University of California
College of Engineering
Computer Science Division -EECS

Sp 1996 D.E. Culler

CS 152 Midterm I

Your Name:_____SOLUTION_____

ID Number:_____

Discussion Section:_____

You may bring two pages of notes and you may use a calculator, but no book or computer. Please print you name clearly on the cover sheet and on every page. The point value of each question is indicated in brackets. There are a total of 120 points. You have 170 minutes. Show your work. Write neatly and be well organized. It never hurts to make it easy to grade.

Good luck.

Problem	**Possible**	**Score**
1	40	
2	20	
3	20	
4	20	
5	20	
Total	120	

Problem 1 (40 points)

1a [3] State the five major components of a computer.

Processor datapath

Processor Control

Memory

Input

Output

1b [5] State five major distinct issues that must be addressed in an instruction set architecture.

programmable storage

data types and encodings

set of operations

instruction formats

number of operands

where besides memory can operands be located

how memory operands are speified (addressing modes)

1c [2] Define Little Endian.

word is addressed by the byte address of the least significant byte (least significant byte is at lowest address in the word.)

1d[3] Decode the following MIPS instruction using the opcode encoding table at the end of the exam (Fig A.18) 10001100111001110000000000000111. Give its RTL (register transfer language) meaning.

lw $7, 7($7) R[7] <– mem(R[7] + 7)

1e [3] What is the value of 1000 1100 1110 0111 0000 0000 0000 0111 as a 32bit 2s complement number?

number is negative so, comp+1=> 0111 0011 0001 1000 1111 1111 1111 1001

-(2^30 + 2^29 + 2^28 + 2^25 + 2^24 + 2^20 + 2^19 + 2^16 - 7) = 1931018233

1f [3] What is the value of 10001100111001110000000000000111 as a single-precision IEEE floating-point number?

-1.11001110000000000000111 x 2^(-102) or

about -3.56 x 10^(-31)

1g. [2] DRAM memory chips increase in capacity by a factor of 16 every how many years?

4x per 3 years => 16x per 6 years

1h. [3] Under what conditions is CPI a valid metric of performance comparison?

Time = Instruction Count x CPI x Cycle Time, so

Same program, same instruction set, and same cycle time

1i. [4] State three different methods for evaluating branch conditions. Explain the advantages and disadvantages of each.

conditions codes. condition is set implicitly when doing normal operations, e.g. arithmetic, so some explicit comparisons can be avoided. Introduces implicit dependences between instructions.

condition registers & compare instructions. Simple to implement, but tends to increase instruction count.

compare&branch instructions. Reduce the number of instructions, but difficult to implement and may impact cycle time.

MIPS uses compare&branch for only the simplest comparisons (EQ, <0, ≥0)

1j[4] What are the four basic addressing modes supported by the MIPS R3000 instruction set? State and give the RTL meaning of each. (Do not include the special cases that arise from setting one of the operands to zero.)

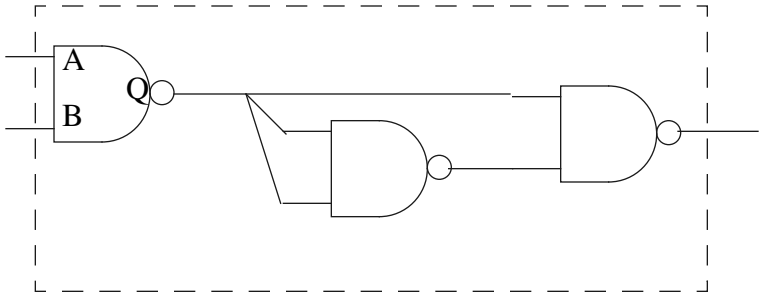
register-addressing: value is contained in a register specified in the instruction

base or displacement addressing : mem[R[rs] + sign_ext(Imm16)]

immediate addressing: value is contained in the instruction

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

1k[5]. Assume the NAND Gate has the following characteristics: Input load = 100fF, propagation delay low-to-high TPlh = 0.5ns, TPhl = 0.1ns, TPlhf=0.002 ns/fF, TPhlf=0.002ns/fF. Identify the critical path in the following cell and fully characterize it using the linear delay model.



