COMPUTER ARCHITECTURE

Architecture refers to the structure of the processor and how computer components are related to each other.

The processor

- Can be referred to generally as Central Processing Unit (CPU) which is responsible for fetching, decoding and executing of all computer instructions.
- It is commonly called the *brain* of the computer
- Computers cannot work without the processor

The processor has the following functions:

- It controls the transmission of data from input device to memory
- It processes the data and instructions held in main memory
- It controls the transmission of information from main memory to output device.
- Controls the sequence of instructions,
- Give commands to all parts of the computer,
- Fetches the next instruction to be executed
- Decodes instructions
- Executes decoded instructions

Co-Processor

This is an additional processor used for a specific task and improves processing speed by executing jobs concurrently, e. g. maths co-processor

Maths Core-Processor: an additional processor which works alongside the main processor, capable of processing large representations using large size registers, particularly used for floating point calculations

Processor Performance

The traditional processor's performance is affected by these four main components:

(a) Clock Speed

- The processor contains a timing device known as the clock. It determines the timing of all operations.
- This sends out signals at a given interval, and all processes within the computer will start with one of these pulses.
- A process may take any amount of time to complete, but it will only start on a pulse.

- It therefore makes sense that a processor with a faster clock speed will perform faster, since more pulses will be sent out in the same time frame.
- A faster clock increases the speed of the processor and/or memory but not the peripherals
- The clock speed is generally quoted in factors of Hertz, with modern processors typically Gigahertz.

(b) Word Size

- The word size of a computer is the number of bits it can process at a time.
- Increasing the word length of the registers benefits programs with many numerical calculations
- Word length is determined by size of registers
- Bits are grouped into words, the length of a word varies but it is typically, 8, 16, 32, 64 or 128 bits.
- Obviously if the processor is able to deal with more bits at a time then it is going to perform better.
- Typically home computers are 32-bit, but 64-bit technology is becoming widespread too.

(c) Bus Width

- The addresses of data, and the data itself, is transmitted along buses inside the computer.
- The width of the bus is the number of bits it can store / carry.
- A wider data bus will allow more data to be sent at a time, and therefore the processor will perform more efficiently.
- A wider address bus will increase the number of memory addresses a computer may use.
- For example an 8 bit bus will only allow a value between 0 and 255 to be transmitted at a time.

(d) Architecture

- The architecture of a processor will affect its performance,
- A better designed processor will perform better than a different processor.

The main components of the processor are:

- ALU (Arithmetic Logic Unit)
- Control Unit (CU)
- Registers

- System Clock

1. Arithmetic and Logic Unit

Responsible for carrying out operations on data, like calculations. It consists of two parts:

(a) Arithmetic Unit

(b) Logic Unit

(a) Arithmetic Unit

Responsible for basic arithmetic functions such as: Addition, Subtraction, Multiplication, Division, etc

(b) Logical Unit

It perform logical operations like comparing two data items to find which data item is > ,= ,< the other, etc

The ALU works together with the accumulator register, which temporarily stores data being processed and the results of processing.

The ALU performs the following:

- Carries out all arithmetic.
- Carries out logic operations.
- Acts as gateway to and from the processor

2. Control unit

It manages the execution of instructions by running the clock.

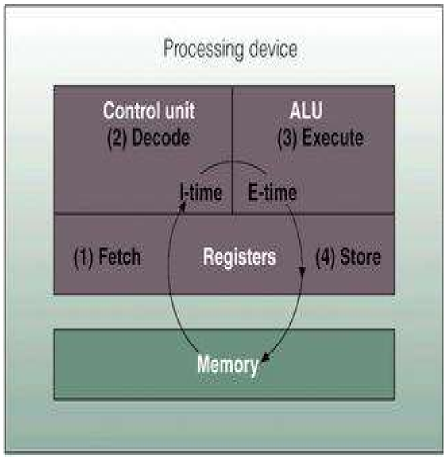
It coordinates and controls all operations of computer system.

It also called the supervisor of the computer. It performs the following:

- Fetches the next instruction to be executed
- Decodes instructions
- Manages execution of instructions
- Executes decoded instructions
- Uses control signals to manage rest of processor.

- It carries out the **Fetch-Execute Cycle as illustrated below:**

The Fetch-Decode-Execute Cycle



The Fetch-Execute Cycle

Step 1. Fetch instruction: In the instruction phase, the computer's control unit fetches the next instruction to be executed from main memory. Microprocessor gets software instruction telling it what to do with data.

Step 2. Decode instruction: Then the instruction is decoded so that the central processor can understand what is to be done. Microprocessor determines what the instructions mean. At this stage, the computer produces signals which control other computer components like the ALU.

Step 3. Execute the instruction: In the execution phase, the ALU does what it is instructed to do, making either an arithmetic computation or a logical comparison. Microprocessor performs the instruction (cause instruction to be executed).

Step 4. Store results: Then the results are stored in the registers or in memory.

Step 3 & 4 are called the execution phase. The time it takes to complete the execution phase is called the EXECUTION TIME (E-time).

After both phases have been completed for one instruction, they are again performed for the second instruction, and so on.

(c) Registers:

- This is a **high-speed storage** area **in the CPU** used to **temporarily** hold small units of program instructions and data immediately before, during and after execution by the CPU.
- It is a small amount of storage available on the CPU whose contents can be accessed more quickly than storage available elsewhere
- Registers are special memory cells that operate at very high speed. They provide the fastest way for a CPU to access data.
- The CPU contains a number of registers and each has a predefined functions

- Most modern computer architectures operate by moving data from main memory into registers, operate on them, then move the result back into main memory
- Register size determines how much information it can store
- The size of register is in bytes: i.e., can be one, two, four or eight byte register
- The processor contains a number of special purpose registers (which have dedicated uses) and general purpose registers (which may be used for arithmetic function and are a sort of "working area")
- The **main types** registers (special purpose registers) found in the Von Neumann Machine are as given below:
  - ✓ program counter
  - ✓ memory address register
  - ✓ memory data register/memory buffer register
  - ✓ current instruction register
  - ✓ index register

Program Counter (PC)

- Contains the address of the next instruction to be fetched/executed
- PC holds address of next instruction
- this register is automatically incremented so that it always holds the memory address of the next instruction
- Keeps check of whereabouts the next program is in the memory.
- After one instruction has been carried out, the PC will be able to tell the processor whereabouts the next instruction is.
- The PC is also called the Sequence Control Register (SCR) as it controls the sequence in which instructions are executed.
- During program execution, the PC:
  - stores the address of the next instruction to be executed
  - Its content is incremented after the address is read
  - Its content is altered to specific address if instruction is a jump instruction

Memory address register (MAR)

- Is used to hold the memory address that contains either the next piece of data or an instruction that is to be used
- It holds the address of a memory location from which data will be read from or written to. Data might be a variable as part of a program, or an instruction for the processor to execute
- Specifies the address for the next read or write
- MAR holds address of instruction/data
- This is where the address that was read from the PC is sent.
- Stored here so that the processor knows where-abouts in the memory the instruction is.

