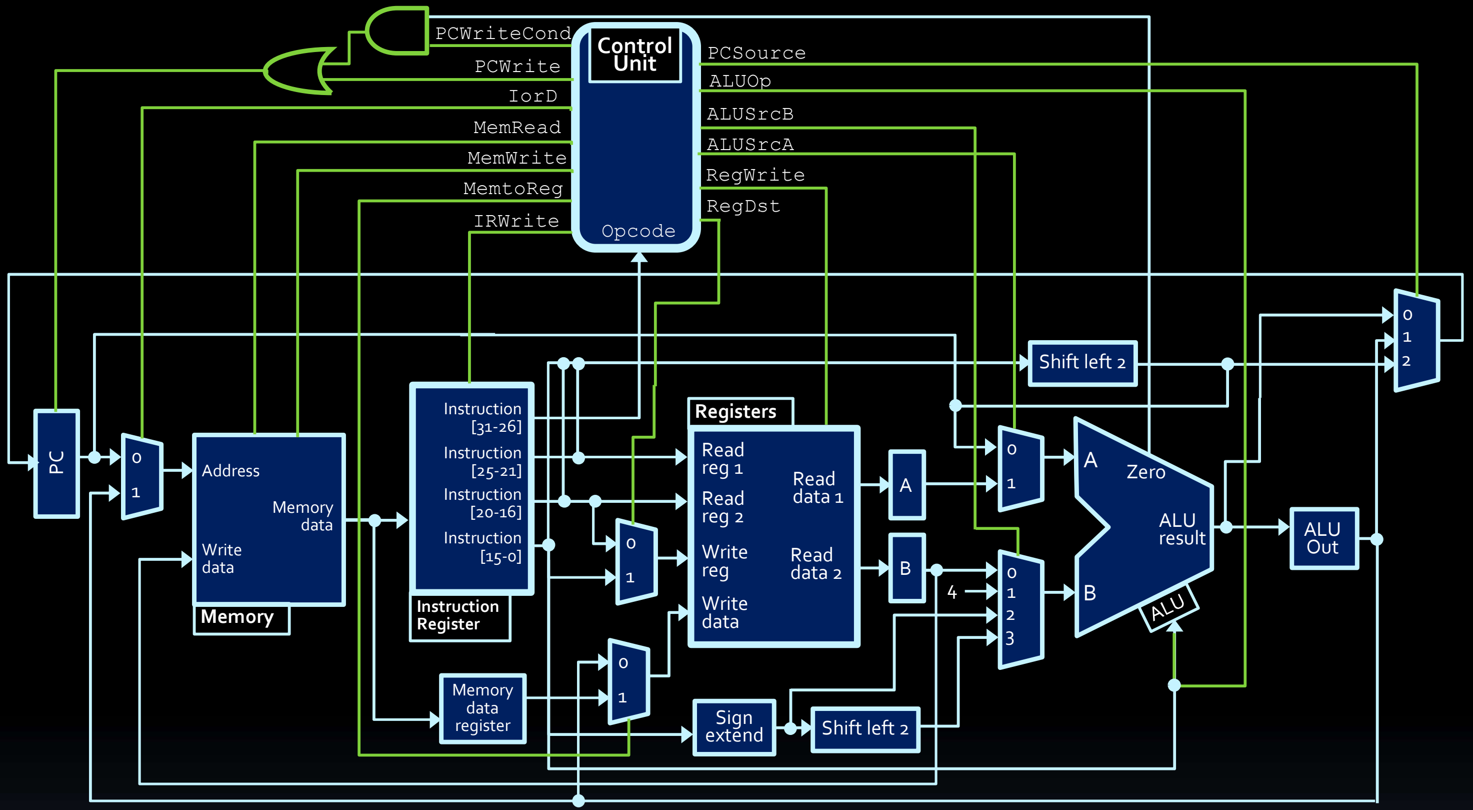
