LC-3, addressing modes

See P&P Appendices A and C: LC-3 ISA, TRAPS, Devices, Interrupts, Exceptions.

1. DATA IS IN REGISTERS (**RegFile[i] , IR , PC**)

modes

ADD R2 R3 R1

OP	DR	SR1		SR
0001	010	011	000	001

R2 <== R3 + R1 (register mode) reg, reg, reg

ADDi R2 R3 #2

OP	DR	SR		imm5
0001	010	011	1	00010

R2 <== R3 + **IR[4:0]** (immediate mode) reg, reg, imm.

LEA R2 myLoc

OP	DR	PCoffset9
1110	010	000 000 001

R2 <== PC + **IR[8:0]** (immediate mode) reg, PC, imm.
(assembly computes offset from label)

2. MEMORY ADDRESS IS IN REGISTERS (**Regfile[i] , PC , IR)**

LDR R2 R3 #2

OP	DR	BaseR	offset6
0110	010	011	000010

MAR <== R3 + **IR[5:0]** (base-offset mode)
R2 <== MDR

LD R2 myVar

OP	DR	offset9
1010	010	000000010

MAR <== PC + **IR[8:0]** (pc-relative mode)
R2 <== MDR (assembly calculates offset from label)

BR z myLoc

OP	CC	PCoffset9
0000	010	000000010

PC <== PC + **IR[8:0]** (if Condition Code Z=1)
(assembly calculates offset from label)

jmp R2

OP		BaseReg	
1100	000	010	000000

PC <== R2

jsr myFunc

OP		PCoffset11
0100	1	00000000010

R7 <== PC (assembly calculates offset from label)
PC <== PC + **IR[10:0]**

jsrr R2

OP			BaseReg	
0100	0	00	010	000000

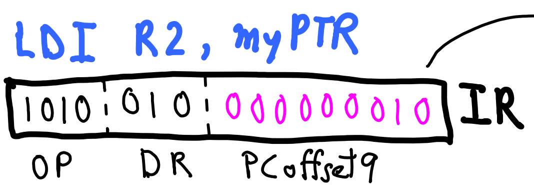
R7 <== PC
PC <== R2

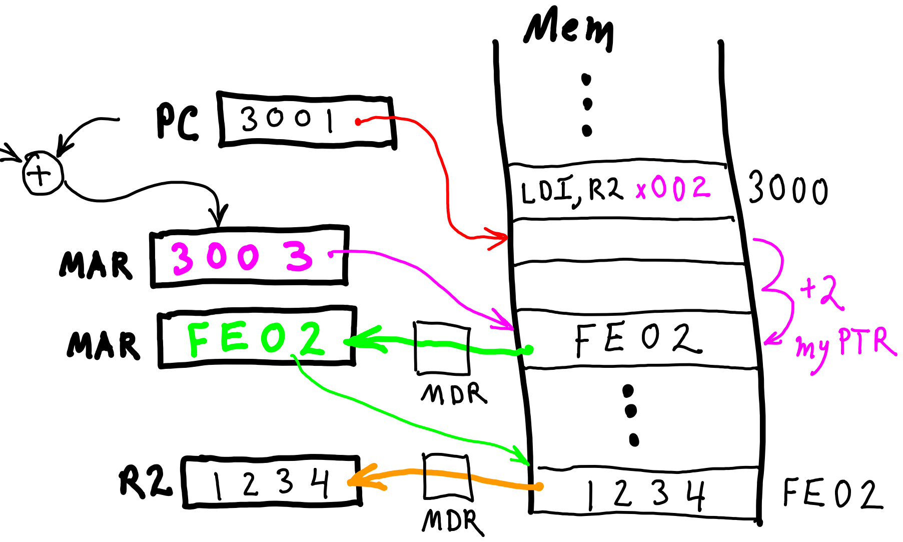
jmp R7

OP		BaseReg	
1100	000	111	000000

PC <== R7 (NB--assembly language shorthand, "ret")

