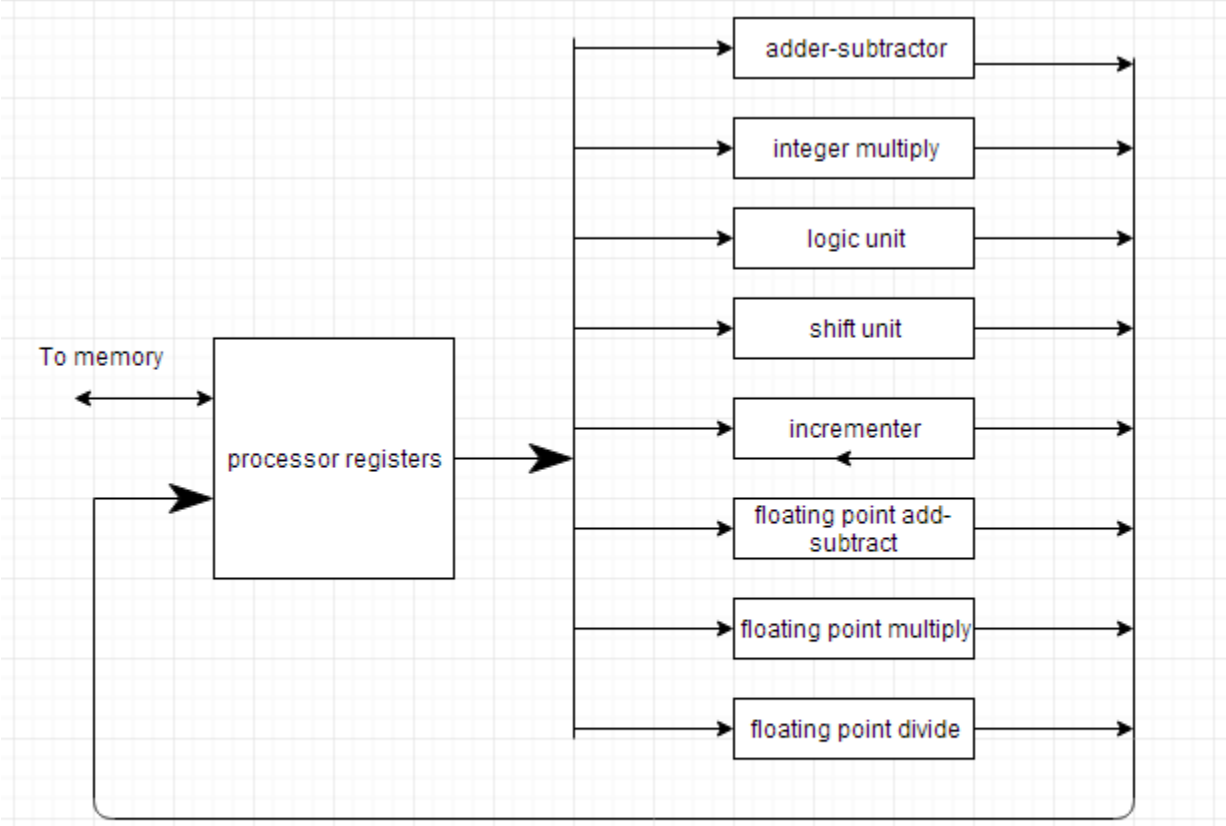
Parallel processing:

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.

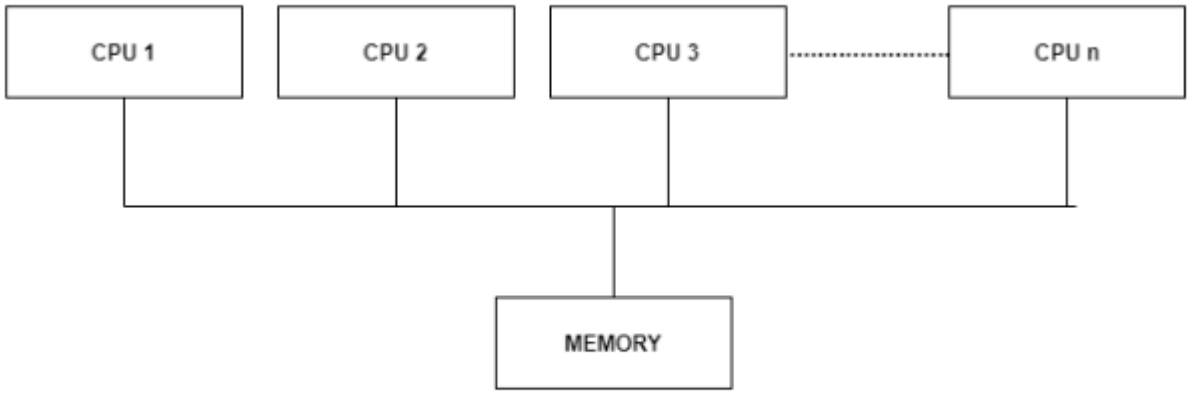
NOTE: Throughput is the number of instructions that can be executed in a unit of time.

