Rose-Hulman Institute of Technology
Computer Science and Software Engineering
Prof. Laurence D. Merkle

Exam 1

Instructions:

- Write all answers on these pages. Use the back as necessary.
- Clearly indicate your final answer.
- For full credit, show your work, and document your code.
- Read the entire examination before starting, and then budget your time.

Authorized resources:

- Reference materials provided by the instructor at the time of the exam.
- Both sides of one 8½"x11" sheet of paper containing only handwritten material.

Unauthorized resources:

You are NOT permitted to use any resources other than those identified above. In particular, you may NOT use books, notes, electronic files, calculators, PDAs, or computers.

Good luck!

Problem 1	/15 points
Problem 2	/15 points
Problem 3	/18 points
Problem 4	/16 points
Problem 5	/16 points
Total	/80 points

Problem 1 – For each pseudoinstruction in the following table, give a minimal sequence of actual MIPS instructions to accomplish the same thing. You may need to use `$at` for some of the sequences. In the following table, `big` refers to a specific number that requires 32 bits to represent.

Pseudoinstruction	What it accomplishes	Actual instructions
`li $t5, big` [6 points]	`$t5 = big`	
`ble $t5, small, L` [6 points]	`if ($t5 <= small)` `  go to L`	
`not $t5, $t3` [3 points]	`$t5 =` `  bitwise-not($t3)`	

Problem 2 – The incomplete MIPS procedure on the following page computes the recursive function

$$bar(a,b,c) = \begin{cases} b\text{-}c, \text{ if } a = 0 \\ bar(a\text{-}1,b*b,c\text{-}a) * b * c, \text{ otherwise} \end{cases}$$

Finish the procedure on the next page by adding instructions for the following:

- the procedure entrance,
- the recursive procedure call, and
- the procedure exit.

Do not modify any of the given code, and be sure to follow the MIPS convention for register usage.

```
bar:
      # Procedure entrance
```

[6 points]

```
      # if (a == 0) return b-c
      bne  $t6, $zero, Else
      sub  $t0, $s2, $s3
      j    Exit

      # otherwise, return bar(a-1,b*b,c-a)*b*c
Else: sub  $t1, $t6, 1
      mul  $t2, $s2, $s2
      sub  $t3, $s3, $t6
      # Recursive procedure call
```

[6 points]

```
      mul  $t0, $t0, $s2
      mul  $t0, $t0, $s3

      #Procedure exit
Exit:
```

[6 points]

Problem 3a - [6 points] List the machine language fields and their DECIMAL values for these MIPS instructions:

```
addiu $t0, $s0, -1
```

```
mfhi  $t0
```

```
jal   Label # where the instruction is at address 0x7000 0018
            # and Label is at address 0x7000 0008
```

Problem 3b - [6 points] Give the MIPS assembly language statements represented by each of the following:

`0x024A 7824`

`0x0147 702B`

`0x0211 0019`

Problem 3c - [3 points] Assume that the MIPS instruction `bne $t0, $s0, Label` is located at address `0x0400 FFFC`, and that the value of the address field is `0x1111`. At what address is `Label` located? Express your answer in hexadecimal. *Hint*: remember that the offset is relative to the instruction following the branch, and that all branch targets must be word aligned.

Problem 3d - [3 points] Assume that the MIPS instruction `j Label` is located at address `0x0FFF FFFC`, and that `Label` is located at address `0x1444 4444`. What will the value of the target address field be? Express your answer in hexadecimal. *Hint*: remember that all branch targets must be word aligned.

Problem 4 - You are designing a processor as part of a larger system. Your architecture uses a special 8-bit register called the accumulator, another special 1-bit register called the status bit, 32 general purpose 8-bit registers, and the instruction set shown below.

