Control Unit

Control Unit basics

The main function of a computer is to execute ***programs.***

The execution of a program consists of a sequential execution of ***instructions*****.**

Each instruction is executed during an **instruction cycle** made up of shorter subcycles (**fetch, execute, interrupt**). The performance of each subcycle involves one or more shorter operations, that is, **micro-operations.**

Micro-operations are functional or atomic operations of a processor – a transfer between registers, a transfer between registers and external bus, a simple arithmetic or logic operation (shift, add, negate).

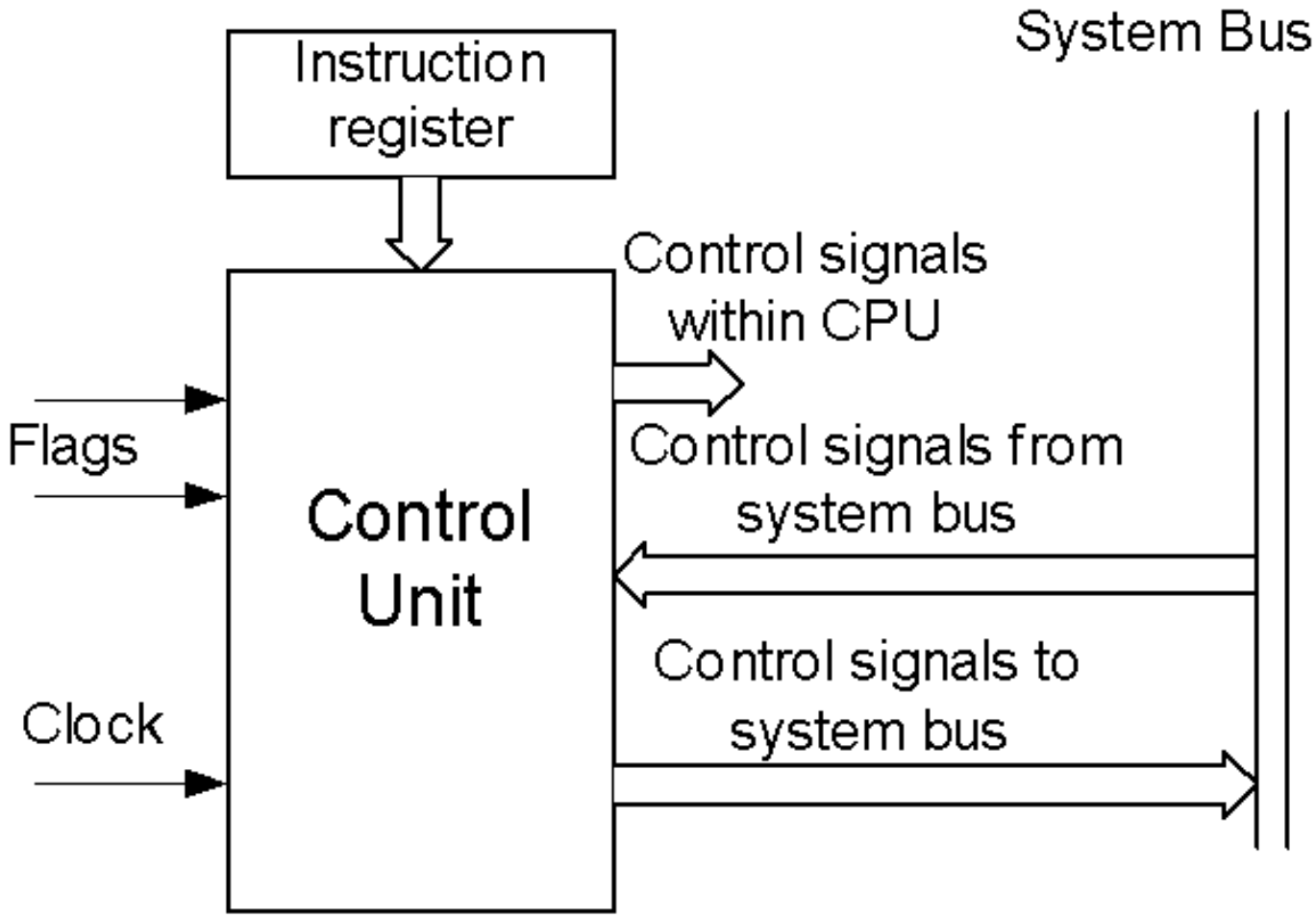
The control unit is the main component that directs the system operations by sending control signals to the datapath.

These signals control the flow of data within the CPU and between the CPU and external units such as memory and I/O.

The control unit performs two basic tasks:

- **Sequencing** – the control unit causes the processor to step through a series of micro-operations in the proper sequence, based on the program being executed
- **Execution** – The control unit causes each micro-operation to be performed.

A general model of a control Unit:



Inputs:

- **Clock** – One or several micro-operations are executed at one clock pulse. It is called a processor cycle.
- **Instruction register** – The opcode of the current instruction is used to determine which micro-operations to perform.
- **Flags** – Are needed to determine the status of the processor and outcome of previous ALU operations.
- **Control signals from Control Bus** – interrupt signals, acknowledgments.

Outputs:

- **Control signals within the CPU** – These are two types: those that cause data to be moved from one register to another and those that activates specific CPU functions.
- **Control signals to Control Bus** – also two types: control signals to memory and control signals to I/O system.

Harwired Control Unit

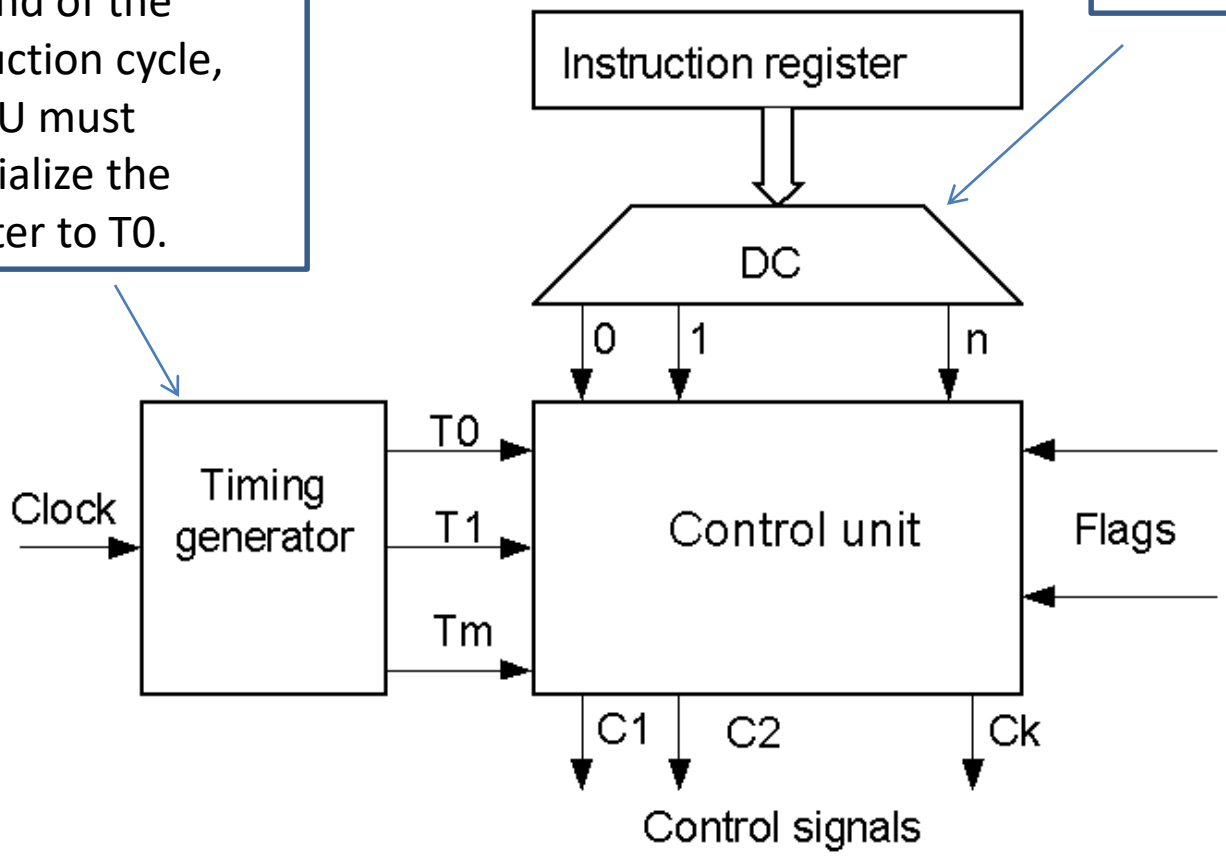
In hardwired control, fixed logic circuits that correspond directly to the Boolean expressions are used to generate the control signals.