Memory Data Register (MDR) also called Memory buffer register (MBR)

- This is used to store data which has been read from or is ready to write to memory. All transfers from memory to CPU go through this buffer
- It acts like a buffer and holds anything that is copied from the memory ready for the processor to use it
- It contains data written into memory or receives data read from memory

Current instruction register (CIR)

- it holds the instruction that is to be executed
- Contains both the operator and operand of the current instruction
- For example the instruction ***MOVE 100,#13***
- "MOVE" is the operator, and "100" and "#13" are the operands.
- It stores an instruction while it is being decoded/executed/carried out
- Its contents change when an instruction from memory has been placed in MDR, and then it is copied from MDR to CIR.

The Status Register (SR).

- This contains status bits which reflect on the results of an instruction. For example, if there is an error in the operation (such as an overflow) this will be recorded in the status register. The Status Registers also store data about interrupts

Instruction Register (IR)

- Contains the instruction most recently fetched or the instruction currently being executed

Index register

- It is a register used for modifying operand addresses during program execution,
- Used in performing vector/array operations.
- Used for indirect addressing where an immediate constant (i.e. which is part of the instruction itself) is added to the contents of the index register to form the address to the actual operand or data

How and why is the index register (IR) used

- used in indexed addressing
- stores a number used to modify an address which is given in an instruction
- allows efficient access to a range of memory locations by incrementing the value in the IR e.g. used to access an array
- It stores an integer value
- The integer value is added to the base address in the instruction
- Used for the successive reading of values from memory locations e.g. in an array
- Can be incremented after use

General Purpose Registers

Accumulator

- A general purpose register used to accumulate results of processing
- it is where the results from other operations are stored temporarily before being used by other processes.
- Available to the programmer and referenced in assembly language programs
- Used for performing arithmetic functions

Flags Register: Used to record the effect of the last ALU operation

HOW REGISTERS OPERATE

Fetch phase:

- The address of next instruction is copied from the PC to the MAR.
- The instruction held at that address is copied to the MDR
- The contents of the MDR are copied to the CIR

Execute Phase:

- The instruction held in the CIR is decoded.
- The instruction is executed.

The Fetch-Decode Execute Cycle

There are three stages of the machine cycle in a Von Neumann architecture, which are: fetch, decode and execute stages.

The stages are summarised as follows:

Fetch

- The PC stores the address of the next instruction which needs to be carried out
  As instructions are held sequentially in the memory, the value in the PC is incremented so that it always points to the next instruction.
- When the next instruction is needed, its address is copied from the PC and placed in the MAR
- The data which is stored at the address in the MAR is then copied to the MDR
- Once it is ready to be executed, the executable part if the instruction is copied into the CIR

Decode

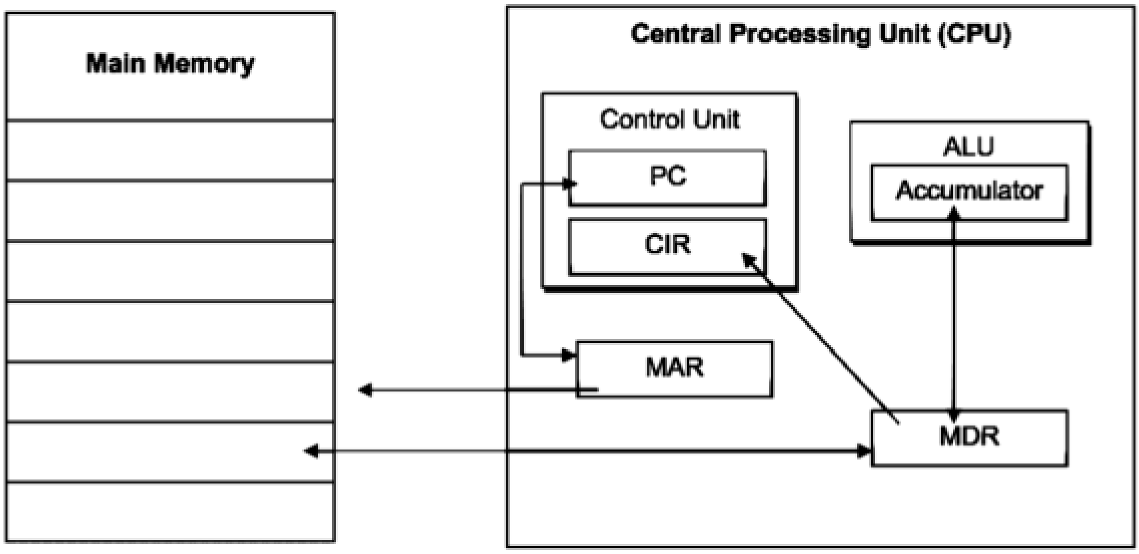
- The instruction in the CIR can now be split into two parts, the address and the operation
- The address part can be placed in the MDR and the data fetched and put in the MAR.

Execute

- The contents of both the memory address register and the memory data register are sent together to the central processor. The central processor contains all the parts that do the calculations, the main part being the CU (control unit) and the

ALU (arithmetic logic unit), there are more parts to the central processor which have specific purposes as well.

- The ALU will keep referring back to where the data and instructions are stored, while it is executing them, the MDR acts like a buffer, storing the data until it is needed
- The CU will then follow the instructions, which will tell it where to fetch the data from, it will read the data and send the necessary signals to other parts of the computer.



The fetch-decode-execute cycle operates in the following way:

- Load the address that is in the program counter (PC) into the memory address register (MAR).
- Increment the PC by 1.
- Load the instruction that is in the memory address given by the MAR into the MDR
- Load the instruction that is now in the MDR into the current instruction register (CIR).
- Decode the instruction that is in the CIR.
- If the instruction is a jump instruction then
    - Load the address part of the instruction into the PC
    - Reset by going to step 1.
- Execute the instruction.
- Reset by going to step 1.

How a jump instruction executed

- by changing contents of PC (to address part of
- instruction)
- copy address part of instruction in CIR to PC

The first step simply places the address of the next instruction into the memory Address Register so that the control unit can fetch the instruction from the correct part of the memory. The program counter is then incremented by 1 so that it contains the address of the next instruction, assuming that the instructions are in consecutive locations.

The memory data register is used whenever anything is to go from the central processing unit to main memory, or vice versa. Thus the next instruction is copied from memory into the MDR and is then copied into the current instruction register.

Now that the instruction has been fetched the control unit can decode it and decide what has to be done. This is the execute part of the cycle. If it is an arithmetic instruction, this can be executed and the cycle restarted as the PC contains the address of the next instruction in order. However, if the instruction involves jumping to an instruction that is not the next one in order, the PC has to be loaded with the address of the instruction that is to be executed next. This address is in the address part of the current instruction, hence the address part is loaded into the PC before the cycle is reset and starts all over again.

