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
; 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:
ifZgt3:

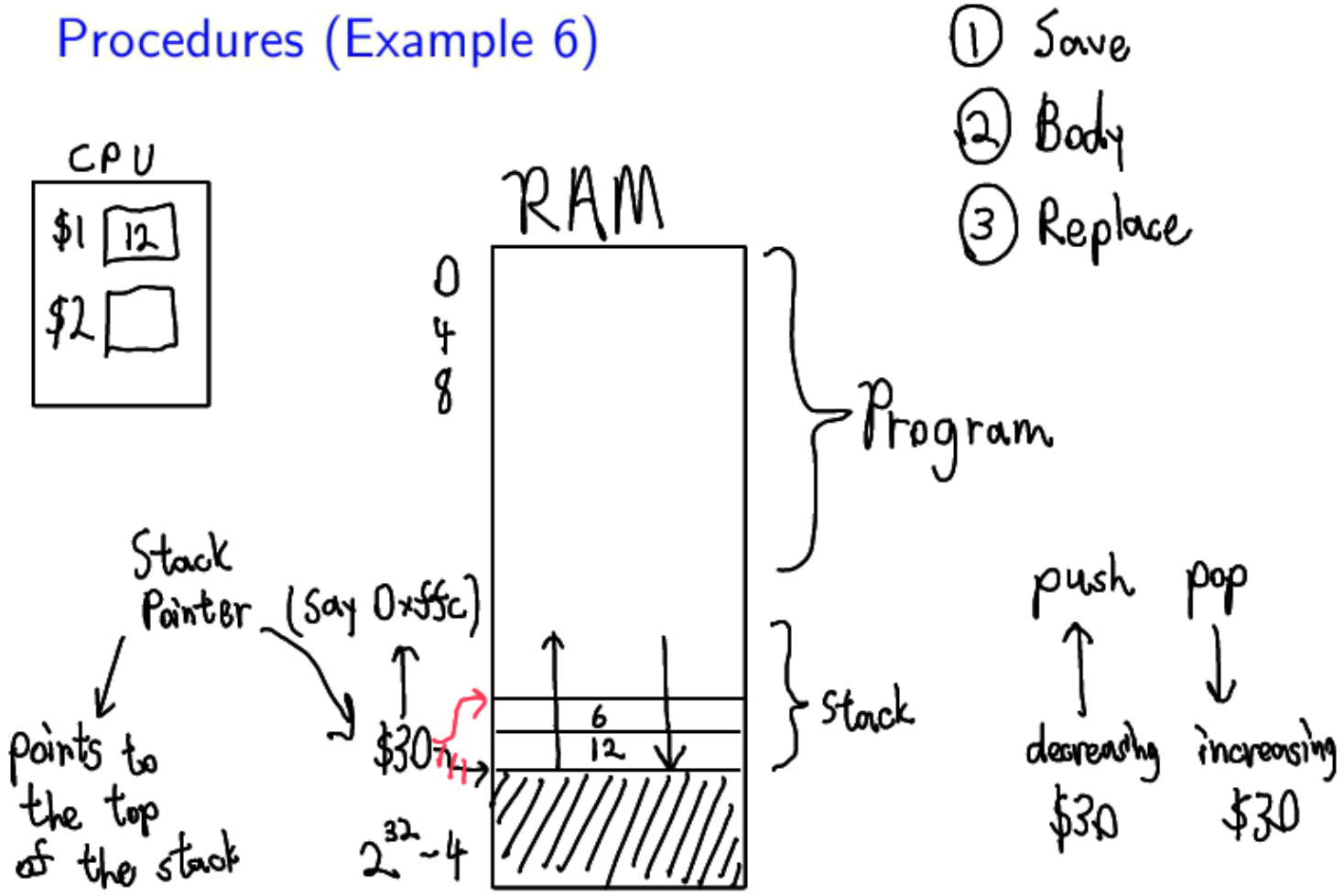
loop: add $3, $3, $2

bne $2, $0, loop

Storing and Restoring Registers

- We wish to keep register values intact

Procedures (Example 6)



Recursion and general MIPS pattern

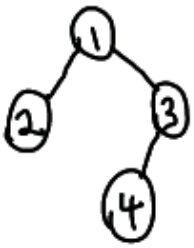
→ calling a subprogram (that is yourself)

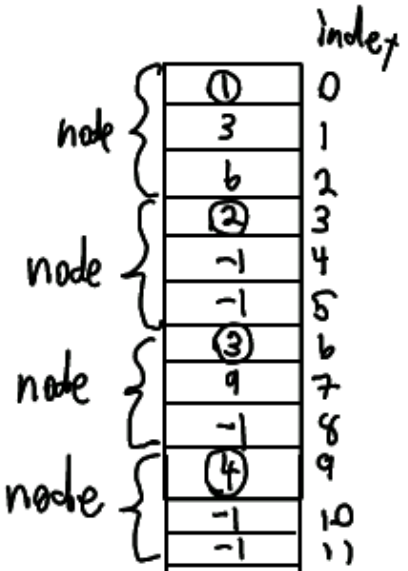
Two cases:

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 3_7
recurse right
combine

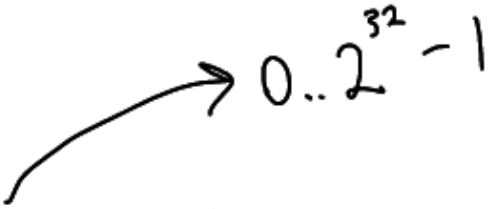




Input/Output

See Example 5

Other MIPS notes



$-2^{31} \ldots 2^{31}-1$

2's compl

(mult slt)

- unsigned operators (e.g., `multu`, `sltu`, ...)
- local variables in MIPS?

  → registers: save + restore
- dynamic memory?

  → $29 → heap pointer