CG 2007 Microprocessor Systems
Lecture 9

Interrupt System

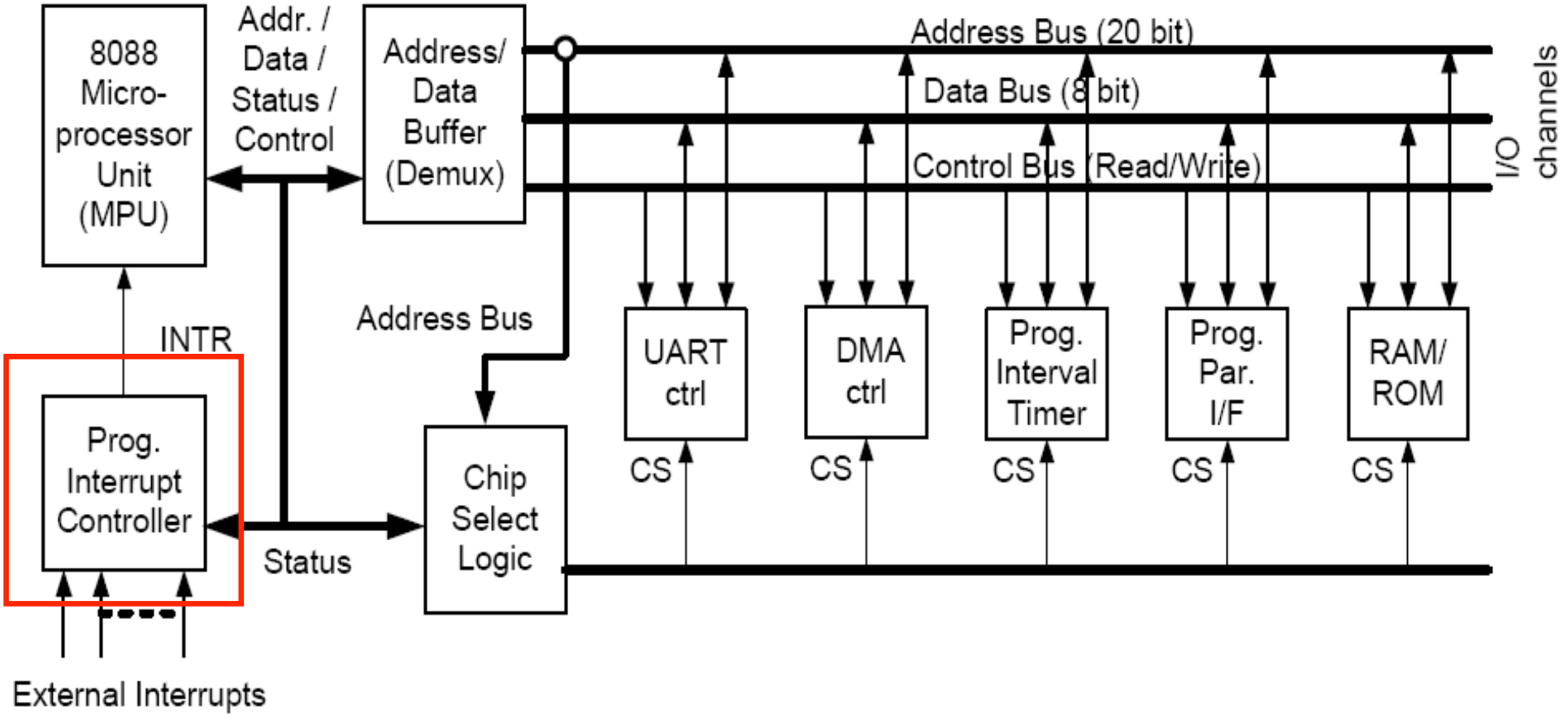
Dr. Ha Yajun
(E1-08-13, *elehy@nus.edu.sg*)

Lecture 9: Objectives and Outline

- **Objectives**
  - Understand interrupt system concepts and develop interrupt application

- **Outline**
  - Interrupt Concepts
  - Intel 8086/8088 Interrupt System
  - Interrupt Controller 8259A

System architecture of an IBM compatible PC



Outline: Interrupt System

- Interrupt Concepts
- Intel 8086/8088 Interrupt System
- Interrupt Controller 8259A

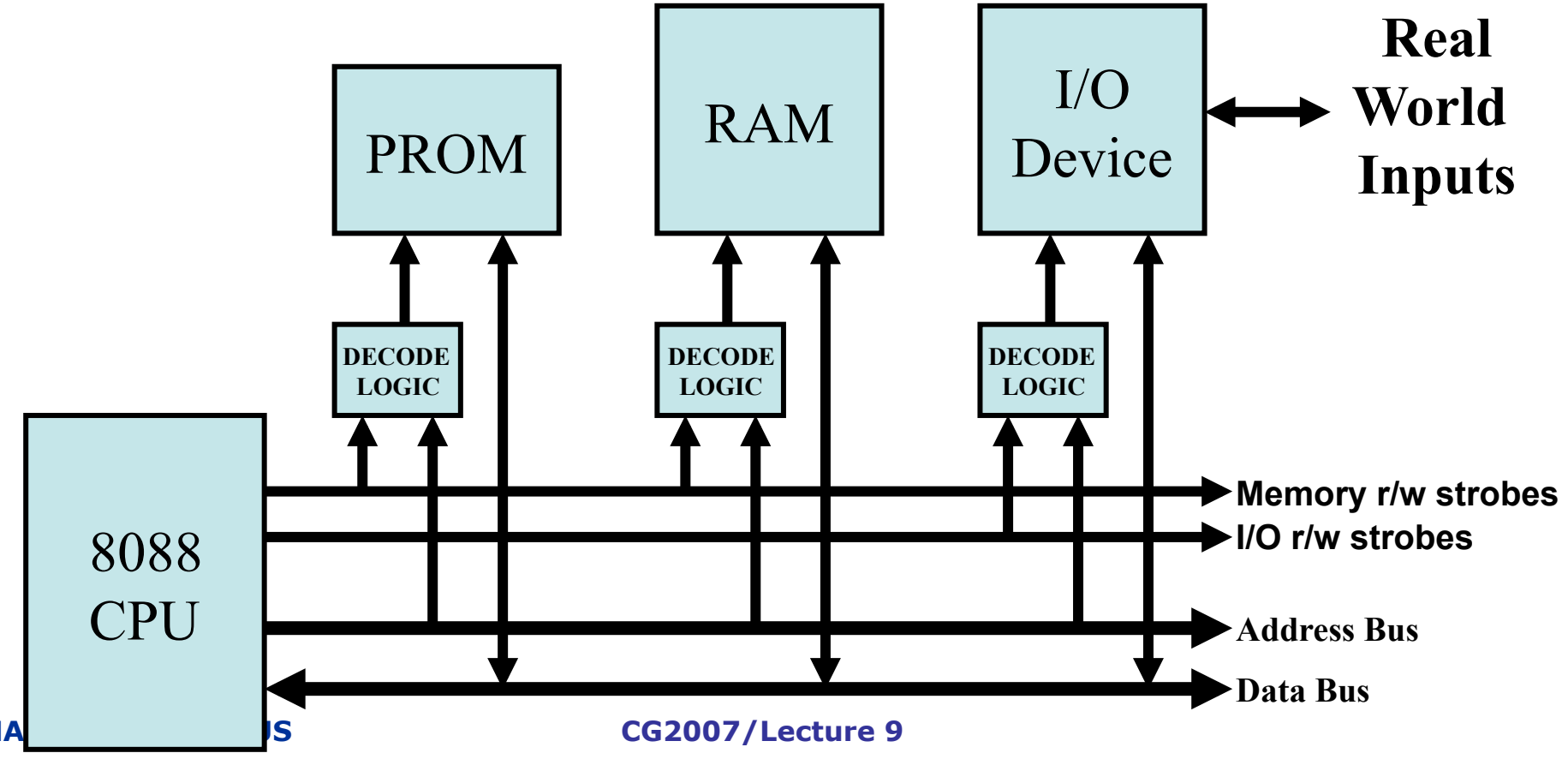
We Want to Serve I/O Devices Efficiently

- **We like to hook up I/O devices to computers**
  - Keyboards, monitors, etc.
  - Particularly true in embedded systems
- **I/O devices may require services from processor for real world inputs at unpredictable times**
  - CPU doesn't know when you're about to hit a key
- **Accessing I/O devices can take a long time**
  - Disk reads/writes have ~10ms latency
  - Would like processor to be able to do something else while waiting

Need an efficient way for processors to determine when I/O devices need services!