- *Advantage:* Hardwired control is very fast and CU has a small size.
- *Disadvantage:* Hardwired control could be very expensive and complicated for complex systems. It will require a redesign of the entire systems in the case of any change (ex. add a new instruction).

Timing generator is a counter of a clock pulses. At the end of the instruction cycle, the CU must reinitialize the counter to T0.

To simplify the CU logic, there should be a unique logic input for each opcode. This function is performed by a decoder which takes an encoded input and produces a single output.

General structure:



Example: Assume that the instruction set of a machine has the three instructions**: x, y, and z**; and A, B, C, D, E, F, G, and H are signals that should be generated for the three instructions at the three steps T0 , T1 , and T2.

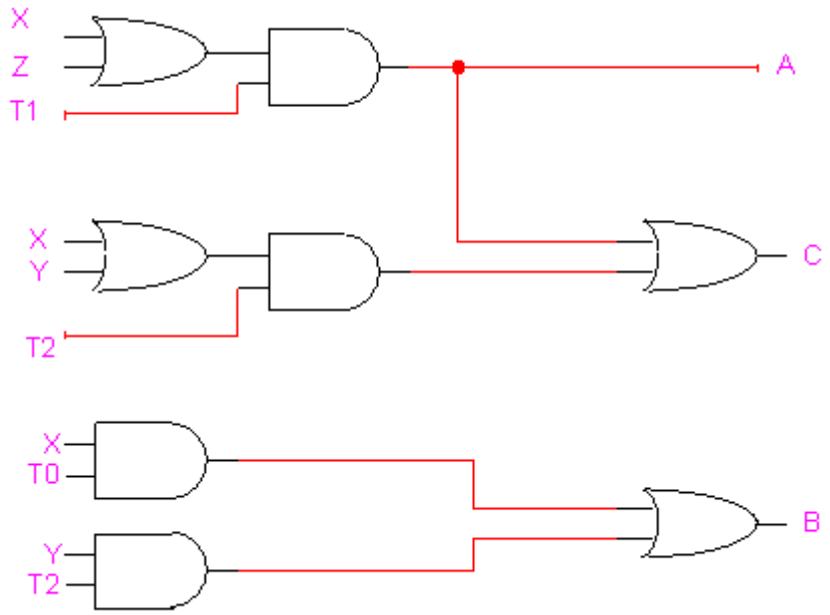
Step	Istruction X	Istruction Y	Istruction Z
T0	D,B,E	F,H,G	E,H
T1	C,A.H	G	D,A,C
T2	G,C	B,C	-

The Boolean expressions for control signals A, B, and C can be obtained as follows:

A=X*T1+Z*T1=(X+Z)*T1

B=X*T0+Y*T2

C=X*T1+ Z*T1+ X*T2+Y*T2=(X+Z)*T1+ (X+Y)*T2



Microprogrammed Control Unit

The idea of microprogrammed control is to store the control signals associated with the implementation of a certain instruction as a microprogram in a special memory called a **control memory (CM).**

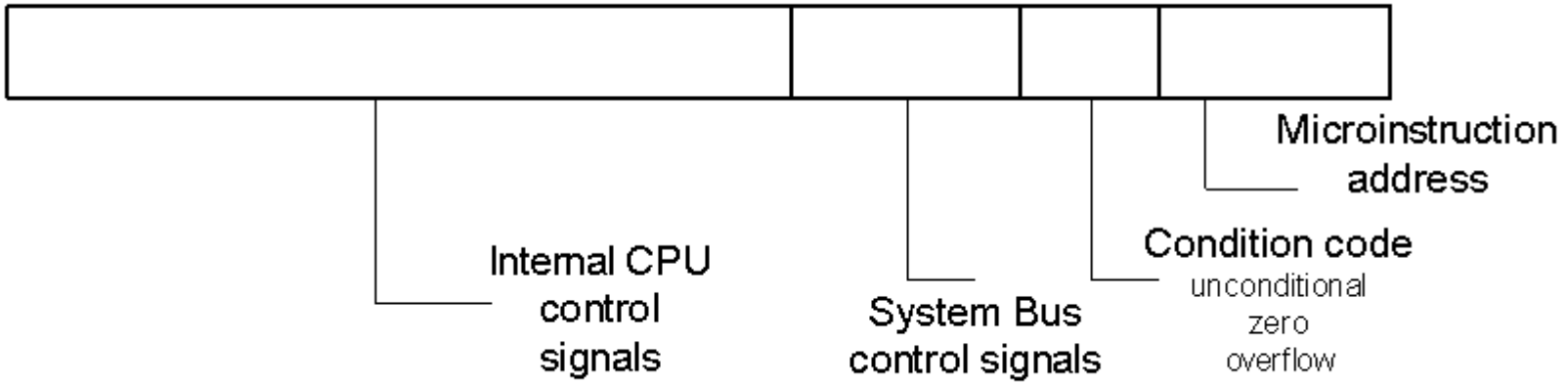
- *Advantage:* It is flexible and could adapt easily to changes in the system design. We can easily add new instructions without changing hardware.
- *Disadvantage:* It is slower than a hardwired control unit of comparable technology.

Microprogramming is the dominant technique in CISC processors, hardwired CU – in RISC processors.

- A **microprogram** is written in a microprogramming language and consists of a sequence of microinstructions.
- A **microinstruction** is a vector of bits, where each bit is a ***control signal***, ***condition code*** and the ***address of the next microinstruction***.

  Microinstructions can be classified as ***horizontal*** or ***vertical***.

Horizontal microinstruction



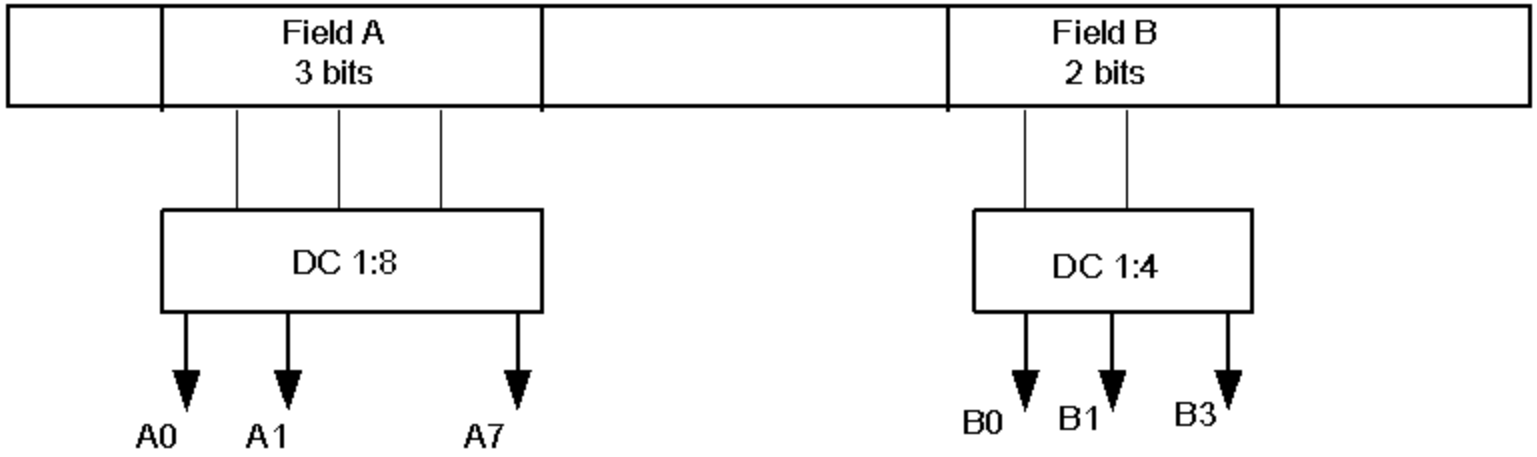
Individual bits in horizontal microinstructions correspond to individual control lines.

If the control bit is equal to 1 – the control line is turned on, if the bit is equal to 0 – the control line is leaved of.

If the condition code is false – the next instruction in the sequence is executed. If the condition is true – the address of the next microinstruction to be executed is indicated in the address field.

Horizontal microinstructions are long and allow **maximum parallelism** since each bit controls a single control line.

