Rec1Extra

Sunday, August 20, 2023 2:49 PM

Rec1Extra

CS2100 Computer Organization
Recitation 1
Extra Exercises

Question 1

Write a program that:

1. Asks for an operator, which can be "+", "-", "*", "/" or "q".
2. If the operator is +, -, * or /, ask for two floating point numbers.
3. Prints "Unrecognized operation" if the operator entered is not +, -, *, / or q.
4. If the person enters "q" the program says "Bye!" and exits. If the user enters +, -, *, /, the program performs the operation on the two numbers entered and prints the result in two decimal places.
5. If the user did not enter "q", go to 1.

Some sample runs are shown here:

```
Please enter an operation (+, -, *, / or q to quit): +

Enter the first nunber: 1.2

Enter the second number: 2.3

1.20 + 2.30 = 3.50

Please enter an operation (+, -, *, / or q to quit): /

Enter the first nunber: 3.22

Enter the second number: 10.15

3.22 / 10.15 = 0.32

Please enter an operation (+, -, *, / or q to quit): ~

Unrecognized operation
Please enter an operation (+, -, *, / or q to quit): *

Enter the first nunber: 1.2

Enter the second number: 2.5

1.20 * 2.50 = 3.00

Please enter an operation (+, -, *, / or q to quit): q

Bye!
```

```
#include <stdio.h>

int main() {
   char op;
   int i, j;

   printf("Please enter an operation (+, -, *, / or q to quit):\n");
   scanf(" %c", &op);  // Use %c to scan a single character, and use & before op to get its address.

   if (op == 'q') {
      printf("Bye!\n");
   } else if (op != '+' && op != '-' && op != '*' && op != '/') { // Change || (logical OR) to && (logical AND)
      printf("Unrecognized operation\n");
   } else {
      printf("Enter the first number:\n");
      scanf("%d", &i);
      printf("Enter the second number:\n");
      scanf("%d", &j);

      if (op == '+') {
         printf("%d %c %d = %d\n", i, op, j, i + j);
      } else if (op == '-') {
         printf("%d %c %d = %d\n", i, op, j, i - j);
      } else if (op == '*') {
         printf("%d %c %d = %d\n", i, op, j, i * j);
      } else if (op == '/') {
         if (j != 0) {
            printf("%d %c %d = %lf\n", i, op, j, (double)i / j);
         } else {
            printf("Division by zero is not allowed.\n");
         }
```

```
    }
  }

  return 0;
}
```

Question 2

Using what you know about number systems and C, show why this while loop may never end:

```
float x = 0.0;
while(x != 0.1)
  x += 0.1;
```

Because 0.1 is not a float. The representation of 0.1 decimal and 0.1 float are different numbers.

tut01qns (1)

Tuesday, September 5, 2023 10:42 AM

tut01qns (1)

CS2100 Computer Organization
Tutorial 1: C and Number Systems
(Week 3: 28 Aug – 1 Sep 2023)

Note: Friday's tutorial groups will have tutorial 1 on 25 August in lieu of 1 September. Refer to Canvas announcement for more information.

1. In 2's complement representation, "sign extension" is used when we want to represent an *n-bit* signed integer as an *m-bit* signed integer, where $m > n$. We do this by copying the MSB (most significant bit) of the *n-bit* number $m - n$ times to the left of the *n-bit* number to create the *m-*bit number.

   For example, we want to sign-extend 0b0110 to an 8-bit number. Here $n = 4$, $m = 8$, and thus we copy the MSB bit 0 four (8 – 4) times, giving 0b00000110.

   Similarly, if we want to sign-extend 0b1010 to an 8-bit number, we would get 0b11111010.

   Show that IN GENERAL, sign extension is value-preserving. For example, 0b00000110 = 0b0110 and 0b11111010 = 0b1010.

2. We generalize $(r - 1)$'s-complement (also called *radix diminished complement*) to include fraction as follows:

   $$\textbf{(}r - 1\textbf{)'s complement of } N = r^n - r^{-m} - N$$

   where n is the number of integer digits and m the number of fractional digits. (If there are no fractional digits, then $m = 0$ and the formula becomes $r^n - 1 - N$ as given in class.)

