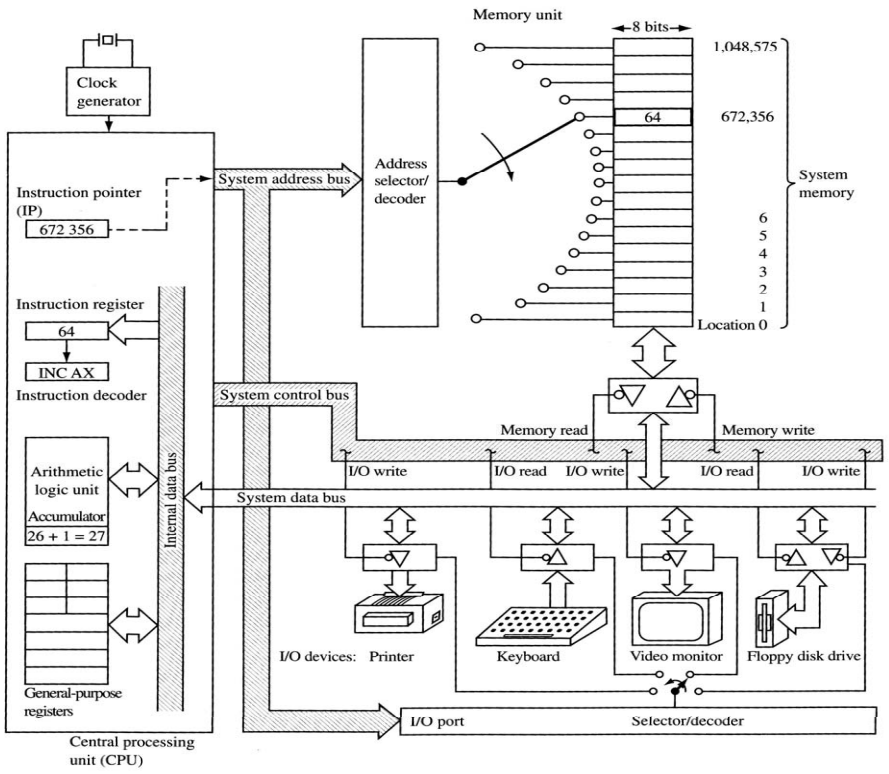
Week 1

Introduction to Microcomputers and Microprocessors, Computer Codes, Programming, and Operating Systems

Stored Program Concept



Stored Program Concept

- There are three major parts
  - The CPU (Central Processing Unit) which acts as the brain coordinating all activities within the computer
  - The memory unit where the program instructions and data are temporarily stored
  - The I/O (Input/Output) devices which allow the computer to input information for processing and then output the result
- Today the CPU circuitry has been reduced to *ICs* called the *microprocessor*, the entire computer with the three parts is called a *microcomputer*
- Several registers (e.g., flip-flops wired in series with each other)
  - Some are general purpose, the accumulator for example is reserved for performing complex mathematical operations like multiply and divide, and all I/O data has to go thru the accumulator
- The basic timing of the computer is controlled by a square wave oscillator or a *clock* generator circuit.
  - Synchronization
  - Determines how fast the program can be fetched from memory and executed
- Memory Read or Fetch Cycle
  - IP: Instruction Pointer

Stored Program Concept

- Memory unit consists of a large number of storage locations each with its own address
  - RAM (Random Access Memory) and its volatility
    - Typically each memory location is 8 bits wide (byte accessible memory)
  - ROM (Read Only Memory)
- The memory unit's address selector/decoder circuit examines the binary number on the address line and selects the proper memory location to be accessed.
  - In this example, CPU is reading from memory, it activates its MEMORY READ control signal
  - This causes the selected data byte in memory to be placed onto the data lines and routed to the instruction register in the CPU
- Once in the CPU, the instruction is decoded and executed
  - In this example, instruction has the decimal code 64 which for a 8086 microprocessor is decoded to be INC AX
  - The ALU (Arithmetic Logic Unit) is instructed to add 1 to the contents of the AX
- The cycle repeats itself

Stored Program Concept

- Memory unit consists of a large number of storage locations each with its own address. As a Prime Memory it can be observed as:
  - RAM (Random Access Memory) and its volatility. *Typically 8 bit wide*
  - ROM (Read Only Memory)

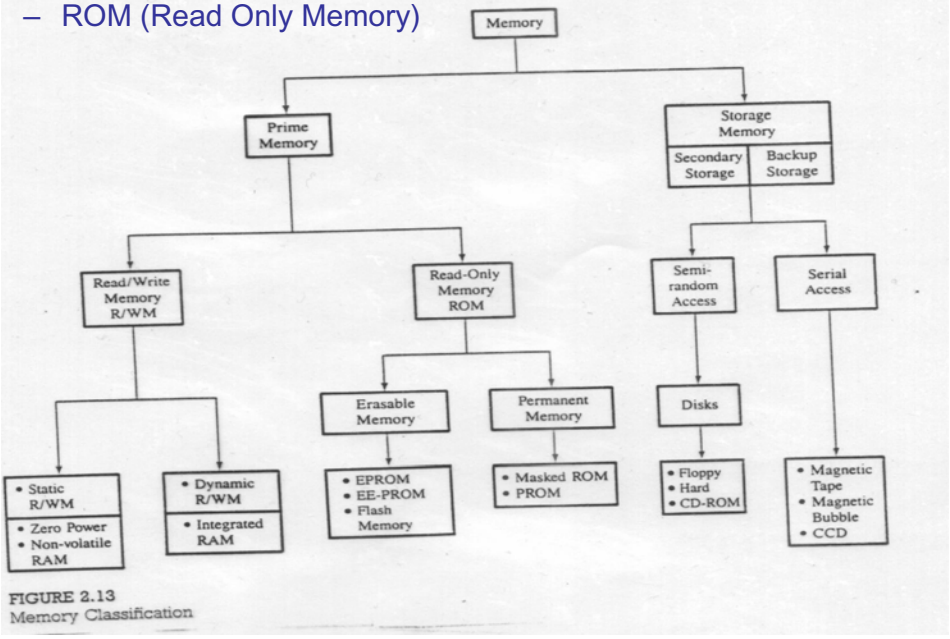


FIGURE 2.13
Memory Classification

Instruction Set

- The list of all recognizable instructions by the instruction decoder is called the instruction set
  - CISC (Complex Instruction Set Computers), e.g., 80x86 family has more than 3000 instructions
  - RISC (Reduced Instruction Set Computers) - A small number of very fast executing instructions
- Most microprocessor chips today are allowed to fetch and execute cycles to overlap
  - This is done by dividing the CPU into
    - EU (Execution Unit)
    - BIU (Bus Interface Unit)
  - BIU fetches instructions from the memory as quickly as possible and stores them in a queue, EU then fetches the instructions from the queue not from the memory
    - The total processing time is reduced
  - Modern microprocessors also use a *pipelined* execution unit which allows the decoding and execution of instructions to be overlapped.

RISC versus CISC

•**Advantages of complex instruction set machines (CISC)**

- •Less expensive due to the use of microcode; no need to hardwire a control unit
- •Upwardly compatible because a new computer would contain a superset of the instructions of the earlier computers
- •Fewer instructions could be used to implement a given task, allowing for more efficient use of memory
- •Simplified compiler, because the microprogram instruction sets could be written to match the constructs of high-level languages
- •More instructions can fit into the cache, since the instructions are not a fixed size

• **Disadvantages of CISC**

Although the CISC philosophy did much to improve computer performance, it still had its drawbacks:

- •Instruction sets and chip hardware became more complex with each generation of computers, since earlier generations of a processor family were contained as a subset in every new version
- •Different instructions take different amount of time to execute due to their variable-length
- •Many instructions are not used frequently; Approximately 20% of the available instructions are used in a typical program

RISC versus CISC

Advantages of RISC

Advantages of a reduced instruction set machine:

- •Faster
- •Simple hardware
- •Shorter design cycle due to simpler hardware

Disadvantages of RISC

Drawbacks of a reduced instruction set computer include

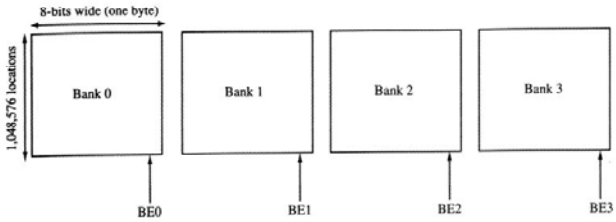
- •Programmer must pay close attention to instruction scheduling so that the processor does not spend a large amount of time waiting for an instruction to execute
- •Debugging can be difficult due to the instruction scheduling Require very fast memory systems to feed them instructions

•Nearly all modern microprocessors, including the Pentium, Power PC, Alpha and SPARC microprocessors are superscalar

Three Bus System Architecture

- A collection of electronic signals all dedicated to particular task is called a *bus*
  - *address bus*
  - *data bus*
  - *control bus*
- Data Bus
  - The width of the data bus determines how much data the processor can read or write in one memory or I/O cycle
  - 8-bit microprocessor has an 8-bit data bus
  - 80386SX 32-bit internal data bus, 16-bit external data bus
  - 80386 32-bit internal and external data buses
- How can a 64-bit (or 16 bit) microprocessor access an 8-bit memory?
  - The trick is to divide the memory into banks
  - 64-bit Pentium requires eight banks of memory with each bank set up to be one-byte wide
  - Bank enable signals are then output by the microprocessor to specify which bank to access

Address Bus



Total amount of memory is 4Mbytes

- Address Bus
  - The address bus is used to identify the memory location or I/O device (also called port) the processor intends to communicate with
  - 20 bits for the 8086 and 8088
  - 32 bits for the 80386/80486 and the Pentium
  - 36 bits for the Pentium II and III
  - The total number of memory locations addressable by a given CPU is always equal to 2^x where x is the number of address bits, regardless of the data bus.
- 8086 has a 20-bit address bus and therefore addresses all combinations of addresses from all 0s to all 1s. This corresponds to 2^{20} addresses or 1M (1 Meg) addresses or memory locations.
- Pentium: 4Gbyte main memory

Control Bus

- How can we tell the address is a memory address or an I/O port address
  - Memory Read
  - Memory Write
  - I/O Read
  - I/O Write
- When Memory Read or I/O Read are active, data is *input* to the processor.
- When Memory Write or I/O Write are active, data is *output* from the processor.
- The control bus signals are defined from the processor's point of view.
- Control and address lines are output lines only but the data bus is bidirectional

Some Important Terminology

- Bit is a binary digit that can have the value 0 or 1
- A byte is defined as 8 bits
- A nibble is half a byte
- A word is two bytes
- A double word is four bytes
- A kilobyte is 2^10 bytes (1024 bytes), The abbreviation K is most often used
  - Example: A floppy disk holding 356Kbytes of data
- A megabyte or meg is 2^20 bytes, it is exactly 1,048,576 bytes
- A gigabyte is 2^30 bytes

Internal Working of Computers - Example

- Assume that an imaginary CPU has registers called A,B,C, and D.
- It has an 8-bit data bus and a 16-bit address bus.
- Therefore the CPU can access memory from addresses 0000h to FFFFh for a total of 2^16 locations
- The action to be performed by the CPU is to put a hexadecimal value 21 into register A, and add to register A values 42h and 12h.
- Assume that the code for the CPU to move a value to register A is 1011 0000b (B0h) and the code for adding a value to register A is 0000 0100b (04h)

Action	Code	Data
Move 21h to A	B0h	21h
Add 42h to A	04h	42h
Add 12h to A	04h	12h

Example Continued

Memory Address	Content of memory
1400h	B0h
1401h	21h
1402h	04h
1403h	42h
1404h	04h
1405h	12h
1406h	F4h (the code for halt)

- Assume program is stored at memory locations starting at 1400h

Internal Working Of Computers

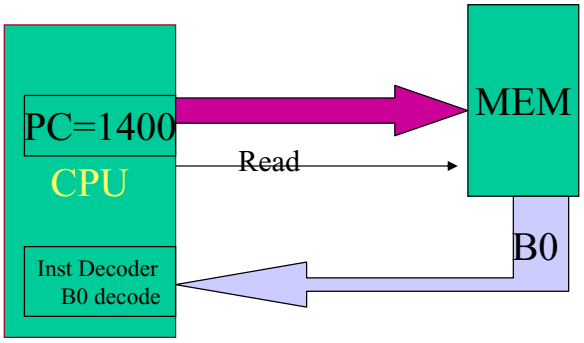
ACTION	Code	Data
Move value 21 into register A	B0H	21H
Add value 42H to register A	04H	42H
Add value 12H to register A	04H	12H

Memory Address	Contents of memory address
1400	(B0) the code for move to A
1401	(21) the value for A
1402	(04) the code for adding a value to A
1403	(42) the value to be added
1404	(04) the code for adding a value to A
1405	(12) the value to be added
1406	(F4) the code for halt

Internal Working Of Computers

1- the CPU program counter can have any value between 0000 → FFFF H. This one is set to start with 1400

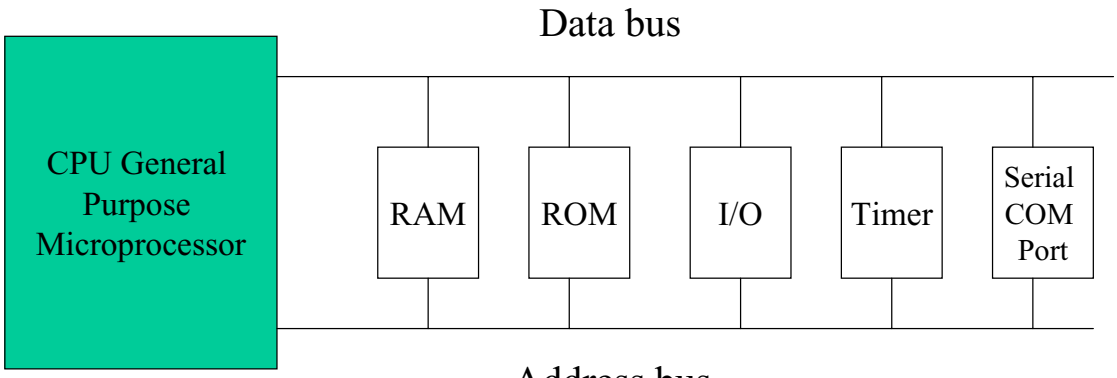
2- the CPU puts out 1400. The memory circuitry finds the location. Activates the read signal, indicating the memory location 1400. B0 is put on the bus and brought to the CPU



3- B0 is decoded internally it now knows it needs to fetch the next byte!. It brings 21h from 1401. The program counter automatically increments itself to the next location to fetch the next data/instruction.

General Purpose Microprocessors

Microprocessors lead to versatile products



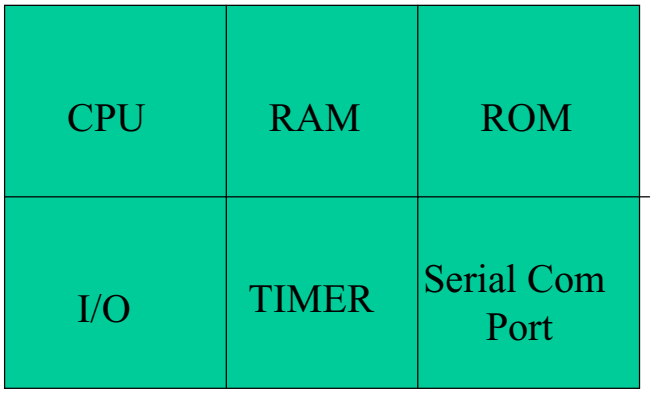
These general microprocessors contain no RAM, ROM, or I/O ports on the chip itself

Ex. Intel's x86 family (8088, 8086, 80386, 80386, 80486, Pentium)

Motorola's 680x0 family (68000, 68010, 68020, etc)

Microcontrollers

Microcontroller



A microcontroller has a CPU in addition to a fixed amount of RAM, ROM, I/O ports on one single chip; this makes them ideal for applications in which cost and space are critical

Example: a TV remote control does not need the computing power of a 486

Embedded Systems

- An embedded system uses a microcontroller or a microprocessor to do one task and one task only
  - Example: toys, garage door openers, answering machines, ABS, keyless entry, etc.
  - Inside every mouse, there is a microcontroller that performs the task of finding the mouse position and sends it to the PC
- Although microcontrollers are the preferred choice for embedded systems, there are times that the microcontroller is inadequate for the task
- Intel, Motorola, AMD, Cyrix have also targeted the embedded market with their general purpose microprocessors
- For example, Power PC microprocessors (IBM Motorola joint venture) are used in PCs and routers/switches today
- Microcontrollers differ in terms of their RAM,ROM, I/O sizes and type.
  - ROM: One time-programmable, UV-ROM, flash memory

Types of Microcomputers

- *Microprocessor:* Processor on a chip
- In 1982, IBM began selling the idea of a *personal computer* . It featured a system board designed around the Intel 8088 8-bit microprocessor, 16 K memory and 5 expansion slots.
  - This last feature was the most significant one as it opened the door for 3rd party vendors to supply video, printer, modem, disk drive, and RS 232 serial adapter cards.
  - Generic PC: A computer with interchangable components manufactured by a variety of companies
- *Microcontroller* is an entire computer on a chip, a microprocessor with on-chip memory and I/O.
  - These parts are designed into (embedded within) a product and run a program which never changes
  - Home appliances, modern automobiles, heat, air-conditioning control, navigation systems
  - Intel's MCS-51 family, for example, is based on an 8-bit microprocessor, but features up to 32K bytes of on-board ROM, 32 individually programmable digital input/output lines, a serial communications channel.
  - PIC Microcontrollers will be studied in this course

Evolution of Intel Microprocessors

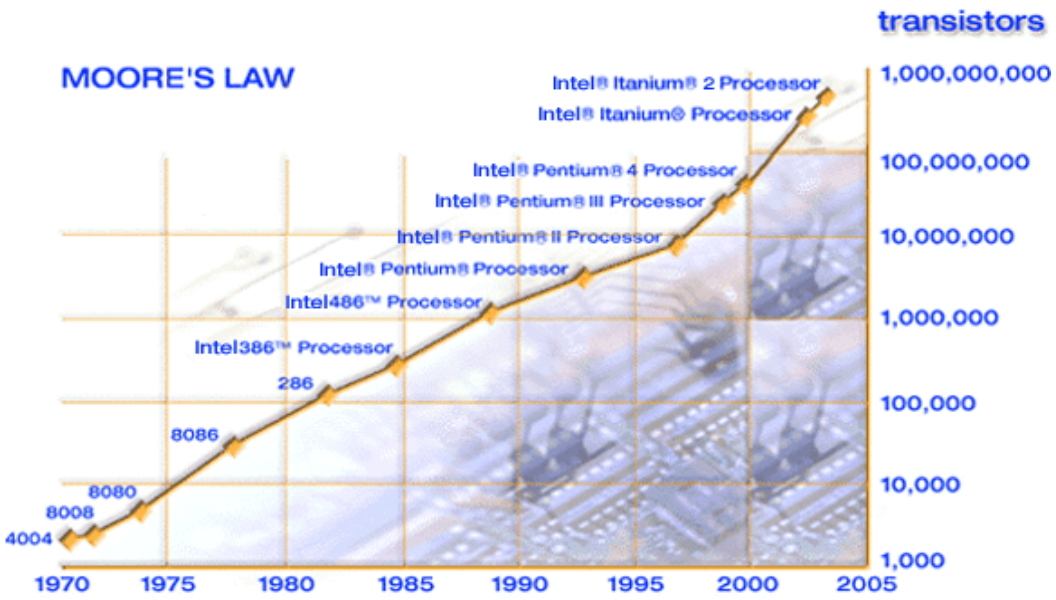
Processor	Codename	Year Introduced	Transistors	Minimum Feature Size (microns)	Package	Socket or Slot	Core/Bus Frequency (Max)[1]	External Data Bus Width	Internal Register Widths	Address Bus Width	NDP[2]	L1 Cache	L2 Cac
4004		1971	2,250	10.0	16 pin DIP		.108 MHz	4	8	12	none	none	no
8008		1972	3,500	10.0	18 pin DIP		.200 MHz	8	8	14	none	none	no
8080		1974	6,000	6.0	40 pin DIP		3 MHz	8	8	16	none	none	no
8085[3]		1976	6,000	6.0	40 pin DIP		6 MHz	8	8	16	none	none	no
8086		1978	29,000	3.0	40 pin DIP		10 MHz	16	16	20	external	none	no
8088		1979	29,000	3.0	40 pin DIP		10 MHz	8	16	20	external	none	no
80286		1982	134,000	1.5	68 pin PLCC or PGA[4]		12.5 MHz	16	16	24	external	none	no
80386DX		1985	275,000	1.0	132 pin PGA or QFP[5]		33 MHz	32	32	32	external	none	exter
80386SX		1988	275,000	1.0	100 pin PQFP[7]		33 MHz	16	32	24	external	none	exte
80486DX		1989	1.2 million	0.8	168 pin PGA	Socket 3	50 MHz	32	32	32	on-chip	8 KB	exte
80486SX		1991	1.185 million	1.0	196 lead PQFP or 168 pin PGA	Socket 3	33 MHz	32	32	32	none	8 KB	exte
80486DX2		1992	1.2 million	0.6	168 pin PGA	Socket 3	66/33 MHz	32	32	32	on-chip	8 KB	exte
80486DX4		1994	1.2 million	0.6	168 pin PGA	Socket 3	100/ 33 MHz	32	32	32	on-chip	8 KB	exte
Pentium Classic	P5	1993	3.1 million	0.8	273 pin PGA	Socket 4, 5	66 MHz	64	32	32	on-chip	8/8 KB C/D[8]	exte
Pentium Classic	P54	1994	3.3 million	0.35, 0.5	296 pin PGA	Socket 7	200/66 MHz	64	32	32	on-chip	8/8 KB C/D	exte
Pentium MMX	P55	1997	4.5 million	0.25, 0.28	296 pin PGA	Socket 7	300/66 MHz	64	32	32	on-chip	16/16 KB C/D	exte
Pentium Pro	P6	1995	5.5 million[9]	0.35, 0.5	387 pin dual cavity PGA or PPGA[10]	Socket 8	200/66 MHz	64	32	36	on-chip	8/8 KB C/D	256 1M

Evolution of Intel Microprocessors

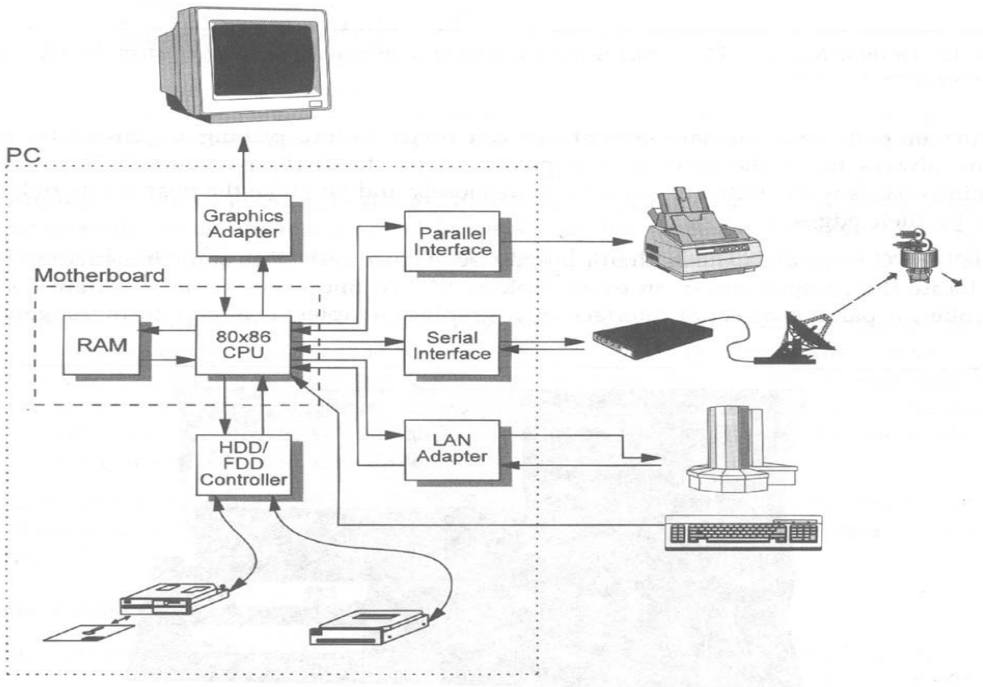
Pentium II	(Klamath) Deschutes[12]	(1997) 1998	7.5 million	(0.28), (0.25)	242 contact SEC cartridge	Slot 1	(233/66 MHz) 450/100 MHz	64	32	36	on-chip	16/16 KB C/D	512 KB[13]
Celeron	(Covington) Mendocino[14]	1998	(7.5 million) 19 million[15]	0.25	(242 contact SEP cartridge)	Slot 1	(300/66 MHz)						
					370 pin PPGA	Socket 370	466/66 MHz	64	32	36	on-chip	16/16 KB C/D	(external) 128 KB[16]
Pentium III	Katmai	1999	9.5 million	0.25	242 contact SEC cartridge	Slot 1	550/100 MHz	64	32	36	on-chip	16/16 KB C/D	512 KB[17]
					330 contact SEC cartridge	Slot 2							
	Coppermine	1999		0.18	370 pin PGA	Socket 370	733/133 MHz						256 KB[18]
Itanium[19]	Merced	2000		0.18			6XX/133 MHz	128	64	64	on-chip		256 KB[20]

[1]It is likely that higher frequency versions of the newer processors will be offered in the future.
[2]Numeric data processor (also called coprocessor or floating point unit).
[3]Improved 8080 with three new instructions to enable/disable three added interrupt pins. Simplified hardware with single +5 V power supply and on-board clock generator.
[4]Plastic leaded chip carrier or pin grid array.
[5]Quad flat package (QFP).
[6]Some 386 computers (and nearly all later processors) incorporated external L2 caches.
[7]Plastic quad flat package.
[8]Separate code and data caches are supplied
[9]On-board 256 KB L2 cache (separate silicon die) has 15.5 million transistors (31 million for 512 KB cache). 1 MB cache has two separate 512 KB die.
[10]Plastic pin grid array
[11]Separate die in package. Cache operates at core speed.
[12]Specifications for Klamath processor are shown in parentheses.
[13]Separate die in SEC package. Cache operates at one-half core speed.
[14]Specifications for the Covington processor are shown in parentheses. The Mendocino processor is also called Celeron A.
[15]Includes integrated 128 KB L2 cache.
[16]128 KB cache is on the same die with the processor and operates at the core frequency of the processor.
[17]Separate die operating at 0.5 times core speed (slot 1) or integrated with the processor operating at core speed (slot 2).
[18]Integrated with the processor and operating at core speed. Includes 256-bit (vs. 64 bit on previous chips) processor–cache data bus.
[19]Specifications for this processor have not yet been finalized by Intel.
[20]Integrated with the processor die and operating at full core speed.

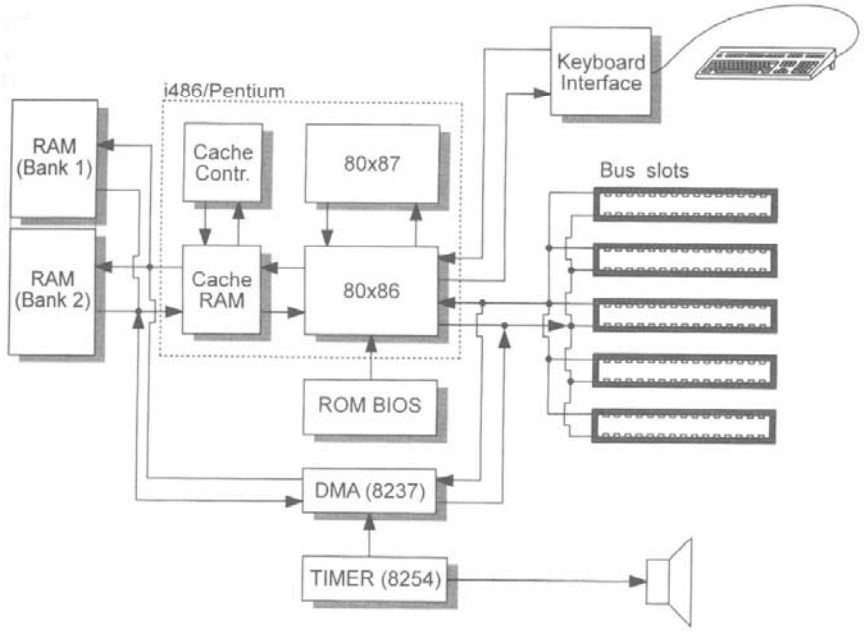
Change in Microprocessors



Data Flow Inside the PC



Motherboard



Motherboard

- The motherboard is the heart of the PC on which all components that are absolutely necessary are located.
- Motherboard and several *slots* into which the circuit boards of the graphics adapter and the interfaces are located.
- 80x86 is the central unit of the board.
  - It executes all data processing, that is, numbers are added, subtracted, multiplied, etc. logic operations with two operations with two items are executed (logical AND, XOR).
- For extensive mathematical operations (like the tangent of a real number), a mathematical coprocessor is available. Intel calls the processor as 80x87.
    - Other companies also supply coprocessors (Weitex, Cyrix).
    - May be 100 times faster than normal processors.
    - Usually PCs are not equipped with a coprocessor when shipped, only with a socket for it.
    - The 486DX and its successors Pentium and Pentium Pro already implement an FPU on-chip so that a coprocessor is obsolete.

Motherboard

- Another important motherboard component is the main memory or RAM.
  - Usually, the main memory is divided into several *banks;* each bank has to be fully equipped with memory chips.
  - AT-386s main memory size is typically 4 Mbytes, fully equipped Pentium PCs have at least 32 Mbytes of RAM.
  - CPU stores data and intermediate results, as well as programs, in its memory and read them later.
  - *Address:* house number of the data unit requested
  - Transferring the address to the memory is carried out by an *address bus* and the transfer of data by a *data bus*.
  - *Bus* means a number of lines through which data and signals are transferred.
  - Address bus is
    - 20 for PC XT/AT
    - 24 for AT
    - 32 for 386, 486, and Pentium
    - *Access time*:Time period between the CPU's command to the memory that data should be read and this data being transferred to the processor.
      - Modern memory chips have an access time of about 60-70 ns; access time is one of the most important restrictions on the operational speed of a PC.

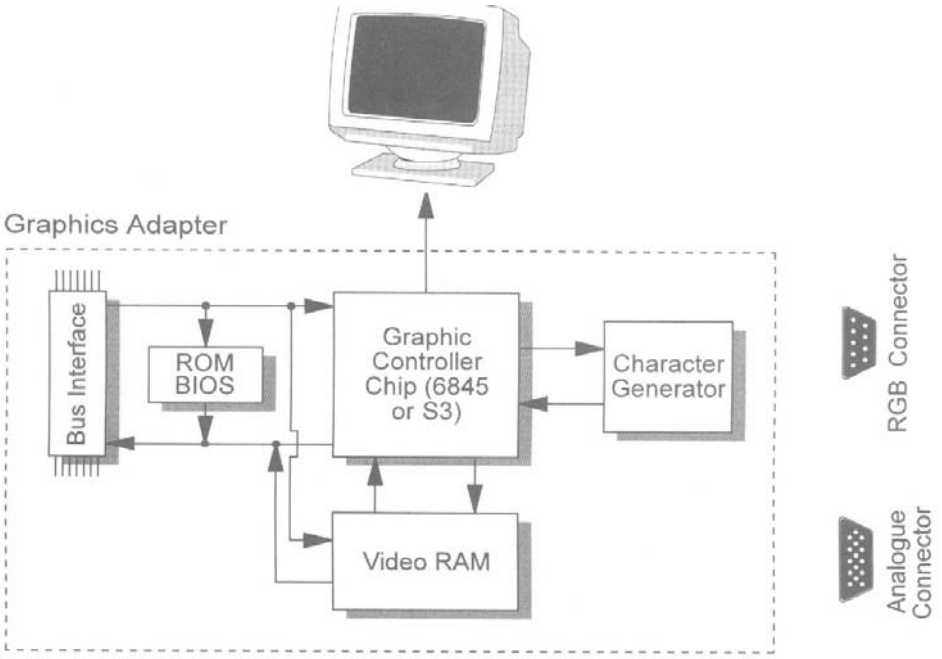
Motherboard

- Fast-clocked computers above 150 Mhz have a *cache* or *cache memory* which is significantly smaller than the main memory but much faster (access time of 10-20 ns).
  - Cache holds data that is frequently accessed by the CPU.
  - Uses a cache controller to check if the required data is in the cache.
  - On the new and powerful 80x86 processors, the processor, coprocessor, cache memory, and a cache controller are all integrated on a single chip to form the i486 or Pentium.
- Motherboard also includes *Read Only Memory (ROM)*
  - Located on this chip are the programs and data that the PC needs at power-up.
  - In the ROM, there are also various support routines for accessing the keyboard, graphics adapter, etc. known collectively as ROM-BIOS.
- To control the data transfer process, additional control signals are required: e.g., write-enable signal for which one bus line is reserved.
  - Data bus, address bus, and and all control lines are known as the *system bus*.
  - 62 contacts for the XT (XT's system bus) and 98 contacts for the AT
- Bus slots: memory expansion card may be inserted in one bus slot.

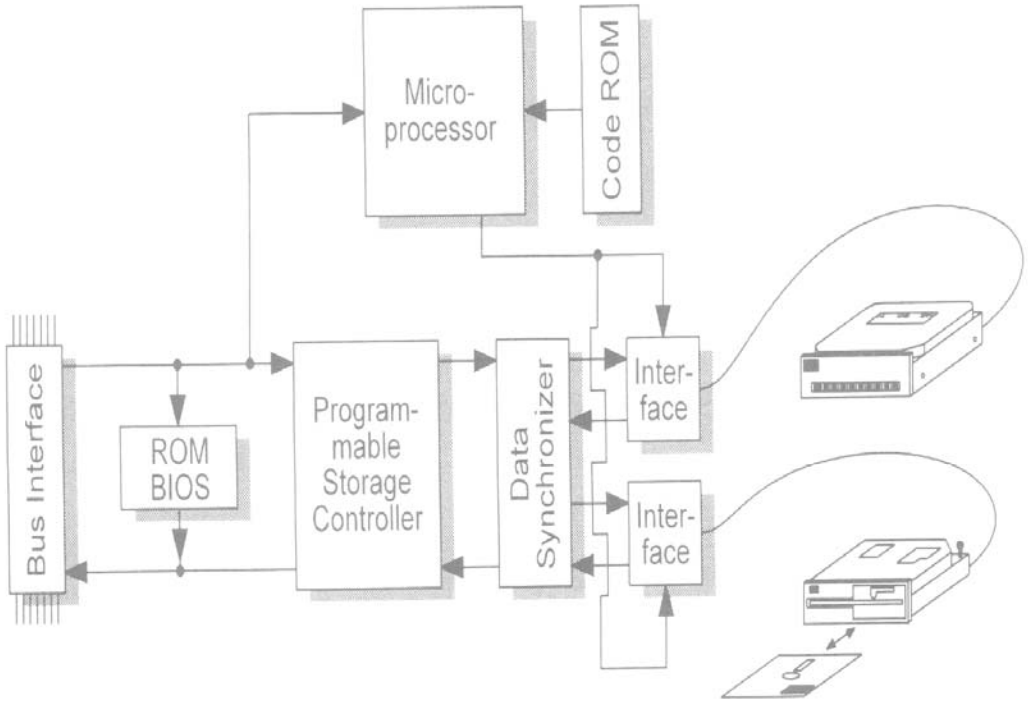
Motherboard

- Frequently, extensive amounts of data must be transferred from a hard or floppy disk into the main memory (word processor application for example). For this purpose, the motherboard has several chips optimized for data transfer within the computer - the *DMA chips (Direct Memory Access).*
  - The CPU is bypassed in this process.
- Timer chip for memory refresh (Dynamic RAM) and for supporting DOS routines time and date.

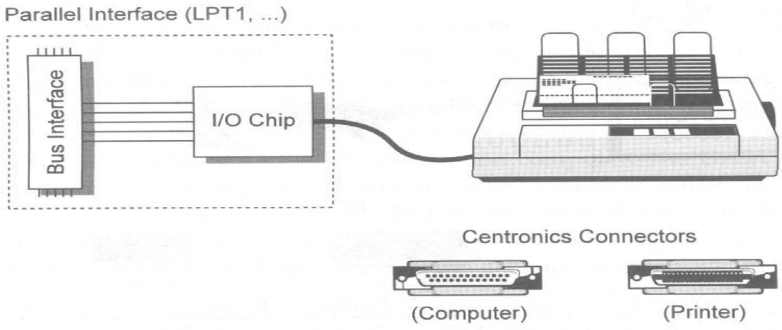
Graphics Adapters and Monitors



Drive Controllers, Floppy and Hard Disk Drives

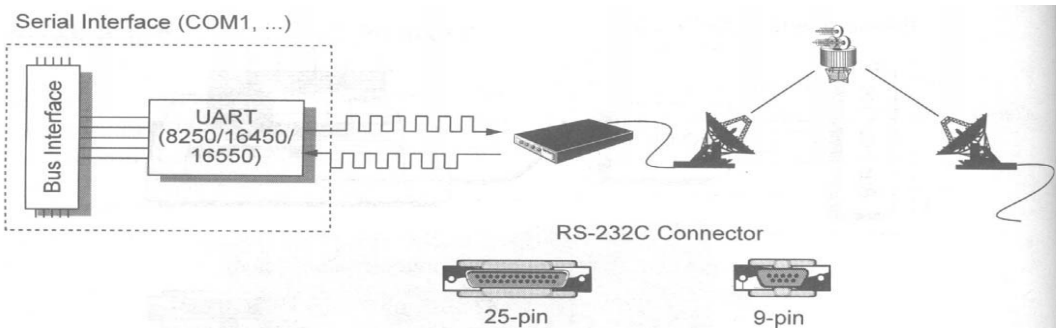


Parallel Interfaces and Printers



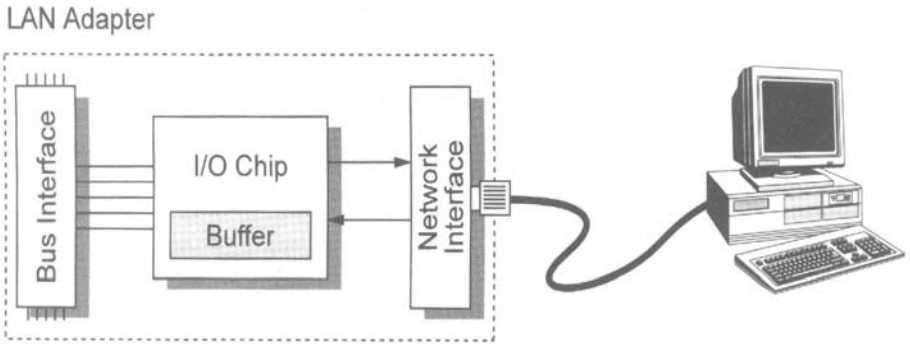
- I/O chip on the interface card accepts eight bits together and transfers them together (that is, in parallel) to the connected device (printer).
- Besides the data byte, control signals are also present to indicate whether the data has arrived.
- Up to 100 Kbytes of data can be transferred per second if the interface and the connected peripheral are correctly adapted.
- On the interface is a jack with 25 holes which supply signals according to the Centronix standard.
- The standard claims 36 holes, IBM uses 25 of them: de-facto standard
- The max distance between the computer and the printer is about 5m
- Data is exchanged via handshaking
- Usually the parallel interface only supplies data but does not receive any.
- New versions of I/O chips can receive data and it is thus possible to exchange data between computers via the parallel interface and a suitable software.

Serial Interfaces and Modems



- A PC usually has one or more serial interfaces; these are integrated on an interface adapter card together with a parallel interface.
- The central component is a so-called UART which transmits via single data line as opposed to eight as in the case for the parallel interface.
- Older PC/XTs have an 8250 chip, the AT has the more advanced 16450/16550 chip.
- UART adds additional bits; start, stop and parity bits.
- Much longer distances are possible (up to 100 m) but the transfer rate is lower.
- Serial interfaces conform to the RS232 standard which requires 25 contacts; only 14 at most are used.
- Used generally in modems but also in mouse, trackball, or joystick.

Network Adapters and LANs



- Networking is essential.
- A network adapter has two interfaces: one to the PC's CPU and a network interface for accessing the network.
- Network adapter can be inserted in any free slot.
- The network interface depends on the network used: Ethernet, token ring, or ATM
- 10/100/G Ethernet
- I/O chip converts the data into a form that is adapted for transmission via the network.

Binary and Hexadecimal Systems - Overview

- Conversion to decimal:
  - 110.101 b = ?
  - 6A.C h = ?
- Conversion from decimal
  - for a whole number: divide by the radix and save the remainder as the significant digits
  - 10 = ? B
  - 10 = ? $_8$
- Converting from a decimal fraction
  - multiply the decimal fraction by the radix
  - save the whole number part of the result
  - repeat above until fractional part of step 2 is 0
  - 0.125 = ? b
  - 0.046875 = ? h

  - 110.101 b = 6.625
  - 6A.C h = 106.75

  - 10 = 1010 b
  - 10 = 12 $_8$

  - 0.125 = 0.001 b
  - 0.046875 = 0.0C h

Two's Complement

- If the number is positive make no changes
- If the number is negative, complement all bits and add by 1
  - -6 => $\overline{0000\ 0110}$ + 1 = 1111 1001 + 1 = FAh
- 8 bit signed numbers
  - 0 to 7Fh (+127) are positive numbers
  - 80h (-128)to FFh (-1) are negative numbers
- Conversion of signed binary numbers to their decimal equivalent
  - $\overline{1101\ 0001}$
    - 1101 0001 + 1 = 0010 1110 + 1 = 0010 1111 = 2Fh => -47
  - 1000 1111 0101 1101
    - 0111 0000 1010 0010 + 1 = 0111 0000 1010 0011 = 70C3h => -28835
- Two's complement arithmetic
  - +14 - 20
  - 0000 1110 + $\overline{0001\ 0100}$ + 1 = FAh
- *Overflow:* Whenever two signed numbers are added or subtracted the possibility exist that the result may be too large for the number of bits allocated Ex: +64 +96 using 8-bit signed numbers

Two's complement

- -128 1000 0000b 80h
- -127 1000 0001b 81h
- -126 1000 0010b 82h

- -2 11111110b FEh
- -1 11111111b FFh
- 0 0000000b 00h
- 1 00000001b 01h

- +127 01111111b 7Fh

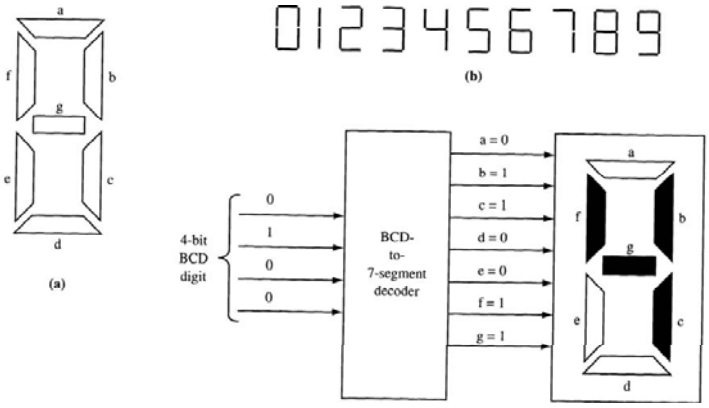
Numbers in the range
-2^n … 2^n –1
are represented by signed arithmetic

ASCII

- The standard for text
- In this code each letter of the alphabet, punctuation mark, and decimal number is assigned a unique 7-bit code number
- With 7 bits, 128 unique symbols can be coded
  - e.g., Uppercase A 41h
- Error detection codes
  - Parity
    - F has the ASCII code 46h or 100 0110
    - Even parity encoded F becomes 1100 0110 or C6h
- Which of the following are errored transmissions if even parity is used?
  - E1h = 1110 0001
  - 20h = 0010 0000
  - 72h = 0111 0010 *error*
- Parity method of error detection can only be used to detect odd numbers of errors
  - 72h => 77h *use more sophisticated checksums*

BCD

- BCD code provides a way for decimal numbers to be encoded in binary form that is easily converted back to decimal
  - 26 => 0010 0110 (BCD) => 11010 (unsigned binary)
  - 243 => 0010 0100 0011 (BCD) => 1111 0011 (unsigned binary)
- Used in seven segment displays



Computer Programming

- Machine Language vs Assembly Language
  - Machine language or object code is the only code a computer can execute but it is nearly impossible for a human to work with
  - E4 27 88 C3 E4 27 00 D8 E6 30 F4 the object code for adding two numbers input from the keyboard
- When programming a microprocessor, programmers often use assembly language
  - This involves 3-5 letter abbreviations for the instruction codes (mnemonics) rather than the binary or hex object codes

Address	Hex Object Code				Mnemonics		Comment
					Op-Code	Operand	
0100	E4	27			IN	AL,27H	Input first number from port 27H and store in AL
0102	88	C3			MOV	BL,AL	Save a copy of register AL in register BL
0104	E4	27			IN	AL,27H	Input second number to AL
0106	00	D8			ADD	AL,BL	Add contents of BL to AL and store the sum in AL
0107	E6	30			OUT	30H,AL	Output AL to port 30H
0109	F4				HLT		Halt the computer

Source code

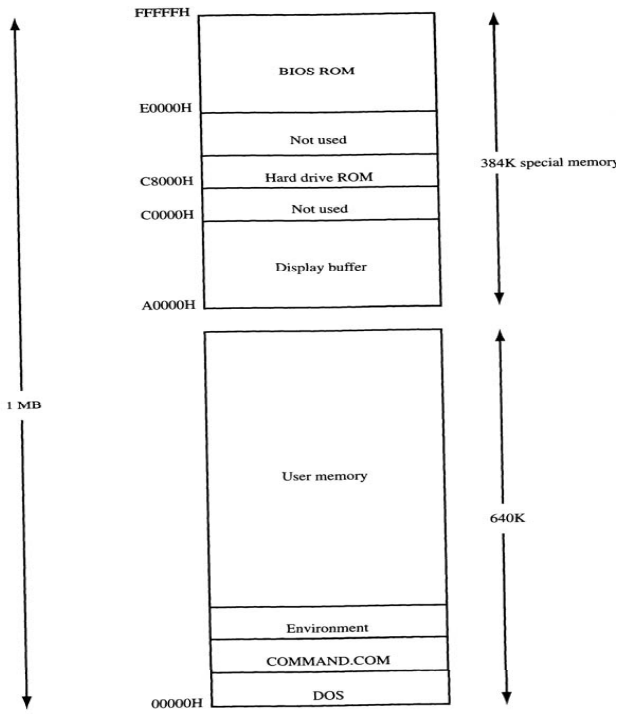
Edit, Assemble, Test, and Debug Cycle

- Using an *editor*, the source code of the program is created. This means selecting the appropriate instruction mnemonics to accomplish the task
- A compiler program which examines the source code file generated by the editor and determines the object code for each instruction in the program, is then run. In assembly language programming, this is called an *assembler*.
- The object code produced by the computer is loaded into the target computer's memory and is then *run.*
- *Debugging:* locating and fixing the source of error
- High-level programming Languages
  - Basic
  - C

Computer Operating Systems

- What happens when the computer is first turned on?
- MS-DOS
  - A startup program in the BIOS is executed
  - This program in turn accesses the master boot record on the floppy or hard disk drive
  - A loader then transfers the system files IO.SYS and MSDOS.SYS from the disk drive to the main memory
  - Finally, the command interpreter COMMAND.COM is loaded into memory which puts the DOS prompt on the screen that gives the user access to DOS's built-in commands like DIR, COPY, VER.
- The 640 K Barrier
  - DOS was designed to run on the original IBM PC
  - 8088 microprocessor, 1Mbyte of main memory
  - IBM divided this 1Mb address space into specific blocks
    - 640 K of RAM (user RAM)
    - 384 K reserved for ROM functions (control programs for the video system, hard drive controller, and the basic input/output system)

Memory Map



MS-DOS Functions and BIOS Services

- Program Support
- BIOS: usually stored in ROM these routines provide access to the hardware of the PC
- Access to the BIOS is done through the software interrupt instruction Int *n*
- For example, the BIOS keyboard services are accessed using the instruction INT 16h
- In addition to BIOS services DOS also provides higher level functions
  - INT 21h
  - More details later

Week 2
The 80x86 Microprocessor Architecture

Brief History of the 80x86 Family

- Evolution from 8080/8085 to 8086
  - In 1987, Intel introduced a 16-bit microprocessor called the 8086
  - It was a major improvement over the previous generation 8080/8085 microprocessors
    - 1 Mbyte memory (20 address lines) vs 8080/8085's capability of 64 Kbytes
    - 8080/8085 was an 8 bit system, meaning that the data larger than 8 bits should be broken into 8-bit pieces to be processed by the CPU; in contrast 8086 is a 16 bit microprocessor
    - 8086 is pipelined vs nonpipelined 8080/8085; in a system with pipelining the data and address busses are busy transferring data while the CPU is processing information
- Evolution from 8086 to 8088
  - 8086 is a microprocessor with a 16-bit data bus internally and externally
  - Internal because all registers are 16 bits wide
  - External because the data bus was 16 bits to transfer data in and out of the CPU
  - There was a resistance in using the 16 bit external data bus since at that time peripherals were designed around 8-bit microprocessors
  - Intel then came out with the 8088 version with 8-bit data bus

Brief History - Continued

- Success of 8088
  - IBM picked up the 8088 as their microprocessor of choice in designing the IBM PC
  - All specification of the hardware and software of the PC are made public by IBM and Microsoft (in contrast with Apple computers)
- Other microprocessors: 80386, 80386, 80486
  - Intel introduced 80286 in 1982
  - 16 bit internal and external data buses
  - 24 address lines (16 Mbyte main memory)
  - Virtual memory: a way of fooling the microprocessor into thinking that it has access to almost unlimited amount of memory by swapping data between disk storage and RAM
  - Real mode vs protected mode
  - Intel unveiled the 80386 (sometimes called the 80386DX) in 1985; internally and externally a 32 bit microprocessor with a 32 bit address bus (4 Gbyte physical memory)
  - Numeric data processing chips were made available: 8087, 80287, 80387 etc.

Evolution of Intel's microprocessors

Processor	Date of Introduction	Transistors on Chip	Maximum MIPS at Introduction[a]	Maximum Clock Frequency at Introduction[b]	On-chip Cache Memory	Maximum Addressable Memory
8086	1978	29K	0.8	8 MHz		1 MB
80286	1982	134K	2.7	12.5 MHz		16 MB
80386	1985	275K	6	20 MHz		4 GB
80486	1989	1.2M	20	25 MHz[c]	8K Level 1	4 GB
Pentium	1993	3.1M	100	60MHz	16K Level 1	4 GB
Pentium Pro	1995	5.5M	440	200 MHz	16K Level 1, 256K/ 512K Level 2	64 GB
Pentium II	1997	7M	466	266 MHz	32K Level 1, 256/ 512K Level 2	64 GB
Pentium III	1999	8.2M	1,000	500 MHz	32K Level 1, 512K Level 2	64 GB

Virtual 8086 Mode

- Real Mode
  - Only one program can be run one time
  - All of the protection and memory management functions are turned off
  - Memory space is limited to 1MB
- Virtual 8086 Mode
  - The 386 hands each real mode program its own 1MB chunk of memory
  - Multiple 8086 programs to be run simultaneously but protected from each other (multiple MSDOS prompts)
  - Due to time sharing, the response becomes much slower as each new program is launched
  - The 386 can be operated in Protected Mode and Virtual 8086 mode at the same time.
  - Because each 8086 task is assigned the lowest privilege level, access to programs or data in other segments is not allowed thus protecting each task.
  - We'll be using the virtual 8086 mode in the lab experiments on PCs that do have either Pentiums or 486s.

The 80286 and above - Modes of Operation

•Real Mode

•The address space is limited to 1MB using address lines A0-19; the high address lines are inactive

•The segmented memory addressing mechanism of the 8086 is retained with each segment limited to 64KB

•Two new features are available to the programmer

–Access to the 32 bit registers

–Addition of two new segments F and G

•Protected Mode

–Difference is in the new addressing mechanism and protection levels

–Each memory segment may range from a single byte to 4GB

–The addresses stored in the segment registers are now interpreted as pointers into a descriptor table

–Each segment's entry in this table is eight bytes long and identifies the base address of the segment, the segment size, and access rights

–In 8088/8086 any program can access the core of the OS hence crash the system. Access Rights are added in descriptor tables.

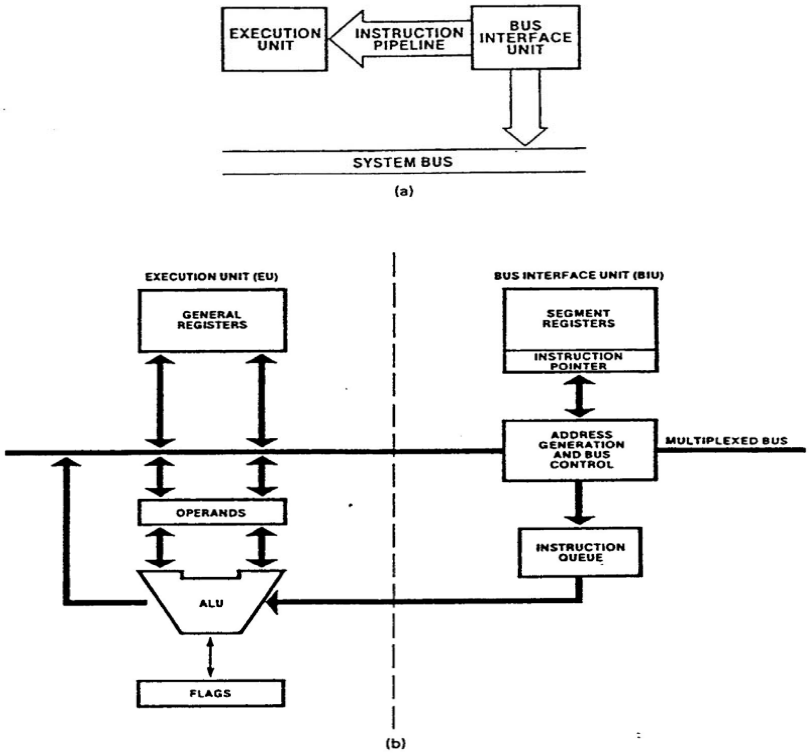
Virtual Memory

- 286 onward supported Virtual Memory Management and Protection™
- Unlimited amount of main memory assumed
- Two methods are used:
  - Segmentation
  - Paging
- Both techniques involve swapping blocks of user memory with hard disk space as necessary
  - If the program needs to access a block of memory that is indicated to be stored in the disk, the OS searches for an available memory block (typically using a least recently used algorithm) and swaps that block with the desired data on the hard drive
  - Memory swapping is invisible to the user
  - Segmentation: the block size is variable ranging up to 4GB
  - Paging: Block sizes are always 4 KB at a time.
- A final protected mode feature is the ability to assign a privilege level to individual tasks (programs). Tasks of lower privilege level cannot access programs or data with a higher privilege level. The OS can run multiple programs each protected from each other.

The 8086 and 8088

- The 8086 microprocessor represents the foundation upon which all the 80x86 family of processors have been built
- Intel has made the commitment that as new generations of microprocessors are developed, each will maintain software compatibility with this first generation part.
  - For example, a program designed to run on an **Intel** 386 microprocessor, which also runs on a Pentium, is **upward** compatible.
- **Processor model**
  - BIU (Bus Interface Unit) provides hardware functions including generation of the memory and I/O addresses for the transfer of data between itself and the outside world
  - EU (Execution Unit) receives program instruction codes and data from the BIU executes these instructions and stores the results in the general registers.
  - EU has no connection to the system busses; it receives and outputs all its data through the BIU.

