Rose-Hulman Institute of Technology
Computer Science and Software Engineering
Prof. Archana Chidanandan

Exam 1

Name:_____Solutions_____Section _____

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 Number	Maximum Points	Points Earned
1	12	
2	32	
3	16	
4	10	
5	14	
6	16	
Total	100	

Problems

Problem 1 – [12 points] 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.

Pseudoinstruction	Description	Actual instructions
`neg $t1, $t2`	Puts the negative of 32 bits of register $t2 into $t1.	sub reg1, $zero, reg2
rol $t1, value	Rotates(not simply shifts) the 32 bits of register $t1 left by `value` bits.	sll $at, reg1, value srl reg1, reg1, 32 - value or reg1, reg1, $at
`bge $t1, $t2, Label`	Branch to Label if $t1 ≥ $t2	slt $at, $t1, $t2 beq $at, $zero, Label

Problem 2a - [12 points] List the machine language fields and their **HEXADECIMAL** values for these MIPS instructions:

```
addi $t0, $t0, 11
8      8    8   0xB
001000 01000 01000 0x11
2   1    0    8    0xB => 0x210800b

jr $31              0x03e00008


sw   $ra, 0($t0)    0xad1f0000
```

Problem 2b - [12 points] Give the MIPS assembly language statements represented by each of the following:

```
0x0004 1080         sll $v0, $a0, 2

0x0810 000e         j 0x00400038

0x0149 5020         add  $t2, $t2, $t1// add $10, $10, $9
```

Problem 2c - [4 points] Assume that the MIPS instruction

```
     bne  $t0, $s0, Label
```

is located at address `0x0740 0560`, and that the 16-bit immediate field has the value

```
0000 0010 1001 1100
```

What is the address of `Label`? 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.

1. Add 4 to current address to get value in PC: (0x740 0560 + 4 = 0x0740 0564).
2. Convert # of instructions to # of addresses(bytes) i.e. left-shift by 2:
   00 0000 1010 0111 0000 => 0x0 0A70
3. Add the number of addresses to address in PC
   0x0740 0564 + 0x0000 0A70 = 0x0740 0FD4

Problem 2d - [4 points] Assume that the MIPS instruction `j Label` is located at address `0xF000 0004`, and `Label` is located at 0xF111 1114. What is the value of the 26-bit address field? Express your answer in decimal.

~~1111~~ 0001 0001 0001 0001 0001 0001 01~~00~~

Problem 3 – [16 points] A processor has 8 general purpose registers, 16 unique instructions, uses condition codes for branching and has 16-bit fixed length instructions. It it a byte-addressable memory and has a 16-bit address bus. For the following questions, mark any unused fields as unused and clearly indicate which bits are used for which field.

a. Draw the format for the R-type instruction such as `add,` where there are two source registers and one destination register.

Opcode	Rs1	Rs2	Rd	Unused bits
4 bits	3 bits	3 bits	3 bits	3 bits

b. Draw a format for the B-type instruction such as `beq,` which does not use any register operands, instead uses 3 bits for 3 condition code flags.

Opcode	CC1	CC2	CC3	immediate/address
4 bits	1 bit	1 bit	1 bit	9 bits

c. In part b, the immediate field is less than 16 bits, implying that all the addresses in memory cannot be accessed. Determine how many instructions(not addresses) can be accessed if only the value in those bits are used for addressing.

9 bits => 2^9 addresses
byte-addressable memory => 2^9 bytes
2 bytes = 1 instruction
=> 2^9 bytes = x instructions => x = $2^9/2 = 2^8$ instructions

d. Suggest an addressing mode for the immediate field in the B-type instruction, that could increase the range of instruction addresses that could be accessed.

Problem 4 – [10 points] For the following high-level language statement, obtain the equivalent assembly language instructions

a. on a stack-based ISA

OR

b. on an accumulator-based ISA

F = A*(C+D)

Assume that A, C and D are values in memory and Mem[A] will access the value A in memory. Also, F must be written back to memory.
You must do either one, not both. If you do both, the only the first one will be graded.

