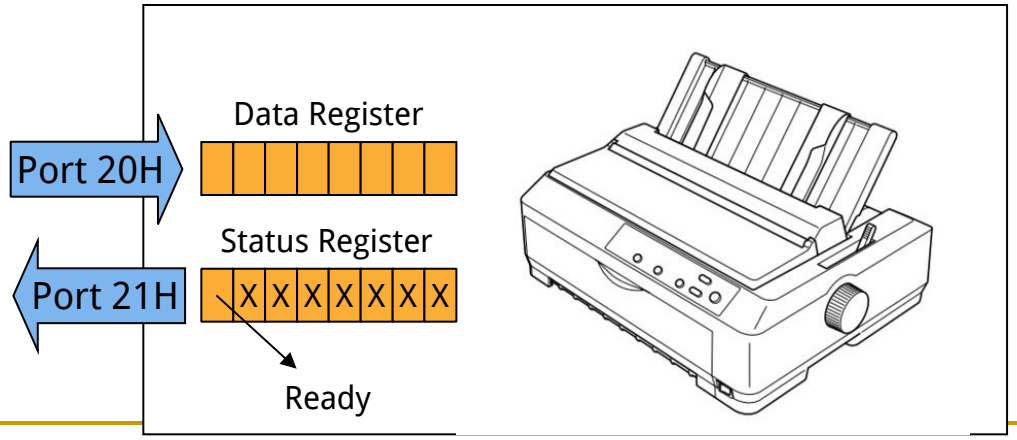
Microprocessors, Lecture 7

Interrupts

Hamid Fadishei
Assistant Professor, University of Bojnord
Spring 2015

Motivation

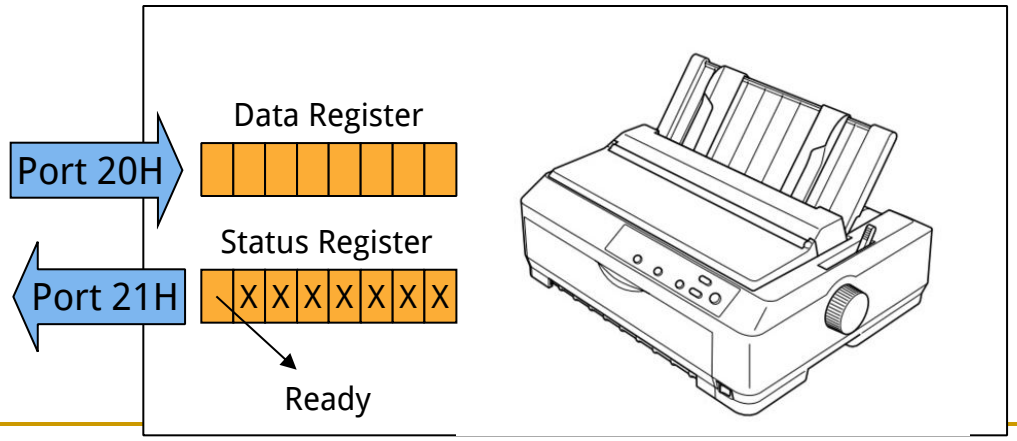
- When software wants to perform an I/O operation, it should make sure that the target I/O device is "Ready" for the operation
  - For output devices, "ready" usually means that the device buffer is empty (for example, the previous output data is consumed) and the device can accept new data
    - examples
      - a character printer device has printed the previous character
      - a network interface card has sent the previous data
  - For input devices, "ready" usually means that new data is available at the device buffer to be read by the microprocessor
- I/O devices usually have some status bits
  - which can be accessed via some input port to check whether the device is ready for new I/O or not



Motivation

- Polling (Programmed I/O)
  - Microprocessor waits in a busy-wait loop until the I/O device becomes ready
  - Not efficient (Wastes CPU time in loops)

```
PUTCHAR: IN       A, (21H)
         BIT      7, A
         JR       NZ, PUTCHAR
         OUT      (20H), A
         RET
```