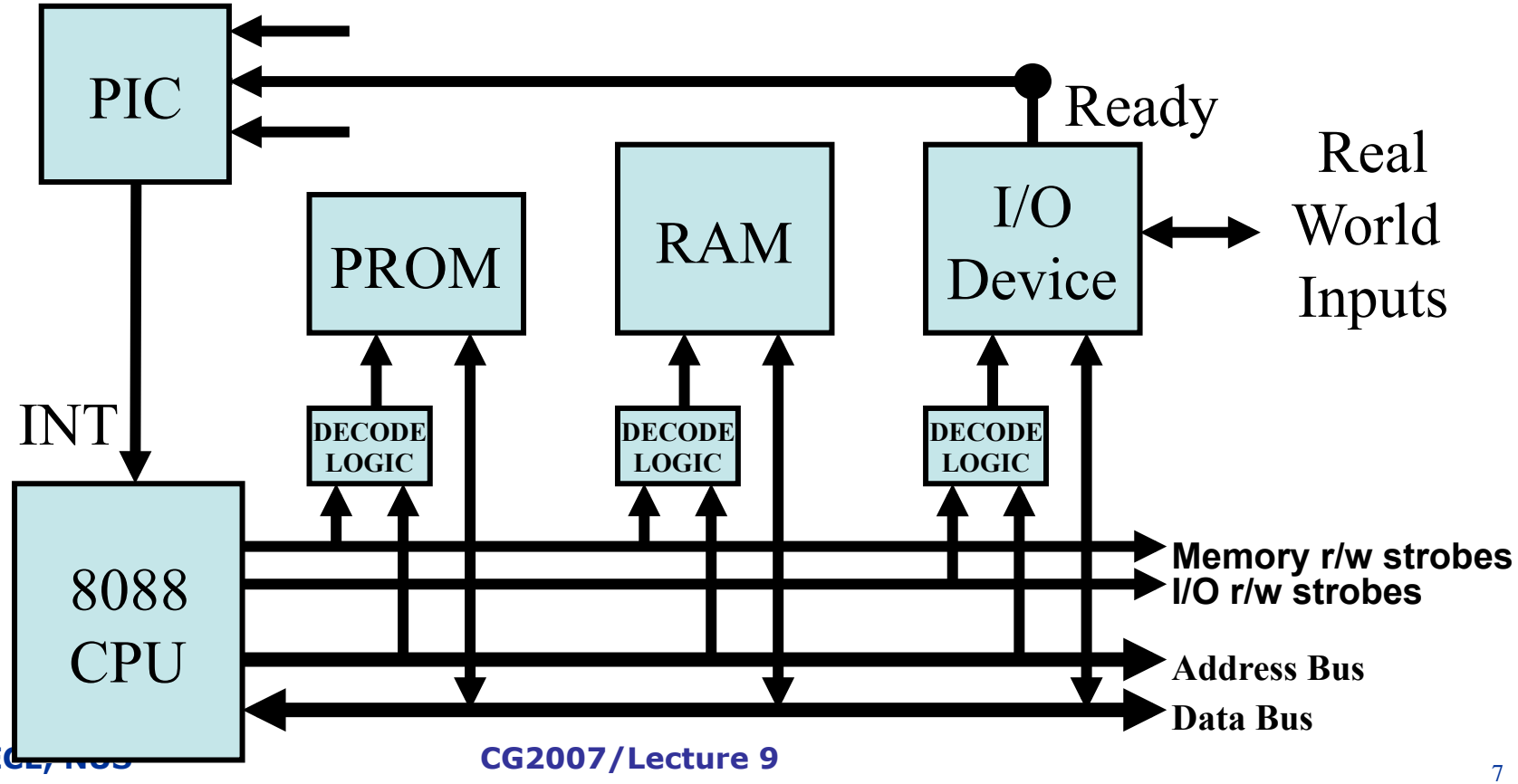
Alternative 1: Polling

- **Every so often, processor checks each device to see if it has a request**
  - Takes CPU time even if no requests pending
  - How does software know when to let the system poll?



Alternative 2: Interrupts

- **Give each device a wire (interrupt line) that it can use to signal the processor**
  - When interrupt signaled, processor executes a routine called an *interrupt service routine (ISR)* to deal with the interrupt
  - No overhead when no requests pending



Polling vs. Interrupts

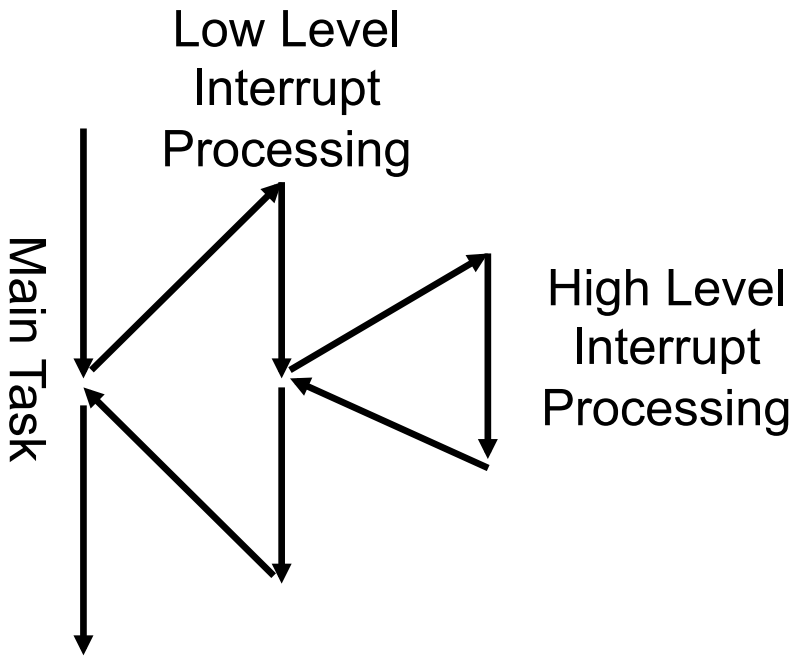
"Polling is like picking up the phone every few seconds to see if you have a call. Interrupts are like waiting for the phone to ring"

- **Interrupts are better if the processor has other work to do and the time to respond to events isn't absolutely critical**
- **Polling can be better if the processor has to respond to an event ASAP**
- **Performance of interrupt hardware is a critical factor on processors for embedded systems.**

Interrupt Processing Sequence (1)