   For example, the 1's complement of 011.01 is $(2^3 - 2^{-2}) - 011.01 = (1000 - 0.01) - 011.01 = 111.11 - 011.01 = 100.10$. (Since 011.01 represents the decimal value 3.25 in 1's complement, this means that -3.25 is represented as 100.10 in 1's complement.)

   Perform the following binary subtractions of values represented in 1's complement representation by using addition instead. (Note: Recall that when dealing with complement representations, the two operands must have the same number of digits.)

   (a) 0101.11 – 010.0101
   (b) 010111.101 – 0111010.11

   Is sign extension used in your working? If so, highlight it.

   Check your answers by converting the operands and answers to their actual decimal values.

3. Convert the following numbers to fixed-point binary in 2's complement, with 4 bits for the integer portion and 3 bits for the fraction portion.

3. Convert the following numbers to fixed-point binary in 2's complement, with 4 bits for the integer portion and 3 bits for the fraction portion.

   (a) 1.75 (b) -2.5 (c) 3.876 (d) 2.1

   Using the binary representations you have derived, convert them back into decimal. Comment on the compromise between range and accuracy of the fixed-point binary system.

4. [AY2010/2011 Semester 2 Term Test #1]
   How would you represent the decimal value –0.078125 in the IEEE 754 single-precision representation? Express your answer in hexadecimal. Show your working.

5. Given the partial C program shown below, complete the two functions: **readArray()** to read data into an integer array (with at most 10 elements) and **reverseArray()** to reverse the array. For **reverseArray()**, you are to provide two versions: an iterative version and a recursive version. For the recursive version, you may write an auxiliary/driver function to call the recursive function.

```
#include <stdio.h>
#define MAX 10

int readArray(int [], int);
void printArray(int [], int);
void reverseArray(int [], int);

int main(void) {
   int array[MAX], numElements;

   numElements = readArray(array, MAX);
   reverseArray(array, numElements);
   printArray(array, numElements);

   return 0;
}

int readArray(int arr[], int limit) {

   // ...
   printf("Enter up to %d integers, terminating with a negative
integer.\n", limit);
   // ...
}
```

```
    // ...
}

void reverseArray(int arr[], int size) {

    // ...
}

void printArray(int arr[], int size) {
    int i;

    for (i=0; i<size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```


6. Trace the following program manually (do not run it on a computer) and write out its output. When you present your solution, draw diagrams to explain.

```
#include <stdio.h>

int main(void) {
    int a = 3, *b, c, *d, e, *f;

    b = &a;
    *b = 5;
    c = *b * 3;
    d = b;
    e = *b + c;
    *d = c + e;
    f = &e;
    a = *f + *b;
    *f = *d - *b;

    printf("a = %d, c = %d, e = %d\n", a, c, e);
    printf("*b = %d, *d = %d, *f = %d\n", *b, *d, *f);

    return 0;
}
```

Remember to post on the Canvas forum or QnA if you have any queries.

Remember to post on the Canvas forum or QnA if you have any queries.

tut02qns

Thursday, September 7, 2023 8:15 AM

tut02qns

CS2100 Computer Organization
Tutorial 2: C and MIPS
(Week 4: 4 – 8 Sep 2023)

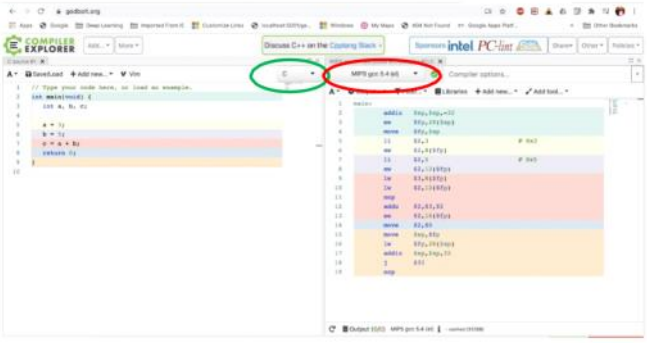
Discussion Questions

You may discuss these on the Canvas Forum. Answers will not be given.

D1. **Exploration: C to MIPS**

Go to this website https://godbolt.org/ and copy the C code below into the left box, and ensure that you choose "C" in the dropdown list in green in the screenshot after the C code, and choose MIPS gcc 5.4 (el) or MIPS gcc 5.4 in the dropdown list circled red (do not choose MIPS64 gcc 5.4 or MIPS64 gcc 5.4 (el):

