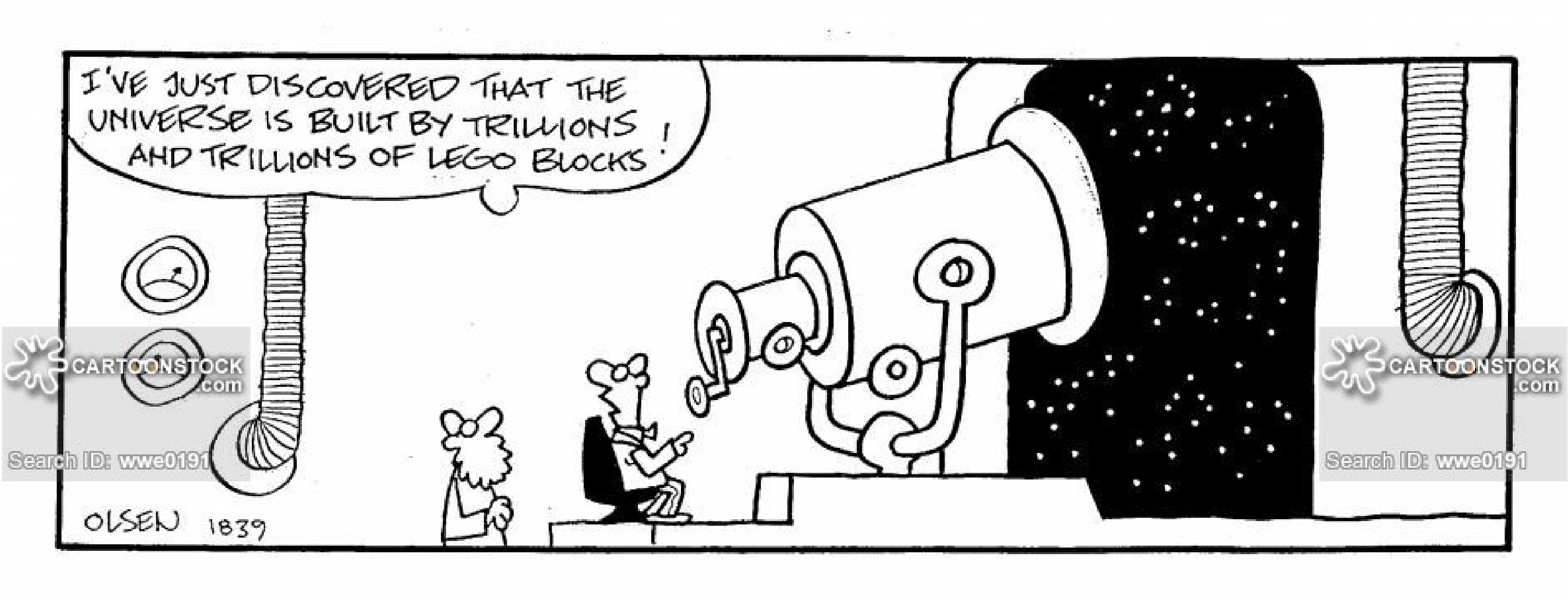
ECE 220 Computer Systems & Programming

Lecture 3 – Repeated Code: TRAPs and Subroutines

January 22, 2019

- MP1 due Thursday, 1/24, by 10pm
- Schedule mock quiz for extra credit

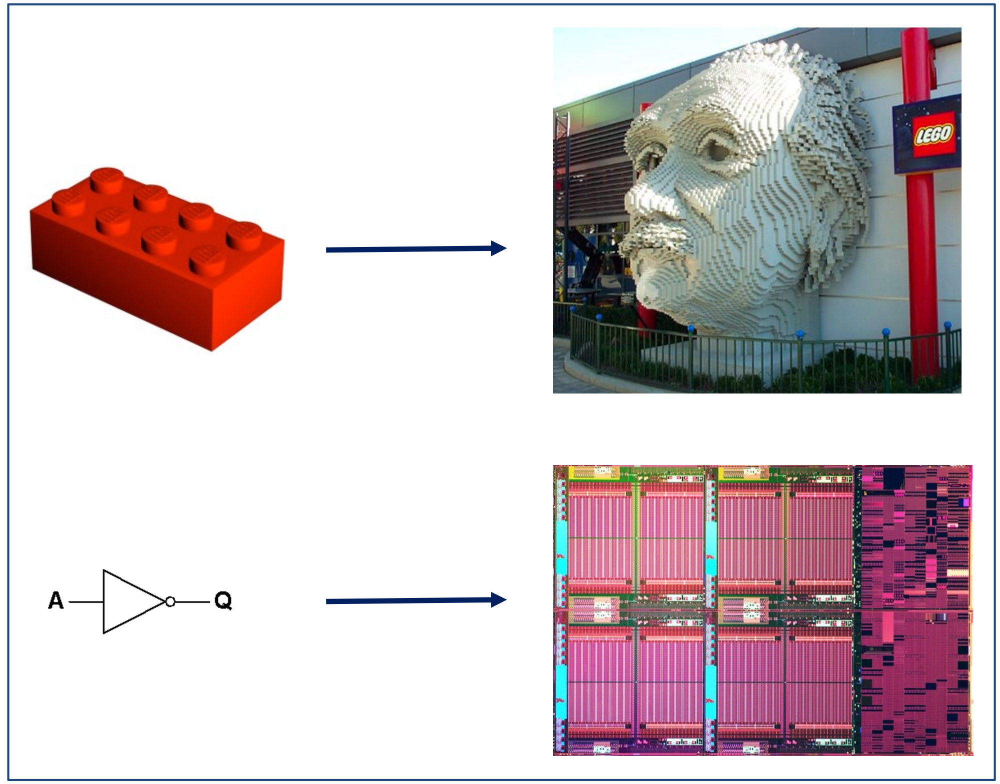
Key Concept: Abstraction



Key Concept: Abstraction

Create building blocks that are tightly specified. Abstract way their details