Stack-based: (No general purpose registers. The stack and memory are the only possible operands. Hence the top of the stack is an implicit operand in all the instructions.)

```
PUSH    Mem[C]
PUSH    Mem[D]
ADD
PUSH    Mem[A]
MUL
POP     Mem[F]
```

Accumulator-based: (1 general purpose register, called the Accumulator. Since it is always one of the operands, it is also implicit in the instruction.)

```
LOAD    Mem[C]    # ACC = C
ADD     Mem[D]    # ACC = ACC + D
MUL     Mem[A]    # ACC = ACC * A
STORE   Mem[F]    # Write contents of ACC to Mem[E]
```

Problem 5 – [14 points] A MIPS program calculates the sum of the elements in an array. If the sum is greater than the value of the last element of the array, then it sets the sum to zero. It then writes the final value of the sum to memory. The equivalent JAVA code fragment and the MIPS program are provided below. Fragments of the MIPS program are missing. You must provide the missing fragments in the space provided.

The JAVA code fragment:

```
sum = 0;
count = 0;
while (count  < N ) {
      sum = sum + A[i];
      count = count + 1;
}

if( sum > A[N - 1]) {
      sum = 0;
}
```

The MIPS program:

```
      .text
main:
      la    $t0, N            # Read the size of the array
      lw    $t0, 0($t0)
      la    $t1, A            # Read the address of the array
      li    $t2, 0            # Initialize sum to 0
      li    $t3, 0            # Initialize count to 0

loop:
      beq   $t3, $t0, done

      # Read the element from the array into $t5
      # Add the element to sum
      # Increment count
```

```
   sll   $t4, $t3, 2
   add   $t4, $t1, $t4
   lw    $t5, 0($t4)

   add   $t2, $t5, $t2     # Add the element to sum
   addi  $t3, $t3, 1       # Increment count by 1
```

```
      j     loop
```

```
done:
      beq   $t3, $0, quit     # If count = 0, quit

      # Check if sum is greater than last element of array
      # If yes, then set sum to 0 and then go to "save"
      # else go directly  to "save"
```

```
slt   $t6, $t5, $t2     # if (sum > A[N-1]) then $t6 = 1 else $t6 = 0
beq   $t6, $zero, save
li    $t2, 0
```

```
save:
      # Write the value of sum to memory
      la    $t0, Sum
      sw    $t2, 0($t0)
quit:
      # Prepare to quit the program
      li    $v0, 10
      syscall

      .data
N:    .word 5
A:    .word -1, 9, 3, -20, 10
Sum:  .word 0
```

Problem 5 –In this problem, you will modify Patterson & Hennessy's multicycle implementation of MIPS (RTL description and datapath) to support the new instruction `swap`.

The effect of the instruction `swap rs, rt` is to move the contents of register `rs` to `rt` and move the contents of register `rt` to `rs`. For example, if