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.



PROCESSOR WITH MULTIPLE FUNCTIONAL UNITS

Most computer systems are single processor systems i.e they only have one processor. However, multiprocessor or parallel systems are increasing in importance nowadays. These systems have multiple processors working in parallel that share the computer clock, memory, bus, peripheral devices etc. An image demonstrating the multiprocessor architecture is:>



Multiprocessing Architecture

Types of Multiprocessors

There are mainly two types of multiprocessors i.e. symmetric and asymmetric multiprocessors. Details about them are as follows:

Symmetric Multiprocessors

In these types of systems, each processor contains a similar copy of the operating system and they all communicate with each other. All the processors are in a peer to peer relationship i.e. no master - slave relationship exists between them.

An example of the symmetric multiprocessing system is the Encore version of Unix for the Multimax Computer.

Asymmetric Multiprocessors

In asymmetric systems, each processor is given a predefined task. There is a master processor that gives instruction to all the other processors. Asymmetric multiprocessor system contains a master slave relationship.

Asymmetric multiprocessor was the only type of multiprocessor available before symmetric multiprocessors were created. Now also, this is the cheaper option.

Advantages of Multiprocessor Systems

There are multiple advantages to multiprocessor systems. Some of these are:

More reliable Systems

In a multiprocessor system, even if one processor fails, the system will not halt. This ability to continue working despite hardware failure is known as graceful degradation. For example: If there are 5 processors in a multiprocessor system and one of them fails, then also 4 processors are still working. So the system only becomes slower and does not ground to a halt.

Enhanced Throughput

If multiple processors are working in tandem, then the throughput of the system increases i.e. number of processes getting executed per unit of time increase. If there are N processors then the throughput increases by an amount just under N.

More Economic Systems

Multiprocessor systems are cheaper than single processor systems in the long run because they share the data storage, peripheral devices, power supplies etc. If there are multiple processes that share data, it is better to schedule them on multiprocessor systems with shared data than have different computer systems with multiple copies of the data.

Disadvantages of Multiprocessor Systems

There are some disadvantages as well to multiprocessor systems. Some of these are:

Increased Expense

Even though multiprocessor systems are cheaper in the long run than using multiple computer systems, still they are quite expensive. It is much cheaper to buy a simple single processor system than a multiprocessor system.

Complicated Operating System Required

There are multiple processors in a multiprocessor system that share peripherals, memory etc. So, it is much more complicated to schedule processes and impart resources to processes.than in single processor systems. Hence, a more complex and complicated operating system is required in multiprocessor systems.

Large Main Memory Required

All the processors in the multiprocessor system share the memory. So a much larger pool of memory is required as compared to single processor systems.

Vector Processing:

Characteristics of a Vector

The two main characteristics of a vector are that it helps with fast processing and that the size of the vector does not need to be decided in advance.

Vector Increases Efficiency

If you have the following 5 integers: 20, 25, 30, 35, 40 and you need to add 10 to each of the integers, one method is to declare 5 variables to store each of the 5 integers. So, you have a = 20, b = 25, c = 30 and so on. The computer first finds 20, add 10 to 20; then find 25 and add 10 to 25 and so on.

Using the vector method, instead of having 5 separate variables, you could have one vector variable z. The vector saves all 5 numbers in its array: z[0] = 20; z[1] = 25; z[2] = 30 and so on. The processor considers the entire vector as one variable and instead of having to add 10 five separate times adds 10 at one time to all the elements in the vector, thereby increasing computing efficiency.

Characteristics of Vector Processors

A **vector processor** is a CPU (Central Processing Unit) in a computer with parallel processors and the capability for vector processing. The main characteristic of a vector processor is that it makes use of the parallel processing capability of the processor where two or more processors operate concurrently. This makes it possible for:

- the processors to perform multiple tasks simultaneously

or

- for the task to be split into different subtasks handled by different processors and combined to get the result.