to a simple interface. And then use them to build more complex things.

Then optimize the building blocks like crazy…



Outline

- Chapter 9
- Repeated code: TRAPs and Subroutines

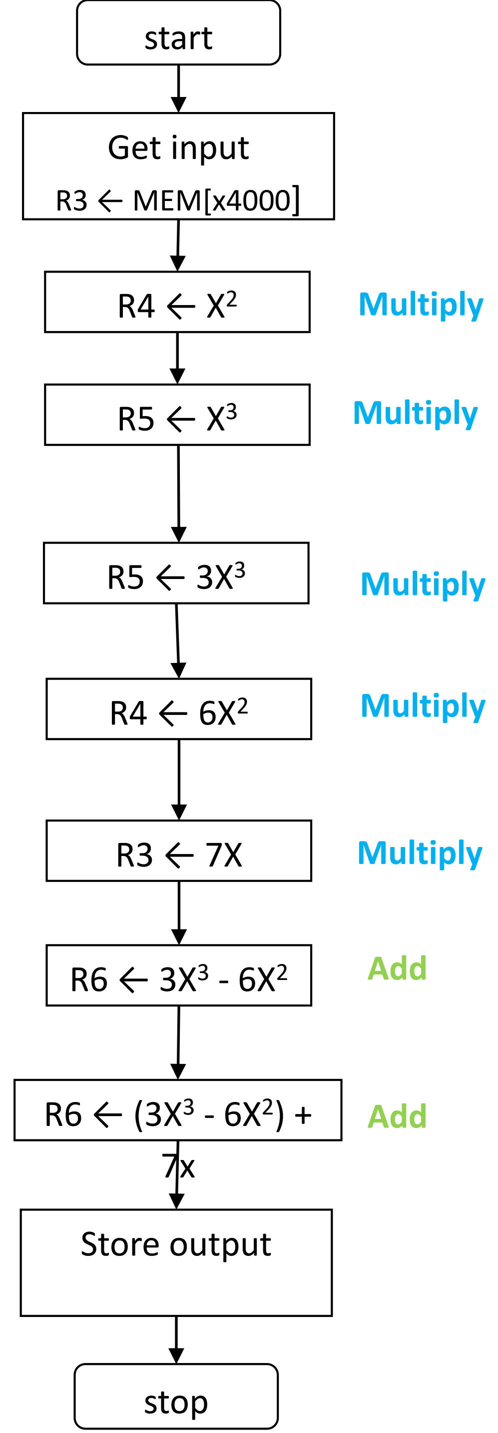
- Key concepts
  - Lookup table: for starting address of subroutines/TRAPS (vector table)
  - Preserving register and PC values
- Instructions
  - TRAP
  - RET
  - JSR, JSRR

