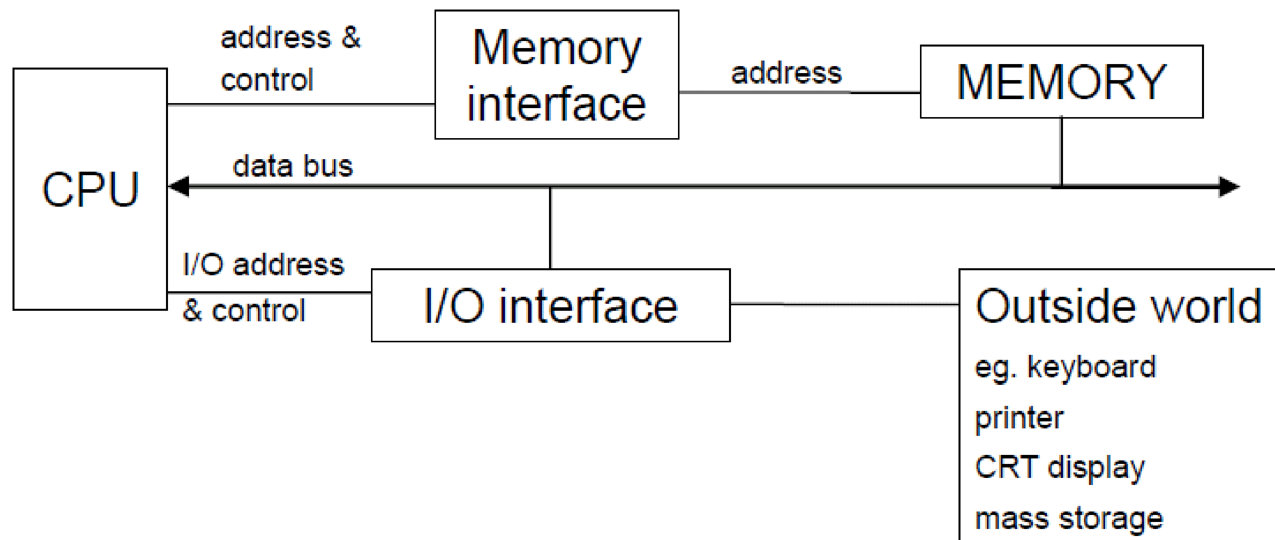
Design with Microprocessors

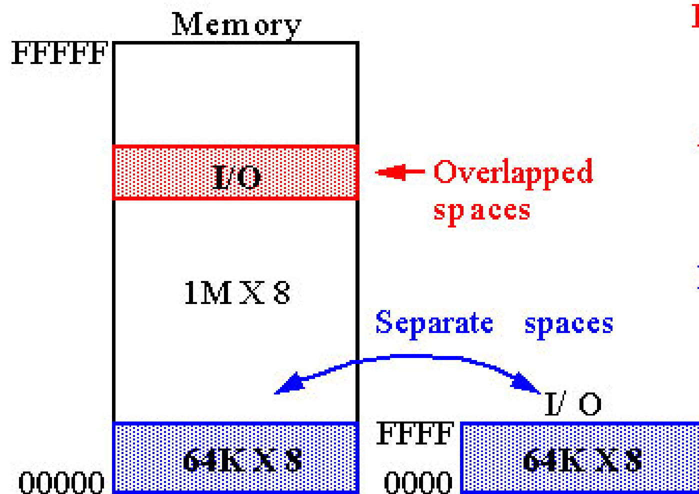
Year III Computer Science

1-st Semester

Lecture 11: I/O transfer with x86

I/O Transfer





Disadvantage:

A portion of the memory space is used for I/O devices.

Advantage:

$\overline{IORC}$ and $\overline{IOWC}$ not required.

Any data transfer instruction.

Disadvantage:

Hardware using M/$\overline{IO}$ and W/$\overline{R}$ needed to develop signals $\overline{IORC}$ and $\overline{IOWC}$.

Requires IN, OUT, INS and OUTS

I/O Instructions

We discussed IN, OUT, INS and OUTS as instructions for the transfer of data to and from an I/O device.

IN and **OUT** transfer data between an I/O device and the microprocessor's accumulator (AL, AX or EAX).

The I/O address is stored in:

A byte, immediately following the opcode (fixed address: 00H .. FFH).

IN *AL*, 19H ;8-bits are saved to AL from I/O port 19H.

OUT 19H, *AX* ;16-bits are written to I/O port 0019H.

Register DX as a 16-bit I/O address (variable addressing: 100H ...)

IN *AX*, DX ;16-bits are saved to AX.

IN *EAX*, DX ;32-bits are saved to EAX (32 bit processors)

OUT *DX*, *AX* ;16-bits are written to port DX from AX.

OUT *DX*, *EAX* ;32-bits are written to port DX from EAX (32 bit processors)

INS and **OUTS** transfer to I/O devices using ES:DI and DS:SI, respectively.

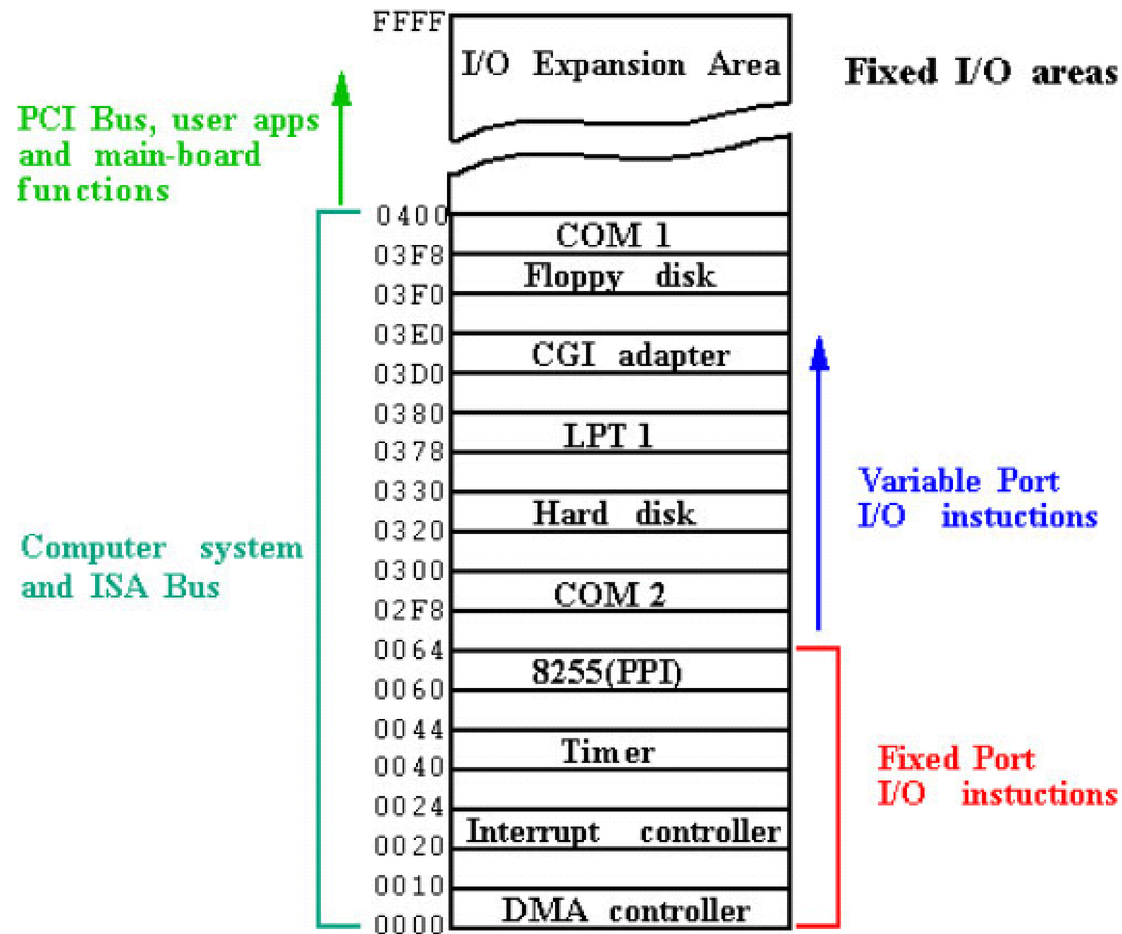
Only 16-bits (A0 to A15) are decoded (Address connections above A15 are undefined for I/O instructions)

I/O Instructions

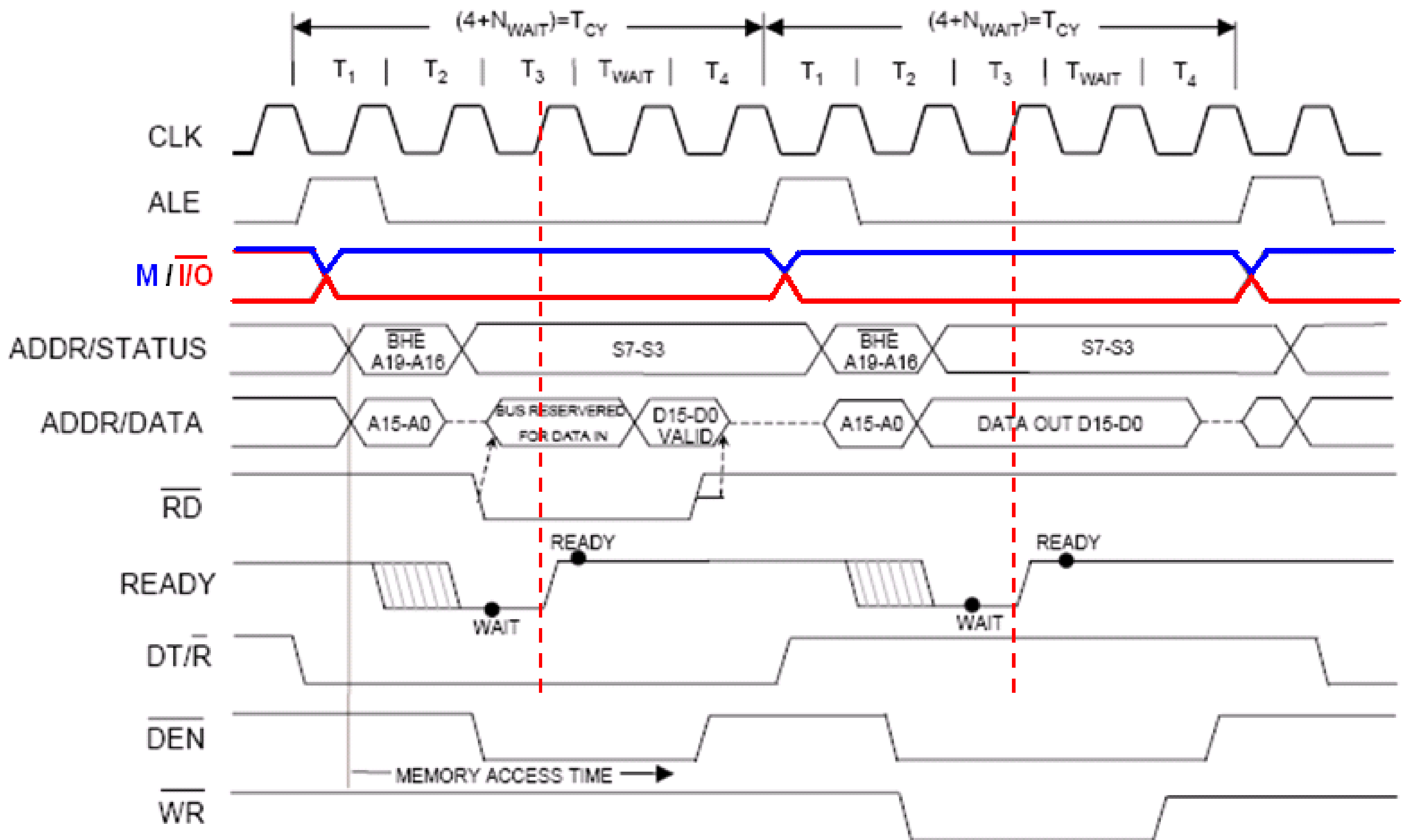
TABLE 10–1 Input/output instructions

Instruction	*Data Width*	*Function*
IN AL,p8	8	A byte is input from port p8 into AL
IN AX,p8	16	A word is input from port p8 into AX
IN EAX, p8	32	A doubleword is input from port p8 into EAX
IN AL,DX	8	A byte is input from the port addressed by DX into AL
IN AX,DX	16	A word is input from the port addressed by DX into AX
IN EAX,DX	32	A word is input from the port addressed by DX into EAX
INSB	8	A byte is input from the port addressed by DX into the extra segment memory location addressed by DI, then DI = DI ± 1
INSW	16	A word is input from the port addressed by DX into the extra segment memory location addressed by DI, then DI = DI ± 2
INSD	32	A doubleword is input from the port addressed by DX into the extra segment memory location addressed by DI, then DI ± 4
OUT p8,AL	8	A byte is output from AL to port p8
OUT p8,AX	16	A word is output from AX to port p8
OUT p8,EAX	32	A doubleword is output from EAX to port p8
OUT DX,AL	8	A byte is output from AL to the port addressed by DX
OUT DX,AX	16	A word is output from AX to the port addressed by DX
OUT DX,EAX	32	A doubleword is output from EAX to the port addressed by DX
OUTSB	8	A byte is output from the data segment memory location addressed by SI to the port addressed by DX, then SI = SI ± 1
OUTSW	16	A word is output from the data segment memory locations addressed by SI to the port addressed by DX, then SI = SI ± 2
OUTSD	32	A doubleword is output from the data segment memory locations addressed by SI to the port addressed by DX, then SI = SI ± 4