Instruction	Function	Type	Opcode
`CondJump addr`	Sets the lower 6 bits of the program counter to the 6-bit immediate `addr` if the status bit is set.	J	
`EqualReg $r`	Sets the status bit to 1 if the contents of the accumulator are equal to those of register `$r`. Otherwise, sets the status bit to 0.	R	
`LoadMem $r`	Uses the contents of register `$r` as an address, and puts the contents of that memory location in the accumulator.	R	
`NandImm imm`	Puts the logical NAND of the accumulator and the zero-extended 4-bit signed integer immediate `imm` in the accumulator.	I	
`NorImm imm`	Puts the logical NOR of the accumulator and the zero-extended 4-bit signed integer immediate `imm` in the accumulator.	I	
`ShiftImm shamt`	Shifts the contents of the accumulator left the number of bits specified by the 4-bit signed integer immediate `shamt` (negative values indicate a shift right).	I	
`StoreMem $r`	Uses the contents of register `$r` as an address, and puts the contents of the accumulator in that memory location.	R	
`SwapReg $r`	Exchanges the contents of the accumulator with those of register `$r`.	R	
`XorImm imm`	Puts the logical XOR of the accumulator and the zero-extended 4-bit signed integer immediate `imm` in the accumulator.	I	

a) [12 points] As indicated in the table, you have assigned each instruction to one of three types based on their operands. You have also decided to use 8-bit instructions. Show the formats for each of the instruction types. Clearly indicate which bits are assigned to which fields. Hint: Design the formats in the order shown, and make the opcode fields as large as possible.

i. J-type:

ii. R-type:

iii. I-type:

b) [4 points] In the table above, specify opcodes for each of the instructions. Remember that every possible instruction must have a unique binary representation. You should also make it as simple as possible to distinguish between instruction types.

Problem 5 –In this problem, you will modify Patterson & Hennessy's multicycle implementation of MIPS (RTL description, datapath, and control unit) to support the new instruction *jump register on equal* (`jreq`).

The effect of the instruction `jreq $rs, $rt, $rd` is to conditionally branch to the address stored in `$rd` if the contents of `$rs` and `$rt` are equal.

