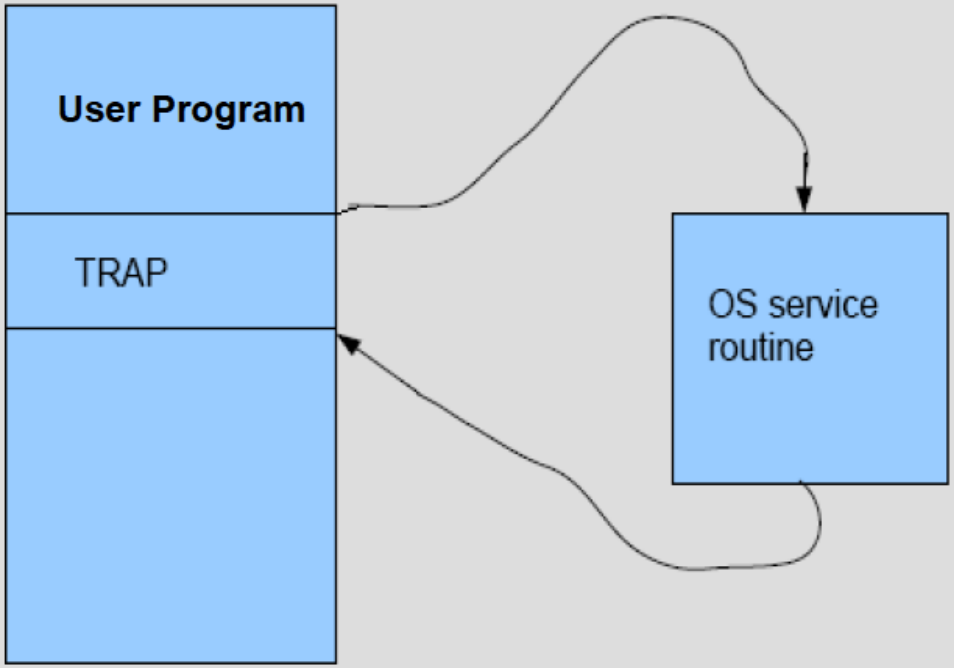
COMP1003 Computer Organization

Lecture 13 Trap Routines

United International College

What's a Trap?

- A TRAP is a system call (service call) to get the OS to do some job



A Real Trap

Why do we need Traps?

- Programming I/O devices is very complicated, application programmers don't have to know the low level detail
- If a user programmer is allowed to access hardware directly, and he/she messed up, it could create troubles to other user programs
- To safely and conveniently perform low-level, privileged operations, system calls or Traps are needed

When do we need a Trap?

- There are some operations that are best executed by the OS rather than a user program:
  - I/O instructions
  - Loading of memory-mapped registers
  - Resetting the clock
  - Halt
- If any user program wants to do the above things, it can do so by calling system routines via Trap instructions

Three Functions of Traps

- Traps (Service routines) provide three main Functions:
  - 1. Information Hiding: shield programmers from system-specific details.
  - 2. Code Reuse: Write frequently-used code just once.
  - 3. Security: Protect system resources from malicious/clumsy programmers.

The Trap Mechanism

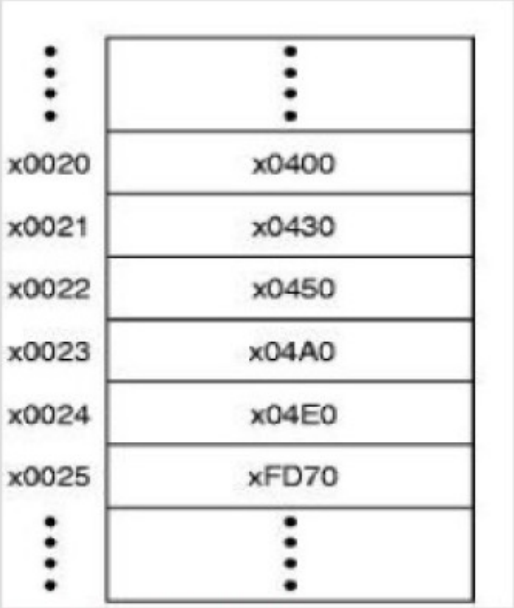
- A set of trap service routines (TSRs)
- A table of the starting addresses of these service routines
  - Located in a pre-defined block of memory …
  - called the Trap Vector Table or System Control Block
  - In the LC-3: from x0000 to x00FF (only 5 currently in use)
- The TRAP instruction
  - which loads the starting address of the TSR into the PC
- A linkage back to the user program at the end of the TSR

