Lecture 4
Assembly language and Procedures

CS 241: Foundations of Sequential Programs
Spring 2017

Troy Vasiga et al. (where Dan Holtby ∈ al)
University of Waterloo
Loop example

;; - Program that computes the sum of 1..13 into $3
;; $1 = 1
;; $2 = loop variable i
;; $3 = sum

;;===setup===
lis $2 ; i <- 13
.word 13
add $3, $0, $0 ; sum <- 0
lis $1 ; $1 <- 1
.word 1

;;===loop body===
add $3, $3, $2 ; sum += i
sub $2, $2, $1 ; --i
bne $2, $0, -3 ; if i is not 0, go back 3 inst

;;===return to OS===
jr $31
Labels

- Can use as a relative address (beq)
- Can use as an absolute address (lis)
Loop example w/labels

;;; - Program that computes the sum of 1..13 into $3
;;; $1 = 1
;;; $2 = loop variable i
;;; $3 = sum

;;;===setup===
lis $2 ; i <- 13
.word 13
add $3, $0, $0 ; sum <- 0
lis $1 ; $1 <- 1
.word 1

;;;===loop body===
loop: add $3, $3, $2 ; sum += i
sub $2, $2, $1 ; --i
bne $2, $0, loop ; if i is not 0, go back to loop

;;;==return to OS===
jr $31
Labels again

Can load the address of a label with lis

```
lis $27
.word myLabel
```

The is the absolute address. No math involved (yet)

Useful for

- Big loops using jr instead of beq (beq can only branch over $2^{15}$ instructions)
- Function calls using jalr
- A global variable using lw / sw for access
Memory

Load from memory with lw

Example

;; - Program that puts the value MEM[$1] into $3

lw $3, 0($1)
jr $31

Things to try on mips.twoints:

▶ $1 = 0 (retrieve the lw instruction)
▶ $1 = 8,12,16,etc. (retrieve the instructions after the program)
▶ $1 = 3 (alignment error)
Arrays

mips.array asks the user for a length \( n \), then \( n \) values
Pointer to array goes into \$1\), length goes into \$2\)

```assembly
;; - snippet that puts element 2 (index=1) into $3
beq $2, $0, 1 ; length 0, skip
lw $3, 4($1)
```

For fun: it should check that length \( \geq 2 \), not just that length \( \neq 0 \)
::< snippet that puts element at index $7 into $3
:: $7 = index i
:: $8 = scratch
:: $4 = 4

lis $4        ; $4 <- 4
.word 4

multu $7, $4
mflo $8       ; $8 <- i * 4
add $8, $8, $1 ; $8 <- arr + i*4
lw $3, 0($8)  ; $3 <- *(8)
Simulator has memory mapped I/O

If you write to the right address, a byte is sent to the screen (stdout)

;; - snippet that prints ! (ascii 33) to the screen

lis $4 ;
.word 0xFFFF000C ; $4 <- special output address
lis $5 ;
.word 33 ; $5 <- ’!’
sw $5, 0($4) ; putchar($5)
If you read from the right address, a byte is read from the user (stdin)

;; - snippet that gets a character from stdin

lis $4 ;
.word 0xFFFF0004 ; $4 <- special input address
lw $3, 0($4) ; $3 <- getchar()
Storing and Restoring Registers

- We wish to keep register values intact
- Can save them to memory
  - Need somewhere not in use
  - $30
Procedures

Didn’t quite finish, so first thing Tuesday!