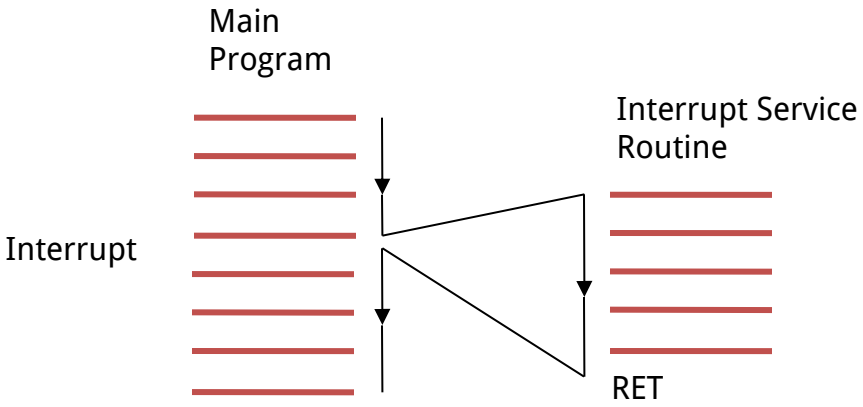


Motivation

- Interrupt-driven I/O
  - Instead of wasting CPU time in loops by polling
  - CPU time is used efficiently by executing the main program logic
  - When a device is ready for I/O, it informs CPU via an “interrupt pin”
  - CPU stops executing the main program and performs the required I/O operation with that device
  - Then CPU continues the execution of the main program from the point it was interrupted

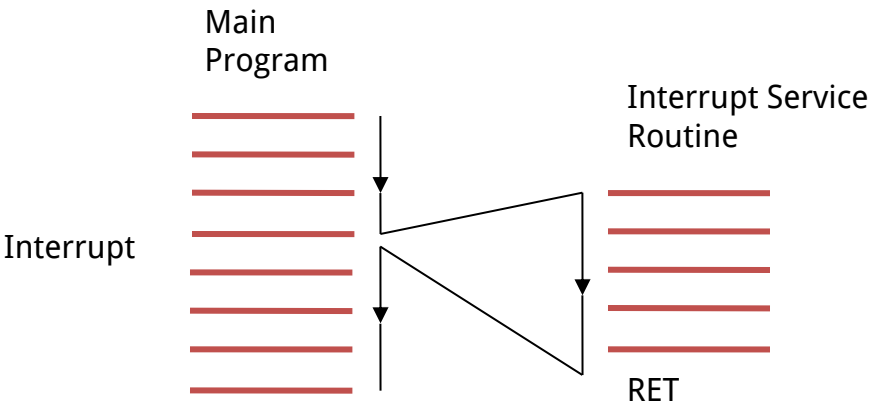
Interrupts

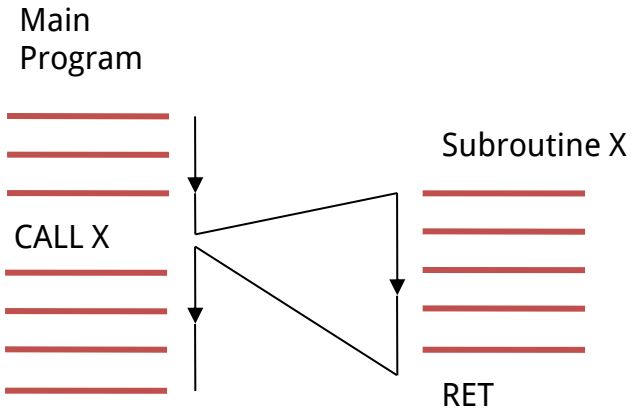
- Interrupt
  - A process in which the computer stops executing the main program and calls a subroutine located somewhere in the memory
  - After returning from that subroutine, main program execution is resumed



Interrupts

- Subroutine calls vs interrupts
  - Interrupts are triggered by external event
  - Subroutine calls are triggered by main program executing CALL instruction





Interrupts

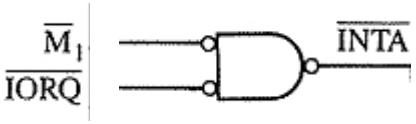
- Common questions and problems
  - How the interrupt request is generated
  - How the ISR address is determined
  - How can we disable/enable interrupts
  - Multiple interrupting devices and simultaneous interrupts