Execution and Bus Interface Units



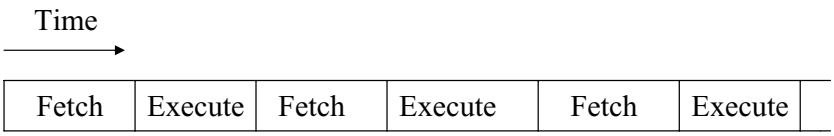
Fetch and Execute Cycle

- Fetch and execute cycles overlap
  - BIU outputs the contents of the IP onto the address bus
  - Register IP is incremented by one or more than one for the next instruction fetch
  - Once inside the BIU, the instruction is passed to the queue; this queue is a first-in-first-out register sometimes likened to a pipeline
  - Assuming that the queue is initially empty the EU immediately draws this instruction from the queue and begins execution
  - While the EU is executing this instruction, the BIU proceeds to fetch a new instruction.
    - BIU will fill the queue with several new instructions before the EU is ready to draw its next instruction
  - The cycle continues with the BIU filling the queue with instructions and the EU fetching and executing these instructions

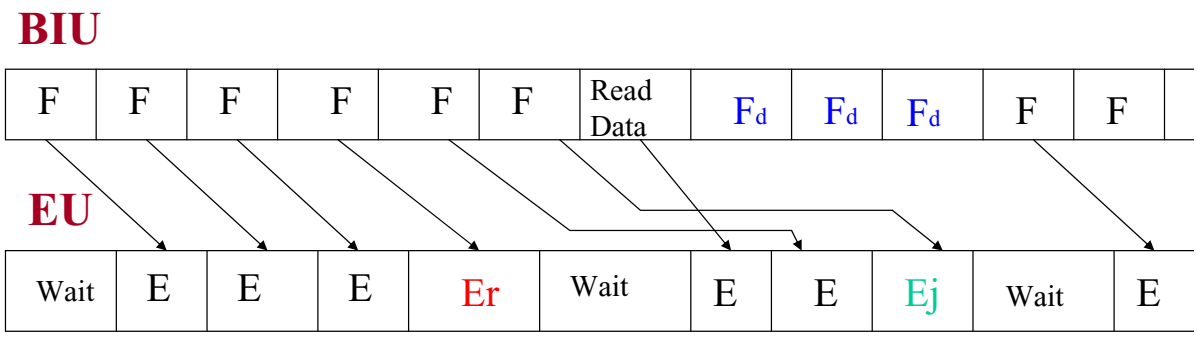
Pipelined Architecture

- Three conditions that will cause the EU to enter a wait mode
  - when the instruction requires access to a memory location not in the queue
  - when the instruction to be executed is a jump instruction; the instruction queue should be flushed out (known as branch penalty too much jumping around reduces the efficiency of the program)
  - during the execution of slow instructions
    - for example the instruction AAM (ASCII Adjust for Multiplication) requires 83 clock cycles to complete for an 8086
- 8086 vs 8088
  - BIU data bus width 8 bits for 8088, BIU data bus width 16 bits for 8086
  - 8088 instruction queue is four bytes instead of six
  - 8088 is found to be 30% slower than 8086
    - WHY
  - Long instructions provide more time for the BIU to fill the queue

Nonpipelined vs pipelined architecture



Non-pipelined architecture



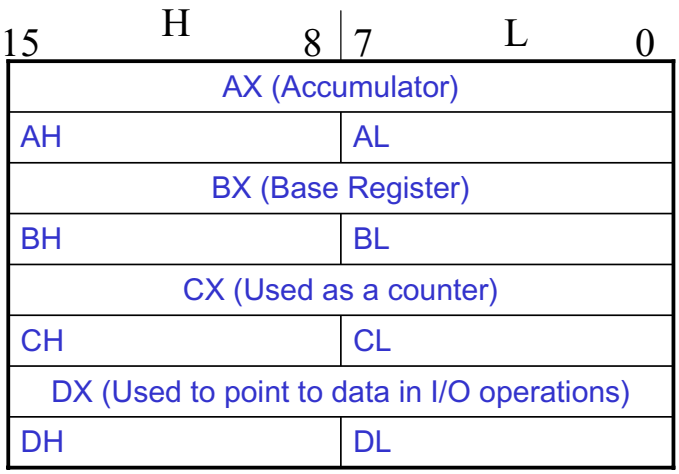
Pipelined architecture

Er: a request for data not in the queue
Ej: jump instruction occurs Fd: Discarded

Registers of the 8086/80286 by Category

Category	Bits	Register Names
General	16	AX,BX,CX,DX
	8	AH,AL,BH,BL,CH,CL,DH,DL
Pointer	16	SP (Stack Pointer), Base Pointer (BP)
Index	16	SI (Source Index), DI (Destination Index)
Segment	16	CS(Code Segment) DS (Data Segment) SS (Stack Segment) ES (Extra Segment)
Instruction	16	IP (Instruction Pointer)
Flag	16	FR (Flag Register)

General Purpose Registers



- **Data Registers** are normally used for storing temporary results that will be acted upon by subsequent instructions
- Each of the registers is 16 bits wide (AX, BX, CX, DX)
- General purpose registers can be accessed as either 16 or 8 bits e.g., AH: upper half of AX, AL: lower half of AX

Data Registers

Register	Operations
AX	Word multiply, word divide, word I/O
AL	Byte multiply, byte divide, byte I/O, decimal arithmetic
AH	Byte multiply, byte divide
BX	Store address information
CX	String operations, loops
CL	Variable shift and rotate
DX	Word multiply, word divide, indirect I/O

Pointer and Index Registers

SP	**Stack Pointer**
BP	**Base Pointer**
SI	**Source Index**
DI	**Destination Index**
IP	**Instruction Pointer**

The registers in this group are all 16 bits wide
Low and high bytes are not accessible
These registers are used as memory pointers
- Example: MOV AH, [SI]

Move the byte stored in memory location
whose address is contained in register SI to register AH

IP is not under direct control of the programmer

Computer Programming

- Machine Language vs Assembly Language
  - Machine language or object code is the only code a computer can execute but it is nearly impossible for a human to work with
  - E4 27 88 C3 E4 27 00 D8 E6 30 F4 the object code for adding two numbers input from the keyboard
- When programming a microprocessor, programmers often use assembly language
  - This involves 3-5 letter abbreviations for the instruction codes (mnemonics) rather than the binary or hex object codes

Address	Hex Object Code				Mnemonics		Comment
					Op-Code	Operand	
0100	E4	27			IN	AL,27H	Input first number from port 27H and store in AL
0102	88	C3			MOV	BL,AL	Save a copy of register AL in register BL
0104	E4	27			IN	AL,27H	Input second number to AL
0106	00	D8			ADD	AL,BL	Add contents of BL to AL and store the sum in AL
0107	E6	30			OUT	30H,AL	Output AL to port 30H
0109	F4				HLT		Halt the computer

Source code

Edit, Assemble, Test, and Debug Cycle

- Using an *editor*, the source code of the program is created. This means selecting the appropriate instruction mnemonics to accomplish the task
- A compiler program which examines the source code file generated by the editor and determines the object code for each instruction in the program, is then run. In assembly language programming, this is called an *assembler* (MASM (Chapter 2 of the textbook, DEBUG: Appendix A of the textbook, etc.,)
- The object code produced by the computer is loaded into the target computer's memory and is then *run.*
- *Debugging:* locating and fixing the source of error
- High-level programming Languages
  - Basic, Pascal, C, C++

MOV Instruction

- MOV destination,source
  - **8 bit moves**
    - MOV CL,55h
    - MOV DL,CL
    - MOV BH,CL
    - Etc.
  - **16 bit moves**
    - MOV CX,468Fh
    - MOV AX,CX
    - MOV BP,DI
    - Etc.

MOV Instruction

- **Data can be moved among all registers but data cannot be moved directly into the segment registers** (CS,DS,ES,SS).
  - To load as such, first load a value into a non-segment register and then move it to the segment register

```
MOV AX,2345h
MOV DS,AX
```

- **Moving a value that is too large into a register will cause an error**

```
MOV BL,7F2h       ; illegal
MOV AX,2FE456h    ; illegal
```

- **If a value less than than FFh is moved into a 16 bit register. The rest of the bits are assumed to be all zeros.**

```
MOV BX,5        ; BX = 0005 with BH  = 00 and BL = 05
```

MOV Instruction

• MOV AX,58FCH	√
• MOV DX,6678H	√
• MOV SI,924BH	√
• MOV BP,2459H	√
• MOV DS,2341H	x
• MOV CX,8876H	√
• MOV CS,3F47H	x
• MOV BH,99H	√

ADD Instruction

- ADD destination,source
- The ADD instruction tells the CPU to add the source and destination operands and put out the results in the destination

DESTINATION = DESTINATION + SOURCE

```
MOV AL,25H
MOV BL,34h
ADD AL,BL   ; (AL should read 59h once the instruction is executed)
MOV DH,25H
ADD DH,34h   ; (AL should read 59h once the instruction is executed)
```

Immediate operand

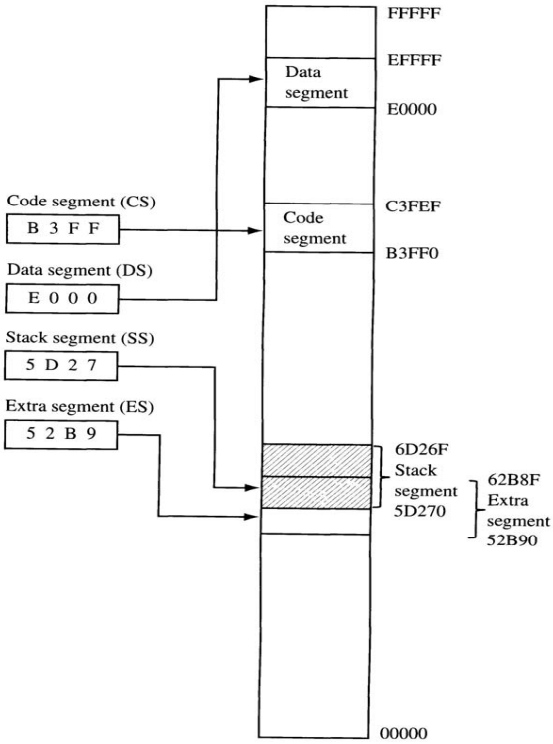
Origin and Definition of a Segment

- A segment is an area of memory that includes up to 64 Kbytes and begins on an address divisible by 16 (such an address ends with an hex digit 0h or 0000b)
  - 8085 could address 64Kbytes 16 address lines
- In the 8085, 64 K is for code, data, and stack
- In the 8086/88, 64 K is assigned to each category
  - Code segment
  - Data segment
  - Stack Segment
  - Extra Segment

Advantages of Segmented Memory

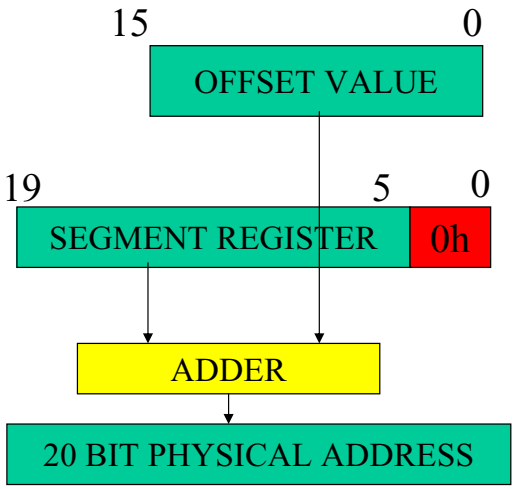
- One program can work on several different sets of data. This is done by reloading register DS to a new value.
- Programs that reference logical addresses can be loaded and run anywhere in the memory: relocatable
- Segmented memory introduces extra complexity in both hardware in that memory addresses require two registers.
- They also require complexity in software in that programs are limited to the segment size
- Programs greater than 64 KB can be run on 8086 but the software needed is more complex as it must switch to a new segment.
- Protection among segments is provided.

Segment Registers



Logical and Physical Addresses

- Addresses within a segment can range from address 0 to address FFFFh. This corresponds to the 64Kbyte length of the segment called an *offset*
- An address within a segment *logical address*
- *Ex.* Logical address 0005h in the code segment actually corresponds to B3FF0h + 5 = B3FF5h.



Example 1:
Segment base value: 1234h
Offset: 0022h

12340h
+ 0022h

12362h is the physical 20 bit address

Two different logical addresses may correspond to the same physical address.
D470h in ES 2D90h in SS
ES:D470h SS:2D90h

Example

•If DS=7FA2H and the offset is 438EH

a) Calculate the physical address

7FA20 + 438E = 83DAE

b) calculate the lower range

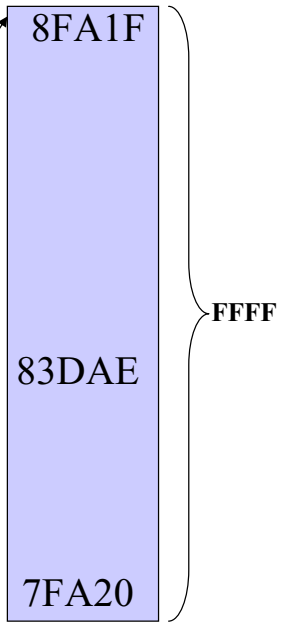
7FA20 + 0000 = 7FA20

c) Calculate the upper range of the data segment

7FA20 + FFFF = 8FA1F

d) Show the logical Address

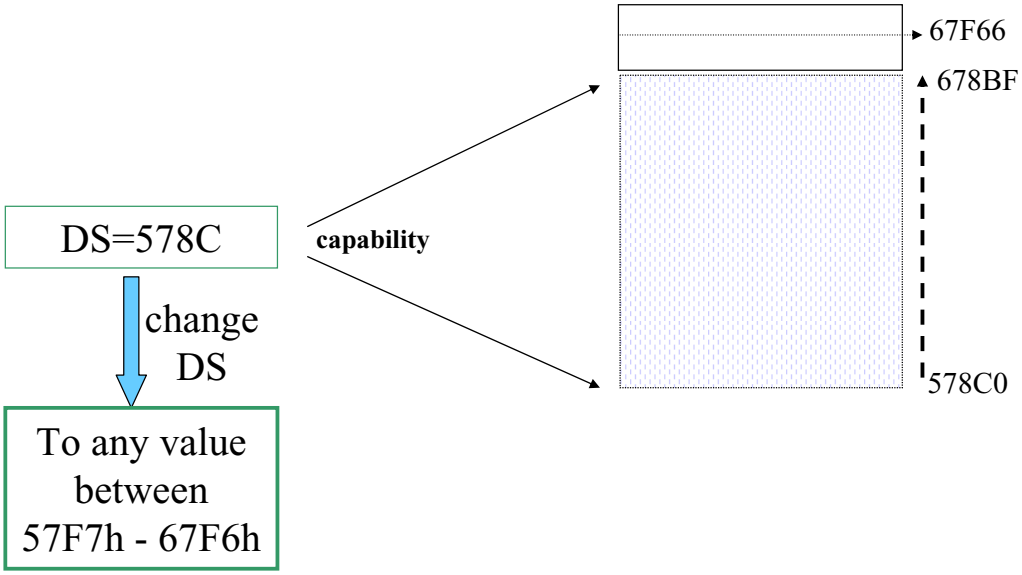
7FA2:438E



Example

Question:

Assume DS=578C. To access a Data in 67F66 what should we do?



Code Segment

- To execute a program, the 8086 fetches the instructions (opcodes and operands) from the code segment
- The logical address is in the form CS:IP

- Example: If CS = 24F6h and IP = 634Ah, show
  - The logical address
  - The offset address

  and calculate
  - The physical address
  - The lower range
  - The upper range

Logical Address vs Physical Address in the CS

CS:IP	Machine Language	Mnemonics
1132:0100	B057	MOV AL,57h
1132:0102	B686	MOV DH,86h
1132:0104	B272	MOV DL,72h
1132:0106	89D1	MOV CX,DX
1132:0108	88C7	MOV BH,AL
1132:010A	B39F	MOV BL,9F
1132:010C	B420	MOV AH,20h
1132:010E	01D0	ADD AX,DX
1132:0110	01D9	ADD CX,BX
1132:0112	05351F	ADD AX, 1F35h

- Show how the code resides physically in the memory

Data Segment

- Assume that a program is written to add 5 bytes of data 25h,12h,15h,1Fh, and 2Bh.
- One way to do it

```
MOV AL,00h
ADD AL, 25h
ADD AL, 12h
ADD AL,15h
ADD AL,1Fh
ADD AL,2Bh
```

- Data and code are mixed in the instructions here
- The problem with it is if the data changes, the code must be searched for every place the data is included and data retyped.
- It is a good idea then to set aside an area of memory strictly for data

Data Segment

- The data is first placed in the memory locations

```
DS:0200 = 25h
DS:0201 = 12h
DS:0202 = 15h
DS:0203 = 1Fh
DS:0204 = 2Bh
```

- Then the program is written as

```
MOV AL,0
ADD AL,[0200]   ; bracket means add the contents of DS:0200 to AL
ADD AL,[0201]
ADD AL,[0202]
ADD AL,[0203]
ADD AL,[0204]
```

- If the data is stored at a different offset address, say 450 h, the program need to be rewritten

Data Segment

- The term pointer is used for a register holding an offset address
- Use BX as a pointer

```
MOV AL,0
MOV BX,0200h
ADD AL,[BX]
INC BX
ADD AL,[BX]
INC BX
ADD AL,[BX]
INC BX
ADD AL,[BX]
INC BX
ADD AL,[BX]
```

- If the offset address of data is to be changed, only one instructions will need to be modified

16 bit Segment Register Assignments

Type of Memory Reference	Default Segment	Alternate Segment	Offset
Instruction Fetch	CS	none	IP
Stack Operations	SS	none	SP,BP
General Data	DS	CS,ES,SS	BX, address
String Source	DS	CS,ES,SS	SI, DI, address
String Destination	ES	None	DI

Little Endian Convention

"Little Endian" means that the low-order byte of the number is stored in memory at the lowest address, and the high-order byte at the highest address. (The little end comes first.)
Intel uses Little Endian Convention.
For example, a 4 byte LongInt

Byte3 | Byte2 | Byte1 |Byte0 will be arranged in memory as follows:
Base Address+0 Byte0
Base Address+1 Byte1
Base Address+2 Byte2
Base Address+3 Byte3

- **Adobe Photoshop** -- Big Endian
- **BMP (Windows and OS/2 Bitmaps)** – little Endian
- **GIF** -- Little Endian
- **IMG (GEM Raster)** -- Big Endian
- **JPEG** -- Big Endian

ENDIAN TYPE	B_0,B_1,B_2,B_3 = 0XAABBCCDD	SAMPLE MICROPROCESSORS
Little-Endian	aa bb cc dd	Intel x86, Digital (VAX, Alpha)
Big-Endian	dd cc bb aa	Sun, HP, IBM RS6000, SGI, "Java"

Computer Operating Systems

- What happens when the computer is first turned on?
- MS-DOS
  - A startup program in the **BIOS** (Basic Input Output System) is executed
  - This program in turn accesses the master boot record on the floppy or hard disk drive
  - A loader then transfers the system files **IO.SYS**
  - IO.SYS calls MSDOS.SYS. MS-DOS.SYS is basically the kernel of the operating system.
  - After initializing, MS-DOS.SYS then calls the command interpreter **COMMAND.COM** which is loaded into memory. This puts the DOS prompt on the screen that gives the user access to DOS's built-in commands like DIR, COPY, VER.

Memory Map of a PC

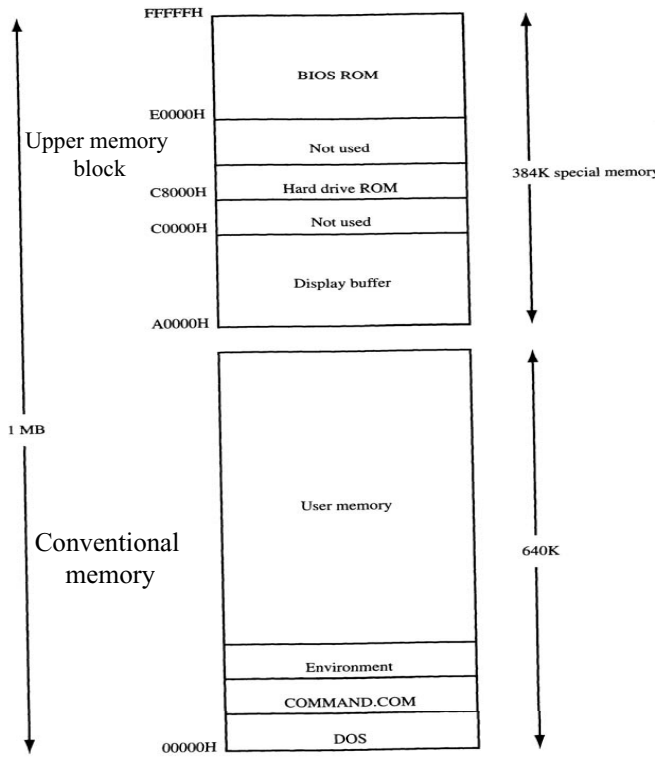
The 640 K Barrier

DOS was designed to run on the original IBM PC 8088 microprocessor, 1Mbytes of main memory

IBM divided this 1Mb address space into specific blocks

640 K of RAM (user RAM)

384 K reserved for ROM functions (control programs for the video system, hard drive



MS-DOS Functions and BIOS Services

BIOS: usually stored in ROM

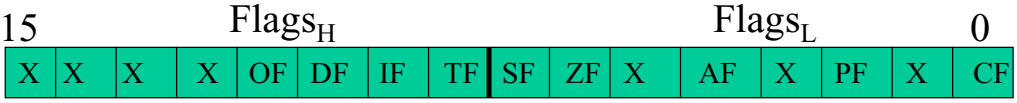
- tells the CPU what to do at startup
- these routines provide access to the peripheral devices of the PC, such as the keyboard, video, printer, and disk
- To test all the devices connected to the PC and alert if error

- Access to the BIOS is done through the software interrupt instruction Int *n*
- For example, the BIOS keyboard services are accessed using the instruction INT 16h
- In addition to BIOS services, DOS also provides higher level functions
  - INT 21h
  - More details later

More About RAM

- Memory management is one of the most important functions of the DOS operating systems and should be left to DOS
- Therefore, we do not assign any values for the DS,CS,SS registers; this is the job of DOS
- It is very important to remember that
  - The DS,CS, and DS values we will experiment will be different than those used by the textbook; do not worry

Flag (Status) Register



- Six of the flags are status indicators reflecting properties of the last arithmetic or logical instruction.
- For example, if register AL = 7Fh and the instruction ADD AL,1 is executed then the following happen
  - AL = 80h
  - CF = 0; there is no carry out of bit 7
  - PF = 0; 80h has an odd number of ones
  - AF = 1; there is a carry out of bit 3 into bit 4
  - ZF = 0; the result is not zero
  - SF = 1; bit seven is one
  - OF = 1; the sign bit has changed
- Can be used to transfer program control to a new memory location

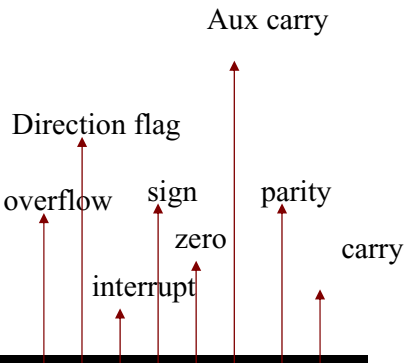
```
ADD AL,1
JNZ 0100h
```

Example

- Show how the flag register is affected by
  - MOV AX, 34F5h
  - ADD AX,95EBh

0011 0100 1111 0101
1001 0101 1110 1011

1100 1010 1110 0000

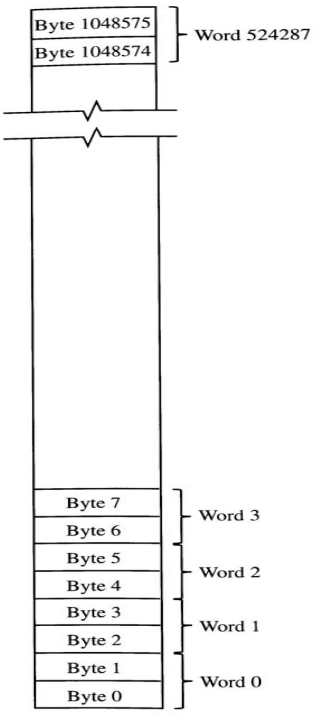



```
-t=1000

AX=34F5  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=1003   NV UP EI PL NZ NA PO NC
0B24:1003 05EB95        ADD     AX,95EB
-t

AX=CAE0  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=0B24  ES=0B24  SS=0B24  CS=0B24  IP=1006   NV UP EI NG NZ AC PO NC
0B24:1006 93            XCHG    BX,AX
```


TF, IF, and DF

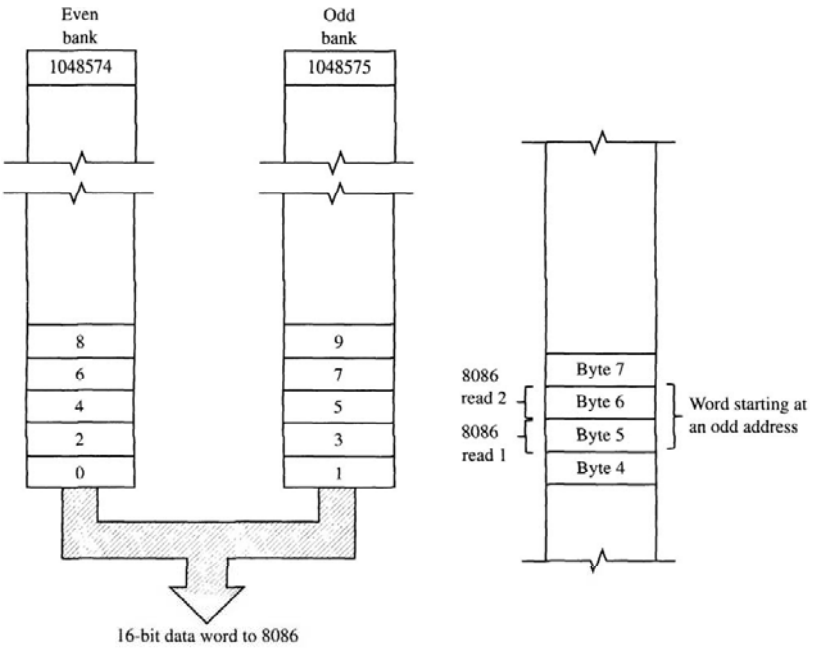
- Three of the flags can be set or reset directly by the programmer ands are used to control the operation of the microprocessor, these are TF, IF, and DF.
- When TF (Trap Flag) is set, control is passed to special address after each instruction is executed. Normally a program to display all the registers and flags is stored there. Single-stepping mode.
- When IF (Interrupt Flag) is set, external interrupt requests on the 8086's interrupt line INTR is enabled.
  - For example a printer may spend several seconds printing a page of text from its internal buffer
  - When it is ready for new data, the printer control circuit drives the 8086's INTR input line
  - The processor then suspends whatever it is doing and begins running the printer *interrupt service routine* (ISR)
  - When the routine has finished via a IRET (interrupt return) instruction control is transferred back to the original instruction in the main program that was executing when the interrupt occurred
  - Hardware and software interrupts
- DF (Direction Flag) is used with block move instructions (more later!!).
  - DF = 1 then the block memory pointer will automatically decrement
  - DF = 0, then the block memory pointer will automatically increment

Memory Address Space and Organization



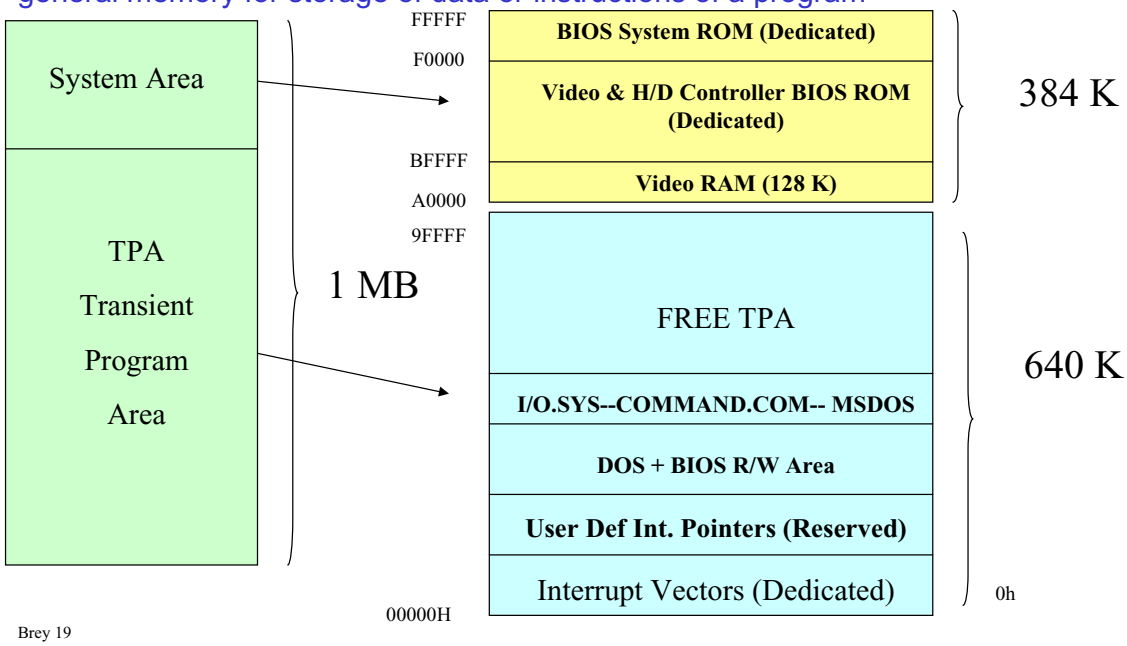
- Word
- Double Word
- Aligned Word
- Misaligned Word

Even addressed and odd-addressed banks



Dedicated, Reserved and General Purpose Memory

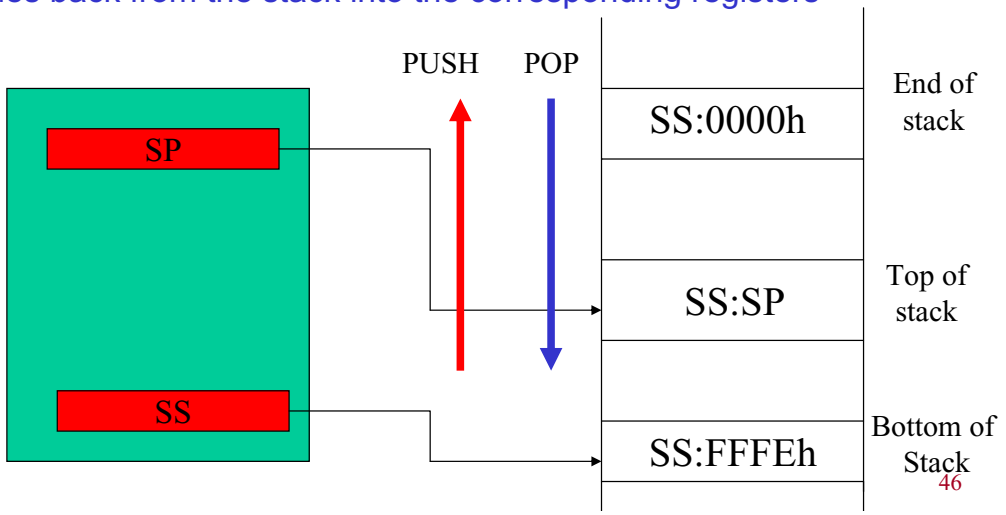
•Some address locations have dedicated functions and should not be used as general memory for storage of data or instructions of a program



Brey 19
Mazidi 32

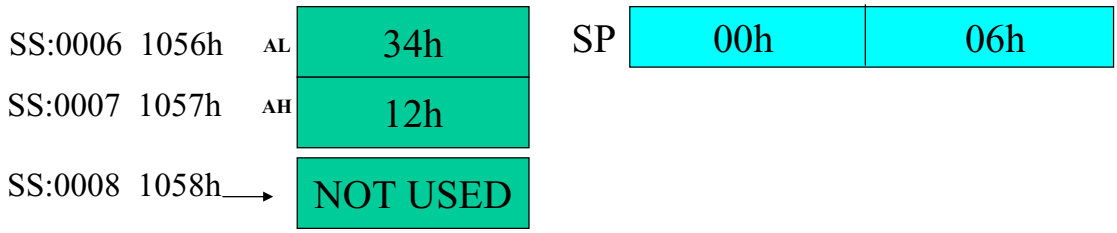
The Stack

- The stack is used for temporary storage of information such as data or addresses; for instance when a call is executed the 8088 automatically pushes the current value of CS and IP onto the stack.
- Other registers can also be pushed
- Near the end of the subroutine, pop instructions can be used to pop values back from the stack into the corresponding registers



Example for PUSH

- Given
  - SS = 0105h
  - SP = 0008h
  - AX = 1234h
  - What is the outcome of the PUSH AX instruction?
- A_{BOS} = 01050 + FFFEh = 1104h
- A_{TOS} = 01050 + 0008h = 1058h
- Decrement the SP by 2 and write AX into the word location 1056h.



Example for POP

- What is the outcome of the following
  - POP AX
  - POP BX
  - if originally 1058h contained AABBh?
- Read into the specified register from the stack and increment the stack pointer for each POP operation
- At the first POP
  - AX = 1234h SP = 0008h
- At the second POP
  - BX = AABBh SP = 000Ah

Week 3

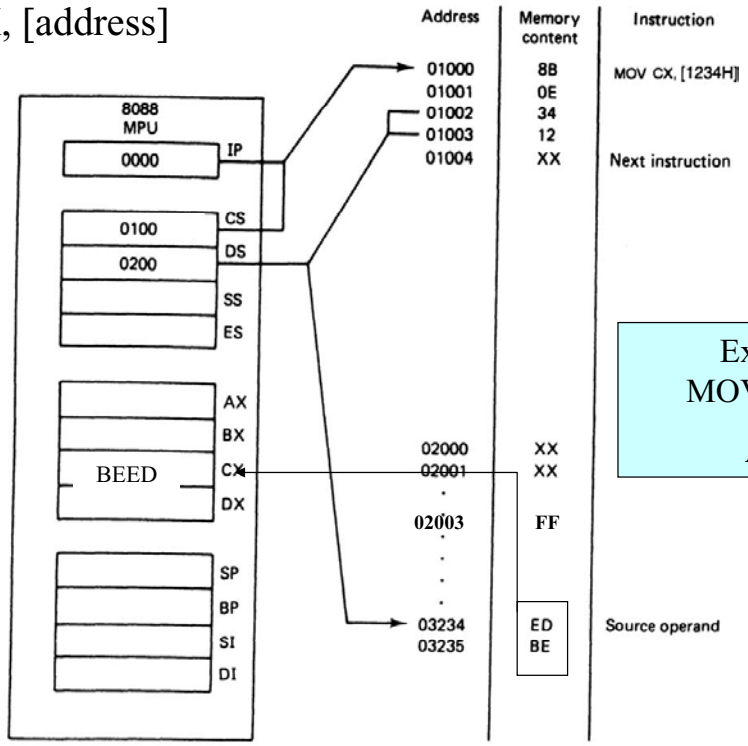
8086/8088 Addressing Modes, Instruction Set & Machine Codes

Addressing Modes

- When the 8088 executes an instruction, it performs the specified function on data
- These data, called operands,
  - May be a part of the instruction
  - May reside in one of the internal registers of the microprocessor
  - May be stored at an address in memory
- **Register Addressing Mode**
  - MOV AX, BX
  - MOV ES,AX
  - MOV AL,BH
- **Immediate Addressing Mode**
  - MOV AL,15h
  - MOV AX,2550h
  - MOV CX,625

Direct Addressing Mode

MOV CX, [address]

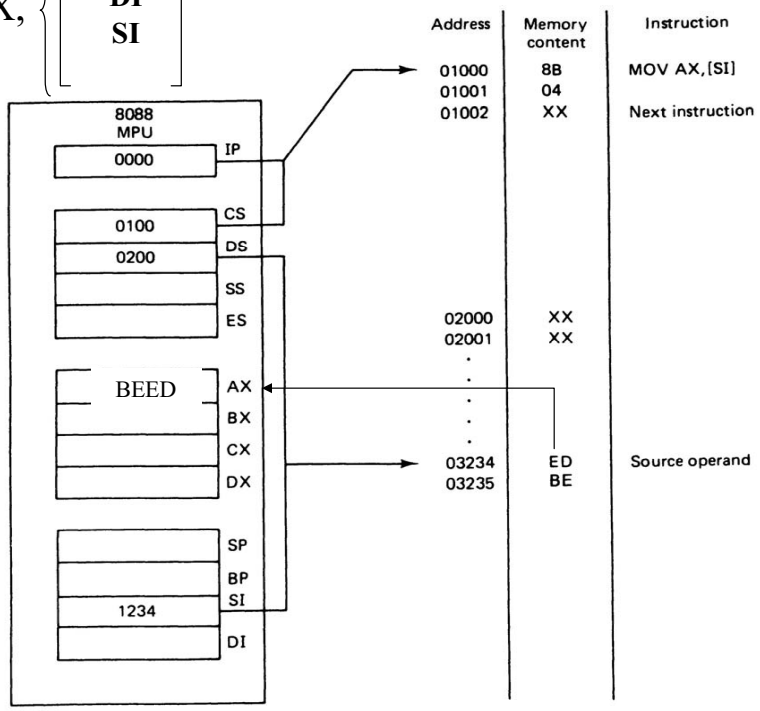


Example:
MOV AL,[03]

AL=?

Register Indirect Addressing Mode

MOV AX, { BX, DI, SI }



Example for Register Indirect Addressing

- Assume that DS=1120, SI=2498 and AX=17FE show the memory locations after the execution of:

MOV [SI],AX

DS (Shifted Left) + SI = 13698.

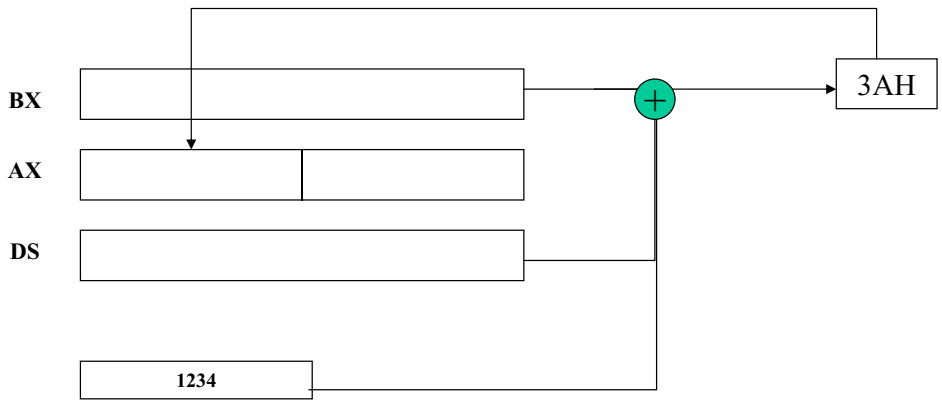
With little endian convention:

Low address 13698 → FE

High Address 13699 → 17

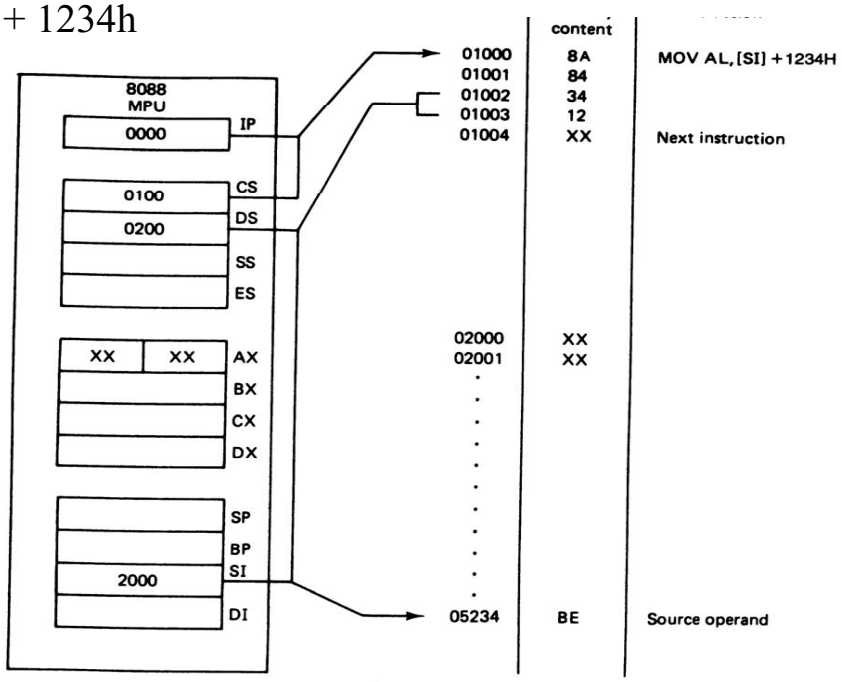
Based-Relative Addressing Mode

MOV AH, [DS:BX / SS:BP] + 1234h



Indexed Relative Addressing Mode

MOV AH, [$\frac{SI}{DI}$] + 1234h



Example: What is the physical address MOV [DI-8],BL if DS=200 & DI=30h ?
DS:200 shift left once 2000 + DI + -8 = 2028

Based-Indexed Relative Addressing Mode

- Based Relative + Indexed Relative
- We must calculate the PA (physical address)

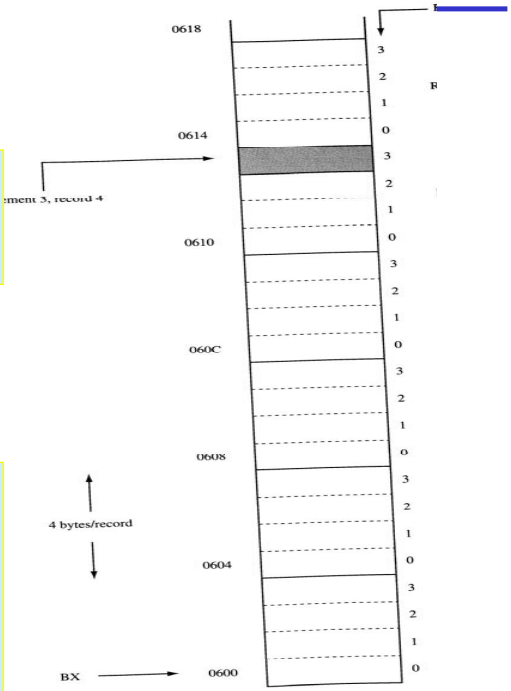
PA= [CS / SS / DS / ES] : [BX / BP] + [SI / DI] + [8 bit displacement / 16 bit displacement]

MOV AH,[BP+SI+29]
or
MOV AH,[SI+29+BP]
or
MOV AH,[SI][BP]+29

} The register order does not matter

Based-Indexed Addressing Mode

ure 4–4.
a base + index +
placement
dressing mode can
used to access a
rticular element in a
rticular record of an
ray.




```
MOV BX, 0600h
MOV SI, 0010h ; 4 records, 4 elements each.
MOV AL, [BX + SI + 3]
```

OR

```
MOV BX, 0600h
MOV AX, 004h ;
MOV CX,04;
MUL CX
MOV SI, AX
MOV AL, [BX + SI + 3]
```


Summary of the addressing modes

Addressing Mode	Operand	Default Segment
Register	Reg	None
Immediate	Data	None
Direct	[offset]	DS
Register Indirect	[BX] [SI] [DI]	DS DS DS
Based Relative	[BX]+disp [BP]+disp	DS SS
Indexed Relative	[DI]+disp [SI]+disp	DS DS
Based Indexed Relative	[BX][SI or DI]+disp [BP][SI or DI]+disp	DS SS

16 bit Segment Register Assignments

Segment Registers	CS	DS	ES	SS
Offset Register	IP	SI,DI,BX	SI,DI,BX	SP,BP

Type of Memory Reference	Default Segment	Alternate Segment	Offset
Instruction Fetch	CS	none	IP
Stack Operations	SS	none	SP,BP
General Data	DS	CS,ES,SS	BX, address
String Source	DS	CS,ES,SS	SI, DI, address
String Destination	ES	None	DI

Segment override

Instruction Examples	Override Segment Used	Default Segment
MOV AX,CS:[BP]	CS:BP	SS:BP
MOV DX,SS:[SI]	SS:SI	DS:SI
MOV AX,DS:[BP]	DS:BP	SS:BP
MOV CX,ES:[BX]+12	ES:BX+12	DS:BX+12
MOV SS:[BX][DI]+32,AX	SS:BX+DI+32	DS:BX+DI+32

Example for default segments

- The following registers are used as offsets. Assuming that the default segment used to get the logical address, give the segment register associated?

a) BP b)DI c)IP d)SI, e)SP, f) BX

- Show the contents of the related memory locations after the execution of this instruction

  MOV [BP][SI]+10,DX

  if DS=2000, SS=3000,CS=1000,SI=4000,BP=7000,DX=1299 (all hex)

SS(0)=30000
30000+4000+7000+10=3B010

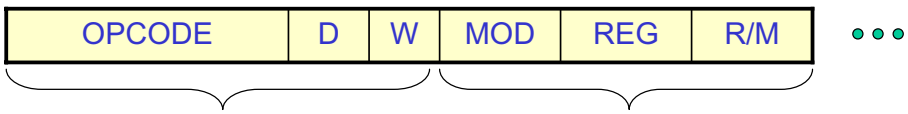
Assembly Language

- There is a one-to-one relationship between assembly and machine language instructions
- What is found is that a compiled machine code implementation of a program written in a high-level language results in inefficient code
  - More machine language instructions than an assembled version of an equivalent handwritten assembly language program
- Two key benefits of assembly language programming
  - It takes up less memory
  - It executes much faster

Languages in terms of applications

- One of the most beneficial uses of assembly language programming is **real-time applications**.
- Real time means the task required by the application must be completed before any other input to the program that will alter its operation can occur
- For example the device service routine which controls the operation of the floppy disk drive is a good example that is usually written in assembly language
- Assembly language not only good for controlling hardware devices but also **performing pure software operations**
  - searching through a large table of data for a special string of characters
  - Code translation from ASCII to EBCDIC
  - Table sort routines
  - Mathematical routines
- Assembly language: perform real-time operations
- High-level languages: Those operations mostly not critical in time.

Converting Assembly Language Instructions to Machine Code



- An instruction can be coded with 1 to 6 bytes
- **Byte 1 contains three kinds of information:**
  - Opcode field (6 bits) specifies the operation such as add, subtract, or move
  - Register Direction Bit (D bit)
    - Tells the register operand in REG field in byte 2 is source or destination operand
      - 1:Data flow to the REG field from R/M
      - 0: Data flow from the REG field to the R/M
  - Data Size Bit (W bit)
    - Specifies whether the operation will be performed on 8-bit or 16-bit data
      - 0: 8 bits
      - 1: 16 bits
- **Byte 2 has two fields:**
  - Mode field (MOD) – 2 bits
  - Register field (REG) - 3 bits
  - Register/memory field (R/M field) – 2 bits

Continued

- **REG field is used to identify the register for the first operand**

REG	W = 0	W = 1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Continued

- 2-bit MOD field and 3-bit R/M field together specify the second operand

CODE	EXPLANATION
00	Memory Mode, no displacement follows*
01	Memory Mode, 8-bit displacement follows
10	Memory Mode, 16-bit displacement follows
11	Register Mode (no displacement)

*Except when R/M = 110, then 16-bit displacement follows

(a)

MOD = 11			EFFECTIVE ADDRESS CALCULATION			
R/M	W = 0	W = 1	R/M	MOD = 00	MOD = 01	MOD = 10
000	AL	AX	000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16
001	CL	CX	001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) + D16
010	DL	DX	010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16
011	BL	BX	011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16
100	AH	SP	100	(SI)	(SI) + D8	(SI) + D16
101	CH	BP	101	(DI)	(DI) + D8	(DI) + D16
110	DH	SI	110	DIRECT ADDRESS	(BP) + D8	(BP) + D16
111	BH	DI	111	(BX)	(BX) + D8	(BX) + D16

(b)

Examples

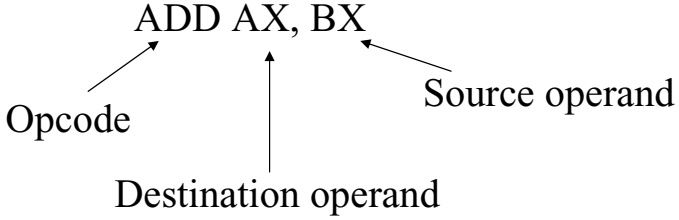
- MOV BL,AL
- Opcode for MOV = 100010
- We'll encode AL so
  - D = 0 (AL source operand)
- W bit = 0 (8-bits)
- MOD = 11 (register mode)
- REG = 000 (code for AL)
- R/M = 011

OPCODE	D	W	MOD	REG	R/M
100010	0	0	11	000	011

MOV BL,AL => 10001000 11000011 = 88 C3h
ADD AX,[SI] => 00000011 00000100 = 03 04 h
ADD [BX][DI] + 1234h, AX => 00000001 10000001 __ __ h
=> 01 81 34 12 h

Software

- The sequence of commands used to tell a microcomputer what to do is called a **program**
- Each command in a program is called an **instruction**
- 8088 understands and performs operations for **117 basic instructions**
- The native language of the **IBM PC** is the machine language of the 8088
- A program written in machine code is referred to as **machine code**
- In 8088 assembly language, each of the operations is described by alphanumeric symbols instead of just 0s or 1s.



DEBUG program instruction set (page 825 mzd)

- Debug instructions
- List of commands
  - a Assemble [address] you can type in code this way
  - c range address ; compare c 100 105 200
  - d [range] ; Dump d 150 15A
  - e address [list] ; Enter e 100
  - f Fill range list F 100 500 ‘ ‘
  - g Go [=address] addresses runs the program
  - h Value1 Value2 ; addition and subtraction H 1A 10
  - i Input port I 3F8
  - r Show & change registers Appears to show the same thing as t, but doesn't cause any code to be executed.
  - t=startaddress Trace either from the starting address or current location.
  - u startaddress UnAssemble

Some examples with debug

0100 mov ax,24b6
0103 mov di, 85c2
0106 mov dx,5f93
0109 mov sp,1236
010c push ax
010d push di
010e int 3

Display the stack contents after execution.
-D 1230 123F

Some examples with DEBUG

- 0100 mov al,9c
- 0102 mov dh,64
- 0104 add al,dh
- 0109 int 3

trace these three commands and observe the flags

T=<start trace location>

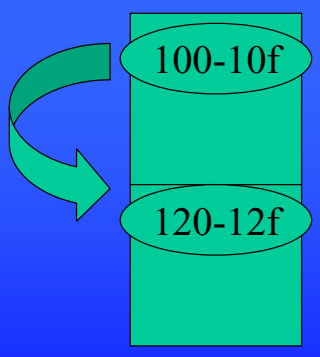
Saving and Loading a file

- After the code has been entered with the A command
- Use CX to store data indicating number of bytes to save. BX is the high word.
- Use N filename.com
- Then W command to write to file.
- L loads this file.

Example

Copy the contents of a block of memory (16 bytes) starting at location 20100h to another block of memory starting at 20120h

```
        MOV AX,2000
        MOV DS,AX
        MOV SI, 100
        MOV DI, 120
        MOV CX, 10
NXTPT:  MOV AH, [SI]
        MOV [DI], AH
        INC SI
        INC DI
        DEC CX
        JNZ  NXTPT
```



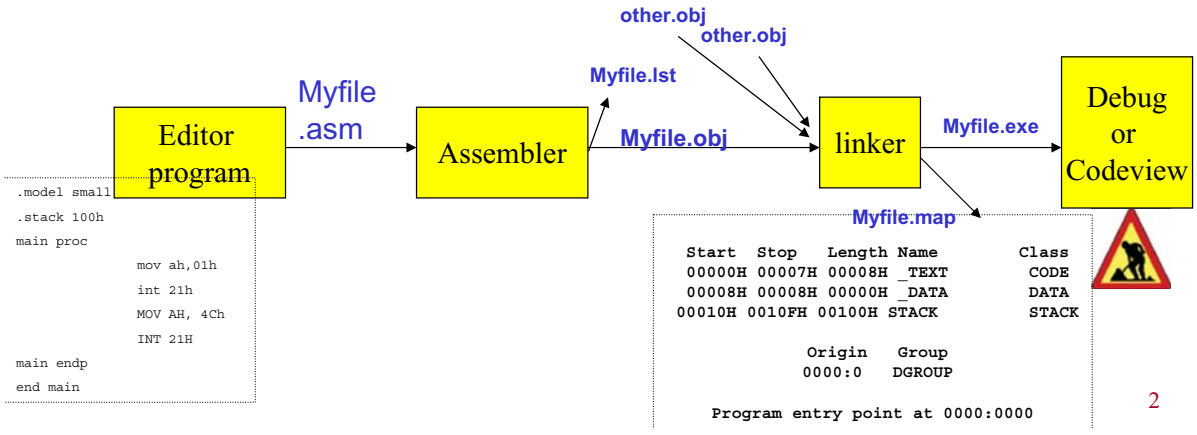
Week 4

8088/8086 Microprocessor Programming

Assemble, Link and Run a Program

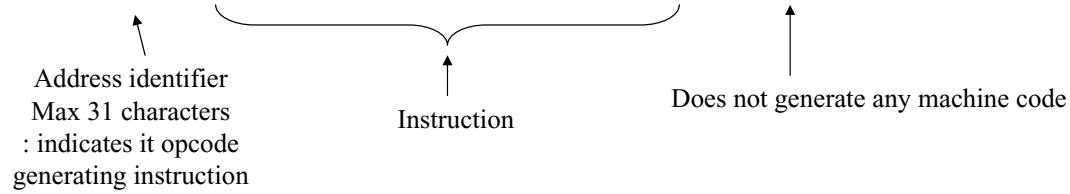
- Steps in creating an executable Assembly Language Program

Step	Input	Program	Output
1. Editing	Usually Keyboard	Editor (Text word editors etc.)	Myfile.asm
2. Assemble	Myfile.asm	MASM	Myfile.obj
3. Link	Myfile.obj	LINK	Myfile.exe



Instructions

[LABEL:] MNEMONIC [OPERANDS] [; COMMENT]



Ex. START: MOV AX,BX ; copy BX into AX

Sample Program

```
title Hello World Program          (hello.asm)
; This program displays "Hello, world!"
.model small
.stack 100h
.data
message db "Hello, world!",0dh,0ah,'$` ;newline+eoc
.code
main proc
    mov  ax,@data ; address of data
    mov  ds,ax
    mov  ah,9
    mov  dx,offset message ;disp.msg.starting at 0
    int  21h   ; or LEA dx,message will do!
    mov  ax,4C00h ; halt the program and return
    int  21h
main endp
end main
```

Assembly Language Basics

- Character or String Constants
  - 'ABC'
  - 'X'
  - "This isn't a test"
  - "4096"
- Numeric Literals
  - 26
  - 1Ah
  - 1101b
  - 36q
  - 2BH
  - 47d

Statements

- longarrayDefinition dw 1000h,1020h,1030h \
  1040h, 1050h, 1060h, 1070h
  Lines may break with "\" character
- Name limit of 247 characters
- Case insensitive
- Variables
  - Count1 db 50 ;a variable (memory allocation)
- Label,
  - If a name appears in the code area of the program it is a label.

```
LABEL1: mov ax,0
        mov bx,1
LABEL2: jmp Label1 ;jump to label1
```

Assembler Directives

.MODEL SMALL ; selects the size of the memory model usually sufficient max 64K code 64K data

.STACK ; size of the stack segment
.DATA ; beginning of the data segment
.CODE ; beginning of the code segment

Ex: .DATA

```
DATAW       DW    213FH
DATA1       DB    52H
SUM         DB    ? ; nothing stored but a storage is assigned
```

Ex: .CODE

```
PROGRAMNAME  PROC;  Every program needs a name
             …. ; program statements
PROGRAMNAME  ENDP
             END PROGRAMNAME
```


DataTypes and Data Definition

```
DATA1   DB      25
DATA2   DB      10001001b
DATA3   DB      12h
                ORG  0010h
DATA4   DB      "2591"
                ORG   0018h
DATA5   DB      ?
```

```
This is how data is initialized in the data
   segment
0000        19
0001        89
0002        12
0010        32 35 39 31
0018        00
```

DB DW DD

```
.data
   MESSAGE2 DB '1234567'
   MESSAGE3 DW 6667H
   data1 db 1,2,3
   db 45h
   db 'a'
   db 11110000b
   data2 dw 12,13
   dw 2345h
   dd 300h
```

```
; how it looks like in
        memory
31 32 33 34 35 36 37
      67 66
      1 2 3
        45
        61
        F0
    0C 00 0D 00
      45 23
    00 30 00 00
```


More Examples

```
DB    6 DUP(FFh); fill 6 bytes with ffh

DW   954
DW   253Fh      ; allocates two bytes
DW   253Fh,'HI'

DD 5C2A57F2h    ;allocates four bytes
DQ        "HI"   ; allocates eight bytes

COUNTER1   DB   COUNT
COUNTER2   DB   COUNT
```

More assembly

- OFFSET
  - The offset operator returns the distance of a label or variable from the beginning of its segment. The destination must be 16 bits
  - mov bx, offset count
- SEG
  - The segment operator returns the segment part of a label or variable's address.

```
Push ds
Mov ax, seg array
Mov ds, ax
Mov bx, offset array
.
Pop ds
```

- DUP operator only appears after a storage allocation directive.
  - db 20 dup(?)
- EQU directive assigns a symbolic name to a string or constant.
  - `Maxint equu 0ffffh`
  - `COUNT  EQU 2`

Memory Models

- Tiny code and data combined must be less than 64K
- Small Code <=64K and Data<= 64K
- Medium Data<=64K code any size multiple code seg
- Compact Code<=64K data any size multiple data seg
- Large Code>64K and data>64K multiple code and data seg
- Huge Same as the Large except that individual vars can be >64K

The PTR Operator

- INC [20h] ; is this byte/word/dword? or
- MOV [SI],5
  - Is this byte 05?
  - Is this word 0005?
  - Or is it double word 00000005?
- Byte or word or doubleword?

- To clarify we use the PTR operator
  - INC BYTE PTR [20h]
  - INC WORD PTR [20h]
  - INC DWORD PTR [20h]
- or for the mov example:
  - MOV byte ptr [SI],5
  - MOV word ptr[SI],5

The PTR Operator

- Would we need to use the PTR operator in each of the following?

