CS 250: Computer Architecture

Midterm Exam - October 22, 2012
Closed-book/notes/discussion

TIME: 90 minutes (8:00 PM - 9:30 PM)

Name:	Solution
Email:	dxu@cs.purdue.edu

- Total points = 100 points.
- Make sure you have **11 pages – including the cover page**.
- Please answer all questions clearly and concisely. If you are making any assumptions, please state them clearly.
- Good luck!

Please do not write below this line

Question	Score (grader)
1	
2	
3	
4	
5	
TOTAL	

1. **(20 points) True or False? Indicate by entering a 'T' (if true) or 'F' (if false) in the appropriate brackets before each question (grading: +2 for each correct answer, 0 for incorrect or no answer).**

(1) [F] Among the five classic components of a computer, the datapath, control, and memory altogether form the processor. memory IS NOT part of processor

(2) [F] 6 bits are sufficient to represent every student taking CS250 this semester. $2^6 = 64$, we need 8 bits for 136 students

(3) [F] A multiplexor with x data input bits should have 2^x input selector bits. x should be 2^x — 2^x should be x

(4) [F] In both one's complement and two's complement schemes, there are two representations for zero. only ONE zero under 2's complement

(5) [F] In MIPS, both caller-saved and callee-saved registers are saved in the stack, which is a special area in the register file. Stack in memory, not in reg. file

(6) [T] In MIPS, *not* all I-type instructions involve accessing the data memory.

(7) [T] In MIPS, the 26-bit "address" field of the `j` instruction is treated as an unsigned integer instead of a signed integer.

(8) [T] A stack grows from high to low addresses.

(9) [T] In the simple MIPS processor, the output of the ALU will determine the value of one of the control signals when executing the **`beq`** instruction.

(10) [T] Some digital circuits, such as the one that implements a register, maintain states.

The rest of this page is intentionally blank.

2. (40 points) Short Q&A. Your answer to each question should have no more than three sentences.

(1) (5 points) What are the five components of a computer system?

Datapath, Control, memory, input, output

processor

(2) (4 points) Under two's complement, what is the decimal value of binary number 1111 0010? What is the binary form of the same number if we sign-extend it to 16 bits?

-14

1111 1111 1111 0010

(3) (5 points) Under one's complement, what are the binary representations of decimal numbers 7 and (-3), respectively? Show how to perform 7 + (-3) in the *binary* form under one's complement.

7_{10} 0 1 1 1

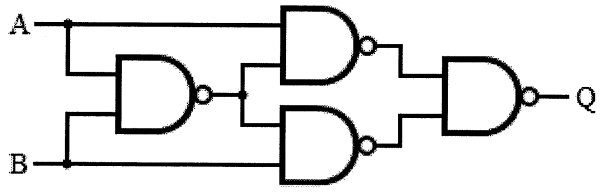
-3_{10} +) 1 1 0 0

(1) 0 0 1 1

1

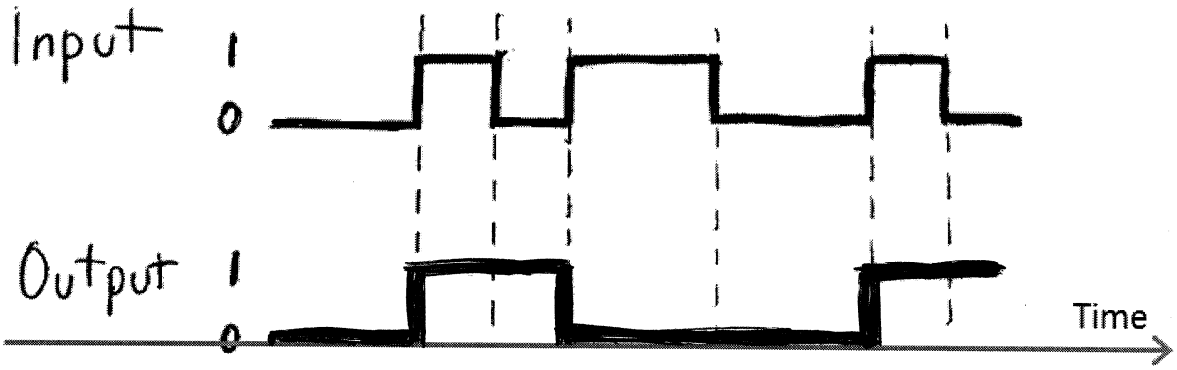
0 1 0 0 ← 4_{10}

(4) (3 points) Give a one-sentence (or even better, one-phrase) description of the function performed by the circuit below:

