# Computer Architecture

# 2020-2021

## المرحلة الثالثة / الفصل الثاني

## أستاذ المادة

## ا.م.د.اسماء عبدالله فهد

## Programming I/O Devices:

**Input/output** or **I/O** is the communication between an information processing system, such as a computer, and the outside world, possibly a human or another information processing system. Inputs are the signals or data received by the system and outputs are the signals or data sent from it.

Figure (1) shows that a device has two important components. The first is the hardware interface it presents to the rest of the system. Just like a piece of software, hardware must also present some kind of interface that allows the system software to control its operation. Thus, all devices have some specified interface and protocol for typical interaction.

The second part of any device is its internal structure. This part of the device is implementation specific and is responsible for implementing the abstraction the device presents to the system. Very simple devices will have one or a few hardware chips to implement their functionality; more complex devices will include a simple CPU, some general purpose memory, and other device-specific chips to get their job done.

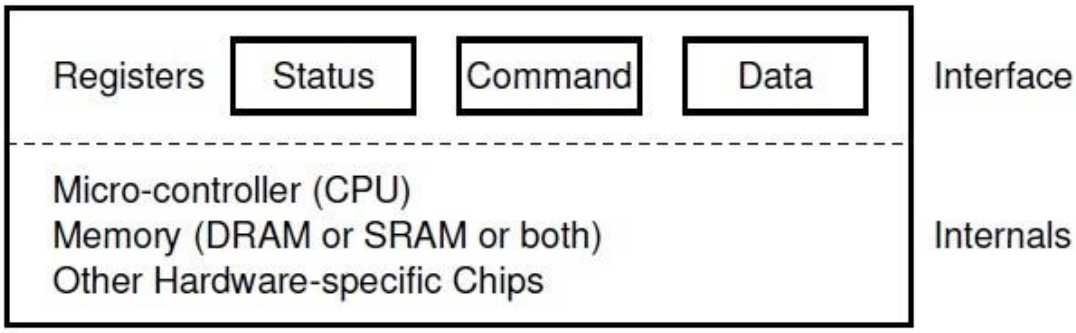



Figure (1) A canonical I/O Device Component

In this figure the (simplified) device interface is comprised of three registers: a status register, which can be read to see the current status of the device; a command register, to tell the device to perform a certain task; and a data register to pass data to the device, or get data from the device. By

reading and writing these registers, the operating system can control device behavior.

## I/O Programming Techniques

There are three techniques for programming the I/O devices:

- Programmed I/O
- Interrupt driven
- Direct Memory Access (DMA)

### 1. Programmed I/O:

Let us now describe a typical interaction that the operating system (OS) might have with the device in order to get the device to do something on its behalf. The protocol is as follows:

While (STATUS == BUSY)

; // wait until device is not busy

Write data to DATA register

Write command to COMMAND register

(Doing so starts the device and executes the command)

While (STATUS == BUSY)

; // wait until device is done with your request

The protocol has four steps. In the first, the OS waits until the device is ready to receive a command by repeatedly reading the status register; we call this polling the device (basically, just asking it what is going on). Second, the OS sends some data down to the data register; one can imagine that if this were a disk, for example, that multiple writes would need to take place to transfer a disk block (say 4KB) to the device. When the main

CPU is involved with the data movement (as in this example protocol), we refer to it as programmed I/O (PIO). Third, the OS writes a command to

the command register; doing so implicitly lets the device know that both the data is present and that it should begin working on the command.

Finally, the OS waits for the device to finish by again polling it in a loop, waiting to see if it is finished (it may then get an error code to indicate success or failure).

This basic protocol has the positive aspect of being simple and working.

However, there are some inefficiencies and inconveniences involved. The first problem you might notice in the protocol is that polling seems inefficient; specifically, it wastes a great deal of CPU time just waiting for

the (potentially slow) device to complete its activity, instead of switching to another ready process and thus better utilizing the CPU.

# Computer Architecture

# 2020-2021

## المرحلة الثالثة / الفصل الثاني

## أستاذ المادة

## ا.م.د.اسماء عبدالله فهد

## Interrupt driven

The invention that many engineers came upon years ago to improve this interaction is something we've seen already: the interrupt. Instead of polling the device repeatedly, the OS can issue a request, put the main process to sleep, and switch to execute another task.

An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing. The processor responds by suspending its current activities, saving its state, and executing a function called an **interrupt handler** (or an **Interrupt Service Routine, ISR**) to deal with the event. This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities.

The information that needs to be saved and restored typically includes the condition code flags and the contents of any registers used by both the interrupted program and the ISR.

Typically, the processor saves only the contents of the program counter and the processor status register. Any additional information that needs to be saved must be saved by program instruction at the beginning of the interrupt-service routine and restored at the end of the routine.

There are various things that need to be done when an interrupt is received:

1. Finish executing the current instruction, as shown in Figure (2).
2. "Recognize" the interrupt (determine which service routine is needed).
3. Save all the CPU register contents (PC, Registers, and Status Word) in memory. The Stack memory is used for this.
4. Jump to the routine, execute it, and return.
5. Restore the PC, registers and status word from the stack.
6. Continue with original program sequence, as if nothing had happened.

So this is mostly like jumping to a standard subroutine. However, as this is a routine which could be called at anytime and hence anywhere,

there are no parameters to be passed. In addition, notice that the registers and status word are saved. When a programmer writes a subroutine, it is assumed that the programmer will write it so that data being worked on is not lost. However, an interrupt routine is not called by the programmer but by the machine, and can occur at any time. The machine must make sure the CPU's state can be fully restored after the interrupt.

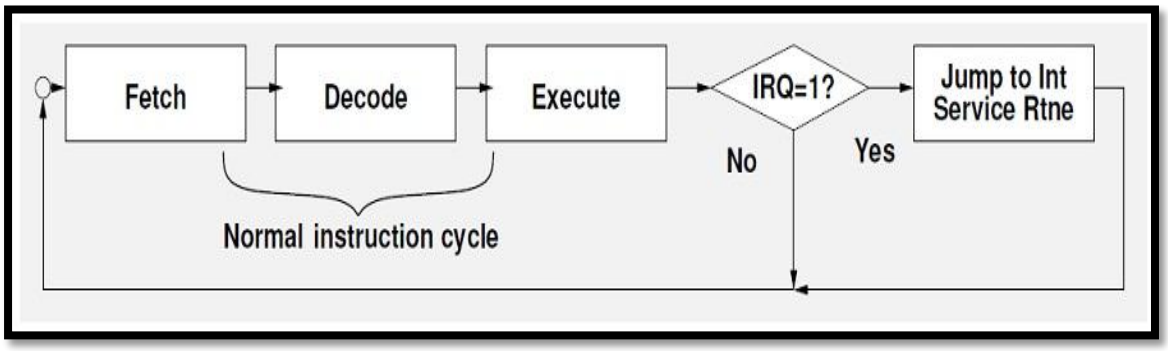



Figure (2) The Interrupt mechanism

**Enabling and Disabling Interrupts**

- Device activates interrupt signal line and waits with this signal activated until processor attends.
- The interrupt signal line is active during execution of ISR and till the device caused interrupt is serviced

**The sequence of events involved in handling an interrupt request from a single device**

Assuming that interrupts are enabled, the following is a typical scenario:

1. The device raises an interrupt request.

2. The processor interrupts the program currently being executed.
3. Interrupts are disabled
4. The device is informed that its request has been recognized, and in response, it deactivates the interrupt request signal.
5. The action requested by the interrupt is performed by the ISR.
6. Interrupts are enabled and execution of the interrupted program is resumed.

