Advanced Processing Technique
College of Technical /Kirkuk –Software Engineering Dept.
Third class Dr. Ann Zeki

Introduction to supercomputers

A **supercomputer** is a computer with a high-level computational capacity compared to a general-purpose computer. Performance of a supercomputer is measured in floating point operations per second (FLOPS) instead of million instructions per second (MIPS).

Supercomputers are typically (but not exclusively) used for scientific computing. Some applications have included weather forecasting, aerodynamic design, fluid mechanics, radiation modeling, molecular dynamics.

Supercomputers are the fastest and most expensive computers, it helps if we understand what a computer is: it's a general-purpose machine that takes in information (data) by a process called input, stores and processes it, and then generates some kind of output (result). A supercomputer is not simply a fast or very large computer: it works in an entirely different way, typically using parallel processing instead of the serial processing that an ordinary computer uses. Instead of doing one thing at a time, it does many things at once.

Uniprocessor system

A **uniprocessor system** is defined as a computer system that has a single central processing unit that is used to execute computer tasks .The term *uniprocessor* is therefore used to distinguish the class of computers where all processing tasks share a single CPU. Most desktop computers are now shipped with multiprocessing architectures. A type of architecture that is based on a single computing unit. All operations (additions, multiplications, etc.) are done sequentially on the unit.

Multiprocessing System

Multiprocessing is the use of two or more central processing units (CPUs) within a single computer system. The term also refers to the ability of a system to support more than one processor and/or the ability to allocate tasks between them. There are many variations on this basic theme, and the definition of multiprocessing can vary with context, mostly as a function of how CPUs are defined multiple packages in one system unit, etc.

According to some on-line dictionaries, a **multiprocessor** is a computer system having two or more processing units (multiple processors) each sharing main memory and peripherals, in order to simultaneously process programs.

Parallel computers

Parallel computing uses multiple processing elements simultaneously to solve a problem. This is accomplished by breaking the problem into independent parts so that each processing element can execute its part of the algorithm simultaneously with the others. The processing elements can be diverse and include resources such as a single computer with multiple processors, several networked computers, specialized hardware, or any combination of the above.

Parallelism

Parallelism is the use of components in a sentence that are grammatically the same; or similar in their construction, sound, meaning or meter. Parallelism examples are found in literary works as well as in ordinary conversations.

Concurrency

In a database management system (DBMS), concurrency control manages simultaneous access to a database. It prevents two users from editing the same record at the same time and also serializes transactions for backup and recovery.

Overlapping

If you have two pieces of something, and one is covering a part of another, then they're overlapping.

If either block (source or destination) straddles a page boundary, a page fault might occur after the move is partially done. In addition, if the source and destination blocks overlap, the source block may have been modified, in which case we cannot simply restart the instruction.
This problem can be solved in two different ways. In one solution, the microcode computes and attempts to access both ends of both blocks. If a page fault is going to occur, it will happen at this step, before anything is modified.
The move can then take place; we know that no page fault can occur, since all the relevant pages are in memory. The other solution uses temporary registers to hold the values of overwritten locations. If there is a page fault, all the old values are written back into memory before the trap occurs. This action restores memory to its state before the instruction was started, so that the instruction can be repeated.
This is by no means the only architectural problem resulting from adding paging to an existing architecture to allow demand paging, but it illustrates some of the difficulties involved. Paging is added between the CPU and the memory in a computer system. It should be entirely transparent to the user process. Thus, people often assume that paging can be added to any system.
Although this assumption is true for a non-demand-paging environment, where a page fault represents a fatal error, it is not true where a page fault means only that an additional page must be brought into memory and the process restarted.

Speedup

If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others. Notice that such a speedup can be achieved only if the computer has multiple processing cores.
In the field of computer architecture, speedup is a metric for relative performance improvement when executing a task. The notion of speedup was particularly

focused in the context of parallel processing. However, speedup can be used more generally to show the effect of any performance enhancement.

Bandwidth

The data transfer rate is commonly used to measure how fast data is transferred from one location to another. For example, a hard drive may have a maximum data transfer rate of 480 Mbps, while an Internet connection with a maximum data transfer rate of only 1.5 Mbps.

Data transfer rates are typically measured in bits per second (bps) as opposed to bytes per second, which can be understandably confusing. Because there are eight bits in a byte, a sustained data transfer rate of 80 Mbps is only transferring 10MB per second. While this is confusing for consumers, Internet service providers must enjoy measuring data transfer rates in bps.

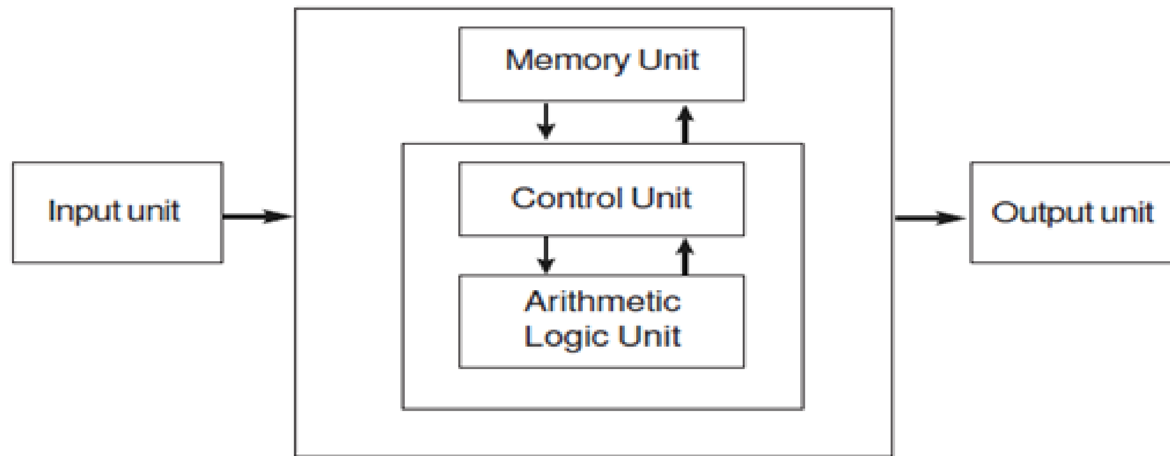


Fig. 1.1 Block diagram of Computer Organisation

WHAT IS A COMPUTER?

Computer is a device that transforms data into meaningful information. Data can be anything like marks obtained by you in various subjects. It can also be name, age, weight, height, etc. of all the students in a class. Computer can also be defined in terms of functions it can perform. A computer can i) accept data, ii) store data, iii) process data as desired, and iv) retrieve the stored data as and when required and v) print the result in desired format .The Block diagram of Computer Organization in fig.1.1.The major characteristics of a computer are high speed, accuracy, diligence, versatility and storage. The computer performs basically five major operations of functions irrespective of their size and make. These are:

1. It accepts data or instruction by way of input.
2. It stores data.
3. It can process data as required by the user.
4. It gives results in the form of output.
5. It controls all operations inside a computer.

We discuss below each of these operations.

1. Input: this is the process of entering data and programs into the computer system
2. Control Unit (CU): The process of input, output, processing and storage is performed under the supervision of a unit called 'Control Unit'. It decides when to start receiving data, when to stop it, where to store data, etc. It takes care of step-by-step processing of all operations inside the computer.

3. Memory Unit: Computer is used to store data and instructions.
4. Arithmetic Logic Unit (ALU): The major operations performed by the ALU are addition, subtraction, multiplication, division, logic and comparison.
5. Output: This is the process of producing results from the data for getting useful information.

The ALU and the CU, carry look ahead adder CLA, carry generate CGA ,FP addition ,FP multiplication of a computer system are jointly known as the central processing unit (CPU). You may call CPU as the brain of any computer system.

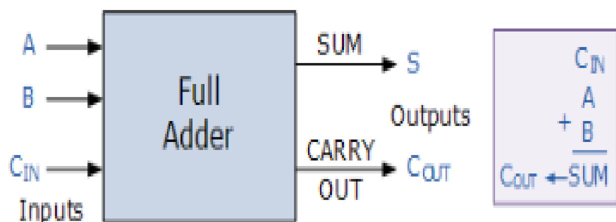
ALU Arithmetic and Logic Unit

An arithmetic logic unit (ALU) is a combinational digital electronic circuit that performs arithmetic and bitwise operations on integer binary numbers. This is in contrast to a floating-point unit (FPU), which operates on floating point numbers. An ALU is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing units (GPUs). A single CPU, FPU or GPU may contain multiple ALUs.

The inputs to an ALU are the data to be operated on, called operands, and a code indicating the operation to be performed; the ALU's output is the result of the performed operation. In many designs, the ALU also has status inputs or outputs, or both, which convey information about a previous operation or the current operation, respectively, between the ALU and external status registers.

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

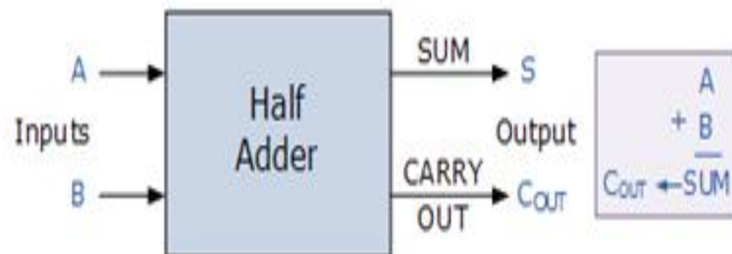
Most of the operations of a CPU are performed by one or more ALUs, which load data from input registers. A **register** is a small amount of storage available as part and the ALU stores the result in an output register.



Binary Adder

Binary Adders are arithmetic circuits in the form of half-adders and full-addersb used to add together two binary digits

Binary Adder Block Diagram



The major difference between **Half Adder** and **Full Adder** is that Half Adder adds two **1-bit numbers** given as input but do not add the carry obtained from previous addition while the Full Adder, along with **two 1-bit numbers** can also add the carry obtained from previous addition.

Another common and very useful combinational logic circuit which can be constructed using just a few basic logic gates allowing it to add together two or more binary numbers is the **Binary Adder**.

A basic Binary Adder circuit can be made from standard AND and Ex-OR gates allowing us to "add" together two single bit binary numbers, A and B.

The addition of these two digits produces an output called the SUM of the addition and a second output called the CARRY or Carry-out, (C_{OUT}) bit according to the rules for binary addition. One of the main uses for the *Binary Adder* is in arithmetic and counting circuits. Consider the simple addition of the two denary (base 10) numbers below.

