Week 10 Review

Question 1

- What are the offsets of the fields?

```
typedef struct {
char a[5]; // this is an array of 5 chars
short b;
int c;
} Foo;
```

Question 1

- What are the offsets of the fields?

```
typedef struct {
char a[5]; // this is an array of 5 chars
short b;
int c;
} Foo;
```

Offset	What?
0	
1	
2	
…	

Question 1

- What are the offsets of the fields?

```
typedef struct {
char a[5]; // this is an array of 5 chars
short b;
int c;
} Foo;
```

- a → 0
  - First field is always at offset 0…
- b → 6
  - Previous field ends at 5, so pad to half-word alignment
- c → 8
  - Previous field ends at 8, which is word-aligned.

Offset	What?
0	a[0]
1	a[1]
2	a[2]
3	a[3]
4	a[4]
5	padding
6	b
7	b
8	c
9	c
10	c
11	c

Question 2

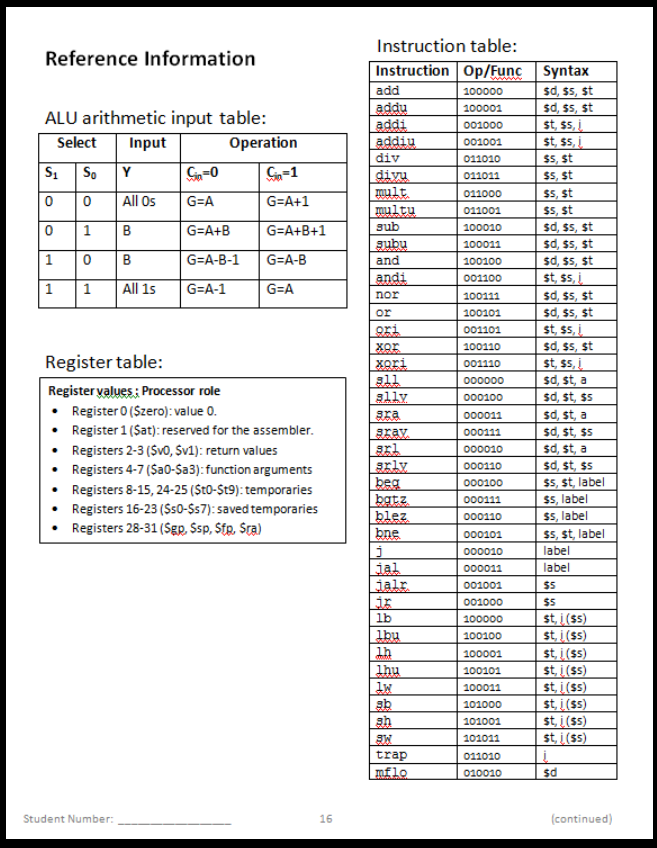
- Final Exam, Winter 2012:

3. In the space below, write a short assembly language program that is a translation of the program on the right. You can assume that `i` has been placed on the top of the stack, and that the return value should be placed on the stack as well before returning to the calling program. Make sure that you comment your code so that we understand what you're doing. **(10 marks)**

```
int sign (int i) {
   if (i > 0)
      return 1;
   else if (i < 0)
      return -1;
   else
      return 0;
```

- How would you convert this to assembly language?

- For the 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)

Code for sign(i)

```
# registers: $t0 = i, $t1 = return value
sign:   lw $t0, 0($sp)          # pop i off the stack
        addi $sp, $sp, 4

        # handle case of i > 0
if_gt:  blez $t0, if_lt
        addi $t1, $zero, 1
        j end

        # handle case of i < 0
if_lt:  beq $t0, $zero, eq
        addi $t1, $zero, -1
        j end

        # handle case of i == 0
eq:     add $t1, $zero, $zero

end:    addi $sp, $sp, -4       # push return value
        sw $t1, 0($sp)
        jr $ra                  # return to caller
```

Question 3: Remember this?

- 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 3

- We divided `$t0` by 2
  - `div` puts remainder in HI
- If remainder is 0 → return 1
- if remainder is not 0 → return 0
- This is a function that returns 1 if a number is even or 0 if it is odd

Question 4

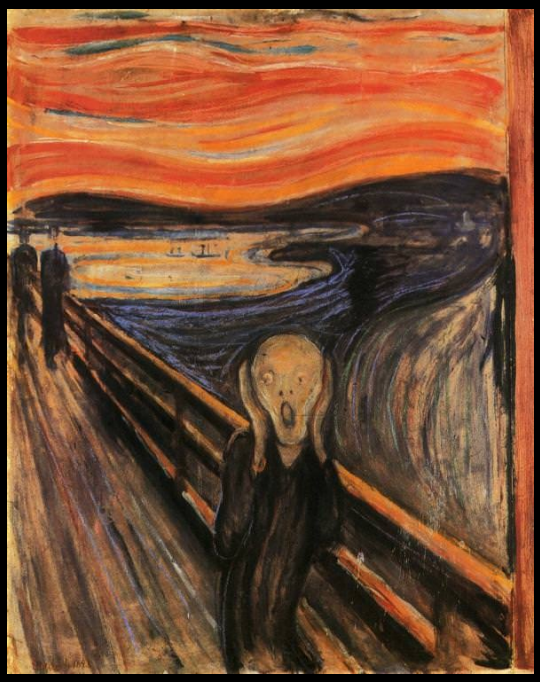
- Write a function `numplus(arr, n)` that returns the number of elements larger than zero in `arr` minus the number of negative elements.
- Weird restrictions:
  - Not allowed to use loops.
  - Only allowed one branch, it must be bne or beq.
  - Not allowed any other branches…
  - …but can use `sign` function.
- How would you do it in C?

Question 4

- Since sign(i) is -1,0,1, we can compute the sum of `sign(arr[i])` for all i.
- Use recursion instead of loops.

```
int numplus(int arr[], int n)
{
      if (n == 0) // here we use the branch
            return 0;
      return sign(arr[0]) + numplus(arr+1, n-1);
}
```

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

- Strategy:
  - Pop arguments into $t0, $t1
  - Deal with base case
  - Save $ra
  - $t2 = sign(arr[0])
  - $t3 = numplus(…)
  - Compute $t2+$t3
  - Restore $ra
  - Return result

Which registers must be saved in each case?

Question 4

- Pop arr, n
- Check for base case n == 0:
  - Return 0 if n is zero
- Save $ra
- Push arr, n to save them
- Get arr[0]
- Call sign(arr[0])
- Pop result into $t2
- Pop arr, n to restore them
- Push $t2 to save it (it contains the result of sign(…))
- Call numplus(arr+4, n-1) ← note arr+4 (in C it would be +1)
- Pop return value into $t3
- Pop $t2 to restore it
- Restore $ra
- Return $t2 + $t3