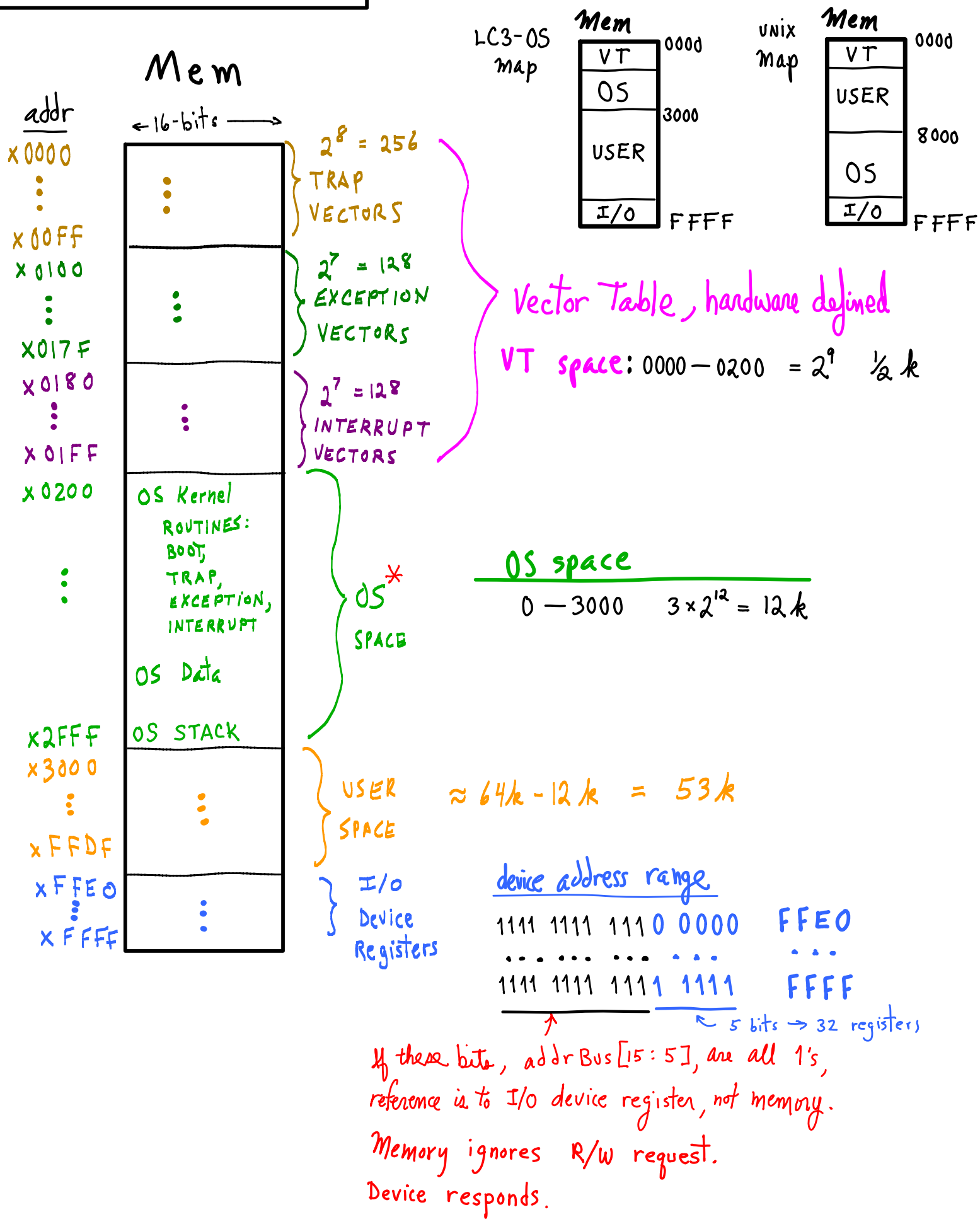
LC3 Memory Map

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



Memory access, register usage

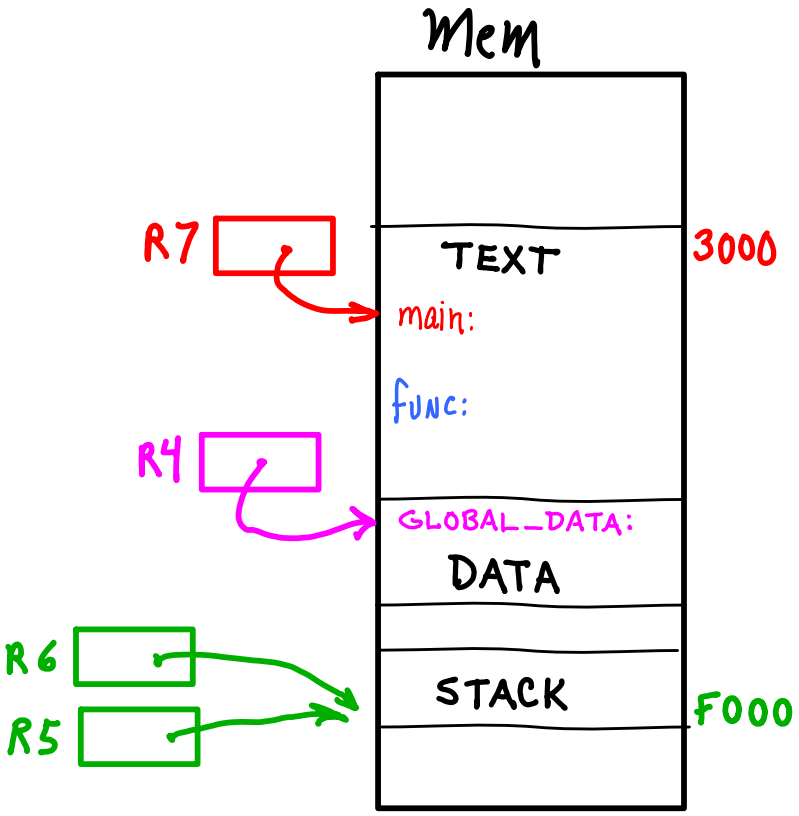
C compiler generated

```
.Orig x3000

LD   R4, DATA_POINTER
LDR  R6, R4, #0
LDR  R5, R4, #0
LDR  R7, R4, #1
jsrr R7
TRAP x25

DATA_POINTER:
     .FILL GLOBAL_DATA

main:
        ...
        JMP R7

func:
        ...
        JMP R7

GLOBAL_DATA:
    .FILL xF000     ;;;  Stack bottom
    .FILL x3007     ;;;  address of main
    .FILL x1234     ;;;  int x = 0x1234
    .FILL x0010     ;;;  int y = 16
    .FILL func      ;;;  address of func
```

assembler fills in offsets

labels become addresses

assembler fills in actual address (loader might edit address)



;------ get data from variable y:

```
ADD R2, R4, #2
LDR R2, R2, #0
```

;----- jump to func's location:

```
ADD R7, R4, #3
JSRR R7
```

MEMORY ADDRESS IN MEMORY

Pointer access and usage

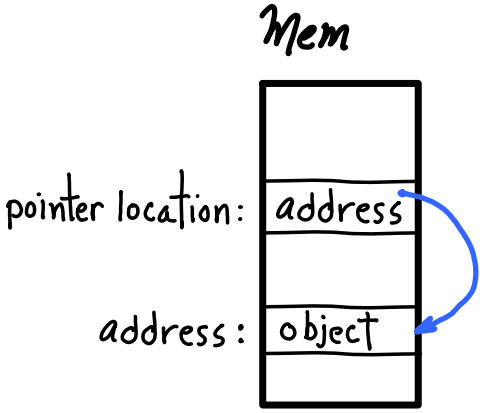
A pointer "points to" an object.