Vertical microinstruction

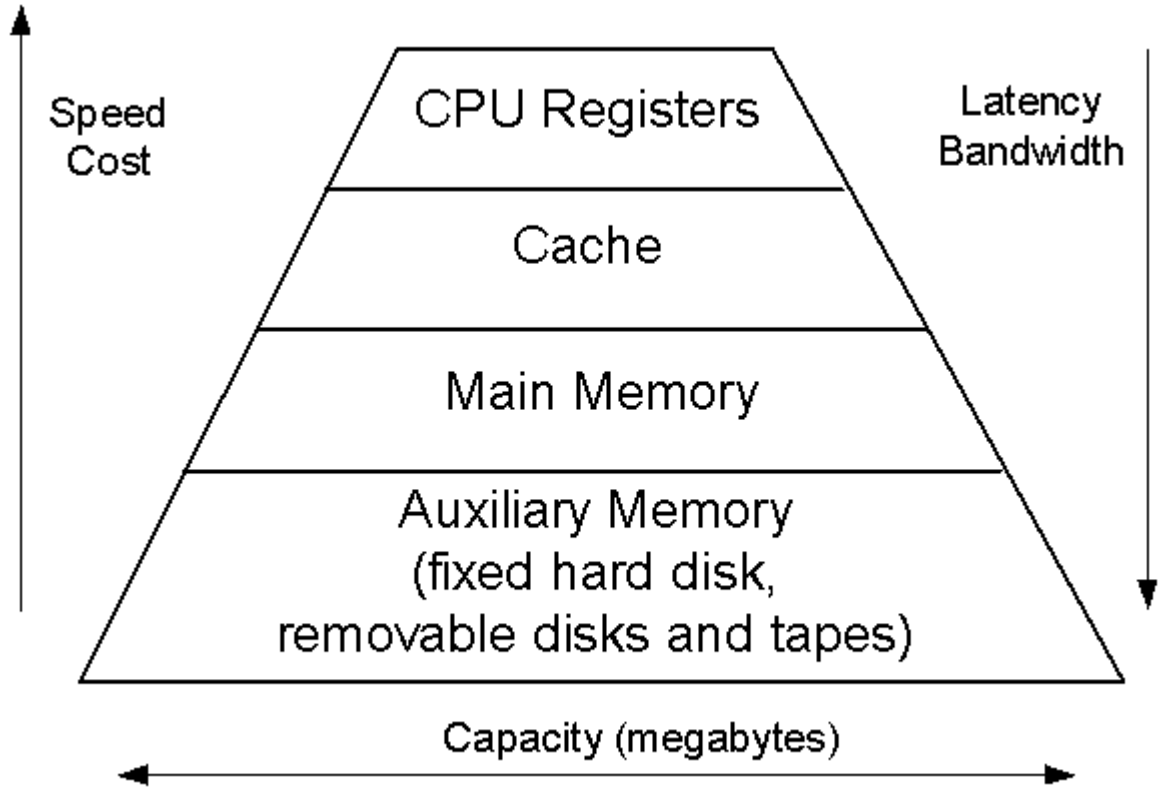


Control lines are coded into specific fields within a microinstruction.
Decoders are needed to map a field of k bits to 2^k possible combinations of control lines.

Because of the encoding, vertical microinstructions are much shorter than horizontal ones.

Control lines encoded in the same field cannot be activated simultaneously. Therefore, vertical microinstructions allow only **limited parallelism**.

Memory System
Memory hierarchy



The memory hierarchy can be characterized by a number of parameters:

1. Access type (sequencial, direct, random and associative)

- **Sequencial access.** It is used in tape units. Memory is organized into units of data, called records. Access must be done in a specific linear sequence. Example: if access to location 100 takes 500 ns, and if a consecutive access to location 101 takes 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.

- **Direct access.** It is used in disk units. Individual blocks have a unique address based on physical location. Access is done by direct access of a block and sequencial searching to reach the final location.

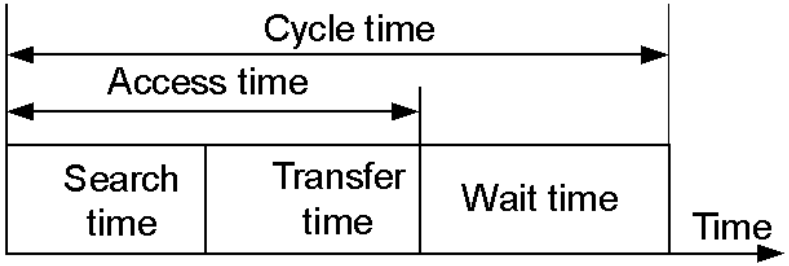
- **Random access.** It is used in main memory and some cache systems. Each addressable location has a unique address. The time to access a given location is constant.

- **Associative access.** It is used in some cache memories. It is a random access type of memory in which a word is stored and retrieved **based on a portion of its content** rather than its address. The access time is also constant.

2. Capacity. Is typically expressed in terms of bytes or words (1KB, 1MB, 1GB).

3. Access time (latency). The time it takes to perform a write or read operation, it represents an interval between the request for information and the access to the first bit of that information.

4. Cycle time It consists of the access time plus any additional time required before a second access can commence.



5. Bandwidth (Transfer rate). This is the rate at which data can be transferred into or out a memory unit. It is egual to 1/cycle time (words per second) or w/cycle time.

6. Cost (is usually specified in money per megabytes).

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

A variety of physical types of memory have been employed. The most common today are **semiconductor memory**, **magnetic,** used for disk and tape, and **optical** and **magneto-optical.**

According to physical characteristics memory can be:

- **Volatile** – information is lost when electrical power is switched off (RAM).
- **Non-volatile** – information once recorded remains without deterioration until it is changed (ROM).

Semiconductor memory types

The main memory of a computer system should be fast enough to not degrade the performance of the system. To achieve this, the semiconductor type memories are used as main memory.

- **read-only memory -ROM** non-volatile memory type
- **read-write memory or random access memory - RAM volatile**

ROM memory

- **ROM**. Its content can not be erasable.
- **PROM** (Programmable read only memory. Once programmed, they can not be erased.
- **EPROM** (erasable programmable ROM). Erasable by ultraviolet lights.
- **EEPROM** (byte-level electrically erasable programmable ROM).
- **Flash memory** (block-level electrically erasable programmable ROM).

RAM memory

- **static random access memory (SRAM)**
- **dynamic random access memory (DRAM)**

In a SRAM binary values are stored using traditional flip-flops (6 transistors configuration).

A DRAM is made with cells that store data as charge on capacitors.

The use of dynamic memory leads to saving in chip area. The presence or absence of charge on a capacitor is interpreted as a binary 1 or 0. Because capacitors have a natural tendence to discharge, DRAMs require periodic charge refreshing by a special circuit.

SRAM and DRAM are both volatile: power must be continuously supplied to the memory.

DRAM cell is smaller than SRAM cell. Thus, a DRAM is **denser and less expensive**, but it requires the supporting **refresh circuitry**.

Thus, DRAMs are used in large memory requirements.

SRAMs are generally faster than DRAMs. They are used and in cache memories.

The effectiveness of a memory hierarchy during a program execution depends on the principle called ***locality of reference.***

According to this principle, the most frequently used information is temporarily moved into the faster memory.

There exist two forms of locality: ***spatial and temporal locality***.

- ***Spatial locality*** refers to the phenomenon that when a given address has been referenced, it is most likely that addresses near it will be referenced within a short period of time, for example, consecutive instructions in a straightline program.
- ***Temporal locality***, on the other hand, refers to the phenomenon that once a particular memory item has been referenced, it is most likely that it will be referenced next, for example, an instruction in a program loop.

The sequence of events that takes place when the processor makes a request for an

item is as follows.

First, the item is sought in the first memory level of the memory hierarchy.

