

# Review Problem 1

---

- ❖ Programming languages have many instructions, but they fall under a few basic types. One is arithmetic (+, -, \*, /, etc). What are the others?

Control flow

- Compare >, !=
- function calls
- if-then, goto, Return
- assign
- object/variable declarations + allocation
- loops: for, while
- Libraries: file I/O, graphics.

# Assembly Language

---

Readings: 2.1-2.7, 2.9-2.10, 2.14

Green reference card

## Assembly language

Simple, regular instructions – building blocks of C, Java & other languages

Typically one-to-one mapping to machine language

## Our goal

Understand the basics of assembly language

Help figure out what the processor needs to be able to do

Not our goal to teach complete assembly/machine language programming

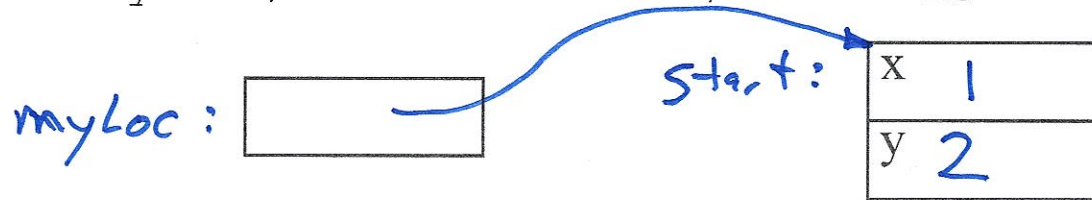
Floating point

Procedure calls

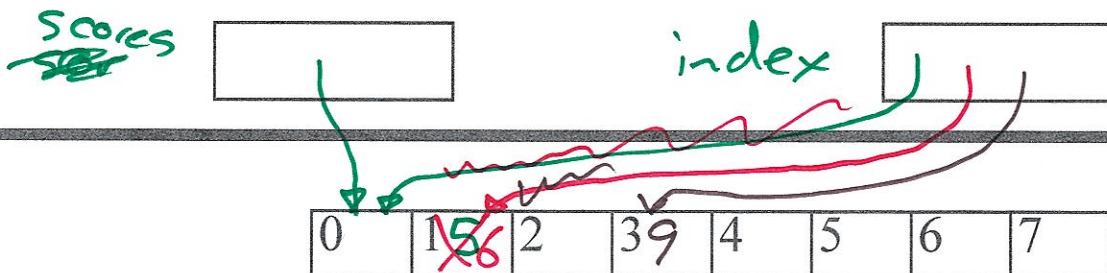
Stacks & local variables

## Aside: C/C++ Primer

→ `struct coord { int x, y; };` /\* Declares a type \*/  
→ `struct coord start;` /\* Object with two slots, x and y \*/  
→ `start.x = 1;` /\* For objects "." accesses a slot \*/  
→ `struct coord *myLoc;` /\* "\*" is a pointer to objects \*/  
→ `myLoc = &start;` /\* "&" returns thing's location \*/  
→ `myLoc->y = 2;` /\* "->" is "\*" plus "." \*/



→ `int scores[8];` /\* 8 ints, from 0..7 \*/  
→ `scores[1]=5;` /\* Access locations in array \*/  
→ `int *index = scores;` /\* Points to scores[0] \*/  
→ `index++;` /\* Next scores location \*/  
→ `(*index)++;` /\* "\*" works in arrays as well \*/  
→ `index = &(scores[3]);` /\* Points to scores[3] \*/  
→ `*index = 9;`



# ARM Assembly Language

---

The basic instructions have four components:

Operator name

Destination

1<sup>st</sup> operand

2<sup>nd</sup> operand

ADD <dst>, <src1>, <src2>      // <dst> = <src1> + <src2>

SUB <dst>, <src1>, <src2>      // <dst> = <src1> - <src2>

Simple format: easy to implement in hardware

More complex:  $A = B + C + D - E$

ADD t1, B, C  
ADD ~~B~~<sup>t1</sup>, t1, D  
SUB A, t2, E

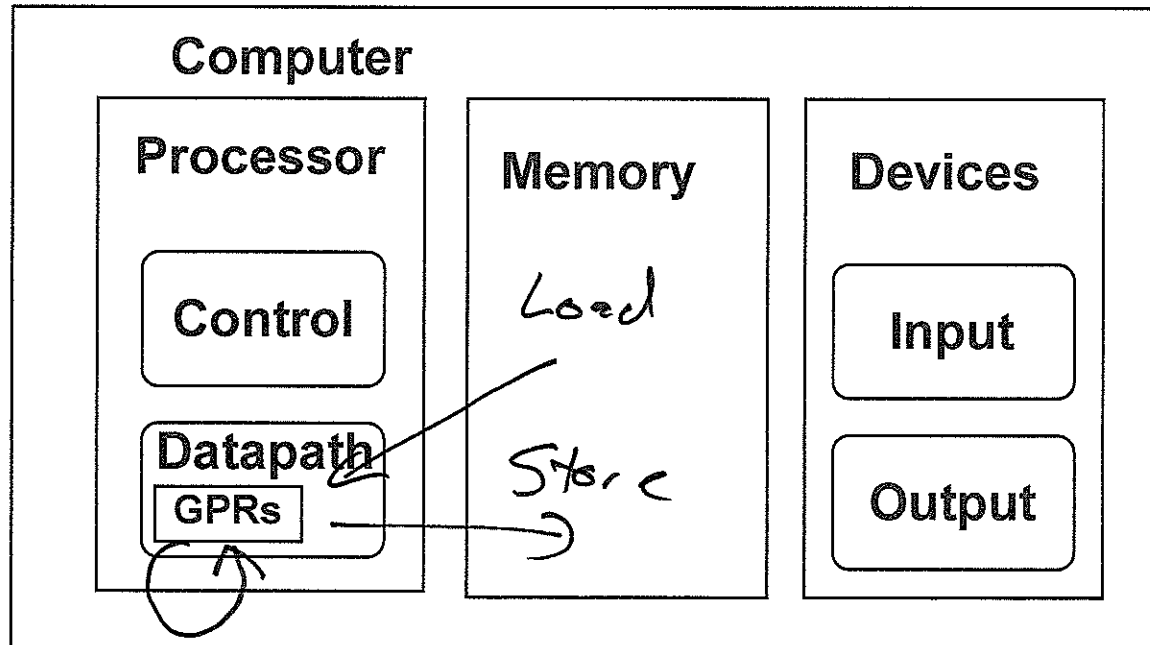
ADD t1, B, C  
SUB t2, D, E  
ADD A, t1, t2



# Operands & Storage

---

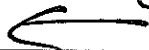
For speed, CPU has 32 general-purpose registers for storing most operands  
For capacity, computer has large memory (multi-GB)



Load/store operation moves information between registers and main memory  
All other operations work on registers

# Registers

32x 64-bit registers for operands



Register	Function	Comment
X0-X7	Function arguments/Results	
X8	Result, if a pointer	
X9-X15	Volatile Temporaries	Not saved on call
X16-X17	Linker scratch registers	Don't use them
X18	Platform register	Don't use this
X19-X27	Temporaries (saved across calls)	Saved on call
X28	Stack Pointer	
X29	Frame Pointer	
X30	Return Address	
X31	Always 0	No-op on write

*temporaries*

*Support for  
subroutines*

469