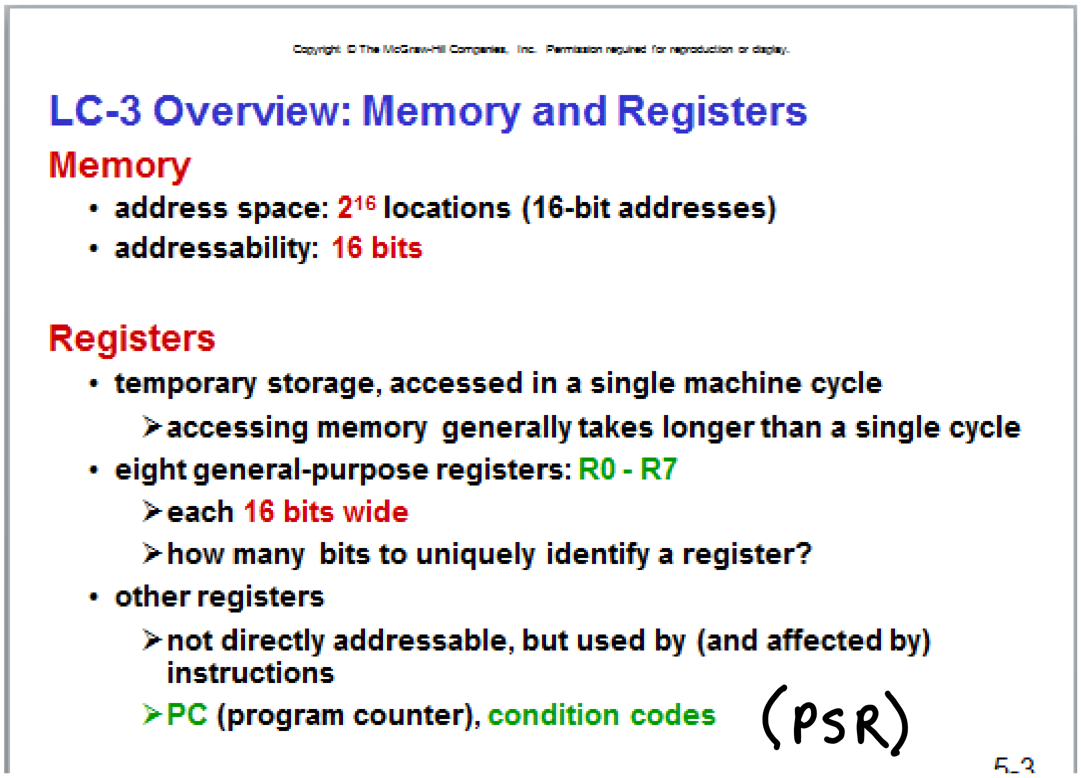
LC3 Intro/Review



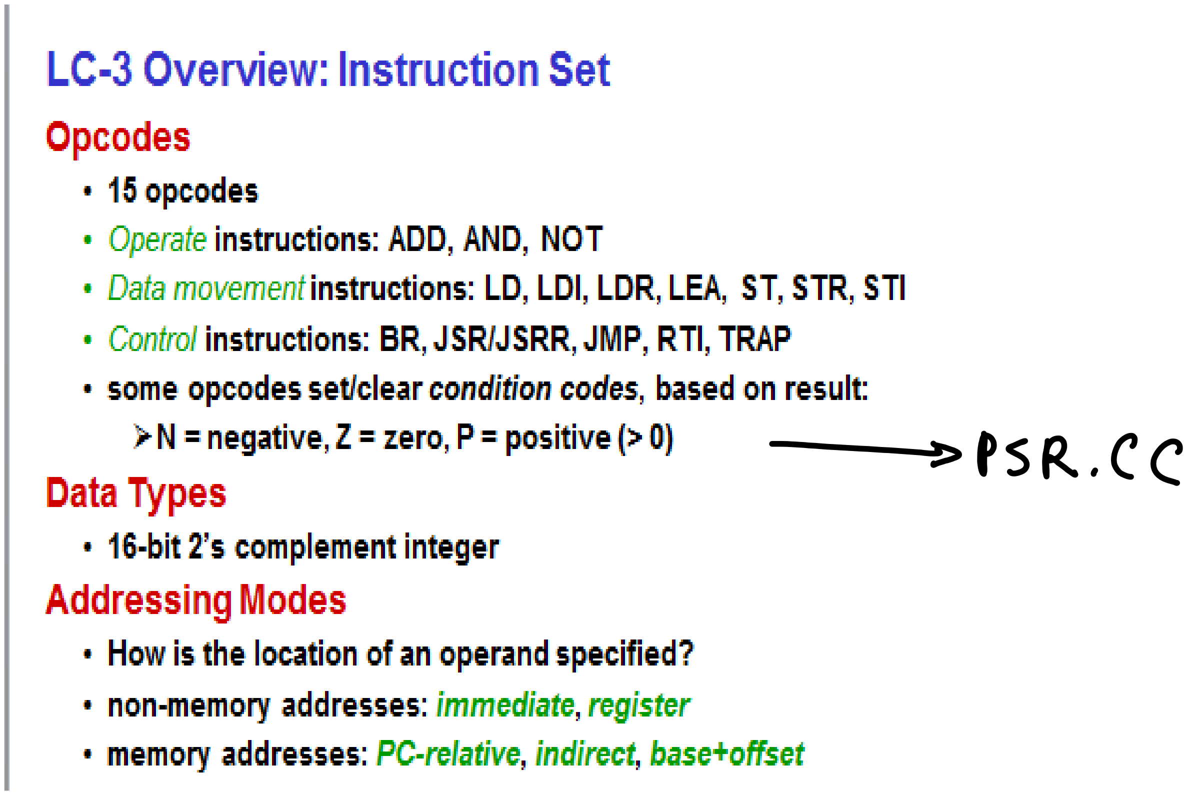
LC-3 Overview: Memory and Registers

Memory

- address space: 2^{16} locations (16-bit addresses)
- addressability: 16 bits

Registers

- temporary storage, accessed in a single machine cycle
  - accessing memory generally takes longer than a single cycle
- eight general-purpose registers: R0 - R7
  - each 16 bits wide
  - how many bits to uniquely identify a register?
- other registers
  - not directly addressable, but used by (and affected by) instructions
  - PC (program counter), condition codes (PSR)



LC-3 Overview: Instruction Set

Opcodes

- 15 opcodes
- *Operate* instructions: ADD, AND, NOT
- *Data movement* instructions: LD, LDI, LDR, LEA, ST, STR, STI
- *Control* instructions: BR, JSR/JSRR, JMP, RTI, TRAP
- some opcodes set/clear *condition codes*, based on result:
  - N = negative, Z = zero, P = positive (> 0) $\longrightarrow$ PSR.CC

Data Types

- 16-bit 2's complement integer

Addressing Modes

- How is the location of an operand specified?
- non-memory addresses: *immediate*, *register*
- memory addresses: *PC-relative*, *indirect*, *base+offset*

NOT (Register)

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NOT	1	0	0	1	Dst			Src			1	1	1	1	1	1
					DR			SR1								

ADD/AND (Register)