# Computer Architecture

# 2020-2021

## المرحلة الثالثة / الفصل الثاني

## أستاذ المادة

## ا.م.د.اسماء عبدالله فهد

## Interrupt Nesting

- Pre-Emption of low priority Interrupt by another high priority interrupt is known as Interrupt nesting.
- Disabling Interrupts during the execution of the ISR may not favor devices which need immediate attention.
- Need a priority of IRQ devices and accepting IRQ from a high priority device.
- The priority level of the processor can be changed dynamically.
- The privileged instruction write in the PS (processor status word), that encodes the processors priority.
- Organizing I/O devices in a prioritized structure.
- Each of the interrupt-request lines is assigned a different priority level.
- The processor is interrupted only by a high priority device.

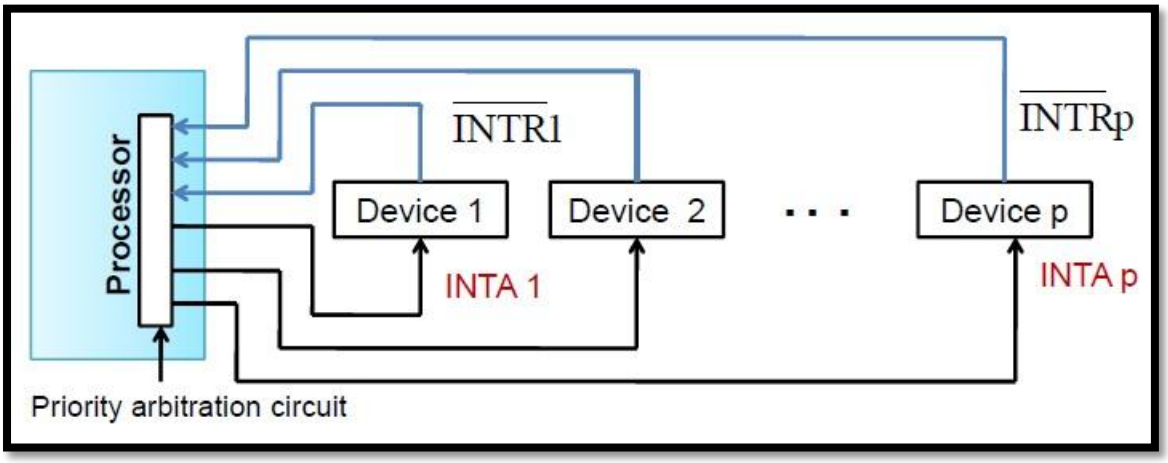



Figure (3) interrupt priority system

## Interrupt Operations

Interrupt operations fall into two classes; software or hardware initiated. Hardware interrupts can be classified as **non-maskable** or **maskable**.

## NON-Maskable Interrupt (NMI)

The processor provides a single non-maskable interrupt pin (NMI) which has higher priority than the maskable interrupt request pin (INTR). A typical use would be to activate a power failure routine.

## Maskable Interrupt (INTR)

The 8086 provides a single interrupt request input (INTR) which can be masked internally by software with the resetting of the interrupt enable FLAG status bit. During the interrupt response sequence further interrupts are disabled. The enable bit is reset as part of the response to any interrupt (INTR, NMI, software interrupt or single-step), although the FLAGS register which is automatically pushed onto the stack reflects the state of the processor prior to the interrupt. Until the old FLAGS register is restored the enable bit will be zero unless specifically set by an instruction.

# Computer Architecture

# 2020-2021

## المرحلة الثالثة / الفصل الثاني

## أستاذ المادة

## ا.م.د.اسماء عبدالله فهد

## Interrupt Vector Table (IVT)

IVT is a data structure that contains the address of the interrupt service routine. The IVT is located in the first 1024 (1 KB) of memory at address 000000H-0003FFH. It contains 256 different 4-byte interrupt vectors as shown in Figure (4). For each Interrupt, the following is stored:

1- Code segment register (16- bits)
2- Instruction Pointer (16-bits)

These two determine the memory location, i.e. where the ISR for the interrupt is located. The total 256 numbers of interrupts may be represented (each in 4 bytes) in the vector table. These are identified as type 0 to 255 or vector 0 to 255 interrupts. The first 32 (0-31 or 00-1F) of which are reserved for processor exceptions; the rest for hardware interrupts, software interrupts.

During the interrupt response a byte is fetched from the external interrupt system (8259A) which identifies the source (type) of the interrupt. This byte is multiplied by four and used as a pointer into the interrupt vector lookup table.

Vector table entry = N*4

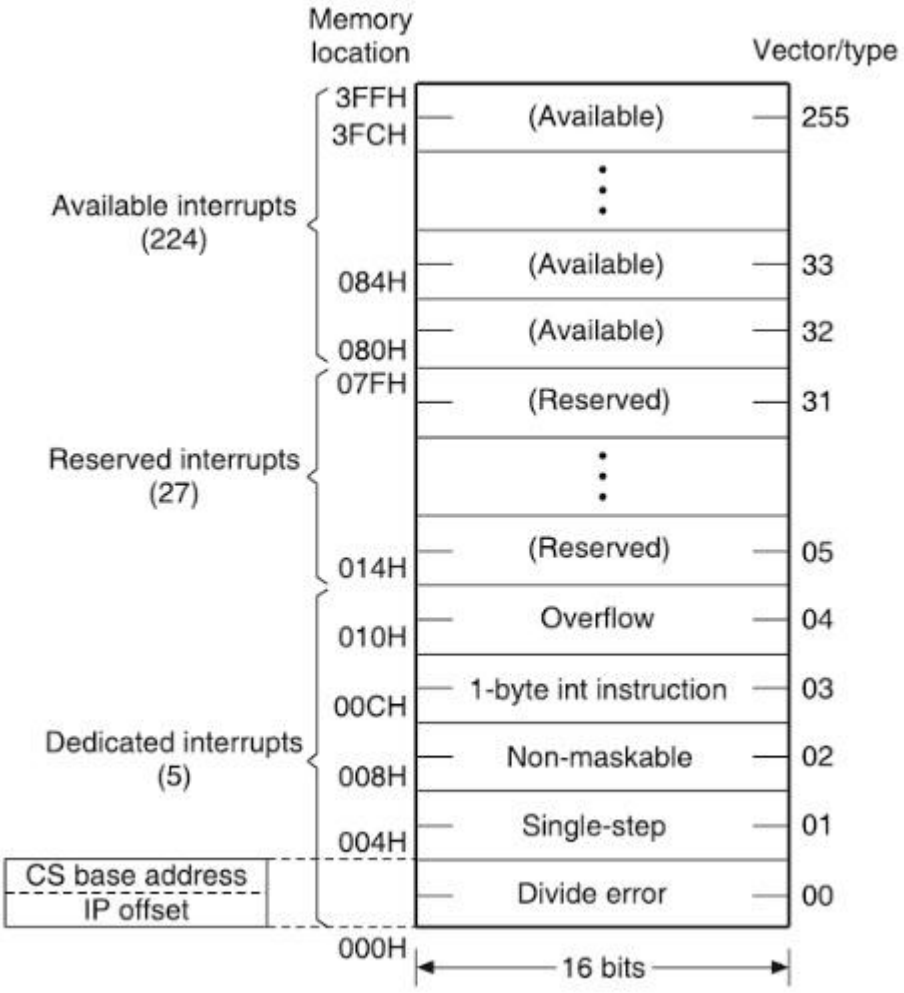



Figure (4) Interrupt Vector Table

## Intel x86 instructions related to Interrupt

All x86 processors provide the following instructions related to interrupts:

- INT nn: Interrupt. Run the ISR pointed by vector nn.
- INT 0 is reserved for the Divide Error
- INT 1 is reserved for Single Step operation
- INT 2 is reserved for the NMI pin
- INT 3 is reserved for setting a Breakpoint

- INT 4 is reserved for Overflow (Same as the INTO (Interrupt on overflow) instruction.
- CLI: Clear Interrupt Flag. IF is set to 0, thus interrupts are disabled.
- STI: Set Interrupt Flag. IF is set to 1, thus interrupts are enabled.
- IRET: Return from interrupt. This is the last instruction in the ISR (Real Mode only). It pops from the stack the Flag register, the IP and

  the CS. After returning from an ISR the interrupts are enabled, since the initial value of the flag register is poped from the stack.

