Introduction to Computing Systems
from bits & gates to C & beyond

Chapter 9

TRAP Routines

- *Privileged Instructions*
- *TRAP Routines*
- *Subroutines*

Privileged Instructions

- There are several instructions that are best executed by a *supervisor* program (OS) rather than a *user* program:
  - IO instructions
  - Loading of memory-mapped registers
  - Resetting the clock
  - Halt

  i.e. instructions where one program can affect the behavior of another.
- The CPU can be designed to enforce two modes of operation:
  - User Mode
  - Privileged Mode (aka. supervisor, kernel, monitor mode)
- Only the supervisor program (OS) can execute privileged instructions.

TRAP Instructions

- **TRAPs insulate critical tasks from the user**
  - **with or without privilege enforcement**
- **The TRAP mechanism:**
  - **A set of trap service routines or TSRs (part of the CPU OS)**
    - ***We have already seen the basic I/O SRs***
  - **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***
  - **Return link**
    - ***from the end of the TSR back to the original program.***

Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

LC-3 TRAP Routines

- **GETC (TRAP x20)**
  - ***Read a single character from KBD.***
  - ***Write ASCII code to R0[7:0], clear R0[15:8].***
- **OUT (TRAP x21)**
  - ***Write R0[7:0] to the monitor.***
- **PUTS (TRAP x22)**
  - ***Write a string to monitor (address of first character of string is in R0).***
- **IN (TRAP x23)**
  - ***Print a prompt to the monitor and read a single character from KBD.***
  - ***Write ASCII code to R0[7:0], clear R0[15:8], echo character to the monitor.***
- **HALT (TRAP x25)**
  - ***Print message to monitor & halt execution.***

⋮	⋮
x0020	x0400
x0021	x0430
x0022	x0450
x0023	x04A0
x0024	x04E0
x0025	xFD70
⋮	⋮

Trap vector table

TRAP Instructions

- **TRAP: A special instruction**
  - A form of subroutine call used to invoke a service routine.
  - If privilege is being enforced, it switches the execution to *privileged* mode, and reverts back to *user* mode when the TSR completes.

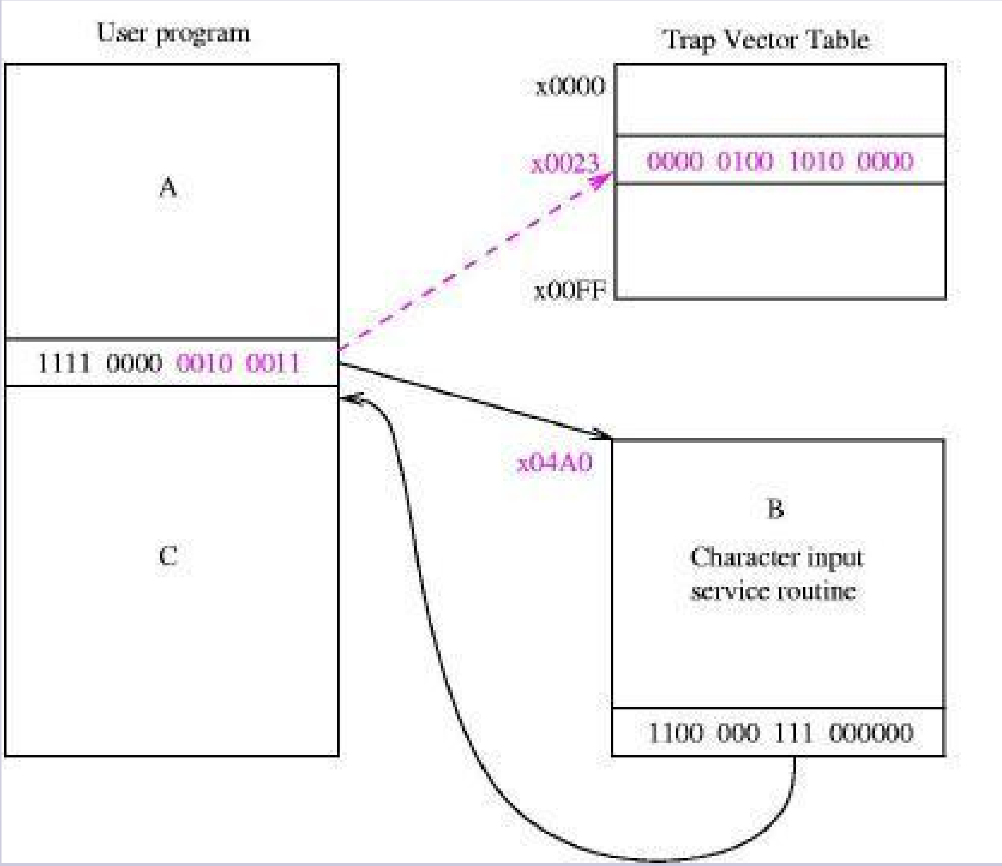
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	0	0	trapvector8							

