Computer System Architecture

Research · August 2015

DOI: 10.13140/RG.2.1.4289.2002

1 author:

Elmustafa sayed ali ahmed
Red Sea University
22 PUBLICATIONS **60** CITATIONS

SEE PROFILE

Computer System Architecture

MEMORY SYSTEM, INSTRUCTION PIPELINING , AND PARALLEL COMPUTATION

Computer System Architecture

Edited By: Elmustafa Sayed Ali Ahmed

Red Sea University

Contents

Chapter 1: Introduction
Basic computer architecture and organization, Basic functions of a computer and its main components, The von Neumann architecture.

Chapter 2: The Memory System
Memory hierarchy, Cache memories, Virtual memories, Memory management and memory system.

Chapter 3: Instruction Pipelining
Organization of pipelined units.

Chapter 4: RISC and CISC Architectures
An analysis of instruction execution for code generated from high-level language programs, Compiling for RISC architectures, Main characteristics of RISC architectures, RISC-CISC trade-offs.

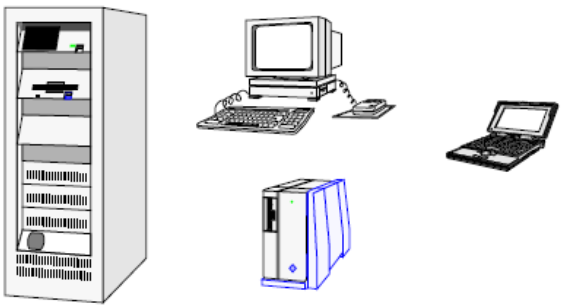
Chapter 5: Superscalar Architectures
Instruction level parallelism and machine parallelism, Hardware techniques for performance enhancement, Data dependencies, Policies for parallel instruction execution,

Chapter 6: Architectures for Parallel Computation
Parallel programs, Performance of parallel computers, a classification of computer architectures, Array processors, Multiprocessors, Multicomputer, Vector processors.

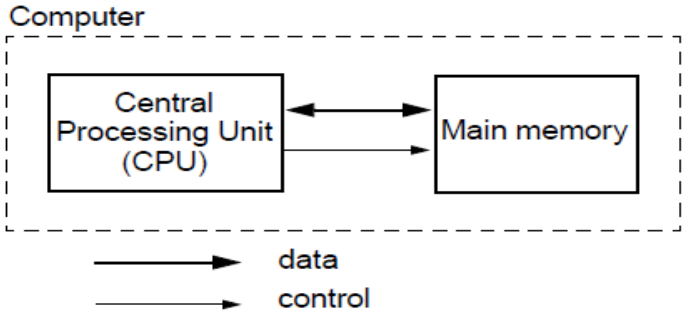
Chapter 1
Introduction

- ✓ What is a Computer/Computer System?
- ✓ The von Neumann Architecture.
- ✓ Representation of Data and Instructions.
- ✓ Instruction Execution.
- ✓ The Control Unit.
- ✓ The Computer System.
- ✓ Main and Secondary Memory.
- ✓ Input - Output Devices.

What is a computer?

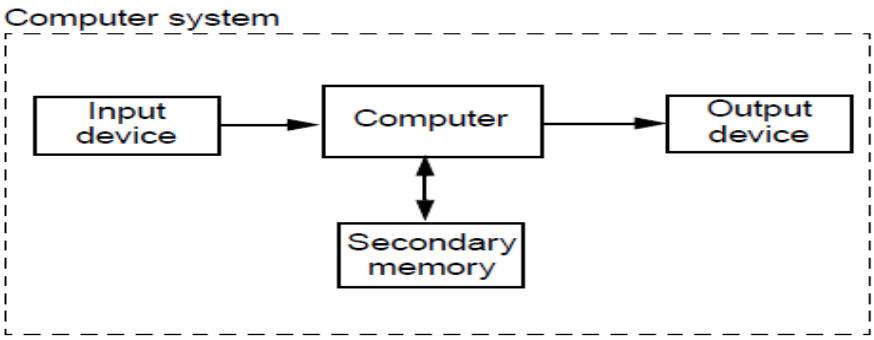


A computer is a data processing machine which is operated automatically under the control of a list of instructions (called a program) stored in its main memory.



What is a computer system?

- A computer system consists usually of a computer and its peripherals.
- Computer peripherals include input devices, output devices, and secondary memories.



The von Neumann Architecture

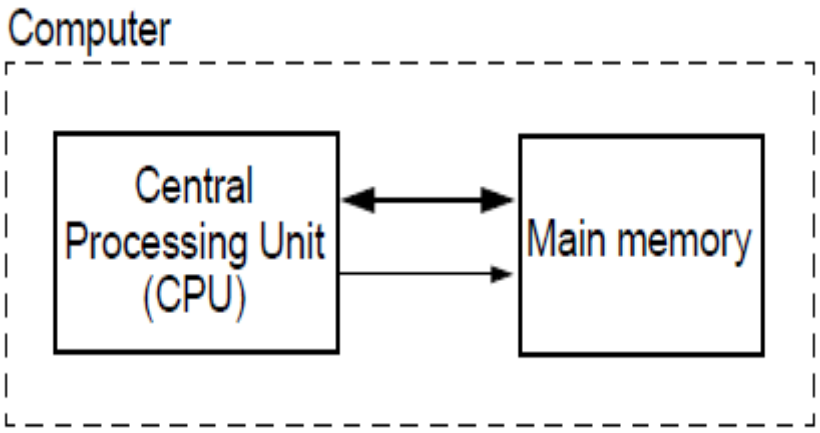
von Neumann computers are general purpose computers. They can solve very different problems depending on the program they got to execute!

The principles:

- Data and instructions are both stored in the main memory.
- The content of the memory is addressable by location (without regard to what is stored in that location).
- Instructions are executed sequentially (from one instruction to the next, in order of their location in memory) unless the order is explicitly modified.

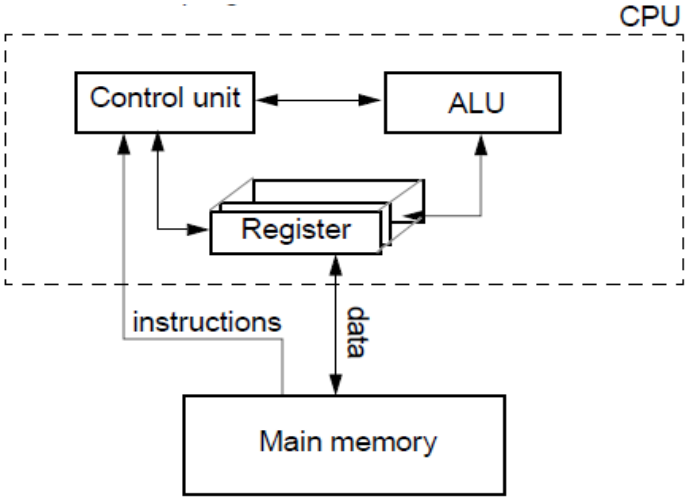
The organization (architecture) of the computer:

- a central processing unit (CPU); it contains the control unit (CU), that coordinates the execution of instructions and the arithmetic/logic unit (ALU) which performs arithmetic and logic operations;
- (main) memory.



General-purpose (von Neumann) Architectures

In the von-Neumann architecture, a small set of circuits can be driven to perform very different tasks, depending on the software program which is executed.



- The primary function of a CPU is to execute the instructions fetched from the main memory.
- An instruction tells the CPU to perform one of its basic operations (an arithmetic or logic operation, to transfer a data from/to main memory, etc.).

- The Control Unit is the one which interprets (decodes) the instruction to be executed and which "tells" the different other components what to do.
- The CPU includes a set of registers which are temporary storage devices typically used to hold intensively used data and intermediate results.

Representation of Data

Inside a computer, data and control information (instructions) are all represented in binary format which uses only two basic symbols: "0" and "1".

- The two basic symbols are represented by electronics signals.
- Numeric data are represented using the binary system, in which the positional values are powers of 2:
-

$$100101 = 1*20 + 0*21 + 1*22 + 0*23 + 0*24 + 1*25$$
$$10110 = 0*20 + 1*21 + 1*22 + 0*23 + 1*24$$

- Binary numbers are added, subtracted, multiplied and divided (by the ALU) directly; it is not needed to convert them to decimal numbers first.

$$100101 + 10110 = 111011$$

Machine Instructions

- A CPU can only execute machine instructions;
- Each computer has a set of specific machine instructions which its CPU is able to recognize and execute.
- A machine instruction is represented as a sequence of bits (binary digits).

These bits have to define:
- What has to be done (the operation code)
- To whom the operation applies (source operands)
- Where does the result go (destination operand)

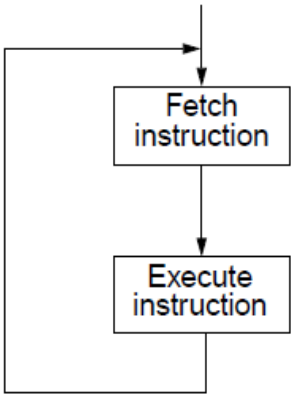
- How to continue after the operation is finished

Type of Machine Instructions

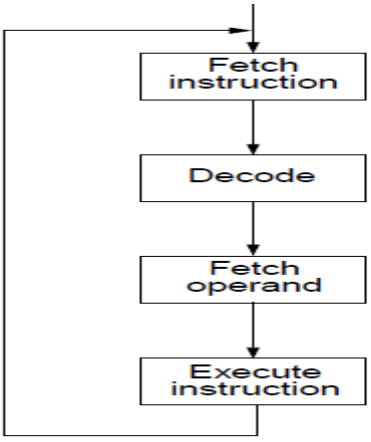
Machine instructions are of four types:
- Data transfer between memory and CPU registers
- Arithmetic and logic operations
- Program control (test and branch)
- I/O transfer

The Instruction Cycle

Each instruction is performed as a sequence of steps; the steps corresponding to one instruction are referred together as an instruction cycle. A simple and refined view of the instruction cycle



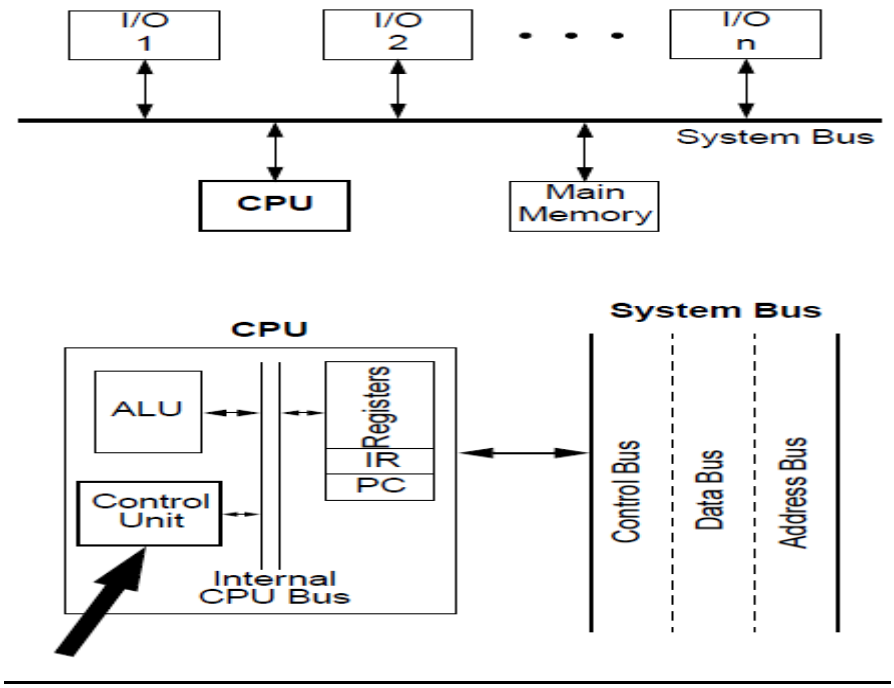
- Simple -



- Refined -

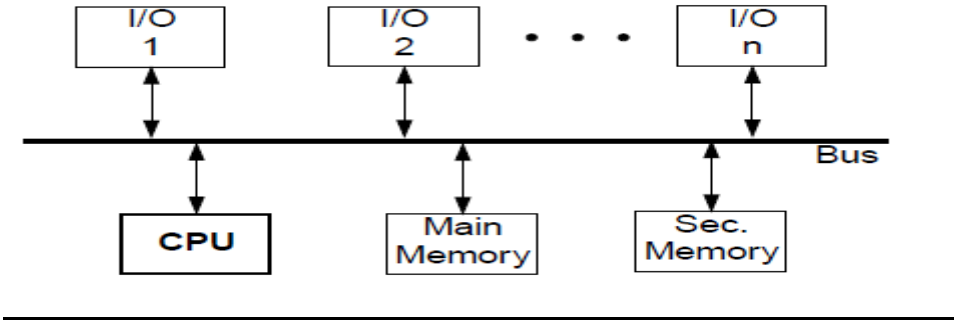
The Control Unit

The task of the Control Unit is to perform the control by question how are the elements inside the CPU and the interface to the external data path controlled (synchronized) in order to work properly?



The Computer System

- CPU + main memory constitute the "core" of the computer system.
- Secondary memory + I/O devices are the so called peripherals.
- Communication between different components of the system is usually performed using one or several buses.



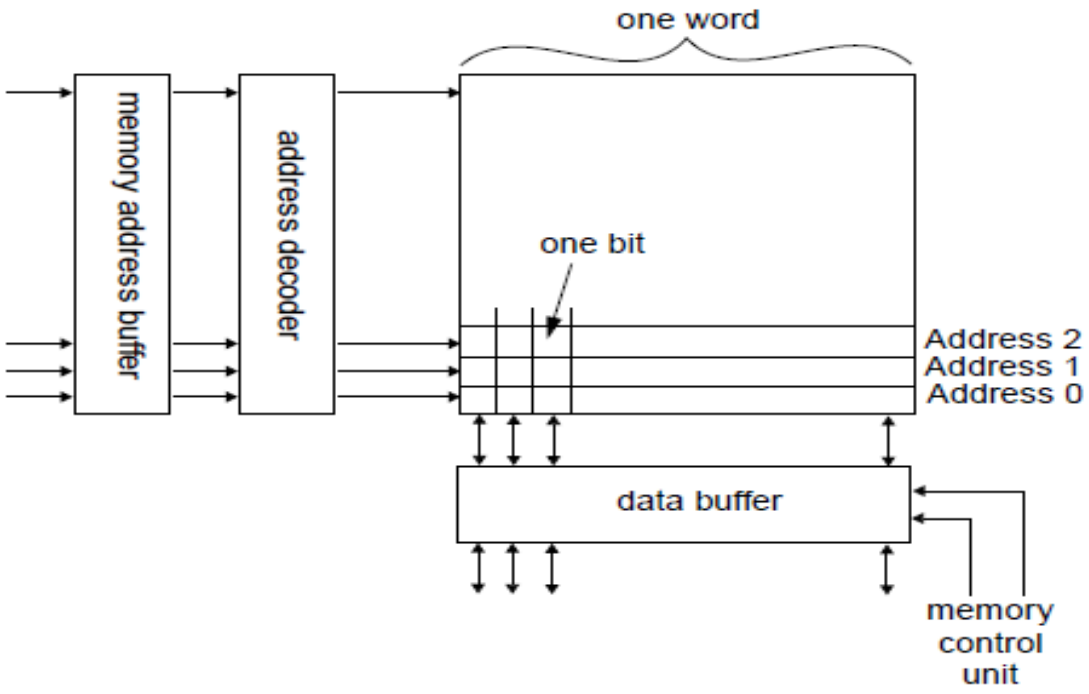
Memories

- The main memory is used to store the program and data which are currently manipulated by the CPU.
- The secondary memory provides the long-term storage of large amounts of data and program.
- Before the data and program in the secondary memory can be manipulated by the CPU, they must first be loaded into the main memory.
- The most important characteristics of a memory is its speed, size, and cost, which are mainly constrained by the technology used for its implementation.
- Typically:

The main memory is fast and of limited size;
Secondary memory is relatively slow and of very large size.