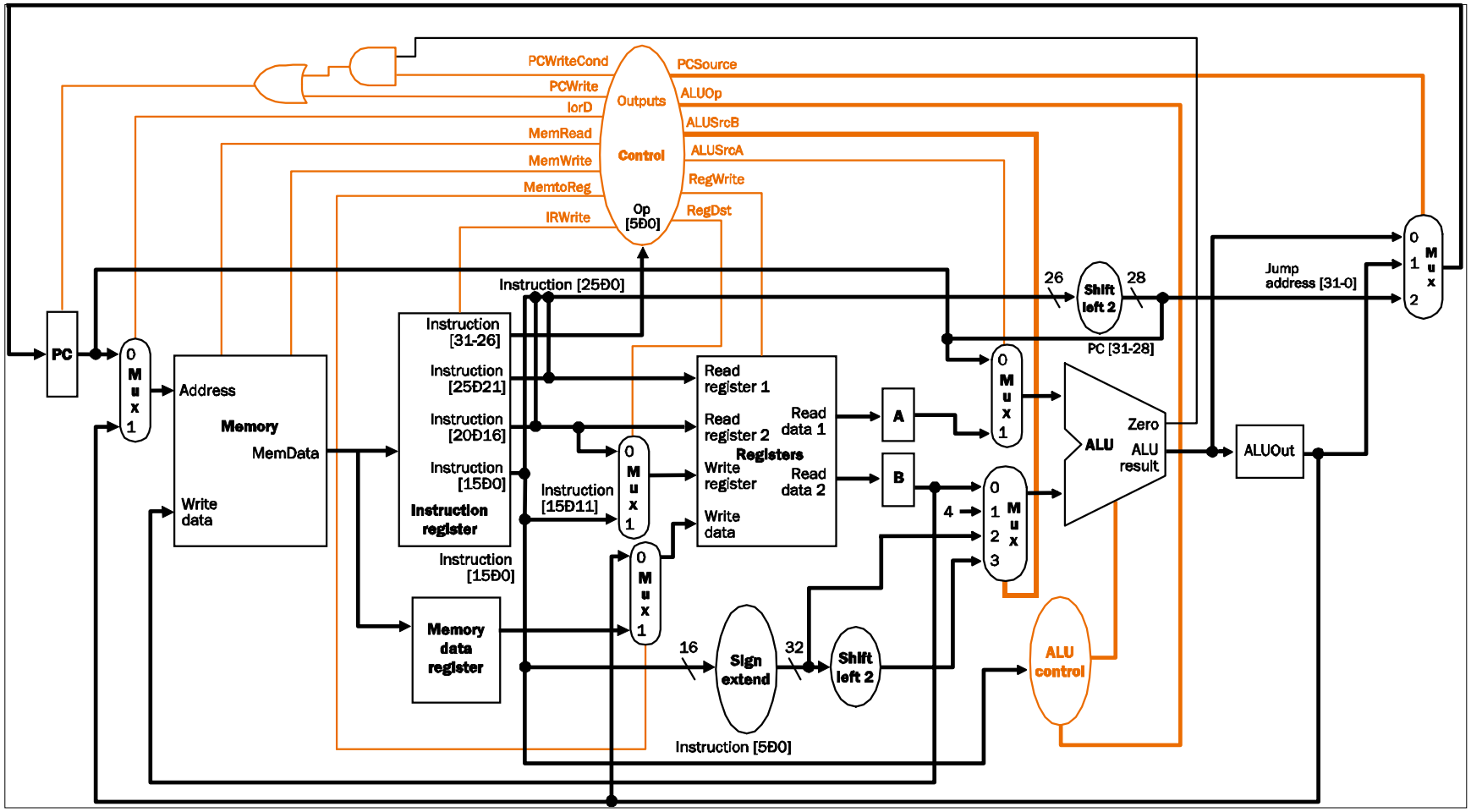
a) [4 points] Give an RTL description for the `jreq` instruction. Use as few cycles as possible without significantly extending the clock cycle. Patterson and Hennessy's RTL descriptions for the `beq` and `j` instructions are shown for reference. Do not modify the existing RTL. Describe any new or modified components required by your RTL (inputs, outputs, controls, and behavior).