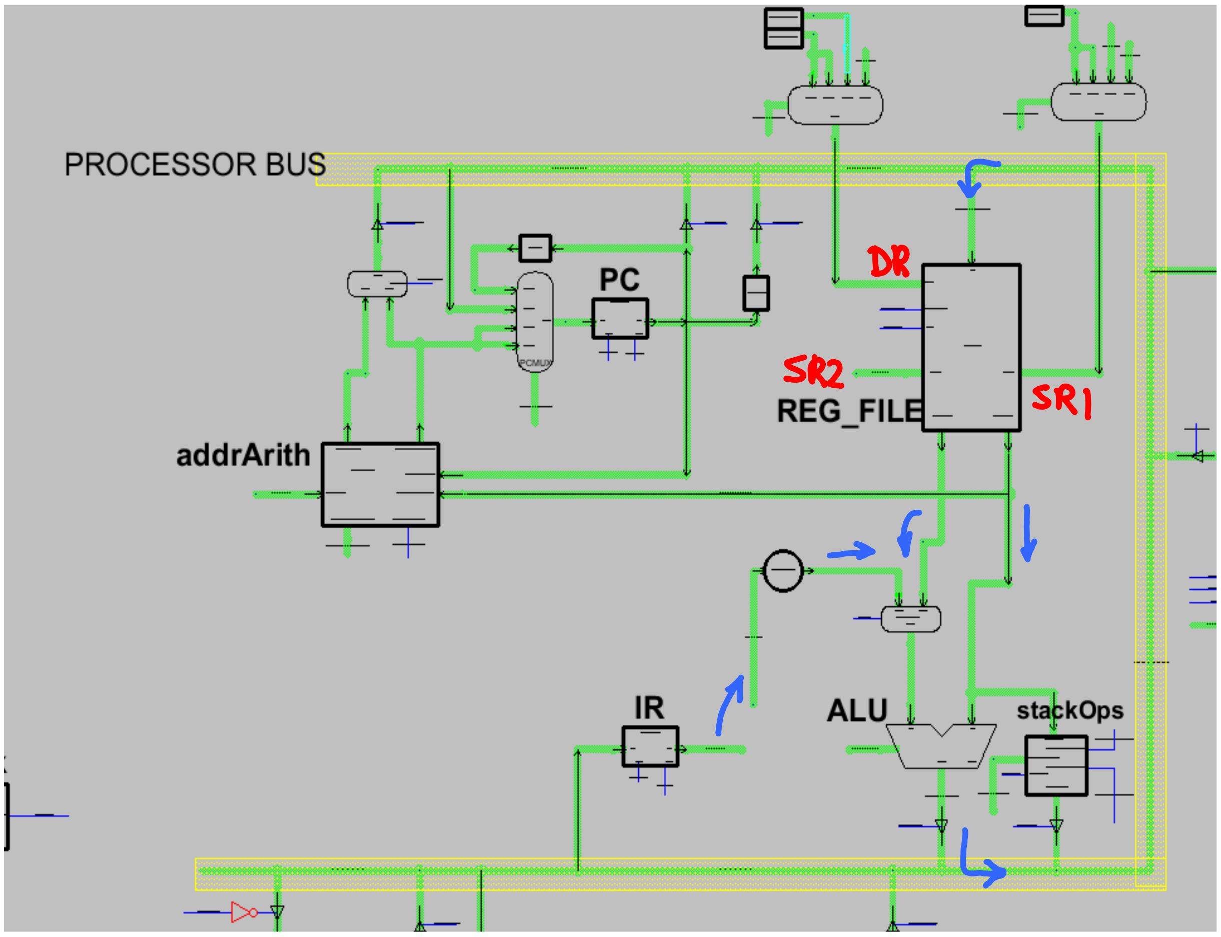
this zero means "register mode"

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0	0	0	1	Dst			Src1			0	0	0	Src2		
AND	0	1	0	1	Dst			Src1			0	0	0	Src2		
					DR			SR1						SR2		

ADD/AND (Immediate)

this one means "immediate mode"

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0	0	0	1	Dst			Src1			1	Imm5				
AND	0	1	0	1	Dst			Src1			1	Imm5				



LD (PC-Relative)

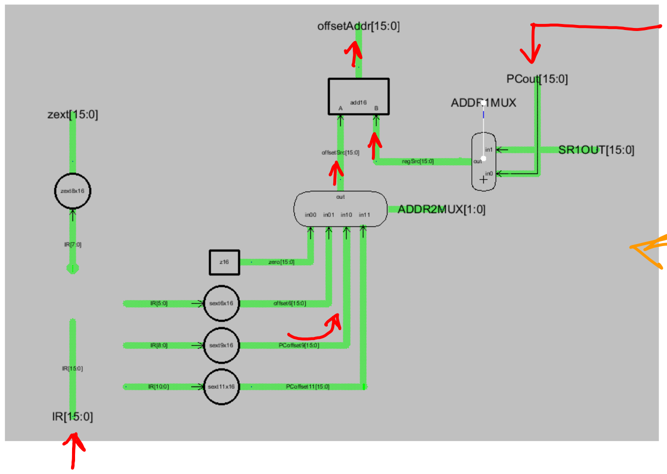
	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
LD	0 0 1 0	Dst	PCoffset9

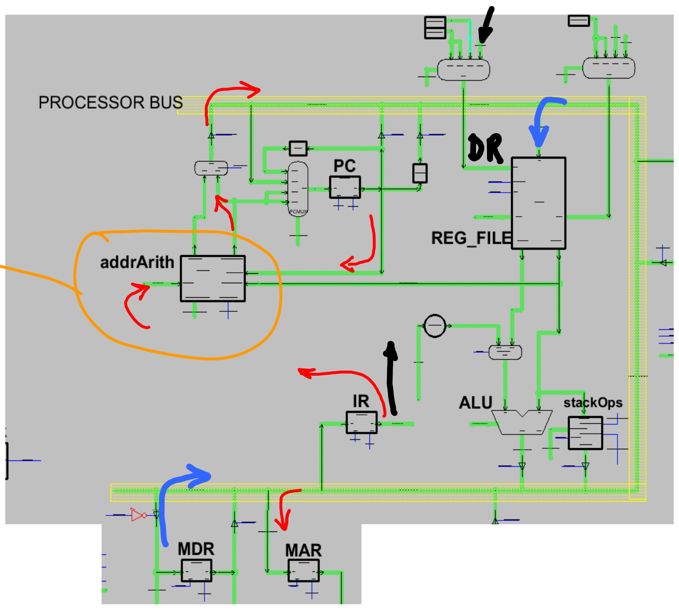
DR

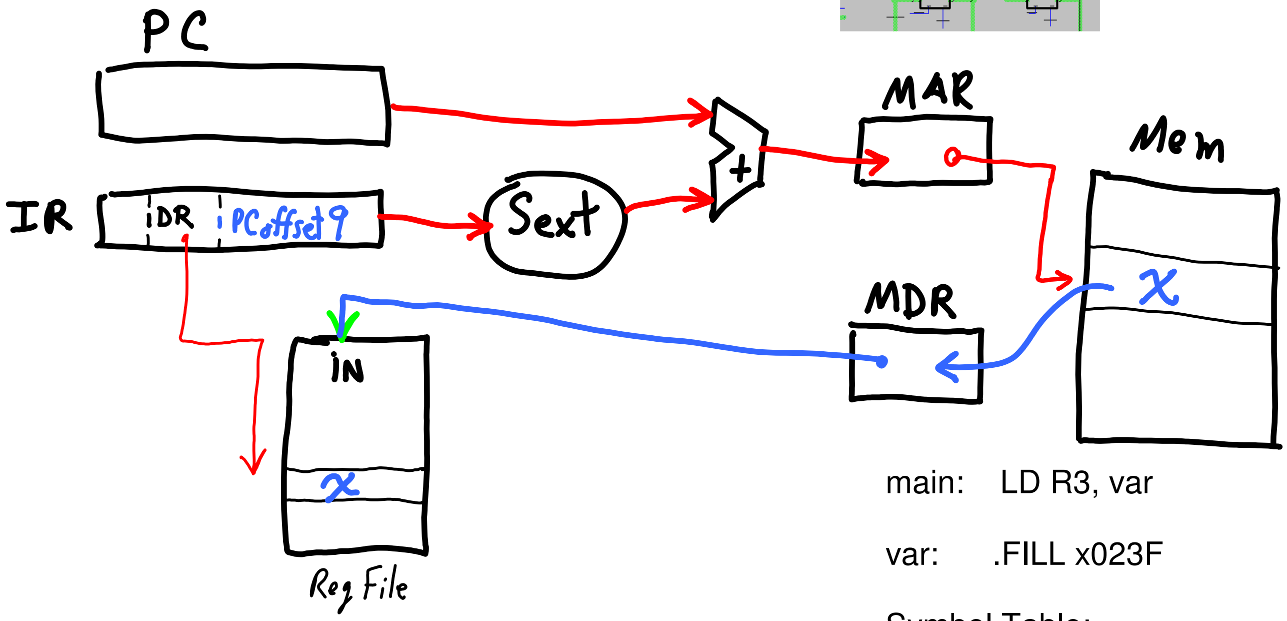
ST (PC-Relative)