The vector processor considers all of the elements of the vector as one single element as it traverses through the vector in a single loop. Computers with vector processors find many uses that involve computation of massive amounts of data such as image processing, artificial intelligence, mapping the human genome, space simulations, seismic data, and hurricane predictions.

Components of a Vector Processor

Vector Registers File

The vector register holds the different elements that make up the vector. In the z vector, the vector register consists of z[0] through z[5].

Vector Address Generator

This generates the start address and the number of elements in each vector.

Vector Functional Units

These handle arithmetic and logical functions - addition, subtraction, multiplication, division, and square root.

Vector Data Switch

Switch to connect the different components - to connect the processor to the memory modules.

What is fault tolerance?

Fault-tolerance or **graceful degradation** is the property that enables a system (often computer-based) to continue operating properly in the event of the failure of (or one or more faults within) some of its components.

Fault tolerance requirements:

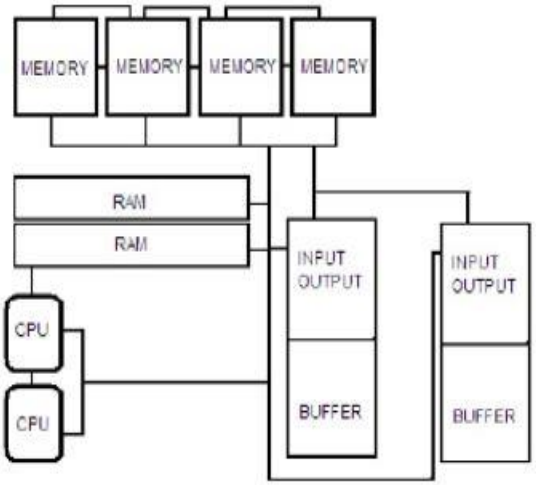
The basic characteristics of fault tolerance require:

- •No single point of failure
- •No single point of repair
- •Fault isolation to the failing component
- •Fault containment to prevent propagation of the failure
- •Availability of reversion modes

Fault-tolerant computer systems:

Fault-tolerant computer systems are systems designed to meet the concepts of fault tolerance. In essence, they have to be able to keep working to a level of satisfaction in the presence of faults.

Basic Block Diagram:



Types of fault tolerance:

Most fault-tolerant computer systems are designed to be able to handle several possible failures, Such as hard disk failures, input or output device failures, or other temporary or permanent failures; software bugs and errors; interface errors between the hardware and software, including driver failures; operator errors, such as erroneous keystrokes, bad command sequences, or installing unexpected software; and physical damage or other flaws introduced to the system from an outside source.

Hardware fault-tolerance is the most common application of these systems, designed to prevent failures due to hardware components. Typically, components have multiple backups and are separated into smaller "segments" that act to contain a fault, and extra redundancy is built into all physical connectors, power supplies, fans, etc. the above figure clearly defines the construction design of fault tolerance system, here all the units having doubled.

.

Evaluation in Fault tolerant Computer systems:

The first known fault-tolerant computer was SAPO, built in 1951 in Czechoslovakia by Antonin Svoboda; its basic design was magnetic drums connected via relays, with a voting method of memory error detection

Most of the development in the so called LLNM (Long Life, No Maintenance) computing was done by NASA during the 1960's, in preparation for Project Apollo and other research aspects.

IBM developed the first computer of this kind for NASA for guidance of Saturn V rockets, but later on BNSF, Unisys, and General Electric built their own.

IBM and Tandem Computers (Now Acquired by HP) built their entire business on such machines, which used single point tolerance to create their **NonStop** systems with uptimes measured in years.

Uses of FTCS:

The fault tolerant system has wide range of applications, now a day each application are considered as Mission Critical application, no company compromise with system down time in their application. To meet business continuity each vendor in market tries to bring every system that they design under Fault – Tolerant design

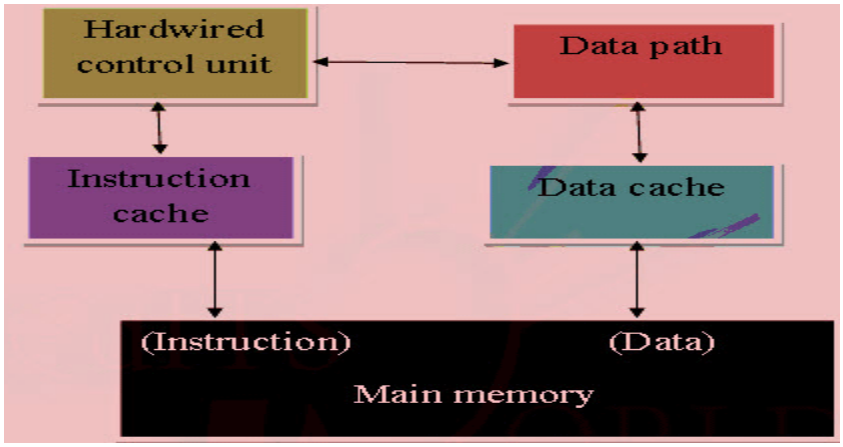
Major industry usage of fault tolerant systems are;