    - *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*

- **RET – return instruction**
  - The TSR ends with the RET instruction
    - *PC ← (R7) ; the program now picks up where it left off*

TRAP Example

- **Trap Vector Table**
  - **Or System Control BLock**
  - **In LC-3**
    - ***8 bits specify one of 256 locations (x0000 to x00FF)***
    - ***The location contains the address of the TRAP service routine.***
- **TRAP & Interrupts**
  - **Similar mechanisms**
  - **A TRAP is an instruction (event internal to a program).**
  - **An interrupt is external to a program (from an I/O device)**
  - **Both invoke a supervisor service routine.**



Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

Character Output TSR (OUT)

```
        .ORIG     X0430           ; System call starting address
        ST   R1, SaveR1     ; R1 will be used for polling

; Write the character
TryWrite  LDI   R1, DSR           ; Get status
        BRzp      TryWrite        ; bit 15 = 1 => display ready
WriteIt   STI   R0, DDR           ; Write character in R0

; Return from TRAP
Return    LD    R1, SaveR1        ; Restore registers
        RET              ; Return (actually JMP R7)
DSR .FILL         xFE04           ; display status register
DDR .FILL         xFE06           ; display data register
SaveR1    .BLKW    1
        .END
```

HALT TSR

- **Clears the RUN latch MCR[15]:**

```
        .ORIG   XFD70           ; System call starting address
        ST      R0, SaveR0      ; Saves registers affected
        ST      R1, SaveR1      ; by routine
        ST      R7, SaveR7      ;

; Print message that machine is halting
        LD      R0, ASCIINewLine
        TRAP    x21             ; Set cursor to new line
        LEA     R0, Message     ; Get start of message
        TRAP    x22             ; and write it to monitor
        LD      R0, ASCIINewLine
        TRAP    x21

; Clear MCR[15] to stop the clock
        LDI     R1, MCR         ; Load MC register to R1
        LD      R0, MASK        ; MASK = x7FFF (i.e. bit 15 = 0)
        AND     R0, R1, R0      ; Clear bit 15 of copy of MCR
        STI     R0, MCR         ; and load it back to MCR
```

HALT TSR (cont.)

```
13   ; Return from the HALT routine
14   ; (how can this ever happen, if the clock is stopped on line 12??)
15   ;
16          LD    R7, SaveR7      ; Restores registers
17          LD    R1, SaveR1      ; before returning
18          LD    R0, SaveR0
19          RET                   ; JMP R7
1A
1B   ; constants
1C   ASCIINewLine      .FILL       x000A
1D   SaveR0   .BLKW    1
1E   SaveR1   .BLKW    1
1F   SaveR7   .BLKW    1
20   Message .STRINGZ        "Halting the machine"
21   MCR       .FILL       xFFFE
22   MASK      .FILL       x7FFF
23          .END
```


Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

Saving & restoring registers

- **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.**
    - ***e.g. in the HALT routine, which has itself been called by another program, the caller knows that it has precious cargo in R7, which will be overwritten by the TRAP instructions (why??), so it saves R7 to memory at the start of the routine, and restores it from memory before returning to the main program.***
    - ***This is known as "caller save"***

Saving & restoring registers (cont.)

- Other times it will be the called program ("callee") that knows what registers it will be using to carry out its task.
  - *again in the HALT routine, R0 and R1 are used as temporary working space to hold addresses, masks, ASCII values, etc., so they are both saved to memory at the start of the routine, and restored from memory before returning to the main program.*
  - *This is known as "callee save"*
- This applies not only to trap service routines but to all subroutine calls, and is the basis of what are called "scope rules" in higher level languages.