## The 8259A Programmable Interrupt Controller

The Intel 8259 is a Programmable Interrupt Controller (PIC) designed for the Intel 8086 microprocessors. The 8259 combines multiple interrupt input sources into a single interrupt output to the host microprocessor, extending the interrupt levels available in a system beyond the one or two levels found on the processor chip, Figure (5).

The main signal pins on an 8259 are as follows:

- D0-D7: Bidirectional data connections
- IR0-IR7: Interrupt request inputs
- INT: Output, connects to μP INTR pin
- INTA´: Input, connects to μP INTA´ pin

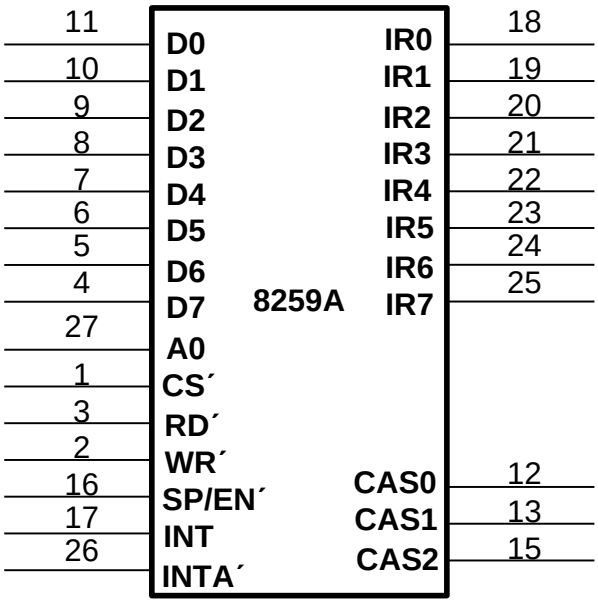



Figure (5) 8259 Interrupt Controler

# Computer Architecture

# 2020-2021

## المرحلة الثالثة / الفصل الثاني

## أستاذ المادة

## ا.م.د.اسماء عبدالله فهد

## Direct Memory Access Technique

Direct memory access (DMA) is a feature of computer systems that allows certain hardware subsystems to access main system memory (RAM), independent of the CPU.

Without DMA, when the CPU is using programmed input/output, it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU first initiates the transfer, then it does other operations while the transfer is in progress, and it finally receives an interrupt from the DMA controller when the operation is done. This feature is useful at any time that the CPU cannot keep up with the rate of data transfer, or when the CPU needs to perform useful work while waiting for a relatively slow I/O data transfer. Many hardware systems use DMA, including disk drive controllers, graphics cards, network cards and sound cards.

DMA can also be used for "memory to memory" copying or moving of data within memory. DMA can offload expensive memory operations, such as large copies operation.

### Basic Operation of DMA

When the processor wishes read or send a block of data, it issues a command to the DMA module by sending some information to DMA module. The information includes:

- Read or write command, sending through read and write control lines.
- Number of words to be read or written, communicated on the data lines and stored in the data count register.
- Starting location in memory to read from or write to, communicated on data lines and stored in the address register.
- Address of the I/O device involved, communicated on the data lines.

After the information is sent, the processor continues with other work. The DMA module then transfers the entire block of data directly to or from memory without going through the processor. When the transfer is

complete, the DMA module sends an interrupt signal to the processor to inform that it has finish using the system bus

## Advantages & Disadvantages of DMA

| | |
|---|---|
| **Advantages** | - allows a peripheral device to read from/write to memory without going through the CPU |
| | - allows for faster processing since the processor can be working on something else while the peripheral can be populating memory |
| **Disadvantages** | - requires a DMA controller to carry out the operation, which increases the cost of the system |
| | - cache coherence problems |

## Steps of DMA in terms of a data transfer from peripheral to memory:

1. Peripheral Device requests DMA service by pulling DREQ line high.
2. DMA controller requests CPU go on hold by pulling CPU's HOLD line high.
3. CPU finishes current bus cycle, then acknowledges with HLDA.The CPU hands over ALL bus control now to the DMA controller.
4. DMA Controller begins transferring data from peripheral to the Memory
    a. Controller sets the address of the memory where data is to be written.

   b. Controller reads data from the I/O Device and places the data on the data BUS.
   c. Controller Writes data from the data bus to the memory location.
   d. The Bus request is dropped, the HOLD pin goes Low, and the controller relinquishes the bus.
5. The Bus grant from the CPU is dropped and the HLDA pin goes Low.
6. The address register is incremented by 1.
7. The byte count is decremented by 1.
8. If the byte count is non-zero, return to step 2, otherwise stop.

# Computer Architecture

# 2020-2021

## المرحلة الثالثة / الفصل الثاني

## أستاذ المادة

## ا.م.د.اسماء عبدالله فهد

# Multiprocessor, Multiprogramming, and Multicore systems

## Multiprocessor system:

is an interconnection of two or more CPUs with memory and input— output equipment. The term "processor" in multiprocessor can mean either a central processing unit (CPU) or an input—output processor (lOP).

As it is most commonly defined, a multiprocessor system implies the existence of multiple CPUs, although usually there will be one or more IOPs as well.

Multiprocessors are classified as multiple instruction stream, multiple data stream (MIMD) systems.

There are some **similarities** between Multiprocessor and Multicomputer systems since both support concurrent operations. However, there exists important distinction between a system with multiple computers and a system with multiple processors.

• Computers are interconnected with each other means of communication lines to form a computer network.

• The network consists of several autonomous computers that may or may not communicate with each other.

• A multiprocessor system is controlled by one operating system that provides interaction between processors and all the components of the system cooperate in the solution of a problem.

***Multiprocessing*** improves the reliability of the system so that a failure or error in one part has a limited effect on the rest of the system. *If a fault causes one processor to fail, a second processor can be assigned to perform the functions of the disabled processor.* The system as a whole can continue to function correctly with perhaps some loss in efficiency.

• The benefit derived from a multiprocessor organization is an improved system performance.

• The system derives its high performance from the fact that computations can proceed in parallel in one of two ways.

1. Multiple independent jobs can be made to operate in parallel.

2. A single job can be partitioned into multiple parallel tasks.

***Tightly coupled***

Multiprocessors are classified by the way their memoryis organized.

• A multiprocessor system with common shared memory is classified as a *shared-memory* or *tightly coupled multiprocessor*.

• In fact, most commercial tightly coupled Tightly coupled multiprocessors provide *a cache memory* with each CPU.

• In addition, there is a global common memory that all CPUs can access.

• Information can therefore be shared among the CPUs by placing it in the *common global memory*.

***Loosely coupled***

• An alternative model of microprocessor is the *distributed-memory or loosely coupled system*. Each processor element in a loosely coupled system has its own private local memory.

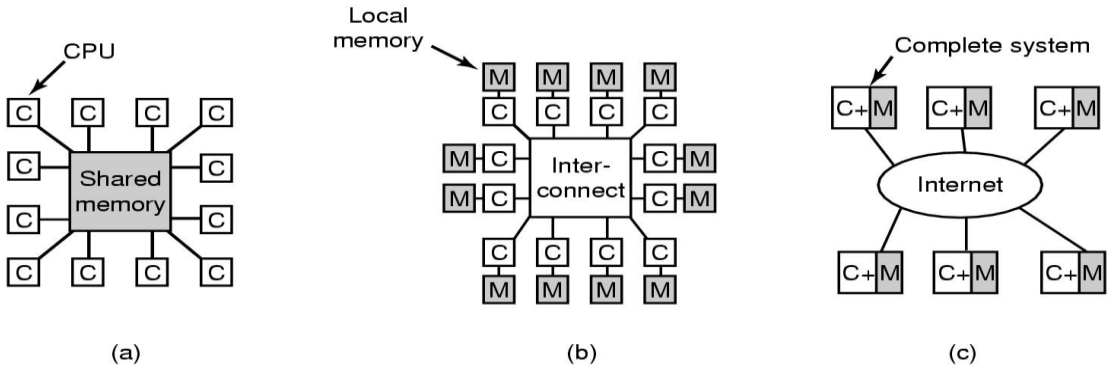
• The processors are tied together by a switching scheme designed to route information from one processor to another through a *message passing scheme.*

• The processors relay program and data to other processors in packets.

