Final Exam Review

Important:
Do not assume that the topic covered here are the only stuff for the preparation of exam.

Recommended:
A comprehensive study from lecture 1 till last lecture is required for better grade

Architecture & Organization

- All Intel x86 family share the same basic architecture
  - 8086, 8088, 80286, 80386, 80486, Pentium
- The IBM System/370 family share the same basic architecture
  - Initial announcement in 1964 included Models 30, 40, 50, 60, 62, and 70
  - Advantage?
    - This gives code compatibility
      - At least backwards
  - The particular functions a microprocessor performs are dictated by software (Assembly Language).
- Organization differs between different versions

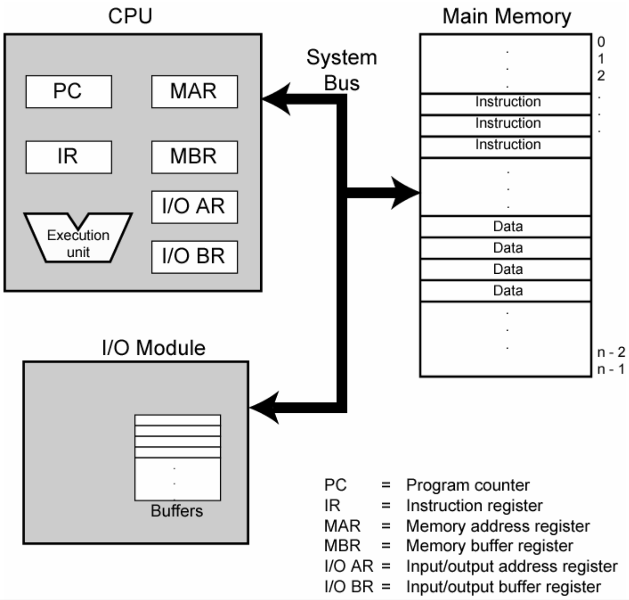
Structure & Function

- A computer is a complex system contains millions of electronic components
  - The components are arranged in a hierarchal way as a set of interrelated subsystems
- The designer need to deal with a particular level of the system at any one time, and concerned with structure and function at each level
  - Structure
    - the way in which components relate to each other
  - Function
    - the operation of individual components as part of the structure

General Architecture

- The hardware of a microcomputer system can be divided in four functional sections
  - The input unit
  - The microprocessor unit
  - The memory unit
  - The output unit
- Each unit has a special function
- The communication medium that is linking these units is bus
  - The address bus
  - The data bus
  - The control bus

Computer Components: Top Level View

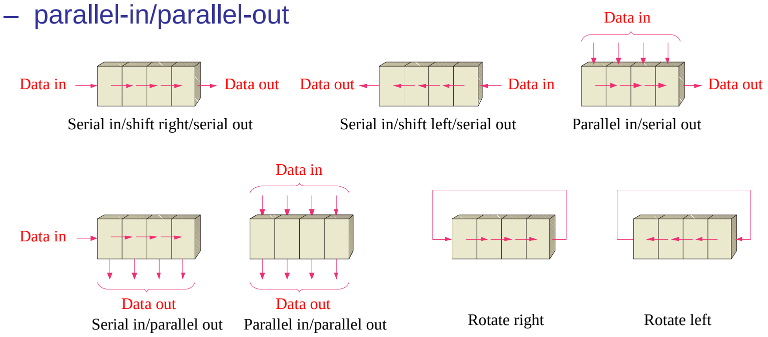


Registers

- Registers → a common sequential device:
  - They are frequently used in building larger sequential circuits.
- A register is a collection of flip-flops
  - basic function is to hold information
- Registers hold larger quantities of data than individual flip-flops.
  - Registers are central to the design of modern processors.
- One register is simply a set of D flip-flops, one per bit
  - Data inputs are D's
  - Data outputs are Q's and Q's
  - Clocks all tied together

Shift Registers

- A shift register is an arrangement of flip-flops with important applications in storage and movement of data.
- A shift register is a register that moves information on the clock signal
  - serial-in/serial-out
  - serial-in/parallel-out
  - parallel-in/serial-out
  - parallel-in/parallel-out

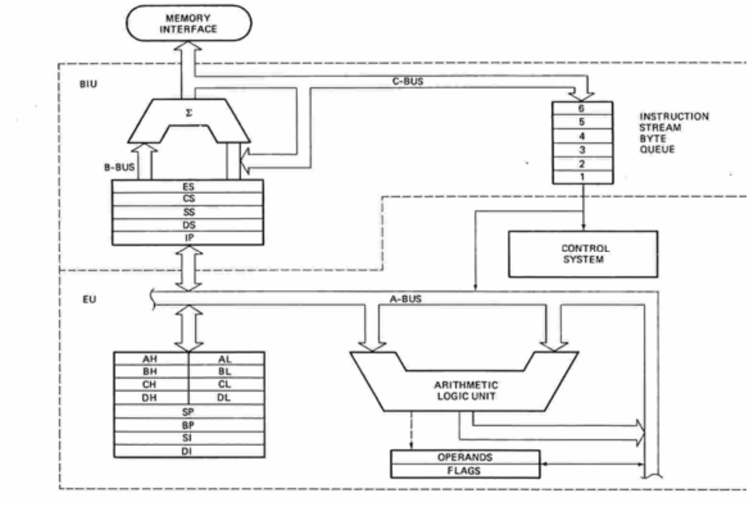


8086 Architecture

- The microprocessors functions as the CPU in the stored program model of the digital computer.
- The microprocessor also has a S/W function.
  - It must recognize, decode, and execute program instructions fetched from the memory unit.
    - Fetch – Decode – Execute – Store Cycle
  - This requires an Arithmetic-Logic Unit (ALU) within the CPU to perform arithmetic and logical (AND, OR, NOT, compare, etc) functions.
- The 8088/ 8086 CPUs are organized as two processing units, called the
  - Bus Interface Unit (BIU) and the Execution Unit (EU).
    - Each unit has a dedicated function and both operate at the same time

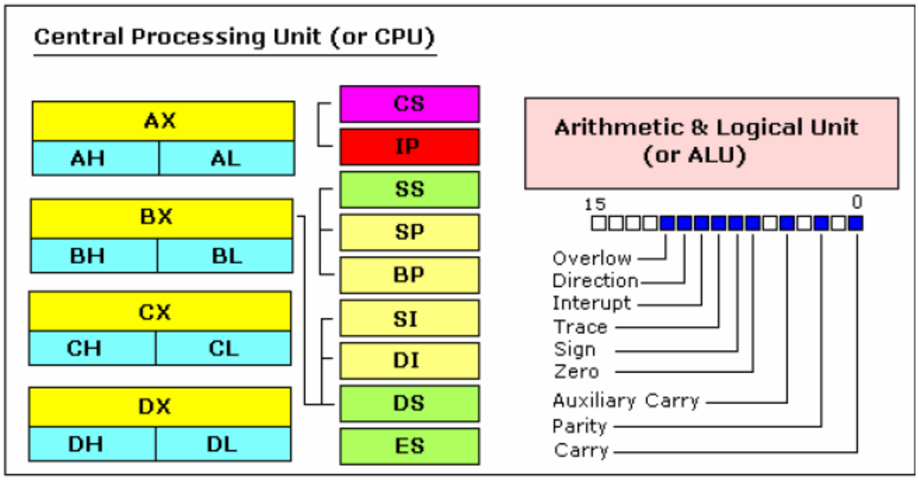
Architecture

A typical x86 Architecture



Intel 8086 Architecture, the 1st member of x86 family

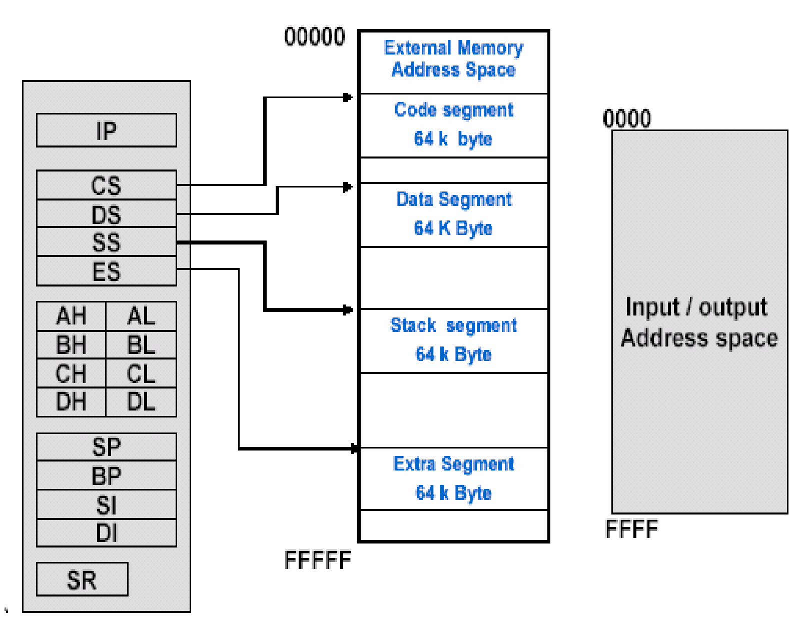
Inside CPU



8086 Registers

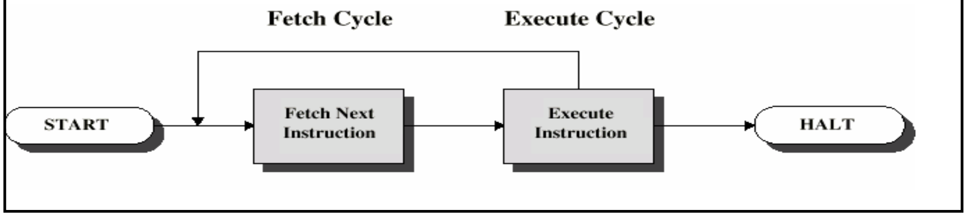
Segment Registers		
CS	Code Segment	16-bit number that points to the active code-segment
DS	Data Segment	16-bit number that points to the active data-segment
SS	Stack Segment	16-bit number that points to the active stack-segment
ES	Extra Segment	16-bit number that points to the active extra-segment
Pointer Registers		
IP	Instruction Pointer	16-bit number that points to the offset of the next instruction
SP	Stack Pointer	16-bit number that points to the offset that the stack is using
BP	Base Pointer	used to pass data to and from the stack
General-Purpose Registers		
AX 16 bit (AH +AL 8 bit)	Accumulator Register	mostly used for calculations and for input/output
BX (BH + BL)	Base Register	Only register that can be used as an index
CX (BH + BL)	Count Register	register used for the loop instruction
DX (BH + BL)	Data Register	input/output and used by multiply and divide
Index Registers		
SI	Source Index	used by string operations as source
DI	Destination Index	used by string operations as destination

