Lecture 8-1

or ""
Announcements

Homework will be assigned at the end of class.
Let’s start with MARS
Some examples (If)

if ((a>b) && (c==d)) e=0; else e=f;
a, b, c, d, e, f in $s0, $s1, $s2, $s3, $s4, $s5

bgt $s0,$s1,next0
# if a>b, go to next0
j nope
# else go to nope
next0:  beq $s2,$s3,yup
# if c==d, go to yup
nope:  move $s4, $s5 # e=f
j out
yup:  xor $s4,$s4,$s4
out:  …
For Loop

for (i=0;i<a;i++) b[i]=i;

a in $s0, i in $s1, and base address of b[] in $s2

addi $s1,$zero, 0    # set i = 0.

loop0:    blt $s1,$s0,loop1    # if i<a, goto loop1

j out    # else jump out of the loop

loop1:    sll $t0,$s1,2    # j= i *4

add $t0, $t0, $s2    # address of b[i]

sw $s1,0($t0)    # b[i] = i

addi $s1,$s1,1    # i = i +1.

j loop0

out: …
Pre-Test While Loop

while (a<b) { a++;

a and b in $s0 and $s1.

loop0: blt $s0, $s1, loop1  # if a<b, go to loop1
    j out  # else jump out the loop

loop1: addi $s0,$s0,1  # a = a+1
    j loop0

out: …
Branches

bne $s0, $s1, Lbl  # go to Lbl if $s0 $s1
beq $s0, $s1, Lbl  # go to Lbl if $s0 = $s1

Format

<table>
<thead>
<tr>
<th>0x05</th>
<th>16</th>
<th>17</th>
<th>16 bit offset</th>
</tr>
</thead>
</table>
Branches

j label  #go to label

| 0x02 | 26-bit address |
## Target Addressing

### Example

Loop location starts at 80000

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Address</th>
<th>pc</th>
<th>op1</th>
<th>op2</th>
<th>op3</th>
<th>op4</th>
<th>op5</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sll $t1, $s3, 2</code></td>
<td>80000</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>9</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td><code>addi $s3, $s3, 1</code></td>
<td>80004</td>
<td>0</td>
<td>9</td>
<td>22</td>
<td>9</td>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td><code>lw $t0, 0($t1)</code></td>
<td>80008</td>
<td>35</td>
<td>9</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><code>bne $t0, $s5, Exit</code></td>
<td>80012</td>
<td>5</td>
<td>8</td>
<td>21</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><code>addi $s3, $s3, 1</code></td>
<td>80016</td>
<td>8</td>
<td>19</td>
<td>19</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><code>j Loop</code></td>
<td>80020</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20000</td>
<td></td>
</tr>
</tbody>
</table>

Exit: ... | 80024 | 0 | 0 | 0 | 0 | 0 | 0 |
## Target Addressing Example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop: sll $t1, $s3, 2</td>
<td>80000</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>9</td>
<td>4</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add $t1, $t1, $s6</td>
<td>80004</td>
<td>0</td>
<td>9</td>
<td>22</td>
<td>9</td>
<td>0</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lw $t0, 0($t1)</td>
<td>80008</td>
<td>35</td>
<td>9</td>
<td>8</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bne $t0, $s5, Exit</td>
<td>80012</td>
<td>5</td>
<td>8</td>
<td>21</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>addi $s3, $s3.1</td>
<td>80016</td>
<td>8</td>
<td>19</td>
<td>19</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>j Loop</td>
<td>80020</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20000</td>
</tr>
<tr>
<td>Exit: ...</td>
<td>80024</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How these work

The jump call takes the address, multiplies it by 4 (20000 words * 4 bytes per word = offset), and goes to that address.

Branch tells it “I want to go forward this many words”, so it goes to the next instruction space, then goes forward X number of words.
What if the branch destination is further away than we can represent in 16 bits?

The assembler helps us!

```
beq $s0, $s1, L1
```

becomes

```
bne $s0, $s1, L2
j L1
```

L2: ...
Functions in HLL (High Level Programming language) provide abstraction

- Mapping actual params to formal params
- Allocation and initialization of temporary local storage
Environment of a Function

- Parameters
- Value of all of the local variables
- Statement being executed
Functions/Procedures

- Before function A calls function B it pushes its environment onto the stack
- Jumps to function B
- When B returns function A restores its environment by popping its environment from the stack
Instructions for Accessing Procedures

MIPS procedure call instruction:

jal ProcedureAddress  #Jump and link

Saves Program Counter (PC) + 4 to $ra to have a link to the next instruction after the procedure returns

Return with

jr $ra  #return
6 steps in Execution

Main routine (caller) places params in a place where the procedure (callee) can access them

$\text{a0 - a3}$

Caller transfers control to the callee

Callee acquires the storage resources needed

Callee performs the desired task
Callee places the result value in a place where the caller can access it

   $v0-$v1: 2 value registers for result values

Callee returns control to the caller

   $ra: one return address register to return to the point of origin
Spilling registers

What if the callee needs more registers than allocated? callee uses a stack (last in-first out)

One of the general registers, $sp, is used to address the stack which grows from high address to low address

Add to stack? $sp = $sp - 4 then put data on

remove from stack? take data out then $sp = $sp + 4
C code

```c
int leaf_example (int g, h, i, j)
{
    int f;
    f = (g + h) - (i + j);
    return f;
}
```

Arguments g, ..., j in $a0, ..., $a3
f in $s0
g+h in $t0, i+j in $t1, result in $v0
Example

leaf_example: addi $sp,$sp,-12 # push $s0,$t0,$t1 on stack
    sw $s0, 0($sp)
    sw $t0, 4($sp)
    sw $t1, 8($sp)
    add $t0, $a0, $a1 # procedure
    add $t1, $a2, $a3
    sub $s0, $t0, $t1
    add $v0, $s0, $zero # result
    lw $s0, 0($sp) # restore $s0
    lw $t0, 4($sp)
    lw $t1, 8($sp)
    addi $sp, $sp, 12
    jr $ra # return
Call a procedure

leaf_example(1,2,3,4)
li $a0, 1
li $a1, 2
li $a2, 3
li $a3, 4
jal leaf_example
move $t0, $v0
Homework

Write a program using MARS that takes 2 integers and adds them together.
Write another program that takes 2 integers and subtracts them, and takes another int \( k \) and shifts the previous result and shifts it left by \( k \).