• A packet consists of an address, the data content, and some error detection code.

• The packets are addressed to a specific processor or taken by the first available processor, depending on the communication system used.

• Loosely coupled systems are most efficient when the interaction between tasks is minimal, whereas tightly coupled systems can tolerate a higher degree of interaction between tasks.

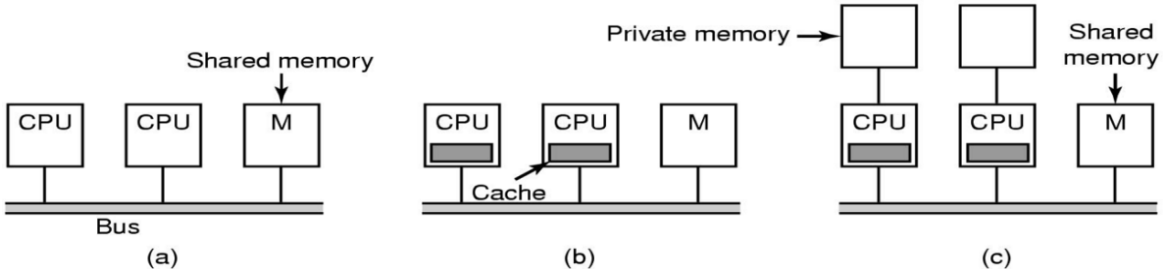



Multiprocessor Multicomputer Distributed System

- Continuous need for faster and powerful computers
  - shared memory model ( access nsec)
  - message passing multiprocessor (access microsec)
  - wide area distributed system (access msec)

Multiprocessor also can be defined as a computer system in which two or more CPUs share full access to a common RAM

## Multiprocessor Hardware (1)




Bus-based multiprocessors

memory coherence

# Computer Architecture

# 2020-2021

## المرحلة الثالثة / الفصل الثاني

## أستاذ المادة

## ا.م.د.اسماء عبدالله فهد

## Multiprogramming system

several programs running at a time. Like Windows..with many programs open.

Why Multiprogramming? When there is a single program running in the CPU, it leads to the degradation of the CPU utilization.

**Advantages :-**

1.highCPUutilization.

2.mainmemoryutilization.

3.the allocation of a computer system and its resources to more than one concurrent application.

**Disadvantages :-**

1-If we have not much resource then it will not work properly, or it will badly affect the performance of OS.

2-We need CPU scheduling

## Multi-Core Architecture

A multi-core processor is a single computing component with two or more independent actual processing units (called "cores"), which are the units that read and execute program instructions as shown in Figure-1. The instructions are ordinary CPU instructions such as add, move data, and branch, but the multiple cores can run multiple instructions at the same time, increasing overall speed for programs amenable to parallel computing. Manufacturers typically integrate the cores onto a single integrated circuit die (known as a chip multiprocessor or CMP), or onto multiple dies in a single chip package.

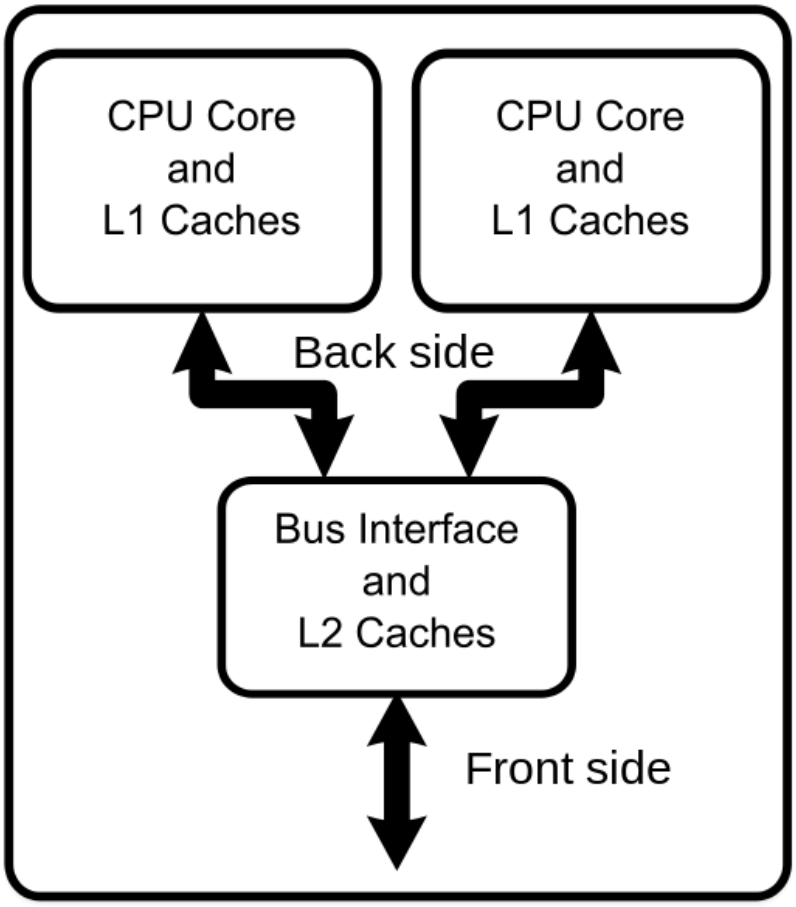



Figure-1 Two independent core architecture

Multi-core processors may have two cores (dual-core CPUs, for example, AMD Phenom II X2 and Intel Core Duo), three cores (tri-core CPUs, for example, AMD Phenom II X3), four cores (quad-core CPUs, for example, AMD Phenom II X4, Intel's i5 and i7 processors), six cores (hexa-core CPUs, for example, AMD Phenom II X6 and Intel Core i7 Extreme Edition 980X), eight cores (octa-core CPUs, for example, Intel Xeon E7-2820 and AMD FX-8350), ten cores (deca-core CPUs, for example, Intel Xeon E7-2850), or more.

A multi-core processor implements multiprocessing in a single physical package. Designers may couple cores in a multi-core device tightly or loosely. For example, cores may or may not share caches, and they may implement message passing or shared-memory inter-core communication methods. Common network topologies to interconnect cores include bus, ring, two-dimensional mesh, and crossbar. Homogeneous multi-core systems include only

identical cores, heterogeneous multi-core systems have cores that are not identical.

The **Intel Core microarchitecture** (previously known as the Next-Generation Micro-**Architecture**) is a multi-**core** processor **microarchitecture** and can be considered an iteration of the P6 **microarchitecture**, introduced in 1995 with Pentium Pro.

# Computer Architecture

# 2020-2021

## المرحلة الثالثة / الفصل الثاني

## أستاذ المادة

## ا.م.د.اسماء عبدالله فهد

# Memory System

Without a memory no information can be stored or retrieved in a computer. Computer memory has to be organized in a hierarchy. In such a hierarchy, larger and slower memories are used to supplement smaller and faster ones. This observation has since then proven essential in constructing a computer memory. If we put aside the set of CPU registers (as the first level for storing and retrieving information inside the CPU), hierarchy starts with a small, expensive, and relatively fast unit, called the **cache**. The cache is followed in the hierarchy by a larger, less expensive, and relatively slow **main memory** unit. Cache and main memory are built using solid-state semiconductor material. They are followed in the hierarchy by far larger, less expensive, and much slower magnetic memories that consist typically of the (hard) **disk** and the **tape**.

## 1. Memory Hierarchy

As mentioned above, a typical memory hierarchy starts with a small, expensive, and relatively fast unit, called the cache, followed by a larger, less expensive, and relatively slow main memory unit. Cache and main memory are built using solid- state typically CMOS transistors). The fast memory level the primary memory. Larger, less expensive, and far slower magnetic memories that consist typically of the (hard) disk called the secondary memory.
Memory hierarchy can be characterized by a number of parameters.

