Final Exam - 1 EECE 344L Spring 2014

Name: Steven Seppälä

Problem	Possible	Score
1	10	10
2	15	13
3	25	25
4	25	19
5	20	18
6	20	17
7	20	18
8	15	12
Total	150	132

General information that may be useful sometime during test:

Table of Powers of Two

N	2^N	N	2^N	N	2^N	N	2^N
0	1	8	256	16	65,536	24	16,777,216
1	2	9	512	17	131,072	25	33,554,432
2	4	10	1,024	18	262,144	26	67,108,864
3	8	11	2,048	19	524,288	27	134,217,728
4	16	12	4,096	20	1,048,576	28	268,435,456
5	32	13	8,192	21	2,097,152	29	536,870,912
6	64	14	16,384	22	4,194,304	30	1,073,741,824
7	128	15	32,768	23	8,388,608	31	2,147,483,648

Caution: Make sure you write *very* legibly.

1. Information representation question: (I warned you there was gonna be one of these!) Consider a weird system that uses **9 bits**, with the radix point four bits from right. (That is, p=4.) So, the representation has 5 whole number bits and four fractional bits, like this: xxxxx.xxxx
Fill in the missing elements of the following table.

Number	Bit Pattern	Value of Representation
Value of this bit pattern; two's complement	10111.0110	$-8\ 5/8$
Value of this bit pattern; unsigned binary	10111.0110	$23\ 3/8$
Generate bit pattern for: two's complement	10010.1011	$-13\ 5/16$
Value of this bit pattern: bit two's complement	10101.0101	$-10\ 11/16$
Value of this bit pattern; unsigned binary	10110.1010	$22\ 5/8$

A) 10111.0110 => 01000.1010 => $2^3\ 5/8 = -8\ 5/8$
3 2 1 0 . 1/2 1/4 1/8

B) 10111.0110 => $2^4+2^2+2^1+2^0\ 3/8 = 16+4+2+1 = 23\ 3/8$
4 3 2 1 0 . 1/2 1/4 1/8

C) 01101.0101 ⇝ 2's ⇝ 10010.1011
13 = 8+4+1
3 − 8 = −5
2 − 4 = −2
0 − 1 = −1
1+4+8 = 13
4/16 + 1/16 = 5/16

D) 10101.0101 => 01010.1011 => $2^3+2^1\{11/16\} = -10\ 11/16$
3 2 1 0 . 8/16 4/16 2/16 1/16

C) 10110.1010 => $2^4+2^2+2^1\{5/8\} = 16+4+2 = 22\ 5/8$
4 3 2 1 0 . 1/2 1/4 1/8
1/2 + 1/8 = 4/8 + 1/8

2. General information question:

a) What is the basic fundamental principle on which all stored program computers are based?

Fetch → Decode → Execute

b) Okay, here's one of those "what's different" questions. The word location 0xA0001000 contains the value 0x3456789A, Also $t0 contains the address 0xA0000000. What is the difference in the values delivered to $t1 and $t2 with the two instructions 'lb $t1,0x1000($t0)' and 'lbu $t2, 0x1000($t0)'

lb → Sign extend ⇒ $t1 ⇐ 0xFFFFFF9A

lbu → zero extend ⇒ $t2 ⇐ 0x0000009A

9A ⇒ 10011010 (sign extend)

c) Why have we been adding a 'nop' instruction after each branch instruction?

Branches take up 2 instructions (-2)

So to effectively use them and give time to calculate branch address we need a nop delay

d) Address 0xBF886010 is used to write directly to PORTA. What happens at address 0xBF88601C?

C	S	I
4	8	C

@ 0x0C of PORTA will invert the selected bits that are written into it

e) List the four instruction types and give an example of each from the MIPS instruction set.

work	movement	program control	system control
add	lw	beq	Syscall

f) TheMIPS architecture calls for a coprocessor used to manage interrupts, handle status, and related responsibilities. What instructions are used to move information to and from the coprocessor?

mfc0 → move from co-processor

mtc0 → move to co-prosecssor

g) True or False: An ASCII character is specified as one of 256 different bit patterns?

0xFF

2^8 = 11111111

False

h) True or False: The following is a MIPS instruction: BEQI – Branch to target if Rs = IMM

