Lecture 9-10 Review

Calling Conventions Review

- Calling conventions define the protocol for calling a function with arguments and getting the return values.
  - An agreement between caller and callee.
  - Some functions are both (for example recursion).
- They define things like:
  - How to pass parameters and get return values.
  - Who manages the stack and when?
  - Which registers must be preserved.

Passing Arguments

- `x,y,z = some_function(a,b,c)`
- Option 1:
  - Caller pushes arguments to stack in order A,B,C.
  - Callee pops arguments (in order C, B, A).
  - Use this unless we say otherwise.
- Option 2:
  - Caller puts arguments in registers `$a0,$a1`,…
- Other options e.g., :
  - Caller pushes arguments to stack in order A,B,C
  - Callee loads argument but does not pop.
  - Caller will clear arguments form stack.

Returning values

- `x,y,z = some_function(a,b,c)`
- Option 1:
  - Callee pushes return value(s) onto the stack in order z,y,x.
  - Caller pops return values (will get order x,y,z)
  - Use this unless we tell you otherwise.
- Option 2:
  - Callee puts the arguments in registers `$v0,$v1`
- Other options e.g.,:
  - Caller preallocates space on stack for return value.
  - Callee stores return value in the prepared space.

Preserving Registers

- `$ra` must be preserved if calling other functions from inside a function.
- Option 1:
  - Push before calling functions (before arguments)
  - Restore after popping return value)
- Option 2:
  - Push when entering a function
  - Pop just before pushing the return value.
- You can use any option as long as it's correct.

Preserving Registers

- Registers $t0 -$t9 are caller-saved
  - If the caller needs their values, save them.
  - Push before calling a function (before arguments)
  - Pop when function returns (after popping ret. val).
- Registers $s0-$s7 are callee-saved
  - If a callee uses them, save them.
  - Push at the beginning, after popping arguments.
  - Pop at the end, before pushing return value.
- You need to do this to maintain correctness.

- For the test & final exam, you'll have a list of available assembly language commands:

Reference Information

ALU arithmetic input table:

Select		Input	Operation	
S_1	S_0	Y	C_{in}=0	C_{in}=1
0	0	All 0s	G=A	G=A+1
0	1	B	G=A+B	G=A+B+1
1	0	B	G=A-B-1	G=A-B
1	1	All 1s	G=A-1	G=A

Register table:

Register values : Processor role

- Register 0 ($zero): value 0.
- Register 1 ($at): reserved for the assembler.
- Registers 2-3 ($v0, $v1): return values
- Registers 4-7 ($a0-$a3): function arguments
- Registers 8-15, 24-25 ($t0-$t9): temporaries
- Registers 16-23 ($s0-$s7): saved temporaries
- Registers 28-31 ($gp, $sp, $fp, $ra)

Instruction table:

Instruction	Op/Func	Syntax
add	100000	$d, $s, $t
addu	100001	$d, $s, $t
addi	001000	$t, $s, i
addiu	001001	$t, $s, i
div	011010	$s, $t
divu	011011	$s, $t
mult	011000	$s, $t
multu	011001	$s, $t
sub	100010	$d, $s, $t
subu	100011	$d, $s, $t
and	100100	$d, $s, $t
andi	001100	$t, $s, i
nor	100111	$d, $s, $t
or	100101	$d, $s, $t
ori	001101	$t, $s, i
xor	100110	$d, $s, $t
xori	001110	$t, $s, i
sll	000000	$d, $t, a
sllv	000100	$d, $t, $s
sra	000011	$d, $t, a
srav	000111	$d, $t, $s
srl	000010	$d, $t, a
srlv	000110	$d, $t, $s
beq	000100	$s, $t, label
bgtz	000111	$s, label
blez	000110	$s, label
bne	000101	$s, $t, label
j	000010	label
jal	000011	label
jalr	001001	$s
jr	001000	$s
lb	100000	$t, i ($s)
lbu	100100	$t, i ($s)
lh	100001	$t, i ($s)
lhu	100101	$t, i ($s)
lw	100011	$t, i ($s)
sb	101000	$t, i ($s)
sh	101001	$t, i ($s)
sw	101011	$t, i ($s)
trap	011010	i
mflo	010010	$d

Student Number: _____ 16 (continued)