<u>Memory Unit</u>

Is the computer memory that temporarily stores the operating system, application programs and data currently use.
It used to store the following:

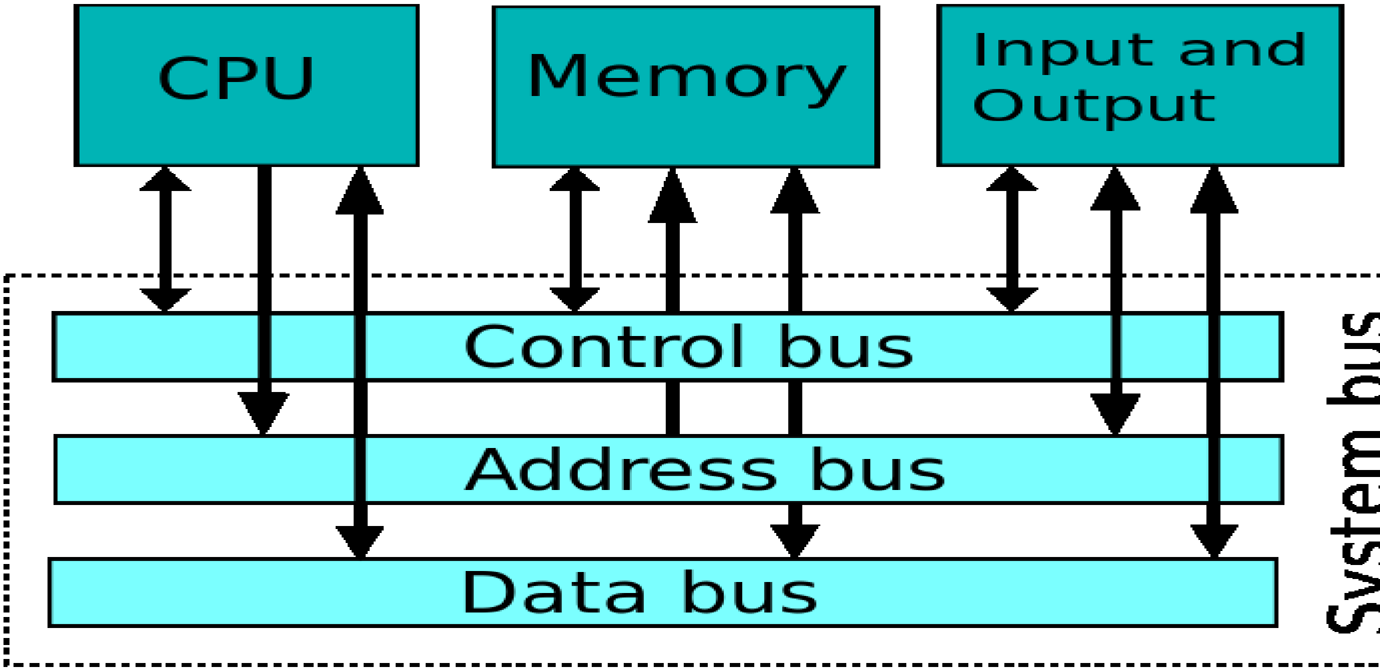
- Program instructions in current use;
- Data in current use;
- Parts of Operating System that are currently in use.

Some architectures have a Memory Unit (Main memory) which has two types: RAM and ROM.

Buses

- A bus is a pathway through which data and signals are transferred from one device to another.
- They are a set of parallel wires connecting two or more components of the computer.
- Buses can be internal or external.
- Buses can be generally referred to as **system** bus and this connect the CPU, memory and I/O devices.
- Each bus is a shared transmission medium, so that only one device can transmit along a bus at any one time.
- Multiple devices can be connected to the same bus

Diagram



- Data and control signals travel in both directions between the processor, memory and I/O controllers.
- Address, on the other hand, travel only one way along the address bus: the processor sends the address of an instruction, or of data to be stored or retrieved, to memory to an I/O controller.

The main types of buses are:

- **Data bus**:
  - Used for carrying data from memory to the processor and between I/O ports.
  - Comprises of either 8, 16, 32 or 64 separate parallel lines
  - Provide a bi-directional path for data and instruction's between computer components. This means that the CPU can read data from memory and input ports and also send data to memory and output ports.
  - The width of the bus determines the overall system performance. For example, if the data bus is 8 bits wide, and each instruction is 16 bits long, then the processor must access the main memory twice during each instruction cycle

- **Address bus**:
  - Used for transferring memory addresses from the processor when it is accessing main memory
  - They are used to access memory during the read or write process
  - The width of the address bus determines the maximum possible memory capacity of the computer.
  - This a uni-directional bus (one way). The address is send from CPU to memory and I/O ports only.

- **Control bus**:
  - The purpose of the control bus is to transmit command, timing and specific status information between system components. Timing signals indicate the validity of data and address information. Command signals specify operations to be performed. Specific status signals indicate the state of a data transfer request, or the status of request by a components to gain control of the system bus
  - This is a bi-directional bus used for carrying control signals (Signals can be transferred in both directions).
  - They carry signals to enable outputs of addressed port and memory devices
  - Control signals regulate activities on the bus.
  - Control buses transmit command, timing and status information between computer components.
  - Typical control signals are:
    - ✓ Memory Read

- ✓ Memory Write
- ✓ I/O Read
- ✓ I/O Write
- ✓ Interrupt Request
- ✓ Interrupt Grant
- ✓ Reset
- ✓ Ready hold
- ✓ etc

- **Timing signals**: indicate validity of data and information.
- **Command signals**: Specify operations to be performed
- **Status signals**: Indicate state of data transfer request or status of a request.

INTERRUPTS

Interrupt: it is a signal generated by a device or software, which may cause a break in the execution of the current routine

An interrupt is a signal send to the processor by a peripheral or software for attention to be turned to that peripheral/software, thereby causing a break in the execution of a program, e.g. printer out of paper.

Control is transferred to another routine and the original routine will be resumed after the interrupt

Interrupt Service Routine (Handler): a small subprogram that is calld when an interrupt occurs and it handles the interrupt.

Interrupt priorities

Interrupts have different priorities

This is important if two interrupts are received simultaneously the processor for it to decide which one is more important to execute first.

There are four levels of priority, which are (highest priority order):

- Hardware Failure: can be caused by power failure or memory parity error.
- Program Interrupts: Arithmetic overflow, division by zero, etc
- Timer Interrupts: generated by the internal clock
- I/O Interrupts:

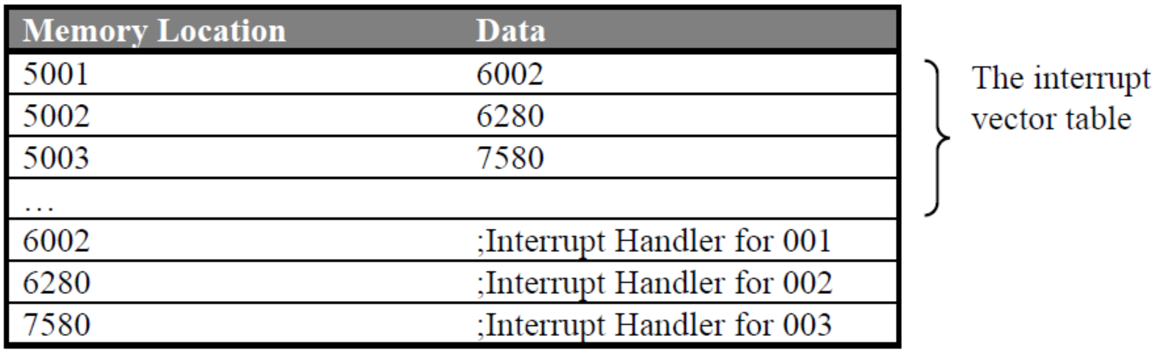
Interrupt Handling

At the end of each Fetch-Execute cycle, the contents of the interrupt registers are checked.