	15 14 13 12	11 10 9	8 7 6 5 4 3 2 1 0
ST	0 0 1 1	Src	PCoffset9

SR







main: LD R3, var

var: .FILL x023F

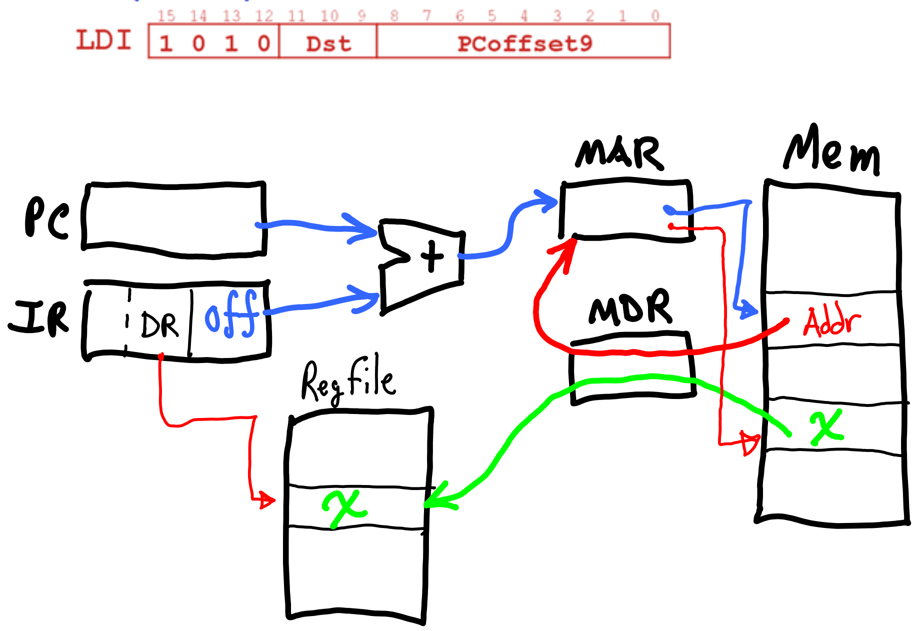
Symbol Table:

"main"	x0200
"var"	x0208

Memory

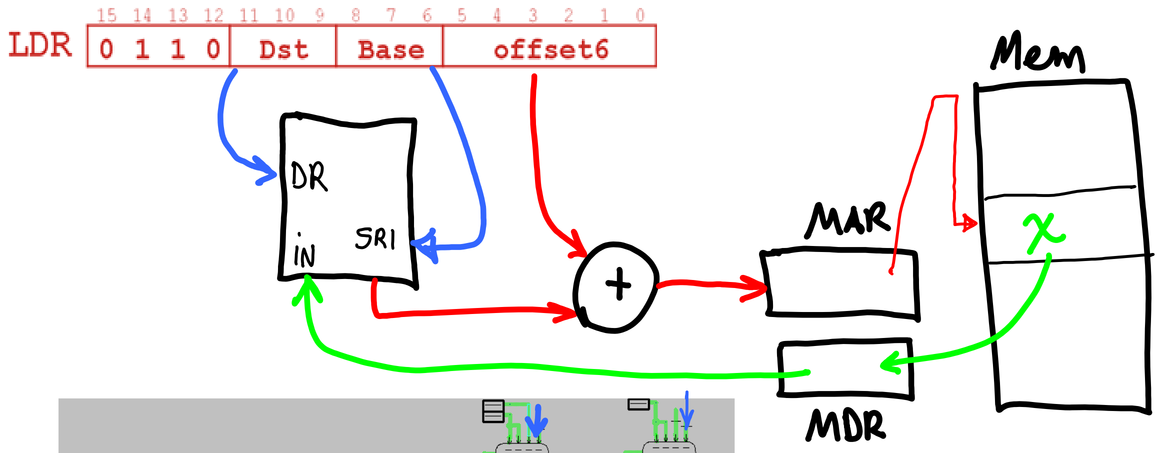
```
-----------------------------
0200:    0010 011 000000111

0208:    0000 0010 0011 1111
```

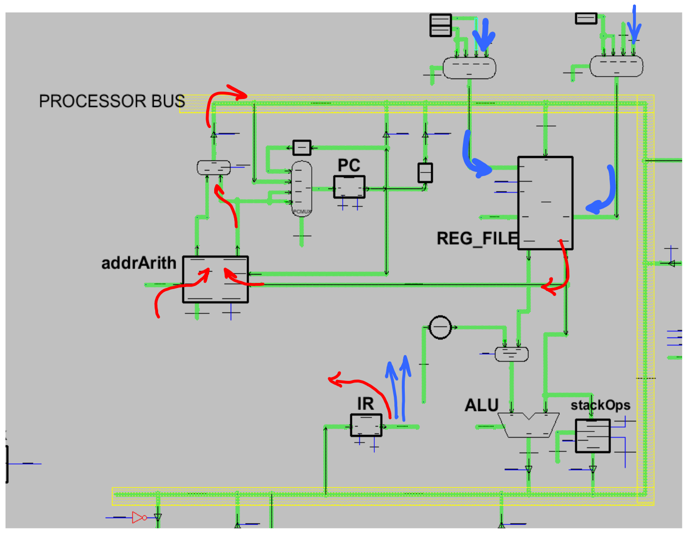
LDI (Indirect)



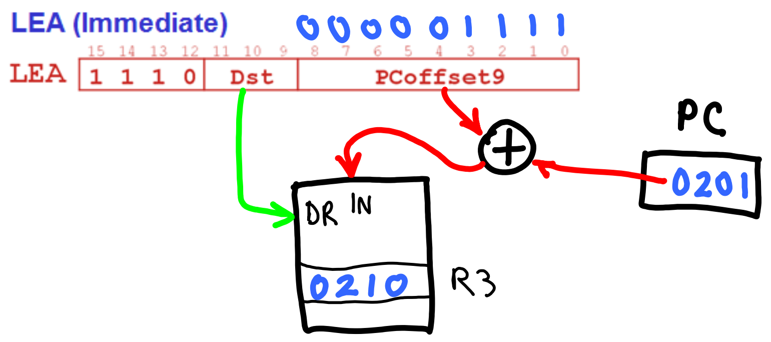
```
main:      LDI R3, dataPtr

dataPtr:   .FILL var

var:       .FILL x0000


Symbol table:
    "main"       x0200
    "dataPtr"    x0210
    "var:        x1234

Memory
---------------------------
0200:   1010 011 000001111

0210:    0001 0010 0011 0100

1234:    0000 0000 0000 0000
```

LDR (Base+Offset)





LEA (Immediate)



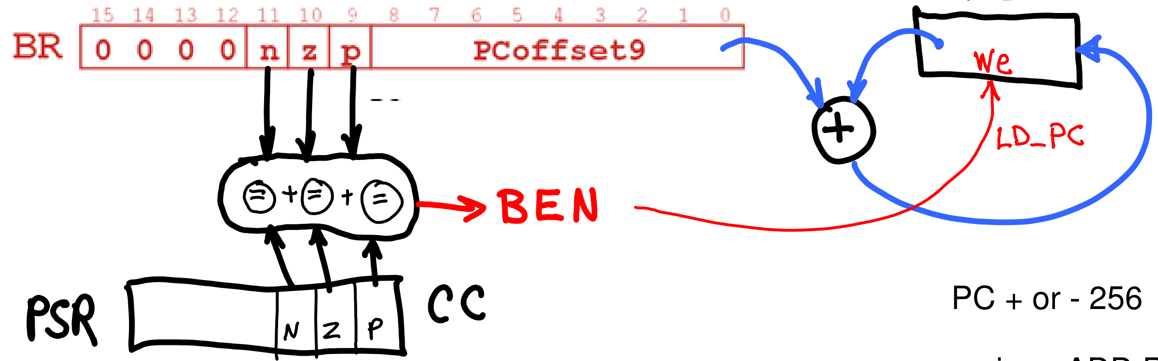
main: LEA R3, array
...
array: .BLKW 100

Symbol Table:
"main"   x0200
"array"   x0210

Memory

0200:   1110 011 000001111
...
0210:   ????
0211:   ????
....