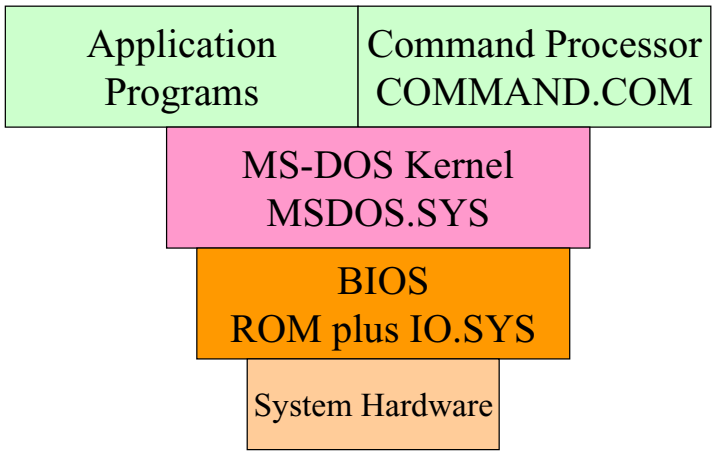
```
MOV AL,BVAL
MOV DL,[BX]
SUB [BX],2
MOV CL,WVAL
ADD AL,BVAL+1

.data
BVAL DB 10H,20H
WVAL DW 1000H
```

```
MOV AL,BVAL
MOV DL,byte ptr [BX]
SUB [BX],byte ptr 2
MOV CL,byte ptr WVAL
ADD AL,BVAL+1
```

Simple Assembly Language Program

```
        .MODEL SMALL
        .STACK 64
        .DATA
DATA1   DB  52h
DATA2   DB 29h
SUM     DB ?
        .CODE
MAIN    PROC FAR
        MOV AX,@DATA; copy the data segment into the DS reg.
        MOV DS,AX
        MOV AL,DATA1
        MOV BL,DATA2; or DATA1+1
        ADD AL,BL
        MOV SUM,AL
        MOV AH,4Ch
        INT 21h
MAIN    ENDP
        END MAIN
```


MS-DOS Functions and BIOS Calls



- BIOS is hardware specific
- BIOS is supplied by the computer manufacturer
- Resident portion which resides in ROM and nonresident portion IO.SYS which provides a convenient way of adding new features to the BIOS

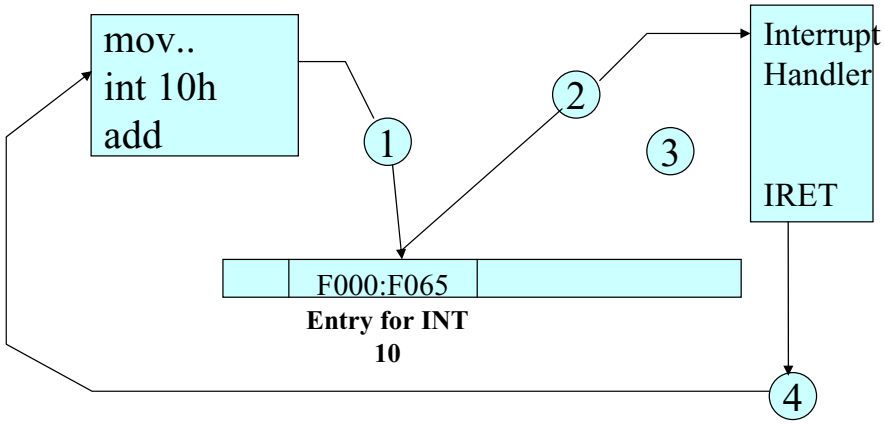
80x86 Interrupts

- An interrupt is an event that causes the processor to suspend its present task and transfer control to a new program called the interrupt service routine (ISR)
- There are three sources of interrupts
  - Processor interrupts
  - Hardware interrupts generated by a special chip, for ex: 8259 Interrupt Controller.
  - Software interrupts
- Software Interrupt is just similar to the way the hardware interrupt actually works!. The INT Instruction requests services from the OS, usually for I/O. These services are located in the OS.
- INT has a range 0→ FFh. Before INT is executed AH usually contains a function number that identifies the subroutine.

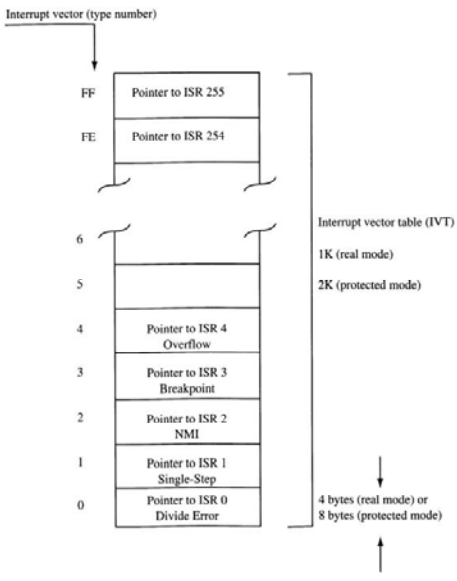
- Each interrupt must supply a type number which is used by the processor as a pointer to an interrupt vector table (IVT) to determine the address of that interrupt's service routine
- Interrupt Vector Table: CPU processes an interrupt instruction using the interrupt vector table (This table resides in the lowest 1K memory)
- Each entry in the IVT=32 bit segment+offset adress in OS, points to the location of the corresponding ISR.
- Before transferring control to the ISR, the processor performs one very important task
  - It saves the current program address and flags on the stack
  - Control then transfers to the ISR
  - When the ISR finishes, it uses the instruction IRET to recover the flags and old program address from the stack
- Many of the vectors in the IVT are reserved for the processor itself and others have been reserved by MS-DOS for the BIOS and kernel.
  - 10-1A are used by the BIOS
  - 20 – 3F are used by the MS-DOS kernel

80x86 Interrupts

- The number after the mnemonic tells which entry to locate in the table. For example INT 10h requests a video service.



Interrupt Vector Table



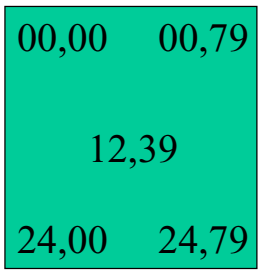
Processor	Pointer Size	IVT Location
Real Mode	4 bytes	Address 00000000–000003FF
Protected Mode	8 bytes	Anywhere in Physical Memory

Interrupts

- There are some extremely useful subroutines within BIOS or DOS that are available to the user through the INT (Interrupt) instruction.
- The INT instruction is like a FAR call; when it is invoked
  - It saves CS:IP and flags on the stack and goes to the subroutine associated with that interrupt.
  - Format:
    - INT xx ; the interrupt number xx can be 00-FFH
  - This gives a total of 256 interrupts
  - Common Interrupts
    - INT 10h Video Services
    - INT 16h Keyboard Services
    - INT 17h Printer Services
    - INT 21h MS-DOS services
  - Before the services, certain registers must have specific values in them, depending on the function being requested.

Some Software Interrupts

- INT 10H Function 06 (AH = 06) Scroll a screen windows.
  - **Moves the data on the video display up or down**. As screen is rolled the bottom is replaced by a blank line. Rows:0-24 from top, bottom: 0-79 from the left. (0,0) to (24,79). Lines scrolled can not be recovered!
  - AL = number of lines to scroll (with AL=00, window will be cleared)
  - BH = Video attribute of blank rows
  - CH, CL = Row,Column of upper left corner
  - DH, DL = Row,Column of lower right corner

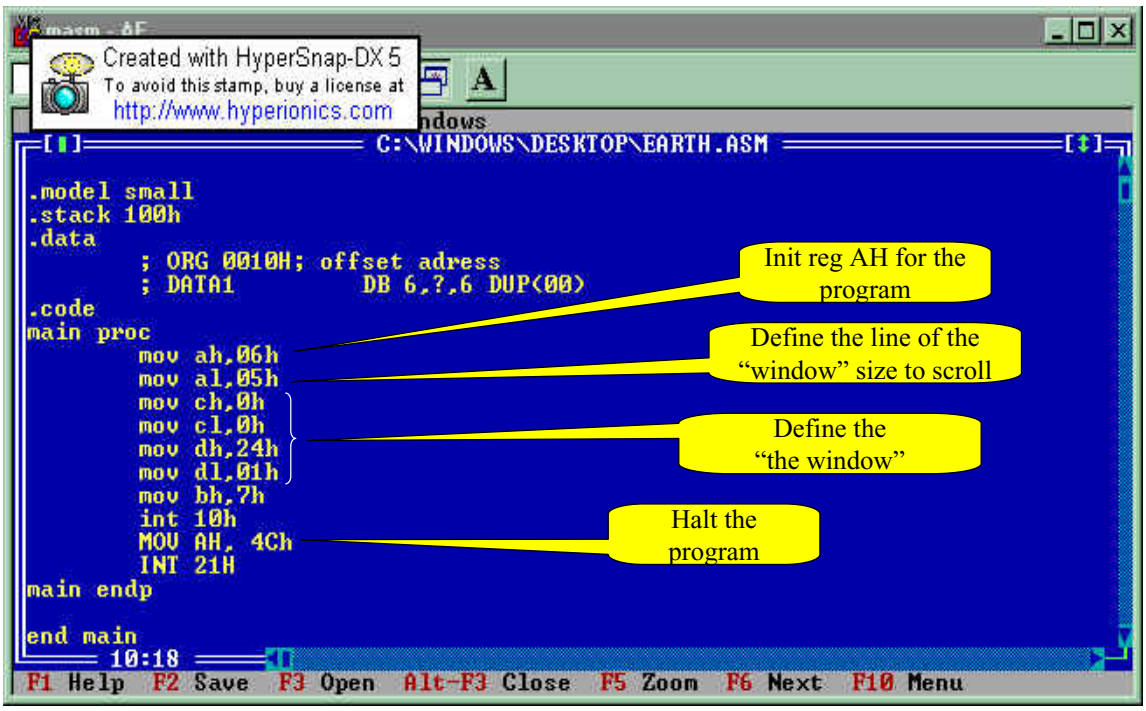


Cursor Locations

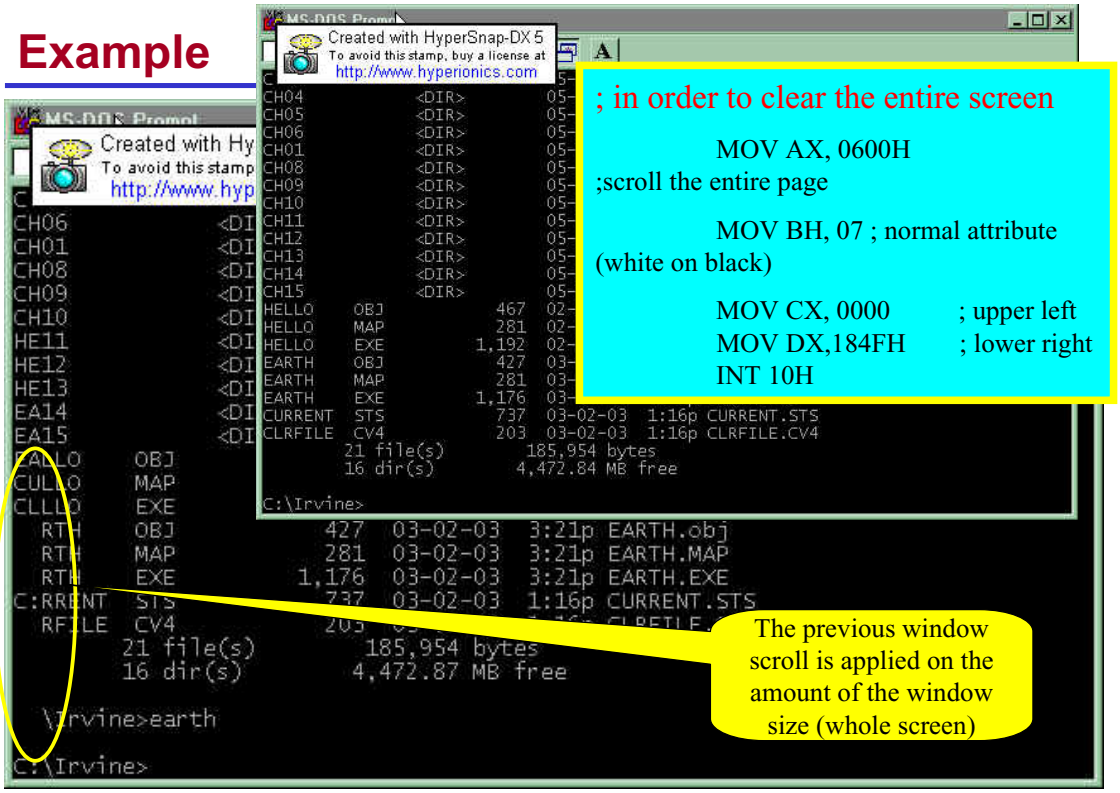
Example: Clear the screen by scrolling it upward with a normal attribute

```
mov ah,6h
mov al,0h
mov ch,0h
mov cl,0h
mov dh,24h
mov dl,01h
mov bh,7h
int 10h
```

Example Int10 06

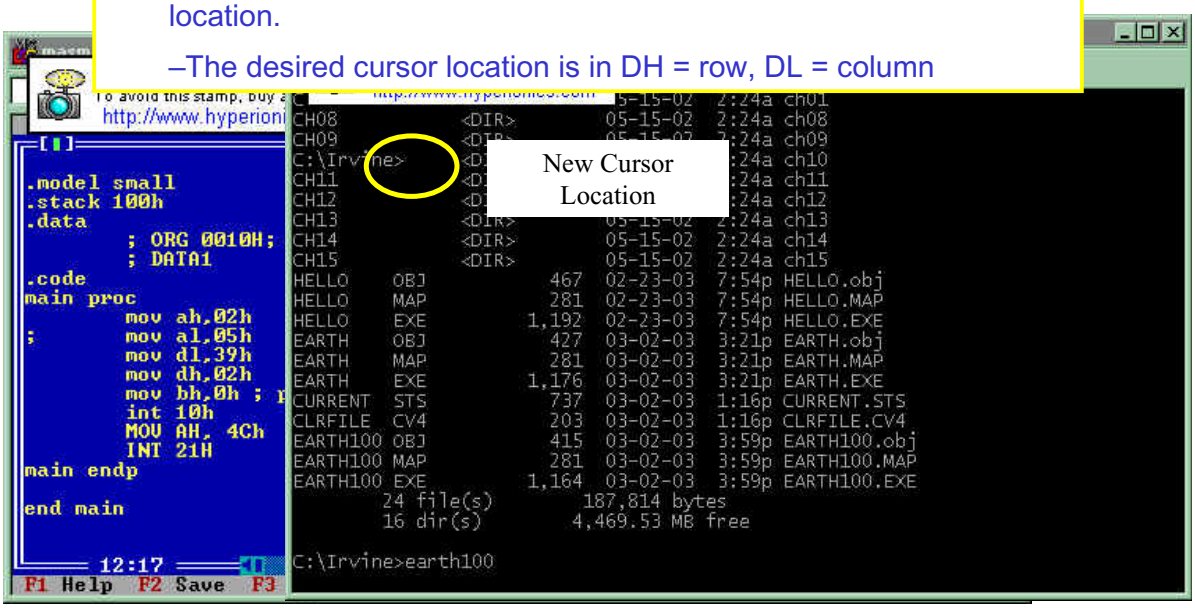


Example



Int 10 02H

- **INT 10H function 02**; setting the cursor to a specific location
  - –Function AH = 02 will change the position of the cursor to any location.
  - –The desired cursor location is in DH = row, DL = column



Int 10 03

- **INT 10H function 03**; get current cursor position

MOV AH, 03
MOV BH, 00
INT 10H

- Registers DH and DL will have the current row and column positions and CX provides info about the shape of the cursor.
- Useful in applications where the user is moving the cursor around the screen for menu selection

Int 10 05

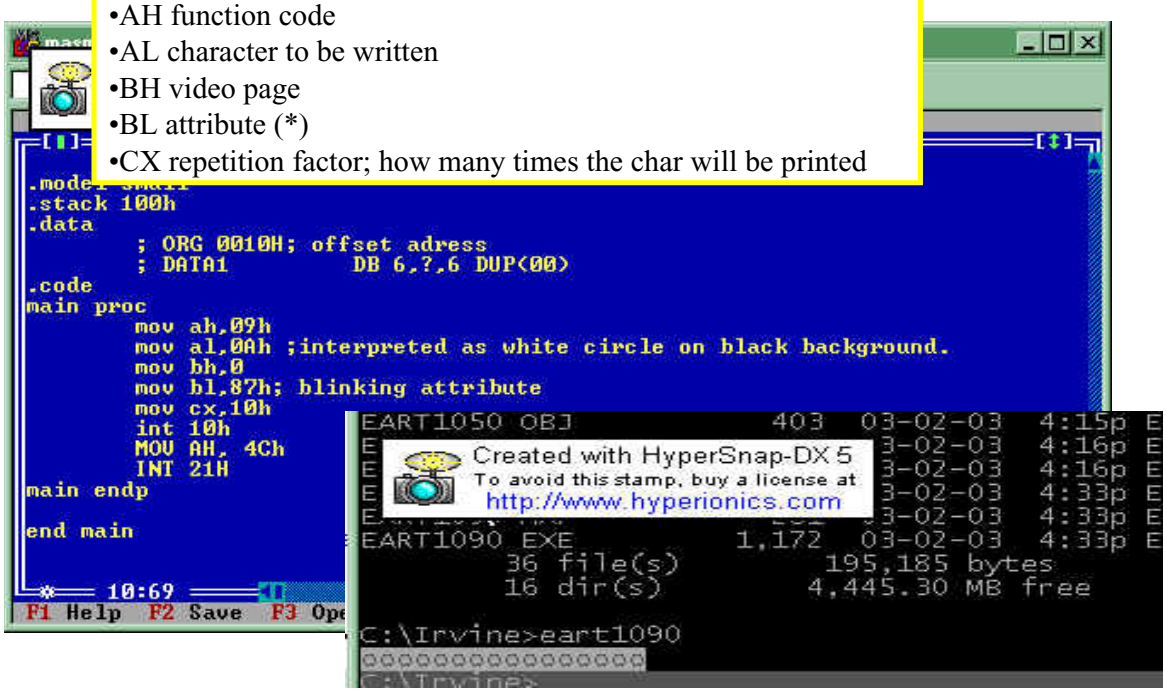
- **INT 10H function 05**; switch between video modes by adjusting AL

MOV AH, 05h
MOV AL, 01H; switch to video page1
INT 10H
; below will switch to video page 0
MOV AH, 05h
MOV AL, 00H; switch to video page0
INT 10H

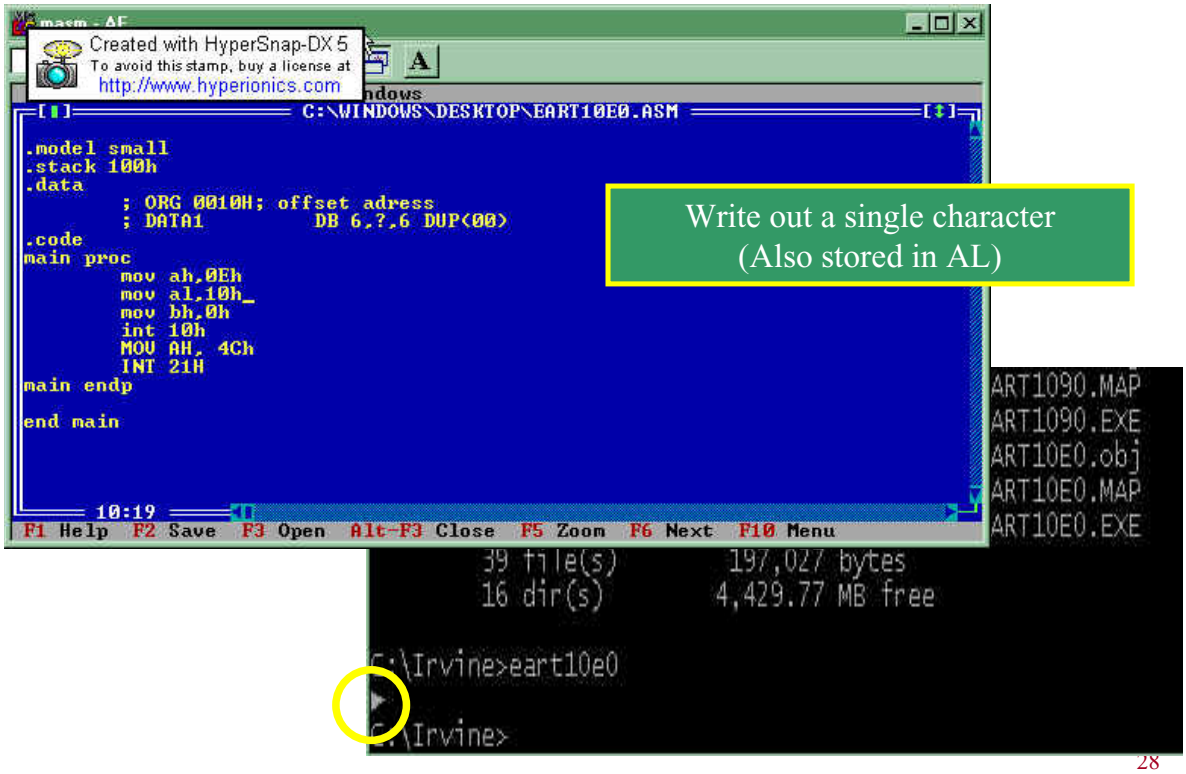
Extremely useful in text modes that support multiple pages!
This is what we had before Windows™

INT 10 - 09h or 0A (* no attribute)

- Write *one or more* characters at the current cursor position
- This function can display any ASCII character.
- AH function code
- AL character to be written
- BH video page
- BL attribute (*)
- CX repetition factor; how many times the char will be printed



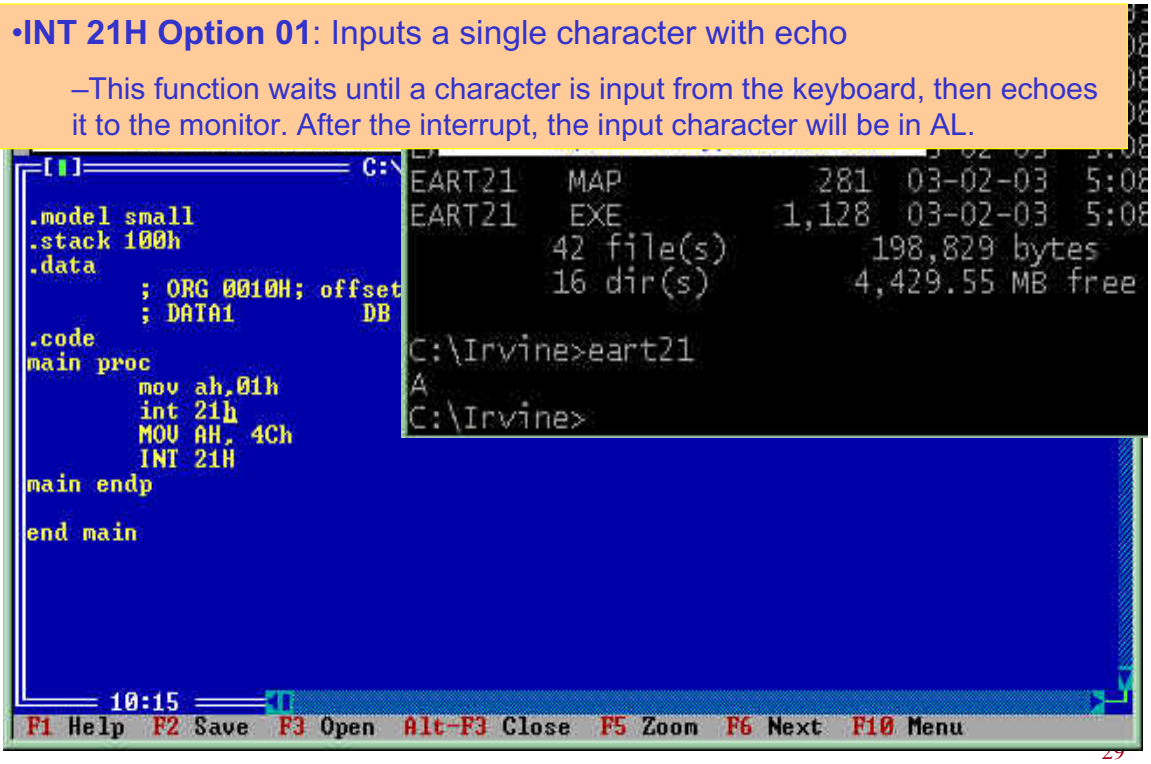
Int 10 - 0e



INT 21h

•**INT 21H Option 01**: Inputs a single character with echo

–This function waits until a character is input from the keyboard, then echoes it to the monitor. After the interrupt, the input character will be in AL.

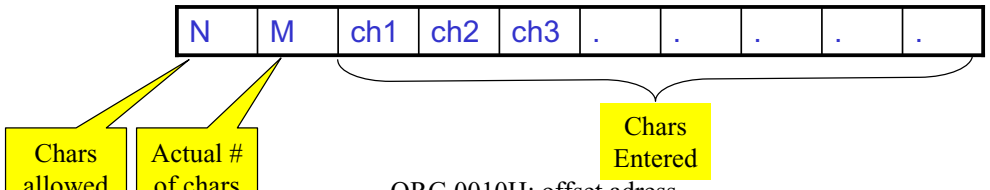


INT 21h

•**INT 21H Option 0AH/09H**: Inputs/outputs a string of data stored at DS:DX

–AH = 0AH, DX = offset address at which the data is located

–AH = 09, DX = offset address at which the data located




```
                ORG 0010H; offset adress
DATA1   DB 6,?,6 DUP(FF)

        MOV AH, 0AH
        MOV DX, OFFSET DATA1
        INT 21H
```

Ex. What happens if one enters USA and then <RETURN>

0010	0011	0012	0013	0014	0015	0016	0017
06	03	55	53	41	0D	FF	FF

INT 16h Keyboard Services

- Checking a key press, we use INT 16h function AH = 01

```
MOV AH, 01
INT 16h
```

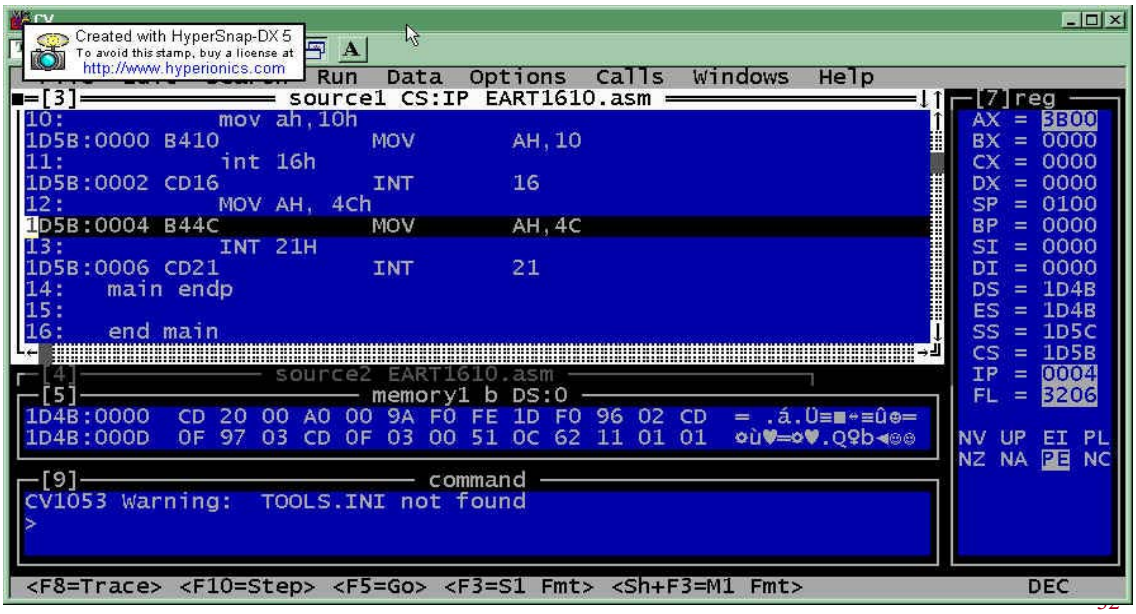
- Upon return, ZF = 0 if there is a key press; ZF = 1 if there is no key press
- Whick key is pressed?
- To do that, INT 16h function can be used immediately after the call to INT 16h function AH=01

```
MOV AH,0
INT 16h
```

- Upon return, AL contains the ASCII character of the pressed key

INT 16 – option 10 or 00

- BIOS Level Keyboard Input (more direct)
- Suppose F1 pressed (Scan Code 3BH). AH contains the scan code and AL contains the ASCII code (0).



Example. The PC Typewriter

- Write an 80x86 program to input keystrokes from the PC's keyboard and display the characters on the system monitor. Pressing any of the function keys F1-F10 should cause the program to end.
- Algorithm:
  1. Get the code for the key pressed
  2. If this code is ASCII, display the key pressed on the monitor and continue
  3. Quit when a non-ASCII key is pressed
- INT 16, BIOS service 0 – Read next keyboard character
  - Returns 0 in AL for non-ASCII characters or the character is simply stored in AL
- To display the character, we use INT 10, BIOS service 0E- write character in teletype mode. AL should hold the character to be displayed.
- INT 20 for program termination

Example

```
        MOV DX, OFFSET MES
        MOV AH,09h
        INT 21h ; to output the characters starting from the offset
AGAIN:  MOV AH,0h
        INT 16h; to check the keyboard
        CMP AL,00h
        JZ QUIT ;check the value of the input data
        MOV AH, 0Eh
        INT 10h; echo the character to output
        JMP AGAIN
QUIT:   INT 20h
MES     DB 'type any letter, number or punctuation key'
        DB 'any F1 to F10 to end the program"
        DB 0d,0a,0a,'$'
```

Data Transfer Instructions - MOV

Mnemonic	Meaning	Format	Operation	Flags Affected
MOV	Move	MOV D, S	(S) →(D)	None

Destination	Source
Memory	Accumulator
Accumulator	Memory
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Seg reg	Reg16
Seg reg	Mem16
Reg 16	Seg reg
Memory	Seg reg

Seg immediate & Memory to memory are not allowed

Data Transfer Instructions - XCHG

Mnemonic	Meaning	Format	Operation	Flags Affected
XCHG	Exchange	XCHG D,S	(Dest) ↔ (Source)	None

Destination	Source
Reg16	Reg16
Memory	Register
Register	Register
Register	Memory

Example: XCHG [1234h], BX

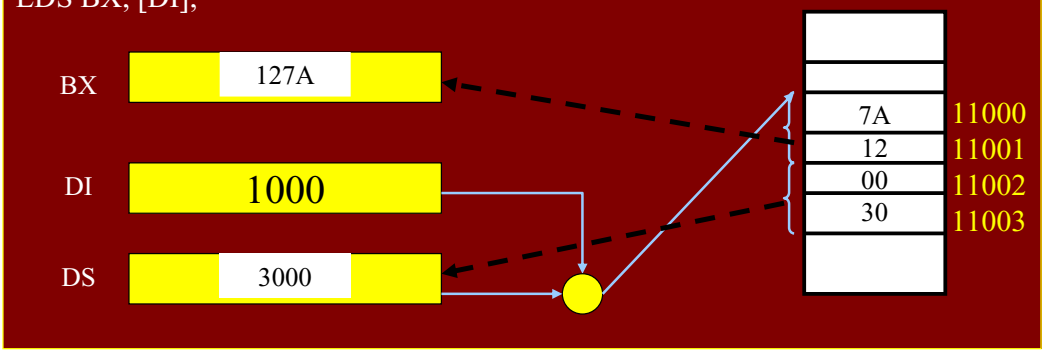
Data Transfer Instructions – LEA, LDS, LES

Mnemonic	Meaning	Format	Operation	Flags Affected
LEA	Load Effective Address	LEA Reg16,EA	EA →(Reg16)	None
LDS	Load Register and DS	LDS Reg16, MEM32	(Mem32) → (Reg16) (Mem32 + 2) → (DS)	None
LES	Load Register and ES	LES Reg16, MEM32	(Mem32) → (Reg16) (Mem32 + 2) → (ES)	None

Examples for LEA, LDS, LES

```
DATAX  DW 1000H
DATAY  DW 5000H
.CODE
LEA SI, DATAX
MOV DI, OFFSET DATAY; THIS IS MORE EFFICIENT

LEA BX,[DI]; IS THE SAME AS…
MOV BX,DI; THIS JUST TAKES LESS CYCLES.

LEA BX,DI; INVALID!
```

LDS BX, [DI];



Arithmetic Instructions – ADD, ADC, INC, AAA, DAA

Mnemonic	Meaning	Format	Operation	Flags Affected
ADD	Addition	ADD D, S	(S) + (D) ➔ (D) Carry ➔ (CF)	All
ADC	Add with carry	ADC D, S	(S) + (D) + (CF) ➔ (D) Carry ➔ (CF)	All
INC	Increment by one	INC D	(D) + 1 ➔ (D)	All but CY
AAA	ASCII adjust after addition	AAA	Use AX for the source	AF,CF
AAD	ASCII adjust before! div	AAD	AX has two unpacked BCD before div	
DAA	Decimal adjust after addition	DAA	Adjusts AX for decimal	All

Examples

Ex. 1 ADD AX, 2
ADC AX, 2

Ex. 2 INC BX
INC word ptr [BX]

Ex. 3 ASCII CODE 0-9 = 30h – 39h
MOV AX, 38H ;(ASCII code for number 8)
ADD AL, 39H ;(ASCII code for number 9)
AAA; used for addition
ADD AX, 3030H; answer to ASCII ➔ 0107

Ex. 4 AL contains 25 (packed BCD)
BL contains 56 (packed BCD)

ADD AL, BL
DAA

```
  25
  56
+ ----------
  7B  → 81
```

Ex. 5 MOV BL,9
MOV AX,0702H
; convert to binary first
AAD; 00-99
DIV BL

Example

Write a program that adds two multiword numbers:

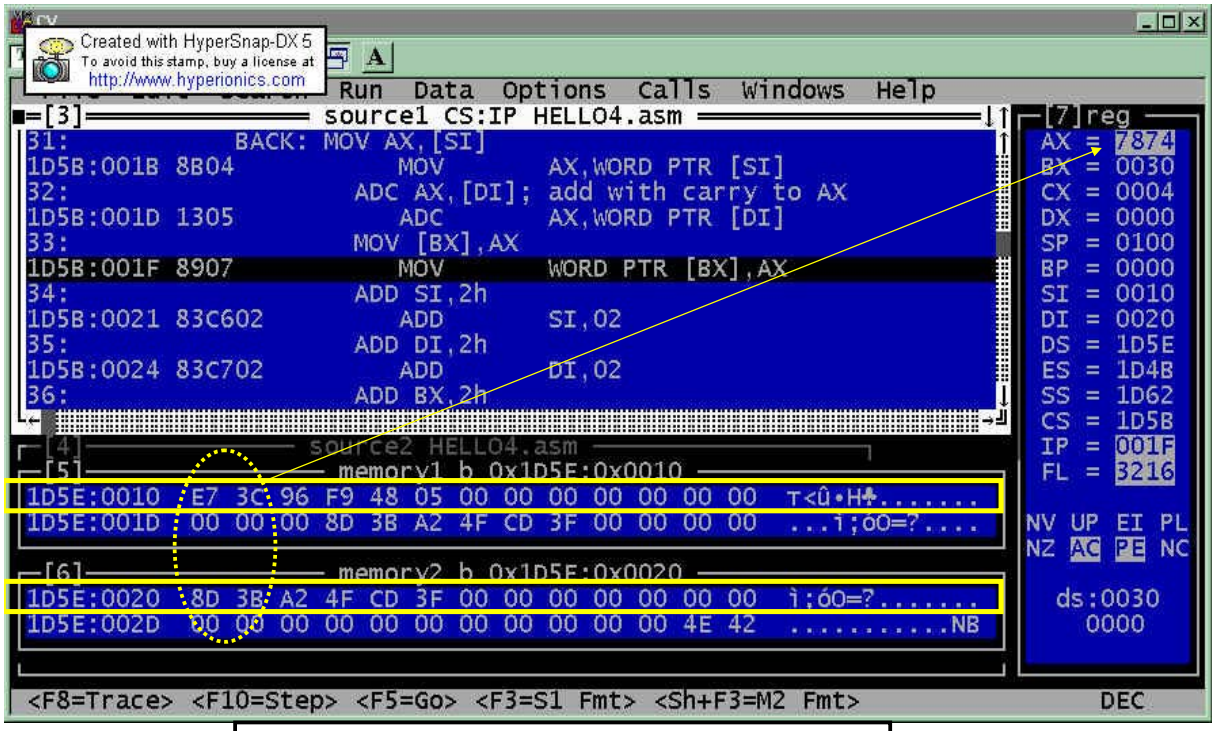
```
.MODEL SMALL
.STACK 64
.DATA
        DATA1 DQ 548F9963CE7h; allocate 8 bytes
ORG 0010h
        DATA2 DQ 3FCD4FA23B8Dh; allocate 8 bytes
ORG 0020h
        DATA3 DQ ?
```


Example Cont'd

```
.CODE
        MAIN PROC FAR
        MOV AX,@DATA; receive the starting address for DATA
        MOV DS,AX
        CLC; clear carry
        MOV SI,OFFSET DATA1; LEA for DATA1
        MOV DI,OFFSET DATA2; LEA for DATA2
        MOV BX,OFFSET DATA3; LEA for DATA3
        MOV CX,04h
        BACK: MOV AX,[SI]
                ADC AX,[DI]; add with carry to AX
                MOV [BX],AX
                ADD SI,2h
                ADD DI,2h
                ADD BX,2h
        LOOP BACK; decrement CX automatically until zero
        MOV AH,4Ch
        INT 21h; halt
MAIN ENDP
END MAIN
```

INC SI
INC SI
INC DI
INC DI
INC BX
INC BX

Example Cont'd



After 1st word addition

Arithmetic Instrutions – SUB, SBB, DEC, AAS, DAS, NEG

Mnemonic	Meaning	Format	Operation	Flags Affected
SUB	Subtract	SUB D, S	(D) - (S) ➔ (D) Borrow ➔ (CF)	All
SBB	Subtract with borrow	SBB D, S	(D) - (S) - (CF) ➔ (D)	All
DEC	Decrement by one	DEC D	(D) - 1 ➔ (D)	All but CY
NEG	Negate	NEG D	2's complement operation	All
DAS	Decimal adjust for subtraction	DAS	(convert the result in AL to packed decimal format)	All
AAS	ASCII adjust after subtraction	AAS	(convert the result in AX to packed decimal format) 37-38 -> 09	CY, AC

Examples with DAS and AAS

```
MOV BL, 28H
MOV AL, 83H
SUB AL,BL;    AL=5BH
DAS           ;    adjusted as AL=55H
```

```
MOV AX, 38H
SUB AL,39H ; AX=00FF
AAS            ; AX=FF09 ten's complement of –1
OR AL,30H   ; AL = 39
```


Example on SBB

- 32-bit subtraction of two 32 bit numbers X and Y that are stored in the memory as
  - X = (DS:203h)(DS:202h)(DS:201h)(DS:200h)
  - Y = (DS:103h)(DS:102h)(DS:101h)(DS:100h)
- The result X - Y to be stored where X is saved in the memory

```
MOV SI, 200h
MOV DI, 100h
MOV AX, [SI]
SUB AX, [DI]
MOV [SI], AX    ;save the LS word of result
MOV AX, [SI] +2 ; carry is generated from the first sub?
SBB AX, [DI] +2 ; then subtract CY this time!
MOV [SI] +2, AX
```

Ex. 12 34 56 78 – 23 45 67 89 = EE EE EE EF

Week 5

8088/8086 Microprocessor Programming

Multiplication and Division

Multiplication (MUL or IMUL)	Multiplicant	Operand (Multiplier)	Result
Byte * Byte	AL	Register or memory	AX
Word * Word	AX	Register or memory	DX :AX
Dword * Dword	EAX	Register or Memory	EDX :EAX

Division (DIV or IDIV)	Dividend	Operand (Divisor)	Quotient : Remainder
Word / Byte	AX	Register or memory	AL : AH
Dword / Word	DX:AX	Register or memory	AX : DX
Qword / Dword	EDX: EAX	Register or Memory	EAX : EDX

Unsigned Multiplication Exercise

```
DATAX    DB    4EH
DATAY    DW    12C3H
RESULT   DQ    DUP (?)
```

Find: Result = Datax * Datay

```
; one possible solution
XOR AX,AX
LEA SI, DATAX
MOV AL,[SI]
MUL DATAY
LEA DI, RESULT
MOV [DI],AX
MOV [DI+2],DX
```


AAM, AAD, CBW, CWD

- AAM: Adjust AX after multiply

```
MOV AL,07   ;  first unpacked number
MUL AL, 09  ; second unpacked number
AAM         ; AX unpacked decimal representation: 06 03
```

- AAD: Adjust AX for divide
- CBW instruction. Division instructions can also be used to divide an 8 bit dividend in AL by an 8 bit divisor.
  - In order to do so, the sign of the dividend must be extended to to fill the AX register
  - AH is filled with zeros if AL is positive
  - AH is filled with ones if the number in AL is negative
  - Automatically done by executing the CBW (convert byte to word) instruction. Simply extends the sign bit into higher byte.
- CWD (convert word to double word)

Ex.
```
MOV AL, 0A1h
CBW; convert byte to word
CWD; convert word to double word (push sign into DX)
```

Compare

Mnemonic	Meaning	Format	Operation	Flags Affected
CMP	Compare	CMP D,S	(D) − (S) is used in setting or resetting the flags	CF, AF, OF, PF, SF, ZF

(a)

Unsigned Comparison		
Comp Operands	CF	ZF
Dest > source	0	0
Dest = source	0	1
Dest < source	1	0

Destination	Source
Register	Register
Register	Memory
Memory	Register
Register	Immediate
Memory	Immediate
Accumulator	Immediate

(b)

Signed Comparison		
Comp Operands	ZF	SF,OF
Dest > source	0	SF=OF
Dest = source	1	x
Dest < source	0	SF<>OF

Compare Example

```
DATA1        DW     235Fh
…

MOV AX, CCCCH
CMP AX, DATA1
JNC OVER
SUB AX,AX
OVER: INC DATA1
```

CCCC – 235F = A96D => Z=0, CF=0 =>
CCCC > DATA1

Compare (CMP)

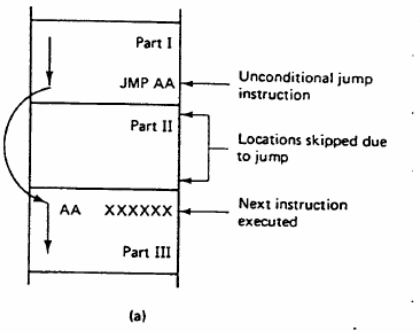
For ex: CMP CL,BL ; CL-BL; no modification on neither operands

Write a program to find the **highest** among 5 grades and write it in **DL**

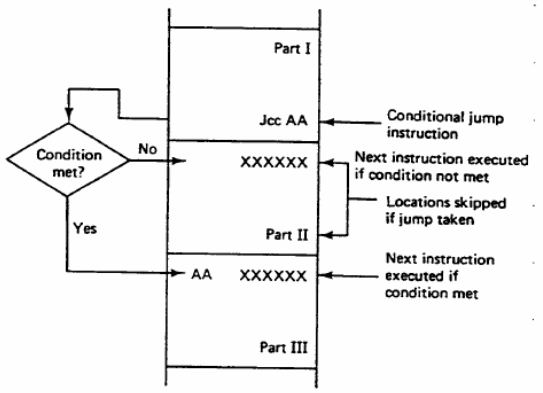
```
DATA    DB      51, 44, 99, 88, 80          ;13h,2ch,63h,58h,50h
        MOV     CX,5                        ;set up loop counter
        MOV     BX, OFFSET DATA             ;BX points to GRADE data
        SUB     AL,AL                       ;AL holds highest grade found so far
AGAIN:  CMP     AL,[BX]                     ;compare next grade to highest
        JA      NEXT                        ;jump if AL still highest
        MOV     AL,[BX]                     ;else AL holds new highest
NEXT:   INC     BX                          ;point to next grade
        LOOP AGAIN                          ;continue search
        MOV  DL, AL
```


Jump Instructions

- Unconditional vs conditional jump



(a)



(b)

Conditional Jump

These flags are based on general comparison

Mnemonic	Description	Flags/Registers
JZ	Jump if ZERO	ZF = 1
JE	Jump if EQUAL	ZF = 1
JNZ	Jump if NOT ZERO	ZF = 0
JNE	Jump if NOT EQUAL	ZF = 0
JC	Jump if CARRY	CF = 1
JNC	Jump if NO CARRY	CF = 0
JCXZ	Jump if CX = 0	CX = 0
JECXZ	Jump if ECX = 0	ECX = 0

Conditonal Jump based on flags

Mnemonic	Description	Flags/Registers
JS	JUMP IF SIGN (NEGATIVE)	SF = 1
JNS	JUMP IF NOT SIGN (POSITIVE)	SF = 0
JP	Jump if PARITY EVEN	PF = 1
JNP	Jump if PARITY ODD	PF = 0
JO	JUMP IF OVERFLOW	OF = 1
JNO	JUMP IF NO OVERFLOW	OF = 0

Jump Based on Unsigned Comparison

These flags are based on unsigned comparison

Mnemonic	Description	Flags/Registers
JA	Jump if above op1>op2	CF = 0 and ZF = 0
JNBE	Jump if not below or equal op1 not <= op2	CF = 0 and ZF = 0
JAE	Jump if above or equal op1>=op2	CF = 0
JNB	Jump if not below op1 not <opp2	CF = 0
JB	Jump if below op1<op2	CF = 1
JNAE	Jump if not above nor equal op1< op2	CF = 1
JBE	Jump if below or equal op1 <= op2	CF = 1 or ZF = 1
JNA	Jump if not above op1 <= op2	CF = 1 or ZF = 1

Jump Based on Signed Comparison

These flags are based on signed comparison

Mnemonic	Description	Flags/Registers
JG	Jump if GREATER op1>op2	SF = OF AND ZF = 0
JNLE	Jump if not LESS THAN or equal op1>op2	SF = OF AND ZF = 0
JGE	Jump if GREATER THAN or equal op1>=op2	SF = OF
JNL	Jump if not LESS THAN op1>=op2	SF = OF
JL	Jump if LESS THAN op1<op2	SF <> OF
JNGE	Jump if not GREATER THAN nor equal op1<op2	SF <> OF
JLE	Jump if LESS THAN or equal op1 <= op2	ZF = 1 OR SF <> OF
JNG	Jump if NOT GREATER THAN op1 <= op2	ZF = 1 OR SF <> OF

Control Transfer Instructions (conditional)

- It is often necessary to transfer the program execution.
  - Short
    - A special form of the direct jump: “short jump”
    - **All conditional jumps are short jumps**
    - Used whenever target address is in range +127 or –128 (single byte)
    - Instead of specifying the address a relative offset is used.

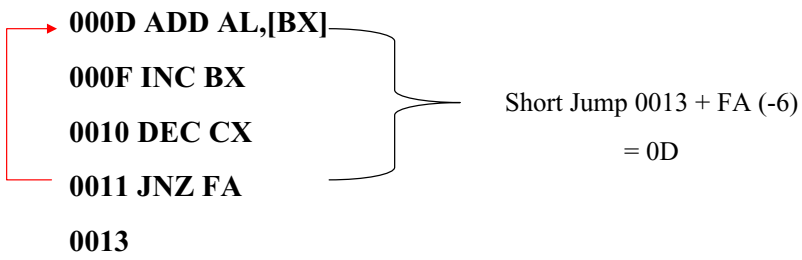
Short Jumps

•Conditional Jump is a two byte instruction.

•In a jump backward the second byte is the 2’s complement of the displacement value.

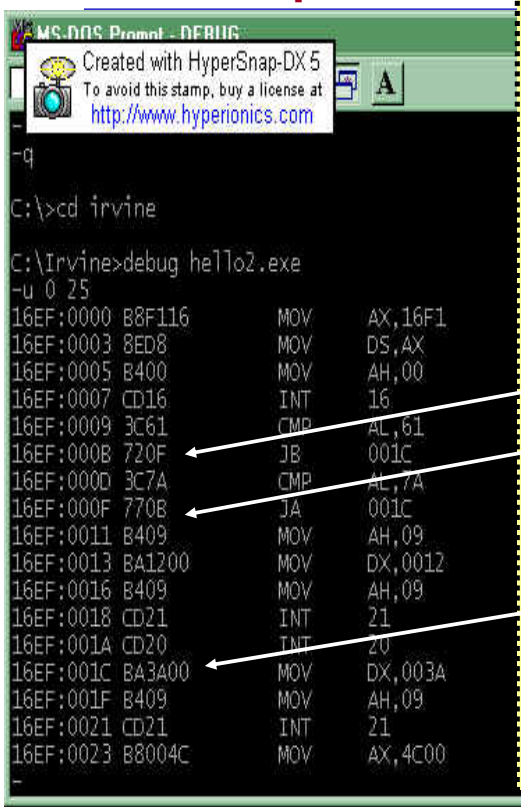
•To calculate the target the second byte is added to the IP of the instruction after the jump.

Ex:



Hello2.exe

SJ Example




```
.model small
.stack 100h
.data
org 0010
message1 db "You now have a small letter
entered !",0dh,0ah,'$'
org 50
message2 db "You have NON small letters
",0dh,0ah,'$'
.code
    main proc
        mov ax,@data
        mov ds,ax
        mov ah,00h
        int 16h
        cmp al,61h
        jb next
        Cmp al,7Ah
        ja next
        mov ah,09h
        mov dx,offset message1
        mov ah,09h
        int 21h
        int 20h
        next: mov dx,offset message2
        mov ah,09h
        int 21h
        mov  ax,4C00h
        int  21h
    main endp
end main
```

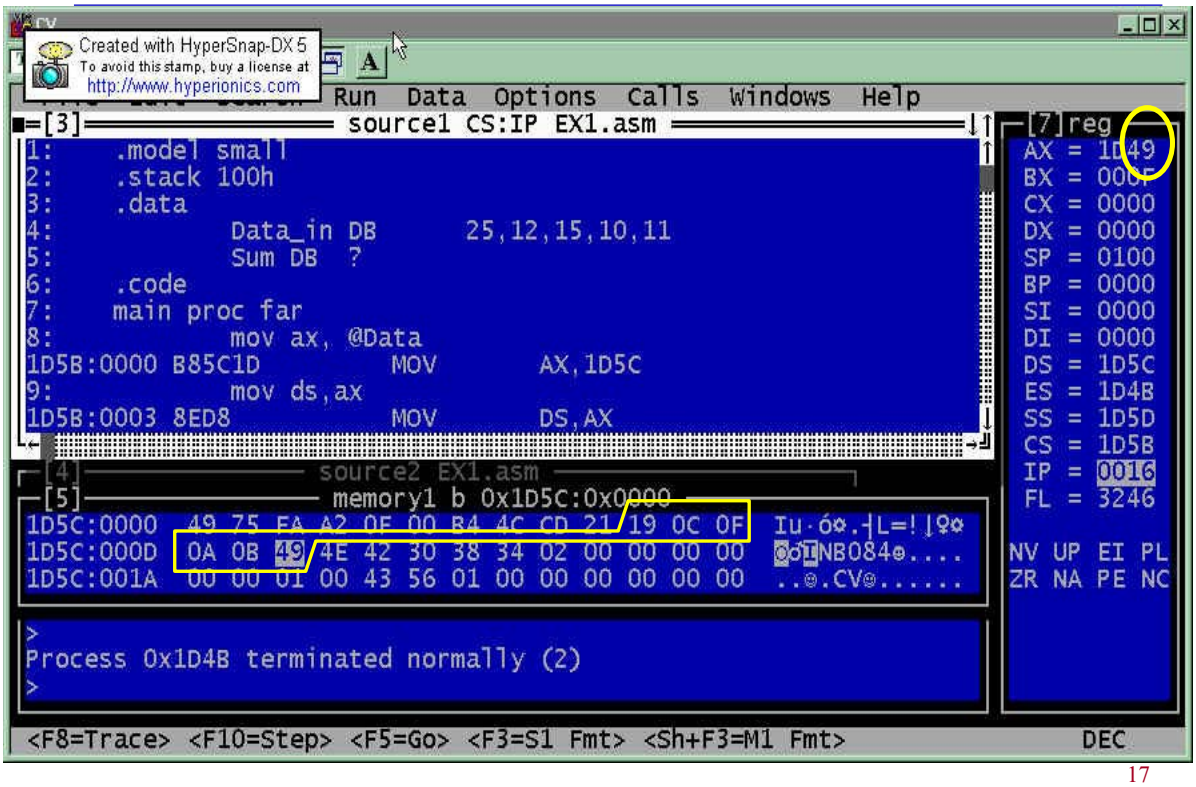
A Simple Example Program finds the sum

- Write a program that adds 5 bytes of data and saves the result. The data should be the following numbers: 25,12,15,10,11

```
.model small
.stack 100h
.data
        Data_in DB 25,12,15,10,11
        Sum DB  ?
.code
main proc far
        mov ax, @Data
        mov ds,ax
        mov cx,05h
        mov bx,offset data_in
        mov al,0
```

```
Again: add al,[bx]
        inc bx
        dec cx
        jnz Again
        mov sum,al
        mov ah,4Ch
        INT 21H
Main    endp
end main
```

Example Output



Unconditional Jump

❖Short Jump: jmp short L1 (8 bit)

❖Near Jump: jmp near ptr Label

If the control is transferred to a memory location within the current code segment (intrasegment), it is NEAR. IP is updated and CS remains the same

➢The displacement (16 bit) is added to the IP of the instruction following jump instruction. The displacement can be in the range of –32,768 to 32,768.

➢The target address can be register indirect, or assigned by the label.

➢**Register indirect JMP:** the target address is the contents of two memory locations pointed at by the register.

➢Ex: JMP [SI] will replace the IP with the contents of the memory locations pointed by DS:DI and DS:DI+1 or JMP [BP + SI + 1000] in SS

❖Far Jump: If the control is transferred to a memory location outside the current segment. Control is passing outside the current segment both CS and IP have to be updated to the new values. ex: JMP FAR PTR label = EA 00 10 00 20

jmp far ptr Label ; this is a jump out of the current segment.

Near Jump

```
0B20:1000 jmp 1200
0B20:1003
-u 1000
0B20:1000 E9FD01        JMP     1200
0B20:1003 200B          AND     [BP+DI],CL
```

Jumps to the specified IP with +/- 32K distance from the next instruction following the jmp instruction

Far Jump

```
0B20:1000 jmp 3000:1200
0B20:1005
-u 1000
0B20:1000 EA00120030    JMP     3000:1200
0B20:1005 FF750B        PUSH    [DI+0B]
```

Jumps to the specified CS:IP

XLAT

- Adds the contents of AL to BX and uses the resulting offset to point to an entry in an 8 bit translate table.
- This table contains values that are substituted for the original value in AL.
- The byte in the table entry pointed to by BX+AL is moved to AL.