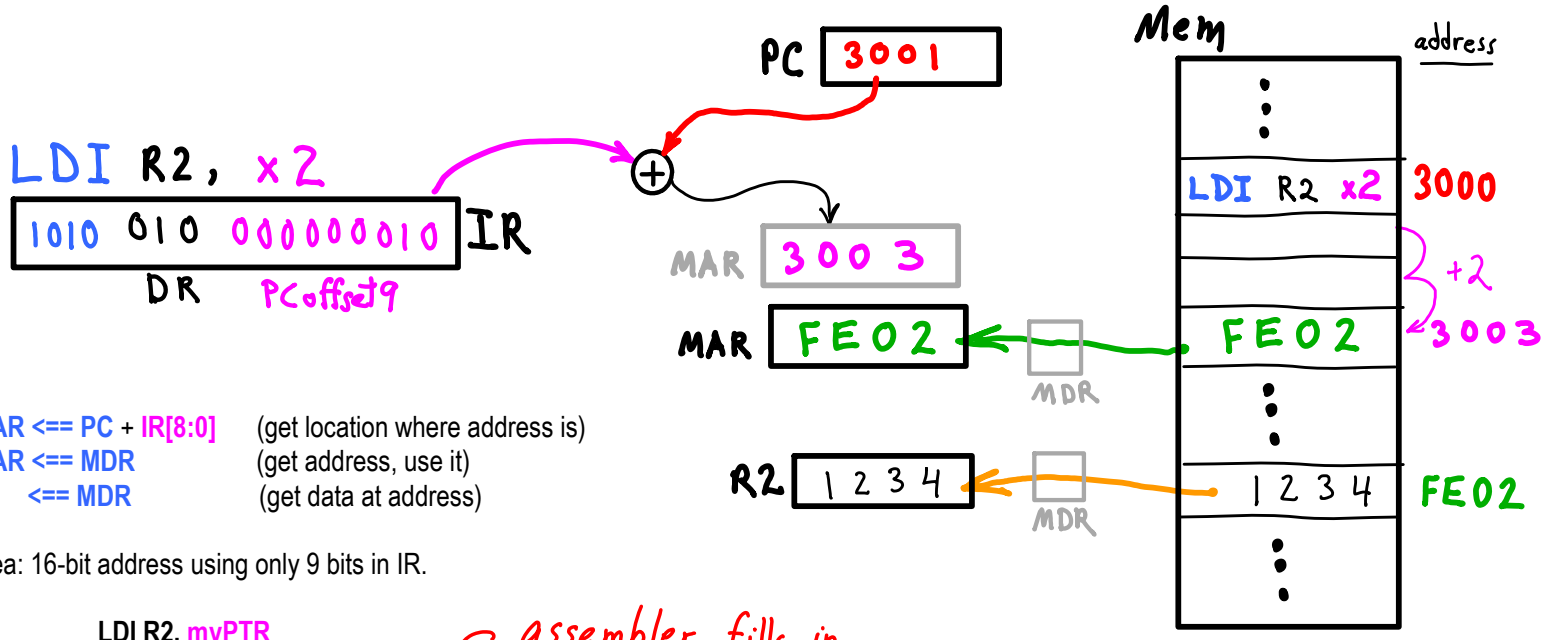
A pointer == an address.
A pointer variable == location containing address.