Input load is 100fF on A and B

TPlhf=0.002 ns/fF, TPhlf=0.002ns/fF

Fixed internal delay is a little wierd because output ends up being high, after a glitch in one case. The important part was understanding the calculation

$TP_{cell} = (TP_{nand} + 3 * TPf_{nand}) + (TP_{nand} + 1* TP_{nand}f) + TP_{nand}$

This gives

TPlh = (0.5 + 0.6) + (0.1 + 0.2) + 0.5 = 1.9 (or 0, since no change on output)

TPhl = (0.1 + 0.6) + (0.5 + 0.2) + 0.1 = 1.5 (glitches, then settles)

1l[3] Give the definition of speedup due an enhancement.

Speedup with E = (Time without E) / (Time with E)

= (Performance with E) / (Performance without E)

Problem 2 (20 points).

This problem looks at performance and cost in the real world. The Feb 20, 1996 issue of PC Magazine provides the following data in its “Pentium or Pro?” cover story. $CPUMark_{32}$ and $CPUMark_{16}$ are indicators of performance (speed) similar to SPECMarks (bigger is better) for 32 and 16 bit programs.

	Pentium (P5)	**Pentium Pro (P6)**
Clock Rate	150 MHz	150 MHz
Transistor Count	3.3 M	5.5 M
Ave. System Price	$3,750	$7,800
$CPUMark_{32}$	273	430
$CPUMark_{16}$	276	270

2a. [3] Assuming CPUmarks are indicative of performance on real programs on these machines, how much faster is the Pentium Pro on 32 bit code? Show your work.

(a) 0.63

Speedup = Performance Pro / Performance P5= 430/273

(b) 0.98

(c) 1.02

(d) 1.58

2b. [3] Assuming CPUmarks are indicative of performance on real programs on these machines, how much faster is the Pentium Pro on 16bit code? Show your work.

(a) 0.63

Speedup = Performance Pro / Performance P5= 270/276

(b) 0.98

(c) 1.02

(d) 1.58

2c. [2] How much faster in performance per dollar is the Pro on 32 and 16 bit code?

32bit:(430/7800) / (270/3750) = 0.76 times faster

16bit:(270/7800) / (276/3750) = 0.47 times faster

Pretty sad, isn’t it.

2d. [5] Assuming that die area is proportional to the number of transistors for these designs, how much more expensive would you expect the Pro die to be? Show your work.

Hint: Here are some equations from the book that may help.

$$\text{dies/wafer} = \frac{(\Pi \times \text{wafer diameter}/2)^2}{\text{die area}} - \frac{\Pi \times \text{wafer diameter}}{\sqrt{2 \times \text{die area}}} - \text{test dies per wafer}$$

$$\text{die yield} = \text{wafer yield} \times \left(1 + \frac{\text{defects pr uinit area} \times \text{die area}}{\alpha}\right)^{-\alpha}$$

α is typically 2.

(a) 1.7

To first order, cost increases as the cube of the Area. Relative area is 1.7, so relative cost is expected to be 1.7^3

(b) 2.8

(c) 3.6

(b) 4.6

2e. [7] Using the answers to 2a and 2b, what fraction of the total workload would have to be 32-bit code in order for the Pro to perform 1.2 times faster than the Pentium?

Assume x is the percentage of the program is 32bit.

1.2 = P5 execution time / P6 execution time

$$1.2 = \frac{1}{\left(\frac{(1-x)}{0.98} + \frac{x}{1.58}\right)}$$

x = 0.483

so 48.3%.

Problem 3. (20 points) Complete the skeleton of MIPS assembly language (with delayed branches) below for the following C function. (Underlines may not be exactly right.)

