CSc 252: Computer Organization

HW 3

due at 5pm, two days **before** the test

Turn in through GradeScope

Solutions

Policy Reminders

- You must turn in a single `PDF` to GradeScope. Other file formats will not be accepted. (Sorry, this is necessary for the sanity of the TAs!)
- You are allowed to work with other students on this homework, as we will not be grading it for correctness. However, **each student must turn in their own copy of the homework.**
- **Show your work for all problems.** While we won't be grading for correctness, you will not receive full credit unless you show your work.

  After all, showing your work is required on the test - and homeworks are intended to help you practice for the test!

Required Problems:

3.1(d), 3.2(d), 3.3(d), 3.4(d)

Allowable Instructions

When writing MIPS assembly, the only instructions that you are allowed to use (so far) are:

- `add, addi, sub, addu, addiu, subu`
- `beq, bne, j, jal, jr`
- `slt, slti`
- `and, andi, or, ori, nor, nori, xor, xori`
- `sll, srl, sra`
- `lw, lh, lb, sw, sh, sb`
- `la`
- `syscall`

While MIPS has many other useful instructions (and the assembler recognizes many pseudo-instructions), do not use them! We want you to learn the **fundamentals** of how assembly language works - you can use fancy tricks after this class is over.

Problem 3.1 - ALUs

In this problem, you will simulate an ALU, following the designs we've seen in class. However, to make the work easier, you will only need to simulate a 4-bit ALU, not a 32-bit one. (Just like in the 32-bit ALU, the `Less` input for ALU Element 0 - the LSB - comes from the adder output from ALU Element 3 - the MSB.)

3.1(a)

Give the proper control bits - that is, the `bNegate` and `aluOp` values - to configure the ALU to Add. Then, consider what would happen if the input values were as follows:

```
a = 1101
b = 0111
```

Give the following values as 4 bit numbers:

- The 4 AND bits
- The 4 OR bits
- The 4 Add results
- The 4 Out values (that is, the output from the various Result MUXes)

3.1(b)

Repeat part (a), except that you must give the configuration to perform subtraction. This time, give the following information:

- The 4 Add results
- The 4 Less inputs
- The 4 Out values (that is, the output from the various Result MUXes)

For this problem, use the following input values:

```
a = 0011
b = 1001
```

3.1(c)

Repeat part (b), except that you must give the configuration bits to perform Set-Less-Than. Report the same output information as from part(b).

For this problem, use the following input values:

```
a = 1111
b = 0110
```

3.1(d) - Turn in this one

Repeat part (a), except that you must give the configuration to perform `OR`.

For this problem, use the following input values:

```
a = 0100
b = 1001
```

Solution: For OR, we must set `bNegate==0` and `aluOp==1`.

The AND bits are: `0000`

The OR bits are: `1101`

The Add results are : `1101`

Since `aluOp==1` we will select the OR result in every ALU Element as our the desired Output. Therefore, the output will be `1101`

Problem 3.2 - Tracking a Stack

3.2(a)

In this problem, `foo()` calls `bar()`. The first column shows the state of the stack while `foo()` is running, just before its startup code.

In the second column, show the state of the stack after the startup code in `foo()` has completed, but before the `jal` instruction. In the third column show the state of the stack after `bar()` has run its function prologue **and** it has saved any registers that are required.