A pointer refers to an object: data or instructions

--- fetch address from location

--- R/W data to/from address OR jump to address



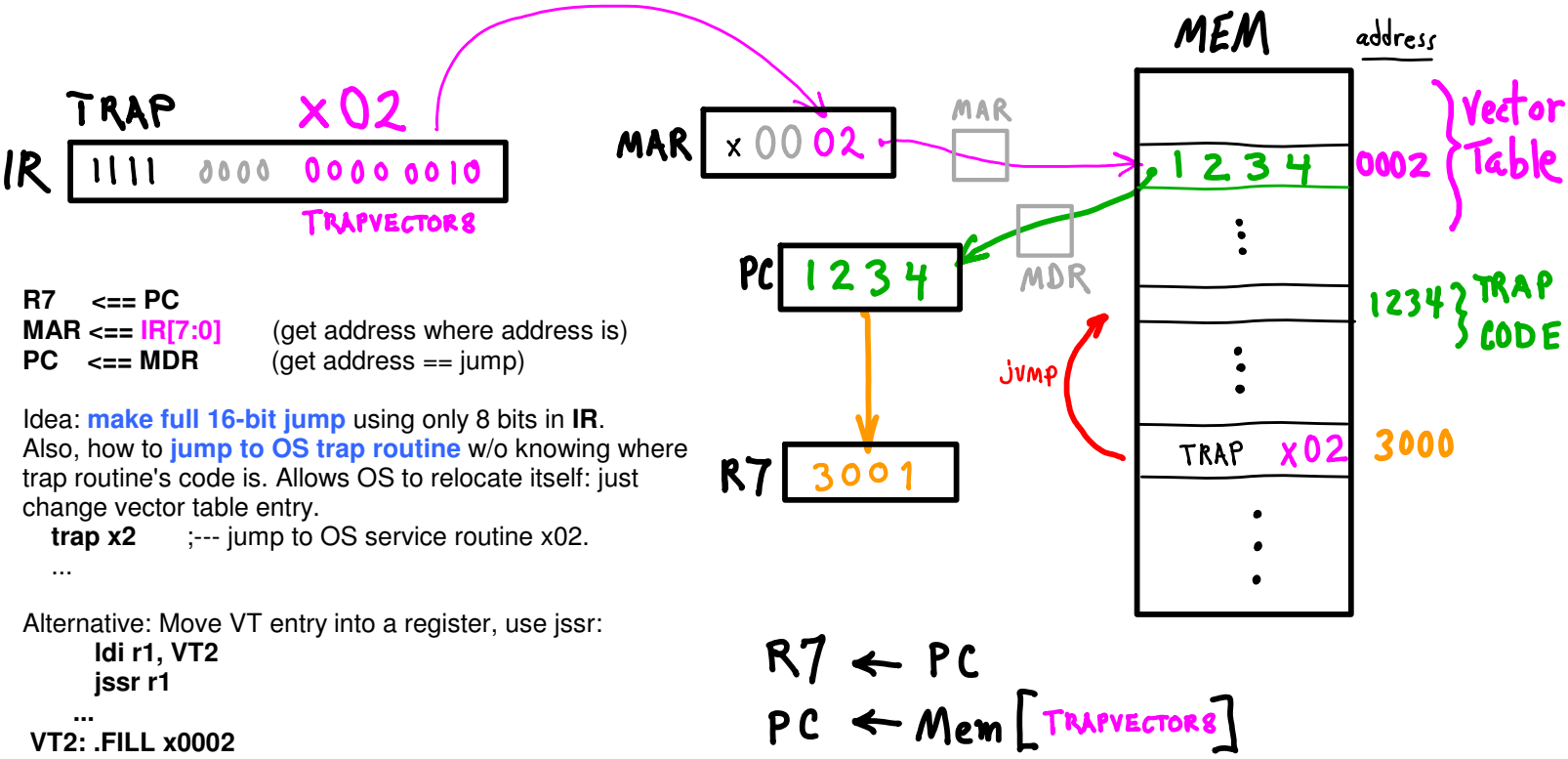


MAR <== PC + IR[8:0] (get location where address is)
MAR <== MDR (get address, use it)
R2 <== MDR (get data at address)

Idea: 16-bit address using only 9 bits in IR.

```
        LDI R2, myPTR
        ...
myPTR:  .FILL xFE02
```

