Microprocessor System Design
Basic I/O

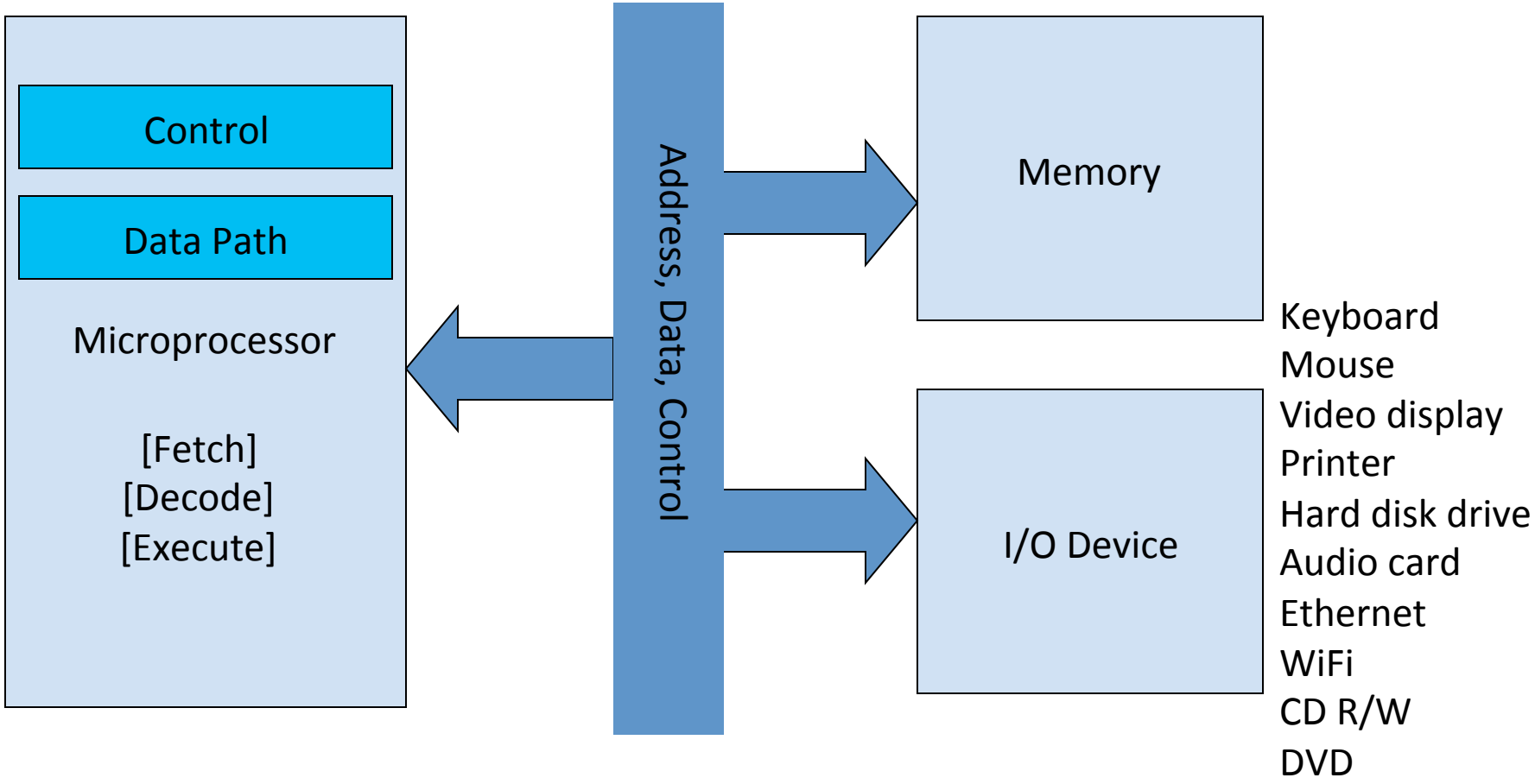
ECE 485/585

Mark G. Faust

Outline

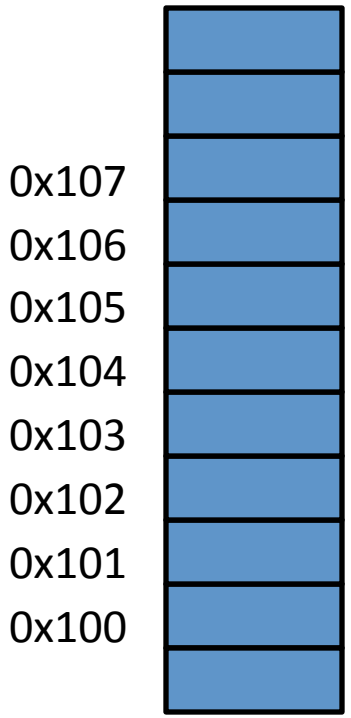
- Simple model of Computation
- Memory Addressing
  - Alignment, Byte Order
- 8088/8086 Bus
- Asynchronous I/O Signaling
- Review of Basic I/O
  - How is I/O performed
    - Dedicated/Isolated /Direct I/O Ports
    - Memory Mapped I/O
  - How do we tell when I/O device is ready or command complete?
    - Polling
    - Interrupts
  - How do we transfer data?
    - Programmed I/O
    - DMA

Simplified Model

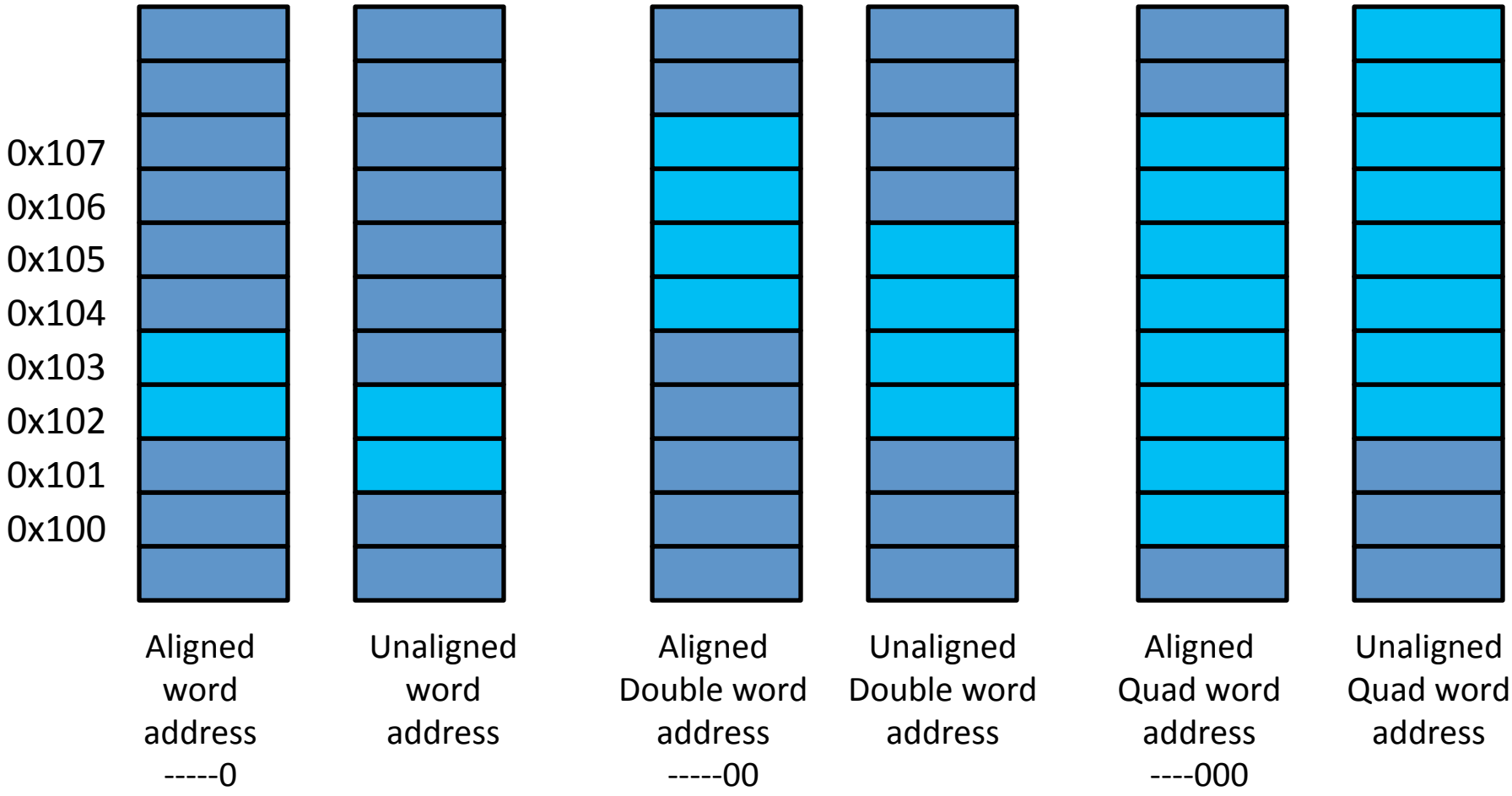


Memory Addressing

- Size of operands
  - Bytes, words, long/double words, quadwords
    - 16-bit half word (Intel: word)
    - 32-bit word (Intel: doubleword, dword)
    - 64-bit double word (Intel: quadword, qword)
  - Note: names are non-standard
    - SUN Sparc word is 32-bits, double is 64-bits
- Alignment
  - Can multi-byte operands begin at any (byte) address?
    - Yes: non-aligned
    - No: aligned. Low order address bit(s) will be zero



Memory Operand Alignment

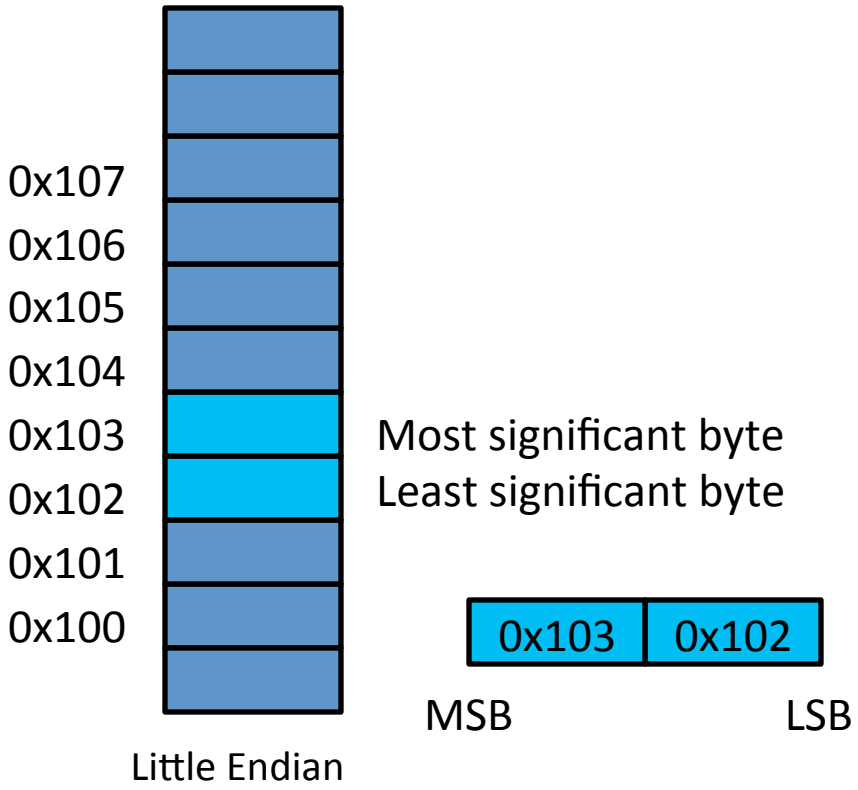


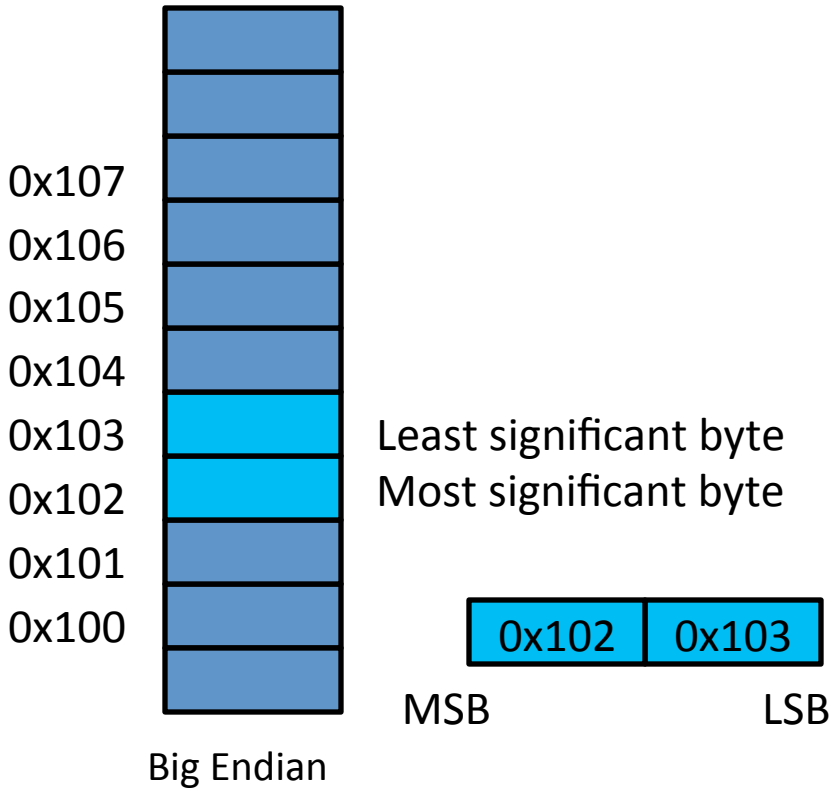
Memory Operand Alignment

- Why do we care?
- Unaligned memory references
  - Can cause multiple memory bus cycles for a single operand
  - May also span cache lines
    - Requiring multiple evictions, multiple cache line fills
  - Complicate memory system and cache controller design

- Some architectures restrict addresses to be aligned
- Even in architectures without alignment restrictions (e.g. Intel x86) assembler directives are typically used (automatically by compiler) to force alignment, improving efficiency of generated code.