Interrupts in Z80 microprocessor

- Z80 has two Interrupt pins
  - INT
    - Maskable interrupt pin
    - Executing DI instruction disables interrupts
    - When disabled, Z80 ignores requests on this pin
    - Interrupts can be enabled by executing EI instruction
  - NMI
    - Non-maskable interrupt pin
    - can not be disabled by software

Interrupts in Z80 microprocessor

- Z80
  - Z80 checks INT pin at the rising edge of the last clock of the each instruction
  - If interrupt is enabled, Z80 acknowledges by activating both IORQ and M1 pins (instead of MEMRQ and M1 for fetching next instruction)
    - After the acknowledge, the interrupting device can deactivate the INT pin



  - How the address of ISR is determined by Z80?
    - it depends on the selected "interrupt mode"
      - INT mode 1: ISR is at 0038H
      - NMI: ISR is at 0066H
      - Other modes (mode 0 and 2): we'll see

Z80 Interrupt modes

- Z80 interrupt system works in 3 modes
- Modes can be switched by program
  - IM 0, IM 1, IM 2 instruction
- Interrupt mode 1
  - The ISR should be located at 0038H
  - Processor calls the ISR at this address
  - Very simple, but what if we need to have several interrupting devices (and several ISRs)?
  - Let's look at the more complicated mode 0

Z80 Interrupt modes

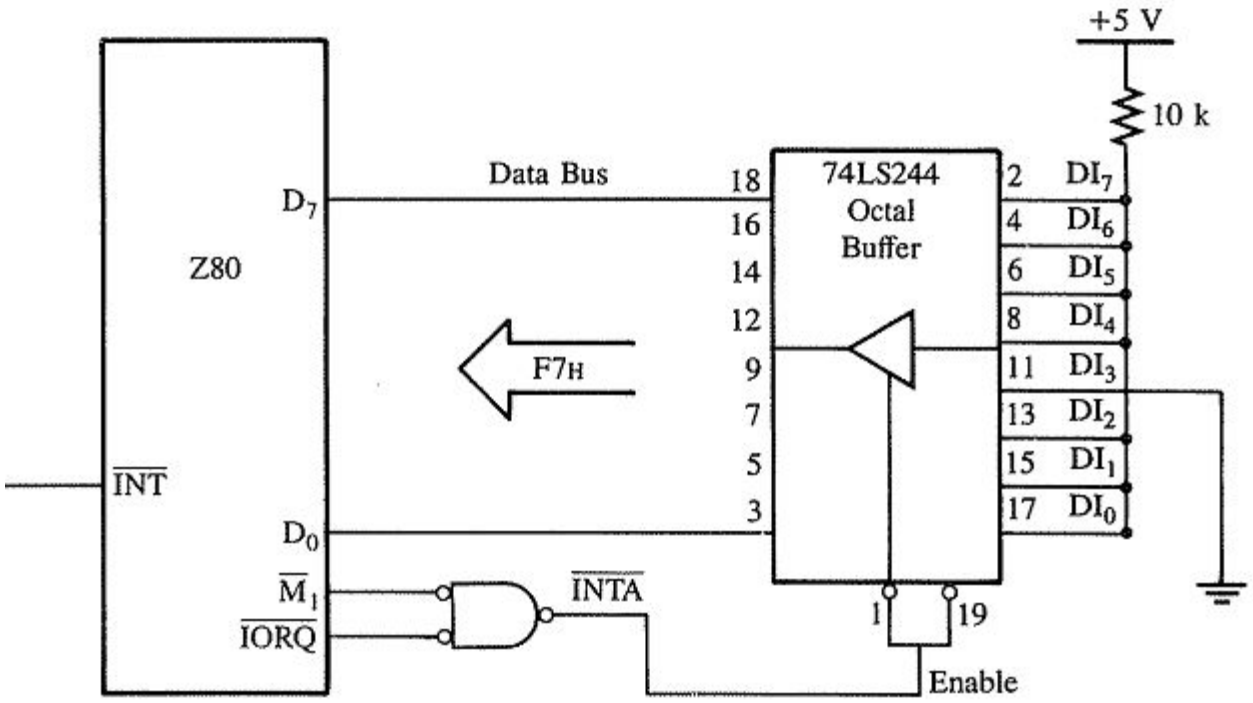
- Interrupt mode 0
  - The interrupting device specifies the proper ISR to be called
  - How?
    - When mode 0 is selected, after an interrupt request, Z80 executes an instruction whose opcode is placed on the data bus
    - The interrupting device should place the opcode
    - This instruction is usually an RST
    - Thus the device can point to its ISR address to be called