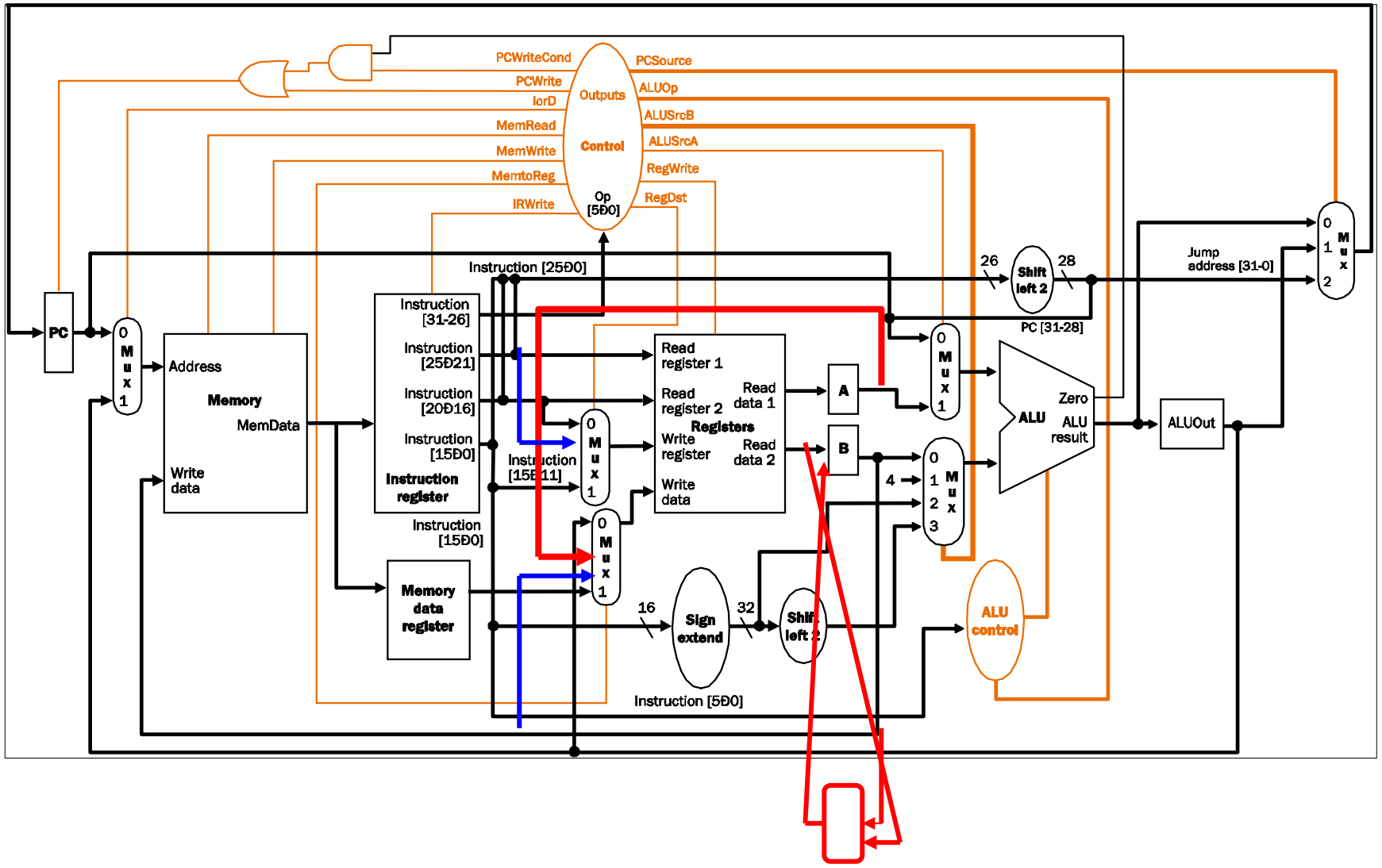
- register `$1` contains `0x0000 0030`,
- register `$2` contains `0x0000 0400`

then the instruction `swap $1, $2` puts `0x0000 0030` in register `$2` and `0x0000 0400` in register `$1`.

a) [8 points] Using the table on the next page, give an RTL description for the `swap` instruction. Use as few cycles as possible without significantly extending the clock cycle. Patterson and Hennessy's RTL descriptions for the `add` and `lw` instructions are shown for reference. Do not modify the existing RTL.

b) [8 points] Patterson & Hennessy's multicycle datapath is shown on page 10. Neatly indicate the appropriate modifications necessary to support your RTL.

Action for `add`	Action for `swap`
IR = Memory[PC] PC = PC + 4	
A = Reg [IR[25-21]] B = Reg [IR[20-16]] ALUOut = PC + (sign-extend (IR[15-0]) << 2)	
ALUOut = A + B	Reg[IR[20:16]] = A B = B
Reg[IR[15-11]] = ALUOut	Reg[IR[25:21]] = B


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]
Memory data register
Instruction [25Đ0]
Instruction [15Đ11]
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
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