Chapter 10
The Stack and More...

To Muddle (According to Webster)
Main Entry: 1 muddy
Pronunciation: ’muːd-dli
Function: verb
Inflected Form(s): muddy, muddy
Etymology: probably from obsolete Dutch muddelen, from Middle Dutch, from modde mud; akin to Middle Low German muddle

transitive senses
1 : to make turbid or muddy
2 : to befog or stupefy especially with liquor
3 : to mix confusedly
4 : to make a mess of : BUNGLE

intransitive senses : to think or act in a confused aimless way
- muddler /’muːd-ler, ’muːd-lər/ noun

Next Semester
Yes
• CSE 121/131: Programming Languages and Techniques II
• CSE 260: Math. Foundations of CS
• CSE 371/372: Digital Systems Organization and Design

Maybe
• CSE 112: Networked Life
• CSE 313: Computational Linear Algebra (Math 114)
• CSE 341: Intro. to Compilers and Interpreters (120/121, 260, time change?)
• CSE 377: Virt. World Design (no freshman, strong programming)
• CSE 391: Introduction to Artificial Intelligence (121, 262?)
• CSE 399/001: Computer Vision (anal. geom, lin. alg., Math 114,115, 240, or perm)

Probably not
• CSE 320: Introduction to Algorithms (262 or A- in 260)
• CSE 398: Quantum Computing and Information Science (260,261?,262, Math 240, and permission)
• CSE 455: Internet and Web Systems (120/121, 330, 380)

Review: Using Memory
Memory
• Just a big “array”
• “Indexed” by address
• Accessed with loads and stores

LD/LDR/LDI
• Read a word out of memory
• Use different addressing mode

ST/STR/STI
• Place a word in memory
• Use different addressing mode
Review: Using Memory (cont.)

Problem
- What if the memory you want to access is far away?
- LD/ST won’t work (PC-relative)
- LDR/STR won’t work alone (need to get address in register)

Solution
- Place address of far away value nearby
- Load address, then load/store from that

```
.ORIG x3000
LD R2, SOMELAB
LDR R2, R3, #0
SOMELAB .FILL xFE00

.ORIG x0000
LDI R2, SOMELAB
```

Constant "local variables"

Review: Using Memory (Summary)

Addresses
- Labels allow programmer to refer to addresses
- Memory and registers may contain addresses
- It’s up to programmer to know difference
- It’s up to programmer to use appropriate load/store instructions

Bottom line
- Don’t be a muddler!
- Without mastery, C programming not possible
- Without C programming, CSE 381 hurts!!!
- Working on tutors!!!

Problems?

What’s the problem with... recursion?

```
Main . . .
Next . . .
Foo . . .
ST R7, SaveR7
AND R0, R0, #0
JSR Foo
After . . .
LD R7, SaveR7
RET
Save? .FILL #0
```

- First call to Foo (SaveR7 contains address of Next)
- Second call to Foo (SaveR7 contains address of After)
- First return from Foo (returns to After)
- Second return from Foo (returns to After again!!!)
Recursion

Need
- Per-subroutine-invocation data space (activation record)

Approach
- Allocate new activation record on a stack whenever a subroutine is called
- Subroutine uses its own activation record to hold invocation-specific data (e.g., local variables, saved registers)

Note
- Given that Breakout is recursive, we will need activation records

Big Picture

Each subroutine invocation gets its own activation record . . . but how?

Stacks (Review)

A LIFO (last-in first-out) storage structure
- The first thing you put in is the last thing you take out
- The last thing you put in is the first thing you take out

Two main operations
- PUSH: add an item to the stack
- POP: remove an item from the stack

A Physical Stack

Coin holder

Last quarter in is the first quarter out (LIFO)
A Software Stack

Data items don’t move in memory, just our idea about where TOP of the stack is

---

Basic Push and Pop Code

Push

ADD R6, R6, #1 ; increment stack ptr
STR R0, R6, #0 ; store data (R0)

Pop

LDR R0, R6, #0 ; load data from TOS
ADD R6, R6, #-1 ; decrement stack ptr

Note

- Stacks can grow in either direction (toward higher address or toward lower addresses)