Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

(Week 06) Lecture 11-12

INPUT/OUTPUT INSTRUCTIONS

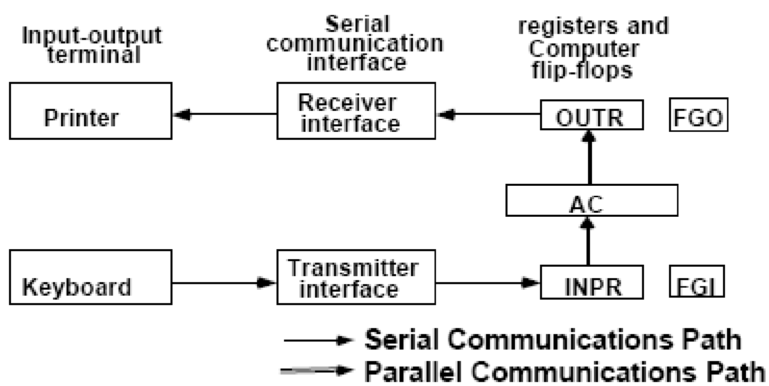
These are the instruction specified by D_7I means when opcode = 111 and I=1. These are instruction deals with I/O devices like keyboard, printers, scanners etc. These are given by the flowing table.

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

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

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

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

INTERRUPTS

It very interesting to note that out I/O devices are very slow in response while the CPU's response time is in MICRON. So, if the CPU waits for the I/O device to get ready or send/receive some data, it has to wait a long and also in the mean while it remains idol. So, to get rid of this situation we introduce a routine called *interrupt.* As its name is concerned is a knock to the CPU door, after receiving interrupt enable signal by IEN flag, CPU handles the interrupt and again get busy in other tasks.

INTERRUPT CYCLE

From the occurrence of an interrupt to its end called interrupt cycle. It obviously contains the same steps we already have been discussed. Mainly from invoking an interrupt, invoking the CPU for interrupt, handling the interrupt and returning back the control.

It can further be categorized as following.

