Announcements

- Reading for this week: finish Chapter 5
- Homework #6 is due on Thursday
- Midterm #2 is Thursday 11/9
- Last day for midterm #1 regrades is today (10/31)

Happy Halloween!

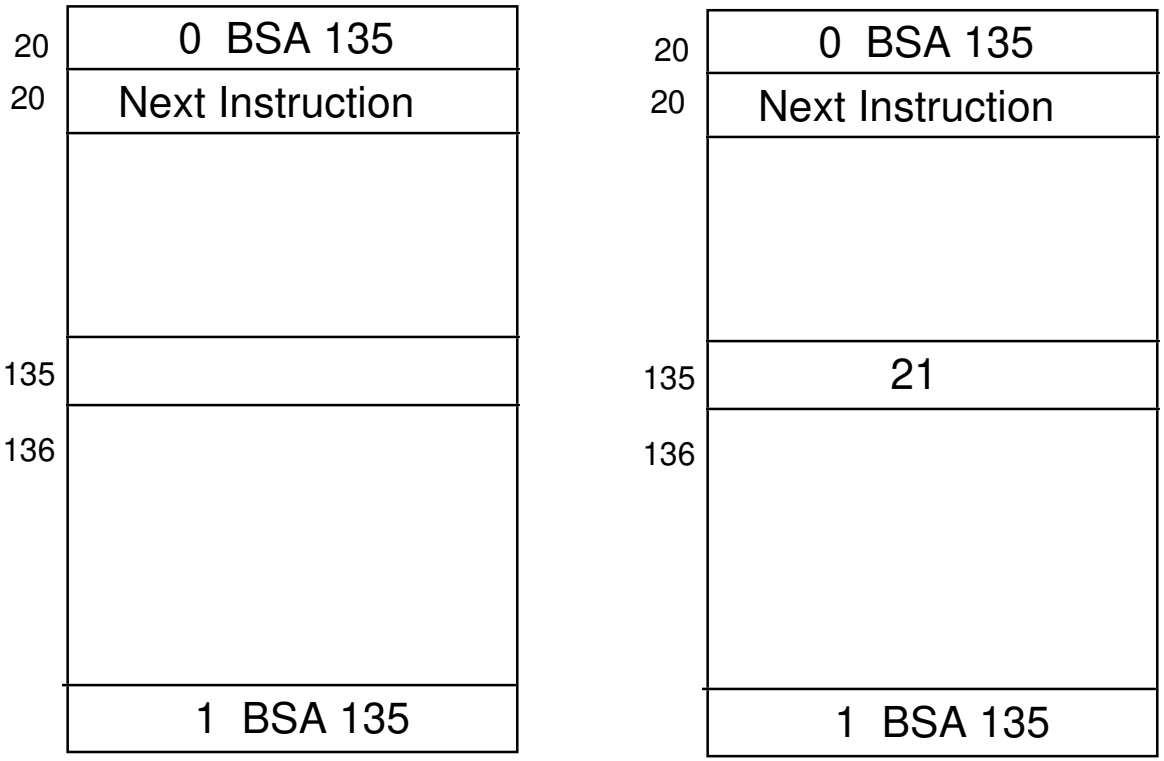
Memory Reference Instructions

- Memory instructions may take more than one clock

AND	D0T4	DR <- M[AR]
	D0T5	AC <- AC ^ DR, SC <- 0
ADD	D1T4	DR <- M[AR]
	D1T5	AC <- AC ^ DR, 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) PC <- PC + 1, SC <- 0

Subroutine Calls

- BSA Instruction can be used to call a subroutine
- BUN Indirect is used to return