The Main Memory

- The main memory can be viewed as a set of storage cells, each of which can be used to store a word.
- Each cell is assigned a unique address and the addresses are numbered sequentially: 0,1,2,... .
- Besides the storage cells, there are a memory address buffer (storing the address of the word to be read/written) and a data buffer (storing the data read/to be written), the address decoder and a memory control unit.
- The most widely used technology to implement main memories is semiconductor memories.
- The most common semiconductor memory type is random access memory (RAM).
- The information stored in a RAM semiconductor memory will be lost when electrical power is removed.



Secondary Memory

Hard Disk:

- Data are recorded on the surface of a hard disk made of metal coated with magnetic material.
- The disks and the drive are usually built together and encased in an air tight container to protect the disks from pollutants such as smoke particle and dust. Several disks are usually stacked on a common drive shaft with each disk having its own read/write head.

- Main features:

- Direct access
- Fast access:

Seek time » 10 ms
Data transfer rate » 5 MB/s

- Large storage capacity (8MB - several GB)

Diskette:

- Data are recorded on the surface of a floppy disk made of polyester coated with magnetic material.
- A special diskette drive must be used to access data stored in the floppy disk. It works much like a record turntable of gramophone.
- Main features:
  - Direct access
  - Cheap
  - Portable, convenient to use
- Main standards:

5 1/4-inch. Capacity » 360 KB/disk
3 1/2-inch. Capacity » 1.44 MB/disk
(about 700 pages of A4 text)

Magnetic tape:

- Magnetic tape is made up from a layer of plastic which is coated with iron oxide. The oxide can be magnetized in different directions to represent data.
- Its operation uses a similar principle as in the case of a tape recorder.
- Main features:

- Sequential access (access time about 1-5 s).
- High value of storage (50 MB/tape).
- Inexpensive.

- It is often used for backup or archive purpose.

Optical Memory:

- CD-ROM (Compact Disk ROM): The disk surface is imprinted with microscopic holes which record digital information. When a low-powered laser beam shines on the surface, the intensity of the reflected light changes as it encounters a hole. The change is detected by a photo sensor and converted into a digital signal. with properties :

- Huge capacity: 775 MB/disk (»550 diskettes).
- Inexpensive replication, cheap production.
- Removable.
- Read-only.
- Long access time (could be half a second).

- WORM (Write-Once Read-Many) CD: A laser beam of modest intensity equipped in the disk drive is used to imprint the whole pattern. Good for archival storage by providing a permanent record of large volumes of data.

- Erasable Optical Disk: combination of laser technology and magnetic surface technique.

- Can be repeatedly written and overwritten
- High reliability and longer life than magnetic disks.

Input-Output Devices

- Input and output devices provide a means for people to make use of a computer.
- Some I/O devices function also as an interface between a computer system and other physical systems. Such interface usually consists of A/D and D/A converters.

Typical Input Devices

Device	Main features	Advantages	Disadvantages
Keyboard	Like a typewriter	Efficient for inputting text	Relatively slow, speed depends on operator
Light pen	Point at screen	Easy to use	Needs much software to make it versatile
Mouse	Move around on desk	Efficient for icon-based input, and menu selection	Needs much software support
Joystick	Used for games and control	As above Fast	Needs much software support
Graphics tablet	Graphics input	Input picture and freehand sketch	Slow
Scanner	Copy pictures	Fast input of graphics	Bit-mapped graphics only
Voice input	User friendly	No hands needed	Limited vocabulary, Speech recognition software needed

Typical Output Devices

Device	Main features	Advantages	Disadvantages	Speed
Display Screen	Most versatile, both text and graphics	No waste of paper etc.	No hard copy	
Line printer	Impact printer, Very fast.	Can cope with high volume	Large versions are very noisy	up to 6000 cps
Dot matrix printer	Versatile text and graphics	Inexpensive	low quality and speed	up to 200 cps
Inkjet printer	Mechanically similar to above; dot produced by ejected ink droplet	small size; inexpensive	lower quality then laser printers	~20 line/sec
Laser printer	High quality text and graphics	Very fast, high volume	(used to be) expensive	20 000 line/min possible
Plotter	High quality graphics	large graphics output possible	Large machine, expensive	Pen up to 1 meter/s
Voice output	Natural for certain applications	Don't need to use eyes	Limited range of sounds	Normal speech

Summary

- Computer = CPU + Main Memory
- Computer System = Computer + Peripherals
- The CPU executes instructions stored together with data in the main memory.
- Von Neumann computers are general-purpose, programmable computers.
- Data and instructions are represented in binary format.
- Machine instructions are specific to each computer and are organized according to a certain instruction format.
- An instruction is performed as a sequence of steps; this is the instruction cycle.
- The currently manipulated program and data are stored in the main memory. This is organized as a set of storage cells each one having a unique address.
- Secondary memory can be a hard disk, diskette, magnetic tape or an optical device.
- Input-output devices provide a means for people to exchange information with the computer.

➢ Home work:

✓ Write the volatile and non volatile memories.

Chapter 2
The Memory System

- ✓ Components of the Memory System.
- ✓ The Memory Hierarchy.
- ✓ Cache Memories.
- ✓ Cache Organization.
- ✓ Replacement Algorithms.
- ✓ Write Strategies.
- ✓ Virtual Memory.

Components of the Memory System

- Main memory: fast, random access, expensive, located close (but not inside) the CPU. Is used to store program and data which are currently manipulated by the CPU.
- Secondary memory: slow, cheap, direct access, located remotely from the CPU.

Problems with the Memory System

- What do we need?
  - ❖ We need memory to fit very large programs and to work at a Speed comparable to that of the microprocessors.
- Main problem:
  - ❖ Microprocessors are working at a very high rate and they need large memories;
  - ❖ Memories are much slower than microprocessors;
- Facts:
  - ❖ The larger a memory, the slower it is;
  - ❖ The faster the memory, the greater the cost/bit.

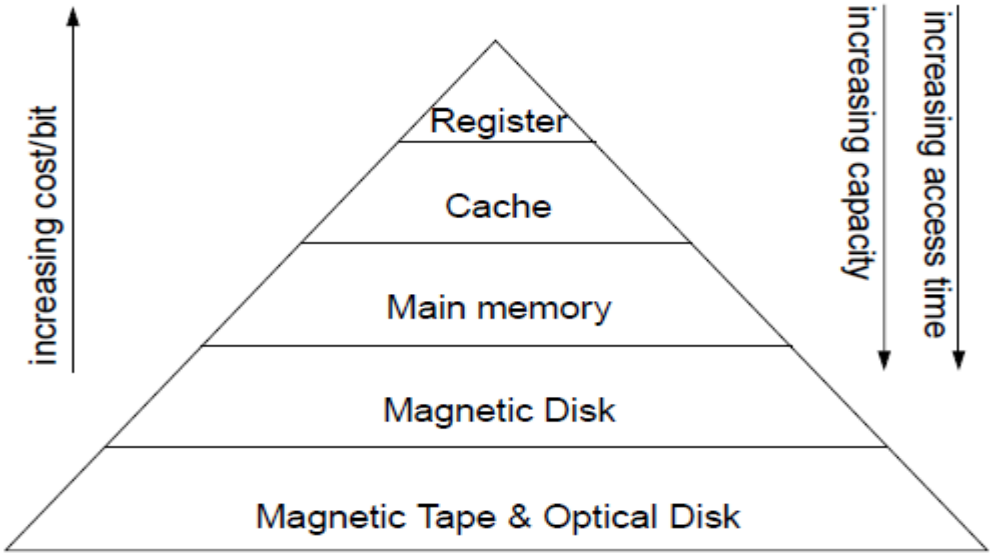
A Solution

- It is possible to build a composite memory system which combines a small, fast memory and a large slow main memory and which behaves (most of the time) like a large fast memory.
- The two level principles above can be extended into a hierarchy of many levels including the secondary memory (disk store).
- The effectiveness of such a memory hierarchy is based on property of programs called the principle of locality .

The Memory Hierarchy

The key to the success of a memory hierarchy is if data and instructions can be distributed across the memory so that most of the time they are available, when needed, on the top levels of the hierarchy.

- The data which is held in the registers is under the direct control of the compiler or of the assembler programmer.
- The contents of the other levels of the hierarchy are managed automatically:
- Migration of data/instructions to and from caches is performed under hardware control.
- Migration between main memory and backup store is controlled by the operating system (with hardware support).

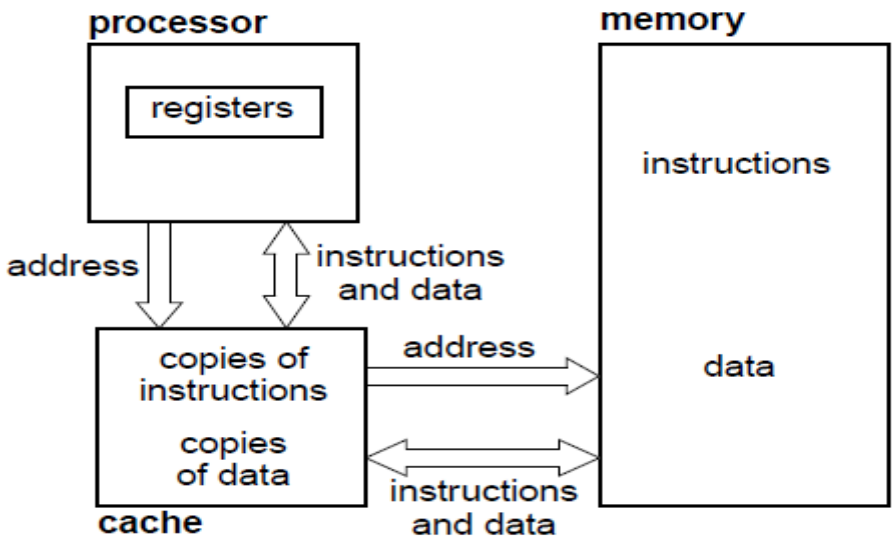


Some typical characteristics

1. Processor registers:
   - 32 registers of 32 bits each = 128 bytes
   - access time = few nanoseconds
2. On-chip cache memory:
   - capacity = 8 to 32 Kbytes
   - access time = ~10 nanoseconds
3. Off-chip cache memory:
   - capacity = few hundred Kbytes
   - access time = tens of nanoseconds
4. Main memory:
   - capacity = tens of Mbytes
   - access time = ~100 nanoseconds
5. Hard disk:
   - capacity = few G bytes
   - access time = tens of milliseconds

Cache Memory

A cache memory is a small, very fast memory that retains copies of recently used information from main memory. It operates transparently to the programmer, automatically deciding which values to keep and which to overwrite.



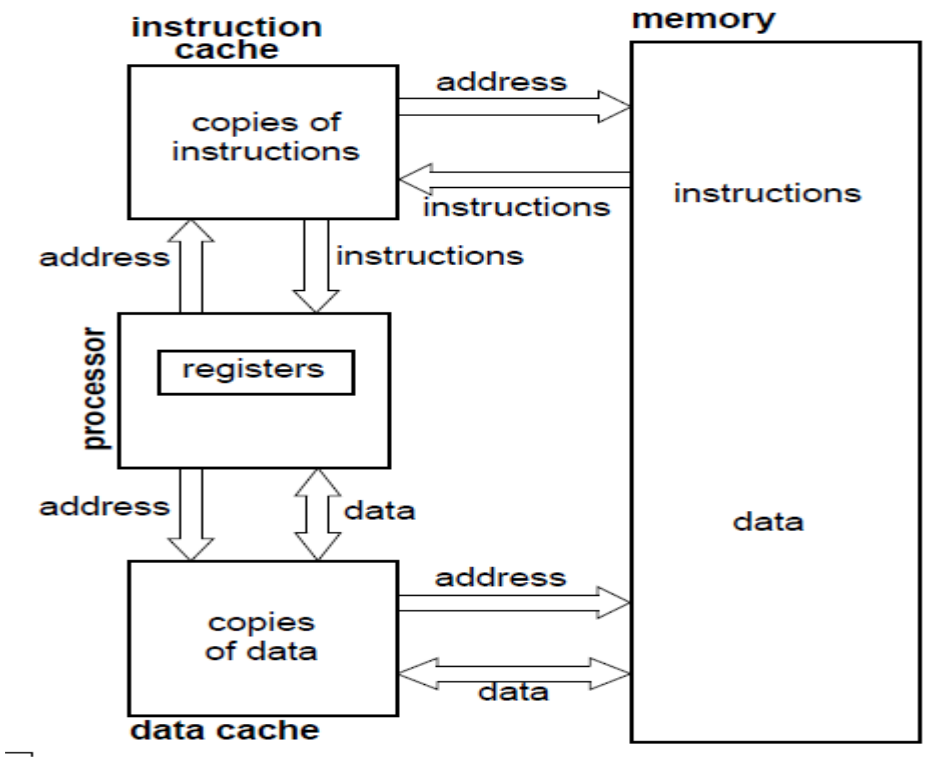
- The processor operates at its high clock rate only when the memory items it requires are held in the cache. The overall system performance depends strongly on the proportion of the memory accesses which can be satisfied by the cache

- An access to an item which is in the cache: hit
- An access to an item which is not in the cache: miss.
- The proportion of all memory accesses that are satisfied by the cache: hit rate.
- The proportion of all memory accesses that are not satisfied by the cache: miss rate.