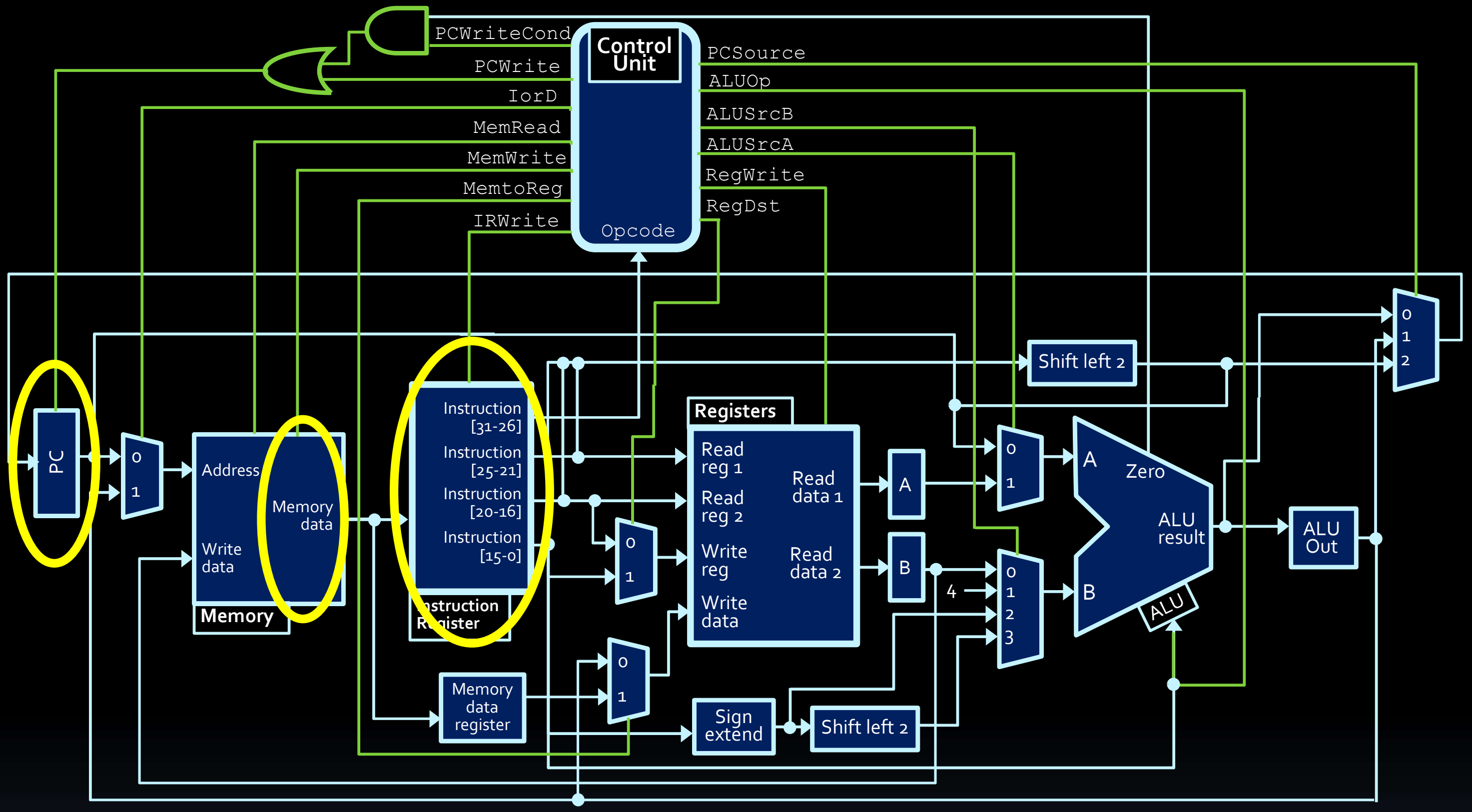
Week 11 Review

