# Computer Architecture

Spring 2020

**Hamed Farbeh**

**farbeh@aut.ac.ir**

Department of Computer Engineering

Amirkabir University of Technology

## BASIC COMPUTER ORGANIZATION AND DESIGN

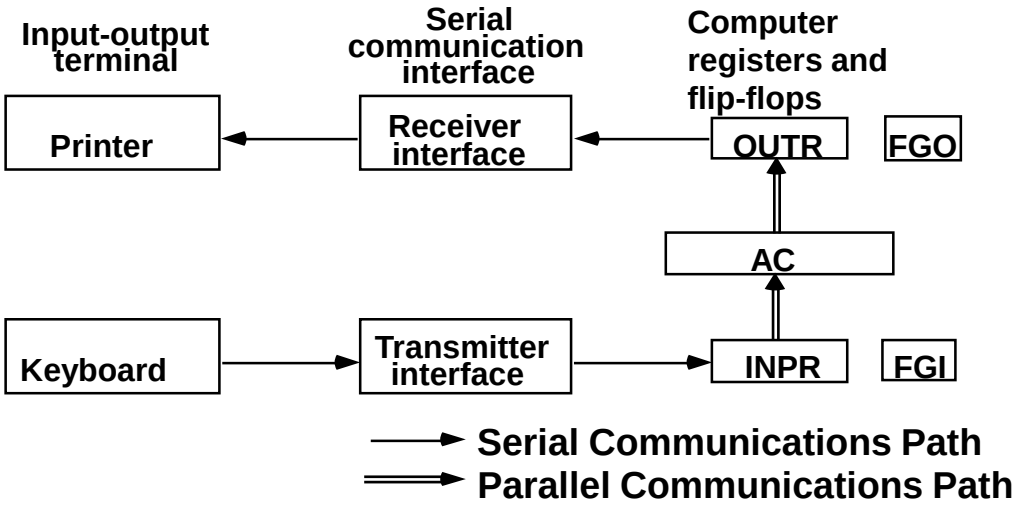
# Outlines

- Instruction Codes
- Computer Registers
- Computer Instructions
- Timing and Control
- Instruction Cycle
- Memory Reference Instructions
- **Input-Output and Interrupt**
- **Complete Computer Description**
- **Design of Basic Computer**
- **Design of Accumulator Logic**

# INPUT-OUTPUT AND INTERRUPT

## A Terminal with a keyboard and a Printer

- **Input-Output Configuration**




| | |
|---|---|
| *INPR* | Input register - 8 bits |
| *OUTR* | Output register - 8 bits |
| *FGI* | Input flag - 1 bit |
| *FGO* | Output flag - 1 bit |
| *IEN* | Interrupt enable - 1 bit |

- The terminal sends and receives serial information
- The serial info. from the keyboard is shifted into INPR
- The serial info. for the printer is stored in the OUTR
- INPR and OUTR communicate with the terminal serially and with the AC in parallel.
- The flags are needed to *synchronize* the timing difference between I/O device and the computer