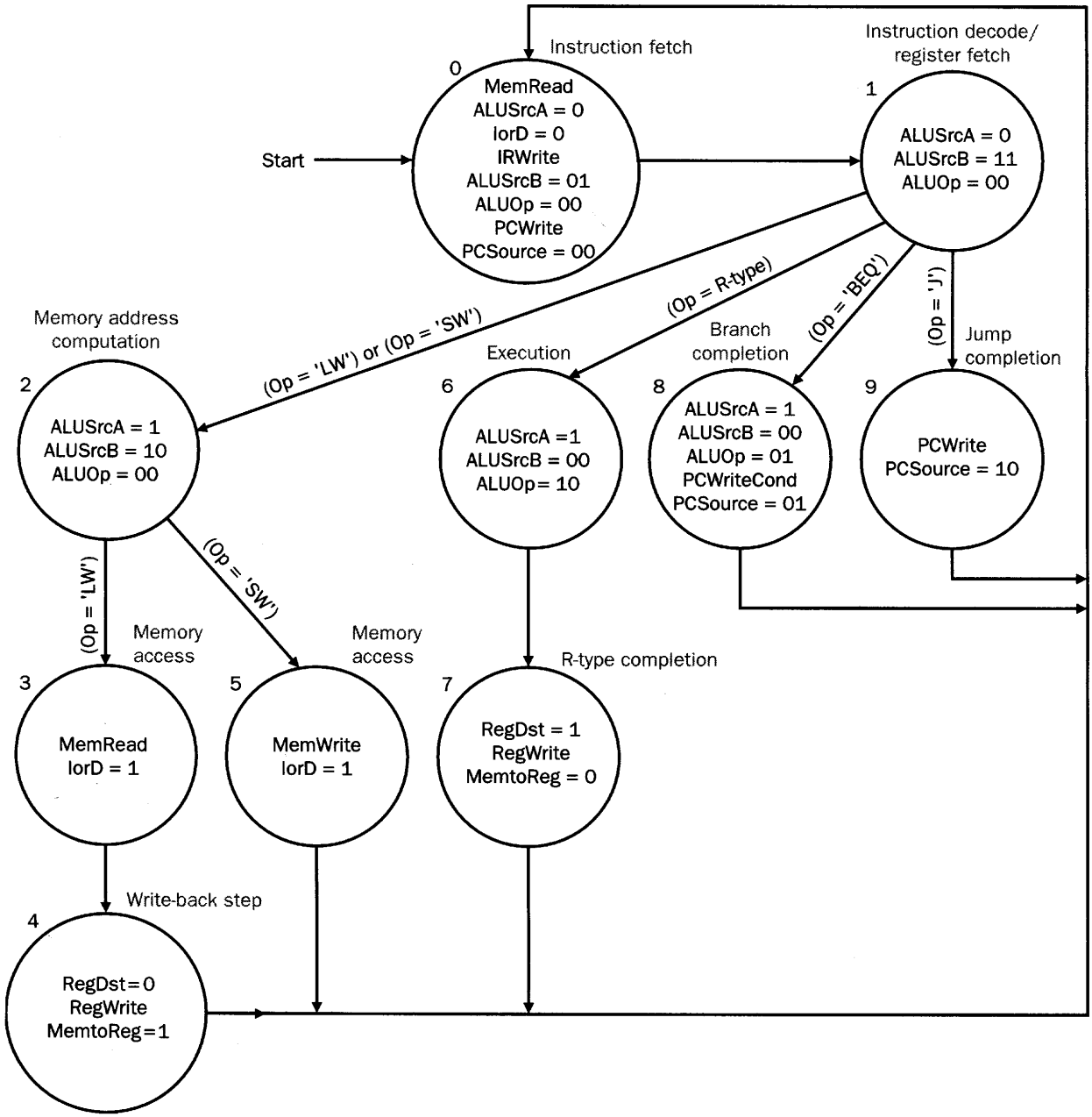
Action for branches	Action for jumps	Action for jreq
IR = Memory[PC] PC = PC + 4		
A = Reg [IR[25-21]] B = Reg [IR[20-16]] ALUOut = PC + (sign-extend (IR[15-0]) << 2)		
if (A == B) then PC = ALUOut	PC = PC [31-28] || (IR[25-0]<<2)	

.

b) [6 points] Patterson & Hennessy's multicycle datapath is shown on the next page. Neatly indicate the appropriate modifications necessary to support your RTL.

c) [6 points] Patterson & Hennessy's finite state machine is shown on the page following the datapath. Neatly modify the diagram to accommodate the changes made to the datapath, including new and modified control signals, as well as new states.


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op [5Đ0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
PC
0 M u x 1
Address
Memory
MemData
Write data
Instruction [31-26]
Instruction [25Đ21]
Instruction [20Đ16]
Instruction [15Đ0]
Instruction register
Instruction [15Đ0]
Instruction [15Đ11]
Memory data register
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
A
B
4
16
Sign extend
32
Shift left 2
ALU
Zero
ALU result
ALUOut
ALU control
Instruction [5Đ0]
Instruction [25Đ0]
26
Shift left 2
28
PC [31-28]
Jump address [31-0]
0 1 2 3 M u x
0 1 2 M u x


Start
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Instruction fetch
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
Instruction decode/
register fetch
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
Memory address
computation
6
ALUSrcA =1
ALUSrcB = 00
ALUOp= 10
Execution
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
Branch
completion
9
PCWrite
PCSource = 10
Jump
completion
(Op = 'LW')
(Op = 'SW')
3
MemRead
IorD = 1
Memory
access
5
MemWrite
IorD = 1
Memory
access
7
RegDst = 1
RegWrite
MemtoReg = 0
R-type completion
4
RegDst=0
RegWrite
MemtoReg=1
Write-back step