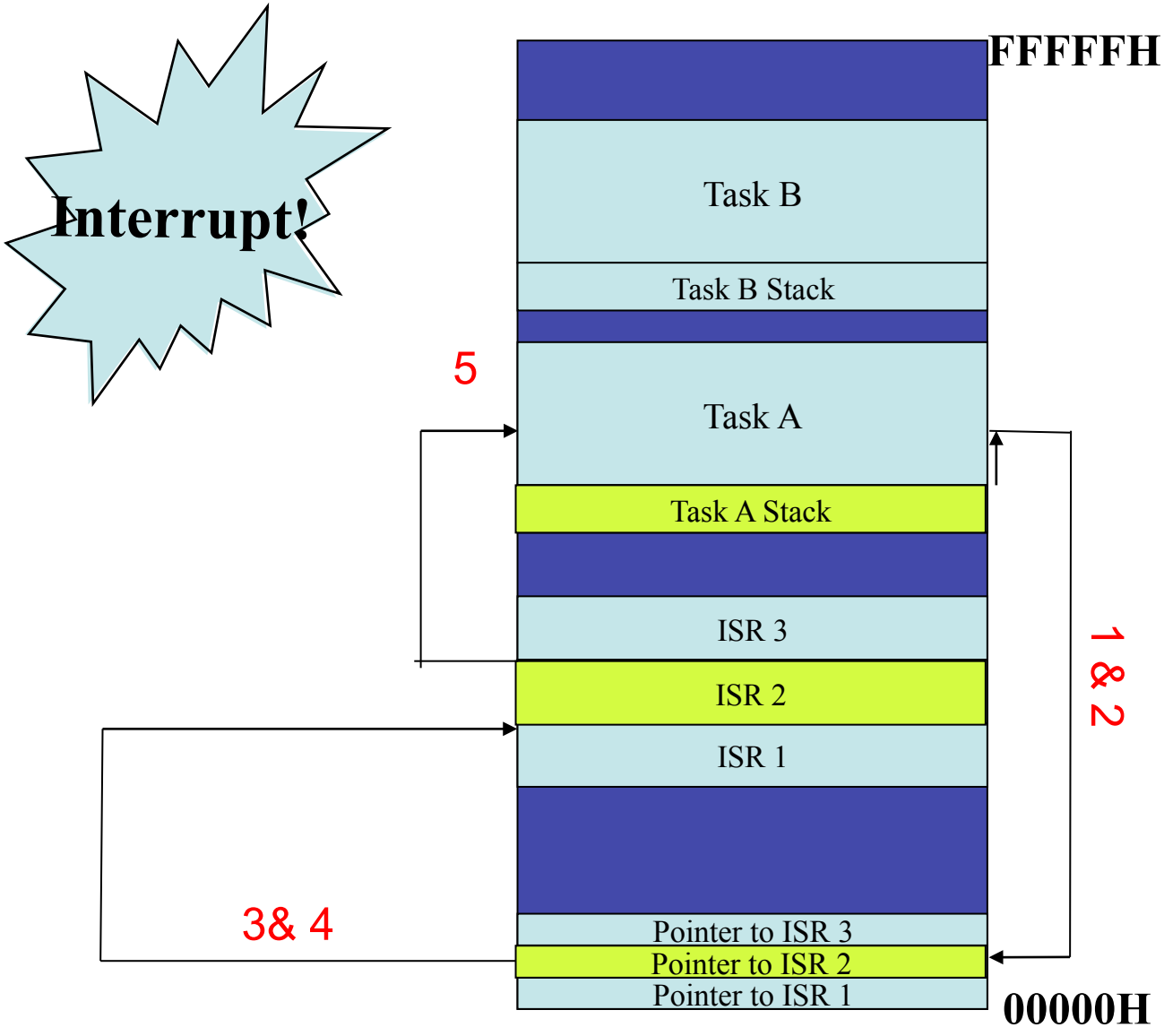
Typical interrupt processing scenario

1. During the execution of main task, a device asserts "ready" line
2. If used, PIC (Programmable Interrupt Controller) checks priority, asserts µP "INT" input.
3. µP finishes current instruction.
4. µP saves current context on stack.
5. µP gets an ISR (Interrupt Service Routine) address from Interrupt Vector Table
6. µP jumps to ISR and begins executing it.



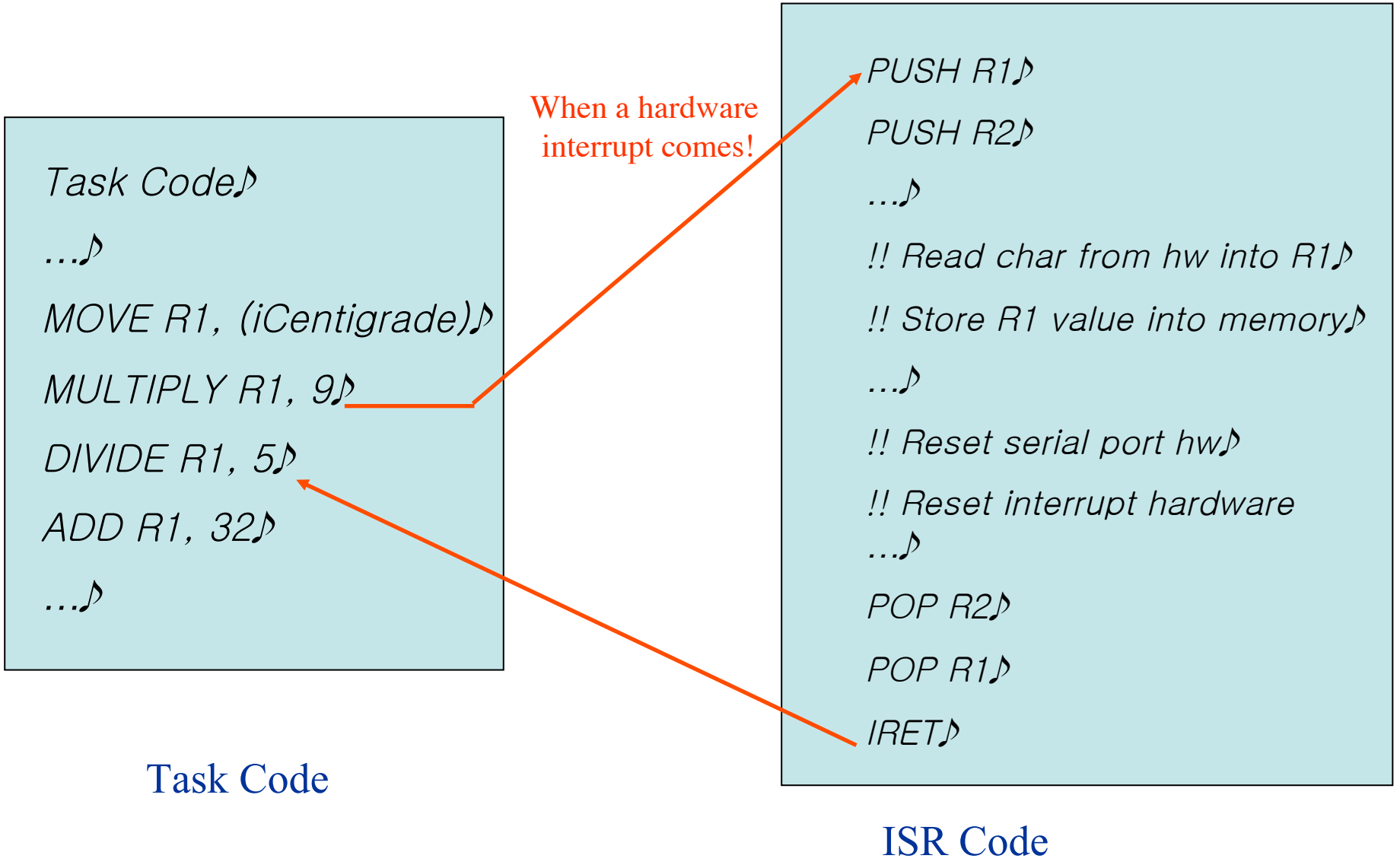
7. Executes steps 2-6 in case a higher level interrupt request comes.
8. Returns to previous suspension points, retrieve contexts from stack and continue to execute.

Interrupt Processing Sequence (2)



1. µP finishes current instruction in Task A.
2. µP saves current context on stack.
3. µP gets an ISR (Interrupt Service Routine) address from Interrupt Vector Table
4. µP jumps to ISR and begins executing it.
5. µP finishes ISR execution, restores previous context from stack and returns to Task A.

