University of California
College of Engineering
Computer Science Division -EECS

Fall 1996 D.E. Culler

CS 152 Midterm I

Your Name:_____

ID Number:_____

Discussion Section:_____

You may bring one double-sided pages of notes and you may use a calculator, but no book or computer. Please print your name clearly on the cover sheet and on every page. Show your work. Write neatly and be well organized. It never hurts to make it easy to grade.

Good luck.

Problem	**Possible**	**Score**
1	25	
2	20	
3	25	
4	20	
5	10	
Total	100	

Mean: 69, Max 97

Distribution: 100 - 85 (11), 84 - 70 (21), 69 - 55 (24), 54 - 40 (7), 39 - (2)

Problem 1 (25 points)

1a [5] State the five major components of a computer (according to Patterson and Hennessey).

1. Processor datapath
2. Processor Control
3. Memory
4. Input
5. Output

1b [5]: Assemble the following MIPS instruction into its binary machine representation

XORI $15, $0, 0x8000

answer: op(6) rsrtimmediate

001110	00000	01111	1000 0000 0000 0000

1c [5]: Decode your answer to 1b as a 32-bit 2's-complement integer

a) $2^{29} + 2^{28} + 2^{27} + 2^{20}$ b) $2^{30} - 2^{27} + 31 \times 2^{15}$ c) $7 \times 2^{27} + 31 \times 2^{20}$ d) $14 \times 2^{27} + 31 \times 2^{15}$

answer: b) 0011 1000 0000 1111 1000 0000 0000 0000

1d [5]: Decode your answer to 1b as an single precision IEEE floating-point number

a) $2^{-71} \times \frac{31}{256}$ b) $2^{-15} \times \frac{31}{256}$ c) $2^{-15} \times \frac{287}{256}$ d) $2^{112} \times \frac{287}{256}$

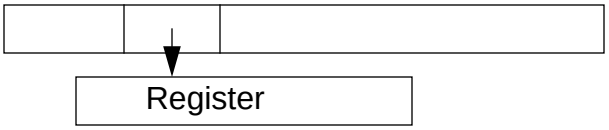
answer: c)

0	01110000	000 1111 1000 0000 0000 0000

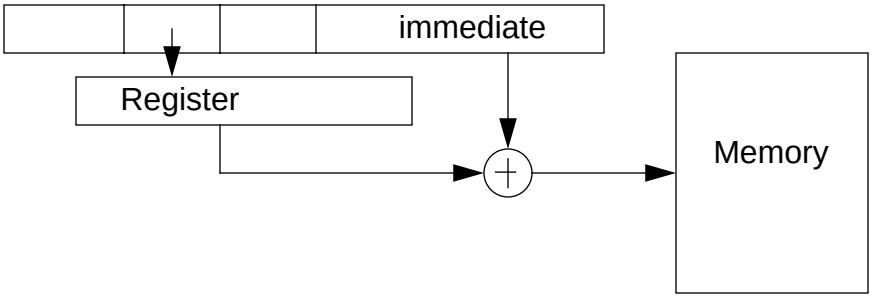
Grading: 1a) 1 point for each component; 1b) 1 point for each field; 1c), 1d) hit or miss for multiple choice.

1e[5] What are the four basic addressing modes supported by the MIPS R3000 instruction set? Draw a diagram of each. (Do not include the special cases that arise from setting one of the operands to zero.)

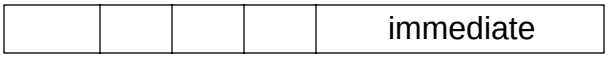
register-addressing: value is contained in a register specified in the instruction



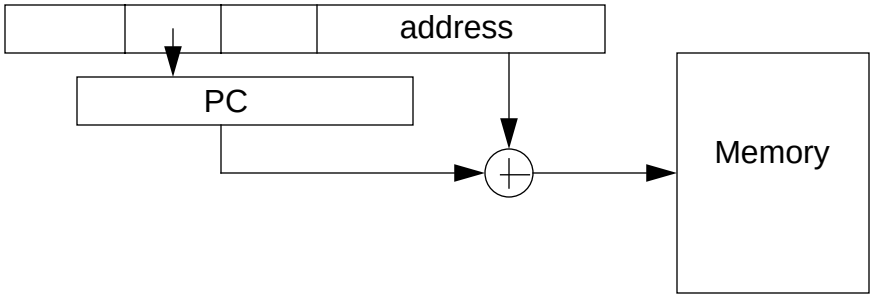
base (or displacement) addressing



immediate addressing: value is contained in the instruction



PC-relative addressing: PC <- PC + sign_ext(Imm16)



Points: 1 for each type. -2 for lack of diagrams, or lack of description. Other addressing modes were also given credit, if labeled correctly.

