35/56 35.0/56

COE608: Computer Organization and Architecture

Midterm-Exam, Winter 2017

Student Name: **Student #:**

i) Total time allowed is 80 minutes. **Total Marks: 56**

ii) This is a Close Book and Note Exam. A MIPS CPU data sheet is allowed to be used in the exam.

iii) This Exam has **5** sheets and **5** questions. Answer all the questions.

iv) Estimated time for each question is equivalent to the marks assigned to it.

iv) All the questions are not of equal difficulty and marks. Read the questions carefully.

Q1. Consider a 16-bit RISC type embedded CPU with the following instruction formats. Opcode occupies either bit 11-15 for R and LS type instructions or bit 13-15 for J type instructions. Source, destination or other register identifications use either bit 8-10, bit 5-7 or bit 2-4 as given below.

i) R-type: Register-to-register Arithmetic or Logical type instructions

Op Code	Source Reg. 1	Destination Reg.	Source Reg. 2 or Immediate Data (5-bit)
15 11	10 8	7 5	4 0

ii) LS-type: Load-store type instructions

Op Code	Source/Dest. Reg.	Base Addr. Reg.	Address offset
15 11	10 8	7 5	4 0

iii) J-type: Jump type instructions

Op Code	Jump Address
15 13	12 0

MARKS: 9 (3+2+2+2)

Write a short answer in the space provided (1-2 lines) for the following questions. Justify your answers.

(a) Maximum number of operations that can be executed by the CPU (i.e. All R, LS and J-type instructions).

R-type = $2^6 = 64$
LS-type = $2^6 = 64$
J-type = $2^3 = 8$
136

The maximum number of operations is all possible combinations of op-codes for R, LS & J type instructions which equals 136.

(b) Number of registers available in the above 16-bit RISC CPU.

1) Source Reg.1
2) Destination Reg.
3) Source/Des Reg.
4) Base Adr. Reg.
5) Source Reg. 2

5

(c) Assuming that page/frame memory addressing is not allowed, determine the maximum size of the main Memory in bytes.

4 bytes

(d) Maximum size of a constant that can be used as an operand in arithmetic and logical instructions.

R-type max operations = $2^6 = 64$

max value is $64-1 = 63_{10} \Rightarrow 11111_2$

12

Q2. If mutiply instruction of a CPU takes 12 cycles and account for 20% of the instructions in the main application program while the other 80% of the instructions require an average of 2 cycles per instruction.

MARKS: (4+8)

(a) What percentage of time does the CPU spend doing multiplications?

$$\frac{\text{Multi}}{\text{Total}} = \frac{0.2 \times 12 \text{ cycles}}{0.2 \times 12 \text{ cycles} + 0.8 \times 2 \text{ cycles}} = \frac{2.4}{4} = 60\%$$

Multi	20%	12 cycles
other	80%	2 cycles

✓ 4

(b) The hardware engineering team is suggesting the following two options of modifying the CPU-hardware to reduce the number of cycles required for multiplication.

(i) The first option reduces the number of cycles required for multiply to 5 but this will require a 25% increase in the clock cycle time.

(ii) The 2nd option reduces the number of cycles required for multiply to 8 but it requires a 10% increase in the clock cycle time.

Nothing else will be affected. Which of the modification is better in reducing the overall execution time? Justify your answer.

i) Multi 20% 5 cycles
other 80% 2 cycles

$$\text{New cycle time}_1 = (0.2 \times 5 + 0.8 \times 2) \times 1.25$$
$$= 3.25 \text{ cycles}$$

✓ 3

ii) Multi 20% 8 cycles
other 80% 2 cycles

$$\text{New cycle time}_2 = [(0.2 \times 8) + (0.8 \times 2)] \times 1.1$$
$$= 3.52 \text{ cycles}$$

✓ 3

The 1st option (2) to reduce the multiplication cycle to 5 and an overall cost of 25% would be the best option.

12

Q3. (a) Consider the following FOR-loop code sequence in a C-like language:

```
i = 0;
for (j = 1; j < 10; j++) {
        i = i + 3
}
...
```

MARKS: 10

The MACHINE-level code for this loop is written below. THERE ARE MULTIPLE ERRORS IN THIS CODE SEQUENCE! You may modify an instruction or reorder instructions; but you should NOT rewrite the program from scratch.