Byte Order: Big Endian or Little Endian

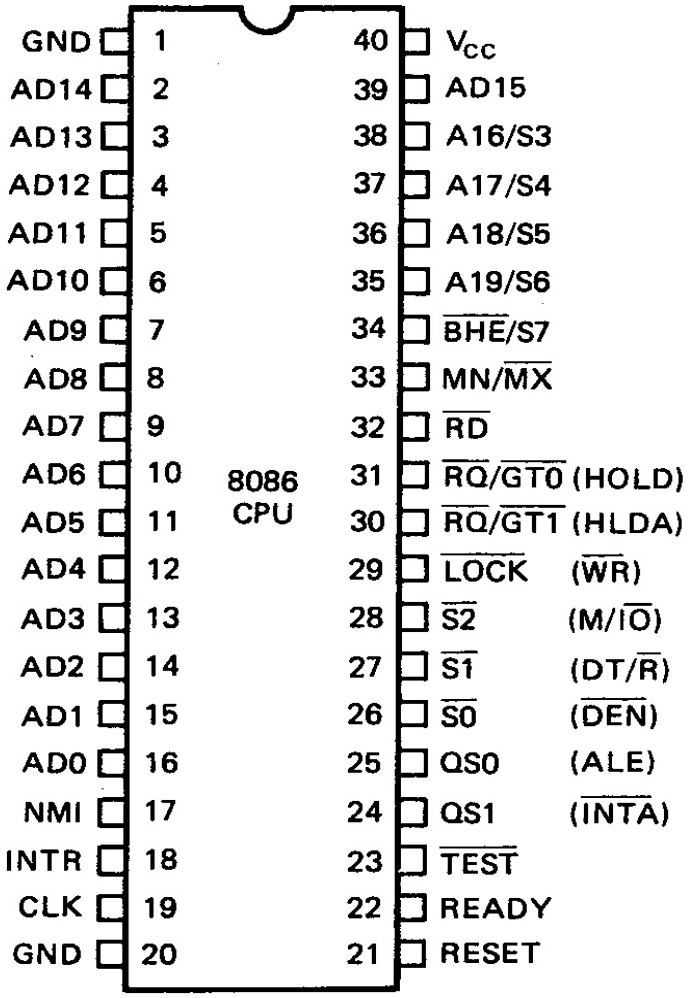




Byte Order

- Pros and cons often exaggerated
  - Big Endian: Motorola 680x0, Sun Sparc, PDP-11
  - Little Endian: VAX, Intel IA32
  - Configurable: MIPS, ARM
- Really only matters when
  - Communicating between two systems
  - Networks and serial interfaces

A Simple Example: Intel 8086/8088



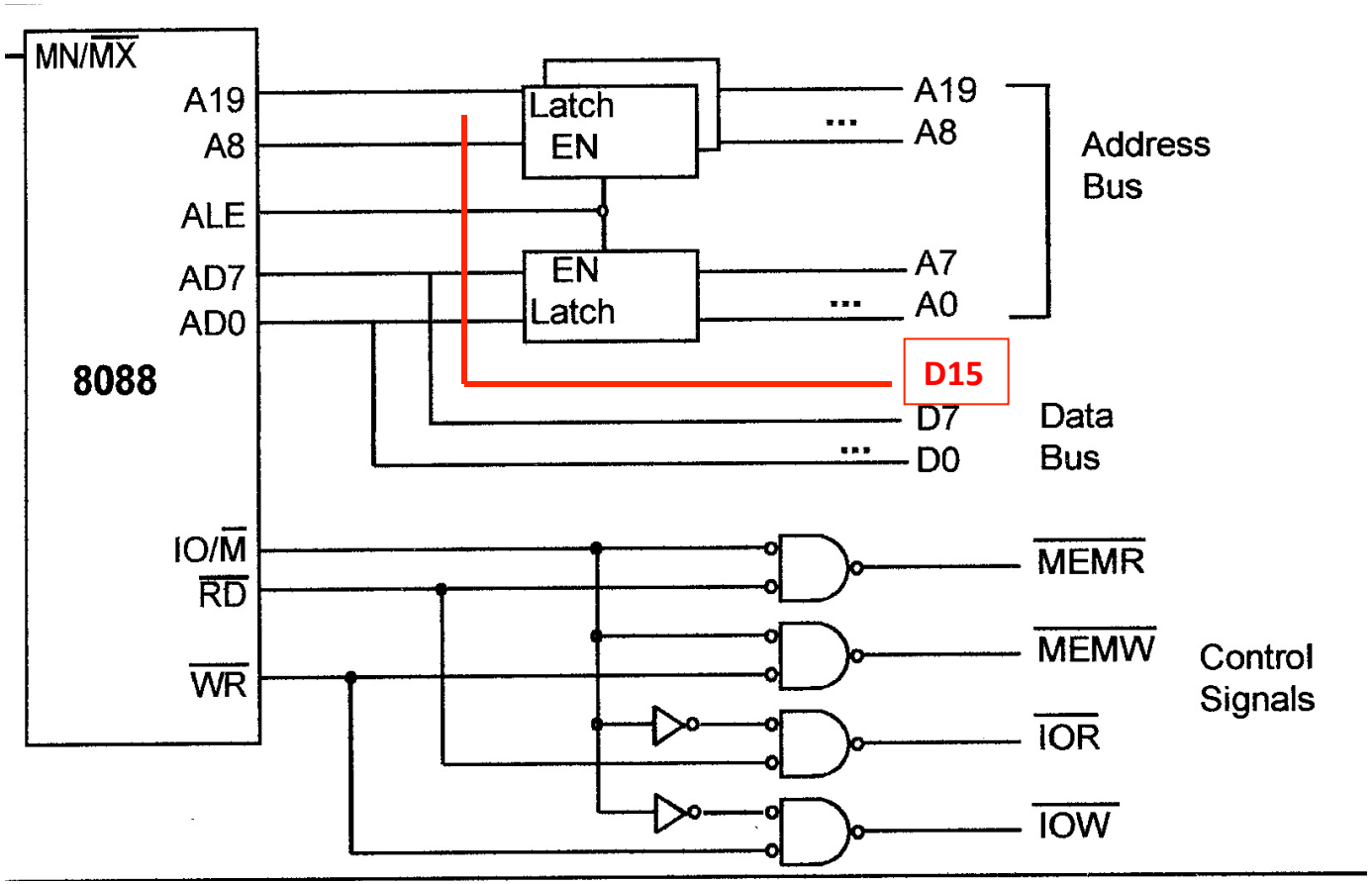
- 16-bit data bus
- 20-bit address (2^{20} = 1M)
- Data/Address Multiplexed
  - AD0..AD15 + A16..A19
- 8088 (first PC) had 8 bit data bus

Minimum/Maximum Mode
Support co-processor
Affects I/O signals

Package Pin Count Constraints

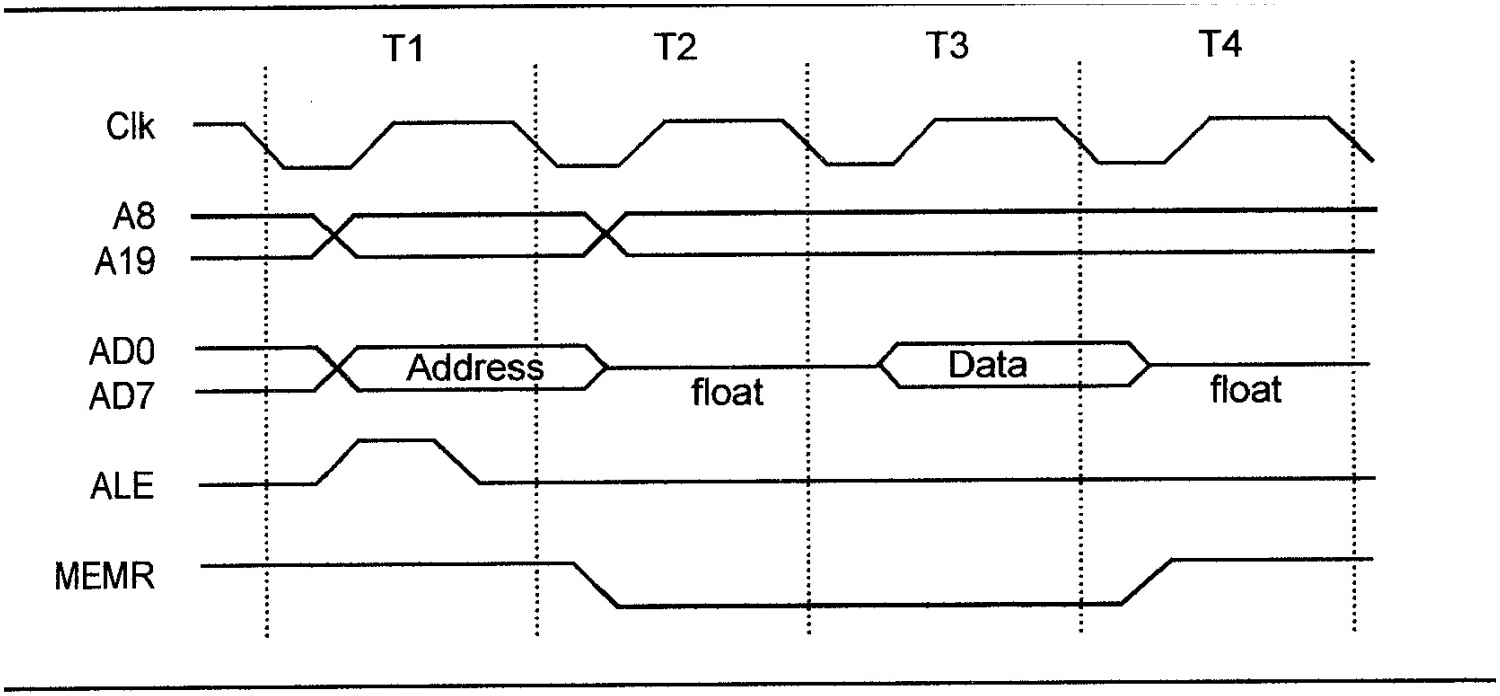
8088 Address, Data, Control Signals

How is 8086 different?

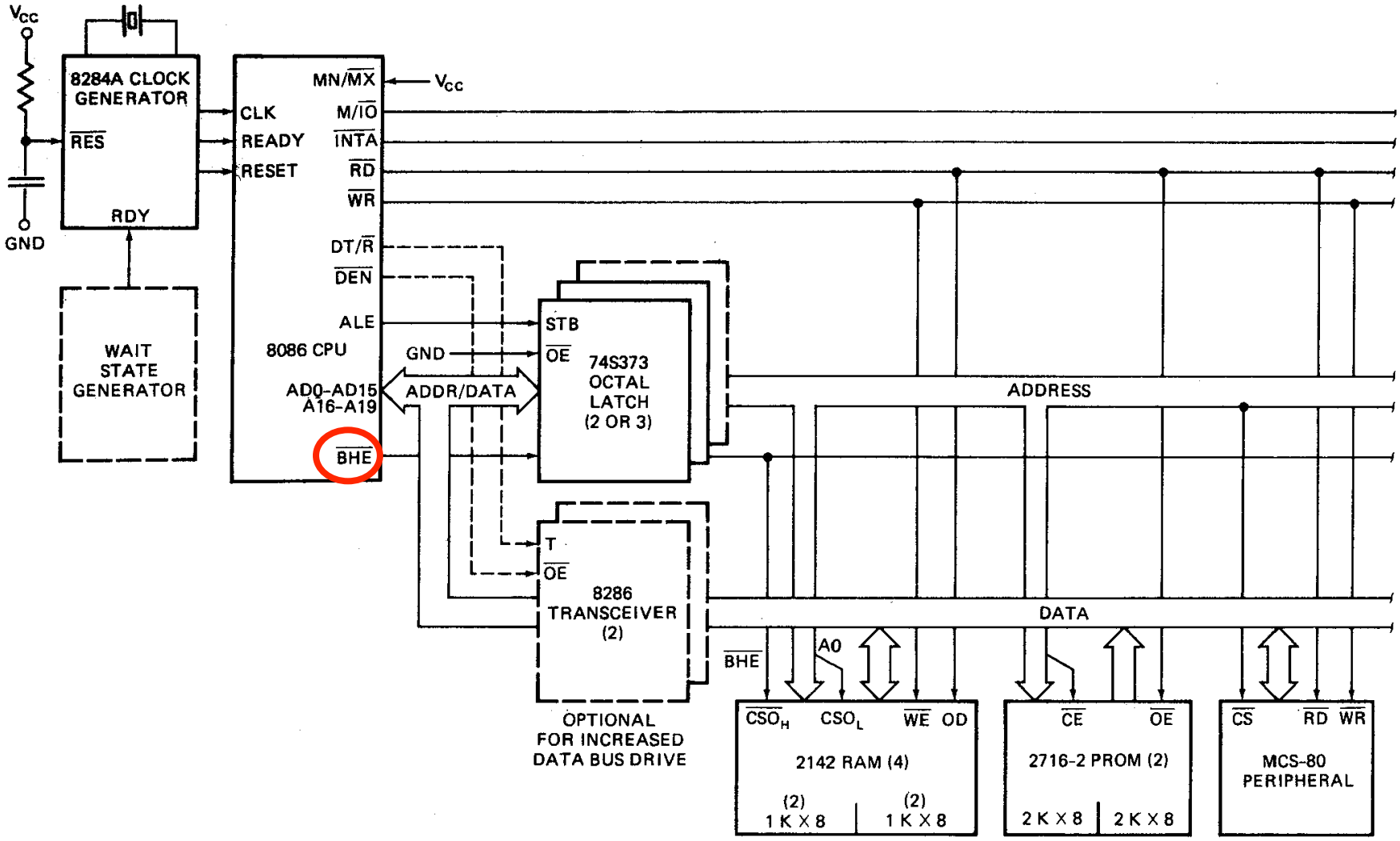


8088 Timing

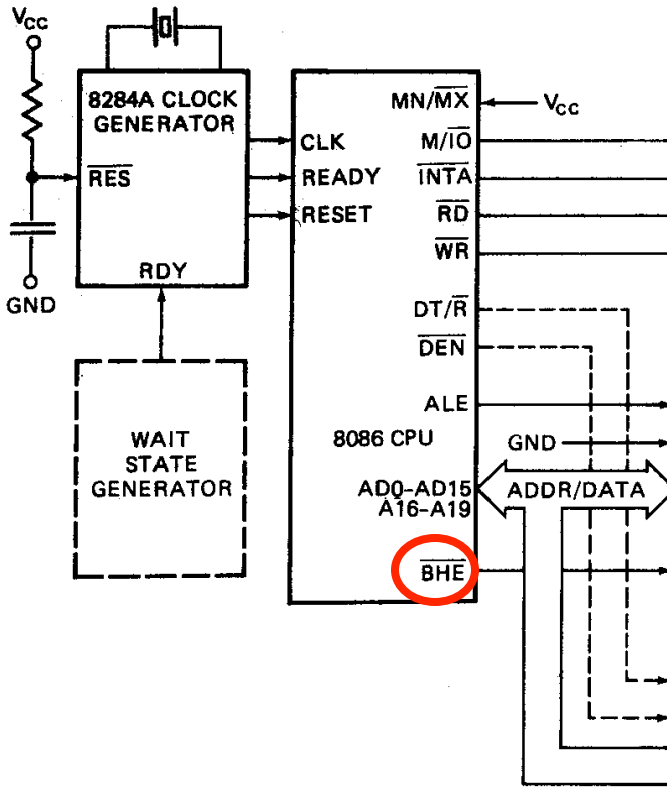
- Instruction Cycle
  - Fetch/Decode/Execute
  - Comprised of one or more Machine Cycles
    - Comprised of one or more states