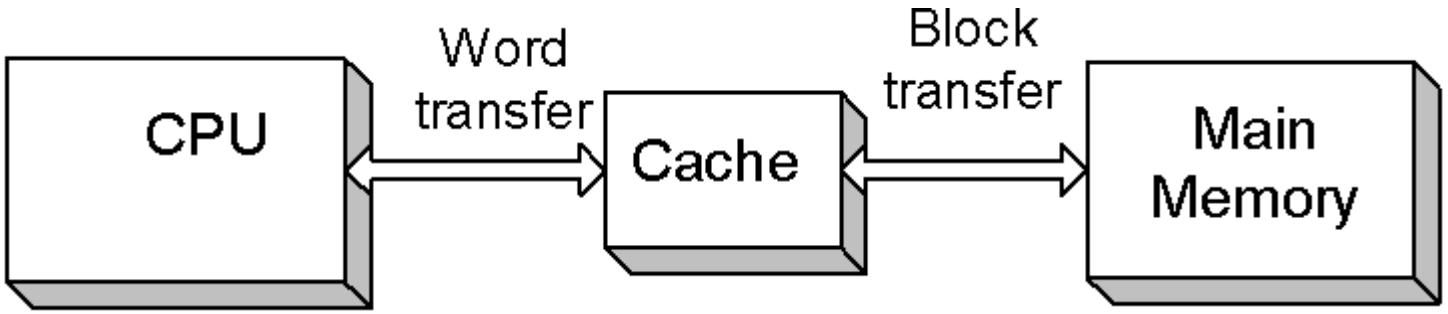
The probability of finding the requested item in the first level is called ***the hit ratio***.

The probability of not finding (missing) the requested item in the first level of the memory hierarchy is called the ***miss ratio***.

When the requested item causes a “miss,” it is sought in the next subsequent memory level.

Cache memory

Cache memory is a small high-speed memory, situated between the processor and the main memory in which the information expected to be used more frequently by the CPU is kept (the term cache means a safe place for hiding or storing things).



The structure of a cache-memory system

At any given time some active portion of the main memory is duplicated in the cache.

Therefore, when the processor makes a request for a memory reference, the request is first sought in the cache.

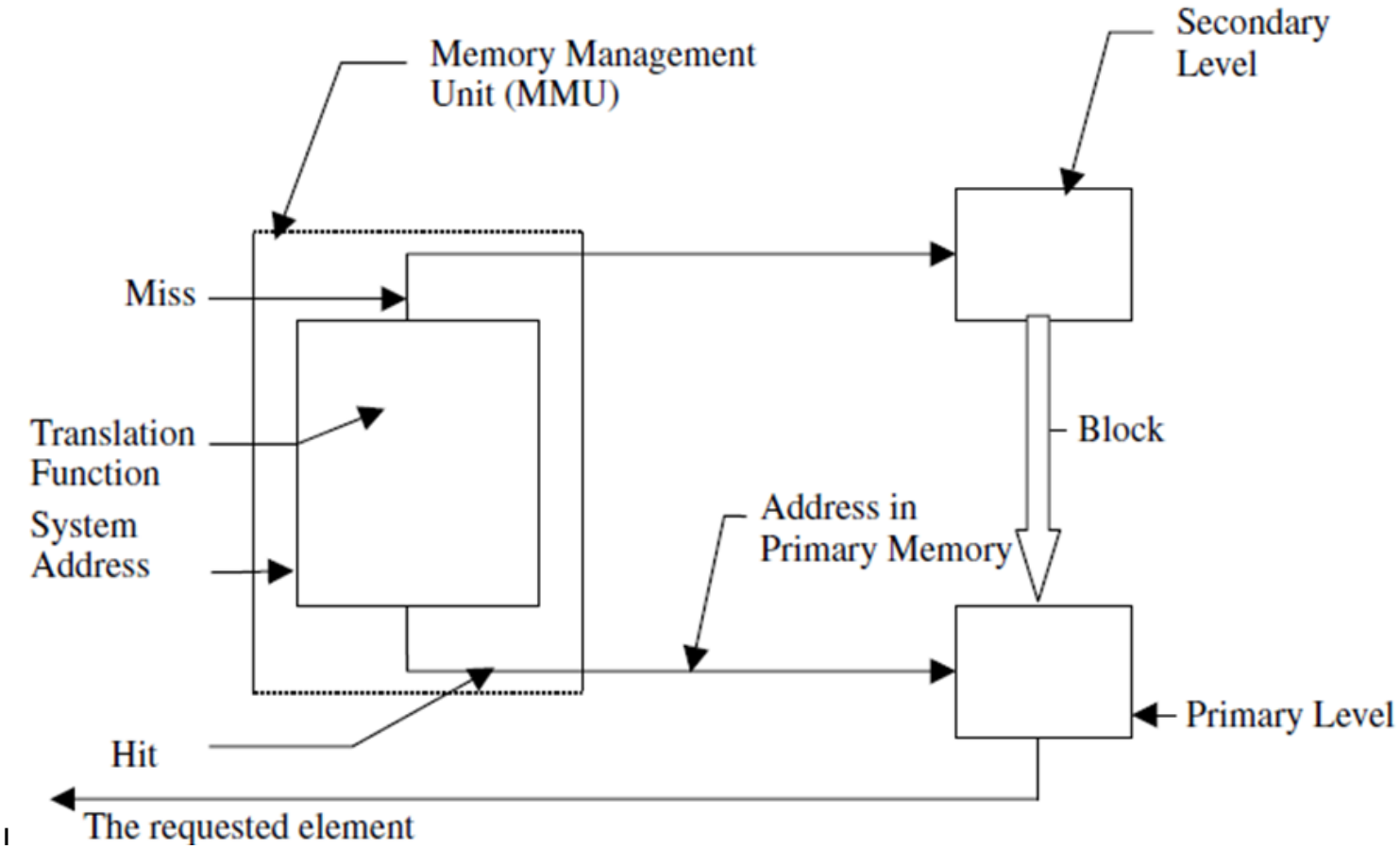
If the request corresponds to an element that is currently residing in the cache, we call that a **cache hit.**

If the request corresponds to an element that is not currently in the cache, we call that a **cache miss**.

After a cache miss, a block of elements is brought from the main memory to cache.

Because of the phenomenon of ***locality of reference,*** we can expect that the next requested element will be residing in the neighboring locality of the current requested element.

A request for accessing a memory element is made by the processor through issuing the logic address of the requested element. It may correspond to that of an element that exists currently in the cache (cache hit); otherwise, it may correspond to an element that is currently residing in the main memory. Therefore, address translation has to be made in order to determine where is the requested element. This is one of the functions performed by the memory management unit (MMU).



The main memory consists of up to 2^n addressable words, with each word having a unique n-bit address.

For mapping purposes, this memory is considered to consist of a number of fixed-lengh blocks of B words each.

That is, there are $M=2^n/B$ blocks.

Cache consists of C lines of B words each and the number of lines is considerably less than the number of main memory blocks ($C<<M$).

Because there are more blocks than lines, an individual line cannot be uniquely dedicated to a particular block. Thus, each line includes a tag that identifies which block is currently being stored. The tag is usually a portion of the main memory address.

There are three main different organization techniques used for cache memory.

1. Direct mapping

2. Associative Mapping

3. Set-Associative Mapping.

These techniques differ in two main aspects:

- - The criterion used to place, in the cache, an incoming block from the main memory.
- - The criterion used to replace a cache block by an incoming block (on cache full).

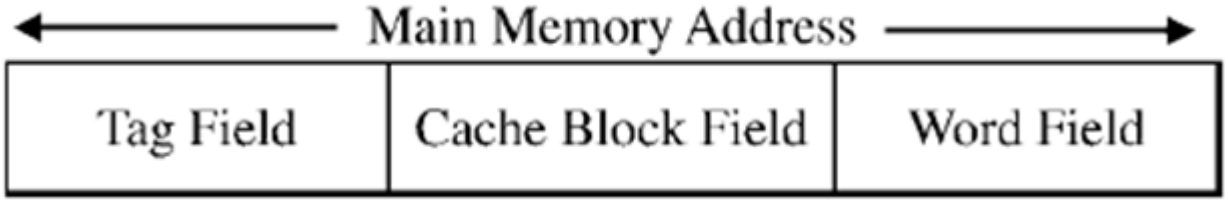
Direct mapping

Each block of main memory is mapped into only one possible cache block.

Consider a main memory with M blocks with B words in each.

If cache contains C lines, then memory is organized as a two-dimensional array with C lines and L columns. C*L=M. So, L memory blocks from one line can be mapped in one cache block.

Each main memory address is divided into three fields:



The word field identifies a unique word within a block. It contains $b=\log_2 B$ bits. B is the number of words in a block.
The cache block field specifies one cache block (line). It contains $c= \log_2 C$ bits. C is the number of cache lines.
The tag field specifies one block in the main memory line. It contains $l = \log_2 L$ bits, L is the number of columns in main memory. $L=M/C$
The total number of bits in the main memory address $n=\log2 (B*M)$. M is a number of total main memory blocks.