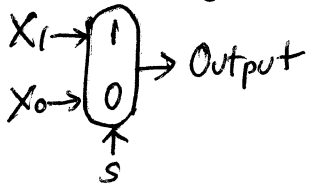


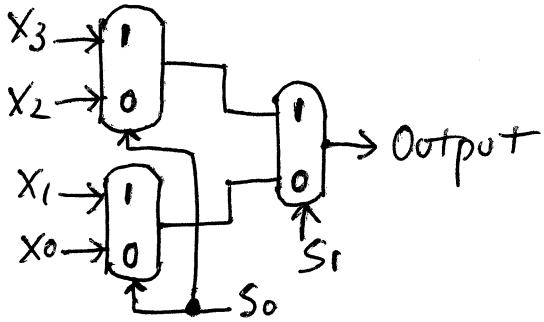
Q = A XOR B

(5) (4 points) Plot the output of a flip-flop in the following diagram.



(6) (4 points) A 2-to-1 multiplexor takes two 1-bit inputs X_0 and X_1 and selects one of the two inputs as its output, depending on the selector signal. Show how to construct a 4-to-1 multiplexor using (exactly) three 2-to-1 multiplexors. No other gates are allowed. The 4-to-1 multiplexor takes four 1-bit inputs X_0, X_1, X_2, X_3 and selects one of the four as its output, according to its 2-bit selector signal S_0 and S_1.



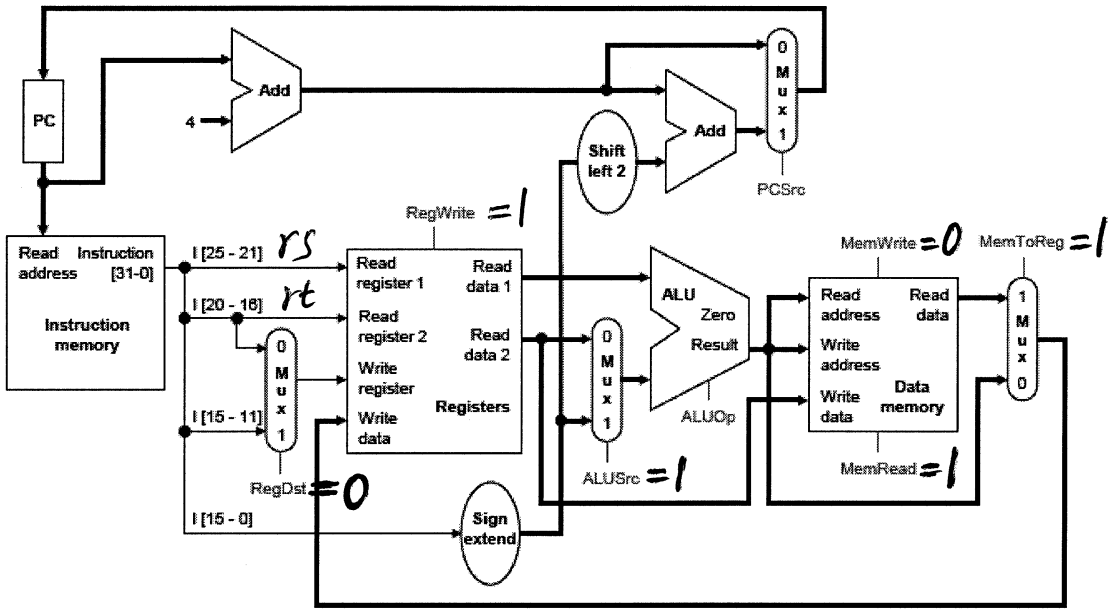


(7) (3 points) What is the *one* line of C code that corresponds to the following instruction sequence?

```
addi    $a0, $zero, 10
addi    $a1, $zero, 22
addi    $a2, $zero, 2012
jal     midterm
```

midterm(10, 22, 2012);