LC3 Trap Routines

- GETC (TRAP x20)
- OUT (TRAP x21)
- PUTS (TRAP x22)
- IN (TRAP x23)
- HALT (TRAP x25)

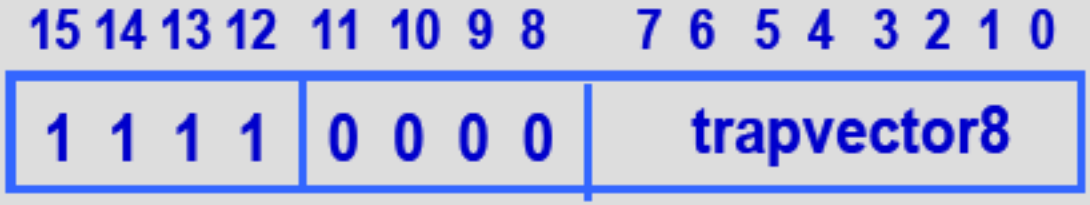
Trap Vector Table

- LC3's TVT: From x000 to x00FF
- Up to 256 entries
- Only 5 are in use



Trap Vector Table

The Trap Instruction



- *R7 ← (PC) ; the current PC is stored in R7*
- *PC ← Mem[Zext(IR[7:0])] ; the 8-bit trap vector is loaded to the PC*