123	A	
+ 789	B	(Addend)
912	SUM	

From our math's lessons at school, we learnt that each number column is added together starting from the right hand side and that each digit has a weighted value depending upon its position within the columns.

When each column is added together a carry is generated if the result is greater or equal to 10, the base number. This carry is then added to the result of the addition of the next column to the left and so on, simple school math’s addition, add the numbers and carry.

The adding of binary numbers is exactly the same idea as that for adding together decimal numbers but this time a carry is only generated when the result in any column is greater or equal to “2”, the base number of binary. In other words 1 + 1 creates a carry.

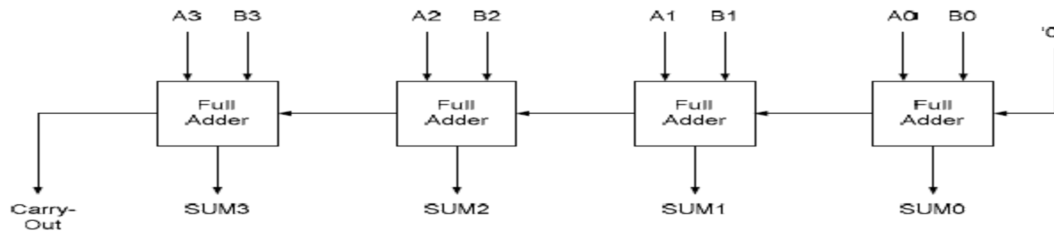
Binary Addition follows these same basic rules as for the denary addition above except in binary there are only two digits with the largest digit being “1”. So when adding binary numbers, a carry out is generated when the “SUM” equals or is greater than two (1+1) and this becomes a “CARRY” bit for any subsequent addition being passed over to the next column for addition and so on. Consider the single bit addition below.

Binary Addition of Two Bits

0	0	1	1
+ 0	+ 1	+ 0	+ 1
0	1	1	(carry) 1←0

When the two single bits, A and B are added together, the addition of “0 + 0”, “0 + 1” and “1 + 0” results in either a “0” or a “1” until you get to the final column of “1 + 1” then the sum is equal to “2”. But the number two does not exists in binary however, 2 in binary is equal to 10, in other words a zero for the sum plus an extra carry bit.

Then the operation of a simple adder requires two data inputs producing two outputs, the Sum (S) of the equation and a Carry (C) bit as shown.



Flags set in the Status Register by the ALU

An important function of the ALU is to set up bits or flags which give information to the control unit about the result of an operation. The flags are grouped together in the status word.

As the ALU has only an adder, to subtract numbers one has to use 2s-complement arithmetic. The ALU has no knowledge of this at all — it simply adds two binary inputs and sets the flags. It is up to control unit (or really the programmer's instructions executed by the control unit) to interpret the results.

Z Zero flag: This is set to 1 whenever the output from the ALU is zero. N **Negative flag**: This is set to 1 whenever the most significant bit of the output is 1. Note that it is not correct to say that it is set when the output of the ALU is negative: the ALU doesn't know or care whether you are working in 2's complement. However, this flag is used by the controller for just such interpretations.

C **Carry flag:** Set to 1 when there is a carry from the adder.

V **oVerflow flag**: Set to 1 when $A_{msb} = 1$, $B_{msb} = 1$, but $O_{msb} = 0$; or when $A_{msb} = 0$, $B_{msb} = 0$, but $O_{msb} = 1$. Allows the controller to detect overflow during 2's complement addition. Note (i) that the ALU does not know *why* it is setting the flag, and (ii) in our BSA, the msb=15.

Counter Registers the PC and SP are counting registers, which either act as loadable registers (loaded from the IR (address) register) or count, both on receipt of a clock pulse. The internal workings appear complicated, but are unremarkable, so we leave them as black boxes here.

Multipliers

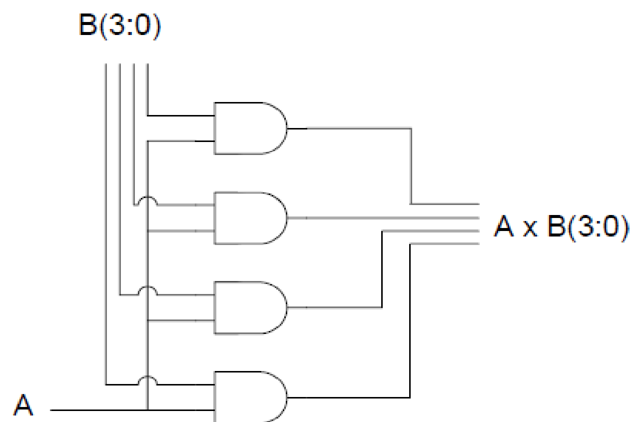
Consider a 1 bit x 1 bit multiplier:

0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1

Multiplication of two bits is simply an AND gate

Multipliers

One bit number x N-bit number
eg. 10111011 x 1 = 10111011
10111011 x 0 = 00000000



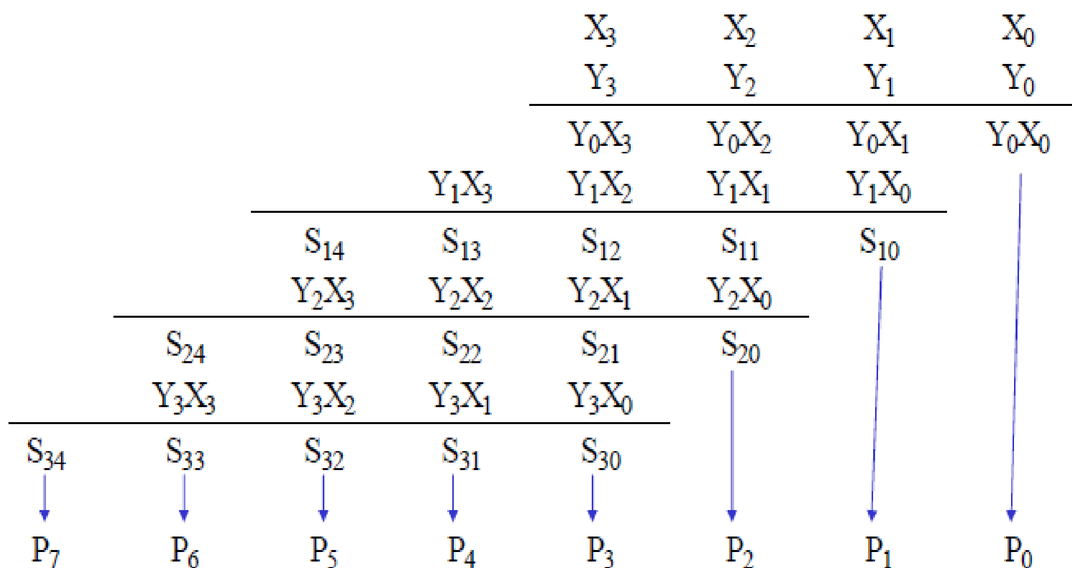
Multipliers

N bit x N bit number (consider 4x4):

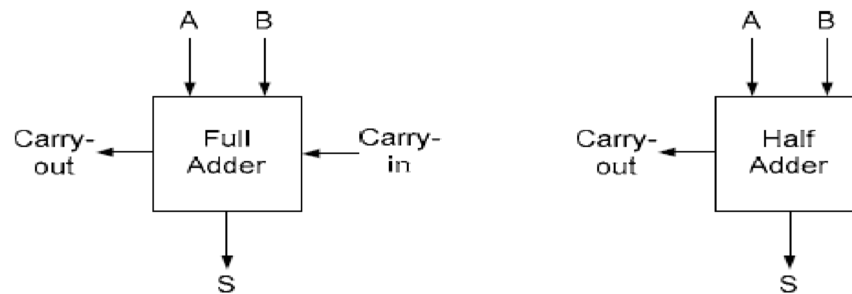
				X_3	X_2	X_1	X_0
				Y_3	Y_2	Y_1	Y_0
				Y_0X_3	Y_0X_2	Y_0X_1	Y_0X_0
			Y_1X_3	Y_1X_2	Y_1X_1	Y_1X_0	
		Y_2X_3	Y_2X_2	Y_2X_1	Y_2X_0		
	Y_3X_3	Y_3X_2	Y_3X_1	Y_3X_0			
S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0

This would require 16 AND gates and a 8 bit, 4 input adder, and lots of wiring (wiring is becoming a big problem in integrated circuits).

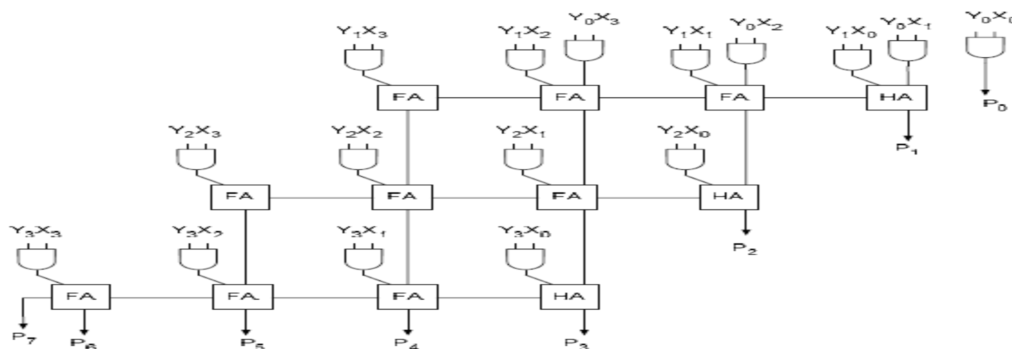
We can reduce the wiring by distributing the adder:



Recall: Half-Adder vs. Full-Adder:



By combining half-adders, full-adders, and AND gates, we can implement our array multiplier



Addressing Technique

Addressing mode is the way of addressing a memory location in instruction that tells us what the type of the operand is and the way they are accessed from the memory for execution of an instruction and how to fetch particular instruction from the memory.

There are various methods of giving source and destination address in instruction, thus there are various types of Addressing Modes. Here you will find the different types of Addressing Modes are explained below:

Following are the types of Addressing Modes:

- Register Addressing Mode
- Direct Addressing Mode
- Register Indirect Addressing Mode
- Immediate Addressing Mode
- Index Addressing Mode

Explanation:

1. **Register Addressing Mode:** In this type of addressing mode both the operands are registers.
   Example:

```
MOV AX, BX
XOR AX, DX
ADD AL, BL
```

2. **Direct Addressing Mode**: The addressing mode in which the effective address of the memory location is written directly in the instruction.