3. MEMORY ADDRESS IS IN MEMORY





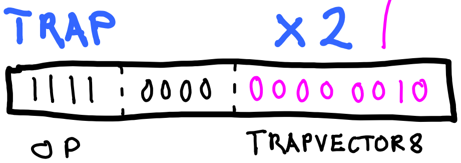
```
MAR <== PC + IR[8:0]     (get address where address is)
MAR <== MDR              (get address, use it)
R2   <== MDR             (get data at address)
```

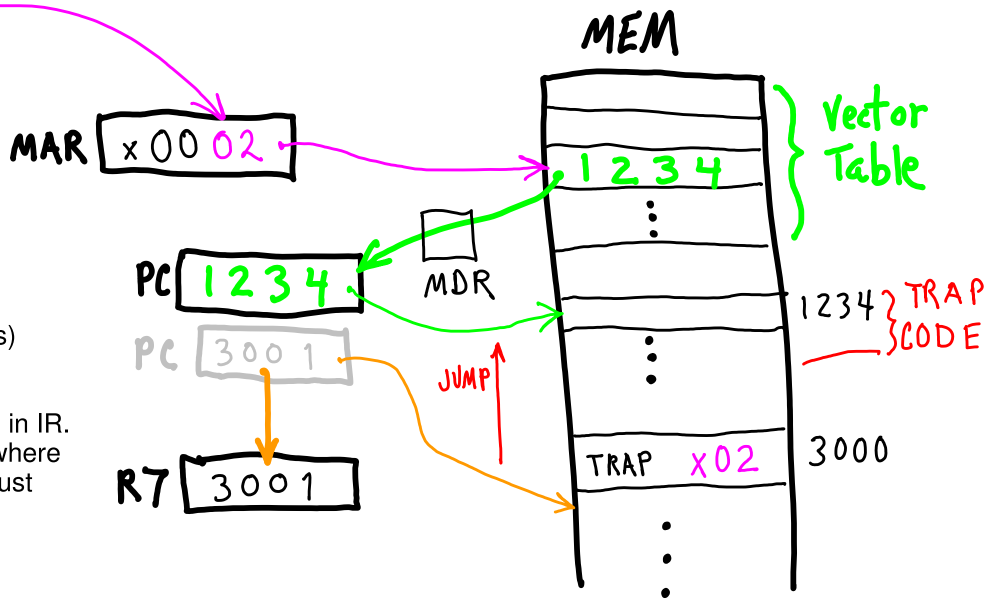
Idea: How to use full 16-bit address using only 9 bits in IR.

```
    ldi r2, myPTR
    ...
 myPTR: .FILL xFE02
```

Alternative: Move myPTR into a register, use base-offset mode:

```
    ld r1, myPTR
    ldr r2, r1, 0
    ...
 myPTR: .FILL xFE02
```





```
R7     <== PC
MAR <== IR[7:0]       (get address where address is)
PC     <== MDR                    (get address, jump)
```

Idea: How to make full 16-bit jump using only 8 bits in IR. Also, how to jump to OS trap routine w/o knowing where trap routine's code is. Allows OS to relocate itself: just change vector table entry.

```
    trap x2       ;--- jump to OS service routine x02.
    ...
```

Alternative: Move VT entry into a register, use jssr:

```
    ldi r1, VT2
    jssr r1
    ...
 VT2: .FILL x0002
```

Note: Using what we had above to eliminate ldi, we could eliminate both LDI and TRAP instructions from the LC3's ISA: we would have two unused opcodes to play with.