```
int main(void) {
   int a, b, c;

   a = 3;
   b = 5;
   c = a + b;
   return 0;
}
```



AY2023/24 Semester 1 **Page 1 of 5** *Tutorial 2*

00010111

The following is extracted from the Assembly output.

```
    addiu   $sp,$sp,-32
    sw      $fp,28($sp)
    move    $fp,$sp
    li      $2,3                    # 0x3
    sw      $2,8($fp)
    movz    $31,$31,$0
    li      $2,5                    # 0x5
    sw      $2,12($fp)
    lw      $3,8($fp)
    lw      $2,12($fp)
    nop
    addu    $2,$3,$2
    sw      $2,16($fp)
    move    $2,$0
    move    $sp,$fp
    lw      $fp,28($sp)
    addiu   $sp,$sp,32
    j       $31
    nop
```

We covered **sw**, **lw** and **j** in lecture, but not the rest. Find out what they are. (Note: **li** will be used in the labs later and **nop** will be mentioned in the topic on Pipelining. **move**, like **li**, is a pseudo- instruction. **$sp**, **$fp**, **movz**, **addiu**, and **addu** are not in the syllabus.)

D2. For each of the following instructions, indicate if it is valid or not. If not, explain why and suggest a correction. Note that the | in (d) is the bitwise OR operation.

a. **`add  $t1, $t2, $t3`** # $t3 = $t1 + $t2
b. **`addi $t1, $0, 0x25`** # $t1 = 0x25
c. **`subi $t2, $t1, 3`** # $t2 = $t1 - 3
d. **`ori  $t3, $t4, 0xAC120000`** # $t3 = $t4 | 0xAC120000
e. **`sll  $t5, $t2, 0x21`** # shift left $t2 33 bits and store in $t5

Tutorial Questions

1. **Bitwise operations**
Find out about the following bitwise operations in C and explain and illustrate each of them with an example.

- | (bitwise OR)
- & (bitwise AND)
- ^ (bitwise XOR)
- ~ (one's complement)
- << (left shift)
- >> (right shift)

You may use the following code template for your illustration. Variables of the data type **char** take up 8 bits of memory.

```
#include <stdio.h>

typedef unsigned char byte_t;

void printByte(byte_t);

int main(void) {
   byte_t a, b;

   a = 5;
   b = 22;
   printf("a    = "); printByte(a);   printf("\n");
   printf("b    = "); printByte(b);   printf("\n");
   printf("a|b  = "); printByte(a|b); printf("\n");
   return 0;
}

void printByte(byte_t x) {
   printf("%c%c%c%c%c%c%c%c",
             (x & 0x80 ? '1' : '0'),
             (x & 0x40 ? '1' : '0'),
             (x & 0x20 ? '1' : '0'),
             (x & 0x10 ? '1' : '0'),
             (x & 0x08 ? '1' : '0'),
             (x & 0x04 ? '1' : '0'),
             (x & 0x02 ? '1' : '0'),
             (x & 0x01 ? '1' : '0'));
}
```


2. **Swapping**
We have discussed in lecture how to swap two variables in a function:

```
void swap(int *a, int *b) {
    int t = *a;
    *a = *b;
    *b = t;
}
```

In the above code, a temporary variable **t** is used.

A possible alternative is to use some bitwise operator to perform the swap, without using any temporary variable. Write a function to do this. What is the limitation of this new function?

3. **MIPS Bitwise Operations**
Implement the following in MIPS assembly code. Assume that integer variables **a**, **b** and **c** are mapped to registers $s0, $s1 and $s2 respectively. Each part is independent of all the other parts. **For bitwise instructions (e.g. ori, andi, etc),** any immediate values you use should be written in binary for this question. This is optional for non-bitwise instructions (e.g. **addi**, etc).

Note that bit 31 is the most significant bit (MSB) on the left, and bit 0 is the least significant bit (LSB) on the right, i.e.:

MSB					LSB
Bit 31	Bit 30	Bit 29	...	Bit 1	Bit 0

a. Set bits 2, 8, 9, 14 and 16 of **b** to 1. Leave all other bits unchanged.

b. Copy over bits 1, 3 and 7 of **b** into **a**, without changing any other bits of **a**.

c. Make bits 2, 4 and 8 of **c** the inverse of bits 1, 3 and 7 of **b** (i.e. if bit 1 of **b** is 0, then bit 2 of **c** should be 1; if bit 1 of **b** is 1, then bit 2 of **c** should be 0), without changing any other bits of **c**.

4. **MIPS Arithmetic**
Write the following in MIPS assembly, using as few instructions as possible. You may rewrite the equations if necessary to minimize instructions.

In all parts you can assume that integer variables **a**, **b**, **c** and **d** are mapped to registers $s0, $s1, $s2 and $s3 respectively. Each part is independent of the others.

a. c = a + b

if necessary to minimize instructions.

In all parts you can assume that integer variables **a**, **b**, **c** and **d** are mapped to registers $s0, $s1, $s2 and $s3 respectively. Each part is independent of the others.

a. c = a + b add $s2, $s0, $s1

b. d = a + b – c sub $s3, $s1, $s2 | add $s3, $s3, $s0

c. c = 2b + (a – 2)

d. d = 6a + 3(b – 2c) = 3(2(a-c)+b)

c. sll $s2, $s1, 1
addi $t0, $s0, -2
add $s2, $s2, $t0

d. sub $t0, $s0, $s2
sll $t0, $t0, 1
add $t0, $t0, $s1
sll $s3, $t0, 2
sub $s3, $s3, $t0

lui $t1, 0b 1111 1111 1111 1111
ori $t1, $t1, 0b 1111 1111 1111 0101
and $s0, $s0, $t1
or $s0, $s0, $t0

c.

5. [AY2013/14 Semester 2 Exam]
The mysterious MIPS code below assumes that **$s0 is a 31-bit binary sequence**, i.e. the MSB (most significant bit) of **$s0** is assumed to be zero at the start of the code.

```
     add  $t0, $s0, $zero   # make a copy of $s0 in $t0
     lui  $t1, 0x8000
lp:  beq  $t0, $zero, e     if $t0 = 0, end
     andi $t2, $t0, 1       place last bit of t0 into t2
     beq  $t2, $zero, s     if t2=0, shift t0 to right by 1
     xor  $s0, $s0, $t1
s:   srl  $t0, $t0, 1
     j    lp
e:
```

a. For each of the following initial values in register **$s0** at the beginning of the code, give the hexadecimal value of the content in register $**s0** at the end of the code.

   i. Decimal value **31**.
   ii. Hexadecimal value **0x0AAAAAAA**.

b. Explain the purpose of the code in one sentence.

tut03qns

Thursday, September 14, 2023 6:21 PM

tut03qns

CS2100 Computer Organization
Tutorial 3: Array and MIPS Instruction Encoding
(Week 5: 11 – 15 Sep 2023)

Discussion Questions

D1. Given two integer arrays *A* and *B* with unknown number of elements, and their base addresses stored in registers **$s0** and **$s1** respectively, study the MIPS code below. Note that an integer takes up 32 bits of memory.