Example

```
MOV AX, [1592H]
MOV AL, [0300H]
```

3. **Register indirect mode –** In this addressing mode the effective address is in SI, DI or BX.
   Example:

```
MOV AX, [DI]
ADD AL, [BX]
MOV AX, [SI]
```

4. **Immediate Addressing Mode**

In this immediate data is the part of the instruction itself.

Example:

```
Mov Ax, 0005H
```

5. Index Addressing Mode

In this type of addressing mode the effective address is sum of index register and displacement.
Example:

```
MOV AX, [SI+2000]
MOV AL, [DI+3000]
```

Basic I/O Techniques

Three basic techniques are available in current computers systems. From the least efficient to the best performing, we find:

1. Programmed I/O where the CPU is strictly tied to the I/O until it is finished.
2. Interrupt-driven I/O per character transfer
3. DMA (Direct Memory Access) a memory controller mechanism programmed by the OS to exchange data at high rate between main memory and the I/O device; the big win with DMA is twofold. First we reduce the number of interrupts from one per character to one per buffer. Second we benefit from the streaming effect of the data transfer.

The Operating System is responsible for choosing and managing the right technique for each specific I/O in order to deliver the best overall performance.

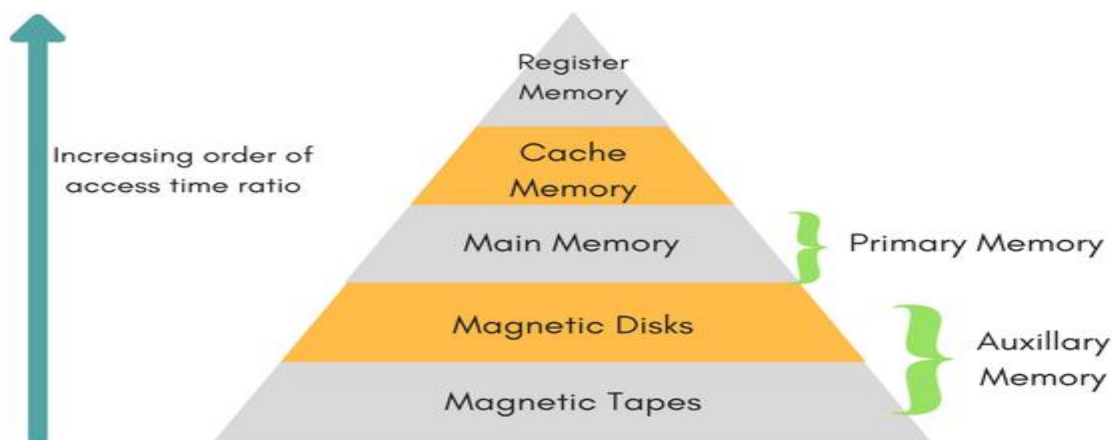
Memory System

A memory unit is the collection of storage units or devices together. The memory unit stores the binary information in the form of bits. Generally, memory/storage is classified into 2 categories:

Volatile Memory: This loses its data, when power is switched off.

Non-Volatile Memory: This is a permanent storage and does not lose any data when power is switched off.

Memory Hierarchy



The total memory capacity of a computer can be visualized by hierarchy of components. The memory hierarchy system consists of all storage devices

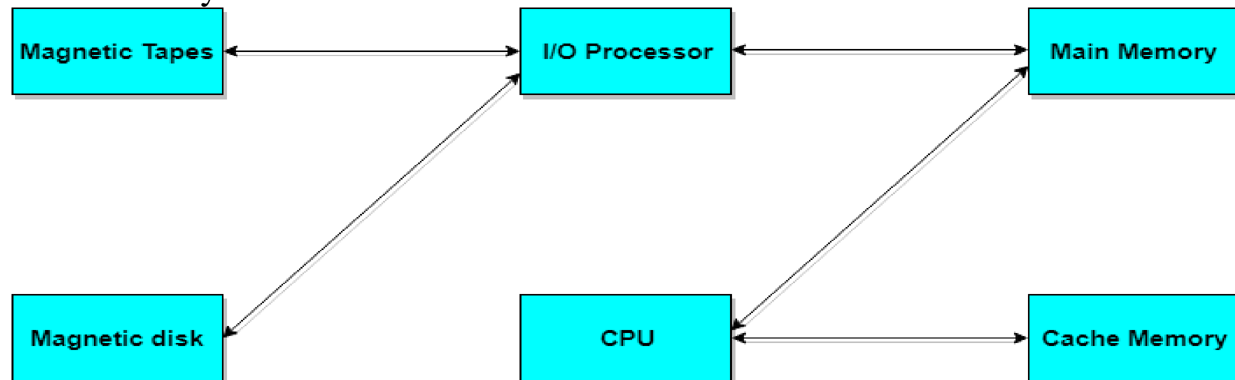
contained in a computer system from the slow Auxiliary Memory to fast Main Memory and to smaller Cache memory.

Auxiliary memory access time is generally **1000 times** that of the main memory, hence it is at the bottom of the hierarchy.

The **main memory** occupies the central position because it is equipped to communicate directly with the CPU and with auxiliary memory devices through Input/output processor (I/O).

When the program not residing in main memory is needed by the CPU, they are brought in from auxiliary memory. Programs not currently needed in main memory are transferred into auxiliary memory to provide space in main memory for other programs that are currently in use.

The **cache memory** is used to store program data which is currently being executed in the CPU. Approximate access time ratio between cache memory and main memory is about **1 to 7~10**



Memory Access Methods

Each memory type is a collection of numerous memory locations. To access data from any memory, first it must be located and then the data is read from the memory location. Following are the methods to access information from memory locations:

Random Access: Main memories are random access memories, in which each memory location has a unique address. Using this unique address any memory location can be reached in the same amount of time in any order.

Sequential Access: This method allows memory access in a sequence or in order.

Direct Access: In this mode, information is stored in tracks, with each track having a separate read/write head.

Main Memory

The memory unit that communicates directly within the CPU, Auxiliary memory and Cache memory, is called main memory. It is the central storage unit of the computer system. It is a large and fast memory used to store data during computer operations. Main memory is made up of **RAM** and **ROM**, with RAM integrated circuit chips holing the major share.

RAM: Random Access Memory

DRAM: Dynamic RAM, is made of capacitors and transistors, and must be refreshed every 10~100 ms. It is slower and cheaper than SRAM.

SRAM: Static RAM, has a six transistor circuit in each cell and retains data, until powered off.

NVRAM: Non-Volatile RAM retains its data, even when turned off. Example: Flash memory.

ROM: Read Only Memory, is non-volatile and is more like a permanent storage for information. It also stores the **bootstrap loader** program, to load and start the operating system when computer is turned on. **PROM** (Programmable ROM), **EPROM** (Erasable PROM) and **EEPROM** (Electrically Erasable PROM) are some commonly used ROMs.

Auxiliary Memory

Devices that provide backup storage are called auxiliary memory. **For example:** Magnetic disks and tapes are commonly used auxiliary devices. Other devices used as auxiliary memory are magnetic drums, magnetic bubble memory and optical disks.

It is not directly accessible to the CPU, and is accessed using the Input/output channels.

Cache Memory

The data or contents of the main memory that are used again and again by CPU, are stored in the cache memory so that we can easily access that data in shorter time.

Whenever the CPU needs to access memory, it first checks the cache memory. If the data is not found in cache memory then the CPU moves onto the main memory. It also transfers block of recent data into the cache and keeps on deleting the old data in cache to accommodate the new one.

Hit Ratio

The performance of cache memory is measured in terms of a quantity called **hit ratio**. When the CPU refers to memory and finds the word in cache it is said to

produce a **hit**. If the word is not found in cache, it is in main memory then it counts as a **miss**.
The ratio of the number of hits to the total CPU references to memory is called hit ratio.

Hit Ratio = Hit/ (Hit + Miss)

The capacity of memories in commercial computers is usually stated as the total number of bytes that can be stored.

Internal Structure of a Memory Unit

The internal structure of a memory unit is specified by the **number of words** it contains and the **number of bits** in each word. Special input lines called **address lines** select one particular word. Each word in memory is assigned an identification number, called an **address**, staring from 0 and continuing with 1, 2, 3, up to 2^k-1 where **k** is the number of address lines. The selection of a specific word inside the memory is done by applying the k-bit binary address to the address lines.

A **decoder** inside the memory accepts this address and opens the paths needed to select the bits of the specified word.

Computer memories may range from 1024 words, requiring an address of 10 bits, to 2^{32} words, requiring 32 address bits. It is customary to refer to the number of words (or bytes) in a memory with one of the letters:

- **K(Kilo)** is equal to 2^{10}
- **M(Mega)** is equal to 2^{20}
- **G(Giga)** is equal to 2^{30}

Two major types of memories are used in computer systems: **Random Access Memory (RAM)** and **Read Only Memory (ROM)**. These semiconductor memories are classified into **Random Access Memories (RAMs)** and **Sequential Access Memories (SAMs)** based on access time.

Memories constructed with shift registers, **Charged Coupled Devices (CCDs)**, or bubble memories are examples of SAMs. RAMs are categorized into ROMs, **Read Mostly Memories (RMMs)**, and **Read Write Memories (RWMs)**.

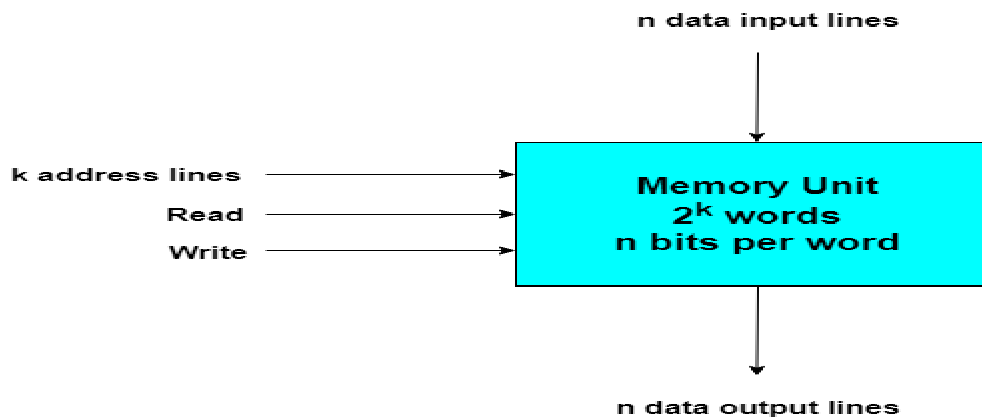
ROMs are of two types: **Masked Programmed ROMs** and **User Programmed PROMs**.