Observation 1

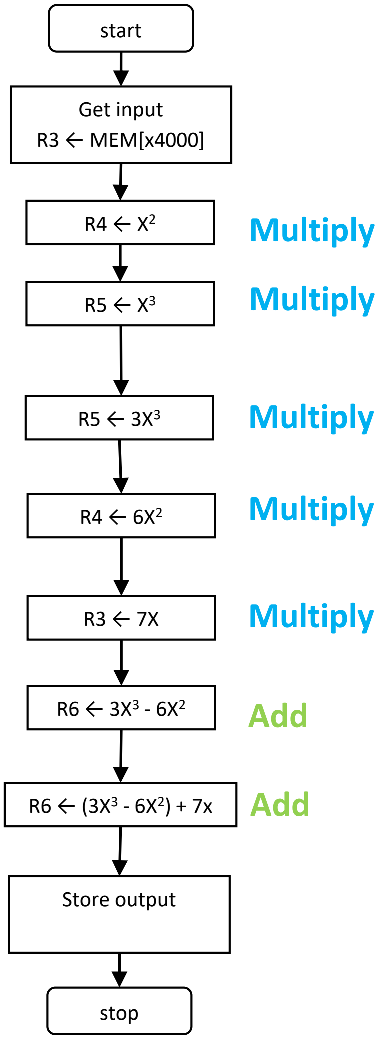
Example problem: Compute $y=3x^3-6x^2+7x$ for any input x > 0