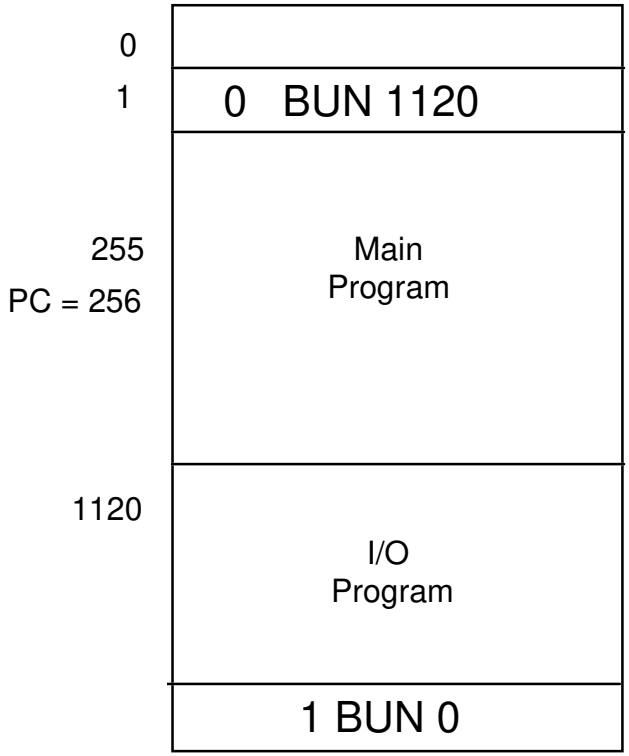
Input-Output and Interrupt

- Input
  - input register (8 bits)
  - FGI (input control FF)
    - set to one when input becomes available
    - cleared when the input register is read
- Output
  - output register (8 bits)
  - FGO (output control FF)
    - set to one when computer is started
    - set to zero when output register is loaded by computer
    - set to one when output is read by external device

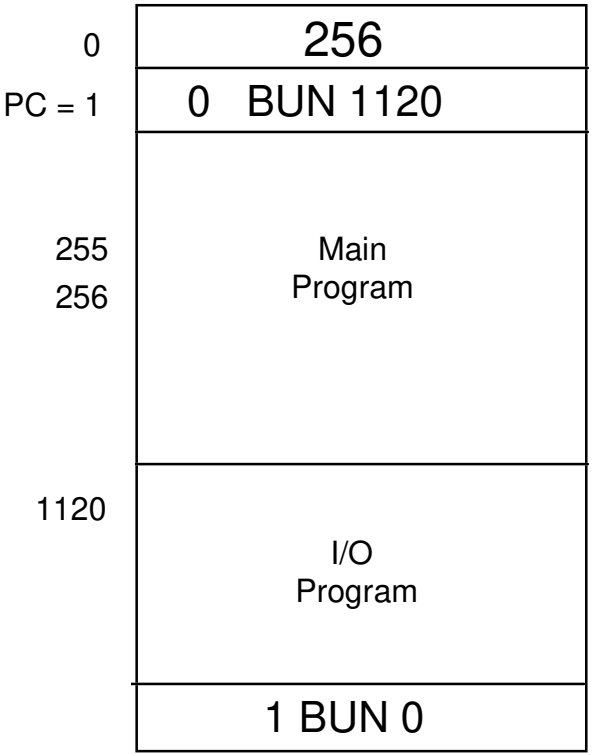
Interrupts

- Computers are faster than I/O devices
- Processor must wait for the output to be done before the next character can be sent
  - we can either keep checking if the device is ready
    - called polling
  - have the device tell us when it is done
    - called interrupt
- How can the device indicate to the program its done?
  - answer: change the program's execution
  - also need to save information about where we were when the interrupt arrived
    - this is sort of like a hardware procedure call