```
      addi $t0, $s0, 0
      addi $t1, $s1, 0
loop: lw   $t3, 0($t0)
      lw   $t4, 0($t1)
      slt  $t5, $t4, $t3      # line A
      beq  $t5, $zero, skip   # line B
      sw   $t4, 0($t0)
      sw   $t3, 0($t1)
skip: addi $t0, $t0, 4
      addi $t1, $t1, 4
      bne  $t3, $zero, loop
```

(a) What is the purpose of register **$t1** in this code?

(b) If array *A* = {7, 4, 1, 6, 0, 5, 9, 0} and
array *B* = {3, 4, 5, 2, 1, 0, 0, 9},
give the final content of these two arrays.

(c) How many **store word** operations are performed given the contents of the arrays in part (b)?

(d) What is the value (in decimal) of the immediate field in the machine code representation of the **bne** instruction?

(e) The two lines indicated as "line A" and "line B" represent the translation of a MIPS pseudo-instruction. Give the corresponding pseudo-instruction.

AY2023/24 Semester 1 Page **1** of **4** *Tutorial 3*

S0 - a
S1 - b

T0 - a[0]
T1 - b[0]
T3 -
T4 -

If T4 < t3 then branch

Tutorial Questions:

1. Below is a C code that performs palindrome checking. A palindrome is a sequence of characters that reads the same backward or forward. For example, "madam" and "rotator" are palindromes.

```
char string[size] = { ... }; // some string
int low, high, matched;