PC I/O map

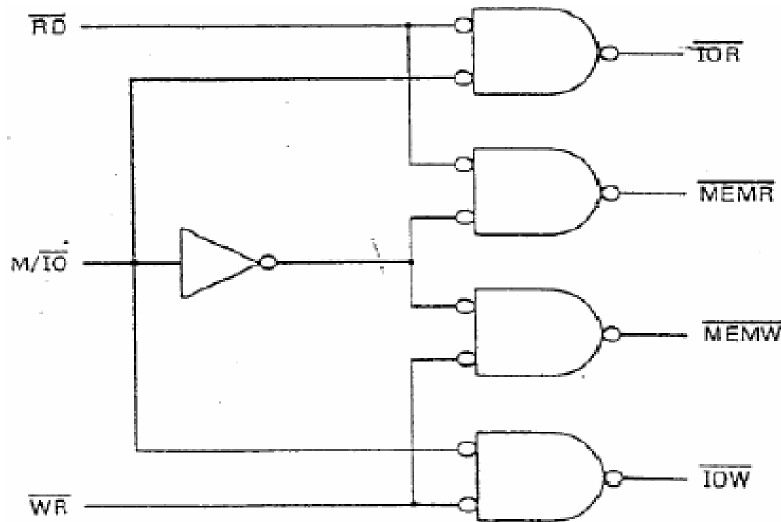


I/O bus cycle

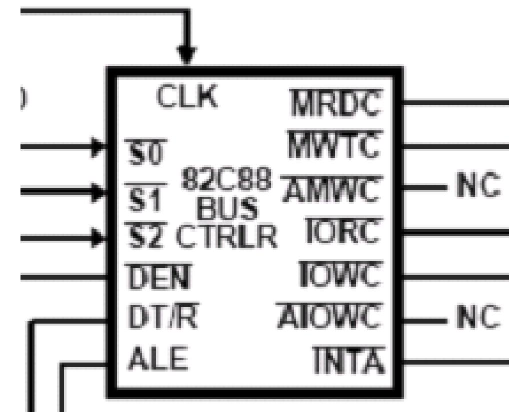


I/O and memory control signals

8086 – min. mode

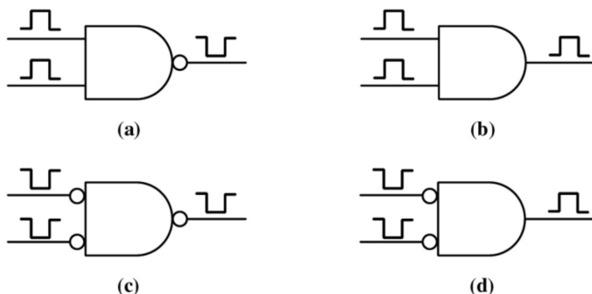


8086 – max. mode



$\overline{\text{S2}}$	$\overline{\text{S1}}$	$\overline{\text{S0}}$	PROCESSOR STATE	82C88 COMMAND
0	0	0	Interrupt Acknowledge	$\overline{\text{INTA}}$
0	0	1	Read I/O Port	$\overline{\text{IORC}}$
0	1	0	Write I/O Port	$\overline{\text{IOWC}}$, $\overline{\text{AIOWC}}$
0	1	1	Halt	None
1	0	0	Code Access	$\overline{\text{MRDC}}$
1	0	1	Read Memory	$\overline{\text{MRDC}}$
1	1	0	Write Memory	$\overline{\text{MWTC}}$, $\overline{\text{AMWC}}$
1	1	1	Passive	None

Logic gates:

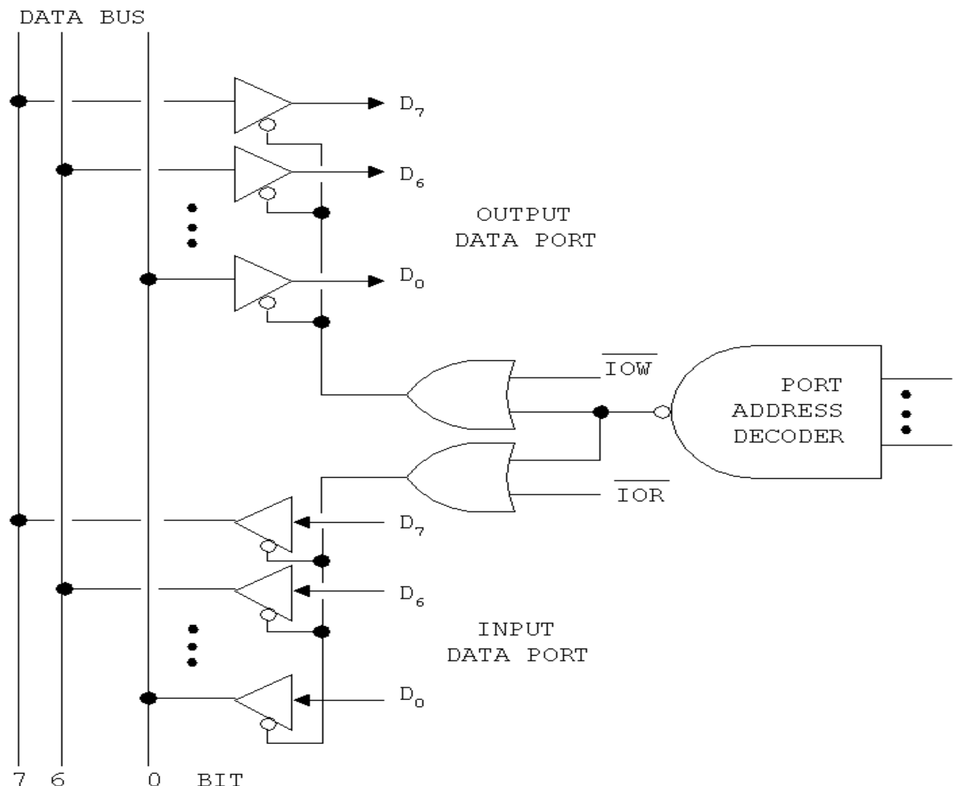


(a) NAND, (b) AND, **(c) OR**, and (d) NOR gates

How is an I/O transfer performed

1. Selection of the I/O port through the I/O address (by an address decoder)
2. Asserting #IORC or #IOWC (low)

- 2 different ports can have the same address if they have different types (input or output) !



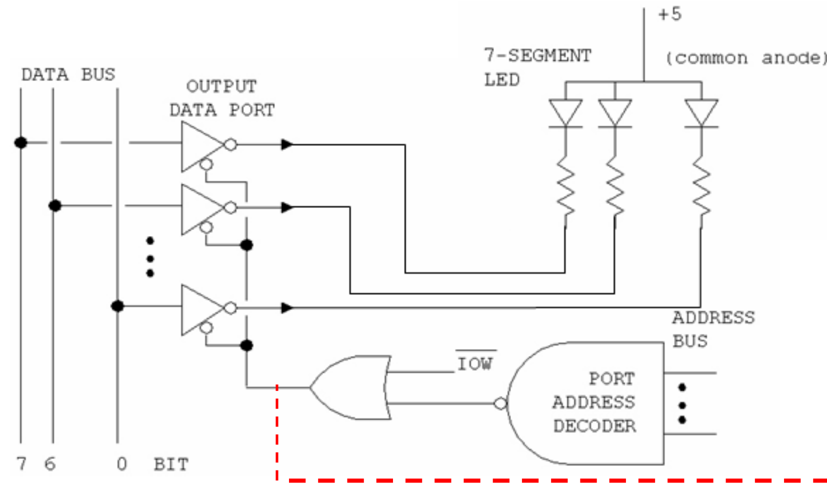
Example:

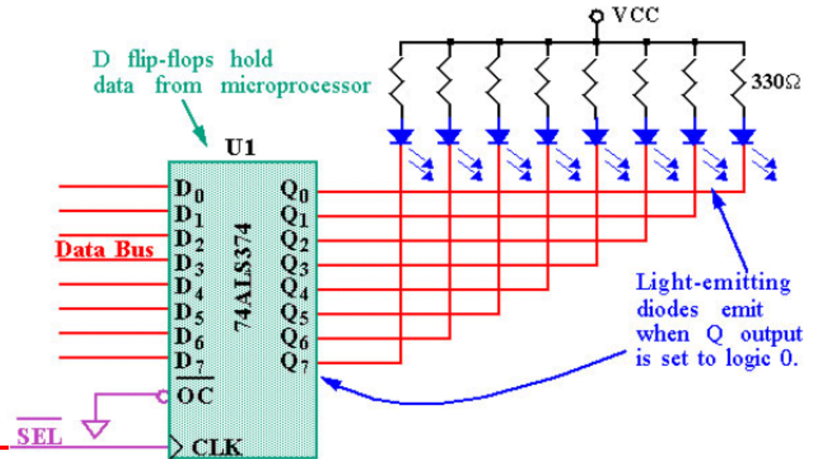
An octal input port & an octal output port decoded at the same address.

Each port (3-state buffer) is activated by the IOR/IOW control signal in conjunction with the address decoder output.

Basic output port – data persistency

Example: driving 8 LEDs with continuous current

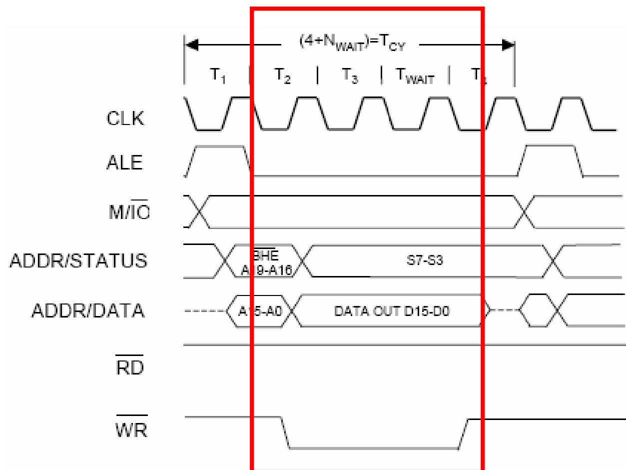


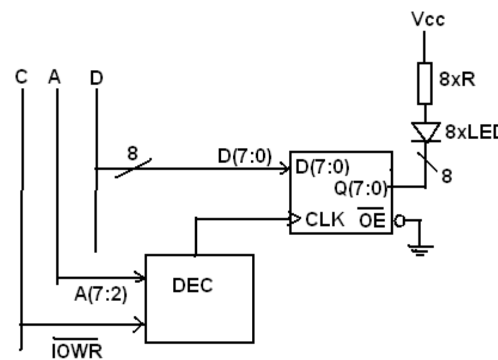


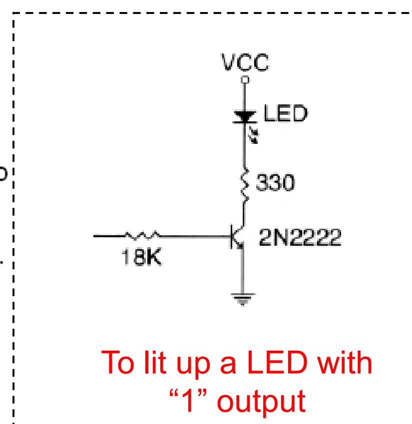
Not recommended: - the LEDs will be driven for about 3 clock cycles and will not lit up

Solution: to latch the output in a latch or flip-flop register

- The LEDs will lit up as long we do not change the content of the register







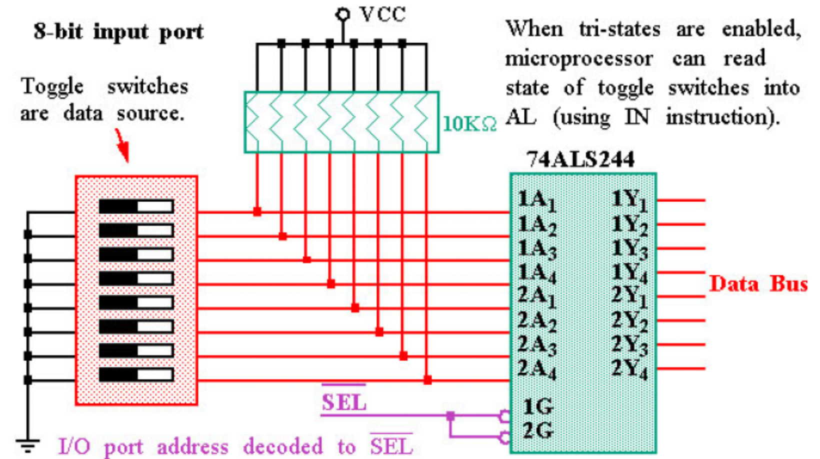
Basic input port – data bus decoupling

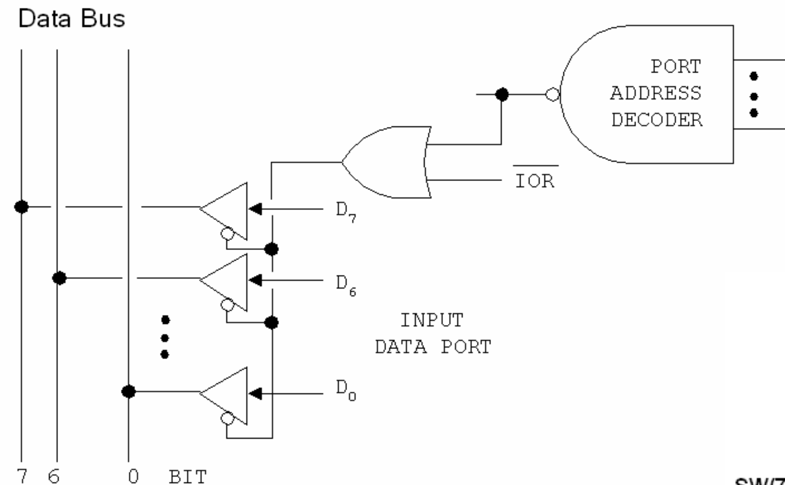
On the data system bus there are many devices (ports) connected).