BR (PC-Relative)



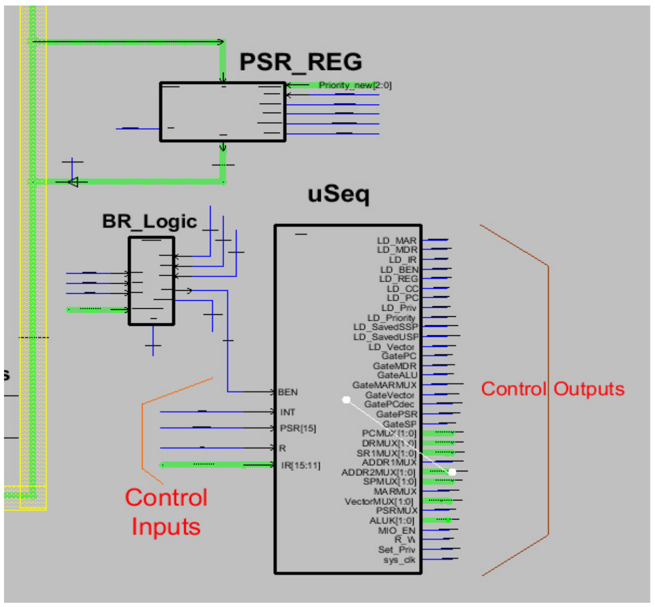
PC + or - 256

main: ADD R0, R0, 1
            BRp main

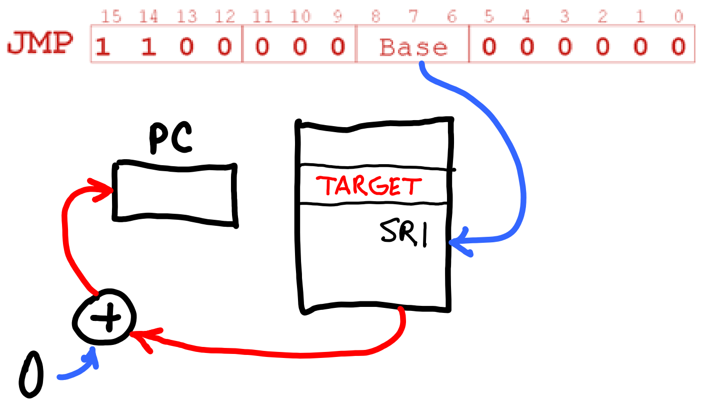
Symbol Table:
"main" x0200

Memory

0200: 0000 001 111111110

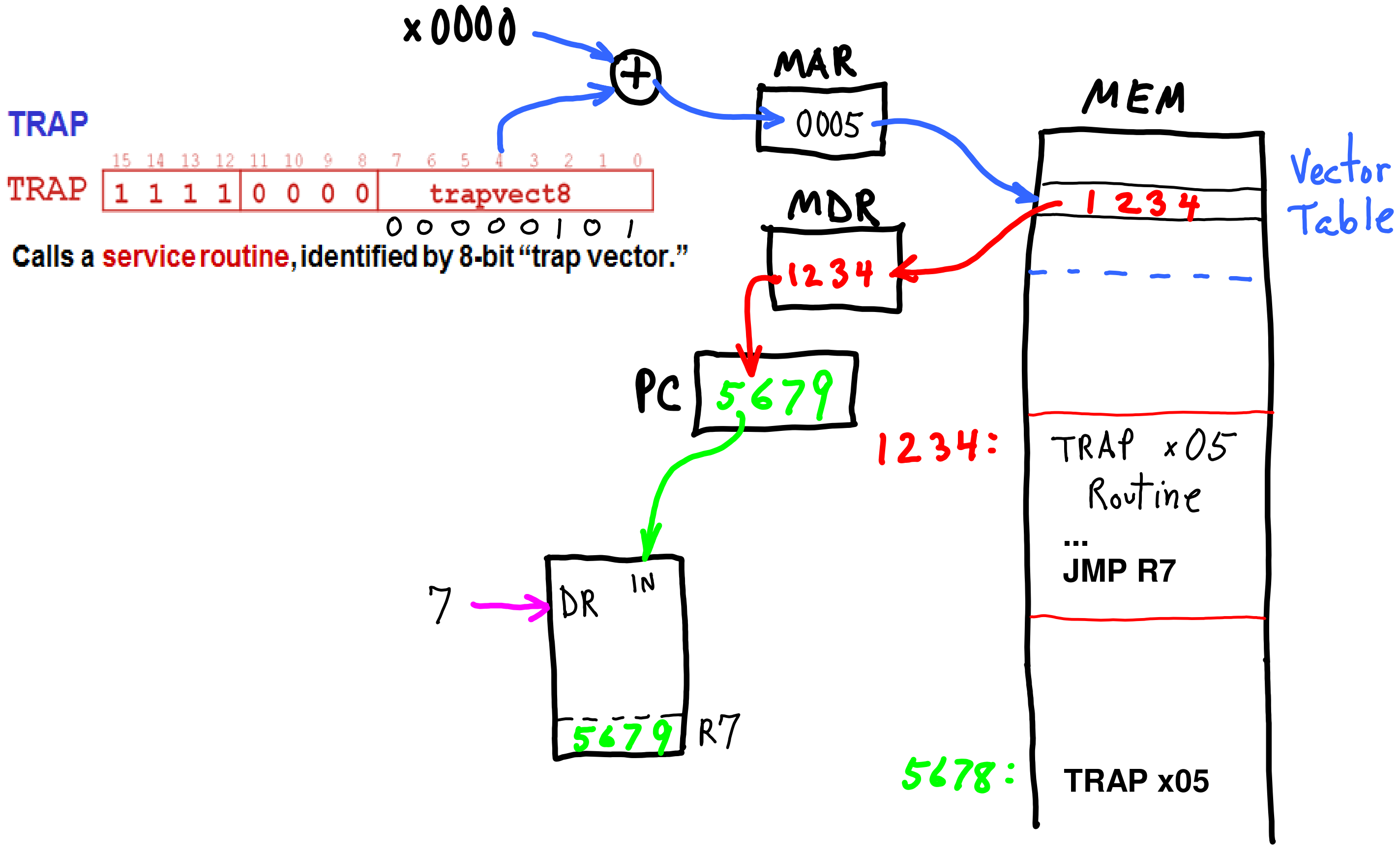


- **only certain instructions set the codes**
  (ADD, AND, NOT, LD, LDI, LDR, LEA)



```
main:    LEA R7, next
next:     ADD R0, R1, #11
...
foo:     ADD R0, R1, #10
         JMP R7
```

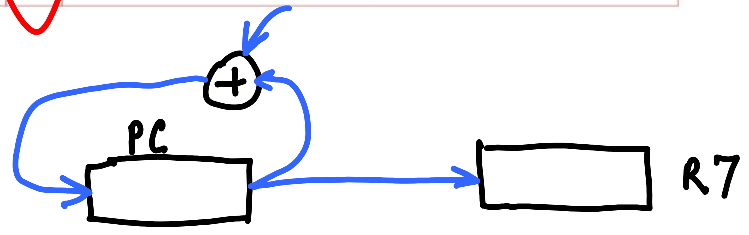
TRAP



Calls a service routine, identified by 8-bit "trap vector."