(8) (6 points) When the simple, single-cycle MIPS processor executes instruction **`lw $rt,2012($rs)`**(**`rs`**: `I[25-21]`; **`rt`**: `I[20-16]`), what are the values of control signals 'RegDst', 'RegWrite', 'ALUSrc', 'MemRead', 'MemWrite', and 'MemToReg'? Indicate the values in the figure.



(9) (3 points) Briefly explain why we need the "Shift left 2" component (see the figure in Question (8)) for the execution of branch (e.g., **`beq`**) instructions.

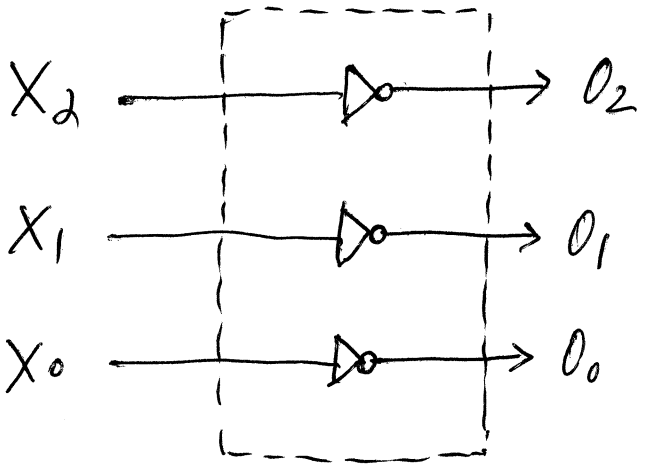
The "offset" field in beq is the distance between the beq instruction and the branch target -- in # of instructions. Hence we need the bit-shifter to convert it to distance in »BYTES«.

(10) (3 points) To implement the `Factorial` function in MIPS assembly, we have studied a recursive version and a non-recursive version. Briefly explain why the non-recursive version tends to run faster than the recursive version.

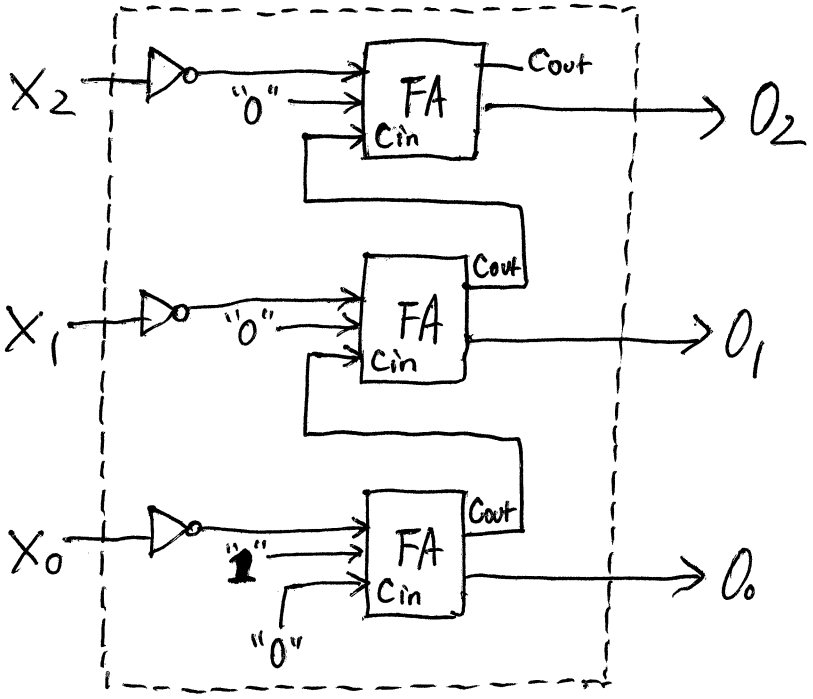
- Recursive version involves many stack push/pop operations, which are memory accesses. And memory accesses are slow.
- Non-recursive version does NOT involve memory accesses. It only involves registers, which are fast.