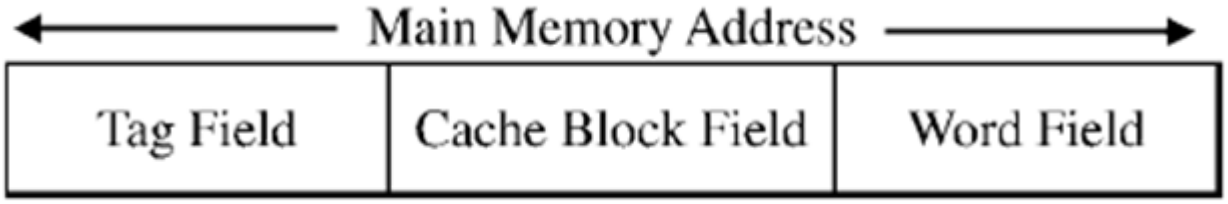
Consider the case of a main memory. consisting of M=4K=2^{12} blocks, a cache memory consisting of C=128=2^7 blocks, and a block size of B=16=2^4 words (bytes). The main memory size is $2^{12}*2^4=2^{16}$=64KB.

The division of the main memory and the cache according to the direct-mapped cache technique: Main memory array: 128 x 32.

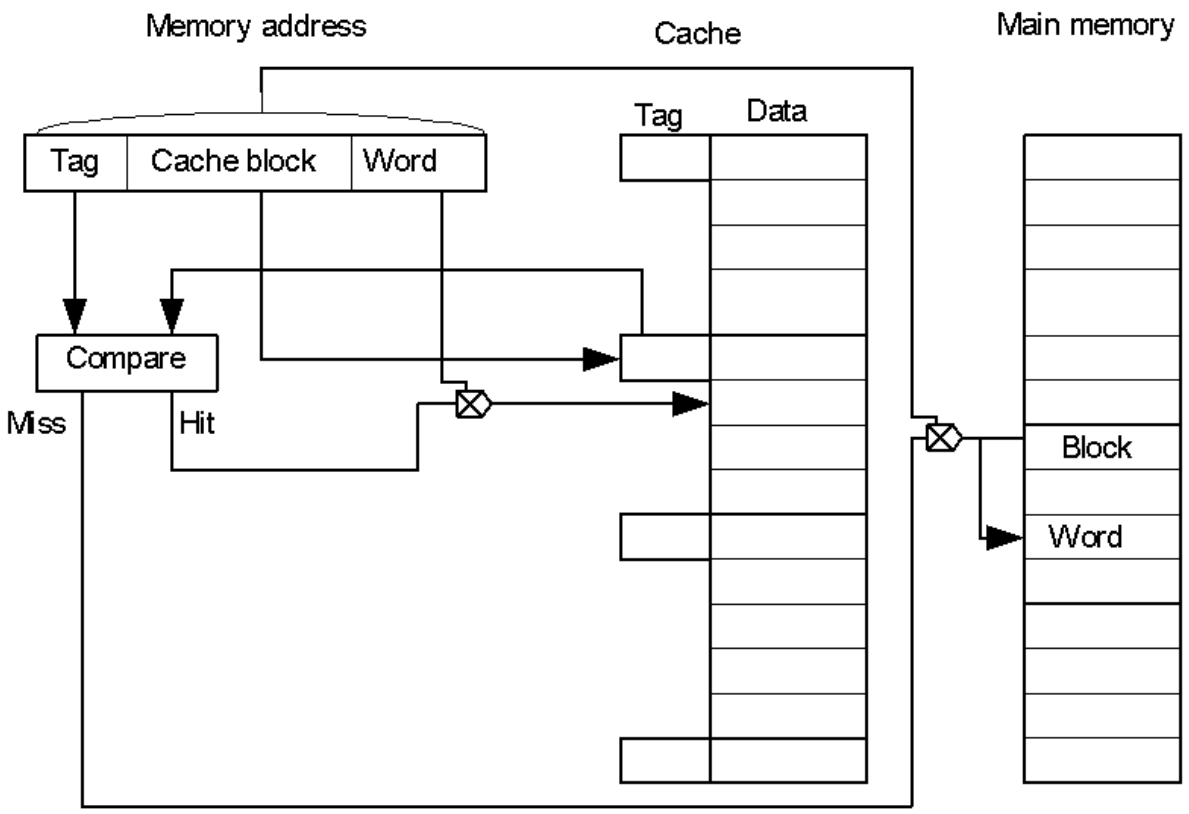
Tag		Cache
3	0	384
1	1	129
0	2	
	126	
31	127	4095

Main Memory

0	128	256	384		3968
1	129	257	385		
2	130	258	386		
127	255	383			4095
0	1	2	3		31



Word field: b= $\log_2 16 = 4$ bits
Cache Block field c= $\log_2 128 = 7$ bits
Tag field l= $\log_2 L = \log_2 (M/C) = \log_2 (2^{12}/2^7) = 5$ bits
The total number of bits in the main memory address
n = $\log_2 (M*B) = \log_2 (2^{12} *2^4) = 16$ bits.



The direct mapping technique is simple and inexpensive to implement.
Disadvantage: a fixed cache location for any given block.

Associative Mapping

According to this technique, an incoming main memory block can be placed in any available cache block.

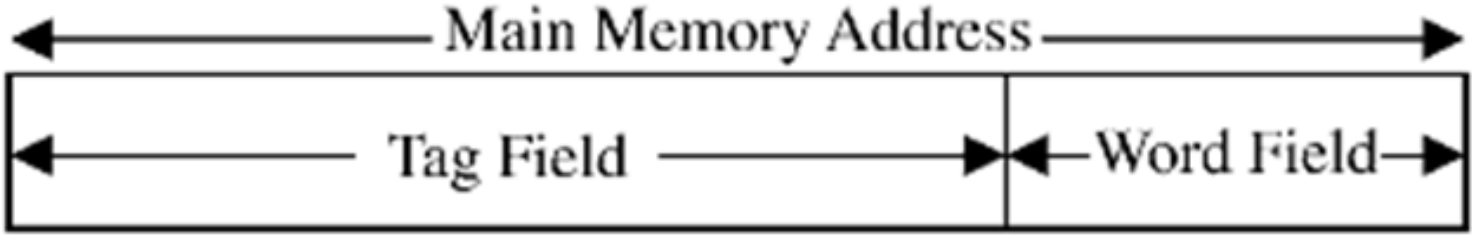
Therefore, the address issued by the processor need only have two fields. These are the Tag and Word fields.

The first uniquely identifies the block while residing in the cache.

The second field identifies the element within the block that is requested by the processor.

To determine whether a block is in the cache, the cache control logic must examine every block's tag in parallel.

No field in the address corresponds to cache block number, so that the number of block in the cache is not determined by the address format.



The length, in bits, of each of the fields:

1. Word field b= $\log_2$ B, where B is the size of the block in words.
2. Tag field m= $\log_2$ M, where M is the size of the main memory in blocks.
3. The number of bits in the main memory address n= $\log_2$ (B * M)