- XLAT [tablename] ; optional because table is assumed at BX

- Table db '0123456789ABCDEF'

Mov AL,0A; index value
Mov bx,offset table
Xlat; AL=41h, or 'A'

Example

• Write a program that calculates the average of five temperatures and writes the result in AX

```
DATA    DB    +13,-10,+19,+14,-18          ;0d,f6,13,0e,ee
        MOV     CX,5                       ;LOAD COUNTER
        SUB     BX, BX          ;CLEAR BX, USED AS ACCUMULATOR
        MOV     SI, OFFSET DATA            ;SET UP POINTER
BACK:   MOV     AL,[SI]                    ;MOVE BYTE INTO AL
        CBW                                ;SIGN EXTEND INTO AX
        ADD     BX, AX                     ;ADD TO BX
        INC     SI                         ;INCREMENT POINTER
        DEC     CX                         ;DECREMENT COUNTER
        JNZ     BACK                       ;LOOP IF NOT FINISHED
        MOV     CL,5                       ;MOVE COUNT TO AL
        DIV     CL                         ;FIND THE AVERAGE
```

Compare (CMP)

For ex: CMP CL,BL ; CL-BL; no modification on neither operands

Write a program to find the highest among 5 grades and write it in DL

```
DATA    DB      51, 44, 99, 88, 80          ;13h,2ch,63h,58h,50h
        MOV     CX,5                        ;set up loop counter
        MOV     BX, OFFSET DATA             ;BX points to GRADE data
        SUB     AL,AL                       ;AL holds highest grade found so far
AGAIN:  CMP     AL,[BX]                     ;compare next grade to highest
        JA      NEXT                        ;jump if AL still highest
        MOV     AL,[BX]                     ;else AL holds new highest
NEXT:   INC     BX                          ;point to next grade
        LOOP AGAIN                          ;continue search
        MOV  DL, AL
```


Logical Instructions

- AND
  - Uses any addressing mode except memory-to-memory and segment registers. Places the result in the first operator.
  - Especially used in clearing certain bits (masking)
    - xxxx xxxx **AND** 0000 1111 = 0000 xxxx (clear the first four bits)
  - Examples: AND BL, 0FH; AND AL, [345H]
- OR
  - Used in setting certain bits
    - xxxx xxxx **OR** 0000 1111 = xxxx 1111
- XOR
  - Used in inverting bits
    - xxxx xxxx XOR 0000 1111 = xxxx yyyy
- **Ex.** Clear bits 0 and 1, set bits 6 and 7, invert bit 5

```
AND CX, OFCH    1111 1100
OR CX, 0C0H     1100 0000
XOR CX, 020H    0010 0000
XOR AX.,AX
```

Turn the CAPS LOCK on

CAUTION

```
push ds ; save the current ds
mov ax,40h ; new ds at BIOS
mov ds,ax
mov bx,17h   ;keyboard flag byte
xor byte ptr[bx],01000000b ;now you altered CAPS
pop ds
MOV Ah,4CH
INT 21H
```


TEST

- TEST instruction performs the AND operation but it does not change the destination operand as in AND but only the flags register.
- Similar to CMP bit it tests a single bit or occasionally multiple bits.
- **Ex.** TEST DL, DH ; TEST EAX, 256

```
TEST AL, 1    ; test right bit
JNZ RIGHT     ; if set
TEST AL, 128 ; test left bit
JNZ LEFT      ; if set
```

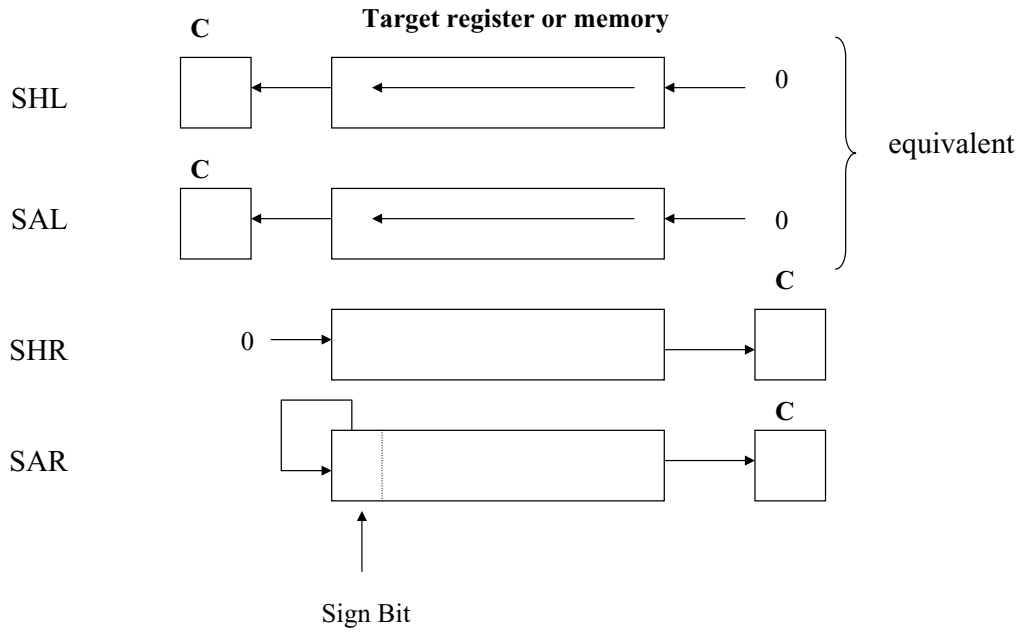
- Bit Test Instructions (80386 thru Pentium 2) BT/BTC/BTR/BTS
  - BT AX,4 tests bit position 4 in AX and result is located in the CF,
  - CF = 1 if bit 4 is 1, 0 otherwise

Compare Example

```
DATA1        DW     235Fh
…
MOV AX, CCCCH
CMP AX, DATA1
JNC OVER
SUB AX,AX
OVER: INC DATA1
```

CCCC – 235F = A96D => Z=0, CF=0 =>
CCCC > DATA1

Shift



Examples

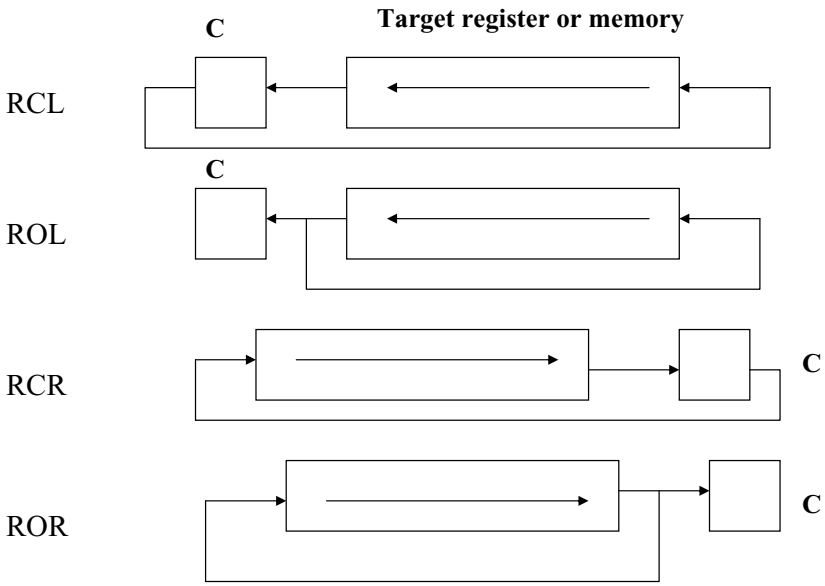
Examples
```
SHL AX,1
SAL DATA1, CL   ; shift count is a modulo-32 count
```

Ex.
```
; Multiply AX by 10
        SHL AX, 1
        MOV BX, AX
        MOV CL,2
        SHL AX,CL
        ADD AX, BX
```

Ex. What are the results of SAR CL, 1 if CL initially contains B6H?

Ex. What are the results of SHL AL, CL if AL contains 75H and CL contains 3?

Rotate



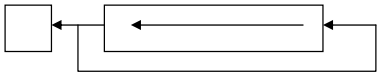
Ex. What is the result of ROL byte ptr [SI], 1 if this memory location 3C020 contains 41H?

What is the result of ROL word ptr [SI], 8 if this memory location 3C020 contains 4125H?

Example

Write a program that counts the number of 1's in a byte and writes it into BL

```
DATA1   DB 97              ; 61h
        SUB   BL,BL        ;clear BL to keep the number of 1s
        MOV    DL,8        ;rotate total of 8 times
        MOV    AL,DATA1
AGAIN:  ROL    AL,1        ;rotate it once
        JNC    NEXT        ;check for 1
        INC    BL          ;if CF=1 then add one to count
NEXT:   DEC    DL          ;go through this 8 times
        JNZ    AGAIN       ;if not finished go back
        NOP
```



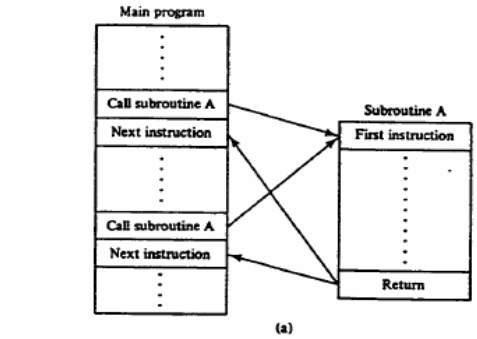
Subroutines and Subroutine Handling Functions

✓A subroutine is a special segment of a program that can be called for execution from any point in the program

✓A RET instruction must be included at the end of the subroutine to initiate the return sequence to the main program environment

Examples. **Call 1234h**
Call BX
Call [BX]

Two calls
•intrasegment
•intersegment



(a)

Mnemonic	Meaning	Format	Operation	Flags Affected
CALL	Subroutine call	CALL operand	Execution continues from the address of the subroutine specified by the operand. Information required to return back to the main program such as IP and CS are saved on the stack.	None

(b)

Operand
Near-proc
Far-proc
Memptr16
Regptr16
Memptr32

(c)

Figure 6–20 (a) Subroutine concept. (b) Subroutine call instruction. (c) Allowed operands.

Calling a NEAR proc

✓The CALL instruction and the subroutine it calls are in the same segment.

✓Save the current value of the IP on the stack.

✓load the subroutine's offset into IP (nextinst + offset)

Calling Program	Subroutine	Stack

```
Main proc          sub1 proc
001A: call sub1    0080: mov ax,1
001D: inc ax       …
.                  ret
Main endp          sub1 endp
```

1ffd	1D
1ffe	00
1fff	(not used)

Calling a FAR proc

✓The CALL instruction and the subroutine it calls are in the "Different" segments.

✓Save the current value of the CS and IP on the stack.

✓Then load the subroutine's CS and offset into IP.

Calling Program	Subroutine	Stack

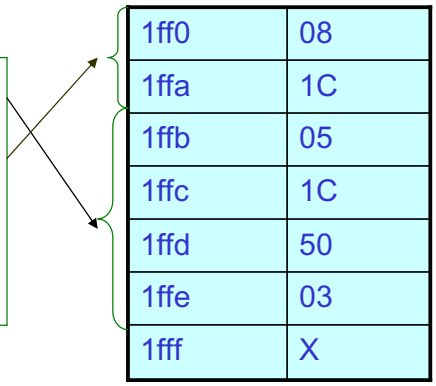
```
      Main proc                  sub1 proc far
1FCB:001A: call far ptr sub1 →   4EFA:0080: mov ax,1
1FCB:001F: inc ax                ….
      …                          ….
      …                          ret (retf opcode generated)
      Main endp                  sub1 endp
```

1ffb	1F	IP
1ffc	00	
1ffd	CB	SEG
1ffe	1F	
1fff	N/A	

Opcode 8000 FA4E

Example on Far/Near Procedure Calls

```
0350:1C00 Call FarProc
0350:1C05 Call NearProc
0350:1C08 nop
```



1ff0	08
1ffa	1C
1ffb	05
1ffc	1C
1ffd	50
1ffe	03
1fff	X

Nested Procedure Calls

A subroutine may itself call other subroutines.

Example:

```
                main proc
000A            call subr1
000C            mov ax,…
…
                main endp
```

```
                subr1  proc
0030            nop
                …
                call subr2
0040            ret …
                subr1 endp
```

```
                subr2  proc
0050            nop
                …
                call subr3
0060            ret …
                subr2 endp
```

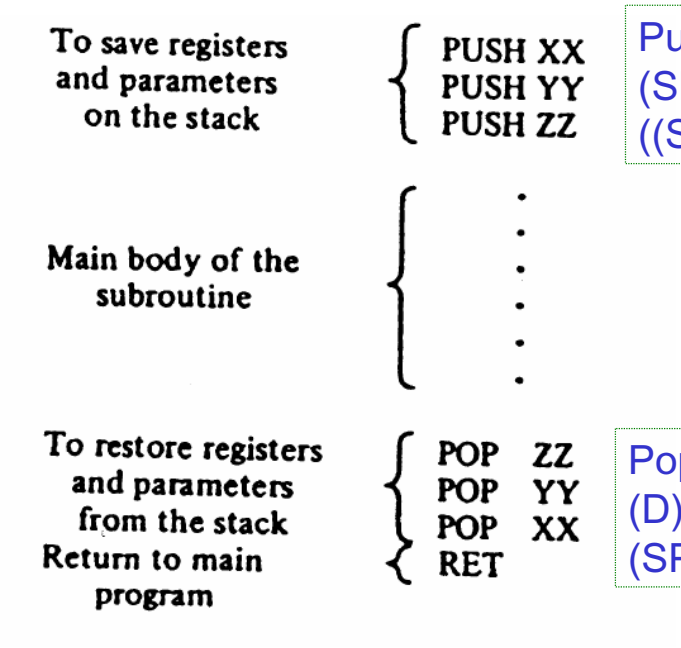
```
                subr3  proc
0070            nop
                …
0079            nop
007A            ret
                subr3 endp
```

Q: show the stack contents at 0079?

1ff0	60
1ffa	00
1ffb	40
1ffc	00
1ffd	0c
1ffe	00
1fff	X

Do NOT overlap Procedure Declarations

Push and Pop Instructions



Push S (16/32 bit or Mem)
(SP) ← (SP) - 2
((SP)) ← (S)

Pop D (16/32 bit or Mem)
(D) ← ((SP))
(SP) ← (SP) + 2

Week 6

8088/8086 Microprocessor Programming

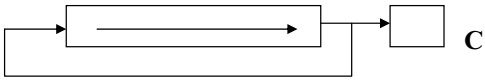
BCD and ASCII Numbers

- BCD (Binary Coded Decimal)
  - Unpacked BCD: One byte per digit
  - Packed BCD: 4 bits per digit (more efficient in storing data)
- ASCII to unpacked BCD conversion
  - Keyboards, printers, and monitors all use ASCII.
  - Digits 0 to 9 are represented by ASCII codes 30 – 39.

- Example. Write an 8086 program that displays the packed BCD number in register AL on the system video monitor
  - The first number to be displayed should be the MS Nibble
  - It is found by masking the LS Nibble and then rotating the MS Nibble into the LSD position
  - The result is then converted to ASCII by adding 30h
  - The BIOS video service is then called to display this result.

ASCII Numbers Example

```
MOV BL,AL; save
AND AL,F0H
MOV CL,4
ROR AL,CL
ADD AL,30H
MOV AH,0EH
INT 10H ;display single character

MOV AL,BL; use again
AND AL,0FH
ADD AL,30H
INT 10H
INT 20H          ; RETURN TO DOS
```



Example

- Write an 8086 program that adds two packed BCD numbers input from the keyboard and computes and displays the result on the system video monitor
- Data should be in the form 64+89= The answer 153 should appear in the next line.

#	?	6	4	+	8	9	=
0	1	2	3	4	5	6	7

Example Continued

```
Mov dx, offset bufferaddress
Mov ah,0a
Mov si,dx
Mov byte ptr [si], 6
Int 21
Mov ah,0eh
Mov al,0ah
Int 10
; BIOS service 0e line feed  position cursor

 sub byte ptr[si+2], 30h
 sub byte ptr[si+3], 30h
 sub byte ptr[si+5], 30h
 sub byte ptr[si+6], 30h
```

6	?	6	4	+	8	9	=
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**

```
Mov cl,4
Rol byte ptr [si+3],cl
Rol byte ptr [si+6],cl
Ror word ptr [si+5], cl
Ror word ptr [si+2], cl

 Mov al, [si+3]
 Add al, [si+6]
 Daa
 Mov bh,al
 Jnc display
 Mov al,1
 Call display
 Mov al,bh
 Call display
 Int 20
```


Flag Control Instructions

SF	ZF		AF		PF		CF

- **LAHF** Load AH from flags (AH) ← (Flags)
- **SAHF** Store AH into flags (Flags) ← (AH)
  - Flags affected: SF, ZF, AF, PF, CF

Bulk manipulation of the flags

- **CLC** Clear Carry Flag (CF) ← 0
- **STC** Set Carry Flag (CF) ← 1
- **CLI** Clear Interrupt Flag (IF) ← 0
- **STI** Set interrupt flag (IF) ← 1

Individual manipulation of the flags

- Example (try with debug)

```
LAHF
MOV AX,0000
ADD AX,00
SAHF
```

  - Check the flag changes!

Example. Binary (Hex) to ASCII (Decimal) Conversion

- CONVERSION PROCEDURE WITH AN EXAMPLE
- 34Dh = 3 x 256 + 4 x 16 + 13 x 1 = 845
- 34Dh / A = 84 remainder 5
- 84h / A = 8 remainder 4
- 8 < A the process stops
- Taking the remainders in reverse order gives : 845 decimal

❖Write a program to convert a word sized hex number in data item BINNUM

❖The result will be five digits, each digit will be converted to ASCII and placed in ASCNUM, the lowest digit will be in high memory as the convention of ASCII storage in DOS

Program

```
                BINNUM          DW 34Dh
                ASCNUM          DB 5 DUP('0')
                .CODE
                MOV     BX, 10
                MOV SI  OFFSET ASCNUM
                ADD     SI, 5
                DEC     SI
                MOV     AX,BINNUM
BACK:           SUB     DX, DX
                DIV     BX;   Dword division DX:AX / BX = AX ; rem DL
                OR      DL,30h
                MOV     [SI], DL
                DEC     SI
                CMP     AX,0
                JA      BACK
                INT     20h
```

Loop and Loop Handling Instructions

Mnemonic	Meaning	Format	Operation
LOOP	Loop	LOOP Short-label	(CX) ← (CX) − 1 Jump is initiated to location defined by short-label if (CX) ≠ 0; otherwise, execute next sequential instruction
LOOPE/LOOPZ	Loop while equal/ loop while zero	LOOPE/LOOPZ Short-label	(CX) ← (CX) − 1 Jump to location defined by short-label if (CX) ≠ 0 and (ZF) = 1; otherwise, execute next sequential instruction
LOOPNE/ LOOPNZ	Loop while not equal/ loop while not zero	LOOPNE/LOOPNZ Short-label	(CX) ← (CX) − 1 Jump to location defined by short-label if (CX) ≠ 0 and (ZF) = 0; otherwise, execute next sequential instruction

Figure 6–28 Loop instructions.

Loop

```
NEXT:   MOV CX,COUNT      Load count for the number of repeats
        .
        .
        .                 Body of routine that is repeated
        .
        .
        .
        LOOP NEXT         Loop back to label NEXT if count not zero
```

(a)

```
        MOV     AX,DATASEGADDR
        MOV     DS,AX
        MOV     SI,BLK1ADDR
        MOV     DI,BLK2ADDR
        MOV     CX,N
NXTPT:  MOV     AH,[SI]
        MOV     [DI],AH
        INC     SI
        INC     DI
        LOOP    NXTPT
        HLT
```

(b)

Nested Loops

single Loop

```
MOV CX,A
BACK: …
…
…
…
LOOP BACK
```

Nested Loops

```
        MOV CX,A
OUTER:  PUSH CX
        MOV CX, 99
INNER:  NOP
        …
        …
        …
        LOOP INNER
        POP CX
        LOOP OUTER
```

How many times will the loop execute, if JCXZ wasn't there

```
MOV CX,0
DLOOP: JCXZ SKIP ;guarding
BACK: MUL AX,2H
ADD AX,05H
LOOP BACK
SKIP: INC AX; if CX=0
```


INT

INT operates similar to Call

- ❖Processor first pushes the flags
- ❖Trace Flag and Interrupt-enable flags are cleared
- ❖Next the processor pushes the current CS register onto the stack
- ❖Next the IP register is pushed

Example: What is the sequence of events for INT 08? If it generates a CS:IP of 0100:0200. The flag is 0081H.

SP-6	00
SP-5	02
SP-4	00
SP-3	01
SP-2	81
SP-1	00

SP initial

MEMORY / ISR table	
00020	10
00021	00
00022	80
00023	05

IP

SEG

0580:
0010

IRET

- IRET must be used for special handling of the stack.
- Must be used at the end of an ISR

SP-6	00
SP-5	02
SP-4	00
SP-3	01
SP-2	81
SP-1	00

SP initial

Return address + flags are loaded

String Instructions

80x86 is equipped with special instructions to handle string operations
String: A series of data words (or bytes) that reside in consecutive memory locations
Operations: move, scan, compare

String Instruction:
Byte transfer, SI or DI increment or decrement by 1
Word transfer, SI or DI increment or decrement by 2
DWord transfer, SI or DI increment or decrement by 4

String Instructions - D Flag

The Direction Flag: Selects the auto increment D=0 or the auto decrement D=1 operation for the DI and SI registers during string operations. D is used only with strings

Mnemonic	Meaning	Format	Operation	Flags Affected
CLD	Clear DF	CLD	(DF) ← 0	DF
STD	Set DF	STD	(DF) ← 1	DF

CLD → Clears the D flag / STD → Sets the D flag

String Instructions

Mnemonic	Meaning	Format	Operation	Flags Affected
MOVS	Move string	MOVSB/MOVSW	((ES)0 + (DI)) ← ((DS)0 + (SI)) (SI) ← (SI) ± 1 or 2 (DI) ← (DI) ± 1 or 2	None
CMPS	Compare string	CMPSB/CMPSW	Set flags as per ((DS)0 + (SI)) − ((ES)0 + (DI)) (SI) ← (SI) ± 1 or 2 (DI) ← (DI) ± 1 or 2	CF, PF, AF, ZF, SF, OF
SCAS	Scan string	SCASB/SCASW	Set flags as per (AL or AX) − ((ES)0 + (DI)) (DI) ← (DI) ± 1 or 2	CF, PF, AF, ZF, SF, OF
LODS	Load string	LODSB/LODSW	(AL or AX) ← ((DS)0 + (SI)) (SI) ← (SI) ± 1 or 2	None
STOS	Store string	STOSB/STOSW	((ES)0 + (DI)) ← (AL or AX) ± 1 or 2 (DI) ← (DI) ± 1 or 2	None

```
        MOV     AX,DATASEGADDR
        MOV     DS,AX
        MOV     ES,AX
        MOV     SI,BLK1ADDR
        MOV     DI,BLK2ADDR
        MOV     CX,N
        CLD
NXTPT:  MOVSB
        LOOP    NXTPT
        HLT
```

Repeat String REP

Basic string operations must be repeated in order to process arrays of data; this is done by inserting a repeat prefix.

Prefix	Used with:	Meaning
REP	MOVS STOS	Repeat while not end of string CX ≠ 0
REPE/REPZ	CMPS SCAS	Repeat while not end of string and strings are equal CX ≠ 0 and ZF = 1
REPNE/REPNZ	CMPS SCAS	Repeat while not end of string and strings are not equal CX ≠ 0 and ZF = 0

Figure 6–36 Prefixes for use with the basic string operations.

Example. Find and replace

- Write a program that scans the name “Mr.Gohns” and replaces the “G” with the letter “J”.


search.asm

```
Data1 db  'Mr.Gones','$'
.code
mov es,ds
cld ;set auto increment bit D=0
mov di, offset data1
mov cx,09; number of chars to be scanned
mov al,'G'; char to be compared against
repne SCASB; start scan AL =? ES[DI]
jne Over; if Z=0
dec di; Z=1
mov byte ptr[di], 'J'
Over:  mov ah,09
mov dx,offset data1
int 21h; display the resulting String
```


Search.exe

Strings into Video Buffer

Fill the Video Screen with a value


```
CLD
MOV AX,0B800H
MOV ES,AX
MOV DI,0
MOV CX,2000H
MOV AL,20h
REP STOSW
```


Example. Display the ROM BIOS Date

- Write an 8086 program that searches the BIOS ROM for its creation date and displays that date on the monitor.
- If a date cannot be found display the message “date not found”
- Typically the BIOS ROM date is stored in the form xx/xx/xx beginning at system address F000:FFF5
- Each character is in ASCII form and the entire string is terminated with the null character (00)
- First we scan the null character, if the null character is found in the first eight characters, an invalid date is assumed
- If not, the entire string is copied into RAM
- Add a ‘$’ character to the end of the string and make it ready for DOS function 09, INT 21

I. Locate Date String in ROM

A. Save Es
B. Search for null byte
   1. Point ES:DI to F000:FFF5
   2. Direction flag = auto increment
   3. 12 bytes to scan
   4. Scan for null character
   5. Null Found?
      i. No: Goto II.E
      ii. Yes: Continue
C. Compute length of string and save in CX
   1. Subtract FFF5 from address of null character
D. Make sure string is the correct format.
   1. Found null at character 1-8?
      i. Yes: Goto II.E
      ii. No: Continue

II. Display Date String

A. Add date string to message header
   1. Recover ES
   2. Save DS
   3. Point DS:SI to F000:FFF5
   4. Point ES:DI to end of message header
   5. Copy date string to end of header

B. Recover DS and append end of message ($) to the full string
C. Display the complete message (header string plus date string)
   1. Point DS:DX to start of header
   2. DOS INT 21, function 9
D. Goto III.A

E. Display date not found message
   1. Recover DS
   2. Point DS:DX to date not found string
   3. DOS INT 21, function 9

III. Return to DOS

A. INT 20H

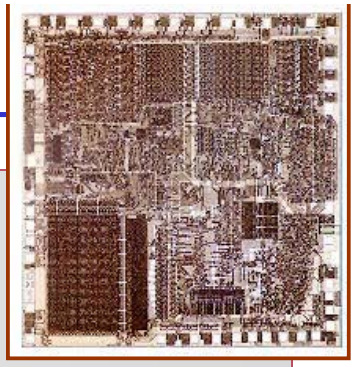
IV. Set up Messages

A. Header: "Your ROM BIOS is dated:"
B. Date: "xx/xx/xxxx"
C. Not found: "Date not found"

```
        push    es

        mov     ax,f000
        mov     es,ax
        mov     di,fff5
        cld
        mov     cx,0c
        mov     al,0
repne   scasb
        jnz     II.E
        mov     cx,di
        sub     cx,fff5
        cmp     cx,9
        jb      II.E

        pop     es
        push    ds
        mov     ax,f000
        mov     ds,ax
        mov     si,fff5
        mov     di,start_of_date_string
rep     movsb
        pop     ds
        mov     [di],'$'

        mov     dx,start_of_header_string
        mov     ah,9
        int     21
        jmp     III.A
II.E    pop     ds
        mov     dx,start_of_not_found_string
        mov     ah,9
        int     21

III.A   int     20


start_of_header:
        db      "Your ROM BIOS is dated"
start_of_date_string:
        db      "xx/xx/xxxx"
```

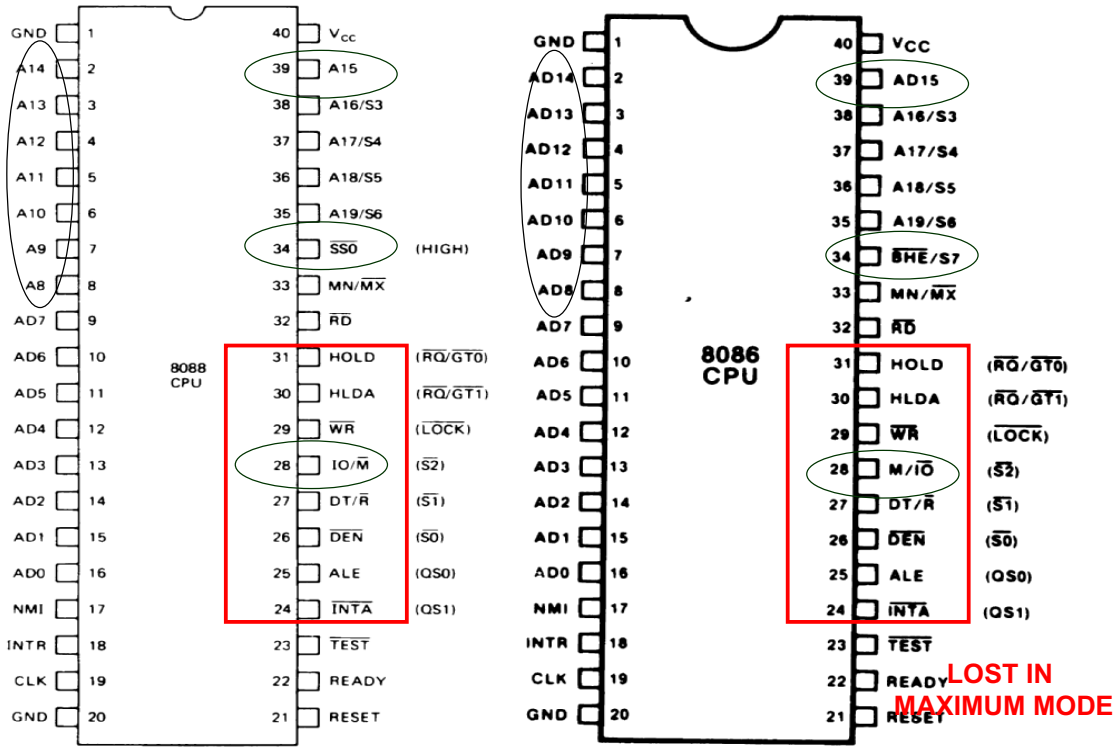
Weeks 7-8
The 8088/8086 Microprocessor Architecture and Memory Interfacing

8086 and 8088 Microprocessors



- 8086 announced in 1978; 8086 is a 16 bit microprocessor with a 16 bit data bus
- 8088 announced in 1979; 8088 is a 16 bit microprocessor with an 8 bit data bus
- Both manufactured using High-performance Metal Oxide Semiconductor (HMOS) technology
- Both contain about 29000 transistors
- Both are packaged in 40 pin dual-in-line package (DIP)
- Address lines A0-A7 and Data lines D0-D7 are multiplexed in 8088. These lines are labelled as AD0-AD7.
  - By multiplexed we mean that the same pysical pin carries an address bit at one time and the data bit another time
- Address lines A0-A15 and Data lines D0-D15 are multiplexed in 8086. These lines are labelled as AD0-AD15.

8088 and 8086 Microprocessors



Minimum-mode and Maximum-mode Systems

- 8088 and 8086 microprocessors can be configured to work in either of the two modes: the minimum mode and the maximum mode

✓ Minimum mode:

- Pull MN/$\overline{MX}$ to logic 1
- Typically smaller systems and contains a single microprocessor
- Cheaper since all control signals for memory and I/O are generated by the microprocessor.

✓ Maximum mode

- Pull MN/**$\overline{MX}$** logic 0
- Larger systems with more than one processor (*designed to be used when a coprocessor (8087) exists in the system)*

Minimum-mode and Maximum-mode Systems

Signals common to both minimum and maximum systems:

Common signals		
Name	Function	Type
AD7–AD0	Address/data bus	Bidirectional, 3-state
A15–A8	Address bus	Output, 3-state
A19/S6–A16/S3	Address/status	Output, 3-state
MN/$\overline{MX}$	Minimum/maximum Mode control	Input
$\overline{RD}$	Read control	Output, 3-state
$\overline{TEST}$	Wait on test control	Input
READY	Wait state control	Input
RESET	System reset	Input
NMI	Nonmaskable Interrupt request	Input
INTR	Interrupt request	Input
CLK	System clock	Input
V_{CC}	+5 V	Input
GND	Ground	

•The following pins are lost when the 8086 operates in
Maximum mode
•ALE
•WR
•IO/ M
•DT/ R
•DEN
•INTA

Minimum-mode Systems

Minimum mode signals (MN/$\overline{MX}$ = V_{CC})		
Name	Function	Type
HOLD	Hold request	Input
HLDA	Hold acknowledge	Output
$\overline{WR}$	Write control	Output, 3-state
IO/$\overline{M}$	IO/memory control	Output, 3-state
DT/$\overline{R}$	Data transmit/receive	Output, 3-state
$\overline{DEN}$	Data enable	Output, 3-state
$\overline{SSO}$	Status line	Output, 3-state
ALE	Address latch enable	Output
$\overline{INTA}$	Interrupt acknowledge	Output

Minimum mode unique signals

8086/8088 Pinout

- *Pin functions:*
- *AD15-AD0*

  Multiplexed address(ALE=1)/data bus(ALE=0).
- *A19/S6-A16/S3 (multiplexed)*

  High order 4 bits of the 20-bit address OR status bits S6-S3.
- *$\overline{M}$/IO*

  Indicates if address is a Memory or IO address.
- $\overline{RD}$

  When 0, data bus is driven by memory or an I/O device.
- *$\overline{WR}$*

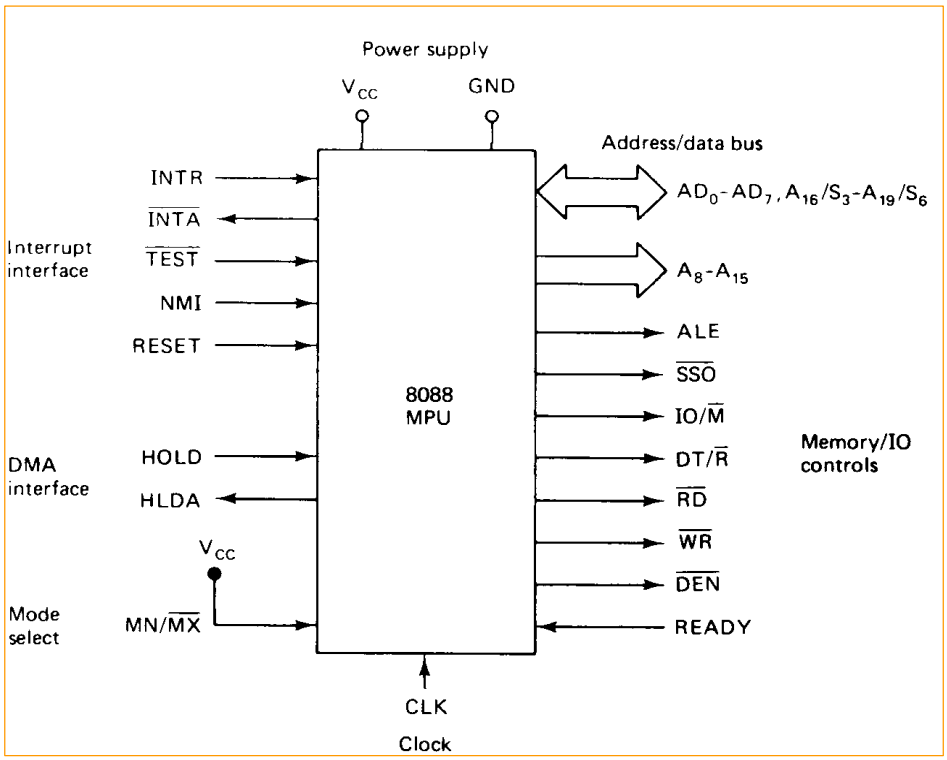
  Microprocessor is driving data bus to memory or an I/O device. When 0, data bus contains valid data.
- *ALE (Address latch enable)*

  When 1, address data bus contains a memory or I/O address.
- *DT/$\overline{R}$ (Data Transmit/Receive)*

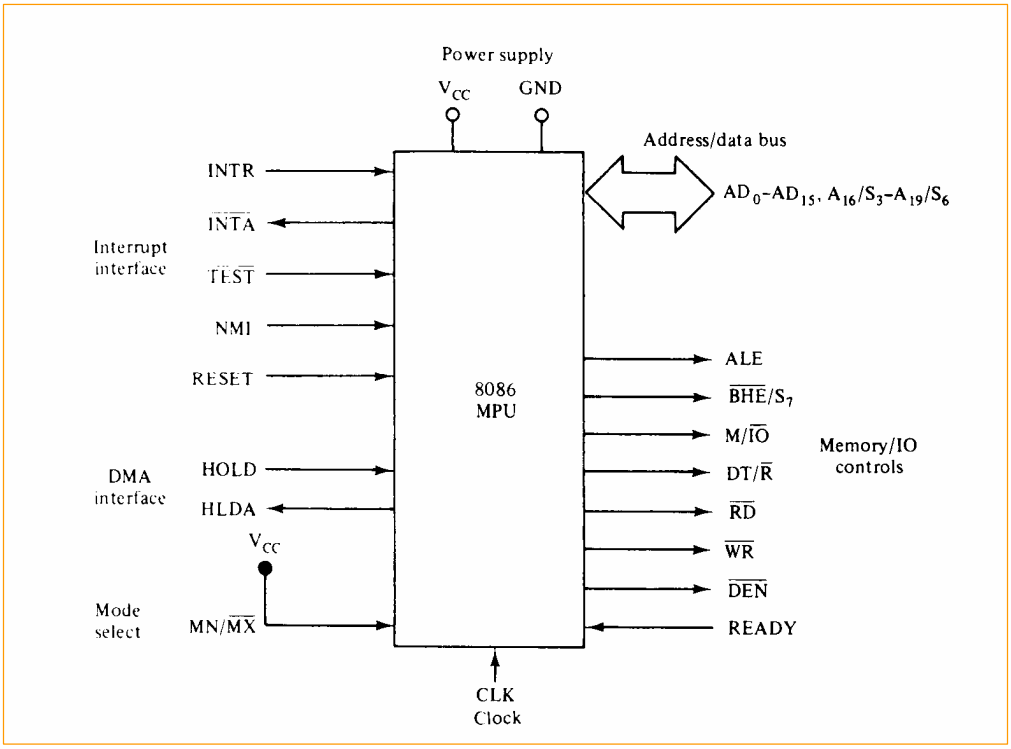
  Data bus is transmitting/receiving data.
- *$\overline{DEN}$ (Data bus Enable)*

  Activates external data bus buffers.

8088 Minimum-mode block diagram



8086 Minimum-mode block diagram

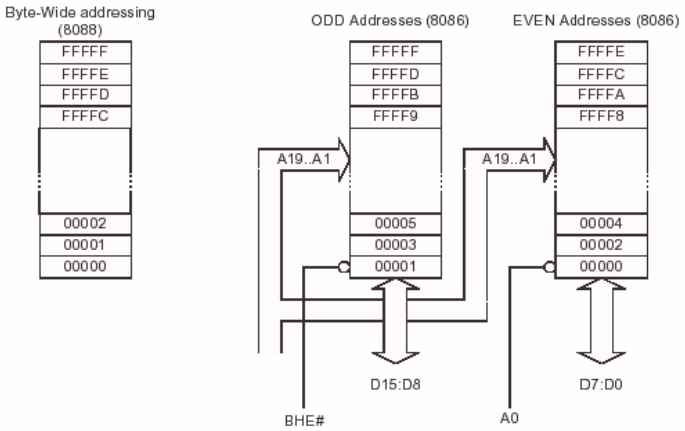


Minimum Mode Interface

✓**$\overline{BHE}$ (Bank High Enable) line** : =0 for most significant byte of data (8086 only) and also carries S_7=1

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

This is how memory is accessed using these signals:



Bus Cycle and Time States

- A bus cycle defines the basic operation that a microprocessor performs to communicate with external devices
- Examples of bus cycles are memory read, memory write, input/output read and input/output write.
- A bus cycle corresponds to a sequence of events that starts with an address being output on the system bus followed by a read or write data transfer
- During these operations, a series of control signals are also produced by the MPU to control the direction and timing of the bus.
- Each bus cycle consists of at least four clock periods, T1, T2, T3, and T4.
- These clock periods are also called the T-States

Bus Cycle and Time States

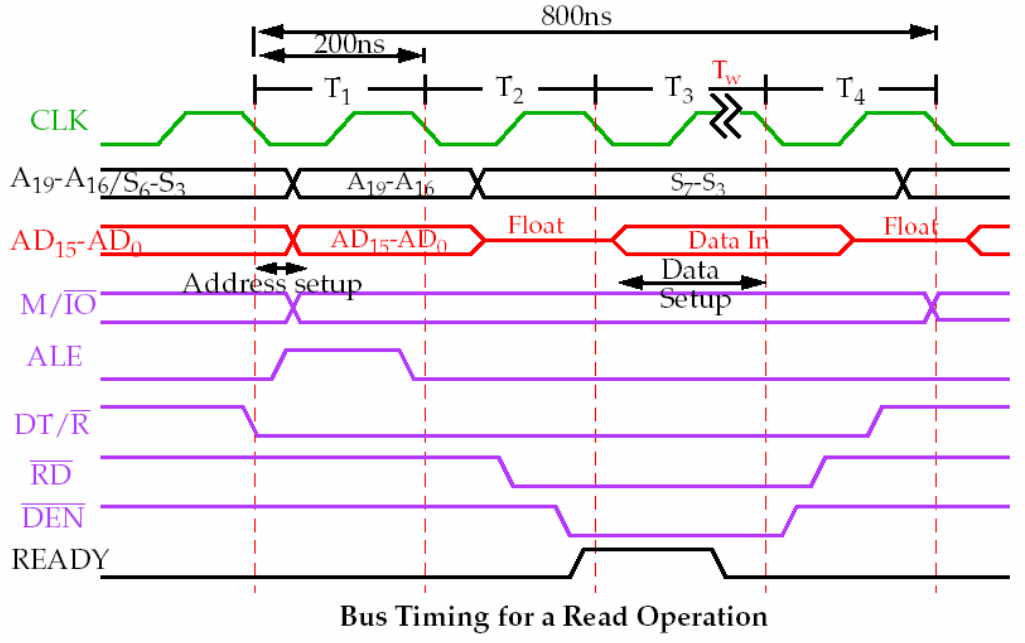
T1 - start of bus cycle. Actions include setting control signals (or S0-S2 status lines) to give the required values for ALE, DT/R,IO/M putting a valid address onto the address bus.

T2 - **the RD or WR control signals are issued, DEN is asserted and in the case of a write, data is put onto the data bus. The DEN turns on the data bus buffers to connect the CPU to the external data bus. The READY input to the CPU is sampled at the end of T2 and if READY is low, a wait state T_W (one or more) is inserted before T3 begins.**

T3 - this clock period is provided to allow memory to access the data. If the bus cycle is a read cycle, the data bus is sampled at the end of T3.

T4 - **all bus signals are deactivated in preparation for the next clock cycle. The 8088 also finishes sampling the data (in a read cycle) in this period. For the write cycle, the trailing edge of the WR signal transfers data to the memory or I/O, which activates and write when WR returns to logic 1 level.**

Read Cycle of the 8088

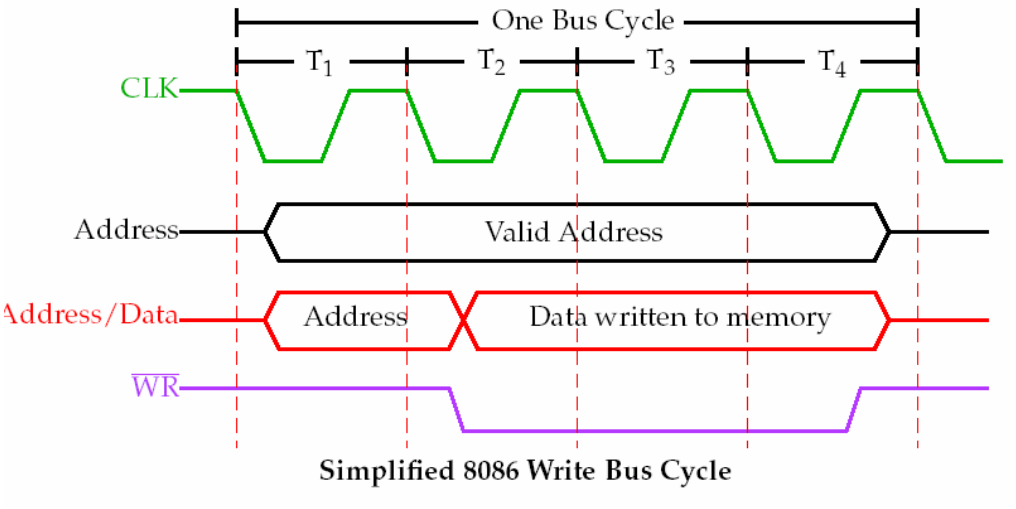


Bus Timing for a Read Operation

Read Cycle

- Each BUS CYCLE on the 8086 equals **four** system clocking periods (T states).
- The clock rate is **5MHz**, therefore one Bus Cycle is *800ns*.
- The transfer rate is **1.25MHz**.
- Memory specs (memory access time) must match constraints of system timing.
- For example, bus timing for a read operation shows almost *600ns* are needed to read data.
- However, memory must access faster due to setup times, e.g.
- Address setup and data setup.
- This subtracts off about *150ns*.
- Therefore, memory must access in at least *450ns* minus another *30-40ns* guard band for buffers and decoders.
- *420ns* DRAM required for the 8086.

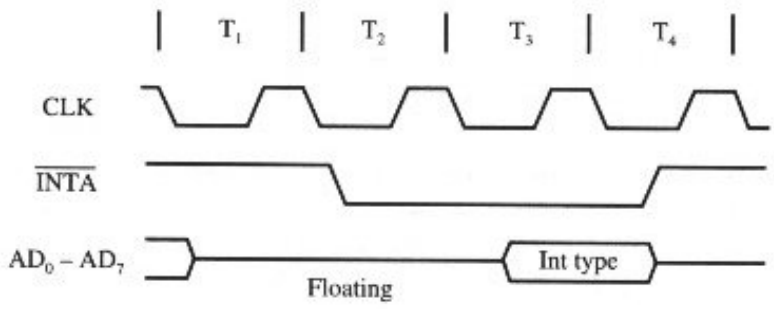
Write Cycle in 8088/8086 Minmode



Simplified 8086 Write Bus Cycle

Minimum Mode Interface

- Interrupt signals:
  - **INTR (Interrupt request)** :=1 shows there is a service request, sampled at the final clock cycle of each instruction acquisition cycle.
  - **INTA** : @T1 tri-states the Address Bus. Processor responds with two pulses going to 0 when it services the interrupt and waits for the interrupt service number after the second pulse.



(a) Minmode

Interrupt Signals cont'd

Interrupt	Logic	Disabled by SW?	Priority
NMI	Rising Edge	No	High
INTR	High	Yes	Low

– **NMI (Nonmaskable interrupt) :** A leading edge transition causes the processor go to the interrupt routine after the current instruction is executed.

– **$\overline{\text{TEST}}$**: Processor suspends operation when =1. Resumes operation when=0. Used to syncronize the processor to external events. (All 8087-capable compilers and assemblers automatically generate a WAIT instruction before each coprocessor instruction. The WAIT instruction tests the CPU's TEST pin and suspends execution until its input becomes "LOW". In all 8086/8087 systems, the 8086 TEST pin is connected to the 8087 BUSY pin. As long as the NEU executes a coprocessor instruction, it forces its BUSY pin "HIGH"; thus, the WAIT opcode preceding the coprocessor instruction stops the CPU until any still-executing coprocessor instruction has finished)

– **RESET** : =0. Need at least 4 clock cycles. Issuing reset causes the processor to fetch the first instruction from the memory FFFF0h.

Minimum Mode Interface

- DMA (Direct Memory Access) Interface Signals:
  - HOLD: External device puts logic level 1 to HOLD input to take control of the bus for DMA request. (sampled at every rising edge of the CLK)
  - $\overline{\text{HLDA}}$ (Hold acknowledge) : Processor responds by putting logic level 1 to HLDA. (at the end of T4)
  - In this state; Address and Data lines, SSO, IO/$\overline{\text{M}}$, DT/$\overline{\text{R}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{DEN}}$ signals are all put to high-Z state
  - RO/GT1 *and* RO/GT0: Request/grant pins request/grant direct memory accesses (DMA) during maximum mode operation.

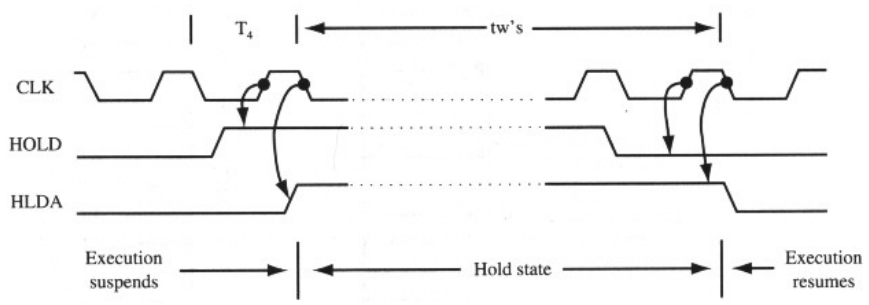


FIGURE 10.14 HOLD timing

Minimum Mode Interface

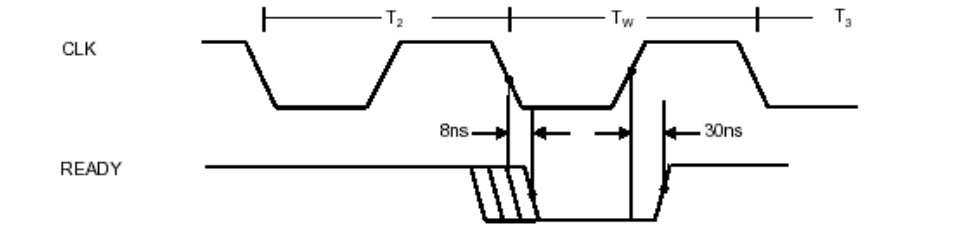
- READY Control line:
  - –can be used to insert wait states into the bus cycle so that it is extended by a number of clock periods.

✓If the access time for a memory device is longer than the memory access time calculated, need to give extra clock periods, wait state Tw, for memory.

✓The READY input is sampled at the end of T2 and again, if applicable, in the middle of Tw. If READY is a logic 0 on 1-to-0 clock transition, then Tw is inserted between T2 and T3. And will check for logic 1 on 0-to-1 clock transition in the middle of Tw to see if it shall go back T3.

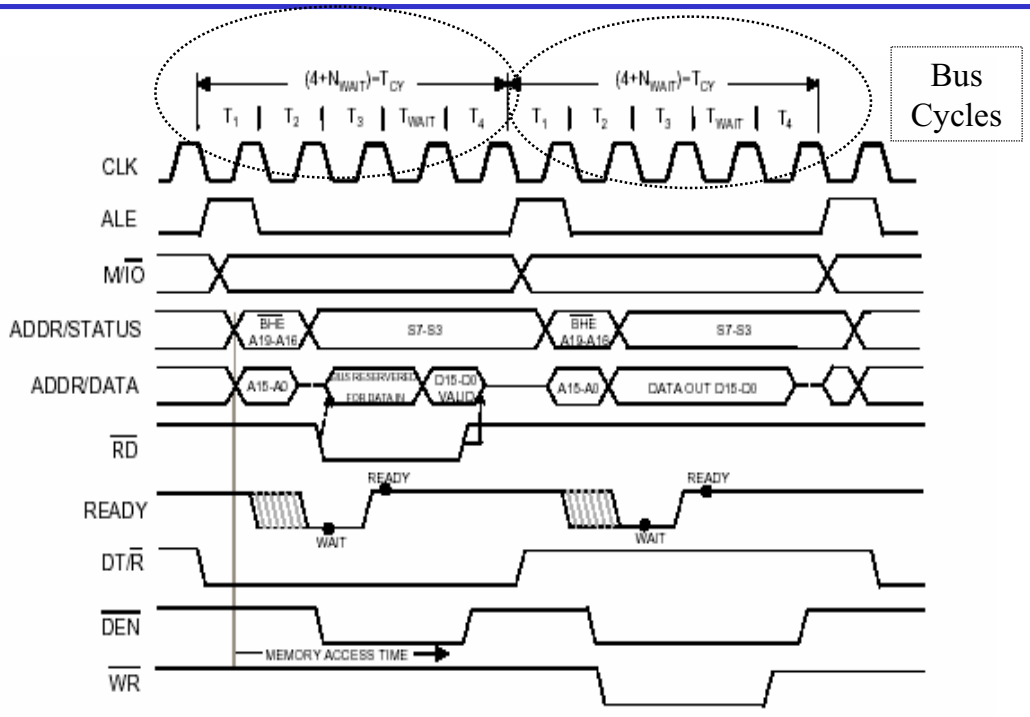
✓During the wait state, signals on the buses remain the same as they were at the start of the WAIT state.

✓By having the WAIT state, slow memory and devices has at least one more cycle (200ns for 5 MHz 8088) to get its data output.