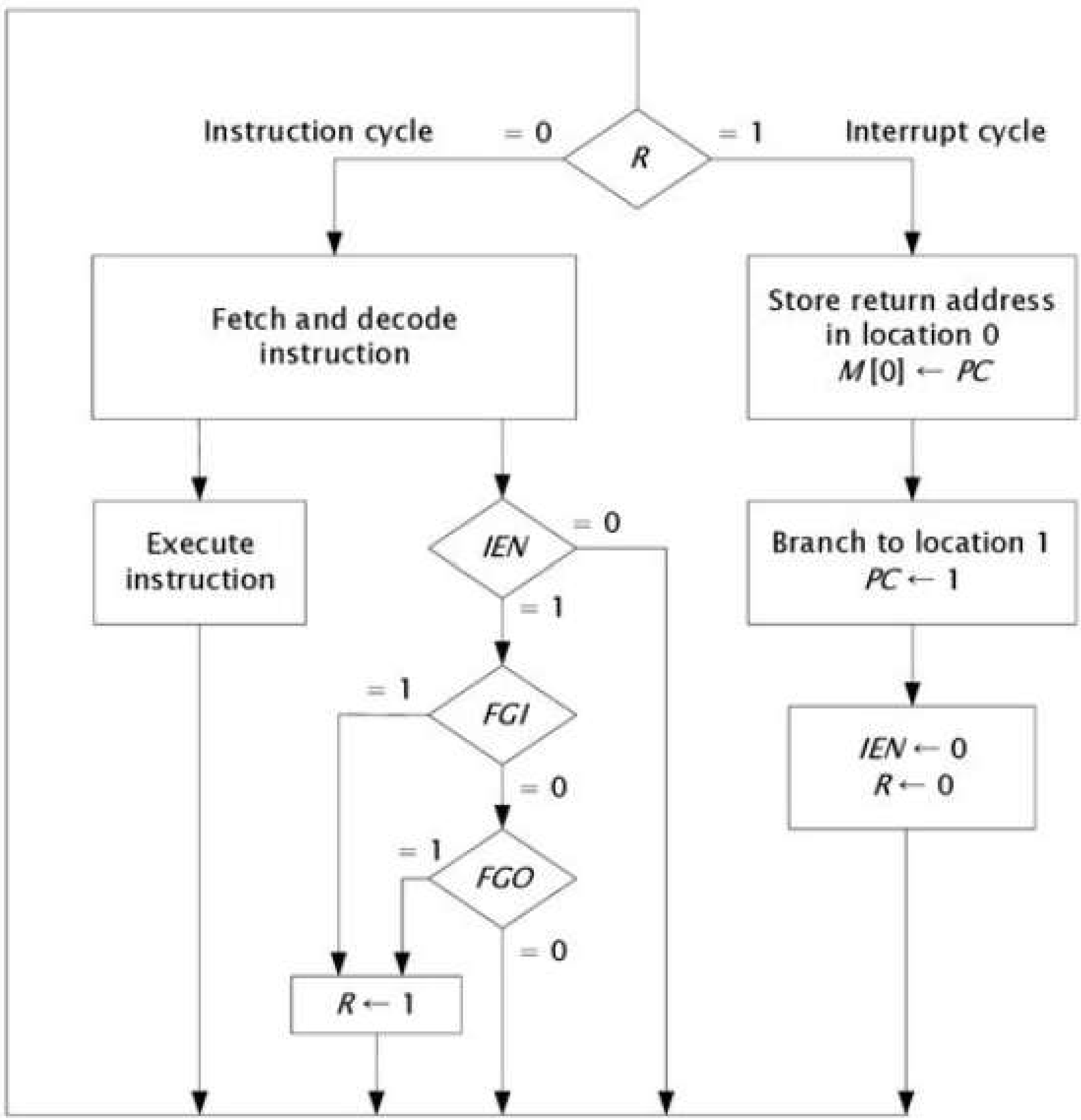
- Open communication only when some data has to be passed --> *interrupt*.
- The I/O interface, instead of the CPU, monitors the I/O device.
- When the interface finds that the I/O device is ready for data transfer, it generates an interrupt request to the CPU
- Upon detecting an interrupt, the CPU stops 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
- 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