Hardware Interrupt Service Routine Sample Code



Outline: Interrupt

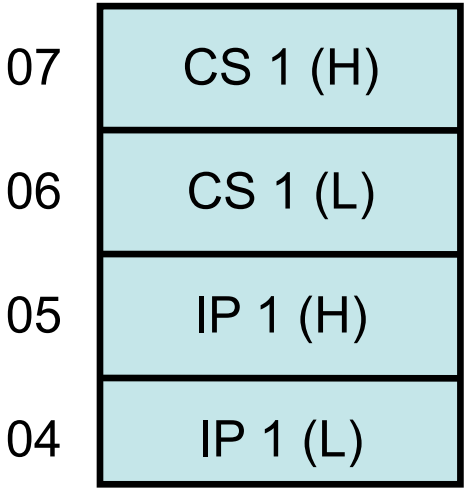
- Interrupt Concepts
- Intel 8086/8088 Interrupt System
- Interrupt Controller 8259A

Classification of Interrupt Sources

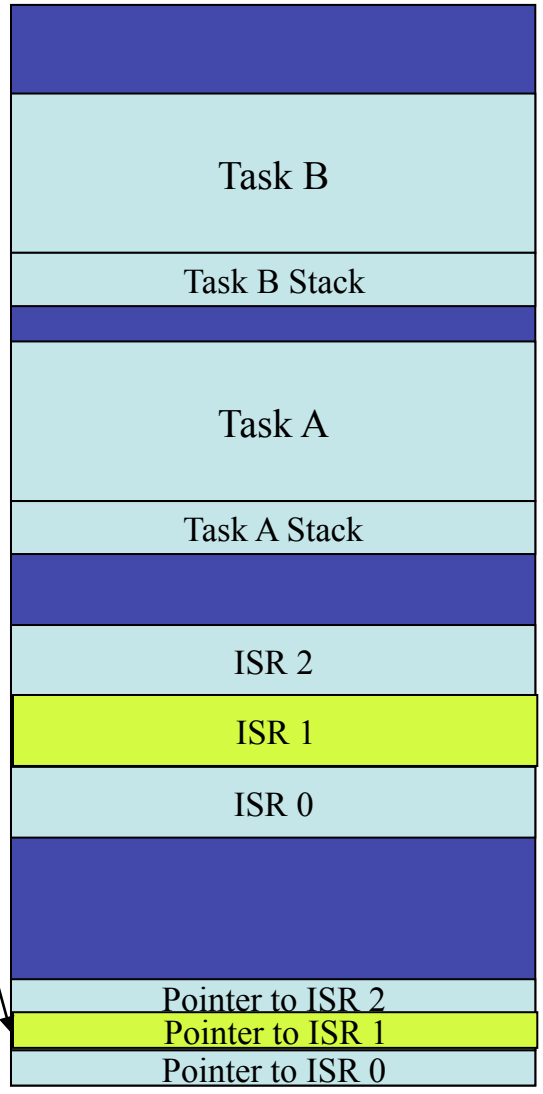
- **External Interrupt (Hardware Interrupt)**
  - Nonmaskable interrupts (NMI)
  - Maskable interrupts (INTR)
- **Internal Interrupt (Software Interrupt)**
  - Divide Error (INT 0)
  - Single Step (INT 1)
  - Breakpoint (INT 3)
  - Overflow (INT 4)
  - User Defined Interrupts (INT N)

Interrupt Vector

- ***Interrupt Vector*** **is the entry address of an Interrupt Service Routine (ISR).**
- **Every *interrupt type number* corresponds to an Interrupt Vector.**
- **Every interrupt vector takes 4 bytes of memory. The first 2 bytes are for IP, and the last 2 bytes are for CS.**



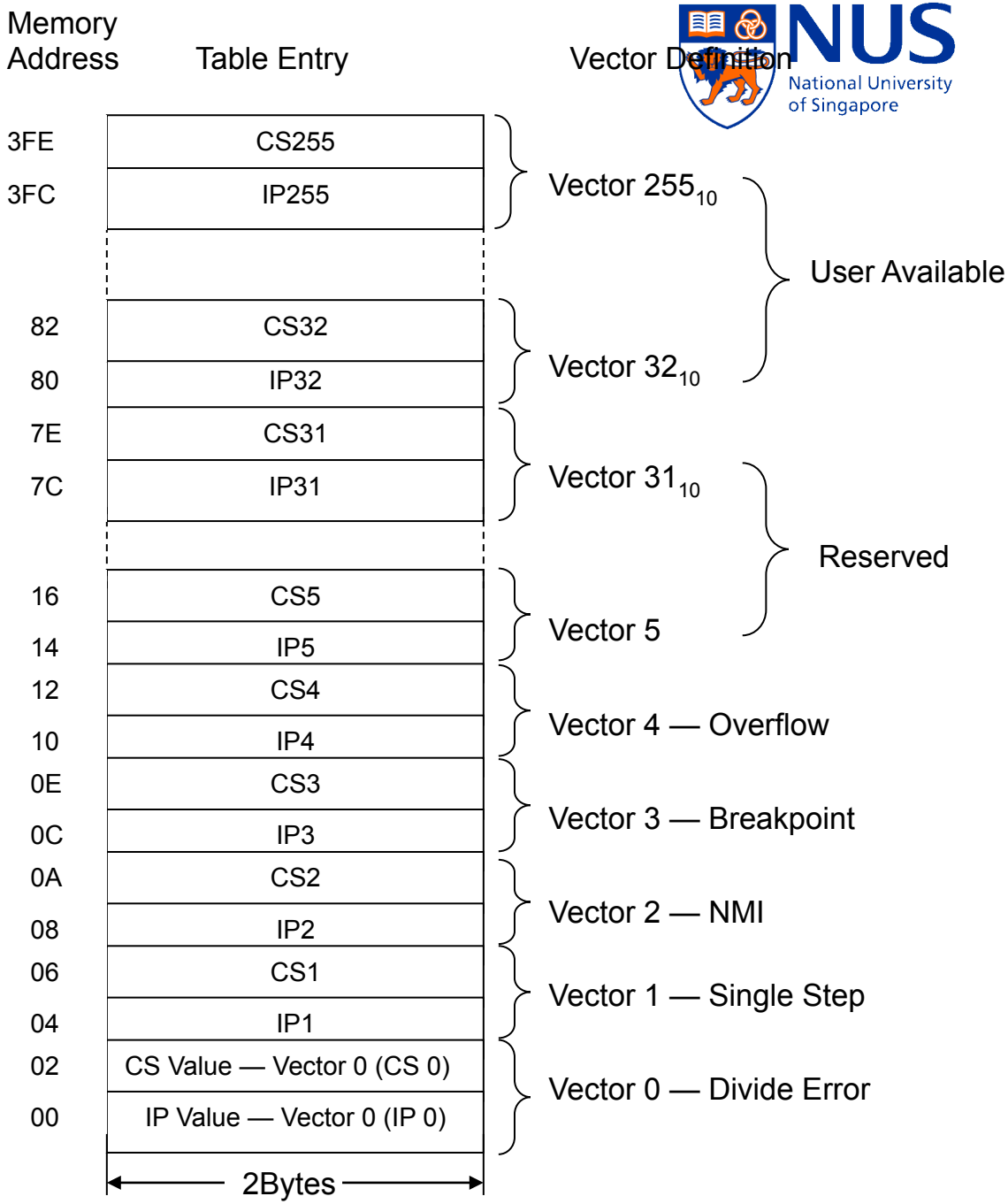
Interrupt Vector for interrupt type number 1



Interrupt Vector Table

- **The part of the memory space that stores interrupt vectors is called *interrupt vector table*, which is generally at the lower end of the memory space.**