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



Issues ?

```
;; 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
```

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

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

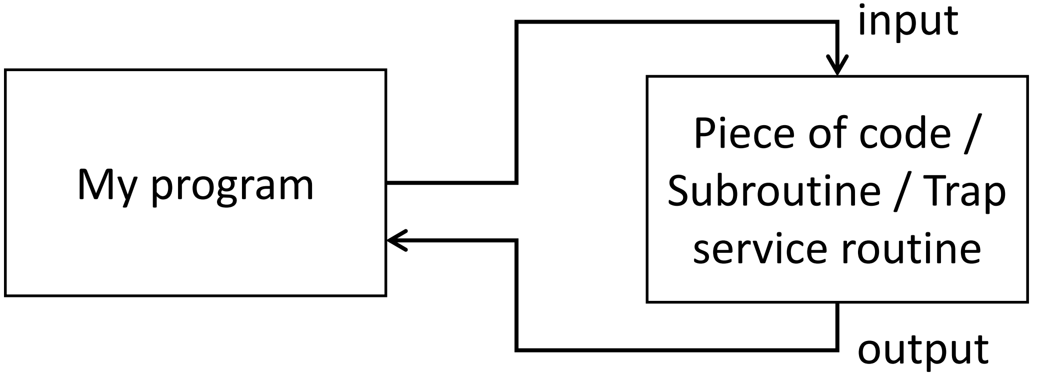
```
START    LDI       R1, KBSR_ADDR
                   BRzp     START

         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

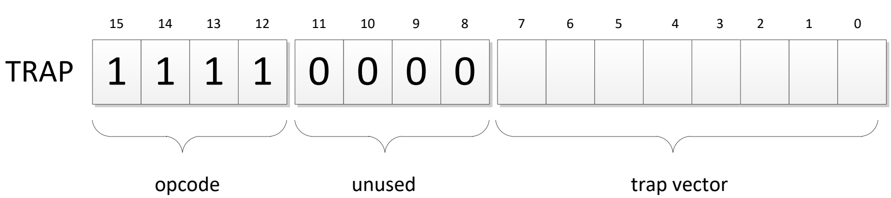
- Package these pieces of code as part of the (operating) system
- In general, provide "pieces of code" or *subroutines, functions, methods, procedures, or service routine* that do some specific subtasks
- User **invokes or calls** subroutine
- Subroutine code performs operation / task
- **Returns** control to user program with no other unexpected changes

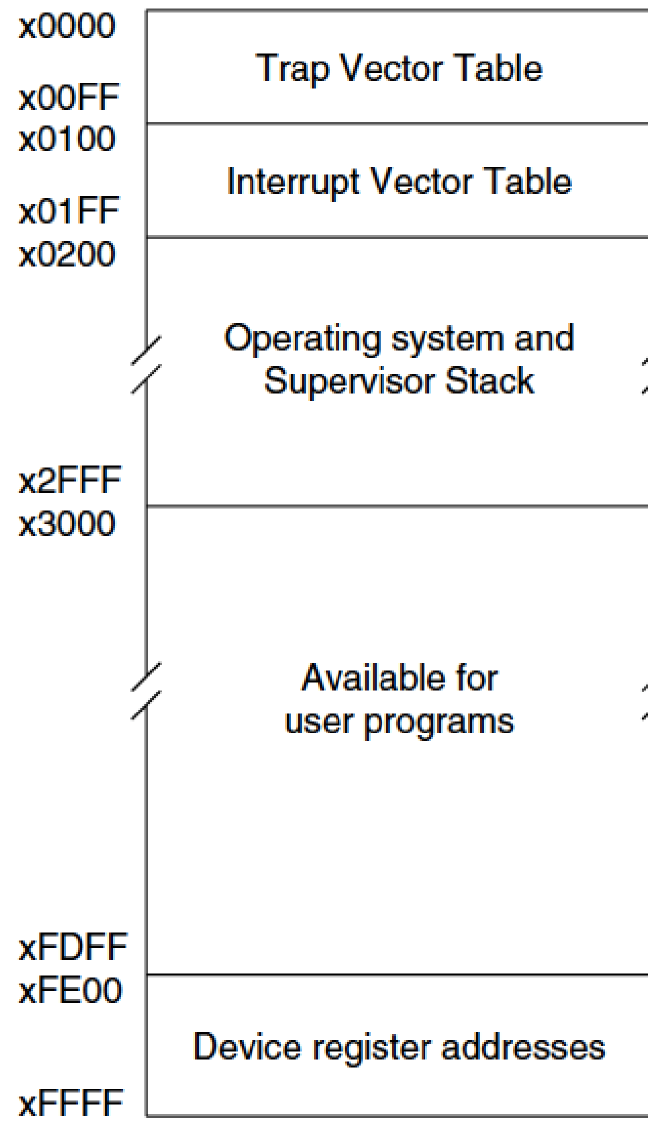


How to make this idea work?

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 x23
  - TRAP vector (8 bits)