- Cache space (~K Bytes) is much smaller than main memory (~M Bytes);

Separate Data and Instruction Caches

- The last consideration shows architecture with a unified instruction and data cache.
- It is common also to split the cache into one dedicated to instructions and one dedicated to data as shown in figure below;



Advantages of unified caches:

- they are able to better balance the load between instruction and data fetches depending on the dynamics of the program execution.
- Design and implementation are cheaper.

Advantages of split caches (Harvard Architectures):

- Competition for the cache between instruction processing and execution units is eliminated instruction fetch can proceed in parallel with memory access from the execution unit.

Cache Organization

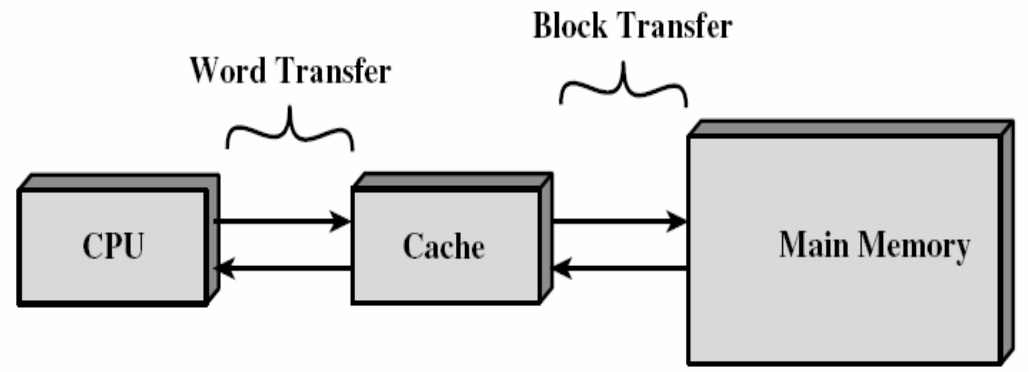
Example:

- a cache of 64 Kbytes.
- data transfer between cache and main memory is in blocks of 4 bytes; we say the cache is organized in lines of 4 bytes;
- a main memory of 16 Mbytes; each byte is addressable by a 24-bit address (2^24=16M)
- The cache consists of 2^14 (16K) lines.
- The main memory consists of 2^22 (4M) blocks.

Cache Memory Principles

Cache memory features:

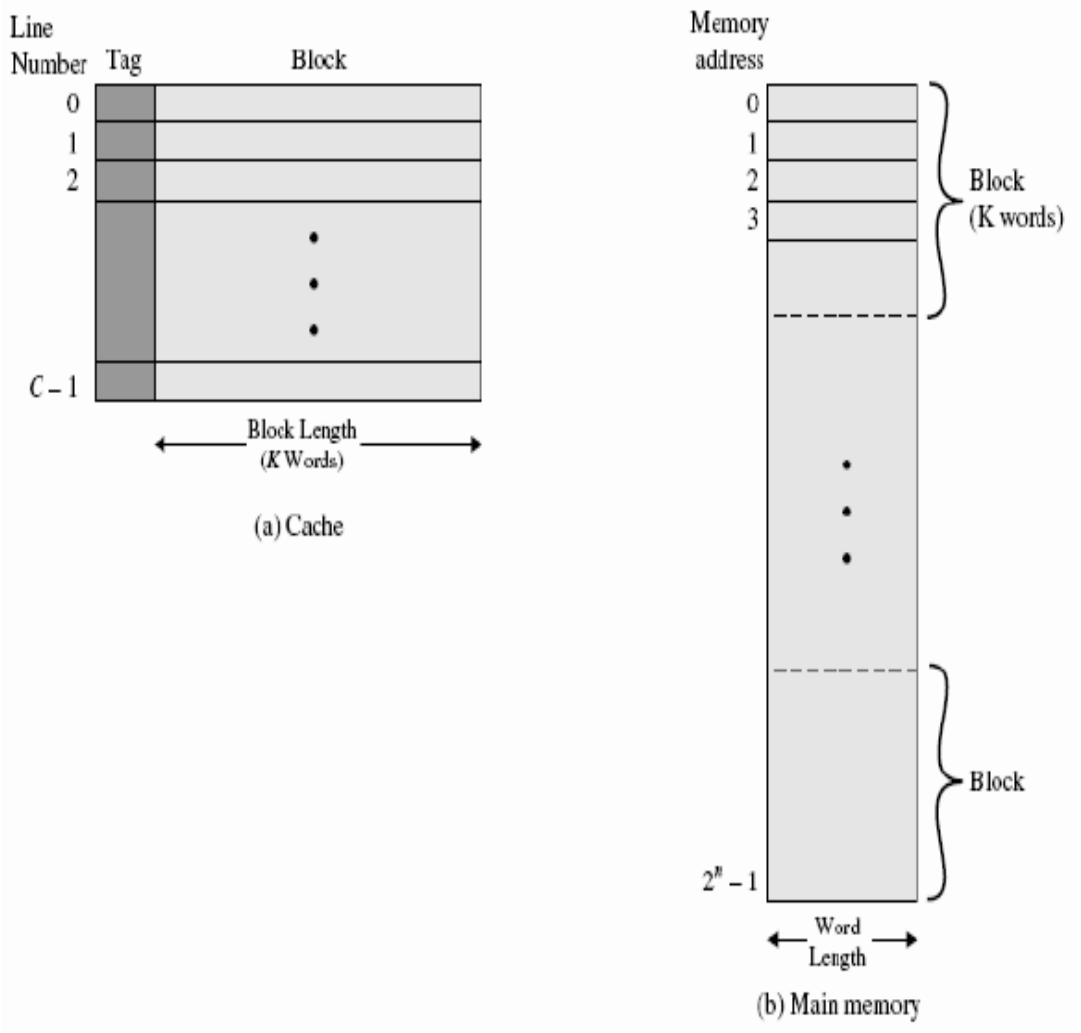
- Small amount of fast memory
- Placed between the processor and main memory
- Located either on the processor chip or on a separate module

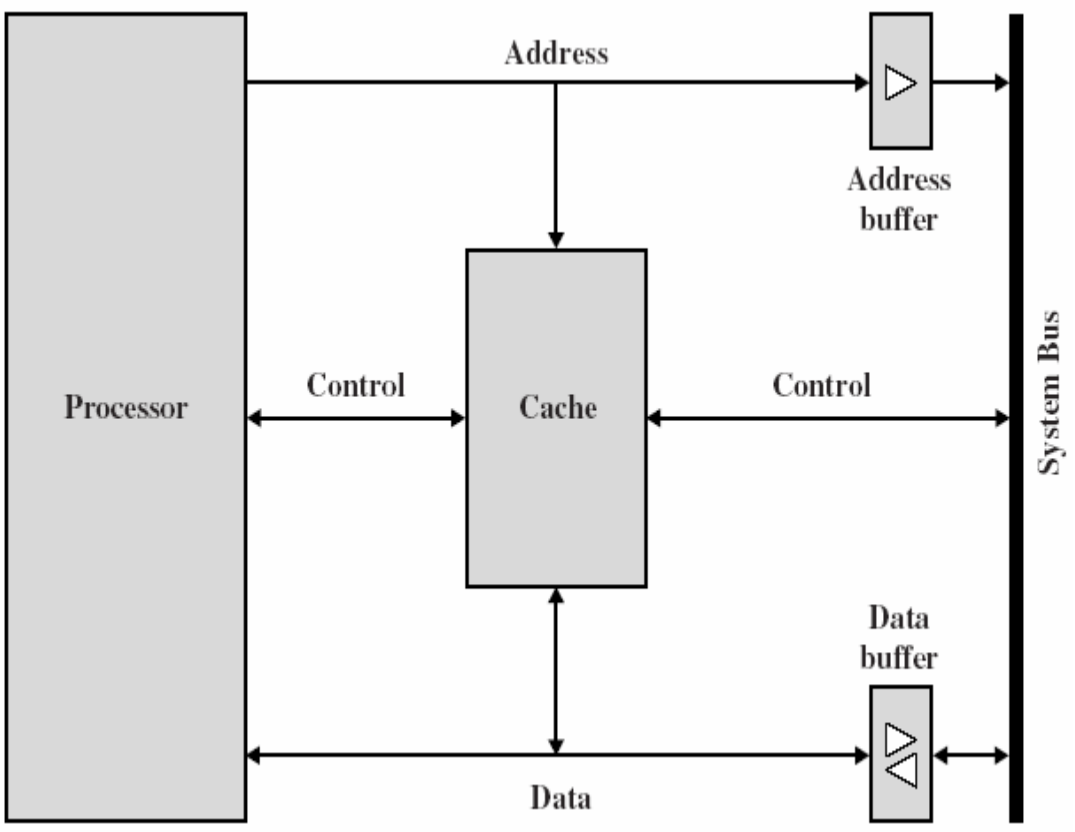


Cache Operation Overview:

- Processor requests the contents of some memory location.
- The cache is checked for the requested data.
- If found, the requested word is delivered to the processor.
- If not found, a block of main memory is first read into the cache, then the requested word is delivered to the processor.

- When a block of data is fetched into the cache to satisfy a single memory reference, it is likely that there will be future references to that same memory location or to other words in the block – locality or reference principle. Each block has a tag added to identify it.





Memory Mapping Organization

An algorithm is needed to map main memory blocks into cache lines. A method is needed to determine which main memory block occupies a cache line. Three techniques used:

- Direct mapping.
- Set Associative Mapping.
- Associative Mapping.

Assume the following:

- Cache of 64 Kbytes.
- Transfers between main memory and cache are in blocks of 4 bytes each – cache organized as 16K = 2^14 lines of 4 bytes each.
- Main memory of 16 Mbytes, directly addressable by a 24-bit address (where 2^24 = 16M) – main memory consists of 4M blocks of 4 bytes each.

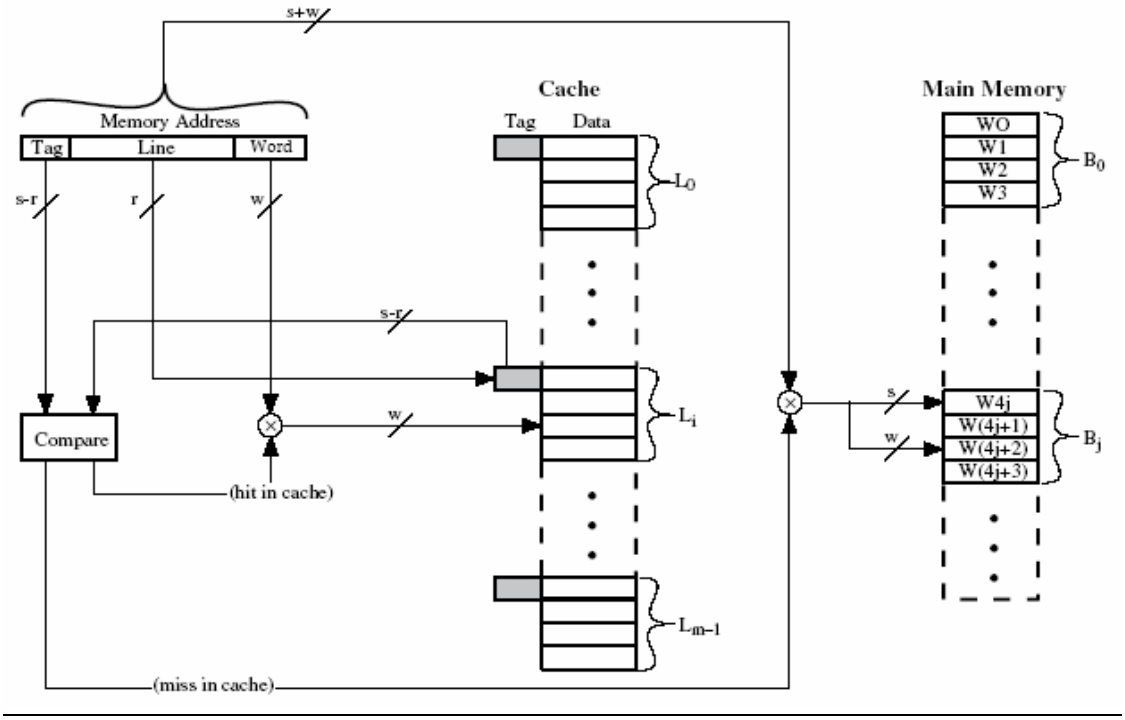
Direct mapping

- Each block of main memory maps to only one cache line
  - "cache line #" = "main memory block #" % "number of lines in cache"
- Main memory addresses are viewed as three fields.
  - Least significant w bits identify a unique word or byte within a block.
  - Most significant s bits specify one of the 2^s blocks of main memory.
    - Tag field of s-r bits (most significant).
    - Line field of r bits – identifies one of the m = 2^r lines of the cache.

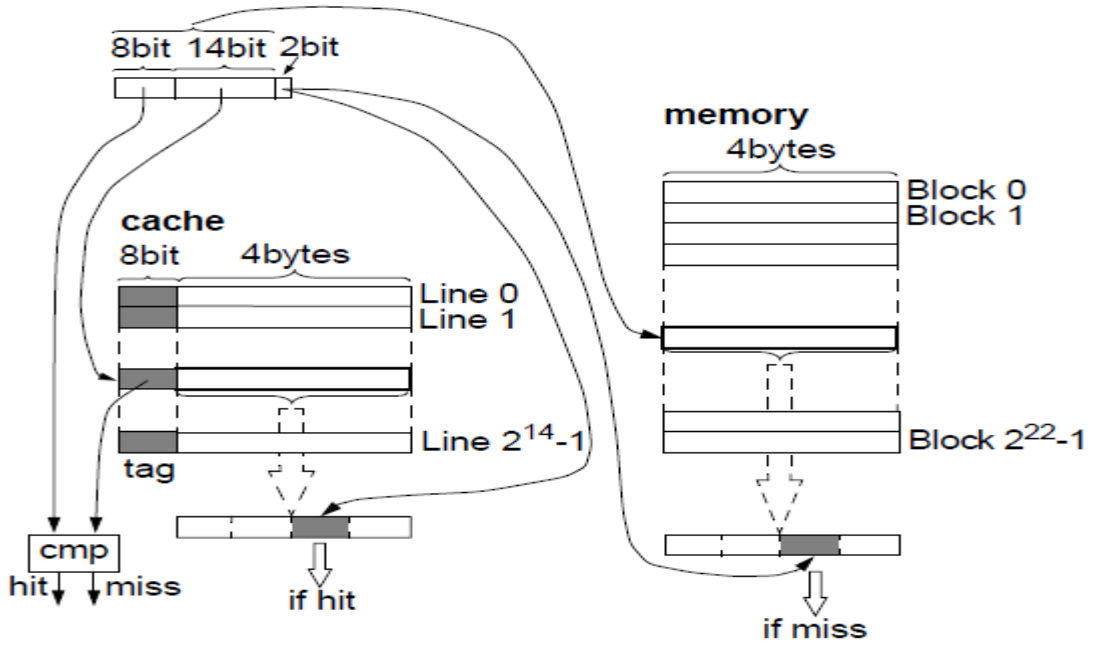
Tag (s-r)	Line or Slot (r)	Word (w)
8 bits	14 bits	2 bits

- 24 bit address.
- 2 bit word identifier.
- 22 bit block identifier.
- 8 bit tag (22-14).
- 14 bit slot or line.
- No two blocks in the same line have the same Tag field.
- Check contents of cache by finding and checking Tag.

