MEGHE GROUP
VALUES REDEFINED

DTEL (Department for Technology Enhanced Learning)

The Centre for Technology enabled Teaching & Learning , N Y S S, India

Teaching Innovation - Entrepreneurial - Global

MEGHE GROUP
VALUES REDEFINED

DEPARTMENT OF COMPUTER TECHNOLOGY

IV-SEMESTER

COMPUTER ARCHITECTURE AND ORGANIZATION

UNIT NO.VI

COMPUTER PERIPHERALS

UNIT 6:- SYLLABUS

1. Memory mapping Techniques
2. Pipelining
3. Computer Peripherals: I/O Devices, DMA
4. I/O Transfers- Program Controlled & Interrupt Driven, DMA
5. Interrupt handling

UNIT-6 SPECIFIC OBJECTIVE / COURSE OUTCOME

The student will be able to:

1. *Differentiate among different types of architectures of processors*
2. Use suitable Cache memory mapping technique
   Compare Memory mapping techniques.

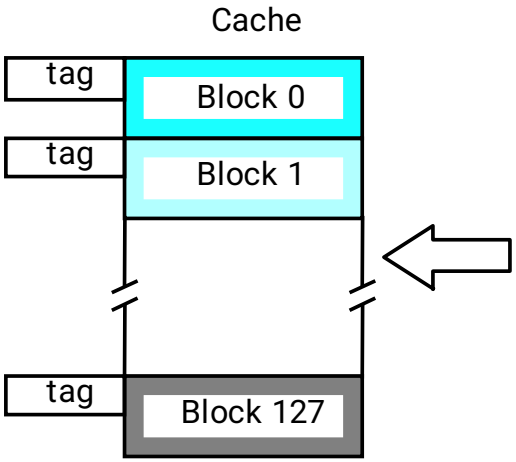
Direct Mapping

Block j of main memory maps onto block j modulo 128 of the cache

4: one of 16 words. (each block has $16=2^4$ words)

7: points to a particular block in the cache ($128=2^7$)

5: 5 tag bits are compared with the tag bits associated with its location in the cache. Identify which of the 32 blocks that are resident in the cache (4096/128).



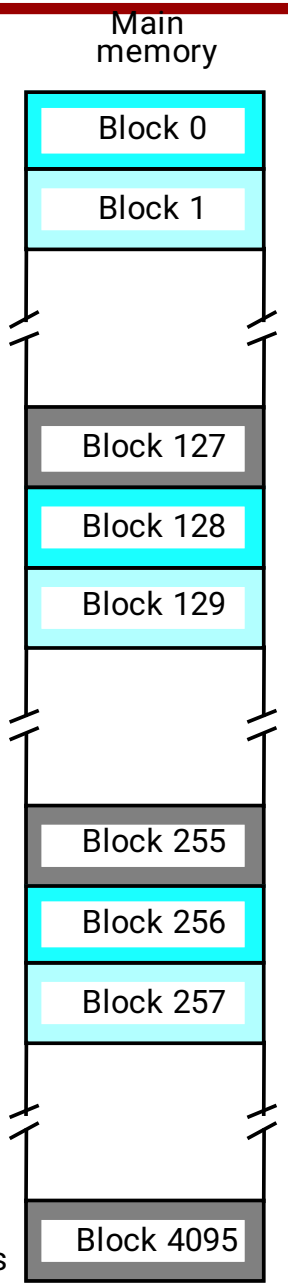
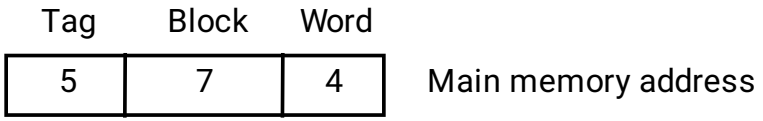
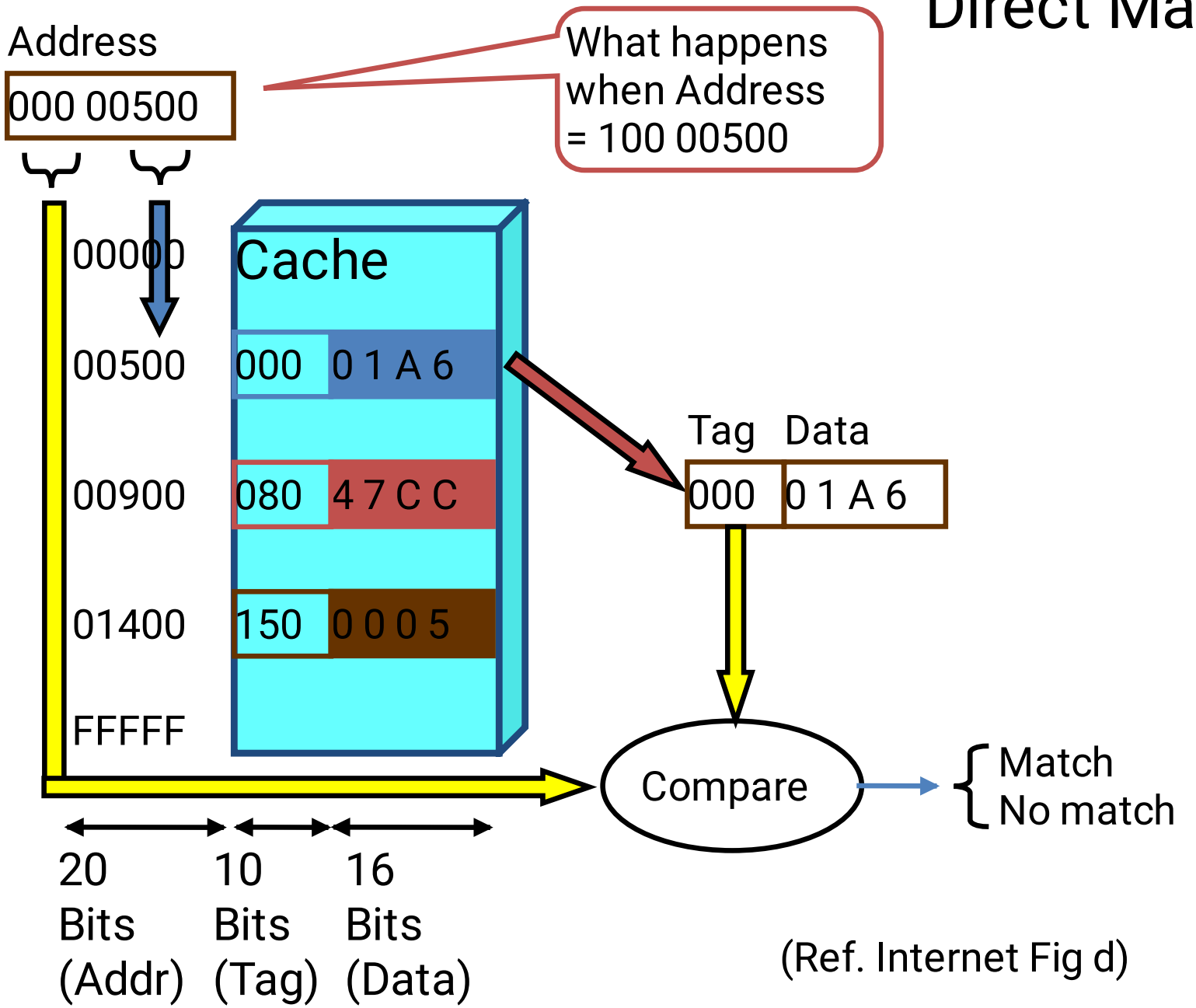


Figure 5.15. Direct-mapped cache.