Should there be an interrupt; the following steps will typically be taken:

a) The current fetch-decode-execute cycle is completed
b) The operating system halts current task
c) The contents of the PC and other registers will be stored safely in a stack.
d) The highest priority interrupt is identified. Interrupts with a lower priority are disabled.
e) The source of the interrupt is identified.
f) The start address of the interrupt handler is loaded into the PC.
g) The interrupt handler is executed.
h) Interrupts are enabled again, and the cycle will restart with any further interrupts.
i) The PC and other registers are “popped” from the stack and restored.
j) The user’s program resumes with the next step in its cycle.

- When dealing with an interrupt, the computer has to know which interrupt handler to call for which interrupt.
- One method of doing this is known as the vectored interrupt mechanism.
- In this approach a complete list of interrupts and the memory address of their handler is stored in a table called the interrupt vector table.
- The interrupt supplies an offset number, which identifies the interrupt uniquely.
- This offset is added to a base term, and the resultant number is the memory address of a *pointer* to the memory location of the handler routine.
- This is explained in the example below:



Memory Location	Data
5001	6002
5002	6280
5003	7580
…	
6002	;Interrupt Handler for 001
6280	;Interrupt Handler for 002
7580	;Interrupt Handler for 003

The interrupt vector table

- If the interrupt 002 is received, the base number 5000 is added to it, which allows the processor to know that the handler can be found by opening the data stored at address 5002.
- The address 5002 simply stores a pointer to another memory location, 6280, where the actual handler routine begins.
- The advantage of this approach is that each interrupt only needs to give the processor an offset number, such as 002, and the processor can determine from that the correct memory location to use. This is more efficient than the interrupt sending the full memory address itself. This approach also allows the interrupt routines to be stored anywhere in the memory, with the pointer table updated to reflect if a handler routine is moved

Types of interrupts

Input / output interrupt e.g. disk full, printer out of paper, etc. they are generated by the I/O devices

Interrupts generated by running process: process may need more storage or to communicate with the operator

Timer interrupts: generated by the processor clock, e.g. control being transferred to another user in a time sharing system

Program check interrupts: caused by errors like division by zero

Machine check interrupts: Caused by malfunctioning hardware**.**

Clock (happens normally in time sharing systems where the clock transfers control from one computer to another.)

Why interrupts are used in a computer system

- to obtain processor time for a higher priority task
- to avoid delays
- to avoid loss of data
- as an indicator to the processor that a device needs to be serviced
- allows computer to shut down if the power off interrupt predicts loss of power, saving data in time

Sources of interrupts

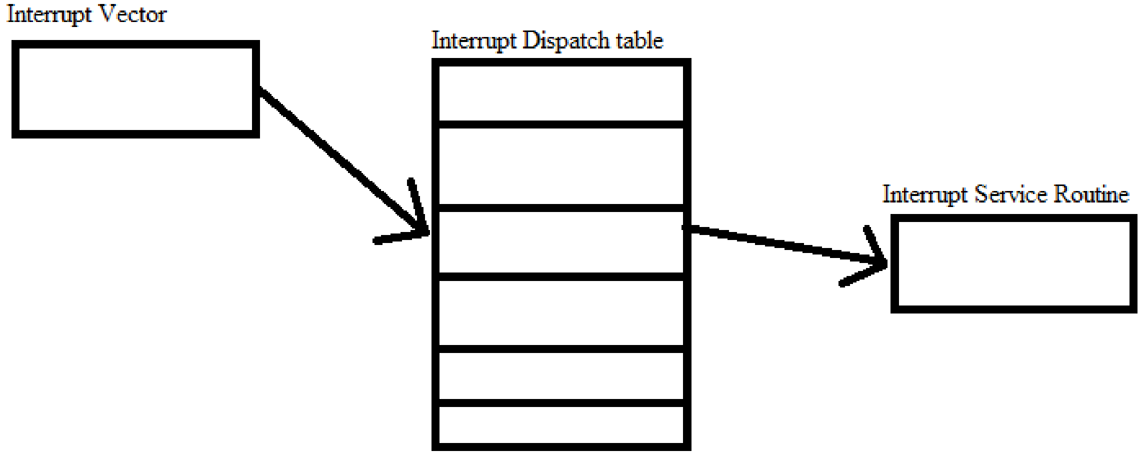
- power failure/system failure
- peripheral e.g. printer (buffer empty)/hardware
- clock interrupt
- user interrupt e.g. new user log on request
- software

Vectored Interrupts

A specific number assigned to each interrupt is called an interrupt vector. Each interrupt is numbered.

Each interrupt vector is the one used to call the interrupt handler

Address of interrupt service routines are stored in an array (known as interrupt dispatch table) and the interrupt vector is used as a subscript to this array.



<u>Buffer</u>

- **Buffer:** This is a temporary memory store for data awaiting processing or output, compensating speed at which devices operate, for example printer buffer.
- A buffer is a memory in the interface between two devices which temporarily store data which is being transmitted from one device to another
- A buffer is a small amount of fast memory outside the processor that allows the processor to get on with other work instead of being held up by the secondary device.
- The buffer is necessary if the two devices work at the different speed
- Buffering is appropriate where an output device processes data slower than the processor. For example, the processor sends data to the printer, which prints much slower and the printer does not need to wait for the printer to finish printing in order for it to carry out the next task.
- It therefore saves the data in a buffer where it will be retrieved by the printer.
- Buffering usually match devices that work at different speeds, e.g. processor and disk.

- Sometimes a device is already busy in executing some instructions.
- **Example**: there are three printing jobs, the printer can print only one job at a time. The OS sends the next two jobs in buffer, a process which is also known as Spooling
- Buffers are a main component of Memory
- The printer buffer is one of the most common type of buffer.

Reasons for using printer buffers:

Stores data or information being sent to the printer temporarily.
Compensates for difference in speed of CPU and printer.
Allows CPU to carry out other tasks whilst printer is printing.

Benefits of increasing size of buffer in a printer:

Reduces the number of data transfers to the printer.
Ensures a more efficient use of the CPU.
Larger files can be sent to the printer without problems

Use of buffers and interrupts in the transfer of data between primary memory and hard disk.