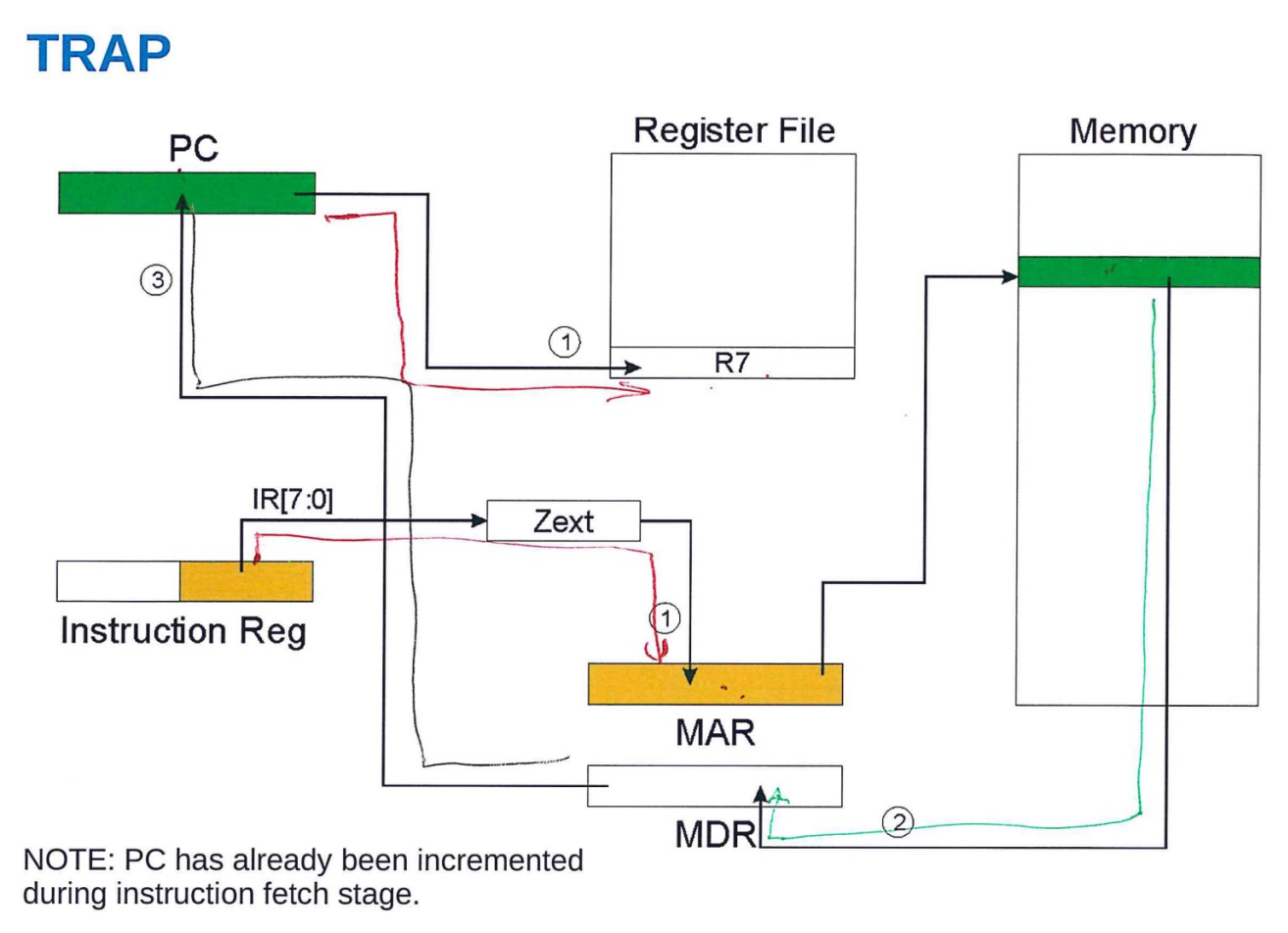
Where to go

- lookup starting address from table; place in PC

How to get back

- save address of next instruction (current PC) in R7

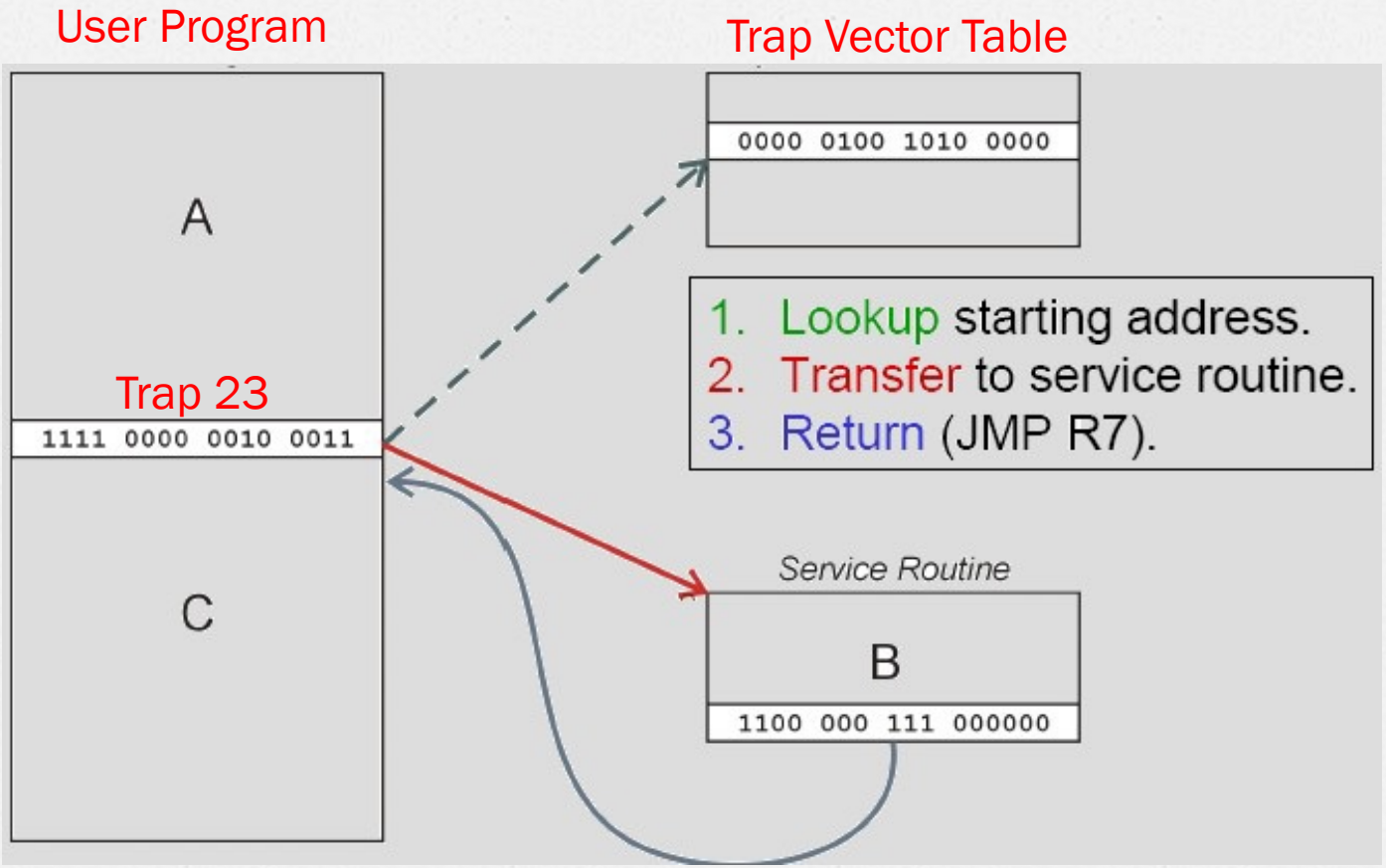
Datapath of Trap Instruction



Return – RET (JMP R7)

- How do we transfer control back to the instruction following the TRAP?
- We saved old PC in R7. JMP R7 gets us back to the user program at the right spot.
- LC-3 assembly language lets us use RET (return)in place of “JMP R7”.
- Must make sure that service routine does notchange R7, or we won’t know where to

Trap Mechanism Operation



Trap Vector Table

Memory

x0020

0x	Label	Hex	Instruction
x0020		x0400	NOP
x0021		x0430	BRz x0052
x0022		x0450	BRz x0073
x0023		x04A0	BRz x00C4
x0024		x04E0	BRz x0105
x0025		xFD70	.FILL xFD70

OUT (Trap x21)

Trap Routine

Memory

x0430 Manag

0x	Label	Hex	Instruction
x0430	TRAP_OUT	x3E0A	ST R7, x043B
x0431		x3208	ST R1, x043A
x0432		xA205	LDI R1, x0438
x0433		x07FE	BRzp x0432
x0434		xB004	STI R0, x0439
x0435		x2204	LD R1, x043A
x0436		x2E04	LD R7, x043B
x0437		xC1C0	RET
x0438		xFE04	.FILL xFE04
x0439		xFE06	.FILL xFE06
x043A		x0000	NOP
x043B		x0000	NOP

DSR

DDR

OUT (Trap x21)

```
01            .ORIG X0430 ;  System call starting address
02            ST R1, SaveR1 ; R1 will be used for polling
03
04  ; Write the character
05  TryWrite LDI R1, DSR ; Get status
06            BRzp TryWrite ; bit 15 = 1 => display ready
07  WriteIt   STI R0, DDR ; Write character in R0
08
09 ; Return from TRAP