Two types of RMMs are **Erasable and Programmable (EPROM)** and **Electrically Erasable (EEPROM)**.

RWMs are **Static RAM (SRAM)** and **Dynamic RAM (DRAM)**. Static RAMs have memory cells as common Flip-Flops. Dynamic RAMs have memory cells that must be refreshed, read and written periodically to avoid loss of memory cells.

Random Access Memory (RAM)

In **random-access memory(RAM)** the memory cells can be accessed for information transfer from any desired random location. That is, the process of locating a word in memory is the same and requires an equal amount of time no matter where the cells are located physically in memory. Communication between a memory and its environment is achieved through data input and output lines, address selection lines, and control lines that specify the direction of transfer. A block diagram of a RAM unit is shown below:



The **n** data input lines provide the information to be stored in memory, and the **n** data output lines supply the information coming out of particular word chosen among the $\mathbf{2^k}$ available inside the memory. The two control inputs specify the direction of transfer desired.

RAM: Write and Read Operations

The two operations that a random access memory can perform are the **write** and **read** operations. The write signal specifies a transfer-in operation and the read signal specifies a transfer-out operation. On accepting one of these control signals. The internal circuits inside the memory provide the desired function. The steps that must be taken for the purpose of transferring a new word to be stored into memory are as follows:

1. Apply the **binary address** of the desired word into the address lines.
2. Apply the **data bits** that must be stored in memory into the data input lines.
3. Activate the **write** input.

The memory unit will then take the bits presently available in the input data lines and store them in the specified by the address lines. The steps that must be taken for the purpose of transferring a stored word out of memory are as follows:

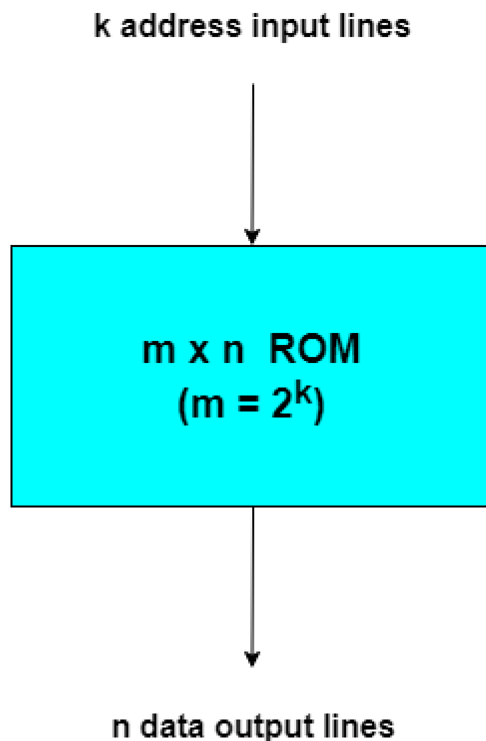
1. Apply the **binary address** of the desired word into the address lines.
2. Activate the **read** input.

The memory unit will then take the bits from the word that has been selected by the address and apply them into the output data lines. The content of the selected word does not change after reading.

Read Only Memory (ROM)

As the name implies, a **read-only memory (ROM)** is a memory unit that performs the read operation only; it does not have a write capability. This implies that the binary information stored in a ROM is made permanent during the hardware production of the unit and cannot be altered by writing different words into it.

Whereas a RAM is a general-purpose device whose contents can be altered during the computational process, a ROM is restricted to reading words that are **permanently** stored within the unit. The binary information to be stored, specified by the designer, is then embedded in the unit to form the required interconnection pattern. ROMs come with special internal electronic fuses that can be **programmed** for a specific configuration. Once the pattern is established, it stays within the unit even when power is turned off and on again.An **m x n** ROM is an array of binary cells organized into **m** words of **n** bits each. As shown in the block diagram below, a ROM has **k** address input lines to select one of $\mathbf{2^k} =$ **m** words of memory, and n input lines, one for each bit of the word. An integrated circuit ROM may also have one or more enable inputs for expanding a number of packages into a ROM with larger capacity.



The ROM does not need a read-control line since at any given time, the output lines automatically provide the n bits of the word selected by the address value. Because the outputs are a function of only the present inputs (the address lines), a ROM is classified as a combinational circuit. In fact, a ROM is constructed internally with decoders and a set of OR gates. There is no need for providing storage capabilities as in RAM, since the values of the bits in the ROM are permanently fixed.

ROMs find a wide range of applications in the design of digital systems. As such, it can implement any combinational circuit with **k** inputs and **n** outputs. When employed in a computer system as a memory unit, the ROM is used for storing fixed programs that are not to be altered and for tables of constants that are not subject to change. ROM is also employed in the design of control units for digital computers. As such, they are used to store coded information that represents the sequence of internal control variables needed for enabling the various operations in the computer. A control unit that utilizes a ROM to store binary control information is called a microprogrammed control unit.

ROM

The required paths in a ROM may be programmed in three different ways.
The first, **mask programming**, is done by the semiconductor company during the last fabrication process of the unit. This procedure is costly because the vendor charges the customer a special fee for custom masking the particular ROM. For this reason, mask programming is economical only if a large quantity of the same ROM configuration is to be ordered.
For small quantities it is more economical to use a second type of ROM called a **Programmable Read Only Memory (PROM)**. The hardware procedure for programming ROMs or PROMs is **irreversible**, and once programmed, the fixed pattern is permanent and cannot be altered. Once a bit pattern has been established, the unit must be discarded if the bit pattern is to be changed.
A third type of ROM available is called **Erasable PROM** or **EPROM**. The EPROM can be restructured to the initial value even though its fuses have been blown previously. Certain PROMs can be erased with electrical signals instead of ultraviolet light. These PROMs are called **Electrically Erasable PROM** or **EEPROM**. Flash memory is a form of EEPROM in which a block of bytes can be erased in a very short duration. Example applications of **EEPROM** devices are: Storing current time and date in a machine. Storing port statuses. Example of **Flash memory device** applications is:
Storing messages in a mobile phone. Storing photographs in a digital camera.

The memory system is a collection of storage locations. Each storage location, or memory word, has a numerical address. A collection of storage locations from an address space. Figure 2 shows the essentials of how a processor is connected to a memory system via address, data, and control lines. When a processor attempts to load the contents of a memory location, the request is very urgent. In virtually all computers, the work soon comes to a halt (in other words, the processor stalls) if the memory request does not return quickly. Modern computers are generally able to continue briefly by overlapping memory requests, but even the most sophisticated computers will frequently exhaust their ability to process data and stall momentarily in the face of long memory delays. Thus, a key performance parameter in the design of any computer, fast or slow, is the effective speed of its memory.
Ideally, the memory system must be both infinitely large so that it can contain an arbitrarily large amount of information, and infinitely fast so that it does not limit the processing unit. Practically, however, this is not possible.

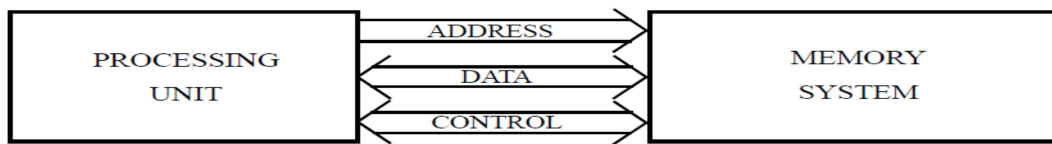


Figure (2)

There are three properties of memory that are inherently in conflict:

Speed, capacity, and cost.

Memory Hierarchies

Technology does not permit memories that are cheap, large, and fast. By recognizing the nonrandom nature of memory requests, and emphasizing the average rather than worst case latency, it is possible to implement a hierarchical memory system that performs well. A small amount of very fast memory, placed in front of a large, slow memory, can be designed to satisfy most requests at the speed of the small memory. This, in fact, is the primary motivation for the use of registers in the CPU: in this case, the programmer makes sure that the most commonly accessed variables are allocated to registers.

A variety of techniques, employing either hardware, software, or a combination of the two, can be employed to assure that most memory references are satisfied by the faster memory. The foremost of these techniques is the exploitation of the locality of reference principle. This principle captures the fact that some memory locations are referenced much more frequently than others. Spatial locality is the property that an access to a given memory location greatly increases the probability that neighboring locations will be accessed immediately. This is largely, but not exclusively, a result of the tendency to access memory locations sequentially. Temporal locality is the property that an access to a given memory location greatly increases the probability that the same location will be accessed again soon. This is largely, but not exclusively, a result of the high frequency of programs' looping behavior. Particularly for temporal locality, a good predictor of the future is the past: the longer a vari4 able has gone unreferenced, the less likely it is to be accessed soon.