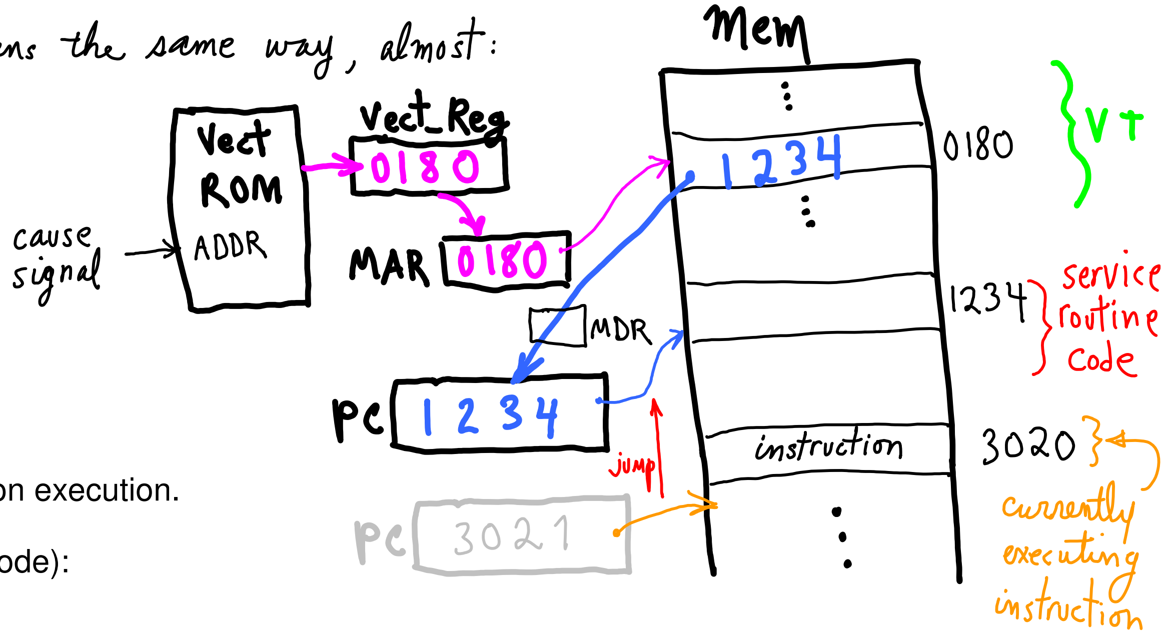
Exceptions Interrupts

Yet another address-in-memory mechanism.
Just like TRAP, but not an instruction.

Something goes wrong: jump to OS routine (exception)
I/O device sends a signal: jump to OS routine (interrupt)

The jump happens the same way, almost:

MAR <== VECT_REG
PC <== MDR



EXCEPTIONS
---- detected during instruction execution.
Eg., "illegal opcode"
detected in state-32 (decode):
VECT_REG <== x0100.

INTERRUPTS
---- generated by device interrupt logic
---- detected in State-18 (fetch)
Eg., a keyboard event:
VECT_REG <== x0180

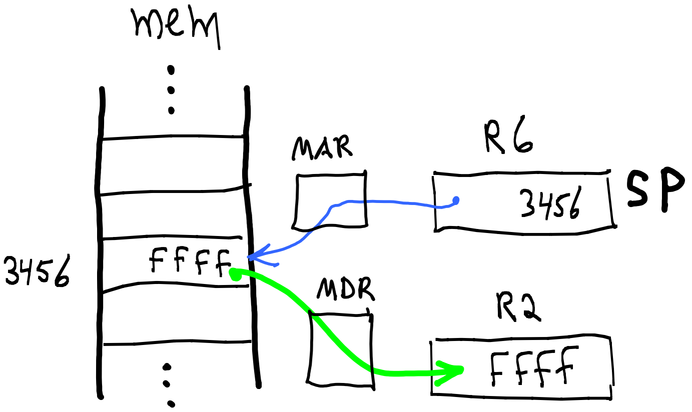
See LC3 Controller States,
13: opcode exception
44: privilege exception
49: interrupt

But,
more needs to be done: Save currently executing code's state!

Not the same as TRAP.
For TRAP, currently executing code,
---- knows a jump is occurring;
---- can SAVE its own STATE beforehand;
---- knows its CC state could change: does not BR immediately after TRAP.

Before we explain saving state, let's see Stack Addressing.