Restart Instructions

Mnemonics	Binary Code								Hex Code	Call Location (Hex)
	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
RST 00H	1	1	0	0	0	1	1	1	C7	0000
RST 08H	1	1	0	0	1	1	1	1	CF	0008
RST 10H	1	1	0	1	0	1	1	1	D7	0010
RST 18H	1	1	0	1	1	1	1	1	DF	0018
RST 20H	1	1	1	0	0	1	1	1	E7	0020
RST 28H	1	1	1	0	1	1	1	1	EF	0028
RST 30H	1	1	1	1	0	1	1	1	F7	0030
RST 38H	1	1	1	1	1	1	1	1	FF	0038

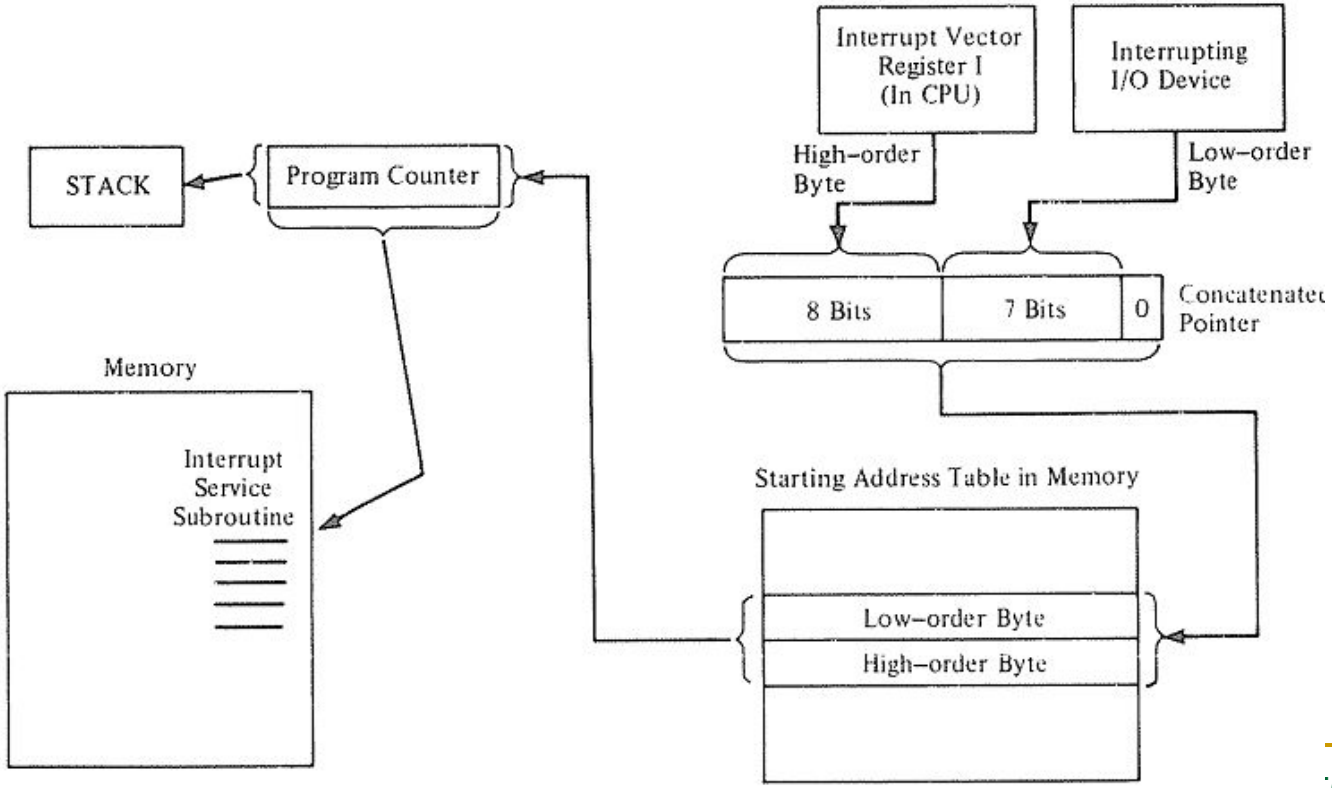
Z80 t terrupt modes

- Mode 0 interrupt hardware example



Z80 t terrupt modes

- Interrupt mode 2
  - After interrupt, Z80 reads the byte on data bus, as the lower byte of a 16 bit address
  - The high byte of the address will be the contents of the I register
  - The memory location addressed by this 16-bit address holds the ISR address