assembler fills in offset by counting words



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

Idea: **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
```

R7 ← PC
PC ← Mem[TRAPVECTOR8]

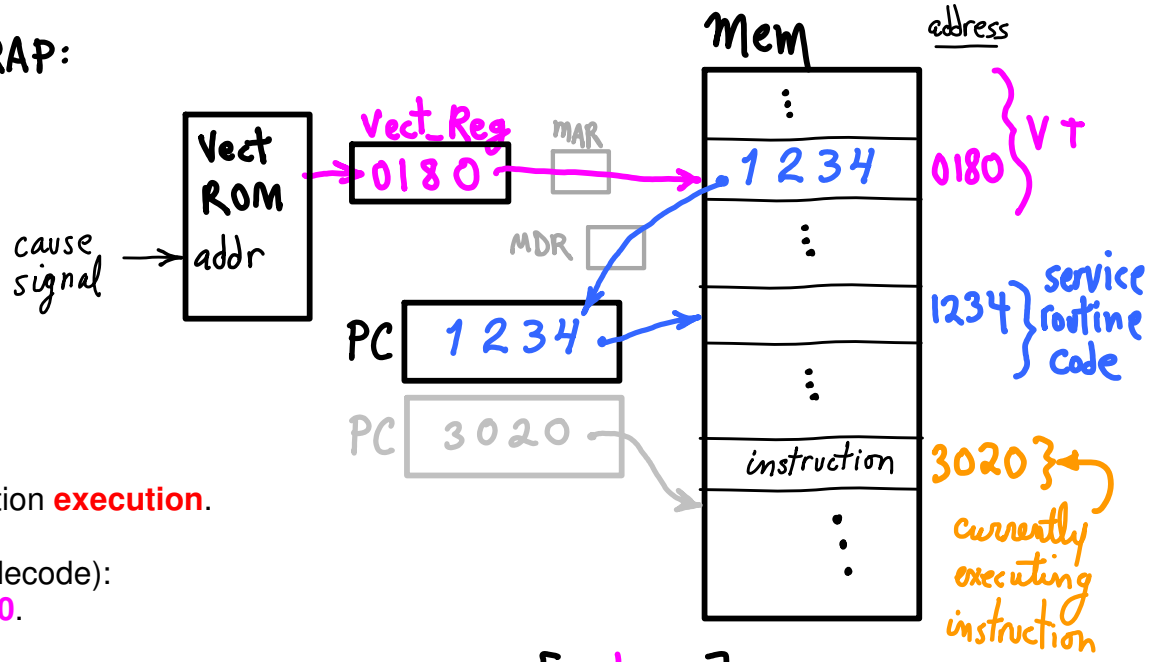
Aside: Using what we had above to eliminate ldi, we could eliminate both LDI and TRAP instructions from the LC3's ISA: we would have two unused opcodes to play with.

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**

Jump is same as TRAP:



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

INTERRUPTS
---- **generated by device** interrupt logic
---- detected in **state-18** (fetch)
keyboard event:
VECT_REG <== x0180

PC ← Mem[vect_reg]

(How To jump back?)

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

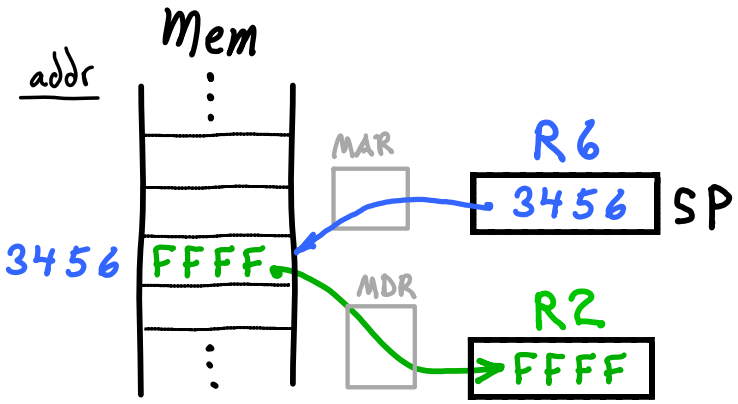
But,
more needs to be done: Save currently executing code's state!

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.

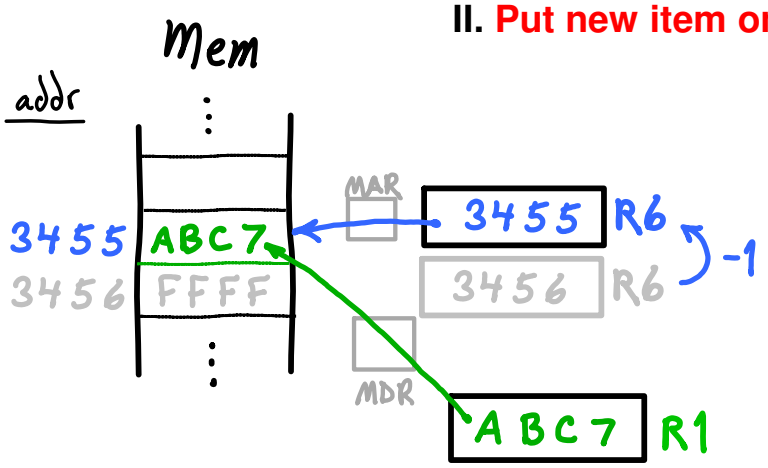


LDR R2, R6, #0

R2 <== MEM[R6]

Stack Pointer (**SP**) is **R6**

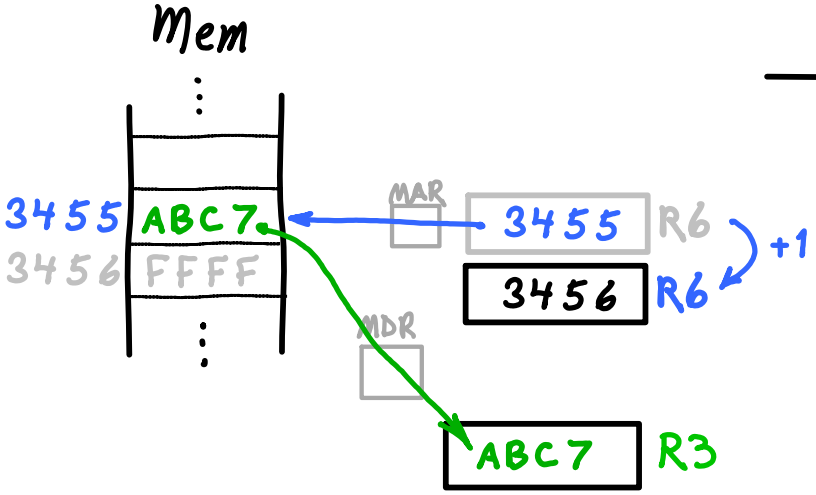
II. Put new item on top of stack: PUSH



PUSH R1