- **The access type**: The term access refers to the action that physically takes place during a read or writes operation.
- **Capacity level**: This is usually measured in bytes.
- **Cycle time**: the time elapsed from the start of a read operation to the start of a subsequent read
- **Latency**: the time interval between the request for information and the access to the first bit of that information.
- **Band width**: measure of the number of bits per second that can be accessed.
- **Cost**: dollars per megabytes.

The term random access refers to the fact that any access to any memory location takes the same fixed amount of time regardless of the actual memory location

For Example,If a write operation to memory location 100 takes 15 ns and if this operation is followed by a read operation to memory location 3000, then the latter operation will also take 15 ns. This is to be compared to sequential access in which if access to location 100 takes 500 ns, and if a consecutive access to location 101 take s 505 ns, then it is expected that an access to location 300 may take 1500 ns. This is because the memory has to cycle through locations 100 to 300, with each location requiring 5 ns.

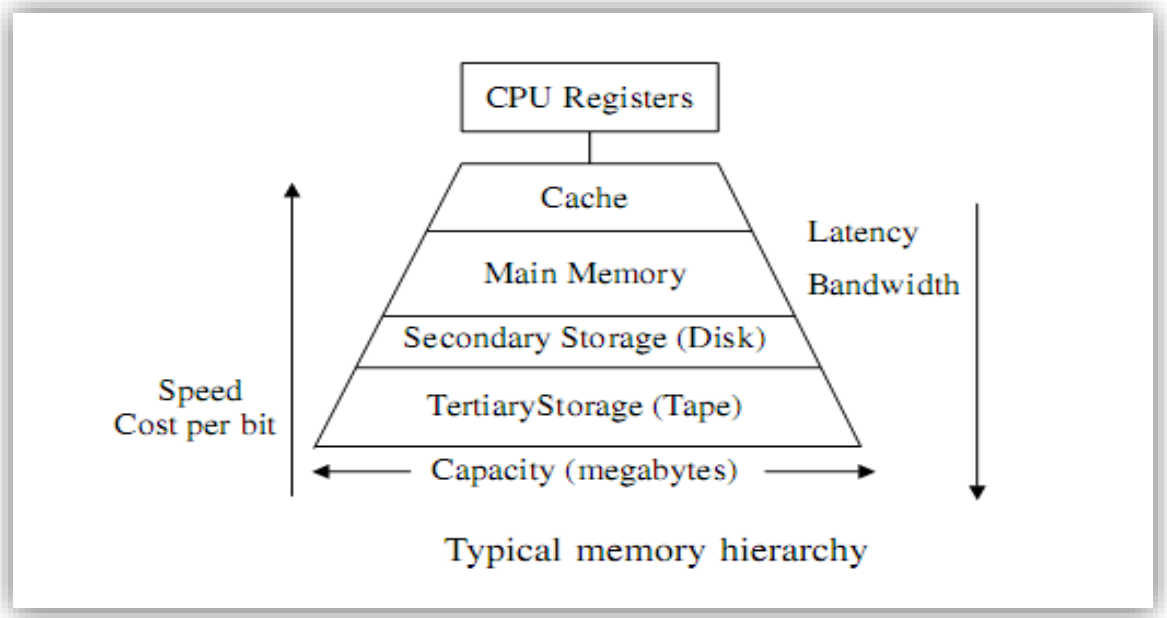



Figure 1Typical Memory Hierarchy

**Memory Hierarchy Parameters**

| | Access type | Capacity | Latency | Bandwidth | Cost/MB |
|---|---|---|---|---|---|
| CPU registers | Random | 64–1024 bytes | 1–10 ns | System clock rate | High |
| Cache memory | Random | 8–512 KB | 15–20 ns | 10–20 MB/s | $500 |
| Main memory | Random | 16–512 MB | 30–50 ns | 1–2 MB/s | $20–50 |
| Disk memory | Direct | 1–20 GB | 10–30 ms | 1–2 MB/s | $0.25 |
| Tape memory | Sequential | 1–20 TB | 30–10,000 ms | 1–2 MB/s | $0.025 |

The effectiveness of a memory hierarchy depends on the principle of moving information into the fast memory infrequently and accessing it many times before replacing it with new information. This principle is possible due to a phenomenon called *locality of reference;* that is, within a given period of time, programs tend to reference a relatively confined area of memory repeatedly.

*Locality of reference; property*

1. *Fetch instruction*
2. *Fetch data*

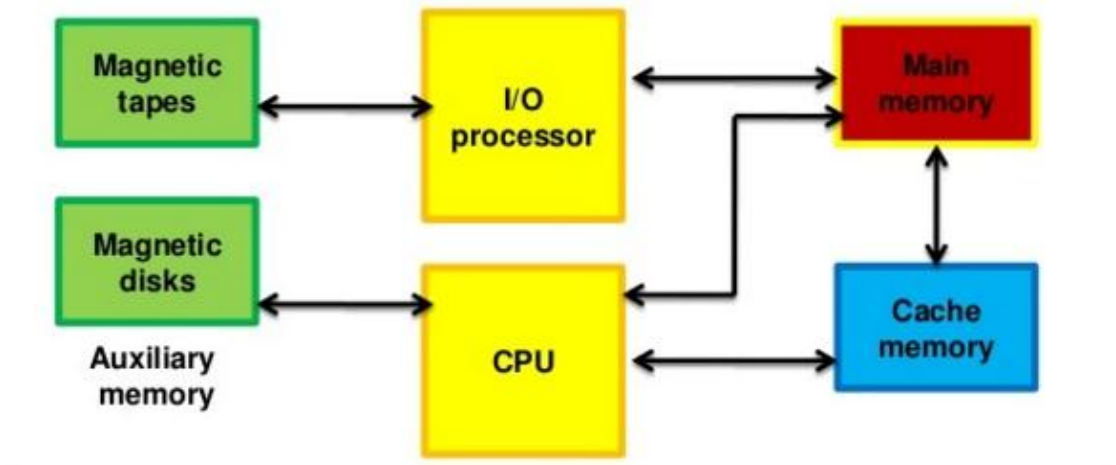



Figure (2) Memory Hierarchy in Computer System

Figure 2 illustrates the components in a typical memory hierarchy. At the bottom of the hierarchy are the relatively slow magnetic tapes used to store removable files. Next are the magnetic disks used as backup storage. The main memory occupies a central position by being able to communicate directly with the CPU and with auxiliary memory devices through an I/0 processor. When programs not residing in main memory are 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 for currently used programs and data.

# Computer Architecture

# 2020-2021

## المرحلة الثالثة / الفصل الثاني

## أستاذ المادة

## ا.م.د.اسماء عبدالله فهد

## Associative Memory

Many data-processing applications require the search of items in a table stored in memory. An assembler program searches the symbol address table in order to extract the symbol's binary equivalent. An account number may be searched in a file to determine the holder's name and account status. The established way to search a table is to store all items where they can be addressed in sequence. The time required to find an item stored in memory can be reduced considerably if stored data can be identified for access by the content of the data itself rather than by an address. A memory unit accessed by content is called an associative memory or content addressable memory (CAM). This type of memory is accessed simultaneously and in parallel on the basis of data content rather than by specific address or location. Moreover, searches can be done on an entire word or on a specific field within a word. An associative memory is more expensive than a random access memory because each cell must have storage capability as well as logic circuits for matching its content with an external argument. For this reason, associative memories are used in applications where the search time is very critical and must be very short.

## 1. Cache Memory

If the active portions of the program and data are placed in a fast small memory, the average memory access time can be reduced, thus reducing the total execution time of the program. Such a fast small memory is referred to as a cache memory. It is placed between the CPU and main memory as illustrated in Fig. 2. The cache memory access time is less than the access time of main memory by a factor of 5 to 10. The cache is the fastest component in the memory hierarchy and approaches the speed of CPU components.