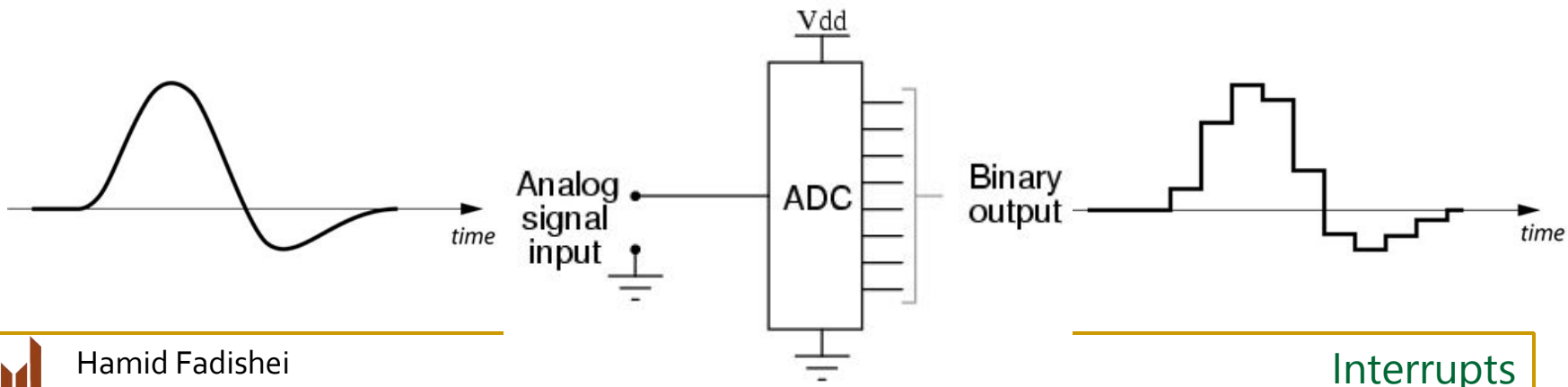


Enabling and disabling Z80 interrupts

- Servicing the interrupt requests (coming from INT pin) can be disabled by program
- Controlled by IFF1 flag
- The interrupt request will be ignored if this flag is 0
- EI instruction enables interrupts (sets IFF1)
- DI instruction disables interrupts (clears IFF1)
- Note: after any interrupt request, Z80 automatically clears IFF1
  - So the ISR itself can not be interrupted again
  - Should be set by program in ISR (usually before RET)

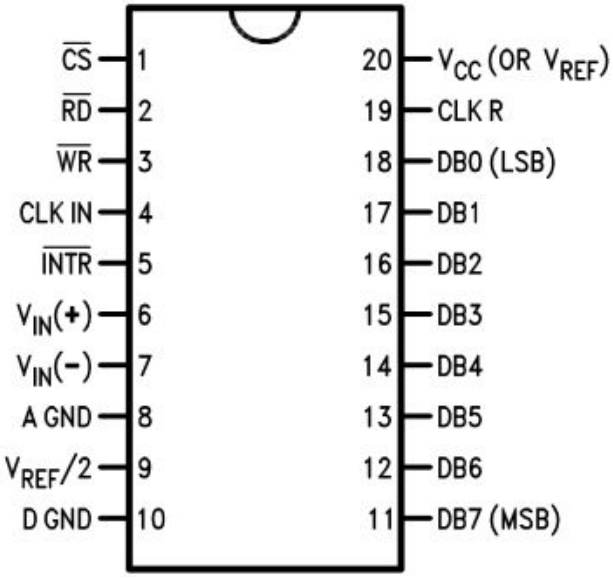
Case study: A/D conversion

- Introduction
  - Some inputs to a microprocessor system may be continuous
    - Temperature
    - Speed
    - ...
  - Microprocessors only understand digital (discrete) values
  - A conversion is needed
  - Analog to digital converter (ADC)
    - a device which converts continuous signals to discrete digital numbers
    - Input: an analog signal (Usually voltage)
    - Output: a binary number proportional to the magnitude of the input voltage
  - The conversion time may vary from some microseconds to a fraction of a second