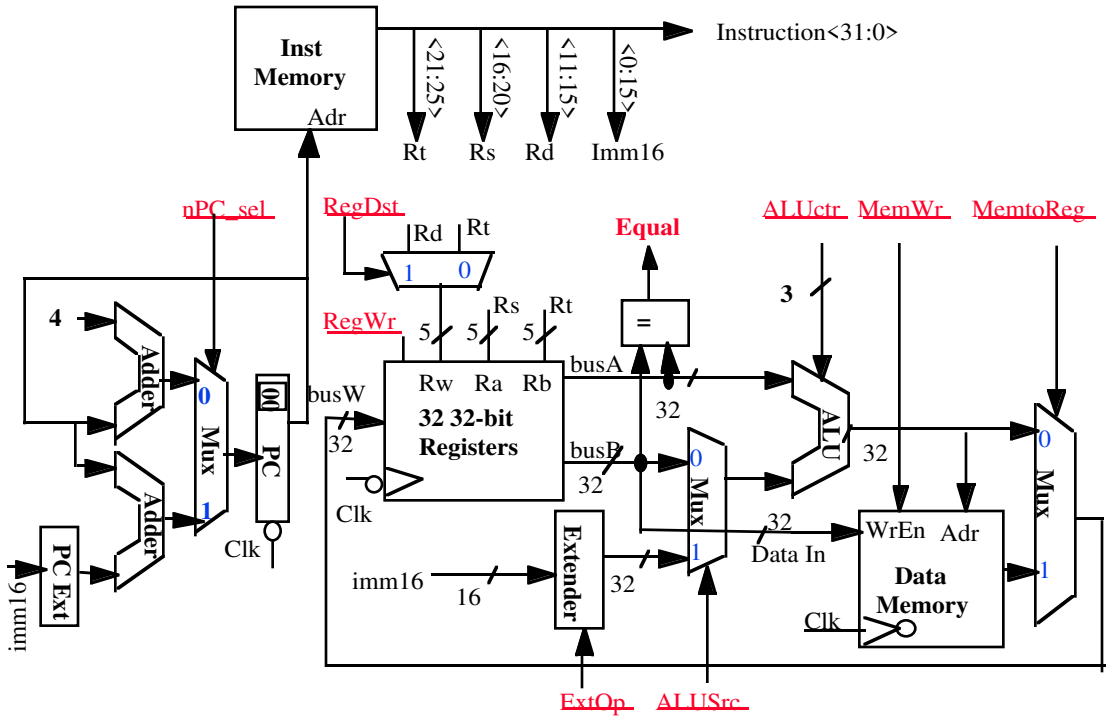
```
extern int f (int);

int foo(int *A, int n)
{
        int i = 0;
        int sum = 0;
        for (i = 0; i<n; i++) {
                sum = sum + f(A[i+1]);
        }
  return(sum);
}
```

```
foo:
      subu    $sp,$sp,40
       sw     $31,32($sp)
       sw     $19,28($sp)          # save A in $19
       sw     $18,24($sp)          # save n in $18
       sw     $17,20($sp)          # i in $17
       sw     $16,16($sp)          # sum in $16
      mov   $16, $0               ; sum = 0 (2 pts for initialization)
      mov   $17, $0               ; i = 0
      slt    $2, $17,$5           # i=0 < n (1 pt)
       beq   $2,$0,$L3            # delayed branch fall-through
      mov   $19, $4               # setup A (2 pts)
      mov   $18, $5               # setup n
$L7:                              # top of loop
      addi  $4, $17, 1            # i+1 (2 pts)
      sll   $4, $17, 2            # convert index to byte address (2 pts)
      addi  $4, $19, $4           # &A[i+1] (2 pts)
      lw    $4, 0($4)             # fetch A[i+1] into argument register (2 pts)
       jal    f                   # delayed jump and link
      addu  $16, $16, $2          # accumulate return value into sum (2 pts)
      slt   $2, $17, $18          # i < n (2 pts)
       bne _$2_,$0,_$L7_          # delayed branch to top of loop
      addi  $17, $17, 1           # i++
$L3:                              # fall-through
      mov   $2, $16               # return sum (1 pt)
      lw    $31, 32($sp)
      lw    $19, 28($fp)          (2 pts)
      lw     $18,24($sp)
      lw     $17,20($sp)
      lw     $16,16($sp)
       j      $31                 # delayed return jump
       addu    $sp,$sp,40
       .end    foo
```

Problem 4 (20 points):

The Single-Cycle processor developed in class below (which was very similar to the one



in the book) supports the following instructions. (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
LOAD	R[rt] <– MEM[R[rs] + sign_ext(Imm16)];	PC <– PC + 4
STORE	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, OR, AND, BLTZAL (branch on less than Zero and Link). On the following pages, write the register transfers for the new instructions. Sketch the modifications to the datapath and specify the control points for each of the new instructions.