The basic operation of the cache is as follows. When the CPU needs to access memory, the cache is examined. If the word is found in the cache, it is read from the fast memory. If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word. A block of words containing the one just accessed is then transferred from main memory to cache memory. In this manner, some data are transferred to cache so that future references to memory find the required words in the fast cache memory. The performance of cache memory is frequently 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 and it counts as a miss. The ratio of the number of hits divided by the total CPU references to memory (hits plus misses) is the hit ratio.

$$\text{Hit ratio} = \text{hit} / (\text{hit} + \text{mis})$$

We define a cache miss to be a reference to an item that is not resident in cache, but is resident in main memory.

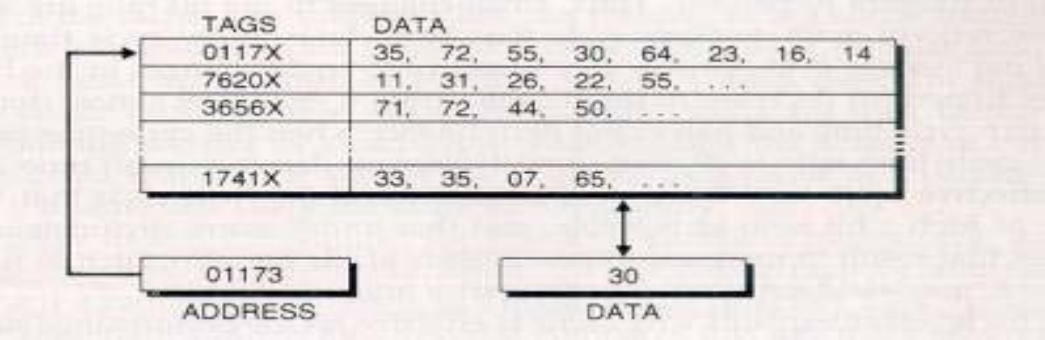



Figure (3) A cache-memory reference. The tag 0117X matches address 01173, so the cache returns the item in the position X=3 of the matched block

Figure 3 shows the structure of a typical cache memory. Each reference to a cell in memory is presented to the cache. The cache searches its directory of address tags shown in the figure to see if the item is in the cache. If the item is not in the cache, a miss occurs.

For READ operations that cause a cache miss, the item is retrieved from main memory and copied into the cache. During the short period available before the main-memory operation is complete, some other item in cache is removed from the cache to make room for the new item.

The cache-replacement decision is critical; a good replacement algorithm can yield somewhat higher performance than can a bad replacement algorithm.

In Fig.3 we show an item in the cache surrounded by nearby items, all of which are moved into and out of the cache together. We call such a group of data a block of the cache.

## 1.1 Cache Memory Organizations

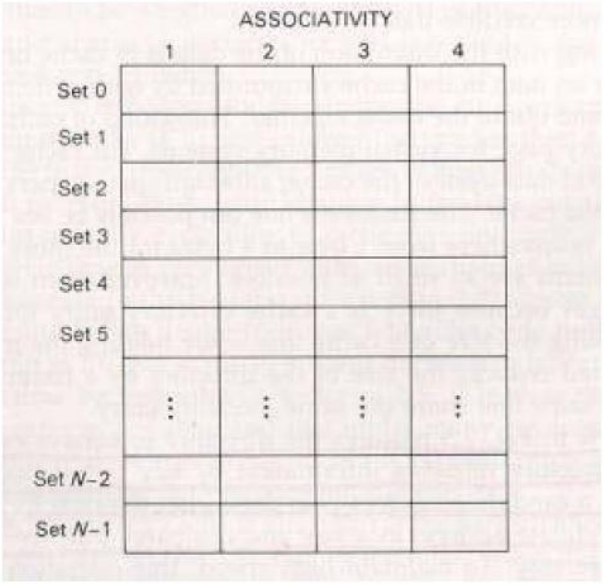



Figure 4: The logical organization of a four-way set-associate cache

Fig.4 shows a conceptual implementation of a cache memory. This system is called set associative because the cache is partitioned into distinct sets of blocks, and each set contains a small fixed number of blocks. The sets are represented by the rows in the figure. In this case, the cache has N sets, and each set contains four blocks. When an access occurs to this cache, the cache controller does not search the entire cache looking for a match. Instead, the controller maps the address to a particular set of the cache and searches only the set for a match.

Fig.4 is only one example, there are various ways that a cache can be arranged internally to store the cached data. In all cases, the processor reference the cache with the main memory address of the data it wants. Hence each cache organization must use this address to find the data in the cache if it is stored there, or to indicate to the processor when a miss has occurred. The problem of mapping the information held in the main memory into the cache must be totally implemented in hardware to achieve improvements in the system operation. Various strategies are possible.

# Computer Architecture

# 2020-2021

## المرحلة الثالثة / الفصل الثاني

## أستاذ المادة

## ا.م.د.اسماء عبدالله فهد

- **Fully associative mapping**

Perhaps the most obvious way of relating cached data to the main memory address is to store both memory address and data together in the cache. This the fully associative mapping approach. A fully associative cache requires the cache to be composed of associative memory holding both the memory address and the data for each cached line. The incoming memory address is simultaneously compared with all stored addresses using the internal logic of the associative memory, as shown in Fig.5. If a match is found, the corresponding data is read out. Single words form anywhere within the main memory could be held in the cache, if the associative part of the cache is capable of holding a full address

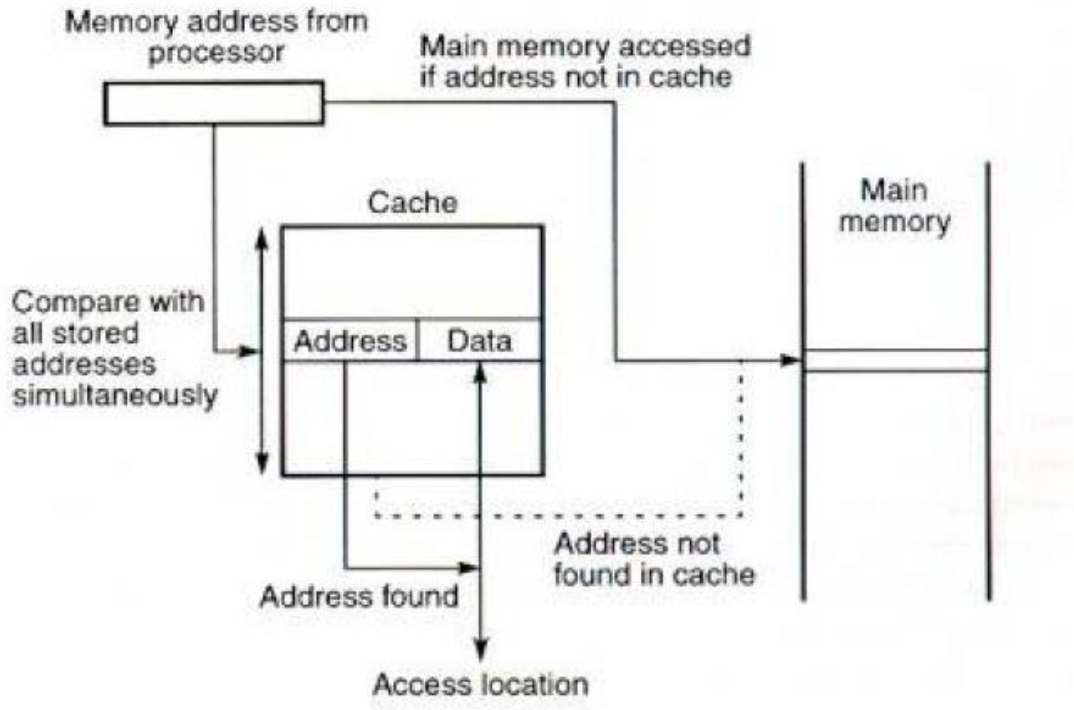



Fig.5 Cache with fully associative mapping

In all organizations, the data can be more than one word, i.e., a block of consecutive locations to take advantage of spatial locality. In Fig.6 a line constitutes four words, each word being 4 bytes. The least significant part of the address selects the particular byte, the next part selects the word, and the remaining bits form the address compared to the address in the cache. The whole line can be transferred to and from the cache in one transaction if there are sufficient data paths between the main memory and the cache. With only one data word path, the words of the line have to be transferred in separate transactions.