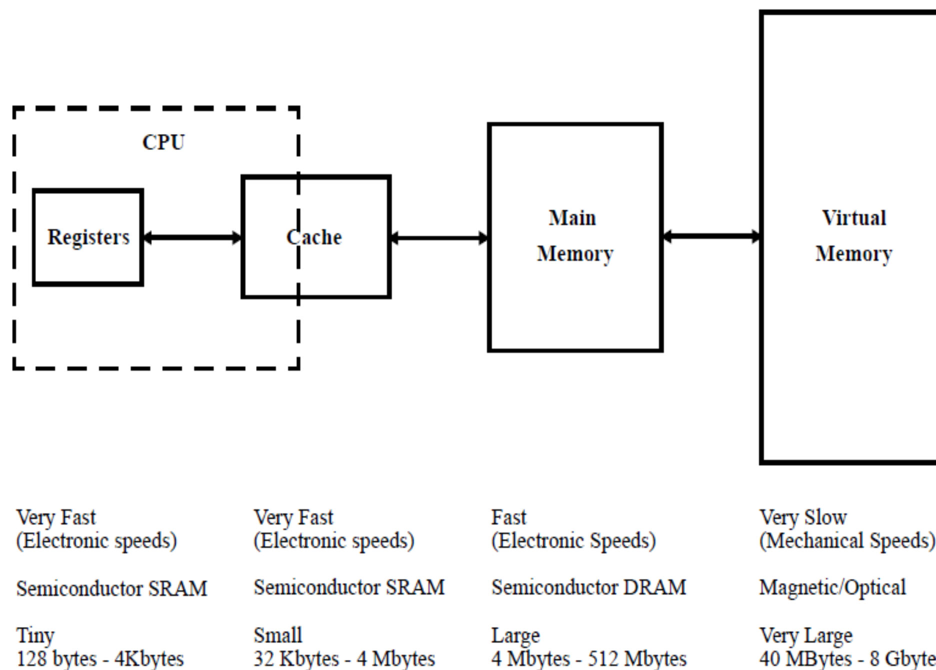


Figure (3) Memory Hierarchy

Figure (3) depicts a common construction of a memory hierarchy. At the top of the hierarchy are the CPU registers, which are small and extremely fast. The next level down in the hierarchy is a special, high-speed semiconductor memory, known as a **cache memory**. The cache can actually be divided into multiple distinct levels; most current systems have between one and three levels of cache. Some of the levels of cache may be on the CPU chip itself, they may be on the same module as the CPU, or they may all be entirely distinct. Below the cache is the conventional memory, referred to as main memory, or backing storage. Like a cache, main memory is semiconductor memory, but it is slower, cheaper, and denser than a cache. Below the main memory is the virtual memory, which is generally stored on magnetic or optical disk. Accessing the virtual memory can be tens of thousands of times slower than accessing the main memory, since it involves moving mechanical parts. As requests go deeper into the memory hierarchy, they encounter levels that are larger (in terms of capacity) and slower than the higher levels. In addition to size and speed, the bandwidth in between adjacent levels in the memory hierarchy is smaller for the lower levels. The bandwidth in between the registers and top cache level, for example, is higher than that between cache and main memory or main memory and virtual memory. Since each level presumably intercepts a fraction of the requests, the bandwidth to the level below need not be as great as that to the intercepting level.

Memory Interleaving

Interleaving: It is a technique for connecting multiple memory modules together in order to improve the bandwidth of the memory system.

The interleaved main memory is demonstrated in a system made up 16 units, 16 memory modules, a split transactions synchronous bus and the arbiter. Each unit can be configured to generate either the single word accesses or the block accesses.

To speed up the memory operations (read and write), the main memory of $2^n = N$ words can be organized as a set of $2^m = M$ independent memory modules (where $m < n)$ each containing 2^{n-m} words. If these M modules can work in parallel (or in a pipeline fashion), then ideally an M fold speed improvement can be expected. The n-bit address is divided into an m-bit field to specify the module, and another (n-m)-bit field to specify the word in the addressed module. The field for specifying the modules can be either the most or least significant m bits of the address. For example, these are the two arrangements of $M = 2^m = 2^2 = 4$ modules ($m = 2$) of a memory of $2^n = 2^4 = 16$ words ($n = 4$):

High−order arrangement

Addr	M0		Addr	M1		Addr	M2		Addr	M3	
0	00	00	4	01	00	8	10	00	12	11	00
1	00	01	5	01	01	9	10	01	13	11	01
2	00	10	6	01	10	10	10	10	14	11	10
3	00	11	7	01	11	11	10	11	15	11	11

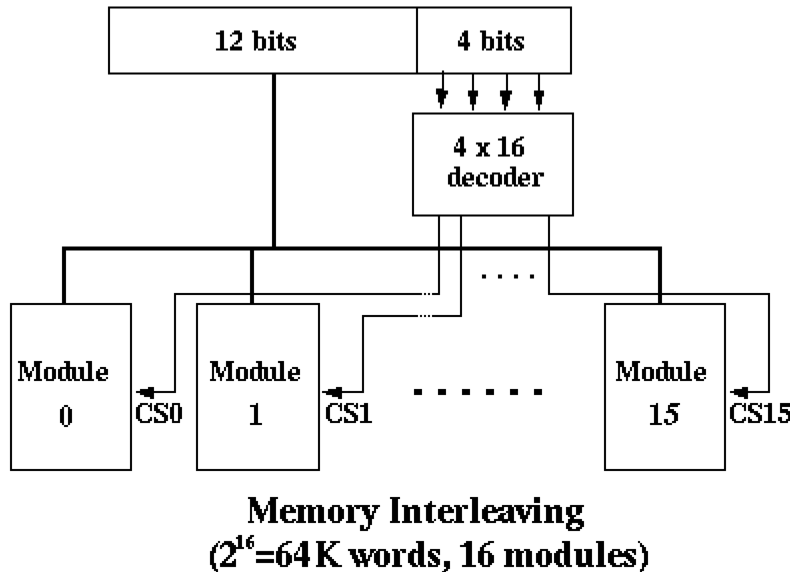
Low−order arrangement (interleaving)

Addr	M0		Addr	M1		Addr	M2		Addr	M3	
0	00	00	1	00	01	2	00	10	3	00	11
4	01	00	5	01	01	6	01	10	7	01	11
8	10	00	9	10	01	10	10	10	11	10	11
12	11	00	13	11	01	14	11	10	15	11	11

In general, the CPU is more likely to need to access the memory for a set of consecutive words (either a segment of consecutive instructions in a program or the components of a data structure such as an array, the interleaved (low-order) arrangement is preferable as consecutive words are in different modules and can be fetched simultaneously. In case of high-order arrangement, the consecutive words

are usually in one module, having multiple modules is not helpful if consecutive words are needed.

Example: A memory of $2^{16} = 64k$ words (n=16) with $2^4 = 16$ modules (m=4) each containing $2^{n-m} = 2^{12} = 4k$ words:



Memory Interleaving
(2^{16}=64K words, 16 modules)

For example: If we have 4 memory banks(4-way Interleaved memory), with each containing 256 bytes, then, the Block Oriented scheme(no interleaving), will assign virtual address 0 to 255 to the first bank, 256 to 511 to the second bank. But in Interleaved memory, virtual address 0 will be with the first bank, 1 with the second memory bank, 2 with the third bank and 3 with the fourth, and then 4 with the first memory bank again.

Hence, CPU can access alternate sections immediately without waiting for memory to be cached. There are multiple memory banks which take turns for supply of data.

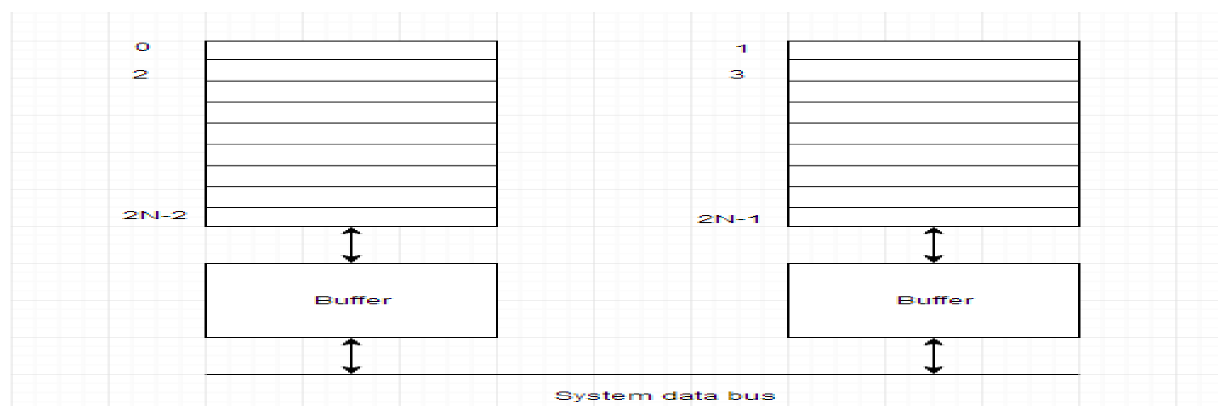
Memory interleaving is a technique for increasing memory speed. It is a process that makes the system more efficient, fast and reliable.

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 followed 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.

The following figure shows the organization of two physical banks of n long words. All even long words of logical bank are located in physical bank 0 and all odd long words are located in physical bank 1.



There are various types of interleaving:
Two-Way Interleaving: Two memory blocks are accessed at same level for reading and writing operations. The chance for overlapping exists.
Four-Way Interleaving: Four memory blocks are accessed at the same time.
Error-Correction Interleaving: Errors in communication systems occur in high volumes rather than in single attacks. Interleaving controls these errors with specific algorithms.
Latency is one disadvantage of interleaving. Interleaving takes time and hides all kinds of error structures, which are not efficient.

<u>Pipelining</u>

Introduction

A **Pipelining** is a series of stages, where some work is done at each stage in parallel. The stages are connected one to the next to form a pipe - instructions enter at one end, progress through the stages, and exit at the other end.

In computers, Pipelining is an implementation technique where multiple instructions are overlapped in execution.

The processor executes a program by fetching and executing instructions, one after the other. Let Fi and Ei refer to the fetch and execute steps for instruction Ii . Execution of a program consists of a sequence of fetch and execute steps, as shown in figure below.

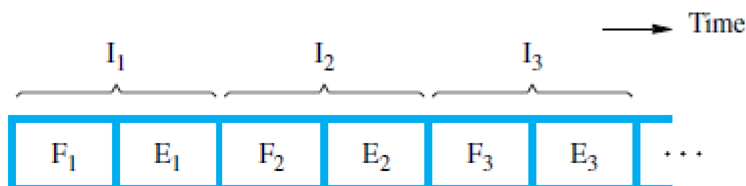


Figure (1)

Now consider a computer that has two separate hardware units, one for fetching instructions and another for executing them, as shown in figure below.

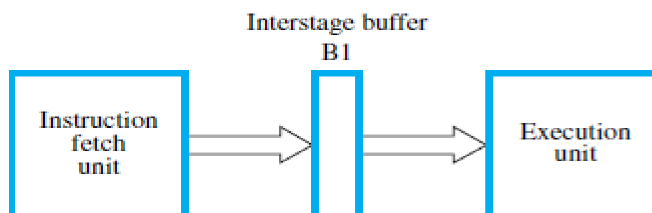


Figure (2)

The instruction fetched by the fetch unit is temporary saved in an intermediate storage buffer, B1. This buffer is needed to enable the execution unit to execute the instruction while the fetch unit is fetching the next instruction. The results of execution are saved in the destination location specified by the instruction. For the purposes of this discussion, we assume that both the source and the destination of the data operated on by the instructions are inside the block labeled "Execution unit."