Direct Mapping Cache Organization:



Example of Direct Mapping:



Direct Mapping Summary:

- Address length = (s+w) bits.
- Number of addressable units = 2^(s+w) words or bytes.
- Block size = line size = 2^w words or bytes.
- Number of blocks in main memory = 2^(s+w)/2^w = 2^s.
- Number of lines in cache = m = 2^r.
- Size of tag = (s-r) bits.

Advantages:

- simple and cheap;
- the tag field is short; only those bits have to be stored which are not used to address the cache (compare with the following approaches);
- Access is very fast.

Disadvantage:

- a given block fits into a fixed cache location Þ a given cache line will be replaced whenever there is a reference to another memory block which fits to the same line, regardless what the status of the other cache lines is.
  - This can produce a low hit ratio, even if only a very small part of the cache is effectively used.

Associative Mapping

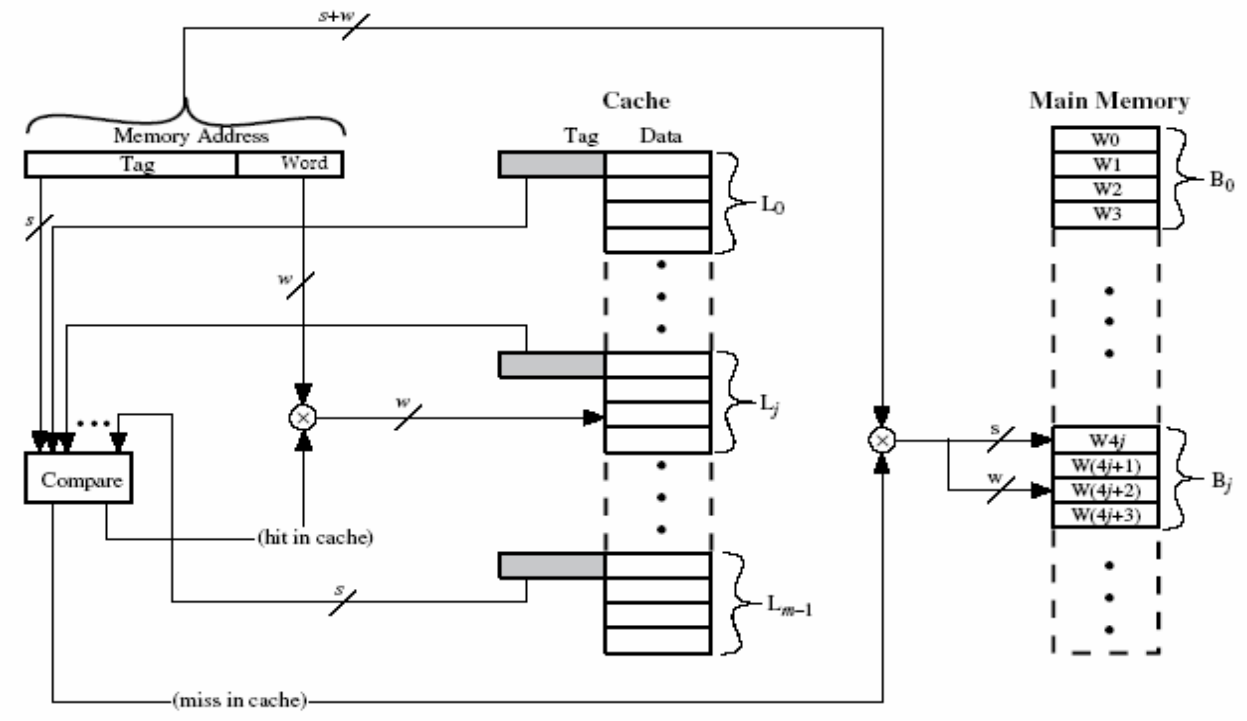
- A main memory block can be loaded into any line of the cache.
- A memory address is interpreted as a tag and a word field.
- The tag field uniquely identifies a block of main memory.
- Each cache line's tag is examined simultaneously to determine if a block is in cache.

- Main memory addresses are viewed as two fields:

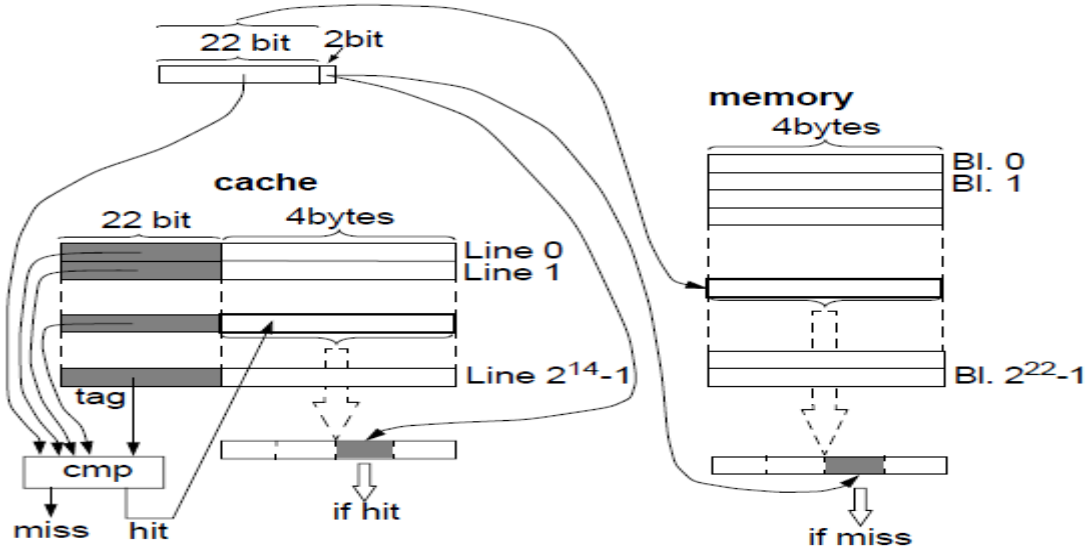
Tag (s)	Word (w)
22 bits	2 bits

- 22 bit tag stored with the 32 bit block of data.
- Tag field compared with tag entry to check for cache hit.
- 2 bit byte number.
-

Associative Mapping Cache Organization:



Example of Associative Mapping:



Associative Mapping Summary:

- Address length = (s+w) bits.
- Number of addressable units = 2^(s+w) words or bytes.
- Block Size = line size = 2^w words or bytes.
- Number of blocks in main memory = 2^(s+w)/2^w = 2^s.
- Number of lines in cache = undetermined.
- Size of tag = s bits.

Advantages:

• Associative mapping provides the highest flexibility concerning the line to be replaced when a new block is read into the cache.

Disadvantages:

• Complex.
• The tag field is long.
• Fast access can be achieved only using high performance associative memories for the cache, which is difficult and expensive.

Set Associative Mapping

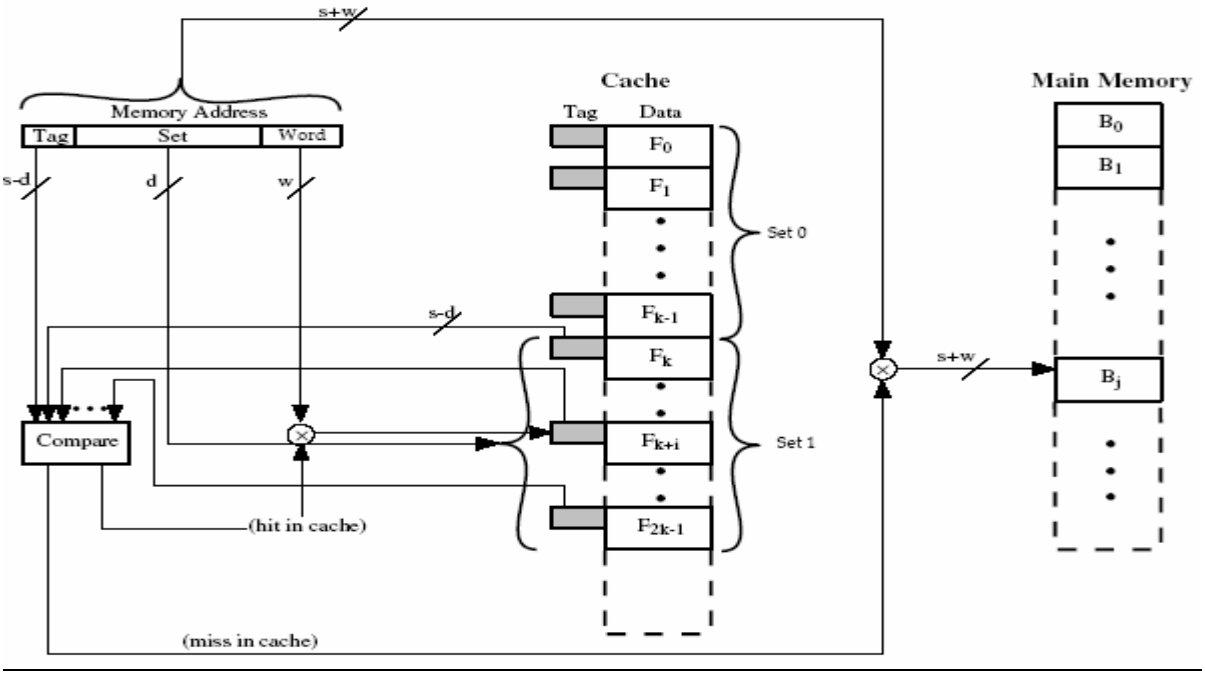
- Compromise between direct and associative mapping.
- Cache divided into v sets.
- Each set contains k lines.
- A given block maps into any line in a given set.
  - Ex: block B can be mapped into any line in set i.
- Ex: Assume that k = 2 – meaning there are 2 lines per set.
  - 2-way associative mapping – k-way mapping.
  - A given block can map into either of the 2 lines in exactly 1 set.
- k-Way Set Associative Cache Organization:

- Main memory addresses are viewed as two fields:

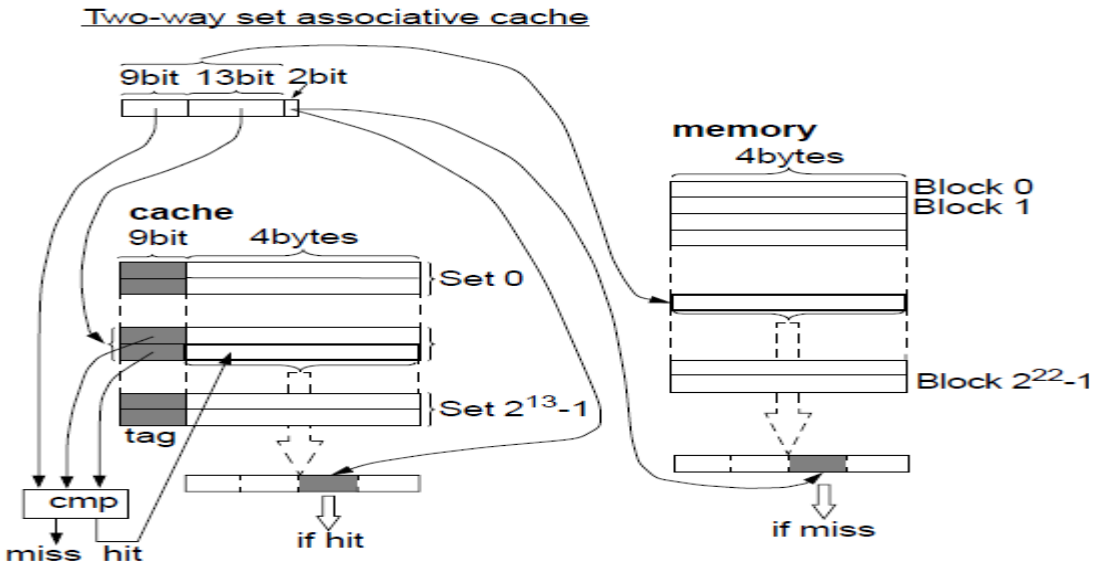
Tag (s-d)	Set (d)	Word (w)
9 bits	13 bits	2 bits

- 9 bit tag field – only compared with k tags in the given set.
- 13 bit set field – specifies one of the v = 2^d sets.
- 2 bit byte number.

Set Associative Mapping Cache Organization:



Example of Set Associative Mapping:



Set Associative Mapping Summary:

- Address length = (s+w) bits.
- Number of addressable units = 2^(s+w) words or bytes.
- Block Size = line size = 2^w words or bytes.
- Number of blocks in main memory = 2^(s+w)/2^w = 2^s.
- Number of lines in set = k.
- Number of sets v = 2^d.
- Number of lines in cache = kv = k*2^d.
- Size of tag = (s-d) bits.
- v = m, k = 1 reduces to direct mapping.
- v = 1, k = m reduces to associative mapping.

- 2-way mapping is most commonly used – significantly improves cache hit ratio over direct mapping.

- 4-way mapping – makes a modest additional improvement for a relatively small additional cost.

- Set associative mapping keeps most of the advantages of direct mapping:

- Short tag field.
- Fast access.
- Relatively simple.

- Set associative mapping tries to eliminate the main shortcoming of direct mapping; certain flexibility is given concerning the line to be replaced when a new block is read into the cache.

- Cache hardware is more complex for set associative mapping than for direct mapping.

- In practice 2 and 4-way set associative mapping are used with very good results. Larger sets do not produce further significant performance improvement.

- if a set consists of a single line direct mapping;
    - If there is one single set consisting of all linesÞ associative mapping.

Replacement Algorithms

- When a new block is to be placed into the cache, the block stored in one of the cache lines has to be replaced.
- With direct mapping there is no choice.
- With associative or set-associative mapping a replacement algorithm is needed in order to determine which block to replace (and, implicitly, in which cache line to place the block);

    - With set-associative mapping, the candidate lines are those in the selected set;
    - With associative mapping, all lines of the cache are potential candidates;

- The replacement is divided into categories they are; random and with use policies to replace.

    - Random replacement: One of the candidate lines is selected randomly.
    - All the other policies are based on information concerning the usage history of the blocks in the cache. The polices used;

        - Least recently used (LRU):
        - First-in-first-out (FIFO):
        - Least frequently used (LFU):

Least recently used (LRU):

The candidate line is selected which holds the block that has been in the cache the longest without being referenced.

