

# Lecture 4

## Assembly language and Procedures

Calling on in transit

*CS 241: Foundations of Sequential Programs*

Fall 2014

Troy Vasiga et al  
University of Waterloo

## Review

- ▶ Arrays
- ▶ Loops

## Loop example

```
; $2 <- 13
lis $2
.word 13

; clear $3
add $3, $0, $0
add $3, $3, $2 ←
; decrement $2
lis $1
.word -1
add $2, $2, $1

; if $2 != 0, loop
bne $2, $0, -5

; return to the OS
jr $31
```

; comment

Document  
or  
die.

## Labels

→ Indicates a particular location / address

start:

end:

clone:

if7gt3:

loop: add \$3, \$3, \$2



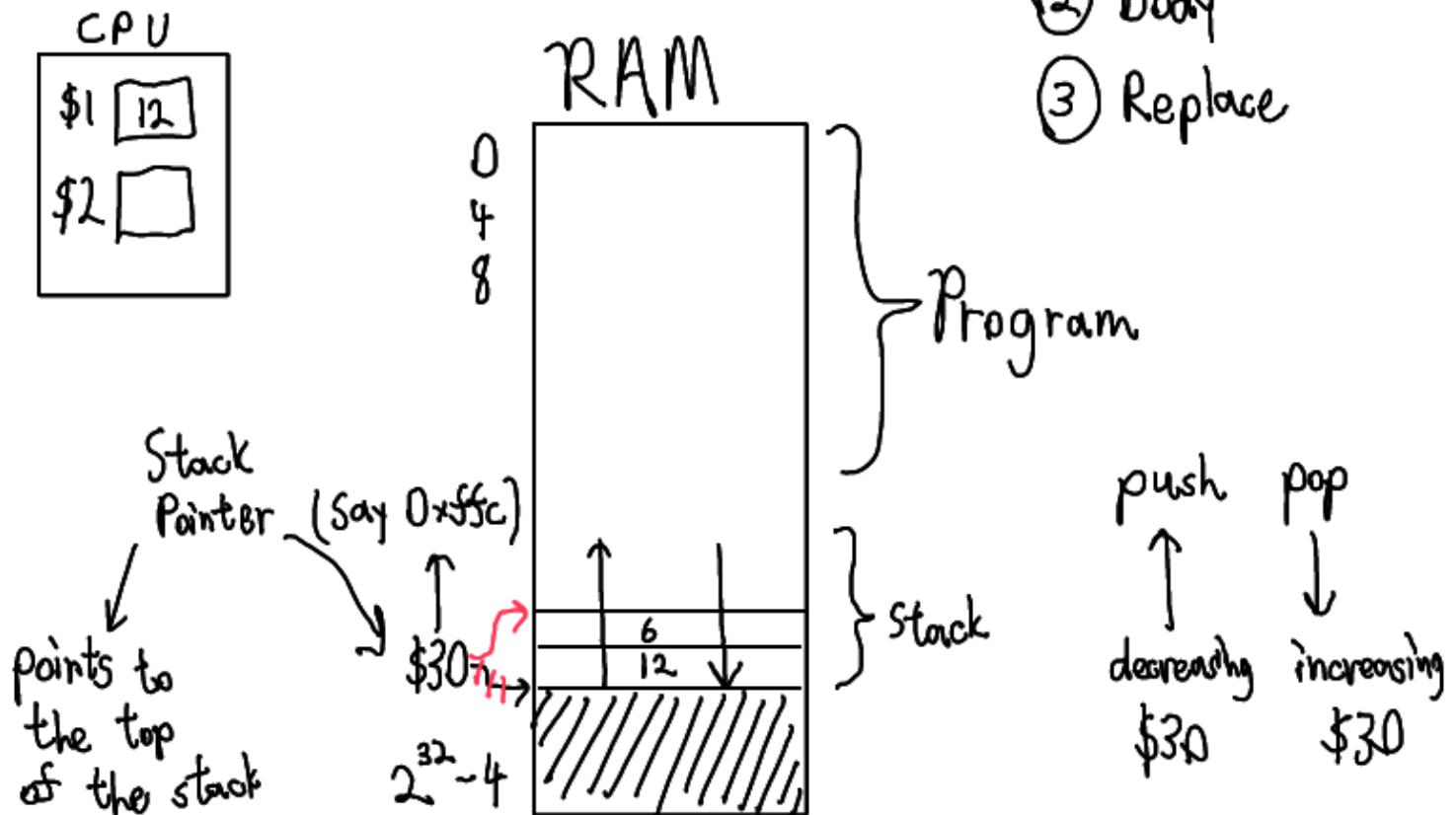
bne \$2, \$0, loop

## Storing and Restoring Registers

- ▶ We wish to keep register values intact

## Procedures (Example 6)

- ① Save
- ② Body
- ③ Replace



## Recursion and general MIPS pattern

↳ calling a subprogram (that is yourself)

Two cases:

① base case(s): ensure they are correct.

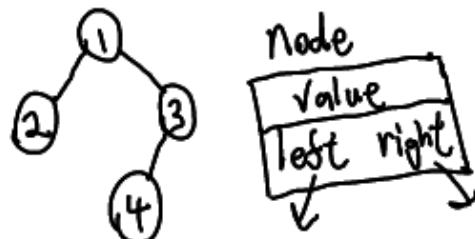
- right time
- right result

② recursive case(s): trust yourself

- don't look ahead
- combine the recursive result with the "current" value.

Linear Recursion  
Binary Recursion

→ recurse left  
save \$37  
recurse right  
combine



index
0
1
2
3
4
5
6
7
8
9
10
11

The table shows indices 0 through 11. Braces on the left group indices into four "node" structures. The first node contains indices 0, 1, and 2. The second node contains indices 3, 4, and 5. The third node contains indices 6, 7, and 8. The fourth node contains indices 9, 10, and 11. The indices 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, and 11 are listed vertically on the right side of the table.

## Input/Output

See Example 5

## Other MIPS notes

- ▶ unsigned operators (e.g., multu, sltu, ...)  
    →  $0..2^{32}-1$
- ▶ local variables in MIPS?  
    → registers: save + restore
- ▶ dynamic memory?  
    →  $\$29 \rightarrow$  heap pointer  
    →  $-2^{31}...2^{31}-1$   
        ↑  
        2's comp:  
        (mult)  
        slt