Question #1



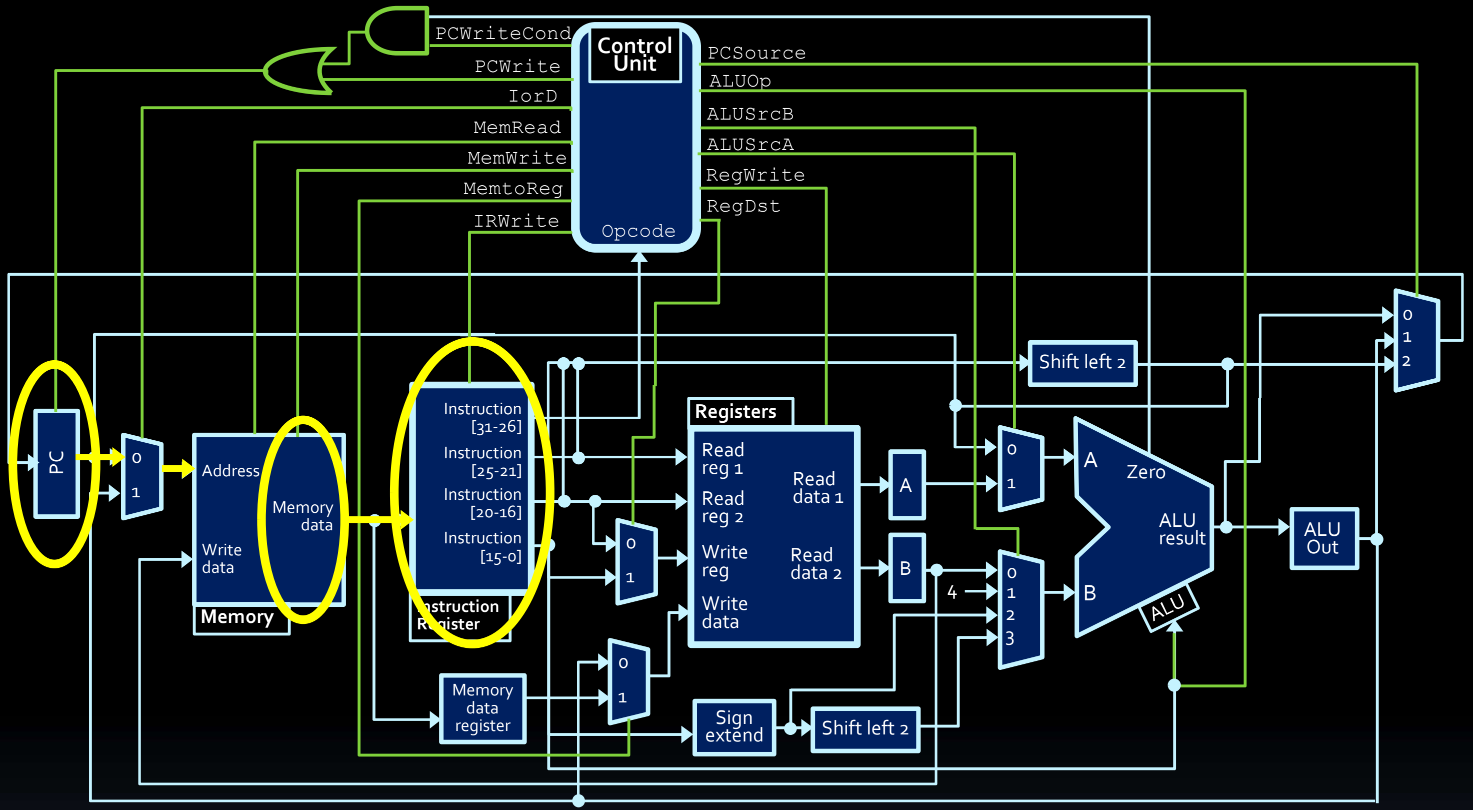
- Given the datapath above, what signals would the control unit turn on and off in order to load a new instruction from memory?