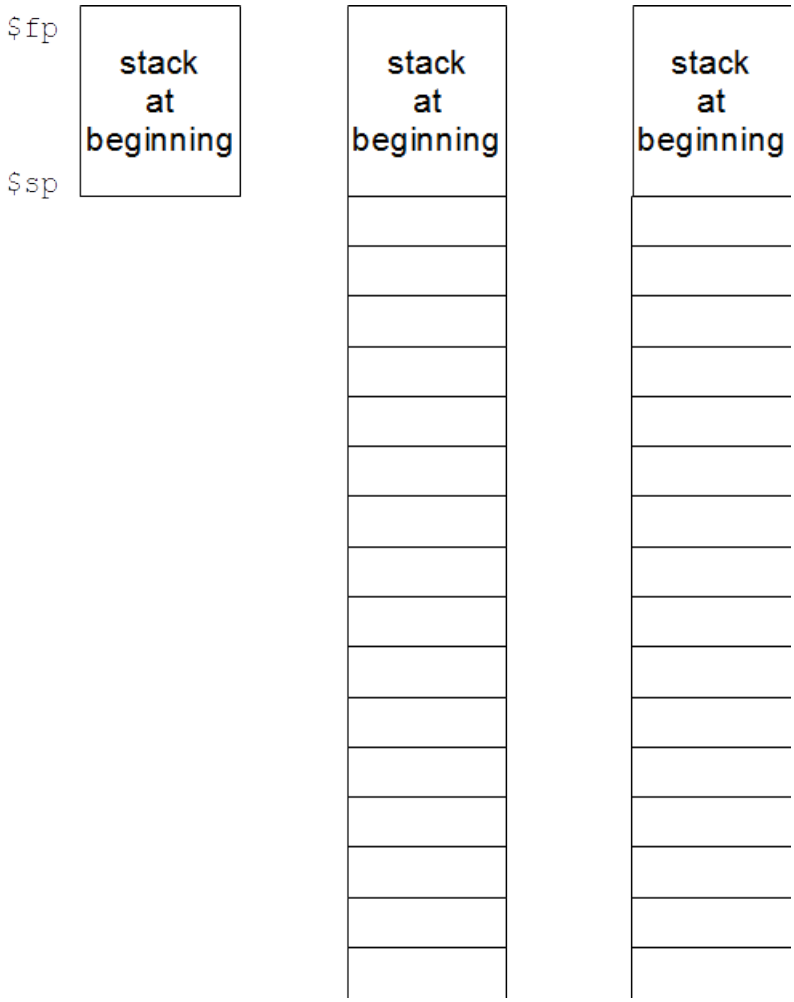
Make sure to mark:

- The positions of `$fp, $sp`
- All values which have been written to stack

  Use arg1, arg2, etc. for the various parameters. Use arg1 for the first parameter - which is stored in `$a0`.

Notes:

- When this problem begins, `foo()` is using (and wants to preserve) the registers `$t0, $t1, $t2, $s3, $s4, $s5`.
- `bar()` takes 5 parameters. It will need to the 1st, 3rd, and 5th on the stack.
- `bar()` will be using the following registers somewhere in its code: `$s0, $s1, $t2, $t3, $t4`.



3.2(b)