3. (12 points) Digital circuit design Given a signed integer X, the additive inverse of X is (-X). For example, the additive inverse of -3_{10} is 3_{10}; and the additive inverse of 1_{10} is -1_{10}.

(1) (6 points) Design a digital circuit to generate the additive inverse of a three-binary-digit signed integer (with bits X_2 X_1 X_0 -- X_2 being the sign bit), under the *1's complement* scheme.



(2) (6 points) Solve the same problem as **(1)** but under the *2's complement* scheme. For convenience, let's assume that the input will *not* be 100_2 (i.e. -4_{10}). (Hint: you can use a number of *one-digit full adders* as building blocks.)



4. (15 points) MIPS assembly programming (recursive) An integer power of x can be efficiently computed using the following recursion (assuming that $x > 0$ and $m \geq 0$):

$x^m = 1$, if m = 0;
$x^m = (x^2)^{m/2}$, if m is even;
$x^m = x * (x^2)^{(m-1)/2}$, if m is odd.

If we define function $p(x, m) = x^m$, then the recursive definition of p will be:

p(x, m) = 1, if m = 0;
p(x, m) = $p(x^2, m/2)$, if m is even;
p(x, m) = $x * p(x^2, (m-1)/2)$, if m is odd.

Here is the C code that implements the above recursive function:

```
p(int x, int m)
{
   if (m = = 0) return 1;
   else if (m & 0x1)      /* i.e. if m is odd*/
              return (x * p(x*x, (m-1)/2));
        else
              return p(x*x, m/2);
}
```

A skeleton of the equivalent MIPS assembly code is given on the next page. Your mission is to understand the assembly code skeleton and complete it.

(1) (6 points) Fill in each of the first three (smaller) boxes with *one* missing instruction.

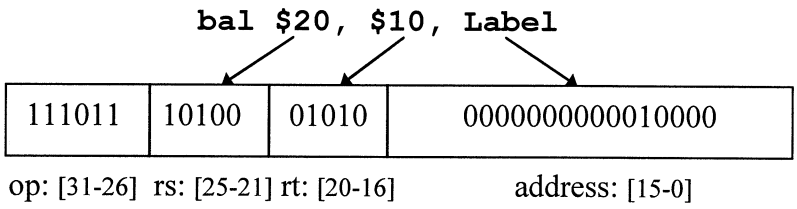
(2) (9 points) Fill in the last (large) box with *a sequence of* instructions to complete the code. And *Comment* each instruction.

The rest of this page is intentionally blank.

See next page

```
# x is the first argument and has been stored in $a0
# m is the second argument and has been stored in $a1
p:      subi $sp, $sp, 12          # create stack frame
        sw   $a0, 0($sp)           # save x
        sw   $a1, 4($sp)           # save m
        sw   $ra, 8($sp)           # save return address

# if m is greater than 0, jump to 'rec'
        bne  $a1, $zero, rec

# if m is equal to 0, return 1
        addi $v0, $zero, 1         # $v0 ← 1
        addi $sp, $sp, 12          # destroy stack frame
        jr $ra                     # return

# if m is greater than 0, do the recursion
rec:  andi  $t1, $a1, 1            # $t1 ← m & 0x1
        beq  $t1, $zero, even      # if m is even, jump to 'even'

odd:    mul  $a0, $a0, $a0         # $a0 ← x*x
        subi $a1, $a1, 1           # compute m - 1
        srl  $a1, $a1, 1           # $a1 ← (m - 1)/2
        jal p                      # call p(x*x, (m-1)/2)
      # after returning from p(x*x, (m-1)/2)
        lw   $a0, 0($sp)           # restore x
        lw   $a1, 4($sp)           # restore m
        lw   $ra, 8($sp)           # restore return address
        addi $sp, $sp, 12          # destroy stack frame
        mul  $v0, $a0, $v0         # $v0 ← x*p(x*x,(m-1)/2)
        jr $ra                     # return

even:   mul $a0, $a0, $a0          # $a0 ← x*x
        srl $a1, $a1, 1            # $a1 ← m/2
        jal p                      # call p(x*x, m/2)
        lw $a0, 0($sp)             # restore x
        lw $a1, 4($sp)             # restore m
        lw $ra, 8($sp)             # restore $ra
        addi $sp, $sp, 12          # destroy stack frame
        jr $ra                     # return
```