(a) 8088/86 READY Input Timing

General Bus Cycle and Time States



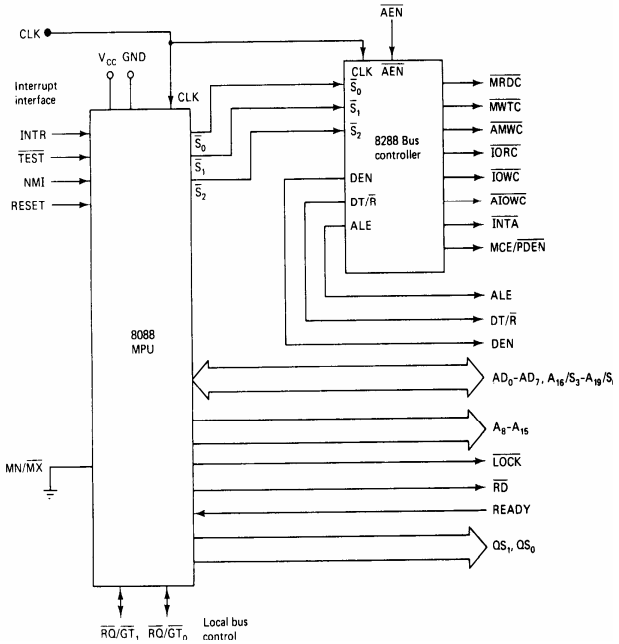
Maximum-mode Systems

Maximum mode signals (MN/$\overline{MX}$ = GND)		
Name	**Function**	**Type**
$\overline{RQ}/\overline{GT1}$, 0	Request/grant bus access control	Bidirectional
$\overline{LOCK}$	Bus priority lock control	Output, 3-state
$\overline{S2}$-$\overline{S0}$	Bus cycle status	Output, 3-state
QS1, QS0	Instruction queue status	Output

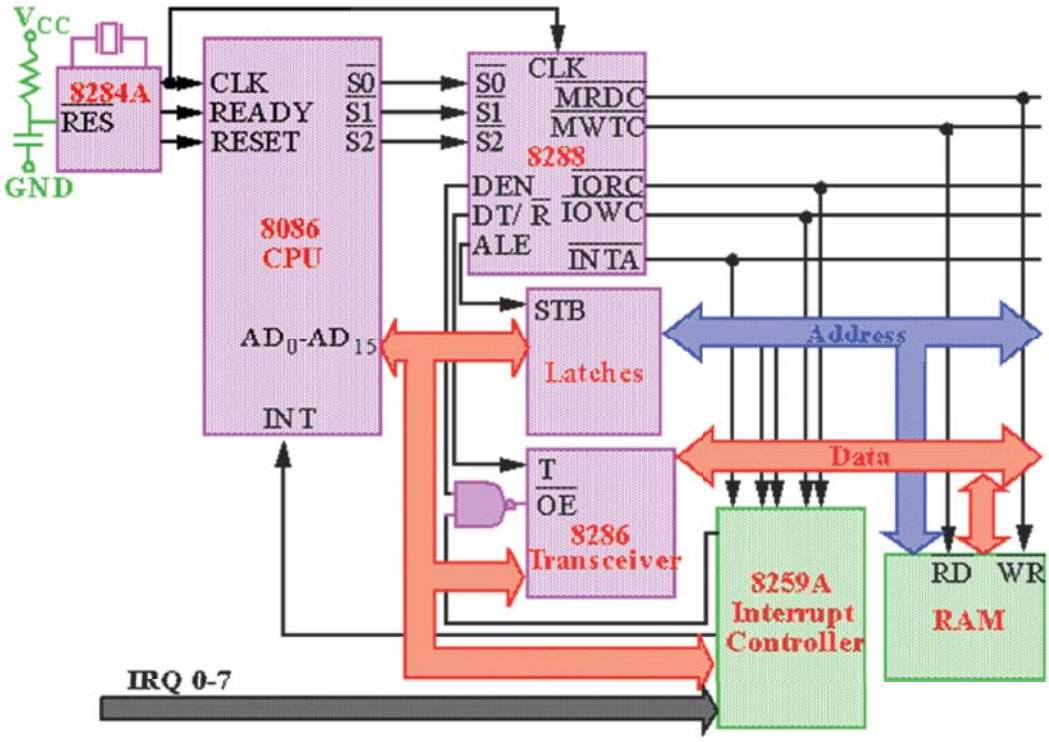
Maximum mode unique signals

Maximum Mode Interface

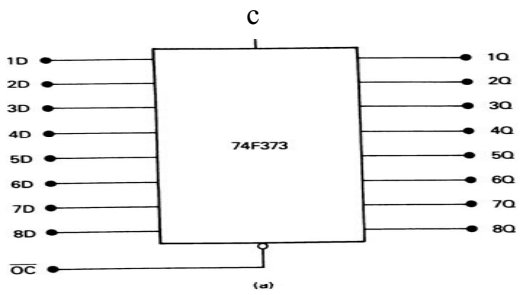
- Used in a multiprocessor environment
- 8288 Bus Controller is used for bus control
- $\overline{WR}$, IO/$\overline{M}$, DT/$\overline{R}$, $\overline{DEN}$, ALE, $\overline{INTA}$ signals are not readily available

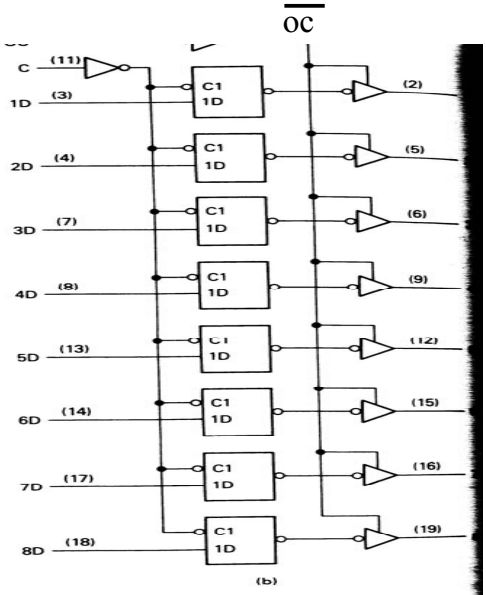


8086 Max Mode Interface



Address Bus Latches and Buffers

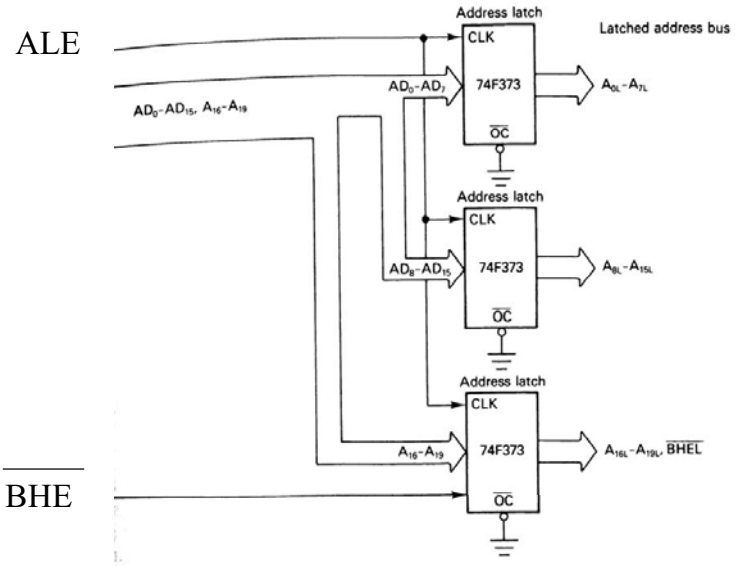




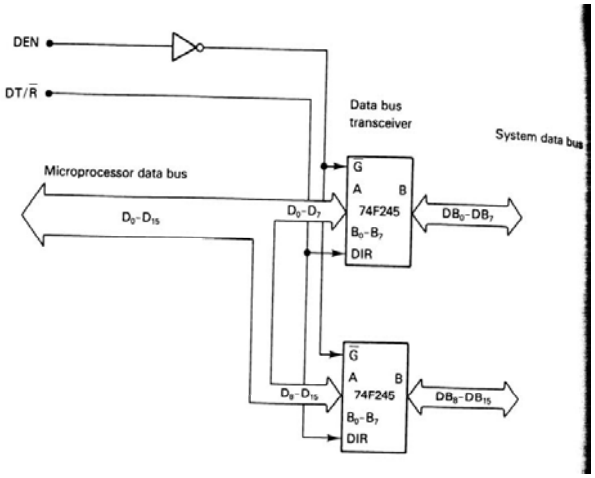
Inputs			Output
$\overline{OC}$	Enable C	D	Q
L	H	H	H
L	H	L	L
L	L	X	Q_0
H	X	X	Z

(c)

Address Latch Circuit



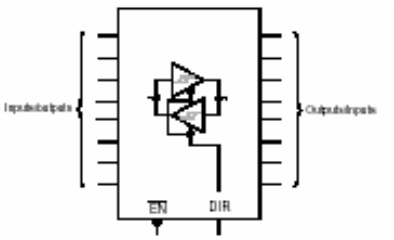
Data Bus Transceiver



G (ACTIVE LOW) ENABLE	DIR	OPERATION
L	L	B data to A
L	H	A data to B
H	x	Isolation

Buffered Systems

- Buffering (boosting) of the control, data, and address busses to provide sufficiently strong signals to drive various IC chips
  - When a pulse leaves an IC chip it can lose some of its strength depending on how far away the receiving IC is located
  - Plus the more pins a signal is connected to (i.e., fanout) the stronger the signal must be to drive them all which requires bus buffering
  - bus buffering = boosting the signals travelling on the busses
  - unidirectional buffer 74LS244
  - bidirectional buffer 74LS245



8088 System

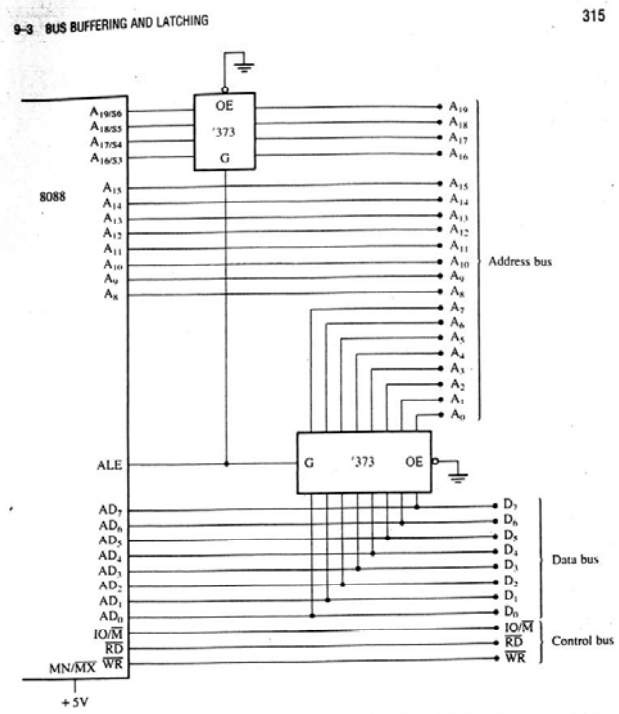
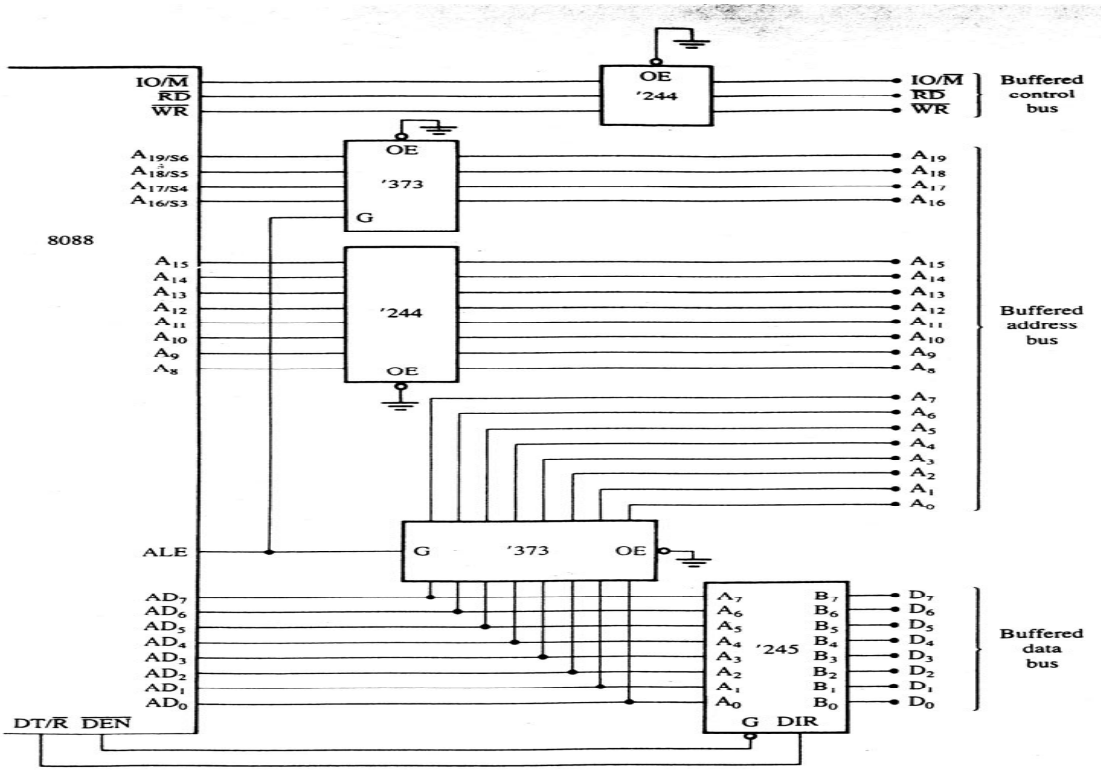
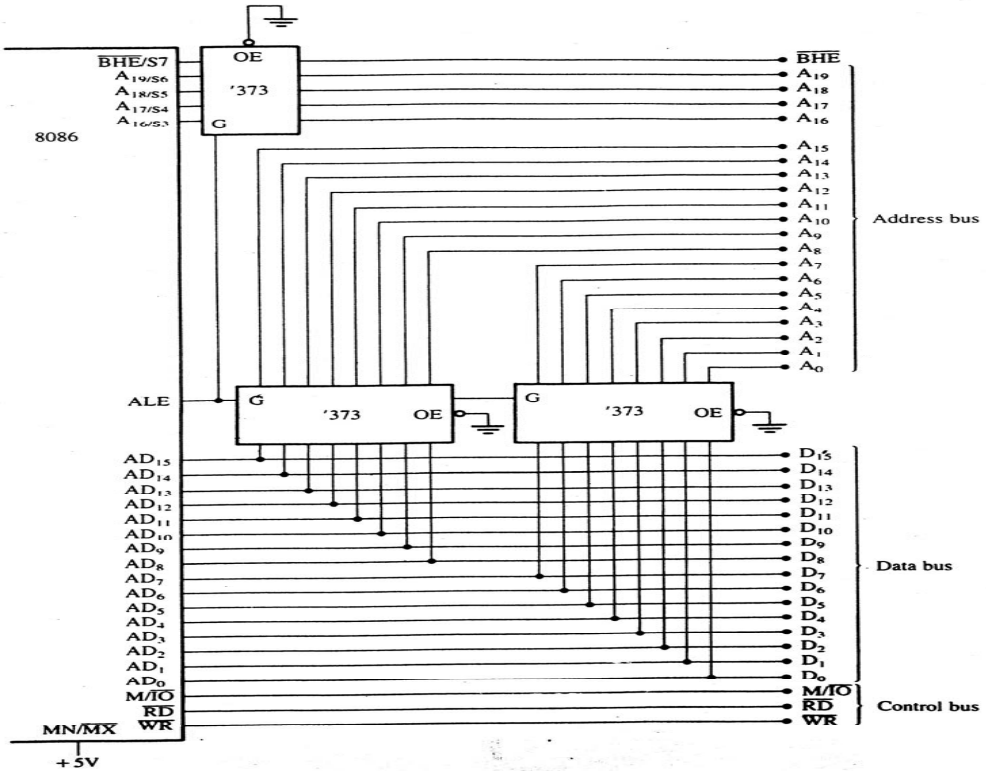


FIGURE 9–5 The 8088 microprocessor shown with a demultiplexed address bus. This is the model used to build many 8088-based systems.

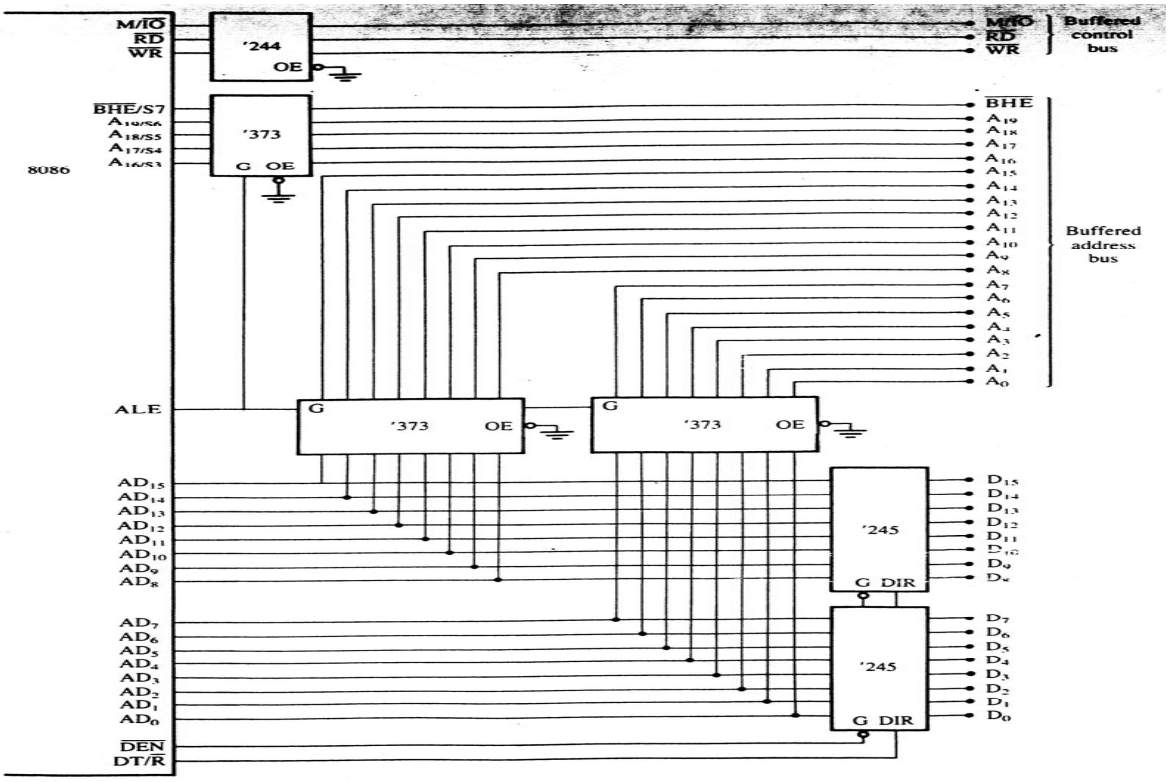
Fully buffered 8088



8086 System



Fully Buffered 8086



Semiconductor Memory Fundamentals

- In the design of all computers, semiconductor memories are used as primary storage for data and code.
- They are connected directly to the CPU and they are the memory that the CPU asks for information (code or data)
- Among the most widely used are RAM and ROM
- Memory Capacity
  - The number of bits that a semiconductor memory chip can store is called its chip capacity (bits or bytes)
- Memory Organization
  - Each memory chip contains 2^x locations where x is the number of address pins on the chip
  - Each location contains y bits, where y is the number of data pins on the chip
  - The entire chip will contain 2^x * y bits
  - Ex. Memory organization of 4K x 4: 2^{12} = 4096 locations, each location holding 4 bits
- Memory Speed (access time)

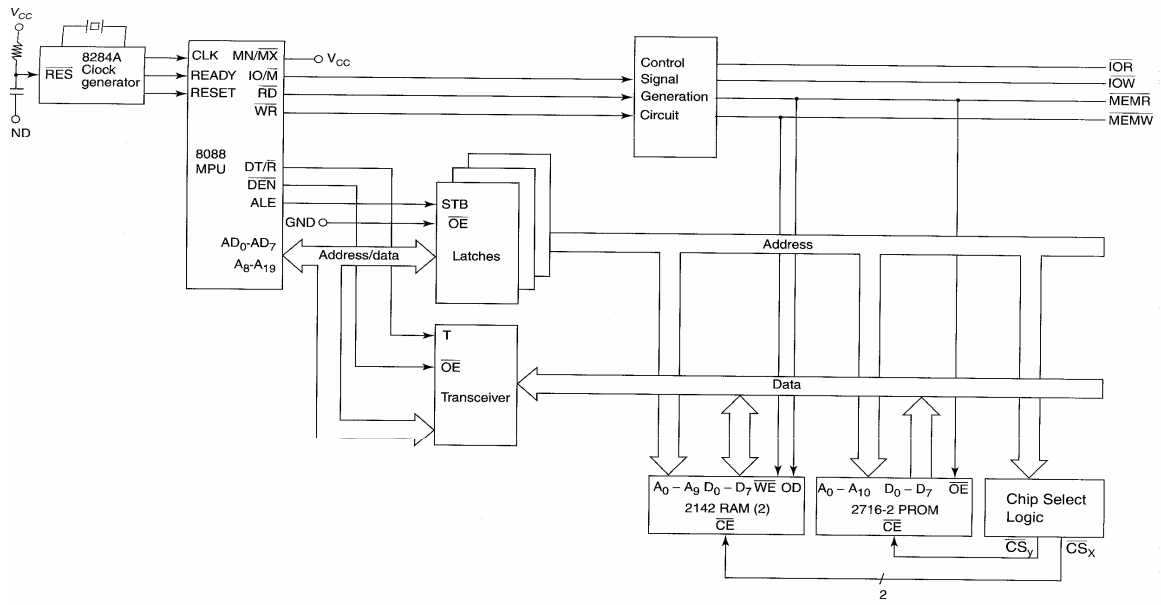
Memory Types

- ROM (Read Only Memory
  - ❖ ROM is the type of memory that does not lose its contents when power is turned off. It is also called nonvolatile memory.
  - ❖PROM (Programmable Memory)
    - ➢User programmable (one-time programmable) memory
    - ➢If the information burned into PROM is wrong, it needs to be discarded since internal fuses are blown permanently.
    - ➢Special equipment needed: ROM burner or ROM programmer
  - ❖EPROM (Erasable Programmable ROM) 2,000 times
    - ➢Allows making changes in the contents of PROM after it is burned
    - ➢One can program the memory chip and erase it thousands of times
    - ➢Erasing its contents can take up to 20 minutes; the entire chip is erased
    - ➢All EPROM chips have a window that is used to shine ultraviolet (UV) radiation to erase its contents
    - ➢Also referred to as UV-EPROM

Memory Types

- ❖ EEPROM (Electrically Erasable ROM) 500,000 times
  - ➢ Method of erasure is electrical
  - ➢ Moreover, one can select which byte to be erased
  - ➢ Cost per bit is much higher than for UV-EPROM
- ❖ Flash Memory EPROM
  - ➢ First, the process of erasure of the entire contents takes less than a second, or one might say in a flash, hence its name: flash memory
  - ➢ When flash memory's contents are erased, the entire device is erased.
  - ➢ Even though flash memories are writeable, like EPROMs they find their widest use in microcomputer systems for storage of firmware
- ❖ RAM (Random Access Memory) infinite times
  - ➢ RAM memory is called volatile memory since cutting off the power to the IC will mean the loss of data.
  - ➢ Also referred to as R/WM (Read And Write Memory)

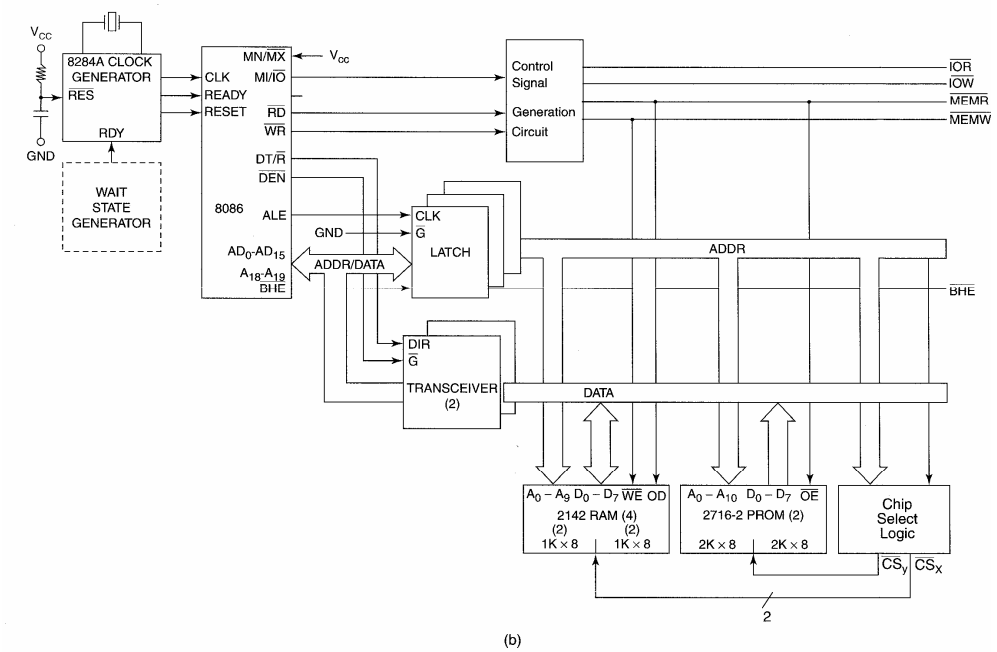
Minmode 8088 Microcomputer system memory circuitry



(a)

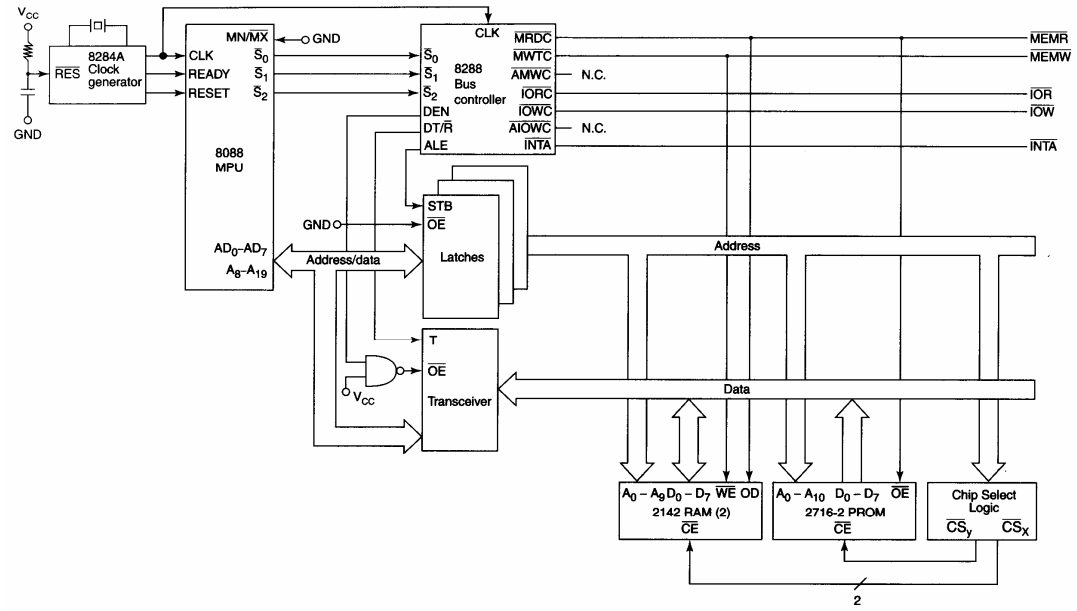
Figure 9–37 (a) Minimum-mode 8088 system memory interface. (Reprinted with permission of Intel Corporation, Copyright/Intel Corp. 1981) (b) Minimum-mode 8086 system memory interface. (Reprinted with permission of Intel Corporation, Copyright/Intel Corp. 1979) (c) Maximum-mode 8088 system memory interface. (Reprinted with permission of Intel Corporation, Copyright/Intel Corp. 1981)

Minmode 8086 Microcomputer system memory circuitry



(b)

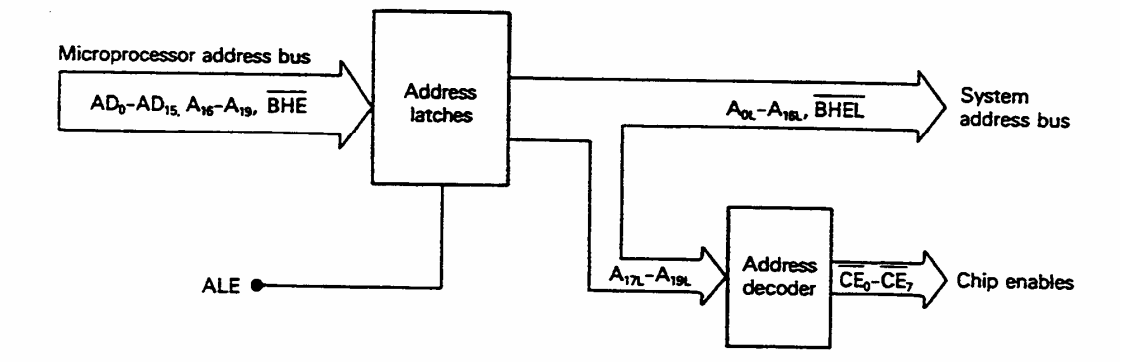
Maxmode 8088 Microcomputer system memory circuitry



(c)

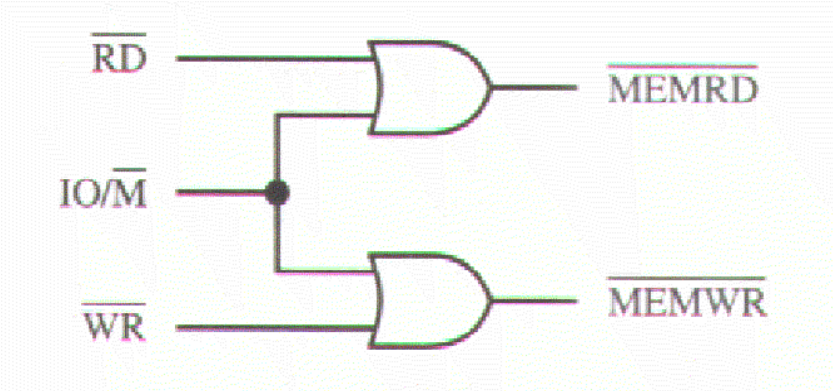
Figure 9–37 (continued)

Address Bus Configuration with Address Decoding



Generation of MEMRD & MEMWR in Minmode

Control Signal Generation Circuit



8088 Memory and I/O address spaces

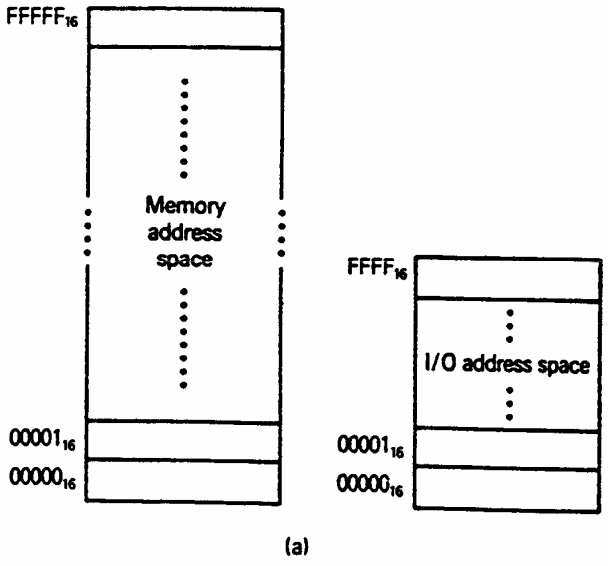
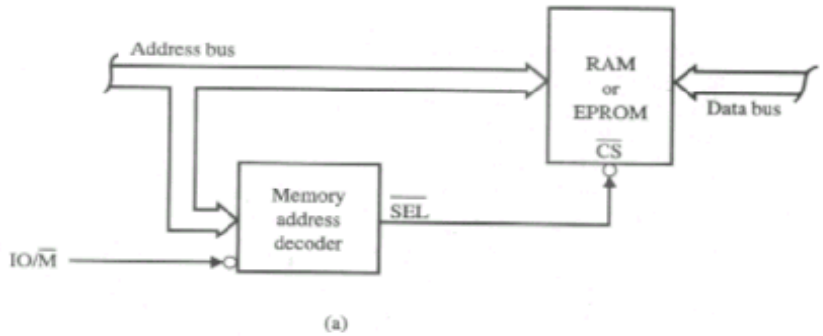


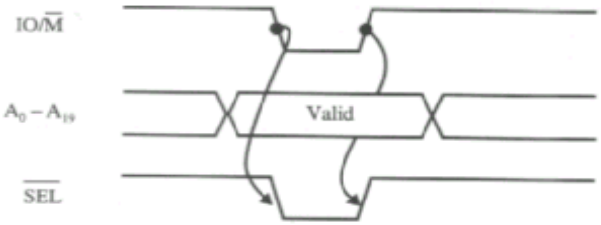
Figure 8–46 8088/8086 memory and I/O address spaces.

We first look at the memory addressing

Address Selection

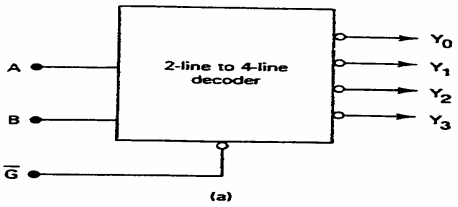


(a)

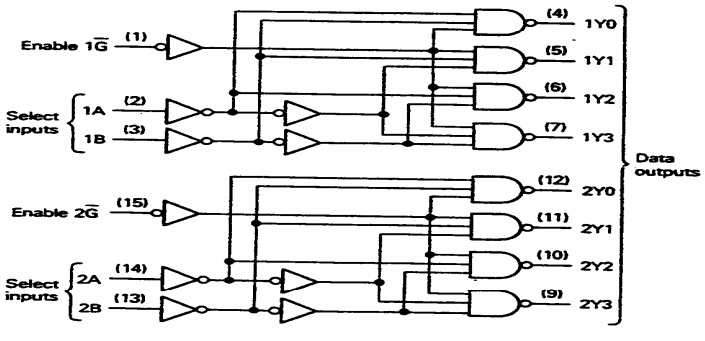


(b)

74F139 2-line to 4-line decoder



(a)

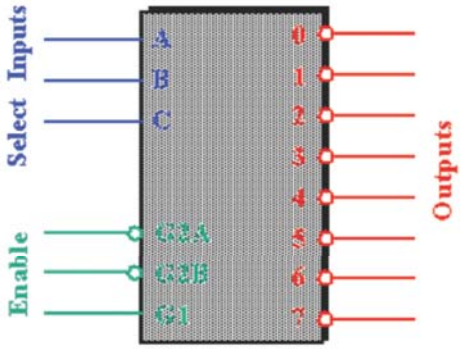


(b)

INPUTS			OUTPUTS			
ENABLE	SELECT					
$\overline{G}$	B	A	Y0	Y1	Y2	Y3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

(c)

Memory Address Decoding



Inputs						Output							
Enable			Select										
G2A	G2B	G1	C	B	A	0	1	2	3	4	5	6	7
1	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	0	X	X	X	1	1	1	1	1	1	1	1
0	0	1	0	0	0	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0	1	1	1	1	1	1
0	0	1	0	1	0	1	1	0	1	1	1	1	1
0	0	1	0	1	1	1	1	1	0	1	1	1	1
0	0	1	1	0	0	1	1	1	1	0	1	1	1
0	0	1	1	0	1	1	1	1	1	1	0	1	1
0	0	1	1	1	0	1	1	1	1	1	1	0	1
0	0	1	1	1	1	1	1	1	1	1	1	1	0

Example on Address Decoding

A circuit containing 32KB of RAM is to be interfaced to an 8088 based system, so that the first address of the RAM is at 48000H. What is the entire range of the RAM Address? How is the address bus used to enable the RAMs? What address lines should be used?

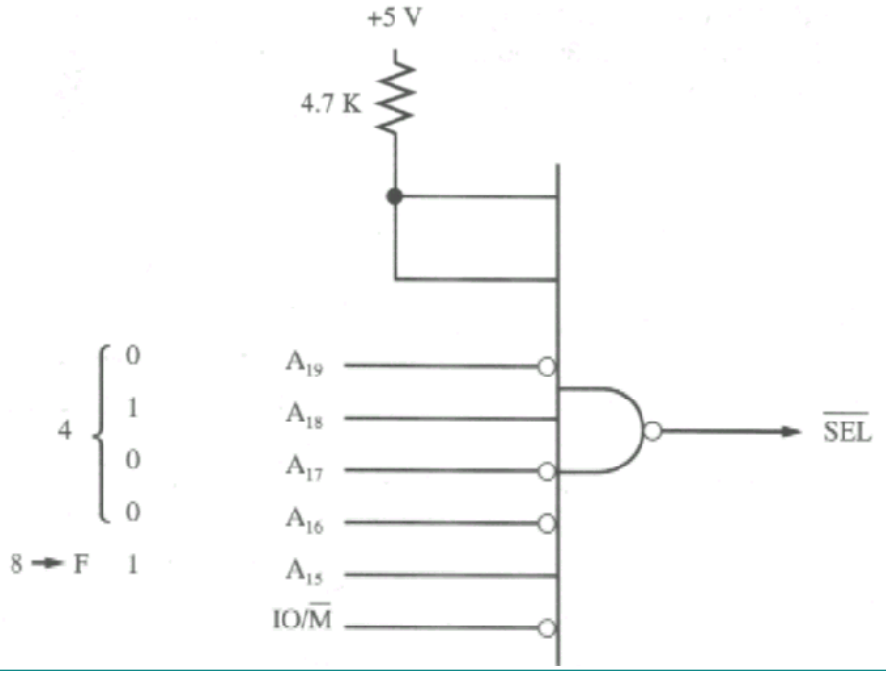
4	8 → F	0 → F	0 → F	0 → F
A_{19} A_{18} A_{17} A_{16}	A_{15} A_{14} A_{13} A_{12}	A_{11} A_{10} A_9 A_8	A_7 A_6 A_5 A_4	A_3 A_2 A_1 A_0
0 1 0 0	1 X X X	X X X X	X X X X	X X X X

X— Don't care (Use 0 or 1)

These 5 address lines set the base address of the memory.

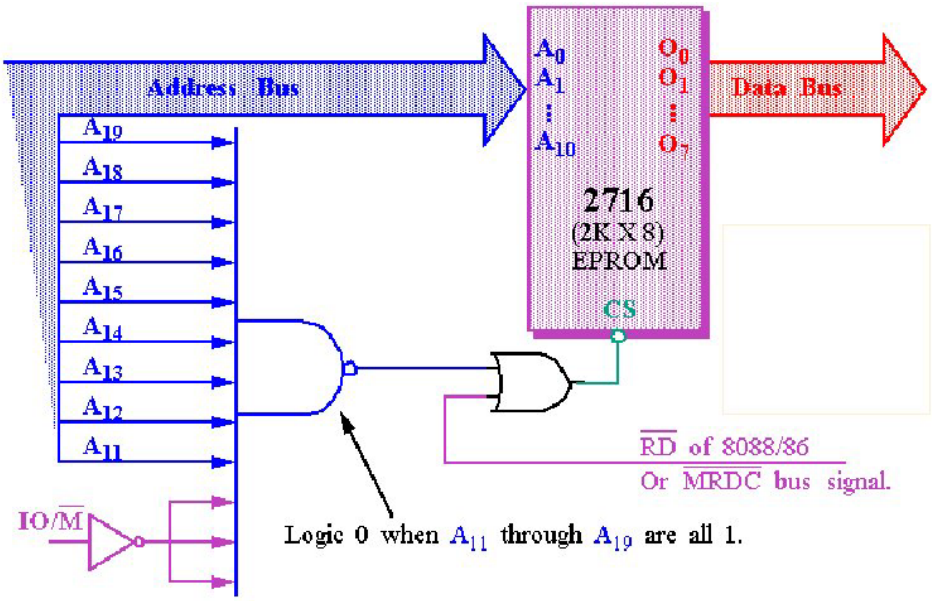
These 15 address lines will select one of 2^{15} (or 32,768) locations inside the RAMs.

Example on Address Decoding

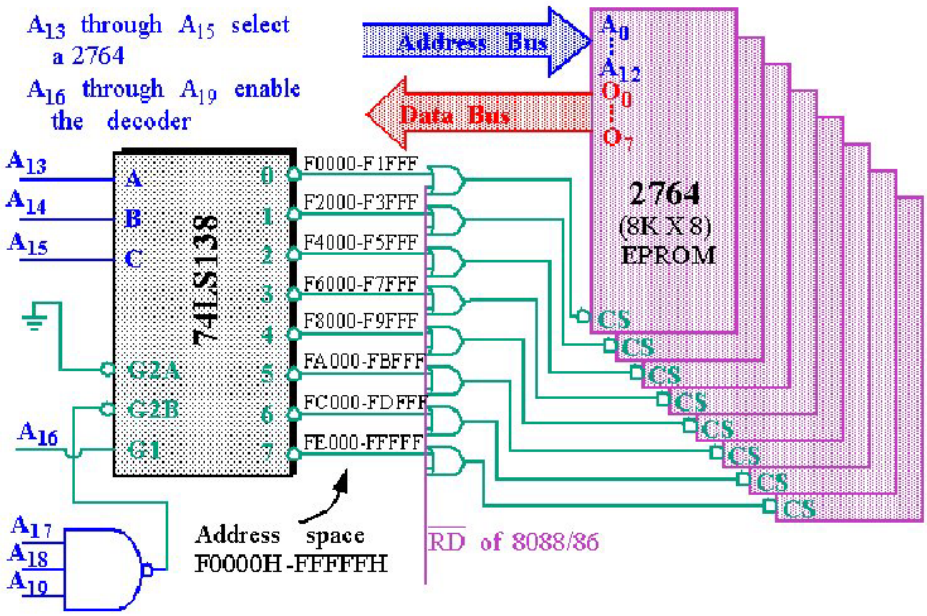


Memory Address Decoder for 48000 to 4FFFF Range

Memory Address Decoding

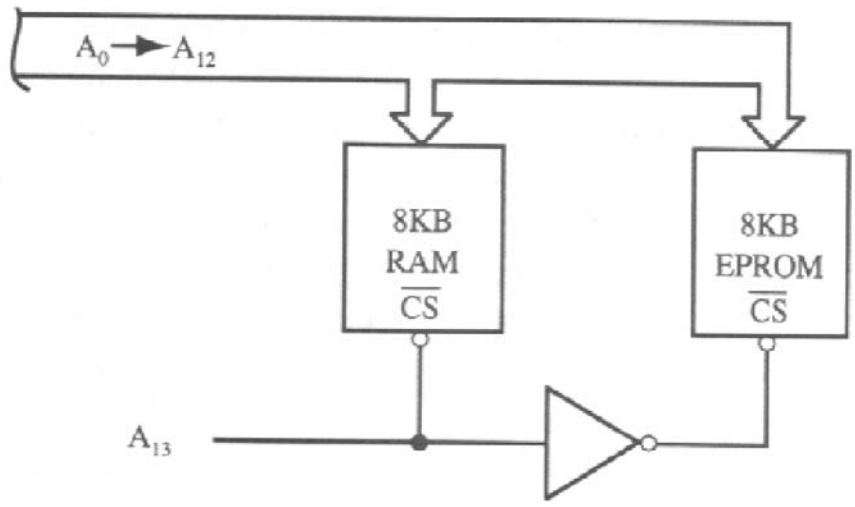


Memory Address Decoding



Partial Address Decoding

- Not all the address lines need to be used.(A14-A19 not used).
So FFFF0, 3BFF0, 07FF0 pr C3FF0 get the same data.
- (+) The purpose is get the job done in minimum hardware.
- (-) Feature expansion of the memory is impossible, and may cause invalid data reads due to overlapping memory segment reads (a fatal error)

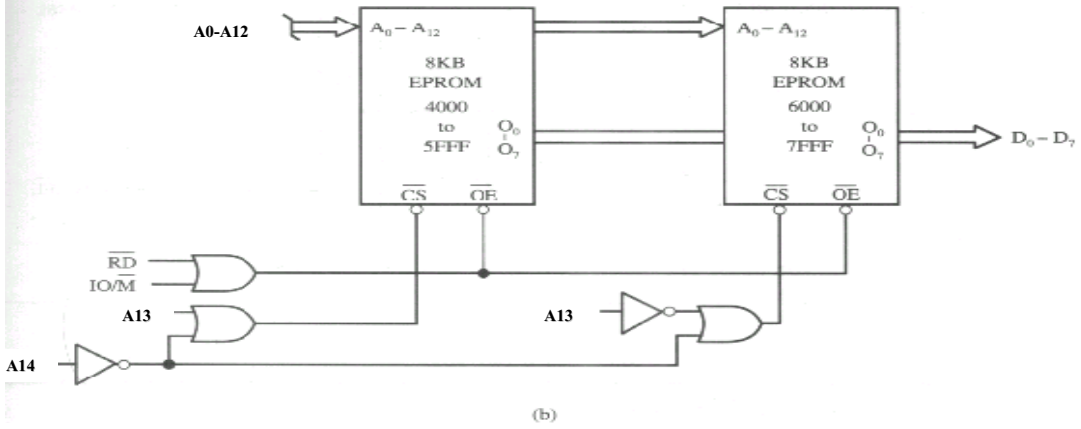


Partial Address Decoding

A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

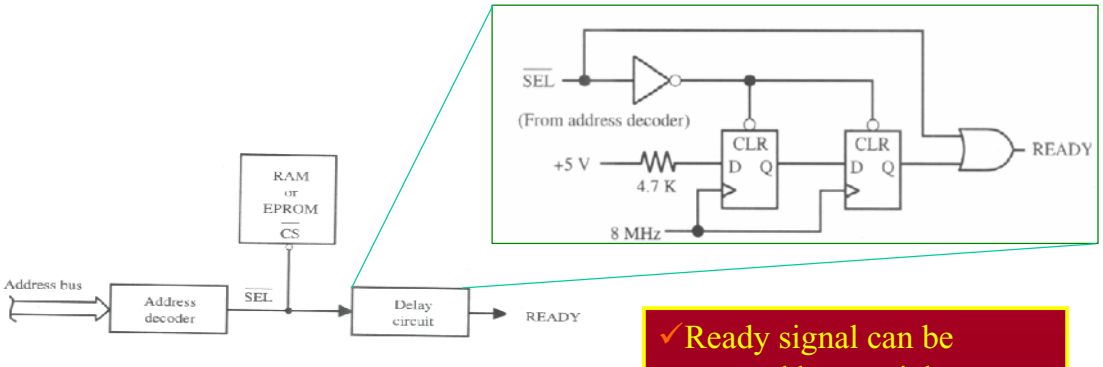
For DECODER — To EPROM

(a)

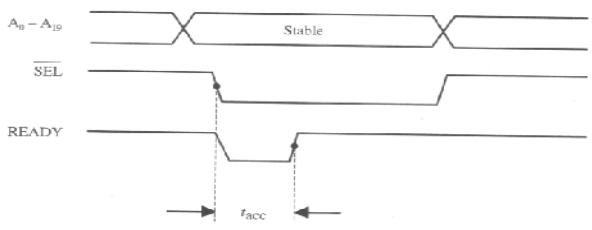


(b)

Generating Wait States in Hardware



(a)

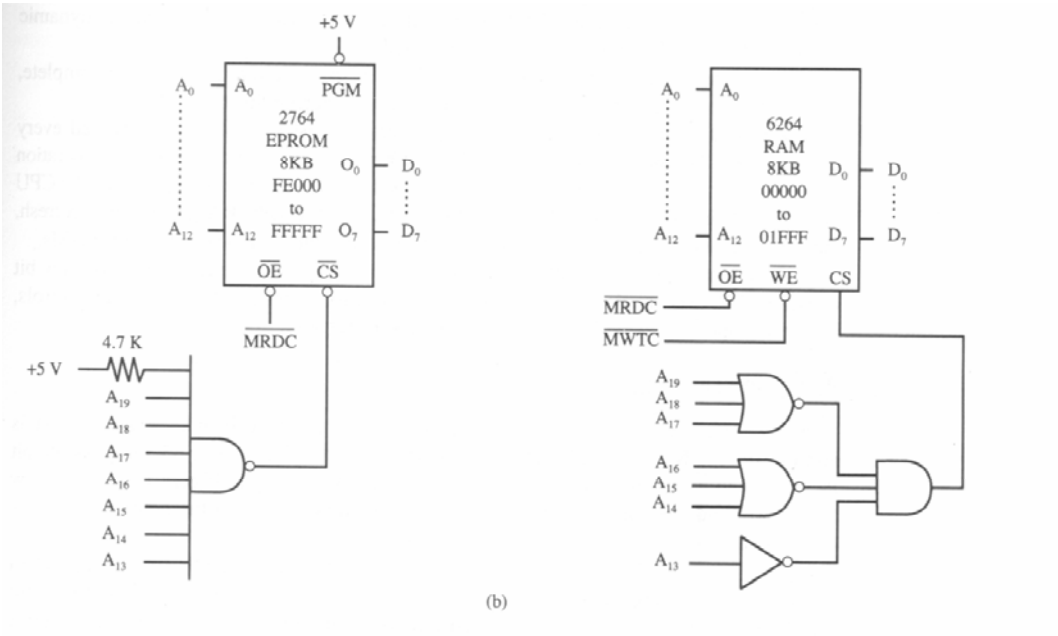


(b)

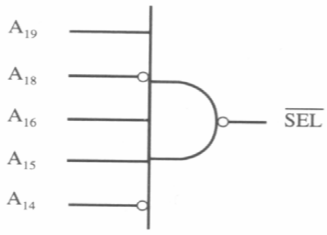
✓Ready signal can be generated by special hardware using the SEL' signal from the address decoder.

✓The circuit above will generate two clock periods of zero signal for the READY output.

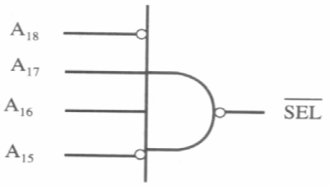
A complete RAM/EPROM Memory



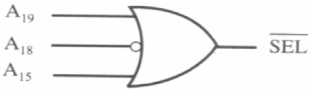
Examples: Find different addressing for CS (A0-A13 used by memories)



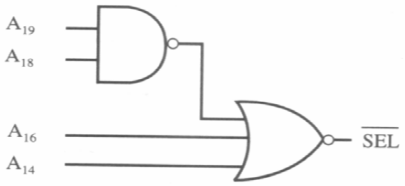
10x1 10/xx xxxx xxxxxxxx



x011 0x/xx xxxx xxxxxxxx



01xx x0/xx xxxx xxxxxxxx



(A19 and A18=0) or A16=1 or A14=1

xx1x xx/xx xxx x xxxxxxxx

ROM

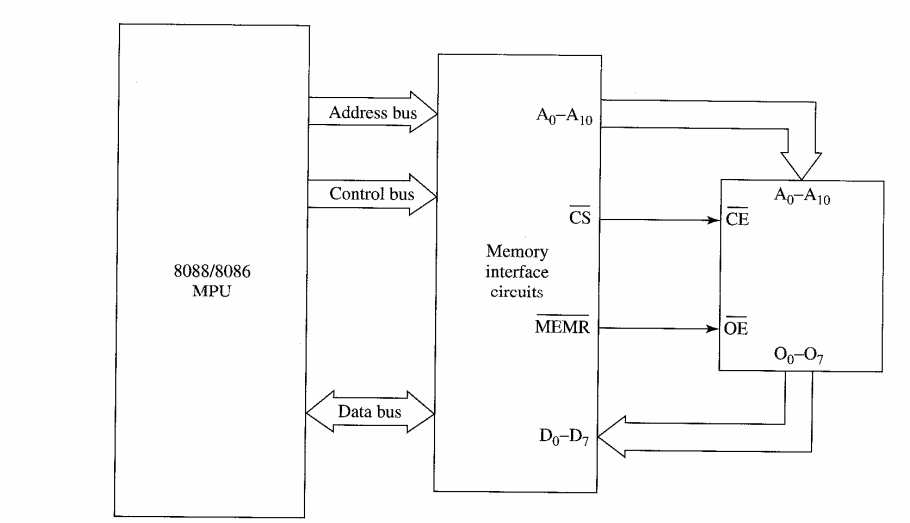


Figure 9–4 Read-only memory interface.

EPROM

EPROM	Density (bits)	Capacity (bytes)
2716	16K	2K × 8
2732	32K	4K × 8
27C64	64K	8K × 8
27C128	128K	16K × 8
27C256	256K	32K × 8
27C512	512K	64K × 8
27C010	1M	128K × 8
27C020	2M	256K × 8
27C040	4M	512K × 8

Figure 9–5 Standard EPROM devices.

RAM types

- SRAM (Static RAM)
  - Storage cells are made of flip-flops and therefore they do not require refreshing to keep their data
  - Cells handling one bit requires 6 or 4 transistors each, which is too many
  - SRAMS are widely used for cache memory and battery-backed memory systems
- DRAM (Dynamic RAM)
  - Uses MOS capacitors to store a bit
  - Requires constant refreshing due to leakage (every 2ms – 4ms)
  - Advantages
    - High density (capacity),
    - Cheaper cost per bit
    - Lower power consumption
  - Disadvantage
    - While it is being refreshed, data cannot be accessed
    - Larger access times
    - Too may pins due to large capacity

SRAM

SRAM	Density (bits)	Organization
4361	64K	64K × 1
4363	64K	16K × 4
4364	64K	8K × 8
43254	256K	64K × 4
43256A	256K	32K × 8
431000A	1M	128K × 8

Figure 9–13 devices.

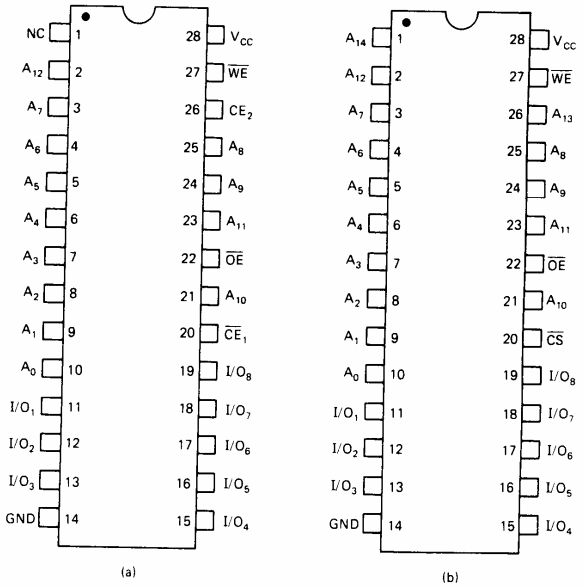


Figure 9–14 (a) 4364 pin layout. (b) 43256A pin layout.

SRAM

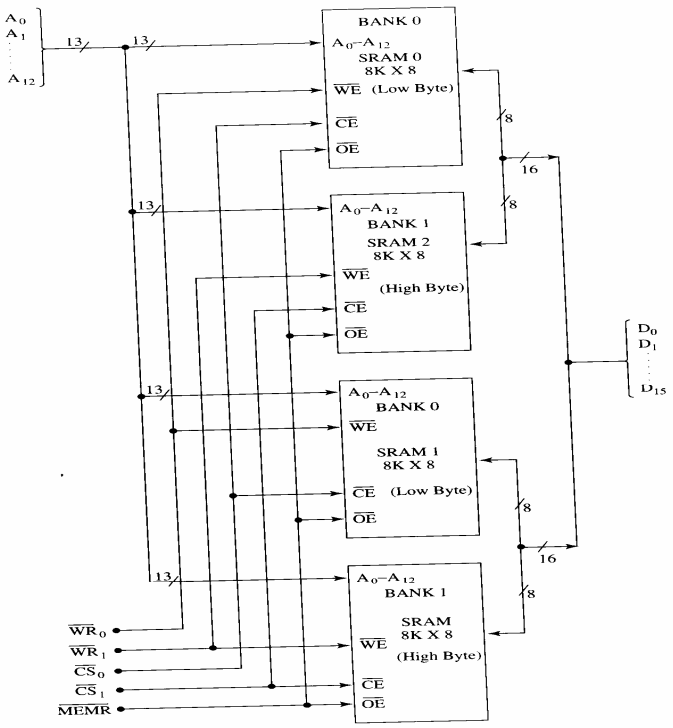


Figure 9–12 16K × 16-bit SRAM circuit.

DRAM

DRAM	Density (bits)	Organization
2164B	64K	64K × 1
21256	256K	256K × 1
21464	256K	64K × 4
421000	1M	1M × 1
424256	1M	256K × 4
44100	4M	4M × 1
44400	4M	1M × 4
44160	4M	256K × 16
416800	16M	8M × 2
416400	16M	4M × 4
416160	16M	1M × 16

Figure 9–19 Standard DRAM devices.

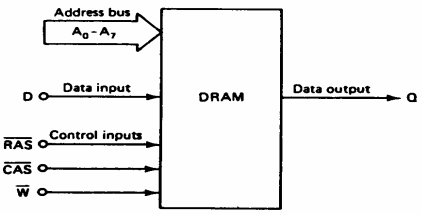


Figure 9–21 Block diagram of the 2164B DRAM.

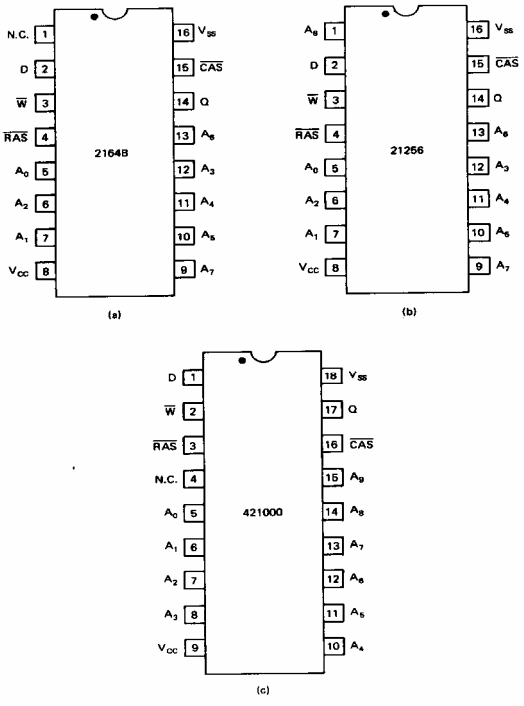
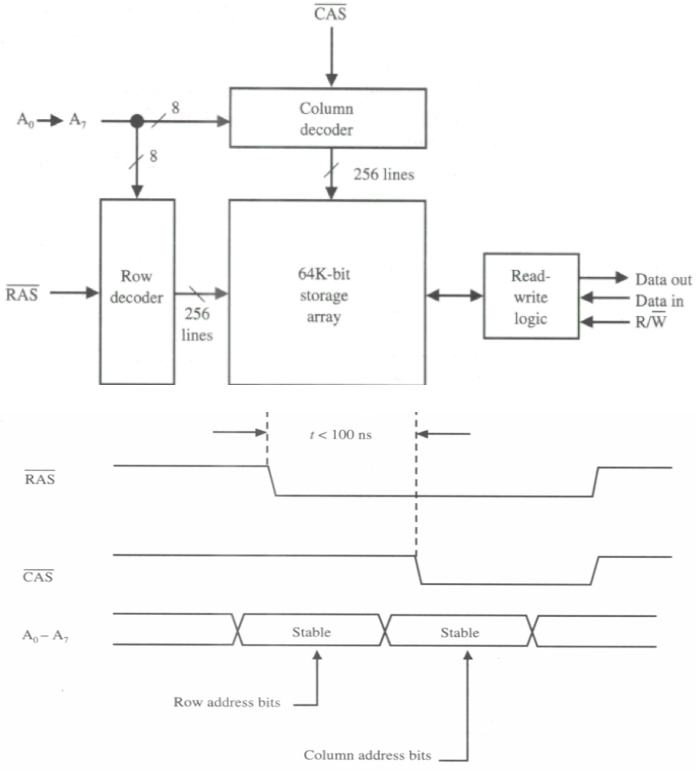


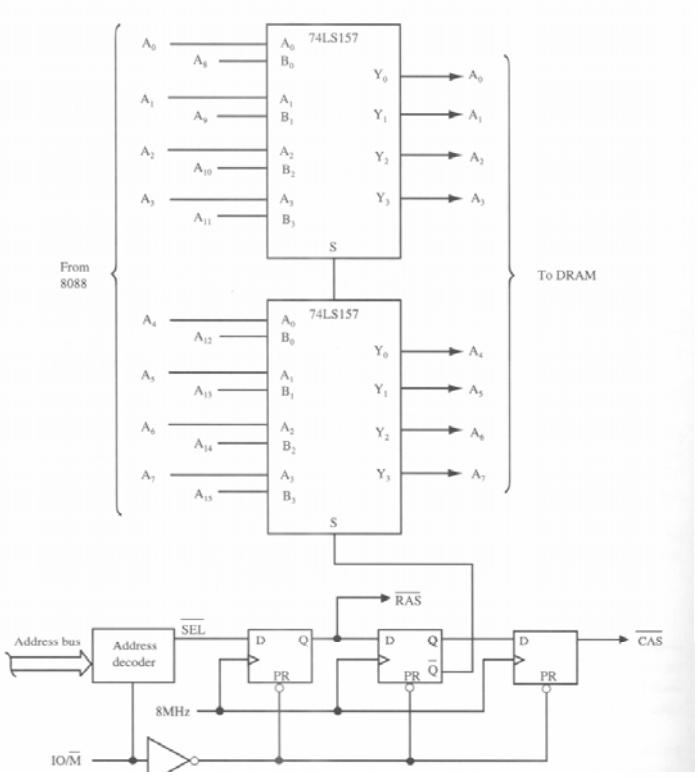
Figure 9–20 (a) 2164B pin layout. (b) 21256 pin layout. (c) 421000 pin layout.

DRAM



- In DRAM, the 8 address lines are latched accordingly by the strobe of the RAS and CAS signals.
- For example: To load a 16 bit address into the DRAM 8 bits of the address are first latched by pulling RAS low, then other 8 bits are presented to A0-A7 and CAS is pulled low.

DRAM Addressing



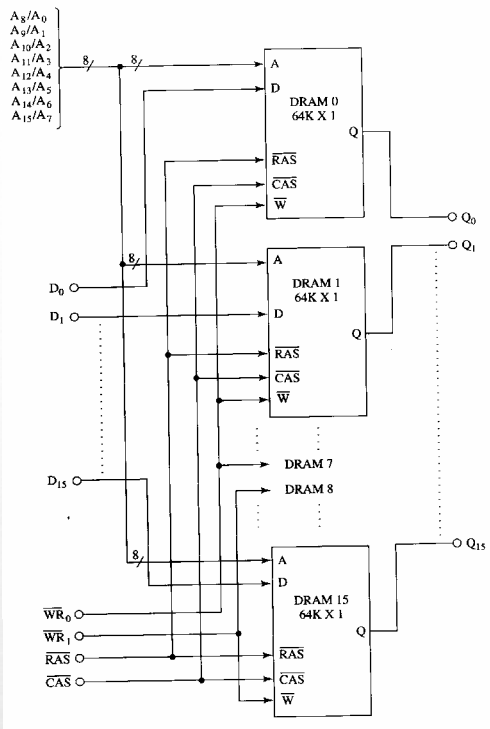


Figure 9–22 64K × 16-bit DRAM circuit.