- **8086/8088 systems have 256 types of interrupts (interrupt type numbers are 0-FFH). It takes 256 * 4 = 1024 Bytes at the lower end of the memory space, which is 00000H-003FFH. This part of memory space is called interrupt vector table in 8086/8088 systems.**



Interrupt Vector Address Calculation

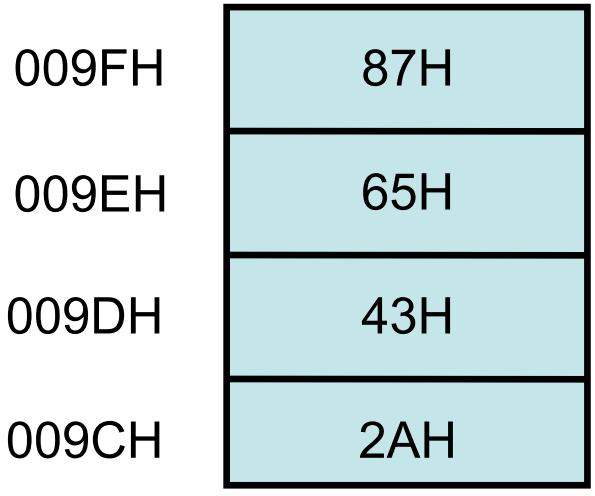
Interrupt Type Number * 4 → (IP)
Interrupt Type Number * 4 + 2 → (CS)

Example: If Interrupt Type Number is 27H, the starting address of its interrupt vector

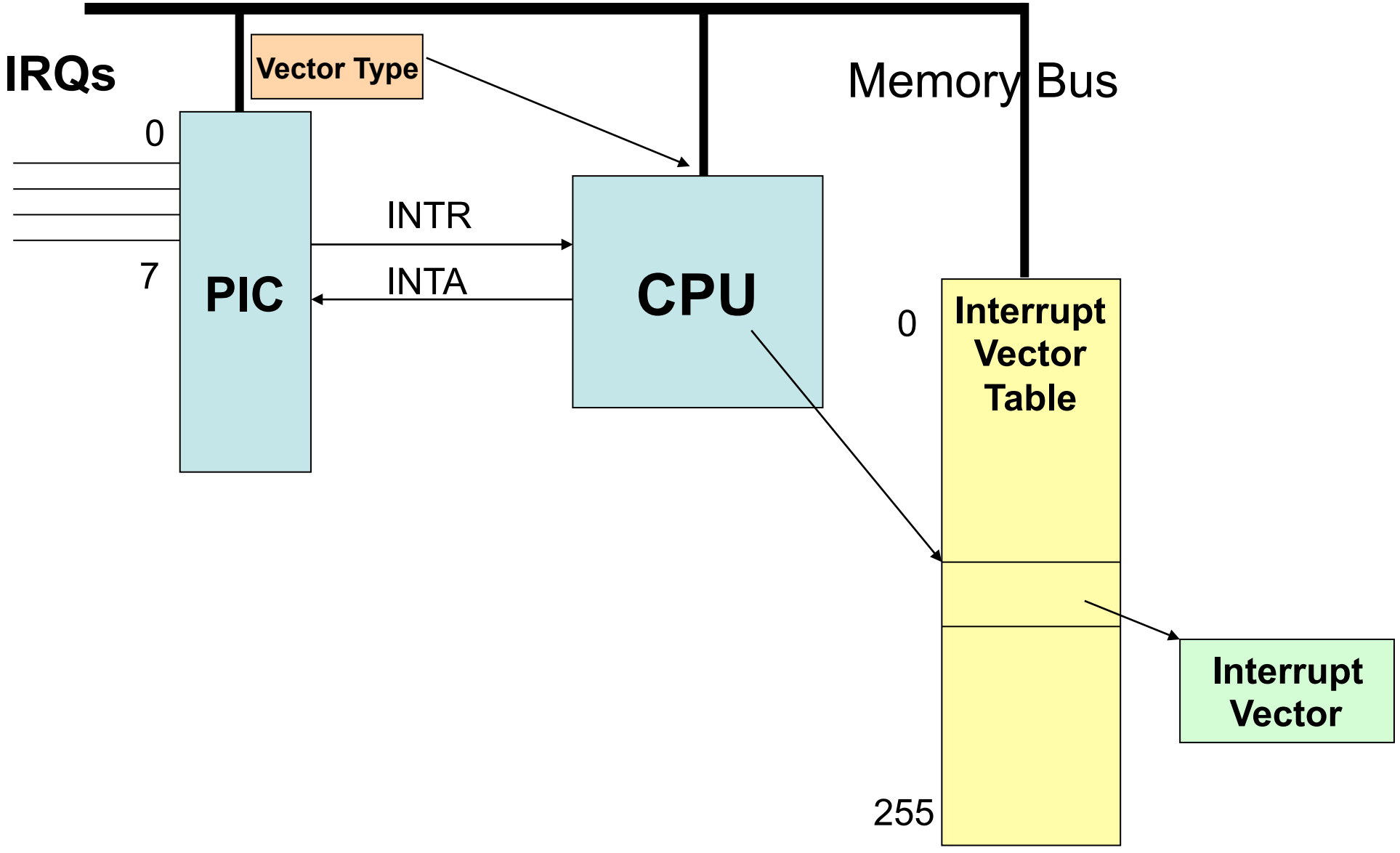
= 27H * 4 = 9CH

Therefore, the interrupt vector is stored in the 4 consecutive bytes starting from 0000H:009CH.

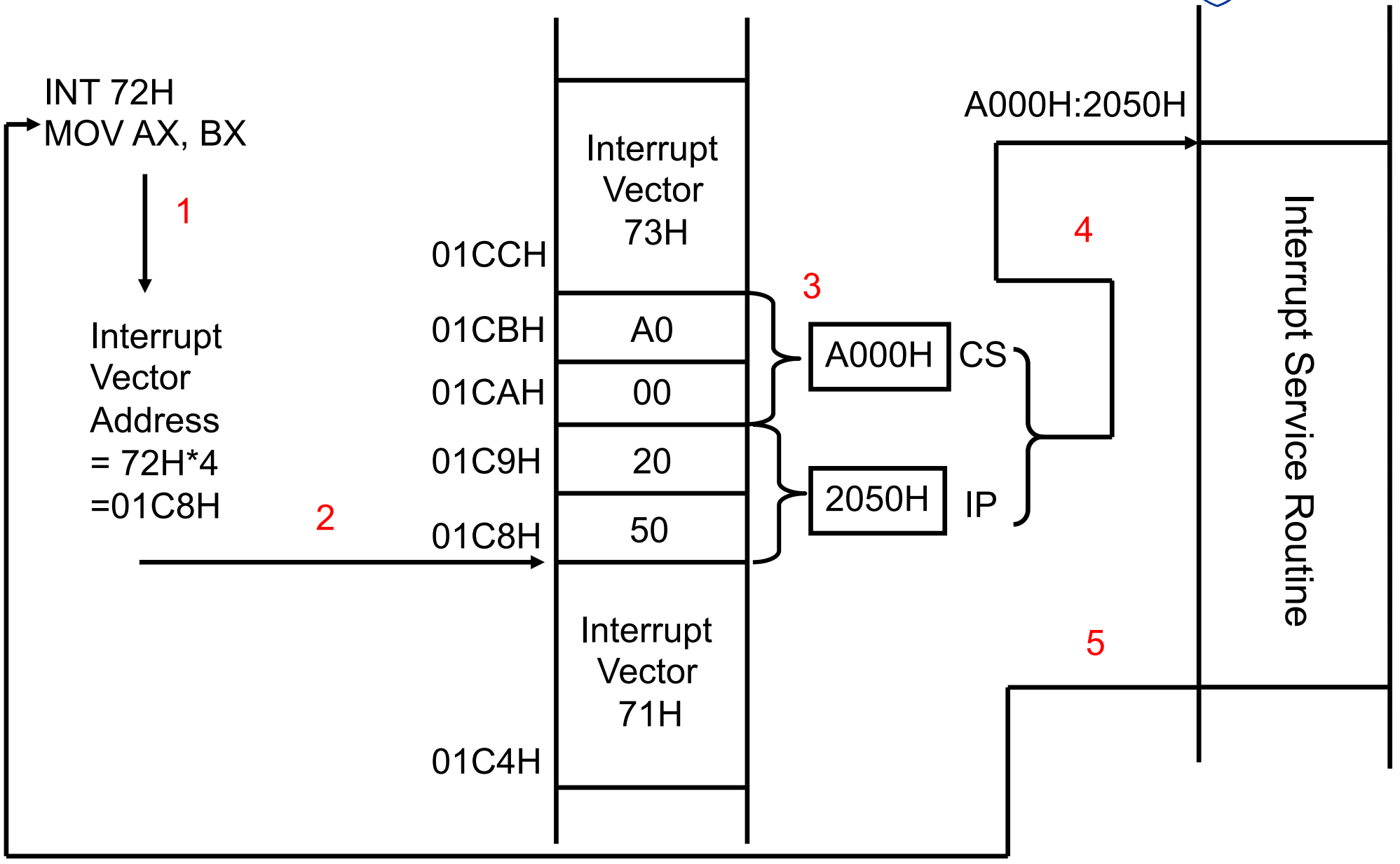
As a result, the logic address of the interrupt service routine for interrupt type 27H is (CS) = 8765H, (IP) = 432AH; the physical address is 8B97AH.



