Indian Institute of Information Technology Allahabad

Review Test - First Component (C1) (February 2020)

Second semester B.Tech (IT): Section A and C

Course Name	Course Code	Date of Exam	MM	Time
Computer organization and Architecture	ICOA230C	Feb. 26, 2020	30	1 Hr

***Important Instructions**: All questions are compulsory. Answer all questions in the order as mentioned in the question paper.*

1. **(5 marks) (*Logic Design*)**:

   The control unit of a chemical process is required to control the temperature and the pressure inside a reactor by binary **(ON/OFF)** control of a *Heater* **(H)** and an inlet *Valve* **(V)** according to the following logic: *Heater is **ON** if the Temperature is **LOW** and the Pressure is not **HIGH**. Valve is **Open** if Pressure is **LOW** and Temperature is not **LOW**.* In addition, *the control unit has also to sound an Alarm **(A)** if the temperature and the pressure are either both **LOW** or both* HIGH.
   Assign two binary variables **TL** and **TH** to represent the three possible ranges – **LOW**, **NORMAL** and **HIGH** – of temperature and two binary variables **PL** and **PH** to represent the three possible ranges of pressure. Obtain Boolean expressions for the output variables **H, V and A** in terms of **TL, TH, PL and PH**.

   Solution:

PL/PH	0	1	TL/TH	0	1
0	Normal	Low	0	Normal	Low
1	High	High	1	High	Low

   H = TL * TH * $\overline{PH}$

   V = PL * $\overline{PH}$ * $\overline{TL}$

   A = PH * TH * $\overline{TL}$ + PL * $\overline{PH}$ * TL

2. **(5 marks) (*Number Representation and Arithmetic Circuits*)**:

   (a) *Nobita's* instructor for the *ICOA230C course* asks him to type 4 bits in a file. *Nobita* opens a text editor and types "0100" into a file. Thus, he believes he has written 4 bits to the file, and has created a binary file. However, *Doremon* explains to him that he has committed a mistake. What do you think was the mistake?

   Solution:

   ```
   When Nobita types characters in a text editor, it gets translated to the ASCII
   code. Thus '1', '0', '0', '0' are written in the file as 0x31, 0x30, 0x30,
   0x30  (or more properly, 0011 0001 0011 0000 0011 0000 0011 0000).
   Thus, Nobita has written 4 bytes into the file, not 4 bits.
   ```

   (b) In a normal *n-bit* ripple carry adder, to find out if an overflow has occured we make use of ———

   Solution:

   ```
   Xor gate
   ```

   (c) Booth's algorithm for integer multiplication gives worst performance when the multiplier pattern is

   ```
    (i) 101010.....1010  (ii) 100000......0001
   (iii) 111111.....1111 (iv) 011111......1110
   ```

   Solution:

   ```
     (i)
   The worst case of an implementation using Booth's algorithm
   is when pairs of 01s or 10s occur very frequently in the multiplier.
   ```

   (d) Assume that for a full adder implementation, the delay for sum and carry generation are 6 nsec and 4 nsec respectively.The worst-case delay of a 16-bit ripple carry adder will be:

(i) 96 nsec (ii) 64 nsec
(iii) 66 nsec (iv) None of the above

Solution:

Correct answer is (iii).
The carry will ripple through the 16 stages, and so the carry propagation time = 16 x 4 = 64 nsec. In the last adder, generation of sum will require 2 nsec more that generation of carry. So the worst-case delay will be 64 + 2 = 66 nsec.

(e) For a full adder stage F in a carry look-ahead adder, which of the following statements are false?

(i) The carry generate function Gi will be 1 if both the inputs Ai and Bi are 1.
(ii)The carry propagate function Pi will be 1 if at least one of the inputs Ai and Bi are 1.
(iii) The carry generate function Gi will be 1 if exactly one of the inputs Ai and Bi are 1.
(iv) None of the above

Solution:

Correct answers are (ii) and (iii)
The carry generate function is defined as Gi= Ai.Bi

The carry propagate function is defined as Pi= Ai xor Bi

The answer follows from the function definitions.

3. **(5 marks) (*Floating Point Number Representation*)**:

Consider the following two **9-bit floating-point** representations based on the *IEEE floating-point* format.
Format A : There is one sign bit, there are k = 5 exponent bits (exponent bias is 15) and there are n = 3 fraction bits.

Format B : There is one sign bit,there are k = 4 exponent bits(exponent bias is 7) and there are n = 4 fraction bits

Below, you are given some bit patterns in **Format A**, and your task is to convert them to the closest value in **Format B**. If rounding is necessary, you should round toward positive infinity. In addition, give the values of number given by **Format A**and **Format B** bit patterns. Give these as whole numbers (i.e. **17**) or as fractions (i.e. **17/64 or 17/26**).