First-in-first-out (FIFO):

The candidate line is selected which holds the block that has been in the cache the longest.

Least frequently used (LFU):

The candidate line is selected which holds the block that has got the fewest references.

- LRU is the most efficient: relatively simple to implement and good results.
- FIFO is simple to implement.
- Random replacement is the simplest to implement and results are surprisingly good.

Write Strategies

The write strategies is used to solve the problem of how to keep cache content and the content of main memory consistent without losing too much performance? This problem arises when a write is issued to a memory address, and the content of the respective address is potentially changed. The write strategies used were;

Write-through:

- All write operations are passed to main memory; if the addressed location is currently hold in the cache, the cache is updated so that it is coherent with the main memory.

- For writes, the processor always slows down to main memory speed.

Write-through with buffered write:

- The same as write-through, but instead of slowing the processor down by writing directly to main memory, the write address and data are stored in a high-speed write buffer; the write buffer transfers data to main memory while the processor continues it's task.

➢ higher speed, more complex hardware

Copy-back:

- Write operations update only the cache memory which is not kept coherent with main memory; cache lines have to remember if they have been updated; if such a line is replaced from the cache, its content has to be copied back to memory.

➢ Good performance (usually several writes are performed on a cache line before it is replaced and has to be copied into main memory), but complex hardware.

➢ Cache coherence problems are very complex and difficult to solve in multiprocessor systems.

Some Cache Architectures

Intel 80486:

- A single on-chip cache of 8 Kbytes.
- Line size: 16 bytes.
- 4-way set associative organization.

Pentium:

- Two on-chip caches, for data and instructions.
- Each cache: 8 Kbytes.
- Line size: 32 bytes.
- 2-way set associative organization.

PowerPC 601:

- A single on-chip cache of 32 Kbytes.
- Line size: 32 bytes.
- 8-way set associative organization.

PowerPC 603:

- Two on-chip caches, for data and instructions.
- Each cache: 8 Kbytes.
- Line size: 32 bytes.
- 2-way set associative organization.

PowerPC 604:

- Two on-chip caches, for data and instructions.
- Each cache: 16 Kbytes.
- Line size: 32 bytes.
- 4-way set associative organization.

PowerPC 620:

- Two on-chip caches, for data and instructions.
- Each cache: 32 Kbytes.
- Line size: 64 bytes.
- 8-way set associative organization.

Home work and research

- Home work:

- ✓ What is the different between physical address and logical address?
- ✓ Define the Memory Management Unit (MMU).

- Research / report:

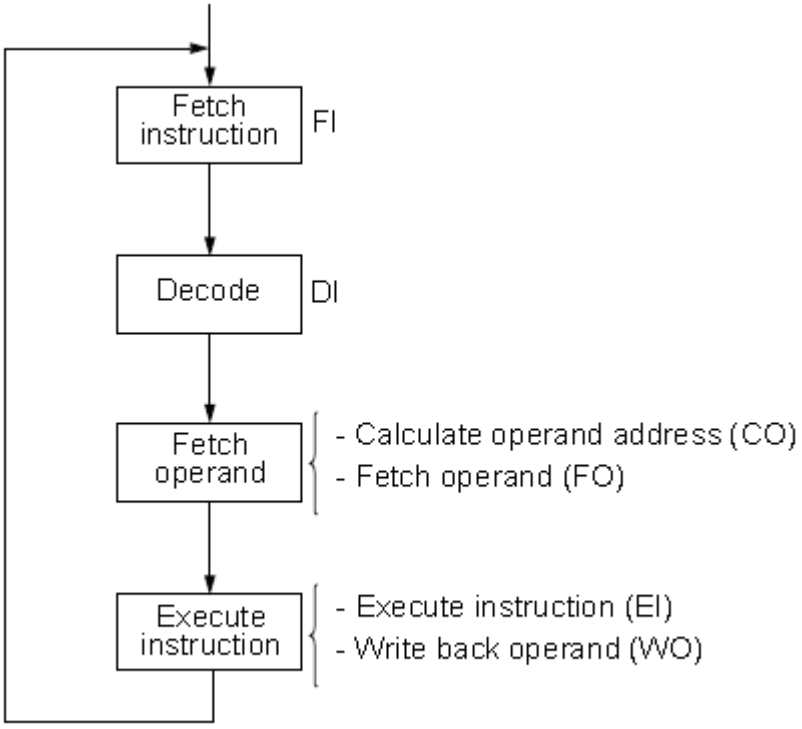
- ✓ Virtual memory.
- ✓ Paging.

Chapter 3
INSTRUCTION PIPELINING

- ✓ The Instruction Cycle.
- ✓ Instruction Pipelining.
- ✓ Pipeline Hazards.
- ✓ Structural Hazards.
- ✓ Data Hazards.
- ✓ Control Hazards.

The Instruction Cycle

Instruction execution is extremely complex and involves several operations which are executed successively This implies a large amount of hardware, but only one part of this hardware works at a given moment.

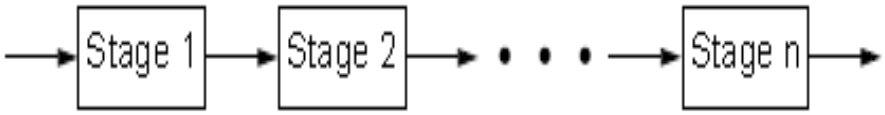


Instruction Pipelining.

- Pipelining is an implementation technique whereby multiple instructions are overlapped in execution.

- This is solved without additional hardware but only by letting different parts of the hardware work for different instructions at the same time.

- The pipeline organization of a CPU is similar to an assembly line: the work to be done in an instruction is broken into smaller steps (pieces), each of which takes a fraction of the time needed to complete the entire instruction. Each of these steps is a pipe stage (or a pipe segment).

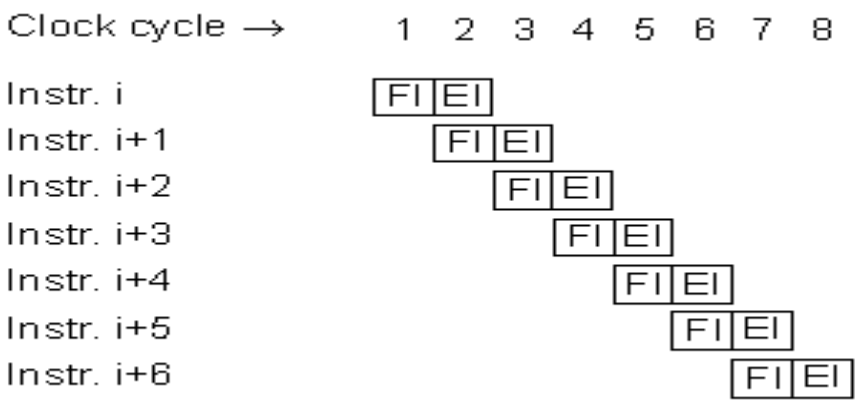


- The time required for moving an instruction from one stage to the next: a machine cycle (often this is one clock cycle). The execution of one instruction takes several machine cycles as it passes through the pipeline.

Acceleration by Pipelining

- Two stage pipeline:

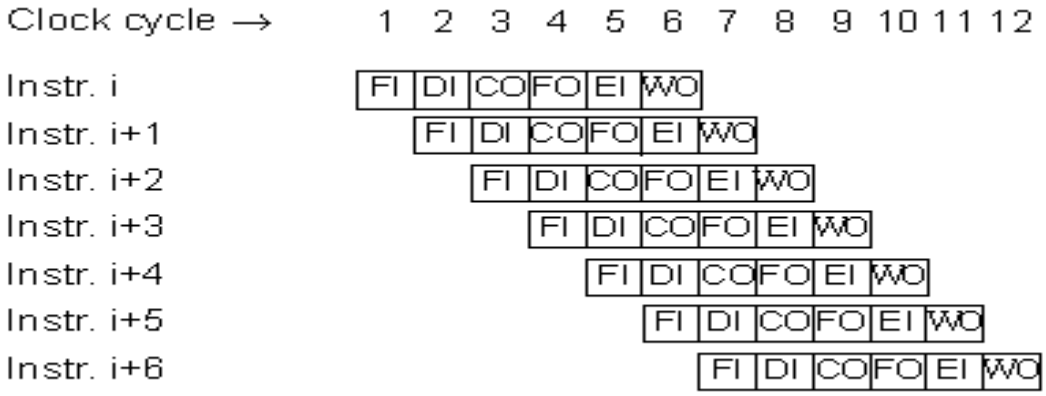
FI: fetch instruction
EI: execute instruction



- We consider that each instruction takes execution time Tex.
- Execution time for the 7 instructions, with pipelining: (Tex/2)*8= 4*Tex

➢ Six stage pipeline:

FI: fetch instruction
DI: decode instruction
CO: calculate operand address
FO: fetch operand
EI: execute instruction
WO: write operand



- Execution time for the 7 instructions, with pipelining: (Tex/6)*12= 2*Tex

- After a certain time (N-1 cycles) all the N stages of the pipeline are working: the pipeline is filled. Now, theoretically, the pipeline works providing maximal parallelism (N instructions are active simultaneously).

- Apparently a greater number of stages always provide better performance. However:

➢ A greater number of stages increase the overhead in moving information between stages and synchronization between stages.

➢ With the number of stages the complexity of the CPU grows.

➢ It is difficult to keep a large pipeline at maximum rate because of pipeline hazards.

Pipeline Hazards

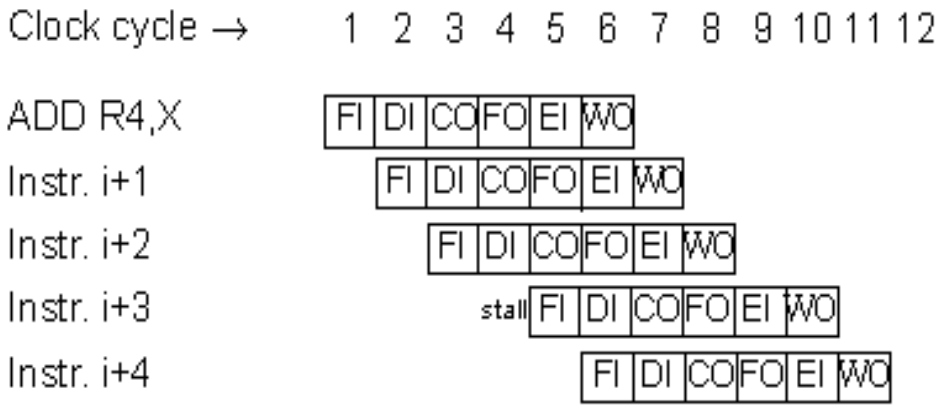
Pipeline hazards are situations that prevent the next instruction in the instruction stream from executing during its designated clock cycle. The instruction is said to be stalled. When an instruction is stalled, all instructions later in the pipeline than the stalled instruction are also stalled. Instructions earlier than the stalled one can continue. No new instructions are fetched during the stall.

Types of hazards:

✓ Structural hazards
✓ Data hazards
✓ Control hazards

Structural Hazards

- Structural hazards occur when a certain resource (memory, functional unit) is requested by more than one instruction at the same time.
- Instruction ADD R4,X fetches in the FO stage operand X from memory. The memory doesn't accept another access during that cycle.



Penalty: 1 cycle

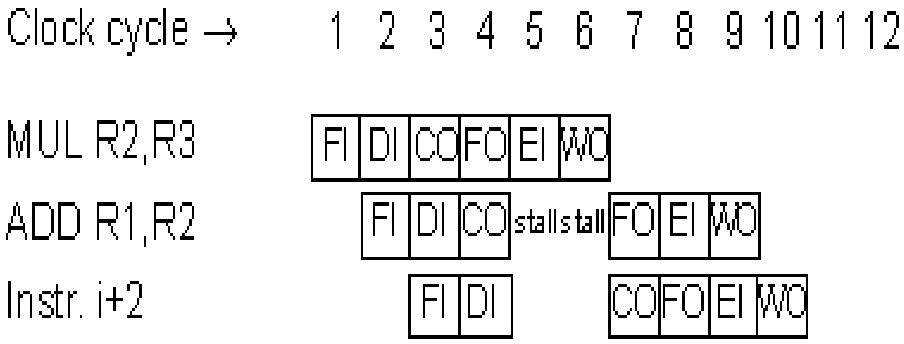
- Certain resources are duplicated in order to avoid structural hazards. Functional units (ALU, FP unit) can be pipelined themselves in order to support several instructions at a time. A classical way to avoid hazards at memory access is by providing separate data and instruction caches.

Data Hazards

- We have two instructions, I1 and I2. In a pipeline the execution of I2 can start before I1 has terminated. If in a certain stage of the pipeline, I2 needs the result produced by I1, but this result has not yet been generated, we have a data hazard.

- I1: MUL R2,R3 R2 ¬ R2 * R3

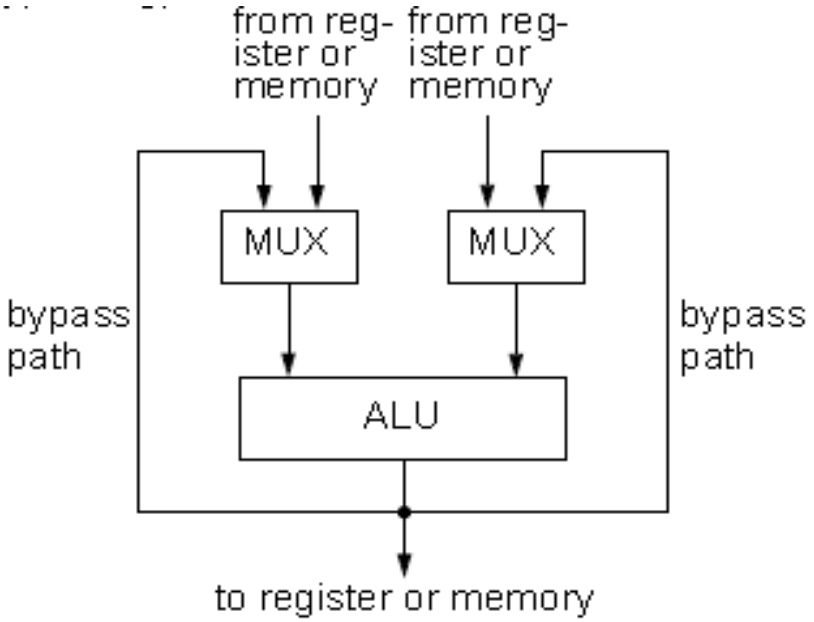
- I2: ADD R1,R2 R1 ¬ R1 + R2



Penalty: 2 cycles

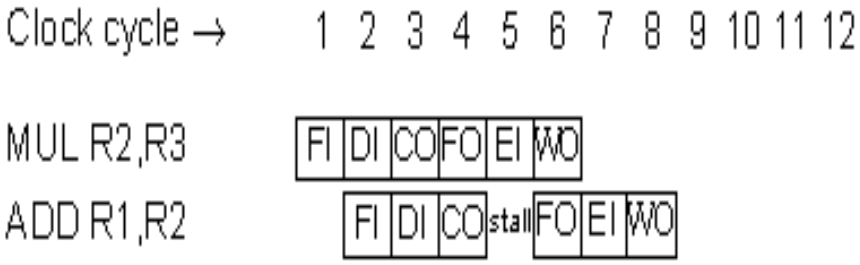
- Before executing its FO stage, the ADD instruction is stalled until the MUL instruction has written the result into R2.