The computer is controlled by a clock whose period is such that the fetch and execute steps of any instruction can each be completed in one clock cycle. Operation of the computer proceeds as in the figure below. In the first clock cycle, the fetch unit fetches an instruction I_1 (step F_1) and stores it in buffer B1 at the end of the clock cycle. In the second clock cycle, the instruction fetch unit proceeds with the fetch operation for instruction I_2 (step F_2). Meanwhile, the execution unit performs the operation specified by instruction I_1, which is available to it in buffer B1 (step E_1).

By the end of the second clock cycle, the execution of instruction I_1 is completed and instruction I_2 is available. Instruction I_2 is stored in B_1, replacing I_1, which is

no longer needed. Step E_2 is performed by the execution unit during the third clock cycle, while instruction I_3 is being fetched by the fetch unit. In this manner, both the fetch and execute units are kept busy all the time. If the pattern in the figure below can be sustained for a long time, the completion rate of instruction execution will be twice that achievable by the sequential operation as shown in figure (1).

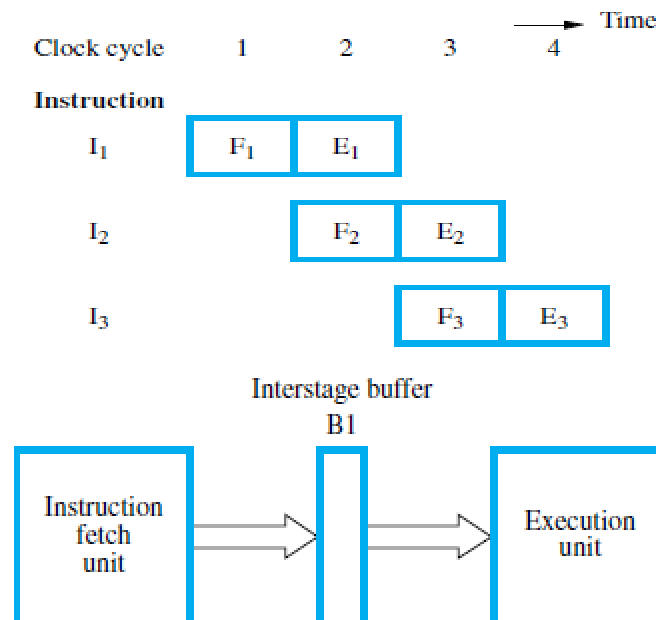


Figure (3)

As a summary:

The fetch and execute units in figure (2) constitute a two-stage pipeline in which each stage performs one step in processing an instruction. An inter stage storage buffer, B1, is needed to hold the information being passed from one stage to the next. New information is loaded into this buffer at the end of each clock cycle.

The processing of an instruction can be divided into 4 steps instead of two. For example, a pipelined processor may process each instruction in four steps, as follows:

IF – Instruction Fetch: read the instruction from the memory.
ID – Instruction Decode: decode the instruction and fetch the source operand(s).
IE – Instruction Execute: perform the operation specified by the instruction.
IW – Instruction Write: store the result in the destination location.

The sequence of events for this case is shown in the below figure

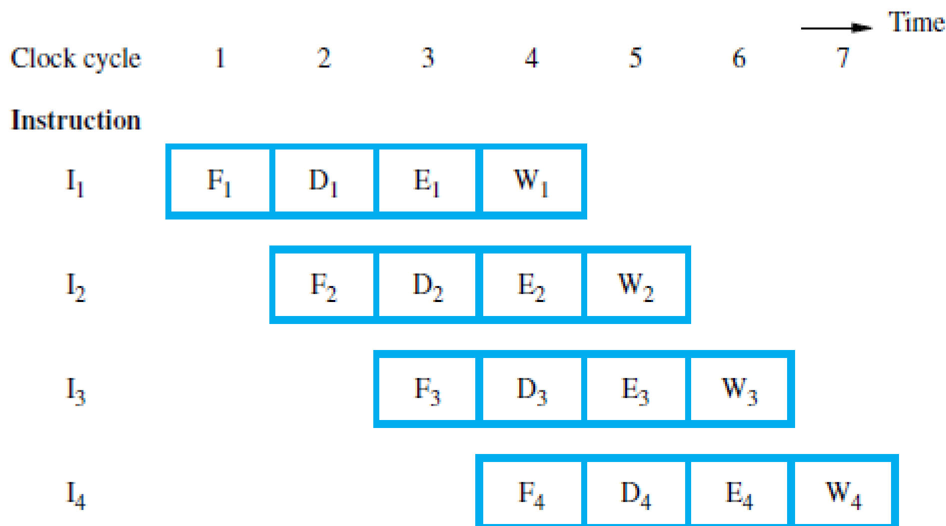
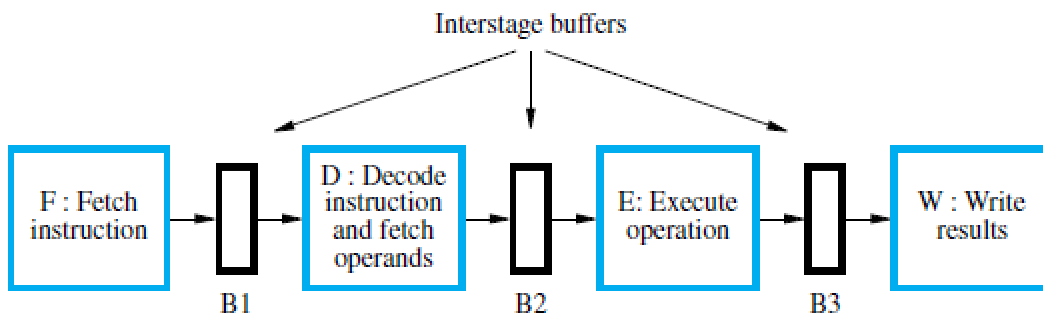


Figure (4)

Four instructions are in progress at any given time. This means that four distinct hardware units are needed, as shown in the figure below. These units must be capable of performing their tasks simultaneously and without interfering with one another. Information is passed from one unit to the next through a storage buffer.



As an instruction progresses through the pipeline, all the information needed by the stages downstream must be passed along. For example, during clock cycle 4, the information in the buffers is as follows:

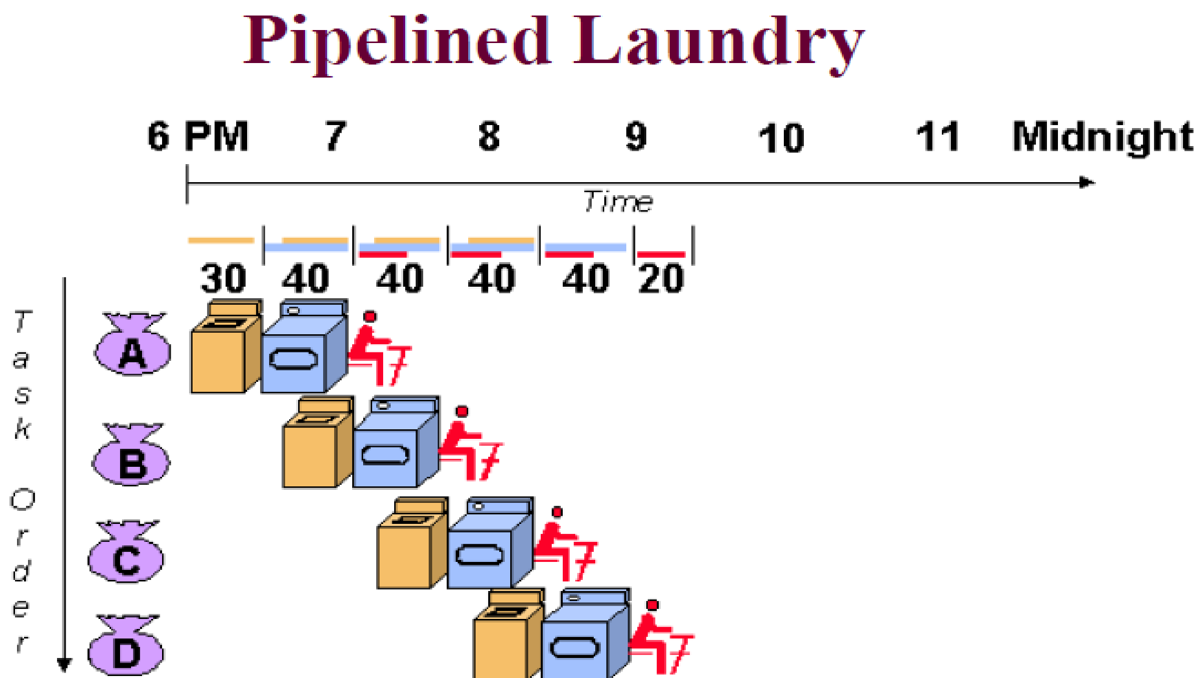
- Buffer B_1 holds instruction I_3, which was fetched in cycle 3 and is being decoded by the instruction-decoding unit.

- Buffer B_2 holds both the source operands for instruction I2 and the specification of the operation to be performed. This is the information

produced by the decoding hardware in cycle 3. The buffer also holds the information needed for the write step of instruction I_2 (stepW_2). Even though it is not needed by stage E, this information must be passed on to stage W in the following clock cycle to enable that stage to perform the required Write operation.

- Buffer B3 holds the results produced by the execution unit and the destination information for instruction I_1.

E.g. Pipelining Case: Laundry
• 4 loads of laundry that need to washed, dried, and folded.
– 30 minutes to wash, 40 min. to dry, and 20 min. to fold.
– We have 1 washer, 1 dryer, and 1 folding station. • What's the most efficient way to get the 4 loads of laundry done?



- Using this method, the laundry would be done at 9:30.