Problem 2 (20 points). You have been running an important program over and over on a 33 MHz DEC 5000 and you decide that you want to understand its performance. So you run it on a detailed simulator and collect the following instruction mix and breakdown of costs for each instruction type.

Instruction Class	Frequence (%)	Cycles
Arithmetic / logical	50	1
Load	20	2
Store	10	2
Jump	10	1
Branch	10	3

2.a. Calculate the CPI and MIPS for this program.

answer: CPI = 1.5 = 0.5 x 1 + 0.2 x 2 + 0.1 x 2 + 0.1 + 1 + 0.1 x 3, 22 MIPS

Points: +6 for correct answer

2.b. Suppose you turn on the optimizer and it eliminates 20% of the arithmetic/logic instructions (i.e. 10% of the instructions overall), but does not affect the other instruction classes. What is the speedup on this program with the optimizer. (Be sure you state and use the correct definition of speedup and show your work.)

a) 0.93 b) 1.07 c) 1.40 d) 1.55 e) none of these

Answer:

cycles per equivalent of average old instruction = 1.4

= 0.4 x 1 + 0.2 x 2 + 0.1 x 2 + 0.1 + 1 + 0.1 x 3.

speedup = 1.5 / 1.4 = 1.05

Points: + 6 for correct answer

2c. Calculate the CPI and MIPS for the optimized version of this program. Show your work. Compare your result to that of 2a and explain the change.

Answer CPI = 1.55, 21.2 MIPS. The program spends more of its time executing slow instruction, although there are fewer instructions overall.

Points: +8 for correct answer

Problem 3. (25 points) Complete the skeleton of MIPS assembly language below, generated by GCC for the following C function. All jumps and branches are delayed. Simple pseudo-ops are used to make it more readable. Annotate the instructions to explain what they do in the C program.

```
int scale (int *A, int n, int x)
{
        int i;
        for (i=0; i<n; i++) {
                A[i] = A[i]*x;
        }
}
```