5. (13 points) Simple, single-cycle MIPS processor Dr. Art Vandelay, a computer architect, proposes to add a new "branch-and-link" instruction (**`bal`**) to the simple, single-cycle MIPS processor. For example, the following instruction



compares the values of registers $20 and $10. If (and only if) they are equal, a function will be called and the return address will be written to $ra (namely $31) – similar to instruction **`jal`**. However, the starting address of the function is computed in a way similar to instruction **`beq`**:

function_address = PC + 4 + shift_left_two_bits(sign_extend(Label))

(1) (4 points) The datapath for **`bal`** is similar to that for **`beq`** – the only difference is that **`bal`** also involves the writing of the return address to $ra ($31) before jumping to the starting address of the function. In the figure on the next page, show your extension to the MIPS datapath to support the execution of **`bal`**.

See first figure on the next page.

(2) (4 points) Suppose the values of $20 and $10 are both 2012 right before the execution of **`bal $20,$10,Label`**. Highlight the portion of the datapath involved in its execution. Use the same figure on the next page.

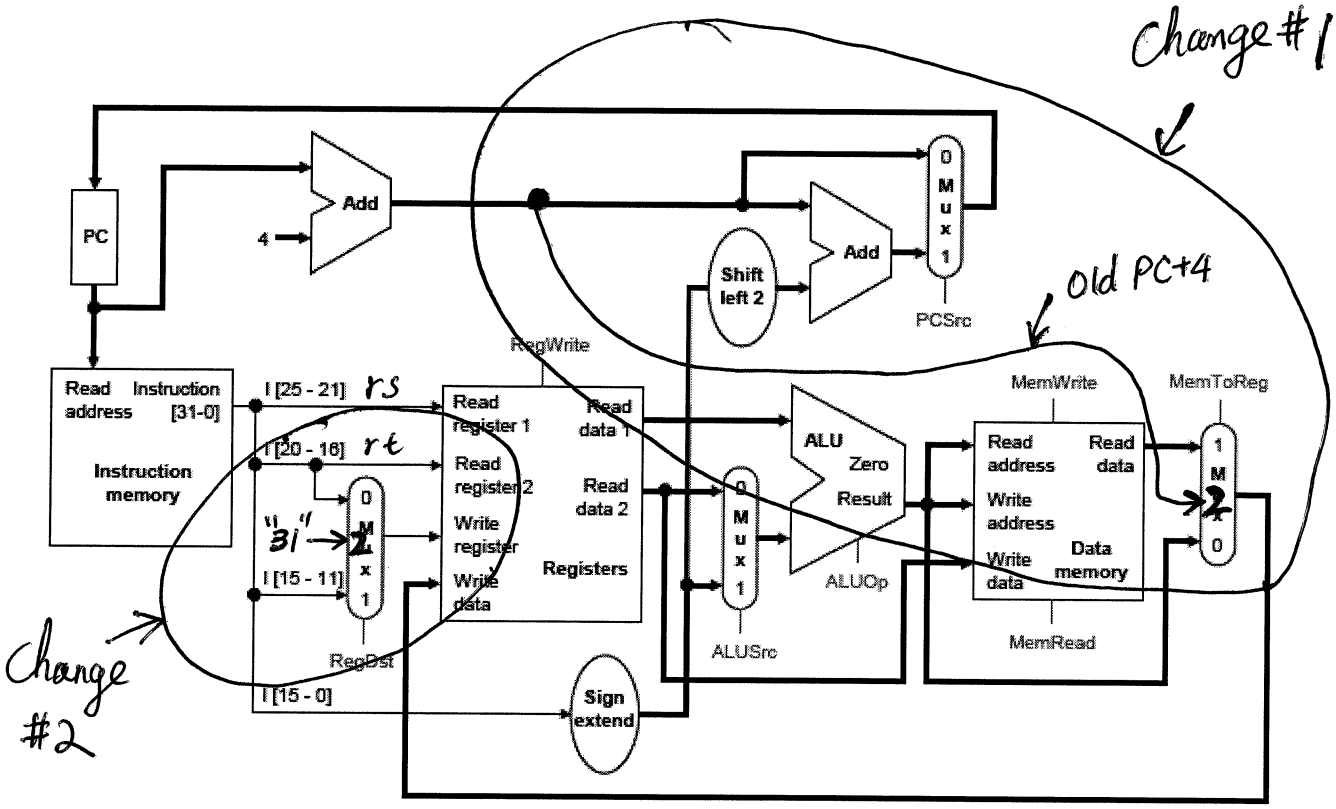
See second figure on the next page

(3) (5 points) In the same situation as in (2), indicate (in the same figure) the values of control signals 'RegDst', 'RegWrite', 'ALUSrc', 'PCSrc', and 'MemToReg'.