- Some of the penalty produced by data hazards can be avoided using a technique called forwarding (bypassing).



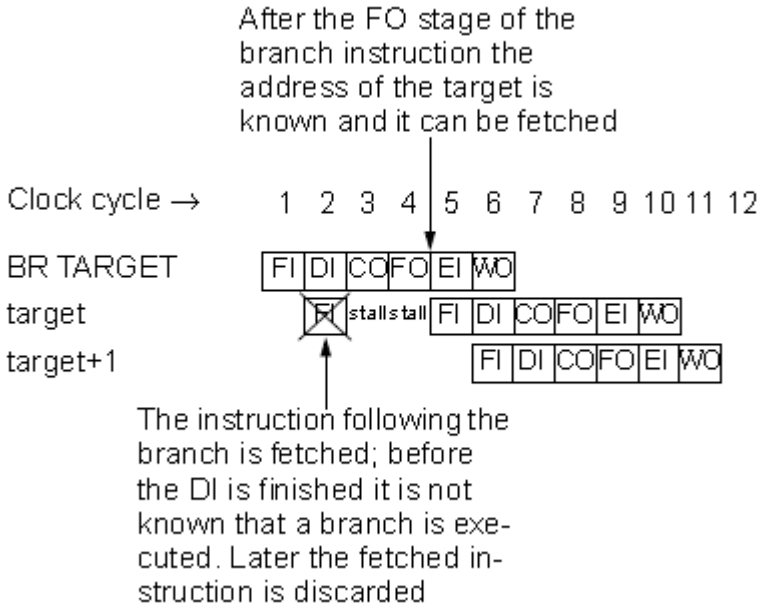
- The ALU result is always fed back to the ALU input.

- If the hardware detects that the value needed for the current operation is the one produced by the previous operation (but which has not yet been written back) it selects the forwarded result as the ALU input, instead of the value read from register or memory.



Control Hazards

- Control hazards are produced by branch Instructions



Penalty: 3 cycles

Chapter 4
INSTRUCTION SET COMPUTERS

- ✓ Why do we need RISCs and CISCs?
- ✓ Some Typical Features of Current Programs.
- ✓ Main Characteristics of RISC Architectures.

What are RISCs and CISCs?

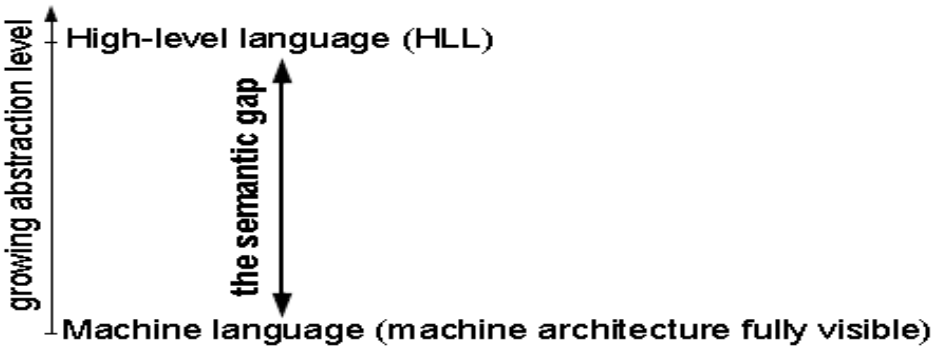
- The RISC architecture is an attempt to produce more CPU power by simplifying the instruction set of the CPU known as Reduced Instruction Set Computers (RISCs) , and it's represent an important innovation in the area of computer organization.

- The opposed trend to RISC is that of Complex Instruction Set Computers (CISCs).

- Both RISC and CISC architectures have been developed as an attempt to cover the semantic gap between level language ad high level language programs.

The Semantic Gap

- In order to improve the efficiency of software development, new and powerful programming languages have been developed (C++, and Java).They provides: high level of abstraction, conciseness, and power.
- By this evolution the semantic gap grows. The problem is how should new HLL programs be compiled and executed efficiently on processor architecture?



- There are Two possible answers:

➢ The **CISC approach:** design very complex architectures including a large number of instructions and addressing modes; include also instructions close to those present in HLL.

➢ The **RISC approach:** simplify the instruction set and adapt it to the real requirements of user programs.

Evaluation of Program Execution

- Evaluation of programs execution understood by what are programs doing most of the time?

- Several studies have been conducted to determine the execution characteristics of machine instruction sequences generated from HLL programs.

- Aspects of interest:

✓ The frequency of operations performed.
✓ The types of operands and their frequency of use.
✓ Execution sequencing (frequency of jumps, loops, subprogram calls).

Frequency of Instructions Executed:

- Frequency distributions of executed machine instructions are:

✓ moves: 33%
✓ conditional branch: 20%
✓ Arithmetic/logic: 16%
✓ others: Between 0.1% and 10%

- Addressing modes: the overwhelming majority of instructions uses simple addressing modes, in which the address can be calculated in a single cycle (register, register indirect, displacement); complex addressing modes (memory indirect, indexed + indirect, displacement + indexed, stack) are used only by ~18% of the instructions.

Operand Types:

- 74 to 80% of the operands are scalars (integers, characters, etc.) which can be hold in registers;
- the rest (20-26%) are arrays/structures; 90% of them are global variables;
- 80% of the scalars are local variables.

Main Characteristics of RISC Architectures

- The instruction set is limited and includes only simple instructions;

➢ The goal is to create an instruction set containing instructions that execute quickly; most of the RISC instructions are executed in a single machine cycle (after fetched and decoded).

✓ Pipeline operation (without memory reference).

- RISC instructions, being simple, are hard-wired, while CISC architectures have to use microprogramming in order to implement complex instructions.

- Having only simple instructions results in reduced complexity of the control unit and the data path; as a consequence, the processor can work at a high clock frequency.

- The pipelines are used efficiently if instructions are simple and of similar execution time.

- Complex operations on RISCs are executed as a sequence of simple RISC instructions. In the case of CISCs they are executed as one single or a few complex instructions.

❖ Let's see some small example:

- Assume: we have a program with 80% of executed instructions being simple and 20% complex;

  - ✓ **on a CISC** machine simple instructions take 4 cycles, complex instructions take 8 cycles; cycle time is 100 ns (10-7 s);

  - ✓ **on a RISC** machine simple instructions are executed in one cycle; complex operations are implemented as a sequence of instructions; we consider on average 14 instructions (14 cycles) for a complex operation; cycle time is 75 ns (0.75 * 10-7 s).

- How much time takes a program of 1 000 000 instructions?

  - ✓ CISC: (106*0.80*4 + 106*0.20*8)*10-7 = 0.48 s
  - ✓ RISC: (106*0.80*1 + 106*0.20*14)*0.75*10-7 = 0.27 s

- complex operations take more time on the RISC, but their number is small;

- Because of its simplicity, the RISC works at a smaller cycle time; with the CISC, simple instructions are slowed down because of the increased data path length and the increased control complexity.

- Load-and-store architecture;

- Only LOAD and STORE instructions reference data in memory; all other instructions operate only with registers (are register-to-register instructions); thus, only the few instructions accessing memory need more than one cycle to execute (after fetched and decoded).

    - ✓ Pipeline operation with memory reference.

- Instructions use only few addressing modes and are usually register, direct, register indirect, displacement.

- Instructions are of fixed length and uniform format;

- This makes the loading and decoding of instructions simple and fast; it is not needed to wait until the length of an instruction is known in order to start decoding the following one;

- A large number of registers is available;

- Variables and intermediate results can be stored in registers and do not require repeated loads and stores from/to memory.

- All local variables of procedures and the passed parameters can be stored in registers.

What is really better RISCs or CISCs?

- RISC architectures have several advantages and they were discussed throughout this lecture.

- A lot of performance comparisons have shown that benchmark programs are really running faster on RISC processors than on processors with CISC Characteristics.

- It is difficult to identify which feature of a processor produces the higher performance. Some "CISC fans" argue that the higher speed is not produced by the typical RISC features but because of technology, better compilers, etc.

- An argument in favor of the CISC; the simpler Instruction set of RISC processors results in a larger memory requirement compared to the similar program compiled for CISC architecture.

- Most of the current processors are not typical RISCs or CISCs but try to combine advantages of both approaches

Some Processor Examples

CISC Architectures:

- VAX 11/780:

- ✓ No. of instructions: 303
- ✓ Instruction size: 2 - 57
- ✓ Instruction format: not fixed
- ✓ Addressing modes: 22
- ✓ Number of general purpose registers: 16

- Pentium:

- ✓ No. of instructions: 235
- ✓ Instruction size: 1 - 11
- ✓ Instruction format: not fixed
- ✓ Addressing modes: 11
- ✓ Number of general purpose registers: 8

RISC Architectures:

- Sun SPARC:

- ✓ No. of instructions: 52
- ✓ Instruction size: 4
- ✓ Instruction format: fixed
- ✓ Addressing modes: 2
- ✓ Number of general purpose registers: up to 520

- PowerPC:

- ✓ No. of instructions: 206
- ✓ Instruction size: 4
- ✓ Instruction format: not fixed (but small differences)
- ✓ Addressing modes: 2
- ✓ Number of general purpose registers: 32

Chapter 5
SUPERSCALAR PROCESSORS

- ✓ What is a Superscalar Architecture?
- ✓ Super pipelining.
- ✓ Features of Superscalar Architectures.
- ✓ Policies for Parallel Instruction Execution.

What is a Superscalar Architecture?

A superscalar architecture is one in which several instructions can be initiated simultaneously and executed independently. Pipelining allows several instructions to be executed at the same time, but they have to be in different pipeline stages at a given moment.

- Superscalar architectures include all features of pipelining but, in addition, there can be several instructions executing simultaneously in the same pipeline stage. They have the ability to initiate multiple instructions during the same clock cycle.

- There are two typical approaches today, in order to improve performance:

- ✓ Super pipelining.
- ✓ Superscalar.

Super pipelining

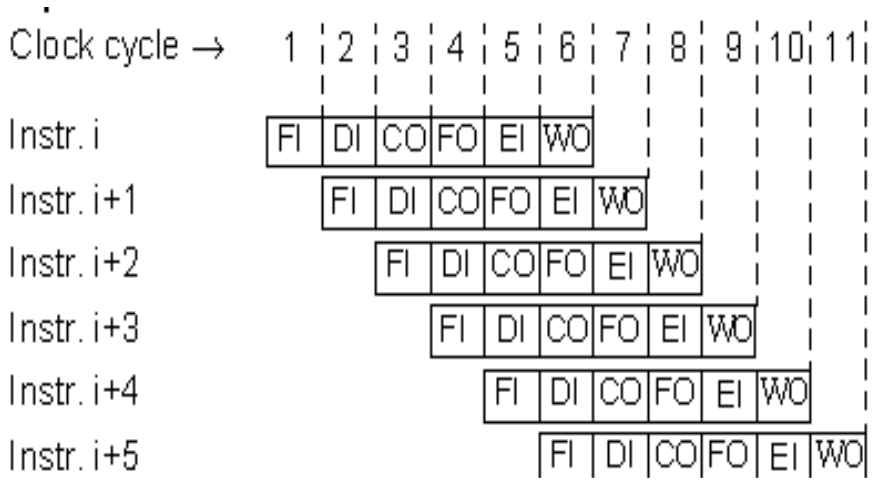
- Super pipelining is based on dividing the stages of a pipeline into sub stages and thus increasing the number of instructions which are supported by the pipeline at a given moment.

- By dividing each stage into two, the clock cycle period t will be reduced to the half, t/2; hence, at the maximum capacity, the pipeline produces a result every t/2 s.

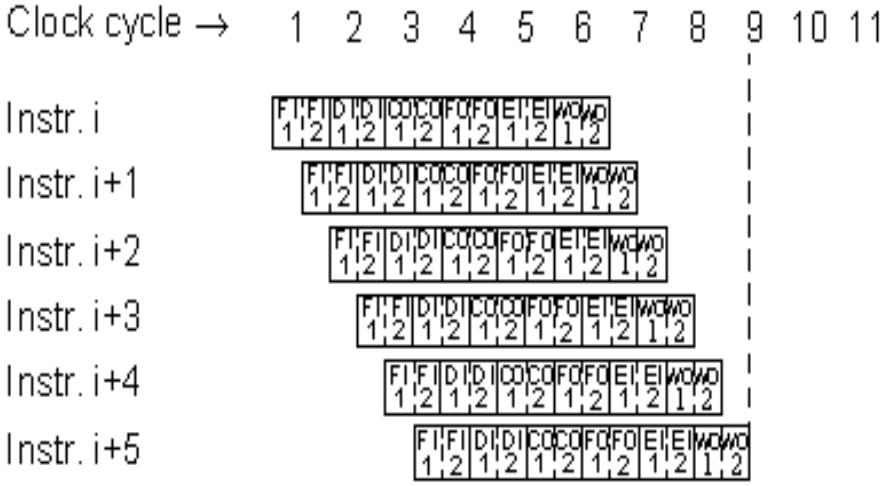
- For a given architecture and the corresponding instruction set there is an optimal number of pipeline stages; increasing the number of stages over this limit reduces the overall performance.

- A solution to further improve speed is the superscalar architecture.