  - How to find address service routine?





TRAP Vector Table for LC3

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

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

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 x23
  - 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 spac
X0023	**x046**	**; Trap ve**
x00FF		End
x0463		
....		
x04...		
...		
x3000		; user
...		
	TRAP x23	; call to
xFE00		; Device registers

What do we need to make this work?

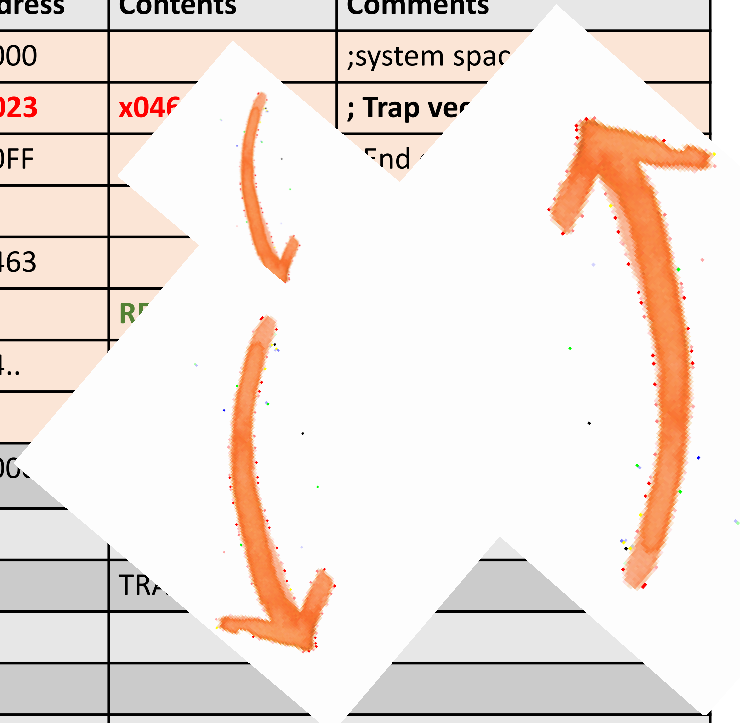
User program **invokes or calls** subroutine; OS code performs operation; **Returns** control to user program

- 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

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

opcode (bits 15–12); R7 (bits 8–6)

Address	Contents	Comments
x0000		;system spac
x0023	**x046**	**; Trap ve**
x00FF		;End
x0463		
....	**R**	
x04..		
…		
x300		
…		
	TR	
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

Example 9.1