0A Return    LD R1, SaveR1 ; Restore registers
0B            RET ; Return (actually JMP R7)
0C DSR       .FILL xFE04 ; display status register
0D DDR       .FILL xFE06 ; display data register
0E SaveR1 .BLKW 1
0F           .END
```

Saving and Storing Registers: Caller Save

- Protect your values! Any subroutine call may change values currently stored in a register.
- Sometimes the calling program ("caller") knows what needs to be protected, so it saves the endangered register before calling the subroutine.
- This is known as "caller save"

Saving and Storing Registers: Callee Save

- Other times it will be the called program ("callee") that knows what registers it will be using to carry out its task.
- This is known as "callee save"
- This applies not only to trap service routines but to all subroutine calls

What does the program do?

```
            .ORIG x3000
            LD    R2, TERM    ; Load negative ASCII '7'
            LD    R3, ASCII   ; Load ASCII difference
AGAIN       TRAP  x23         ; input character
            ADD   R1, R2, R0  ; Test for terminate
            BRz   EXIT        ; Exit if done
            ADD   R0, R0, R3  ; Change to lowercase
            TRAP  x21         ; Output to monitor...
            BRnzp  AGAIN      ; ... again and again...
TERM        .FILL xFFC9       ; -'7'
ASCII       .FILL x0020       ; lowercase bit
EXIT        TRAP  x25         ; halt
            .END
```

Exercise

- How to define your own trap routine? For example, TRAP x26?

What's Next?

Stack & Interrupt-based IO