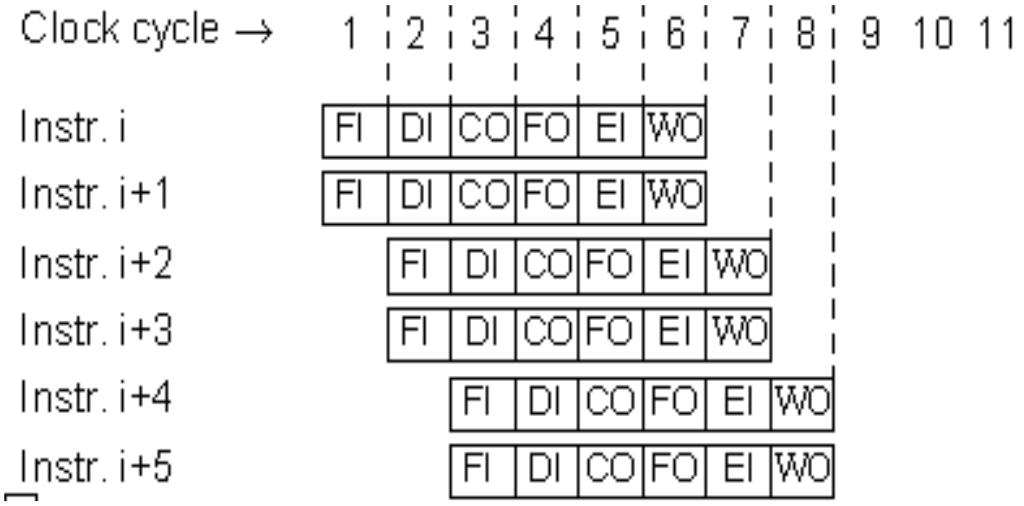
➢ Pipelined execution



➢ Super pipelined execution



➢ Superscalar execution



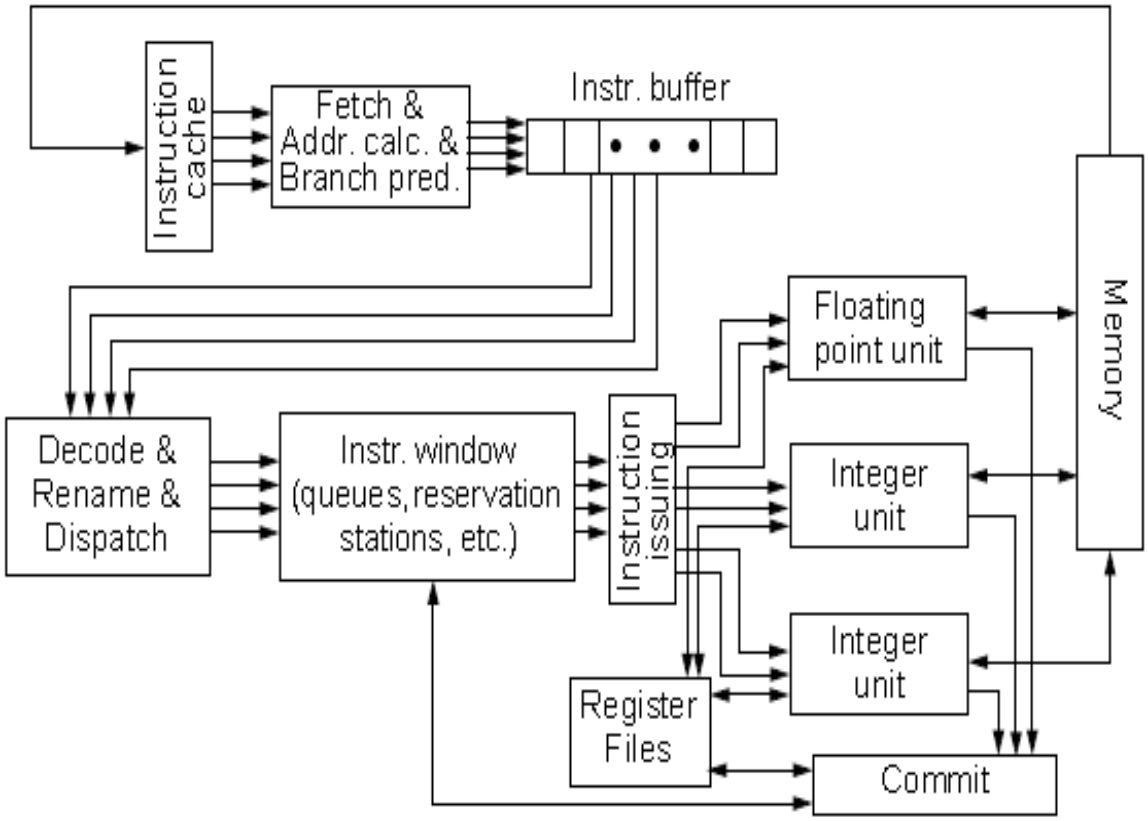
Superscalar Architectures

- Superscalar architectures allow several instructions to be issued and completed per clock cycle.

- A superscalar architecture consists of a number of pipelines that are working in parallel.

- Depending on the number and kind of parallel units available, a certain number of instructions can be executed in parallel.

- In the following example as shown in figure below, a floating point and two integer operations can be issued and executed simultaneously; each unit is pipelined and can execute several operations in different pipeline stages.



(Superscalar Architectures model)

Policies for Parallel Instruction Execution

- The ability of a superscalar processor to execute instructions in parallel is determined by:

✓ The number and nature of parallel pipelines (this determines the number and nature of instructions that can be fetched and executed at the same time);

✓ The mechanism that the processor uses to find independent instructions (instructions that can be executed in parallel).

- The policies used for instruction execution are characterized by the following two factors:

✓ The order in which instructions are issued for execution;
✓ The order in which instructions are completed (they write results into registers and memory locations).

- The simplest policy is to execute and complete instructions in their sequential order. This, however, gives little chances to find instructions which can be executed in parallel.

- In order to improve parallelism the processor has to look ahead and try to find independent instructions to execute in parallel.

- The execution policies are:

✓ In-order issue with in-order completion.
✓ 3. Out-of-order issue with out-of-order completion.

➢ <u>In-Order Issue with In-Order Completion</u>

- Instructions are issued in the exact order that would correspond to sequential execution; results are written (completion) in the same order.

- An instruction cannot be issued before the previous one has been issued;

- An instruction completes only after the previous one has completed.

- To guarantee in-order completion, instruction issuing stalls when there is a conflict and when the unit requires more than one cycle to execute;

- The processor detects and handles (by stalling) true data dependencies and resource conflicts.
- As instructions are issued and completed in their strict order, the resulting parallelism is very much dependent on the way the program is written/ compiled.

❖ Note: With superscalar processors we are interested in techniques which are not compiler based but allow the hardware alone to detect instructions which can be executed in parallel and to issue them.

- With in-order issue in-order completion the processor has not to bother about output dependency and anti dependency! It has only to detect true data dependencies.

➢ Out-of-Order Issue with Out-of-Order Completion

- With in-order issue, no new instruction can be issued when the processor has detected a conflict and is stalled, until after the conflict has been resolved.

- The processor is not allowed to look ahead for further instructions, which could be executed in parallel with the current ones.

- Out-of-order issue tries to resolve the above problem. Taking the set of decoded instructions the processor looks ahead and issues any instruction, in any order, as long as the program execution is correct.

- With out-of-order issue out-of-order completion the processor has to bother about true data dependency and both about output-dependency and anti dependency!

Some Architecture

- PowerPC 604

- six independent execution units:
- Branch execution unit
- Load/Store unit
- 3 Integer units
- Floating-point unit
- in-order issue
- register renaming

- Power PC 620

- provides in addition to the 604 out-of-order issue

Pentium

- three independent execution units:
- 2 Integer units
- Floating point unit
- in-order issue

- Pentium II

- provides in addition to the Pentium out-of-order issue
- five instructions can be issued in one cycle

Chapter 6
ARCHITECTURES of PARALLEL COMPUTATION

- ✓ Why Parallel Computation.
- ✓ Parallel Programs.
- ✓ A Classification of Computer Architectures.
- ✓ Performance of Parallel Architectures.
- ✓ The Interconnection Network.
- ✓ Array Processors.
- ✓ Multiprocessors.
- ✓ Multicomputer.

Why Parallel Computation?

- Because of The need for high performance! Parallel computation is needed.

- Two main factors contribute to high performance of modern processors:
- ✓ Fast circuit technology.
- ✓ Architectural features:

- ➢ Large caches.
- ➢ Multiple fast buses.
- ➢ Pipelining.
- ➢ Superscalar architectures (multiple function units).

- However Computers running with a single CPU, often are not able to meet performance needs in certain areas:

- ➢ Fluid flow analysis and aerodynamics;
- ➢ Simulation of large complex systems, for example in physics, economy, biology, technical; Computer aided design; Multimedia.

- Applications in the above domains are characterized by a very high amount of numerical computation and/or a high quantity of input data.

- One of the solutions to get a high performance is to use parallel computers.

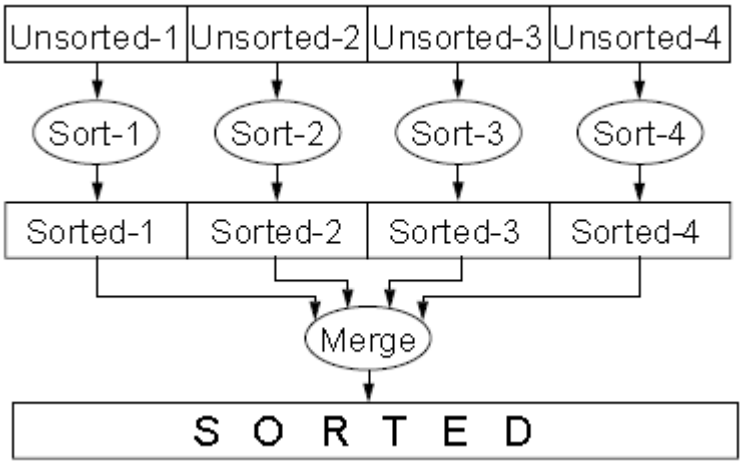
Parallel Computers

- One solution to the need for high performance: architectures in which several CPUs are running in order to solve a certain application.

- Such computers have been organized in very different ways. Some key features:

- Number and complexity of individual CPUs.
- Availability of common (shared memory).
- Interconnection topology.
- Performance of interconnection network.
- I/O devices.

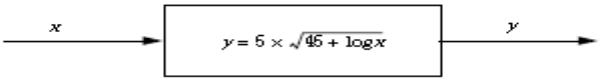
Parallel sorting

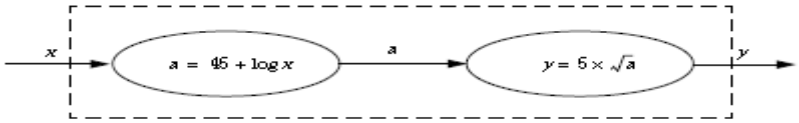


Matrix addition

$$\begin{vmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ a_{13} & a_{23} & \cdots & a_{m3} \\ \cdots & \cdots & \cdots & \cdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{vmatrix} + \begin{vmatrix} b_{11} & b_{21} & \cdots & b_{m1} \\ b_{12} & b_{22} & \cdots & b_{m2} \\ b_{13} & b_{23} & \cdots & b_{m3} \\ \cdots & \cdots & \cdots & \cdots \\ b_{1n} & b_{2n} & \cdots & b_{mn} \end{vmatrix} = \begin{vmatrix} c_{11} & c_{21} & \cdots & c_{m1} \\ c_{12} & c_{22} & \cdots & c_{m2} \\ c_{13} & c_{23} & \cdots & c_{m3} \\ \cdots & \cdots & \cdots & \cdots \\ c_{1n} & c_{2n} & \cdots & c_{mn} \end{vmatrix}$$

Pipeline model computation



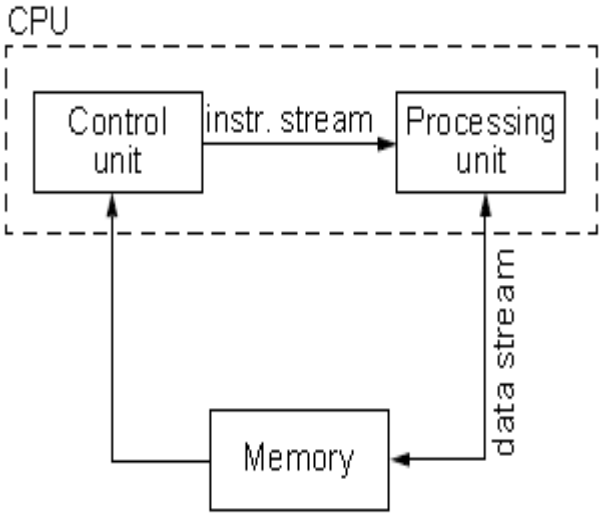


Classification of Computer Architectures

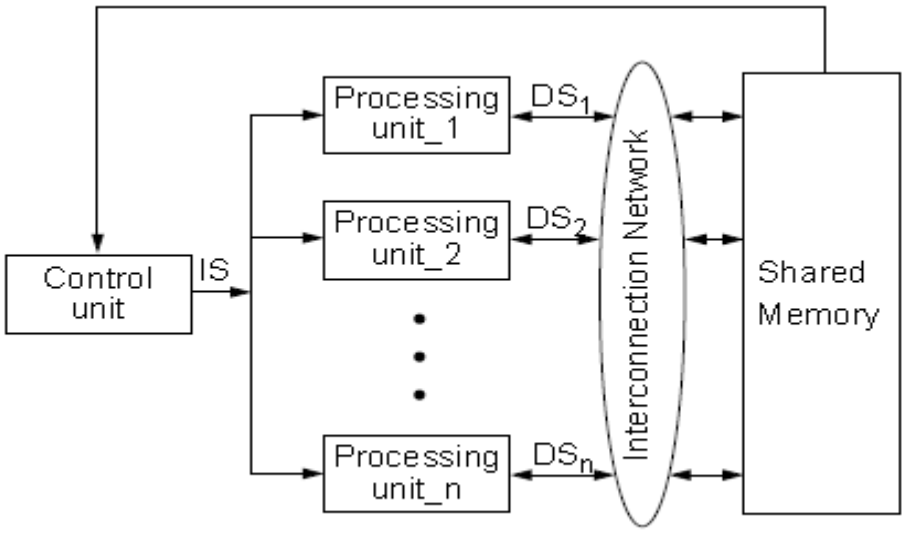
Flynn's classification is based on the nature of the instruction flow executed by the computer and that of the data flow on which the instructions operate. The classifications are:

- ✓ Single Instruction stream, Single Data stream (SISD).
- ✓ Single Instruction stream, Multiple Data stream (SIMD), with shared memory.
- ✓ Single Instruction stream, Multiple Data stream (SIMD), with no shared memory.
- ✓ Multiple Instruction stream, Multiple Data stream (MIMD), with shared memory.
- ✓ Multiple Instruction stream, Multiple Data stream (MIMD), with no shared memory.