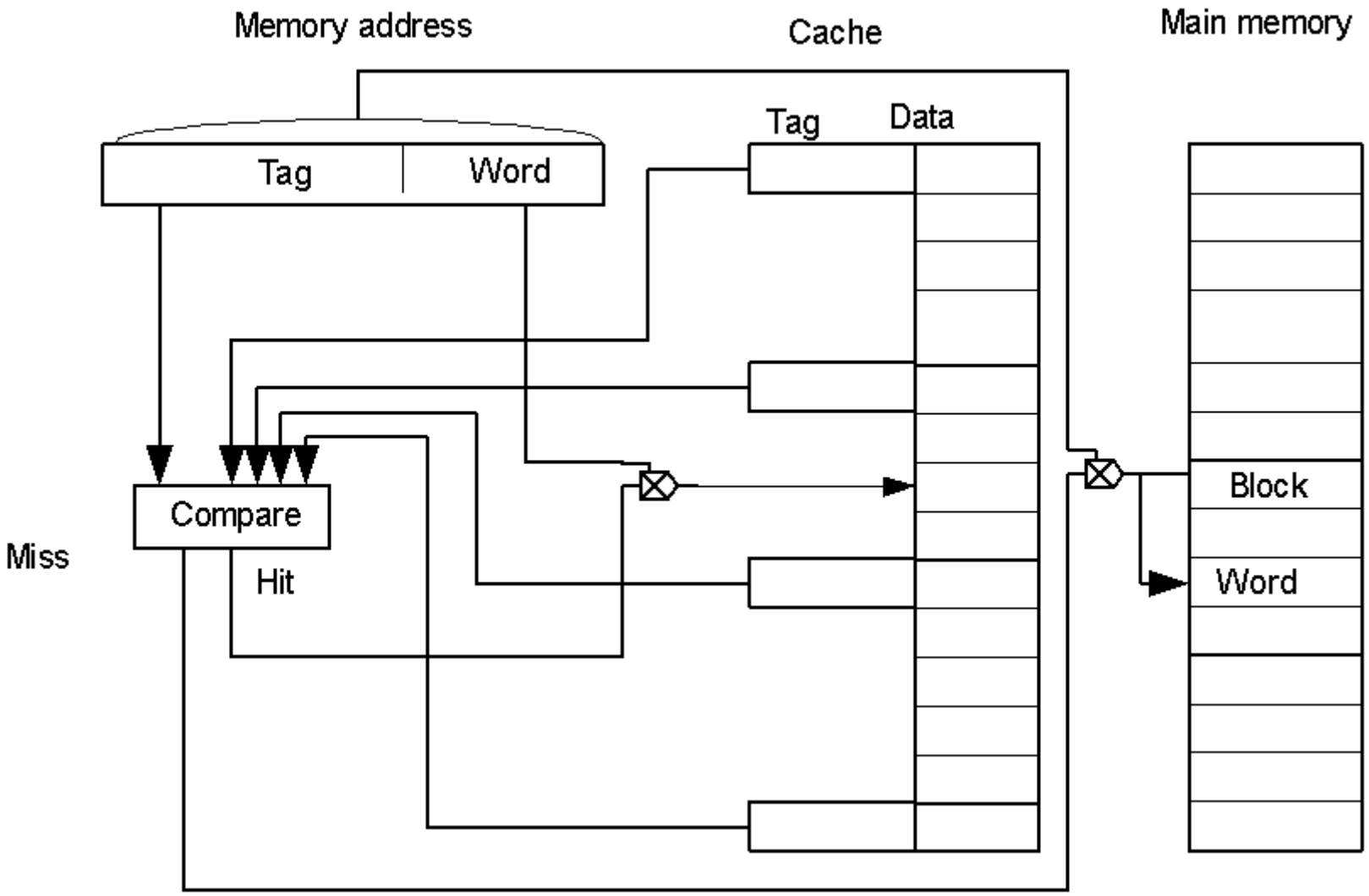
Let's compute these parameters for a memory system having the following specification: size of the main memory is 4K blocks, size of the cache is 128 blocks, and the block size is 16 words.

Word field b= $\log_2$ B = $\log_2$ 16= $\log_2$ 2^4 = 4 bits

Tag field m = $\log_2$ M = $\log_2$ 4K= $\log_2$ 2^{12} = 12 bits

The number of bits in the main memory address

n=$\log_2$(B * M)=$\log_2$(2^4 * 2^{12}) = 16 bits.


Memory address
Cache
Main memory
Tag
Data
Tag
Word
Compare
Miss
Hit
Block
Word

The main **advantage** of the associative-mapping technique is the **efficient use of the cache.**

The main **disadvantage** of the technique, however, is the hardware overhead required to perform the **associative (parallel) search** conducted in order to find a match between the tag field and the tag memory.

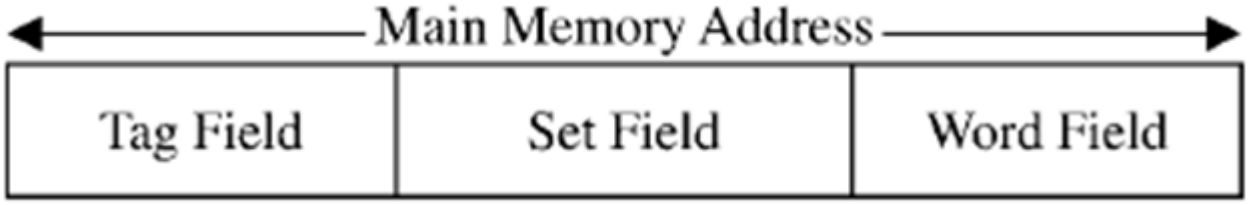
Set-Associative Mapping

A set-associative mapping is a compromise between direct and associative mapping.

According to set-associative mapping technique, the cache is **divided into a number of sets**.

An incoming block maps to any block in the assigned cache set.

Therefore, the address issued by the processor is divided into three distinct fields. These are the **Tag, Set, and Word fields.**



The length, in bits, of each of the fields is given by:

1. Word field b= $\log_2$ B, where B is the size of the block in words.

2. Set field s= $\log_2$ S, where S is the number of sets in the cache.

3. Tag field m= $\log_2$ (M/S), where M is the size of the main memory in blocks.

S = C/Bs, where C is the number of cache blocks and Bs is the number of blocks per set.

4. The number of bits in the main memory address n = $\log_2$ (B * M).

Example. Compute the above three parameters (Word, Set, and Tag) for a memory system having the following specification: size of the main memory M is 4K blocks, size of the cache C is 128 blocks, and the block size B is 16 words. One cache set Bs has four blocks.

Word field b= $\log_2 B = \log_2 16 = \log_2 2^4 = 4$ bits

Set field s=$\log_2 (128/4) = \log_2 32 = 5$

Tag field m = $\log_2 (M/S) = \log_2 2^{10}/32 = \log_2 2^7 = 7$ bits

The number of bits in the main memory address n=$\log_2(B * M)=\log_2(2^4 * 2^{12}) = 16$ bits.

The protocol used by the MMU to satisfy a request made by the processor for accessing a given element.

1. Use the Set field (5 bits) to determine (directly) the specified set (1 of the 32 sets).

2. Use the Tag field to find a match with any of the (four) blocks in the determined set. A match in the tag memory indicates that the specified set determined in step 1 is currently holding the targeted block, that is, a cache hit.

3. Among the 16 words (elements) contained in hit cache block, the requested word is selected using a selector with the help of the Word field.

4. If in step 2, no match is found, then this indicates a cache miss. Therefore, the required block has to be brought from the main memory.

Input–Output system

Peripheral devices can not be connected directly to the system bus, only through I/O interface circuits – **I/O module.**

1. There are a wide variety of peripherals with various methods of operation;
2. The data transfer rate is very different.

Device	The data transfer rate
keyboard	10 characters (bytes)/second
scanner	200,000 characters/second
laser printer	100,000 characters/second
graphic display	30,000,000 characters/second

3. Peripherals often use different data formats and word lengths than the computer to which they are attached.

The functions of an I/O module

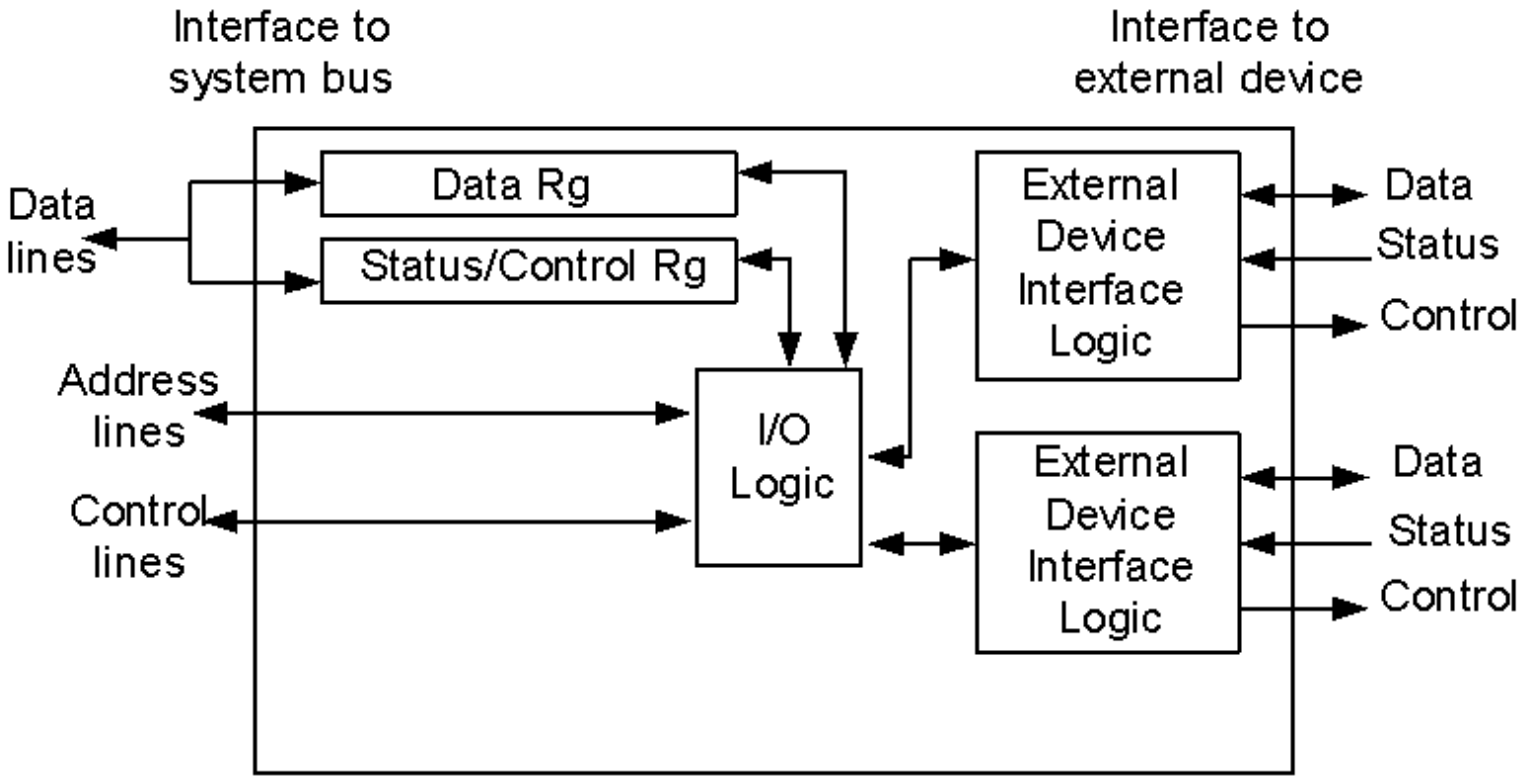
1. **Control and timing** to coordinate the flow of traffic between internal resources and external devices.