Interrupt Handling Components



Example Interrupt Handling Sequence

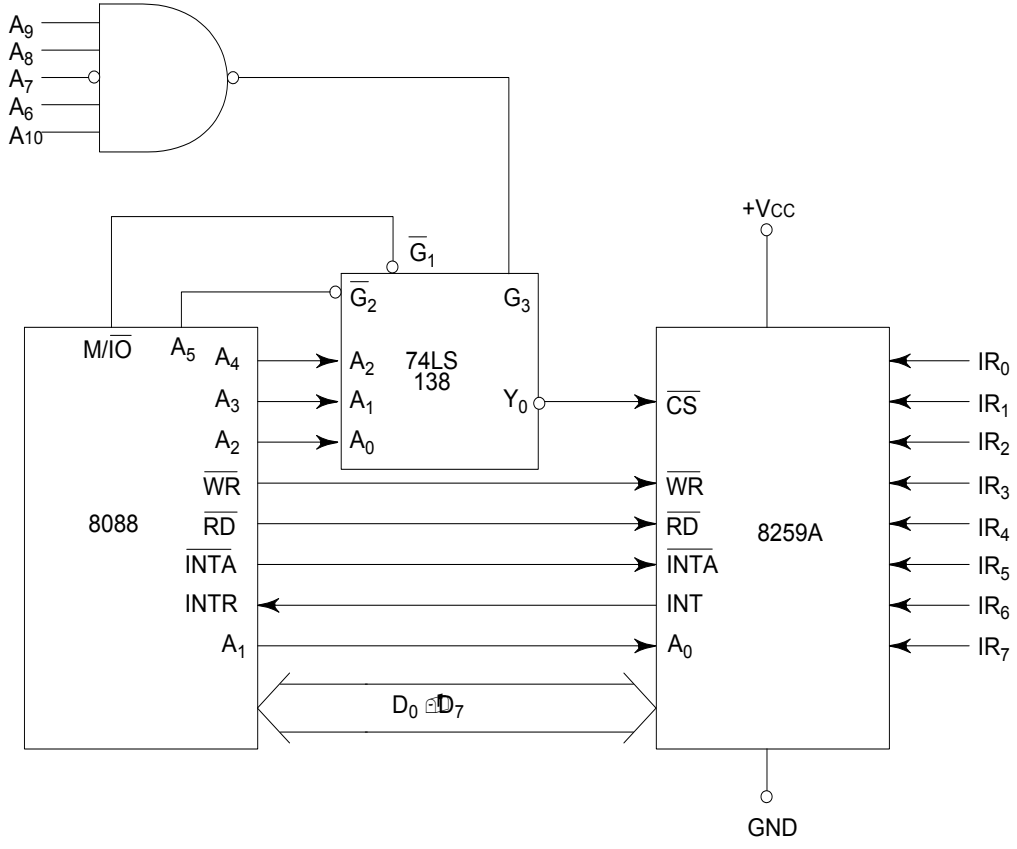


Outline: Interrupt

- Interrupt Concepts
- Intel 8086/8088 Interrupt System
- Interrupt Controller 8259A