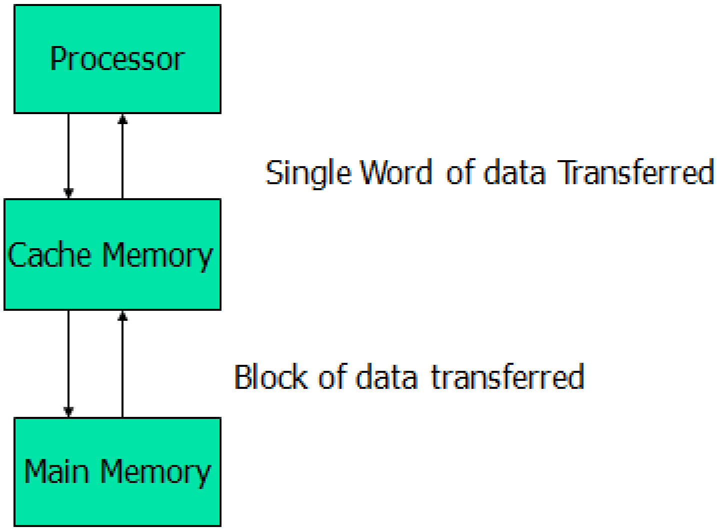
- Buffer is temporary storage area for data
- Data transferred from primary memory to buffer (or vice versa)
- When buffer full, processor can carry on with other tasks
- Buffer is emptied to the hard disk
- When buffer empty, interrupt sent to processor requesting more data to be sent to buffer.
- Works according to priorities

<u>Cache Memory</u>

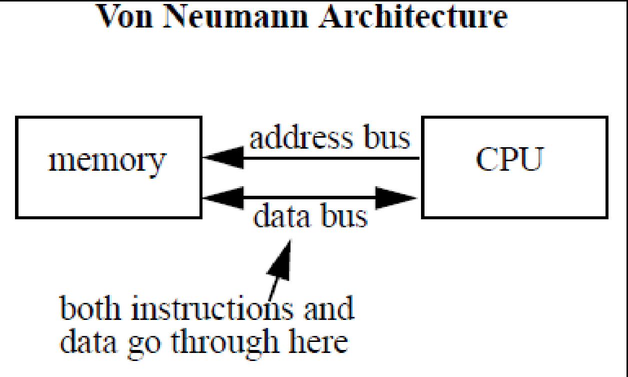
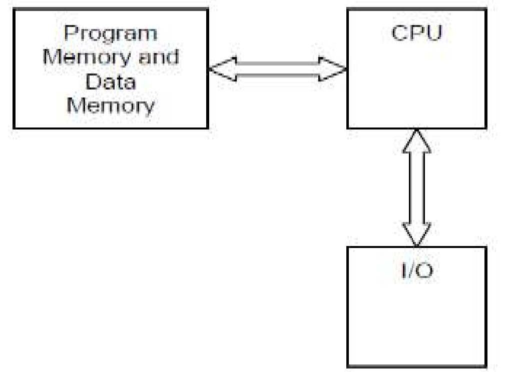
- A cache is a small and very high speed memory used to speed up the transfer of data and instructions, doubling the speed of the computer in some cases.
- It can located inside or close to the CPU Chip
- it is placed between the CPU and the main memory.
- It stores frequently or most recently used instructions and data
- It is faster than RAM
- The data and instructions that are most recently or most frequently used by CPU are stored in cache memory.

- it is used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate
- CPU processes data faster than main memory access time, thus processing speed is limited primarily by the speed of main memory.
- It compensates the speed difference between the main memory access time and processor logic.
- It is used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate.
-
- The cache thus used for storing segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations
-
- The amount of cache memory is generally between 1kb and 512kb

How cache memory works

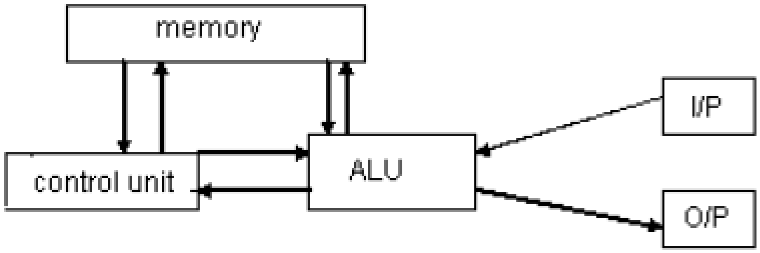


Von Neumann Architecture

Diagram A		**Diagram B**
 Von Neumann Architecture memory address bus CPU data bus both instructions and data go through here		

The Von Neumann Computer architecture has the following characteristics

- Stores both data and instructions in the **same** memory.
- Has a single processor which follows a **linear sequence** of the fetch-decode-execute cycle. Therefore it only processes one **job at a time with one set of data**. Execution occurs in a sequential fashion from one instruction to the next, unless explicitly modified.
- Uses serial processing of instructions. Allows for **one instruction to be rea**d from memory or data to be read/written from/to memory at a time.
- Instructions and data are stored in the same memory and **share a communication pathway or bus** to the CPU. Because program memory and data memory cannot be accessed at the same time, throughput is much smaller than the rate at which the CPU can work. This constraint (problem) is called the Von Neumann **bottleneck** and directly impacts the performance of the system. This can however be solved by use of cache memory.
- The von Neumann architecture allows instructions and data to be mixed and stored in the same memory module and the contents of this memory are addressable by location only. The execution occurs in a sequential fashion.
- Von Neumann architectures usually have a single unified cache
- The Von Neumann machine had five basic parts:- (i) Memory (ii) ALU (iii) control unit (iv) Input equipment & (v) output equipment: as shown below



- Processor needs two clock cycles to complete an instruction.

- In the first clock cycle the processor gets the instruction from memory and decodes it. In the next clock cycle the required data is taken from memory. For each instruction this cycle repeats and hence needs two cycles to complete an instruction
- Pipelining the instructions is not possible with this architecture.
- A **stored-program** digital computer is one that keeps its programmed instructions, as well as its data, in read-write, random access memory (RAM), that is the Von Neumann computer. This makes the machines much more flexible.
- By treating those instructions in the same way as data, a stored-program machine can easily change the program, and can do so under program control.
- Once in the computer's memory a program will be executed one instruction at a time by repeatedly going through
- In the vast majority of modern computers, the same memory is used for both data and program instructions.

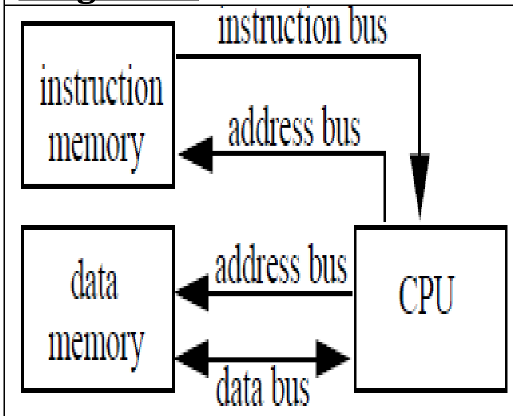
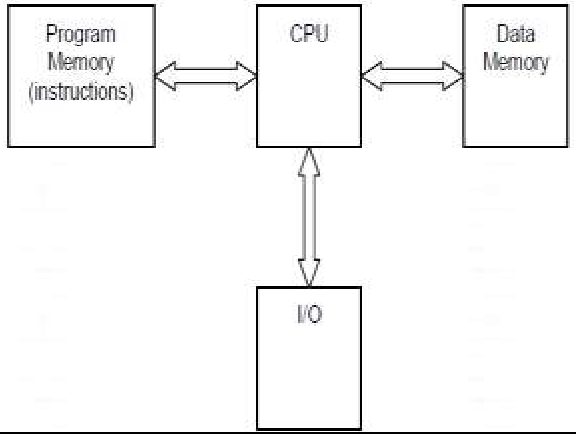
Advantages

- Almost all data can be processed by the von Neumann computer
- Cheaper than alternative types of processing
- Its design is very simple

Disadvantages

- Slower than other architectures
- Limited by bus transfer rate
- Does not maximise CPU utilisation
- Poorly written programs can have their data mixed up as both data and instructions share the same memory

Harvard architecture.