Cache Memories (Mapping Functions)

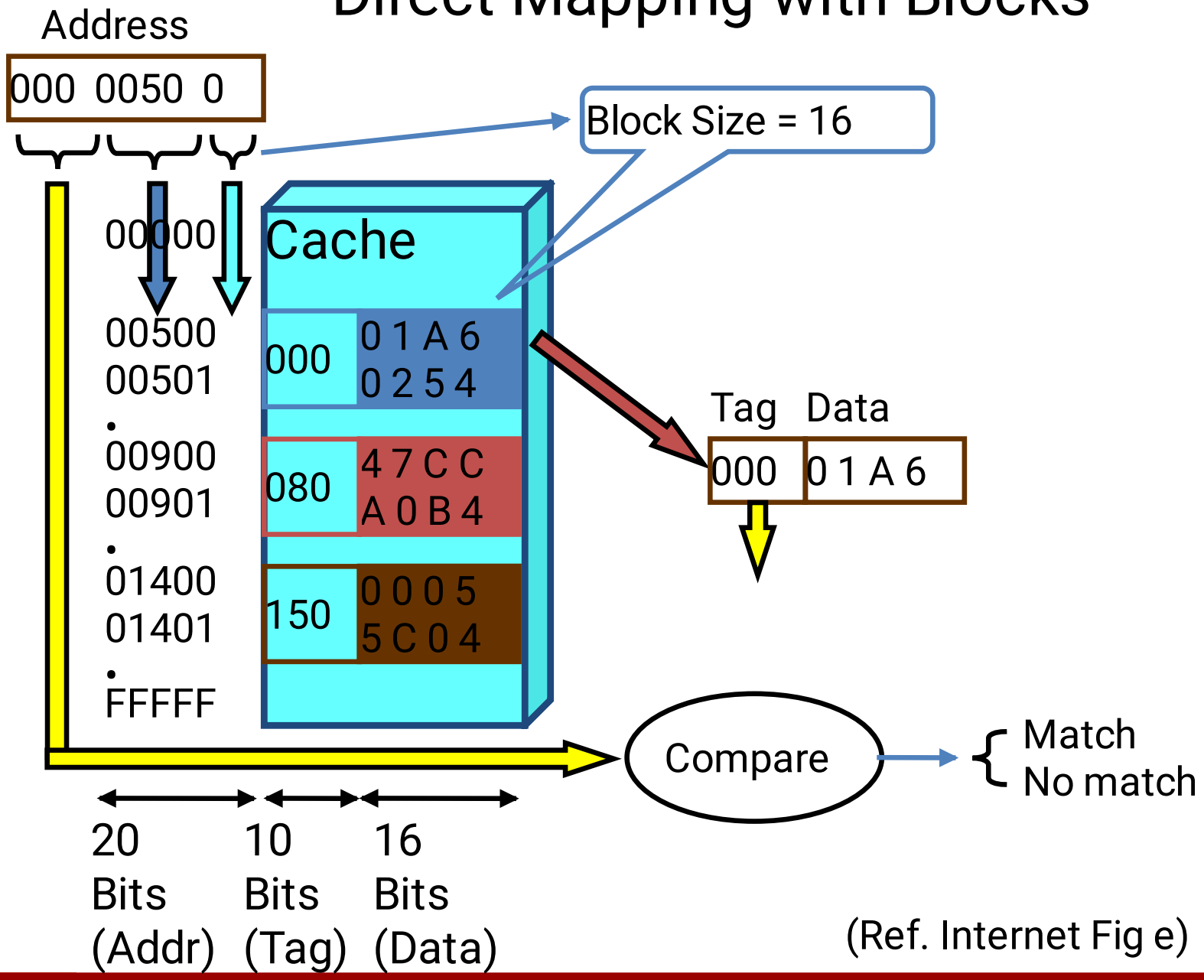
Direct Mapping



(Ref. Internet Fig d)

Cache Memories (Mapping Functions)

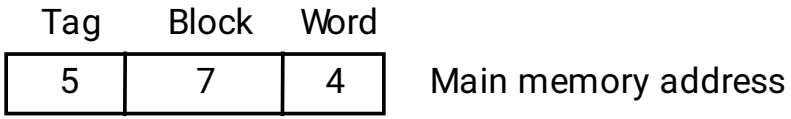
Direct Mapping with Blocks



(Ref. Internet Fig e)

Cache Memories (Mapping Functions)

Direct Mapping



11101,1111111,1100

- Tag: 11101
- Block: 1111111=127, in the 127th block of the cache
- Word:1100=12, the 12th word of the 127th block in the cache

Cache Memories (Mapping Functions)

Associative Mapping

4: one of 16 words. (each block has $16=2^4$ words)

12: 12 tag bits Identify which of the 4096 blocks that are resident in the cache $4096=2^{12}$.

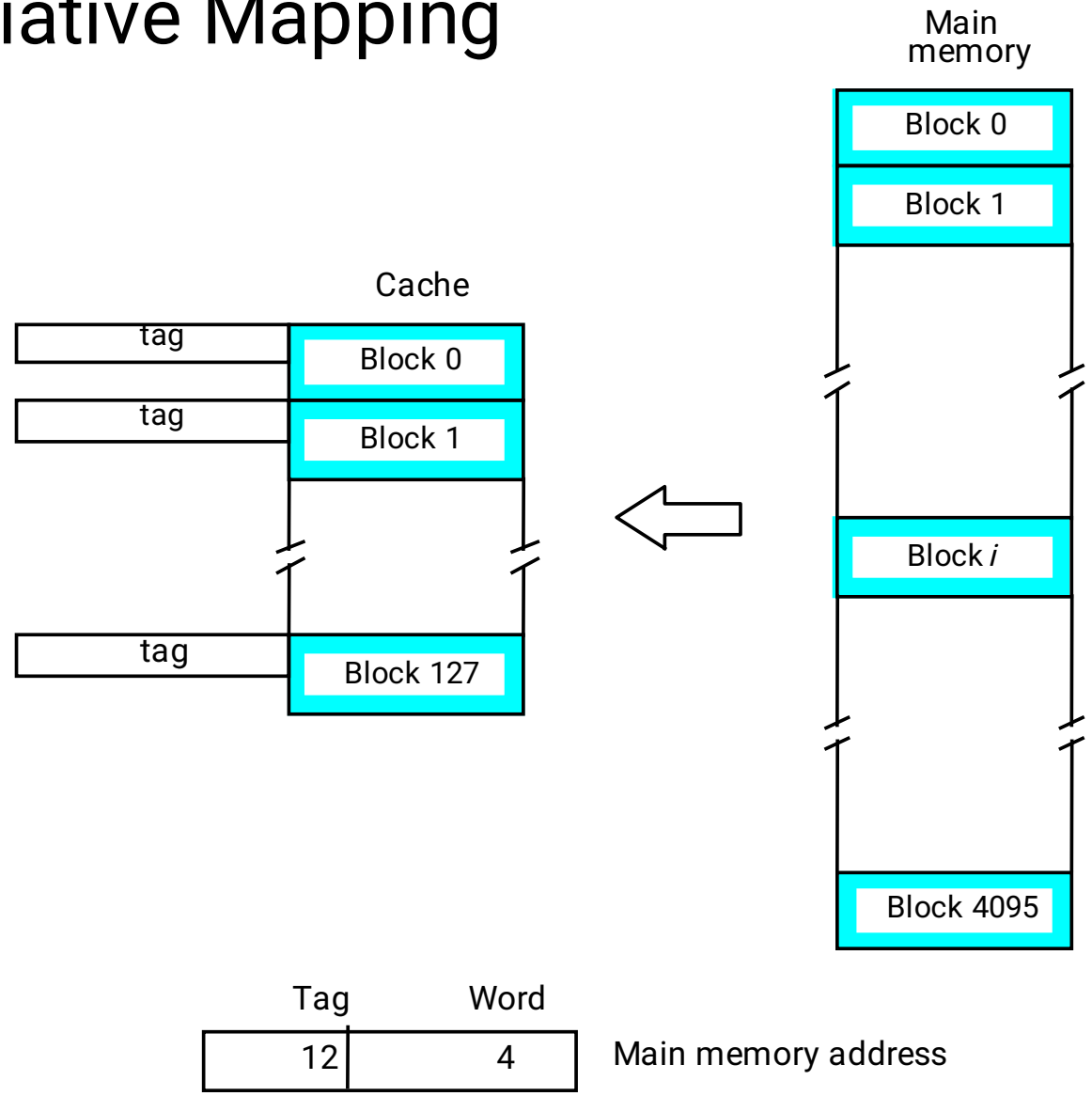
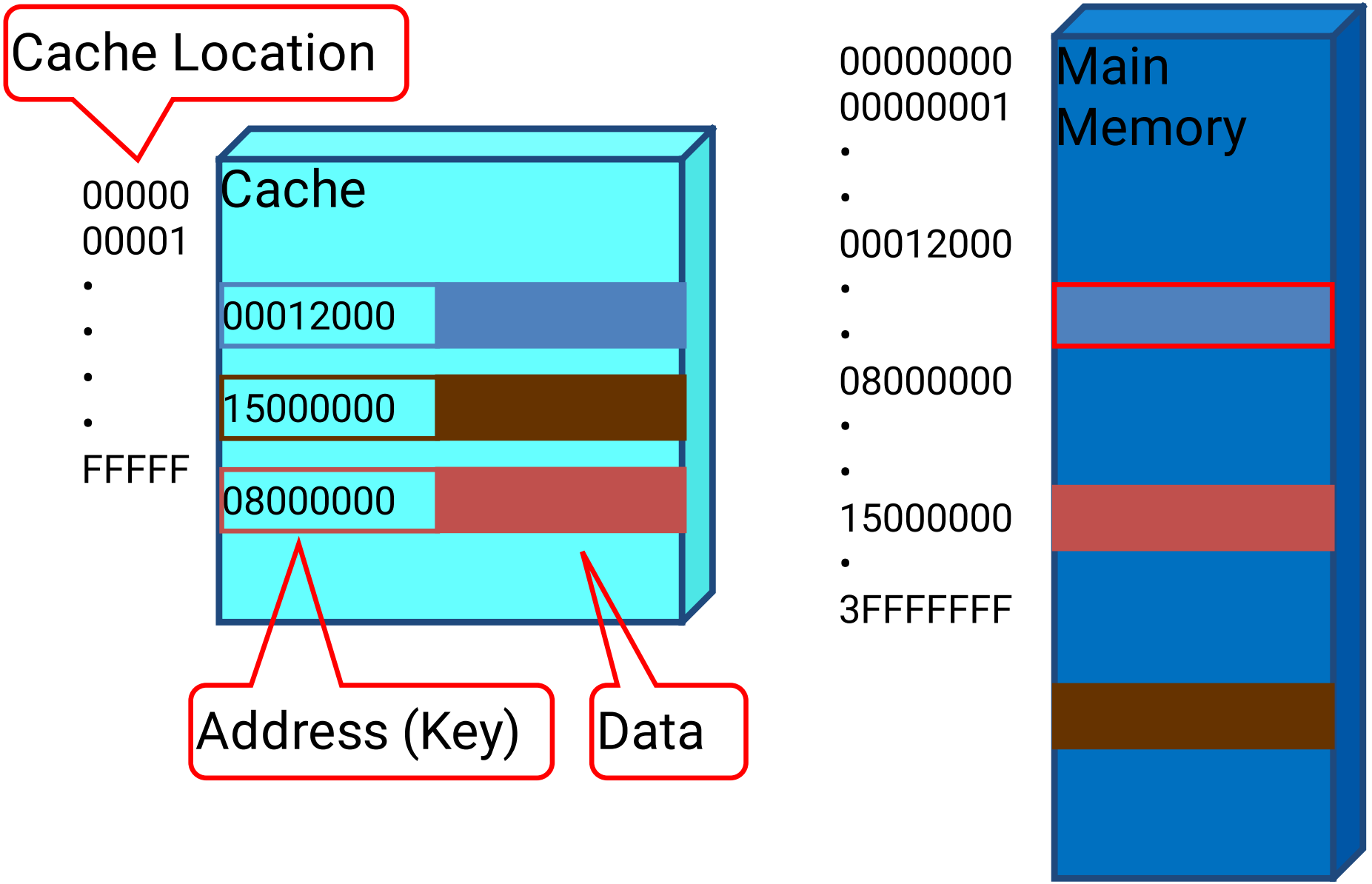