False

3. Instruction execution question 1: The following routine has created by a user for some strange manipulations on data. Hence, there appears to the observer some un-explainable operations (in other words, don't try to make sense out of what this routine does…) The routine "ABC3" is called in the code below, and the purpose of this question is to determine the final value of the registers *after* the code fragment executes. When the program activity is at the location indicated "before", a snapshot of the registers is given in the "Before" area of the table. Please indicate in the "After" area of the table the register contents when "after" is reached after execution of the subroutine (remember to enter values only for those registers that have changed). *Note that only half of the MIPS registers are shown; the instructions do not touch the other registers.*

```
    Addr      Bits        Instruction
              before:
9D0000EC  00000000          nop
9D0000F0  04110006          bal target
9D0000F4  00000000          nop
9D0000F8  00000000          nop
              after:

9D000108  00000000          nop
              target:
9D00010C  3C0C2222          lui $t4,0x2222
9D000110  358C1111          ori $t4,$t4,0x1111
9D000114  3C0E0000          lui $t6,0
9D000118  35CE000F          ori $t6,$t6,15
9D00011C  3C0F0000          lui $t7,0
9D000120  35EF0001          ori $t7,$t7,1
              again:
9D000124  01EF7821          addu $t7,$t7,$t7
9D000128  25EF0001          addiu $t7,$t7,1
9D00012C  25CEFFFF          addiu $t6,$t6,-1
9D000130  1DC0FFFC          bgtz $t6,again
9D000134  314C0F0F          andi $t4,$t2,0x0F0F
9D000138  01175825          or $t3,$t0,$s7
9D00013C  03E00008          jr $ra
9D000140  00000000          nop
```

Before		After	
S0 = 0x00000000	T0 = 0x11111111	S0 =	T0 =
S1 = 0x22222222	T1 = 0x33333333	S1 =	T1 =
S2 = 0x44444444	T2 = 0x55555555	S2 =	T2 =
S3 = 0x66666666	T3 = 0x77777777	S3 =	T3 = 0xFFFFFFFF
S4 = 0x88888888	T4 = 0x99999999	S4 =	T4 = 0x505
S5 = 0xAAAAAAAA	T5 = 0xBBBBBBBB	S5 =	T5 =
S6 = 0xCCCCCCCC	T6 = 0xDDDDDDDD	S6 =	T6 = 0x00000000
S7 = 0xEEEEEEEE	T7 = 0xFFFFFFFF	S7 =	T7 = 65535
	RA = 0x00000000		RA = 0x9D0000F8

4. Instruction execution question 2: Consider the code segment below. As in the previous question, this question basically is to determine the work done by the instructions, and to modify the registers and the memory segment shown below to reflect the instructions shown. And, once again, don't try to attach any meaning to the work done. And, once again, only half the registers are shown. And yes, this format was borrowed from the first exam.

```
                        start:
9D00034C  3C0F0000          lui $t7,0
9D000350  35EF1000          ori $t7,$t7,0x1000
9D000354  91E80017          lbu $t0,0x17($t7)
9D000358  026B4821          addu $t1,$s3,$t3
9D00035C  02B25025          or $t2,$s5,$s2
9D000360  8DEB001C          lw $t3,0x1C($t7)
9D000364  ADF70028          sw $s7,0x28($t7)
9D000368  A1F10032          sb $s1,0x32($t7)
9D00036C  95EE000A          lhu $t6,0x0A($t7)
9D000370  02566027          nor $t4,$s2,$s6
9D000374  A5F60022          sh $s6,0x22($t7)
                        end:
```

Before		After	
S0 = 0x00000000	T0 = 0x11111111	S0 =	T0 = 000000FF
S1 = 0x22222222	T1 = 0x33333333	S1 =	T1 = CCCCCCCC
S2 = 0x44444444	T2 = 0x55555555	S2 =	T2 = EEEEEEEE
S3 = 0x66666666	T3 = 0x77777777	S3 =	T3 = 19181716
S4 = 0x88888888	T4 = 0x99999999	S4 =	T4 = 33333333
S5 = 0xAAAAAAAA	T5 = 0xBBBBBBBB	S5 =	T5 =
S6 = 0xCCCCCCCC	T6 = 0xDDDDDDDD	S6 =	T6 = 00003322
S7 = 0xEEEEEEEE	T7 = 0xFFFFFFFF	S7 =	T7 =

Addr	3	2	1	0	Addr	3	2	1	0
1000	56	45	23	01	1020	DD	AD	0025	
1004	EF	CD	AB	89	1024				
1008	33	22	11	00	1028	EE	EE	EE	EE
100C	77	66	55	44	102C				
1010	BB	AA	99	88	1030		22		
1014	FF	EE	DD	CC	1034				
1018	15	14	13	12	1038				
101C	19	18	17	16	103C				

5. Interrupt question #1: Let's make this real simple. In the space provided below, give code for initializing the UART to operate with receive interrupt enabled, but not transmit interrupt. Also, assume baud rate generator value is 50. Please identify the registers needed and in the comments state the purpose of the register.

```
.global main
.text
.data
.set noreorder
.ent main
main: # assume regesters are all zero'd @ this point
    nop
    li $s0, 0xBF806000 # The address of UART mode
    li $t0, 0x8300 # the value to turn mode on & enable UART Tx & Rx
    sw $t0, 0x00($s0) # put them into the memory mapped I/O reg
    li $t0, 0x1440 # Value to enable Rx interupt on keypress?
    sw $t0, 0x10($s0) # put value into U2STATE Regester
    li $t0, 50
    sw $t0, 0x40($s0) # put 50 into Baud rate SFR
    li $s1, 0xBF881000 # INTCON Register set for interupt controls
    li $t1, 0x1000 # Value for multivector mode
    sw $t1, 0($s1) # turn on multivector mode
    li $t1, 0x200 # Value to enable Rx interupt — U2RXIE
    sw $t1, 0x70($s1) # IEC(1) will enable interupt to be handled
    li $t1, 0x1F # give highest priority to UART interupt
    sw $t1, 0xF8($s1) # put priority level into SFR to enable the level
    nop # and set only this priority
    li $t0 0x9FC01000 # Value of Vector table
    mtc0 $t0, $15, 1 # put Vector table into e-base
    ei # enable interupts — IFS0
    steady_state # done, waiting.
    nop
end main
```