Problem 4 (cont)

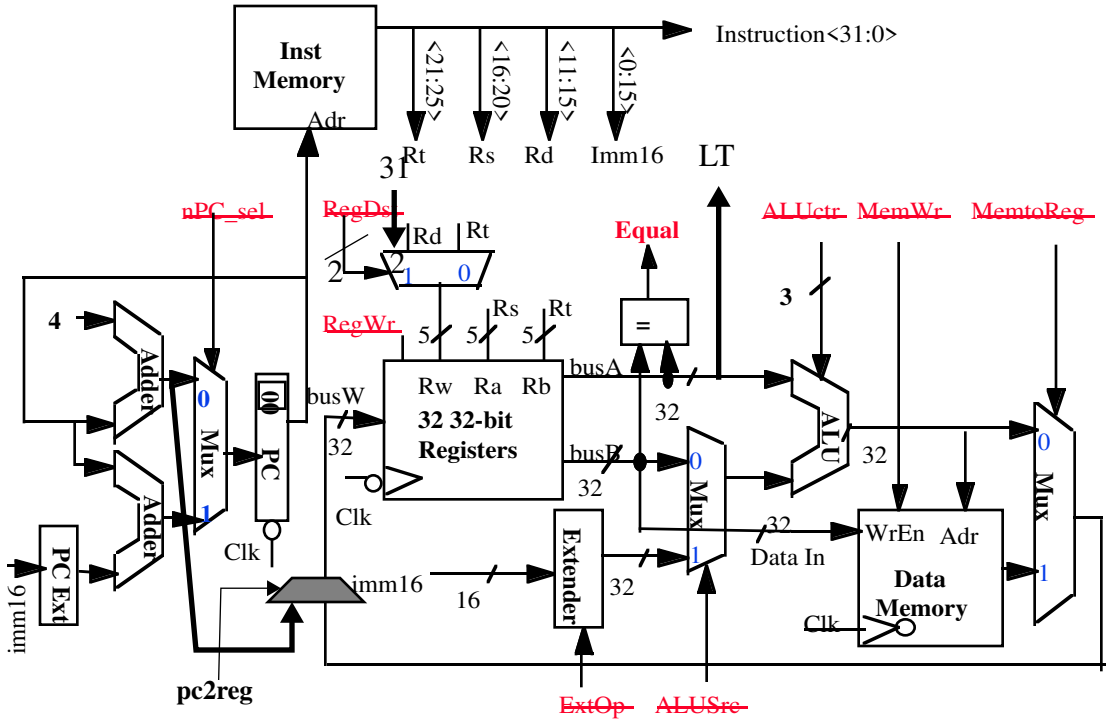
op | rs | rt | rd | shamt | funct = MEM[PC]
op | rs | rt | Imm16 =

inst	Register Transfers	
ADDIU	R[rd] <– R[rs] + SignExt(Imm16)	PC <– PC + 4
OR	R[rd] <– R[rs] or R[rt];	PC <– PC + 4
AND	R[rt] <– R[rs] and R[rt]	PC <– PC + 4
BLTZAL	R[31] <– PC+4;	
	if (R[rs] < 0) then PC <- PC + Sign_Ext(Im16)‖00 else PC <– PC + 4	

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 BLTZAL we need to

- detect R[rs] < 0, this is just the sign bit of bus_A
- 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
OR	x	0	OR	0	0	0	1	1	0
AND	x	0	AND	0	0	0	1	1	0
BLTZAL	x	x	x	x	x	1	2	1	LT

Problem 5 (20 points)

5a [7] Write a MIPS subroutine to perform a unsigned 64-bit subtraction. 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 registers v1:v0 with the same convention. Explain on the reverse side why your code works.

```
/* v1:v0 = a1:a0 -a3:a2 */

/* 4 instructions/cycles */

sltu t0, a0, a2

subu v0, a0, a2

subu v1, a1, a3

subu v1, v1, t0
```

5b.[13] In class (and in the book) we developed an unsigned multiplier that required 32 shift-and-add steps for a 32-bit multiply. We also developed a multi-bit shifter. Using adders, registers, and multiplexors, design a 32-bit multiply unit that skips over sequences of trailing zeros in the multiplier. Give the algorithm and the block diagram and explain how it works.

You could do this with any of the four multipliers that we discussed in class. You hang a piece of logic off the multiplier to determine the number of trailing zeros. (This is essentially like a carry-chain, but simpler.) If you used a barrel shifter the unary shift amount is exactly right, otherwise you need to do a unary-to-binary conversion. If the entire multiplier is zero (32 zeros) it is time to stop the algorithm. Otherwise, shift the multiplier right over the string of zeros and (logically) shift the product left by this amount. If you used the third version of the multiplier-design, this was just shifting the entire 64-bit product register right by the shift amount. Whenever there is a one in the lsb of the multiplier, do the usual add step.