Software Model of 8086



Instruction fetch and execute

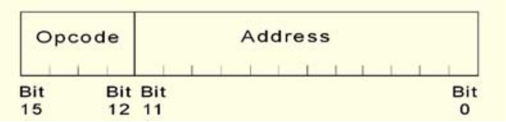
- At the beginning of each instruction cycle
  - Processor fetches an instruction from the memory
    - A register "PC"- program counter holds the address of the instruction to be fetched next
    - Processor always increment the PC after each instruction unless told otherwise
  - The fetched instruction is loaded into a register located in the processor – " the instruction Register (IR)"
  - The instruction contains bits – to specify the action the processor is to take



Example of Program Execution

Example

Add the contents of the memory at address 940 to the contents of the memory at address 941 and store the result in the next location



Internal CPU registers:

Program counter (PC): Address of next instruction

Instruction Register (IR): Current instruction

Accumulator (AC): Temporary Storage

Partial List of Opcodes:

0001 = (1h) = Load AC from Memory

0010 = (2h) = Store AC to memory

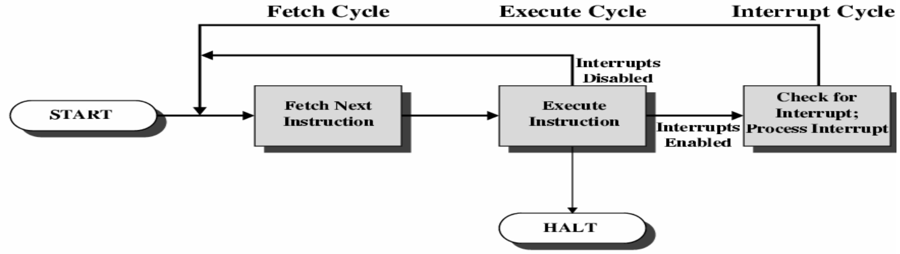
0101 = (5h) = Add to AC from Memory

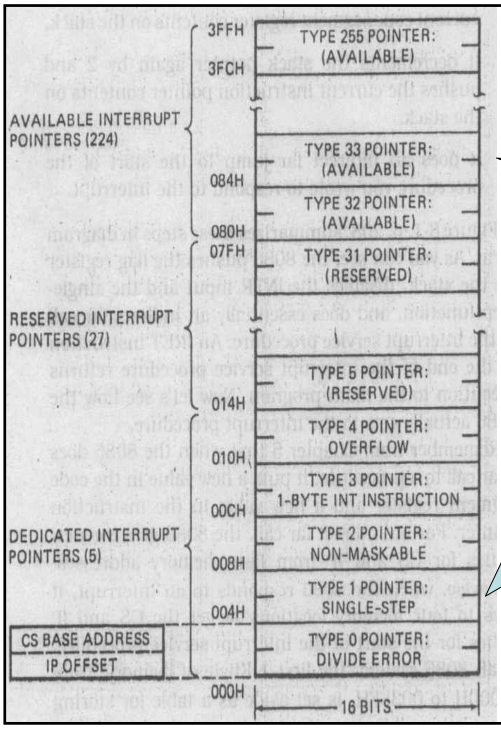
Interrupts

- All computers provide the mechanism by which other modules (e.g. I/O) may interrupt normal sequence of processing
- Most common classes of interrupts are:
  - Program: generated by some condition e.g. overflow, division by zero
  - Timer: Generated by internal processor timer, allow Operating system to perform certain functions at regular intervals
  - I/O: from I/O controller
  - Hardware failure: generated by a failure such as power

Interrupt and Instruction Cycle

- To accommodate interrupts, in interrupt cycle is added to the instruction cycle
  - Processor check for the occurrence of interrupt
  - If no interrupt the processor proceeds to fetch cycle and fetch next instruction of the current program
  - In case of interrupt
    - It suspends execution of current program and saves the address of next instruction
    - Set the PC to starting address of interrupt handler routine

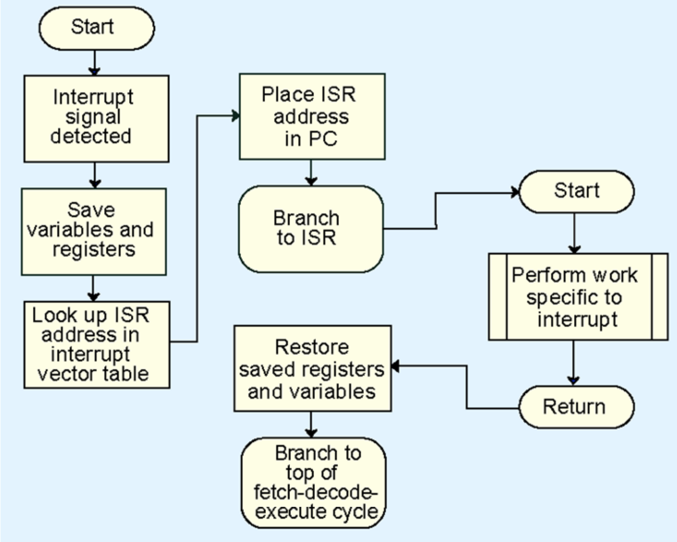




224 out of 256 interrupt types are available for hardware and software interrupts

Vector for each interrupt type requires four memory location.
To respond an interrupt 8086 multiplies the type by 4 to produce the desired address in the vector table

Instruction Processing with interrupt

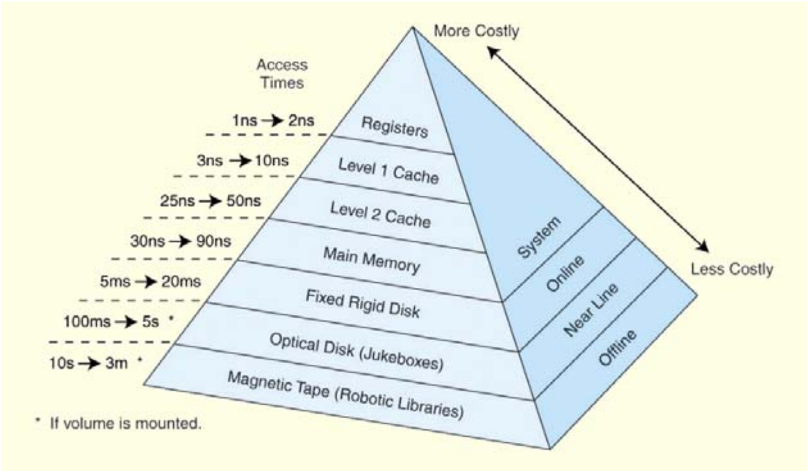

Start
Interrupt signal detected
Save variables and registers
Look up ISR address in interrupt vector table
Place ISR address in PC
Branch to ISR
Start
Perform work specific to interrupt
Return
Restore saved registers and variables
Branch to top of fetch-decode-execute cycle

Types of Memory

- There are two kinds of main memory:
  - *Random Access Memory, RAM, and* Read-only-*Memory, ROM*.
- There are two types of RAM:
  - Dynamic RAM (DRAM) and Static RAM (SRAM).
    - Dynamic RAM consists of capacitors that slowly leak their charge over time. Thus they must be refreshed every few milliseconds to prevent data loss.
    - DRAM is “cheap” memory owing to its simple design.
  - SRAM consists of circuits similar to the D flip-flop
    - SRAM is very fast memory and it doesn’t need to be refreshed like DRAM does. It is used to build cache memory, which we will discuss in detail later.

The Memory Hierarchy

- This storage organization can be thought of as a pyramid:



Memory Organization

- Build 32K x 16 memory with 2K x 8 RAM chips
  - Connect 16 rows and 2 columns of chips together
- Each row addresses 2k words
  - It requires two chips to handle the full width
- Address of this memory must be of 15 bits
  - For 32 k = 2^{5} X 2^{10}
- But each chip pair (row) requires only 11 address lines 2^{11}
- So a decoder will be require to decode the left most 4 bits of the address to determine which chip is holding the desire address
- Once the proper chip has been identifies
  - Remaining 11 bits will be input to another decoder to find the desired address within that chip

2k x 8	2k x8	Row 0
2k x 8	2k x8	Row 1
2k x 8	2k x8	Row15

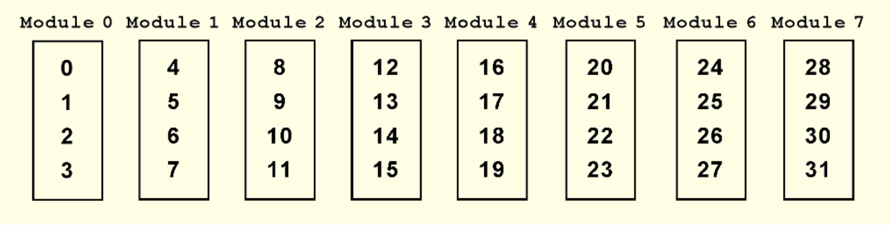
Memory Organization

- Physical memory usually consists of more than one RAM chip.
  - Access is more efficient when memory is organized into banks of chips with the addresses interleaved across the chips
- With low-order interleaving, the low order bits of the address specify which memory bank contains the address of interest.