IBM PC Memory Map

- 00000h – 9FFFFh: RAM (640 Kb)
  - The first 1K used for the interrupt vector table (00000h to 003FFh)
  - 00400h to 004FFh is set aside for the BIOS temporary area
  - 00500h to 005FFh is set aside for the temporary storage of certain parameters in DOS and BASIC
  - A certain number of Kbytes is occupied by the operating system itself
- A0000h – BFFFFh: Video Display RAM (128 Kb)
  - A total of 128 Kbytes is allocated for video
  - Of that 128K, only a portion is used for VDR, the amount depending on which type of video adapter card is installed in the system
- C0000h – FFFFh: ROM (256 Kb)
  - 256 K is set aside for ROM
  - Used in
    - BIOS ROM, Basic language compiler ROM, hard disk controller, other peripheral board ROMS and the rest for expansion by the user

IBM PC Memory Map

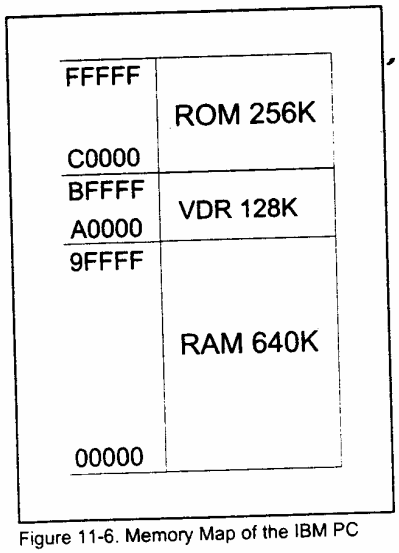


Figure 11-6. Memory Map of the IBM PC

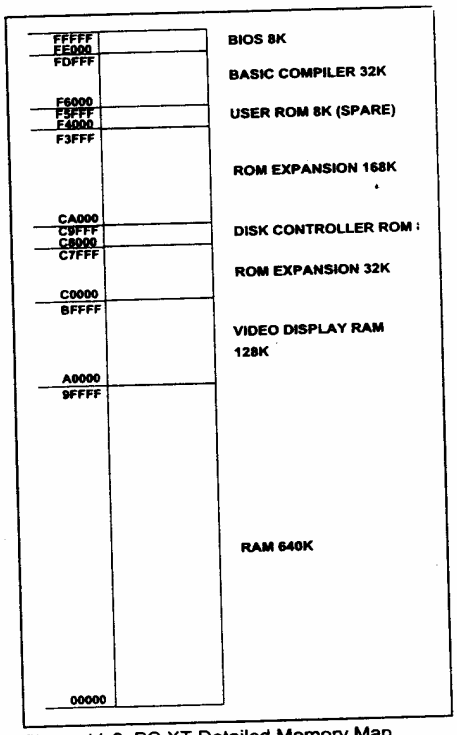


Figure 11-8. PC.XT Detailed Memory Map

Data Transfer

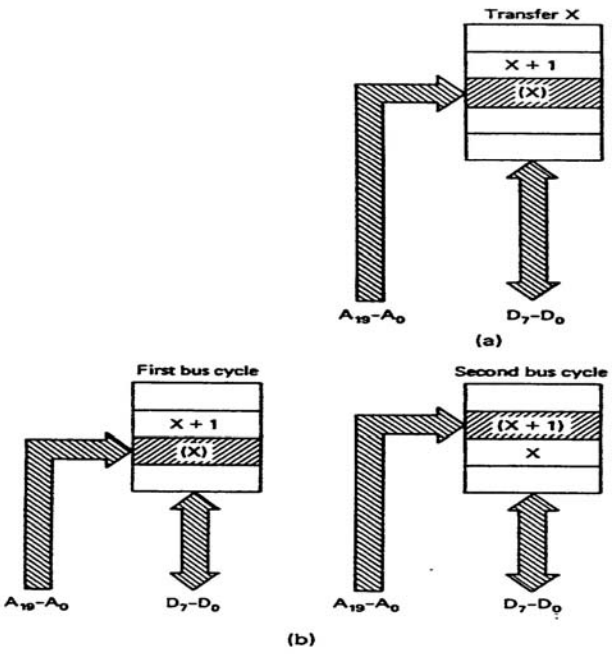
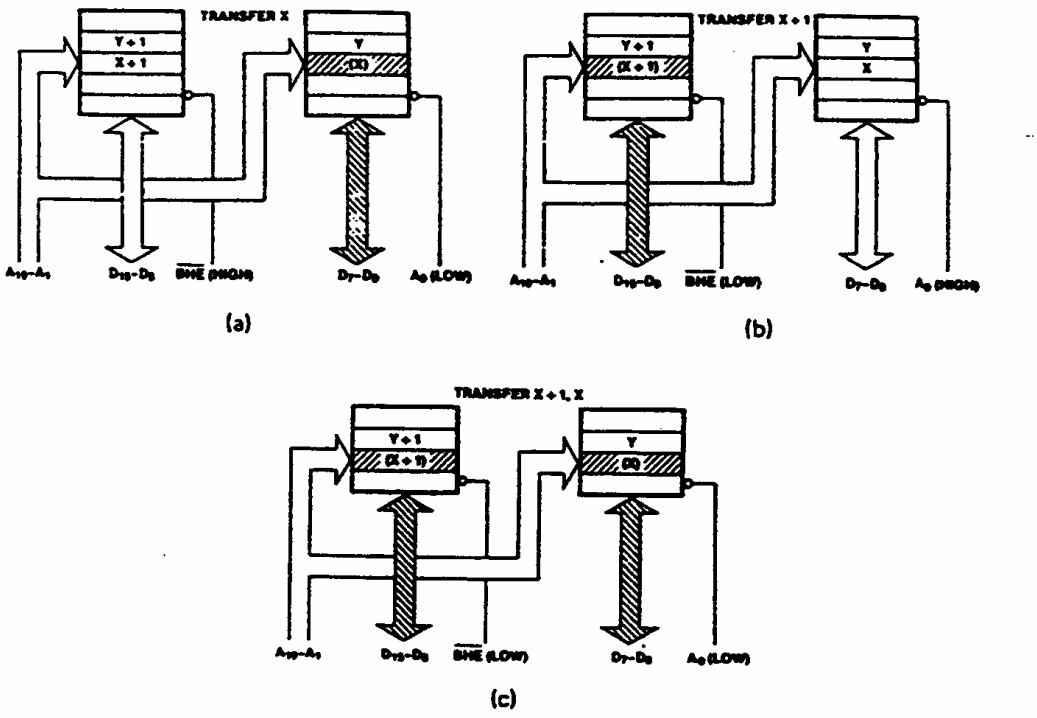
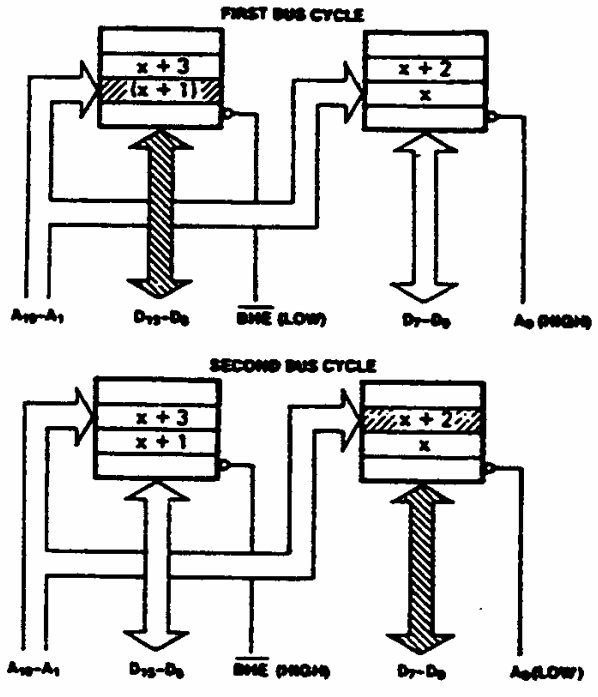


Figure 8–16 (a) Byte tmasfer by the 8088. (b) Word transfer by the 8088.

Data Transfer



Data Transfer

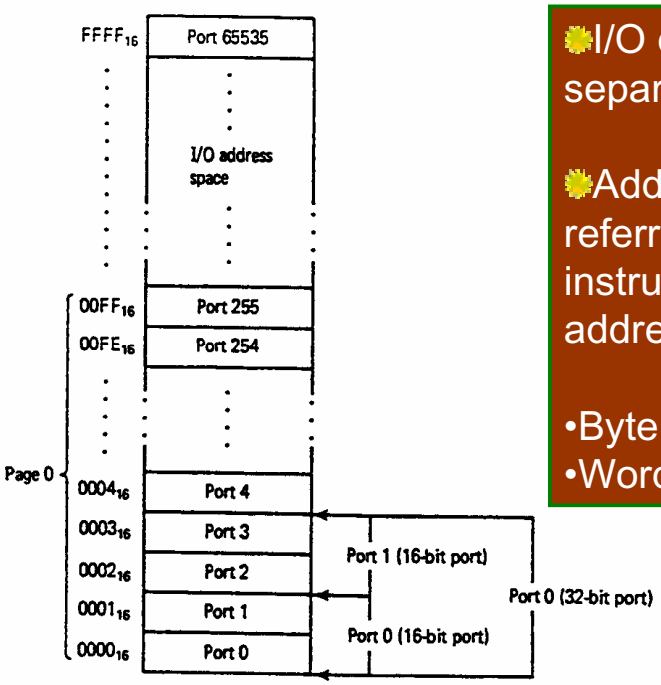


Weeks 9-10-11
Input/Output Interface Circuits and LSI Peripheral Devices

Core and Special Purpose I/O Interfaces

- Special purpose I/O interfaces are implemented as add-on cards on the PC
  - display
  - parallel printer interface
  - serial communication interface
  - local area network interface
  - not all microcomputer systems employ each of these types
- Core input/output interfaces are considered to be the part of the I/O subsystem such as:
  - parallel I/O to read the settings of the DIP switches on the processor board
  - interval timers used in DRAM refresh process
- We will study both

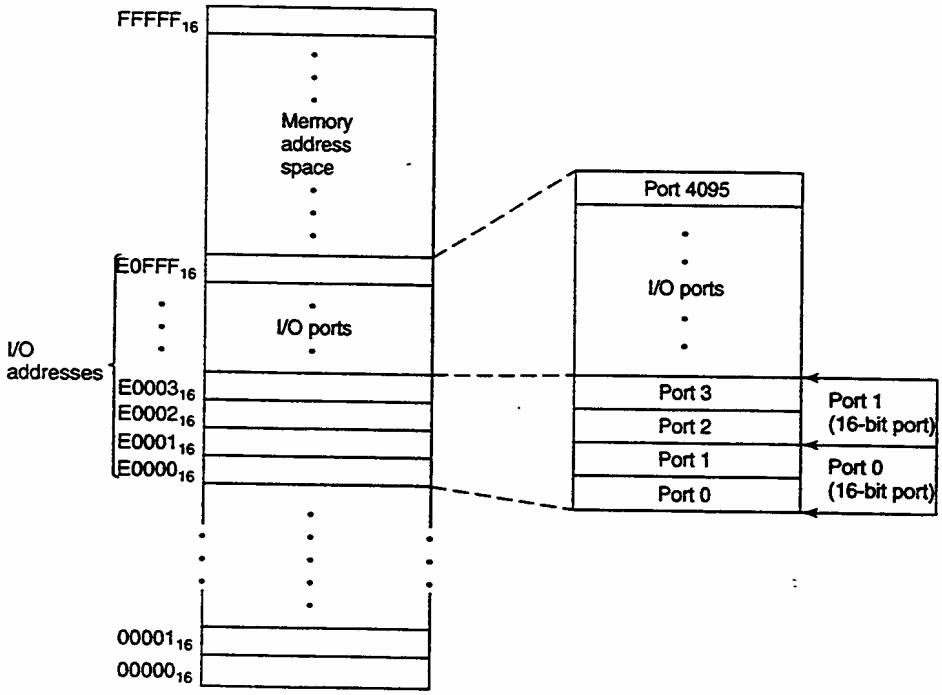
Isolated I/O



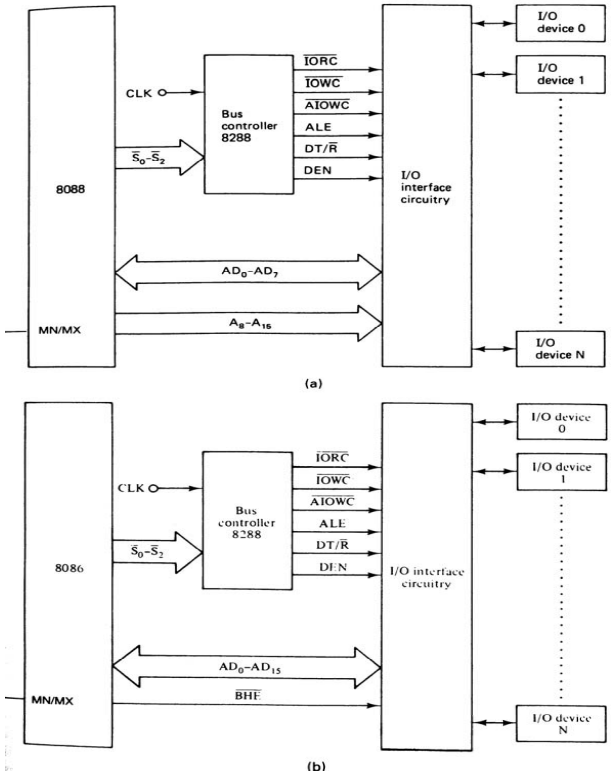
- I/O devices are treated separately from memory
- Address 0000 to 00FF: referred to page 0. Special instructions exist for this address range

  •Byte wide ports
  •Word wide ports

Memory Mapped I/O



Maximum Mode I/O interface - 8088/8086



I/O Bus Cycle Status Codes

Status inputs			CPU cycle	8288 command
$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$		
0	0	0	Interrupt acknowledge	$\overline{INTA}$
0	0	1	Read I/O port	$\overline{IORC}$
0	1	0	Write I/O port	$\overline{IOWC}$, $\overline{AIOWC}$
0	1	1	Halt	None
1	0	0	Instruction fetch	$\overline{MRDC}$
1	0	1	Read memory	$\overline{MRDC}$
1	1	0	Write memory	$\overline{MWTC}$, $\overline{AMWC}$
1	1	1	Passive	None

I/O Instructions

Mnemonic	Meaning	Format	Operation
IN	Input direct	IN Acc,Port	(Acc) ← (Port) Acc = AL or AX
	Input indirect (variable)	IN Acc,DX	(Acc) ← ((DX))
OUT	Output direct	OUT Port,Acc	(Port) ← (Acc)
	Output indirect (variable)	OUT DX,Acc	((DX)) ← (Acc)

Example. Write a sequence of instructions that will output the data FFh to a byte wide output at address ABh of the I/O address space

```
MOV AL,0FFh
OUT 0ABh, AL
```

Example. Data is to be read from two byte wide input ports at addresses AAh and A9h and then this data will then be output to a word wide output port at address B000h

```
IN AL, 0AAh
MOV AH,AL
IN AL, 0A9h
MOV DX,0B00h
OUT DX,AX
```


Input Bus Cycle of the 8088

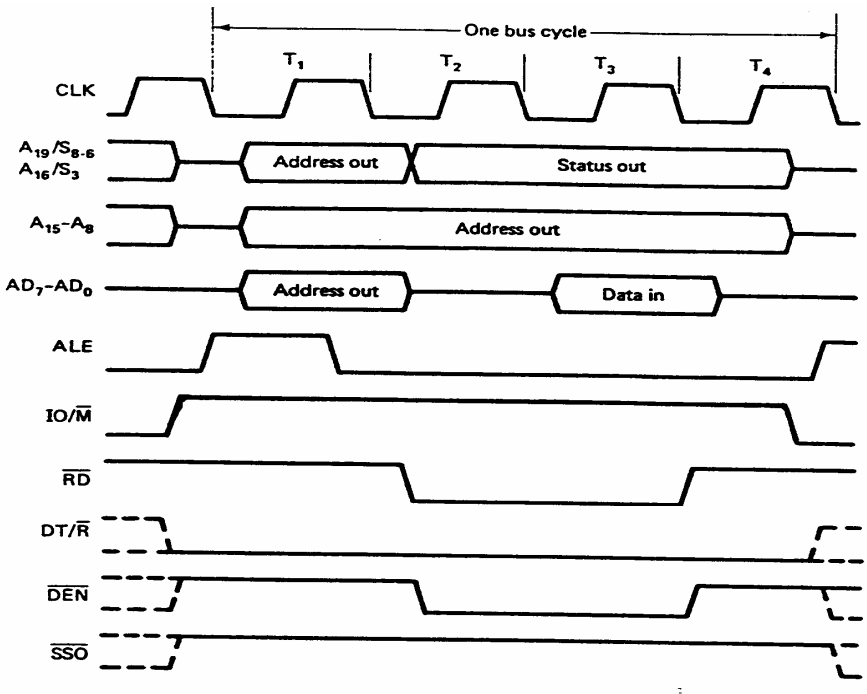


Figure 8–52 Input bus cycle of the 8088.

Output Bus Cycle of the 8088

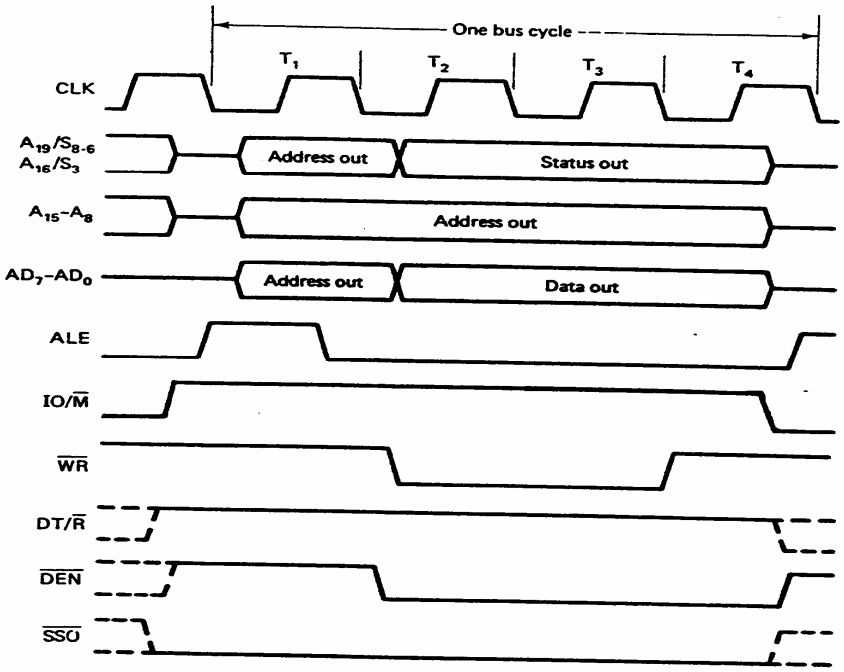


Figure 8–53 Output bus cycle of the 8088.

I/O Example

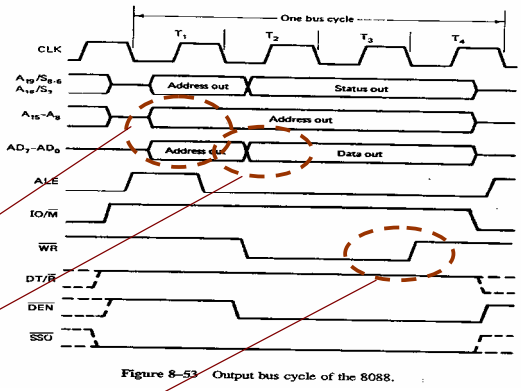
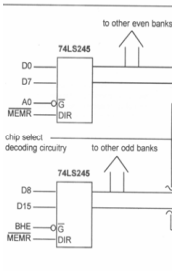


Figure 8–53 Output bus cycle of the 8088.

- Assume that AX = 76A9h.: Analyze the data transfer for a) 8088 b) 8086 when
  MOV DX, 648h
  OUT DX,AX
- 8088 case
  - 1st bus cycle
    - T1: Address 0648h is put on pins AD0-AD7, A8-15 and latched when ALE is activated
    - T2: The low byte A9h is put on the data bus pins AD0-AD7 and $\overline{\text{IOWC}}$ is activated
    - T3: Setup time
    - T4: Byte is written to the port assuming zero wait states
  - 2nd Bus Cycle (Similar to 1st Bus Cycle)
    - T1: Address 0649h is put on pins AD0-AD7, A8-15 and latched when ALE is activated
    - T2: The high byte 76h is put on the data bus pins and $\overline{\text{IOWC}}$ is activated
    - T3: Setup time
    - T4: Byte is written to the port assuming zero wait states

Example continued

- 8086 case
  - T1: Address 0648h is put on pins AD0-AD15 plus BHE=low is latched by the 74LS373 when ALE is activated
  - T2: 76A9h, the contents of AX, is put on AD0-AD15 (A9h on AD0-AD7, 76h on AD8-AD15) and IOWC is activated
  - T3: Setup time
  - T4: During this interval, with the help of the signals A0=0 and BHE=0, the low and high bytes are written to the appropriate ports.
    - ➢It must be noted that since the operand is a 16 bit word and the port address is an even address, the 8086 CPU does not generate address 0649h
    - ➢Port address 648h is connected to the D0-D7 data bus and port address 649h is connected to the D8-D15 data bus

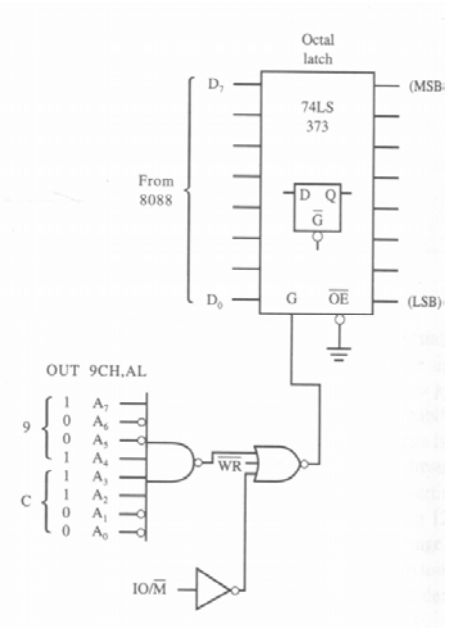


I/O Design in the 8088/86

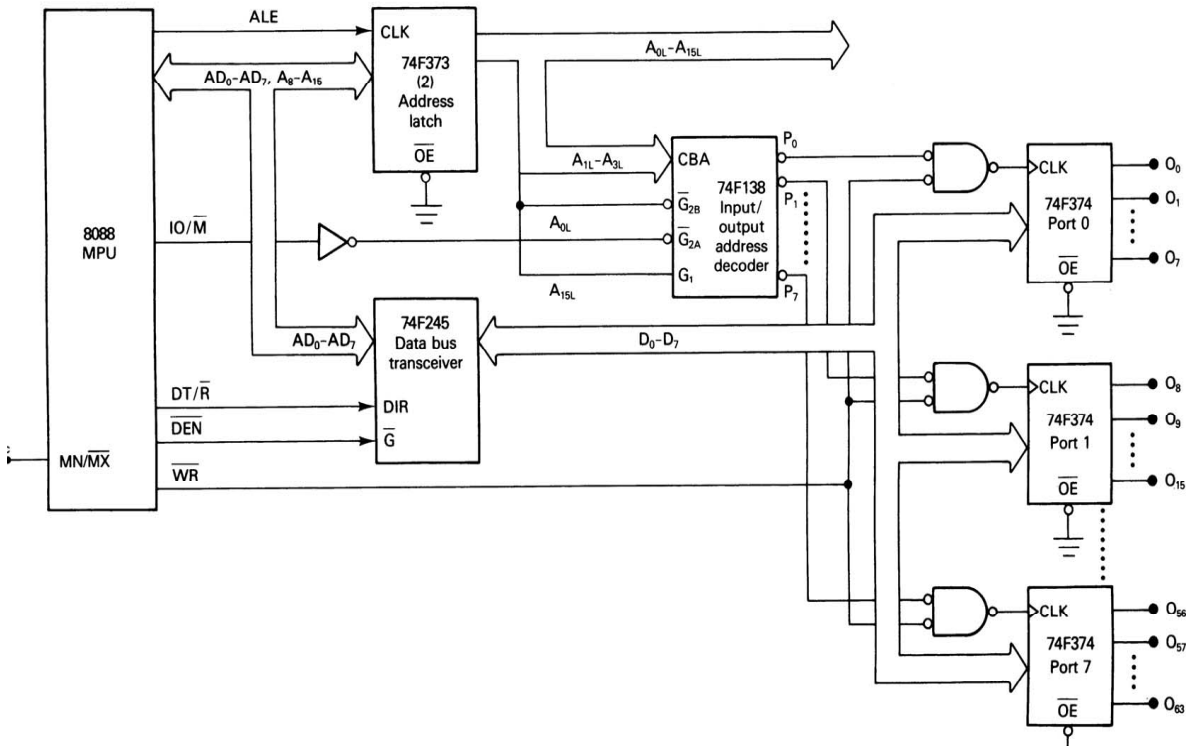
- In every computer, when data is sent out by the CPU, the data on the data bus must be latched by the receiving device
- While memories have an internal latch to grab the data on the data bus, a latching system must be designed for ports
- Since the data provided by the CPU to the port is on the system data bus for a limited amount of time (50 - 1000ns) it must be latched before it is lost
- Likewise, when data is coming in by way of a data bus (either from port or memory) it must come in through a three-state buffer

I/O Design

- Design for OUT 9CH,AL



Example - 64 line parallel output circuit - 8088



Examples

- To which output port in the previous figure are data written when the address put on the bus during an output bus cycle is 8002h?
  - A15 .. A0 = 1000 0000 0000 0010b
  - A15L = 1
  - A0L = 0
  - A3L A2L A1L = 001
  - $\overline{P1}$ = 0

- Write a sequence of instructions that output the byte contents of the memory address DATA to output port 0 in the previous figure

```
MOV DX, 8000h
MOV AL,DATA
OUT DX, AL
```


Time Delay Loop and Blinking a LED at an Output

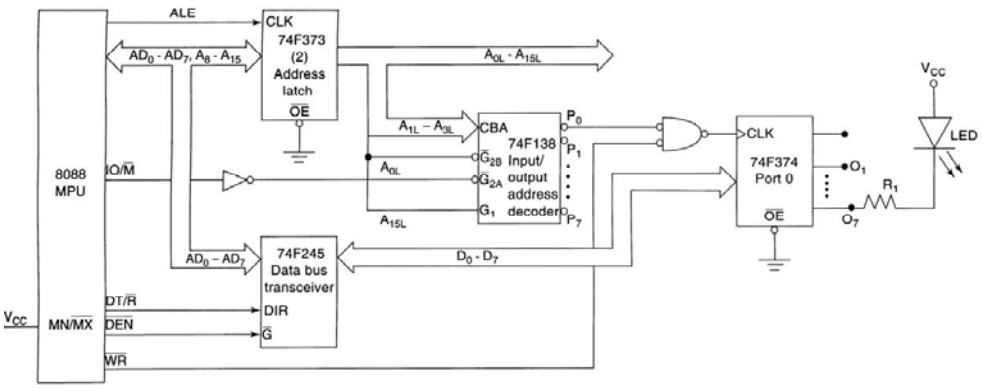


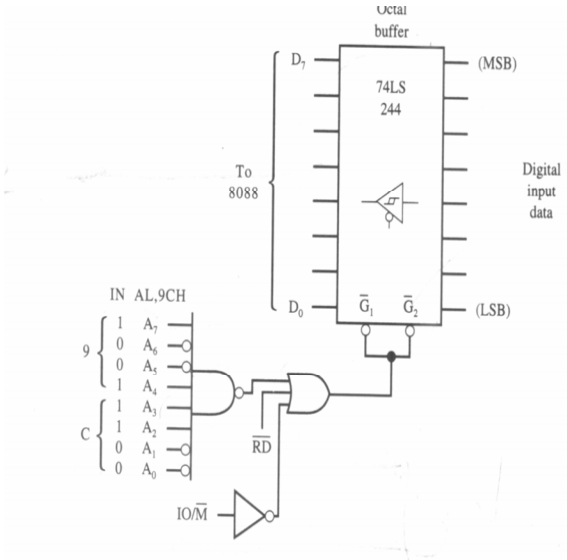
Figure 10–2 Driving an LED connected to an output port.

```
        MOV DX, 8000h     ;initialize address of port 0
        MOV AL, 00h       ; load data with bit 7 as logic 0
ON_OFF: OUT DX,AL         ; turned on
        MOV CX,0FFFFh     ; load delay count of FFFFh
HERE:   LOOP HERE  ------------------------------>
        XOR  AL,80h       ; complement bit 7
        JMP  ON_OFF
```

Aprox.
17 T states
* 64K *
Frequency

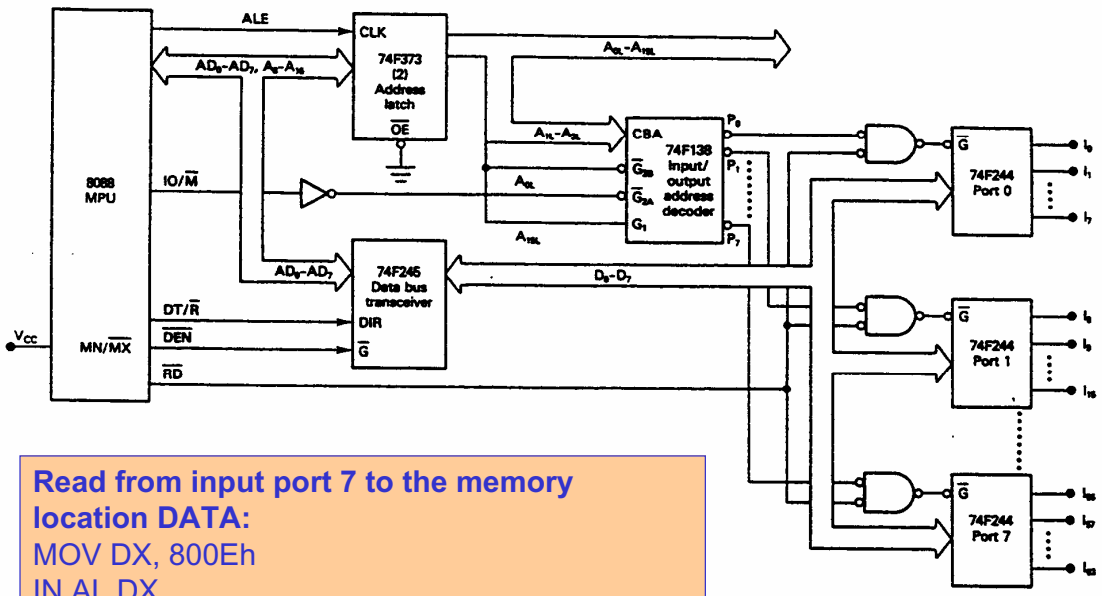
IN port design using the 74LS244

➢Design for IN AL,9CH



- In order to prevent any unwanted data (garbage) to come into the system (global) data bus, all input devices must be isolated through the tri-state buffer. The 74LS244 not only plays this role but also provides the incoming signals sufficient strength (driving capability) to travel all the way to the CPU
- It must be emphasized that every device (memory, peripheral) connected to the global data bus must have a latch or a tri-state buffer. In some devices such as memory, they are internal but must be present.

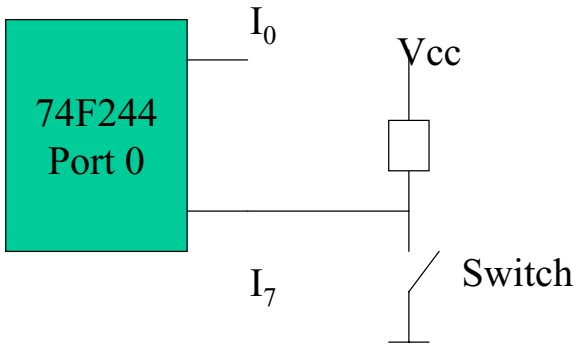
Example - 64 line parallel input circuit



Read from input port 7 to the memory location DATA:
MOV DX, 800Eh
IN AL,DX
MOV DATA, AL

Example

- In practical applications, it is sometimes necessary within an I/O service routine to repeatedly read the value at an input line and test this value for a specific logic level.

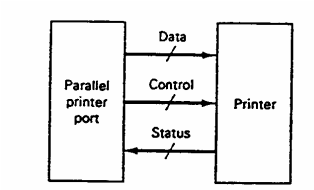


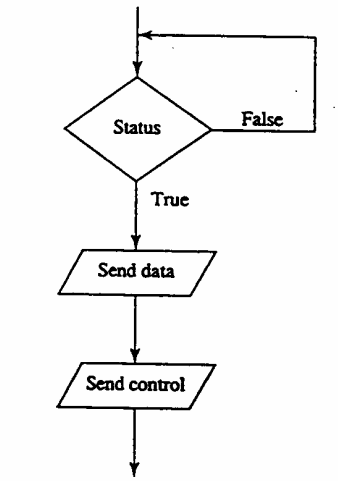

```
Poll the switch waiting for it to close
        MOV DX,8000h
POLL:   IN AL,DX
        SHL AL,1
        JC POLL
```


Input Output Handshaking

- The I/O ports of a computer typically operate at different data rates
- A hard disk drive, for example, might require the computer to input data at 10Mbps → 100Mbps
- CD-ROM drives operate at 300-600 Kbps
- However when inputting keystrokes from the operator, the data rate may fall to only one or two characters per sec.
- If the processor is to operate efficiently, one needs to develop a strategy to control or synchronize the flow of data between the processor and the widely varying rates of its I/O devices
- This type of synchronization is achieved by implementing what is known as handshaking as part of the input/output interface
- Printers typically have buffers that can be filled by the computer at high speed
- Once full the computer must wait while the data in the buffer is printed
- Most printer manufacturers have settled on a standard set of data and control signals Centronics Parallel Printer Interface

Parallel Printer Interface





Pin	Assignment
1	Strobe
2	Data 0
3	Data 1
4	Data 2
5	Data 3
6	Data 4
7	Data 5
8	Data 6
9	Data 7
10	Ack
11	Busy
12	Paper Empty
13	Select
14	Auto Foxt
15	Error
16	Initialize
17	Slctin
18	Ground
19	Ground
20	Ground
21	Ground
22	Ground
23	Ground
24	Ground
25	Ground

```
Data:       Data0, Data1, ........., Data7

Control:    Strobe
            Auto Foxt
            Initialize
            Slctin

Status:     Ack
            Busy
            Paper Empty
            Select
            Error
```

ACK is used by printer to acknowledge receipt of data and can accept a new character.

BUSY high if printer is not ready to accept a new character

SELECT when printer is turned on

ERROR goes low when there are conditions such as paper jam, out of paper, offline

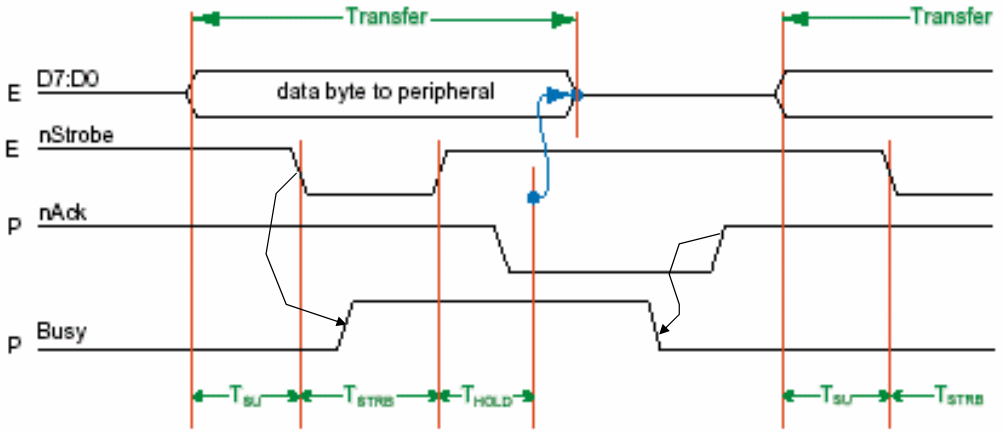
STROBE when PC presents a character

INITIALIZE Clear Printer Buffer and reset control

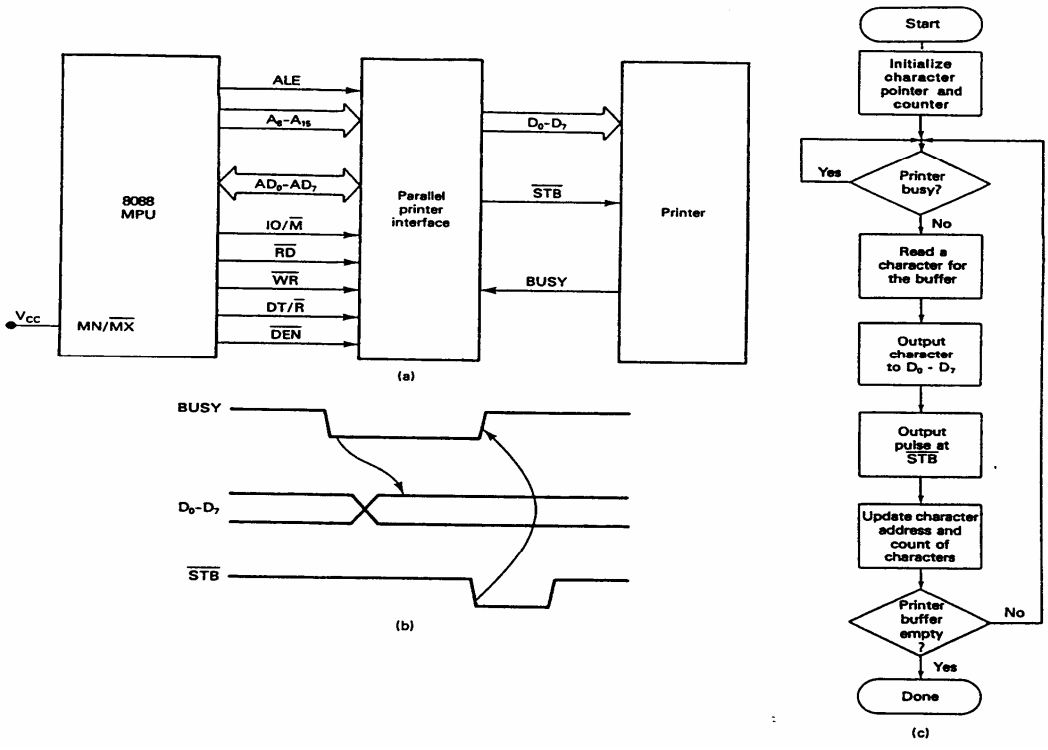
Operational Principle - Parallel Printer Port

- The computer checks the BUSY signal from the printer, if not BUSY then
- When the PC presents a character to the data pins of the printer, it activates the STROBE pin, telling it that there is a byte sitting at the data pins. Prior to asserting STROBE pin, the data must be at at the printer's data pins for at least 0.5 microsec. (data setup time)
- The STROBE must stay for 0.5 microsec

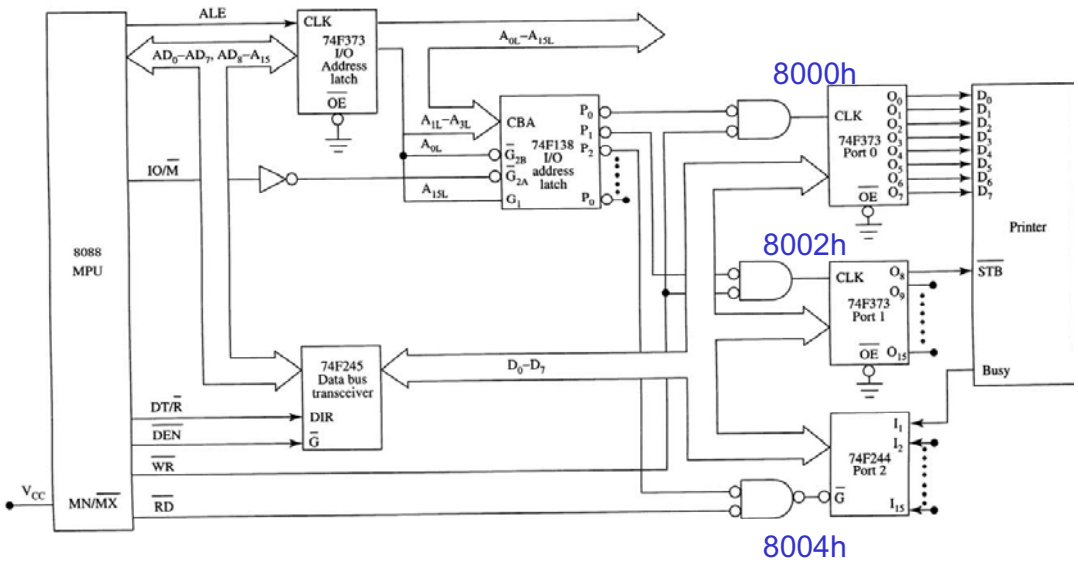
- The printer asserts BUSY pin indicating the computer to wait
- When the printer picks up the data, it sends back the ACK signal, keeps ACK low for 5 microsec.
- As the ACK signal is going high, the printer makes the BUSY pin low to indicate that it is ready to accept the next byte
- The CPU can use ACK or BUSY signals from the printer to initiate the process of sending another byte



Handshaking



Printer Interface Circuit



Example

- Write a program that implements the flowchart. Character data is held in memory starting at address PRNT_BUFF, the number of characters held in the buffer is identified by the count address CHAR_COUNT.

```
                MOV CL, CHAR_COUNT
                MOV SI,  OFFSET PRNT_BUFF
POLL_BUSY:      MOV DX,8004h
                IN AL,DX
                AND AL,01h                    BUSY input checked
                JNZ POLL_BUSY
                MOV AL, [SI]
                MOV DX,8000h                  Character is output
                OUT DX,AL
                MOV AL, 00h           ;STB = 0
                MOV DX,8002h                  So as the strobe
                OUT DX,AL
                MOV BX,0Fh            ; delay for STB = 0
STROBE: DEC BX
                JNZ STROBE
                MOV AL,01h
                OUT DX,AL             ; STB bar = 1
                 INC SI
                 DEC CL
                JNZ POLL_BUSY
```

IBM PC Printer Interfacing

- Base I/O port address of each printer is written into the BIOS data area 0040:0008 to 0040:000Fh
- 0040:0008 and 0040:0009 LPT1 (e.g., 0378h)
  - 0378h: data port
  - 0379h: status port read only
  - 037Ah: control port
- 0040:000A and 0040:000B LPT2
- BIOS INT 17h provides three services: printing a character, initializing the printer port and getting the printer status port
- AH = 0; print a character in AL and have the LPT number in DX (0 for LPT1, 1 for LPT2) also returns status
- AH = 01; initialize the printer port by setting the printer to the top of the page
- AH = 2; get the printer status after calling AH = status as:
  - bit 7: 1 ready 0 busy
  - bit 3: 1 I/O error
  - bit 5: 1 out of paper

Printer Time Out

Timeout occurs due to: printer offline / not able to print for any reason although it is properly connected

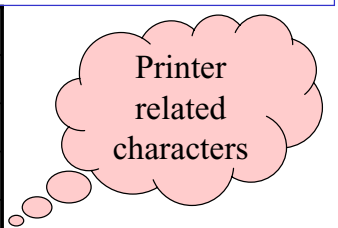
BIOS repeatedly tries for a period of 20 seconds to see if the printer is ready to accept data.

It is possible to alter the entered by BIOS at boot time.

0040:0078 → 0040:007B holds the wait times.

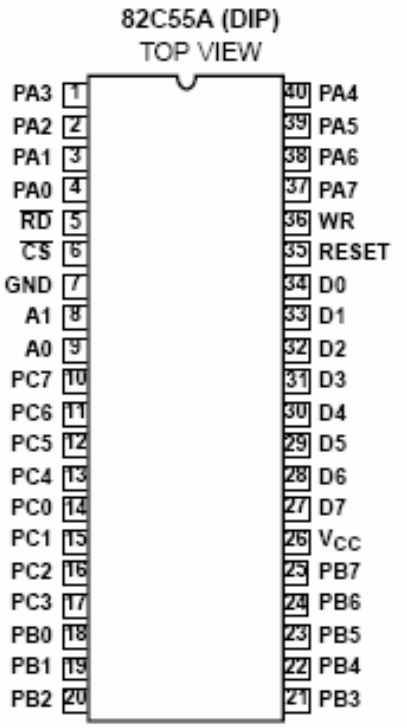
```
MOV AH,0 ; print character option
MOV DX,0 ; select LPT1
MOV AL,41h     ; ASCII code for letter A
INT 17h
```

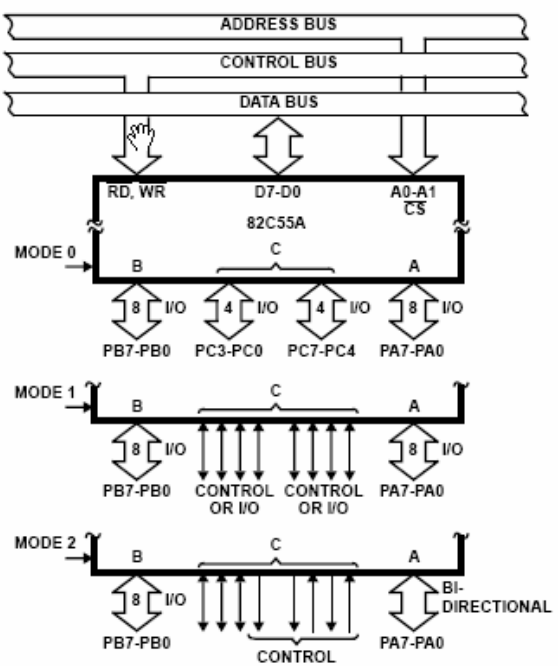
ASCII Table	Hex Code	Function
BS	08	Backspace
HT	09	Horizontal Tab
LF	0A	Line Feed
VT	0B	Vertical Tab
FF	0C	Form Feed
CR	0D	Carriage Return



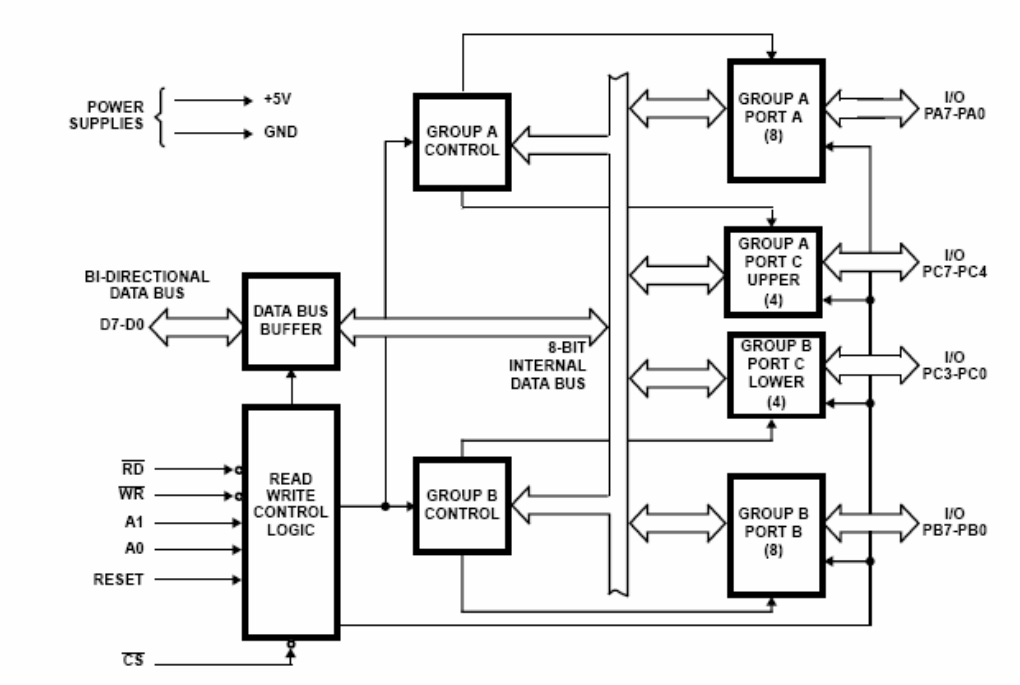
The 8255 Programmable Peripheral Interface

- Intel has developed several peripheral controller chips designed to support the 80x86 processor family. The intent is to provide a complete I/O interface in one chip.
- 8255 PPI provides three 8 bit input ports in one 40 pin package making it more economical than 74LS373 and 74LS244
- The chip interfaces directly to the data bus of the processor, allowing its functions to be programmed; that is in one application a port may appear as an output, but in another, by reprogramming it as an input. This is in contrast with the 74LS373 and 74LS244 which are hardwired and fixed
- Other peripheral controller chips include the 8259 Programmable Interrupt Controller (PIC), the 8253/54 Programmable Interval Timer (PIT) and the 8237 DMA controller

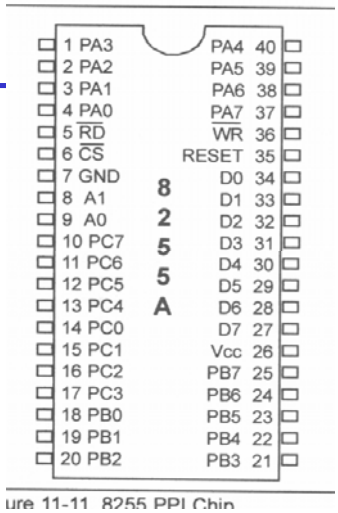




8255A internal



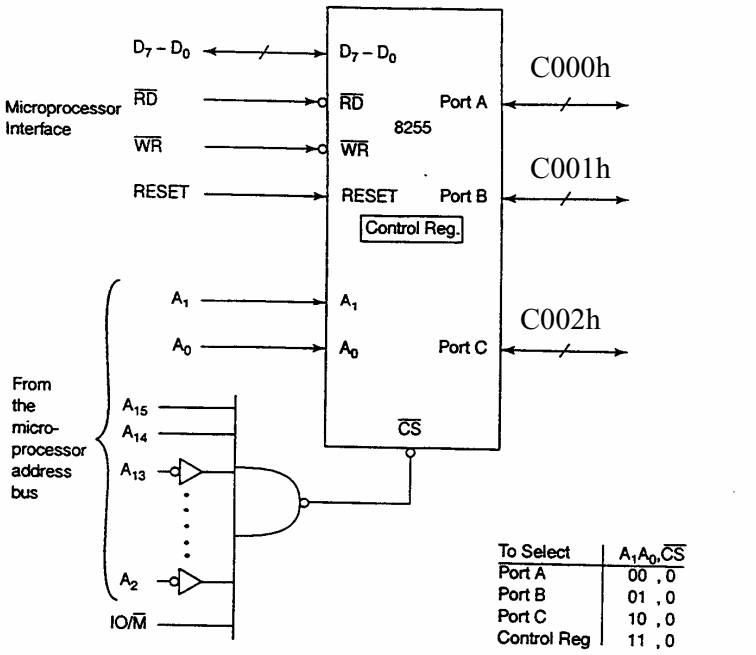
8255 Pins



- PA0 - PA7: input, output, or bi-directional port
- PB0 - PB7: input or output
- PC0 - PC7: This 8 bit port can be all input or output. It can also be split into two parts, CU (PC4 - PC7) and CL (PC0 - PC3). Each can be used for input and output.
- $\overline{RD}$ or $\overline{WR}$
  - $\overline{IOR}$ and $\overline{IOW}$ of the system are connected
- RESET
- A0, A1, and $\overline{CS}$
  - $\overline{CS}$ selects the entire chip whereas A0 and A1 select the specific port (A, B, or C)

CSBAR	A1	A0	SELECTS:
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control Register
1	x	x	8255 not selected

Addressing an 8255



8255 Control Word Format

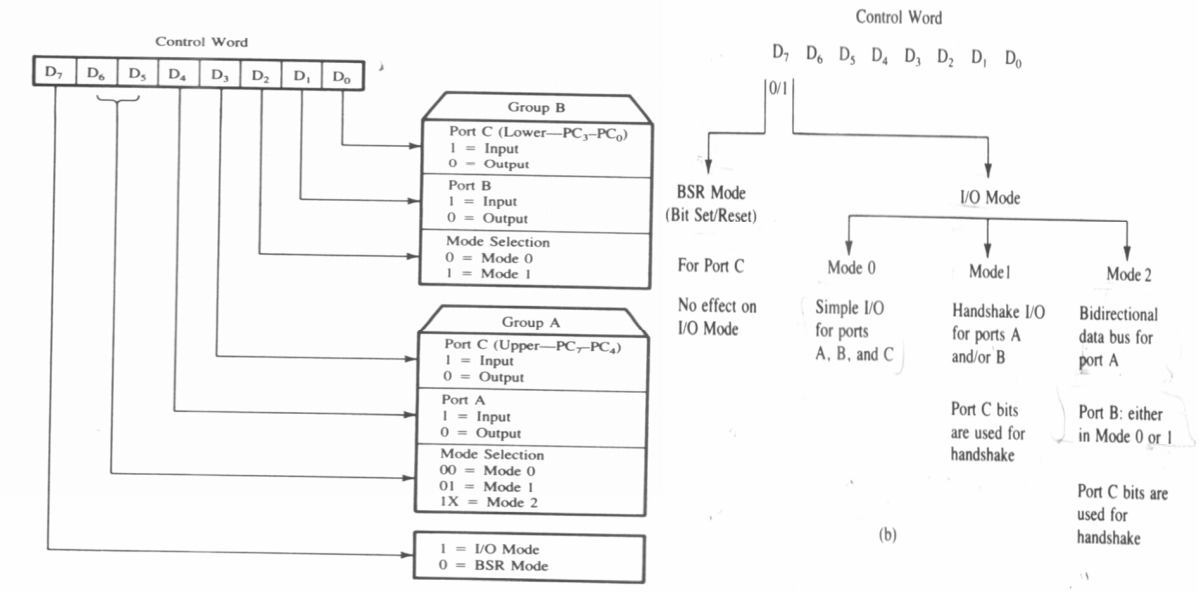


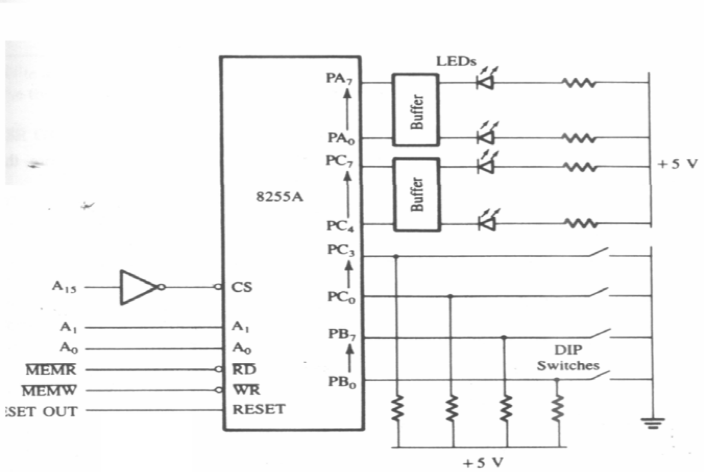
FIGURE 15.4
8255A Control Word Format for I/O Mode
SOURCE: Adapted from Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), p. 3–

Mode 0 - Simple input/output

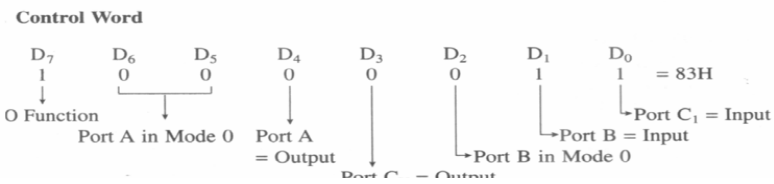
- Simple I/O mode: any of the ports A, B, CL, and CU can be programmed as input or output.
- Example: Configure port A as input, B as output, and all the bits of port C as output assuming a base address of 50h
- Control word should be 1001 0000b = 90h

```
PORTA  EQU 50h
PORTB  EQU 51h
PORTC  EQU 52h
CNTREG EQU 53h
MOV AL, 90h
OUT CNTREG,AL
IN AL, PORTA
OUT PORTB, AL
OUT PORTC, AL
```


Mod 0 Simple I/O



GURE 15.5
terfacing 8255A I/O Ports in Mode 0



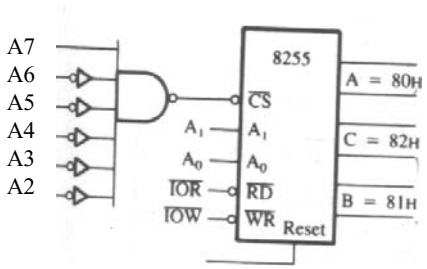
Initialize this device with the appropriate Control Word. Read from PORT C_L and display at PORT A.