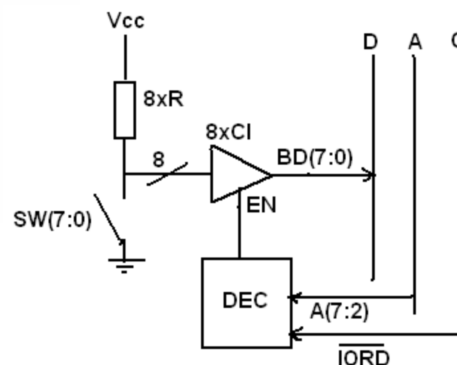
Only one device at a time (in one bus cycle) can have its outputs connected!

⇒ Output pins of each input device (port / register) should be 3-stated (enabled in the bus cycle when they are active and disabled – 3rd state – when are not active) !!!

⇒ Enabling the output can be done trough the output of the address decoder and/or IOR control signal







Inputs		Outputs
Enable A, Enable B	A,B	YA,YB
L	L	L
L	H	H
H	X	Z

I/O port decoding

Using gates:

Address: EFXXh = 11101111XXXXXXXX

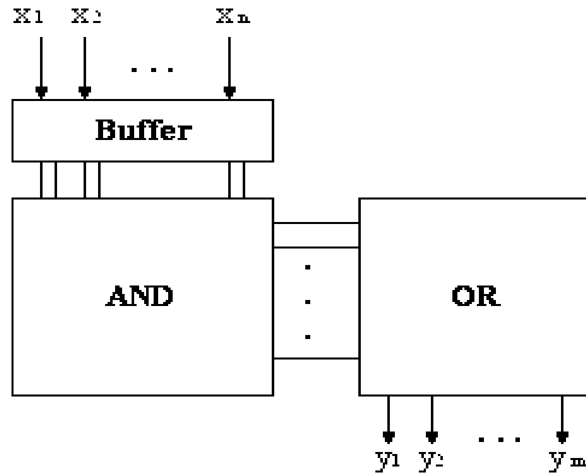
Using LS 138 (3 to 8 line decoder):

The 3-to-8 Line Decoder (74LS138)

Inputs						Output							
Enable			Select										
G2A	G2B	G1	C	B	A	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	$\overline{7}$
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

I/O port decoding

Using PLA, PAL ….

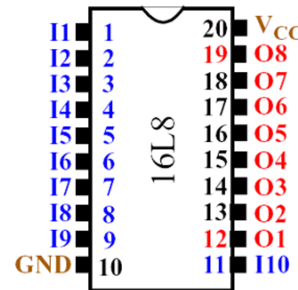


PLA - AND & OR matrices are programmable
PAL - AND matrix is programmable, OR matrix is fix

AMD 16L8 PAL decoder.

It has 10 fixed inputs (Pins 1-9, 11), two fixed outputs (Pins 12 and 19) and 6 pins that can be either (Pins 13-18).

Programmed to decode address lines A_{19} - A_{13} onto 8 outputs.

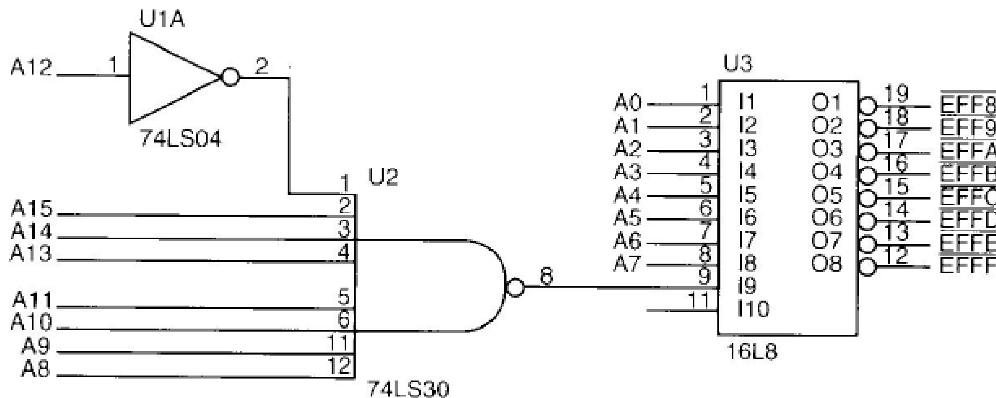


;pins	1	2	3	4	5	6	7	8	9	10
	A19	A18	A17	A16	A15	A14	A13	NC	NC	GND
;pins	11	12	13	14	15	16	17	18	19	20
	NC	O8	O7	O6	O5	O4	O3	O2	O1	VCC

Equations:

/O1 = A19 * A18 * A17 * A16 * /A15 * /A14 * /A13
/O2 = A19 * A18 * A17 * A16 * /A15 * /A14 * A13
/O3 = A19 * A18 * A17 * A16 * /A15 * A14 * /A13
/O4 = A19 * A18 * A17 * A16 * /A15 * A14 * A13
/O5 = A19 * A18 * A17 * A16 * A15 * /A14 * /A13
/O6 = A19 * A18 * A17 * A16 * A15 * /A14 * A13
/O7 = A19 * A18 * A17 * A16 * A15 * A14 * /A13
/O8 = A19 * A18 * A17 * A16 * A15 * A14 * A13

Example: address decoder using logic gates and PAL



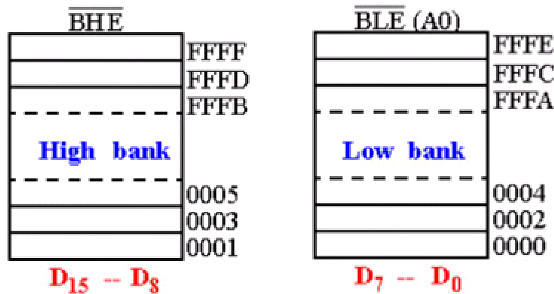
Homework:

-write the /O1 .. /O8 programming functions

-explain why the base address is EFF8

I/O port decoding

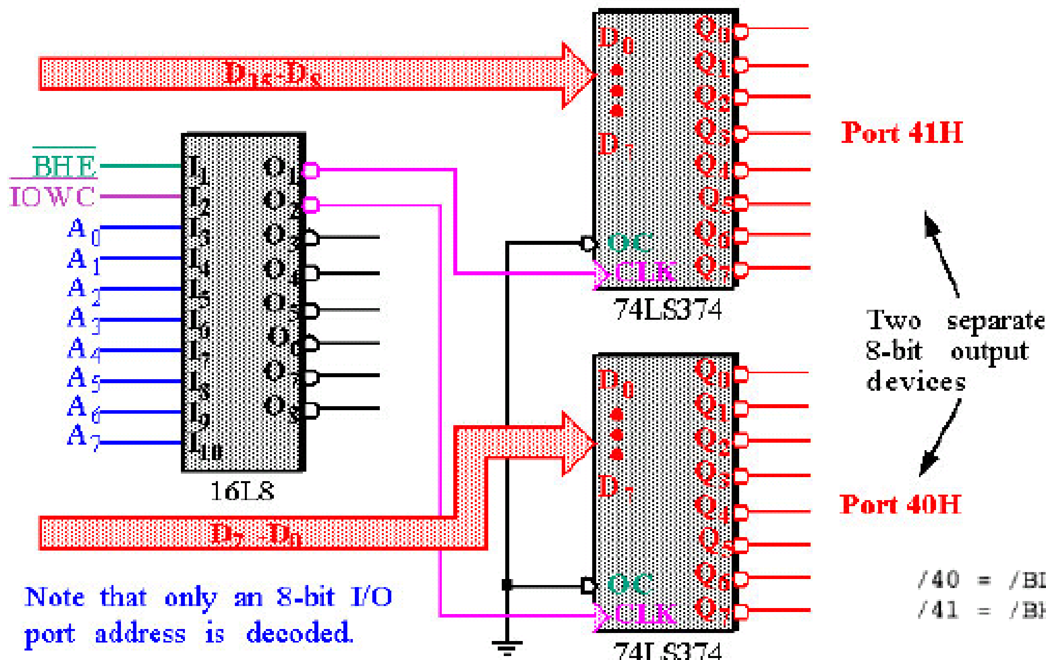
I/O space for 16 bit x86 family



$\overline{BHE}$	A_0	Characteristics
0	0	Whole word
0	1	Upper byte from/to odd address
1	0	Lower byte from/to even address
1	1	None

16 bit I/O port (very rare: ADC and DAC, video, disk interfaces)

Example: 16 bit output port




```
/40 = /BLE * /IOWC * /A7 * A6 * /A5 * /A4 * /A3 * /A2 * /A1
/41 = /BHE * /IOWC * /A7 * A6 * /A5 * /A4 * /A3 * /A2 * /A1
```

Rules:

- 16 bit write needs separate BHE and BLE (A0) strobes
- 16 bit read does not (explain why ?)

Homework: Design a 16 bit input interface using 2x LS244 buffers

Additional reading:

Barry B. Brey, The Intel Microprocessors: 8086/8088, 80186,80286, 80386 and 80486. Architecture, Programming, and Interfacing, 4-th edition, Prentice Hall, 1994, pp. 362-375.

I/O transfer techniques

1. Transfer trough the CPU (internal registers ~ buffers)

Programmed I/O (CPU polls peripherals to check if I/O is needed)

- unconditioned (peripheral status is NOT checked)
- conditioned (peripheral status is checked)

Interrupt based I/O

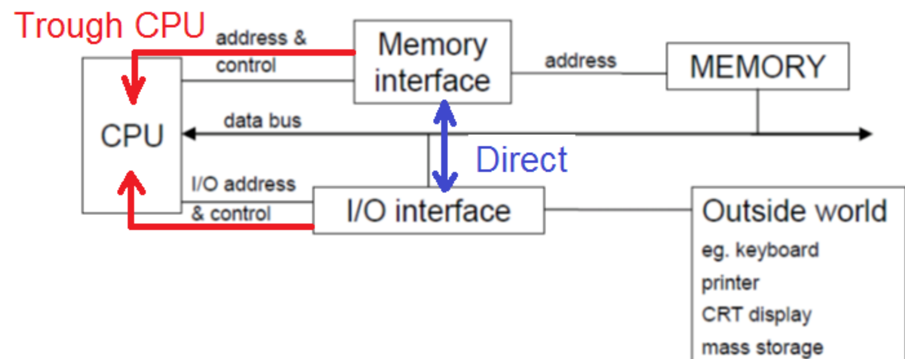
- peripheral sends an interrupt request to CPU for I/O transfers