Figure 5.16. Associative-mapped cache.

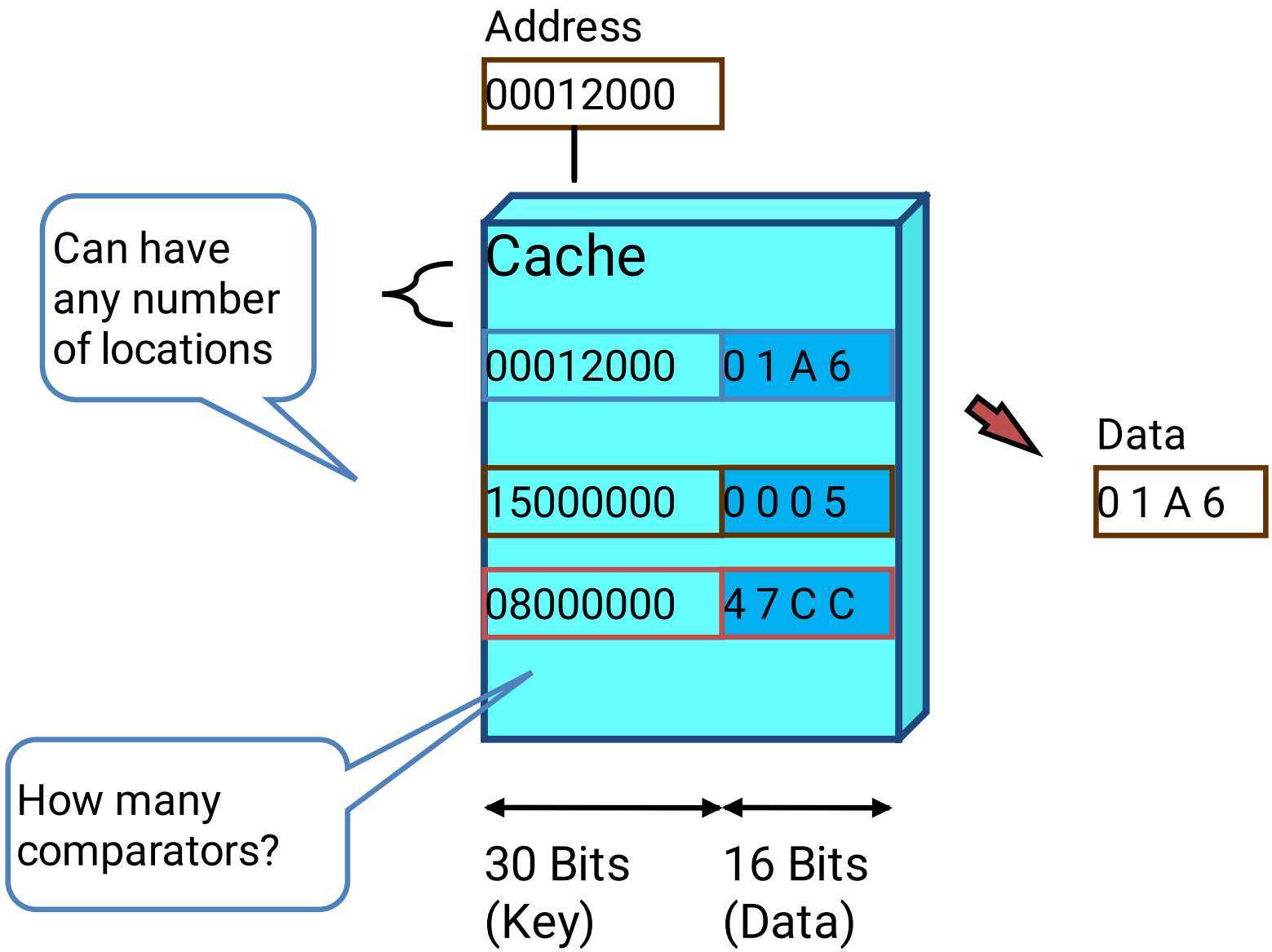
LECTURE 1: Cache Memories (Mapping Functions)

- Associative Memory (Fig f: Ref. Internet)



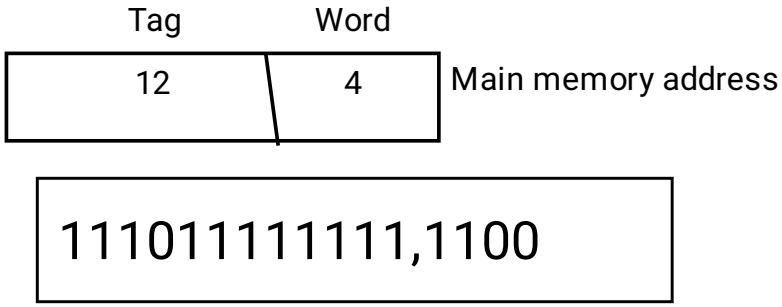
LECTURE 1: Cache Memories (Mapping Functions)

- Associative Mapping (Fig g:Ref. Internet)



Cache Memories (Mapping Functions)

Associative Mapping



- Tag: 111011111111
- Word:1100=12, the 12^{th} word of a block in the cache

Set-Associative Mapping

4: one of 16 words. (each block has $16=2^4$ words)

6: points to a particular set in the cache ($128/2=64=2^6$)

6: 6 tag bits is used to check if the desired block is present ($4096/64=2^6$).