Question 1

- Write a sign function
  - Use the regular (simple) calling convention (callee will pop arguments)

```
def sign(i):
  if(i > 0):
    result = 1
  else if(i < 0):
    result = -1
  else:
    result = 0
  return result
```

Question 1

```
def sign(i):
  if(i > 0):
    result = 1
  else if(i < 0):
    result = -1
  else:
    result = 0
  return result
```

Instruction	Op/Func	Syntax
add	100000	$d, $s, $t
addu	100001	$d, $s, $t
addi	001000	$t, $s, i
addiu	001001	$t, $s, i
div	011010	$s, $t
divu	011011	$s, $t
mult	011000	$s, $t
multu	011001	$s, $t
sub	100010	$d, $s, $t
subu	100011	$d, $s, $t
and	100100	$d, $s, $t
andi	001100	$t, $s, i
nor	100111	$d, $s, $t
or	100101	$d, $s, $t
ori	001101	$t, $s, i
xor	100110	$d, $s, $t
xori	001110	$t, $s, i
sll	000000	$d, $t, a
sllv	000100	$d, $t, $s
sra	000011	$d, $t, a
srav	000111	$d, $t, $s
srl	000010	$d, $t, a
srlv	000110	$d, $t, $s
beq	000100	$s, $t, label
bgtz	000111	$s, label
blez	000110	$s, label
bne	000101	$s, $t, label
j	000010	label
jal	000011	label
jalr	001001	$s
jr	001000	$s
lb	100000	$t, i($s)
lbu	100100	$t, i($s)
lh	100001	$t, i($s)
lhu	100101	$t, i($s)
lw	100011	$t, i($s)
sb	101000	$t, i($s)
sh	101001	$t, i($s)
sw	101011	$t, i($s)
trap	011010	i
mflo	010010	$d

Question 2

- Write a function n_sol(a,b,c) that returns the number of solutions for the equation $ax^2+bx+c = 0$ (Use the sign function from Q1)

```
def n_sol(a,b,c):
      delta = b*b - 4*a*c
      s = sign(delta)
      if s < 0:
            return 0
      else s == 0:
            return 1
      else:
            return 2
```

Question 2

- Write a function n_sol(a,b,c) that returns the solutions for the equation $ax^2+bx+c = 0$
  - TRICK! since sign(delta) is -1,0,1, we can do:

```
def n_sol(a,b,c):
      delta = b*b - 4*a*c
      s = sign(delta)
      return s + 1
```

Question 2

```
def n_sol(a,b,c):
      delta = b*b - 4*a*c
      s = sign(delta)
      return s + 1
```

Instruction	Op/Func	Syntax
add	100000	$d, $s, $t
addu	100001	$d, $s, $t
addi	001000	$t, $s, i
addiu	001001	$t, $s, i
div	011010	$s, $t
divu	011011	$s, $t
mult	011000	$s, $t
multu	011001	$s, $t
sub	100010	$d, $s, $t
subu	100011	$d, $s, $t
and	100100	$d, $s, $t
andi	001100	$t, $s, i
nor	100111	$d, $s, $t
or	100101	$d, $s, $t
ori	001101	$t, $s, i
xor	100110	$d, $s, $t
xori	001110	$t, $s, i
sll	000000	$d, $t, a
sllv	000100	$d, $t, $s
sra	000011	$d, $t, a
srav	000111	$d, $t, $s
srl	000010	$d, $t, a
srlv	000110	$d, $t, $s
beq	000100	$s, $t, label
bgtz	000111	$s, label
blez	000110	$s, label
bne	000101	$s, $t, label
j	000010	label
jal	000011	label
jalr	001001	$s
jr	001000	$s
lb	100000	$t, i($s)
lbu	100100	$t, i($s)
lh	100001	$t, i($s)
lhu	100101	$t, i($s)
lw	100011	$t, i($s)
sb	101000	$t, i($s)
sh	101001	$t, i($s)
sw	101011	$t, i($s)
trap	011010	i
mflo	010010	$d

Question 3

- What does the following function do?:

```
myfunc:     lw $t0, 0($sp)
            addi $sp, $sp, 4
            addi $t1, $zero, 2
            div $t0, $t1
            mfhi $t0
            beq $t0, $zero, LABEL1
            add $t2, $zero, $zero
            j LABEL2

LABEL1:     addi $t2, $zero, 1
LABEL2:     addi $sp, $sp, -4
            sw $t2, 0($sp)
            jr $ra
```