Interrupt Execution



Before Interrupt



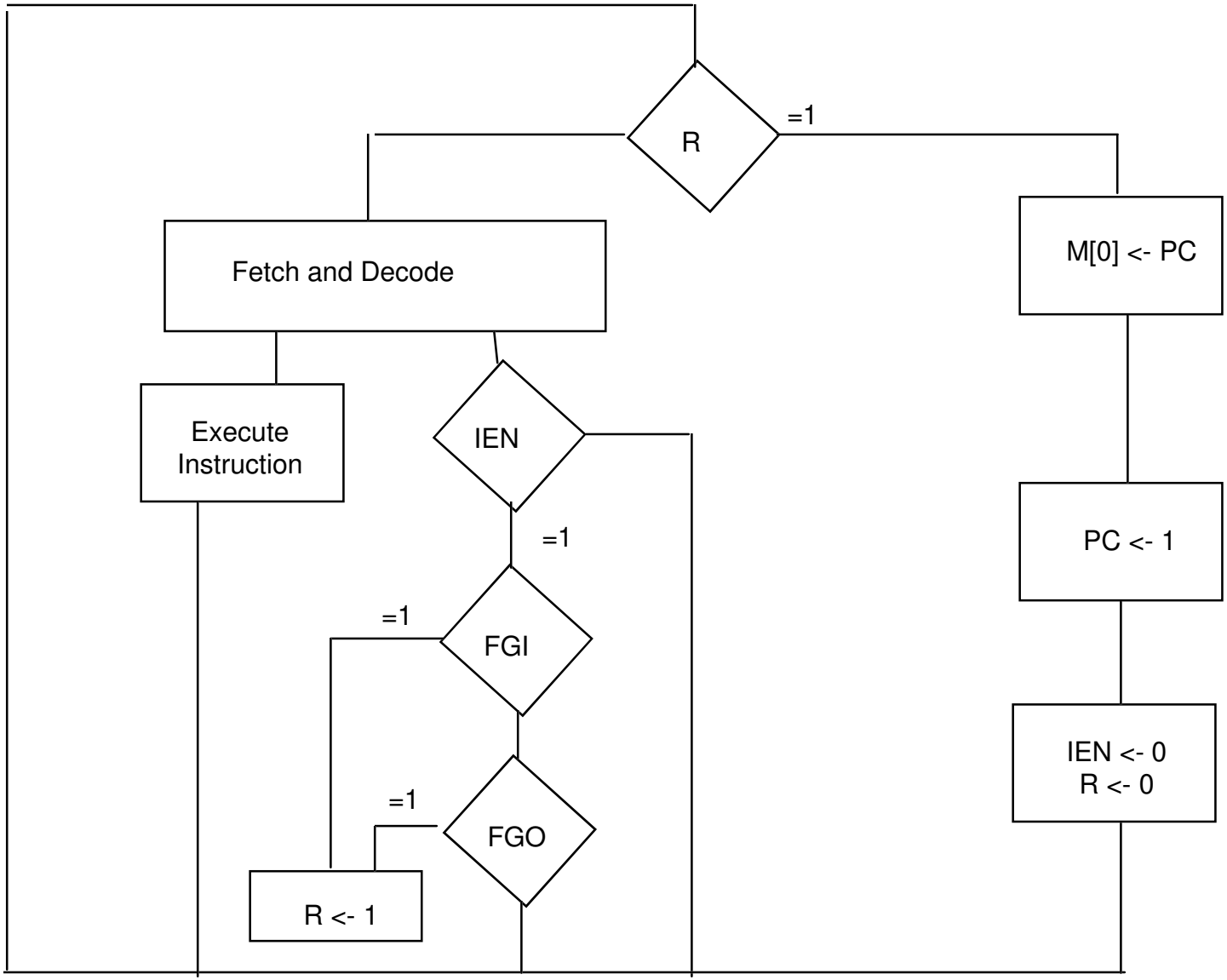
After Interrupt

Input-Output Instructions

- $D_7IT_3 = p$
- IR(I) = Bi

	P:	SC ← 0
INP	pB_{11}:	AC(0-7) ← INPR, FGI ← 0
OUT	pB_{10}:	OUTR ← AC(0-7), FGO ← 0
SKI	pB_9:	if (FGI = 1) PC ← PC + 1
SKO	pB_8:	if (FGO = 1) PC ← PC + 1
ION	pB_7:	IEN ← 1
IOF	pB_6:	IEN ← 0

Interrupts


R
=1
Fetch and Decode
M[0] <- PC
Execute Instruction
IEN
=1
PC <- 1
=1
FGI
IEN <- 0
R <- 0
=1
FGO
R <- 1

Interrupt Cycle

- Setting the Flag
  - $T_0'\ T_1'\ T_2'$ (IEN) (FGI + FGO): R <- 1
- Interrupt Cycle
  - R T_0: AR <- 0, TR <- PC
  - R T_1: M[AR] <- TR, PC <- 0
  - R T_2: PC <- PC + 1, IEN <- 0, R <- 0, SC <- 0
- Modified Fetch and Execute
  - R' T_0: AR <- PC
  - R' T_1: IR <- M[AR], PC <- PC + 1
  - R' T_2: $D_0,\ldots D_7$ <- Decode(IR(12-14)), AR <- IR(0-11), I<-IR(15)

Change

Questions About Interrupt Cycle

- Why can't we set the flag during T_0, T_1, or T_2?
  - Answer: It could interfere with the interrupt cycle which uses R as a flag to decide if we are executing a normal instruction.
- Why not only set the flag on T_3?
  - Answer: We want to detect an interrupt as soon as possible rather than waiting a full instruction cycle.

What is Required to Build the Computer?

- A memory with 4096 words each 16 bits wide
- Nine registers:
  - AR, PC, DR, AC, IR, TR, OUTR, INPR, SC
- Seven Flip-flops: I,S,E,R,IEN,FGI,FGO
- Two decoders:
  - 3x8 and 4x16
- a 16 bit bus
- control logic gates
- added and logic unit

Control Logic Gates

- To make the computer work we need control logic to:
  - control inputs of nine registers (load, clear, increment)
  - control read and write of memory
  - signals to set clear or complement flip-flops
  - signals to control bus (S_2,S_1,S_0)
  - signals to control the ALU
- To design control logic for one component
  - extract all of the micro-instructions that use the component
  - use the guard conditions to figure out the required logic