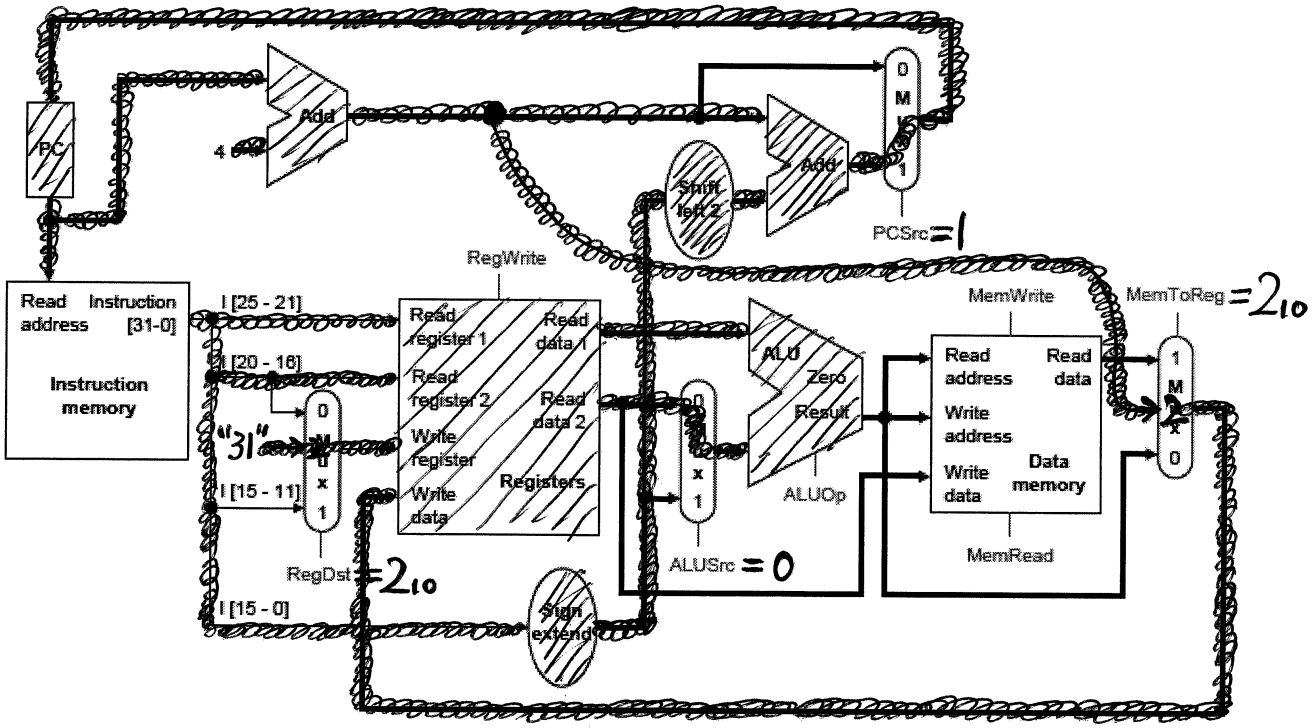
See second figure on the next page

(4) (0 points – for your entertainment only, DON'T spend more than 10 seconds on it!) Let's play jeopardy! *"Art Vandelay is an imaginary character in this nine-year-long (1989-1998) American sitcom."*

"What is Seinfeld ?"



A backup figure in case you need one:



Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

Example
add $rd, $rs, $rt
bne $rs, $rt, addr
J addr

FIGURE 2.26 MIPS instruction formats in this chapter. Highlighted portions show instruction formats introduced

Category	Example Instruction	Meaning
Arithmetic	add $t0, $t1, $t2	$t0 = $t1 + $t2
	sub $t0, $t1, $t2	$t0 = $t1 – $t2
	rem $t0, $t1, $t2	$t0 = $t1 % $t2
	div $t0, $t1, $t2	$t0 = $t1 / $t2
Logical	and $t0, $t1, $t2	$t0 = $t1 & $t2 (Logical AND)
	or $t0, $t1, $t2	$t0 = $t1 | $t2 (Logical OR)
	sll $t0, $t1, $t2	$t0 = $t1 << $t2 (Shift Left Logical)
	srl $t0, $t1, $t2	$t0 = $t1 >> $t2 (Shift Right Logical)
	sra $t0, $t1, $t2	$t0 = $t1 >> $t2 (Shift Right Arithmetic)
Register Setting	move $t0, $t1	$t0 = $t1
	li $t0, 100	$t0 = 100
Data Transfer	lw $t0, 100($t1)	$t0 = Mem[100 + $t1] 4 bytes
	lb $t0, 100($t1)	$t0 = Mem[100 + $t1] 1 byte