2. Direct transfer - peripheral writes directly to memory

Direct memory access (DMA) using a DMA controller

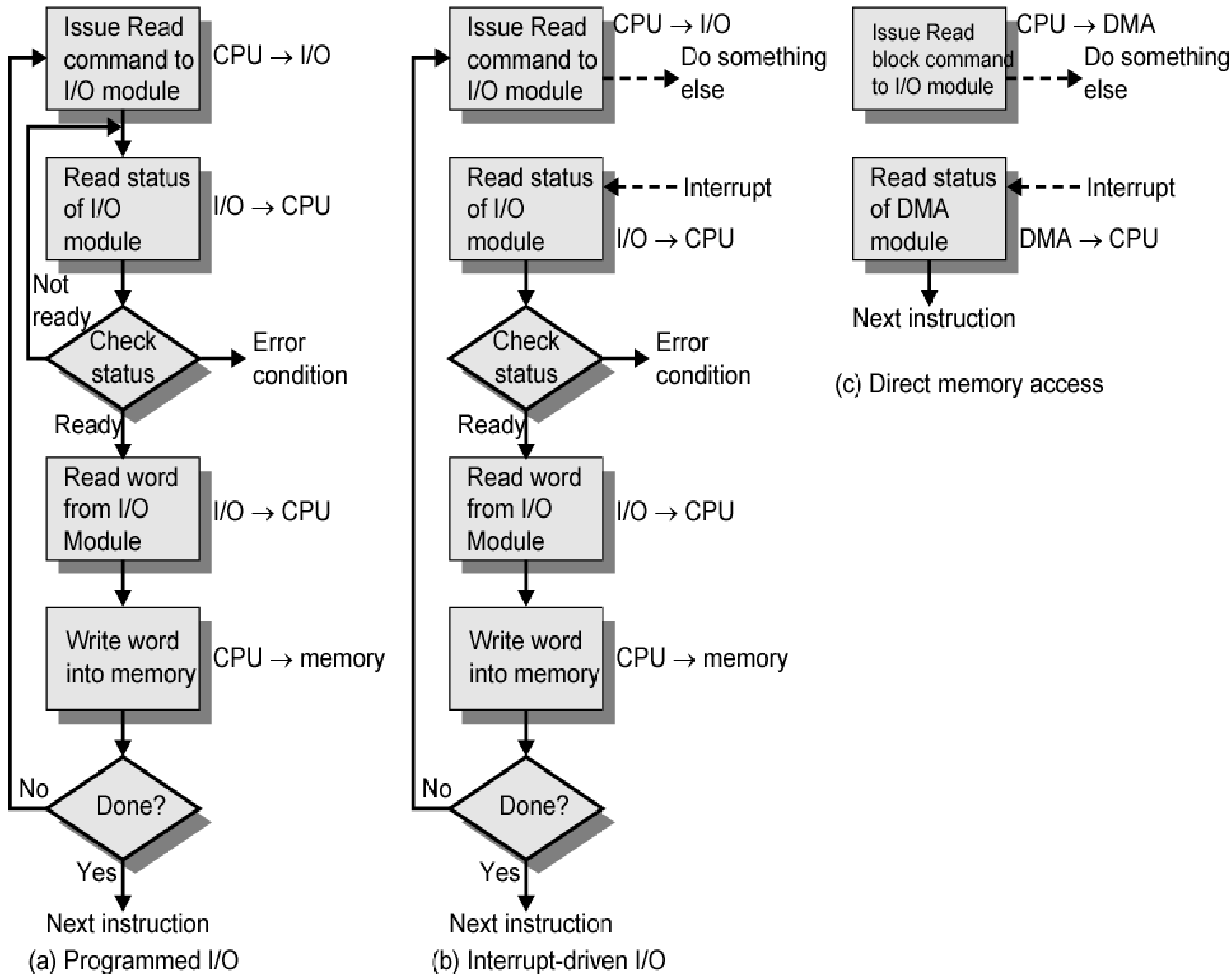
Using a dedicated I/O processor

Using a multiplexer channel



I/O transfer techniques

Example: I/O $\Rightarrow$ memory transfer



Interrupt system (x86)

References

[1] CompE475_chapter12.pdf, pp.1-12

[2] ch13-interrupt.pdf, pp.1-20

[3] Barry B. Brey, The Intel Microprocessors: 8086/8088, 80186,80286, 80386 and 80486. Architecture, Programming, and Interfacing, 4-th edition, Prentice Hall, 1994, pp. 430-450 (optional: 450-462)

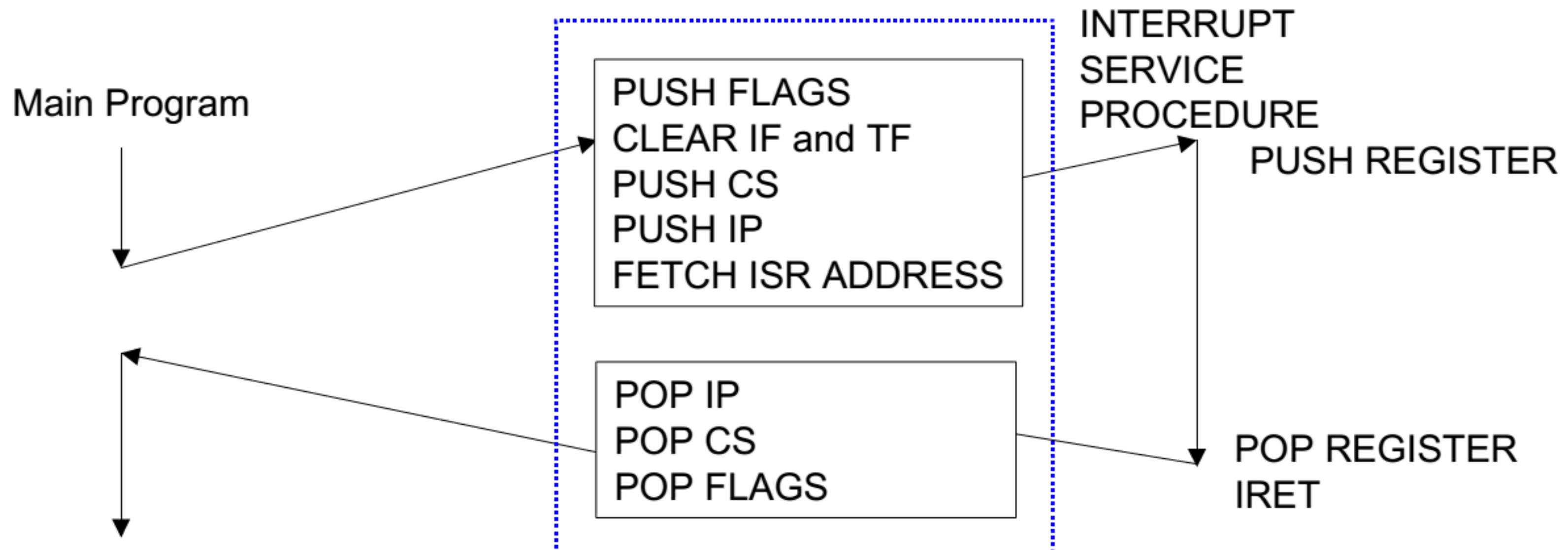
Three basic categories of interrupts:

1. **Software** interrupts: generated by an instruction: **INT** interrupt-vector
2. **Hardware** interrupts: generated by asserting the **NMI** or **INTR** pin
3. **Automatic** interrupts: generated by certain error conditions (eg. Divide by 0), or at the end of every instruction if the TRAP flag is set in the FLAGS register.

Interrupt system (x86)

Interrupt Response Sequence [2]

1. PUSH Flag Register
2. Clear IF and TF
3. PUSH CS
4. PUSH IP
5. Fetch Interrupt vector contents and place into both IP and CS to start the Interrupt Service Procedure (ISP)



Interrupt system (x86)

The **Interrupt Vector Table (IVT)** [3]

Address	Entry	Vector
	32-255 User defined	
080H	14-31 Reserved	
040H	Coprocessor error	16
03CH	Unassigned	15
038H	Page fault	14
034H	General protection	13
030H	Stack seg overrun	12
02CH	Segment not present	11
028H	Invalid task state seg	10
024H	Coproc seg overrun	9
020H	Double fault	8
01CH	Coprocessor not avail	7
018H	Undefined Opcode	6
014H	Bound	5
010H	Overflow (INTO)	4
00CH	1-byte breakpoint	3
008H	NMI pin	2
004H	Single-step	1
000H	Divide error	0

The interrupt vector table is located in the first 1024 bytes of memory at addresses 000000H through 0003FFH.

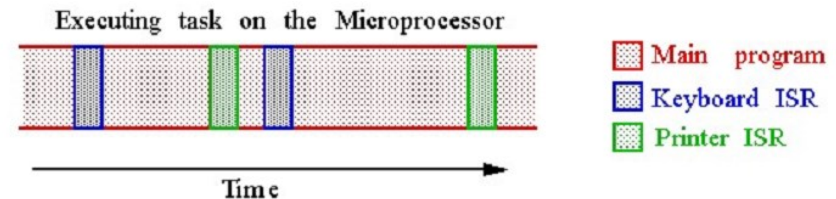
There are 256 4-byte entries (segment and offset in real mode).

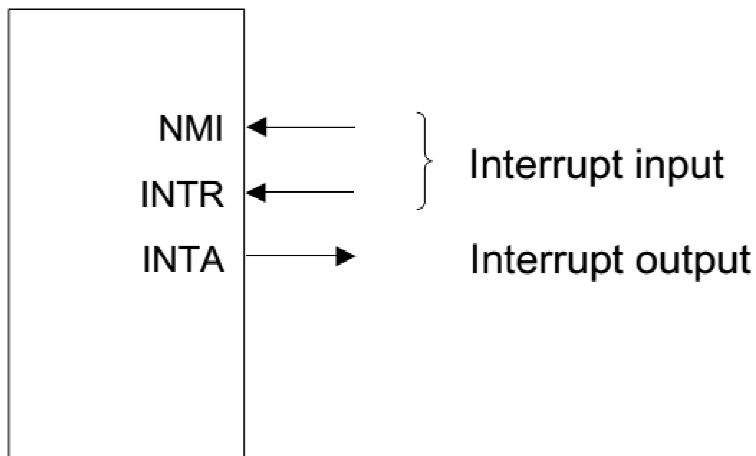
Seg high	Seg low	Offset high	Offset low
Byte 3	Byte 2	Byte 1	Byte 0

Interrupt system (x86)

Hardware Interrupts

- very efficient in handling peripheral devices, particularly I/O operations, since the processor can perform other tasks and only services the peripheral when required.





NMI is usually reserved for the most urgent type of interrupt (eg. imminent power failure of a peripheral).

- Int. no. 2
- edge triggered (0-to-1 transition)

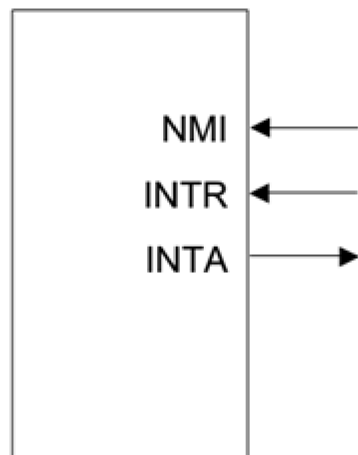
INTR is used for "normal" interrupts (where the peripheral can wait if necessary until the interrupt flag is set).

- level sensitive (must be held at logic "1"until it is acknowledged by INTA.
- INTR is automatically disabled when the μP is already servicing an INTR
- INTR is re-enabled at the end of the interrupt service routine
- Int. no. generated by an INTR is read from the data bus

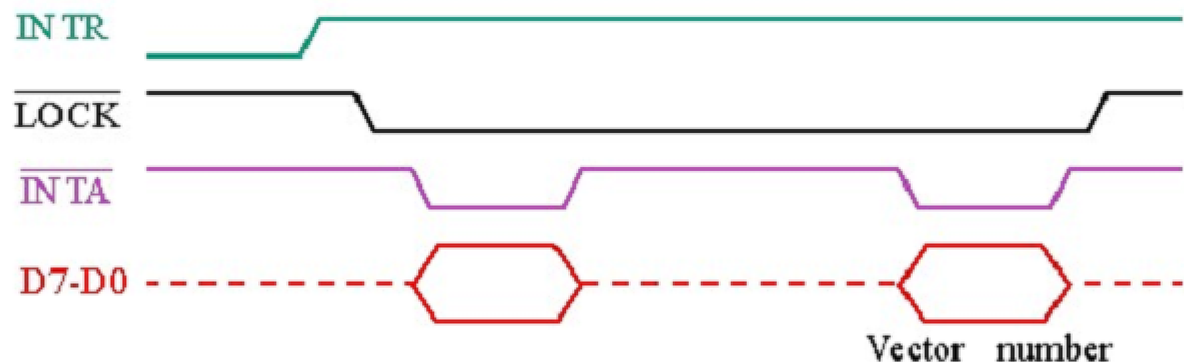
Interrupt system (x86)

Hardware Interrupts

- The INTR pin must be externally decoded to select a vector.
- Any vector is possible, but the interrupt vectors between 20H and FFH are usually used (Intel reserves vectors between 00H and 1FH).
- INTA is an output of the microprocessor to signal the external decoder to place the interrupt number on data bus connections D7-D0.