PORT A 8000h
PORT B 8001H
PORT C 8002H
CONTROL 8003H

Be careful Memory I/O!

```
MOV AL,83H
MOV BX,8003H
MOV [BX],AL
MOV BX,8002H
MOV AL,[BX]
AND AL,0FH
DEC BX
DEC BX
MOV [BX],AL
```

BSR Mode



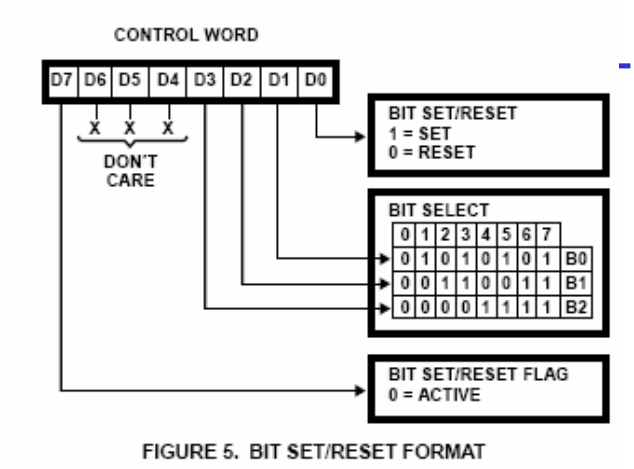


FIGURE 5. BIT SET/RESET FORMAT

- Concerned with the eight bits of port C only which can be set or reset by writing appropriate control word with D7=1
- It does not alter the previously transmitted control word with D7=0

- Ex: Write a BSR word subroutine to set PC7 and PC3

To Set PC7 → OFH ; To set PC3 → 07H

```
MOV AL,0FH
OUT 83H,AL
MOV AL,07H
OUT 83h,AL
```


PC Interface Trainer Decoding Circuitry

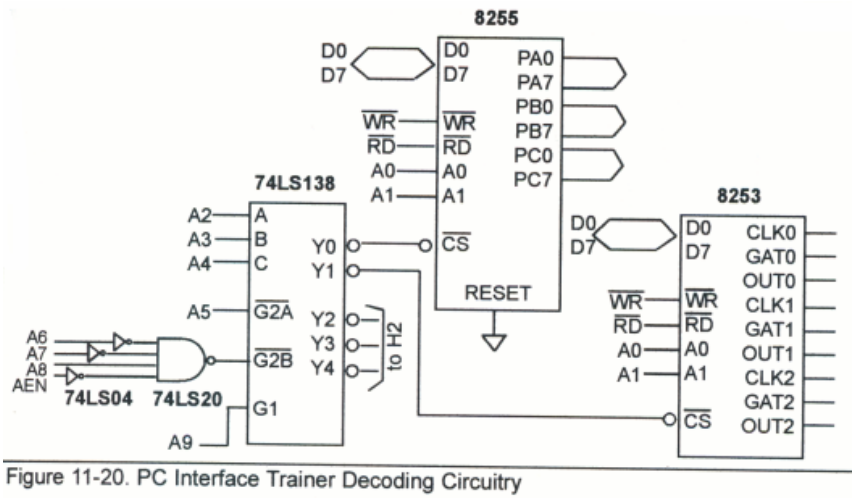
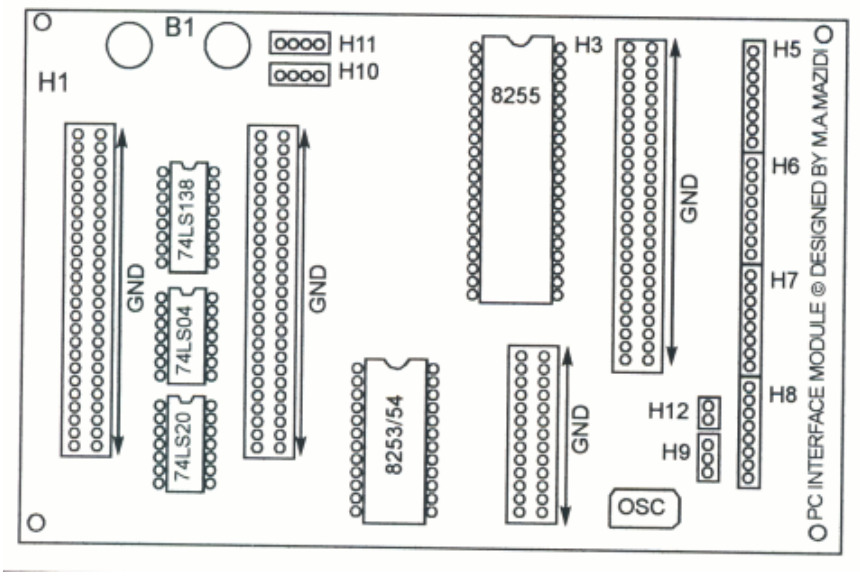


Figure 11-20. PC Interface Trainer Decoding Circuitry

PC Interface Trainer Module



Example 11.4 of Textbook

- Find the control word
  - PA = out
  - PB = in
  - PC0 – PC3 = in
  - PC4 – PC7 = out
- Program the 8255 to get data from port B and send it to port A; in addition data from PCL is sent out to the PCU
- Use port addresses 300h – 303h for the 8255 chip

Control Word:
The control word should be
1000 0011b = 83h

Program

```
B8255       EQU   300h
CNTL        EQU   83h

MOV DX,B8255+3
MOV AL,CNTL
OUT DX,AL
MOV DX,B8255+1
IN AL,DX
MOV DX,B8255
OUT DX,AL
MOV DX,B8255+2
IN AL,DX
AND AL,0Fh
MOV CL,4
ROL AL,CL
OUT DX,AL
```


Example 11-6 Textbook

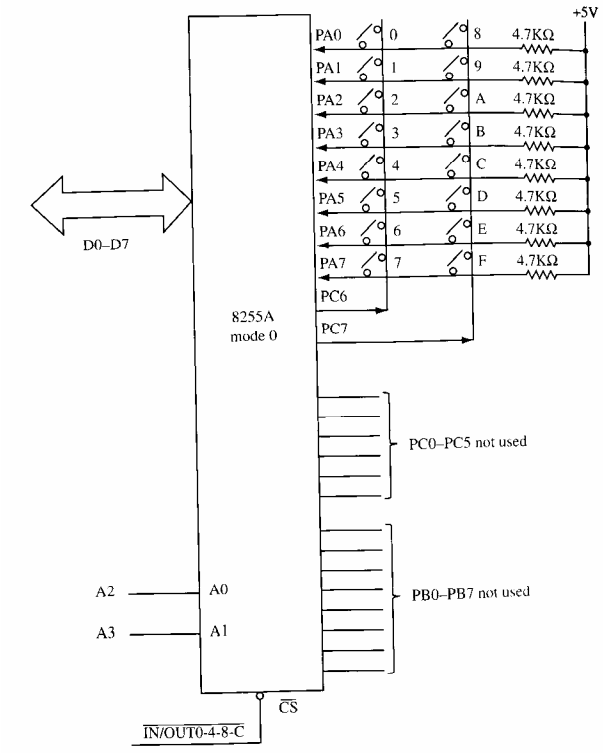
- Assume 8255 has a base address 300h
- Write a program to toggle all bits of port A continuously with a ¼ sec. Delay
- Use int 16h to exit if there is a key press

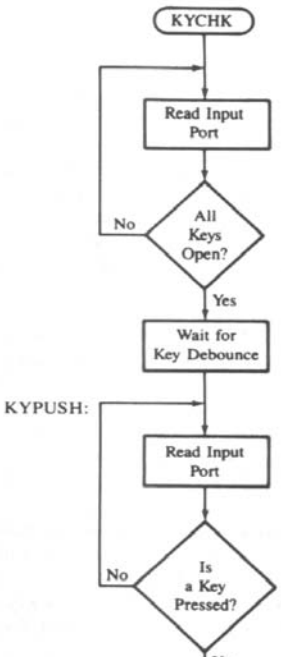
```
            MOV   DX,303h
            MOV AL,80h
            OUT DX,AL
AGAIN:      MOV DX,300h
            MOV AL,55h
            OUT DX,AL
            CALL QSDELAY
            MOV AL,0AAh
            OUT DX,AL
```

Example Contd

```
                    CALL  QSDELAY
                    MOV AH,01
                    INT 16h
                    JZ AGAIN
                    MOV AH,4Ch
                    INT 21h
; to create  a processor independent delay IBM made PB4 of port 61h to toggle every
    15.085 microsec. (for 286 and higher processors)
QSDELAY             PROC  NEAR
                    MOV CX,16572  ;16572*15.085 microsec = ¼ s
                    PUSH  AX
              W1:   IN AL,61h
                    AND AL,00010000b
                    CMP AL,AH
                    JE   W1
                    MOV AH,AL
                     LOOP W1
                    POP AX
                    RET
 QSDELAY            ENDP
```


Mode 0 Design Example - Interfacing a matrix keyboard

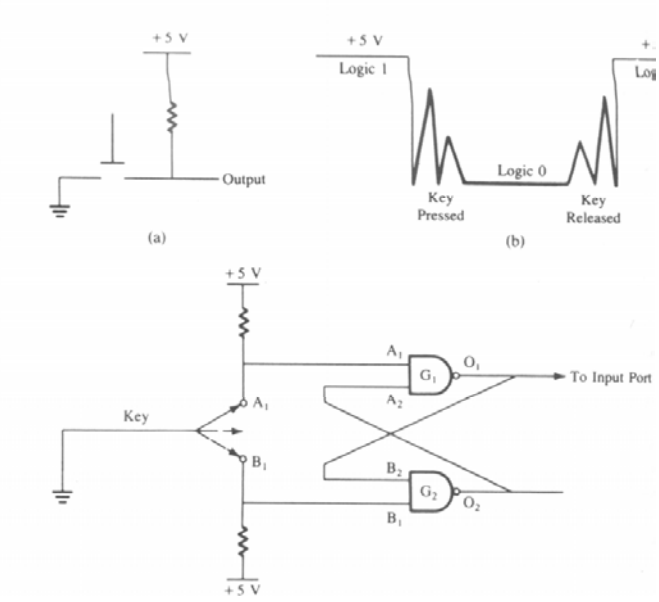




Key Debounce

✓When a mechanical push button key is pressed or released the metal contacts of the key momentarily bounce before giving a steady-state reading. Therefore it is necessary that the bouncing of the key not be read as an input.

✓Key debounce can be eliminated using either software or hardware.



Key debounce technique in hardware


```
                        ;Function:      Scan the keyboard shown in Fig. 8.14
                        ;               and return with the encoded key
                        ;               value in register AL.
                        ;Inputs:        none
                        ;Outputs:       hex key value in AL.
                        ;Calls:         10 ms delay procedure for debouncing
                        ;Destroys:      AX and flags

                        ;*******
                        ;  Set up segment to store key values
                        ;*******

0000                    KEY_CODE        SEGMENT BYTE
0000  00 01 02 03 04    COL1            DB      0,1,2,3,4
0005  05 06 07                         DB      5,6,7
0008  08 09 0A 0B 0C    COL2            DB      8,9,0AH,0BH,0CH
000D  0D 0E 0F                         DB      0DH,0EH,0FH
0010                    KEY_CODE        ENDS

0000                    CODE            SEGMENT BYTE
                                        ASSUME  CS:CODE,DS:KEY_CODE

                        ;*******
                        ;Program equates
                        ;*******

= 00F0                  PORT_A          EQU     00H             ;PPI port A address (see Fig. 8.12)
= 00F2                  PORT_C          EQU     08H             ;PPI port C address
= 00BF                  COL_1_LOW       EQU     10111111B       ;PC6 low
= 007F                  COL_2_LOW       EQU     01111111B       ;PC7 low
= 003F                  BOTH_COL_LOW    EQU     00111111B       ;PC6 and PC7 low
= 00FF                  KEY_UP          EQU     0FFH            ;Input 0FFH when no keys are down
= 16FA                  T1              EQU     8B82H           ;~ 10 ms time delay assuming 25 MHz 80
```

```
;*******
;   10 ms time delay for debouncing
;*******

DELAY           PROC    NEAR
                MOV     CX,T1
COUNT:          LOOP    COUNT
                RET
DELAY           ENDP


;*******
;  Main program begins here
;*******

KEYBOARD        PROC    NEAR
                PUSH    DS                      ;Save registers about to be used
                PUSH    CX
                PUSH    SI
                MOV     AX,KEY_CODE             ;Point DS to the key codes
                MOV     DS,AX

;Wait for previous key to be released

                MOV     AL,BOTH_COL_LOW         ;Scan both columns
                OUT     PORT_C,AL               ;Column lines on PC6 and PC7
POLL1:          IN      AL,PORT_A               ;Read keyboard
                CMP     AL,KEY_UP               ;All keys up?
                JNE     POLL1                   ;No - so wait
                CALL    DELAY                   ;Yes - wait for bounce on release

;Wait for a new key to be pressed

POLL2:          IN      AL,PORT_A               ;Read keyboard
                CMP     AL,KEY_UP               ;Any keys down?
                JE      POLL2                   ;No - so wait
                CALL    DELAY                   ;Yes - wait for bounce
```

```
                        ;See if the key is in column 1

0024  B0 BF                             MOV     AL,COL_1_LOW        ;Test for column 1
0026  E6 08                             OUT     PORT_C,AL           ;PC6 low
0028  E4 00                             IN      AL,PORT_A           ;Read column 1 keys
002A  3C FF                             CMP     AL,KEY_UP           ;Any key down?
002C  74 07                             JE      CHECK_COL_2         ;No - check for column 2
002E  8D 36 0000 R                      LEA     SI,COL1             ;Yes - point SI at the key values 0-7
0032  EB 0F 90                          JMP     LOOKUP              ;Now lookup code

                        If not column 1 then column 2

0035  B0 7F             CHECK_COL_2:    MOV     AL,COL_2_LOW        ;Test for column 2
0037  E6 08                             OUT     PORT_C,AL           ;PC7 low
0039  E4 00                             IN      AL,PORT_A           ;Read column 2 keys
003B  3C FF                             CMP     AL,KEY_UP           ;Any key down?
003D  74 DC                             JE      POLL2               ;No - false input so repeat
003F  8D 36 0008 R                      LEA     SI,COL2             ;Yes - point SI at key values 8-F

                        ;Now lookup the key's value and store in AL

0043  D0 D8             LOOKUP:         RCR     AL,1                ;Rotate keyboard input code right
0045  73 03                             JNC     MATCH               ;If 0 key is found - so retrieve it
0047  46                                INC     SI                  ;No - advance pointer to next value
0048  EB F9                             JMP     LOOKUP              ;Repeat the loop

004A  8A 04             MATCH:          MOV     AL,[SI]             ;Get the key code
004C  5E                                POP     SI                  ;Restore all registers
004D  59                                POP     CX                  ;(except AX and flags)
004E  1F                                POP     DS
004F  C3                                RET
0050                    KEYBOARD        ENDP
0050                    CODE            ENDS
                                        END     KEYBOARD
```

8 Digit LED

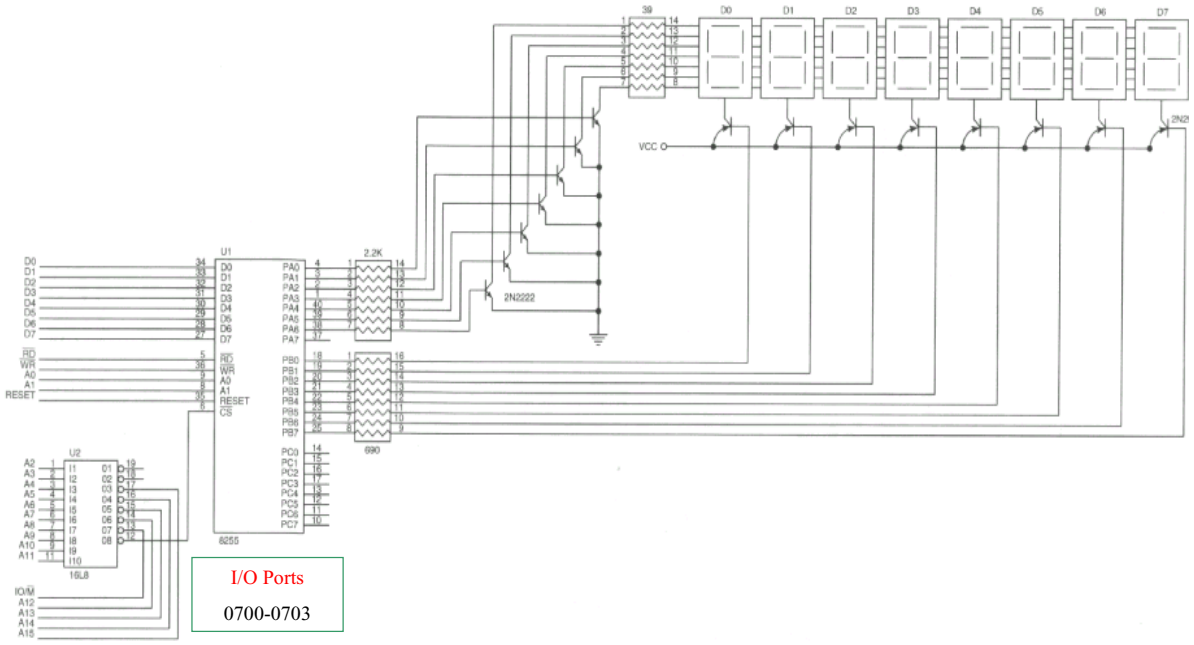
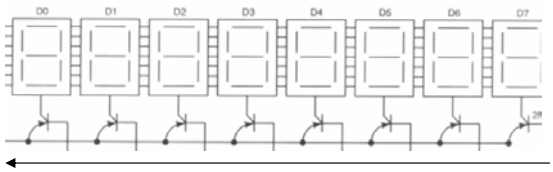


FIGURE 11–20 An 8-digit LED display interfaced to the 8088 microprocessor through an 82C55 PIA.

8 Digit LED

```
; INIT 8255
MOV AL,10000000B
MOV DX,703H
OUT DX,AL

;SETUP REGISTERS TO DISPLAY
MOV BX,8
MOV AH,7FH
MOV SI,OFFSET MEM-1
MOV DX.701H

; DISPLAY 8 DIGITS

DISP1: MOV AL,AH  ;select digit
OUT DX,AL
DEC DX            ;address PORT A
MOV AL,[BX+SI]
OUT DX,AL
CALL DELAY   ;wait 1 ms
ROR AH,1
INC DX            ;address PORT B
DEC BX            ;adjust count
JNZ DISP1

RET
```



Motor Application

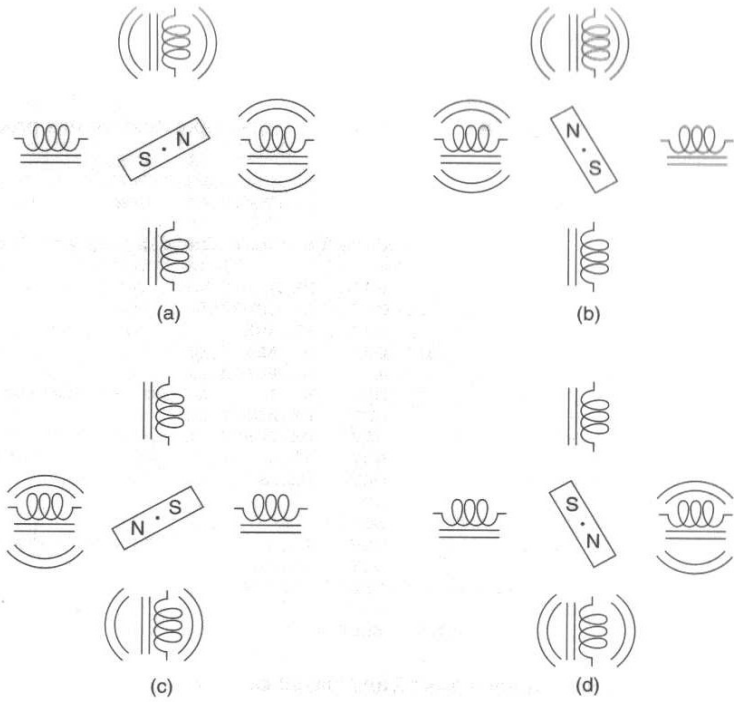


FIGURE 11–22 The stepper motor showing full-step operation. (a) 45° (b) 135° (c) 225° (d) 315°.

Motor Application

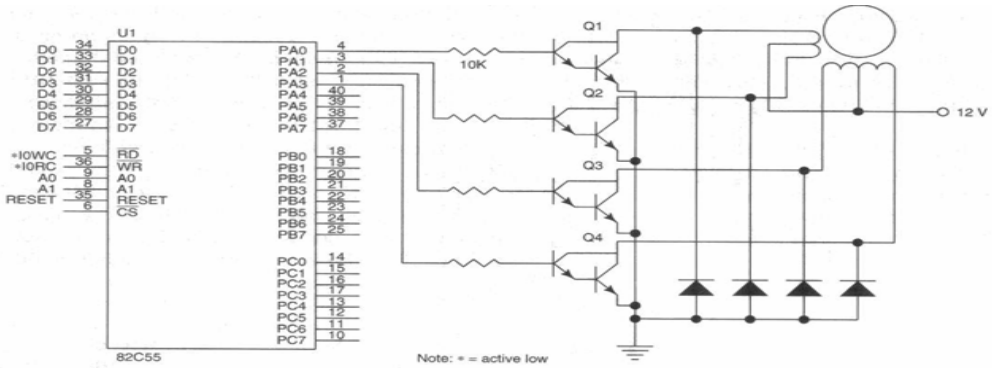


FIGURE 11–23 A stepper motor interfaced to the 82C55. This illustration does not show the decoder.

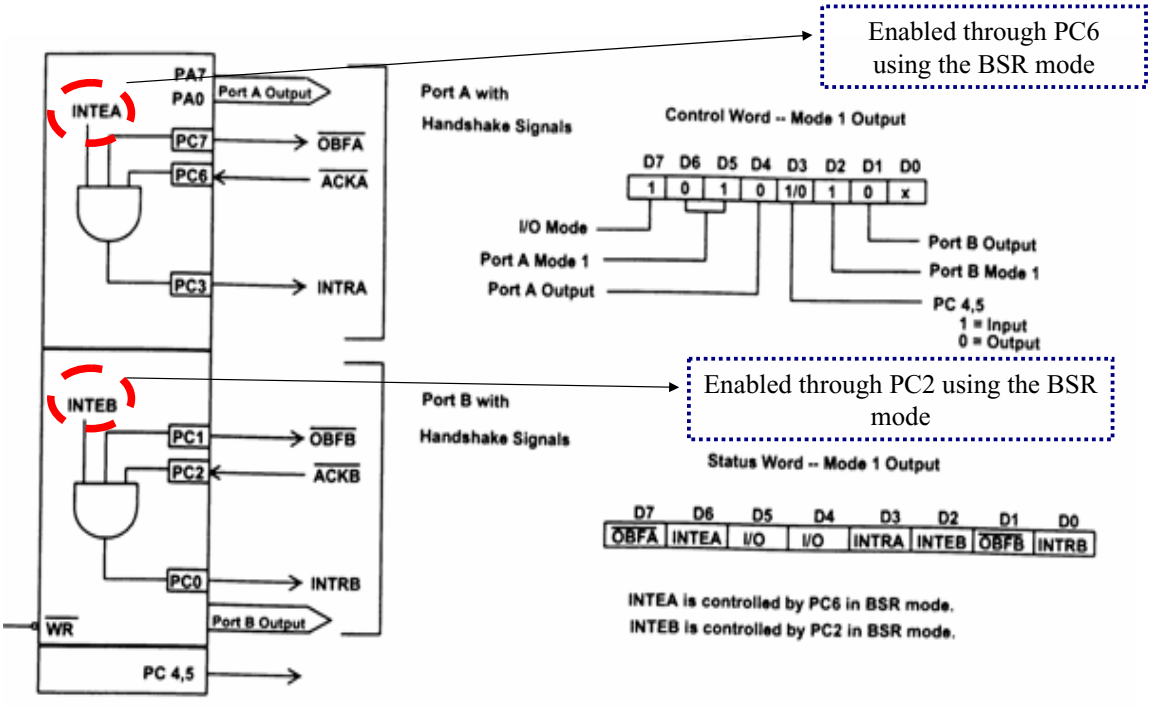
Rotate the motor clockwise if A15 is 1
else rotate counterclockwise

```
STEP:   PROC NEAR
        MOV AL, POS ;current position
        CMP CX, 8000H ;rotation amount
        JA RH
        CMP CX,0
        JE STEP_OUT
STEP1:  ROL AL,1
        OUT PORT, AL
        CALL DELAY
        LOOP STEP1
        JMP STEP_OUT
RH:     AND CX, 7FFFH; Clr 15
RH1:    ROR AL,1
        OUT PORT,AL
        CALL DELAY
        LOOP RH1
STEP_OUT: MOV POS,AL
        RET
```

Mode 1: I/O with Handshaking Capability

- Handshaking refers to the process of communicating back and forth between two intelligent devices
- Example: Process of communicating with a printer
  - a byte of data is presented to the data bus of the printer
  - the printer is informed of the presence of a byte of data to be printed by activating its strobe signal
  - whenever the printer receives the data it informs the sender by activating an output signal called ACK
  - the ACK signal initiates the process of providing another byte of data to the printer
- 8255 in mode 1 is equipped with resources to handle handshaking signals

Mode 1 Strobed Output



Mode 1 Strobed Output

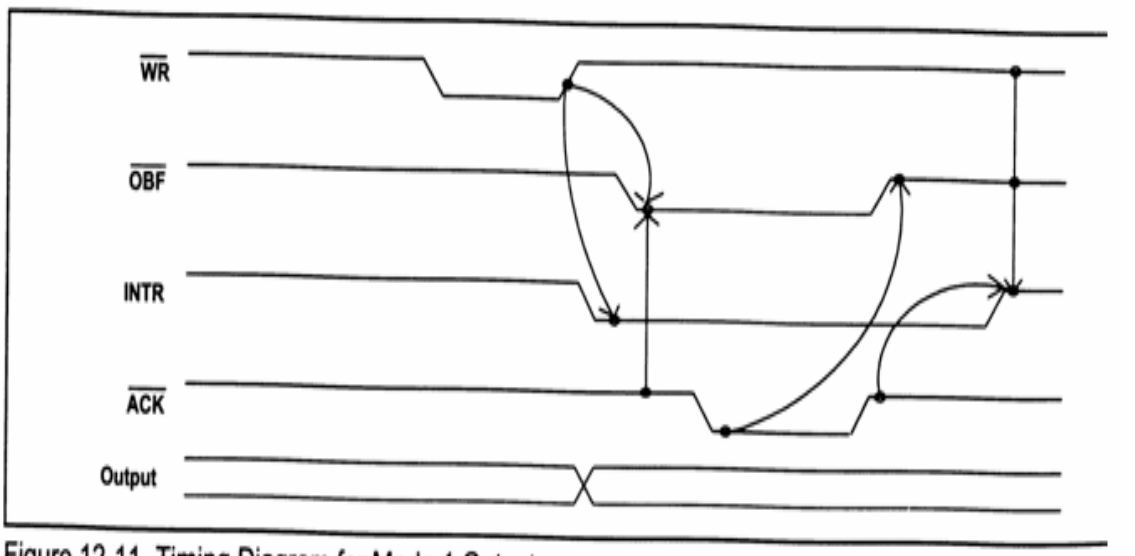


Figure 12-11. Timing Diagram for Mode 1 Output
(Reprinted by permission of Intel Corporation, Copyright Intel Corp. 1983)

Mode 1 Strobed Output Signals

- $\overline{\text{OBFa}}$ (output buffer full for port A)
  - indicates that the CPU has written a byte of data into port A
  - must be connected to the STROBE of the receiving equipment
  - Goes back high again after ACK'ed by the peripheral.
- $\overline{\text{ACKa}}$ (acknowledge for port A)
  - through ACK, 8255 knows that data at port A has been picked up by the receiving device
  - 8255 then makes $\overline{\text{OBFa}}$ high to indicate that the data is old now. $\overline{\text{OBFa}}$ will not go low until the CPU writes a new byte of data to port A.
- INTRa (interrupt request for port A)
  - it is the rising edge of ACK that activates INTRa by making it high. INTRa is used to get the attention of the microprocessor.
  - it is important that INTRa is high only if INTEa, OBFa, ACKa are all high
  - it is reset to zero when the CPU writes a byte to port A
- The 8255 enables the monitoring the status signals INTR, OBF, and INTE for both ports A and B. This is done by **reading port C** into the accumulator and testing the bits. This feature allows the implementation of polling

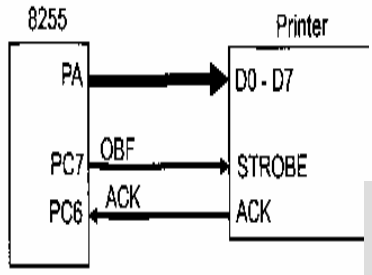
Interrupts vs Polling

CPU services devices in two ways:

Interrupts and Polling

- In the interrupt method, whenever the device needs the service of the CPU, the device informs the CPU by sending an interrupt signal.
  - The CPU interrupts whatever it is doing and serves the request
  - The advantage of interrupts is that the CPU can serve many devices
  - Each receives a service based on its priority
  - Disadvantage of interrupts is that they require more hardware and software
- In polling, CPU monitors continuously a status condition and when the conditions are met, it will perform the service.
  - In contrast, polling is cheap and requires minimal software
  - But it ties down the CPU

Example



OBF signal can also be checked (easier)instead of enabling INTA

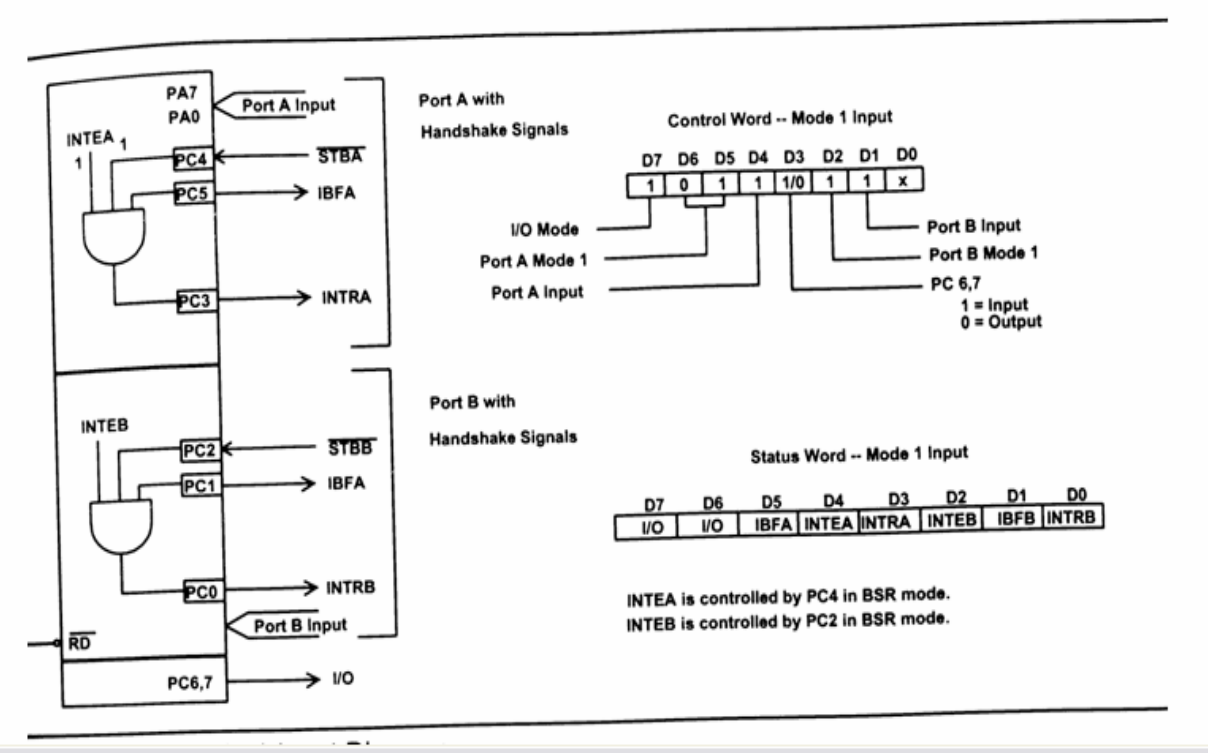
From the data segment:

```
MY_DATA   DB    "Hi. How are you?",CR,LF
          DB    "I am fine. How are you?",CR,LF,"$"
PA        EQU   300H          ;port A address
PB        EQU   301H          ;port B address
PC        EQU   302H          ;port C address
CWP       EQU   303H          ;control word port
LF        EQU   0AH           ;line feed
CR        EQU   0DH           ;carriage return
```

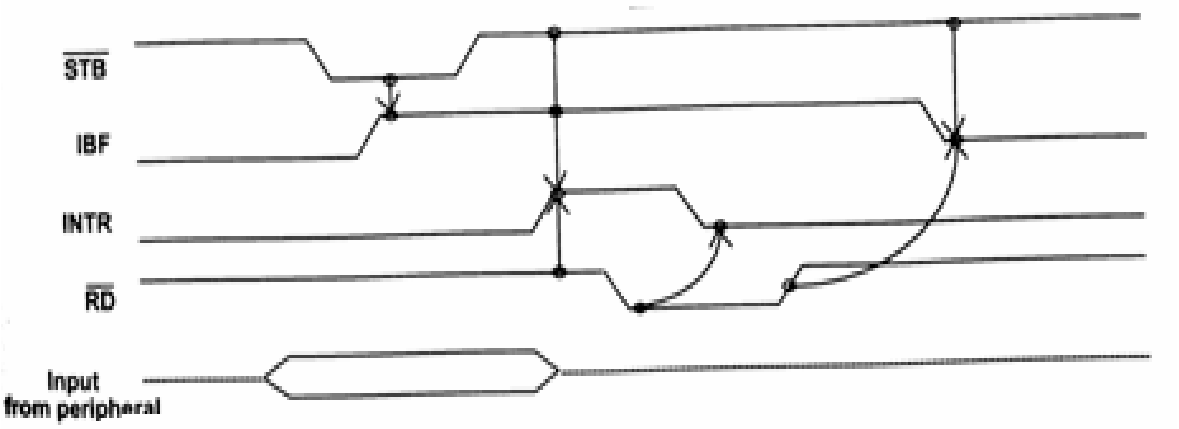
From the code segment:

```
          MOV   AL,10100000B  ;control word PA=out mode 1
          MOV   DX,CWP        ;DX =  303 control word  port
          OUT   DX,AL         ;issue control word
          MOV SI,OFFSET MY_DATA
AGAIN:    MOV AH,[SI]
          CMP AH,'$'
          JZ OVER
BACK:     MOV DX,PC         ;check port C
          IN AL,DX
          AND AL, 80        ;check OBF
          JZ BACK
          MOV DX,PA
          MOV AL,AH
          OUT DX,AL
          INC SI
          JMP AGAIN
OVER:     ...               ;go back to DOS
```

Mode 1 Strobed Input



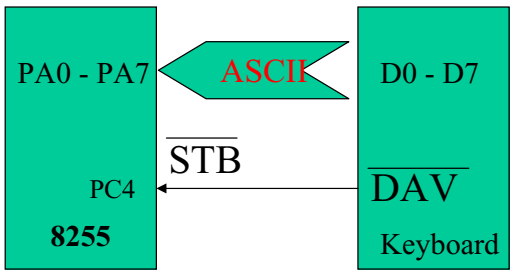
Mode 1 Strobed Input



Mode 1 Input Ports with Handshaking Signals

- $\overline{\text{STB}}$
  - When an external peripheral device provides a byte of data to an input port, it informs the 8255 through the STB pin. STB is of limited duration
- IBF (Input Buffer Full)
  - In response to STB, the 8255 latches into its internal register the data present at PA0-PA7 or PB0-PB7.
  - Through IBF it indicates that it has latched the data but it has not been read by the CPU yet
  - To get the attention of the CPU, IBF activates INTR
- INTR
  - Falling edge of RD makes INTR low
  - The RD signal from the CPU is of limited duration and when it goes high the 8255 in turn makes IBF inactive by setting it low
  - IBF in this way lets the peripheral know that the byte of data was latched by the 8255 and read into the CPU as well.
- The two flip flops INTEA and INTB are set/reset using the BSR mode. The INTEA is enabled or disabled through PC4 and INTEB is enabled or disabled through PC2.

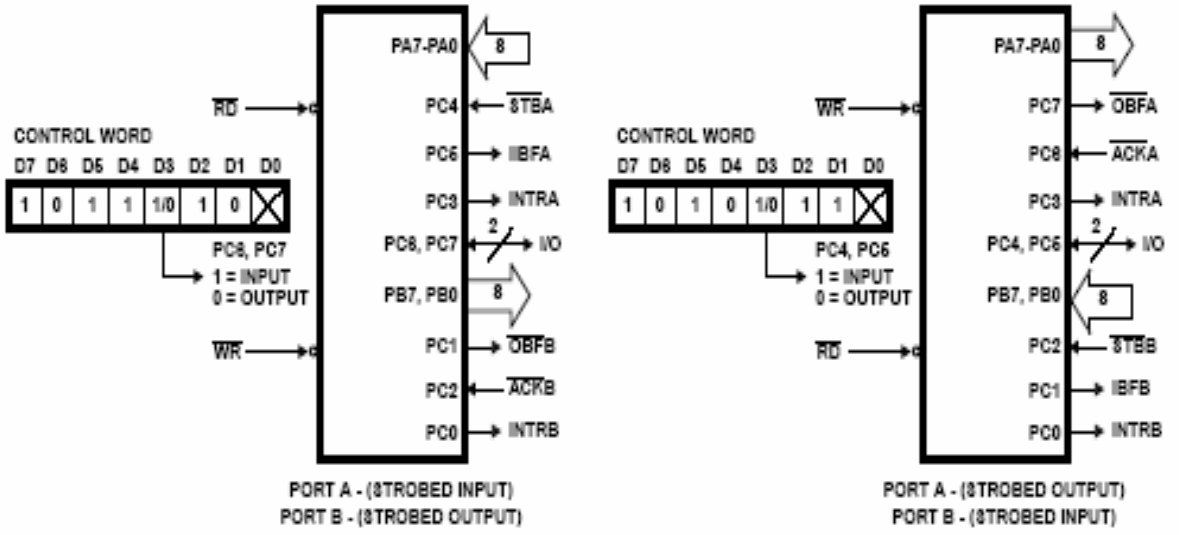
Mode 1 Strobed Input Example



Write a procedure that reads data from the keyboard each time a key is typed.

```
BIT5    EQU 20h
PORTC   EQU 22h
PORTA   EQU 20h
READ    PROC NEAR
        IN AL, PORTC
        TEST AL, BIT5 ;check on IBF?
        JZ  READ
        IN AL, PORTA
        RET
READ    ENDP
```

Mode 1 Other Configurations



Combinations of Mode 1: Port A and Port B can be individually defined as input or output in Mode 1 to support a wide variety of strobed I/O applications.

FIGURE 10. COMBINATIONS OF MODE 1

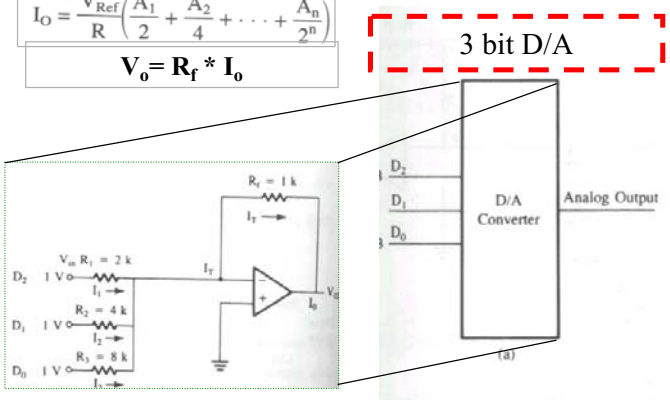
Interfacing DAC to a PC

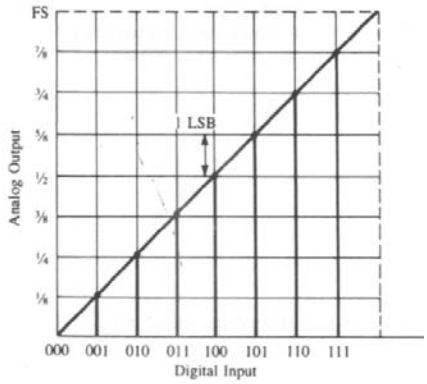
- The digital to analog converter (DAV) is a device widely used to convert digital pulses to analog signals.
- The first criterion to judge a DAC is its resolution which is the a function of the number of binary inputs
- The number of analog input levels is 2^n where n is the number of data bit inputs.
- Therefore the 8 input DAC such as the MC 1408 (DAC 808) provides 256 discrete voltage (or current) levels of output.
- $I_{out} = I_{ref}$ (D7/2 + D6/4 + D5/8 + D4/16 + D3/32 + D2/64 + D1/128 + D0 / 256)

DAC BASICS

$$I_O = \frac{V_{Ref}}{R}\left(\frac{A_1}{2} + \frac{A_2}{4} + \cdots + \frac{A_n}{2^n}\right)$$

$$V_o = R_f * I_o$$

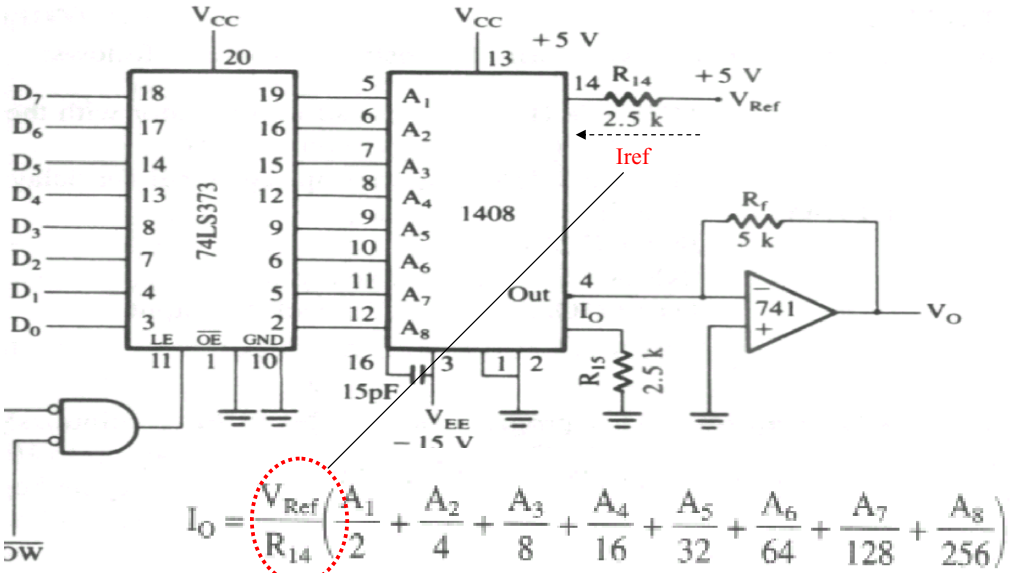




Calculate the values of the LSB, MSB and full scale output for an 8 bit DAC for the 0-10v range?

1. LSB = $1/2^8 = 1/256$
   for 10 v, LSB = 10/256
2. MSB = ½ full scale = 5v
3. Full Scale Output = (Full Scale Value – 1 LSB)
   = 10 v – 0.039v = 9.961v

Interfacing DAC via 8255



$$I_O = \frac{V_{Ref}}{R_{14}}\left(\frac{A_1}{2} + \frac{A_2}{4} + \frac{A_3}{8} + \frac{A_4}{16} + \frac{A_5}{32} + \frac{A_6}{64} + \frac{A_7}{128} + \frac{A_8}{256}\right)$$

For ex:

If $R_{14} = 2.5$

$$I_O = \frac{5\text{ V}}{2.5\text{ k}}\left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \frac{1}{128} + \frac{1}{256}\right)$$

$$= 2\text{ mA}\ (255/256)$$

$$= 1.992\text{ mA}$$

Example

- Assume that R=5K and Iref= 2ma calculate Vout for: 10011001

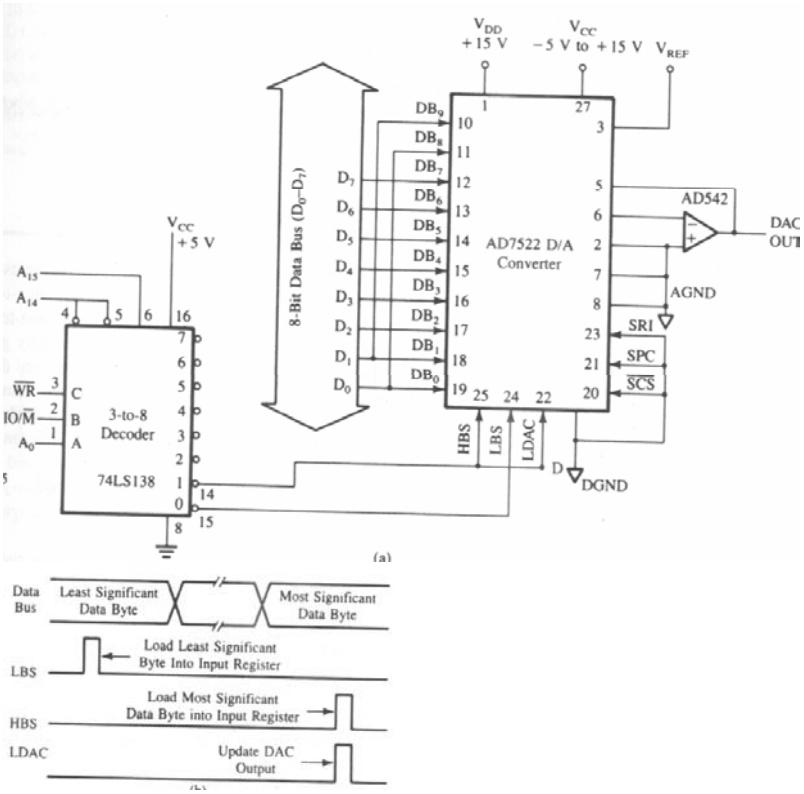
Iout = 2mA (153/255) = 1.195 mA

Vout = 1.195 x 5 K = 5.975V

Example - Generate a Stair-Step Ramp

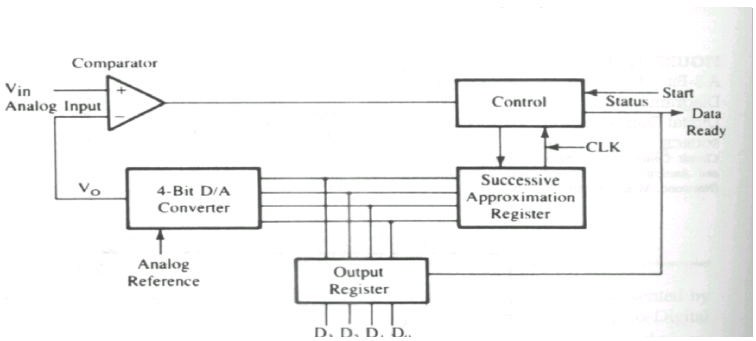
```
        MOV   AL,80H
        MOV   DX,303H
        OUT   DX,AL
A1:     MOV   AH,01
        INT   16H
        JNZ   STOP
        SUB   AL,AL
        MOV   DX,300H
A2:     OUT   DX,AL
        INC   AL
        CMP   AL,0
        JZ    A1
        MOV   CX,02FFH
WT:     LOOP  WT
        JMP   A2
        MOV   AH,4CH
        INT   21H
```

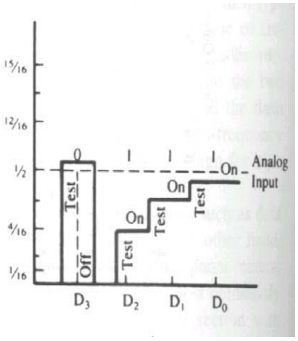
10 bit D/A operation



Interfacing ADC to a PC

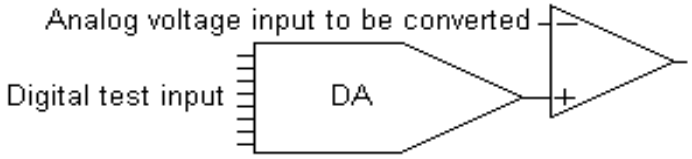
- A physical quantity (temperature, pressure, humidity, velocity) is converted to electrical (voltage, current) signals using a device called a transducer (sensor)
- We need an analog-to-digital converter (ADC) to translate the analog signals to digital numbers so that the PC can read them
- The techniques for ADC are
  - Successive Approximation Technique
    - Conversion Speed is important (Critical Conversion Time)



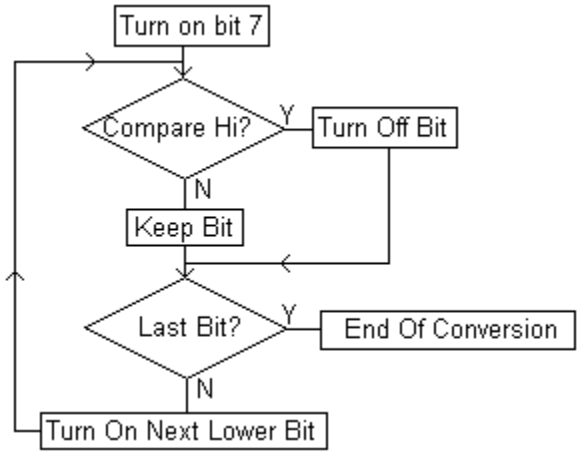


  - Integrating Type Converters
    - Conversion Accuracy is important

Successive Approximation Technique



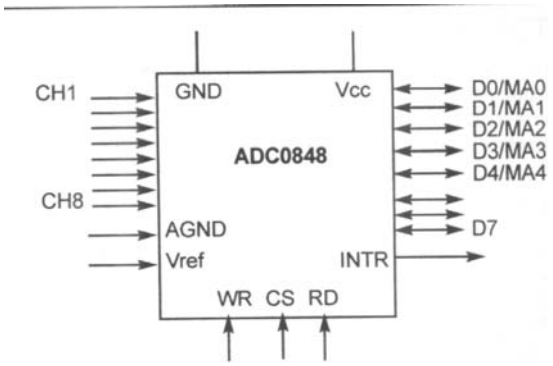
:



Contd

Test Bit	DA Binary Value	Decimal	(5 * DAvalue)/256	Comparison Result
10000000	10000000	128	2.5	low - keep bit
01000000	11000000	192	3.75	high - drop bit
00100000	10100000	160	3.125	low - keep bit
00010000	10110000	176	3.4375	high - drop bit
00001000	10101000	168	3.28125	high - drop bit
00000100	10100100	164	3.203125	low - keep bit
00000010	10100110	166	3.2421875	high - drop bit
00000001	10100101	165	3.22265625	high - drop bit

ADC0848



MA0 – MA4 (multiplexed address)

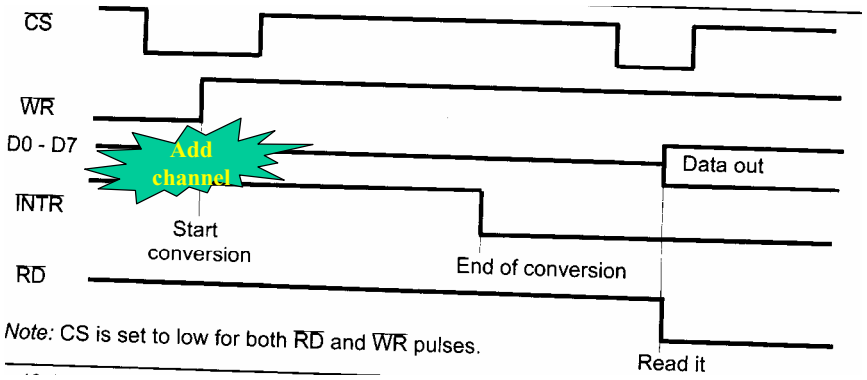
Table 12-11: ADC0848 Analog Channel Selection (Single-Ended Mode)

Selected Analog Channel	MA4	MA3	MA2	MA1	MA0
CH1	0	1	0	0	0
CH2	0	1	0	0	1
CH3	0	1	0	1	0
CH4	0	1	0	1	1
CH5	0	1	1	0	0
CH6	0	1	1	0	1
CH7	0	1	1	1	0
CH8	0	1	1	1	1

Note: Channel is selected when CS = 0, RD =1, and an L-to-H pulse is applied to WR.

- ADC 0848
  - –conversion time less than 110 μs
  - –RD active low: ADC converts the analog input to its binary equivalent and holds it in an internal register; with RD from high to low the data in the register shows at the D0-D7 pins
  - –WR start conversion: low to high input to inform the 804 to start the conversion process
  - –when the data conversion is complete, the INTR pin is forced low by the 804
  - –after INTR goes low, we make CS = 0 and send a high to low pulse to the RD pin to get the data out of the 804

ADC operation



1. Make CS = 0 and send a low to high pulse to the WR pin
2. Keep monitoring the INTR pin. If INTR is low, the conversion is finished and e can go to the next step
3. We then make CS = 0 again and send a high to low pulse to the RD pin to get the data pout of the 804 chip

ADC 804 Pins

- Vin (+) and Vin (-) are the differential analog inputs
- Vin = Vin (+) - Vin (-) ; to set this mode use MA4 and MA3 low.
- Vref is an input voltage used as the reference voltage

Vref (volts)	Vin (volts)	Step Size (mv)
5	0 to 5	5/256 = 19.53
4	0 to 4	4/256 = 15.62
3	0 to 3	3/256 = 11.71
2	0 to 2	2/256 = 7.81
1	0 to 1	1/256 = 3.90

Dout = Vin / (step size)

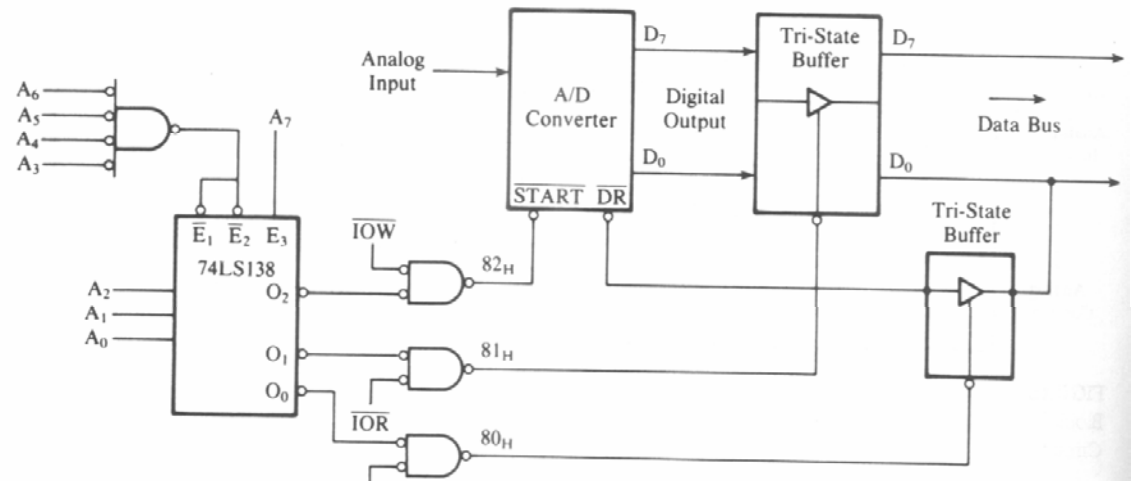
For given ADC, Vref = 2.56. Calculate the D0-D7 output if the analog input is

a) 1.7 v

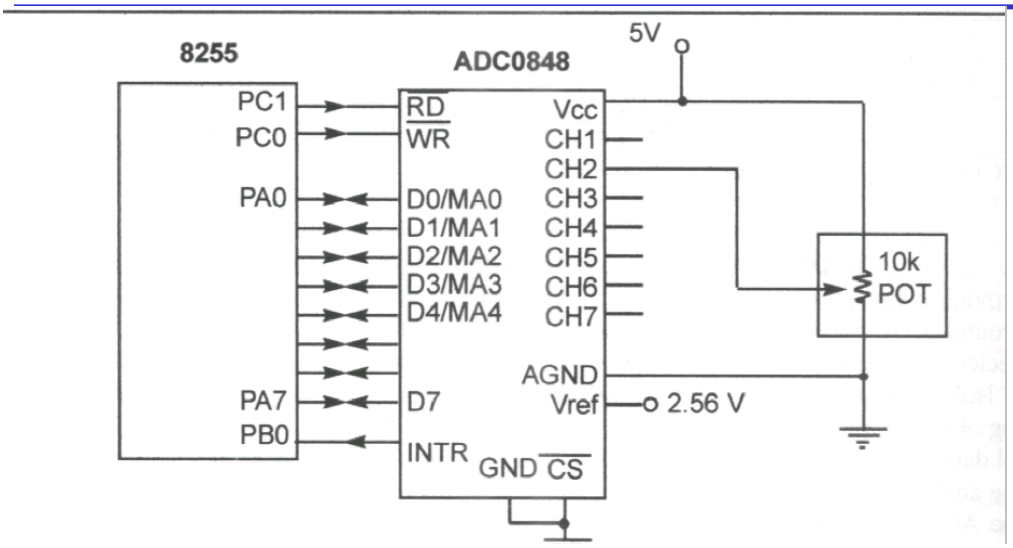
b) 2.1 v

Answer(a): Step size is 2.56/256=10mv
Dout = 1.7/10mv =170
170 is 10101011

Using A/D with status check



Connecting the 804



PA0-PA7 to DO-D7 of ADC	CHANNEL SELECTION(out), DATA READ (in)
PB0 TO INTR	PORT B AS INPUT
PC0 TO WR	PORT C AS OUTPUT
PC1 TO RD	PORT C AS OUTPUT

Required Steps

- CS=0 WR=0
  - Provide the address of the selected channel on DB0 – DB7
  - Apply a WR pulse
  - Channel 2 address is 09h, Channel 1 address is 08h , etc.
  - Not only we select the channel but conversion starts!
- While WR=1
  - Keep monitoring INTR asserted low
  - When INTR goes low, conversion finished
- After INTR becomes low
  - CS=0 and WR=1 and apply a low pulse to the RD pin to get the data from the 848 IC chip

Example

```
MOV AL,82h   ;PA=out PB=in PC=out
MOV DX,CNT_PORT
OUT DX,AL
MOV AL,09  ;channel 2 address
MOV DX,PORT_A
OUT DX,AL
MOV AL,02  ;WR=0 RD=1
MOV DX,PORT_C   ; not only selects channel but also
                ; starts conversion
OUT DX,AL
CALL DELAY          ;few microsecs
MOV AL,03           ; WR=1 RD=1
OUT DX,AL
CALL DELAY
MOV AL,92h          ; PA=in PB=in PC=out
MOV DX,CNT_PORT
OUT DX,AL
```


Example

```
MOV DX,PORT_B
B1: IN AL,DX
AND AL,01
CMP AL,01
JNE B1
MOV AL,01  ;RD=0
MOV DX,PORT_C
OUT DX,AL
MOV DX,PORT_A
IN AL,DX  ;get the converted data
```

Weeks 14-15
Interrupt Interface of the 8088 and 8086 Microprocessors

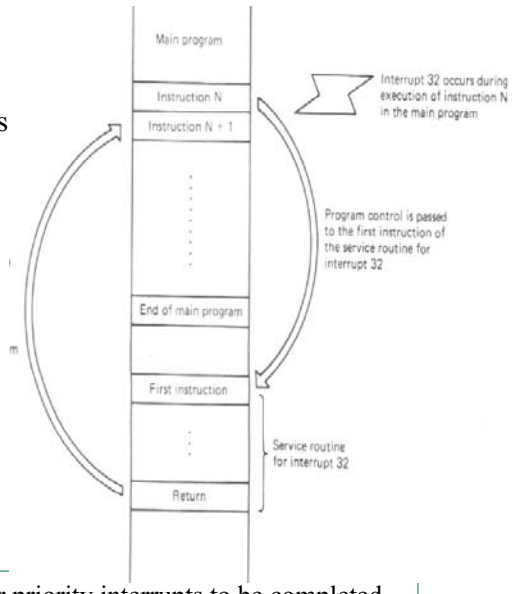
INTERRUPT INTERFACE

Interrupts provide a mechanism for quickly changing program environment.