JSR Instruction

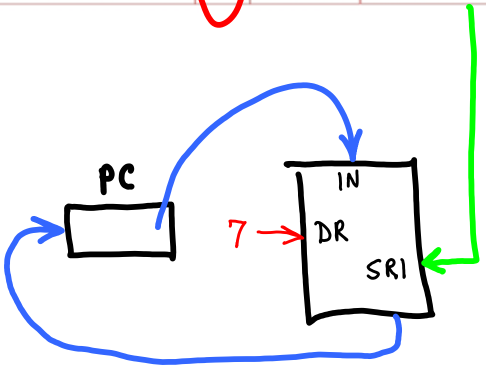
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JSR	0	1	0	0	1	PCoffset11										



"RET" is a synonym for "JMP R7"

JSRR Instruction

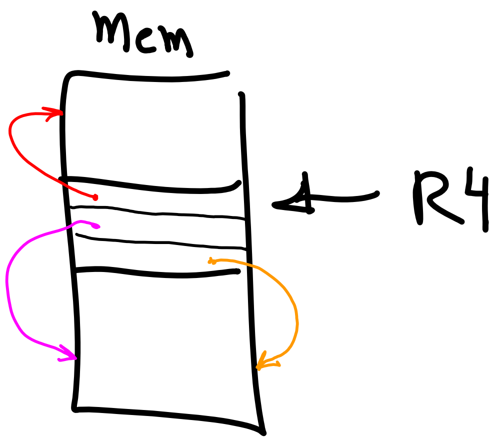
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JSRR	0	1	0	0	0	0	0	Base			0	0	0	0	0	0



linking

```
        ...
        .EXTERNAL SQRT
        ...
        LD   R2, SQAddr
        JSRR R2
        ...
SQAddr  .FILL SQRT
```

Not supported by LC3 assembler, lc3as, but see lcc, C compiler for LC3.



Problem
Jumping or accessing data far away requires having the distant address available. LD can get the address into a register, then JMP REG or LDR can reference the distant location. But then LD must use w/ a local pointer variable.

Solution
Have a data table in memory containing memory address, and set a Global Data Pointer, GDP/R4, to point to it. Now all remote address are available via "LDR REG, R4, offset".

568 appendix c The Microarchitecture of the LC-3

~ 50 states

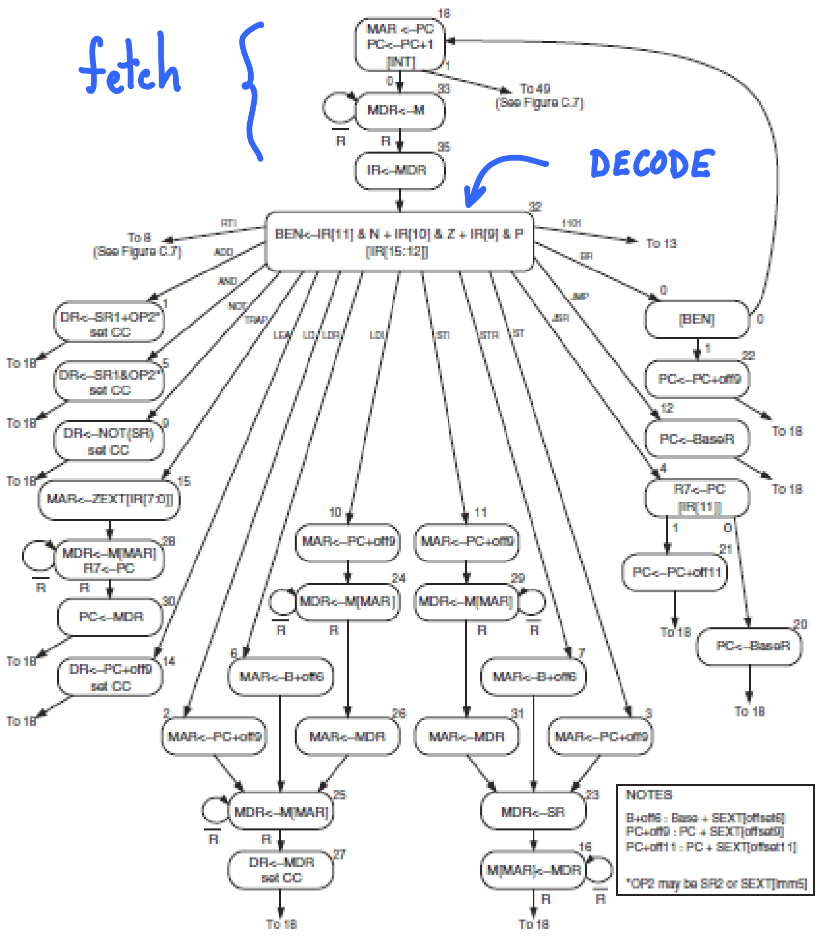
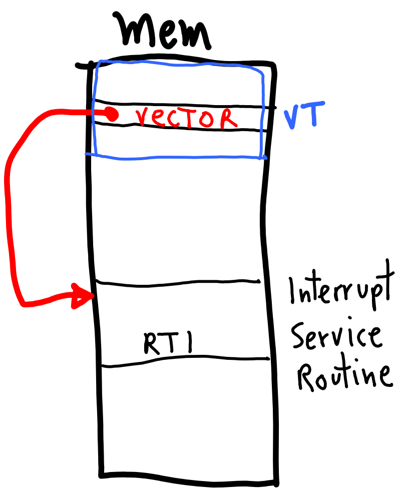


Figure C.2 A state machine for the LC-3

Interrupts, Exceptions



C.6 Interrupt and Exception Control 577

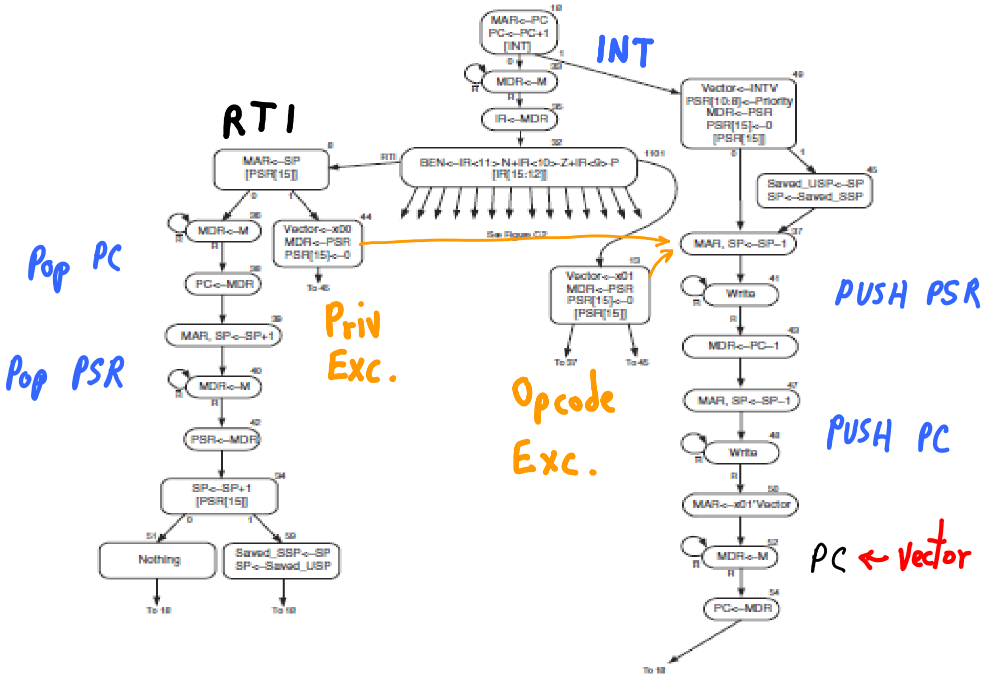
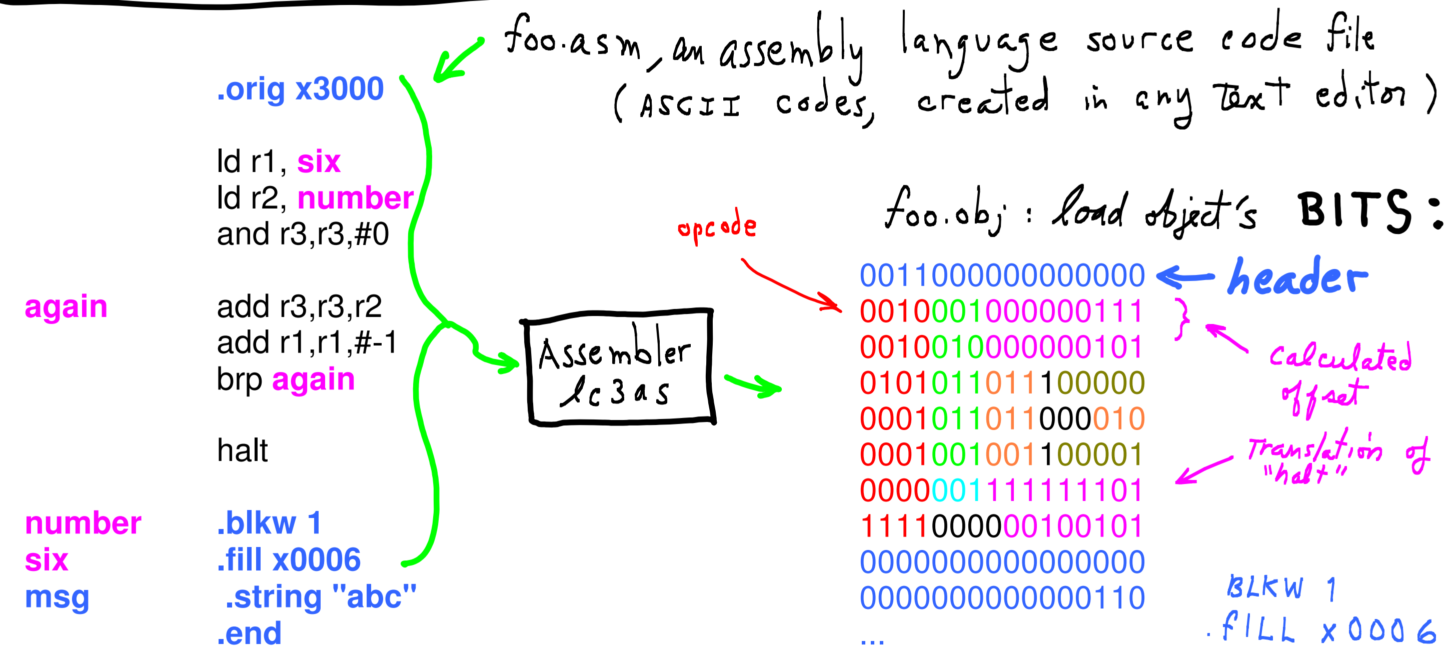