Word Address	MIPS Machine Instruction	Comment	If instruction requires a change then indicate the fix in the box below
0	ADDI $t1, $t0, #0	Clear $t1	ADDI $t1, #0, #0; ✓
1	ADD $t2, $t1, 1	t2 ← t1 + 1	ADDI $t2, $t1, #1; ✓
2	ADD $s3, $t1, #10	s3 ← t1 + 10	ADDI $S3, $t1, #10; ✓
3	SUBI $s4, $t2, $s3	s4 ← t2 – s3	SUB $S4, $t2, $S3; ✓
4	BEQ $s4, #8	IF s4 == 0 THEN GOTO address 8	BEQ $S4, $8; ✗
5	ADDI $t1, $t1, #4	tl ← tl + 3	ADDI $t1, $t1, #3; ✓
6	J #2	GOTO address 2	J $2; ✗
7	ADD $t2, $t2, #1	t2 ← t2 + 1	ADDI $t2, $t2, #1; ✓
8	…	Loop exit	

8

5 ✗

(b) Determine IEEE754 floating-point single-precision representation of $(-13.6666)_{10}$. Show your complete work.

MARKS: 5

$13/2 =$ 6 R1
3 R0
1 R1
0 R1

$13_{10} = 1101_2 = 1 \times 2^3$

$127 + 3 = 130 \Rightarrow$ ✗

$0.6666 \times 2 =$ 1.3332 1
0.6664 0
1.3328 1
0.6656 0
1.3312 1
0.6624 0
1.3248 1
0.6496 0
1.2992 1
0.5984 0 …

Continous

4

1	1000 0010	1010 1010 1010 1010 1010 101

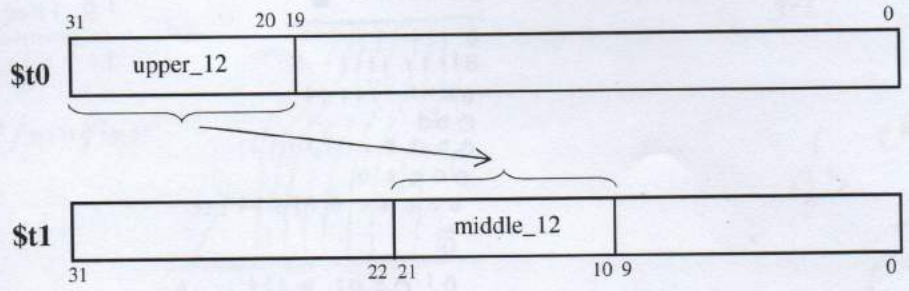
Sign exp binary points.

Not Continous, you'll have to find all the digits

6

Q4. (a) Write a shortest sequence of MIPS CPU code that can transfer the most significant 12 bits of a register $t0 in the middle of a register $t1 as given below. The remaining bits of register $t1 must be set to zero.

MARKS: 6




```
ADDI  $t1, #0, #$FFF0 0000 ; // $t1 = 1111 1111 1111 0000 0000 0000 0000 0000
AND   $t1, $t1, $t0 ;       // $t0= xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
SRL   $t1, $t1, #10 ;       // $t1 = xxxx xxxx xxxx 0000 0000 0000 0000 0000
                            // Right shift 10 -> 0000 0000 00xx xxxx xxxx xx00 0000 0000
```

AND

0	0
1	0
0	0
1	1

3.5

SRR $t1, $t0,20;//Shift reg$t0 all the way right, store result in$t1
SRL $t1, $t1,10;//Shift reg$t1 10 bits to the left, store result in $t1

(b) Write a VHDL entity for a 3-bit wide 2-4 decoder.

MARKS: 4

```
use ieee_library.all;
entity decoder_3 ( A in logic_vector(4 downto 2);
                   B out logic_vector(4 downto 2))

END decoder_3 ;
```

2.5

4

Q5. (a) Add $(-86)_{10}$ to $(-43)_{10}$ by using 8 bit 2's compliment notation. Does it produce an overflow condition?

MARKS: 4 (3+1)

$-86 = 1110\ 1010$

$+\ -43 = 1111\ 0101$

$-129 = 1\ 1101\ 1111$

↳ carry / overflow

↓ Yes

2's compliment?

1

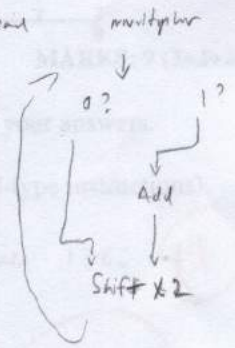
(b) Instead of using special multiplier hardware, it is possible to multiply by shift, add, etc. This is particularly attractive when multiplying by small constants. We want to put 15 times the value of $s0 into $s1. Write the shortest sequence of MIPS instructions for doing this without using a multiply instruction. Assume, there will be no overflow.

MARKS: 6

```
        lw   $t0 , #0 ; // t0 = 0
        cond;
Adder   addi $t0 , $t0 , #15;
        Shift
        ;

Shift   srl  $0 , 1 ;   // shift multiplier
        srl  $t0 , 1 ;  // Shift product

Cond:   AND  $t1 , $s0 , 1 ; // multiplier & 1 = 0000 000x
        beq  $t1 , Adder ;
        Shift ;
```



3