Diagram A		**Diagram B**
		

The Harvard architecture has the following characteristics:

- Stores instructions and data in separate memory, thus has physically separate storage for data and instructions
- Has separate data and instruction buses, allowing transfers to be performed simultaneously on both buses.
- Data and instructions are treated separately
- May employ pipelining. Efficient Pipelining - Operand Fetch and Instruction Fetch can be overlapped.
- Harvard Architecture have a system would have separate caches for each bus.

Using a simple, unified memory system together with a Harvard architecture is highly inefficient. Unless it is possible to feed data into both busses at the same time, it might be better to use a von Neumann architecture processor.

Disadvantages

· Not widely used.
· More difficult to implement.
· More pins needed for buses.

System clock

- It is an electronic component that generates clock pulses to step the control unit through its operation.
- This sends out a sequence of timing pulses or signals, which are used to step the control unit through its operations.
- It generates electric signals at a fast speed
- It controls all functions of computer using clock ticks
- These ticks of system clock are known as clock cycle and speed of CPU
- The speed at which the CPU executes instructions is called clock speed or clock rate.
- It generates a continuous sequence of clock pulse to step the control unit through its operations

Serial Processing

Each instruction is executed in turn until the end of the program.

Advantages

- Nearly all programs can run on serial processing and therefore no additional complex code can be written.
- All data types are suitable for serial processing
- Program can use the previous result in the next operation
- Data set are independent of each other
- Cheaper to handle than parallel

Disadvantages

- Slows data processing especially in the Von Neumann architecture (bottleneck)
- Too much thrashing especially with poorly designed programs

Parallel Processing

- Parallel processing is the ability of a computer system to divide a job into many tasks which are executed simultaneously, using more than one processor, thus allowing multiple processing.
- Multiple CPUs can be used to carry out different parts of the fetch-execute cycle.
- The computer is able to perform concurrent data processing to achieve faster execution time.
- The system may have two or more ALUs and be able to execute two or more instructions at the same time.
- It may also have two or more processors operating concurrently

- The objective is to increase throughput
- Mostly applies to Single Instruction Single Data computer (SISD)
- Supercomputers utilizing parallel processing are used to maintain the safety.
- Scientists are using parallel processing to design computer-generated models of vehicles.
- Airlines use parallel processing to process customer information, forecast demand and decide what fares to charge.
- The medical community uses parallel processing supercomputers.

NB:-

- **Instruction Stream**:-the sequence of instructions read from memory
- **Data stream**: operations performed on the data in the processor

Parallel processing occurs in the instruction stream, the data stream or both.

(a) **Single Instruction Stream, Single Data Stream (SISD)** – Instructions are executed sequentially and parallel processing can be achieved by multiple functional units or by pipelining.

(b) **Single Instruction Stream, Multiple Data Stream (SIMD)-** includes multiple processing units with a single control unit. All processors receive the same instruction but operate on different data

(c) **Multiple Instruction Stream, Single Data Stream (MISD) –** Involves parallel computing where may functional units perform different operations by executing different instructions on the same data set

(d) **Multiple Instruction Stream, Multiple Data Stream (MIMD) –** processor capable of processing several programs at the same time

Advantages of parallel processing

- allows faster processing especially when handling large amounts of data
- more than one instruction (of a program) is processed at the same time
- Not limited (affected) by bus transfer rate
- Can make maximum CPU utilisation as long as it is kept full
- different processors can handle different tasks/parts of same job

- Memory is scalable with number of processors. Increase the number of processors and the size of memory increases proportionately.
- Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache

Disadvantages of parallel processing:

- Only certain types of data is appropriate for parallel processing
- Data that relies on previous operation cannot be made parallel.
- Each data set must be independent of each other
- Usually more expensive
- The programmer is responsible for the details associated with data communication.
- operating system is more complex to ensure synchronisation
- program has to be written in a suitable format
- Program is more difficult to test/write/debug
- It may be difficult to map existing data structures, based on global memory, to this memory organization.

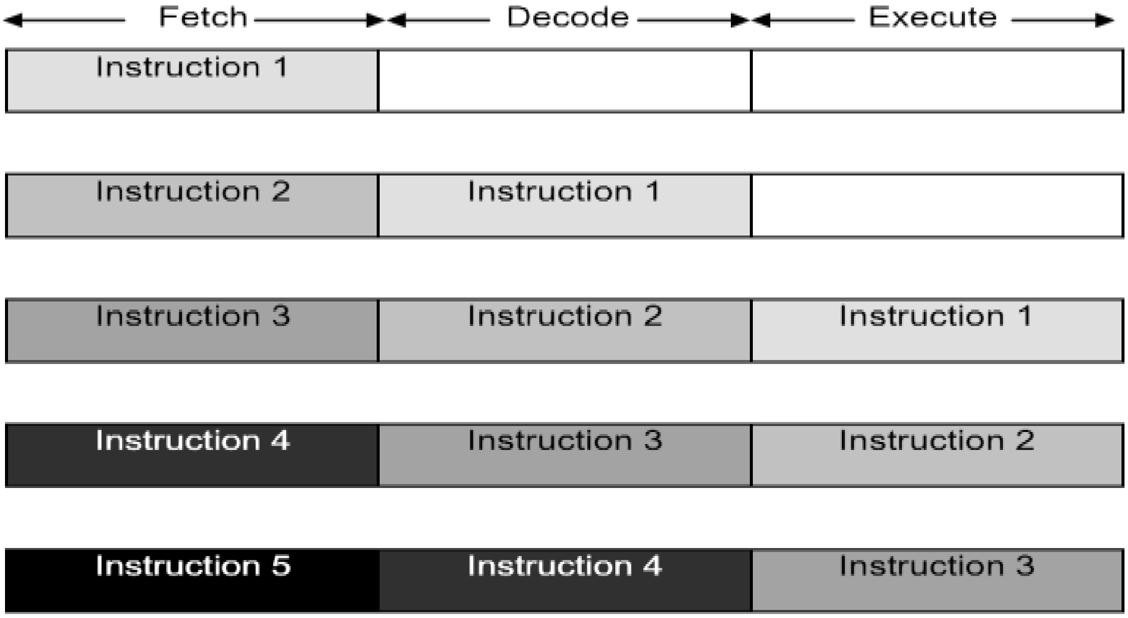
Parallel processing includes **Vector (Array) Processing** and **Pipeline Processing**
Involves the use of a several processors to perform a single job.