Module 0	Module 1	Module 2	Module 3	Module 4	Module 5	Module 6	Module 7
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

Memory Organization

- With high-order interleaving, the high order address bits specify the memory bank.



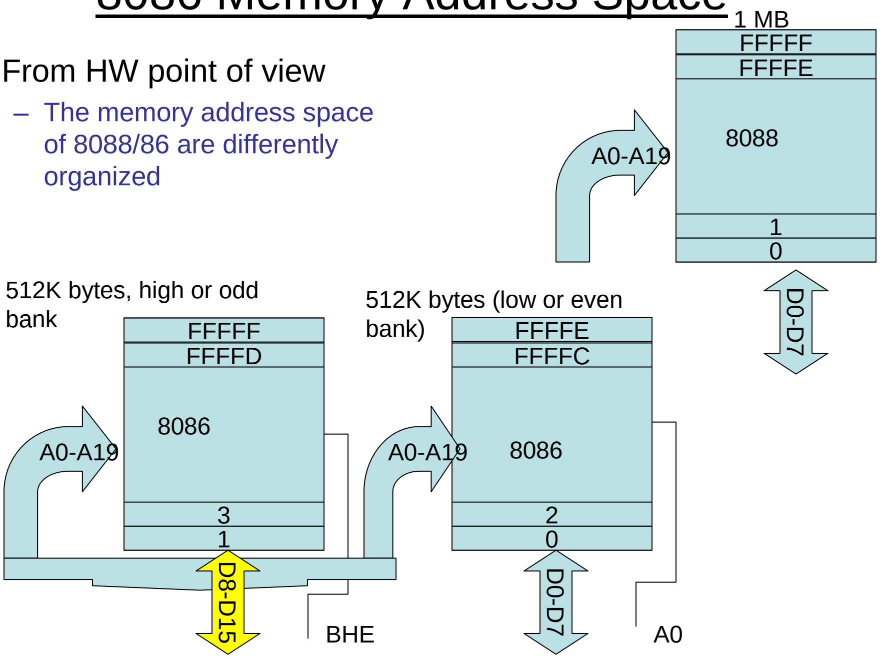
Module 0	Module 1	Module 2	Module 3	Module 4	Module 5	Module 6	Module 7
0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31

Decoding Memory Address

- The processor can usually address a memory space that is much larger than the memory space covered by an individual memory chip.
  - In order to splice a memory device into the address space of the processor, decoding is necessary.
    - For example, the 8086 issues **20-bit**addresses for a total of **1MB** of memory address space.
    - However, the BIOS on a 2716 EPROM has only 2KB of memory and **11**address pins.
  - A decoder can be used to decode the additional 9 address pins and allow the EPROM to be placed in **any** 2KB section of the 1MB address space

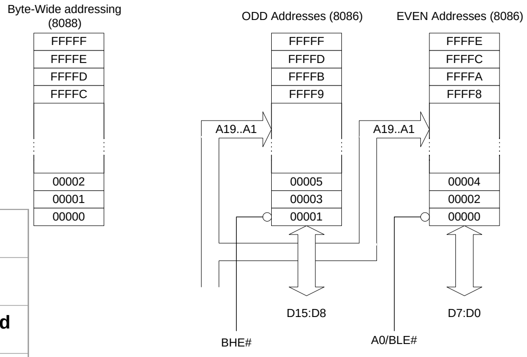
8086 Memory Address Space

- From HW point of view
  - The memory address space of 8088/86 are differently organized



BHE Signal (8086 Only)

- The 8086 processor can address memory a byte at a time
  - Its data bus is 16b wide
  - It uses the BHE# signal and A0 (sometimes called BLE#) to address bytes using its 16b bus
- Use of BHE#/A0

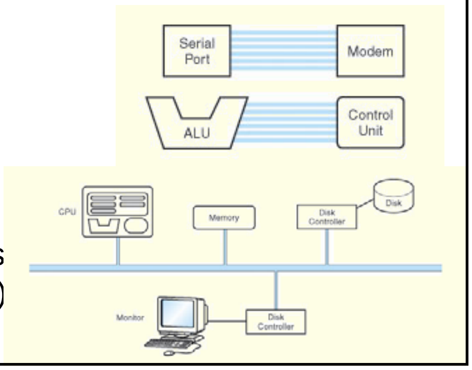


BHE	A0	Selection
0	0	Whole word (16-bits)
0	1	High byte to/from odd address
1	0	Low byte to/from even address
1	1	No selection

The Bus

- The speed of the bus is affected by its length as well as by the number of devices sharing it.
  - Quite often, devices are divided into *master* and *slave* categories,
    - A master device is one that initiates actions and
    - A slave is one that responds to requests by a master.

A bus can be *point-to-point***,** connecting two specific components or it can be a *common pathway* that connects a number of devices, requiring these devices to share the bus (*multipoint* bus)



The Bus

- In a master-slave configuration, where more than one device can be the bus master, concurrent bus master requests must be arbitrated.
- Four categories of bus arbitration are:

- **Daisy chain:**
  - Permissions are passed from the highest-priority device to the lowest.
    - "starved out", simple but not fair.
- **Centralized parallel:**
  - Each device is directly connected to an arbitration circuit.
    - Bottlenecks can result

- **Distributed using self-detection:**
  - Devices decide which gets the bus among themselves.
    - Devices not the central arbiter
- **Distributed using collision-detection:**
  - Any device can try to use the bus. If its data collides with the data of another device, it tries again.
    - Ethernet

Types

- Synchronous Bus: (Processor–Memory)
  - Occurrence of event on the bus is coordinated by clock
  - Advantage:
    - Can run very fast
  - Disadvantages:
    - Devices on the bus must run at the same clock rate
    - Clock skew (clock signal sent from the clock circuit arrives at different components at different times) effects bus length
      - Bus must be kept as short as possible

Types

- Asynchronous Bus: (I/O)
  - To ensure timings, control bus coordinate the operations through handshake protocol
    - Occurrence of event on the bus depends upon occurrence of previous event and additional control lines (ReadReq, Ack, DataRdy etc) are required
  - Advantage:
    - It can accommodate a wide range of devices
  - Disadvantage:
    - Complex communication protocol is needed

Performance Equation

- Clock speed should not be confused with CPU performance.
- The CPU time required to run a program is given by the general performance equation:

$$\texttt{CPU Time} = \frac{\texttt{seconds}}{\texttt{program}} = \frac{\texttt{instructions}}{\texttt{program}} \times \frac{\texttt{avg. cycles}}{\texttt{instruction}} \times \frac{\texttt{seconds}}{\texttt{cycle}}$$

  - We can improve CPU throughput when we:
    - reduce the number of instructions in a program,
    - reduce the number of cycles per instruction, or
    - reduce the number of nanoseconds per clock cycle.

I/O subsystem

- Exchange of data is performed
  - Programmed I/O
  - Memory mapped I/O
    - where the I/O device behaves like main memory from the CPU's point of view.
    - The registers in the interface appears in memory map
      - Advantage faster access, disadvantage memory space compromised
  - Isolated I/O
    - Or isolated /instruction-based, where the CPU has a specialized I/O instruction set.
      - Advantage saving to memory space, disadvantage requires specialized I/O instructions
  - Direct Memory Access

Amdahl's Law

- The overall performance of a system is a result of the interaction of all of its components.
- System performance is most effectively improved when the performance of the most heavily used components is improved.
- This idea is quantified by Amdahl's Law:

$$S = \frac{1}{(1-f) + \frac{f}{k}}$$

where S is the overall speedup; f is the fraction of work performed by a faster component; and k is the speedup of the faster component.

CPU Execution Time: Example

- A Program is running on a specific machine with the following parameters:
  - Total instruction count (I): 10,000,000 instructions
  - Average CPI for the program (CPI): 2.5 cycles/instruction.
  - CPU clock rate (C): 200 MHz.
- What is the execution time for this program:

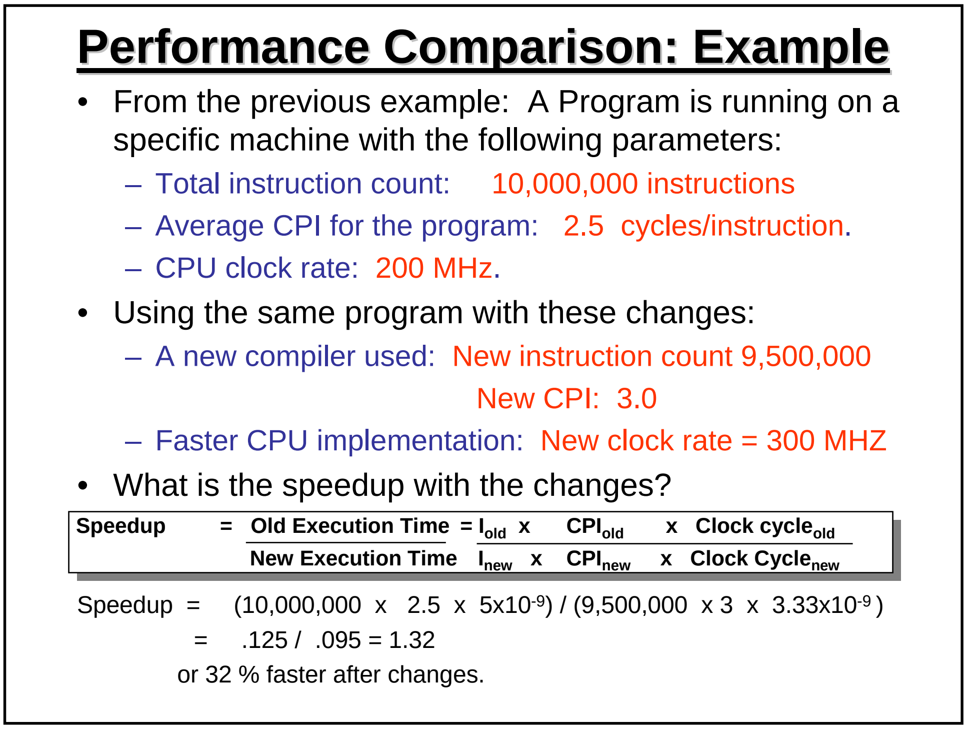
$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