STACK OPERATIONS



I. Access top item in stack.

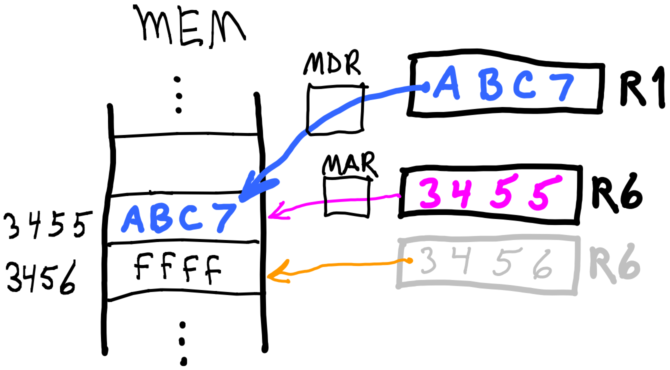
R2 ← mem[R6]

LDR R2, R6, #0

MAR <== R6
R2 <== MDR

Stack Pointer (SP) is R6

II. Put new item on top of stack: PUSH

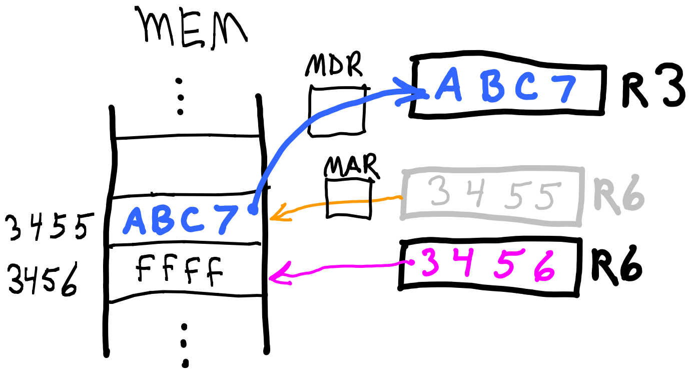


PUSH R1

ADD R6, R6, #-1 — 1. R6--
STR R1, R6, #0 — 2. MEM[R6] ← R1

```
R6  <== R6 - 1
MAR <== R6 + IR[5:0]
MDR <== R1
```

III. Remove item from top of stack: POP



PoP R3

LDR R3, R6, #0 — 1. R3 ← MEM[R6]
ADD R6, R6, #1 — 2. R6++

```
MAR <== R6 + IR[5:0]
R3  <== MDR
R6  <== R6 - 1
```

Saving State

We need to restart currently executing code in its Same execution state (PSR, PC, SP, RegFile)

When an exception/interrupt occurs

---- The PSR gets altered immediately, before the next instruction is fetched.

---- The PC gets altered, i.e., a jump.

---- PC could go to R7, but what about nested execeptions/interrupts?

---- The SP (R6) is used to save state, it needs to be saved.

---- Regs can be saved by service routine code.

===> Hardware, not instruction execution, must save state!

49 INT

```
   MDR         <= PSR          1.
   PSR[10:8] <= IntPriority    1.
   PSR[15]    <= 0             1.
 <PSR[15] == 1?> save SP
```