The section of the program which the control is passed: Interrupt Service Routine, ex: For printers it is the printer driver.

8088 and 8086 interrupts:

- ✓ External Hardware Interrupts
- ✓ Nonmaskable Interrupt
- ✓ Software Interrupts
- ✓ Internal Interrupts
- ✓ Reset



Lower priority interrupts need to wait for the higher priority interrupts to be completed

8088/8086 Interrupts

- An interrupt is an external event which informs the CPU that a device needs service
- In the 8088 & 8086 there are are a total of 256 interrupts (or interrupt types)
  - INT 00
  - INT 01
  - …
  - INT FF
- When an interrupt is executed, the microprocessor automatically saves the flags register (FR), the instruction pointer (IP) and the code segment register (CS) on the stack and goes to a fixed memory location.
- In 80x86, the memory location to which an interrupt goes is always four times the value of the interrupt number
- INT 03h goes to 000Ch

Interrupt Service Routine

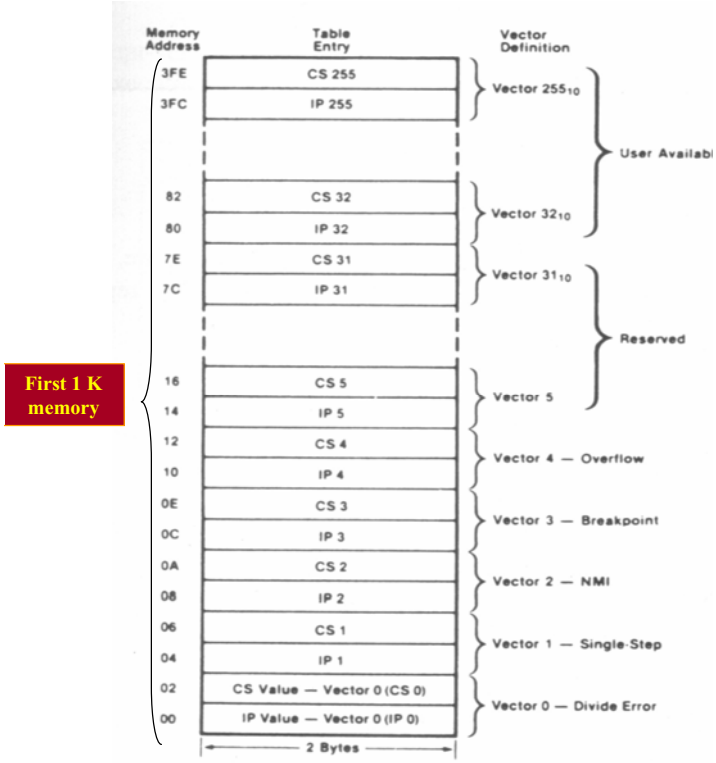
- For every interrupt, there must be a program associated with it
- This program is called an Interrupt Service Routine (ISR)
- It is also called an interrupt handler
- When an interrupt occurs, CPU runs the interrupt handler but where is the handler?
  - In the interrupt Vector Table (IVT)

INT Number	Physical Address	Contains
INT 00	00000h	IP0:CS0
INT 01	00004h	IP1:CS1
INT 02	00008h	IP2:CS2
. . .	. . .	. . .
INT FF	003FCh	IP255:CS255

Interrupt Vector Table

- Interrupt vector table consists of 256 entries each containing 4 bytes.
- Each entry contains the offset and the segment address of the interrupt vector each 2 bytes long.
- Table starts at the memory address 00000H.
- First 32 vectors are spared for various microprocessor operations.
- The rest 224 vectors are user definable.
- **The lower the vector number, the higher the priority.**

Interrupt Vector Table



- Contains 256 address pointers (vectors)
- These pointers identify the starting location of their service routines in program memory.
- Held as firmware or loaded as system initialization

Examples

For example: vector 50: CS and IP?

Physical Address 200 = (4 x 50) = 200 = 11001000 = C8H
000C8 contains IP: and 000CA contains CS information

- INT 12h (or vector 12)
- The physical address 30h (4 x 12 = 48 = 30h) contains
  - 0030h and 0031h contain IP of the ISR
  - 0032h and 0033h contain CS of the ISR

Interrupt Instructions

Mnemonic	Meaning	Format	Operation	Flags Affected
CLI	Clear interrupt flag	CLI	0 → (IF)	IF
STI	Set interrupt flag	STI	1 → (IF)	IF
INT n	Type n software interrupt	INT n	(Flags) → ((SP) − 2) 0 → TF,IF (CS) → ((SP) − 4) (2 + 4 · n) → (CS) (IP) → ((SP) − 6) (4 · n) → (IP)	TF, IF
IRET	Interrupt return	IRET	((SP)) → (IP) ((SP) + 2) → (CS) ((SP) + 4) → (Flags) (SP) + 6 → (SP)	All
INTO	Interrupt on overflow	INTO	INT 4 steps	TF, IF
HLT	Halt	HLT	Wait for an external interrupt or reset to occur	None
WAIT	Wait	WAIT	Wait for $\overline{\text{TEST}}$ input to go active	None

Differences between INT and CALL

- A CALL FAR instruction can jump any location within the 1 MB address range but INT nn goes to a fixed memory location in the Interrupt Vector Table to get the address of the interrupt service routine
- A CALL FAR instruction is used by the programmer in the sequence of instruction in the program but externally activated hardware interrupt can come at any time
- A CALL FAR cannot be masked but INT nn in hardware can be blocked.
- A CALL FAR saves CS:IP but INT nn saves Flags and CS:IP
- At the end of the subroutine RET is used whereas for Interrupt routine IRET should be the last statement

Interrupt Mechanisms, Types, and Priority

INTERRUPT TYPES SHOWN WITH DECREASING PRIORITY ORDER

1.Reset
2.Internal interrupts and exceptions
3.Software interrupt
4.Nonmaskable interrupt
5.Hardware interrupt

All the interrupts are serviced on priority basis. The higher priority interrupt is served first and an active lower priority interrupt service is interrupted by a higher priority one. Lower priority interrupts will have to wait until their turns come.

The section of program to which the control is passed called **Interrupt-service routine** (ISR)

Interrupt instructions

- Interrupt enable flag (IF) causes external interrupts to be enabled.
- INT n initiates a vectored call of a subroutine.
- INTO instruction should be used after each arithmethic instruction where there is a possibility of an overflow.
- HLT waits for an interrupt to occur.
- WAIT waits for TEST input to go high.

The Operation of Real Mode Interrupt

1. **The contents of the FLAG REGİSTERS are pushed onto the stack**
2. **Both the interrupt (IF) and (TF) flags are cleared. This disables the INTR pin and the trap or single-step feature. (Depending on the nature of the interrupt, a programmer can unmask the INTR pin by the STI instruction)**
3. **The contents of the code segment register (CS) is pushed onto the stack.**
4. **The contents of the instruction pointer (IP) is pushed onto the stack.**
5. **The interrupt vector contents are fetched, and then placed into both IP and CS so that the next instruction executes at the interrupt service procedure addressed by the interrupt vector.**
6. **While returning from the interrupt-service routine by the instruction IRET, flags return to their state prior to the interrupt and and operation restarts at the prior IP address.**

INT 00 (divide error)

```
MOV AL,92
SUB CL, CL
DIV CL  ; 92/0 undefined
```

```
; Also invoked if the quotient is too large to fit into the assigned register

MOV AX,0FFFh
MOV BL,2
DIV BL
```

```
; WRITE A DIVIDE ERROR ISR

Prompt db 'Division by zero attempted$'

Diverr: PUSH DX
Mov ah,09h
Mov dx, offset prompt
int 21h
POP DX
```


INT 01 (Single Step)

✱In executing a sequence of instructions, there is often a need to examine the contents of the CPU's registers and system memory.

✱This is done by executing one instruction at a time and then inspecting the registers and memory

✱This is called the tracing or the single stepping

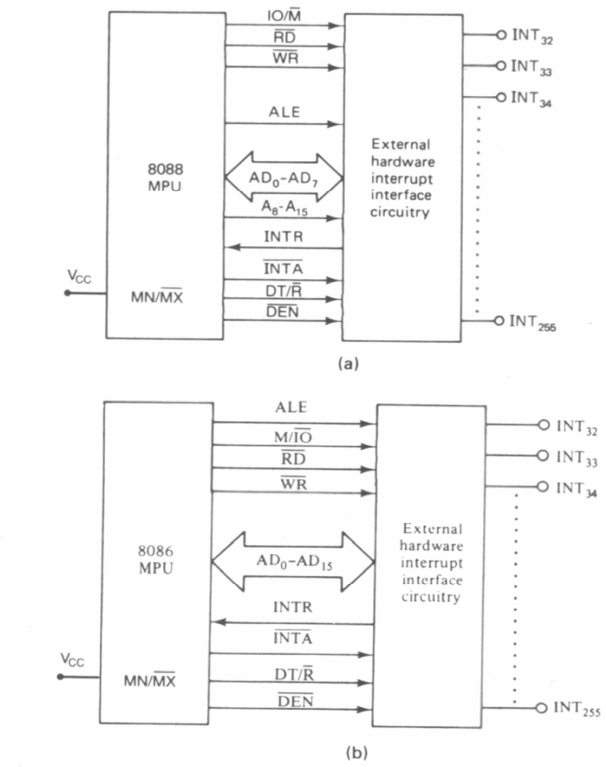
✱TF must be set (D8 of the flag register)

```
PUSHF
POP AX
OR AX,0000000100000000B
PUSH AX
POPF
```

Other Interrupts

- INT 02h
  - Intel has set aside INT 02h for the NMI interrupt
  - There is an NMI pin on the CPU
  - If the NMI pin is activated by a H signal, the CPU jumps to 00008H to fetch the CS:IP of the ISR associated with NMI
- INT 03h (breakpoint)
- INT 04H (signed number overflow) or INTO
  - If OF=0 goes to 00010h to get the address of the ISR
  - Otherwise, it is equivalent to NOP
- Example: Use debug dump command to see the IVT
  - D 0000:0000 0013

External Hardware Interrupt Interface



Minimum Mode

✓The interrupt circuitry must identify which of the pending interrupts has the highest priority.

✓Then passes its type number to the MPU

✓The MPU samples the INTR at the **last clock period** of **each** instruction execution cycle. Its active high level must be maintained.

✓When recognized INTRA generated.

External hardware-interrupt Interface

- Minimum mode hardware-interrupt interface:
  - 8088 samples INTR input during the last clock period of each instruction execution cycle. INTR is a level triggered input; therefore logic 1 input must be maintained there until it is sampled. Moreover, it must be removed before it is sampled next time. Otherwise, the same Interrupt Service is repeated twice.
  - $\overline{\text{INTA}}$ goes to 0 in the first interrupt bus cycle to acknowledge the interrupt after it was decided to respond to the interrupt.
  - It goes to 0 again the second bus cycle too, to request for the interrupt type number from the external device.
  - The interrupt type number is read by the processor and the corresponding int. CS and IP numbers are again read from the memory.

External hardware-interrupt Sequence

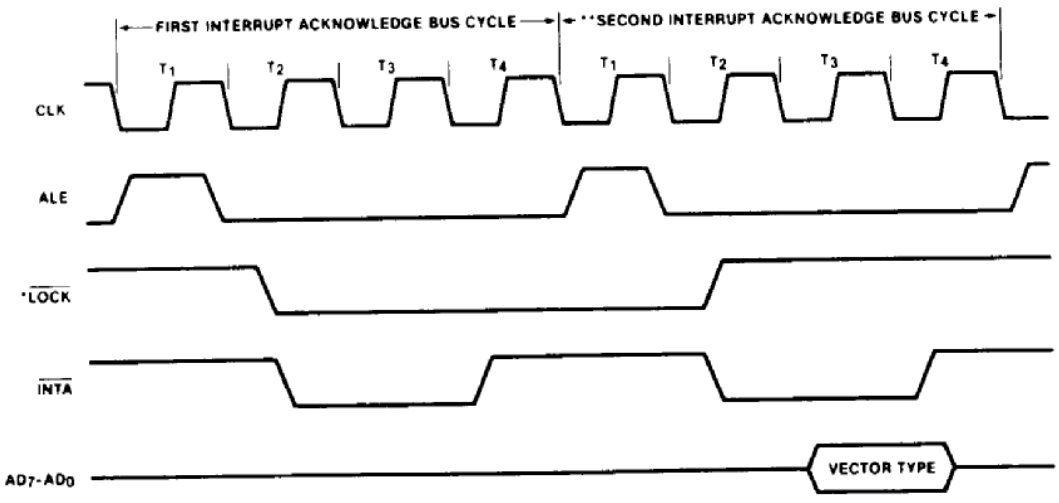


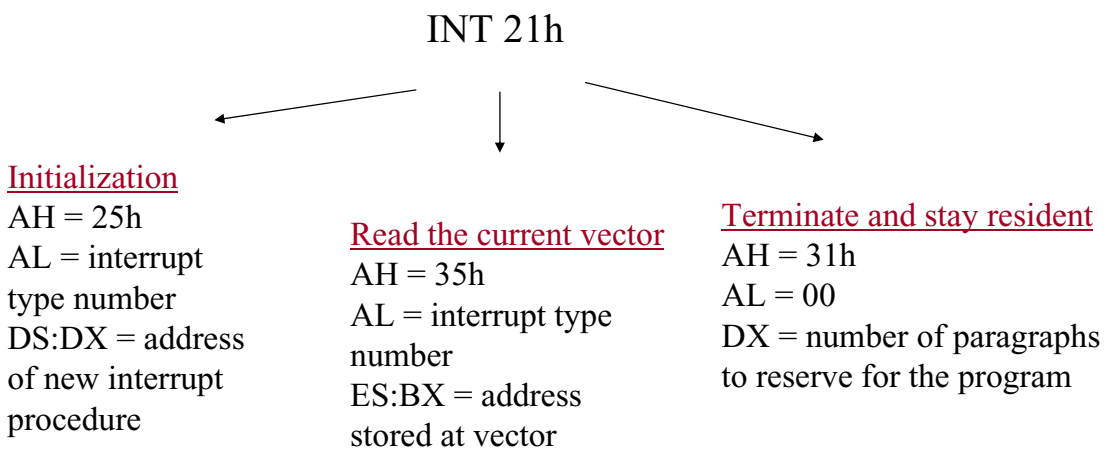
Figure 11–9 Interrupt-acknowledge bus cycle. (Reprinted by permission of Intel Corporation. Copyright/Intel Corp. 1979)

Resident Programs

- Usually non-resident program is a file, loaded from disk by DOS. Termination of such program is the passing control back to DOS. DOS frees all memory, allocated for and by this program, and stays idle to execute next program.
- Resident program passes control to DOS at the end of its execution, but leaves itself in memory whole or partially.
- Such way of program termination was called TSR - Terminate-and-Stay-Resident. So resident programs often called by this abbreviations - TSR.
- For example, TSR can watch keypresses to get passwords, INT 13h sectors operations to substitute info, INT 21h to watch and dispatch file operations and so on.
- TSR stays in memory to have some control over the processes. Usually, TSRs takes INTerrupt vectors to its code, so, when interrupt occurs, vector directs execution to TSR code.

Storing an Interrupt Vector in the Vector Table

In order to install an interrupt vector – sometimes called a hook – the assembler must address absolute memory



Example-storing Interrupt Vector

Storing an Interrupt Vector in the Vector Table

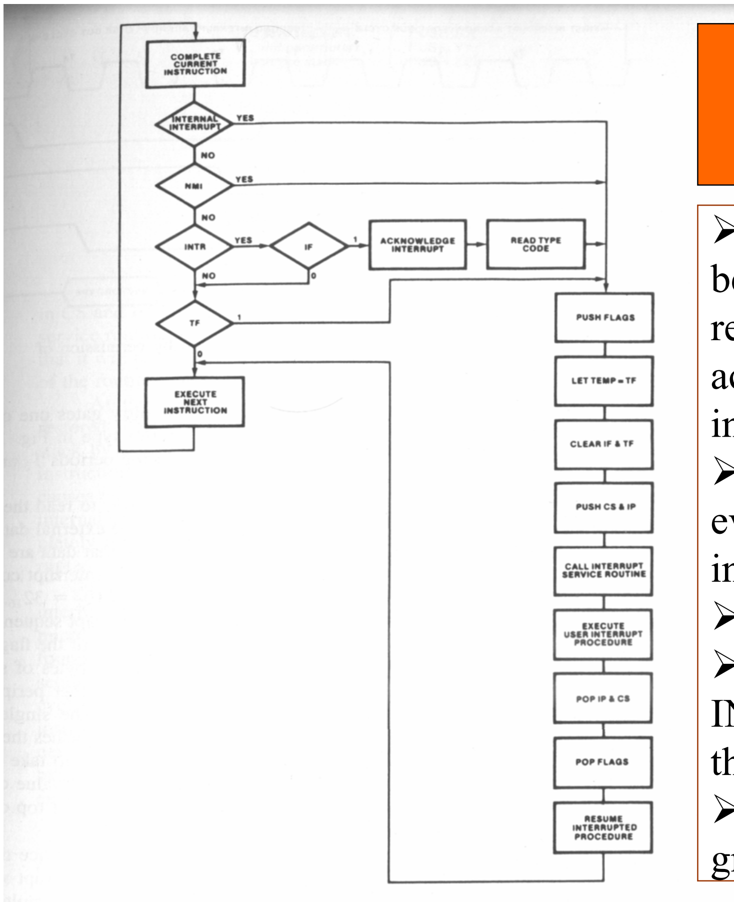
In order to install an interrupt vector—sometimes called a **hook**—the assembler must address absolute memory. Example 12–4 shows how a new vector is added to the interrupt vector table by using the assembler and a DOS function call. Here, INT 21H function call number 25H initializes the interrupt vector. Notice that the first thing done in this procedure is to save the old interrupt vector number by using DOS INT 21H function call number 35H to read the current vector. See Appendix A for more detail on DOS INT 21H function calls.

EXAMPLE 12–4

```
                            .MODEL TINY
                            .CODE
                            ;A program that installs NEW40 at INT 40H.
                            ;
                            .STARTUP
0100  EB 05                       JMP    START
0102  00000000             OLD   DD     ?
                            ;
                            ;new interrupt procedure
                            ;
0106                        NEW40 PROC   FAR

0106  CF                          IRET

0107                        NEW40 ENDP

0107                  START:
0107  8C C8                      MOV    AX,CS     ;get data segment
0109  8E D8                      MOV    DS,AX
```

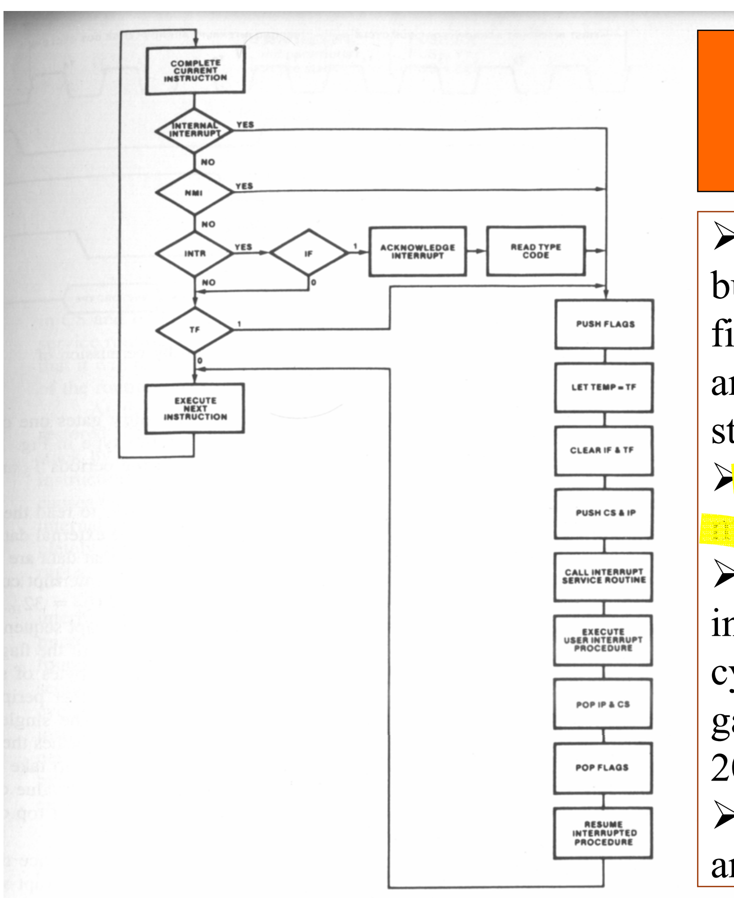
Example-storing Interrupt Vector

```
010B  B4 35                       MOV    AH,35H    ;get old interrupt vector
010D  B0 40                       MOV    AL,40H
010F  CD 21                       INT    21H
0111  89 1E 0102 R                MOV    WORD PTR OLD,BX
0115  8C 06 0104 R                MOV    WORD PTR OLD+2,ES
                            ;
                            ;install new interrupt vector 40H
                            ;
0119  BA 0106 R                   MOV    DX,OFFSET NEW40
011C  B4 25                       MOV    AH,25H
011E  B0 40                       MOV    AL,40H
0120  CD 21                       INT    21H
                            ;
                            ;leave NEW40 in memory
                            ;
0122  BA 0107 R                   MOV    DX,OFFSET START
0125  D1 EA                       SHR    DX,1
0127  D1 EA                       SHR    DX,1
0129  D1 EA                       SHR    DX,1
012B  D1 EA                       SHR    DX,1
012D  42                          INC    DX
012E  B8 3100                     MOV    AX,3100H
0131  CD 21                       INT    21H
                                  END
```



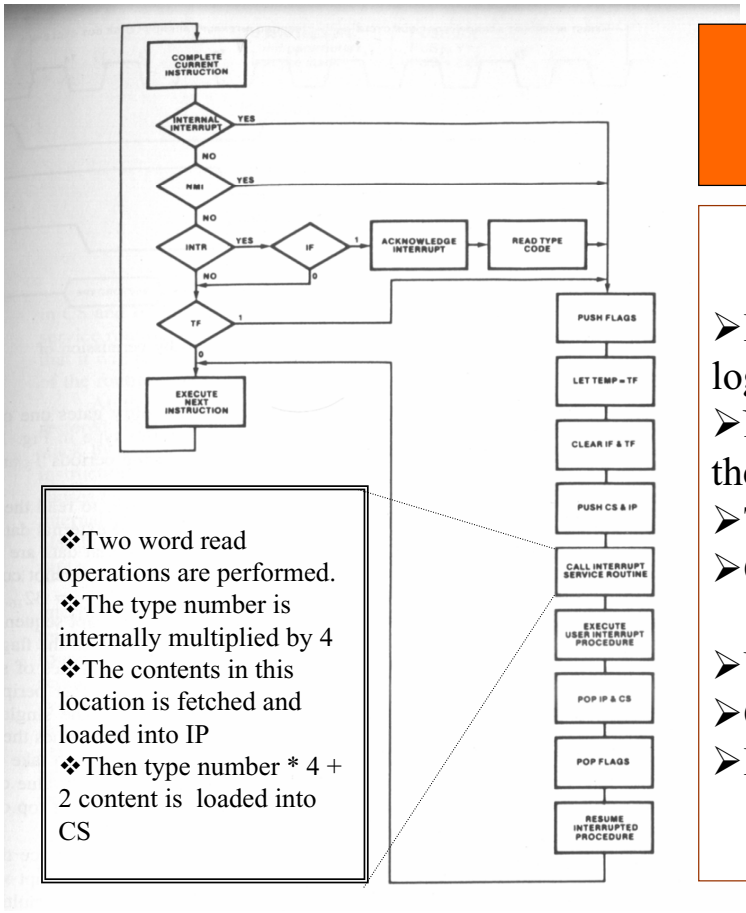
Interrupt Sequence

- The interrupt sequence begins when external device requests service by activating one of the interrupt inputs.
- The external device evaluates the priority of this interrupt
- INTR → 1
- 80x86 checks for the INTR at the last T state of the instruction
- Check for IF before granting INTA



Interrupt Sequence

- 80x86 initiates the INTA bus cycle. During T1 of the first bus cycle ALE is sent and bus is at Z state and stays high for the bus cycle.
- LOCK is provided in maxmode operation
- During the second interrupt acknowledge bus cycle, external circuitry gates one of the interrupts 20→FF onto data bus lines
- Must be valid during T3 and T4 of second bus cycle



❖Two word read operations are performed.
❖The type number is internally multiplied by 4
❖The contents in this location is fetched and loaded into IP
❖Then type number * 4 + 2 content is loaded into CS

Interrupt Sequence

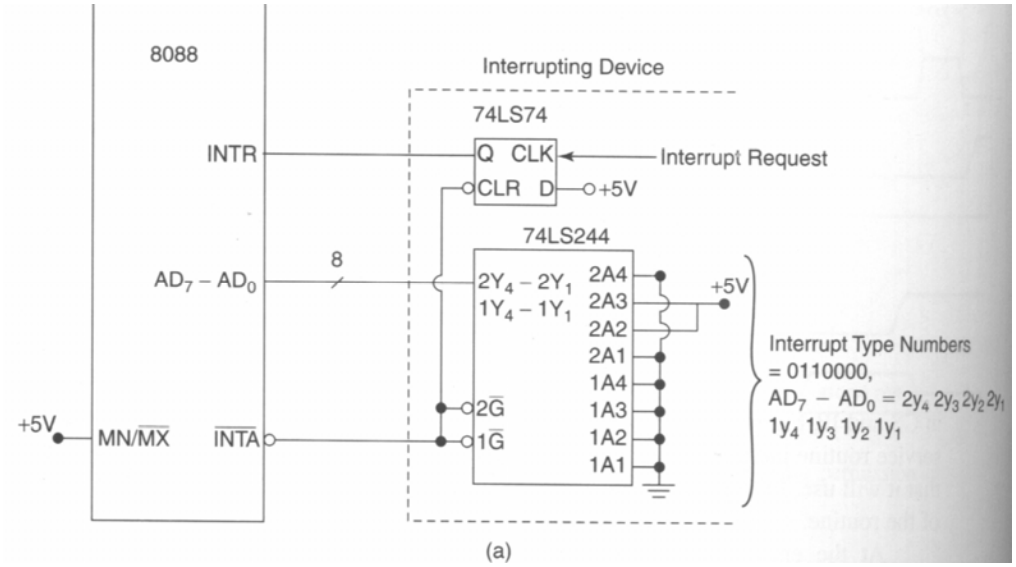
➢DT/R and DEN are at logic zero and IO/M is at 1.
➢Next save the contents of the flag register
➢TF and IF are cleared
➢CS and IP are pushed

➢Upon return by IRET
➢CS and IP are popped
➢Flags are popped

Interrupt Example

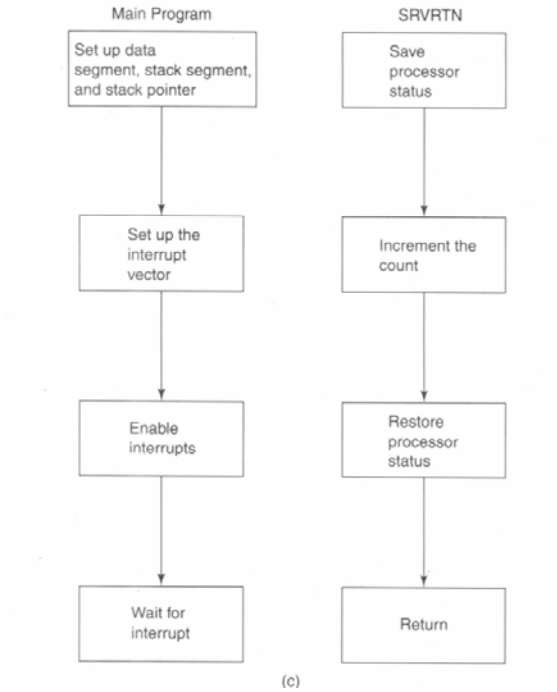
- An interrupting device interrupts the microprocessor each time the interrupt request input has a transition from 0 to 1.
- 74LS244 creates the interrupt type number 60H as a response to INTA
- Assume:
  - CS=DS=1000H
  - SS=4000H
  - Main program offset is 200H
  - Count (counts the number of interrupts) offset is 100H
  - Interrupt-service routine code segment is 2000H
  - Interrupt-service routine code offset is 1000H
  - Stack has an offset of 500H to the current stack segment
  - Make a map of the memory space organisation
  - Write a main program and a service routine to count the number of positive interrupt transitions.

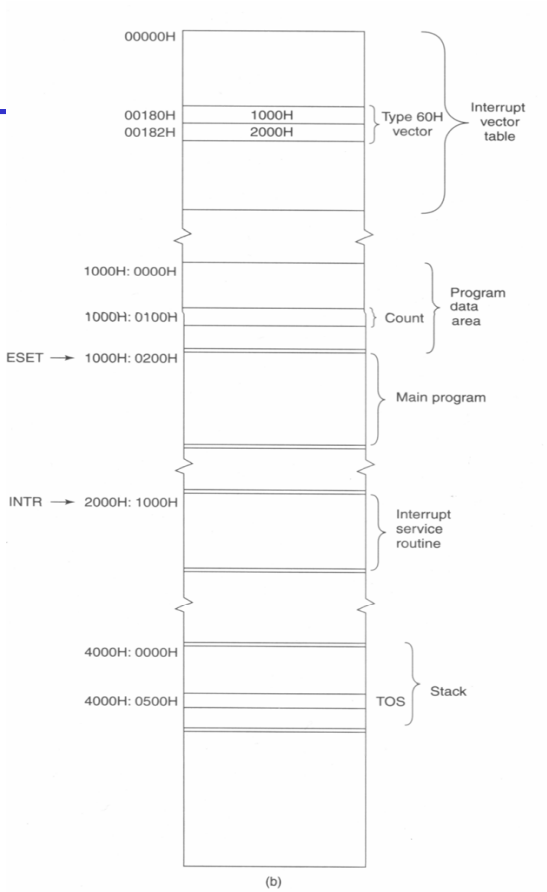
Interrupt Example



Interrupts the microprocessor each time the interrupt request signal has a transition from 0→ 1. The corresponding interrupt number generated by the hardware in response to INTA is 60H

Memory organization



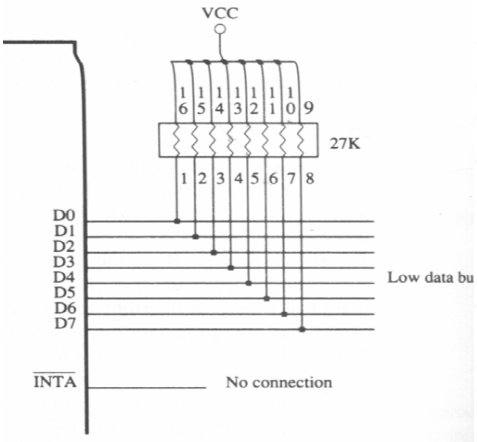


Program

```
;Main Program, START = 1000H:0200H

START:      MOV AX,1000H          ;Setup data segment at 1000H:0000H
            MOV DS,AX
            MOV AX,4000H          ;Setup stack segment at 4000H:0000H
            MOV SS,AX
            MOV SP,0500H          ;TOS is at 4000H;0500H
            MOV AX,0000H          ;Segment for interrupt vector table
            MOV ES,AX
            MOV AX,0000H          ;Service routine offset
            MOV [ES:180H],AX
            MOV AX,2000H          ;Service routine segment
            MOV [ES:182H],AX
            STI                   ;Enable interrupts
HERE:       JMP HERE              ;Wait for interrupt

;Interrupt Service Routine, SRVRTN = 2000H:1000H

SRVRTN:     PUSH AX               ;Save register to be used
            MOV AL,[0100H]        ;Get the count
            INC AL                ;Increment the count
            DAA                   ;Decimal asdjust the count
            MOV [0100H],AL        ;Save the updated count
            POP AX                ;Restore the register used
            IRET                  ;Return from the interrupt
```

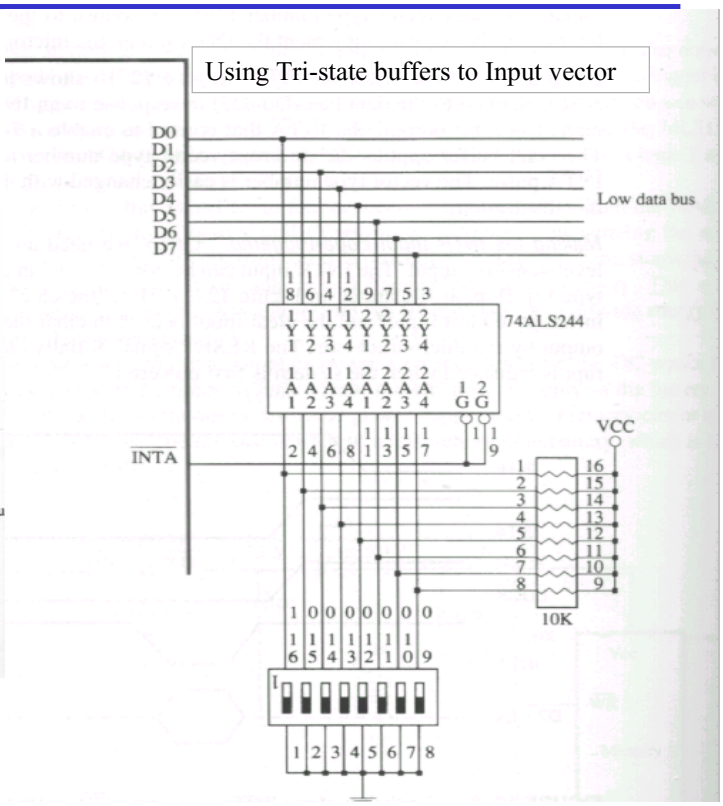
(d)

Using hardware interrupt

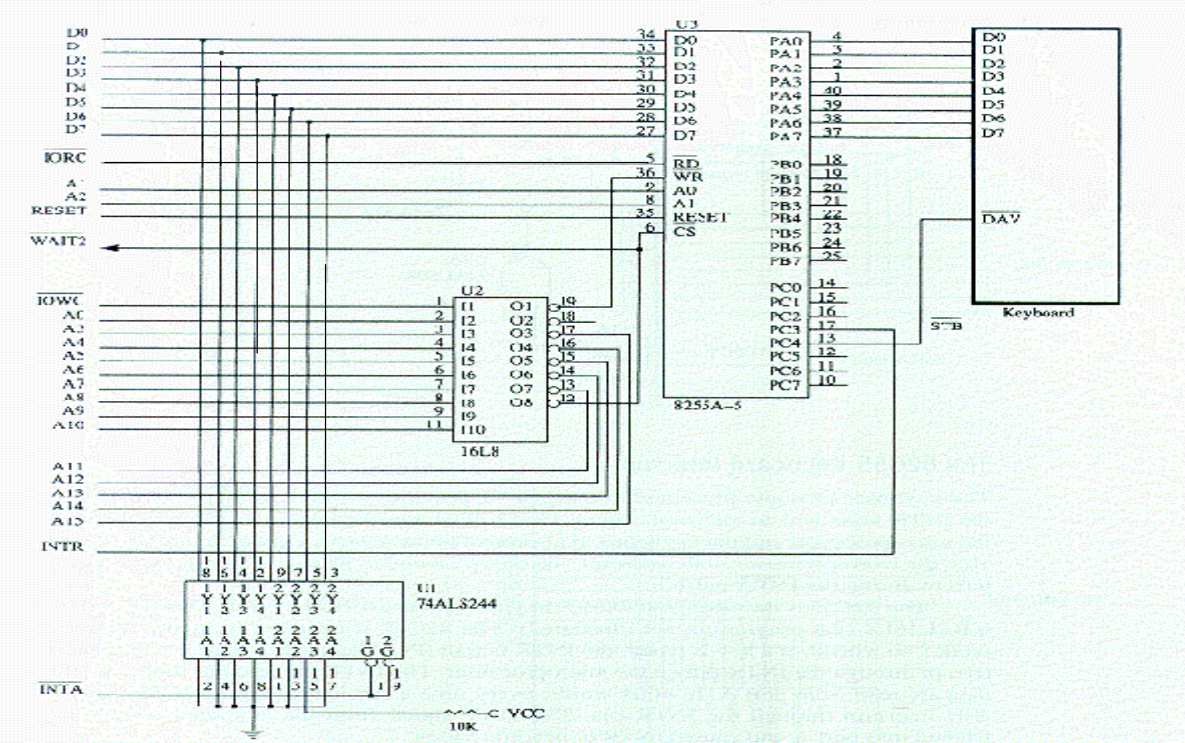


Cheapest Way (FF applied)

Using Tri-state buffers to Input vector



Interrupt circuits

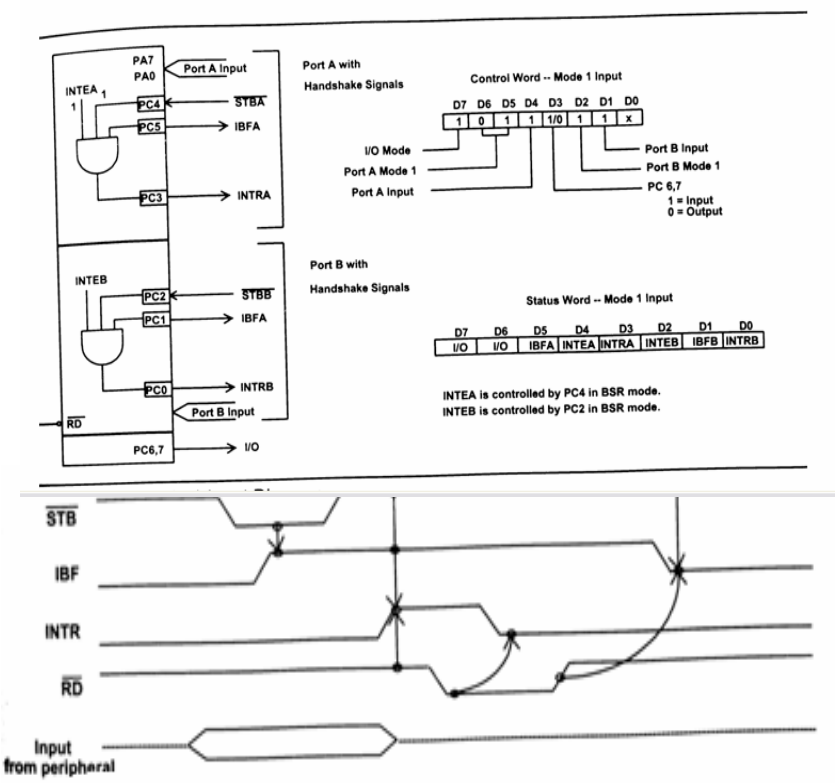


GURE 12–12 An 82C55 interfaced to a keyboard from the microprocessor system using interrupt vector 40H.

Description

- 8255 is decoded at 0500h, 0502h, 0504h, and 0506h
- 8255 is operated at Mode 1 (strobed input) B0 CONTROL WORD
- Whenever a key is typed , the INTR output (PC3) becomes a logic 1 and requests an interrupt thru the INTR pin on the microprocessor
- The INTR remains high until the ASCII data are read form port A.
- In other words, every time a key is typed the 8255 requests a type 40h interrupt thru the INTR pin
- The DAV signal from the keyboard causes data to be latched into port A and causes INTR to become a logic 1
- Data are input from the keyboard and then stored in the FIFO (first in first out) buffer
- FIFO in our example is 256 bytes
- The procedure first checks to see whether the FIFO is full.
- A full condition is indicated when the input pointer (INP) is one byte below the output pointer (OUTP)

Remembering Mode 1 with Interrupts this time



Example: "Read from the Keyboard routine" into FIFO

```
                ; interrupt service routine to read a key from the keyboard
                PORTA   EQU   500h
                CNTR    EQU   506h
                FIFO    DB    256  DUP (?)
                INP     DW    ?      ; SET AS OFFSET FIFO IN MAIN PROG
                OUTP    DW    ?      ; SET AS OFFSET FIFO IN MAIN PROG
KEY:            PROC FAR             ;USES AX BX DI DX
                MOV     BX, INP
                MOV     DI, OUTP
                INC     BL
                CMP     BX, DI       ;test for queue full
                JE      FULL         ; if queue is full
                DEC     BL
                MOV     DX, PORTA
                IN      AL,DX        ; read the key
                MOV     [BX], AL
                INC     WORD PTR INP
                JMP     DONE
FULL:           MOV     AL,8 ;DISABLE THE INTERRUPT
                MOV     DX, CNTR
                OUT     DX,AL
DONE:           IRET
                KEY     ENDP
```

Example contd: “Read from the FIFO into AH”

```
READ:   PROC    FAR USES  BX DI DX
EMPTY:  MOV     BX, INP
        MOV     DI, OUTP
        CMP     BX,DI
        JE      EMPTY
        MOV     AH, CS:DI
        MOV     AL,9 ; enable 8255 intEa
        MOV     DX, CNTR
        OUT     DX,AL
        INC     BYTE PTR CS:OUTP
        RET
READ :  ENDP
```


Multiple Interrupts - Another Interrupt Structure

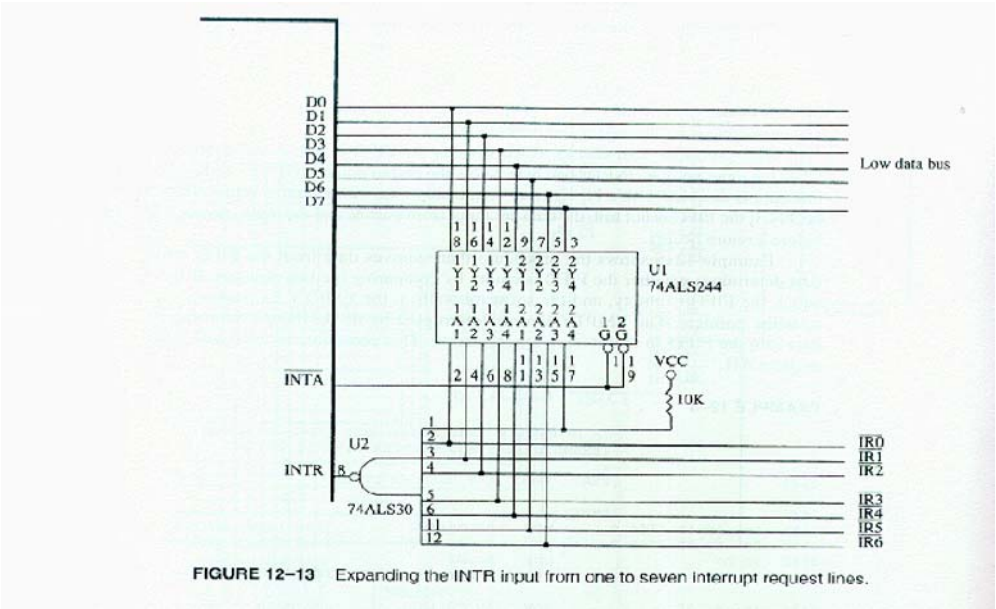


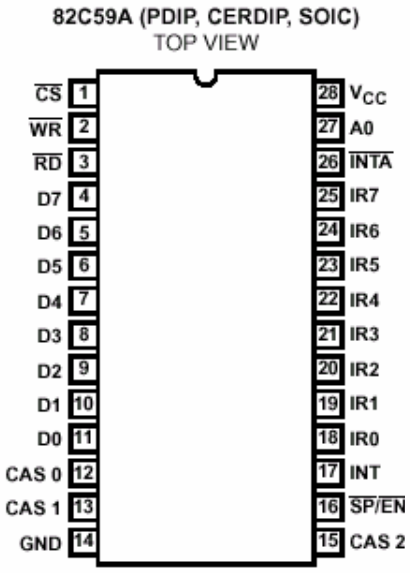
FIGURE 12–13 Expanding the INTR input from one to seven interrupt request lines.

8255 Programmable Interrupt Controller

8259 Programmable Interrupt Controller

•The 8259 programmable interrupt controller (PIC) adds eight vectored priority encoded interrupts to the microprocessor.
•This controller can be expanded to accept up to 64 interrupt requests. This requires a master 8259 and eight 8259 slaves.
•Vector an Interrupt request anywhere in the memory map.
•Resolve eight levels of interrupt priorities in a variety of modes, such as fully nested mode, automatic rotation mode, and specific rotation mode.
•Mask each of the interrupt request individually
•Read the status of the pending interrupts, in-service interrupts and masked interrupts.

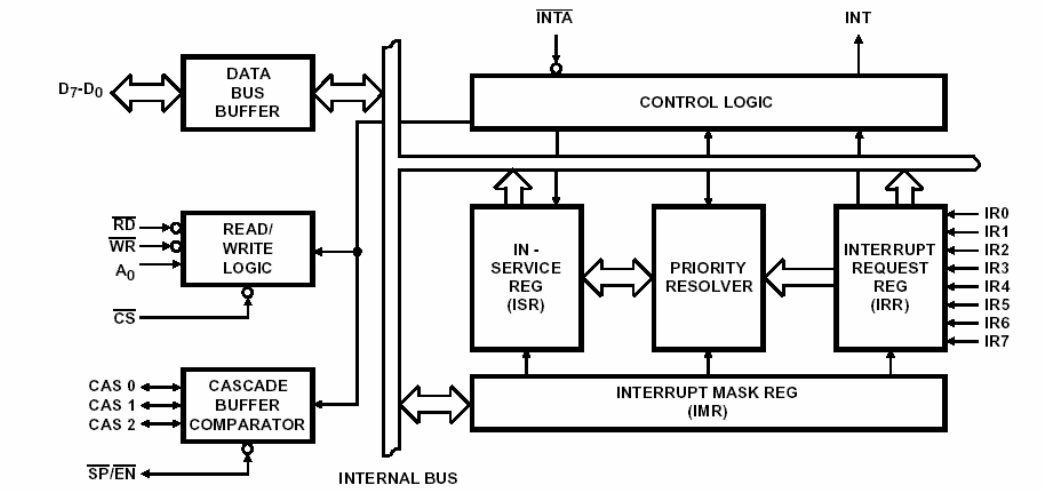
Block Diagram



PIN	DESCRIPTION
D7 - D0	Data Bus (Bidirectional)
$\overline{RD}$	Read Input
$\overline{WR}$	Write Input
A0	Command Select Address
$\overline{CS}$	Chip Select
CAS 2 - CAS 0	Cascade Lines
$\overline{SP/EN}$	Slave Program Input Enable
INT	Interrupt Output
$\overline{INTA}$	Interrupt Acknowledge Input
IR0 - IR7	Interrupt Request Inputs

82C59A Programmable Interrupt Controller

- Block diagram of 82C59A includes 8 blocks
  - 8259 is treated by the host processor as a peripheral device.
  - 8259 is configured by the host pocessor to select functions.
- **Data bus buffer and read-write logic**: are used to configure the internal registers of the chip.
  - A0 address selects **different** command words within the 8259



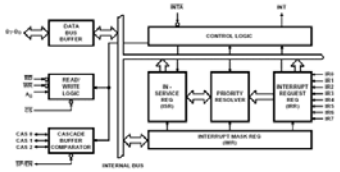
82C59A Programmable Interrupt Controller

- Control Logic INT and $INTA^-$ ared used as the handshaking interface.
  - INT output connects to the INTR pin of the master and is connected to a master IR pin on a slave. $INTA^-$ is sent as a reply.
  - In a system with master and slaves, **only the master $INTA^-$ signal** is connected.
- Interrupt Registers and Priority Resolver: Interrupt inputs IR_0 to IR_7 can be configured as either *level-sensitive* or *edge-triggered* inputs. Edge-triggered inputs become active on 0 to 1 transitions.
  1. **Interrupt request register (IRR):** is used to indicate all interrupt levels requesting service.
  2. **In service register (ISR):** is used to store all interrupt levels which are currently being serviced.
  3. **Interrupt mask register (IMR):** is used to enable or mask out the individual interrupt inputs through bits M0 to M7. 0= enable, 1= masked **out.**
  4. **Priority resolver**: This block determines the priorities of the bits set in the IRR. The highest priority is selected and strobed into the corresponding bit of the ISR during the $INTA^-$ sequence.
     - The priority resolver examines these 3 registers and determines whether INT should be sent to the MPU

82C59A Programmable Interrupt Controller

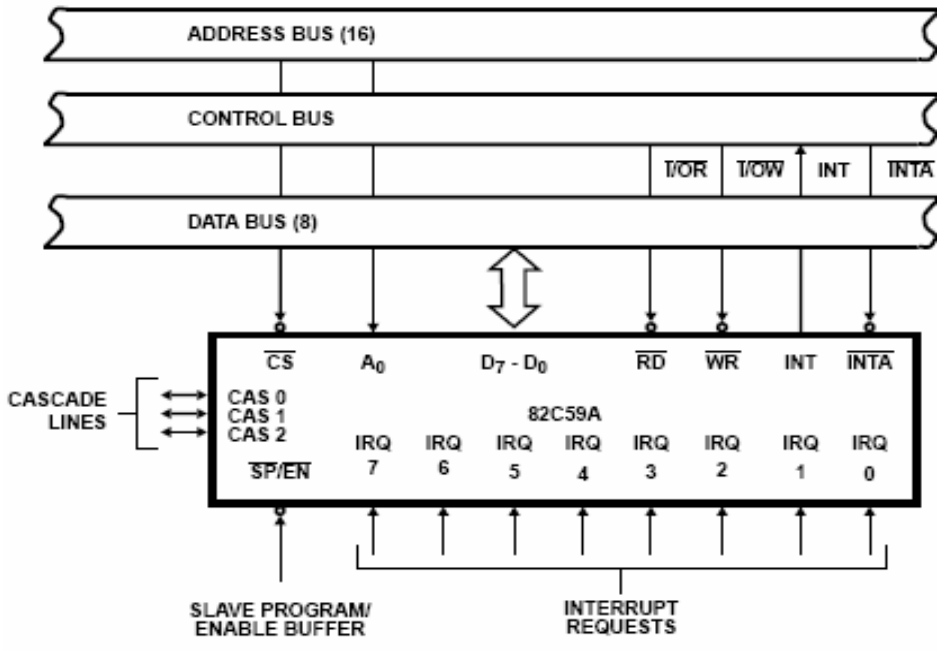
- **Cascade-buffer comparator:** Sends the address of the selected chip to the slaves in the master mode and decodes the status indicated by the master to find own address to respond.
  - Cascade interface CAS_0-CAS_2 and SP^-/EN^- :
    - Cascade interface CAS_0-CAS_2 carry the address of the slave to be serviced.
    - SP^-/EN^- :=1 selects the chip as the master in cascade mode
      :=0 selects the chip as the slave in cascade mode
      :in single mode it becomes the enable output for the data transiver

Interrupt Sequence



1) One or more of the INTERRUPT REQUEST lines (IR0 - IR7) are raised high, setting the corresponding IRR bit(s).
2) The 82C59A evaluates those requests in the priority resolver with the **IMR** and **ISR**, resolves the priority and sends an interrupt (INT) to the CPU, if appropriate.
3) The CPU acknowledges the INT and responds with first INTA pulse.
4) During this INTA pulse, the appropriate ISR bit is set and the corresponding bit in the IRR is reset (to remove request). The 82C59A does not drive the data bus during the first INTA pulse.
5) The 80C86/88/286 CPU will initiate a second INTA pulse. The 82C59A outputs the 8-bit pointer onto the data bus to be read by the CPU.
6) This completes the interrupt cycle. In the **Automatic End of Interrupt** (AEOI) mode, the ISR bit is reset at the end of the second INTA pulse. Otherwise, the ISR bit remains set until an appropriate End of Interrupt (EOI) command is issued at the end of the interrupt subroutine.

8259 System Bus



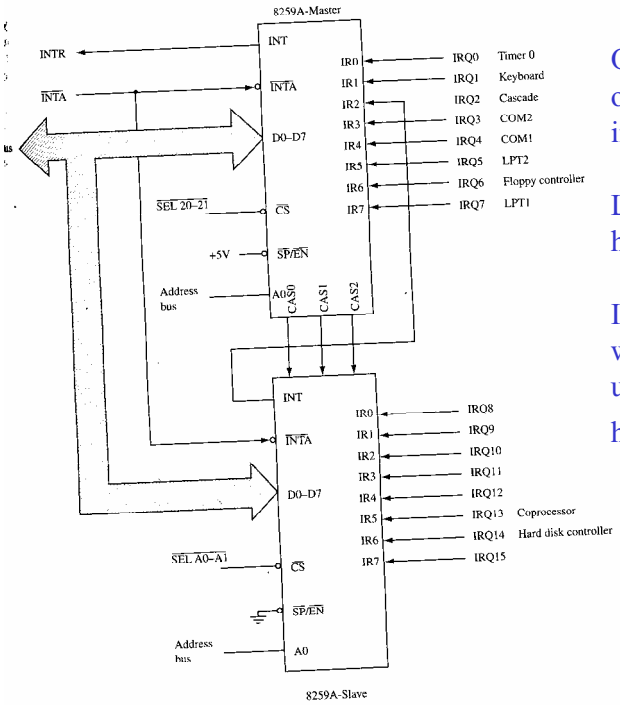
82C59A STANDARD SYSTEM BUS INTERFACE

Content of the Interrupt Vector Byte

CONTENT OF INTERRUPT VECTOR BYTE FOR 80C86/88/286 SYSTEM MODE

	D7	D6	D5	D4	D3	D2	D1	D0
IR7	T7	T6	T5	T4	T3	1	1	1
IR6	T7	T6	T5	T4	T3	1	1	0
IR5	T7	T6	T5	T4	T3	1	0	1
IR4	T7	T6	T5	T4	T3	1	0	0
IR3	T7	T6	T5	T4	T3	0	1	1
IR2	T7	T6	T5	T4	T3	0	1	0
IR1	T7	T6	T5	T4	T3	0	0	1
IR0	T7	T6	T5	T4	T3	0	0	0

Two controllers wired in cascade



On the PC, the controller is operated
in the fully nested mode

Lowest numbered IRQ input has highest priority

Interrupts of a lower priority will not be acknowledged until the higher priority interrupts have been serviced

Fully Nested Mode

- It prioritizes the IR inputs such that IR0 has highest priority and IR7 has lowest priority
- This priority structure extends to interrupts currently in service as well as simultaneous interrupt requests
- For example, if an interrupt on IR3 is being serviced (IS3 = 1) and a request occurs on IR2, the controller will issue an interrupt request because IR2 has higher priority.
- But if an IR4 is received (or any interrupt higher than IR2), the controller will not issue the request
- Note however that the IR2 request will not be acknowledged unless the processor has set IF within the IR3 service routine
- In all operating modes, the IS bit corresponding to the active routine must be reset to allow other lower priority interrupts to be acknowledged
- This can be done by outputting **manually** a special nonspecific EOI instruction to the controller just before IRET
- Alternatively, the controller can be programmed to perform this nonspecific EOI **automatically** when the second INTA pulse occurs

Interrupt Process Fully Nested Mode

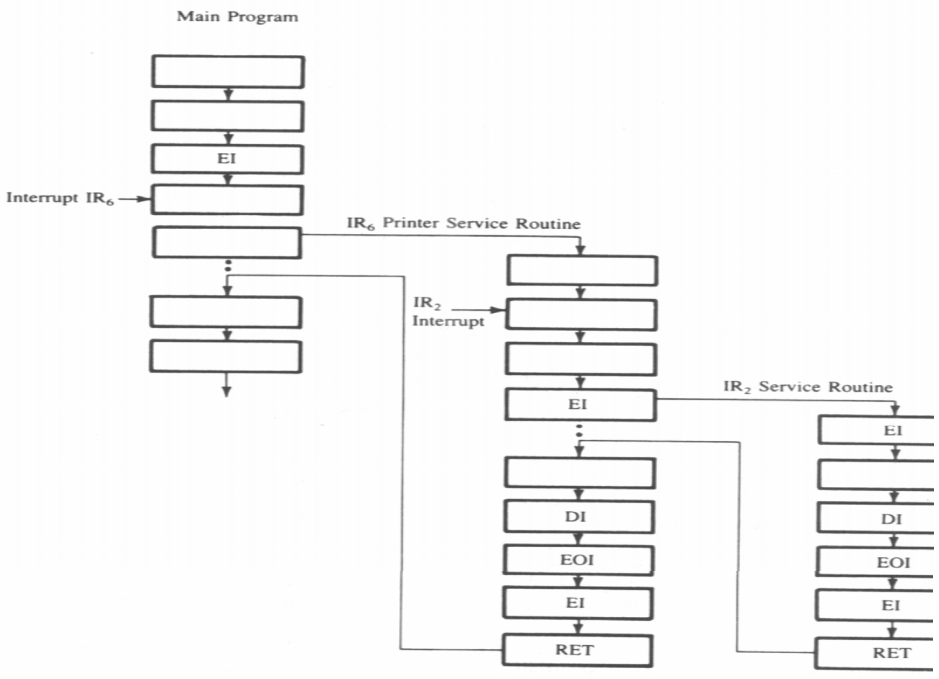


FIGURE 15.32
Interrupt Process: Fully Nested Mode