ECE 220
Lectures 03: Repeated Code
TRAPs and Subroutines

Slides by Sayan Mitra,, Sanjay Patel, Yih-Chun Hu

Outline

- Chapter 9
- Repeated code: TRAPs and Subroutines

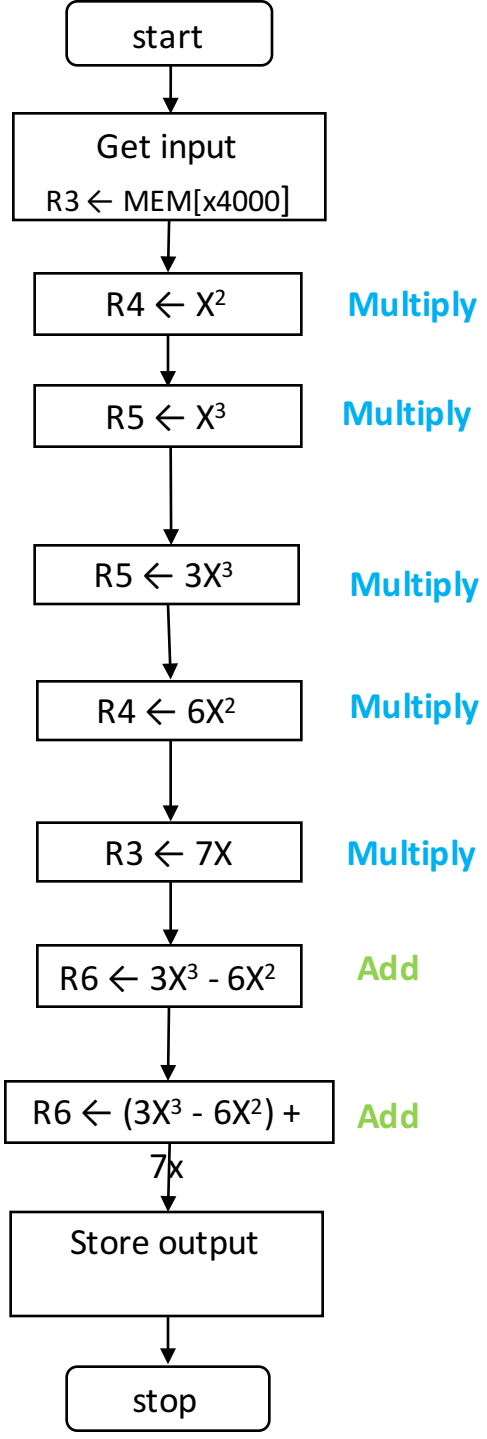
- Key concepts
  - Lookup table: for starting address of subroutines/TRAPS (vector table)
  - Preserving register and PC values
- Instructions
  - TRAP
  - RET
  - JSR, JSRR

Observation 1

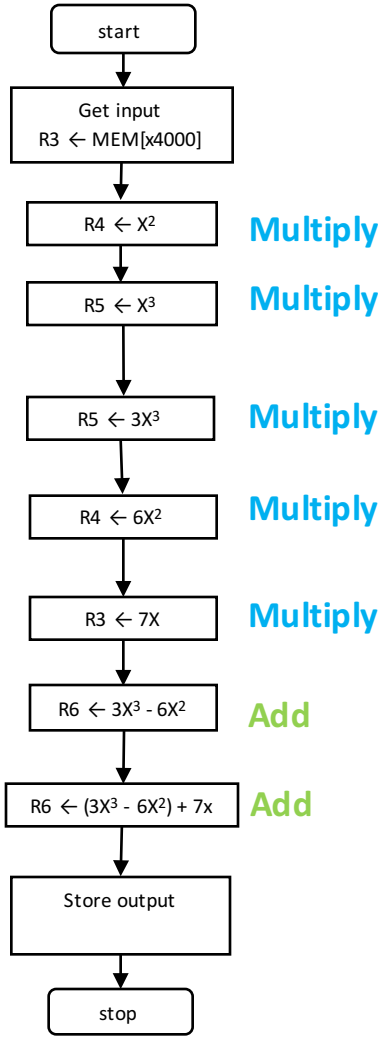
Example problem: Compute $y=3x^3-6x^2+7x$ for any input $x > 0$