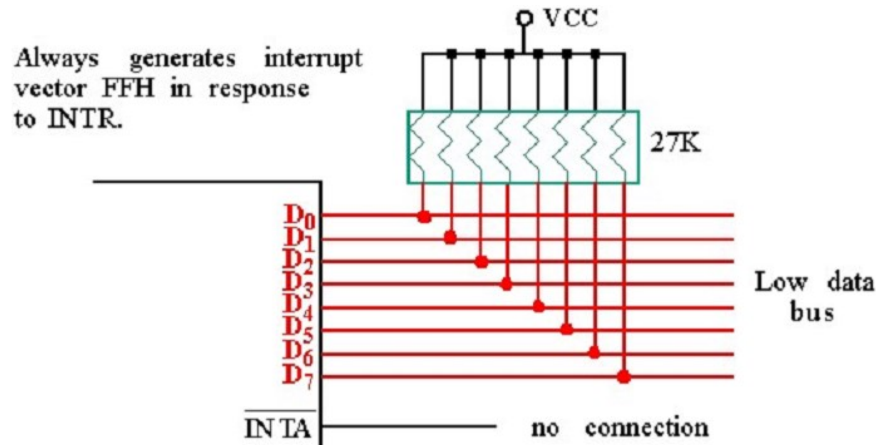


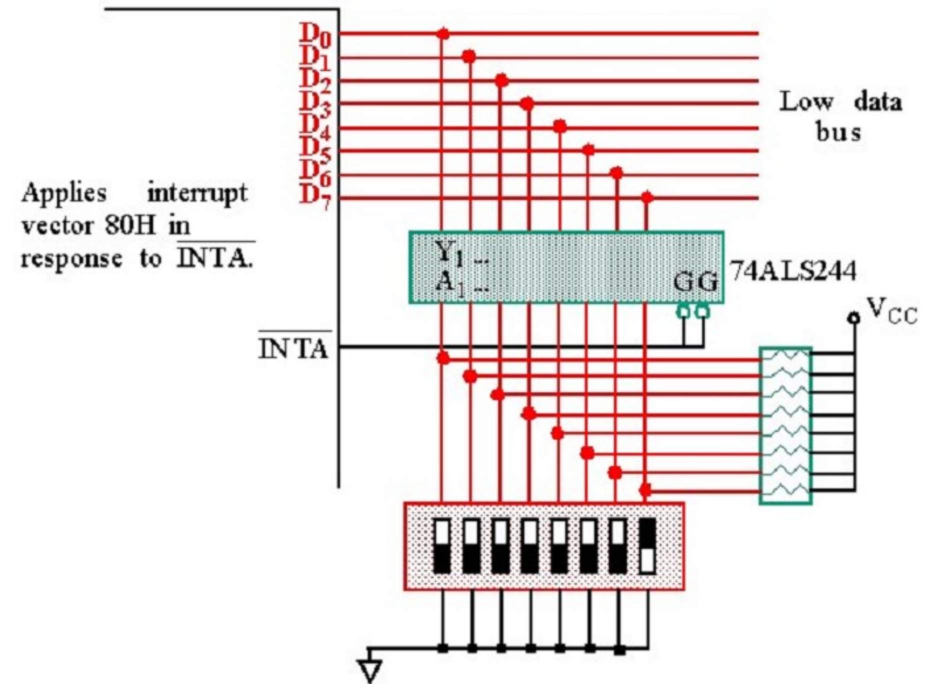
Timing diagram of the handshake (2x INTA cycles) [1]



Interrupt system (x86)

Hardware Interrupts - simple methods to generating interrupt vectors [1]





Interrupt system (x86)

Hardware Interrupts - handling more than 1 IRQ

- If several INTR are generated from different peripherals simultaneously ⇒ is necessary to decide their priority and send the INTA signal only to the highest priority peripheral.
- Two different methods can be used to establish the priority of interrupt requests from different peripherals
  1. Polling and daisy chaining
  2. Interrupt priority management hardware

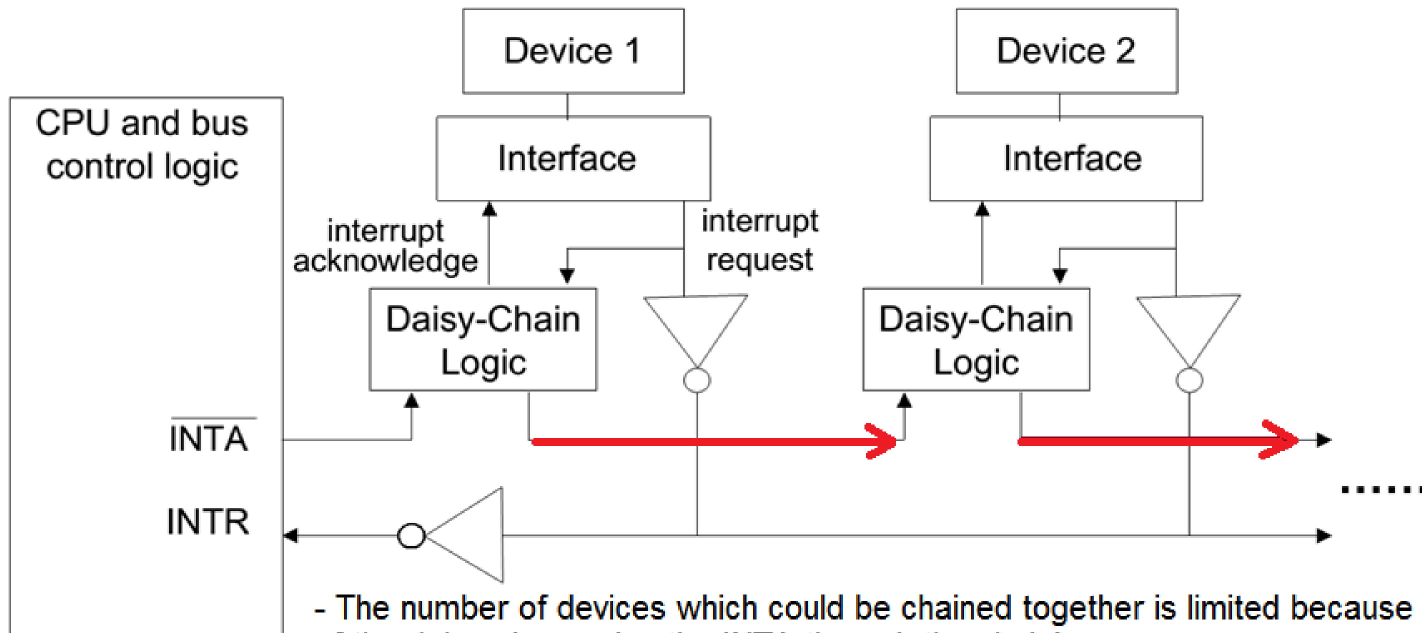
Priority Allocation by Polling and Daisy-Chaining

- Polling = asking each peripheral, in a predetermined order, whether it needs attention from the microprocessor. The first peripheral which responds "yes" is served by the appropriate routine.
- Daisy-chaining is a method of implementing the polling scheme by hardware

Interrupt system (x86)

Hardware Interrupts - Priority Allocation by Daisy-Chaining [2]

- The INTA signal passes from one peripheral to the next only if the peripheral is not requesting an interrupt.
- The first peripheral in the daisy-chain has the highest priority and the last peripheral has the lowest priority (fixed scheme)
- Daisy chaining may be combined with software polling to determine which routine is needed by the peripheral



- The number of devices which could be chained together is limited because of the delays in passing the INTA through the chain!
- The microprocessor expects the interrupt number to be placed on the data bus within a certain time after the INTA is sent out.

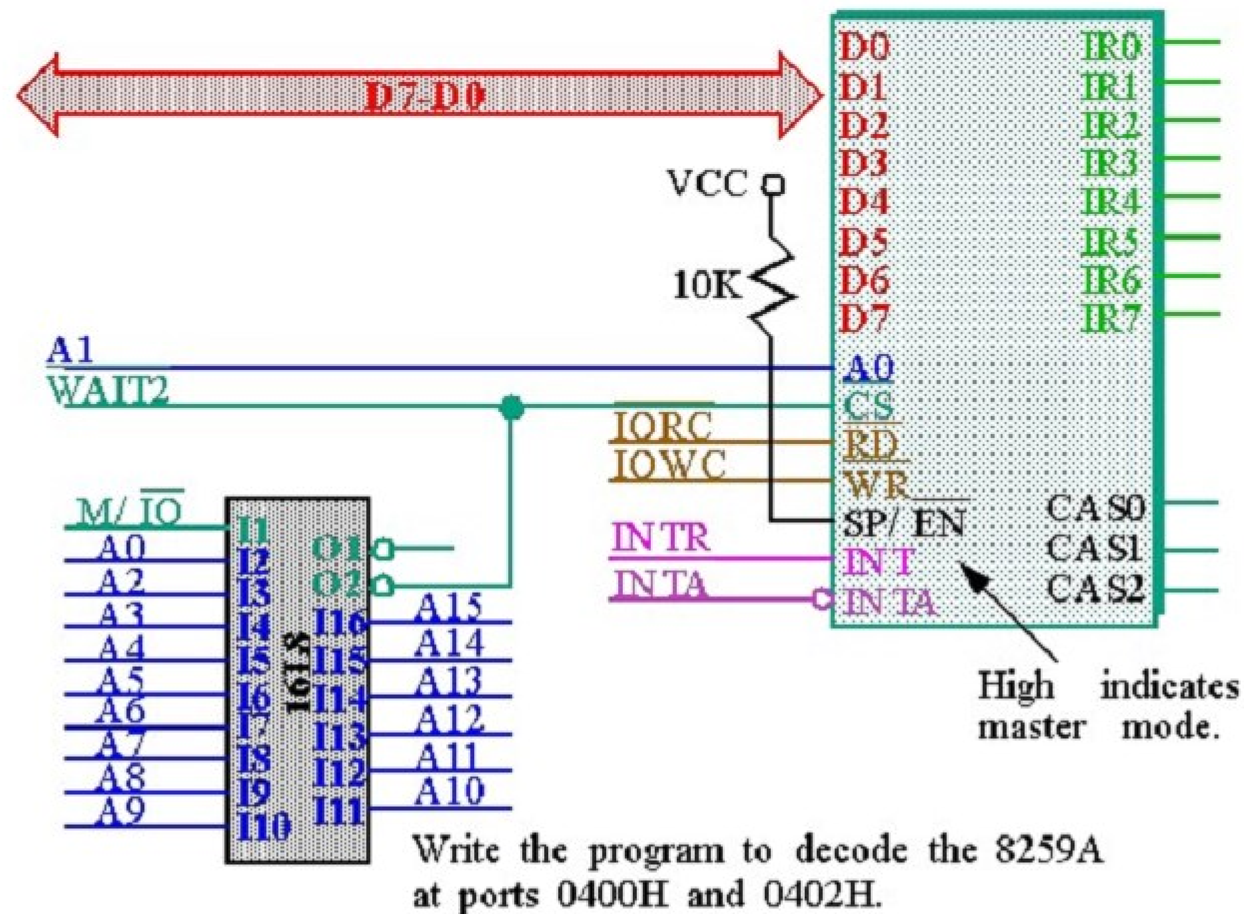
Interrupt system (x86)

Hardware Interrupts - Interrupt priority management hardware

A Programmable interrupt controller(8259A) is usually used in practical systems to determine the priority of interrupts

Example: a single 8259A connected in the 8086 [1]

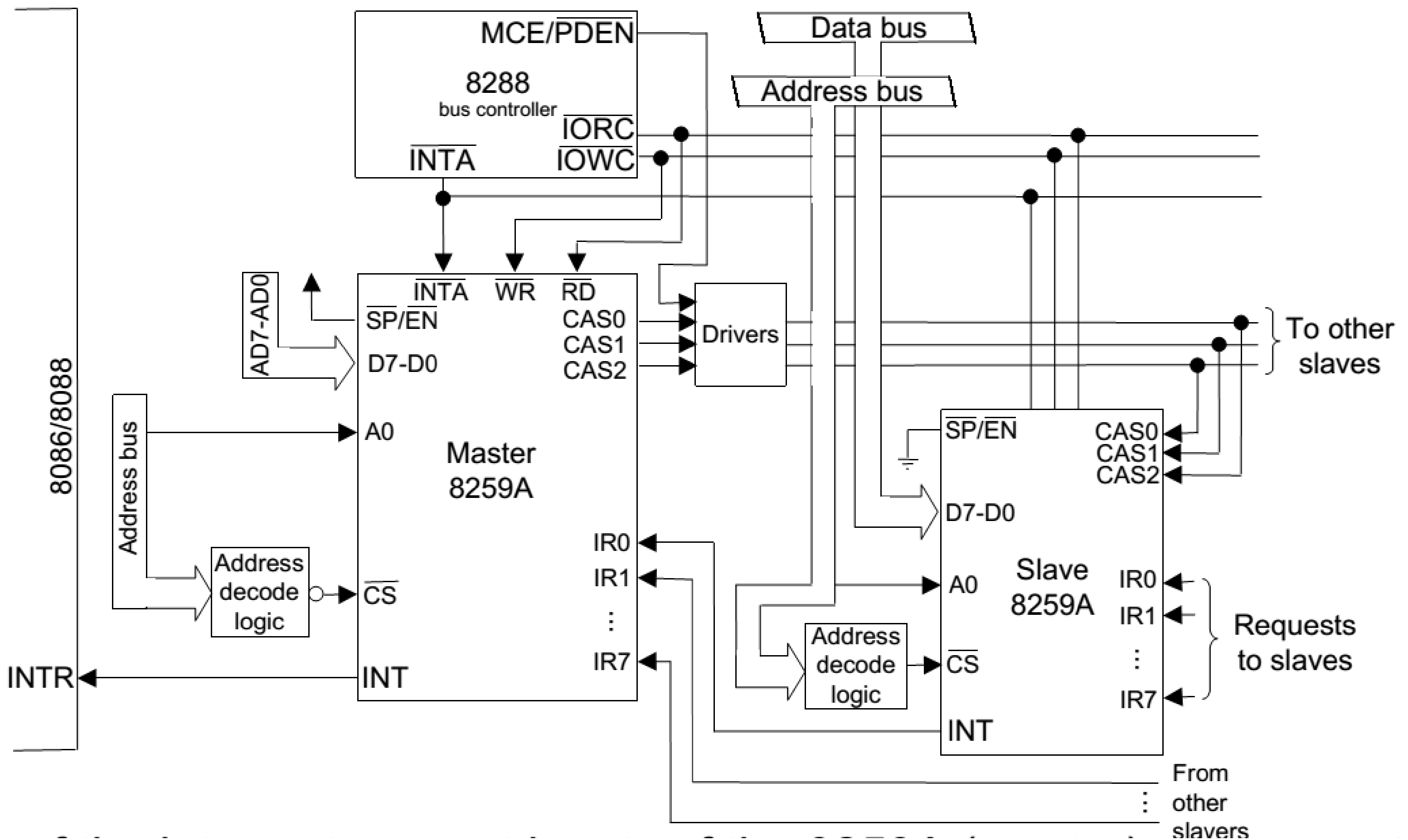
- /CS must be decoded. Other connections are direct to micro
- Programmable IR priority (defult: IR0 highest … IR7 lowest)