In this problem, `foo()` calls `bar()`. The first column shows the state of the stack while `foo()` is running, just before its startup code.

In the second column, show the state of the stack after the startup code in `foo()` has completed, but before the `jal` instruction. In the third column show the state of the stack after `bar()` has run its function prologue, but before it saved any other than the prologue (not even `aX` registers). In the third column show the state of the stack after `bar()` has saved all necessary registers.

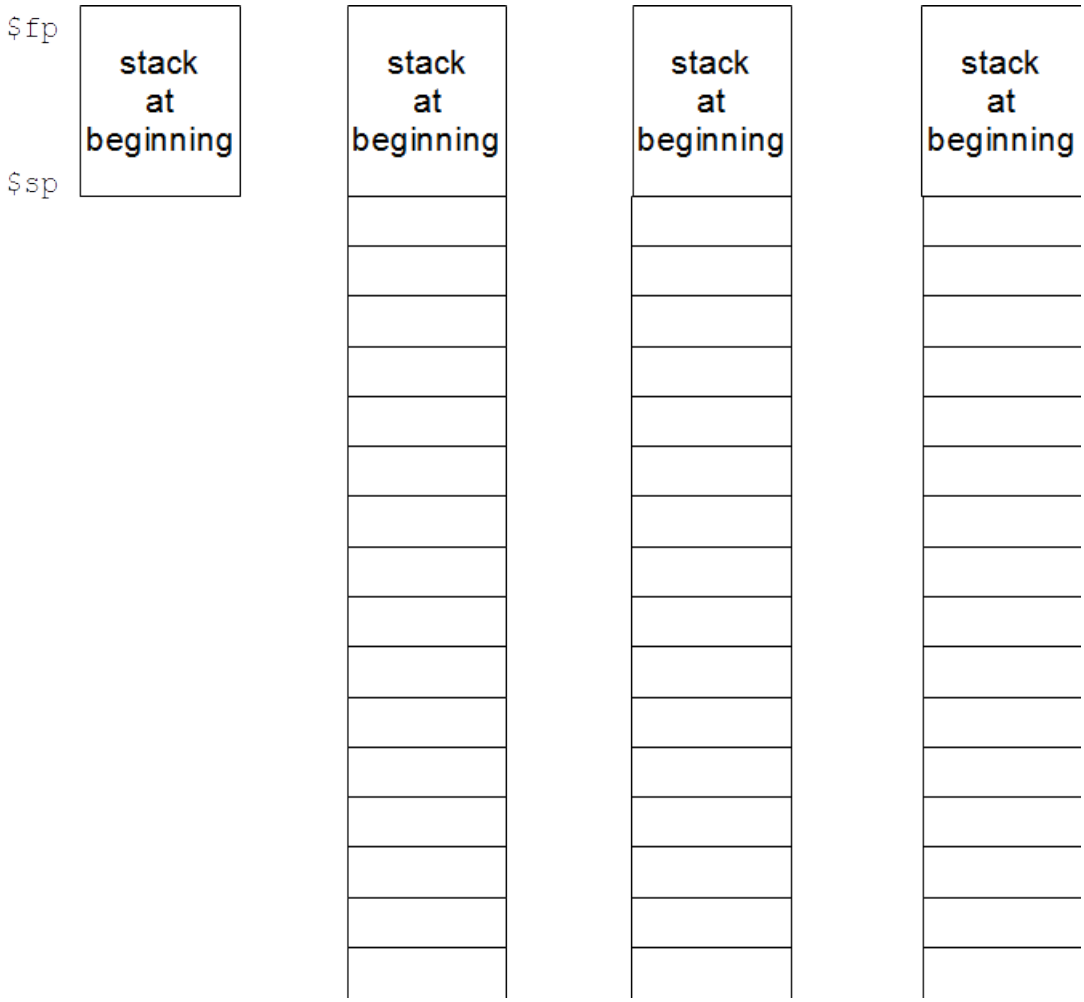
Make sure to mark:

- The positions of `$fp, $sp`
- All values which have been written to stack

  Use arg1, arg2, etc. for the various parameters. Use arg1 for the first parameter - which is stored in `$a0`.

Notes:

- When this problem begins, `foo()` is using (and wants to preserve) the registers `$s3, $s4, $t0, $t6, $t7`.
- `bar()` takes 2 parameters. It will need to store both on the stack.
- `bar()` will be using the following registers somewhere in its code: `$s3, $s4, $t0, $t1`.



3.2(c)

In this problem, `foo()` calls `bar()`. The first column shows the state of the stack while `foo()` is running, just before its startup code.

In the second column, show the state of the stack after the startup code in `foo()` has completed, but before the `jal` instruction. In the third column show the state of the stack after `bar()` has run its function prologue, but before it saved any other than the prologue (not even `aX` registers). In the fourth column show the state of the stack after `bar()` has saved all necessary registers.

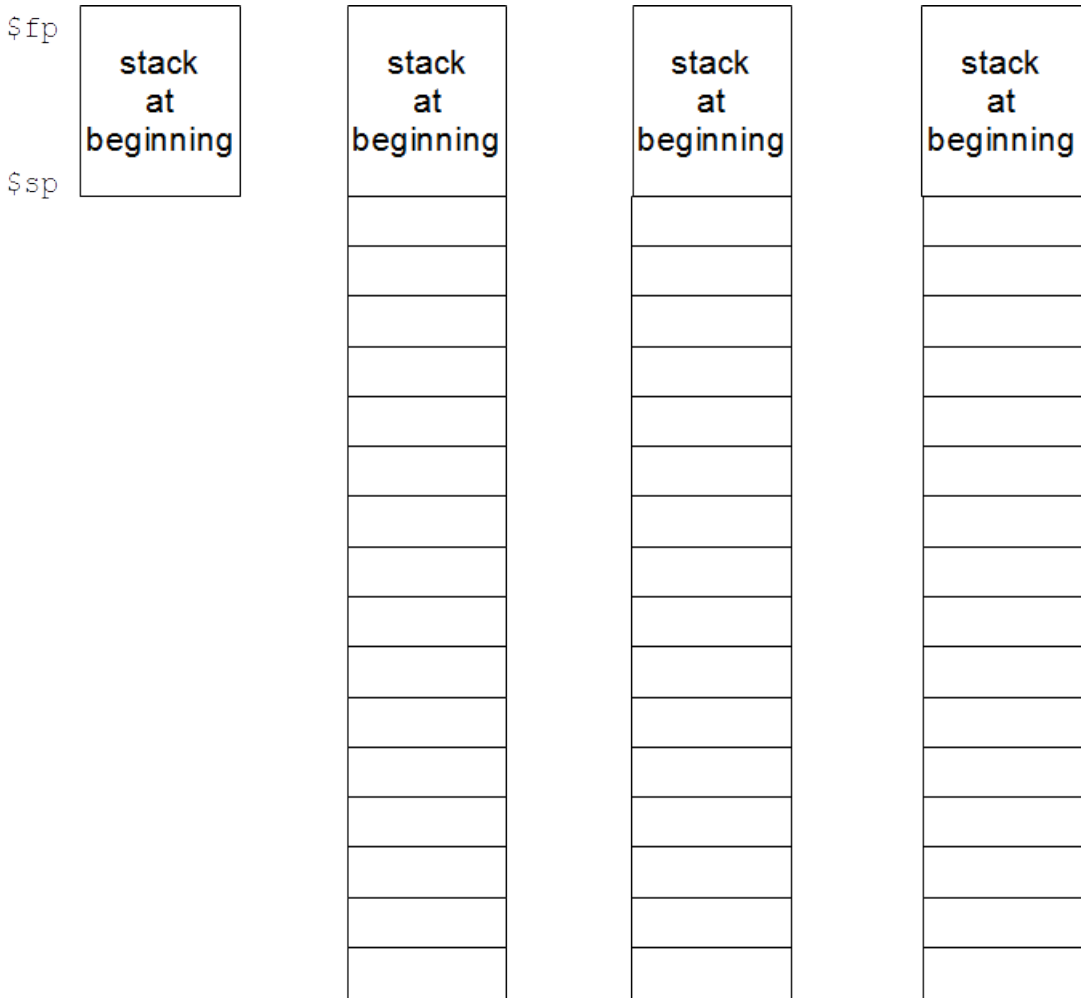
Make sure to mark:

- The positions of `$fp, $sp`
- All values which have been written to stack

  Use arg1, arg2, etc. for the various parameters. Use arg1 for the first parameter - which is stored in `$a0`.

Notes:

- When this problem begins, `foo()` is using (and wants to preserve) the registers `$t2, $s0, $s1, $s2, $s3`.
- `bar()` takes 8 parameters. It will need to store the first and fourth on the stack.
- `bar()` will be using the following registers somewhere in its code: `$t0, $t1, $t3, $t4, $s4, $s5, $s6, $s7`.



3.2(d) - Turn in this one

In this problem, `foo()` calls `bar()`. The first column shows the state of the stack while `foo()` is running, just before its startup code.

In the second column, show the state of the stack after the startup code in `foo()` has completed, but before the `jal` instruction. In the third column show the state of the stack after `bar()` has run its function prologue, but before it saved any other than the prologue (not even `aX` registers). In the fourth column show the state of the stack after `bar()` has saved all necessary registers.