Instruction Pipeline

• **Instruction execution process lends itself naturally to pipelining**

– overlap the subtasks of instruction fetch, decode and execute

 Fetch instruction (FI)
 Decode instruction (DI)
 Calculate operands (CO)
 Fetch operands (FO)
 Execute instructions (EI)
 Write result (WR)

Overlap these operations

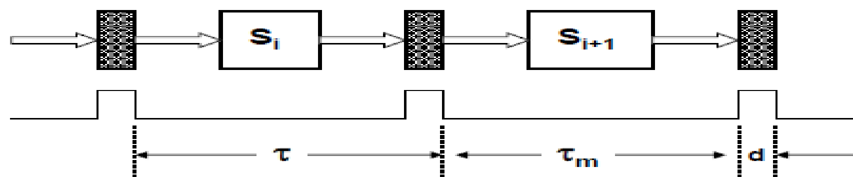
- Instructions Fetch: The IF stage is responsible for obtaining the requested instruction from memory. The instruction and the program counter are stored in the register as temporary storage.
- Decode instruction: The DI stage is responsible for decoding the instruction and sending out the various control lines to the other parts of the processor.
- Calculate Operands: The CO stage is where any calculations are performed. The main component in this stage is the ALU. The ALU is made up of arithmetic, logic and capabilities.
- Fetch Operands: and Execute Instruction. The FO and EI stages are responsible for storing and loading values to and from memory. They also responsible for input and output from the processor respectively.
- Write Operands: The WO stage is responsible for writing the result of a calculation, memory access or input into the register file.

Timing Diagram for Instruction Pipeline Operation

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

Pipeline Performance: Clock & Timing



Clock cycle of the pipeline : τ

Latch delay : *d*

$$\tau = \max \{\tau_m\} + d$$

Pipeline frequency : *f*

$$f = 1 / \tau$$

Types of Pipeline

It is divided into 2 categories:

1. Arithmetic Pipeline
2. Instruction Pipeline

1. 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.
5. Registers are used for storing the intermediate results between the above operations.

2. 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

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

Disadvantages of Pipelining

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

Pipeline Hazards

There are situations, called hazards that prevent the next instruction in the pipeline stages from executing during its designated clock cycle. Hazards reduce the performance from the ideal speedup gained by pipelining. There are three classes of hazards:

- **Structural hazards**
  Arise from resource conflicts when the hardware cannot support all the requests of overlapped execution of instructions simultaneously.

- **Data hazards**
  Arise when an instruction depends on the results of a previous instruction by the overlapping of instructions in the pipeline.

- **Control hazards**
  Arise from the pipelining of branches and other instructions that change the Program Counter (PC).

Hazards in pipelines will stall the pipeline. Avoiding a hazard requires that some of the instruction in the pipeline to proceed execution while others are delayed.
In another word, when an instruction is stalled, all instructions issued Later than the stalled instruction are also stalled. Instructions issued earlier than the stalled instruction must continue, since otherwise the hazard will never clear as it will propagate from one stage to another. As a result, no new instructions are fetched during the stall.

1	2	3	4	5	6	7	8	9	10
IF	ID	EX	MEM	WB					
	IF	ID	EX	MEM	WB				
		IF	ID	EX	MEM	WB			
			stall	IF	ID	EX	MEM	WB	
					IF	ID	EX	MEM	WB
						IF	ID	EX	MEM
							IF	ID	EX

Structural Hazards

When a processor is pipelined, the overlapped execution of instructions requires pipelining of hardware units and duplication of resources to allow all possible combinations of instructions in the pipeline. If some combination of instructions cannot be accommodated because of resource conflicts, the processor is said to have a structural hazard.

The most common cases of structural hazards arise when:

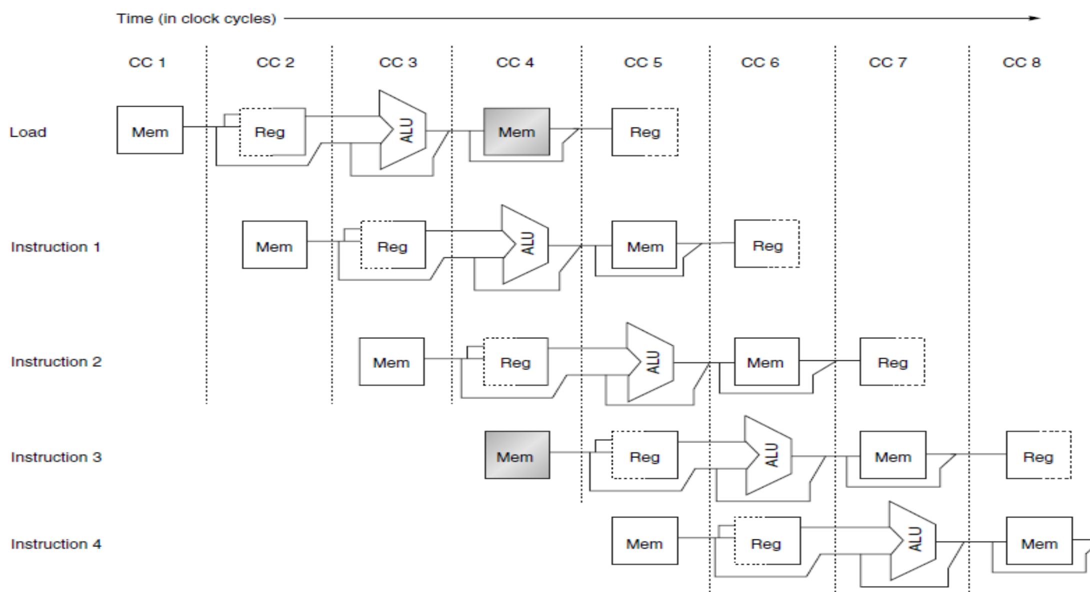
- Case 1: Some functional unit such as a printer or other hardware is not fully pipelined. Then a sequence of instructions using that unpipelined unit cannot proceed at the rate of one per clock cycle.

- Case 2: Some resource has not been duplicated enough to allow all combinations of instructions in the pipeline to execute. For example, a processor may have only one register-file write port, but under certain circumstances, the pipeline might want to perform two writes in a clock cycle. This will generate a structural hazard.

When a sequence of instructions encounters this hazard, the pipeline will stall one of the instructions until the required unit is available. Such stalls will increase the CPI (Clock Cycles per Instruction) from its usual ideal value of 1.

Example:

Some pipelined processors have shared a single-memory pipeline for data and instructions. As a result, when an instruction contains a data memory reference, it will conflict with the instruction reference for a later instruction, as shown in the figure below.



There are two solutions for this problem:

- To resolve this hazard, we stall the pipeline for 1 clock cycle when the data memory access occurs. A stall is commonly called a pipeline Bubble or just bubble, since it floats through the pipeline taking space but carrying no useful work.

- As an alternative solution to this structural hazard, the designer could provide a separate memory access for instructions, either by splitting the cache into separate instruction and data caches, or by using a set of buffers, usually called instruction buffers, to hold instructions.

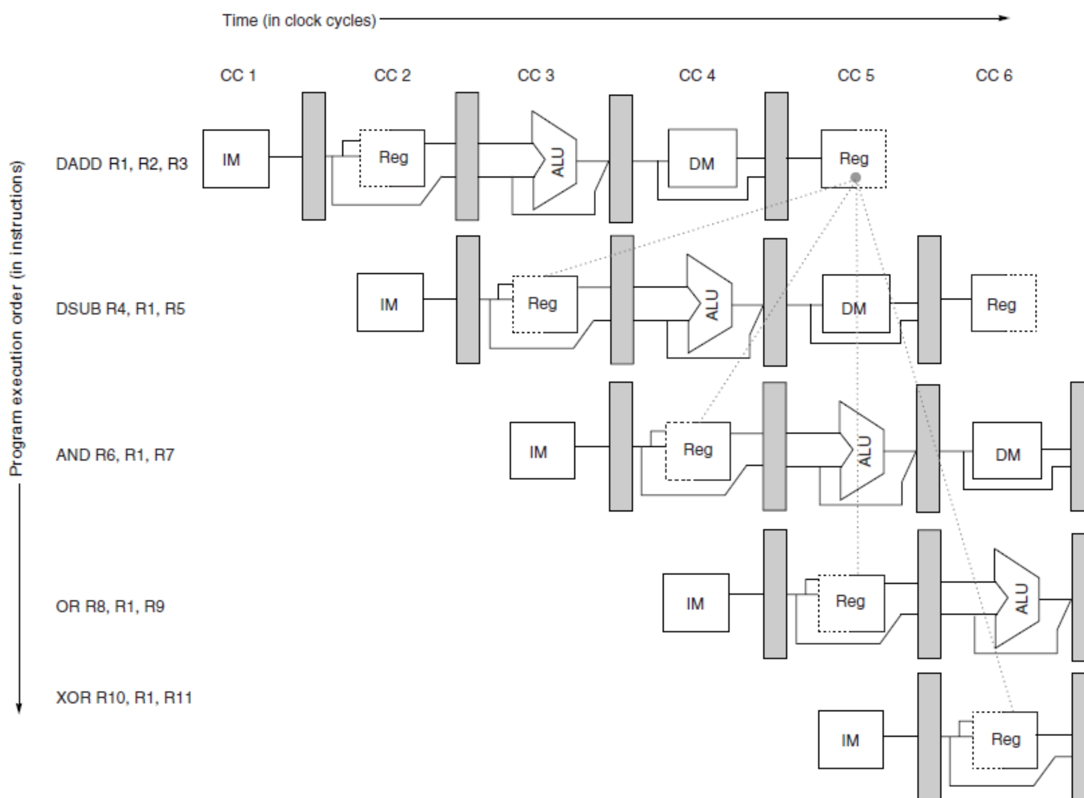
	Clock cycle number									
Instruction	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Load instruction	IF	ID	EX	MEM	WB					
Instruction $i + 1$		IF	ID	EX	MEM	WB				
Instruction $i + 2$			IF	ID	EX	MEM	WB			
Instruction $i + 3$				stall	IF	ID	EX	MEM	WB	
Instruction $i + 4$						IF	ID	EX	MEM	WB
Instruction $i + 5$							IF	ID	EX	MEM
Instruction $i + 6$								IF	ID	EX

Data Hazards

Data hazards occur when the pipeline changes the order of read/write accesses to operands so that the order differs from the order seen by sequentially executing of instructions on an unpipelined processor. Consider the pipelined execution of these instructions:

ADD	R1,R2
SUB	R4,R1
AND	R6,R1
OR	R8,R1
XOR	R10,R1

As you can see all the instructions after the ADD use the result of the ADD instruction which is R1. As shown in the figure below, the ADD instruction writes the value of R1 in the WB pipe stage, but the SUB instruction reads the value during its ID stage. This problem is called a data hazard.