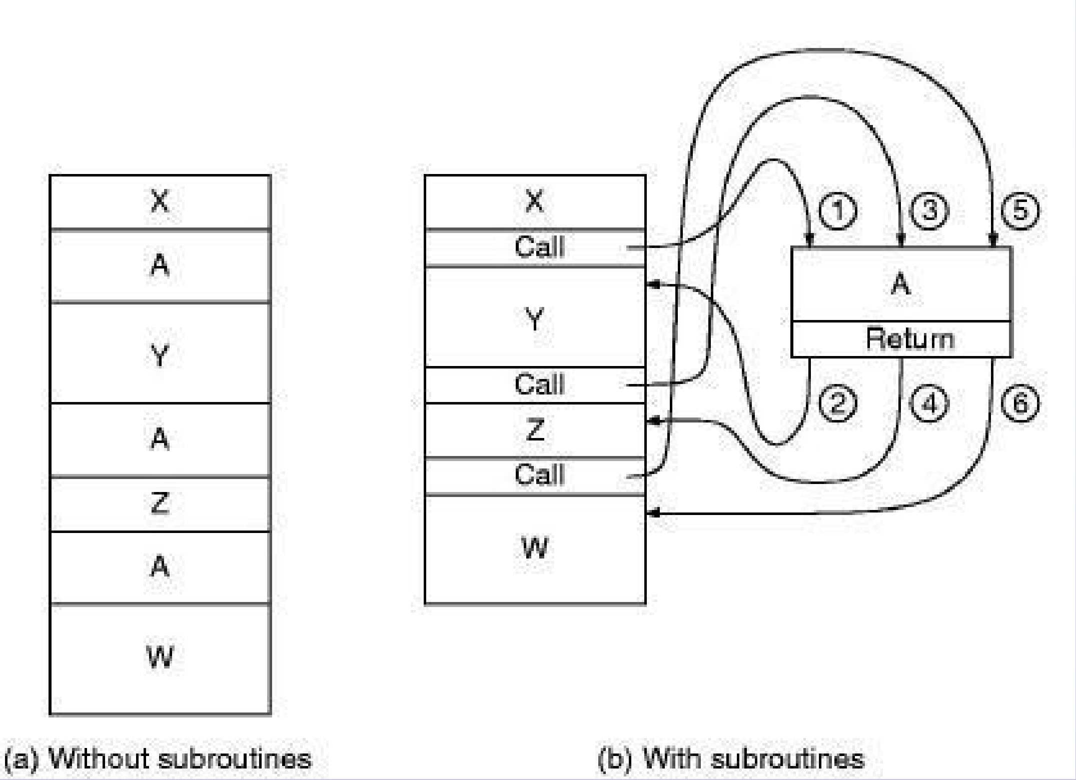
Subroutines

- **Used for**
  - **Frequently executed code segments**
  - **Library routines**
  - **Team-developed systems**
    - ***in other words, all the same reasons for using subroutines in higher level languages, where they may be called functions, procedures, methods, etc.***
- **Requirements:**
  - **Pass parameters and return values, via registers or memory.**
  - **Call from any point & return control to the same point.**

Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

The Call / Return mechanism

- **The figure illustrates the execution of a program comprising code fragments A, W, X, Y and Z.**
  - ***Note that fragment A is repeated several times, and so is well suited for packaging as a subroutine:***



Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

Jump to Subroutine : JSR/JSRR

- **JSR: jump to subroutine (PC-Relative)**

15 14 13 12	11	10 9 8 7 6 5 4 3 2 1 0
0 1 0 0	1	Address eval. bits
JSR(R)	A	

  - A = IR[11] specifies the addressing mode
  - JSR: IR[11] = 1:
    - *R7 ← (PC)   i.e. PC is saved to R7*
    - *PC ← (PC) + Sext(IR[10:0])*
    - *i.e PC-Relative addressing, using 11 bits => label can be within +1024 / -1023 lines of JSR instruction*

JSR/JSRR (cont.)

- **JSRR: jump to subroutine (Base+Offset)**

15 14 13 12	11	10 9	8 7 6	5 4 3 2 1 0
0 1 0 0	0	0 0	BaseR	0 0 0 0 0 0
JSR(R)	A			

- **JSRR: IR[11] = 0:**
  - *R7 ← (PC)   i.e. PC is saved to R7*
  - *PC ← (BaseR)*
  - *i.e Base+Offset addressing, with offset = 0*

- **In both cases, the RET instruction restores the PC from R7, and the calling program resumes.**

Subroutine call example

```
; Calling program
        .ORIG x3000
        LD   R1, num1
        LD   R2, num2
        JSR       multi
        ST   R3, prod
        HALT
;
; Input data & result
num1    .FILL     x0006
num2    .FILL     x0003
prod    .BLKW     1
```

```
; Subroutine multi
; Multiply 2 positive numbers
; Parameters:
; In: R1, R2;   Out: R3
;
multi   AND     R3, R3, #0
        ADD     R4, R1, #0
        BRzzero
loop    ADD     R3, R2, R3
        ADD     R1, R1, # -1
        BRp     loop
zero    RET
        .END
```