CPU time = Instruction count x CPI x Clock cycle

= 10,000,000 x 2.5 x 1 / clock rate

= 10,000,000 x 2.5 x $5\text{x}10^{-9}$

= 0.125 seconds



Performance Comparison: Example

- From the previous example: A Program is running on a specific machine with the following parameters:
  - Total instruction count: 10,000,000 instructions
  - Average CPI for the program: 2.5 cycles/instruction.
  - CPU clock rate: 200 MHz.
- Using the same program with these changes:
  - A new compiler used: New instruction count 9,500,000

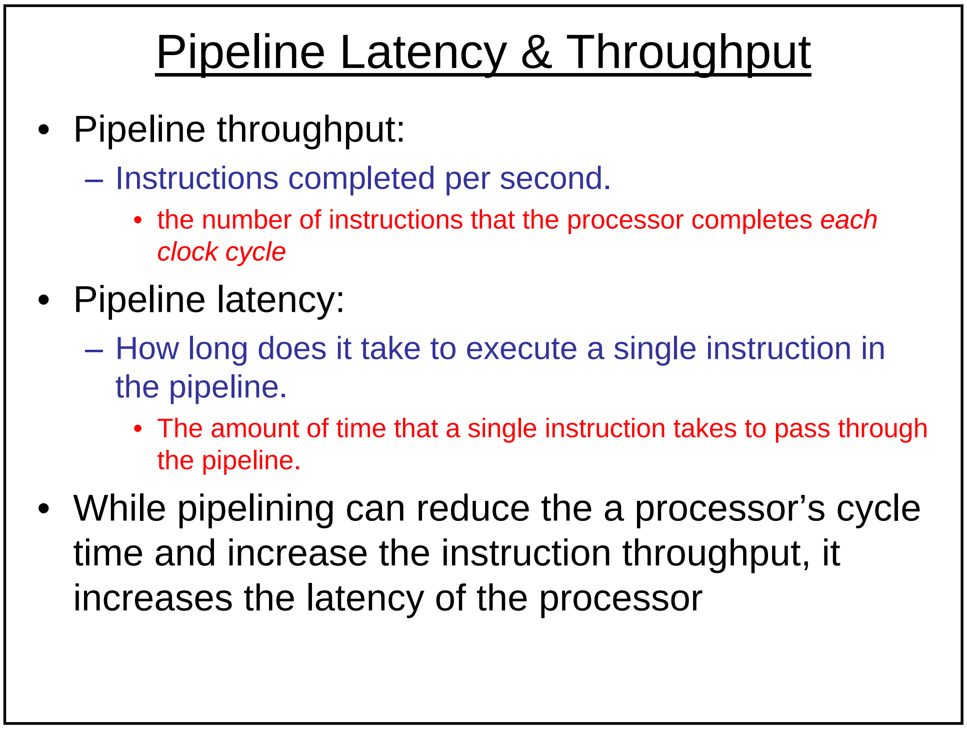
    New CPI: 3.0
  - Faster CPU implementation: New clock rate = 300 MHZ
- What is the speedup with the changes?

$$\text{Speedup} = \frac{\text{Old Execution Time}}{\text{New Execution Time}} = \frac{I_{old} \times \text{CPI}_{old} \times \text{Clock cycle}_{old}}{I_{new} \times \text{CPI}_{new} \times \text{Clock Cycle}_{new}}$$

Speedup = $(10{,}000{,}000 \times 2.5 \times 5\text{x}10^{-9}) / (9{,}500{,}000 \times 3 \times 3.33\text{x}10^{-9})$

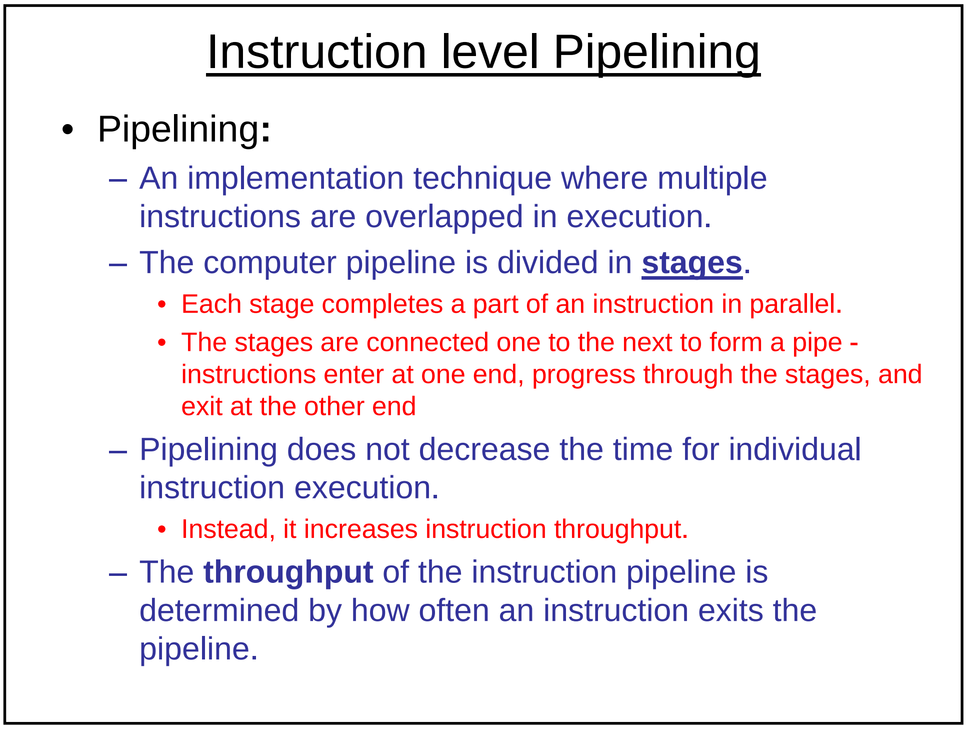
= .125 / .095 = 1.32

or 32 % faster after changes.



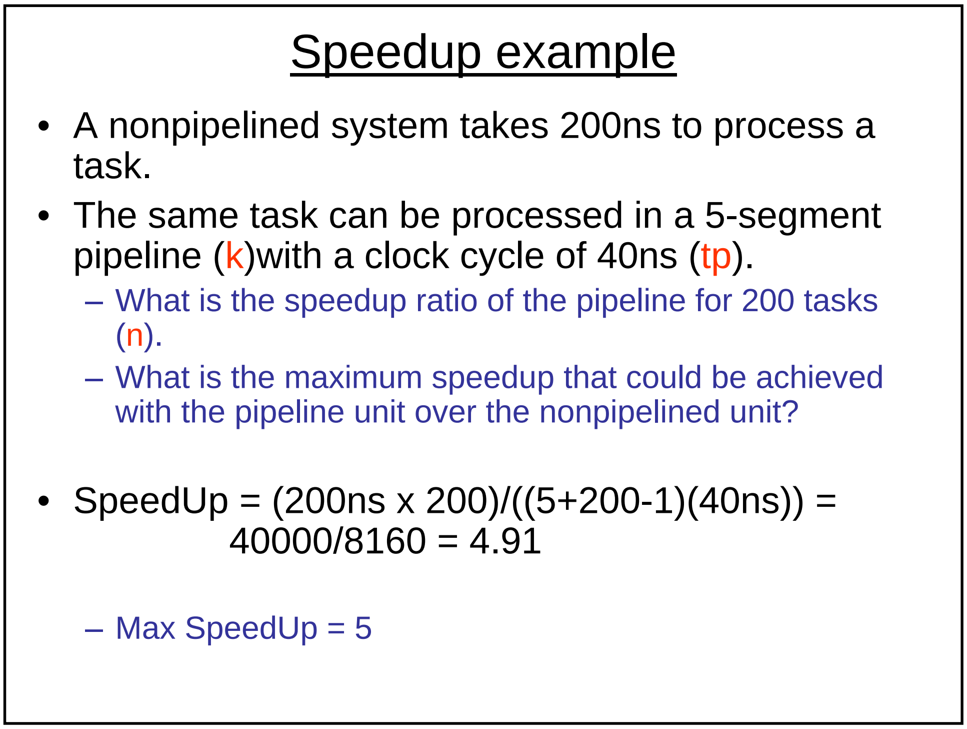
Pipeline Latency & Throughput

- Pipeline throughput:
  - Instructions completed per second.
    - the number of instructions that the processor completes *each clock cycle*
- Pipeline latency:
  - How long does it take to execute a single instruction in the pipeline.
    - The amount of time that a single instruction takes to pass through the pipeline.
- While pipelining can reduce the a processor’s cycle time and increase the instruction throughput, it increases the latency of the processor



Instruction level Pipelining

- Pipelining**:**
  - An implementation technique where multiple instructions are overlapped in execution.
  - The computer pipeline is divided in **stages**.
    - Each stage completes a part of an instruction in parallel.
    - The stages are connected one to the next to form a pipe - instructions enter at one end, progress through the stages, and exit at the other end
  - Pipelining does not decrease the time for individual instruction execution.
    - Instead, it increases instruction throughput.
  - The **throughput** of the instruction pipeline is determined by how often an instruction exits the pipeline.



Speedup example

- A nonpipelined system takes 200ns to process a task.
- The same task can be processed in a 5-segment pipeline (k)with a clock cycle of 40ns (tp).
  - What is the speedup ratio of the pipeline for 200 tasks (n).
  - What is the maximum speedup that could be achieved with the pipeline unit over the nonpipelined unit?