37, 41 push PSR

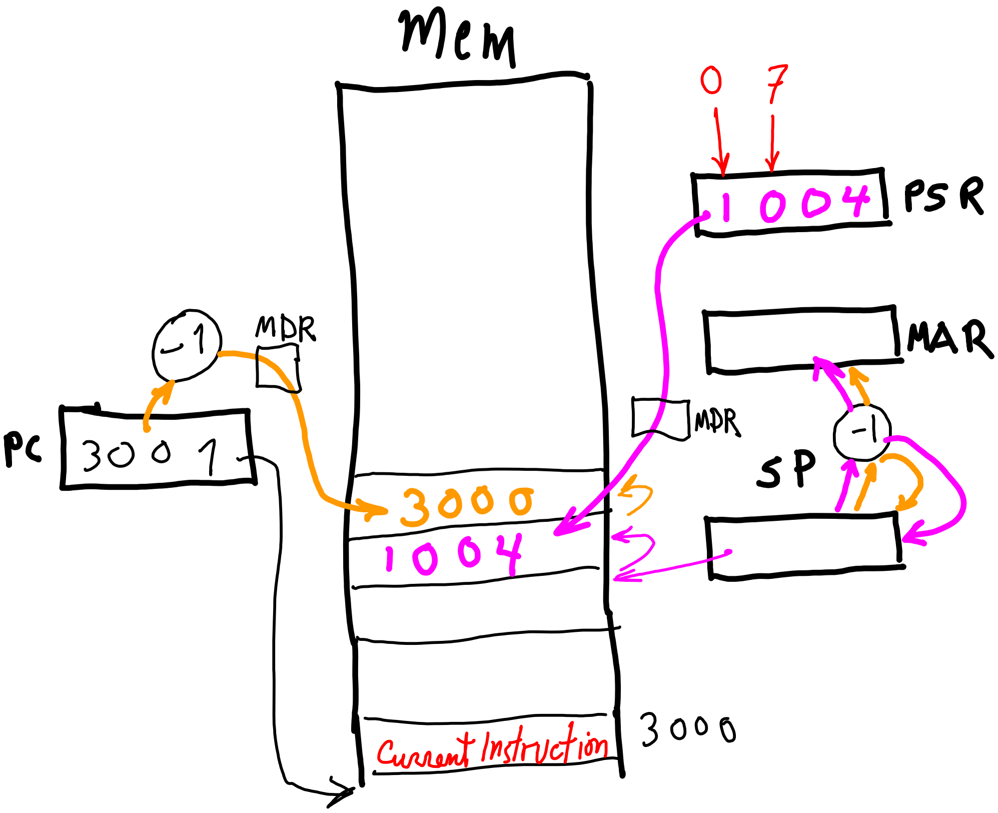
```
   SP    <= SP-1   }  1.
   MAR <= SP-1     }
   Mem <= MDR      }
```

43, 47, 48 push PC

```
   MDR <= PC-1     }
   SP    <= SP-1   }  2.
   MAR <= SP-1     }
   Mem <= MDR      }
```

50, 52, 54 jump

```
   MAR <= Vector
   MDR <= M
   PC    <= MDR
```



ALSO, if PSR[15] == 1, must save SP, and switch to SUPER's STACK. See R6 save/restore hardware near ALU.

When exception/interrupt routine COMPLETES

--- RESTORE Regs, done in software

--- RESTORE PC, PSR: the **RTI** instruction:

```
   PC    <== POP
   PSR <== POP
```