2. **Communication between the processor and the device.** It involves commands decoding, status information (common status signals are BUSY and READY), address recognition and data exchange.

3. **Data buffering.** The transfer rate of the processor and peripheral is different. So data are first stored in special input and output registers (ports).

Block diagram of an I/O module

- The processor check the status of the device
- The I/O module returns the status of the device
- If the device is ready to transmit, the processor requests transfer of data by means of a command to the I/O module
- The I/O module obtains a unit of data (8 or 16 bits) from the device
- The data are transferred from the I/O module to the processor.



There are two arrangements to address input and output registers.

1. **Shared I/O.** I/O devices are assigned particular addresses, isolated from the address space assigned to the memory.

The main advantage of the shared I/O arrangement is the separation between the memory address space and that of the I/O devices.

Its main disadvantage is the need to have special input and output instructions in the processor instruction set.

The shared I/O arrangement is mostly adopted by Intel.

2. **Memory-mapped I/O.** Input and output registers are addressed as memory locations.

The main advantage of the memory-mapped I/O is the use of the read and write instructions of the processor to perform the input and output operations, respectively. It eliminates the need for introducing special I/O instructions.

The main disadvantage - is the need to reserve a certain part of the memory address space for addressing I/O devices, that is, a reduction in the available memory address space.

The memory-mapped I/O has been mostly adopted by Motorola.

I/O Handling in in Operating Systems (I/O techniques of data transfer)

- ***programmed I/O (polling)***, in which I/O data transfer occurs under the control of the CPU program;
- ***interrupt driven I/O,*** in which I/O data transfer is controlled by CPU after the external interrupt request that initiates the transfer;
- ***direct memory access (DMA)***, in which a specialized I/O controller takes over the control of an I/O operation to move a large block of data.

Polling & interrupts work best with low-bandwidth devices, DMA works best with high-bandwidth devices (like hard disks)

Programmed I/O

I/O data transfer occurs under the control of the CPU **program**.

The program must check the device status, send a read or write command and transfer the data.

The processor must wait until the I/O operation is complete. If the processor is faster then the I/O module, this is wasteful of the processor time.

The process of checking the status of I/O devices in order to determine their readiness for receiving and/or sending characters, is called ***I/O polling.***

To execute an I/O instruction, the processor issues an address, specifying the particular I/O module and external device, and an I/O command.

There are four types of commands:

1. Control: Used to activate a peripheral and tell it what to do.

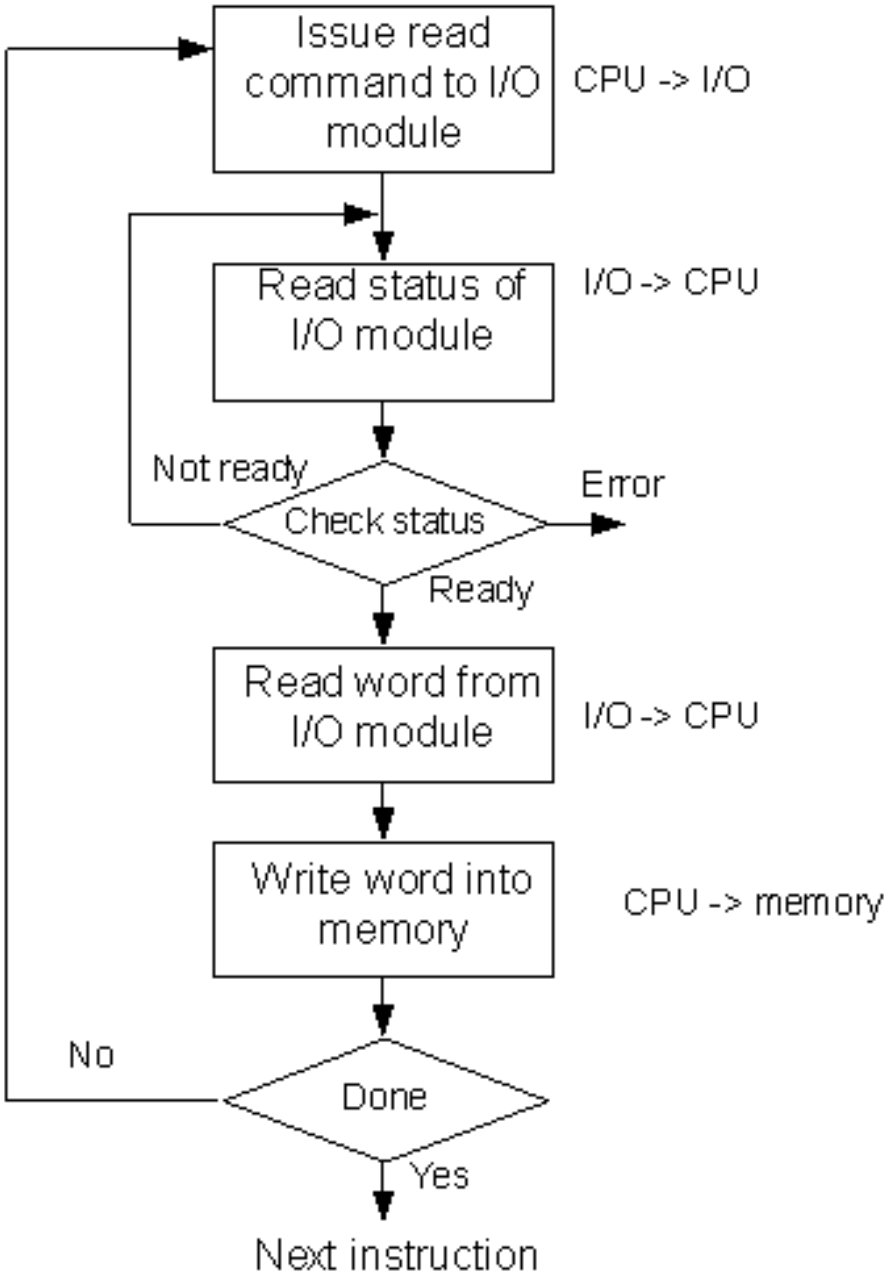
2. Test: Used to test various status conditions associated with an I/O module and its peripherals (if it is powered on, if the I/O operation is completed, if any errors occurred).

3. Read: Causes the I/O module to obtain the word of data from the peripheral and place it in an internal buffer – data register and then to the data bus;

4. Write: Causes the I/O module to take a word of data from the data bus and transmit it to the peripheral.

Flowchart of ***reading in*** a block of data:

For each word that is read in, the processor must remain in status checking cycle until it determines that the word is available in the I/O module's data register.
This flowchart highlights the main disadvantage of this technique: it is a time-consuming process.



Interrupt-driven I/O

The processor issues an I/O command, continues to execute other instructions, and is interrupted by the I/O module when the latter is ready to exchange data with the processor.