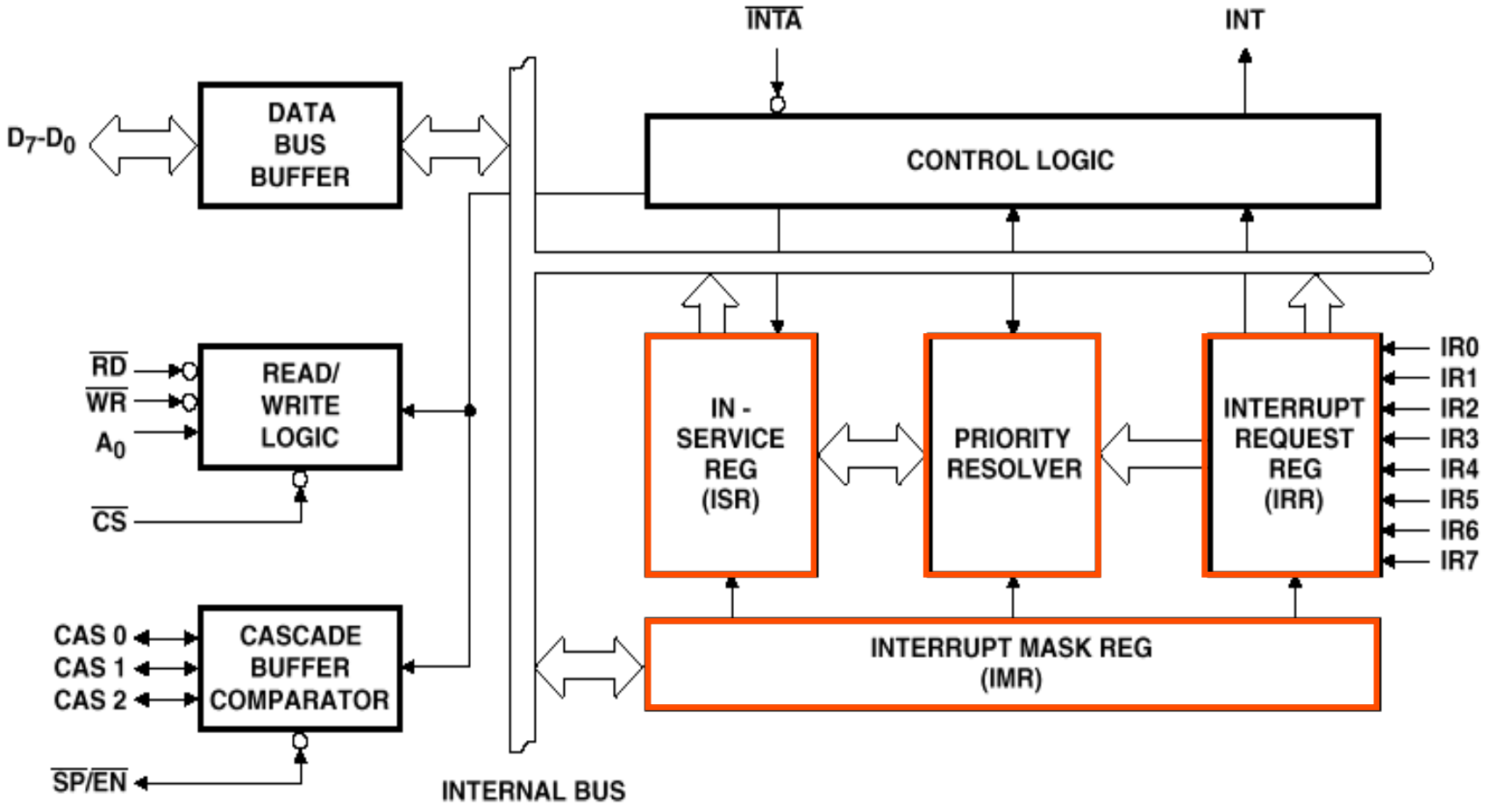
Programmable Interrupt Controller 8259A

- **Programmable Interrupt Controller (PIC)**
  - chip between devices and CPU
  - Fixed number of wires in from devices
    IRQs: Interrupt ReQuest lines
  - Single wire to CPU + some registers
- **PIC translates IRQ to vector type number**
  - Raises interrupt to CPU
  - Vector available in register
  - Waits for ack from CPU
- **Other interrupts may be pending**
- **Possible to "mask" interrupts at PIC or CPU**
- **Early systems cascaded two 8 input chips (8259A)**



Interrupt – 8259A

- **Programmable Interrupt Controller**
  - Operates with both 8-b and 16-b processors



Interrupt – 8259A

- **Registers**
  - IRR: Interrupt Request Register, stores all the requests, in order for the controller to serve one-by-one on the priority basis;
  - ISR: In-Service Register, stores all the requests being served;
  - IMR: Interrupt Mask Register, stores the mask bits, operates on IRR under the direction of the resolver.

- **Priority Resolver**
  - This logic block determines the priorities of the bits set in the lRR. The highest priority is selected and strobed into the corresponding bit of the lSR during the INTA sequence.

- **Modes**
  - EOI: End of Interrupt
  - AEOI: Automatic EOI

Interrupt - sequence

- **Typical interrupt sequence in an 8088 system**

  — Devices request by raising one or more IR lines. These lines **set** the corresponding **IRR (Interrupt Request Register)** bits;

  — 8259A resolves priority and sends an INT signal to 8088;

  — 8088 acknowledges with the first INTA\ pulse;

  — Upon receiving INTA\, the ***highest priority*** **ISR (In-Service Register)** bit is **set** and the corresponding **IRR** bit is **reset**. (Note: 8259A does not yet drive the bus);

  — 8088 will initiate a second INTA\ pulse. 8259A then releases an 8-b vector type to data bus which will be read by the 8088;

  — This completes the interrupt cycle.

  If in AEOI mode, ISR **is** reset **at the end of the second INTA**

  If in EOI mode, ISR **remains** set **until an appropriate EOI command is issued at the end of the interrupt subroutine.**

The Seven Command Words!

- **There are seven command words for the 8259.**
- **There are 4 Initialization Command Words (ICW1- ICW4)**
- **There are 3 Operation Command Words (OCW1 - OCW3).**
- **Normally the ICW are sent once for initial configuration of the 8259. The OCW are sent as needed during programming.**

Initialization Command Word ICW1

ICW1 This is sent to the base address of the 8259 as indicated at A0 bit. Recall that there are only two possible addresses for the 8259 registers. Bit0 of ICW1 indicates whether ICW4 will be used or not. Bit1 indicates single or multiple 8259s have been used. Bit3 indicates level or edge triggering will be used. Bit4 always is 1. Bit2 and Bit5-Bit7 are not needed for 86/88 systems, can be put to 0s.

A0	D7	D6	D5	D4	D3	D2	D1	D0
0	A7	A6	A5	1	LTIM	ADI	SNGL	IC4

Not needed for 8086/8088 system

1-level Triggered
0-Edge Triggered

1 = ICW4 Needed;
0 = no ICW4 needed

1- Single 8259A
0- Cascaded

Call Address Interval
1-4 bytes; 0-8 bytes
not needed for 86/88

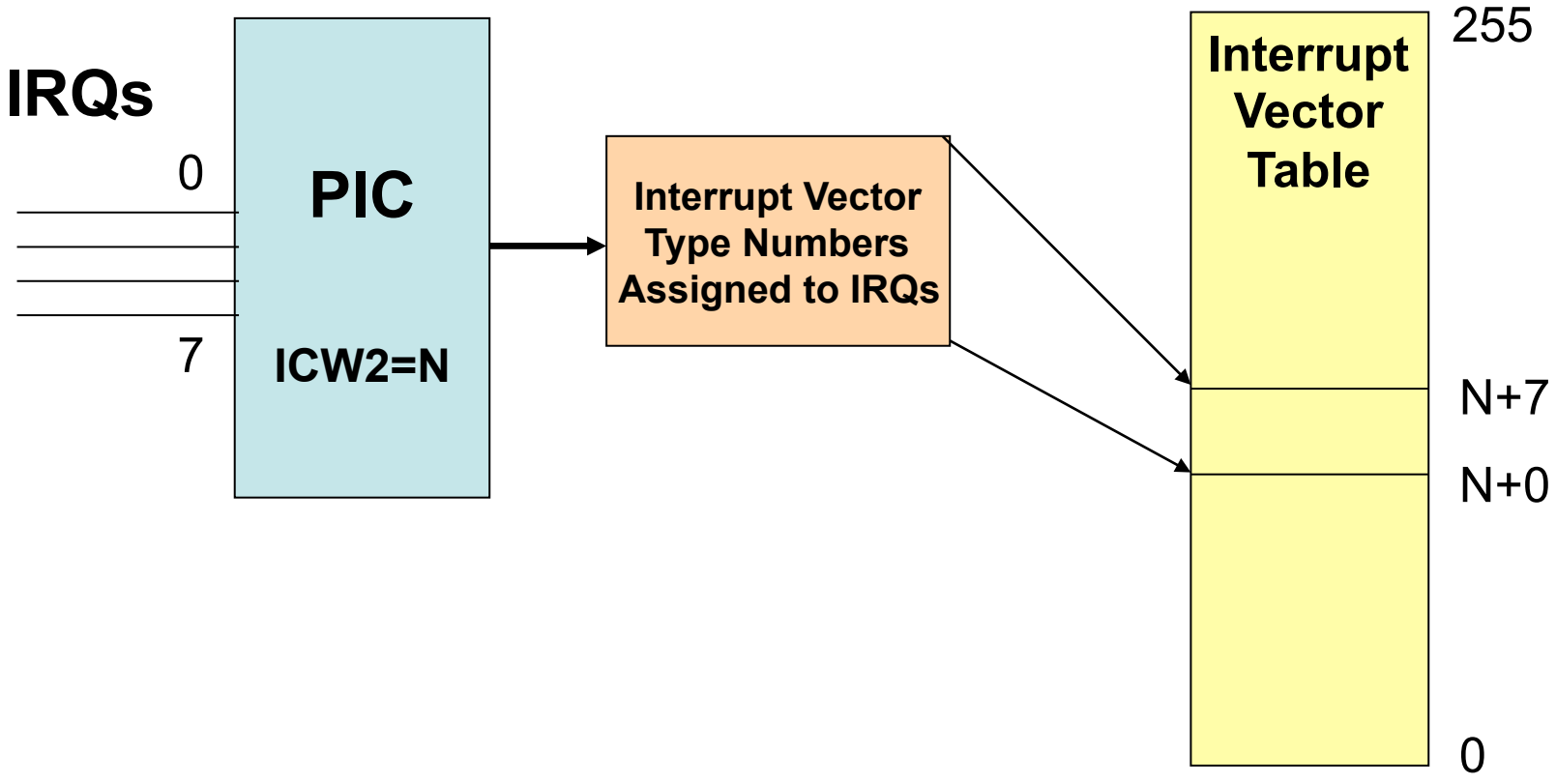
Initialization Command Word ICW2

• **ICW2** **This register specifies the type number assigned to the IR0 input of 8259 interrupt controller. Bits 0 - 2 must be zero. The 8-bit type number is for IR0.**

• **The 8259 automatically assigns consecutive type numbers to the other seven interrupts. For instance, if you specify ICW2 to be 40h then the type numbers for IR0 - IR7 will be 40h - 47h.**

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	T7	T6	T5	T4	T3	0	0	0

ICW2 Calculation



N must be multiples of 8! -> D_2-D_0 of ICW2 must be 0s!