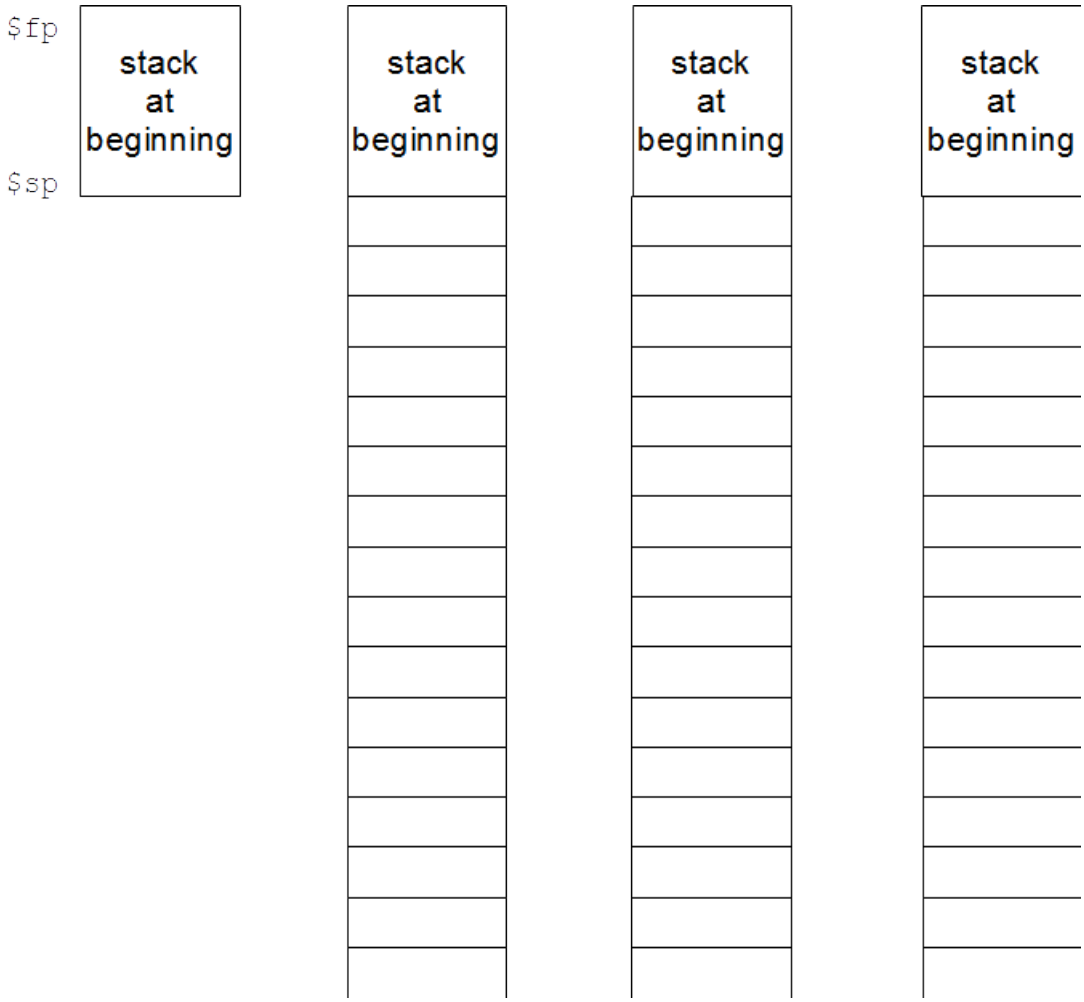
Make sure to mark:

- The positions of `$fp, $sp`
- All values which have been written to stack

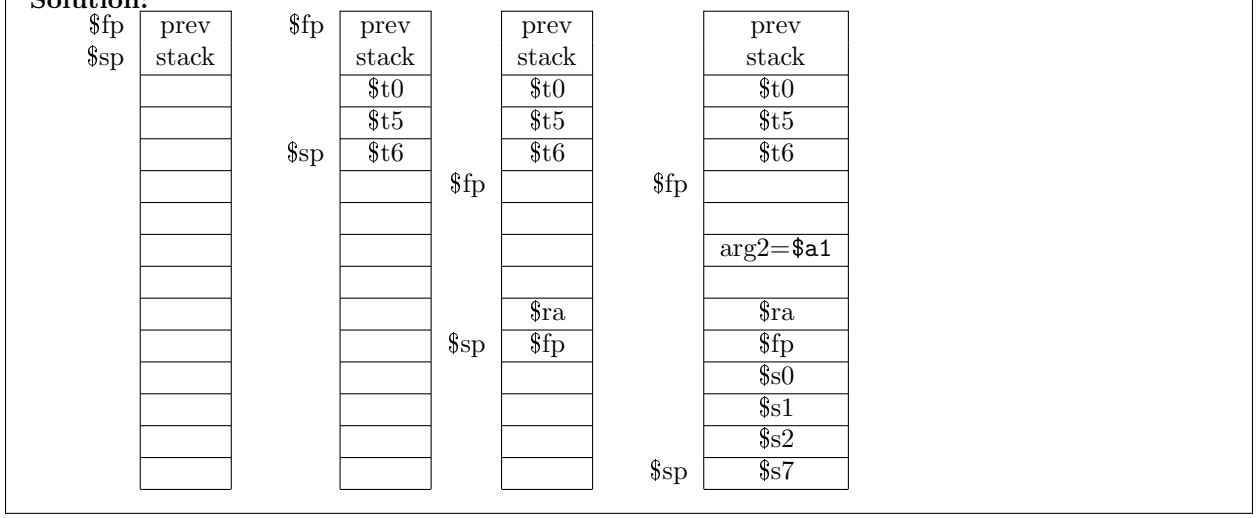
  Use arg1, arg2, etc. for the various parameters. Use arg1 for the first parameter - which is stored in `$a0`.