Interrupt system (x86)

Hardware Interrupts - Interrupt priority management hardware

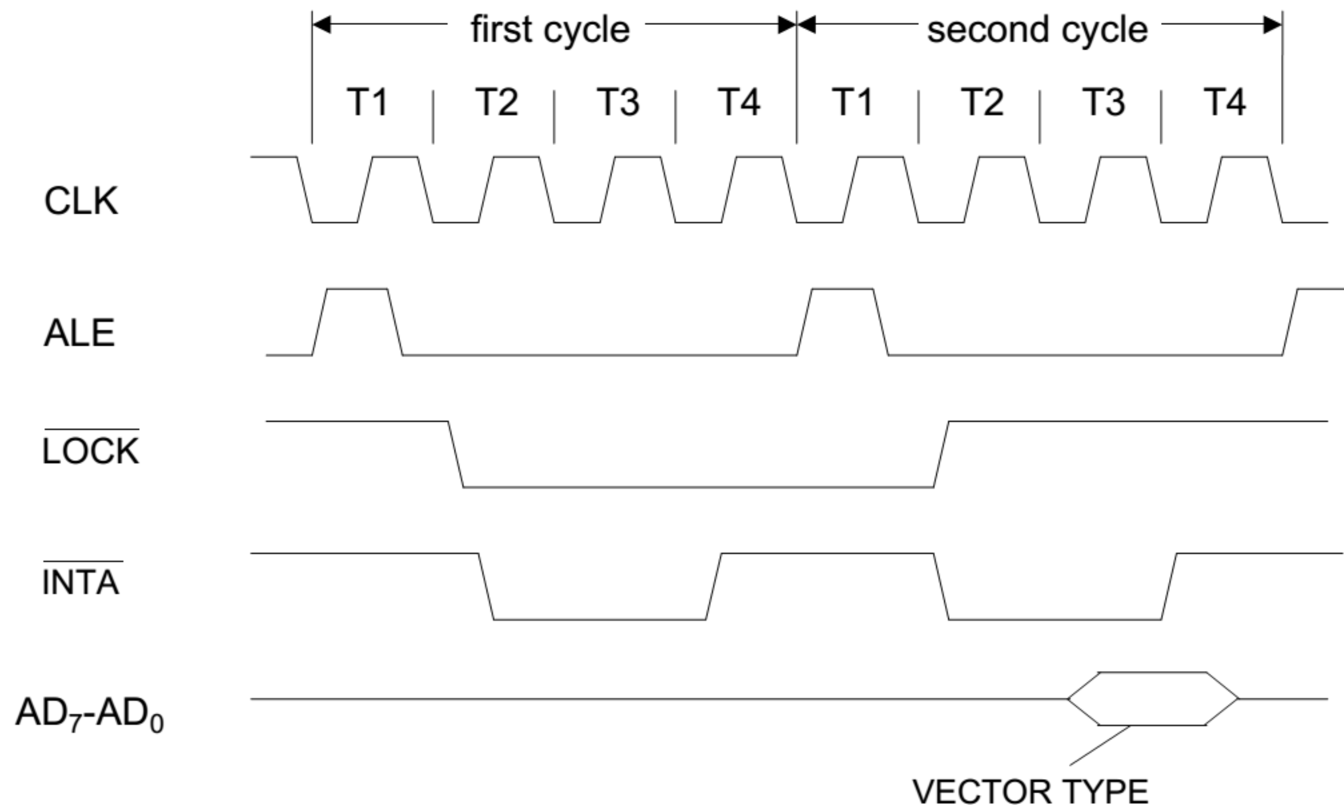
The 8259A may be cascaded (one master 8259A and eight slave 8259A) to provide up to 64 interrupt lines).



Interrupt system (x86)

Hardware Interrupts - Interrupt priority management hardware

Interrupt vector number decoding: timing diagram of the handshake (2x INTA cycles)

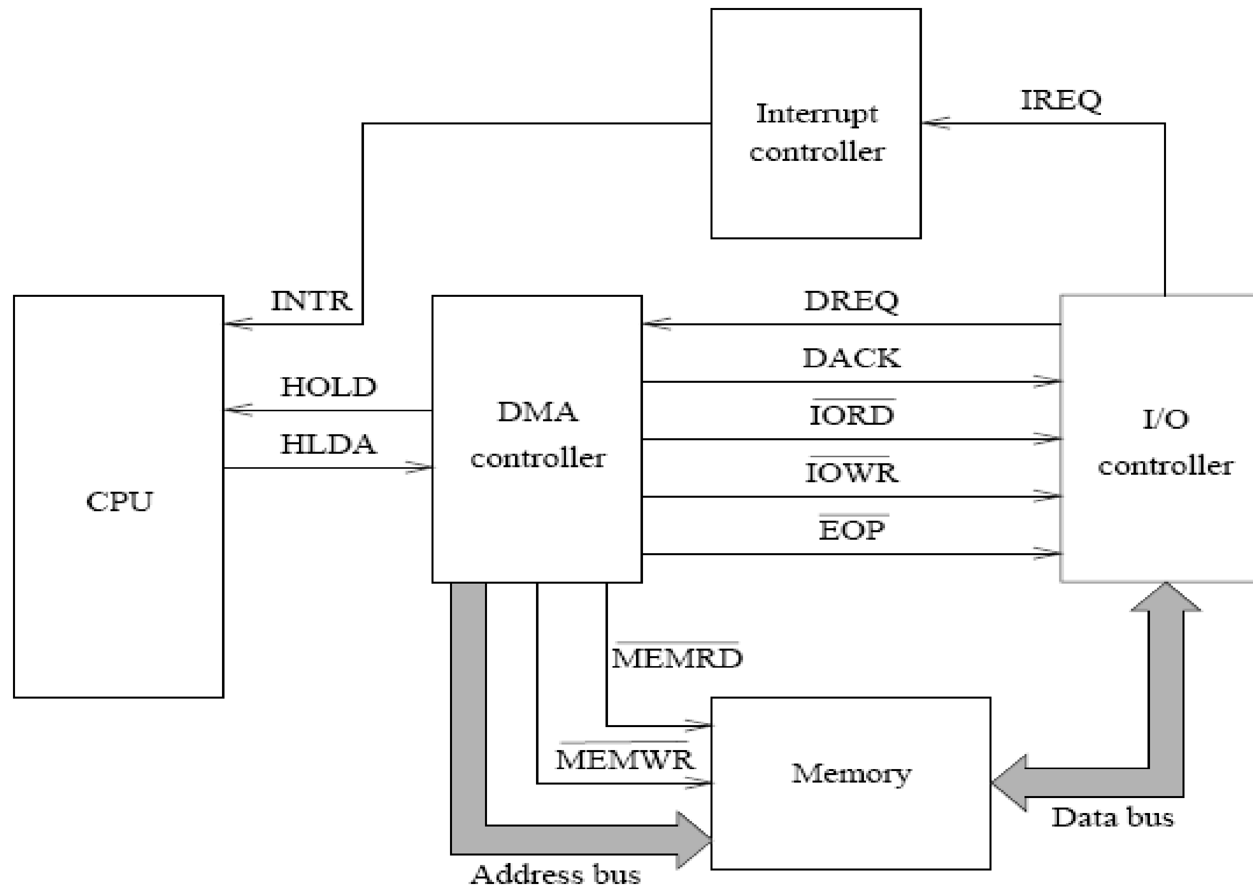


Direct memory access (DMA)

- **Direct memory access (DMA)** is a process in which an external device takes over the control of system bus from the CPU.
- DMA is for **high-speed data transfer** from/to mass storage peripherals, e.g. hard-disk drive,CD-ROM, and sometimes video controllers.
- The basic idea of **DMA** is to transfer *blocks of data* directly between memory and peripherals: the data don't go through the microprocessor but the system data bus is occupied.
- "Normal" I/O transfer of one data byte takes up to 29 clock cycles. The DMA transfer requires only 5 clock cycles.
- Nowadays, DMA can transfer data as fast as 60 MB per second or more. The transfer rate is limited by the speed of memory and peripheral devices.

Direct memory access (DMA)

DMA transfer – general schematics



Direct memory access (DMA)

Basic process of DMA: 8088/8086 in minimum mode

The HOLD and HLDA pins are used to receive and acknowledge the hold request respectively.

Normally the CPU has full control of the system bus. In a DMA operation, the DMA controller takes over the system bus control temporarily.

Sequence of events of a typical DMA process

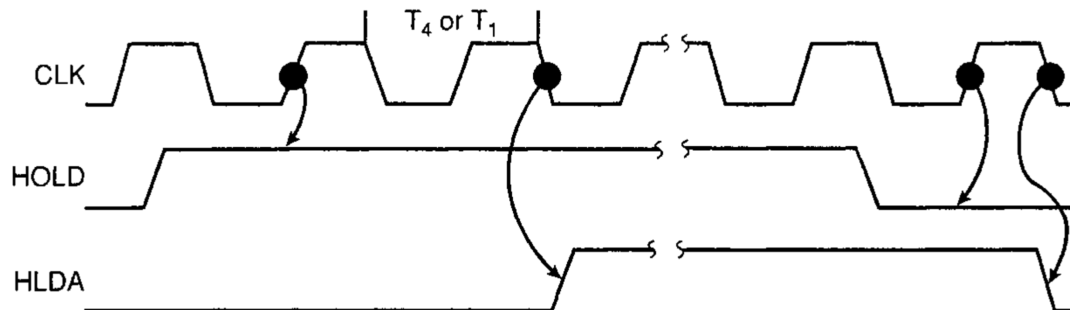
1) DMA controller asserts the request on the HOLD pin

2) 8086 completes its current bus cycle and enters into a HOLD state

3) 8086 grants the right of bus control by asserting a grant signal via the HOLDA pin. 8086 pins (Address, Data, C-trol pins ⇒ 3-rd state)

4) DMA operation starts

5) Upon completion of the DMA operation, the DMA controller asserts low the HOLD pin again to relinquish bus control.



Direct memory access (DMA)

Basic process of DMA - **8088/8086 in** maximum mode

The RQ/GT1 and RQ/GT0 pins are used to issue DMA request and receive acknowledge signals.

Sequence of events of a typical DMA process

1) DMA controller asserts one of the request pins, e.g. RQ/GT1 or RQ/GT0 (RQ/GT0 has higher priority)

2) 8086 completes its current bus cycle and enters into a HOLD state

3) 8086 grants the right of bus control by asserting a grant signal via the same pin as the request signal.

4) DMA operation starts

5) Upon completion of the DMA operation, the DMA controller asserts the request/grant pin again to relinquish bus control.