Question #1



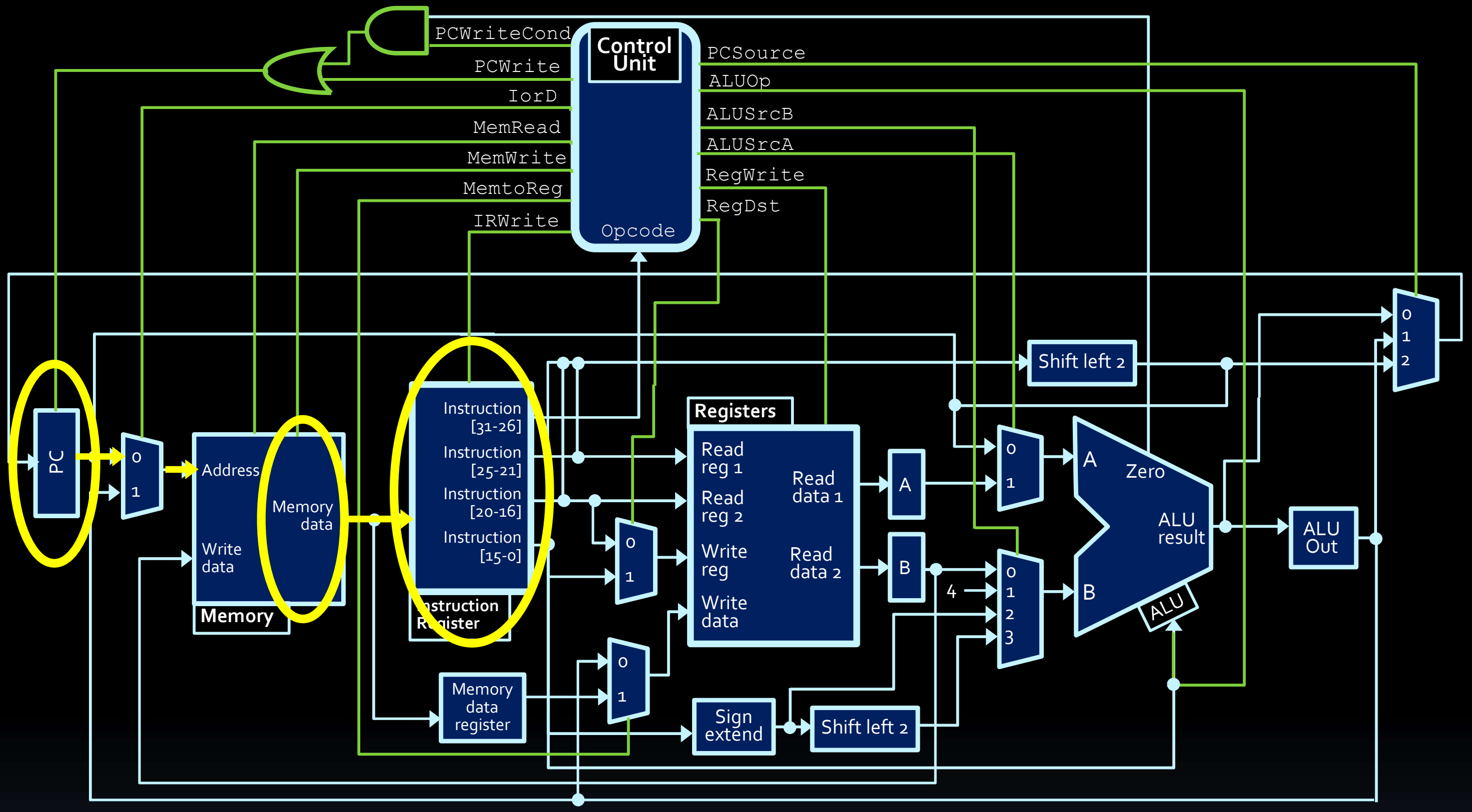
- <u>Step #1</u>: Data source and destination
  - Data starts in memory, ends in instruction register
  - Address of instruction is coming in from PC

Question #1



- <u>Step #2</u>: Data path
  - Data doesn't actually go very far on this diagram.
  - (it actually travels very far, relatively speaking)

Question #1



- <u>Step #3</u>: Signals
  - `MemRead`, `IRWrite` high. All other write signals low.
  - `IorD=0`, and a LOT of don't care values.

Question #1 (final answer)

- PCWrite = 0
- PCWriteCond = 0
- IorD = 0
- MemRead = 1
- MemWrite = 0
- MemToReg = X
- IRWrite = 1
- PCSource = X
- ALUOp = XXX
- ALUSrcA = X
- ALUSrcB = XX
- RegWrite = 0
- RegDst = X

Question #2

- What are the following assembly language instructions doing?

```
beq $t2, $zero, top
```

Jump to the line with label "top" if register `$t2` is equal to `0` (`$zero`)

```
jalr $t0
```

Store the current PC location into $ra (register $31) and jump to the location stored in register `$t0`

```
sw $t1, 16($t1)
```

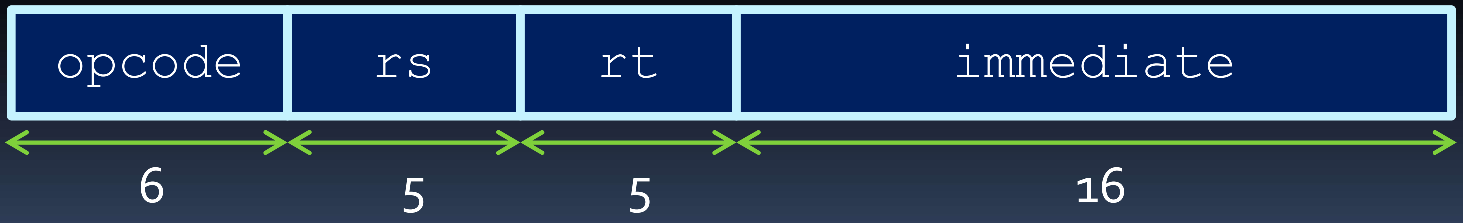
Store the value in register `$t1` 16 bytes ahead of the location in memory specified by `$t1`

Question #3

- How do you translate the following assembly language instruction into machine code?

```
xori $t7, $t0, -1
```

I-type instruction!