Figure C.7 LC-3 state machine showing interrupt control

Assembly Language

P&P, Figure 7.1


foo.asm, an assembly language source code file (ASCII codes, created in any text editor)
Assembler lc3as
foo.obj : load object's BITS:
opcode
header
calculated offset
Translation of "halt"
BLKW 1
.FILL x0006

```
            .orig x3000

            ld r1, six
            ld r2, number
            and r3,r3,#0

again       add r3,r3,r2
            add r1,r1,#-1
            brp again

            halt

number      .blkw 1
six         .fill x0006
msg          .string "abc"
            .end
```

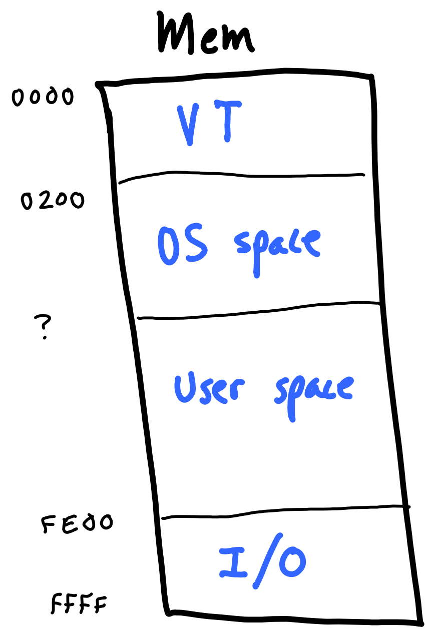
```
0011000000000000
0010001000000111
0010010000000101
0101011011100000
0001011011000010
0001001001100001
0000001111111101
1111000000100101
0000000000000000
0000000000000110
...
```

Assembler (lc3as) **Directives** (to control the assembly process):
.orig: puts a load address into the .obj load-object file's header.
.end: tells assembler, this is the end of source code.
.blkw: tells assembler, create *n* blank words (all zeroes).
.fill: tells assembler, put these bits into a word.
.string: convert text to .FILL w/ one ascii code per word, NUL terminated.

The assembler produces machine code words:
--- ONE PER LINE expressing an LC3 instruction
--- ONE PER LINE where there is a .fill directive
--- n PER LINE where there is a .blkw directive
The assembler also calculates offsets for us using **symbols**. Symbols stand for memory addresses (starting for the .orig address). Offsets are calculated by subtraction. Symbols refer to the next instruction's location.

LC-3
Memory-mapped I/O (Table A.3)

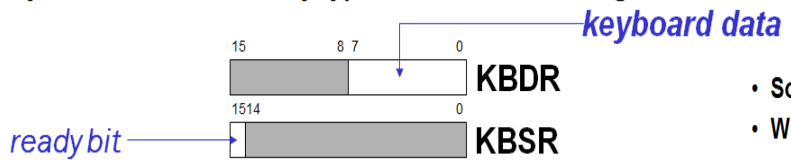
Location	I/O Register	Function
xFE00	Keyboard Status Reg (KBSR)	Bit [15] is one when keyboard has received a new character.
xFE02	Keyboard Data Reg (KBDR)	Bits [7:0] contain the last character typed on keyboard.
xFE04	Display Status Register (DSR)	Bit [15] is one when device ready to display another char on screen.
xFE06	Display Data Register (DDR)	Character written to bits [7:0] will be displayed on screen.



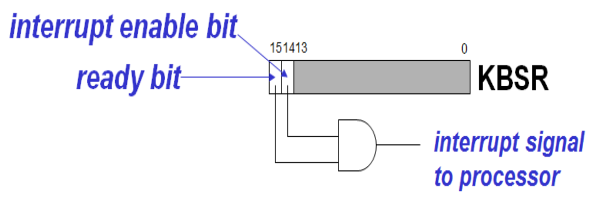
Input from Keyboard

When a character is typed:

- its ASCII code is placed in bits [7:0] of KBDR (bits [15:8] are always zero)
- the “ready bit” (KBSR[15]) is set to one
- keyboard is disabled -- any typed characters will be ignored