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

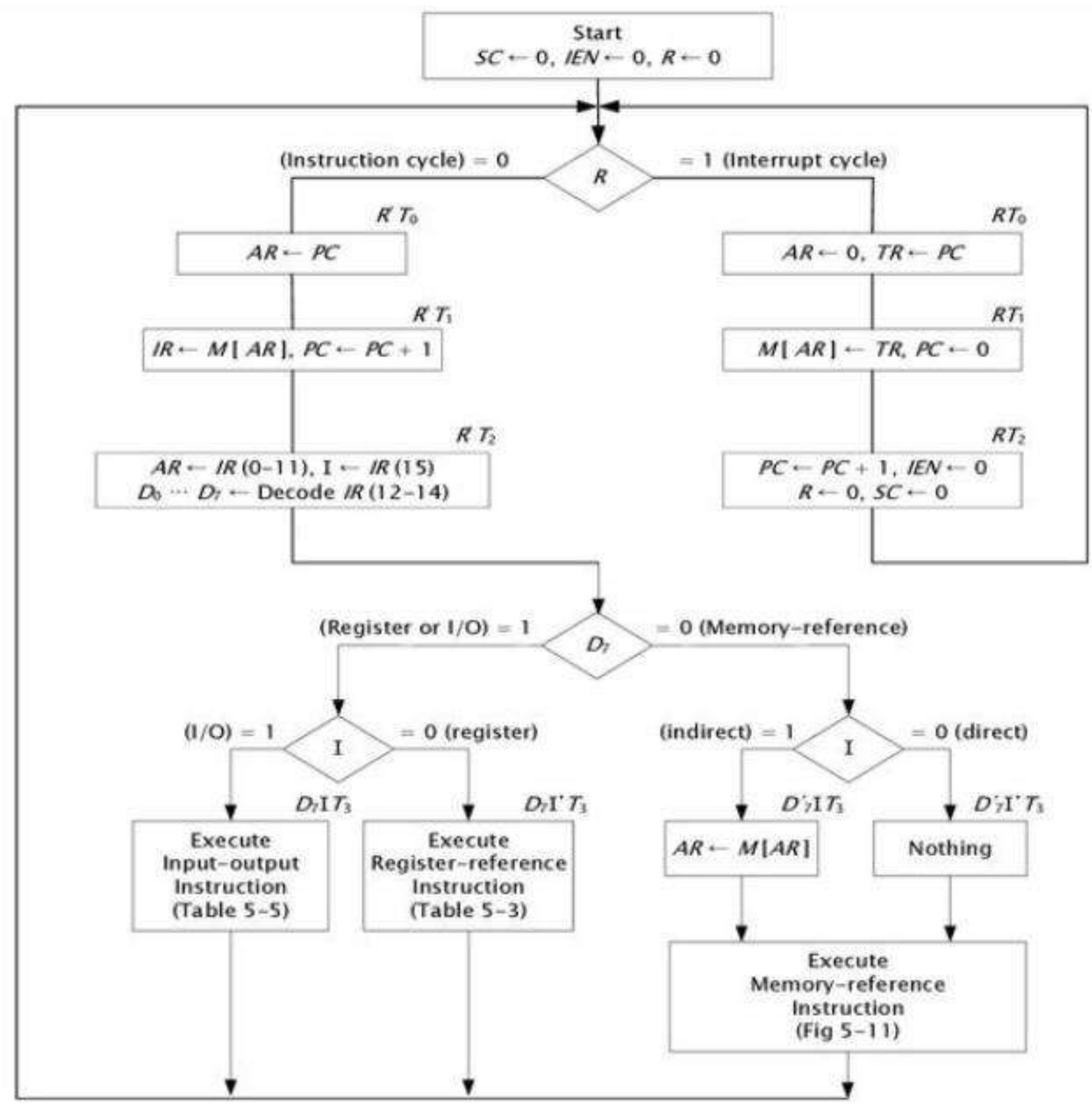
- The instruction that returns the control to the original program is 'indirect BUN 0'



Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

COMPLETE COMPUTER DESCRIPTION

After discussing each and every component explicitly, now we are able to put all the things together to build a complete basic computer. Following is the complete schema for the said scenario.



This was complete flowchart of operations. Now the micro-operations associated with each step are discussed at the next page.

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

- **The way that the interrupt** is handled by the computer can be explained by means of the flowchart of Fig. above. An interrupt flip-flop R is included in the computer. When R = 0, the computer goes through an instruction cycle.
- **During the execute phase of the instruction cycle IEN** is checked by the control. If it is 0, it indicates that the programmer does not want to use the interrupt, so control continues with the next instruction cycle. If IEN is 1, control checks the flag bits.
- **If both flags are 0**, it indicates that neither the input nor the output registers are ready for transfer of information. In this case, control continues with the next instruction cycle. If either flag is set to 1 while IEN = 1, flip-flop R is set to 1.
- **At the end of the execute phase**, control checks the value of R, and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle. The interrupt cycle is a hardware implementation of a branch and save return address operation.
- **The return address** available in PC is stored in a specific location where it can be found later when the program returns to the instruction at which it was interrupted.
- **This location may be a processor register**, a memory stack, or a specific memory location. Here we choose the memory location at address 0 as the place for storing the return address. Control then inserts address 1 into PC and clears IEN and R so that no more interruptions can occur until the interrupt request from the flag has been serviced.
- **An example** that shows what happens during the interrupt cycle is shown in Fig. below. Suppose that an interrupt occurs and R is set to 1 while the control is executing the instruction at address 255. At this time, the return address 256 is in PC.
- **The programmer** has previously placed an input-output service program in memory starting from address 1120 and a BUN 1120 instruction at address 1. This is shown in Part (a) in Fig. below. When control reaches timing signal T_0 and finds that R = 1, it proceeds with the interrupt cycle.
- **The content of PC (256)** is stored in memory location 0, PC is set to 1, and R is cleared to 0. At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1 since this is the content of PC.
- **The branch instruction** at address 1 causes the program to transfer to the input-output service program at address 1120. This program checks the flags, determines which flag is set, and then transfers the required input or output information.
- **Once this is done**, the instruction ION is executed to set IEN to 1 (to enable further interrupts), and the program returns to the location where it was interrupted. This is shown in Part (b) in Fig. below.
- **The instruction** that returns the computer to the original place in the main program is a branch indirect instruction with an address part of 0. This instruction is placed at the end of the UO service program.
- **After** this instruction is read from memory during the fetch phase, control goes to the indirect phase (because I = 1) to read the effective address.