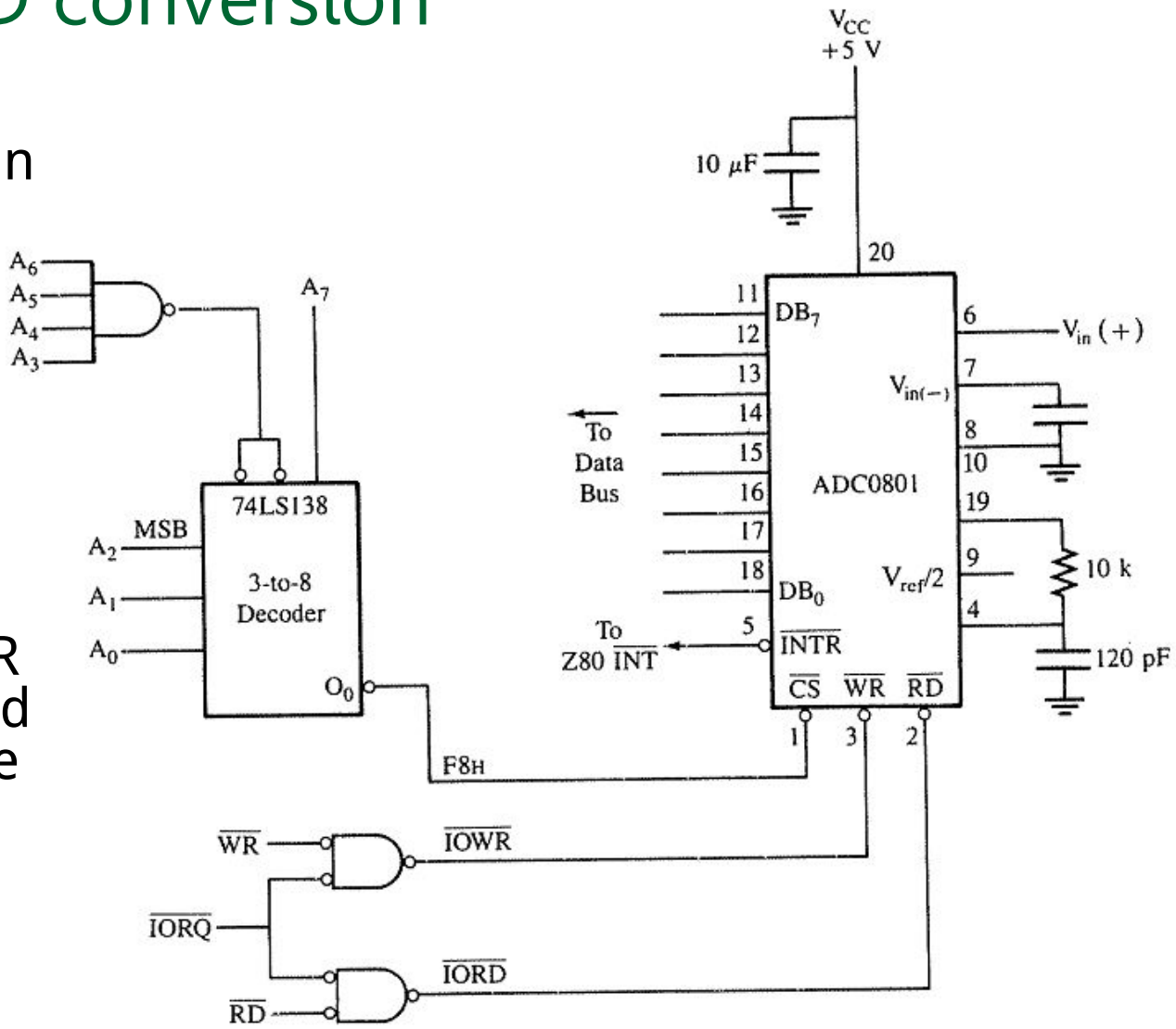
Case study: A/D conversion

- ADC0801 is an A/D converter chip
- The conversion process
  - Microprocessor asserts a START command
    - Writing to a command output port
  - ADC starts conversion
  - At the end of the conversion process, the converted value can be read from ADC
    - How to know the conversion is finished?
      - By polling a status input port; or getting an interrupt