As a reminder…

- MIPS register values:
  - Register 0 ($zero): value 0 -- always.
  - Register 1 ($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): memory and function support
  - Registers 27-28: reserved for OS kernel

Question #3

```
xori $t7, $t0, -1
```

- Step #1: The opcode
  - I-type instructions start with the opcode value:

```
001110 sssss ttttt iiiiiiiiiiiiiiii
```

- Step #2: The register values
  - Register `$t0` translates to register `8`, and register `$t7` translates to register `15`
  - 16-bit immediate value is `-1`.

```
001110 01000 01111 1111111111111111
```

Question #4

- How do you write an assembly language program that can swap the values in `$t0` and `$t1`, using `$t2` as a temp value?

```
add $t2, $zero, $t0
add $t0, $zero, $t1
add $t1, $zero, $t2
```

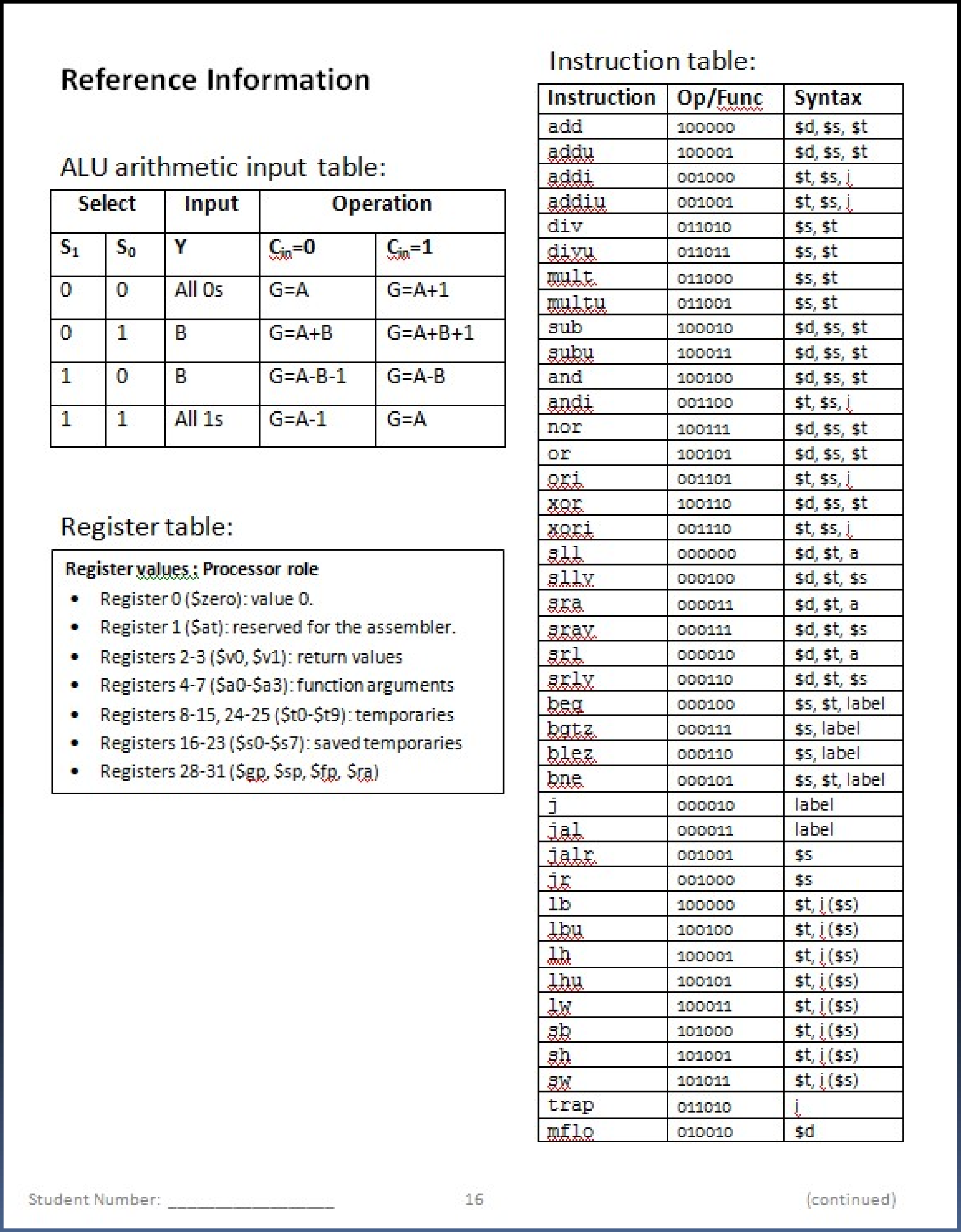
Question #5

- How do you write an assembly language program that performs `$t0` = `$t1` x `$t2` without using `mult` or `multu`?

- Coming up with a solution is easier if you ask yourself certain questions:
  - How can multiplication be done using `add`?
  - What if `$t2` stores a zero value?