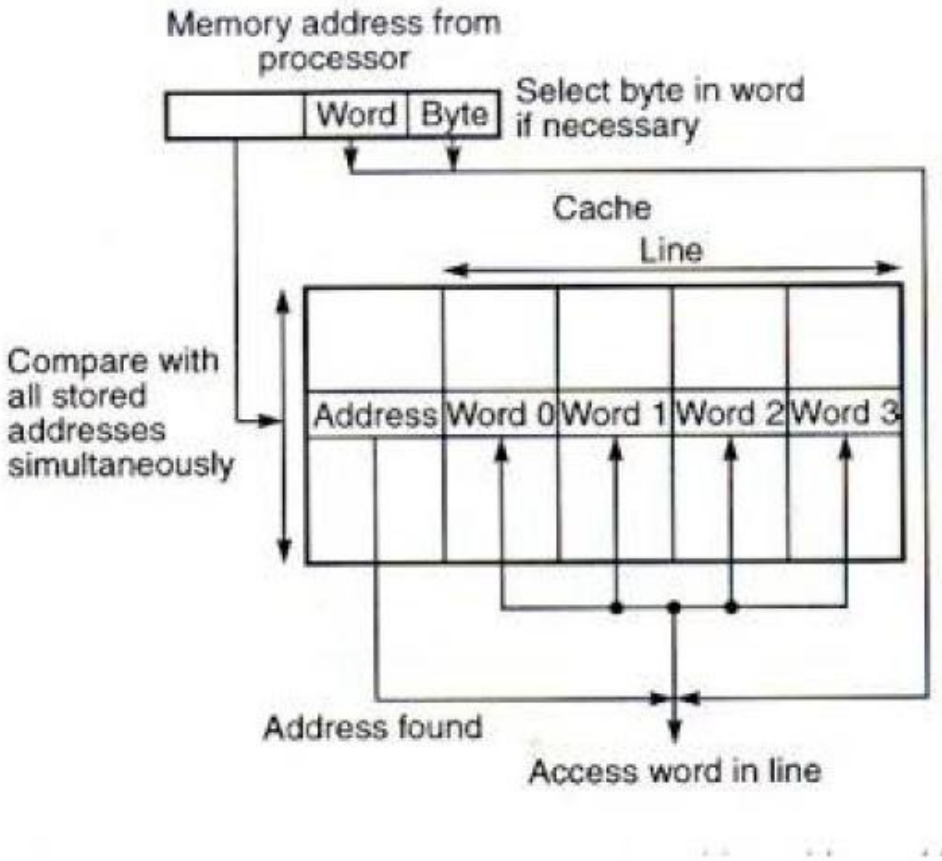



Fig.6 Fully associative mapped cache with multi-word lines

The fully associate mapping cache gives the greatest flexibility of holding combinations of blocks in the cache and minimum conflict for a given sized cache, but is also the most expensive, due to the cost of the associative memory. It requires a replacement algorithm to select a block to remove upon a miss and the algorithm must be implemented in hardware to maintain a high speed of operation. The fully associative cache can only be formed economically with a moderate size capacity. Microprocessors with small internal caches often employ the fully associative mechanism.

- **Direct mapping**

The fully associative cache is expensive to implement because of requiring a comparator with each cache location, effectively a special type of memory. In direct mapping, the cache consists of normal high speed random access memory, and each location in the cache holds the data, at an address in the cache given by the lower significant bits of the main memory address. This enables the block to be selected directly from the lower significant bits of the memory address. The remaining higher

significant bits of the address are stored in the cache with the data to complete the identification of the cached data.

Consider the example shown in Fig.7. The address from the processor is divided into two fields, a tag and an index. The tag consists of the higher significant bits of the address, which are stored with the data. The index is the lower significant bits of the address used to address the cache.

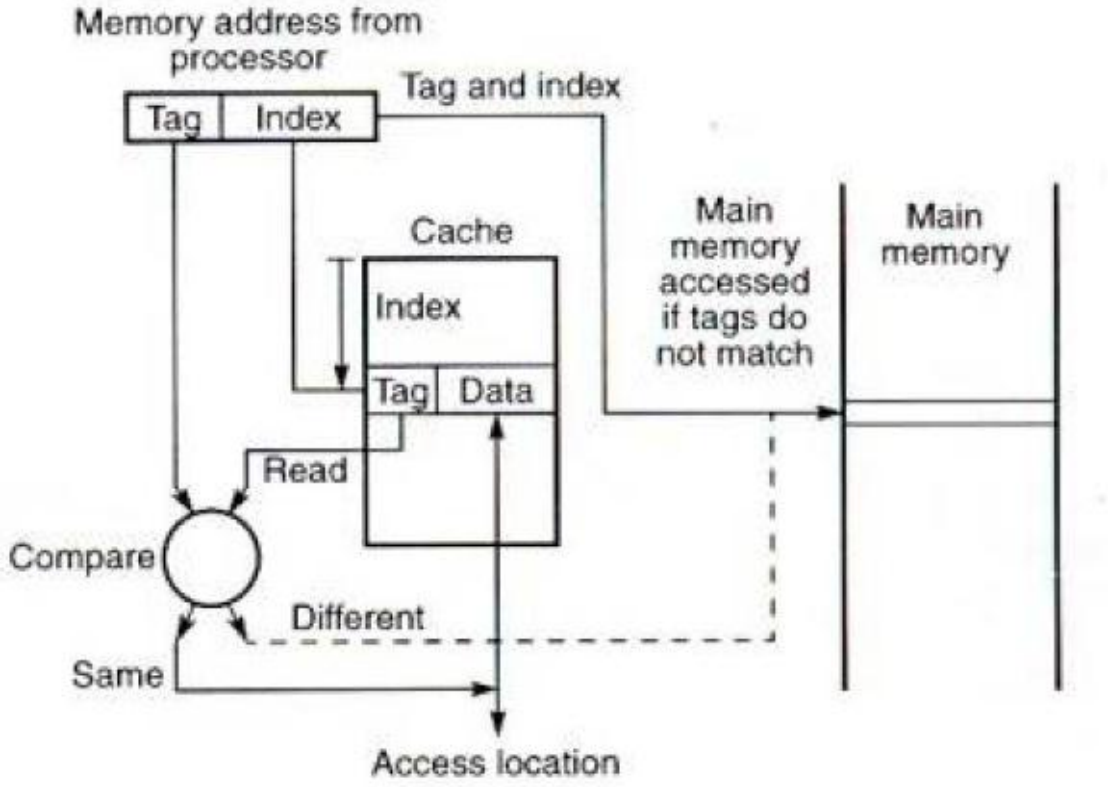



Fig.7 Cache with direct mapping

When the memory is referenced, the index is first used to access a word in the cache. Then the tag stored in the accessed word is read and compared with the tag in the address. If the two tags are the same, indicating that the word is the one required, access is made to the addressed cache word. However, if the tags are not the same, indicating that the required word is not in the cache, reference is made to the main memory to find it. For a memory read operation, the word is then transferred into the cache where it is accessed. It is possible to pass the information to the cache and the processor simultaneously, i.e., to read-through the cache, on a miss. The cache location is altered for a write operation. The main memory may be altered at the same time (write-through) or later.

Fig 8. shows the direct mapped cache with a line consisting of more than one word. The main memory address is composed of a tag, an index, and a word within a line. All the

words within a line in the cache have the same stored tag. The index part to the address is used to access the cache and the stored tag is compared with required tag address. For a read operation, if the tags are the same the word within the block is selected for transfer to the processor. If the tags are not the same, the block containing the required word is first transferred to the cache.