Notes:

- When this problem begins, `foo()` is using (and wants to preserve) the registers `$t0, $t5, $t6, $s0, $s2, $s5, $s6`.
- `bar()` takes 2 parameters. It will need to store the second on the stack.
- `bar()` will be using the following registers somewhere in its code: `$t1, $t2, $t3, $t4, $t5, $t6, $s0, $s1, $s2, $s7`.



Solution:


$fp
$sp
prev stack
$fp
$sp
prev stack
$t0
$t5
$t6
prev stack
$t0
$t5
$t6
$fp
$ra
$sp
$fp
prev stack
$t0
$t5
$t6
arg2=$a1
$ra
$fp
$s0
$s1
$s2
$sp
$s7

Problem 3.3 - Calling Functions

For each problem below, I have given a set of registers which are currently in use and which need to be saved, and then a function call with a certain set of parameters. Write the code that should run in the **caller** - including the startup and cleanup code. Save every register which might be destroyed by the called function, but do **not** save any register which would be saved by the called function.

3.3(a)

Registers to save: `$s0, $s1, $s3, $t0, $t1, $t2`
Call to make: `foo(1,2)`

3.3(b)

Registers to save: `$s2, $s4, $s6, $t1, $t3, $t5`
Call to make: `bar(0, 0x1234, str)`

(Assume that `str` is a label of a string. Pass the **address** of the first character as the third parameter.)

3.3(c)

Registers to save: `$t1, $t2, $t3, $t4, $s0, $s1`
Call to make: `fred(123, 456, 0xabc, 0xdef, 0, 10)`

3.3(d) - Turn in this one

Registers to save: `$t0, $t1, $s1, $s3, $s5, $s7`
Call to make: `qwerty('a', 10, 'B', -2, 0xffff)`

Solution:

```
addiu  $sp, $sp, -8
sw     $t0,  4($sp)
sw     $t1,  0($sp)

addi   $a0, $zero, 'a'       # 'a' is the same as 97
addi   $a1, $zero, 10
addi   $a2, $zero, 'B'       # 'B' is the same as 66
addi   $a3, $zero, -2

# we don't want to save t5, so it's OK to use it as a
# temporary in this code.
ori    $t5, $zero, 0xffff   # t5   = 0xffff
sw     $t5,   -4($sp)       # arg5 = 0xffff

jal    qwerty

lw     $t1,  0($sp)
lw     $t0,  4($sp)
addiu  $sp, $sp, 8
```

Problem 3.4 - Function Prologues and Epilogues

For each problem below, I have given a set of registers which will be modified by a certain function, along with the number of parameters it has; I will also list if any of the parameters need to be copied to the stack (so that they can be saved across a future function call). Write the appropriate prologue and epilogue **plus** the register save/restore code) for this function.

3.4(a)

Registers which will be modified: `$s0, $s1, $s3, $t0, $t1, $t2`
Number of Parameters: 3
Parameters to Save on Stack: `$a2`

3.4(b)

Registers which will be modified: `$s2, $s4, $s6, $t1, $t3, $t5`
Number of Parameters: 3
Parameters to Save on Stack: None

3.4(c)

Registers which will be modified: `$t1, $t2, $t3, $t4, $s0, $s1`
Number of Parameters: 6
Parameters to Save on Stack: `$a0, $a1, $a2, $a3`

3.4(d) - Turn in this one

Registers which will be modified: `$t2, $t3, $t4, $t5, $t6, $s1`
Number of Parameters: 5
Parameter(s) to Save on Stack: `$a1`

Solution:

```
addiu  $sp, $sp, -28
sw     $ra, 4($sp)
sw     $fp, 0($sp)
addiu  $fp, $sp, 24

# save the aX registers
sw     $a1, 12($sp)

# save the sX registers that we're going to use.
addiu  $sp, $sp, -4
sw     $s1, 0($sp)

... body ...

lw     $s1, 0($sp)
addiu  $sp, $sp, 4

lw     $fp, 0($sp)
lw     $ra, 4($sp)
addiu  $sp, $sp, 28
jr     $ra
```

EXAMPLES

Example: Problem 3.1(a)

For addition, we must set `bNegate==0` and `aluOp==2`.

The AND bits are: `0101`

The OR bits are: `1111`

The Add results are : `0100` **Instructors Note:** You will notice that the value had a carry-out, but our 4-bit addeer simply **discarded** the extra bits in the result.