Block Diagram of 8086-based System



Block Diagram of 8086-based System



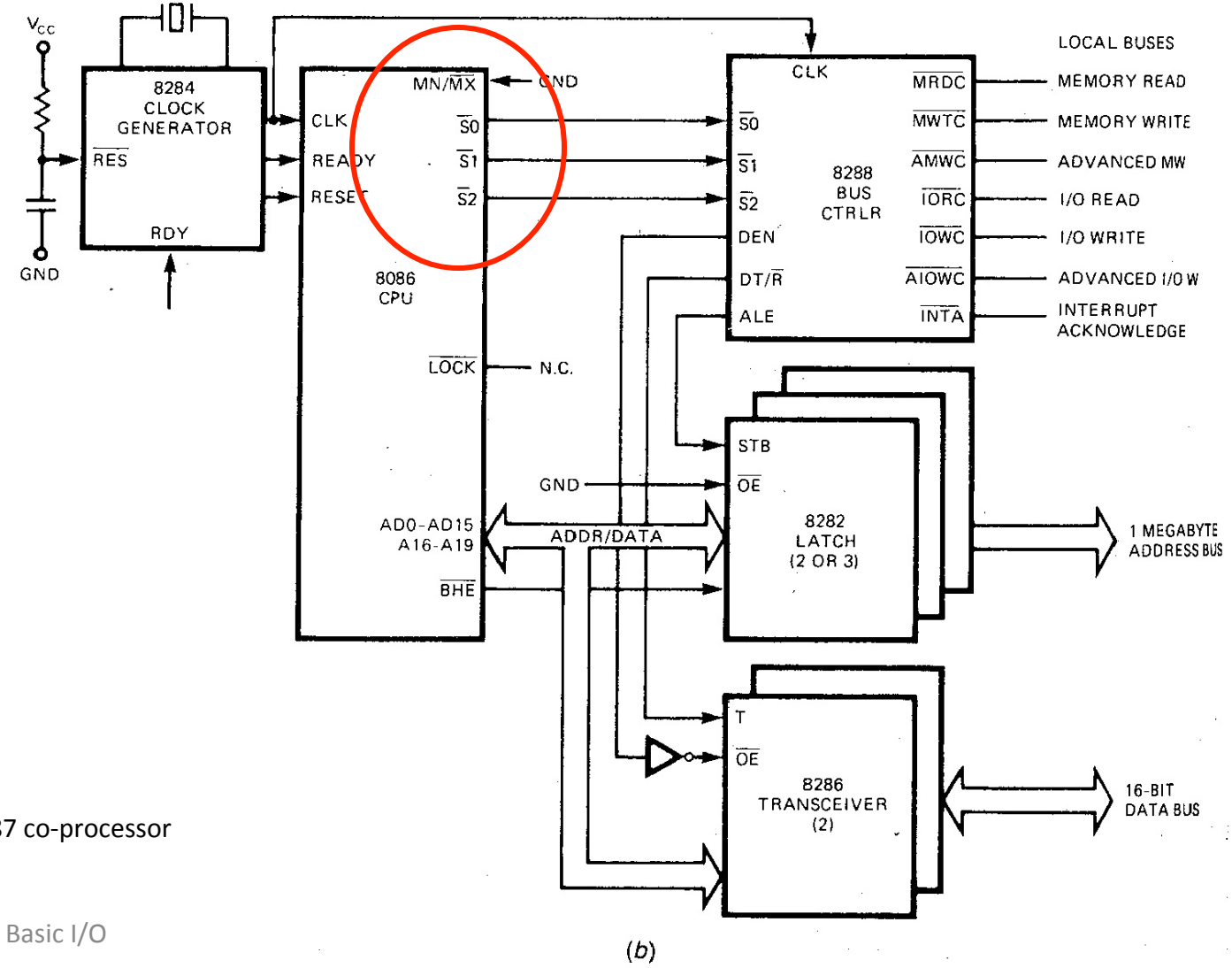
8086 can access bytes or words
References do not need to be aligned

Address alone is not enough to convey both location and size of data

BHE# (Byte High Enable)

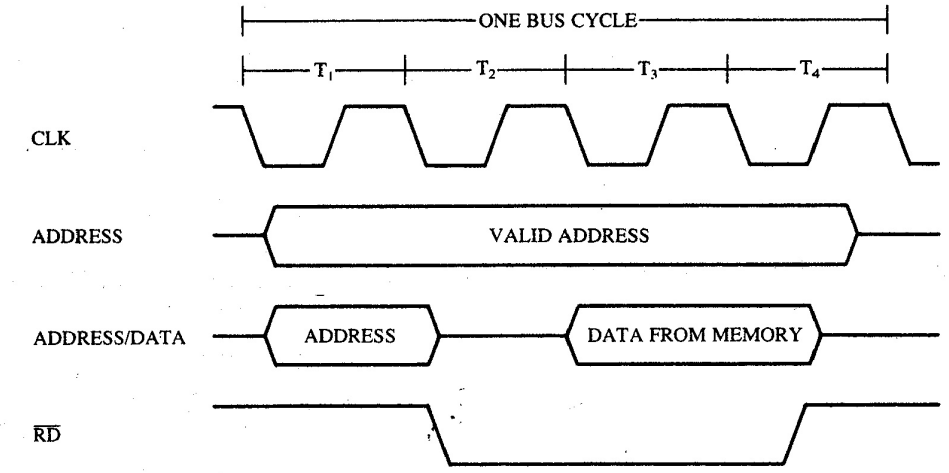
BHE#	A0	Refers to
0	0	16 bits (D0..D15)
0	1	8 bits (D8..D15)
1	0	8 bits (D0..D7)
1	1	<reserved>

