Recursion:

PICK UP HANDOUT

Definition:

Recursion: If you still don't get it, see: "Recursion".

Today's lecture

- **Control flow review & If/Else**
- **Recursion**
  - My turn (Example)
  - Your turn
- **Callee-saved Registers**
  - My turn again

MIPS control instructions

- **Earlier, we saw some of MIPS's control-flow instructions**

  j // for unconditional jumps

  bne and beq // for conditional branches

  slt and slti // set if less than (w/ and w/o an immediate)

- **And how to implement loops**

- **And branch pseudo instructions:**

```
blt  $t0, $t1, L1   // Branch if $t0 < $t1
ble  $t0, $t1, L2   // Branch if $t0 <= $t1
bgt  $t0, $t1, L3   // Branch if $t0 > $t1
bge  $t0, $t1, L4   // Branch if $t0 >= $t1
```

Translating an if-then statement

- We can use branch instructions to translate if-then statements into MIPS assembly code.

```
v0 = A[0];
if (v0 < 0)
    v0 = -v0;
v1 = v0 + v0;
```

```
        lw   $v0, 0($a0)
        bge  $v0, 0, skip
        sub  $v0, $zero, $v0
skip:   add  $v1, $v0, $v0
```

- Sometimes it's easier to *invert* the original condition.
  - In this case, we changed "continue if v0 < 0" to "skip if v0 >= 0".
  - This saves a few instructions in the resulting assembly code.

$0 or $zero

Translating an if-then-else statements

- If there is an else clause, it is the target of the conditional branch
  - And the then clause needs a jump over the else clause

```
// increase the magnitude of v0 by one
if (v0 < 0)
    v0 --;

else
    v0 ++;
v1 = v0;
```

```
    bge  $v0, $0, E
    sub  $v0, $v0, 1
    j    L
E:  add  $v0, $v0, 1
L:  move $v1, $v0
```

- Dealing with else-if code is similar, but the target of the first branch will be another if statement.

Translating an if-then-else statements

- If there is an else clause, it is the target of the conditional branch
  - And the then clause needs a jump over the else clause

```
// increase the magnitude of v0 by one
if (v0 < 0)
    v0 --;

else
    v0 ++;
v1 = v0;
```

```
    bge  $v0, $0, E
    sub  $v0, $v0, 1
    j    L

E:  add  $v0, $v0, 1
L:  move $v1, $v0
```

- Dealing with else-if code is similar, but the target of the first branch will be another if statement.

Recursion in MIPS

- **Last time we talked about function calls…**
  - Recursion is just a special case of function calls
- **Two parts:**
  - Base case: no recursion, often doesn't call any functions
  - Recursive body: calls itself

Recursion in MIPS (single function call)

Suggestions for implementing recursive function calls in MIPS

1. Handle the base case first
   - Before you allocate a stack frame if possible
2. Allocate stack frame
3. Save return address
4. Recursive Body:
   a) Save any registers needed after the call
   b) Compute arguments
   c) Call function
   d) Restore any registers needed after the call
   e) Consume return value (if any)
5. Deallocate stack frame and return.

Recursion in MIPS (multiple calls – caller save)

Suggestions for implementing recursive function calls in MIPS

1. Handle the base case first
   - Before you allocate a stack frame if possible
2. Allocate stack frame
3. Save return address
4. For each function call: *(suggestion: use **$s** registers if >1 call)*
   a) Save any registers needed after the call
   b) Compute arguments
   c) Call function
   d) Restore any registers needed after the call
   e) Consume return value (if any)
5. Deallocate stack frame and return.

Recursion in MIPS (multiple calls – callee save)

Suggestions for implementing recursive function calls in MIPS

1. Handle the base case first
   - Before you allocate a stack frame if possible
2. Allocate stack frame
3. Save return address
4. Save enough **$s** registers to hold your local variables
5. Copy your local variables to **$s** registers
6. For each function call:
   a) ~~Save any registers needed after the call~~
   b) Compute arguments
   c) Call function
   d) ~~Restore any registers needed after the call~~
   e) Consume return value (if any)
7. Restore **$s** registers
8. Deallocate stack frame and return.