I/O module actions.

For input, the I/O module receives a READ command from the processor.

Then it proceeds to read data in from an associated peripheral.

Once the data are in the module's data register, the module signals an interrupt to the processor over a control line.

The module then waits until its data are requested by the processor. When the request is made, the module places data on the data bus.

Processor's actions.

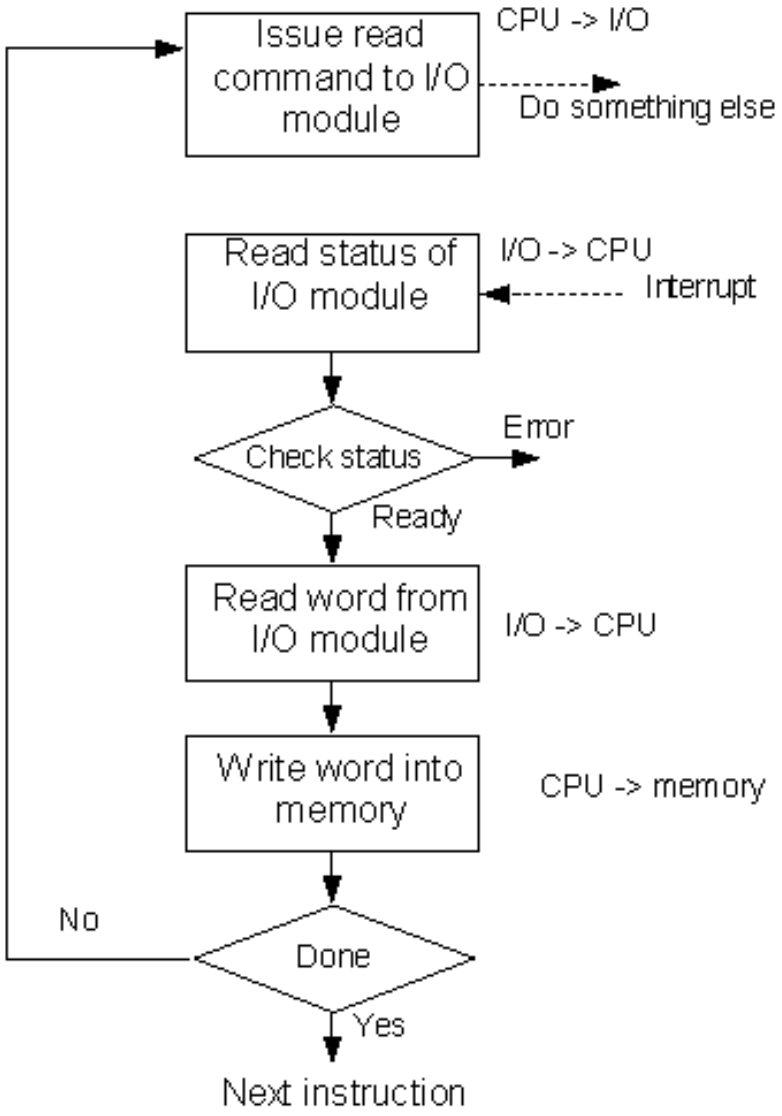
The processor issues a READ command and then goes off and executes other instructions.

At the end of each instruction cycle, the processor checks for interrupts.

When the interrupt occurs, the processor stores FLAGS register, current IP and CS values into stack, disables further interrupts, fetches from the bus one byte representing interrupt number, and jumps to Interrupt Service Routine (ISR). In this case, it reads the word of data from the I/O module and stores it in memory.

It then restores the content of the registers from stack and resumes execution.

A flowchart of ***reading in*** a block of data:



Interrupt-driven I/O is more efficient than programmed I/O because it eliminates needless waiting. However, it still consumes a lot of processor time, because every word of data that goes from memory to I/O module or vice-versa must pass through the processor.

Programmed I/O and interrupt-driven I/O suffer from two drawbacks:

- The I/O transfer rate is limited by the speed with which the processor can test and service a device.

- The processor must execute a number of instructions for each I/O transfer.

Direct memory access (DMA)

When large volumes of data are to be moved, a more efficient technique is required: direct memory access (DMA).

DMA involves an additional module on the system bus, the **DMA controller**. It takes over the control of the system from the processor.

When the processor wishes to read or write a block of data, it issues a command to the DMA controller, by sending the following information:

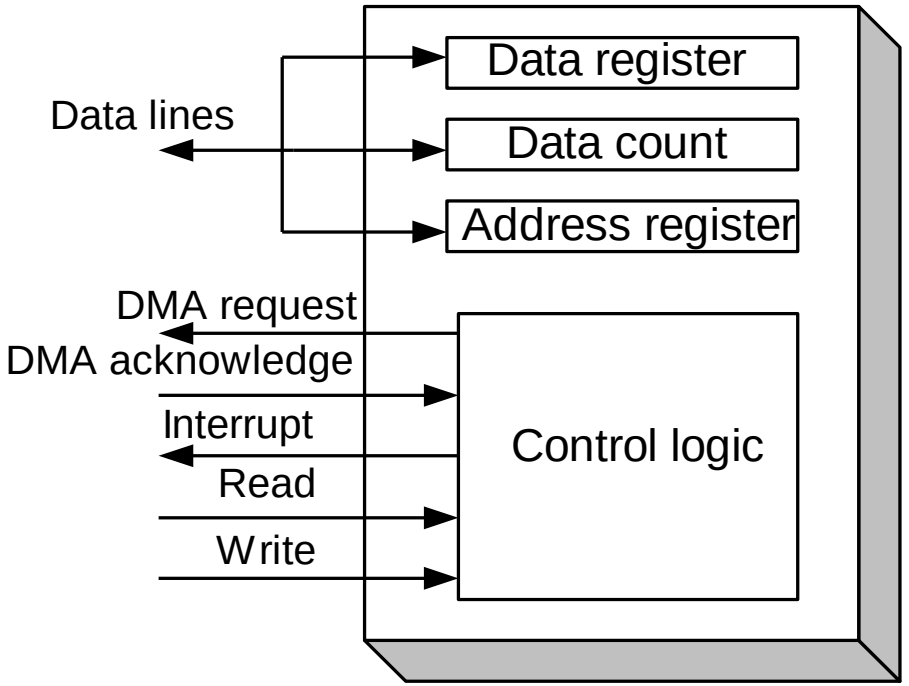
. If read or write is requested (read or write control lines).

. The address of the I/O device (data lines).

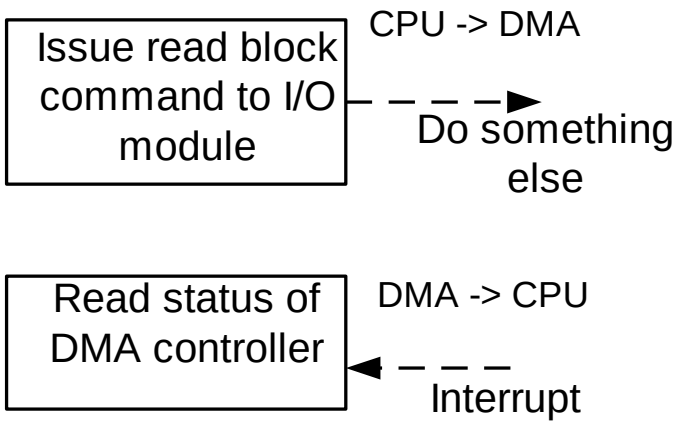
. The starting location in memory to read or write (it is stored in address register)

. The number of words to be read or written (send on data lines and stored in the data count register).

When the transfer is complete, The DMA controller sends an interrupt signal to the processor. Thus, the processor is involved only at the beginning and end of the transfer.



A flowchart of ***reading in*** a block of data:



1. DMA controller initiates data transfer.
2. Data is moved (increasing the address in memory, and reducing the count of words to be moved).
3. When word count reaches zero, the DMA informs the CPU of the termination by means of an interrupt.
4. The CPU regains access to the memory bus.

Direct memory access data transfer can be performed in **burst mode** or **single cycle mode**.

In burst mode, the DMA controller keeps control of the bus until all the data has been transferred to (from) memory from (to) the peripheral device.

This mode of transfer is needed for fast devices where data transfer cannot be stopped until the entire transfer is done.

In single-cycle mode (cycle stealing), the DMA controller relinquishes the bus after each transfer of one data word.

This minimizes the amount of time that the DMA controller keeps the CPU from controlling the bus, but it requires that the bus request/acknowledge sequence be performed for every single transfer. This overhead can result in a degradation of the performance.