8086 in Maximum Mode*

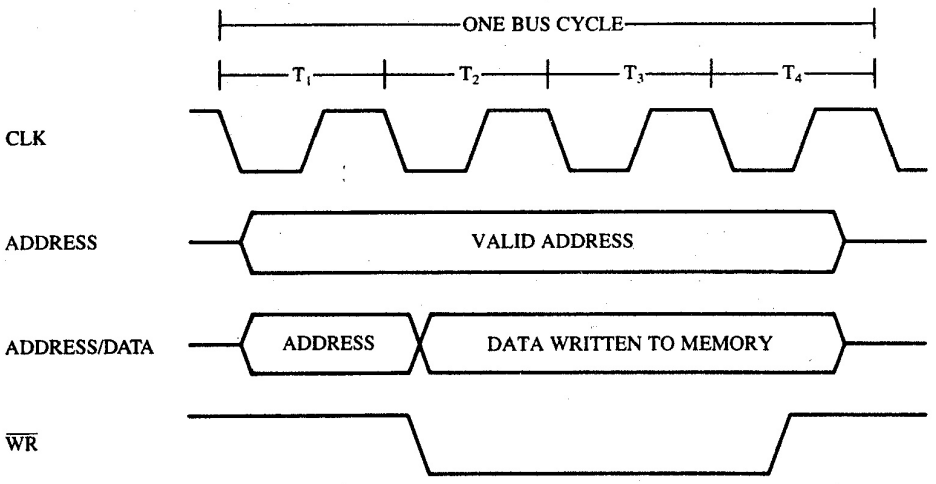


*With 8087 co-processor

8086 Bus Timing

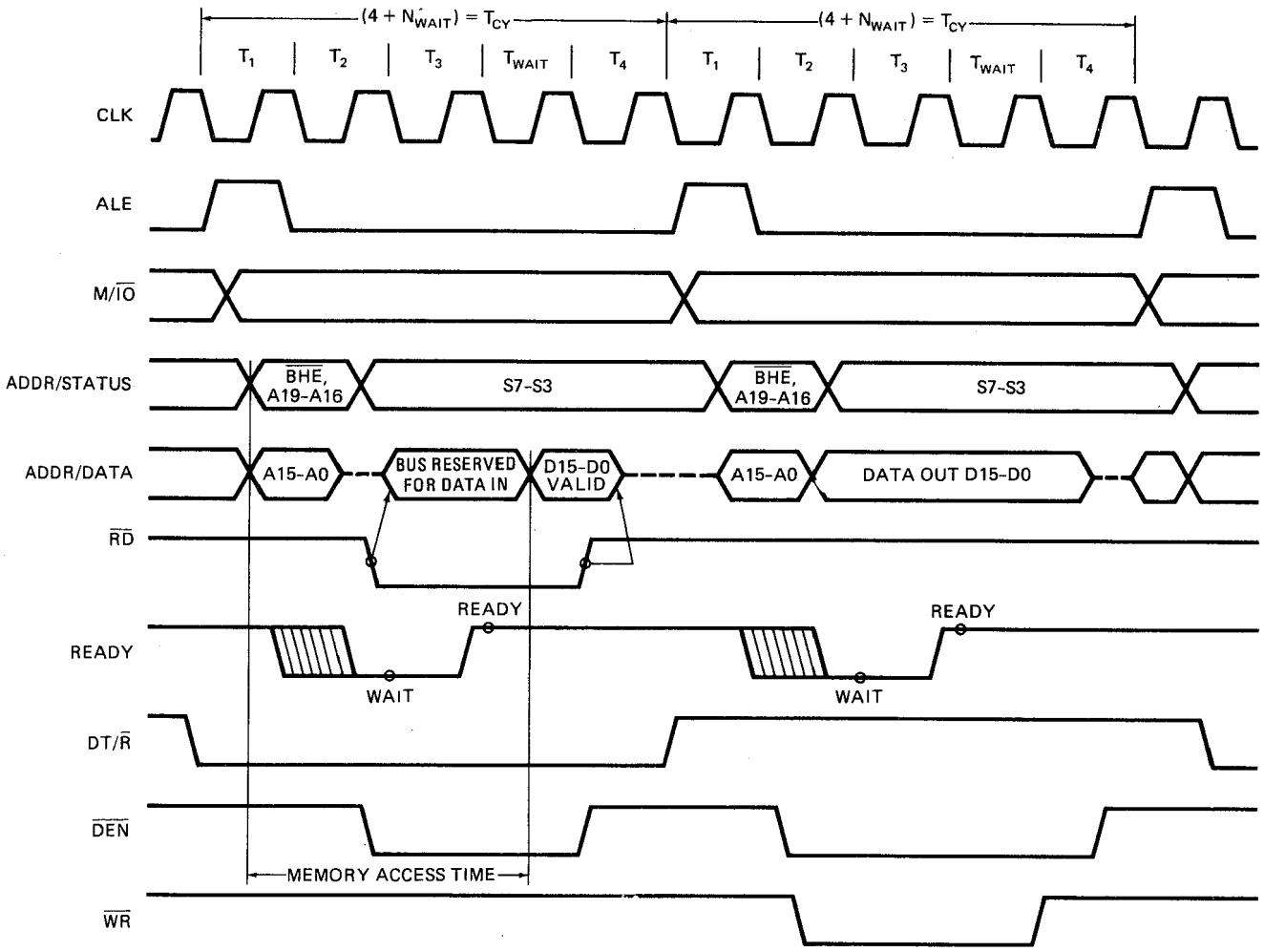


Read Cycle

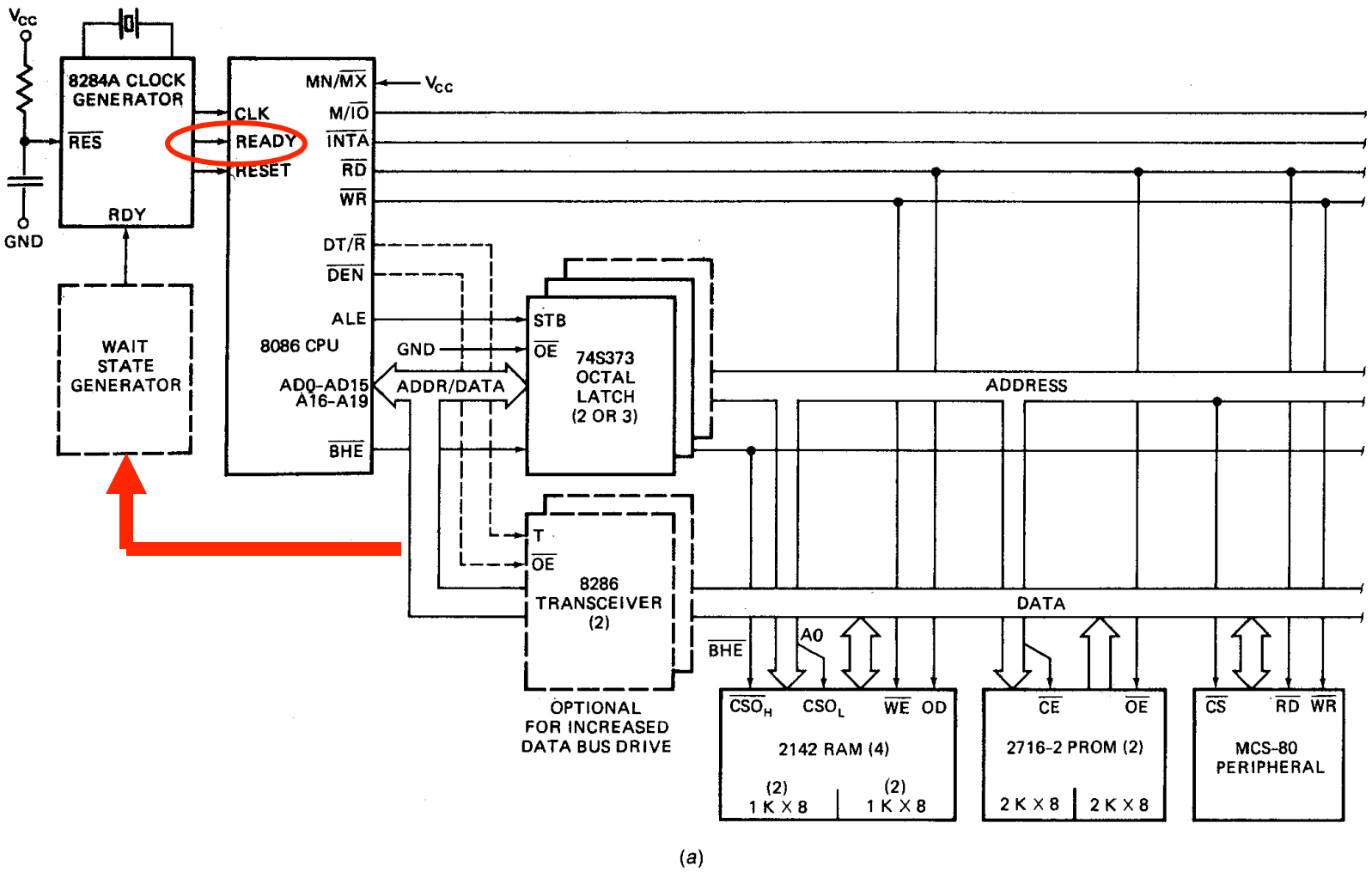


Write Cycle

8086 Bus Timing



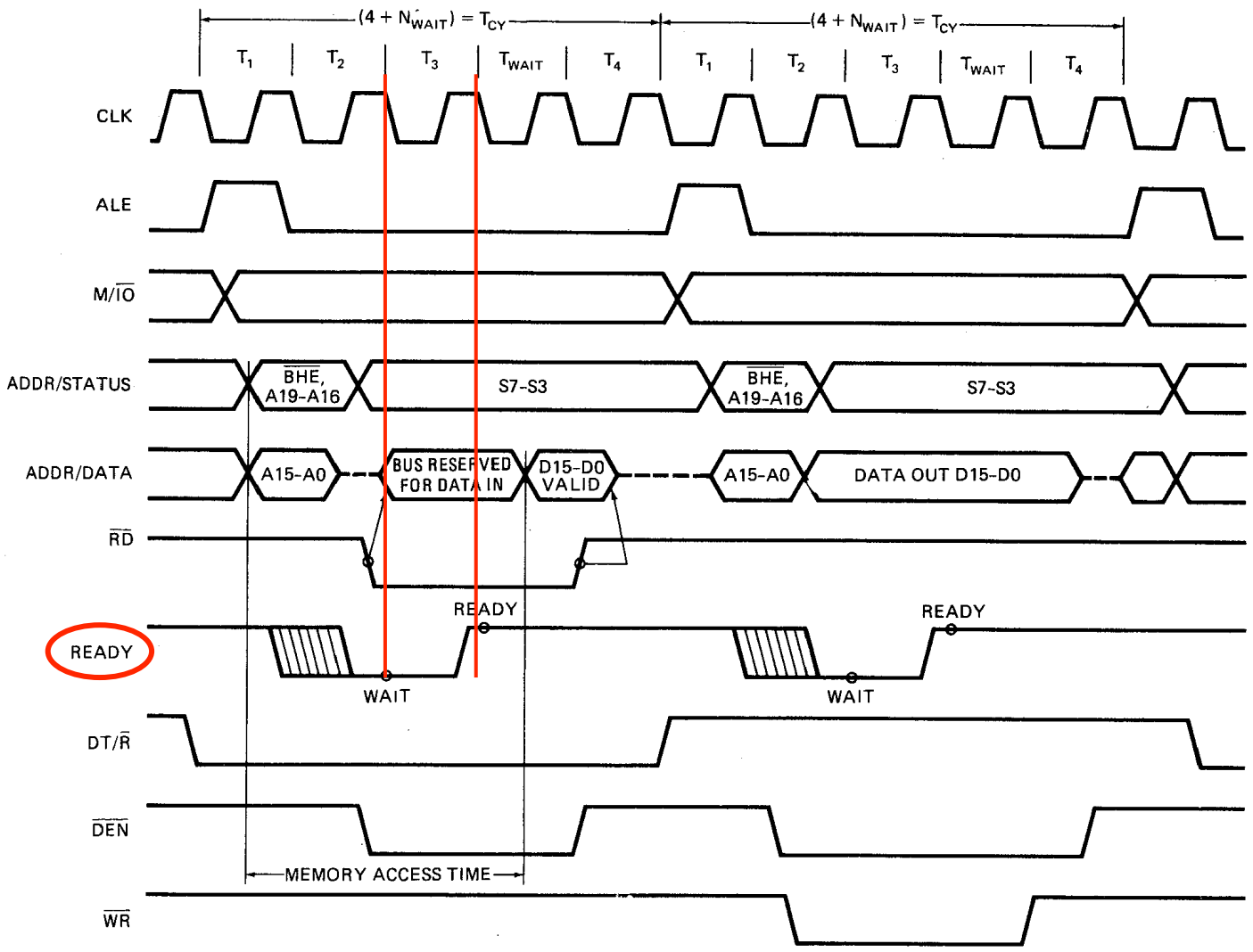
Wait State Generation



(a)

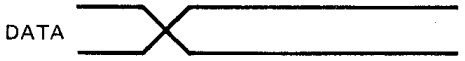
FIGURE 7-1 (a) Block diagram of a simple 8086-based microcomputer. (*See also next page.*)

8086 Bus Timing



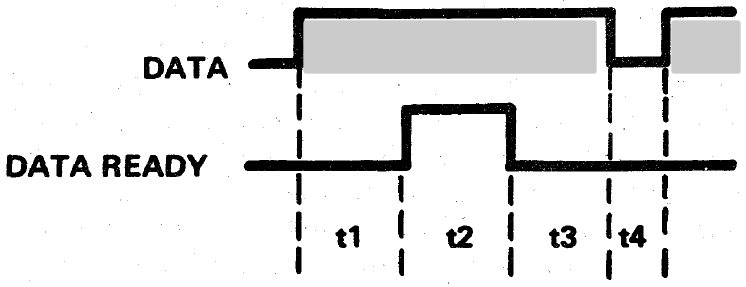
Asynchronous I/O Protocols

Use when sender/receiver may operate at differing speeds
Examples below: level sensitive (not edge triggered)



Asynchronous, source-controlled, no command communication

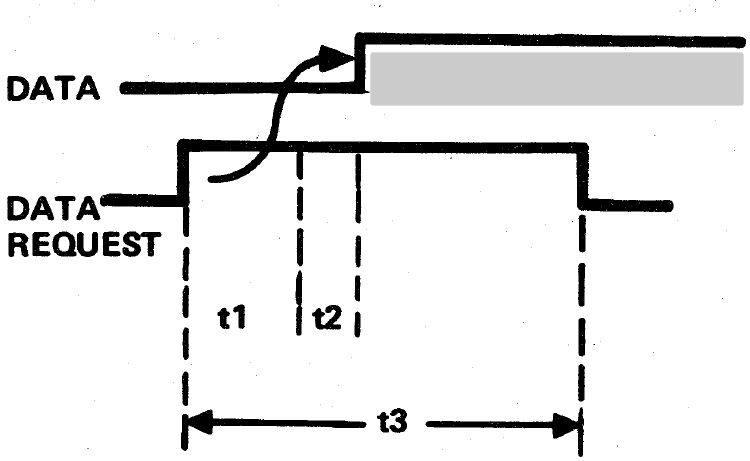
Suitable for display (e.g. signal to LED, temp value to 7-segment display) where no issue of lost data



Asynchronous, source-controlled, one-way command communication

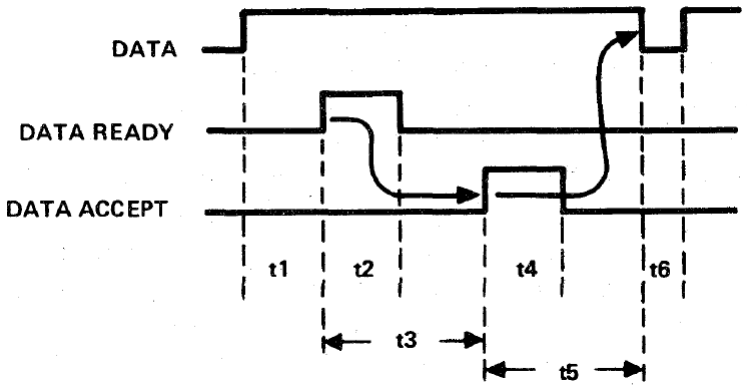
Assumes destination is capable of keeping up (e.g. keyboard to processor)

Asynchronous I/O Protocols



Asynchronous, destination-controlled, one-way command communication

Assumes source is capable of keeping up

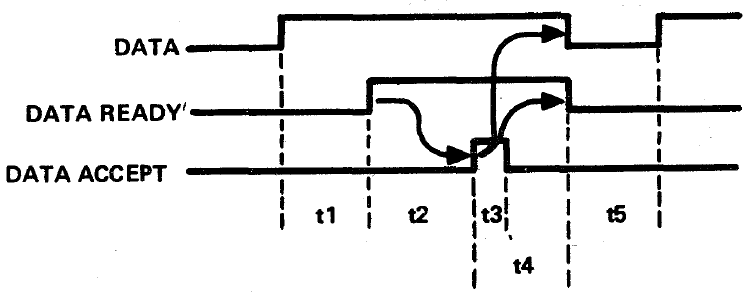


Asynchronous, non-interlocked, request/ acknowledge communication

Suitable for interfacing between devices of different speeds, helps to ensure both devices ready

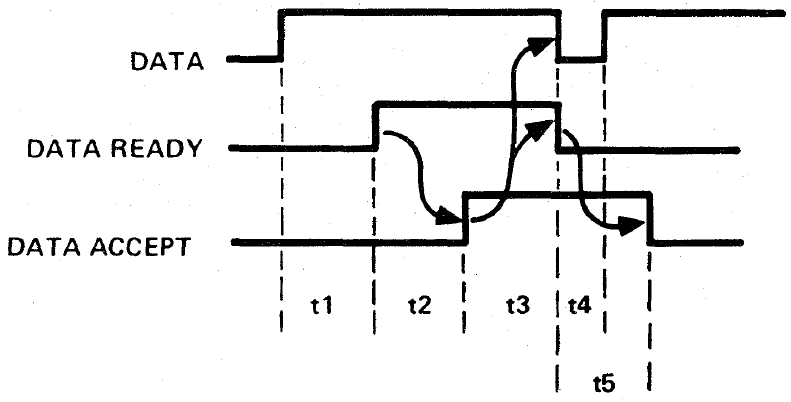
From Thurber et.al.

Asynchronous I/O Protocols



Asynchronous, half-interlocked, request/ acknowledge communication

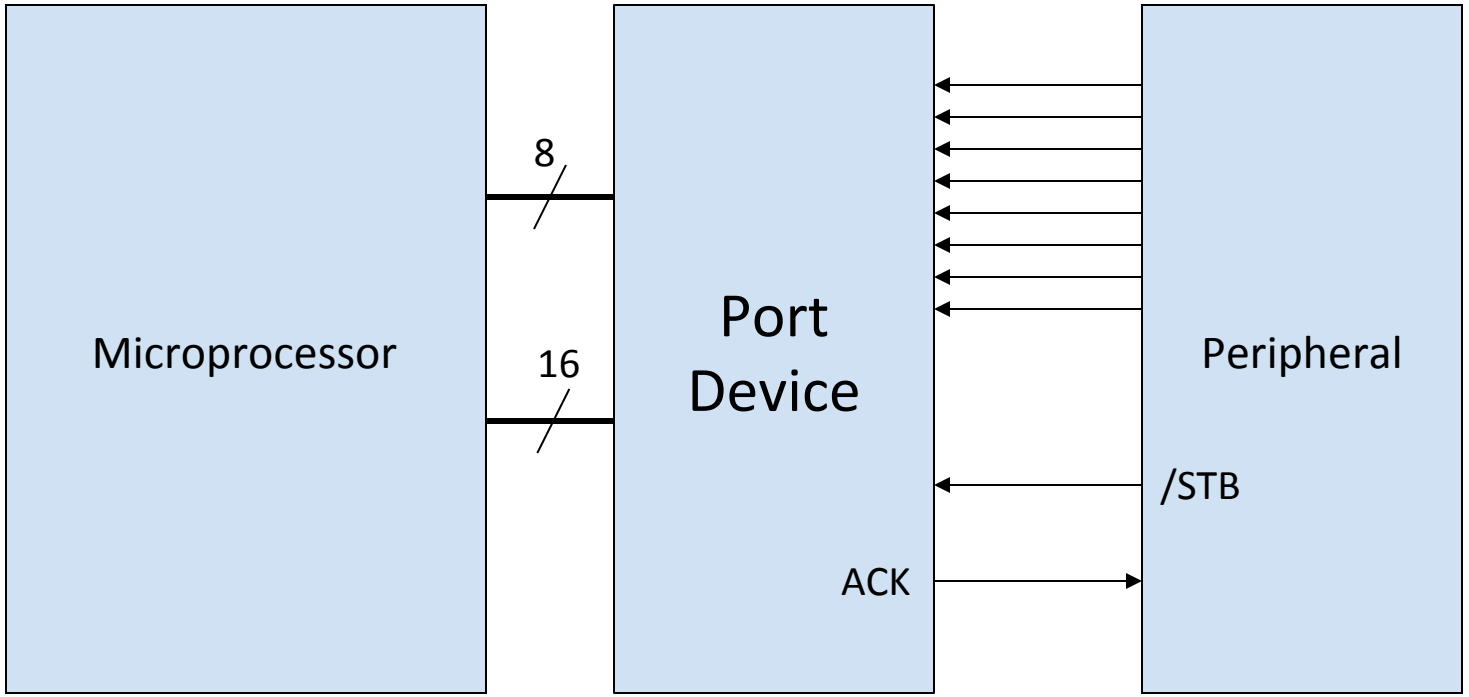
Suitable for interfacing between devices of different speeds, more reliable



Asynchronous, fully-interlocked, request/ acknowledge communication

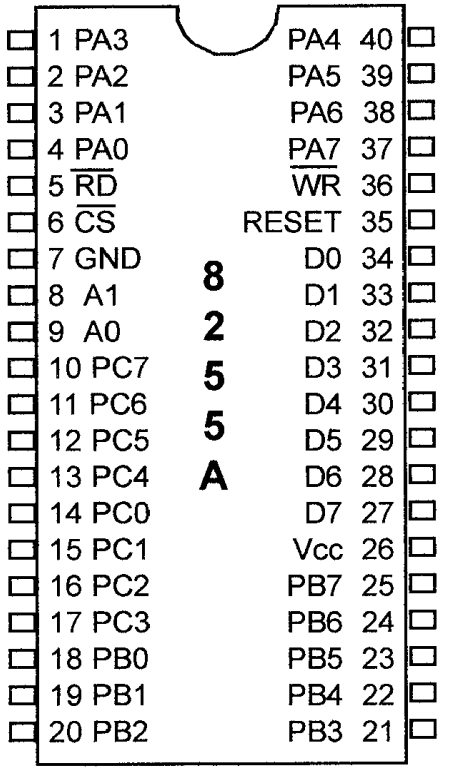
Suitable for interfacing between devices of different speeds, more reliable

Port Devices



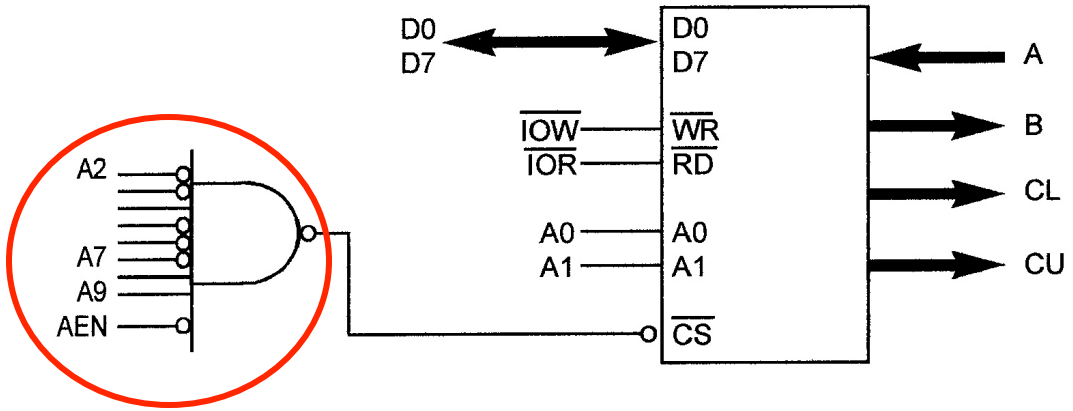
8255 PPI Chip (Programmable Peripheral I/F)

- Provides 3 configurable ports (A,B,C)
  - Port A: 8-bit input or output
  - Port B: 8-bit input or output
  - Port C
    - 8-bit input or output
    - 2 4-bit ports
    - Handshaking signals for ports A, B
- 3 Modes
  - Mode 0: No handshake
  - Mode 1: Strobe handshake
  - Mode 2: Bidirectional handshake
    - (Port A only)



CS	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 is not selected

Addressing the 8255



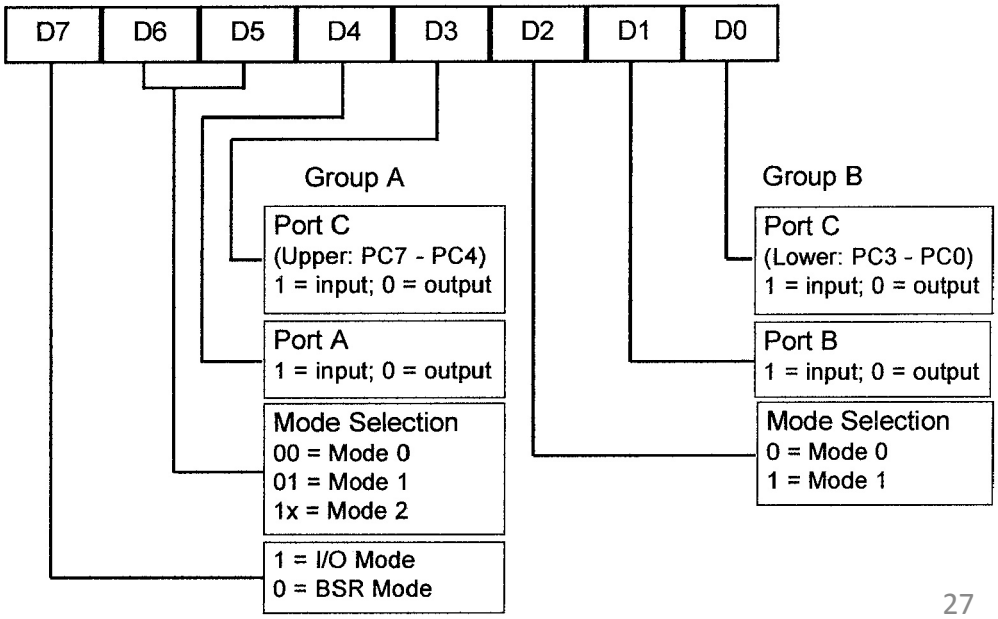
Why not all address bits used?
Cheap!
"Linear Select Decoding"
Results in "aliased" Port IDs!