---- RESTORE SP, see R6 save/restor hardware

8 RTI

```
   MAR  <= SP
```

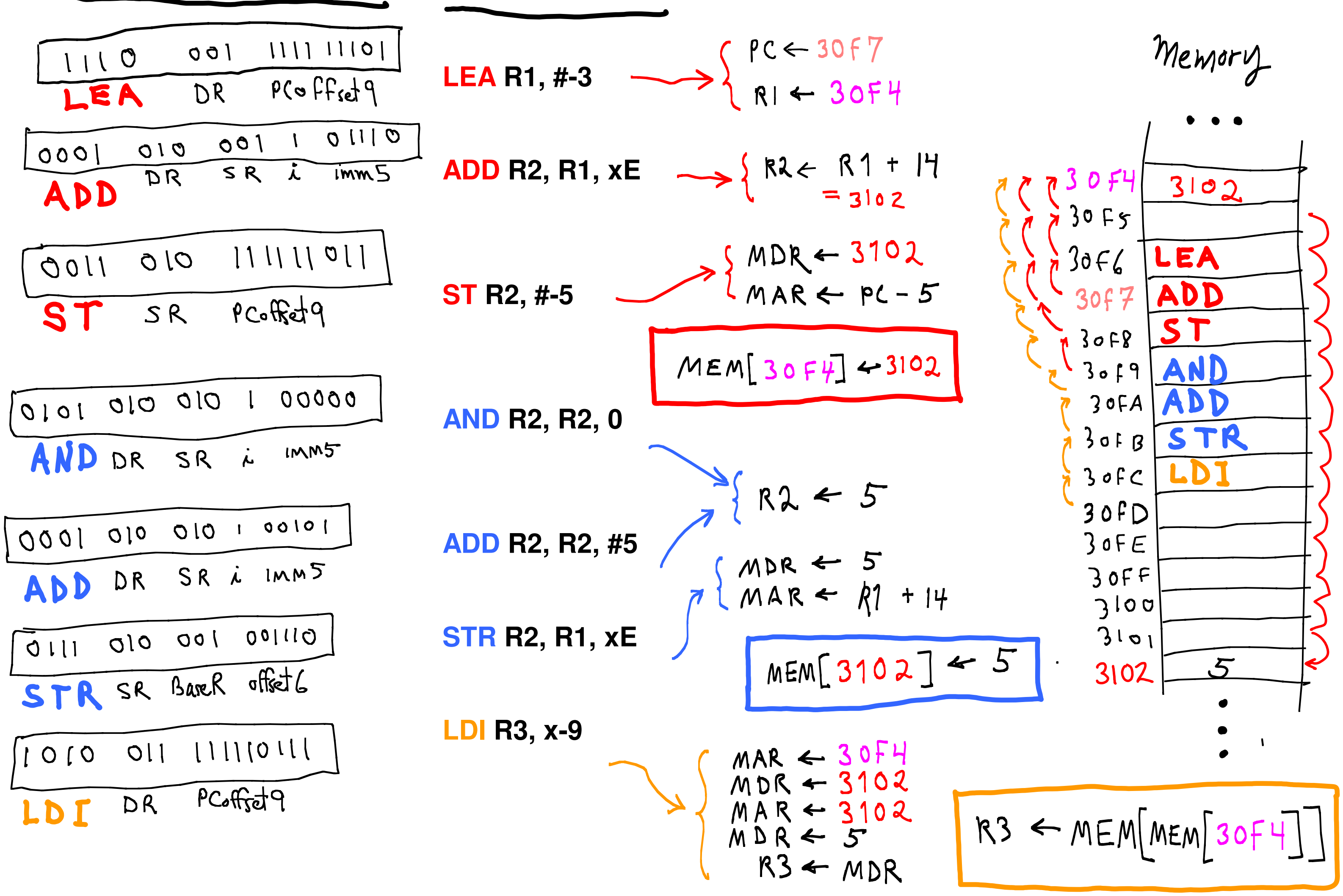
36, 38, 39 pop PC

```
   MDR <= Mem
   PC    <= MDR
   SP    <= SP+1
   MAR <= SP+1
```

40, 42, 34 pop PSR

```
   MDR <= Mem
   PSR <= MDR
   SP    <= SP+1
  <PSR[15] == 1?> (restore SP)
```

(PP, example, Section 5.3.5)



LEA R1, #-3

ADD R2, R1, xE

ST R2, #-5

AND R2, R2, 0

ADD R2, R2, #5

STR R2, R1, xE

LDI R3, x-9

;-- R1 <== &pointer R1 gets (address of pointer variable)

;-- R2 <== &data R2 gets (address of pointer variable + 14) == (address of data variable)

;-- pointer <== &data pointer variable gets (R2, address of data variable)