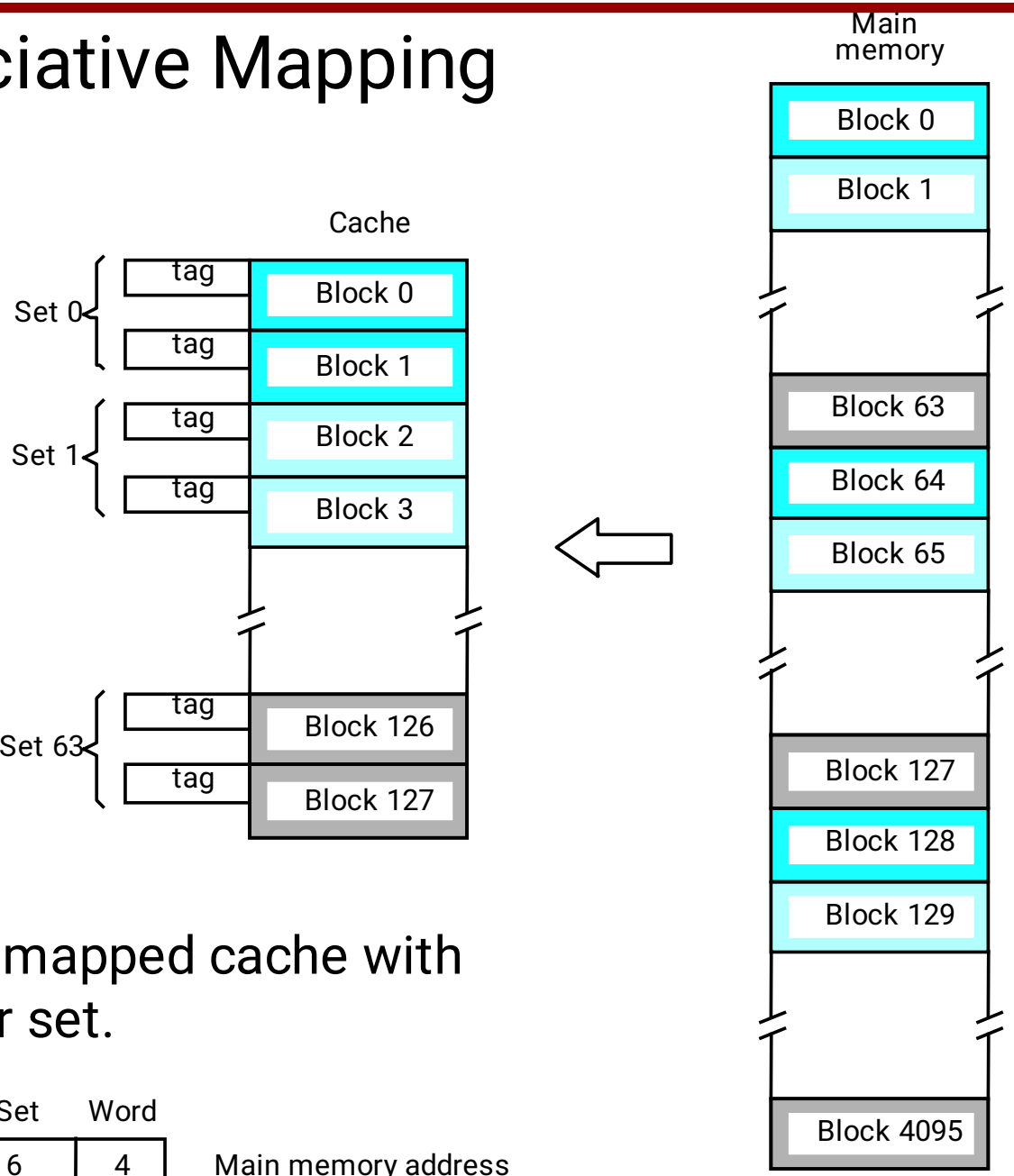
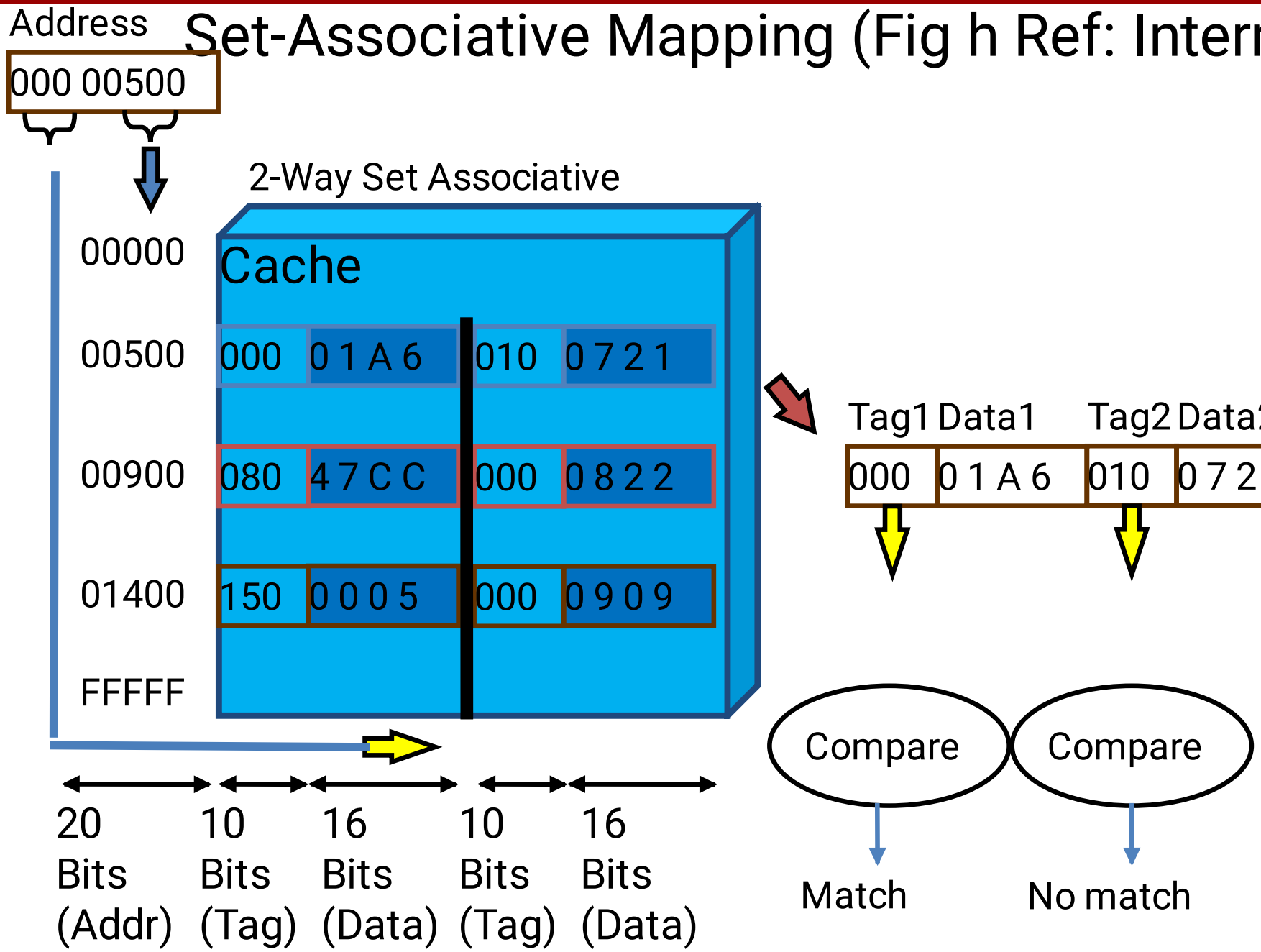
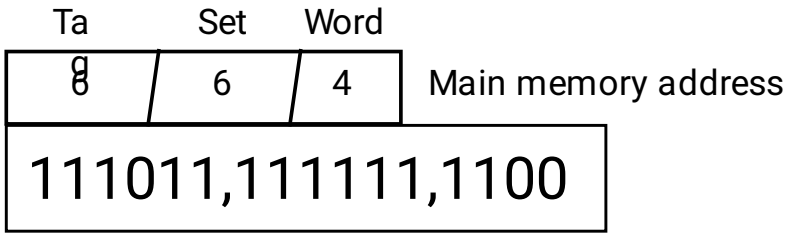


Figure 5.17. Set-associative-mapped cache with two blocks per set.