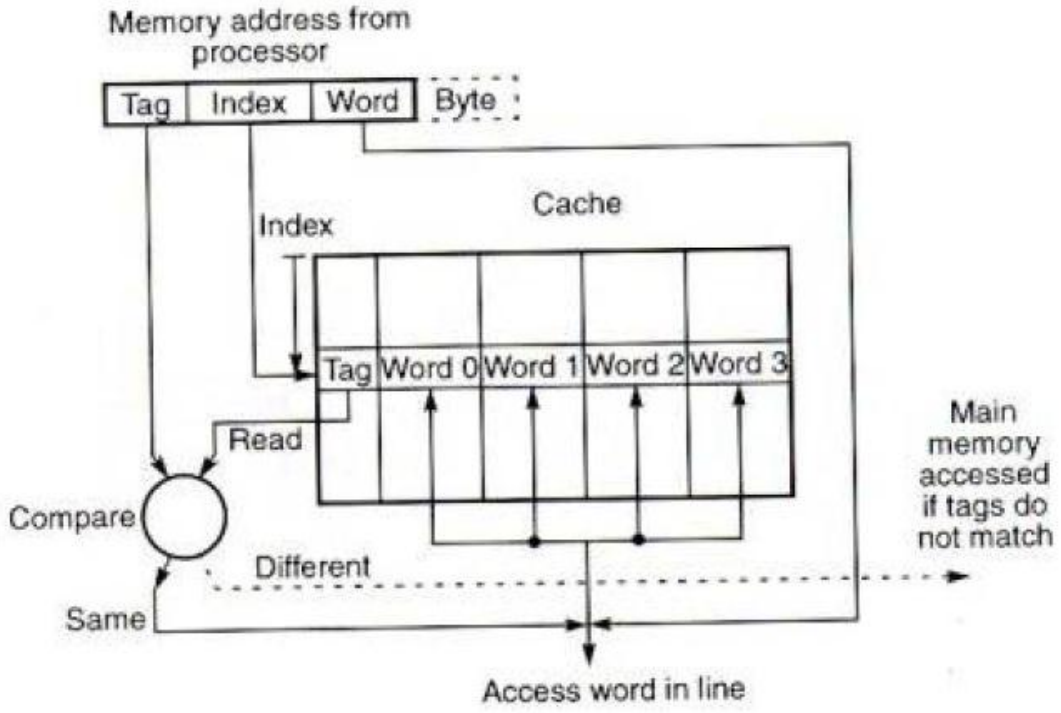



Fig.8 Direct mapped cache with a multi-word block

In direct mapping, the corresponding blocks with the same index in the main memory will map into the same block in the cache, and hence only blocks with different indices can be in the cache at the same time. A replacement algorithm is unnecessary, since there is only one allowable location for each incoming block. Efficient replacement relies on the low probability of lines with the same index being required. However there are such occurrences, for example, when two data vectors are stored starting at the same index and pairs of elements need to processed together. To gain the greatest performance, data arrays and vectors need to be stored in a manner which minimizes the conflicts in processing pairs of elements. Fig.8 shows the lower bits of the processor address used to address the cache location directly. It is possible to introduce a mapping function between the address index and the cache index so that they are not the same.

- **Set-associative mapping**

In the direct scheme, all words stored in the cache must have different indices. The tags may be the same or different. In the fully associative scheme, blocks can displace any

other block and can be placed anywhere, but the cost of the fully associative memories operate relatively slowly.

Set-associative mapping allows a limited number of blocks, with the same index and different tags, in the cache and can therefore be considered as a compromise between a fully associative cache and a direct mapped cache. The organization is shown in Fig.9. The cache is divided into "sets" of blocks. A four-way set associative cache would have four blocks in each set. The number of blocks in a set is know as the associativity or set size. Each block in each set has a stored tag which, together with the index, completes the identification of the block. First, the index of the address from the processor is used to access the set. Then, comparators are used to compare all tags of the selected set with the incoming tag. If a match is found, the corresponding location is accessed, otherwise, as before, an access to the main memory is made.

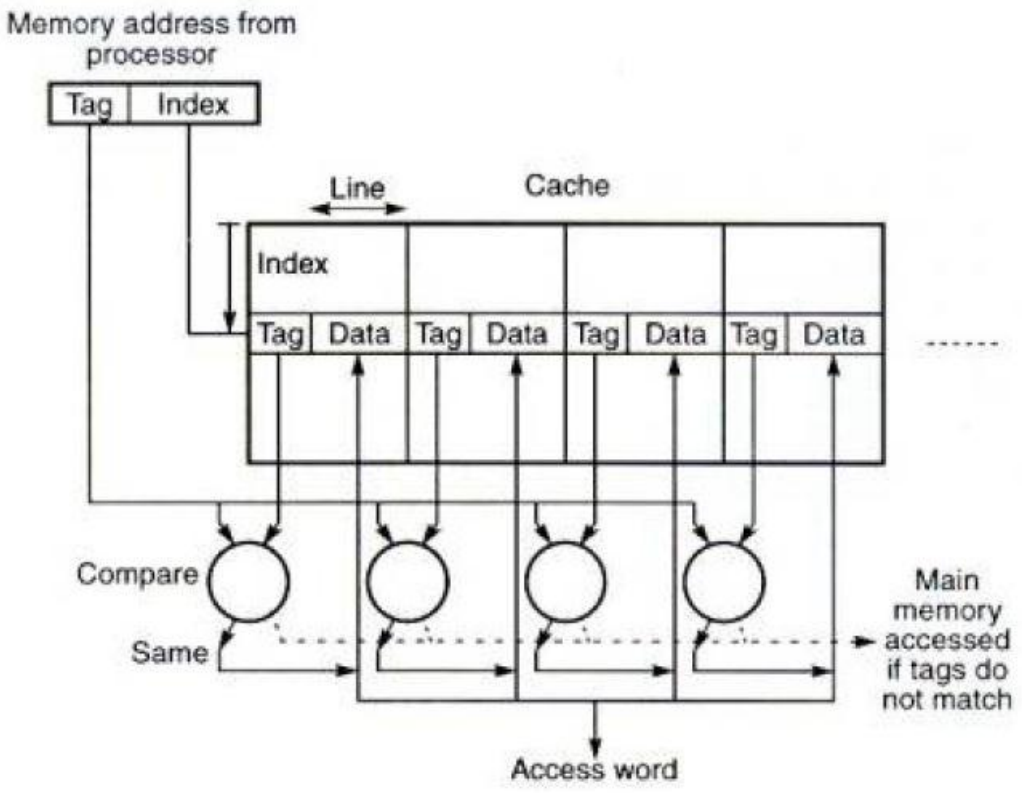



**Fig.9 Cache with set-associative mapping**

The tag address bits are always chosen to be the most significant bits of the full address, the block address bits are the next significant bits and the word/byte address bits form the least significant bits as this spreads out consecutive man memory blocks throughout

consecutive sets in the cache. This addressing format is known as bit selection and is used by all known systems. In a set-associative cache it would

be possible to have the set address bits as the most significant bits of the address and the block address bits as the next significant, with the word within the block as the least significant bits, or with the block address bits as the least significant bits and the word within the block as the middle bits.

Notice that the association between the stored tags and the incoming tag is done using comparators and can be shared for each associative search, and all the information, tags and data, can be stored in ordinary random access memory. The number of comparators required in the set-associative cache is given by the number of blocks in a set, not the number of blocks in all, as in a fully associative memory. The set can be selected quickly and all the blocks of the set can be read out simultaneously with the tags before waiting for the tag comparisons to be made. After a tag has been identified, the corresponding block can be selected.

The replacement algorithm for set-associative mapping need only consider the lines in one set, as the choice of set is predetermined by the index in the address. Hence, with two blocks in each set, for example, only one additional bit is necessary in each set to identify the block to replace.