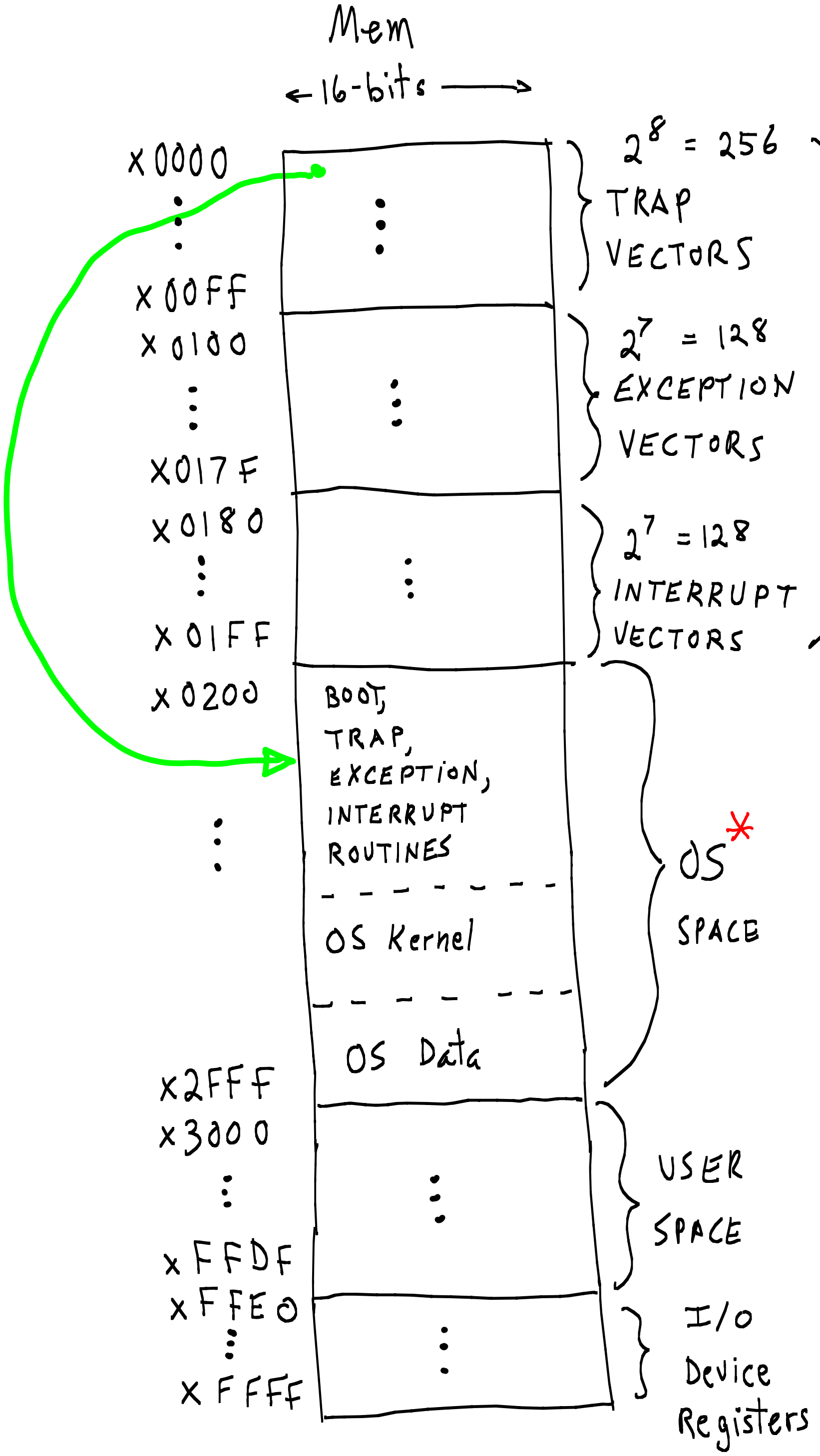
;-- R2 <== 0 data calculation into R2

;-- R2 <== 5 data calculation into R2

;-- data <== 5 MEM[(R1, address of pointer variable) + 14] gets data, R2

;-- R3 <== data R3 gets data from MEM via de-referencing pointer variable.

LC3 Memory Map



Vector Table, hardware defined

$3 \cdot 2^{12} - \frac{1}{2}k = (12 - \frac{1}{2})k = 11.5k$

*which OS?

$\approx 64k - 12k = 53k$

$2^5 = 32$

device address range

```
1111 1111 111|0 0000
1111 1111 111|0 0001
        ...
1111 1111 111|1 1111
```

If these address bits on Addr Bus are 1's then reference is to I/O device register, not for memory.