Set-Associative Mapping (Fig h Ref: Internet)



Set-Associative Mapping



- Tag: 111011
- Set: 111111=63, in the 63th set of the cache
- Word:1100=12, the 12th word of the 63th set in the cache

Replacement Algorithms

- Difficult to determine which blocks to kick out
- Least Recently Used (LRU) block
- The cache controller tracks references to all blocks as computation proceeds.
- Increase / clear track counters when a hit/miss occurs

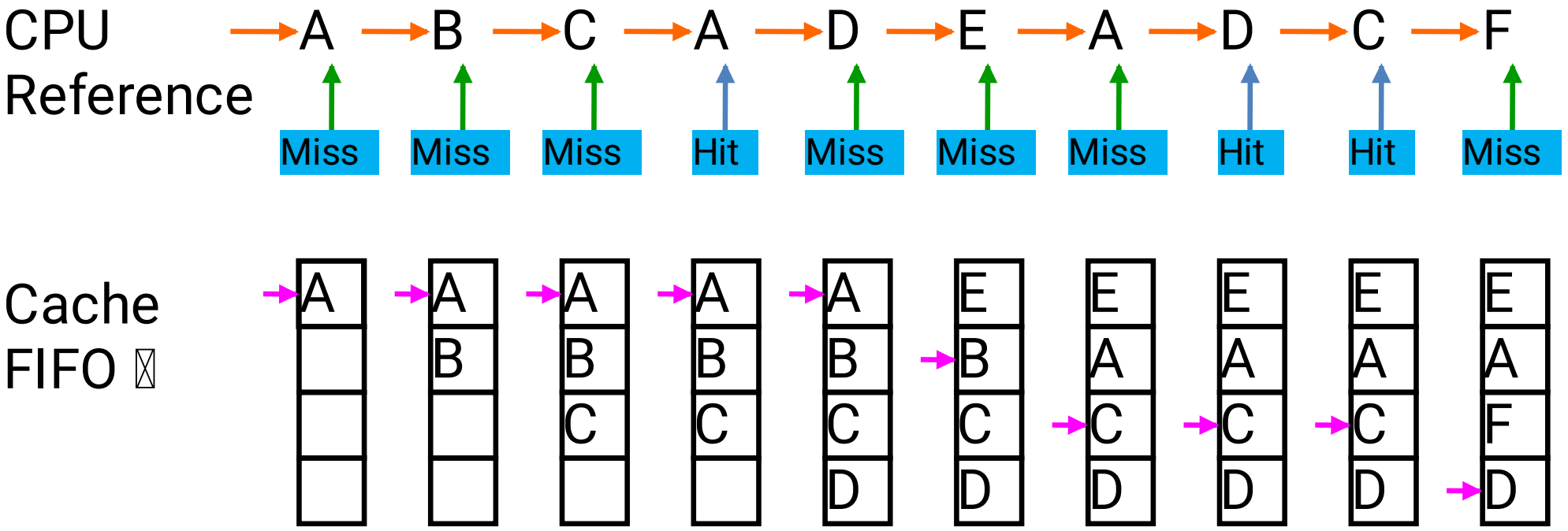
Cache Memories

Replacement Algorithms

- For Associative & Set-Associative Cache
  - Which location should be emptied when the cache is full and a miss occurs?
    - First In First Out (FIFO)
    - Least Recently Used (LRU)
- Distinguish an *Empty* location from a *Full* one
  - Valid Bit

Cache Memories

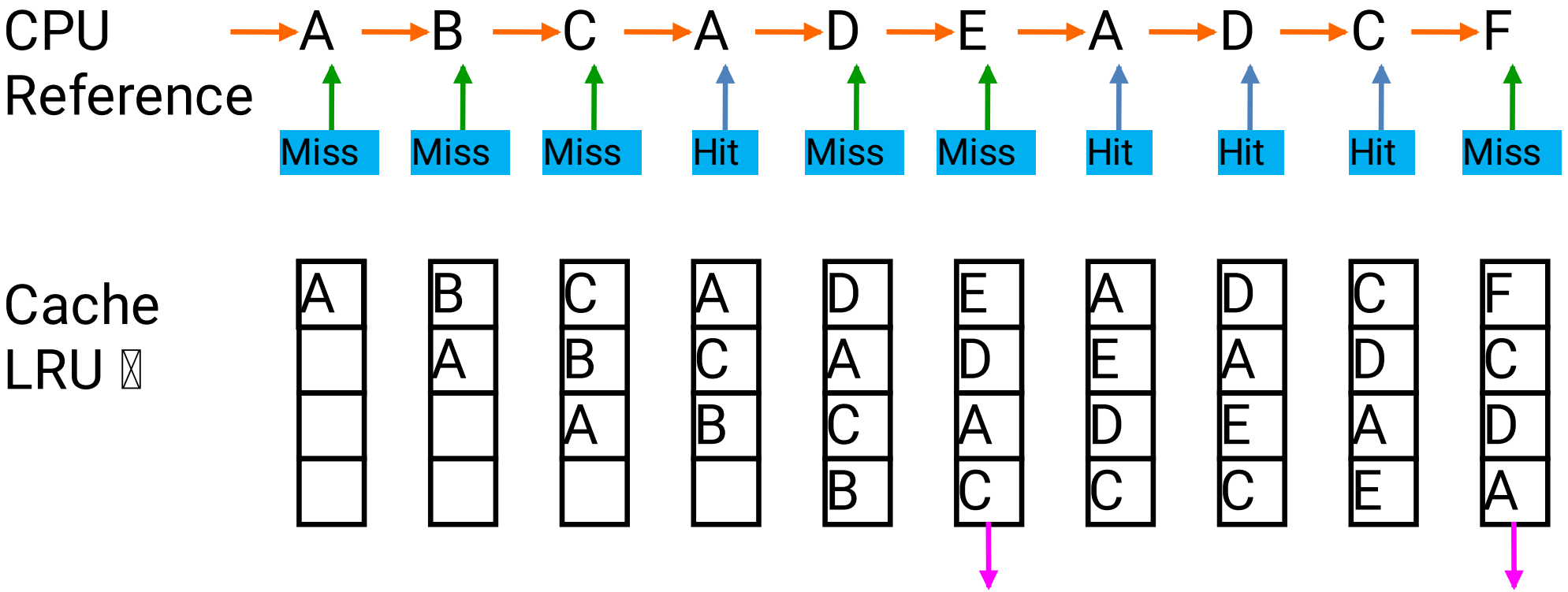
Replacement Algorithms



Hit Ratio = 3 / 10 = 0.3

Cache Memories

Replacement Algorithms



Hit Ratio = 4 / 10 = 0.4

THANK YOU

Pipelining

- Pipeline is a effective way of organizing concurrent activity in a computer system
- Pipelining is widely used in modern processors.
- Pipelining improves system performance in terms of throughput.

Making the Execution of Programs Faster

- Use faster circuit technology to build the processor and the main memory.
- Arrange the hardware so that more than one operation can be performed at the same time.
- In the latter way, the number of operations performed per second is increased even though the elapsed time needed to perform any one operation is not changed.