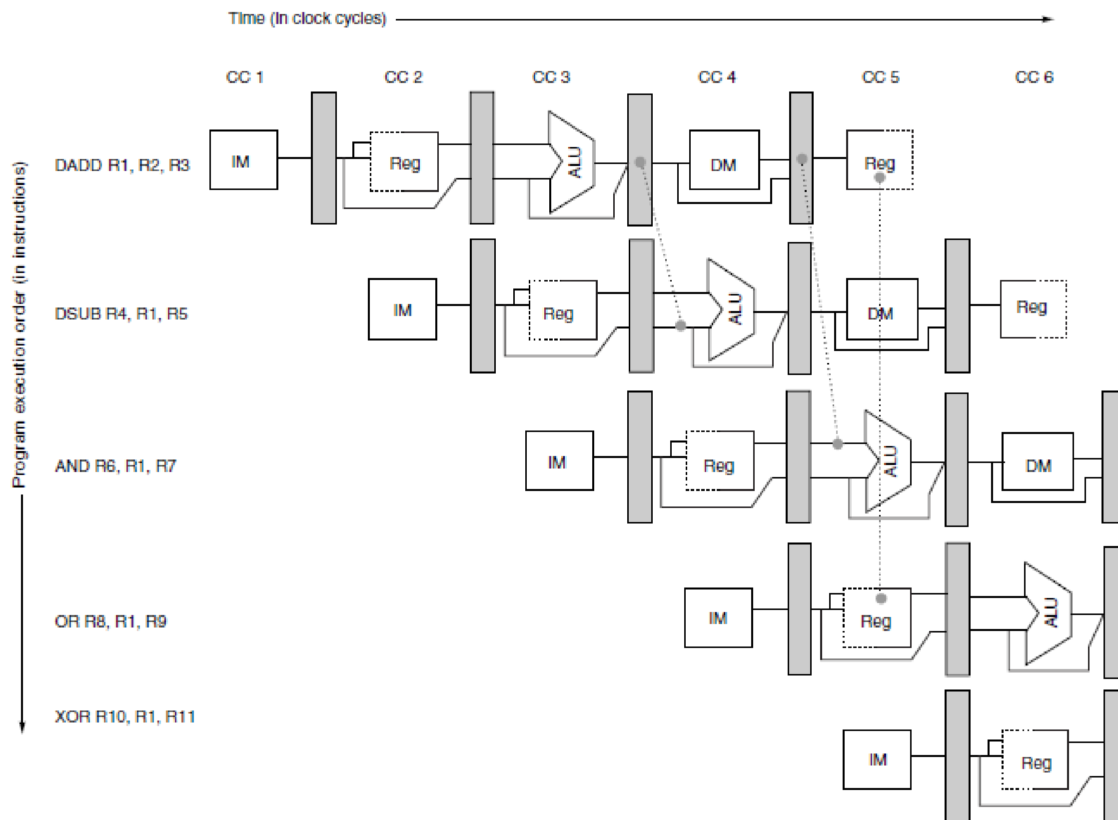


Minimizing Data Hazard Stalls by Forwarding

The problem in the figure above can be solved with a simple hardware technique called forwarding (also called bypassing). The key insight in forwarding is that the result is not really needed by the SUB until after the ADD actually produces it. If the result can be moved from the pipeline register where the ADD stores it to where the SUB needs it, then the need for a stall can be avoided. Using this observation, forwarding works as follows:

1. The ALU result from both the EX/MEM and MEM/WB pipeline registers is always fed back to the ALU inputs.

2. If the forwarding hardware detects that the previous ALU operation has written the same register value of the current ALU operation, the forwarding hardware selects the forwarded result as the ALU input and neglect the register value.



Vector (Array) Processing and Superscalar Processors

A Scalar processor is a normal processor, which works on simple instruction at a time, which operates on single data items. But in today's world, this technique will prove to be highly inefficient, as the overall processing of instructions will be very slow.

What is 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 instruction, 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

Computer with vector processing capabilities are in demand in specialized applications. The following are some areas where vector processing is used:
Petroleum exploration, Medical diagnosis ,Data analysis ,Weather forecasting.
Aerodynamics and space flight simulations , Image processing ,Artificial intelligence.

Superscalar Processors

It was first invented in 1987. It is a machine which is designed to improve the performance of the scalar processor. In most applications, most of the operations are on scalar quantities. Superscalar approach produces the high performance general purpose processors.
The main principle of superscalar approach is that it executes instructions independently in different pipelines. As we already know, that Instruction pipelining leads to parallel processing thereby speeding up the processing of instructions. In Superscalar processor, multiple such pipelines are introduced for different operations, which further improves parallel processing.
There are multiple functional units each of which is implemented as a pipeline. Each pipeline consists of multiple stages to handle multiple instructions at a time which support parallel execution of instructions.
It increases the throughput because the CPU can execute multiple instructions per clock cycle. Thus, superscalar processors are much faster than scalar processors.
A scalar processor works on one or two data items, while the vector processor works with multiple data items. A superscalar processor is a combination of both. Each instruction processes one data item, but there are multiple execution units within each CPU thus multiple instructions can be processing separate data items concurrently. While a superscalar CPU is also pipelined, there are two different performance enhancement techniques. It is possible to have a non-pipelined superscalar CPU or pipelined non-superscalar CPU. The superscalar technique is associated with some characteristics, these are:
Instructions are issued from a sequential instruction stream. CPU must dynamically check for data dependencies. Should accept multiple instructions per clock cycle.

Vector(Array) Processor and its Types

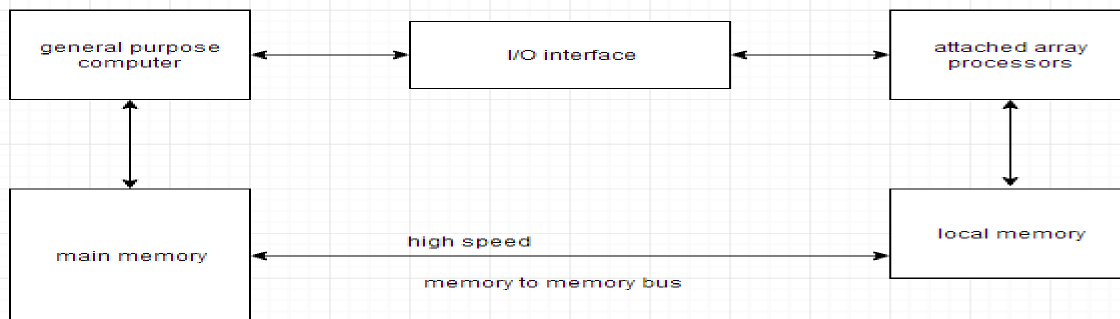
Array processors are also known as multiprocessors or vector processors. They perform computations on large arrays of data. Thus, they are used to improve the performance of the computer.

Types of Array Processors

There are basically two types of array processors:
Attached Array Processors
SIMD Array Processors

Attached Array Processors

An attached array processor is a processor which is attached to a general purpose computer and its purpose is to enhance and improve the performance of that computer in numerical computational tasks. It achieves high performance by means of parallel processing with multiple functional units.

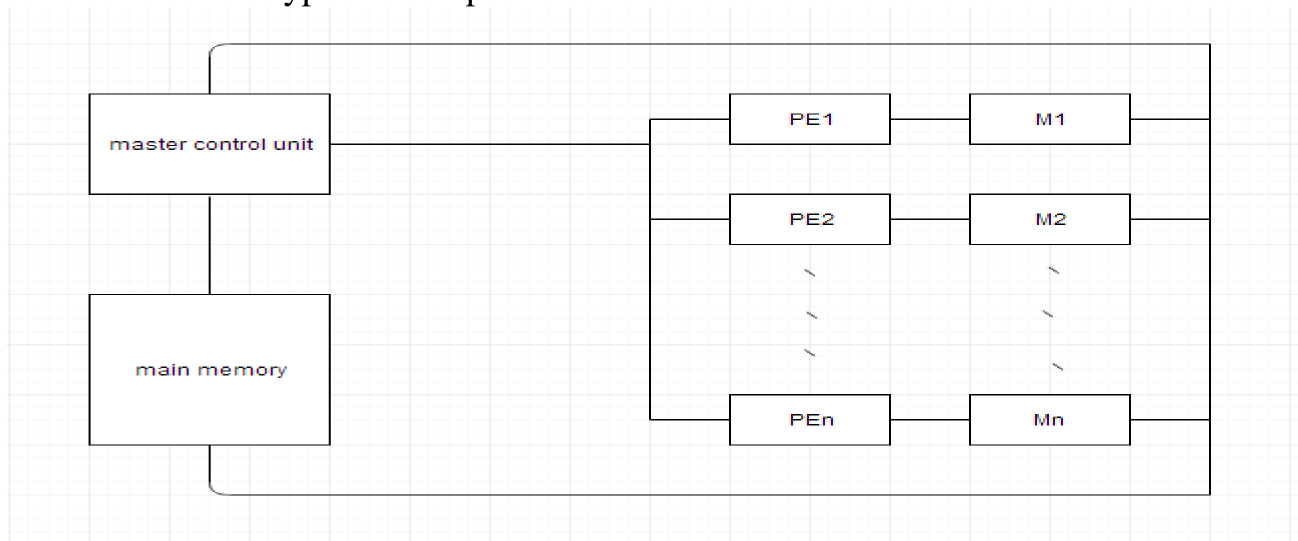


SIMD Array Processors

SIMD is the organization of a single computer containing multiple processors operating in parallel. The processing units are made to operate under the control of a common control unit, thus providing a single instruction stream and multiple data streams.

A general block diagram of an array processor is shown below. It contains a set of identical processing elements (PE's), each of which is having a local memory M. Each processor element includes an **ALU** and **registers**. The master control unit controls all the operations of the processor elements. It also decodes the instructions and determines how the instruction is to be executed.

The main memory is used for storing the program. The control unit is responsible for fetching the instructions. Vector instructions are sending to all PE's simultaneously and results are returned to the memory.
SIMD processors are highly specialized computers. They are only suitable for numerical problems that can be expressed in vector or matrix form and they are not suitable for other types of computations.



Why use the Array Processor

Array processors increase the overall instruction processing speed.
As most of the Array processors operate asynchronously from the host CPU, hence it improves the overall capacity of the system. Array Processors has its own local memory, hence providing extra memory for systems with low memory.