	sw $t0, 100($t1)	Mem[100 + $t1] = $t0 4 bytes
	sb $t0, 100($t1)	Mem[100 + $t1] = $t0 1 byte
Branch	beq $t0, $t1, Label	if ($t0 = $t1) go to Label
	bne $t0, $t1, Label	if ($t0 ≠ $t1) go to Label
	bge $t0, $t1, Label	if ($t0 ≥ $t1) go to Label
	bgt $t0, $t1, Label	if ($t0 > $t1) go to Label
	ble $t0, $t1, Label	if ($t0 ≤ $t1) go to Label
	blt $t0, $t1, Label	if ($t0 < $t1) go to Label
Set	slt $t0, $t1, $t2	if ($t1 < $t2) then $t0 = 1 else $t0 = 0
	slti $t0, $t1, 100	if ($t1 < 100) then $t0 = 1 else $t0 = 0
Jump	j Label	go to Label
	jr $ra	go to address in $ra
	jal Label	$ra = PC + 4; go to Label

The second source operand of the arithmetic, logical, and branch instructions may be a constant.

Register Conventions

The *caller* is responsible for saving any of the following registers that it needs, before invoking a function.

$t0-$t9 $a0-$a3 $v0-$v1

The *callee* is responsible for saving and restoring any of the following registers that it uses.

$s0-$s7 $s8/$fp $sp $ra

Pointers in C:

Declarartion: either *char *char_ptr* -or- *char char_array[]* for *char c*

Dereference: *c = c_array[i]* -or- *c = *c_pointer*

Take address of: *c_pointer = &c*