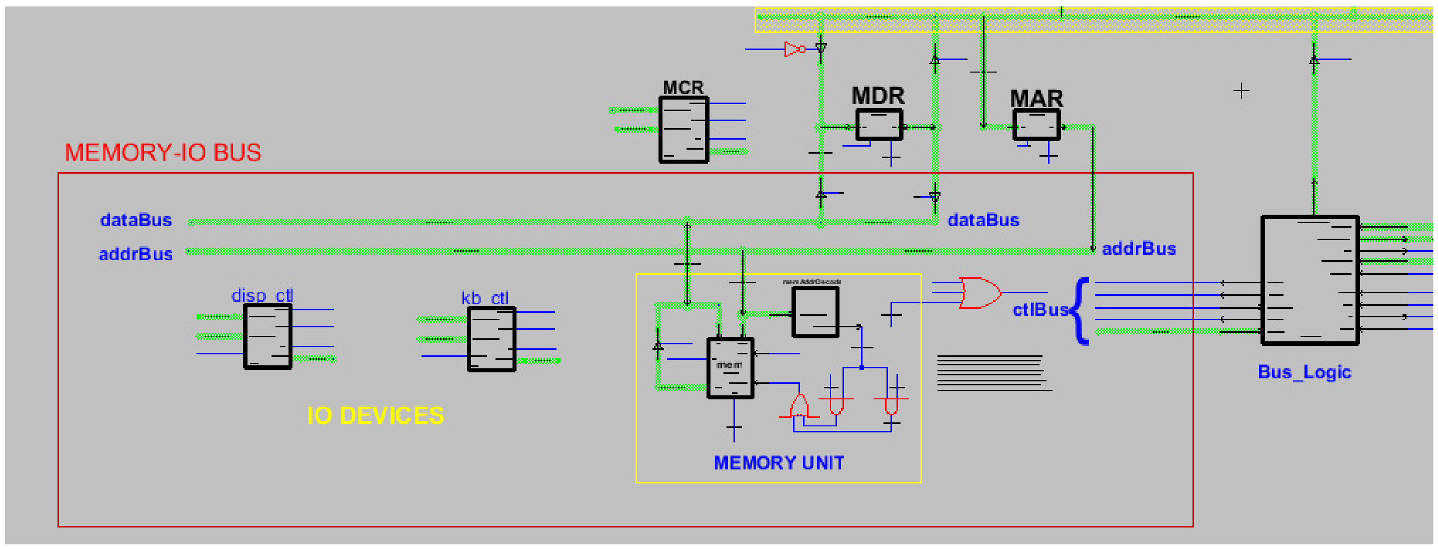


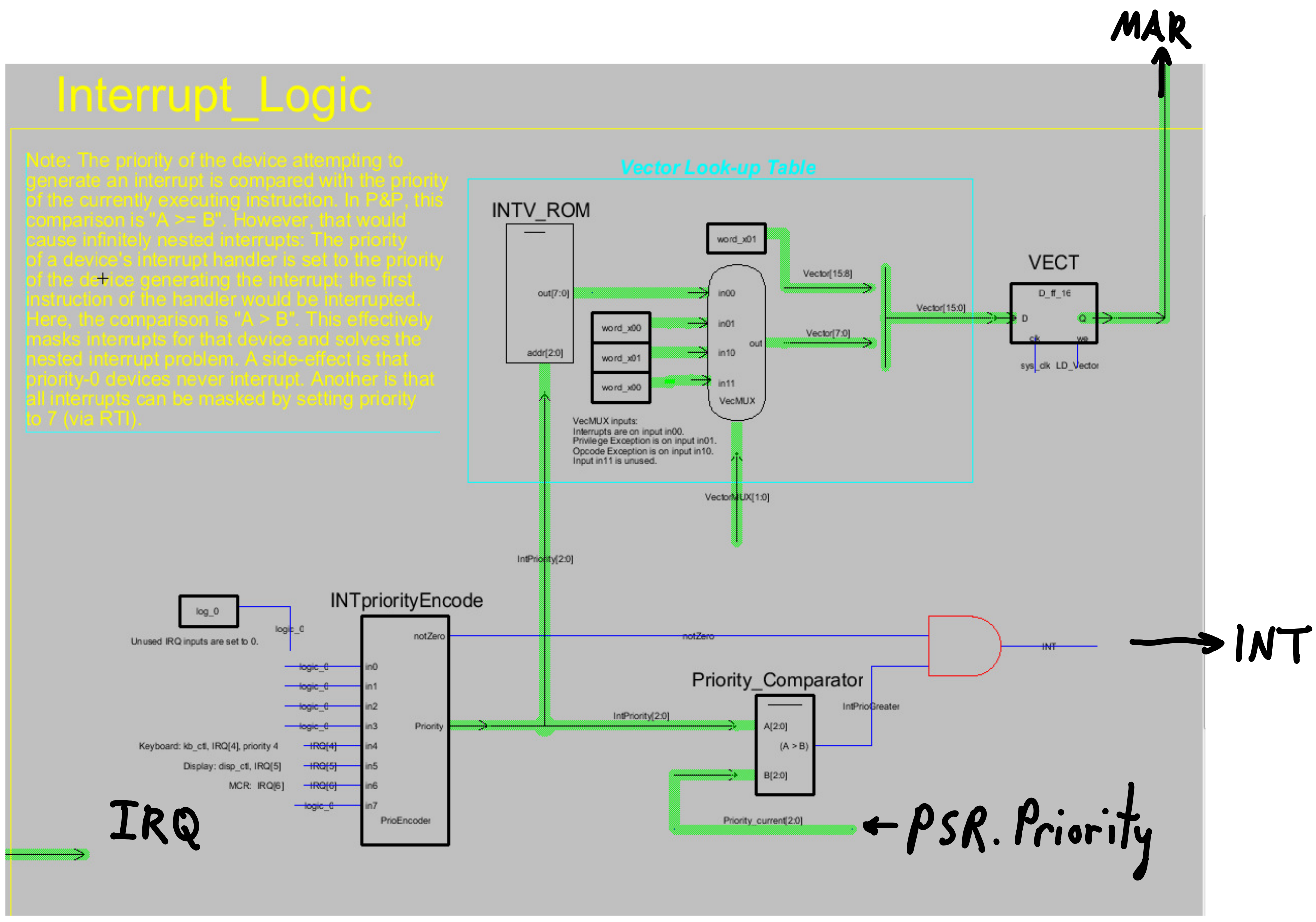
- Software sets "interrupt enable" bit in device register.
- When ready bit is set and IE bit is set, interrupt is signaled.



When KBDR is read:

- KBSR[15] is set to zero
- keyboard is enabled





Note: The priority of the device attempting to generate an interrupt is compared with the priority of the currently executing instruction. In P&P, this comparison is "A >= B". However, that would cause infinitely nested interrupts: The priority of a device's interrupt handler is set to the priority of the device generating the interrupt; the first instruction of the handler would be interrupted. Here, the comparison is "A > B". This effectively masks interrupts for that device and solves the nested interrupt problem. A side-effect is that priority-0 devices never interrupt. Another is that all interrupts can be masked by setting priority to 7 (via RTI).

VecMUX inputs:
Interrupts are on input in00.
Privilege Exception is on input in01.
Opcode Exception is on input in10.
Input in11 is unused.

Push

```
ADD  R6, R6, #-1  ; decrement stack ptr
STR  R0, R6, #0   ; store data (R0)
```

Pop

```
LDR  R0, R6, #0   ; load data from TOS
ADD  R6, R6, #1   ; decrement stack ptr
```

$$\left.\begin{array}{l}\text{R7 - linkage}\\ \text{R6 - SP}\end{array}\right\}\ \text{Hardware defined}$$

R5 - BP

R4 - GDP

-

RTI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1. **Pop PC from supervisor stack.** (PC = M[R6]; R6 = R6 + 1)
2. **Pop PSR from supervisor stack.** (PSR = M[R6]; R6 = R6 + 1)
3. **If PSR[15] = 1, R6 = Saved.USP.**
   (If going back to user mode, need to restore User Stack Pointer.)

Processor Status Register

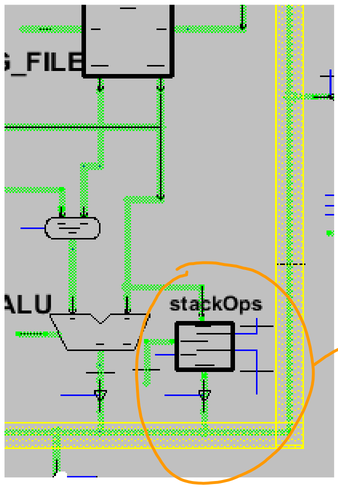
- **Privilege [15], Priority Level [10:8], Condition Codes [2:0]**