```
;Example 9.1
;Converting Uppercase to Lowercase Letter
;Exits when press 7

        .ORIG x3000
        LD    R2,TERM  ; Load -7
        LD    R3,ASCII ; Load ASCII difference
AGAIN   TRAP  x23      ; Request keyboard input
        ADD   R1,R2,R0 ; Test for terminating
        BRz   EXIT     ; character
        ADD   R0,R0,R3 ; Change to lowercase
        TRAP  x21      ; Output to the monitor
        BRnzp AGAIN    ; ... and do it again!
TERM    .FILL xFFC9    ; FFC9 = -7
ASCII   .FILL x0020
EXIT    TRAP  x25      ; Halt
        .END
```

What do we need to make this work?

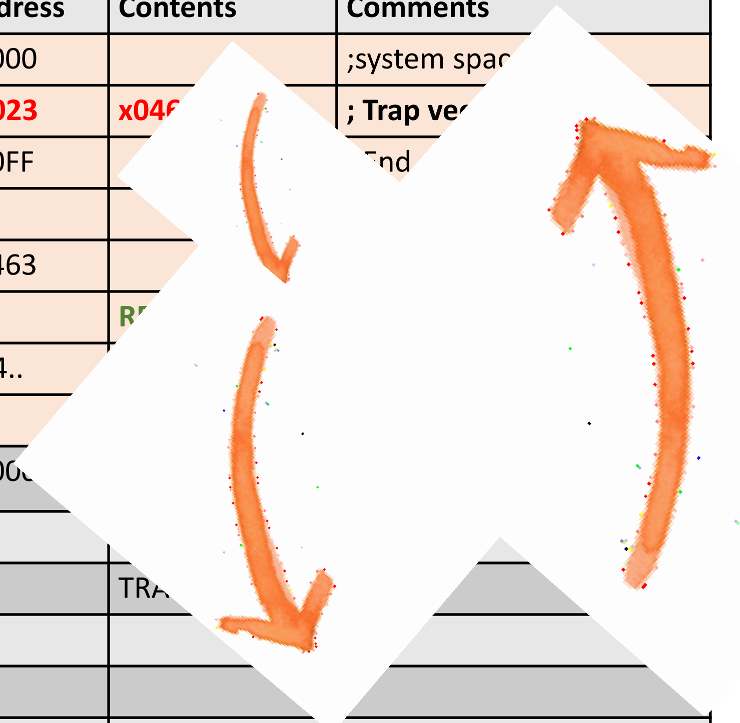
User program **invokes or calls** subroutine; OS code performs operation; **Returns** control to user program

- 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

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

$\underbrace{1100}_{\text{opcode}}$ $\underbrace{111}_{\text{R7}}$

Address	Contents	Comments
x0000		;system spac
x0023	**x046**	**; Trap ve**
x00FF		nd
x0463		
….	R	
X04..		
…		
x300		
…		
	TRA	
xFE00		; Device registers



TRAP Example (Needs special attention)