1. Airline industry
2. Research Labs
3. Aerospace Centers
4. BFSI
6. ATM Transaction
7. Telemedicine
8. Military services

Like the list extends according to the applications, these systems ensure the Business continuity and continuous access.

RISC architecture

RISC stands for reduced instruction set computer. It is a type of microprocessor architecture which utilizes a small, highly optimized instructions set rather than a more specialized set of instructions normally found in other types of architecture. Hence each instruction that a computer must perform, requires additional transistors and circuitry, a larger set of computer instructions makes the microprocessormore complicated and slower in operation. The RISC processor has hardwired control unit. Below figure shows the RISC architecture with hardwired control unit with split cache.



RISC architecture

The RISC processor can operate at a higher speed i.e. perform more millions of instructions per second. Register to register operations support only load and store operations to access memory and rest of the operations on a register to register basis. RISC processor has simple and few addressing modes. RISC processor has large number of registers which are needed to support register to register operations minimize the procedure call and return overhead. The RISC processor has fixed-length instructions facilitates efficient instruction execution and easy to pipeline. It easier to develop code with a smaller instruction set for operating system and application. RISCprocessors have a clock per instruction (CPI) of one cycle due to the optimization of each instruction on the CPU.

Use of large Register file in RISC

The register storage is the faster storage device, faster than even the main memory and the cache. Thus, a strategy is needed that will allow the most frequently accessed operands to be kept in registers and to minimize register memory operations.

Two basic approaches are possible, one based on the software and the other based on the hardware. The software approach is to rely on the compiler to maximize register usage. The compiler will attempt to allocate the registers to those variables that will be used most in the

given time period. This approach requires the use of sophisticated program-analysis algorithms. The hardware approach is simply to use more registers so that more variables can be held in registers for longer periods of time. RISC follows the hardware approach.

Since most operand references are to local scalars, the obvious approach is to store these in registers, with perhaps a few registers reserved for global variables. The problem is that the definition of local changes with each procedure call and return, operations that occur frequently. On every call, local variables must be saved from the registers into memory, so that the registers can be reused by the called program. Furthermore, the parameters must be passed. On return, the variables of the parent program must be restored and the results must be passed back to the parent program.

RISC takes care of these with the help of register windows. Multiple small sets of registers are used, each assigned to different procedure,. A procedure call automatically switches the CPU to use a different fixed size window of registers, rather than saving registers in memory. Windows for adjacent procedures are overlapped to allow parameter passing.