CS	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 is not selected

$\overline{CS}$	A1	A0	Address	Port
11 0001 00	0	0	310H	Port A
11 0001 00	0	1	311H	Port B
11 0001 00	1	0	312H	Port C
11 0001 00	1	1	313H	Control register

8255 Control Word Format

- MSB (D7) Controls
  - Mode Control Word
  - Bit Set/Reset
    - Set/clear bits in Port C
    - In Mode 1/2



I/O Issues

- What instructions does the processor use to communicate with I/O devices? Example: disk read: address, number of bytes, "read"
  - Isolated (Direct) I/O
  - Memory mapped I/O
- How do we know if an I/O device is ready or an I/O operation is complete
  - Polling
  - Interrupts
- How do we transfer data between the I/O device and memory?
  - Programmed I/O (PIO)
  - Direct memory access (DMA)
    - Bus Mastering

Isolated (Direct) I/O

- I/O Address Space Separate from Memory Address Space
- I/O "Ports" (Numbered)
- Dedicated I/O Instructions
  - IN <destination>, <port> ; fixed or variable port
  - OUT <port>, <source> ; fixed or variable port
- Pros
  - Very easy to interface peripherals
  - Port Device ICs (e.g. 8255)
- Cons
  - Additional H/W and instructions required
  - May not be as flexible as memory-mapped I/O
    - Depending upon addressing modes implemented for IN, OUT

Simple 8255 Example

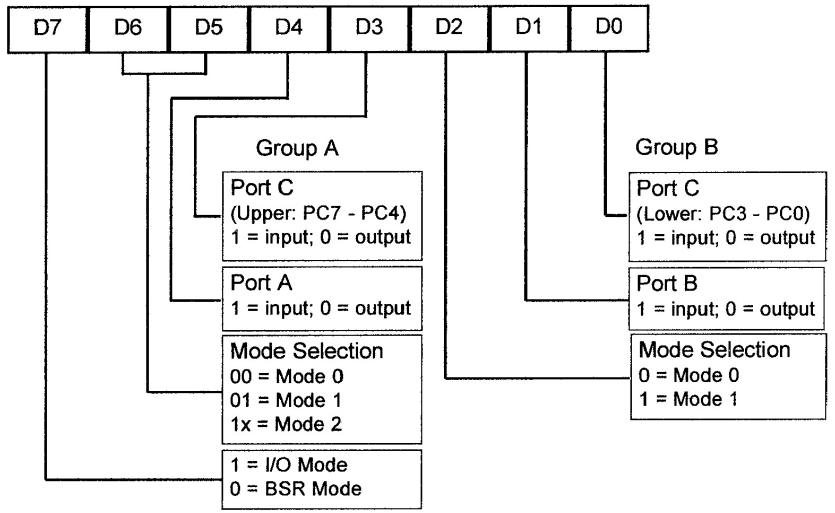
(b) The control word is 90H, or 1001 0000.
(c) One version of the program is as follows:

```
        MOV   AL,90H        ;control byte PA=in, PB=out, PC=out
        MOV   DX,313H       ;load control reg address
        OUT   DX,AL         ;send it to control register
        MOV   DX,310H       ;load PA address
        IN    AL,DX         ;get the data from PA
        MOV   DX,311H       ;load PB address
        OUT   DX,AL         ;send it to PB
        MOV   DX,312H       ;load PC address
        OUT   DX,AL         ;and to PC
```

Configure Port A as input, B as output
Configure both halves of Port C as output
Input data from Port A
Output it to Port B and Port C

Using the EQU directive one can rewrite the above program as follows:

```
CNTLBYTE   EQU   90H     ;PA=in, PB=out, PC=out
PORTA      EQU   310H
PORTB      EQU   311H
PORTC      EQU   312H
CNTLREG    EQU   313H
           ....
           ...
           MOV   AL,CNTLBYTE
           MOV   DX,CNTLREG
           OUT   DX,AL
           MOV   DX,PORTA
           IN    AL,DX
           ;and so on.
```



Hex Range	Device
000 - 01F	DMA controller 1, 8237A-5
020 - 03F	Interrupt controller 1, 8259A, Master
040 - 05F	Timer, 8254-2
060 - 06F	8042 (keyboard)
070 - 07F	Real-time clock, NMI mask
080 - 09F	DMA page register, 74LS612
0A0 - 0BF	Interrupt controller 2, 8237A-5
0C0 - 0DF	DMA controller 2, 8237A-5
0F0	Clear math coprocessor busy
0F1	Reset math coprocessor
0F8 - 0FF	Math coprocessor
1F0 - 1F8	Fixed disk
200 - 207	Game I/O
20C - 20D	Reserved
21F	Reserved
278 - 27F	Parallel printer port 2
2B0 - 2DF	Alternate enhanced graphics adapter
2E1	GPIB (adapter 0)
2E2 & 2E3	Data acquisition (adapter 0)
2F8 - 2FF	Serial port 2
300 - 31F	Prototype card
360 - 363	PC network (low address)
364 - 367	Reserved
368 - 36B	PC network (high address)
36C - 36F	Reserved
378 - 37F	Parallel printer port 1
380 - 38F	SDLC, bisynchronous 2
390 - 393	Cluster
3A0 - 3AF	Bisynchronous 1
3B0 - 3BF	Monochrome display and printer adapter
3C0 - 3CF	Enhanced graphics adapter
3D0 - 3DF	Color/graphics monitor adapter
3F0 - 3F7	Diskette controller
3F8 - 3FF	Serial port 1
6E2 & 6E3	Data acquisition (adapter 1)
790 - 793	Cluster (adapter 1)
AE2 & AE3	Data acquisition (adapter 2)
B90 - B93	Cluster (adapter 2)
EE2 & EE3	Data acquisition (adapter 3)
1390 - 1393	Cluster (adapter 3)
22E1	GPIB (adapter 1)
2390 - 2393	Cluster (adapter 4)
42E1	GPIB (adapter 2)
62E1	GPIB (adapter 3)
82E1	GPIB (adapter 4)
A2E1	GPIB (adapter 5)
C2E1	GPIB (adapter 6)
E2E1	GPIB (adapter 7)

Memory-Mapped I/O

- I/O Treated Exactly Like Memory Access
- I/O Devices Share Address Space with Memory
- Use Normal Instructions for I/O
  - MOV AL, MemAddr ; read byte
  - MOV MemAddr, AL ; write byte
  - Full addressing modes available (++, [])
  - Caution: I/O registers read as side-effect
- I/O Devices Do Address Decoding

Memory-Mapped I/O

- Pros
  - "Regular" architecture
  - No special I/O instructions
    - Simplifies processor architecture
    - Can write I/O device drivers in C
  - Full addressing modes available
    - Increment, indirect
    - Memory to memory instructions
      - Can test/set bits in a control word without using an additional I/O instruction to bring the control word into a register
  - Can protect against random process accessing I/O by using page tables (don't map I/O addresses into process address space)
- Cons
  - Must have solution to avoid caching device registers!
  - Address space consumed (not really an issue!)
- What does Intel IA32 architecture use?

Both!

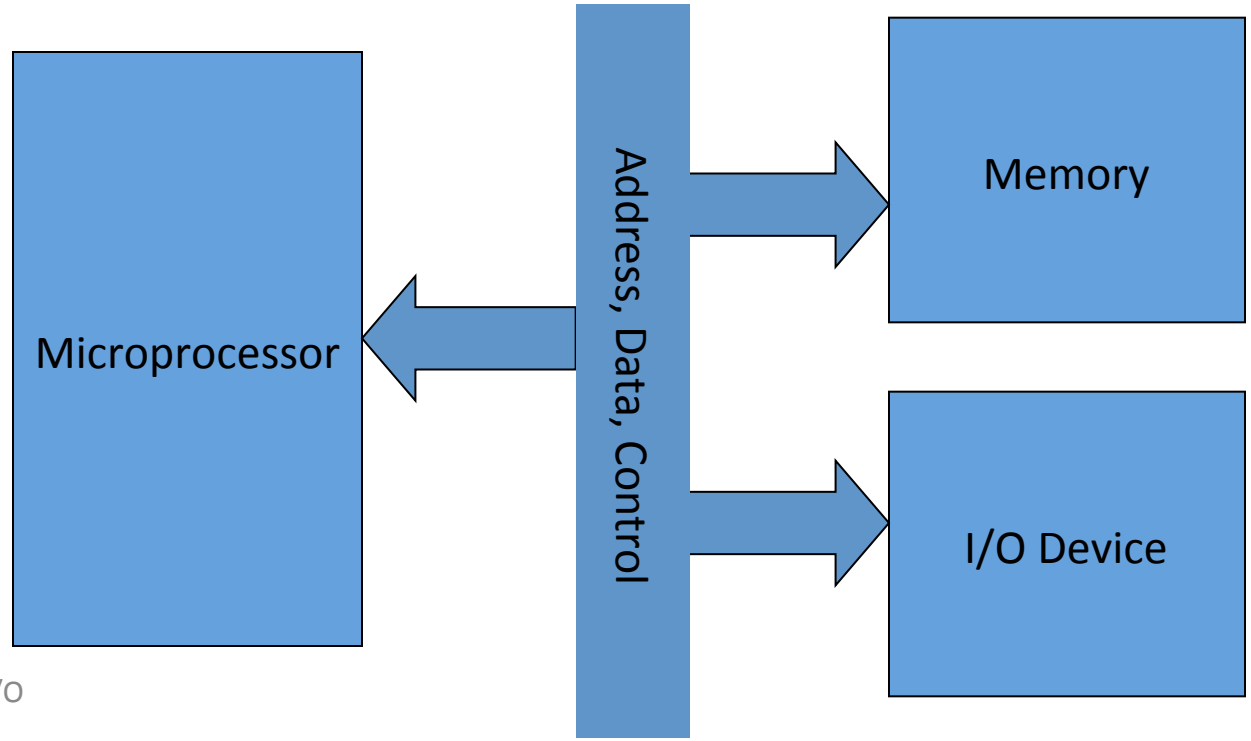
How Do You Get Notified of I/O Completion/Ready?

- Polled I/O
- Interrupt Driven I/O

Polling

```
READB:  IN      AL, BUSY        ; get BUSY flag
        TEST    AL, BUSY_BIT    ; test BUSY bit
        JNE     READB           ; still busy -- loop
        IN      AL, DATAP       ; read data
```