Case study: A/D conversion

- ADC starts conversion when an OUT to F8H is performed. No matter what the output byte is)
- It asserts INTR low when the conversion is finished
- It de-activates INTR when the converted value is read by the CPU



Case study: A/D conversion

MAIN PROGRAM

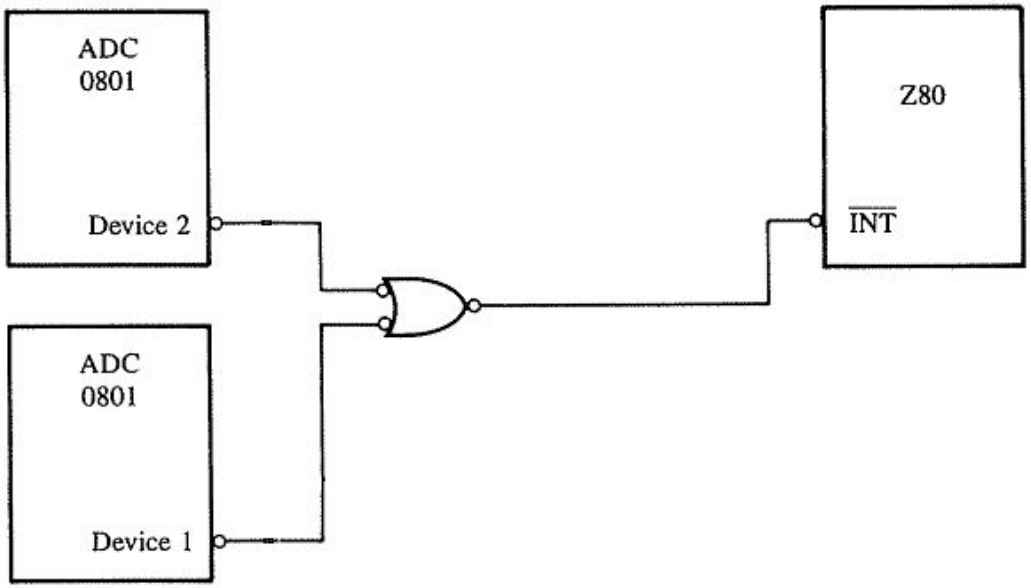
```
START:  LD SP, STACK      ;Initialize stack pointer
        IM 1              ;Set up interrupt Mode 1
        LD HL, BUFFER     ;Set up HL as memory pointer
        LD B, BYTE        ;Set up counter to count the number of readings
        EI                ;Enable interrupt flip-flops
        OUT (F8H), A      ;Start conversion
WAIT:   NOP
        JP NZ, WAIT       ;If all data readings are not yet recorded, wait
        HALT
0038    JP ADC            ;This is Mode 1 restart location; go to data
                          ;  converter service routine
```

SERVICE ROUTINE

```
ADC: ;This service routine reads data from the A/D converter, saves data in
     ;memory, and starts conversion for the next reading.
     ;Input: Address of memory pointer in HL and the number of readings to be
     ;recorded in register B.
     ;Modifies registers A, B, and HL
          IN A, (F8H)      ;Read data byte from the converter
          EI               ;Enable the interrupt
          LD (HL), A       ;Save data in memory
          INC HL           ;Next memory location
          DEC B            ;One reading is recorded, decrement counter
          OUT (F8H), A     ;Start conversion for the next reading
          RET
```