```
ADD R6, R6, #-1      R6--
STR R1, R6, #0       MEM[R6] ← R1
```

III. Remove item from top of stack: POP



POP R3

```
LDR R3, R6, #0       R3 ← MEM[R6]
ADD R6, R6, #1       R6++
```

Saving State

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

When an **exception/interrupt occurs**

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

---- **PC altered**, i.e., a jump.

PC could go to **R7**, but what is in **R7** **(function calls, nested interrupts)**?

---- **SP (R6) altered** to push state, it needs to be saved.

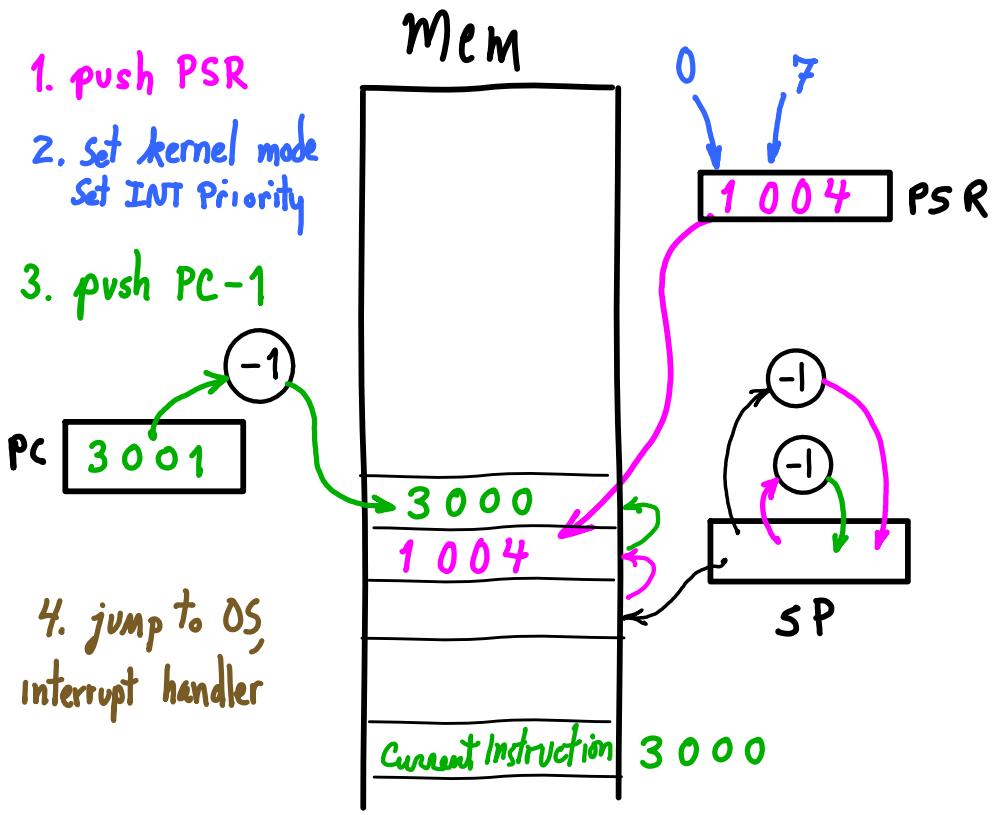
---- **Regs** can be **saved by service routine** code.

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

LC3 States for Interrupt

49 INT
MDR <== PSR
PSR[10:8] <== 3'b111
PSR[15] <== 1'b0
<PSR[15] == 1?> save R6

37, 41 push PSR
SP <== SP-1
MAR <== SP-1
Mem <== MDR
43, 47, 48 push PC
MDR <== PC-1
SP <== SP-1
MAR <== SP-1
Mem <== MDR
50, 52, 54 jump
MAR <== Vector
MDR <== Mem
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 service routine
execute **LD** instructions

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

Pop PC
Pop PSR
Restore SP (see **R6** save/restore hardware)

8 RTI
MAR <== SP

Pop PC

36, 38, 39
MDR <== Mem
PC <== MDR
SP <== SP+1
MAR <== SP+1

Pop PSR

40, 42, 34
MDR <== Mem
PSR <== MDR
SP <== SP+1

< PSR[15] == 1? >

Restore R6 SP