Simplified Polling Code Fragment

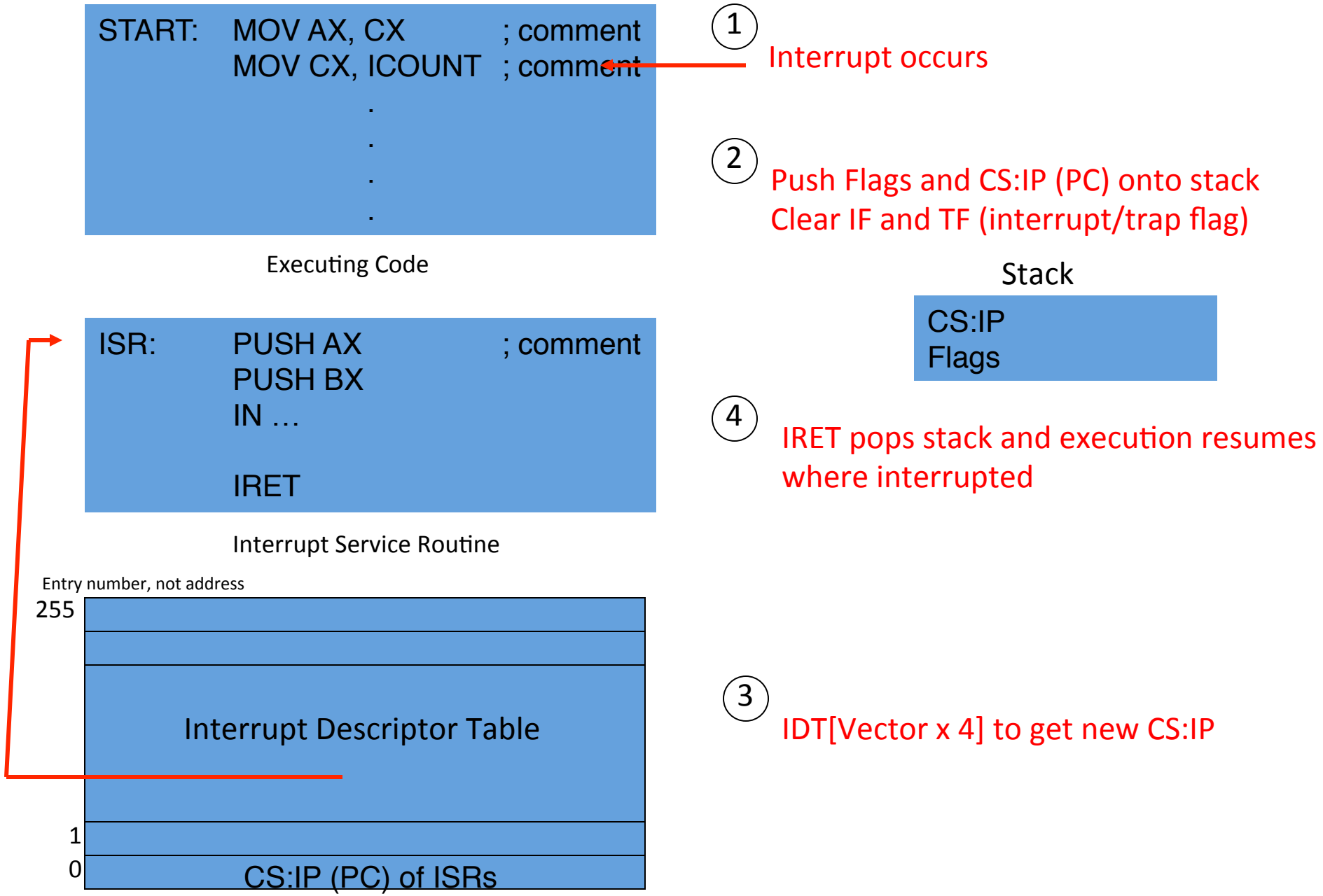


Interrupt-Driven I/O

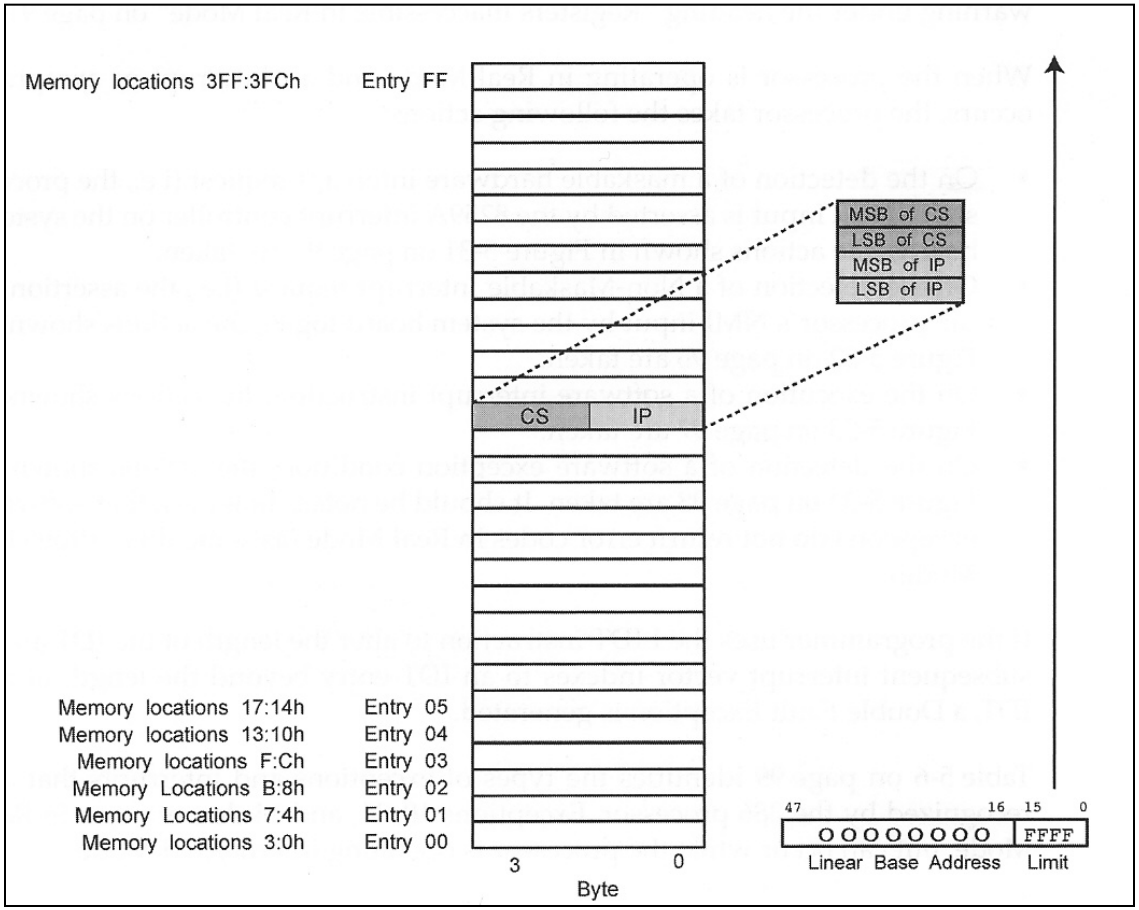
- Permits the processor to execute useful instructions instead of polling an I/O device
- The I/O device interrupts the processor when it needs attention (e.g. has data) or an I/O operation has completed
  - Keyboard character typed
  - Earlier requested disk block has been read

Interrupt Processing Sequence

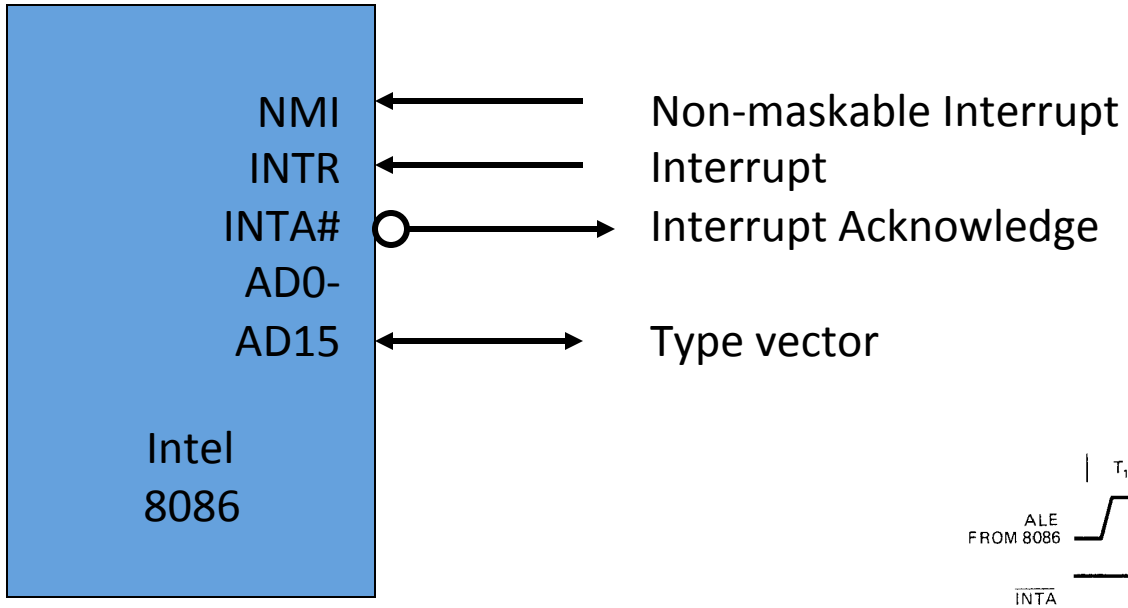
- Interrupt occurs
- Push current processor state onto stack
  - Flags
  - CS:IP
- Clear IF and TF (Interrupt and Trap Flags)
  - Prevents further interrupts
- Retrieve Instruction Pointer (CS:IP) Using Interrupt Vector as index into Interrupt Descriptor Table (IDT)
- Begin executing with new CS:IP of ISR
- Service the interrupt (Interrupt Service Routine)
- ISR ends by executing IRET instruction (explicit)
  - Pops CS:IP and Flags from Stack

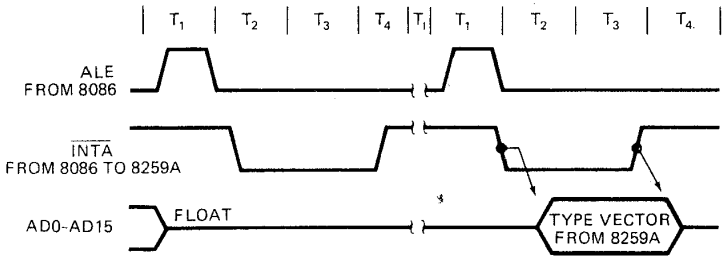

START: MOV AX, CX ; comment
MOV CX, ICOUNT ; comment
Executing Code
ISR: PUSH AX ; comment
PUSH BX
IN …
IRET
Interrupt Service Routine
Entry number, not address
255
Interrupt Descriptor Table
1
0
CS:IP (PC) of ISRs
1
Interrupt occurs
2
Push Flags and CS:IP (PC) onto stack
Clear IF and TF (interrupt/trap flag)
Stack
CS:IP
Flags
4
IRET pops stack and execution resumes where interrupted
3
IDT[Vector x 4] to get new CS:IP

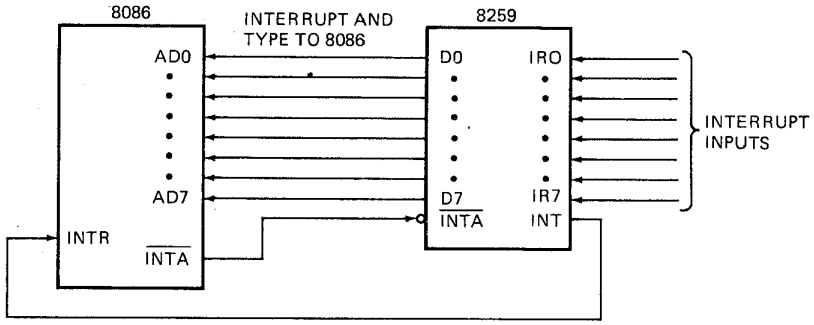
Why multiply interrupt by 4?



Interrupt Hardware: Vectors







The Interrupt Mechanism

- Hardware Interrupts (Asynchronous)
  - Non-Maskable Interrupts (NMI)
  - Priority Maskable Interrupts (INTR)
- Software Interrupts (Synchronous)
  - Nomenclature varies
    - Intel calls these “exceptions” to distinguish them from H/W “interrupts”
    - Often called “traps” or “faults”

The Interrupt Mechanism

- Non-Maskable Interrupts
  - Power Failure
    - Enough time to save state and shutdown gracefully
- Priority Maskable Interrupts
  - I/O Devices Assigned Different Priorities
    - Relative importance of servicing request
      - (e.g. keyboard or disk)
    - Ability of device to buffer data

The Interrupt Mechanism

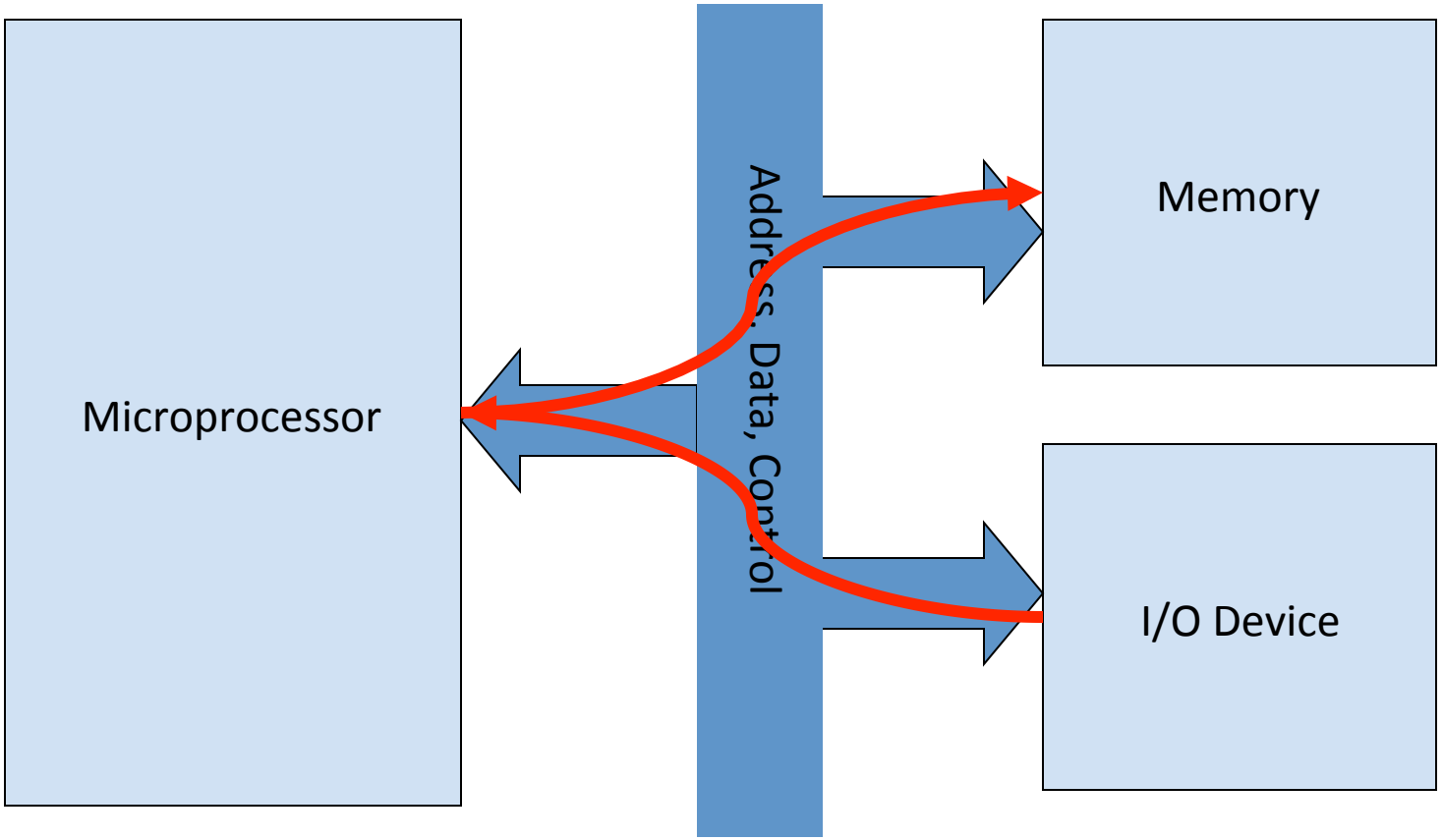
- Software Interrupts (Synchronous)
  - Divide-by-Zero
  - Page fault or segmentation violation
  - Debugging (breakpoints and single step)
  - Operating System Calls
    - User program makes O.S. request
    - INT ## instruction

Vector Number	Description
0	Divide Error
1	Debugger Call
2	NMI Interrupt
3	Breakpoint
4	INTO-detected Overflow
5	BOUND Range Exceeded
6	Invalid Opcode
7	Device Not Available
8	Double Fault
9	(Intel reserved. Do not use. Not used by Pentium™ processor.)
10	Invalid Task State Segment
11	Segment Not Present
12	Stack Exception
13	General Protection
14	Page Fault
15	(Intel reserved. Do not use.)
16	Floating-Point Error
17	Alignment Check
18	Machine Check Exception
19-31	(Intel reserved. Do not use.)
32-255	Maskable Interrupts

How Do You Transfer Data?