Memory

8086 and bus control logic

HOLD
HLDA

Data bus

Address bus

Control bus

I/O device

Interface

DMA controller

1 Interface is ready to receive data, DMA request is made

2 Bus request is made

3 Bus grant is returned

4 DMA device places address on address bus

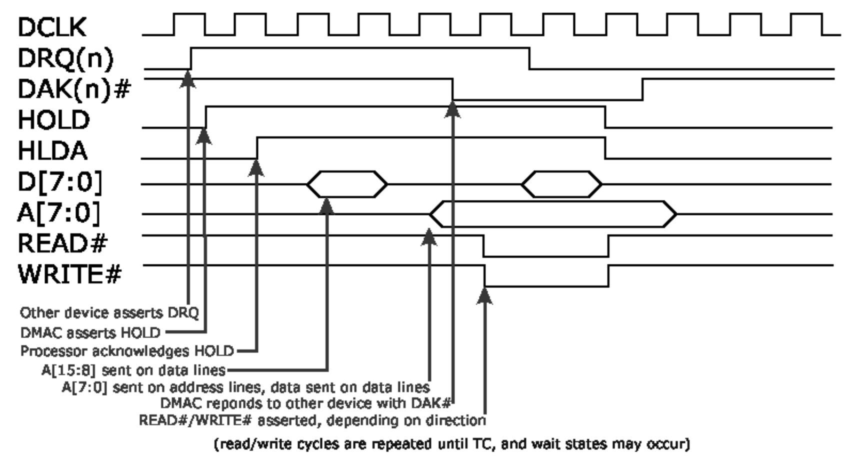
5 DMA request is acknowledged

6 Memory places data on data bus

7 Interface latches data

8 Bus request is dropped and control is returned to the 8086

9 Bus grant is dropped by the 8086



Direct memory access (DMA)

DMA transfer types

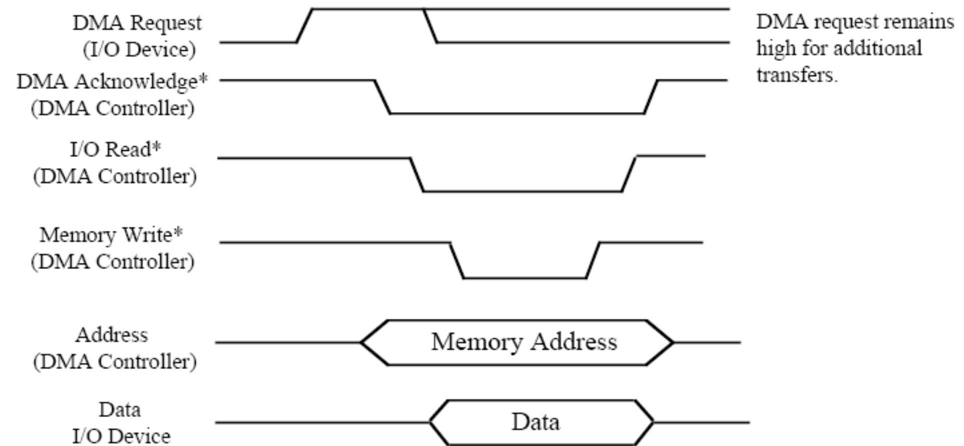
Fly-by DMA Transfer

-Data don’t pass through the DMA controller

-1 bys-cycle / transfer

- Mem ↔ I/O

- Simultaneous control signals

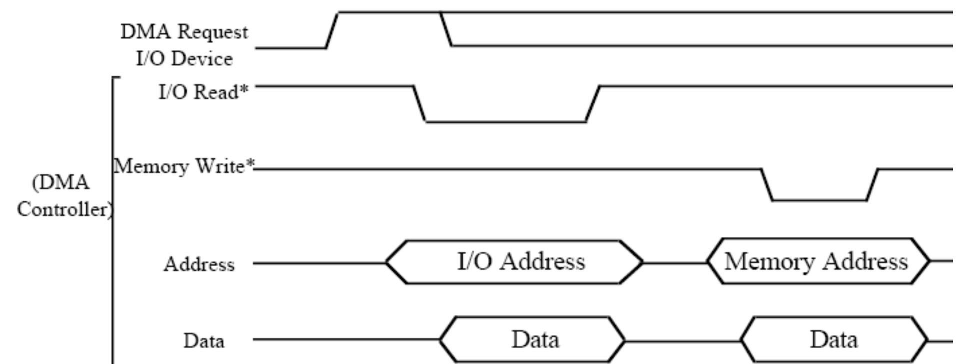


Flow-through DMA Transfer

- Data pass through controller

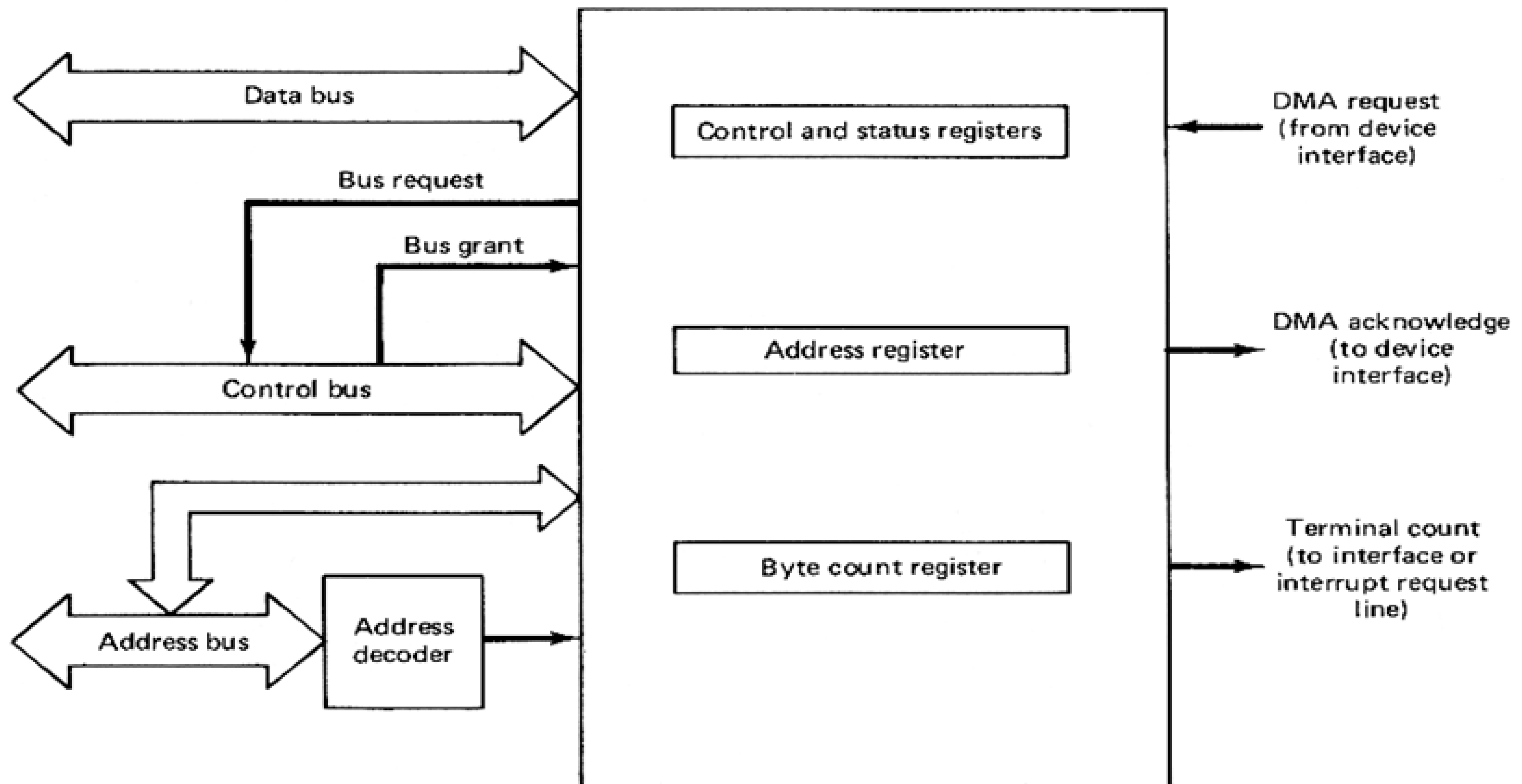
- Fetch-and-Deposit DMA Transfer: 2 cycles/transfer

- Mem ↔ Mem, I/O ↔ I/O, Mem ↔ I/O



Direct memory access (DMA)

General organization of the DMA controller

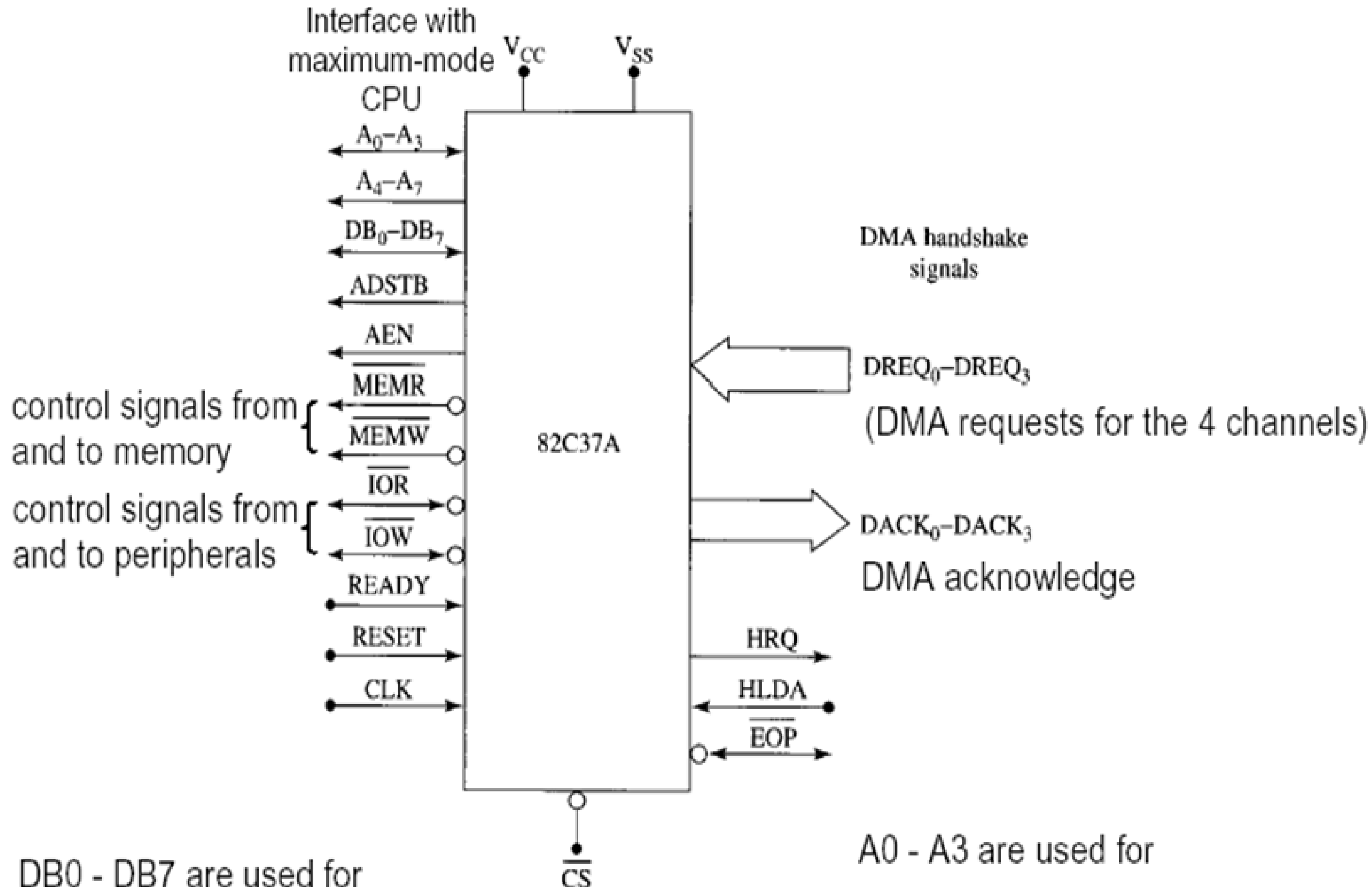


- Nowadays is part of the system controller chipset
- DMA controller commonly used with 8088/8086 is the **8237 programmable device**

Direct memory access (DMA)

The 8237 DMA controller

4-channel device (each addressing a 64 K bytes section of memory)