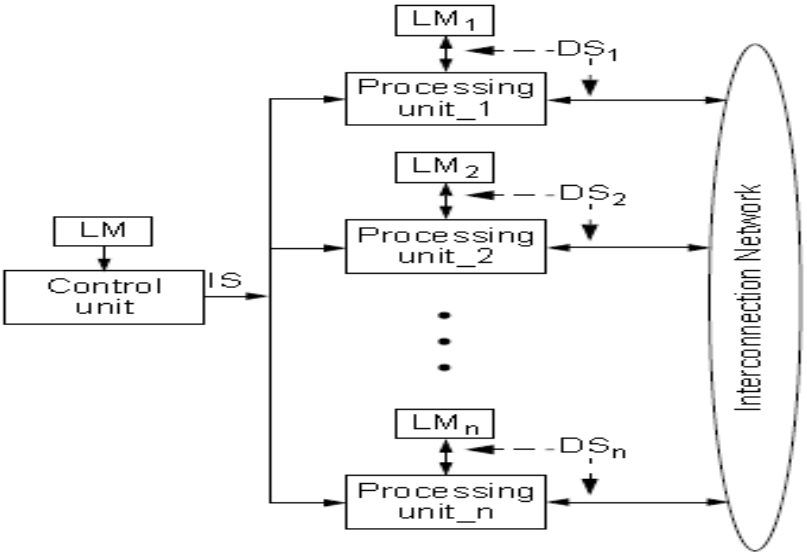
Single Instruction stream, Single Data stream (SISD)



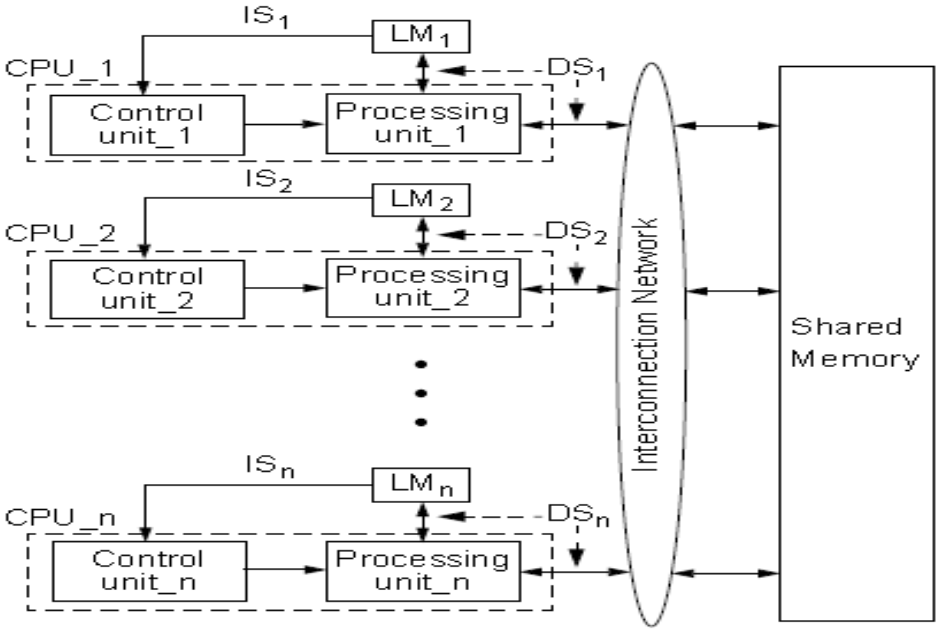
Single Instruction stream, Multiple Data stream (SIMD), with shared memory



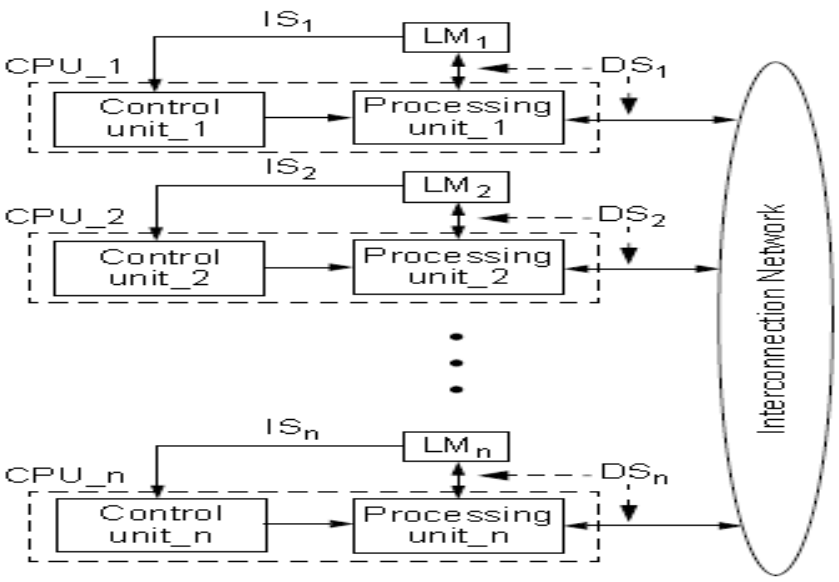
Single Instruction stream, Multiple Data stream (SIMD) , with no shared memory



Multiple Instruction stream, Multiple Data stream (MIMD) , with shared memory



Multiple Instruction stream, Multiple Data stream (MIMD) , with no shared memory



Performance of Parallel Architectures

- Performance of parallel architecture based on questions below;

- How fast runs a parallel computer at its maximal potential?
- How fast execution can we expect from a parallel computer for a concrete application?
- How do we measure the performance of a parallel computer and the performance improvement we get by using such a computer?

- The solutions are to use the performance metrics below;

- **Peak rate**: the maximal computation rate that can be theoretically achieved when all modules are fully utilized. The peak rate is of no practical significance for the user. It is mostly used by vendor companies for marketing of their computers.

- **Speedup**: measures the gain we get by using a certain parallel computer to run a given parallel program in order to solve a specific problem.

$$S = \frac{T_S}{T_P}$$

- ✓ TS: execution time needed with the best sequential algorithm;
- ✓ TP: execution time needed with the parallel algorithm.

- ➢ **Efficiency**: this metric relates the speedup to the number of processors used; by this it provides a measure of the efficiency with which the processors are used.

$$E = \frac{S}{p}$$

- ✓ S: speedup;
- ✓ P: number of processors.

- For the ideal situation, in theory:

$$S = \frac{\frac{T_S}{T_S}}{p} = p;$$

- ✓ Which means E = 1. Note: Practically the ideal efficiency of 1 cannot be achieved!

- Beside the intrinsic sequentially of some parts of an algorithm there are also other factors that limit the achievable speedup:

- ✓ Communication cost.
- ✓ Load balancing of processors.
- ✓ Costs of creating and scheduling processes.
- ✓ I/O operations.

- There are many algorithms with a high degree of parallelism; for such algorithms the value of f is very small and can be ignored. These algorithms are suited for massively parallel systems; in such cases the other limiting factors, like the cost of communications, become critical.

- Consider a highly parallel computation, so that f (the ratio of sequential computations) can be neglected.

- define fc, the fractional communication overhead of a processor:

- ✓ Tcalc: time that a processor executes computations;
- ✓ Tcomm: time that a processor is idle because of communication;

$$f_c = \frac{T_{comm}}{T_{calc}}$$

$$T_p = \frac{T_S}{p} \times (1 + f_c)$$

$$S = \frac{T_S}{T_P} = \frac{p}{1 + f_c}$$

$$E = \frac{1}{1 + f_c} \approx 1 - f_c$$

- With algorithms that have a high degree of parallelism, massively parallel computers, consisting of large number of processors, can be efficiently used if fc is small; this means that the time spent by a processor for communication has to be small compared to its effective time of computation.

- In order to keep fc reasonably small, the size of processes cannot go below a certain limit.

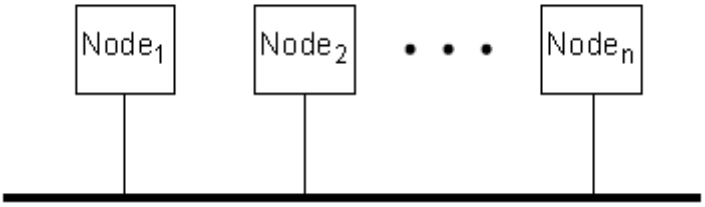
The Interconnection Network

- The interconnection network (IN) is a key component of the architecture. It has a decisive influence on the overall performance and cost.
- The traffic in the IN consists of data transfer and transfer of commands and requests.
- The key parameters of the IN are:

✓ Total bandwidth: transferred bits/second.
✓ Cost.

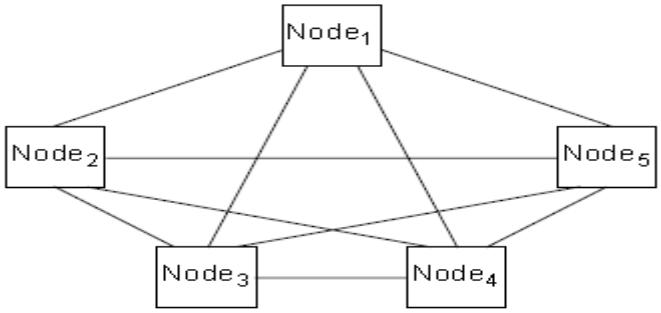
Single Bus

- Single bus networks are simple and cheap.
- One single communication is allowed at a time; the bandwidth is shared by all nodes.
- Performance is relatively poor.
- In order to keep a certain performance, the number of nodes is limited (16 - 20).



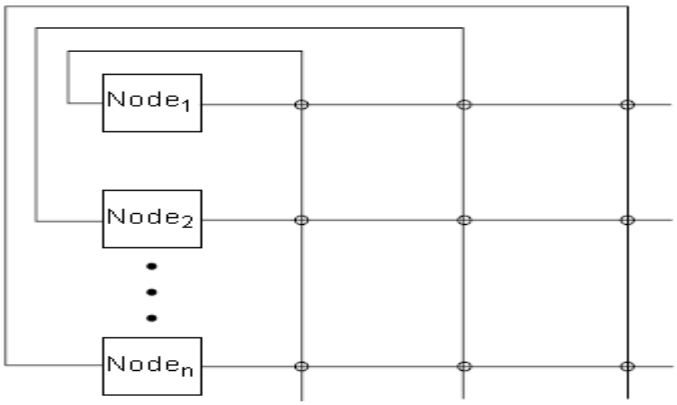
Completely connected network

- Each node is connected to every other one.
- Communications can be performed in parallel between any pair of nodes.
- Both performance and cost are high.
- Cost increases rapidly with number of nodes.



Crossbar network

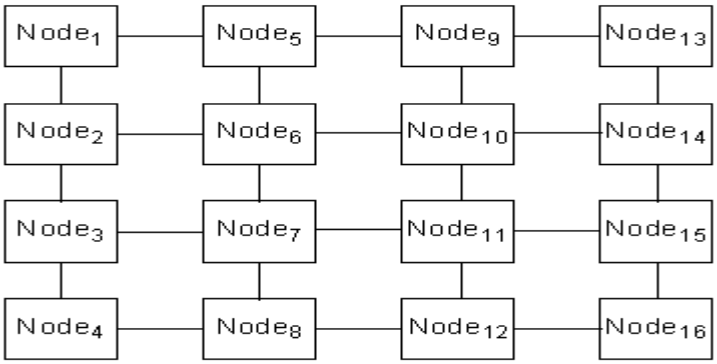
- The crossbar is a dynamic network: the interconnection topology can be modified by positioning of switches.
- The crossbar switch is completely connected: any node can be directly connected to any other.
- Fewer interconnections are needed than for the static completely connected network; however, a large number of switches is needed.
- A large number of communications can be performed in parallel (one certain node can receive or send only one data at a time).



Mesh network

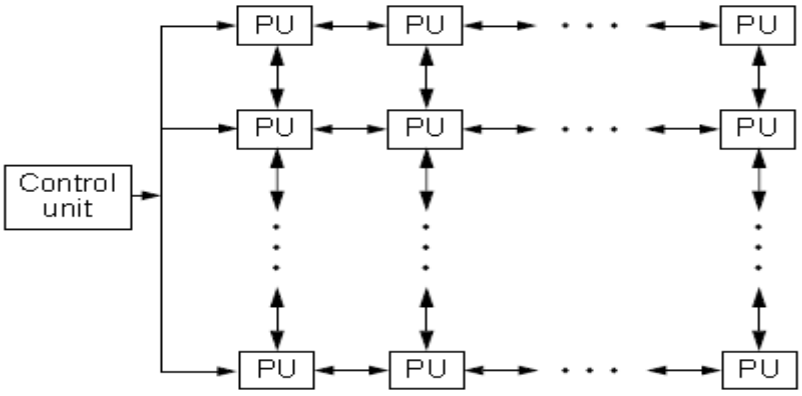
- Mesh networks are cheaper than completely connected ones and provide relatively good performance.
- In order to transmit an information between certain nodes, routing through intermediate nodes is needed (maximum 2*(n-1) intermediates for an n*n mesh).

- It is possible to provide wraparound connections: between nodes 1 and 13, 2 and 14, etc.
- Three dimensional meshes have been also implemented.



Array Processors

- SIMD computers are usually called array processors.
- Array processors are highly specialized for numerical problems that can be expressed in matrix or vector format.



- PU's are usually very simple: an ALU which executes the instruction broadcast by the CU, a few registers, and some local memory.

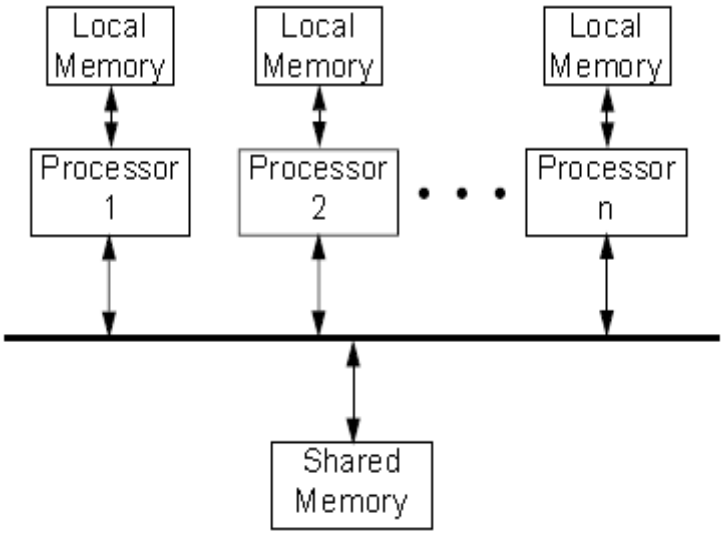
Multiprocessors

- Shared memory MIMD computers are called multiprocessors.

- Some multiprocessors have no shared memory which is central to the system and equally accessible to all processors. All the memory is distributed as local memory to the processors. However, each processor has access to the local memory of any other processor.

- This memory organization is called distributed shared memory.

- Communication between processors is through the shared memory. One processor can change the value in a location and the other processors can read the new value.



Multicomputer

- MIMD computers with a distributed address space, so that each processor has its one private memory which is not visible to other processors, are called multicomputer:

- Communication between processors is only by passing messages over the interconnection network.

- From the programmers point of view this means that no shared variables are available (a variable can be accessed only by one single process). For communication between parallel activities (processes) the programmer uses channels and send/receive operations (see program in slide 10).

- The speed of the interconnection network is an important parameter for the overall performance.