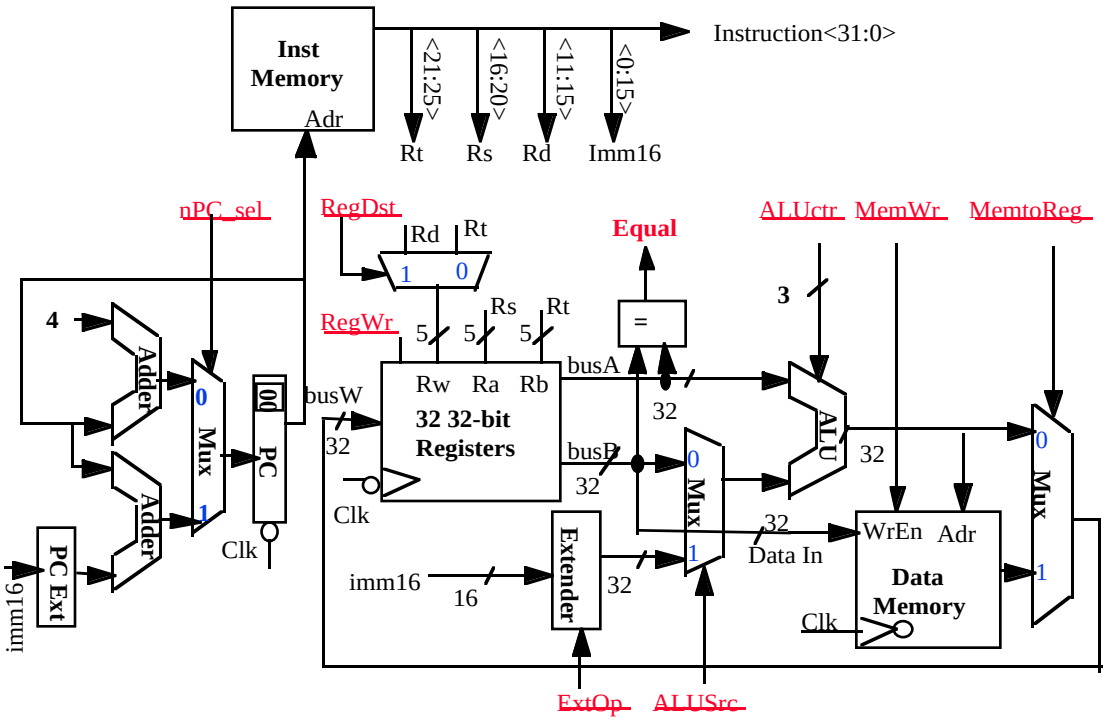
Register Usage
$4 - *A
$5 - n
$6 - x

```
.ent    scale
scale:
        subu    $sp,$sp,8       Adjust Stack pointer_____________________
        blez    $5,$L3          test n = 0, if so fall through_____________
        move    $7,$0           i = 0_____________________________________
$L5:
        sll     $3,$7,2         multiple i by 2___________________________
        addu    $3,$3,$4        address of A[i]___________________________
        lw      $2,0($3)        fetchA[i]_________________________________
        #nop
        mult    $6,$2           A[i]*x____________________________________
        mflo    $8              extract low 32 bits_______________________
        #nop
        addu    $7,$7,1         increment i_______________________________
        slt     $2,$7,$5        test i<n__________________________________
        bne     $2,$0,$L5       if i<n go to top of loop__________________
        sw      $8,0($3)        store A[i] <- A[i]*x______________________
$L3:
        addu    $sp,$sp,8       restore stack_____________________________
        j       $31             return____________________________________
        #nop
.end    scale
```

Points: 3 points for Register Usage; 18 points for instruction fill-ins; 4 points for explanation on left side

Problem 4 (20 points):

The Single-Cycle processor developed in class below (which was very similar to the one in the book) supports the instructions in the table below. (Note that, as in the virtual machine, the branch is not delayed.)



op | rs | rt | rd | shamt | funct = MEM[PC]
op | rs | rt | Imm16 =

inst	Register Transfers	
ADDU	R[rd] <– R[rs] + R[rt];	PC <– PC + 4
SUBU	R[rd] <– R[rs] – R[rt];	PC <– PC + 4
ORi	R[rt] <– R[rs] + zero_ext(Imm16);	PC <– PC + 4
LW	R[rt] <– MEM[R[rs] + sign_ext(Imm16)];	PC <– PC + 4
SW	MEM[R[rs] + sign_ext(Imm16)] <– R[rs];	PC <– PC + 4
BEQ	if (R[rs] == R[rt]) then PC <– PC + sign_ext(Imm16)] || 00	
	else PC <– PC + 4	

Consider adding the following instructions to our subset: ADDIU, AND, BLTZ, JAL . On the following pages, (1) write the register transfers for the new instructions, (2) modify the datapath to support these instructions and (3) specify the control points for each of the new instructions.

op | rs | rt | rd | shamt | funct = MEM[PC]
op | rs | rt | Imm16 =

inst	Register Transfers	
ADDIU	R[rt] <– R[rs] + SignExt(Imm16)	PC <– PC + 4
AND	R[rd] <– R[rs] and R[rt];	PC <– PC + 4
BLTZ	if $R[rs]_{31}$ = 1 then PC <– PC + SignExt(Imm16) || 00 else PC <– PC + 4	
JAL	R[31] <– PC + 4	PC <– $PC_{31..28}$ || Imm26|| 00