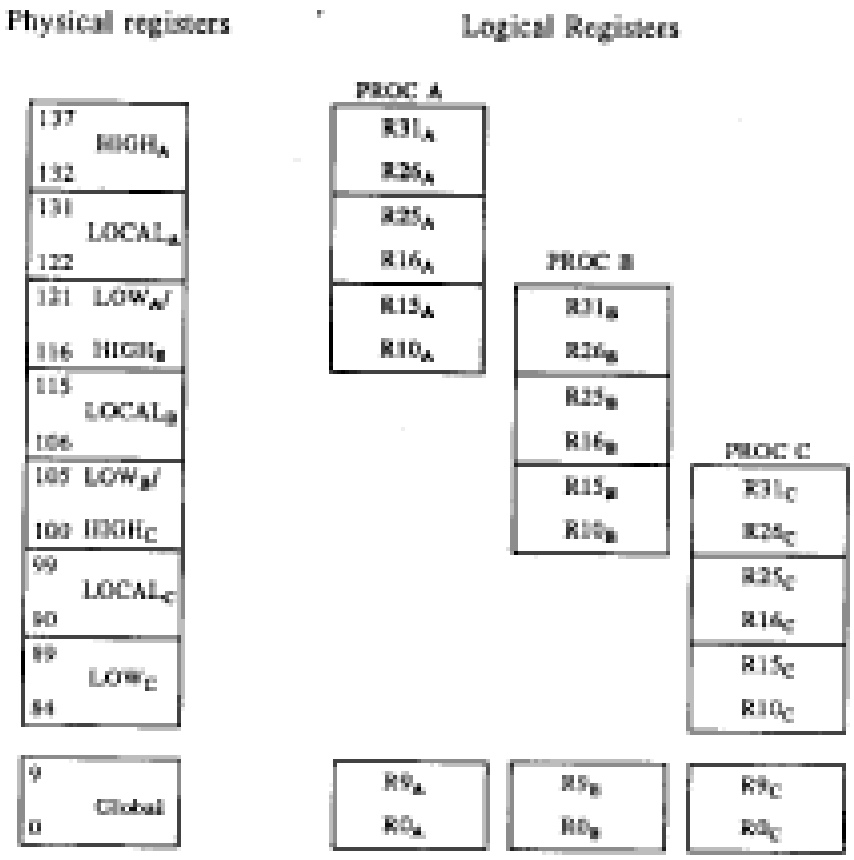


Figure 4: Use of three overlapped register windows

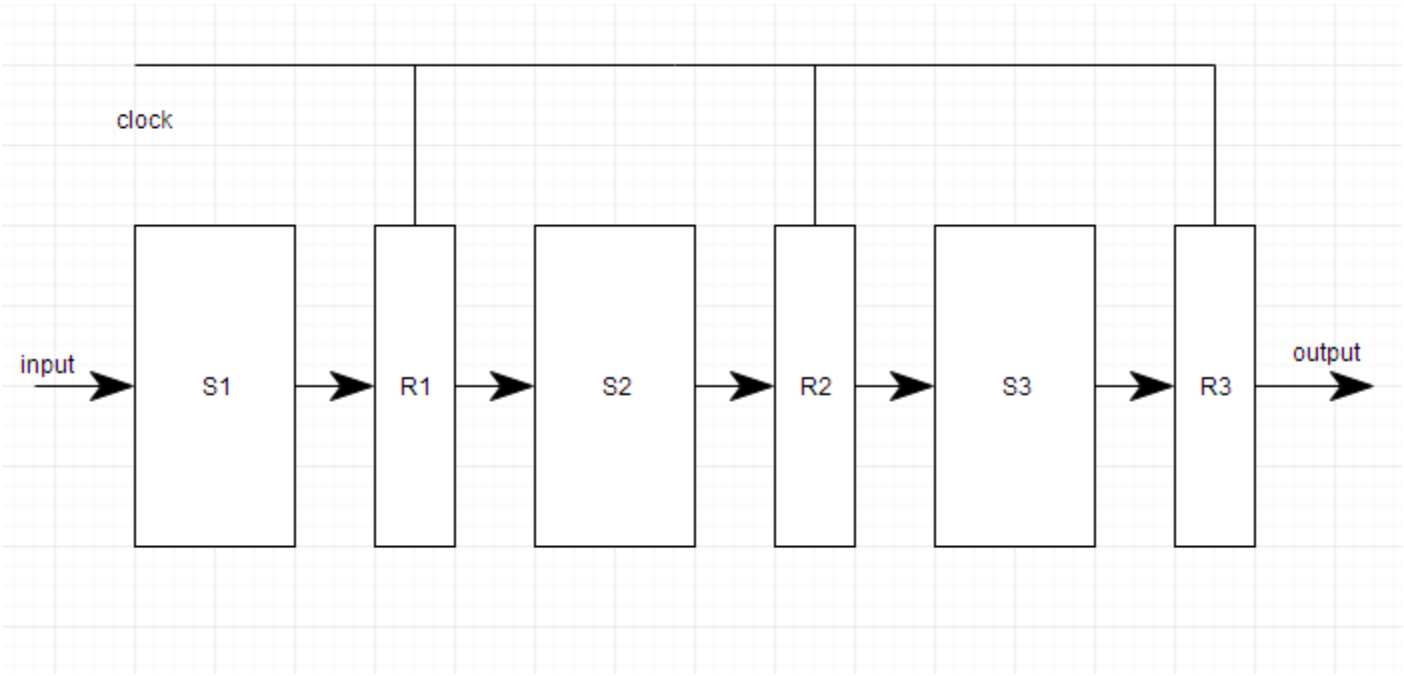
Pipelining:

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.

Types of Pipeline

It is divided into 2 categories:

1. Arithmetic Pipeline

2. Instruction Pipeline

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:

1. 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.

2. 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.

3. 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.

4. Interrupts

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

5. 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.