Programs have lots of repetitive code fragments

```
; multiply  R0 ←R1 * R2
MULT      AND R0, R0, #0      ; R0 = 0
LOOP      ADD R0, R0, R2      ; R0 = R0 + R2
          ADD R1, R1, #-1     ; decrease counter
          BRp LOOP
```



Implementation Option 1




```
;; LC-3 Assembly Program
.ORIG x3000
LDI R3, Xaddr;  R3 ← x
ADD R1, R3, #0;
; Multiply R4 ← R1 * R3 x^2
...
...
; Multiply R5 ← R4 * R3 x^3
...
; Multiply R5 ← R5 * 3 (3x^3)
...
; Multiply R4 ← 6 * R4
```

Issues ?

*Sometimes programs are compiled to look like this for better performance: inline functions

3 virtues of great programmer --- Larry Wall (Author of Perl)

- **Laziness**: write labor-saving programs that other people will find useful and document what you wrote so you don't have to answer so many questions about it.
- **Impatience**: write programs that don't just react to your needs, but actually anticipate them.
- **Hubris**: write programs that other people won't want to say bad things about.

Another example: Code (from last lecture) for reading characters from keyboard

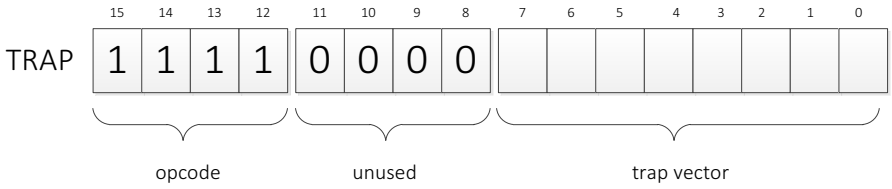
```
START           LDI     R1,
KBSR_ADDR
                BRzp    START

                LDI     R0,
KBDR_ADDR
                BRnzp   NEXT_TASK
                ...
KBSR_ADDR       .FILL     xFE00
KBDR_ADDR       .FILL     xFE02
```

- Very common… would be replicated often
- Too many specific details for most programmers
  - know address of KBDR (xFE02) and KBSR (xFE00)
  - use the registers correctly (polling, data format)
- Improper usage could breach security of the system!
- Idea: Make this part of the (operating) system
- In general, provide "pieces of code" or *subroutines* or *system calls* or *service routine* that do all of these low-level tasks correctly
  - User **invokes or calls** subroutine
  - Operating system code performs operation
  - **Returns** control to user program

How to make this idea work?

User program **invokes or calls** subroutine; OS code performs operation; **Returns** control to user program

- The actual code of the service routine
- Mechanism for invocation
  - TRAP Instruction, e.g., TRAP x20
  - TRAP vector (8 bits)
  - How to find address service routine?



Address	Contents	Comments
x0000		;system space;
…		
x00FF		
x0400		; code for GETC
….		
x0430		; code for OUT
…		
x3000		; user program
…		
	TRAP x20	; call to
xFE00		; Device registers

TRAP Vector Table for LC3

vector	address	symbol	routine
…			
x20	x0400	GETC	read a single character (no echo)
x21	x0430	OUT	output a character to the monitor
x22	x0450	PUTS	write a string to the console
x23	x04A0	IN	print prompt to console, read and echo character from keyboard
X23	x04E0	PUTSP	write a string to the console; two chars per memory location
x25	xFD70	HALT	halt the program
…			