Pipelining

Use the Idea of Pipelining in a Computer

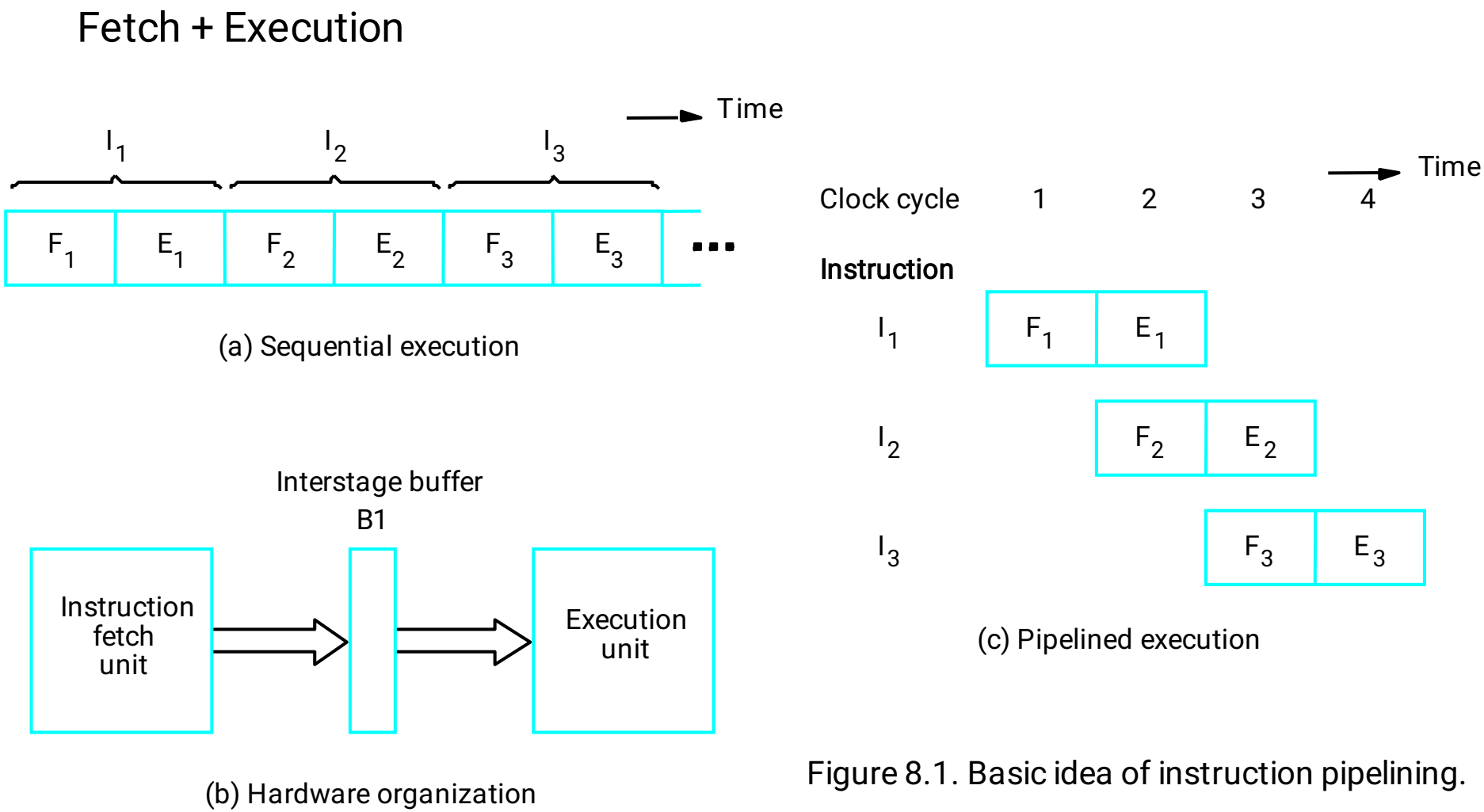


Figure 8.1. Basic idea of instruction pipelining.

Use the Idea of Pipelining in a Computer

Fetch + Decode
+ Execution + Write

Textbook page: 457

Role of Cache Memory

- Each pipeline stage is expected to complete in one clock cycle.
- The clock period should be long enough to let the slowest pipeline stage to complete.
- Faster stages can only wait for the slowest one to complete.
- Since main memory is very slow compared to the execution, if each instruction needs to be fetched from main memory, pipeline is almost useless.

  (particularly important for the instruction fetch step)
- Fortunately, we have cache.

Role of Cache Memory

- Each pipeline stage is expected to complete in one clock cycle.
- The clock period should be long enough to let the slowest pipeline stage to complete.
- Faster stages can only wait for the slowest one to complete.
- Since main memory is very slow compared to the execution, if each instruction needs to be fetched from main memory, pipeline is almost useless.

  (particularly important for the instruction fetch step)
- Fortunately, we have cache.

Pipeline Performance

- The potential increase in performance resulting from pipelining is proportional to the number of pipeline stages.
- However, this increase would be achieved only if all pipeline stages require the same time to complete, and there is no interruption throughout program execution.
- Unfortunately, this is not true.

Pipeline Performance

Pipeline Performance

- The previous pipeline is said to have been stalled for two clock cycles.
- Any condition that causes a pipeline to stall is called a hazard.
- Data hazard – any condition in which either the source or the destination operands of an instruction are not available at the time expected in the pipeline. So some operation has to be delayed, and the pipeline stalls.
- Instruction (control) hazard – a delay in the availability of an instruction causes the pipeline to stall.
- Structural hazard – the situation when two instructions require the use of a given hardware resource at the same time.

[**pipeline stall** is a delay in execution of an instruction in order to resolve a hazard. ... It also **stalls** the instruction in the fetch stage, to prevent the instruction in that stage from being overwritten by the next instruction in the program].

Pipeline Performance

Instruction hazard

Idle periods – stalls (bubbles)

Pipeline Performance

Structural hazard

Load X(R1), R2
source,destination

Pipeline Performance

- Again, pipelining does not result in individual instructions being executed faster; rather, it is the throughput that increases.
- Throughput is measured by the rate at which instruction execution is completed.
- Pipeline stall causes degradation in pipeline performance.
- We need to identify all hazards that may cause the pipeline to stall and to find ways to minimize their impact.

Data Hazards

- Data hazards occur when instructions attempt to use resource before it is ready

Three types:

- read after write (RAW)
- write after read (WAR)
- write after write (WAW)

Data Hazards

Read after write (RAW)

- (i2 tries to read a source before i1 writes to it) A read after write (RAW) data hazard refers to a situation where an instruction refers to a result that has not yet been calculated or retrieved.

- This can occur because even though an instruction is executed after a prior instruction, the prior instruction has been processed only partly through the pipeline
- For example:

i1: ADD R1, R2, R3; //R1<- R2 + R3

i2: SUB R4, R1, R6; //R4<- R1 - R6

A data dependency occurs with instruction i2, as it is dependent on the completion of instruction i1.

Data Hazards

Write after Read (WAR)

- (i2 tries to write a destination before it is read by i1) A write after read (WAR) data hazard represents a problem with concurrent execution.

- For example:

i1: ADD R4, R1, R5; //R4<- R1 + R5

i2: ADD R5, R1, R2; //R5<- R1 + R2

In any situation with a chance that i2 may finish before i1 (i.e., with concurrent execution), it must be ensured that the result of register R5 is not stored before i1 has had a chance to fetch the operands.

Data Hazards

Write after write (WAW)

- i2 tries to write an operand before it is written by i1) A write after write (WAW) data hazard may occur in a concurrent execution environment.

- For Example:

I1:Add R2, R4, R7 // R2 <- R4 + R7

i2:Add R2, R1, R3 // R2 <- R1 + R3

The write back (WB) of i2 must be delayed until i1 finishes executing.

Control Hazards

Also called as instruction hazard

Overview

- Branching hazards (also termed control hazards) occur with branches.
- PC must contain address of next instruction before we know it!!!

Unconditional Branches

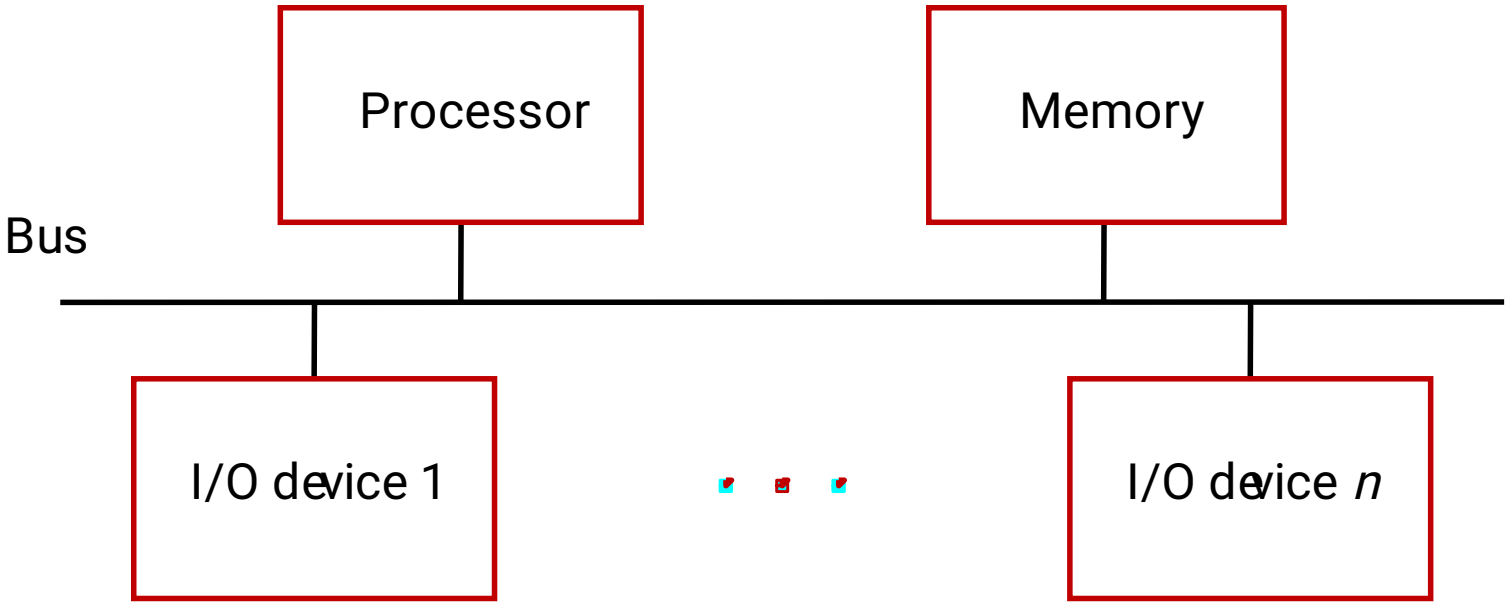
Structural Hazards

- When two or more different instructions want to use same hardware resource in same cycle