Format A		Format B	
BITS	VALUE	BITS	VALUE
101111001	-9/8	101110010	-9/8
010110011			
100111010			
000000111			
111100000			
010111100			

Solution:

Format A		Format B		
BITS	VALUE	BITS	VALUE	Comments
101111001	-9/8	101110010	-9/8	
010110011	176	011100110	176	
100111010	-5/1024	100000101	-5/1024	Norm $->$ denorm
000000111	7/131072	000000001	1/1024	smallest positive denorm
111100000	-8192	111101111	-2148	smallest no. $>$ - infinity
010111100	384	011110000	+infinity	Round to infinity

4. **(5 marks) (*Instruction Set Architecture*)**:

(a) Find The most appropriate matching for the following pairs (provide examples to justify your choice):

```
X : Indirect Addressing       1. Loop
Y : Immediate Addressing        2. Pointers
Z : Auto Decrement Addressing   3. Constants

(i) X - 3, Y - 2, Z - 1   (ii) X - 2, Y - 3, Z - 1
(iii) X - 3, Y - 1, Z - 2 (iv) X - 2, Y - 1, Z - 3
```

Solution:

```
 1) Constants : Immediate addressing ; Example : mov R1, 2;
 Intialized in the instruction itself. Therefore, Y-3

2) Loop : auto decrement/increment addressing;
Example : while [*A++];
The memory locations are automatically incremented.
Therefore, Z-1

3) Pointers : Indirect addressing ; Ex.  int temp = *p;
Here temp is assigned the value of int type stored at the
address contained in p. Therefore X-2

Hence (ii) is the correct solution.
```

(b) In Assembly language programming, minimum number of operands required for an instruction is/are —-. Give examples to justify your choice.

```
(i) zero (ii) one
(iii) two (iv) both i and ii
```

Solution:

```
(i)
```

(c) When a function (subroutine) is called, the address of the instruction following the CALL instructions is stored in/on the

```
(i) stack pointer (ii) accumulator
(iii) stack (iv) program counter
```

Solution:

```
(i)
```

(d) Stack-organized Computer uses instruction of ——

```
(i) indirect addressing  (ii) index addressing
(iii) two addressing     (iv) zero addressing
```

Solution:

```
(iv)
```

(e) The maximum addressing capacity of a CPU which uses 16 bit data bus & 32 bit address bus is

```
(i) 64K  (ii) 4G
(iii) both (i) and (ii) (iv) None of these
```

Solution:

```
(ii). 32 bit address bus can address upto 2^32 addresses
2^32 = 2*2^30 = 2GB
```

5. **(2+2+3+3 = 10 marks) (*Assembly Language Programming*)**:

(a) Consider an architecture where addresses are 64 bits and integers are 32 bits. Consider the following *C* code snippet.

```
int[3] x;
x[0] = 4;
x[1] = 5;
x[2] = 6;
```

Suppose the array starts at address *0x80*.

i. Mention the memory contents after this code runs. You should label each byte with an address, and show the value of each byte (in hexadecimal).

Solution:

```
Address Contents
...
0x80 0x04
0x81 0x00
0x82 0x00
0x83 0x00
0x84 0x05
0x85 0x00
0x86 0x00
0x87 0x00
0x88 0x06
0x89 0x00
0x8A 0x00
0x8B 0x00
```

ii. What is the address of the last word of the array?

Solution:

```
0X88
```

(b) An 8-bit register contains the binary value 10011100. What is the register value after arithmetic shift right? Starting from the initial number 10011100, determine the register value after an arithmetic shift left, and state whether there is an overflow.

Solution:

```
 R = 10011100
Arithmetic shift right: 11001110
Arithmetic shift left: 00111000
overflow because a negative number changed to positive.
```

(c) *lui $r1, immediate* is the "load upper immediate" instruction. It allows you to load the upper 16 bits of a register with the immediate value, and it zeros out the lower 16 bits. Use this instruction (and additional instructions) to load the following 32 bit value **0xcafebabe**. Assume that you can write immediate values in hex.

Solution:

```
lui  $r1, 0xcafe        # Load upper 16 bits with 0xcafe
ori  $r1, $r1, 0xbabe   # Load lower 16 bits with 0xbabe

Notice that we use ori (which is bitwise OR with an immediate value)
instead of addi. addi does signed addition. Thus, it assumes
the immediate value (which uses 16 bits) is signed. To add a signed
16 bit 2's Complement representation to a 32 bit representation
(located in the register), the CPU sign-extends the 16 bit 2C to
a 32 bit 2C then adds.

Since 0xbabe has its MSb equal to 1, this will sign extend,
and cause the result of the addition to mess up the upper 16 bits.

In effect the addition looks like:

1100 1010 1110 1111 0000 0000 0000 0000 (register 1)
1111 1111 1111 1111 1011 1010 1011 1110 (sign extended 0xbabe)
---------------------------------------
1100 1010 1110 1110 1011 1010 1011 1110 (result of adding)

This is NOT the result we want. However, if you do bitwise or with ori,
you get the desired answer. ori is a logical operator.
When it extends the immediate from 16 bits to 32 bits,
it zero-extends it. The assumption is that logical
operations work with bits, not with signed numbers, thus
it zero-extends instead of sign-extends.

Yes, you could use addiu (add immediate unsigned),
which behaves in the same way, instead of ori.
However, the book postpones this instruction until later.
```

(d) Fill out the 32 bit value for addi $r3$,r7, 12. Use all 0's for the opcode. Clearly label bits (i.e. indicate which bits are the opcode, register, etc).

Solution:

```
 000000 00111 00011 0000 0000 0000 1100
------ ----- ----- ------------------
 (1)    (2)   (3)        (4)

For the above,

This is the 6-bit opcode.
Set it as 000000 so you wouldn't have to remember
the value of the opcode, just its location.

This is the second register (register 7).
You encode the register's number as 00111.

This is the first register (register 3).
You encode the register's number as 00011.
This is the immediate value written as 16 bits, 2C.
```