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

Programmed I/O: Input Example



Requires fetching of instructions to be executed to do the I/O
Requires moving data from I/O device to processor and then to memory

Disk Read Example

- Initiate an I/O operation (e.g. read a block of 512 bytes from disk)
  - Write to device indicating: operation (read), disk address, bytes, etc
  - Write to device instructing it to begin
- Determine when device has completed operation and has data
  - Polling
    - Read the device's status
  - Interrupt
    - Receive interrupt when I/O device operation complete and data ready
- Transfer data from device to memory
  - Programmed I/O (read a byte at a time from I/O device)
  - DMA
    - Rely on hardware to transfer the data without involving the CPU)

Programmed I/O
Read Count Bytes

```
START:  MOV     BX, OFFSET BUFFER
        MOV     DX, FFF8H           ; set up I/O port
        MOV     CX, COUNT           ; set up byte count
READB:  IN      AL, DX              ; read byte
        MOV     [BX], AL            ; copy to buffer
        INC     BX                  ; increment pointer
        LOOP    READB               ; decrement/test CX
```

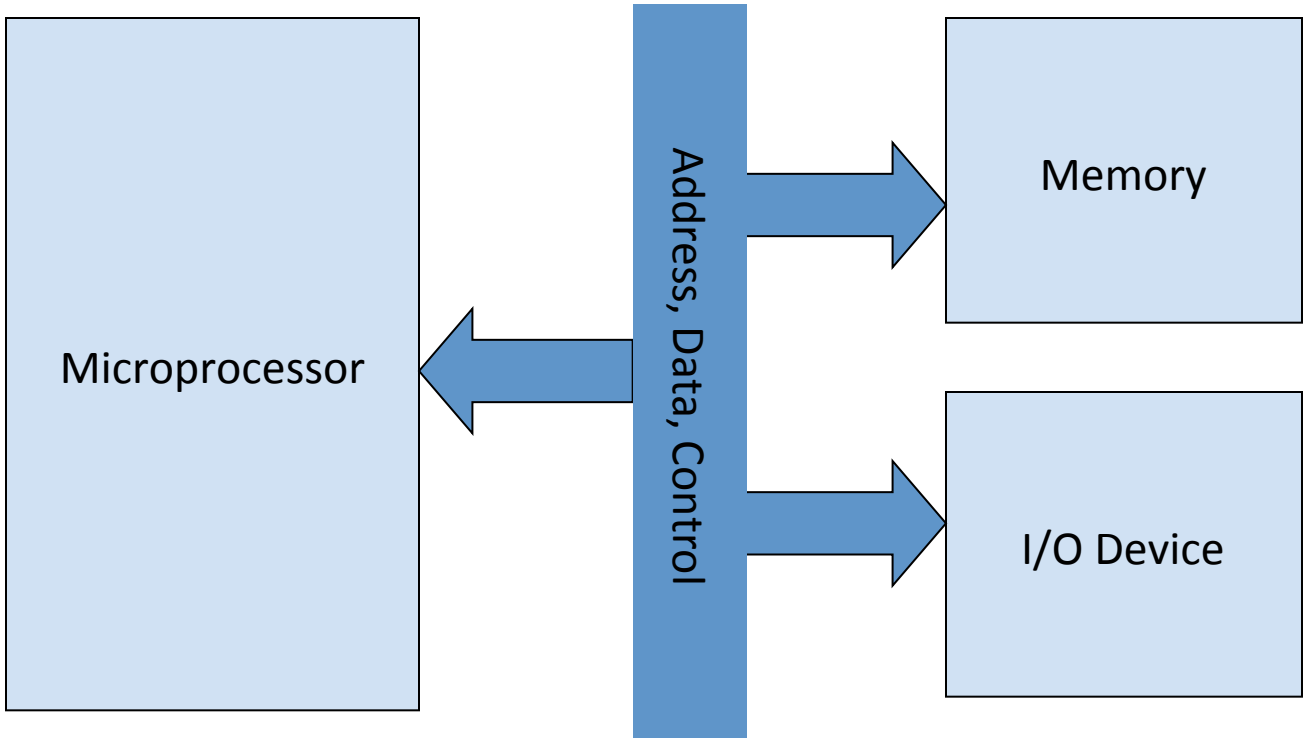
8
9+9 (EA)
3
5/17 if loop taken
34-46

Simplified I/O Read Code Fragment

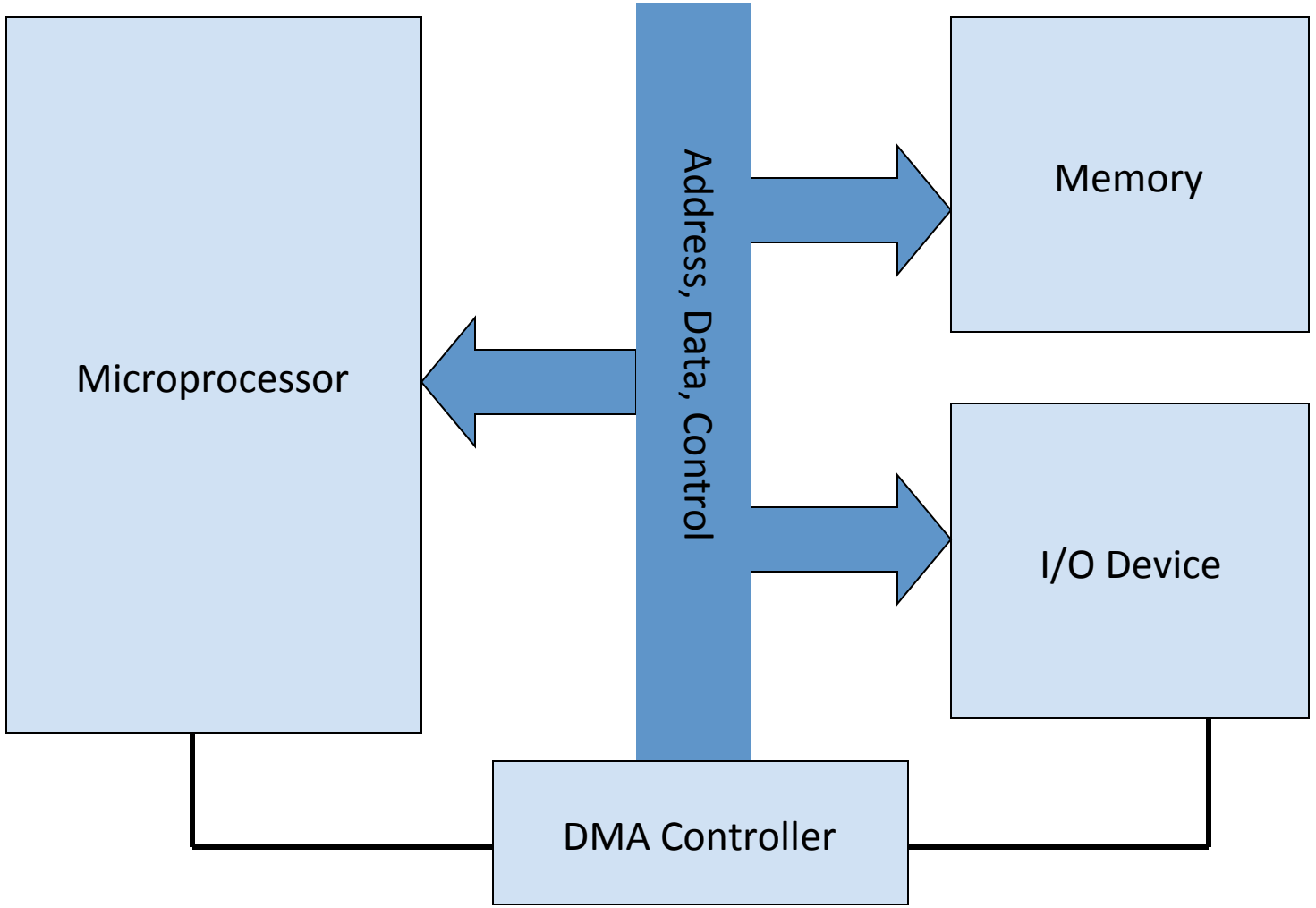
How many clock cycles?

```
START:  MOV     BX, OFFSET BUFFER
        MOV     DX, FFF8H           ; set up I/O port
        MOV     CX, COUNT           ; set up byte count
READB:  IN      AL, DX              ; read byte
        MOV     [BX], AL            ; copy to buffer
        INC     BX                  ; increment pointer
        LOOP    READB               ; decrement/test CX
```

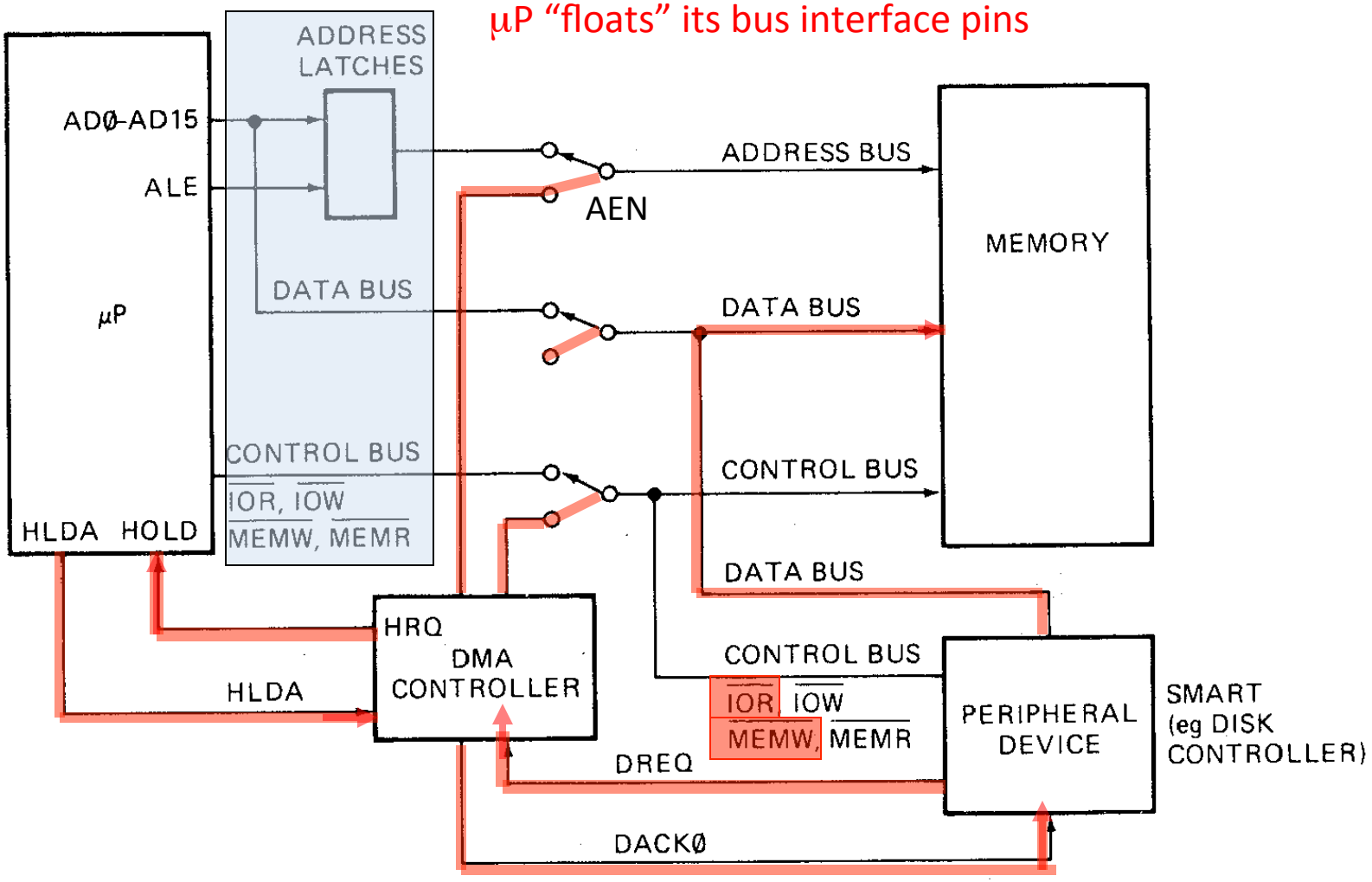
Simplified I/O Read Code Fragment



Direct Memory Access (DMA)



DMA (Detail)


µP "floats" its bus interface pins
ADDRESS LATCHES
AD0-AD15
ALE
µP
DATA BUS
CONTROL BUS
IOR, IOW
MEMW, MEMR
HLDA HOLD
AEN
ADDRESS BUS
MEMORY
DATA BUS
CONTROL BUS
DATA BUS
HRQ
DMA CONTROLLER
HLDA
CONTROL BUS
IOR, IOW
MEMW, MEMR
PERIPHERAL DEVICE
SMART (eg DISK CONTROLLER)
DREQ
DACK0