1. Array / Vector Processing

- It is when a processor allows the same instruction to operate simultaneously on multiple data locations and apply the same calculation on different data very fast on a processor with several ALUs
- Is a Single Instruction Multiple Data (SIMD)
- It applies to any example of a mathematical problem involving large number of similar calculations, e. g weather forecasting, airflow simulation around new aircraft, games, etc.
- It is a simple processor used for data that can be processed independently of one another –a single instruction can be applied to multiple *bits* of data all at the same time, and none of the results of that data will be needed to process the next bit of data.
- These types of processors are normally used for things like controlling input or output devices. They can be used to track the point of the mouse on a screen, or display the data on the monitor.
- They are sometimes called array processors, because the data is stored in an array, similar to that used in programing, the array can have one to many dimensions.
- Array processors are an extension to the CPU’s arithmetic unit.
- The only disadvantage to array processing is that it relies on the data sets all acting on the same instruction, and it is impossible to use the results of one data set to process the next, although this does not matter in their use.
- It is also important to know, that an array processor is a single instruction multiple data (SIMD) processor, meaning that, lots of bits of data get processed with a single instruction.
- They are expensive to use it processing methods
- They have limitation on type of data to process

2. Pipeline Processing

- It is a technique which allows the overlapping of the fetch-decode-execute cycle for different instructions.
- A parallel processing architecture in which several processors are used, each one doing a different part of the fetch, decode, execute cycle, so the fetch-decode-execute cycle is staggered.
- The processor is split up into three parts (fetch, decode, execute), each of which handles one of the three stages.
- Each part is called a line, where each single line is a pipeline.
- This can be best illustrated with the diagram below.



Fetch	Decode	Execute
Instruction 1		
Instruction 2	Instruction 1	
Instruction 3	Instruction 2	Instruction 1
Instruction 4	Instruction 3	Instruction 2
Instruction 5	Instruction 4	Instruction 3

As long as the pipelines can be kept full, it is making best use of the CPU. This is an example of single instruction single data (SISD) processor, again it should be quite clear why, the processor is processing a single instruction to a single bit of data.

In pipelining, three instructions are dealt with at the same time. This reduces the execution time considerably.
However, this would only be true for a very linear program.
Once jump instructions are introduced the problem arises that the wrong instructions are in the pipeline waiting to be executed, so every time the sequence of instructions changes, the pipeline has to be cleared and the process started again.

A non-pipeline architecture is inefficient because some CPU components (modules) are idle while another module is active during the instruction cycle. Pipelining does not completely cancel out idle time in a CPU but making those modules work in parallel improves program execution significantly.

Processors with pipelining are organized inside into stages which can semi-independently work on separate jobs. Each stage is organized and linked into a 'chain' so each stage's output is fed to another stage until the job is done. This organization of the processor allows overall processing time to be significantly reduced

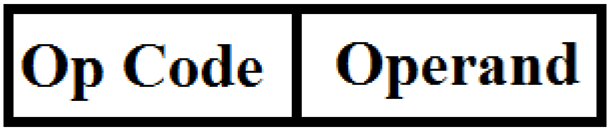
Difference s between Vector and Serial Processors Program Codes

Serial Processor	**Array/ Vector processor**
- execute this loop 10 times - read the next instruction and decode it - fetch this number - fetch that number - add them - put the result here - end loop	- read instruction and decode it - fetch these 10 numbers - fetch those 10 numbers - add them - put the results here

ADDRESSING MODES

Each instruction specifies an operation on certain data.
The different ways in which a computer calculate addresses holding the source and/or destination of the data being processed in a particular instruction is called addressing mode.

Addressing modes are mostly found in assembly language for microprocessors
Each assembly language instruction has the following structure:



Op Code	Operand

Op-code (operator):

- is the part that represent the operations that the computer can understand and carry out. It is the mnemonic part of the instruction/that indicates what it is to do/code for the operation. They are easier to remember. They can be represented by mnemonics which are the pseudo names given to the different operations that make it easier. E.g. ADD.

Operand:

- it is the address field in an instruction that holds data to be used by the operation given in the opcode, e.g. in ADD 12, “12” is the operand
- is the data to be manipulated, there’s no point telling the computer what to ADD if there’s no data to apply it to. It can hold the address of the data, or just the data.

The data is what the operation is being applied to, there are a number of different ways in which this data can be represented, and this is known as addressing.

Symbolic addressing: the use of characters to represent the address of a store location

Effective Address: the actual address of operand to be used by the instruction.

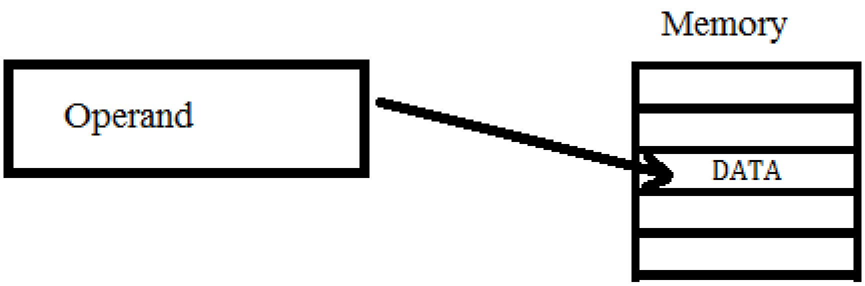
The most common addressing modes are: direct, indirect, indexed, relative and immediate addressing.

1. Immediate Addressing

- This is where the value to be used is stored in the instruction.
- This is when the value in the instruction is not an address at all but the actual data (constant to be used in the program).
- The data to be operated on is held as part of the instruction format.
- The data to be used is stored immediately after the op code for the instruction. Thus the operand field actually contains the data
- e.g: LDA #&80 : Means that Load the hexadecimal value of 80 into the accumulator register.
- MOVE #8, R1: Moves the value 8 into register R1
- Immediate addressing uses the # symbol.
- This is very simple, although not often used because the program parameters cannot be changed.
- This means that the data being operated on can't be adjusted and only uses constants.
- Can be used to initialize constants.

2. Direct Addressing

- The address in the instruction is the address to be used to get to the operand.
- The operand gives the address of the data to be used in the program.