Since `aluOp==2` we will select the Add result in every ALU Element as our the desired Output. Therefore, the output will be `0100`

Example: Problem 3.1(b)

For subtraction, we must set `bNegate==1` and `aluOp == 2.`

Since we are performing subtraction, we have a Carry-In to our first element - and the B input is negated. Thus, the adder result will be:
`1010`

The Less input #0 will be 1, of course, because the MSB of the Adder result is wired to the Less input of ALU ELemenet 0. All three other Less inputs are zero; thus, the entire Less value is `0001`.

Since `aluOp==2` we will select the Add result in every ALU Element as our the desired Output. Therefore, the output will be `1010`.

Example: Problem 3.1(c)

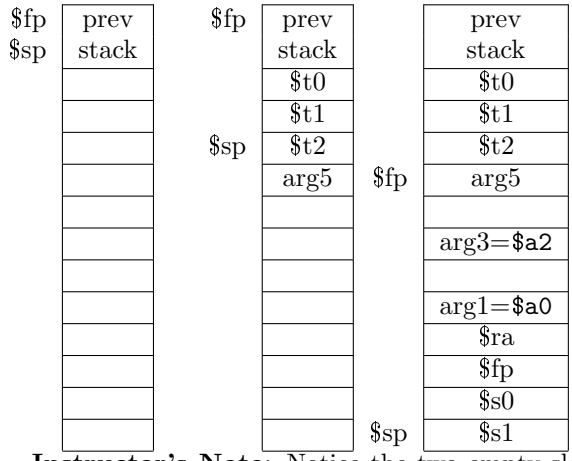
For set-less-than, we must set `bNegate==1` and `aluOp==3`.

The Add results are : `1001`

The MSB of the Add result wraps around, to be the LSB of the Less input. Therefore, the Less inputs are : `0001`. **Instructor's Note:** Since `a` is negative (MSB=1) and `b` is positive, it is reasonable that `slt` would produce a 1 result.

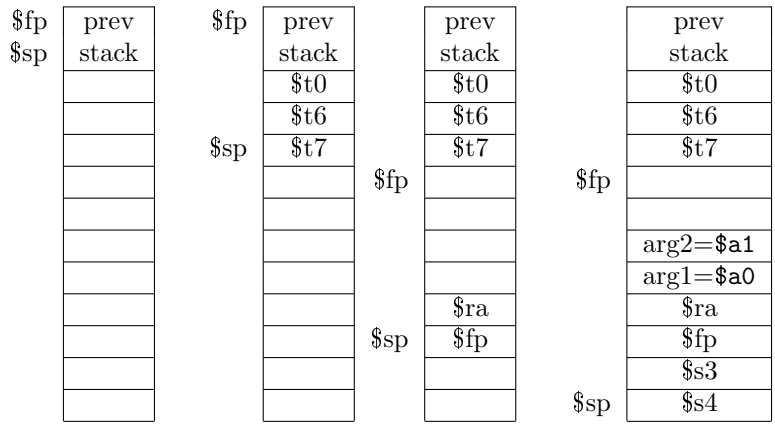
Since `aluOp==3`, we will select the Less input in every ALU Element as our the desired Output. Therefore, the output will be `0001` .

Example: Problem 3.2(a)

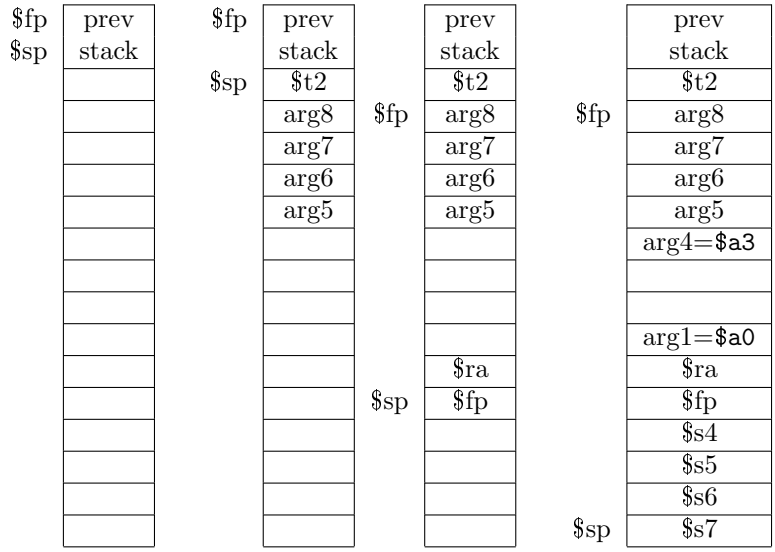


Instructor's Note: Notice the two empty slots in the last column. These are where the 2nd and 4th arguments (`$a1,$a3`) would be stored on the stack, if they need to be saved somewhere. But in this problem, it wasn't required that they be saved.