- e.g., MEM uses the same memory port as IF

THANK YOU

Accessing I/O Devices

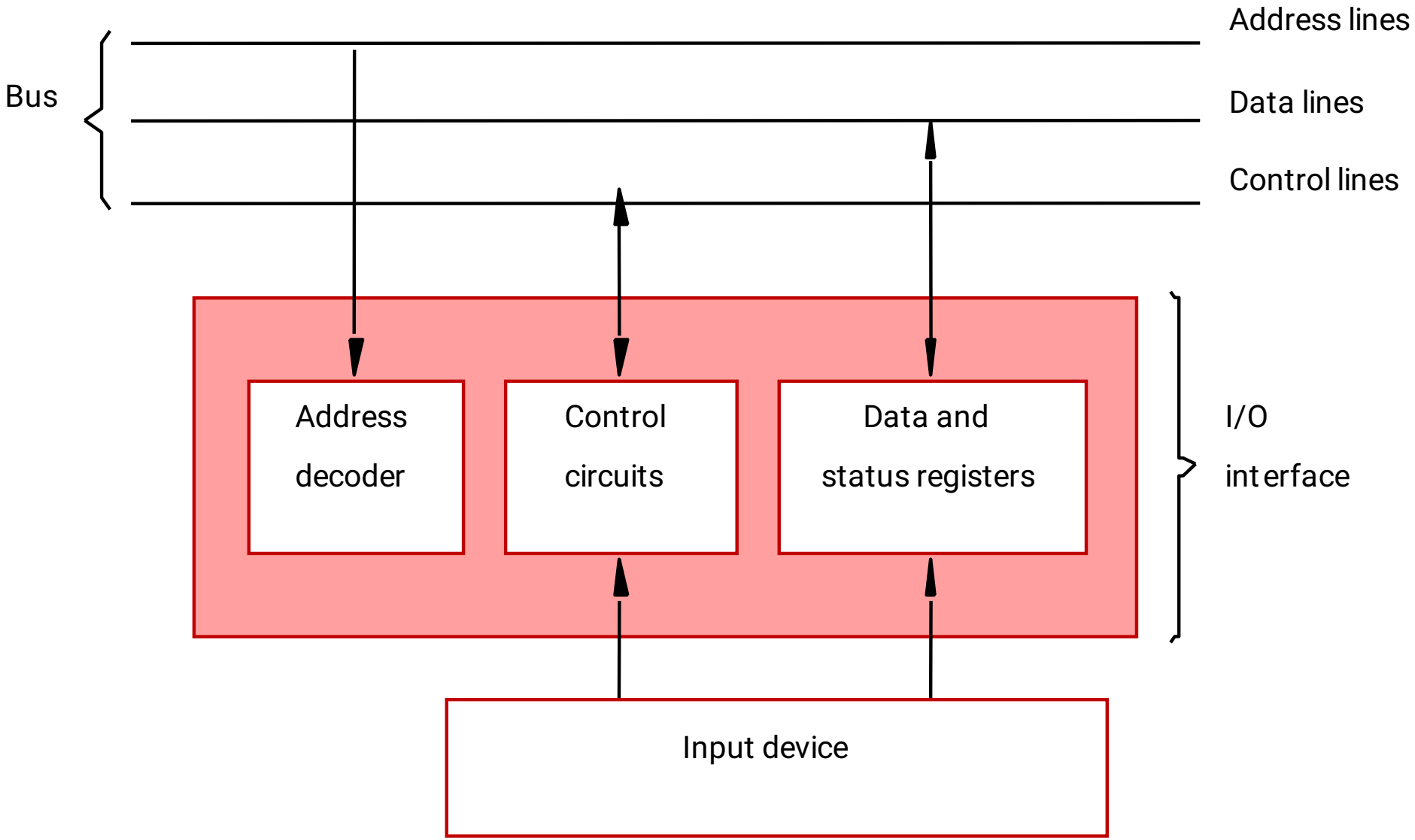


- Multiple I/O devices may be connected to the processor and the memory via a bus.
- Three sets of lines to carry address, data and control signals.
- Each I/O device is assigned an unique address.

Accessing I/O Devices

- I/O devices and the memory may share the same address space:
  - Memory-mapped I/O.
  - Any machine instruction that can access memory can be used to transfer data to or from an I/O device. Simpler software.
- I/O devices and the memory may have different address spaces:
  - Address lines may be shared between I/O devices and memory, with a control signal to indicate whether it is a memory address or an I/O address.

Accessing I/O Devices



Accessing I/O Devices

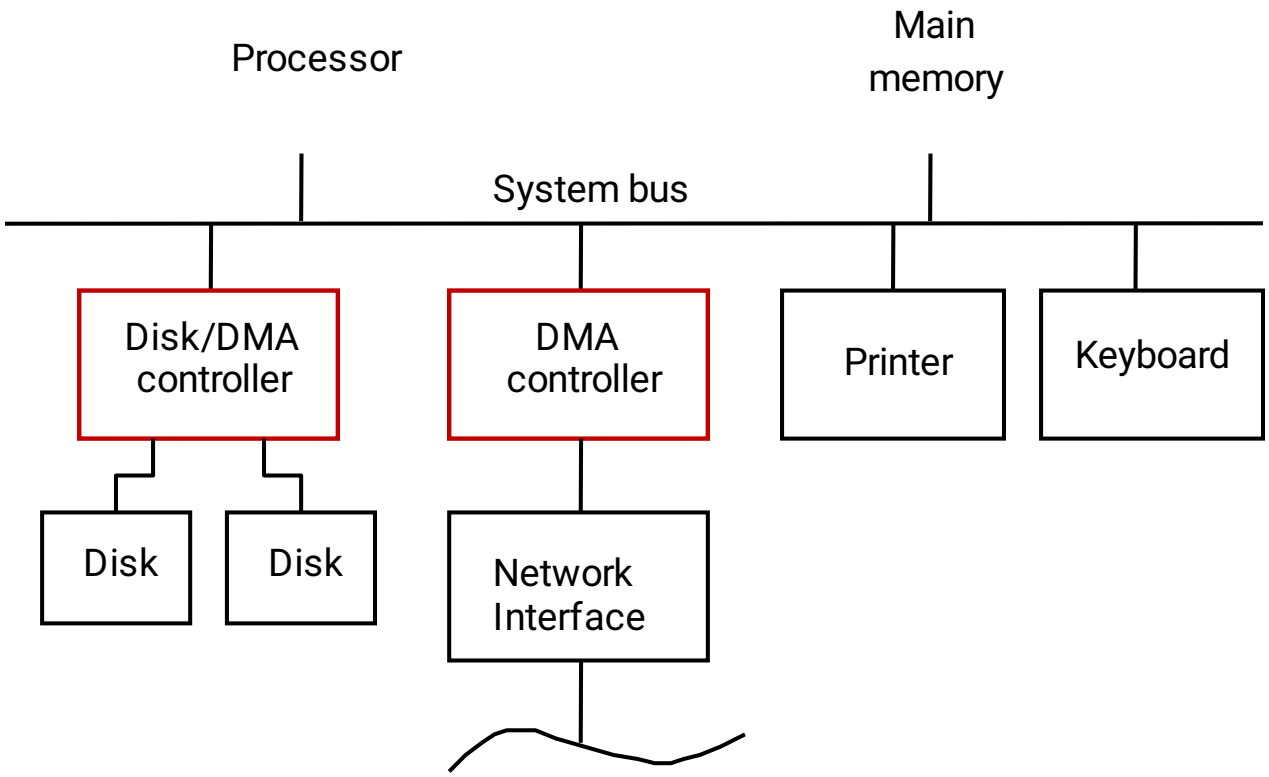
- Two other mechanisms used for synchronizing data transfers between the processor and memory:
  - Interrupts.
  - Direct Memory Access.