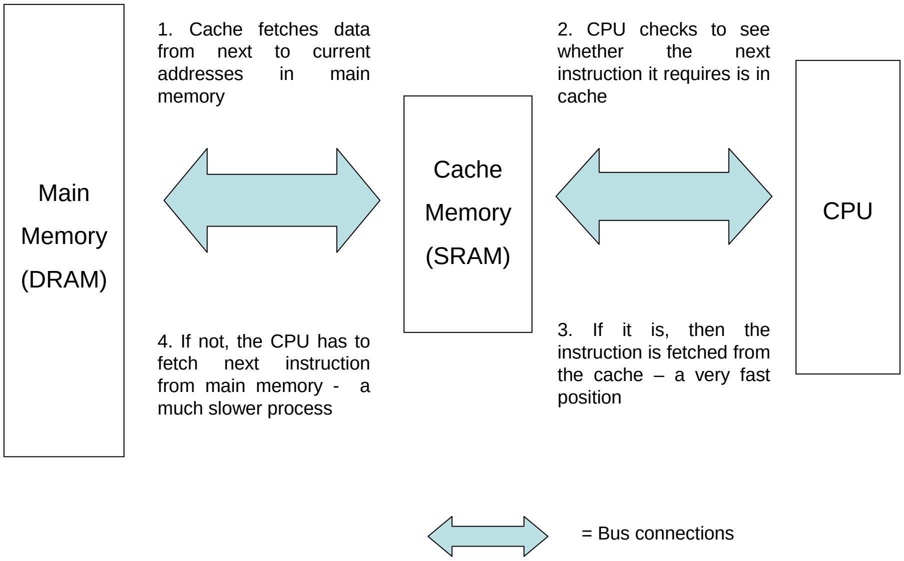
- SpeedUp = (200ns x 200)/((5+200-1)(40ns)) =
  40000/8160 = 4.91

  - Max SpeedUp = 5

Cache Memories

- Cache:
  - Small, Fast SRAM based memory, sit between large memory and the CPU
    - May be located on CPU chip or module
  - Contains the copy of the portion of main memory
    - Holds the operands and instructions most likely to be used by the CPU
  - Temporary store for often used instructions
    - Level 1 cache is built within the CPU (internal)
    - Level 2 cache may be on chip or nearby (external)
  - Faster for CPU to access than main memory
- The cache is placed both physically closer and logically closer to the CPU than the main memory.

The operation of cache memory



1. Cache fetches data from next to current addresses in main memory

2. CPU checks to see whether the next instruction it requires is in cache

3. If it is, then the instruction is fetched from the cache – a very fast position

4. If not, the CPU has to fetch next instruction from main memory - a much slower process

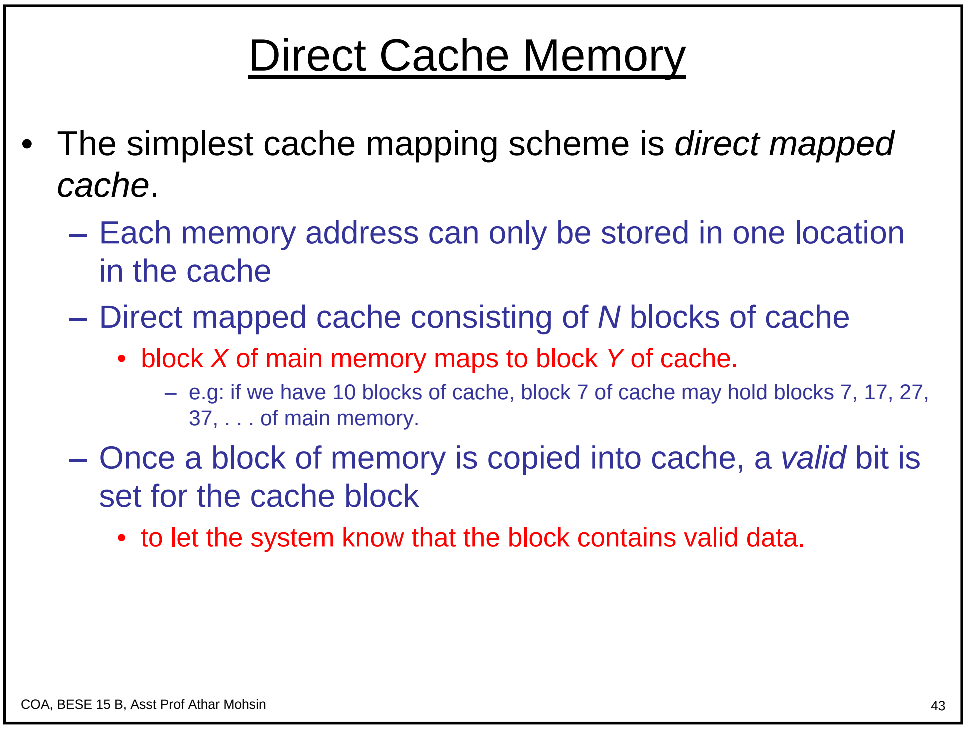
= Bus connections

Cache Terminologies

- Cache Line/Block
  - Cache memory is subdivided into cache lines
  - Cache Lines / Blocks:
    - The smallest unit of memory than can be transferred between the main memory and the cache.
- Tag / Index
  - Every address field consists of two primary parts:
    - A dynamic (tag) which contains the higher address bits, and
    - A static (index) which contains the lower address bits
      - The Dynamic Tag may be modified during run-time while the Static Tag one is fixed.

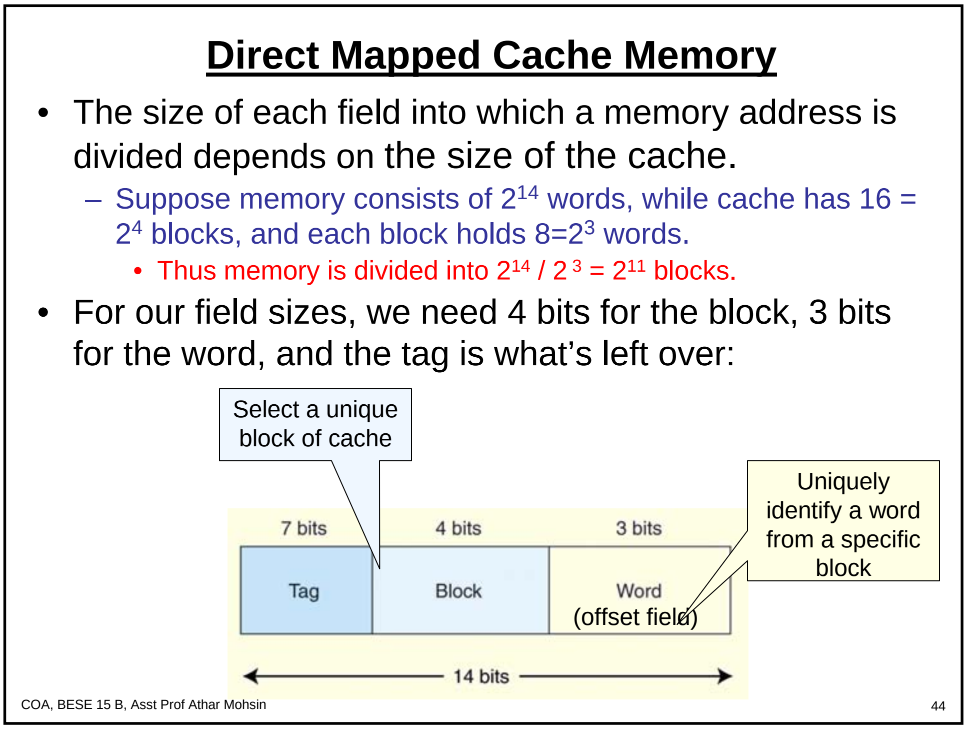
Describing the Cache

- Capacity
  - It is simply the amount of data that can be stored in the cache
    - A 32 KB can store 32 KB of the data
- Line length
  - It is the cache's block size
    - The size of the data that are brought into and thrown out of the cache in response to a miss
      - When a cache with 32 byte cache lines has a cache miss, it brings a 32 byte block of data containing the address of miss into cache, evicting a 32 byte to make room
- Associatively
  - It determine how many locations within the cache may contains a given memory address



Direct Cache Memory

- The simplest cache mapping scheme is *direct mapped cache*.
  - Each memory address can only be stored in one location in the cache
  - Direct mapped cache consisting of *N* blocks of cache
    - block *X* of main memory maps to block *Y* of cache.
      - e.g: if we have 10 blocks of cache, block 7 of cache may hold blocks 7, 17, 27, 37, . . . of main memory.
  - Once a block of memory is copied into cache, a *valid* bit is set for the cache block
    - to let the system know that the block contains valid data.



Direct Mapped Cache Memory

- The size of each field into which a memory address is divided depends on the size of the cache.
  - Suppose memory consists of 2^{14} words, while cache has 16 = 2^4 blocks, and each block holds 8=2^3 words.
    - Thus memory is divided into 2^{14} / 2^{3} = 2^{11} blocks.
- For our field sizes, we need 4 bits for the block, 3 bits for the word, and the tag is what's left over:

Select a unique block of cache

Uniquely identify a word from a specific block

Mapping Functions

- A mapping function is the method used to locate a memory address within a cache
  - It is used when copying a block from main memory to the cache and it is used again when trying to retrieve data from the cache
- There are three kinds of mapping functions
  - Direct
  - Full Associative
  - Set Associative

Fully Associative Cache

- Allowing a main memory block to be placed anywhere in cache.
  - The only way to find a block mapped this way is to search all of cache
    - requires the entire cache to be built from *associative memory* so it can be searched in parallel.
  - Compare the requested tag to *all* tags in cache to determine whether the desired data block is present in cache.
- The main memory address is partitioned into two pieces, the tag and the word.

Tag	word id bits

Fully Associative

- Any block from main memory can map to any location in the cache
  - word id bits are used to identify which word in the block is needed,
  - The tag becomes all of the remaining bits.
- we could allow a block to go anywhere in cache.