Look-up table decouples names of subroutines (GETC) from the location of its implementation in memory

```
1. exit                 801d4a74 T
2. fork                 801d7980 T
3. read                 801eb584 T
4. write                801eb958 T
5. open                 800b13a4 T
6. close                801ccab4 T
7. wait4                801d56bc T
9. link                 800b18e8 T
10. unlink              800b1ff0 T
12. chdir               800b0c60 T
13. fchdir              800b0af0 T
14. mknod               800b14bc T
15. chmod               800b2b40 T
16. chown               800b2c9c T
18. getfsstat           800b088c T
20. getpid              801dc20c T
23. setuid              801dc4c0 T
24. getuid              801dc290 T
25. geteuid             801dc2a0 T
26. ptrace              801e812c T
27. recvmsg             8020a8fc T
28. sendmsg             8020a444 T
29. recvfrom            8020a528 T
30. accept              80209dfc T
31. getpeername         8020abc8 T
32. getsockname         8020ab18 T
33. access              800b24ac T
34. chflags             800b2928 T
35. fchflags            800b29f0 T
36. sync                800b0320 T
37. kill                801dfdcc T
39. getppid             801dc214 T
41. dup                 801cab04 T
42. pipe                801edbe4 T
43. getegid             801dc318 T
46. sigaction           801deee8 T
47. getgid              801dc308 T
48. sigprocmask         801df42c T
49. getlogin            801dd0e8 T
50. setlogin            801dd160 T
51. acct                801c54ec T
52. sigpending          801df5d0 T
53. sigaltstack         801dfd10 T
54. ioctl               801ebd1c T
55. reboot              801e8090 T
56. revoke              800b43f8 T
57. symlink             800b1b58 T
```

For comparison… some iOS system calls

What do we need to make this work?

User program **invokes or calls** subroutine; OS code performs operation; **Returns** control to user program

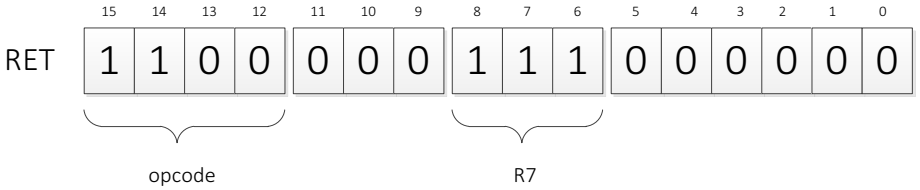
- The actual code of the service routine
- Mechanism for invocation
  - TRAP Instruction, e.g., TRAP x20
  - TRAP vector
  - MAR ←ZEXT[trapvector]
  - MDR ←MEM[MAR]
  - PC←MDR
- How to return to user program after execution of OUT completes?

Address	Contents	Comments
x0000		;system space;
x0020	**x0400**	**; Trap vector table**
x00FF		; End of trap vecto
x0400		; code for GETC
….		
x0430		; code for OUT
…		
x3000		; user program
…		
	TRAP x20	; call to
xFE00		; Device registers

What do we need to make this work?

User program **invokes or calls** subroutine; OS code performs operation; **Returns** control to user program

- The actual code of the service routine
- Mechanism for invocation
  - TRAP Instruction, e.g., TRAP x20
  - TRAP vector
  - MAR ←ZEXT[trapvector]
  - MDR ←MEM[MAR]
  - R7 ← PC
  - PC←MDR
- Mechanism for resuming user program
  - RET ≡ JMP R7



Address	Contents	Comments
x0000		;system space;
x0020	x0400	; Trap vector table
x00FF		; End of trap vecto
x0400		; code for GETC
….	RET	
x0430		; code for OUT
…		
x3000		; user program
…		
	TRAP x20	; call to
xFE00		; Device registers

Putting it all together: 4 Things make TRAPs work

1. TRAP instruction
    - used by program to transfer control to OS subroutines
    - 8-bit trap vector names one of the 256 subroutines
2. Trap vector table: stores starting addresses of OS subroutines
    - stored at x0000 through x00FF in memory
3. A set of OS subroutines
    - part of operating system -- routines start at arbitrary addresses (convention is that system code is below x3000) up to 256 routines
4. A linkage back to the user program (RET)
    - want execution of the user program to resume immediately after the TRAP instruction

Subroutines

- Hide details of code and package them with an interface
  - Abstract away details
  - Expose only input/output interface
  - **Arguments** and **return values**
- Why is this a good idea in programming?
  - Reuse; shorter programs
  - Protect; system resources and IP from users
  - Simplify; code comprehension
  - Teamwork; allows multiple developers to work on different pieces; libraries
- TRAPs are examples of OS subroutines