THANK YOU

Direct Memory Access

- Direct Memory Access (DMA):
  - A special control unit may be provided to transfer a block of data directly between an I/O device and the main memory, without continuous intervention by the processor.
  - For each word, it provides the memory address and all the control signals.
  - To transfer a block of data, it increments the memory addresses and keeps track of the number of transfers.

Direct Memory Access



Direct Memory Access

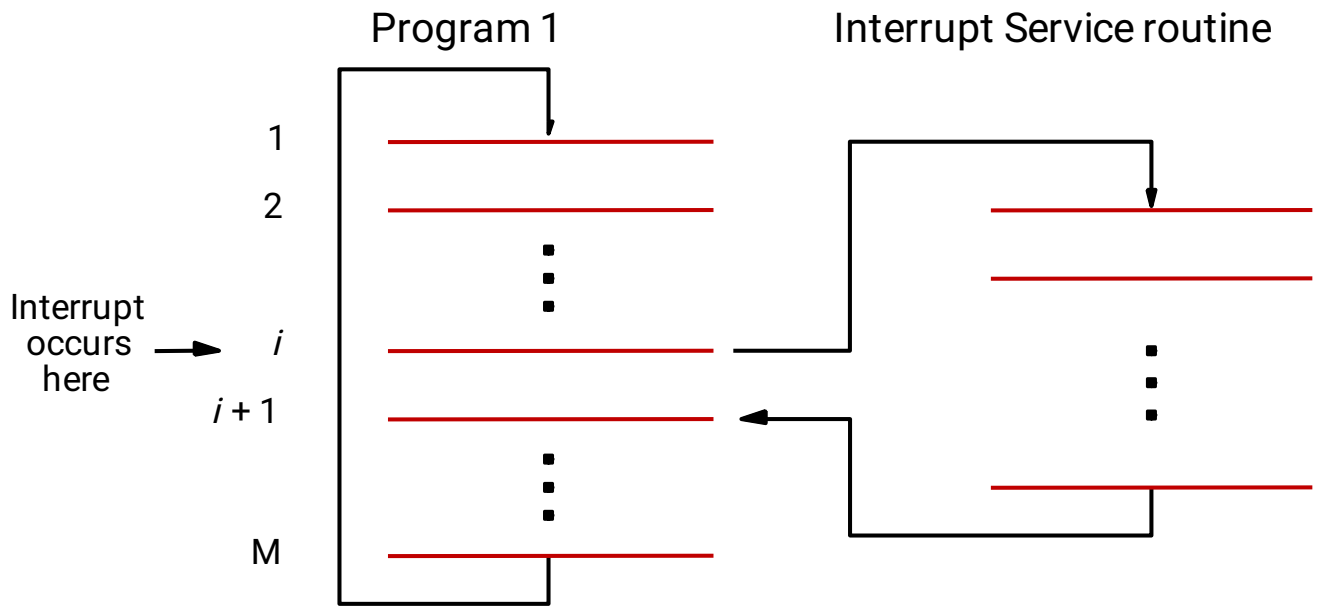
- Processor and DMA controllers have to use the bus in an interwoven fashion to access the memory.
- Processor originates most memory access cycles on the bus.
- An alternate approach is the provide a DMA controller an exclusive capability to initiate transfers on the bus, and hence exclusive access to the main memory. This is known as the block or burst mode.

Interrupts

- In program-controlled I/O, when the processor continuously monitors the status of the device, it does not perform any useful tasks.
- An alternate approach would be for the I/O device to alert the processor when it becomes ready.
  - Do so by sending a hardware signal called an interrupt to the processor.
  - At least one of the bus control lines, called an interrupt-request line is dedicated for this purpose.
- Processor can perform other useful tasks while it is waiting for the device to be ready.

THANK YOU

Interrupts


Program 1
Interrupt Service routine
1
2
Interrupt occurs here
i
i + 1
M

Interrupts

- Treatment of an interrupt-service routine is very similar to that of a subroutine.
- However there are significant differences:
  - A subroutine performs a task that is required by the calling program.
  - Interrupt-service routine may not have anything in common with the program it interrupts.

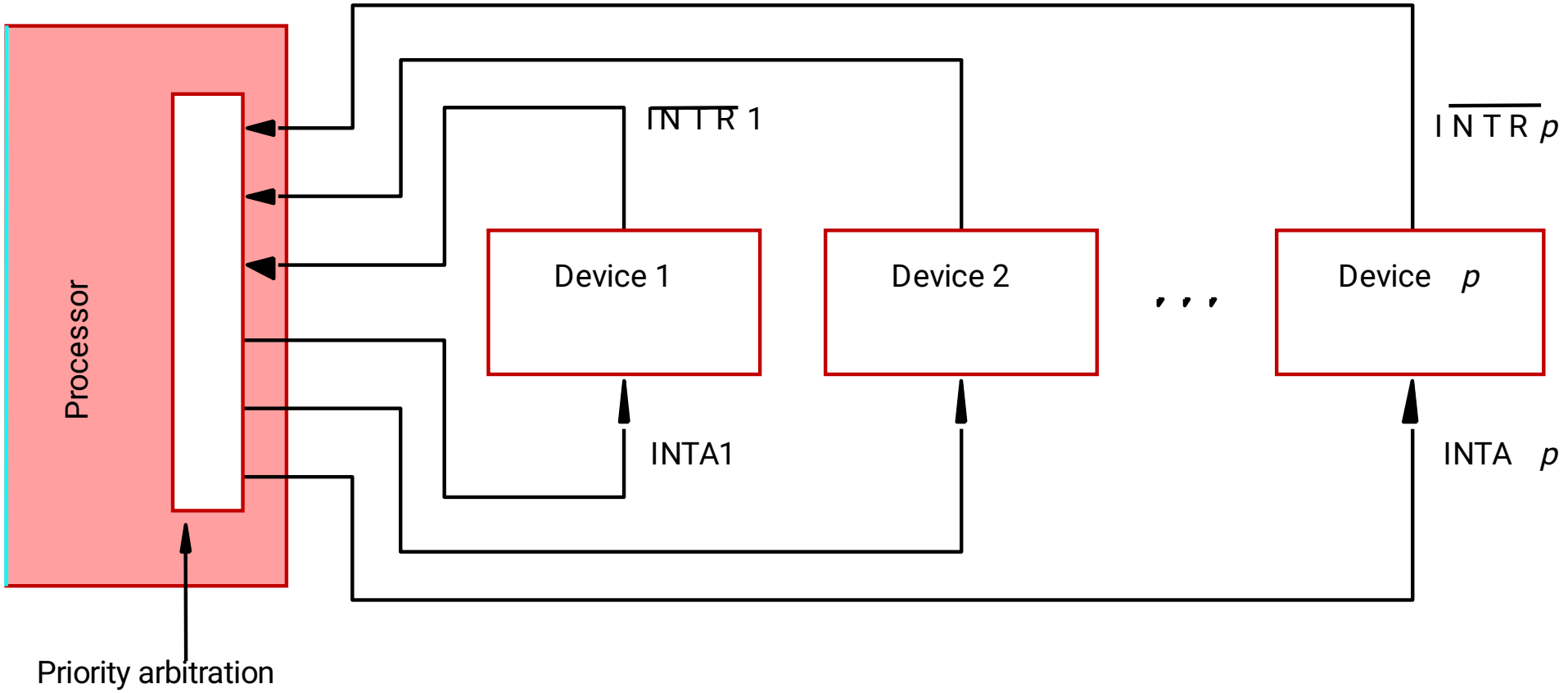
Interrupts

- Interrupt-service routine and the program that it interrupts may belong to different users.
- As a result, before branching to the interrupt-service routine, not only the PC, but other information such as condition code flags, and processor registers used by both the interrupted program and the interrupt service routine must be stored.
- This will enable the interrupted program to resume execution upon return from interrupt service routine.

Interrupts

- Which interrupt request does the processor accept if it receives interrupt requests from two or more devices simultaneously?.
- If the I/O devices are organized in a priority structure, the processor accepts the interrupt request from a device with higher priority.
- However, if the devices share an interrupt request line, then how does the processor decide which interrupt request to accept?

Interrupts


Processor
INTR 1
INTR p
Device 1
Device 2
. . .
Device p
INTA1
INTA p
Priority arbitration

THANK YOU

References Books:

1. Computer Organization by V.Carl Hamacher, Zvonko Vranesic, Safwat Zaky , 5th Edition-McGrawHill Publication.
2. Computer Organization and Architecture by Willaiam Staliing 6th edition Pearson Education
3. Computer Architecture & Organization By J.P. Hayes 3rd edition McGrawHill Publication