CS 241, Lecture 4 - Procedures

1 Example

Write an assembly program that takes in a value in reg 1 and stores the sum of the digits to reg 2.

```
1  lis $3
2  .word 10
3  add $4, $1, $0
4  top:
5  div $4, $3
6  mfhi $5
7  mflo $4
8  add $2, $2, $5
9  bne $4, $0, top
10 jr $31
```

2 Procedures

- Store old values in RAM when we call a function.
- We use a stack pointer to keep track of where we track our registers.
- Register 30 (29 in MIPS standard) keeps track of the bottom of your free RAM. Note that this is assuming that our registers grow upward.
- When we store a word with sw, we store at the beginning of the function at $-x * 4(\$30)$, where $x >= 1$.
- After, update register 30/stack pointer by how many bytes you've stored by.
- For example:

```
1 f:
2       sw $1, -4($30)
3       sw $2 -8($30)
4       lis $2
5       .word $8
```

```
 6       sub $30, $30, $2
 7       ;rest of function
 8       lis $2
 9       .word $8
10       add $30, $30, $2
11       lw $2, -8($30)
12       lw $1, -4($30)
13       jr $31
```

- Now how would we call function f?
- We will violate our previous rule and overwrite register $31! First, we store it in RAM, then we use it as a scratch register and overwrite it with lis and immediate value 4.
- Then, we decrement $30 by 4 and then jump with jalr to the register containing f. The decrementing is the same idea as moving the stack pointer, to accomodate for us storing $31 in RAM.
- After the function call, we once again set $31 to 4, increment $30 by 4, then restore $31 with lw.

```
1  main:
2       lis $8
3       .word f
4       sw $31, -4($30) ; push 31 to stack
5       lis 31 ; use 31 since it has been saved
6       .word 4
7       sub $30, $30, $31
8       jalr $8
9       lis $31
10      .word 4
11      add $30, $30, $31
12      lw $31, -4($30)
13      jr $31
```

2.1 Parameters

- Typically, we store parameters in registers (though if you somehow, for some forsaken reason, have more parameters than registers, you CAN push

them all to stack and then pop them from stack).

- Documentation is very important - you must tell users what you are modifying.
- For example, let's sum even numbers from 1 to N:

```
1  ; sumEven1ToN adds all even numbers from 1 to N,
     where N is even
2  ; Registers:
3  ; $1 : Scratch register, will save previous value
4  ; $2 : Input register, will save previous value
5  ; $3 : Output register, will not save previous
     value
6  sumEven1ToN:
7       sw $1, -4($30)
8       sw $2, -8($30)
9       lis $1
10      .word 8
11      sub $30, $30, $1
12
13      ; Actual function starts
14      add $3 $3 $0 ; zero out $3
15      lis $1 ; set $1 to imm. 2
16      .word 2
17      loopStart:
18          add $3, $3, $2
19          sub $2, $2, $1
20          bne $2, $0, loopStart
21      ; Actual function ends
22
23      lis $1
24      .word 8
25      add $30, $30, $1
26      lw $2, -8($30)
27      lw $1, -4($30)
28      jr $31
```

3 Input and Output

- To output, use sw to store words into location 0xffff000c. The LSB will be printed.
- To input, use lw to store words in location 0xffff0004. LSB will be the next character from stdin.
- For example:

```
1 outputFn:
2       lis $1
3       .word 0xffff000c
4       lis $2
5       .word 67 ; the character C
6       sw $2, 0($1)
```