6. Interrupt question #2. This is also a simple question. Give code for the interrupt service routine to handle the UART of question 5. That is, when the interrupt is detected, your ISR should handle the flags appropriately and place the received character in the mailbox at 0xA0000120. When the character shows up in the mailbox, it is the responsibility of some other routine to do something with it.

```
.global ISR
.text
.data
.set mailbox, 0xA0000120  # set place of mail box
.set noreorder
.ent ISR
ISR: # assume zero'd Registers
    nop
    li $s1, 0xBF88 1000  # INT con Reg set
    li $t0, 0x200 # U2 Rx flag interrupt
    sw $t0, 0x04($s1) # clear only U2Rx flag
    li $s0, 0xBF80 6000

    lw $t0, 0x30($s0) # load word from Rx Register
    sw $t0, mailbox # put char in mail box, yes, this is correct
    nop                 # Syntax (Using it in LAB 4)
    eret  # Return from interrupt
    nop   # eret need; nop
.end ISR
```

CSI

Why bother INTCON? not needed

1

2

Syntax

$s0$ 0 - max 65373
$s1$ qqqqqqq - min max i-type

7. The data structure question. Assume there is an array that starts at location 0xA0004000 that contains 2500 values, 32 bits each. These values are all less than 1,000,000. Give code that will look through the values stored in the array and determine the maximum and minimum values. Place the maximum value in memory at location 0xA0004040, and the minimum at location 0xA0004044.

```
.global main
.text
.data
.set num, 0xA0004000
.set max, 0xA0004040
.set min, 0xA0004044
.set noreorder
.ent main
main: # assume zero'd registers
    nop
    li $s4, 2502
    lw $s0, num      # load 1st value, correct syntax (using same in lab4)
    lw $s1, num+0x04 # load next val, still correct syntax
    la $s3, num+0x08 # get address of rest of nums (correct syntax)
loop: sub $s4, $s4, 1 # decriment
    lw $t0, 0($s3) # get val
    slt $t1, $t0, $s0 # s0 < t0? 1:0
    addi $s3, $s3, 0x04 # get next word
    beq $t1, 0, greater
    nop
    slt $t2, $s1, $t0 # s1 < t0? 1:0
    beq $t2, 0, lesser
    nop
    bne $s4, $zero, loop
    nop
    j compare
    nop
greater: nop
    move $s0, $t0 # if t0 > s0, put t0 into s0, New max
    b loop # go back
    nop
lesser: nop
    move $s1, $t0 # if t0 < s1, put t0 into s1, New min
    b loop
    nop
```

```
compare: nop
    slt $a0, $s0, $s1
    beq $a0, 1, swap
    nop # if $s0 is min val, need to
    nop # swap s1 & s0
    j end
    nop
swap: move $t0, $s0
    move $t1, $s1
    move $s0, $t1
    move $s1, $t0
    # swaps s0 & s1
end: sw $s0, max
    # correct syntax to store
    # max val into mem
    sw $s1, min
    nop
.end main
```

Val
Val2
Val3 — 1+ 1? 1+2?
Val4

s0 <= max
s1 <= min

50000
2 | 100000

8. And finally, a real easy I/O problem. Provide code for initializing the 8 LED module (Port E) and providing a rotating 1's pattern: a single LED on that rotates in steady state, with a count of 100,000 instruction times between rotates.

```
.global main
.data
.text
.set porte, 0xBF886100 # base for Port E set
.set noreorder
.ent main
main: # assume all Registers zeroed
    sw $t0, porte # This will clear Trise, correct syntax
    sw $t0, porte+0x30 # set ODC to digital out, this syntax works.
    sw $t0, porte+0x10 # clear the port
    li $s0, 128 # max rotate val
    li $s1, 1 # reset val
    li $s3, 50 000 # count val
    move $t0, $s3 # copy to to s3
loop: subu $t0, $t0, 1 # these two instructions will execute
    bltz $t0, zero, loop # 50,000 times each, thus giving us
    nop # 100,000 instruction times
    move $t0, $s3 # reset the count
    sw $a0, porte+0x10 # put pattern out, correct val, syntax works
    beq $a0, 128, reset
    nop
    sll $a0, 0x01 # shift right by 1 (multiply by 2 to get next light)
    b loop # keep looping when this is hit
    nop
reset: nop
    move $a0, $s1 # reset the rotate pattern
    b loop # go back to loop
    nop
end main
```

This some new syntax you worked out with PIC

move $a0, $s1 # initalize rotate pattern