# PROGRAM CONTROLLED DATA TRANSFER

-- CPU --

/* Input */ /* Initially FGI = 0 */

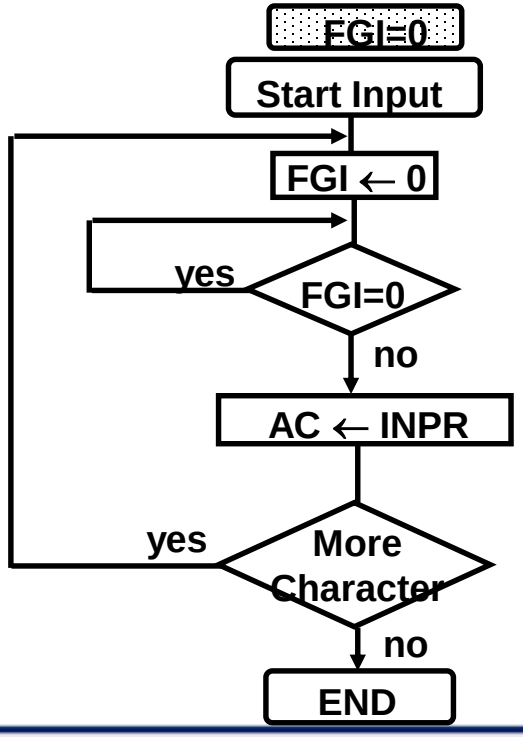
loop: If FGI = 0 goto loop

AC ← INPR, FGI ← 0

/* Output */ /* Initially FGO = 1 */

loop: If FGO = 0 goto loop

OUTR ← AC, FGO ← 0



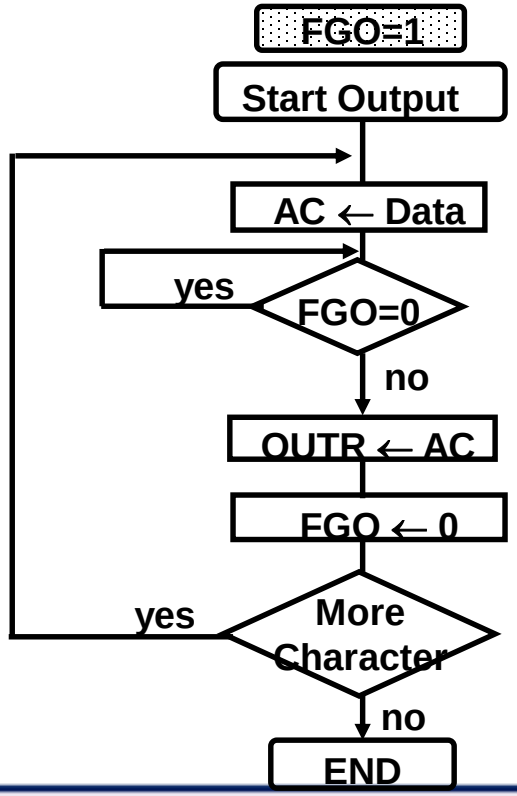

-- I/O Device --

loop: If FGI = 1 goto loop

INPR ← new data, FGI ← 1

loop: If FGO = 1 goto loop

consume OUTR, FGO ← 1



# INPUT-OUTPUT INSTRUCTIONS

$D_7IT_3 = p$
$IR(i) = B_i$, i = 6, …, 11

| | | | |
|---|---|---|---|
| | p: | SC ← 0 | Clear SC |
| INP | $pB_{11}$: | AC(0-7) ← INPR, FGI ← 0 | Input char. to AC |
| OUT | $pB_{10}$: | OUTR ← AC(0-7), FGO ← 0 | Output char. from AC |
| SKI | $pB_9$: | if(FGI = 1) then (PC ← PC + 1) | Skip on input flag |
| SKO | $pB_8$: | if(FGO = 1) then (PC ← PC + 1) | Skip on output flag |
| ION | $pB_7$: | IEN ← 1 | Interrupt enable on |
| IOF | $pB_6$: | IEN ← 0 | Interrupt enable off |

# PROGRAM-CONTROLLED INPUT/OUTPUT

- **Program-controlled I/O**
  - **Continuous CPU involvement**
    **I/O takes valuable CPU time**
  - **CPU slowed down to I/O speed**
  - **Simple**
  - **Least hardware**

**Input**

```
LOOP,   SKI
        BUN   LOOP
        INP
```

**Output**

```
LOOP,   LDA   DATA
LOP,    SKO
        BUN   LOP
        OUT
```

# INTERRUPT INITIATED INPUT/OUTPUT

- Open communication only when some data has to be passed --> *interrupt*.

- The I/O interface, instead of the CPU, monitors the I/O device.

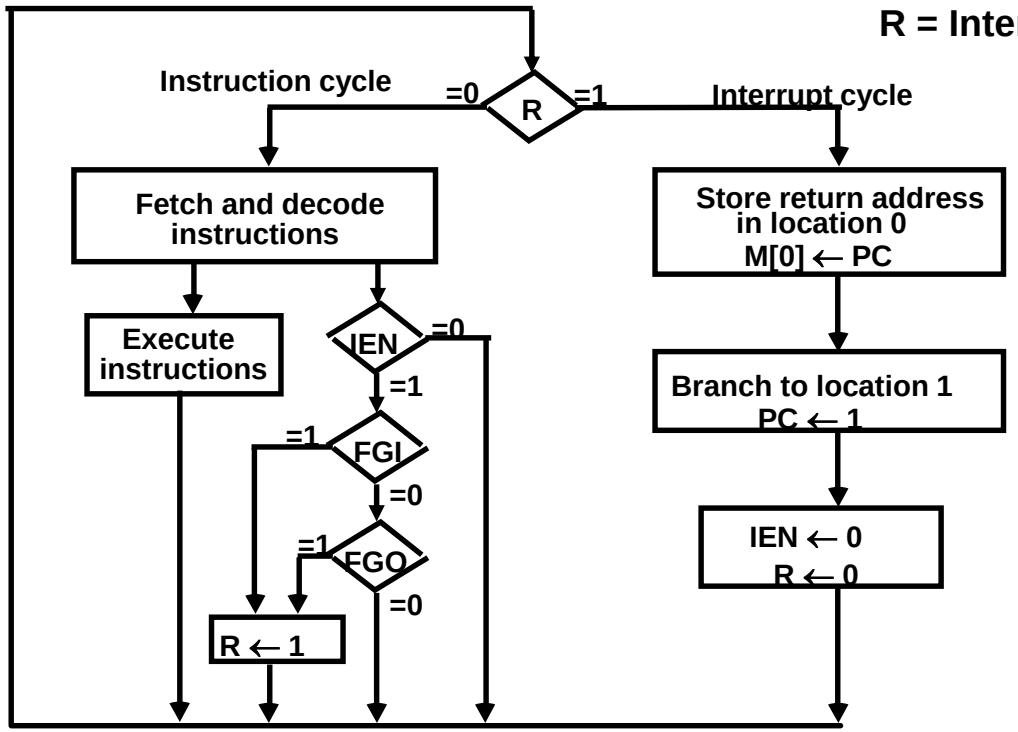
- When the interface founds that the I/O device is ready for data transfer, it generates an interrupt request to the CPU