Or say: N can only be of the values of 0, 8, 16,…,248.

Initialization Command Word ICW3

- **ICW3 This is sent only in a cascaded system (master and one or more slaves)**

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	S7	S6	S5	S4	S3	S2/ID2	S1/ID1	S0/ID0

Initialization Command Word ICW4

• **ICW4 Bit 0 must be 1 for Pentium. Bit1 is one if an AEOI is desirable or the programmer must issue an EOI (by means of an OCW2) before IRET at the end of the interrupt handler.**

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	SFNM	BUF	M/S	AEOI	µPM

Special fully nested mode

1-buffer mode
0-non buffer mode

Master or slave
1-master;
0-8 slave

1- Automatic EOI
0- not AEOI

1 = 8086/88 system;
0 = Mcs-85 system

Operational Command Word OCW1

- **OCW1 This is the IMR (Interrupt Mask Register) and one can read from or write to this register.**

- **A 1 in any bit will disable the IR corresponding to that bit e.g. in al, 21h reads the IMR of the master in the PC/AT**

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	M7	M6	M5	M4	M3	M2	M1	M0

Operational Command Word OCW2

• **OCW2** The EOI command is sent by this command word. Note that a **non-specific** EOI clears the last serviced interrupt and a **specific** clears the one specified by you. A **rotate on non-specific** EOI (command word A0h) would rotate the priorities in the priority resolver so that the one most recently served will go to the end of the priority line.

• **We only use automatic EOI and do not require you to use OCW2. OCW2 is not required in the exam.**

A0	D7	D6	D5	D4	D3	D2	D1	D0
0	R	SL	EOI	0	0	L2	L1	L0

Operational Command Word OCW3

- **OCW3 Bits 0 and 1 indicate an intention to read ISR or IRR. An immediate read of the address of OCW3 will yield the desired information.**

- **Please self study OCW2 and OCW3 for bit definitions.**

- **OCW3 is not required in the exam.**

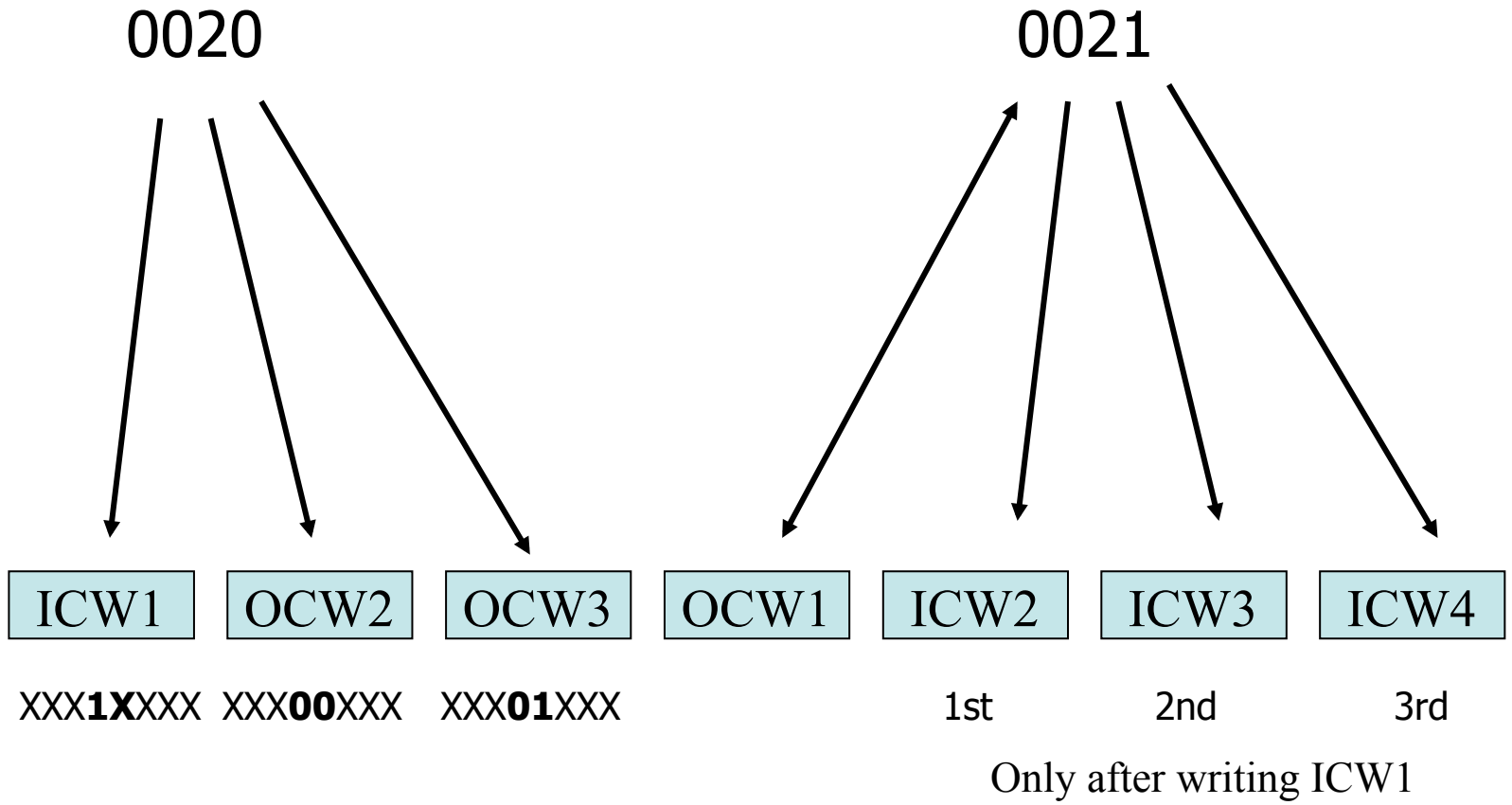
A0	D7	D6	D5	D4	D3	D2	D1	D0
0	×	ESMM	SMM	0	1	P	RR	RIS

Sequential Access to I/O Ports

- **Sometimes multiple read-only ports assigned to one address are distinguished by the *order* in which the data is read.**

- **Multiple write-only ports assigned to one address can also be distinguished by order, or by the *value* of specific bits in part of the data that is written.**

I/O Ports of the 8259 Interrupt Controller in the IBM/PC.



Example Code to Use 8259A

Initialization

```
MOV AL, 13H;        writes ICW1,single chip, edge trigger, need ICW4
OUT 20H, AL;

MOV AL, 08H;        writes ICW2, interrupt type number starts from 8
OUT 21H, AL;

MOV AL, 09H;        writes ICW4, buffered mode, 8088/8086 configuration
OUT 21H, AL;

MOV AL, 00H;        writes OCW1, allows all IRQ0-IRQ7
OUT 21H, AL;
……

MOV AL, 20H;        writes OCW2, issues EOI command
OUT 20H, AL;
IRET                return
```

For IRQ4, its interrupt type number is 8+4 =12, interrupt vector is stored at the four bytes starting from 12*4=48 (30H). 30H and 31H store IP, while 32H and 33H store CS of interrupt vector for IRQ4.

Outline: Interrupt

- Interrupt Concepts
- Intel 8086/8088 Interrupt System
- Interrupt Controller 8259A
- To be covered in the next lecture:

  Direct Memory Access (DMA) Controller, the i8237