JSR and JSRR and RET

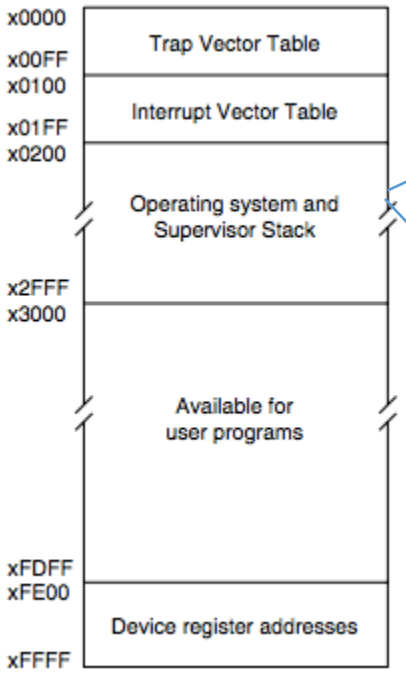
```
JSR:
        R7 <- PC;
        PC <- PC + SEXT(PCOffset11);

JSRR:
        R7 <- PC;
        PC <- BaseR;

RET:
        PC <- R7;
```

Question:

Now that we have TRAPS, and subroutines, can we go back and program using subroutines as planned?

No.

Example of OUT from LC-3




```
                .ORIG   x0430
                ST R1, SaveR1
Ready           LDI     R1, DSR_ADDR
                BRzp    READY
Write           STI     R0, DDR_ADDR
Return  LD      R1, SaveR1
                RET
DSR_ADDR        .FILL       xFE04
DDR_ADDR        .FILL       xFE06
SaveR1  .FILL           x0000
```

Observation 2

- When TRAP is called, PC is saved in R7 before service routine starts to execute so that after it finishes, the user program can resume from the right address
- What if R7 had important information? What about other registers that are used internally by the subroutine? (E.g., R1 in OUT)
- Bookmark and more generally bookkeeping!
- We need to save (useful) values in registers in memory before invocation
  - Caller of service routine can save (and restore): **Caller-save**
  - Called service routine saves (and restore): **Callee-save**
- Saving and restoring values of registers is an example of a task computers need to perform in **context switching**

```
; Caller-save user program
…
ST R0, SaveR0        ; store R0 in memory
ST R7, SaveR7        ; store R7 in memory
GETC                 ; call TRAP which
                     ; destroys R0 and R7
LD R7, SaveR7        ; restore R7
…                    ; consume input in R0
LD R0, SaveR0        ; restore R0
…
HALT

SaveR0 .BLKW 1
SaveR7 .BLKW 1
```

Multiply

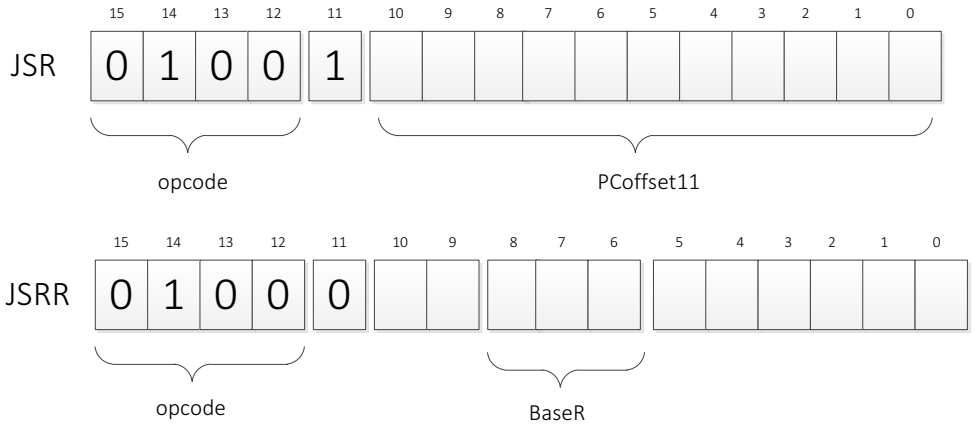
- Write an LC-3 assembly language program to calculate:

$$y = ax^2 + bx + c;$$

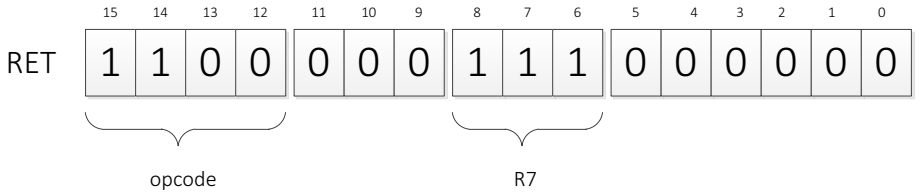
Summary of Lecture 3

- Need: reuse code (MULT, OUT), hide details from user (xFE02), protect system critical resources (e.g., access to KBDR)
- Idea: package repeated code into subroutines or system routines
- Implementation
  - Invocation: TRAP x21 (R7 ← PC, PC ← x0430)
  - System routine code for OUT at x0430 (in OS space) executes
  - RET (PC ← R7); resume execution of user program
- Side effect: May lose useful information in registers (R7, R1)
- Caller save or callee save the relevant registers

JSR and JSRR



TEMP←PC
If (IR[11] == 0) PC←BaseR
Else PC←PC+SEXT(IR[10:0])
R7←TEMP



RET ≡ JMP R7
PC ← R7

Examples: a Subtraction subroutine

; SUBTR

A Multiplication subroutine

; IN

; OUT

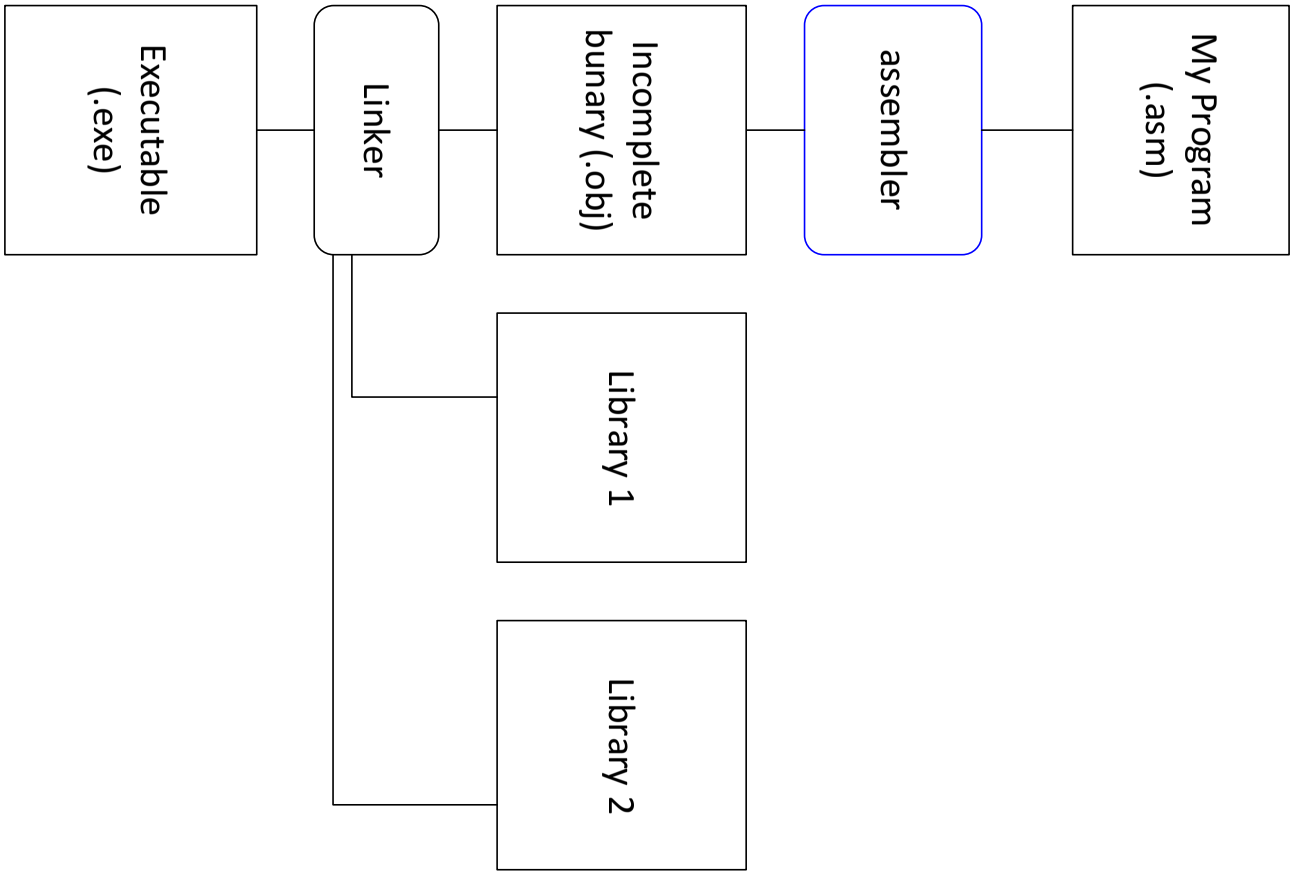
;