- Upon detecting an interrupt, the CPU stops momentarily the task it is doing, branches to the service routine to process the data transfer, and then returns to the task it was performing.

* IEN (Interrupt-enable flip-flop)

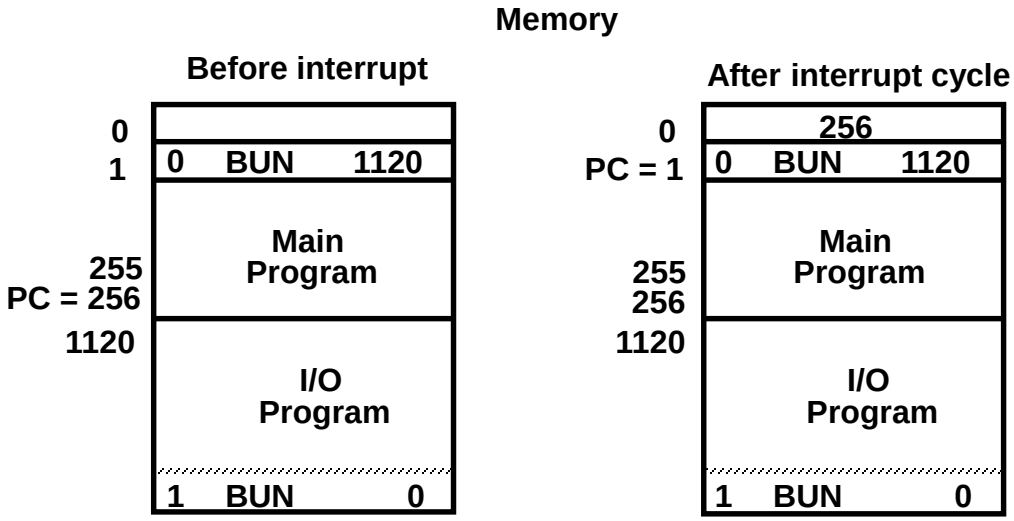
- can be set and cleared by instructions
- when cleared, the computer cannot be interrupted

# FLOWCHART FOR INTERRUPT CYCLE




- The interrupt cycle is a HW implementation of a branch and save return address operation.
- At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1.
- At memory address 1, the programmer must store a branch instruction that sends the control to an interrupt service routine
- The instruction that returns the control to the original program is "indirect BUN 0"

# REG. TRANSFER OPERATIONS IN INTERRUPT CYCLE




**Register Transfer Statements for Interrupt Cycle**

- R F/F ← 1 if IEN (FGI + FGO)$T_0'T_1'T_2'$

  ⇔ $T_0'T_1'T_2'$ (IEN)(FGI + FGO): R ← 1

- The fetch and decode phases of the instruction cycle must be modified ➔Replace $T_0$, $T_1$, $T_2$ with $R'T_0$, $R'T_1$, $R'T_2$
- The interrupt cycle:

  $RT_0$: AR ← 0, TR ← PC

  $RT_1$: M[AR] ← TR, PC ← 0

  $RT_2$: PC ← PC + 1, IEN ← 0, R ← 0, SC ← 0

# FURTHER QUESTIONS ON INTERRUPT

**How can the CPU recognize the device requesting an interrupt?**

**Since different devices are likely to require different interrupt service routines, how can the CPU obtain the starting address of the appropriate routine in each case?**

**Should any device be allowed to interrupt the CPU while another interrupt is being serviced ?**

**How can the situation be handled when two or more interrupt requests occur simultaneously?**

to be continued