```
.ORIG x3000

AND R0, R0, #0
ADD R0, R0, #5     ;init R0 and set it to 5
LD R7, COUNT       ;Initialize to 10

IN                 ;same as 'TRAP x23'

ADD R0, R0, #1     ;increment R0
ADD R7, R7, #-1    ;decrement COUNT

HALT
.END
COUNT   .FILL #10
```

- Question: What could go wrong?
- What are the values in R0 and R7 before and after IN statement?

Suggested approach:

- 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
IN                     ; 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
```

Subroutines

Service routines (TRAP) provide 3 main functions

- Shield programmers from system-specific details
- Write frequently-used code just once
- Protect system recourses from malicious/clumsy programmers

Subroutines provide the same functions for non-system (user) code

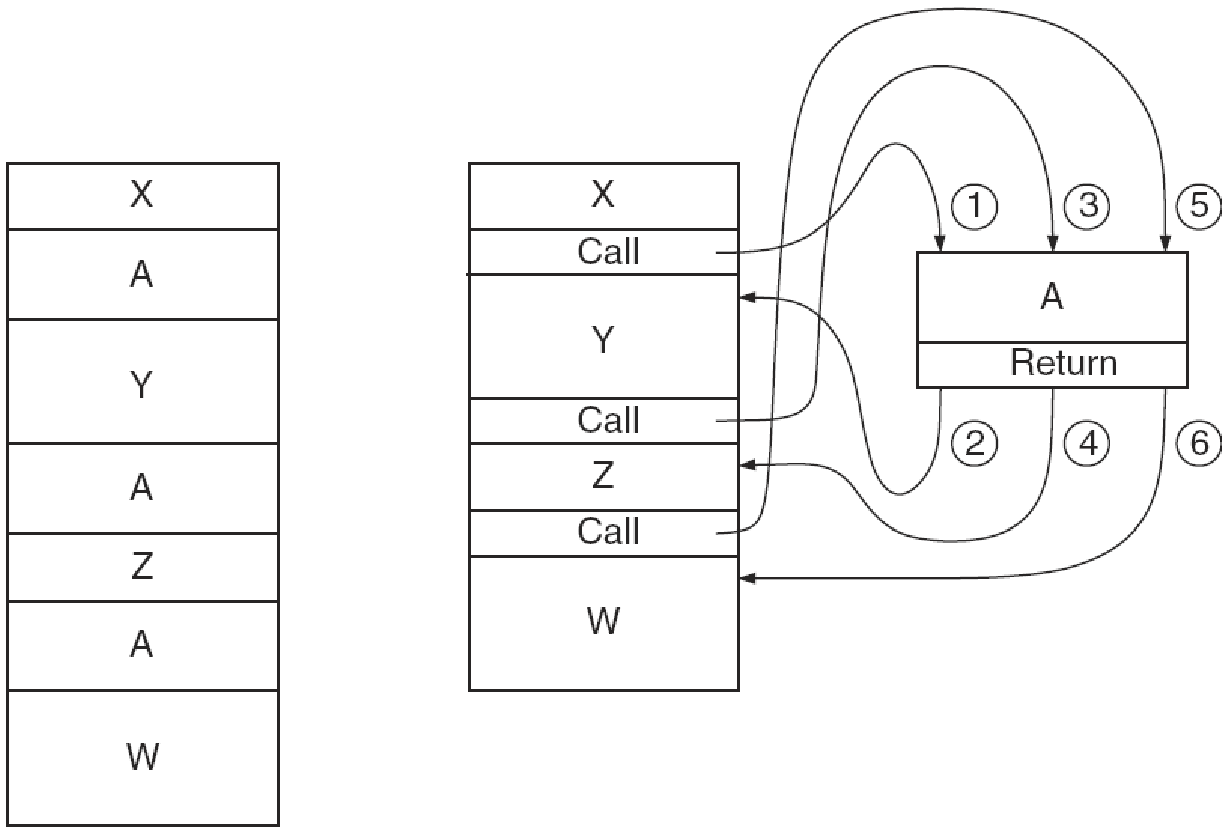
-
-
-
-

- **What are some of the reasons to use subroutines?**

Subroutines

- A sequence of instructions that performs a specific task (and nothing else---no side effects). This unit can then be used in programs wherever that particular task should be performed.
- Hide details of code and package them with an interface
  - Abstract away details
  - **Needs only Arguments** and **return values**
- Why is this a good idea in programming?
  - Reuse; shorter programs
  - Simplify; code comprehension
  - Teamwork; allows multiple developers to work on different pieces; libraries
- TRAPs are examples of OS subroutines

Figure 9.7



(a) Without subroutines (b) With subroutines

JSR and JSRR

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JSR	0	1	0	0	1											
	opcode					PCoffset11										

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

R7←PC

If (IR[11] == 0) PC←BaseR

Else PC←PC+SEXT(IR[10:0])

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RET	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0
	opcode							R7								

RET ≡ JMP R7

PC ← R7

Compare Example 9.4 and 9.8

(observe using subroutine)

Subroutine Examples

- **Can you find the bugs in the following piece of code?**

```
; SUBTR subroutine computes difference of two 2's complement numbers
; IN: R1, R2
; JSR SUBTR
OUT: R0 <- R1-R2
SUBTR   NOT R2, R2
        ADD R7, R2, #1
        ADD R0, R1, R7
        RET
```