- In this way, cache would have to fill up before any blocks are evicted.
  - This is how *fully associative* cache works.

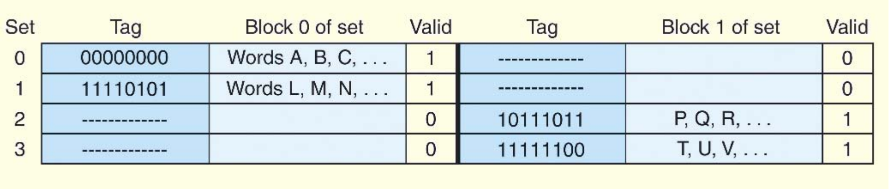
Tag	word id bits

N-way set associative cache mapping

- A combination of Direct and Fully Associative Mapping approaches.
  - Scheme is similar to direct mapped cache,
    - The address is used to map the block to a certain cache location.
  - The important difference:
    - Instead of mapping to a single cache block, an address maps to a *set* of several cache blocks.
  - All sets in cache must be the same size.
- similar to the way in which fully associative cache works.
  - Instead of mapping anywhere in the entire cache, a memory reference can map only to the subset of cache slots.

Set Associative Mapping

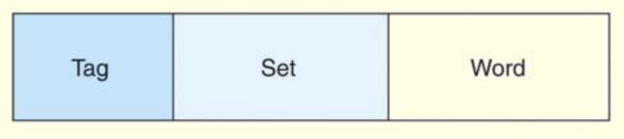
- The number of cache blocks per set in set associative cache varies according to overall system design.
  - For example, in a 2-way set associative cache, there are two cache blocks per set,
  - Each set contains two different memory blocks.
    - set 0 contains two blocks, one that is valid and holds the data A, B, C, . . . , and another that is not valid.



Set	Tag	Block 0 of set	Valid	Tag	Block 1 of set	Valid
0	00000000	Words A, B, C, . . .	1	-------------		0
1	11110101	Words L, M, N, . . .	1	-------------		0
2	-------------		0	10111011	P, Q, R, . . .	1
3	-------------		0	11111100	T, U, V, . . .	1

Set Associative cache

- In set associative cache mapping, a memory reference is divided into three fields:
  - Tag, set, and word.
- As with direct-mapped cache,
  - The word field chooses the word within the cache block, and
  - The tag field uniquely identifies the memory address.
- The set field determines the set to which the memory block maps.



Virtual Memory

- Cache memory:
  - Enhances performance by providing faster memory access speed.
- Virtual memory:
  - Enhances performance by providing greater memory capacity, without the expense of adding main memory.
- Instead, a portion of a disk drive serves as an extension of main memory.

Virtual Memory Terminologies

- **Paging:**
  - One of the memory-management schemes by which a computer can store and retrieve data from hard disk for use in main memory
- The operating system retrieves data from hard disk in same-size blocks called *pages*
  - The advantage of paging:
    - It allows the physical address space of a process to be noncontiguous
- **Page Fault**
  - when a program tries to access pages that are not currently mapped to physical memory. This is known as a page fault.

Virtual Memory Terminologies

- Page Frame:
  - If a system uses paging, virtual memory partitions main memory into individually managed *page frames:*
    - That are written *(or paged)* to disk when they are not immediately needed.
- A *physical address:*
  - The actual memory address of physical memory.

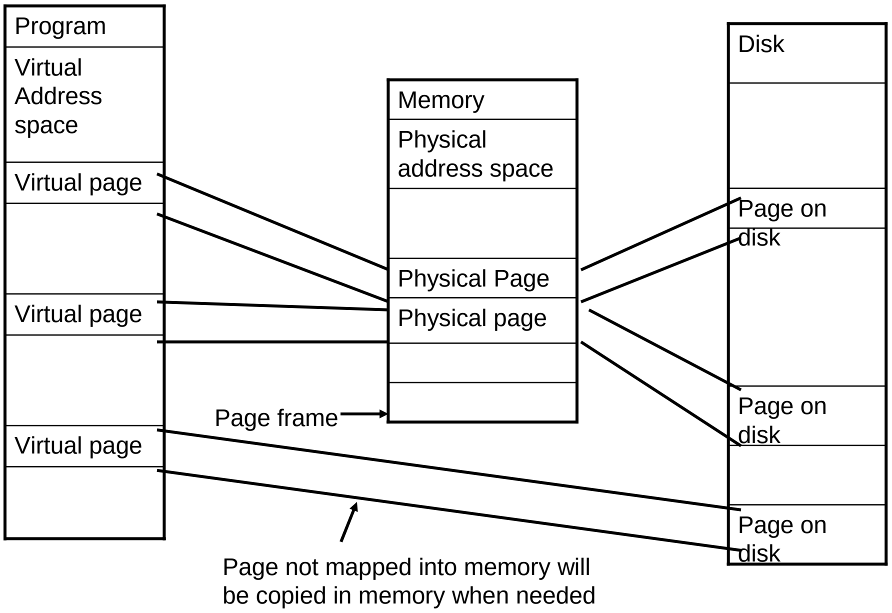
Virtual Memory- Terminologies

- Programs create *virtual addresses:*
  - That are *mapped* to physical addresses by the memory manager.
- Page Table
  - Information concerning the location of each page, whether on disk or in memory, is maintained in a data structure called a page table.
    - There is one page table for each active process.

Virtual Memory

- Virtual and physical address are used to describe address in virtual and physical address space
  - Virtual address space is divided in pages – hard disk
    - Virtual page
  - Some of the pages are copied into memory – **page frame**
    - Physical page

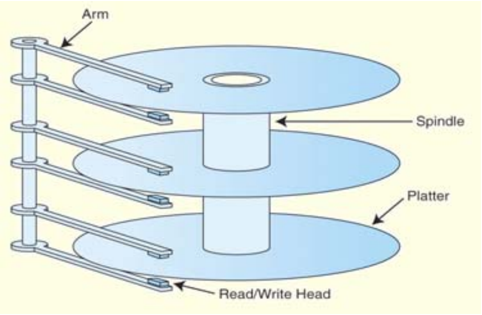
Virtual memory



Magnetic Disk Technology

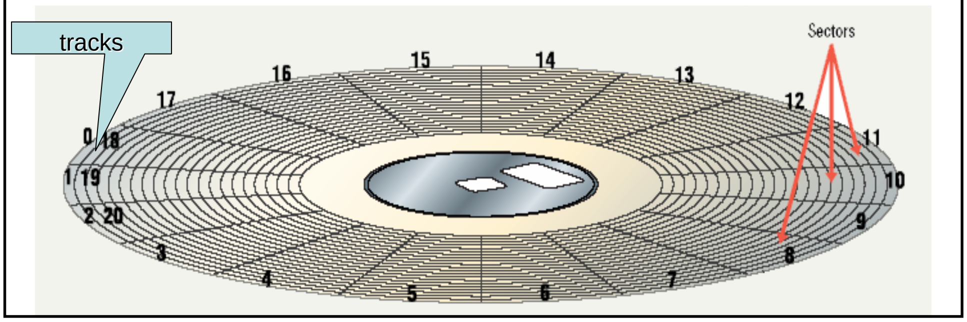
- Hard disk platters are mounted on spindles.
- Read/write heads are mounted on a comb that swings radially to read the disk.
  - Heads never touch the platter, they maintain a safe distance
    - If head touches the surface of the disk, the disk will become unusable, condition is known as head crash
  - Once the power is off, heads retreat to safe place known as parking of heads

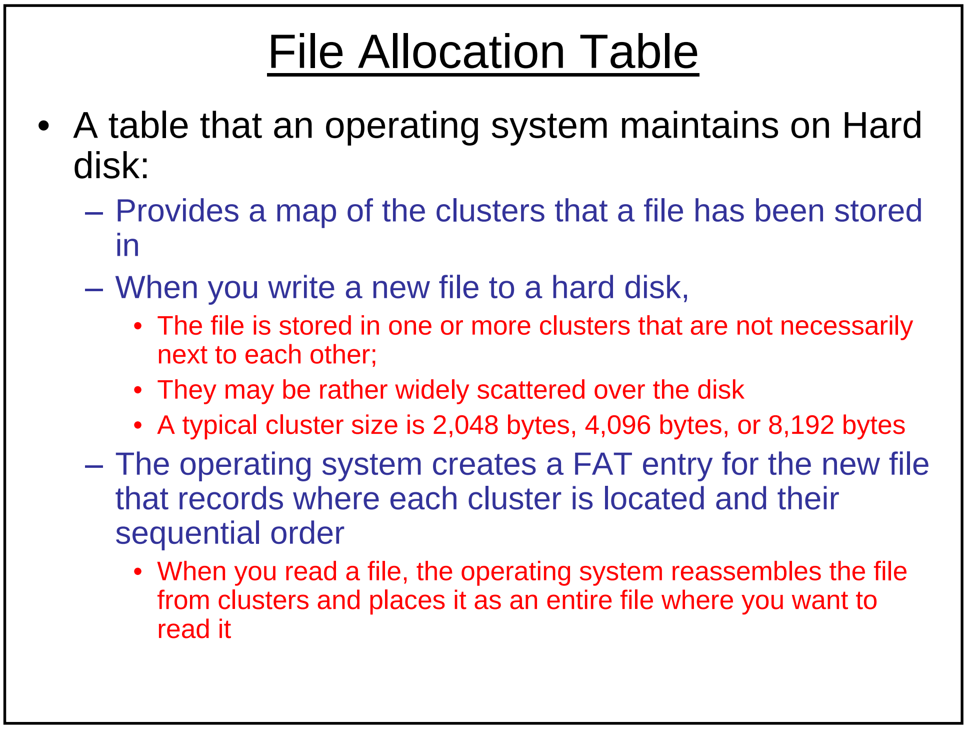
- The rotating disk forms a logical cylinder beneath the read/write heads.
- Data blocks are addressed by their cylinder, surface, and sector.