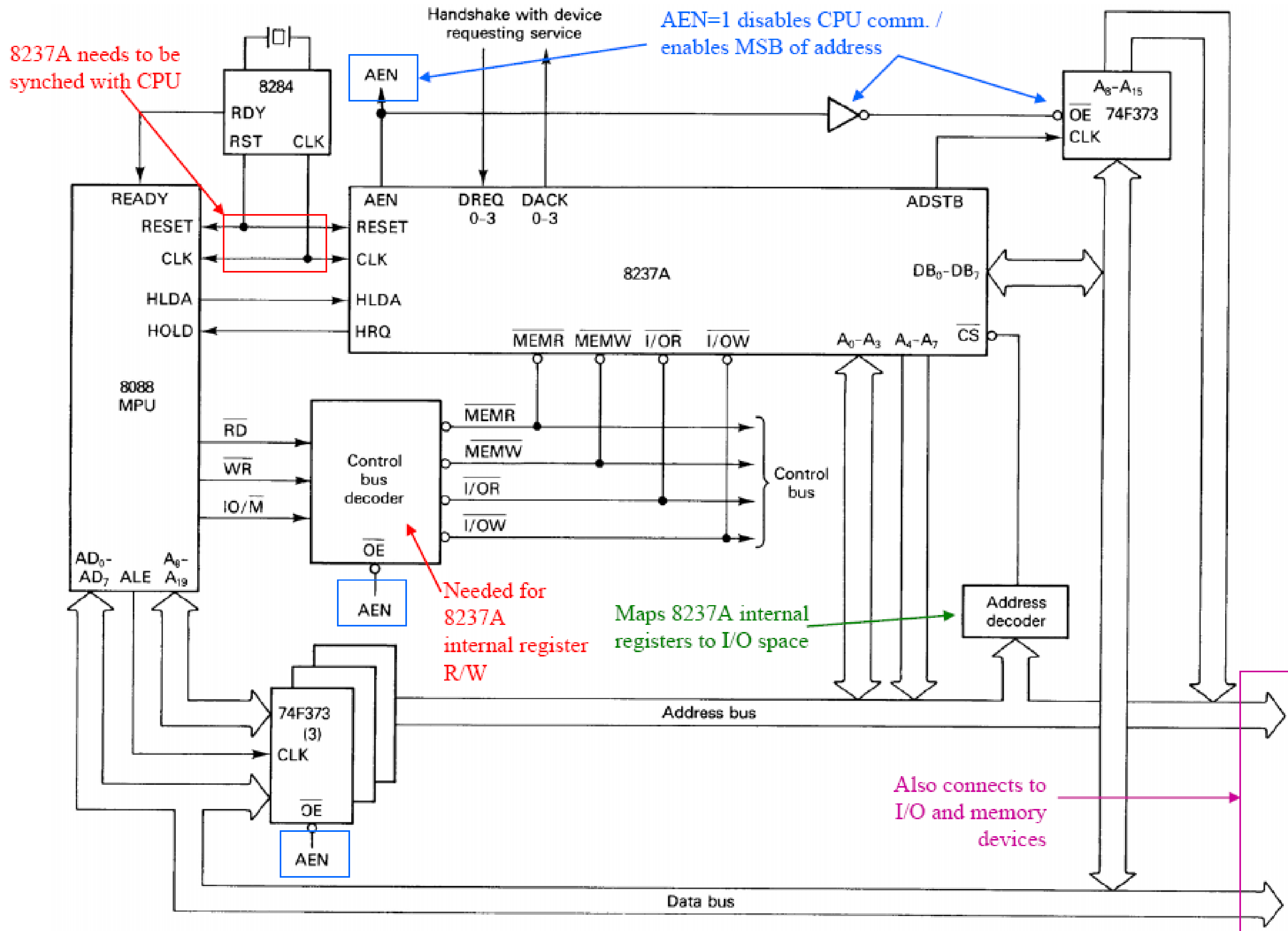
DB0 - DB7 are used for

1) transfer of data
2) 8237 programming

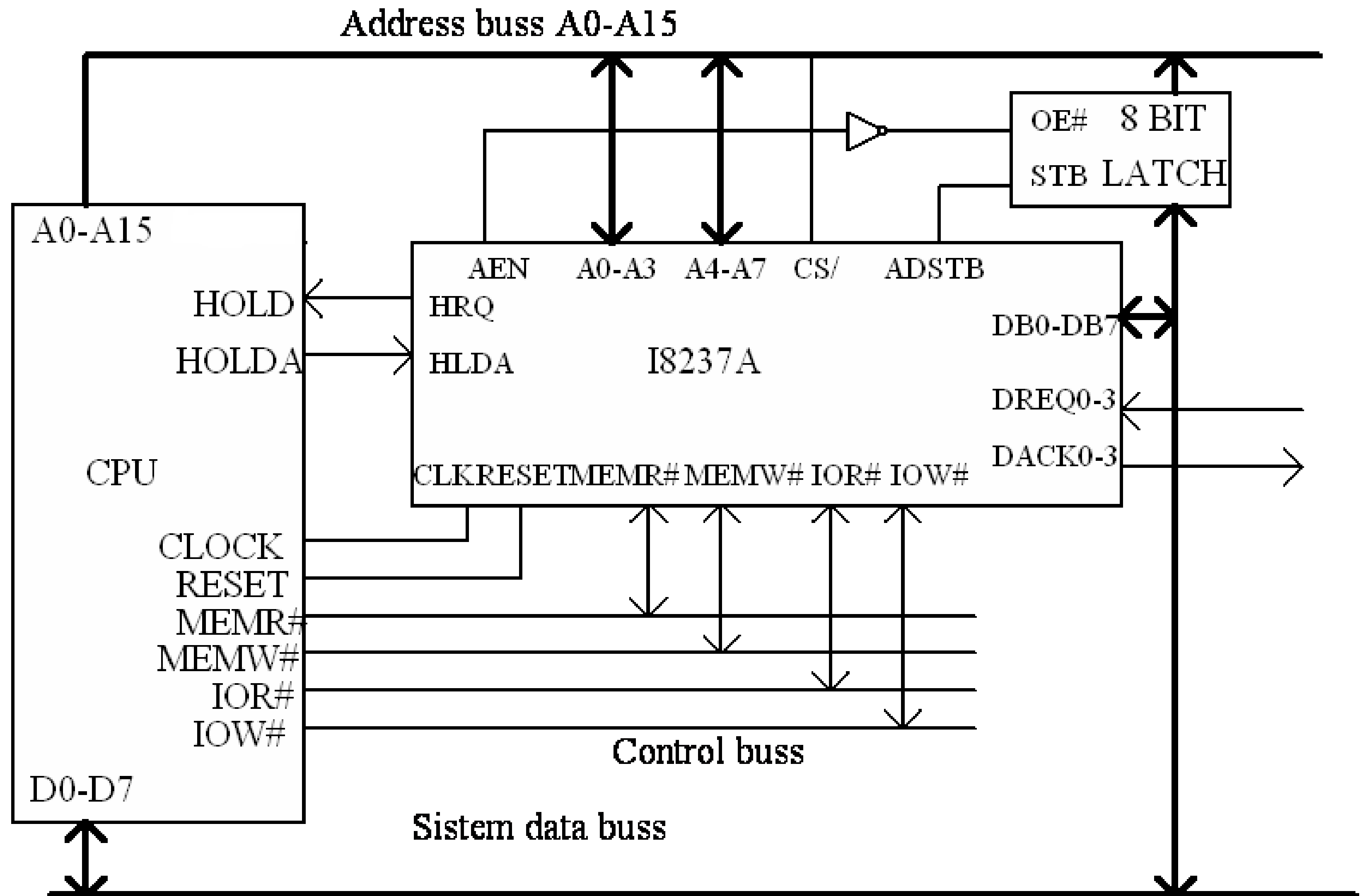
A0 - A3 are used for

1) accessing 8237 internal ports
2) carrying memory address in DMA read and write operations

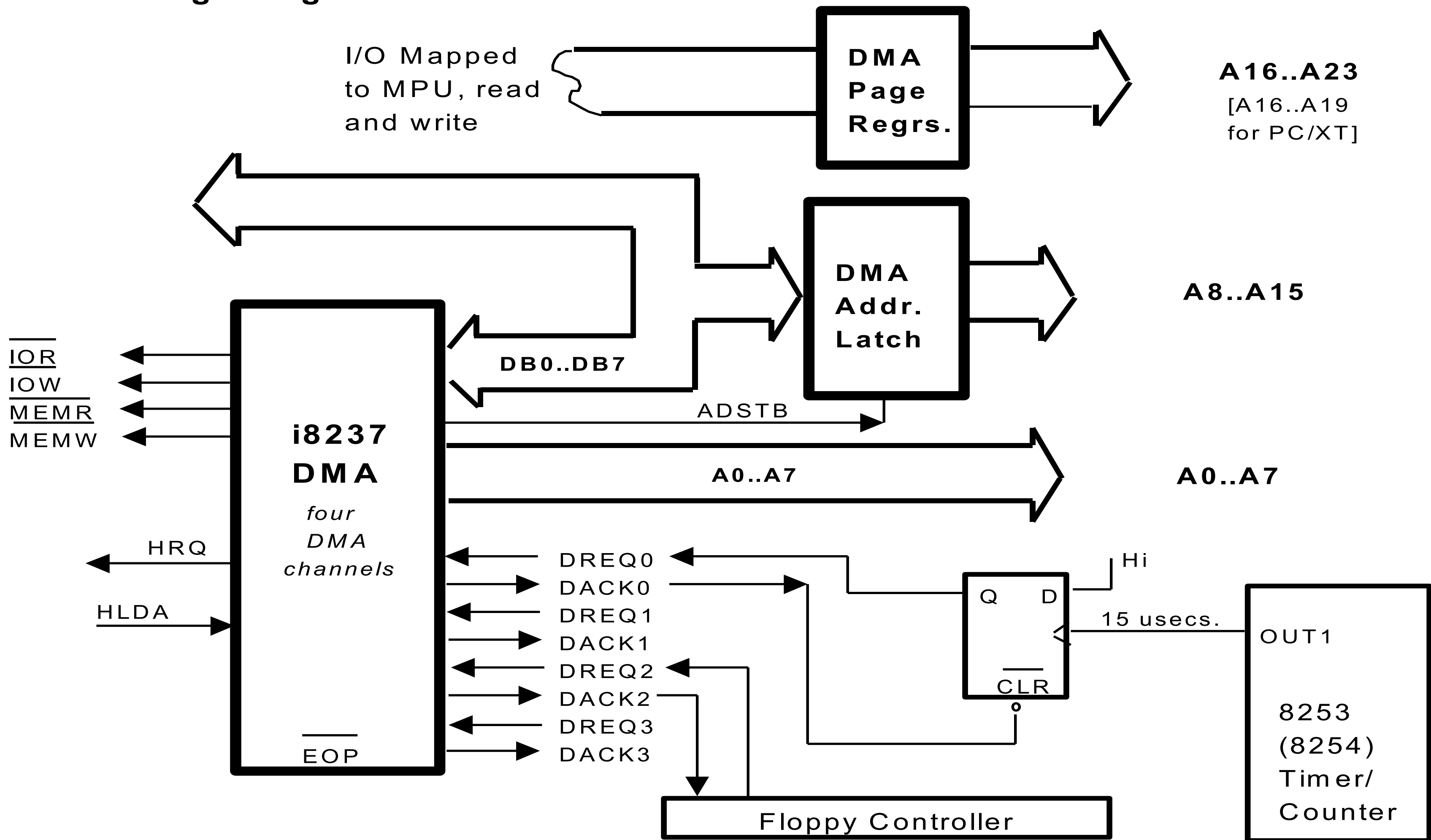
8088/8086 + 8237A



Generating 16 bit addresses



Generating addresses > 16 bits



Direct memory access (DMA)

Functioning modes

Idle (slave)

– programming the device (#CS=0, HOLDA=0)

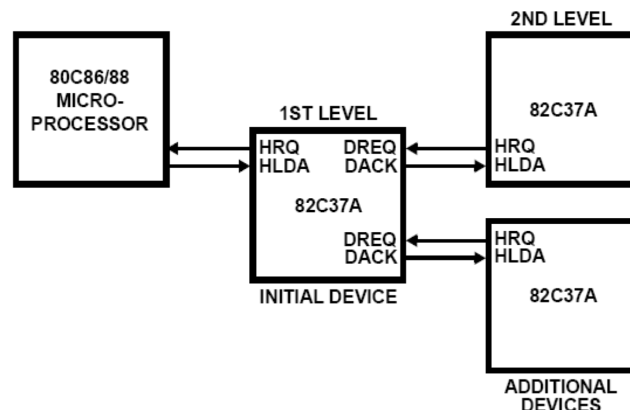
Active (master) – DMA transfer:

Single transfer Mode - release HOLD after each byte transferred. If DREQ is held active HOLD is issued again.

Bloc transfer Mode – transfers a block of size specified in the count register (DREQ need not to be held active).

Demand Transfer Mode – transfers data until external #EOP is received or until DREQ becomes inactive

Cascade Mode -



Direct memory access (DMA)

Transfer Types

Write transfers move data from an I/O device to the memory by activating MEMW and IOR.

Read transfers move data from memory to an I/O device by activating MEMR and IOW.

Verify transfers are pseudo-transfers (memory and I/O control lines all remain inactive). Verify mode is not permitted for memory-to-memory operation.

Memory-to-memory

- Channel 0 – source address & counter
- Channel 1 – destination address & counter
- The data byte read from the memory is stored in the 82C37A internal Temporary register
- The transfer is initiated by setting the software or hardware DREQ for channel 0. The 82C37A requests a DMA service in the normal manner.

Autoinitialize – a channel may be set up as an Autoinitialize channel. During

Autoinitialization, the original values of the Current Address and Current Word Count registers are automatically restored from the Base Address and Base Word Count registers of the channel following EOP. Following Autoinitialization, the channel is ready to perform another DMA service, without CPU intervention, as soon as a valid DREQ is detected, or software request made.