➢ **How should we compute $2x^2 - 3x + 1$?**

Examples: a Subtraction subroutine

; SUBTR

A Multiplication subroutine

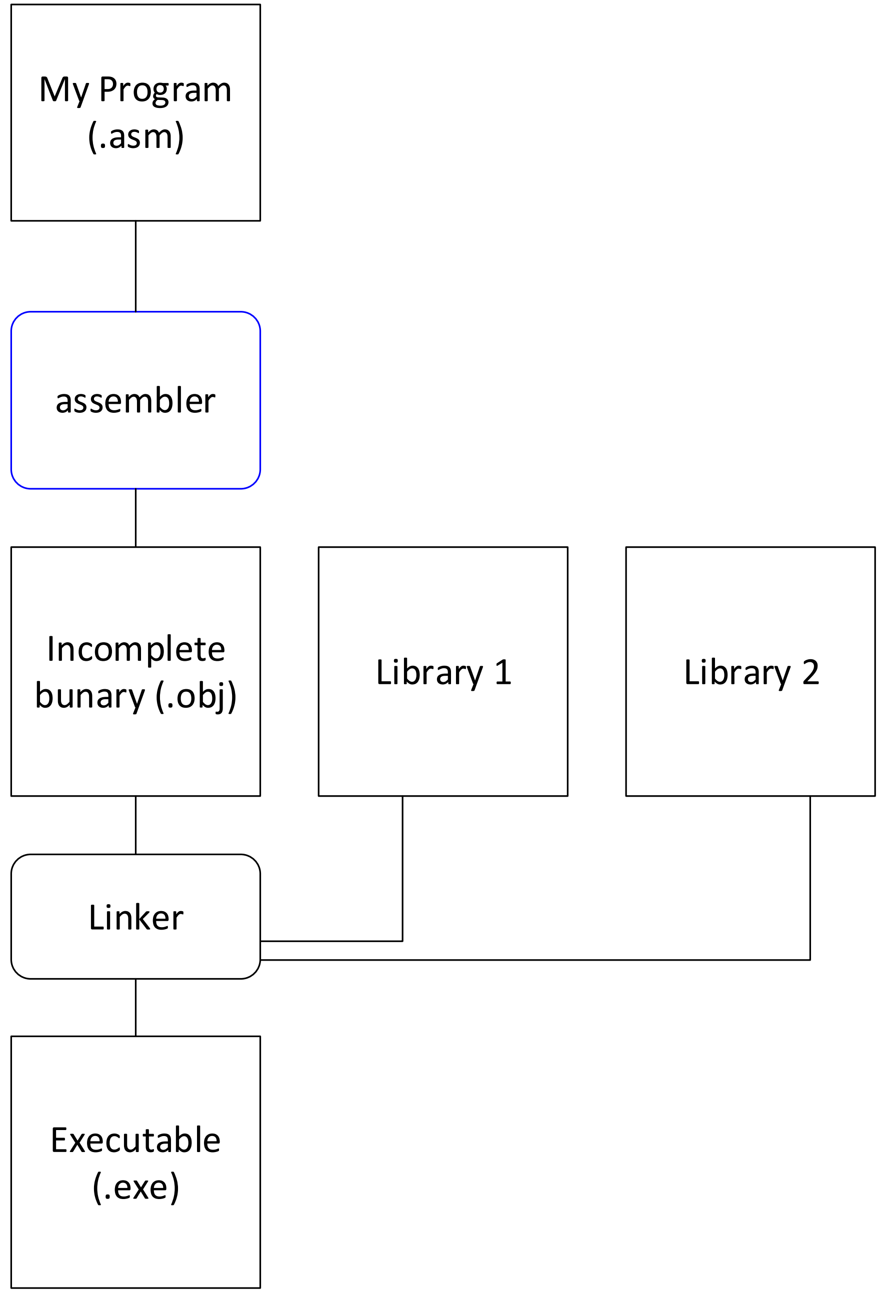
```
; IN
; OUT
;
```

Subroutine abstraction

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

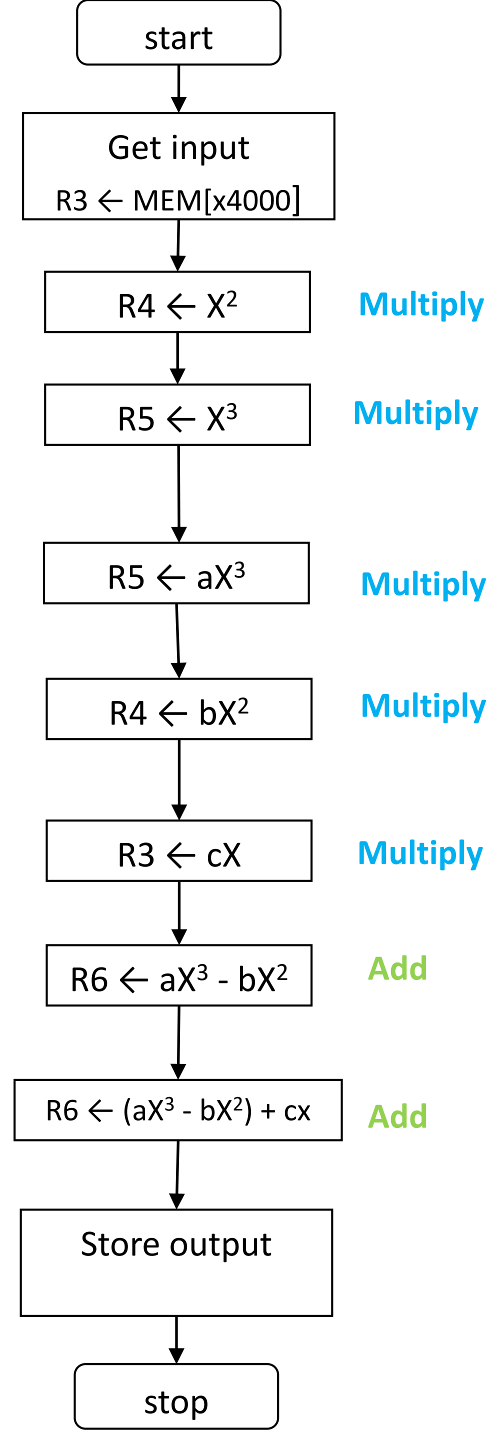
- Does not specify?

Libraries and compilation



Exercise

Compute $y=ax^3-bx^2+cx$ 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