Subroutine abstraction

- Interface specifies
  - Input: type (ASCII, int) and location (Registers)
  - Output: type and location
- Optionally used resources (Registers)

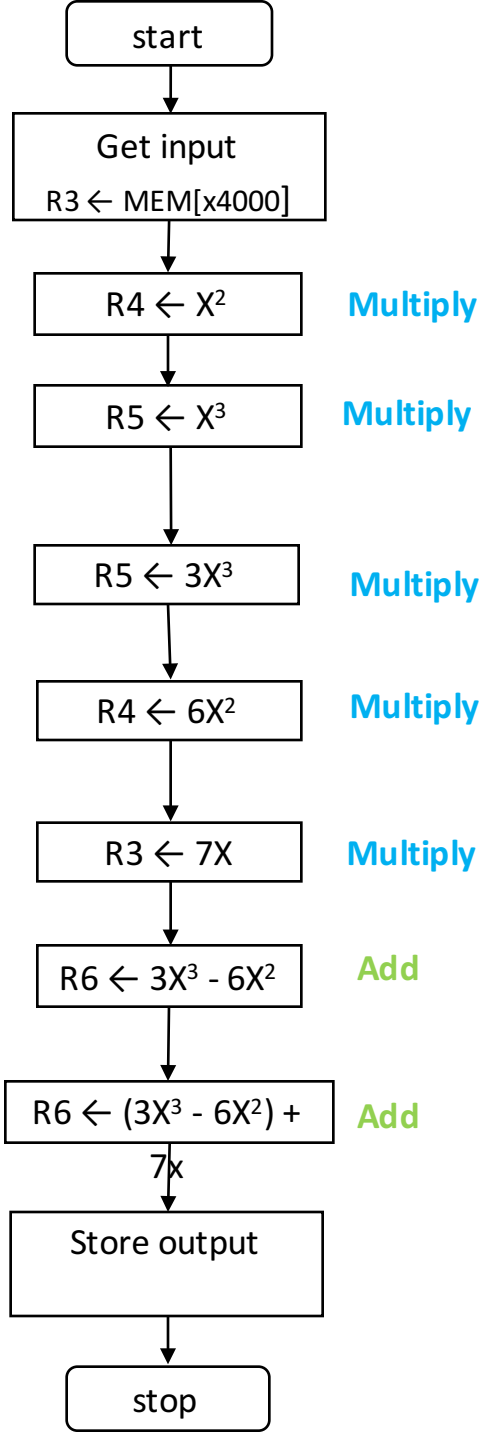
- Does not specify?

Libraries and compilation



Exercise

Compute $y=3x^3-6x^2+7x$ for any input $x > 0$



Exercises

- Write an LC-3 assembly language program to calculate: $y = ax^2 + bx + c$;
- Lab exercise: encode a given string by shifting each character by an offset
- Count the number of occurrences of the word “code” in a given string of text.

Nested Subroutines

Can a subroutine call itself?

Summary and tradeoffs

- Idea: package repeated code into subroutines: easier to program, debug, maintain, share
  - TRAP subroutines addressed by their trapvector using the trapvector table
- Jump and return accomplished by setting PC values
- Side effect: May lose useful information in registers
  - Caller-save or callee-save register values
- But, **context switch** (saving and restoring registers) incurs additional cost---cycles, memory accesses
  - Inline functions
- **Next:** stack data structure