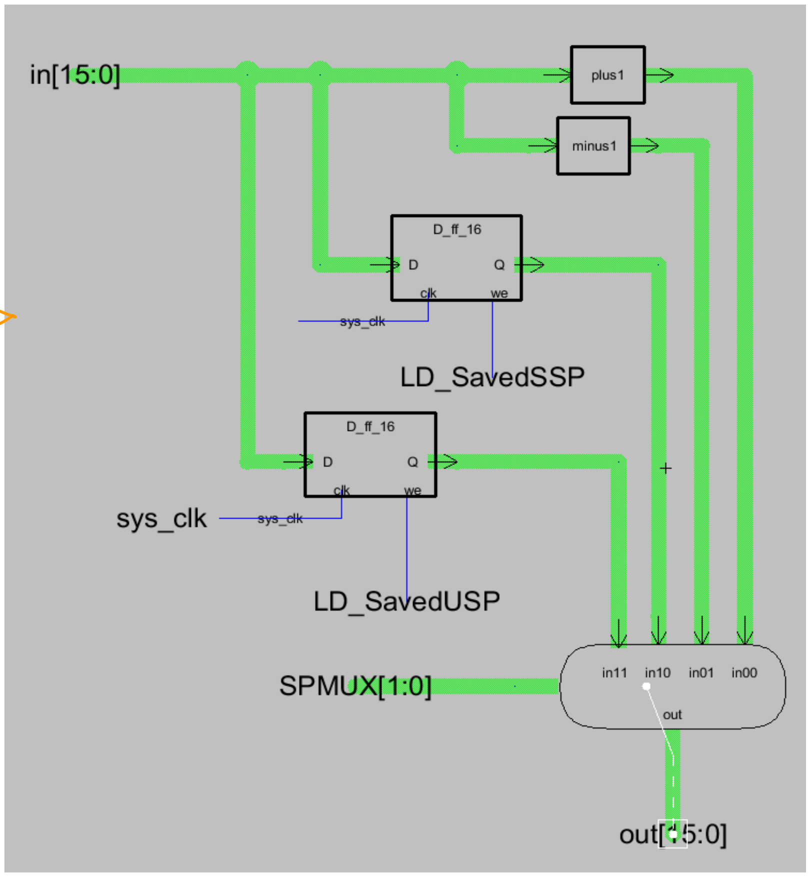
15	14–11	10–8	7–3	2	1	0
P		PL		N	Z	P

0 Supervisor

1 User mode

Switch stacks?
Save and restore SP, R6?





Hardware stack operations:

push, pop

Switch stacks:

UsersSP <==> SupersSP

C.6 Interrupt and Exception Control 577

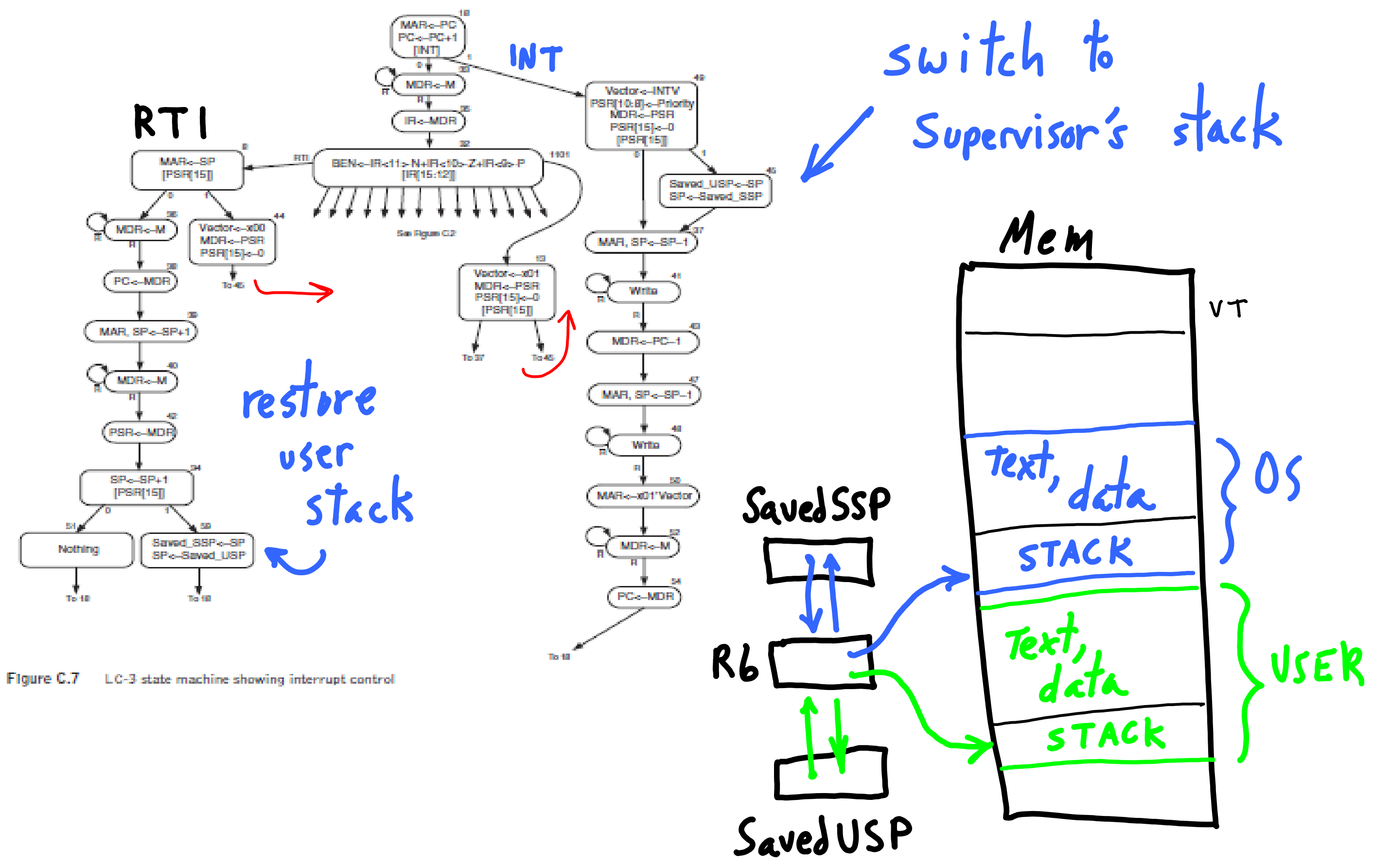


Figure C.7 LC-3 state machine showing interrupt control

Supplemental Readings
(PP) Patt & Patel, *Introduction to Computing Systems, 2e* (revised printing). McGraw-Hill. (Also see online material in LCA3-trunk/docs/ or on the Web.)

PP, Chp. 7: 7.1-7.4 (LC-3 Assembly language: instruction syntax, labels, comments, assembler directives, 2-pass assembly, object files and linking, executable images).

PP, Chp 8: 8.1.1-8.3.3 (device registers, memory-mapped I/O, keyboard and display I/O), 8.5 (interrupts).

PP, Chp 9: 9.1.1-9.2.2 (TRAP/JSR subroutine calls, register saving).

PP, Chp 10: 10.1-10.2 (stacks, push/pop, stack under/overflow, interrupt I/O, saving/restoring program state).

PH, Chp 8: 8.1-8.5 (I/O, disks, disk structure, RAID, buses, polling vs. interrupts, interrupt priority, DMA). [Also see on the CD, 8.3 (networks).]

PP Appendices A and C (also see LC3-turnk/docs/):

LC-3 Instruction notation definitions:	App. A.2
LC-3 Instruction descriptions:	App. A.3
LC-3 TRAP routines:	App. A.3, Table A.2
LC-3 I/O device registers:	App. A.3, Table A.3
LC-3 Interrupt and exception execution:	App. A.4 and C.6
LC-3 FSM state diagram:	App. C, Fig. C.2 and C.7
LC-3 Complete datapath:	App. C, Fig. C.8
LC-3 memory map:	App. A.1

Supplemental Exercises from PP

PP, Chp 6:
6.13 (shift right [NB-use assembly language])

PP, Chp 7:
7.1 (instr. assembly w/ labels [NB-show instr. bits])
7.14 (replace an opcode in assembled prog. to debug)
7.24 (debug loop control)

PP, Chp 8:
8.5 (what is KBSR[15]?)
8.11 (polling vs. intr. efficiency)
8.14 (I/O addr. decode)
8.15 (KBSR[14] and intr. handling)

PP, Chp 9:
9.2 (TRAP execution)
9.13 (debugging JSR and RET)
9.19 (complete the intr. priority service call)

PP, Chp 10:
10.10 (CCs pushed on INT)
10.11 (device registers and IVT)
10.24 (prog. and INT service interaction)