Multiple interrupt sources

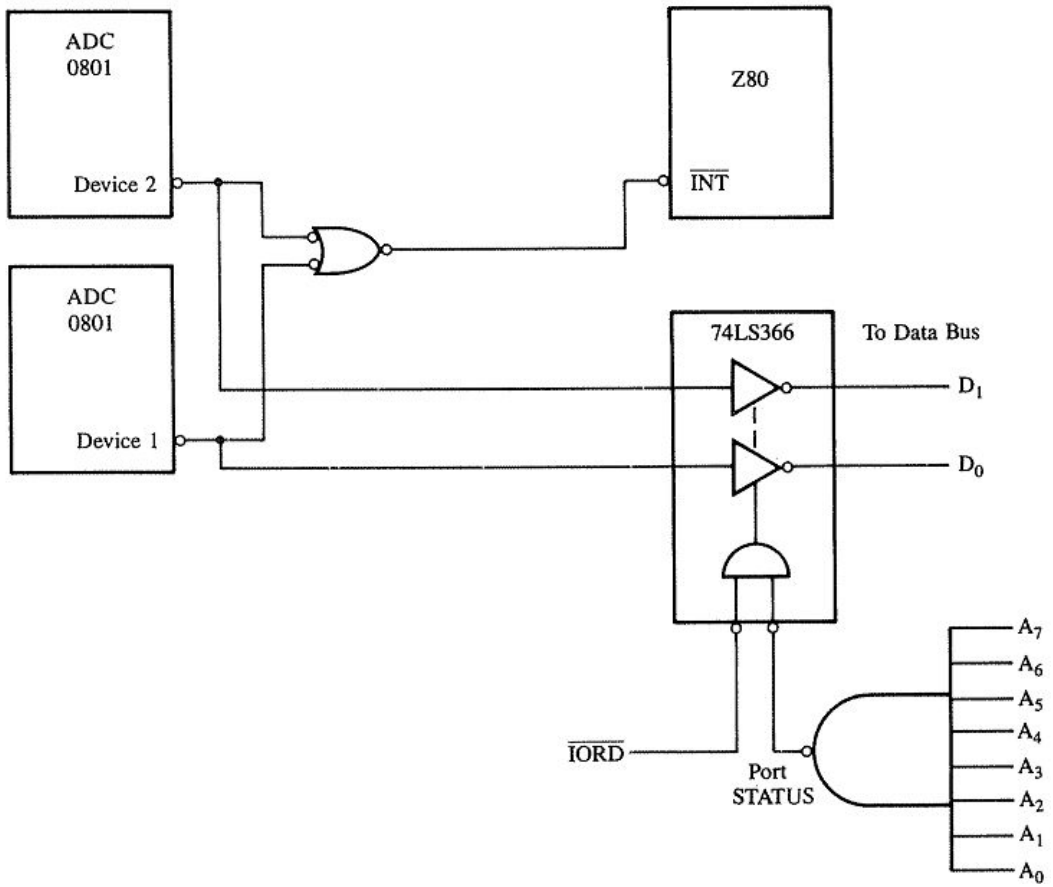
- Z80 has only one maskable interrupt pin (INT)
- What if more than one interrupt source exists?
  - Example: two A/D converters
  - Requests are ORed



- But how can we know who interrupted?

Multiple interrupt sources

- The ISR reads some status ports to find out who the source of interrupt is



Multiple interrupt sources

```
        ;identifying the interrupting converter(s), it reads and stores data
        ;received from the converter(s), and initiates the next conversion.
        PUSH AF             ;Save register contents
        IN A, (STATUS)      ;Read tri-state inverter port
        AND 00000011B       ;Mask data lines D7–D2
        RRA                 ;Place D0 in CY flag
        CALL C, DVICE1      ;If D0 = 1, go to DVICE1 to read data
        RRA                 ;Place D1 in CY flag
        CALL C, DVICE2      ;If D1 = 1, go to DVICE2 to read data
        POP AF              ;Retrieve register contents
        EI                  ;Enable interrupt
        RET
DVICE1: PUSH AF             ;Save interrupt status
        IN A, (ADC1)        ;Read data from Device 1 and turn off INT
        LD (HL), A          ;Save data in memory
        OUT (ADC1), A       ;Start next conversion
        POP AF              ;Retrieve register contents
        RET
DVICE2: PUSH AF             ;Save interrupt status
        IN A, (ADC2)        ;Read data from Device 2 and turn off INT
        LD (DE), A          ;Save data in memory
        OUT (ADC2), A       ;Start next conversion
        POP AF              ;Retrieve register contents
        RET
```