Example: Problem 3.2(b)



Example: Problem 3.2(c)



Example: Problem 3.3(a)

```
addiu  $sp, $sp, -12
sw     $t0, 8($sp)
sw     $t1, 4($sp)
sw     $t2, 0($sp)

addi   $a0, $zero, 1
addi   $a1, $zero, 2

jal    foo

lw     $t2, 0($sp)
lw     $t1, 4($sp)
lw     $t0, 8($sp)
addiu  $sp, $sp, 12
```

Example: Problem 3.3(b)

```
addiu  $sp, $sp, -12
sw     $t1, 8($sp)
sw     $t3, 4($sp)
sw     $t5, 0($sp)

addi   $a0, $zero, 0
addi   $a1, $zero, 0x1234
la     $a2, str

jal    bar

lw     $t5, 0($sp)
lw     $t3, 4($sp)
lw     $t1, 8($sp)
addiu  $sp, $sp, 12
```

Example: Problem 3.3(c)

```
addiu  $sp, $sp, -16
sw     $t1, 12($sp)
sw     $t2,  8($sp)
sw     $t3,  4($sp)
sw     $t4,  0($sp)

addi   $a0, $zero, 123
addi   $a1, $zero, 456
addi   $a2, $zero, 0xabc
addi   $a3, $zero, 0xdef

# we don't want to save t5, so it's OK to use it as a
# temporary in this code.
sw     $zero, -8($sp)       # arg5 = 0
addi   $t5, $zero, 10
sw     $t5,   -4($sp)       # arg6 = 10

jal    fred

lw     $t4,  0($sp)
lw     $t3,  4($sp)
lw     $t2,  8($sp)
lw     $t1, 12($sp)
addiu  $sp, $sp, 16
```

Example: Problem 3.4(a)

```
addiu  $sp, $sp, -24
sw     $ra, 4($sp)
sw     $fp, 0($sp)
addiu  $fp, $sp, 20

sw     $a2, 16($sp)

addiu  $sp, $sp, -12
sw     $s0, 8($sp)
sw     $s1, 4($sp)
sw     $s3, 0($sp)

... body ...

lw     $s3, 0($sp)
lw     $s1, 4($sp)
lw     $s0, 8($sp)
addiu  $sp, $sp, 12

lw     $fp, 0($sp)
lw     $ra, 4($sp)
addiu  $sp, $sp, 24
jr     $ra
```

Example: Problem 3.4(b)

```
addiu  $sp, $sp, -24
sw     $ra, 4($sp)
sw     $fp, 0($sp)
addiu  $fp, $sp, 20

addiu  $sp, $sp, -12
sw     $s2, 8($sp)
sw     $s4, 4($sp)
sw     $s6, 0($sp)

... body ...

lw     $s6, 0($sp)
lw     $s4, 4($sp)
lw     $s2, 8($sp)
addiu  $sp, $sp, 12

lw     $fp, 0($sp)
lw     $ra, 4($sp)
addiu  $sp, $sp, 24
jr     $ra
```

Example: Problem 3.4(c)

```
addiu  $sp, $sp, -32
sw     $ra, 4($sp)
sw     $fp, 0($sp)
addiu  $fp, $sp, 28

# save the aX registers
sw     $a0,  8($sp)
sw     $a1, 12($sp)
sw     $a2, 16($sp)
sw     $a3, 20($sp)

# save the sX registers that we're going to use.
addiu  $sp, $sp, -8
sw     $s0, 4($sp)
sw     $s1, 0($sp)

... body ...

lw     $s1, 0($sp)
lw     $s0, 4($sp)
addiu  $sp, $sp, 8

lw     $fp, 0($sp)
lw     $ra, 4($sp)
addiu  $sp, $sp, 32
jr     $ra
```