DMA Timing

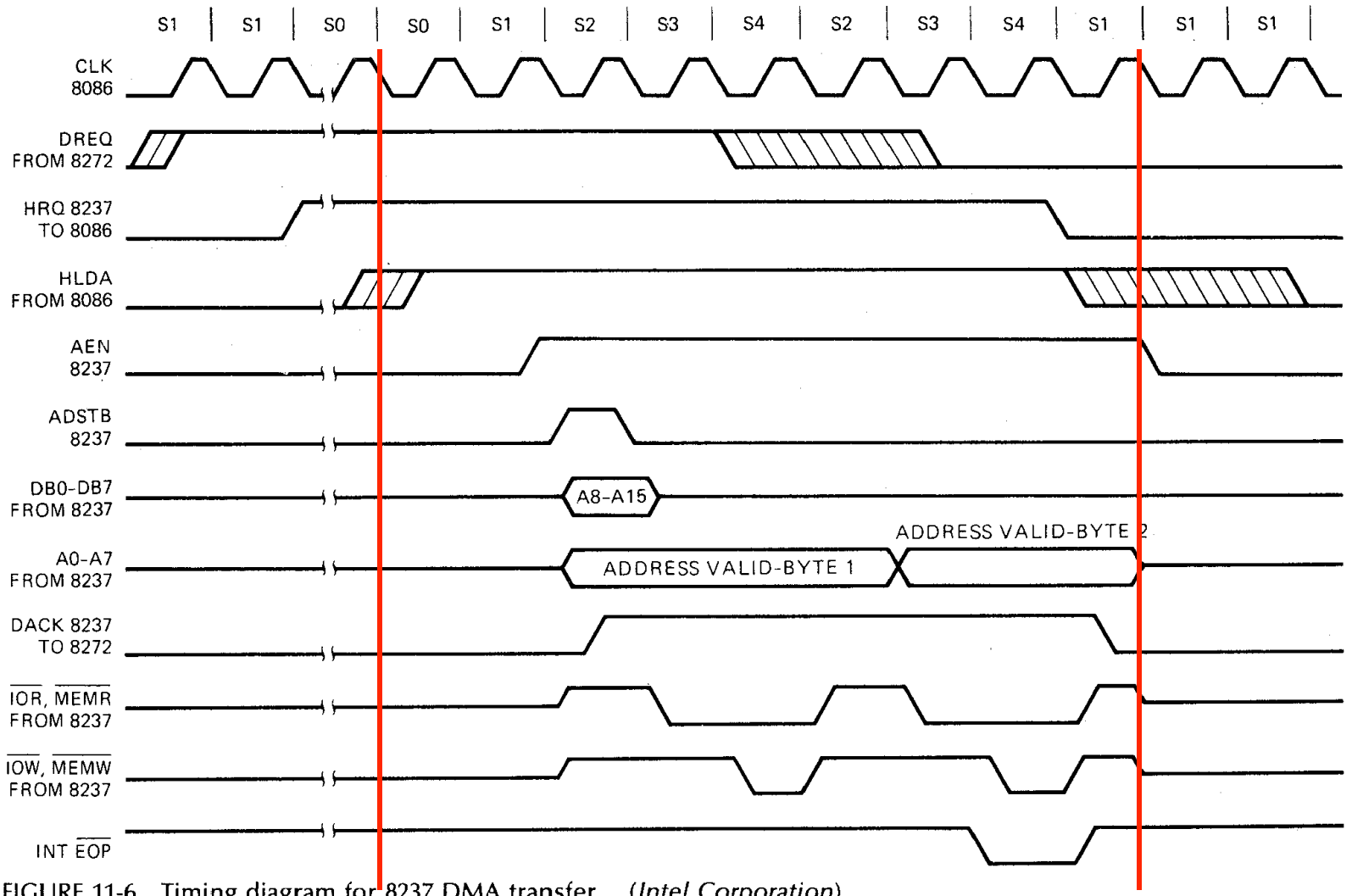
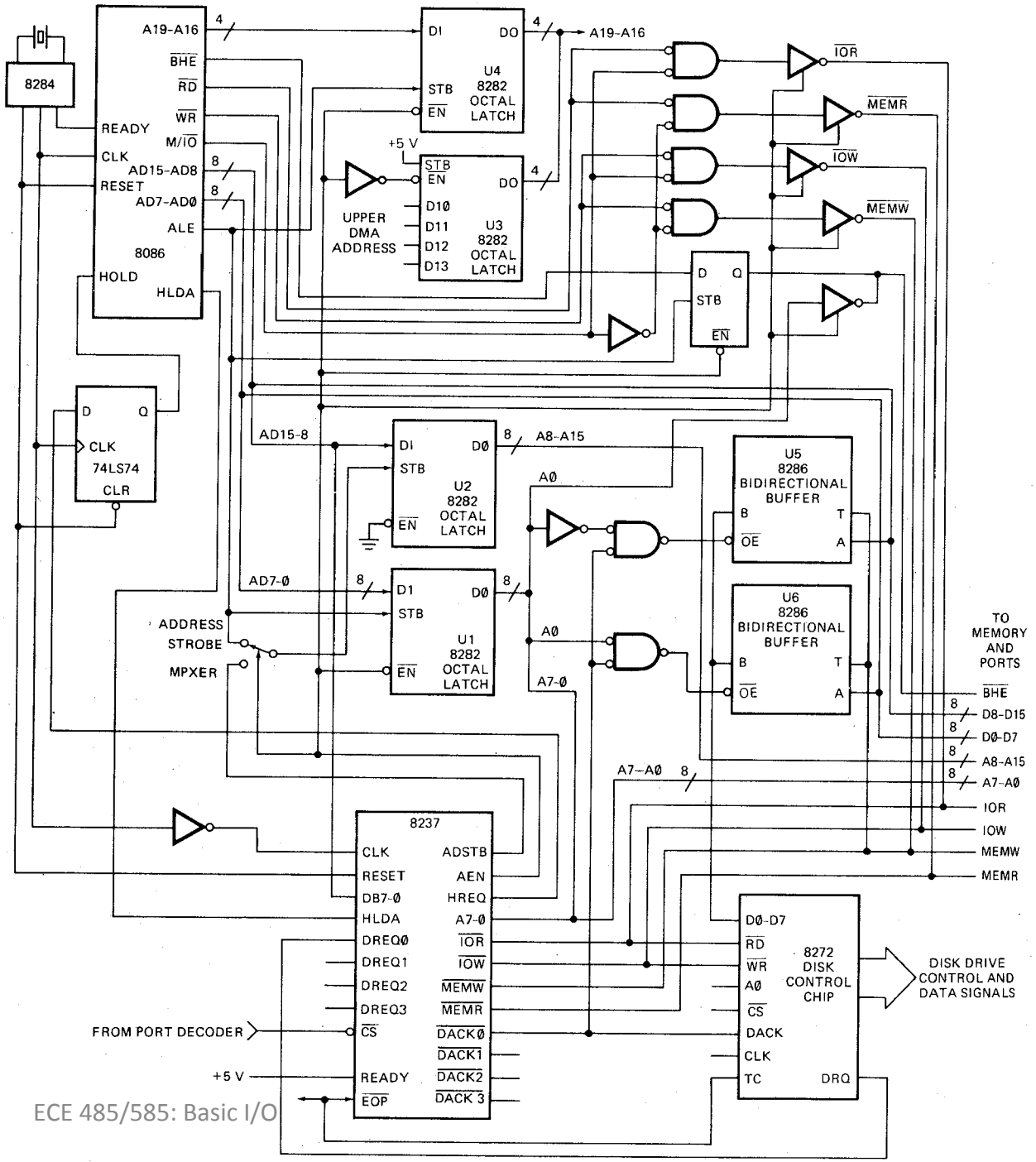
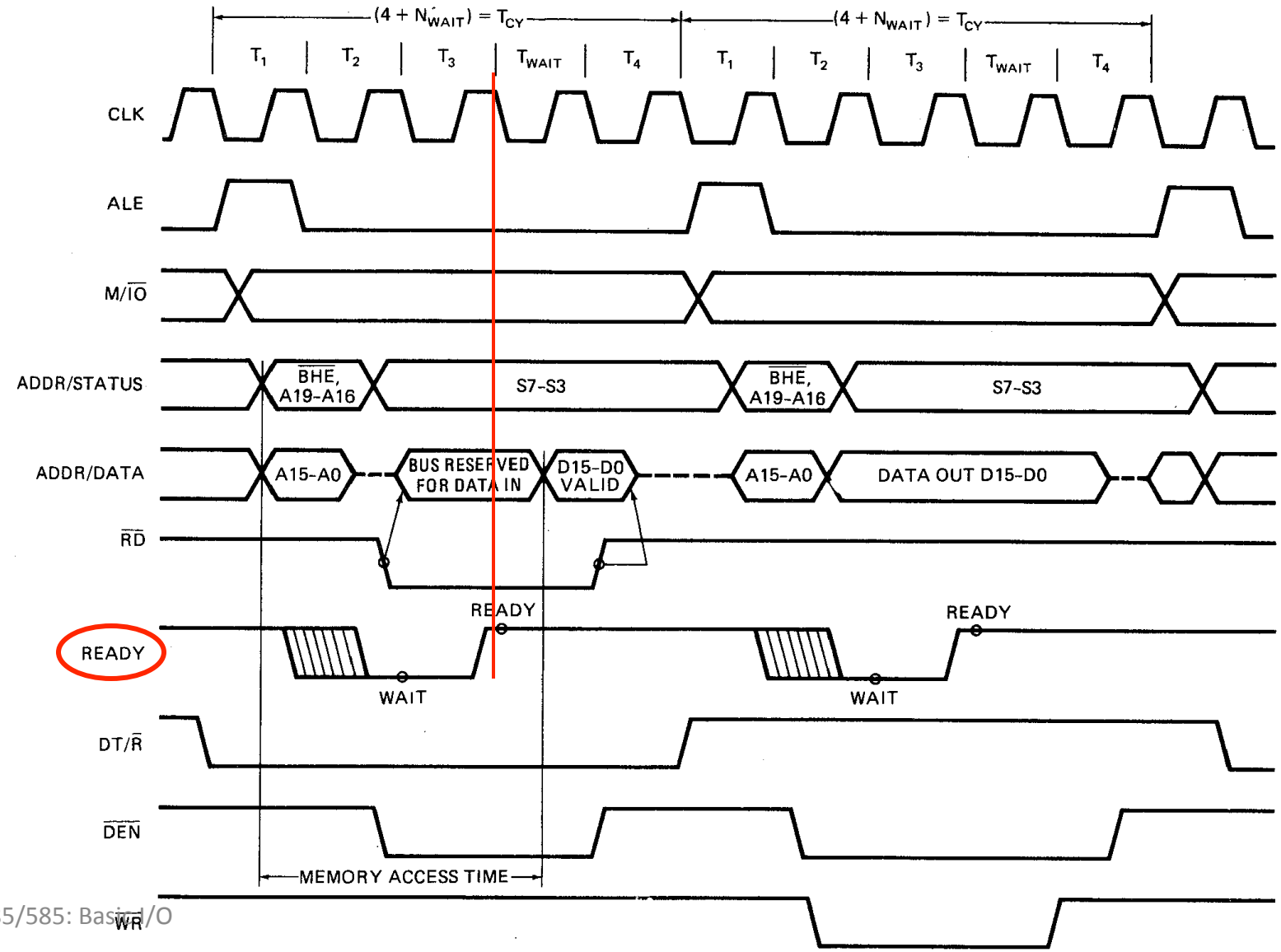


FIGURE 11-6 Timing diagram for 8237 DMA transfer. (*Intel Corporation*)

8237 DMA Controller



8086 Bus Timing



Summary

- Architecture
  - Dedicated (Isolated or Direct) I/O Ports
  - Memory-Mapped I/O
- Event Notification/Synchronization
  - Polling
  - Interrupt-Driven
- Data Transfer Programming
  - Programmed I/O
  - Direct Memory Access (DMA)

System Bus/Global Bus

Local Bus

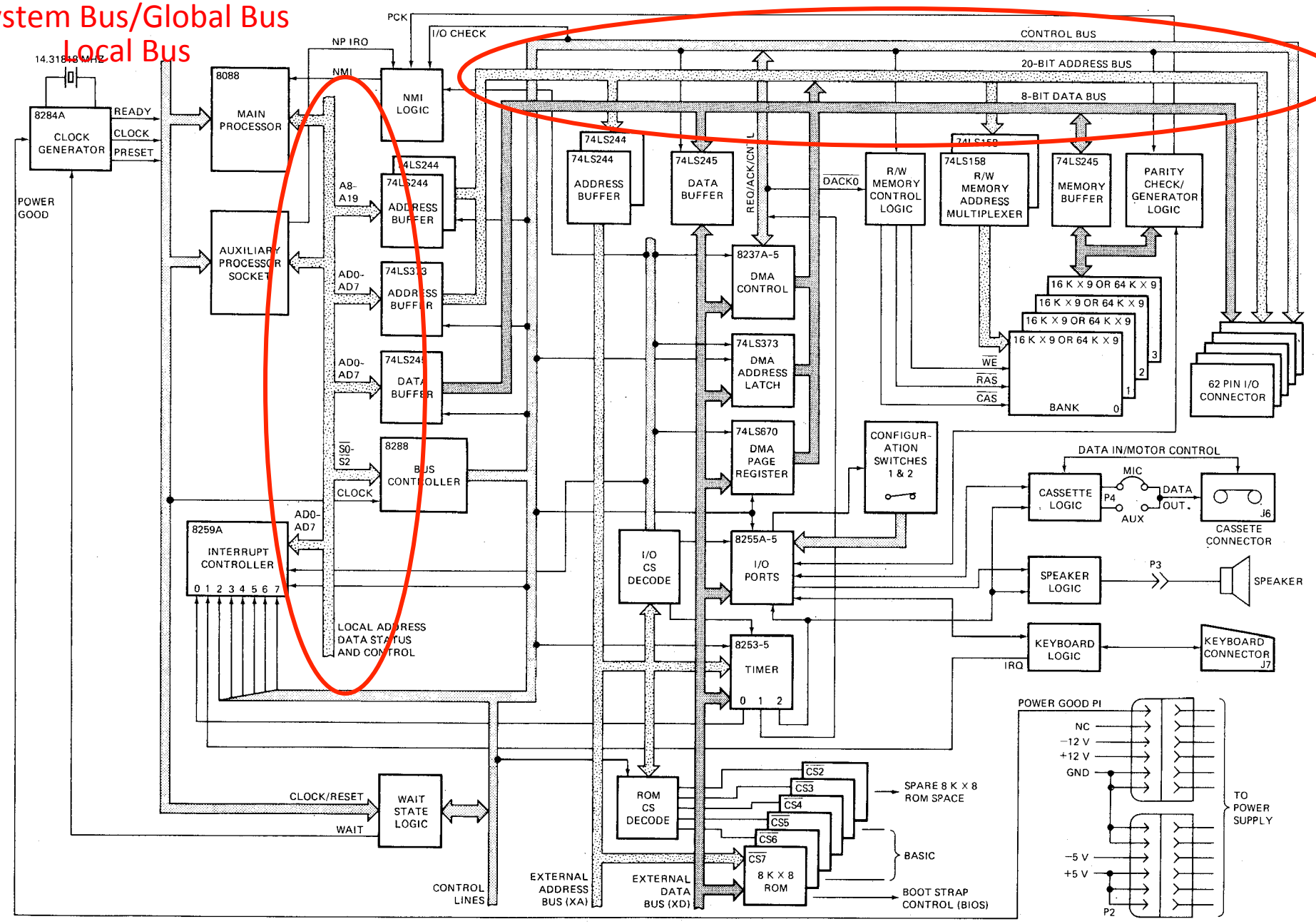


FIGURE 11-2 Block diagram of circuitry on IBM PC motherboard. (*IBM Corporation*)

Technical Specifications for Intel Microprocessors

Technical Specifications

1970s Processors

	4004	8008	8080	8086	8088
Introduced	11/15/71	4/1/72	4/1/74	6/8/78	6/1/79
Clock Speeds	108KHz	200KHz	2MHz	5MHz, 8MHz, 10MHz	5MHz, 8MHz
Bus Width	4 bits	8 bits	8 bits	16 bits	8 bits
Number of Transistors	2,300 (10 microns)	3,500 (10 microns)	4,500 (6 microns)	29,000 (3 microns)	29,000 (3 microns)
Addressable Memory	640 bytes	16 KBytes	64 KBytes	1 MB	1 MB
Virtual Memory	--	--	--	--	--
Brief Description	First microcomputer chip, Arithmetic manipulation	Data/character manipulation	10X the performance of the 8008	10X the performance of the 8080	Identical to 8086 except for its 8-bit external bus

Table 2-2. Key Features of Previous Generations of IA-32 Processors

Intel Processor	Date Intro-duced	Max. Clock Frequency at Intro-duction	Transis-tors per Die	Register Sizes[1]	Ext. Data Bus Size[2]	Max. Extern. Addr. Space	Caches
8086	1978	8 MHz	29 K	16 GP	16	1 MB	None
Intel 286	1982	12.5 MHz	134 K	16 GP	16	16 MB	Note 3
Intel386 DX Processor	1985	20 MHz	275 K	32 GP	32	4 GB	Note 3
Intel486 DX Processor	1989	25 MHz	1.2 M	32 GP 80 FPU	32	4 GB	L1: 8KB
Pentium Processor	1993	60 MHz	3.1 M	32 GP 80 FPU	64	4 GB	L1:16KB
Pentium Pro Processor	1995	200 MHz	5.5 M	32 GP 80 FPU	64	64 GB	L1: 16KB L2: 256KB or 512KB
Pentium II Processor	1997	266 MHz	7 M	32 GP 80 FPU 64 MMX	64	64 GB	L1: 32KB L2: 256KB or 512KB
Pentium III Processor	1999	500 MHz	8.2 M	32 GP 80 FPU 64 MMX 128 XMM	64	64 GB	L1: 32KB L2: 512KB

NOTES:

1. The register size and external data bus size are given in bits. Note also that each 32-bit general-purpose (GP) registers can be addressed as an 8- or a 16-bit data registers in all of the processors.
2. Internal data paths that are 2 to 4 times wider than the external data bus for each processor.

Binary compatibility from 1978 to 2010!!