Magnetic Disk Technology

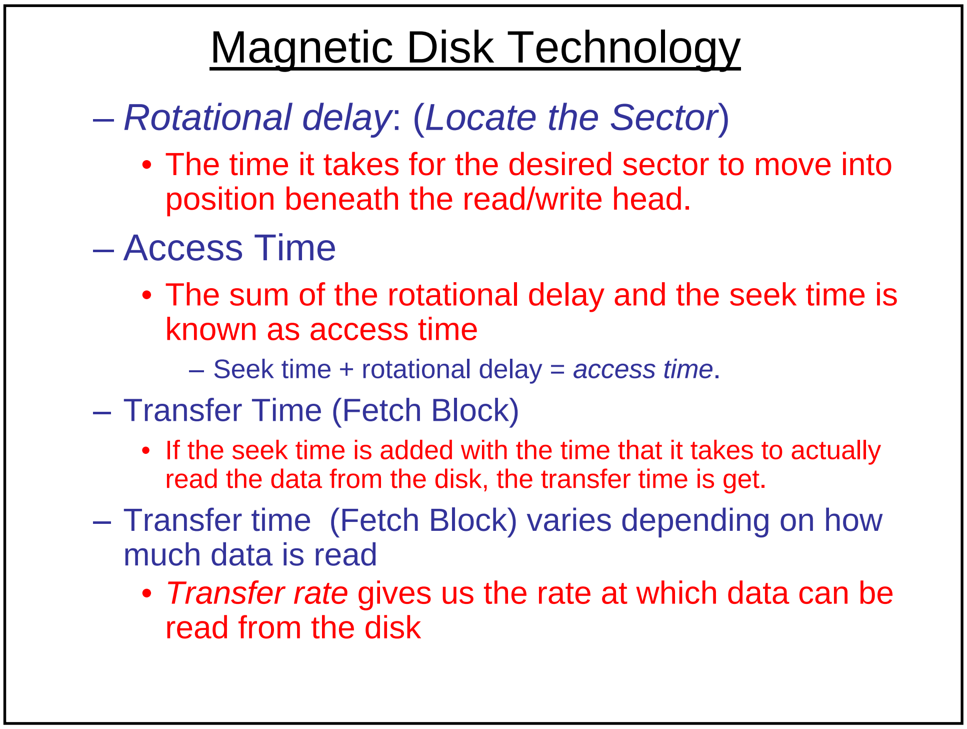
- Each unit of storage, Sector, has a unique address that can be accessed independently
  - blocks of data can be accessed according to their location on the disk.
- Sectors are division of concentric rings- tracks,
  - Every track contains exactly same sectors
  - Each sector contains same number of bytes





File Allocation Table

- A table that an operating system maintains on Hard disk:
  - Provides a map of the clusters that a file has been stored in
  - When you write a new file to a hard disk,
    - The file is stored in one or more clusters that are not necessarily next to each other;
    - They may be rather widely scattered over the disk
    - A typical cluster size is 2,048 bytes, 4,096 bytes, or 8,192 bytes
  - The operating system creates a FAT entry for the new file that records where each cluster is located and their sequential order
    - When you read a file, the operating system reassembles the file from clusters and places it as an entire file where you want to read it



Magnetic Disk Technology

- *Rotational delay*: (*Locate the Sector*)
  - The time it takes for the desired sector to move into position beneath the read/write head.
- Access Time
  - The sum of the rotational delay and the seek time is known as access time
    - Seek time + rotational delay = *access time*.
- Transfer Time (Fetch Block)
  - If the seek time is added with the time that it takes to actually read the data from the disk, the transfer time is get.
- Transfer time (Fetch Block) varies depending on how much data is read
  - *Transfer rate* gives us the rate at which data can be read from the disk

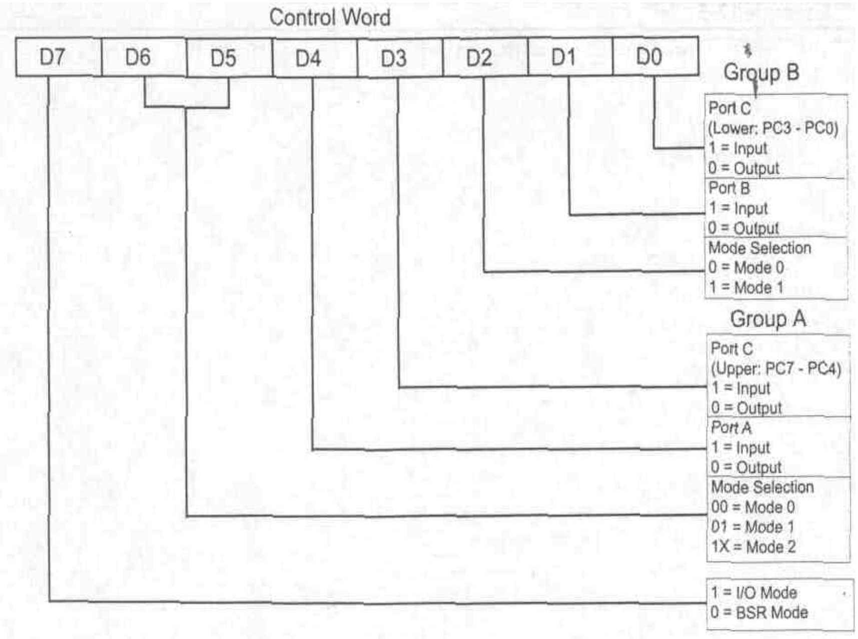
8255 Programmable Peripheral Interfacing

- one of the most widely used I/O chips.
  - It can be used for both memory-mapped and peripheral I/O
  - Provide a flexible parallel interface including, single bit, 4-bit and byte-wide input and output port
    - All these features configuration through software
- The 8255 is a 40-pin DIP chip
  - It has three separately accessible ports
    - Port A, Port B and Port C
    - The source and destination registers within 8255A is selected by a 2 bit register select code applied by MP on A0 and A1 lines of 8255A
      - Port A → A1A0 → 00
      - Port B → A1A0 → 01
      - Port C → A1A0 → 10

Mode selection of the 8255A

- While ports A, B, and C are used for I/O data, it is the control register that must be programmed to select the operation mode of the three ports A, B, and C
- The ports of the 8255 can be programmed in any of the following modes:
  - Mode 0:
    - Port A, B and C can be programmed as simple input and output without handshake
  - Mode 1:
    - Port A and B can be programmed as input out with single hand shake
  - Mode 2:
    - Port A can be used as bidirectional input output port

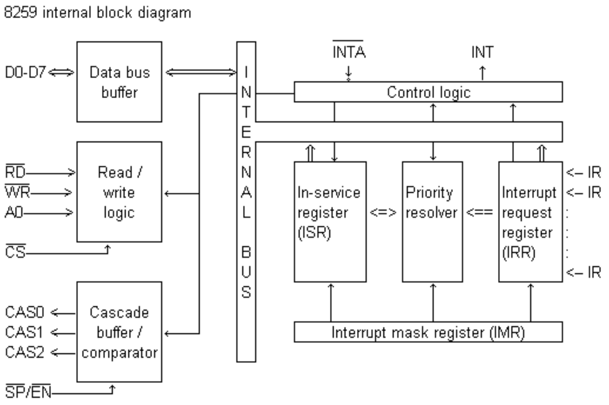
Control Word Format (I/O Mode)



Interrupt handling with 8259A

- 8259A accepts interrupts
- 8259A determines priority
- 8259A signals 8086 (raises INTR line)
- CPU Acknowledges
- 8259A puts correct vector on data bus
- CPU processes interrupt
  - Each interrupt line has a priority
  - Higher priority lines can interrupt lower priority lines
- 8259 can service up to eight hardware devices
  - Interrupts are received on IRQ0 through IRQ7

8259 Block Diagram



$\overline{\text{CS}}$	1		28	Vcc
$\overline{\text{WR}}$	2		27	A0
$\overline{\text{RD}}$	3		26	$\overline{\text{INTA}}$
D7	4		25	IR7
D6	5		24	IR6
D5	6		23	IR5
D4	7	8259	22	IR4
D3	8	PIC	21	IR3
D2	9		20	IR2
D1	10		19	IR1
D0	11		18	IR0
CAS0	12		17	INT
CAS1	13		16	$\overline{\text{SP}}/\overline{\text{EN}}$
gnd	14		15	CAS2

8259A working

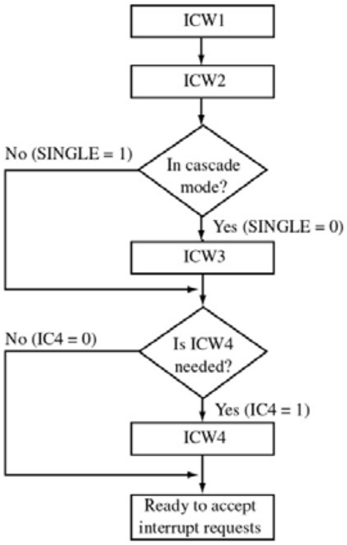
- If 8086 interrupt flag is set and INTR input receives a high signal, then 8086 will:-
  - Send out two interrupt ack pulses on its INTA pin to the INTA pin of 8259A
  - The INTA pulses tells the 8259A to send the desired interrupt type on data bus
  - Multiply the interrupt type by 4 to get the address
  - Push the flag on the stack
  - Clear IF and TF
  - Push the return address on the stack
  - Get the starting address of interrupt procedure
  - Execute the interrupt service procedure

Initializing and programming 8259A

- 8259 is programmed by initialization and Operation command word
  - Initialization command word (ICW) are programmed before 8259 is able to function
    - ICW dictates the basic mode of operation
  - Operation Command word (OCW) are used during the normal course of operation, to make 8259 function properly

