# NATIONAL UNIVERSITY OF SINGAPORE

## SCHOOL OF COMPUTING

**EXAMINATION FOR**

**Semester 2 AY2012/2013**

## CS2100 – COMPUTER ORGANISATION

**April 2013** **Time allowed: 2 hours**

---

## INSTRUCTIONS TO CANDIDATES

1. This examination paper consists of **TEN (10)** questions and comprises **NINE (9)** printed pages.
2. This is a **CLOSED BOOK** examination. Two handwritten A4 reference sheets are allowed. Calculators are not allowed.
3. Answer all questions.
4. Write your answers in the **ANSWER BOOKLET** provided.
5. Fill in your Matriculation Number with a pen, clearly on odd-numbered pages of your ANSWER BOOKLET.
6. You may use pencil to write your answers.
7. You are to submit only the **ANSWER BOOKLET** and no other document.

**Questions 1 - 6:** Each question has only one correct answer. Write your answers in the boxes provided in the Answer Booklet. One mark is awarded for a correct answer and no penalty for wrong answer.

1. A code *C* with 4 code values are given as follows:

   { 010011, 001100, 111101, 110100 }

   What is the **Hamming distance** and **code efficiency** of *C*?

   A. Hamming distance = 2; code efficiency = 1/4
   B. Hamming distance = 2; code efficiency = 1/3
   C. Hamming distance = 3; code efficiency = 1/4
   D. Hamming distance = 3; code efficiency = 1/3
   E. Hamming distance = 4; code efficiency = 1/4

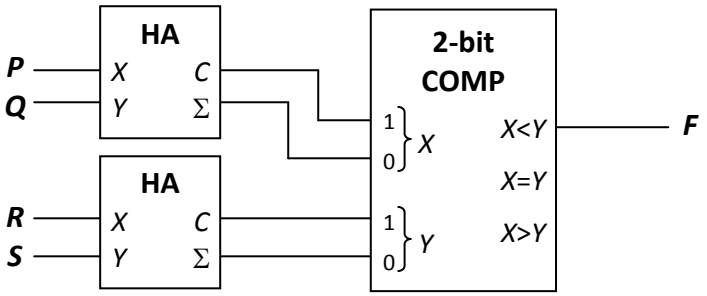
2. Given the following hexadecimal representation in **IEEE 754 single-precision floating-point number system**:

   4 3 0 5 C 0 0 0

   What decimal value does it represent?

   A. 5.75
   B. 66.875
   C. 133.75
   D. 267.5
   E. None of the above.

3. Given the following circuit using a 2-bit magnitude comparator and two half adders (HA), what is the function ***F(P,Q,R,S)***? (*C* and Σ are respectively the carry and sum outputs of the half adder.)




   A. Σ*m*(1, 2, 3)
   B. Σ*m*(1, 2, 3, 6, 7)
   C. Σ*m*(1, 2, 3, 7, 11)
   D. Σ*m*(1, 2, 3, 6, 7, 11)
   E. None of the above.

4. Some processors have a "turbo switch", which increases its clock frequency when activated. Suppose we have a program A with an average CPI of 4.0 executing on one such processor. What is the new average CPI and the speedup gained for program A if the turbo switch increases the clock frequency by 25%?

   A. Average CPI is 3.0, speedup is 1.25 after turbo switch is activated.
   B. Average CPI is 3.0, speedup is 1.67 after turbo switch is activated.
   C. Average CPI is 4.0, speedup is 1.25 after turbo switch is activated.
   D. Average CPI is 4.0, speedup is 1.50 after turbo switch is activated.
   E. None of the above.

5. Suppose we have a MIPS program with both "j" (jump) and "beq" (branch-if-equal):

```
##  code insertion  point ##
here:  …….              #instruction address at 0x00 40 00 00
       beq ……., here
        ……
       j here
```

   If we insert 1024 ($2^{10}$) instructions before the "here" label, which of the following statements is **TRUE**? $XXX_{new}$ refers to the **instruction encoding for instruction XXX** after insertion, $XXX_{old}$ refers to the **original instruction encoding** before insertion.

   A. $beq_{new} - beq_{old} = 0$ and $j_{new} - j_{old} = 1024$
   B. $beq_{new} - beq_{old} = 1024$ and $j_{new} - j_{old} = 0$
   C. $beq_{new} - beq_{old} = 1024$ and $j_{new} - j_{old} = 256$
   D. $beq_{new} - beq_{old} = 0$ and $j_{new} - j_{old} = 256$
   E. None of the above.