- It requires **one** memory reference to read the operand from the given location
- The address given in the instruction is the one that contains the data to be used in the operation **without any modification.**
- It is also called memory addressing

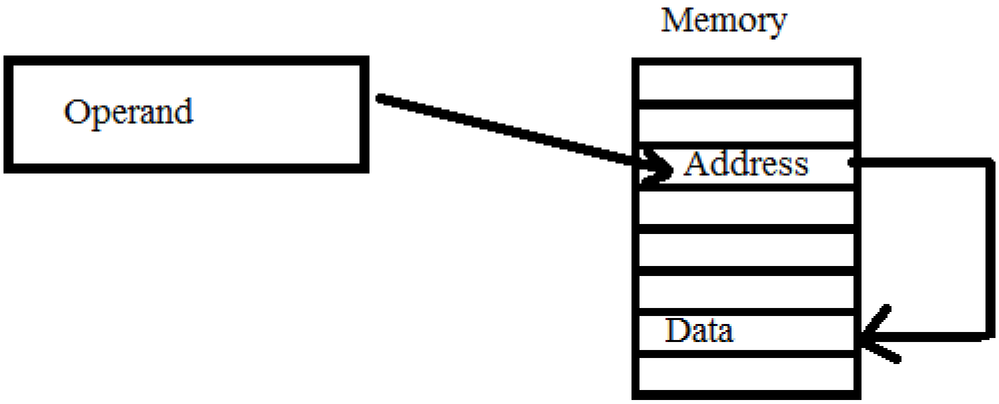
- e.g In the instruction **ADD 23**,
- we first go to memory address 23 which stores the instruction to be executed.
- It provides only a limited address space
- It is very simple, although does not make best use of memory
- It is slow as too much memory is used

Why is it not possible to use only direct addressing in assembly languages?

- Because the number of addresses available in memory is limited by the size
- the address field code is not re-locatable/code uses fixed memory locations

3. Indirect Addressing

- In this mode of addressing, the address given in the instruction holds the address of where the data is stored.
- This is whereby the real address is stored in the memory so the value in the address part of the instruction is pointing to the address of the data.
- The address of data in memory is held in another memory location and the operand of the instruction holds the address of this memory location.

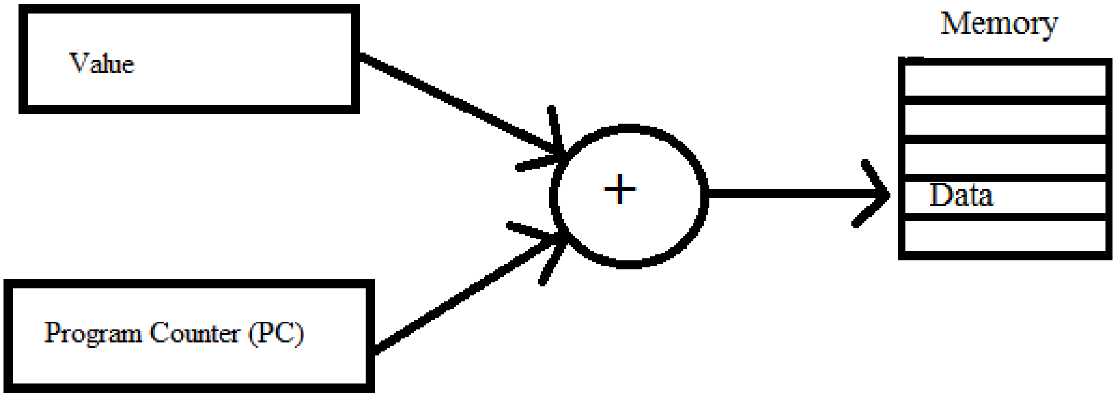


- It is MOSTLY used when access areas of memory that are not accessible using the space available for the address in the instruction code
- using our example, ADD 23
- we go to memory address 23
- there we are given another memory address, e.g 32, where the actual instruction will be stored

- This method is useful because the amount of space in a location is much bigger than the space in the address part of the instruction.
- It gives flexibility as the original program does not need to be altered if the position of the routines (sub-programs) change.
- Therefore we can store larger addresses and use more memory.
- It is used where memory larger than can be accessed by address in instruction
- It is also used when one wants to allow full size of register to be used for address
- used if memory locations are 32 bits are used and thus allowing more memory to be accessed
- there is a problem that some areas of memory cannot be addressed because size of memory address is larger than space available in instruction
- Indirect addressing solves this problem as the Memory address will fit in a memory location

Relative Addressing

- The same as Indexed Addressng except that the PC replces the Index Register.
- E.g Load R_i, X (PC)
- This loads register R_i with the contents of the memory location whose address is the sum of the contents of the PC and the value X.

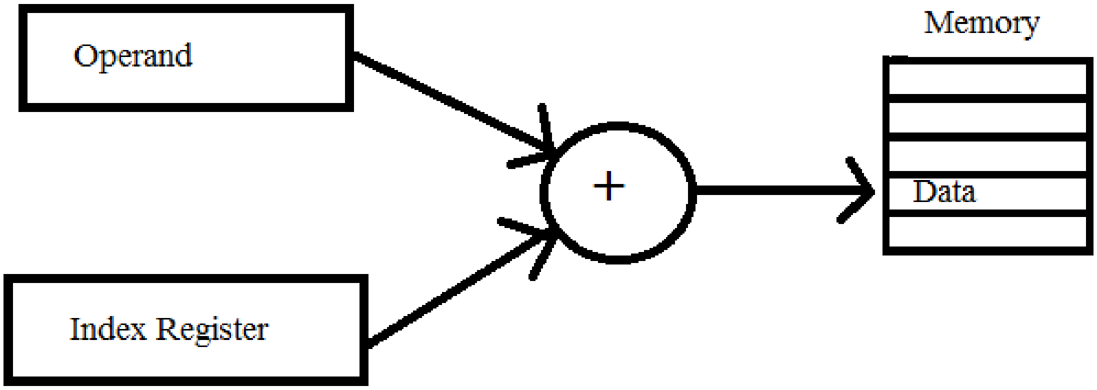


- This is direct addressing that does not commence from the start of the address of the memory.
- It begins from a fixed point, and all addresses are relative to that point.
- allows a real address to be calculated from a base address by adding the relative address
- relative address is an offset and can be used for arrays

- can be used for branching instructions
- it adds the PC contents to the base address to get the effective address
- it is appropriate when the code is going to be loaded at a random place in memory to be executed.
- Use to refer to jump instructions

Indexed Addressing

- The address part of the instruction is added to a value held in the index register.
- It is where the actual address is found by adding a displacement to the base address.



- The result of this is then the required address.
- The instructions specify two registers. The processor then adds contents of these two registers to get the effective address.
- One of the registers is an address register and it holds the base address.
- The other one is a data register or the displacement or index register
- Jmp +10: *branch to instruction 10 bytes on.*
- Used when a number of contiguous locations need to be accessed in order-e.g. contents of array
- Used **when** address in instruction does not change (need not to be altered-like constants), only contents of IR need to be changed

Questions

1. The Program Counter (Sequence Control Register) is a special register in the processor of a computer.

a) Describe the function of the *program counter.* (2)

b) Describe **two** ways in which the program counter can change during the normal execution of a program, explaining, in each case, how this change is initiated. (4)

c) Describe the initial state of the program counter before the running of the program. (2)

2. Explain what is meant by the term Von Neumann Architecture. (2)

3. Describe the fetch/decode part of the fetch/decode/execute/reset cycle, explaining the purpose of any special registers that you have mentioned. (7)

4. a) Describe how pipelining normally speeds up the processing done by a computer. (2)

b) State one type of instruction that would cause the pipeline system to be reset, explaining why such a reset is necessary. (3)

5). Give 3 differences between the Von Neumann and the Harvard Computer architectures.