Mr. Abdul Rehman **email id: abdul@northern.edu.pk** **Whatsapp# 0308-7792217**

- **The effective address** is in location 0 and is the return address that was stored there during the previous interrupt cycle. The execution of the indirect BUN instruction results in placing into PC the return address from location 0.

Fetch	R'T0:	AR <- PC
	R'T1:	IR <- M[AR], PC <- PC + 1
Decode	R'T2:	D0, ..., D7 <- Decode IR(12 ~ 14), AR <- IR(0 ~ 11), I <- IR(15)
Indirect	D7'IT3:	AR <- M[AR]
Interrupt		
T0'T1'T2'(IEN)(FGI + FGO):		R <- 1
	RT0:	AR <- 0, TR <- PC
	RT1:	M[AR] <- TR, PC <- 0
	RT2:	PC <- PC + 1, IEN <- 0, R <- 0, SC <- 0
Memory-Reference		
AND	D0T4:	DR <- M[AR]
	D0T5:	AC <- AC . DR, SC <- 0
ADD	D1T4:	DR <- M[AR]
	D1T5:	AC <- AC + DR, E <- Cout, SC <- 0
LDA	D2T4:	DR <- M[AR]
	D2T5:	AC <- DR, SC <- 0
STA	D3T4:	M[AR] <- AC, SC <- 0
BUN	D4T4:	PC <- AR, SC <- 0
BSA	D5T4:	M[AR] <- PC, AR <- AR + 1
	D5T5:	PC <- AR, SC <- 0
ISZ	D6T4:	DR <- M[AR]
	D6T5:	DR <- DR + 1
	D6T6:	M[AR] <- DR, if(DR=0) then (PC <- PC + 1), SC <- 0
Register-Reference		
	D7I'T3 = r	(Common to all register-reference instr)
	IR(i) = Bi	(i = 0,1,2, ..., 11)
	r:	SC <- 0
CLA	rB11:	AC <- 0
CLE	rB10:	E <- 0
CMA	rB9:	AC <- AC'
CME	rB8:	E <- E'
CIR	rB7:	AC <- shr AC, AC(15) <- E, E <- AC(0)
CIL	rB6:	AC <- shl AC, AC(0) <- E, E <- AC(15)
INC	rB5:	AC <- AC + 1
SPA	rB4:	If(AC(15) =0) then (PC <- PC + 1)
SNA	rB3:	If(AC(15) =1) then (PC <- PC + 1)
SZA	rB2:	If(AC = 0) then (PC <- PC + 1)
SZE	rB1:	If(E=0) then (PC <- PC + 1)
HLT	rB0:	S <- 0
Input-Output	D7IT3 = p	(Common to all input-output instructions)
	IR(i) = Bi	(i = 6,7,8,9,10,11)
	p:	SC <- 0
INP	pB11:	AC(0-7) <- INPR, FGI <- 0
OUT	pB10:	OUTR <- AC(0-7), FGO <- 0
SKI	pB9:	If(FGI=1) then (PC <- PC + 1)
SKO	pB8:	If(FGO=1) then (PC <- PC + 1)
ION	pB7:	IEN <- 1
IOF	pB6:	IEN <- 0