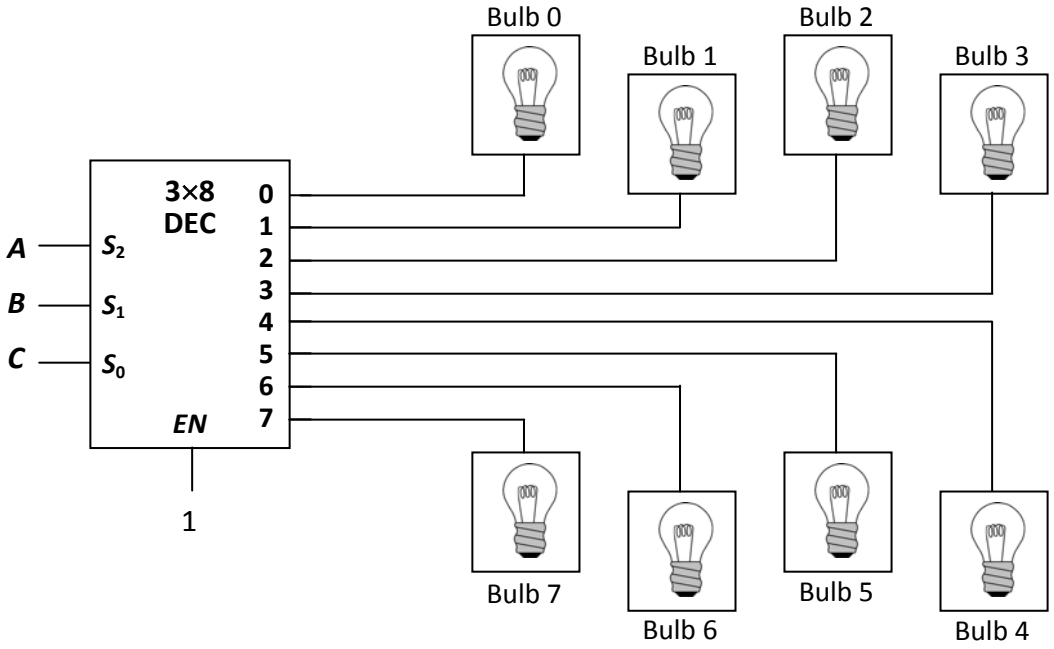
6. Suppose we are given a cache of X number of bytes in data capacity. Which of the following statements regarding cache is/are **TRUE**?

   i. To exploit temporal locality, we should have more cache blocks of smaller size rather than fewer cache blocks of larger size.
   ii. Increasing the number of sets in N-way set associative cache while keeping the cache block size constant will increase the number of cold misses.
   iii. To exploit spatial locality, we should have more cache blocks of smaller size rather than fewer cache blocks of larger size.

   A. Only (i) is TRUE.
   B. Only (i) and (ii) are TRUE.
   C. Only (ii) and (iii) are TRUE.
   D. (i), (ii) and (iii) are all TRUE.
   E. None of the above.

7. **[6 marks]**

The diagram below shows how a light bulb is chosen to be lit among eight light bulbs depending on the 3-bit input *ABC*. For example, if *ABC* = 011, only bulb 3 is lit.




In a game show, to add an element of surprise, the mapping of *ABC* to bulb number is modified as follows:

| *A* | *B* | *C* | Bulb |
|---|---|---|---|
| 0 | 0 | 0 | 3 |
| 0 | 0 | 1 | 6 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 7 |
| 1 | 0 | 1 | 4 |
| 1 | 1 | 0 | 5 |
| 1 | 1 | 1 | 2 |

This means that if *ABC* = 000, bulb 3 is lit instead.

You may implement this change by adding suitable components you have learned in class. You may use only **<u>one type</u>** of component, though there is no limit on the number of components. (Full credit will be given if you use the fewest possible number.)

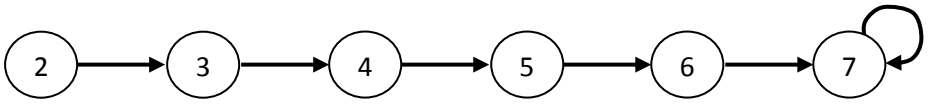
Only logic constants 0 and 1, and *A*, *B*, *C* are available. Complemented variables are not available.

As the game show producer may decide to change the mapping of *ABC* to bulb number on each show, your design must be able to adapt to such changes easily.

On the answer sheet, the eight bulbs are represented by a block diagram labelled "8 bulbs" for convenience. You are not to change what is given on the answer sheet.

8. **[10 marks]**

The state diagram of a certain sequential circuit is shown below, with state values shown in decimal.




a. Implement the above sequential circuit using *JK* flip-flops, which are named *A*, *B* and *C*. Write out the **simplified SOP expressions** for all the flip-flop inputs. You do not need to draw the logic diagram. [6 marks]

b. Using the fewest number of logic gates besides the flip-flops, how many logic gates are needed to implement this circuit and what are the gates? [2 marks]

c. What is a self-correcting circuit? Is this sequential circuit, if correctly implemented using the simplified SOP expressions for all the flip-flops inputs, a self-correcting one and why? [2 marks]

9. [**7 marks**]

Given two binary values X and Y, **hamming distance**(X, Y) is the number of bits that's different between them. For example, X = 1**<u>010</u>**1, Y = 1**<u>101</u>**1 ➔ hamming distance = 3, with the different bits in bold font and underlined.