Question 3b

Now write some code to do the following:

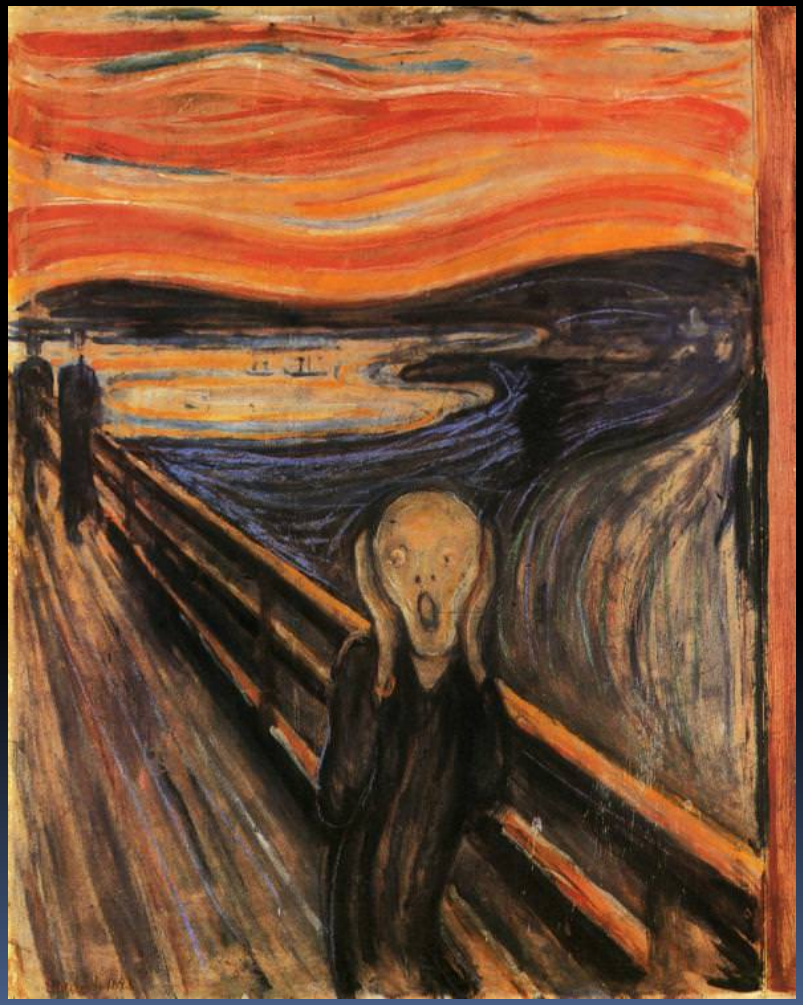
- Create an array of integers
  - The last value in the array is zero (to stop the loop)
- Use the function we just saw to count the number of even values in the array.

Question 4

- Use recursion to compute integer square root of n:

```
def isqrt(n):
    if n < 0:
        return -1 # error!
    if n < 2:
        return n
    low = isqrt(n >> 2) << 1
    high = low + 1
    if high*high > n:
        return low
    else:
        return high
```

Question 4



Question 4

- Don't panic.
- Remember your training!
  - A recursive function is just another function.
  - Save `$ra` and any other regsiters.
- Use "assembly pseudocode" then convert to assembly

Question 4 Pseudocode

- Pop argument into `$t1`
- If `$t1` >= 0 (bgez) skip to next step
  - Otherwise: push -1 then return to caller
- If `$t2` >= 2: skip to next step
  - Otherwise: push `$t1` and return to caller
- Push `$ra`
- Push `$t2` to save n
- Compute n >> 2 and push it
  - Logical shift? (doesn't matter)
- Call isqrt (the recusive call)
- Pop result into `$t1`
- Pop `$t2` to restore n
- …compute…
- Pop `$ra`
- Push return value
- Return to caller

```
def isqrt(n):
    if n < 0:
        return -1 # error!
    if n < 2:
        return n
    low = isqrt(n >> 2) << 1
    high = low + 1
    if high*high > n:
        return low
    else:
        return high
```