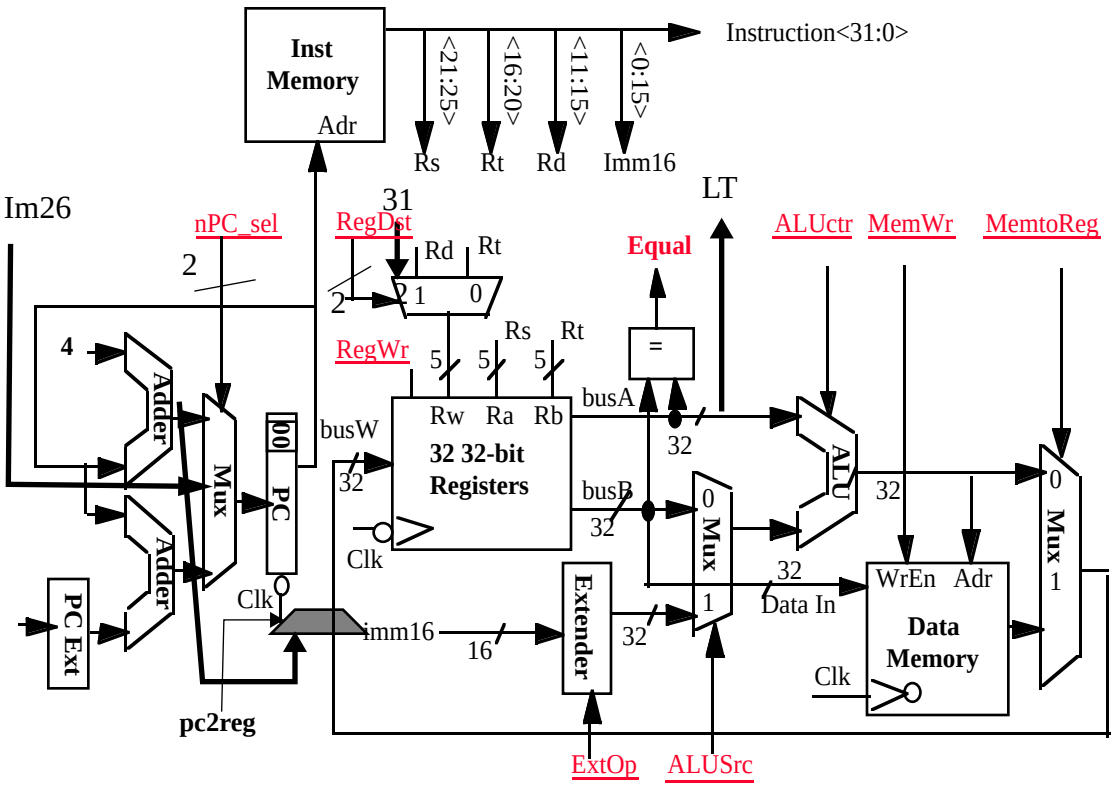
No changes to the datapath are required the the three arithmetic/logical instructions, except the ALU needs to support AND. There is already the capability to sign extend immediates and to operate on pairs of registers.

To support the BLTZ we need to:

- detect R[rs] < 0, this is just the sign bit of bus_A

To support JAL, we need to:

- provide a path from the PC+4 adder onto the register input bus (bus_w)
- provide 31 as the destination register number.



Problem 4 (cont)

TABLE 1.

	Ext	**ALUsrc**	**ALUCtr**	**MemWr**	**Mem2Reg**	**PC2Reg**	**RegDst**	**RegWr**	**nPC**
ADDIU	sign	1	add	0	0	0	0	1	0
AND	x	0	AND	0	0	0	1	1	0
BLTZ	x	x	x	0	x	x	x	0	lt
JAL	x	x	x	0	x	1	2	1	2

Points: 8 points for register transfers; 8 points for alterations to datapath; 4 points for control signals

Problem 5 (10 points)

Write a series of instructions to MIPS instructions to perform a signed 64-bit SLT. The operands are passed in registers A1:A0 and A3:A2 with the MSW in the higher numbered register. The result should be returned in register t0 with the same convention. Explain why your code works.

```
/* t0 = a1:a0 <? a3:a2 */

        BNE   a1, a3, L:

        slt     t0, a1, a3

        sltu   v0, a0, a2

L:
```

Points: +5 for sltu for least significant word; +5 for understanding when to use sltu comparator on least significant word (when $a1 = $a3). No points were taken off if you did not use the delay slots efficiently.