Library Routines

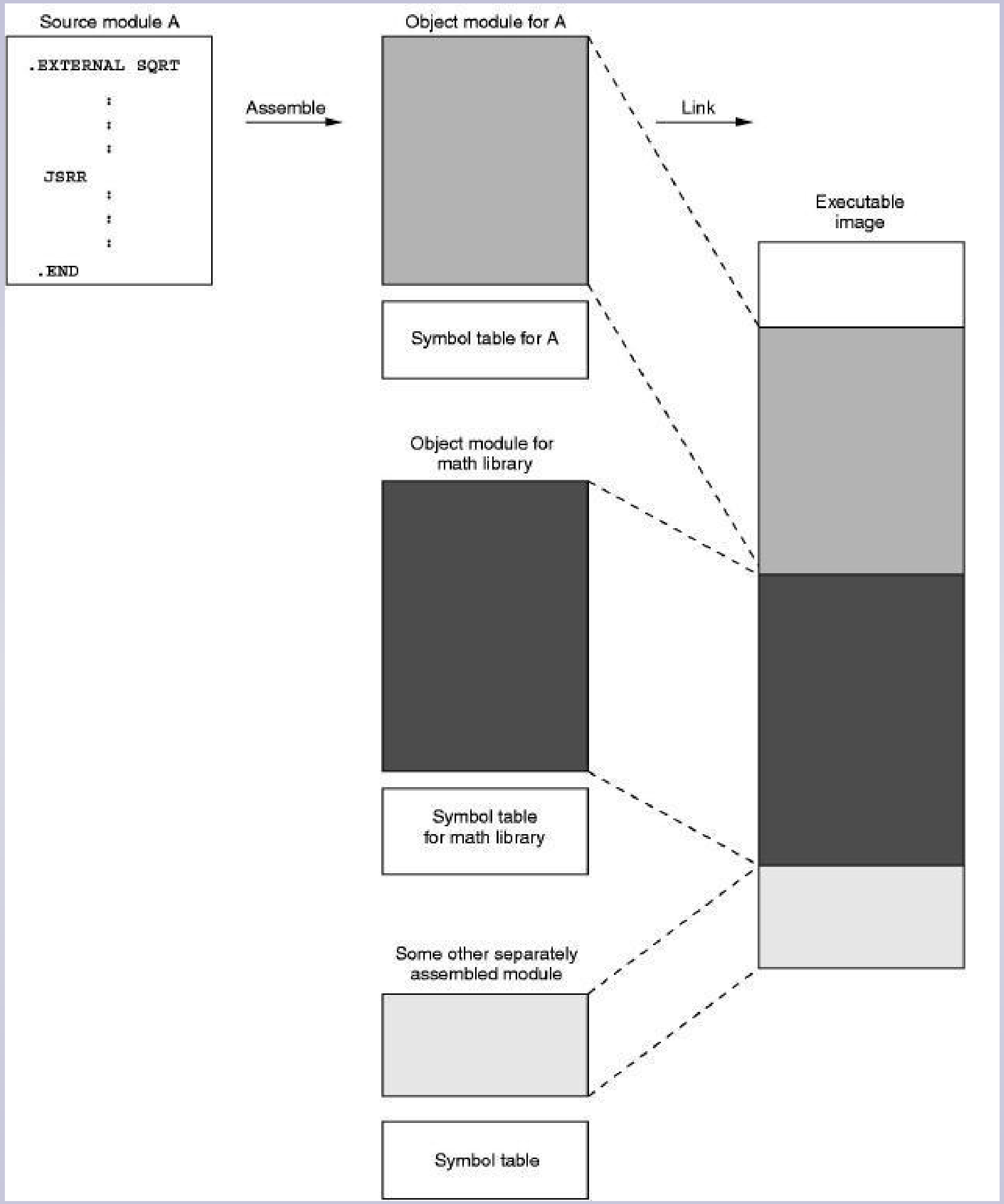
- **Library**
  - **A set of routines for a specific domain application.**
  - **Example: math, graphics, GUI, etc.**
  - **Defined outside a program.**

- **Library routine invocation**
  - **Labels for the routines are defined as *external*. In LC-3:**

    .External   Label
  - **Each library routine contains its own symbol table.**
  - **A linker resolves the external addresses before creating the executable image.**

Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside

Linking multiple files



Slides prepared by Walid A. Najjar & Brian J. Linard, University of California, Riverside