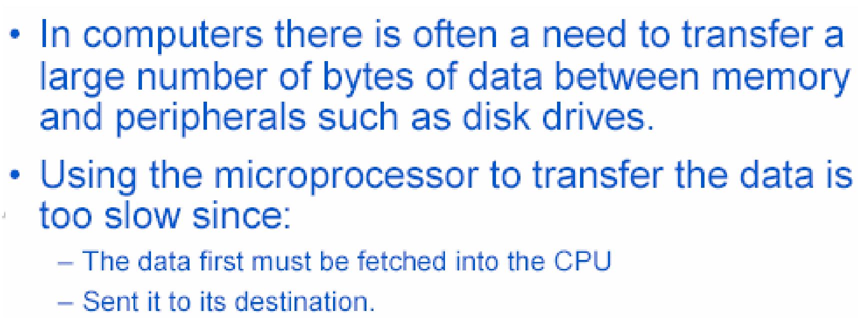
ICWs

- There are four ICWs for 8259A
- When 8259 is first powered
  - ICW1 and ICW2 are sent
  - ICW3 is only needed when ICW1 is programmed for cascade operation (SNLG =1)
  - ICW4 will be programmed for 8086 operation



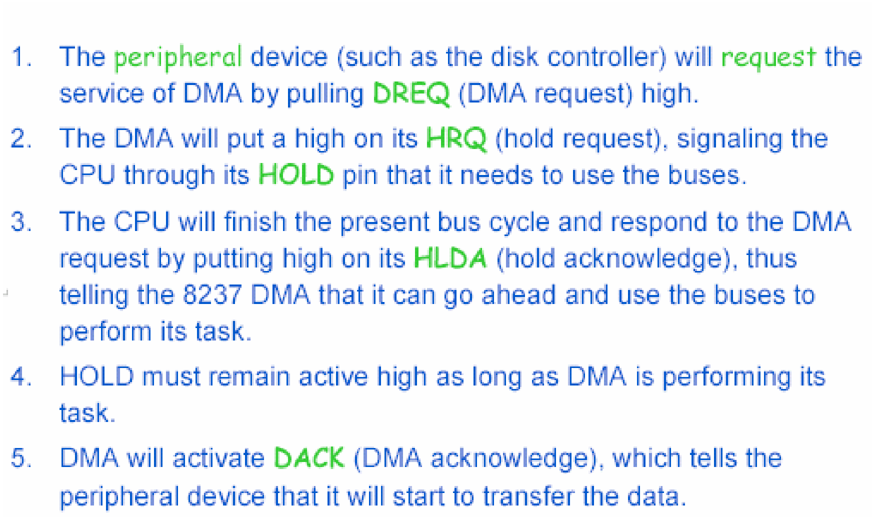
Direct Memory Access - DMA

- A system that can control the memory system without using the CPU
  - On a specified stimulus, the module will move data from one memory location or region to another memory location or region



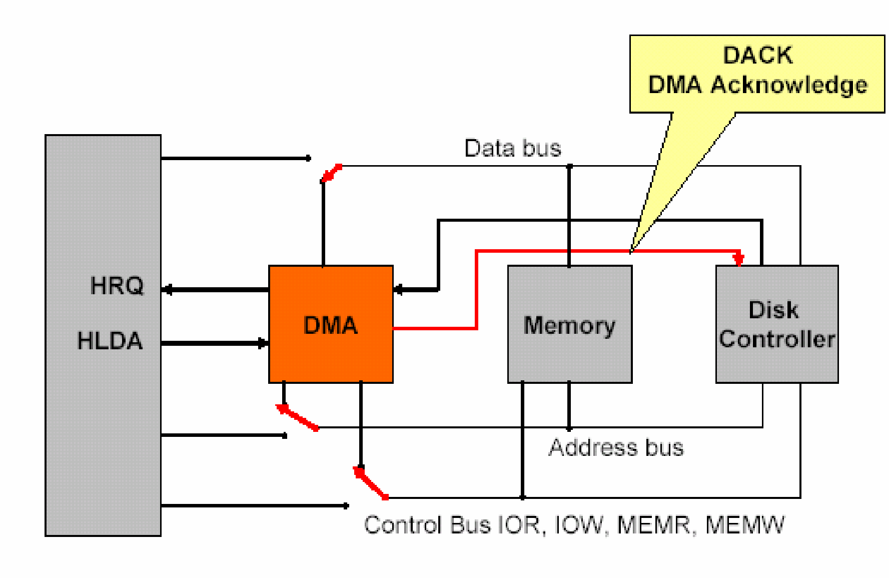
- In computers there is often a need to transfer a large number of bytes of data between memory and peripherals such as disk drives.
- Using the microprocessor to transfer the data is too slow since:
  - The data first must be fetched into the CPU
  - Sent it to its destination.

Steps - DMA

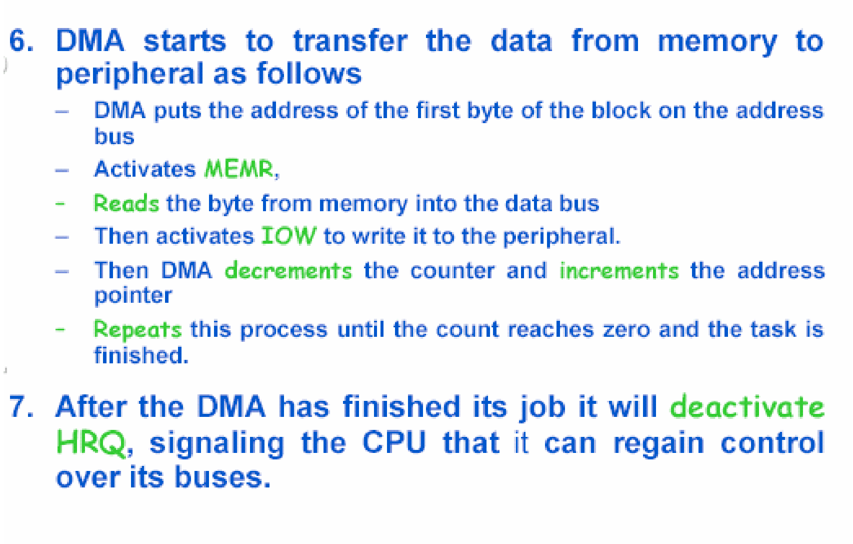


1. The peripheral device (such as the disk controller) will request the service of DMA by pulling DREQ (DMA request) high.
2. The DMA will put a high on its HRQ (hold request), signaling the CPU through its HOLD pin that it needs to use the buses.
3. The CPU will finish the present bus cycle and respond to the DMA request by putting high on its HLDA (hold acknowledge), thus telling the 8237 DMA that it can go ahead and use the buses to perform its task.
4. HOLD must remain active high as long as DMA is performing its task.
5. DMA will activate DACK (DMA acknowledge), which tells the peripheral device that it will start to transfer the data.

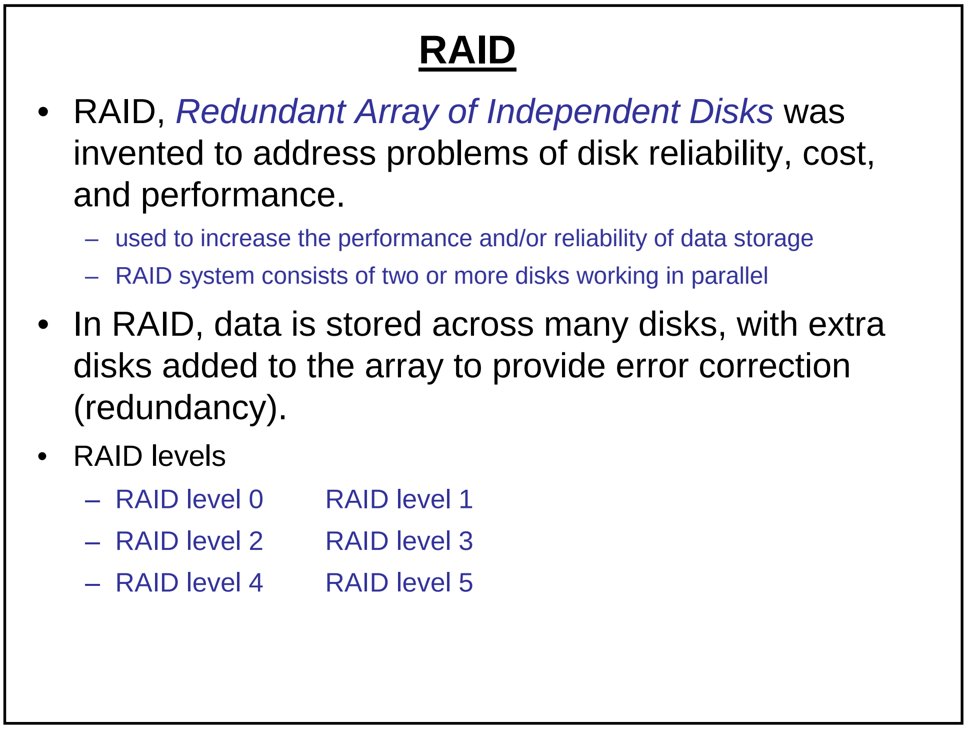
DMA controlled Buses



DMA Steps



6. DMA starts to transfer the data from memory to peripheral as follows
   - DMA puts the address of the first byte of the block on the address bus
   - Activates MEMR,
   - Reads the byte from memory into the data bus
   - Then activates IOW to write it to the peripheral.
   - Then DMA decrements the counter and increments the address pointer
   - Repeats this process until the count reaches zero and the task is finished.
7. After the DMA has finished its job it will deactivate HRQ, signaling the CPU that it can regain control over its buses.



RAID

- RAID, *Redundant Array of Independent Disks* was invented to address problems of disk reliability, cost, and performance.
  - used to increase the performance and/or reliability of data storage
  - RAID system consists of two or more disks working in parallel
- In RAID, data is stored across many disks, with extra disks added to the array to provide error correction (redundancy).
- RAID levels
  - RAID level 0   RAID level 1
  - RAID level 2   RAID level 3
  - RAID level 4   RAID level 5