Let us implement a simple MIPS program to calculate the hamming distance between two 32-bit values, X and Y. The idea can be split into the following two steps:

<u>Step 1. Turn the different bits into "1"s</u> [2 marks]

Let us define:

**T = X op Y**

such that T contains "1"s in positions where X and Y are different, and "0"s in positions where X and Y have the same bit.

For example, if X = 1**<u>010</u>**1, Y = 1**<u>101</u>**1, then **T = 01110**.

Suppose X is in register $s0, and Y is in register $s1, give a sequence of MIPS instructions to generate T in register $t0. You will receive 2 marks if only **one instruction is used** to generate T. Correct answer with more than one instruction will receive at most 1 mark.

<u>Step 2. Count the number of "1"s</u> [5 marks]

Instead of counting the "1"s one bit at a time, we can pre-compute the number of "1"s in all 4-bit values, e.g. 0000 = {0}, 0001 = {1}, …., 0110 = {2}, …, 1111 = {4}, and store them into an array of 16 values. The array is indexed by the 4-bit value, and contains the number of "1"s as content. e.g. $A[0_{10}] = 0$, $A[6_{10}] = 2$, $A[15_{10}] = 4$ etc.

We can now count the number of "1"s in a 32-bit value by chopping it into chunks of 4-bits, then perform a lookup in the array A[]. Suppose the **byte array** A[] is precomputed as described and its address is stored in $s2, complete the partial MIPS program given in the answer booklet to count the "1"s stored in register $t0, and store the result in register $s3.

The content of A[] is given below for your reference:

| idx | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 1 | 2 | 1 | 2 | 2 | 3 | 1 | 2 | 2 | 3 | 2 | 3 | 3 | 4 |

10. [**21 marks**]

We will study the execution behaviour of a program with almost all ideas you've learned in the second portion of this course.

**Note:** Although there are multiple parts for this questions, most of them are independent. So, don't panic if you are stuck on one part, just move on to later parts.

Study the following MIPS program. The annotation is a C-like high level program to help your understanding. The program is duplicated on each page to ease your attempt.

| | | |
|---|---|---|
| address of A[] ➔ $s0<br>address of B[] ➔ $s1<br>i ➔ $s2, initialized to 0 | | `int A[1024] = {...some data...}`<br>`int B[2048] = {...some data...}`<br>`int i = 0` |
| $s4 initialized to 1024 as array bound.<br>$t1 is an array pointer to A[i], initialized to address of A[0].<br>$t3 is an array pointer to B[i*2], initialized to address of B[0]. | | |
| i1 | `loop:`<br>`  beq  $s2, $s4, end` | `while (i < 1024) {` |
| i2<br>i3 | `  lw  $t2, 0($t1)`<br>`  lw  $t4, 4($t3)` | `  //load A[i] into $t2`<br>`  //load B[i*2+1] into $t4` |
| i4<br>i5 | `  andi $t5, $s2, 0x1`<br>`  beq  $t5, $zero, else` | `  if (i % 2 == 1) //odd` |
| i6<br>i7<br>i8 | `  add  $t2, $t2, $t4`<br>`  sw  $t2, 0($t3)`<br>`  j  endW` | `  B[i*2] = A[i] + B[i*2+1]` |
| i9<br>i10 | `else:`<br>`  sub  $t2, $t2, $t4`<br>`  sw  $t2, 0($t3)` | `  else`<br>`  B[i*2] = A[i] - B[i*2+1]` |
| i11<br>i12<br>i13<br>i14 | `endW:`<br>`  addi $s2, $s2, 1`<br>`  addi $t1, $t1, 4`<br>`  addi $t3, $t3, 8`<br>`  j  loop` | `  i++`<br>`  //move $t1 by 1 element`<br>`  //move $t3 by 2 elements` |
| | `end:` | `}` |

a. (Basic statistics) For **one iteration of the loop** (from "beq $s2, …" to "j loop"), count the number of instructions. We split the counting into two cases: (i) when "i" is odd, (ii) when "i" is even. [2 marks]

b. (Performance for non-pipelined processor) Suppose the code is executed on a non-pipelined MIPS processor, with the following capabilities:

| | |
|---|---|
| Cycle Per Instruction | 5 |
| Clock Frequency | 100Mhz |

For the **full program (i.e. 1024 iterations),** give the **total number of instructions executed** and the **total execution time**. You can give the expressions without calculating it, e.g. (7800 / 25000, 35 * 1024 etc). [2 marks]