  - How do you make a loop happen?
  - How do you make it stop looping?
  - What needs to be done at the beginning?

Question #5

- 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)

Question #5: The Math

- How can multiplication be done using `add`?

```
add $t0, $t0, $t1
```

(repeat this many times)

- What if `$t2` stores a zero value?

```
start:     beq $t2, $zero, end
           ...
           ...   # multiplication code here
           ...
end:       ...
```

Question #5: The Loop

- How do you make the loop happen?

```
start:      ...
            ...
            j start
end:        ...
```

- How do you make it stop looping?

```
start:      beq $t2, $zero, end
            ...
            addi $t2, $t2, -1
            j start
end:        ...
```

Question #5: The combination

- What needs to be done at the beginning?

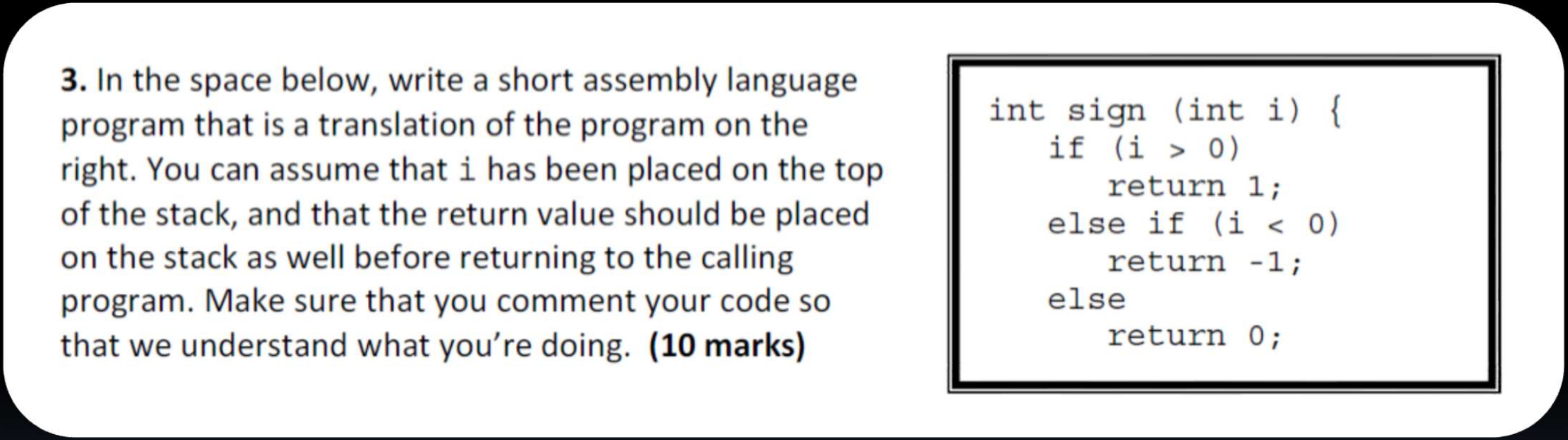
```
add $t0, $zero, $zero
```

- Final solution:

```
          add $t0, $zero, $zero
start:    beq $t2, $zero, end
          add $t0, $t0, $t1
          addi $t2, $t2, -1
          j start
end:      ...
```

Question #6

- 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?