| | | |
|---|---|---|
| i1 | **loop:**<br>**beq $s2, $s4, end** | while (i < 1024) { |
| i2<br>i3 | **lw $t2, 0($t1)**<br>**lw $t4, 4($t3)** | //load A[i] into $t2<br>//load B[i*2+1] into $t4 |
| i4<br>i5 | **andi $t5, $s2, 0x1**<br>**beq $t5, $zero, else** | if (i % 2 == 1) //odd |
| i6<br>i7<br>i8 | **add $t2, $t2, $t4**<br>**sw $t2, 0($t3)**<br>**j endW** | B[i*2] = A[i] + B[i*2+1] |
| i9<br>i10 | **else:**<br>**sub $t2, $t2, $t4**<br>**sw $t2, 0($t3)** | else<br>B[i*2] = A[i] - B[i*2+1] |
| i11<br>i12<br>i13<br>i14 | **endW:**<br>**addi $s2, $s2, 1**<br>**addi $t1, $t1, 4**<br>**addi $t3, $t3, 8**<br>**j loop** | i++<br>//move $t1 by 1 element<br>//move $t3 by 2 elements |
| | **end:** | } |

c. (Pipelined Processor – RAW hazard) Identify all pairs of instructions that have RAW hazard under the MIPS processor. Use the following notation to help:

```
add $t1, $t2, $t3  # i1
sub $t4, $t1, $t9  # i2
```
Indicate the dependency as "**i1 → i2**".

For each RAW hazard indentified, indicate whether it is resolved by the forwarding path. If the dependency is not resolved, indicate the number of stall cycles incurred. Note that marks will be deducted for incorrectly identified pair. [3 marks]

d. (Pipelined Processor – Early Branching) For the two conditional branches "`beq $s2, $s4, end` **`#i1`**" and "`beq $t5, $zero, else` **`#i5`**", indicate the number of stall cycles incurred if **early branching** is used. [3 marks]

e. (Pipelined Processor – Branch Prediction) Suppose we use "predicted taken" branch prediction instead of early branching. Wrong prediction incurs "waste cycles" as incorrect instructions are pumped into pipeline but discarded eventually. How many wasted cycles are incurred for the each of the branches for the whole program run? [2 marks]

| | | |
|---|---|---|
| | **loop:** | |
| **i1** | **beq $s2, $s4, end** | while (i < 1024) { |
| **i2** | **lw $t2, 0($t1)** | //load A[i] into $t2 |
| **i3** | **lw $t4, 4($t3)** | //load B[i*2+1] into $t4 |
| **i4** | **andi $t5, $s2, 0x1** | if (i % 2 == 1) //odd |
| **i5** | **beq $t5, $zero, else** | |
| **i6** | **add $t2, $t2, $t4** | |
| **i7** | **sw $t2, 0($t3)** | B[i*2] = A[i] + B[i*2+1] |
| **i8** | **j endW** | |
| | **else:** | |
| **i9** | **sub $t2, $t2, $t4** | else |
| **i10** | **sw $t2, 0($t3)** | B[i*2] = A[i] - B[i*2+1] |
| | **endW:** | |
| **i11** | **addi $s2, $s2, 1** | i++ |
| **i12** | **addi $t1, $t1, 4** | //move $t1 by 1 element |
| **i13** | **addi $t3, $t3, 8** | //move $t3 by 2 elements |
| **i14** | **j loop** | |
| | **end:** | } |

f. (Instruction Cache – Direct Mapped ) Given the **instruction cache** below:

- Total number of cache blocks = 4
- Size of each cache block = 2 words (i.e. 8 bytes)

What is the number of cold and conflict misses for the **first two iterations** of the program? You can assume that **instruction i1** is at address 0x400000. Please note that this question is about **instructions accessed during execution**. [3 marks]

g. (Data Cache – Direct Mapped) Given the **data cache** below:

- Total number of cache blocks = 8
- Size of each cache block = 4 words (i.e. 16 bytes)

Suppose array **B[]** is placed right after array **A[]** in the memory. For simplicity, you can assume array **A[]** is at address 0x0. What is the **total number of cache accesses** and the **total number of cache misses** for the whole program run? [3 marks]

h. (Data Cache – Set Associative) Given the **data cache** below:

- Total number of cache blocks = 8
- 2-way set associative, i.e. total of 4 sets with 2 cache blocks each.
- Size of each cache blocks = 4 words (i.e. 16 bytes)
- Least recently used block is replaced.

The arrays A[] and B[] are laid out in the same way. What is the **total number of cache misses** for the first four iterations (i = 0 to 3) and the whole program run? [3 marks]

**~~~ END OF PAPER ~~~**