// Translate to MIPS from this point onwards
low = 0;
high = size-1;
matched = 1;          // assume this is a palindrome
                      // In C, 1 means true and 0 means false
while ((low < high) && matched) {
   if (string[low] != string[high])
      matched = 0;  // found a mismatch
   else {
      low++;
      high--;
   }
}
// "matched" = 1 (palindrome) or 0 (not palindrome)
```

Given the following variable mappings:

low ➔ $s0;
high ➔ $s1;
matched ➔ $s3;
base address of string[] ➔ $s4;
size ➔ $s5

(a) Translate the C code into MIPS code by keeping track of the indices.

(b) Translate the C code into MIPS code by using the idea of "array pointer". Basically, we keep track of the actual addresses of the elements to be accessed, rather than the indices. Refer to lecture set #8, slide 34 for an example.

Note: Recall the "short circuit" logical AND operation in C. Given condition (A && B), condition B will not be checked if A is found to be false.

2. (a) You accidentally spilled coffee on your best friend's MIPS assembly code printout. Fortunately, there are enough hints for you to reconstruct the code. Fill in the missing lines (shaded cells) below to save your friendship.

Answer:

Instruction Encoding	MIPS Code
	# **$s1** stores the result, **$t0** stores a non-negative number
	addi $s1, $zero, 0 #Inst. address is 0x00400028
0x00084042	**loop: srl $t0, $t0, 1**
0x11000002	beq $t0, $0, exit
0x22310001	addi $s1, $s1, 1
	j loop
	exit:

(b) Give a simple mathematic expression for the relationship between **$s1** and **$t0** as calculated in the code.

3. [AY2012/13 Semester 2 Assignment 3]
Your friend Alko just learned **binary search** in CS2040S and could not wait to impress you. As a friendly gesture, show Alko that you can do the same, but in MIPS! ☺

Complete the following MIPS code. To simplify your tasks, some instructions have already been written for you, so you only need to fill in the missing parts in **[ ]**. Please translate as close as possible to the original code given in the comment column. You can assume registers $s0 to $s5 are properly initialized to the correct values before the code below.

(a)

Variable Mappings	Comments
address of array[] → $s0	
target → $s1	// value to look for in array
low → $s2	// lower bound of the subarray
high → $s3	// upper bound of the subarray
mid → $s4	// middle index of the subarray
ans → $s5	// index of the target if found, -1 otherwise. Initialized to -1.

loop: slt $t9, $s3, $s2 bne $t9, $zero, end	#while (low <= high) {
add $s4, $s2, $s3 [srl $s4, $s4, 1]	# mid = (low + high)/ 2
sll $t0, $s4, 2 add $t0, $s0, $t0 [lw $t1, 0($t0)]	# t0 = mid*4 # t0 = &array[mid] in bytes # t1 = array[mid]
slt $t9, $s1, $t1 beq $t9, $zero, bigger	# if (target < array[mid])
addi $s3, $s4, -1 j loopEnd	# high = mid - 1
bigger: [slt $t9, $t1, $s1] [beq $t9, $zero, equal]	# else if (target > array[mid])
addi $s2, $s4, 1 j loopEnd	# low = mid + 1

equal:
add $s5, $s4, $zero
[j end]
	# else {
	ans = mid
	break
	# }

loopEnd: [j loop]	#) //end of while-loop
end:	

the sum of instructions

(b) What is the immediate value in decimal for the "**bne $t9, $zero, end**" instruction? 16
You should count only the instructions; labels are not included in the machine code.

(c) If the first instruction is placed in memory address at 0xFFFFFF00, what is the **hexadecimal representation** of the instruction "**j loopEnd**" (for "high = mid – 1")?

(d) Is the encoding for the second "**j loopEnd**" different from part (c)? If yes, give the new encoding, otherwise briefly explain the reason.

tut04qns (1)

Thursday, September 21, 2023 10:03 PM

tut04qns (1)

CS2100 Computer Organization
Tutorial 4: Data and Control Path
(Week 6: 18 Sep – 22 Sep 2023)

1. [Past-year's exam question]

   An ISA has 16-bit instructions and 5-bit addresses. There are two classes of instructions: class *A* instructions have one address, while class *B* instructions have two addresses. Both classes exist and the encoding space for opcode is completely utilized.

   (a) What is the minimum total number of instructions?

   (b) What is the maximum total number of instructions?

2. You have seen how **`blt`** ("branch less than") can be implemented in the lecture slides. MIPS assembly also allows for "pseudo-instructions" which the assembler will expand to multiple instructions to implement the functionality.Show how the following pseudo-instructions are implemented using real MIPS instructions:

   (a) **`bgt $r1, $r2, L`** ("bgt = branch greater than")

   (b) **`bge $r1, $r2, L`** ("bge = branch greater than or equal")

   (c) **`ble $r1, $r2, L`** ("ble = branch less than or equal")

   (d) **`li $r, imm`** ("load immediate" where the immediate can be any length up to 32-bits)

   (e) **`nop`** ("No operation", i.e., a null operation)

Questions 3 and 4 refer to the complete datapath and control design covered in lectures #11 and #12. Please use the diagram in Lecture #12 slide 29 or in the COD MIPS 4th edition textbook, Figure 4.17. For your convenience, Lecture #12 slide 29 is also included at the end of this tutorial sheet.

a. Maximize B

b.

review load upper immediate

3. Let us perform a complete trace to understand the working of the complete datapath and control implementation. Given the following three hexadecimal representations of MIPS instructions:

   i. `lw $24, 0($15)`
   ii. `beq $1, $3, 12`
   iii. `sub $25, $20, $5`

   For each instruction encoding, do the following:

   (a) Give the binary representation for the instruction.

   (b) Fill in the tables below. The first table concerns with the various data (information) at each of the datapath elements, while the second table records the control signals generated. Use the notation $8 to represent register number 8, [$8] to represent the content of register number 8 and Mem(X) to represent the memory data at address X.

Registers File				ALU		Data Memory	
RR1	RR2	WR	WD	Opr1	Opr2	Address	Write Data
$15	$24	$24	Mem[$15]	[$15]	0	[$15]+0	[$24]
$1	$3	$0/$3		[$1]	[$3]	([$1])-([$3])	([$8])
$20	[$5]	$25		[$20]	[$5]		

   [Wr = Write; Rd = Read; M = Mem; R = Reg]

RegDst	RegWr	ALUSrc	MRd	MWr	MToR	Brch	ALUop	ALUctrl
0	1	1	1	0	1	0	00	0
X	0	0	0	0	X	1	01	0110
1								

   (c) Indicate the value of the PC after the instruction is executed.

4. We shall now estimate the latency (time needed for a task) for the various type of instructions. Given below are the resource latencies of the various hardware components (ps = picoseconds = 10^{-12} second):

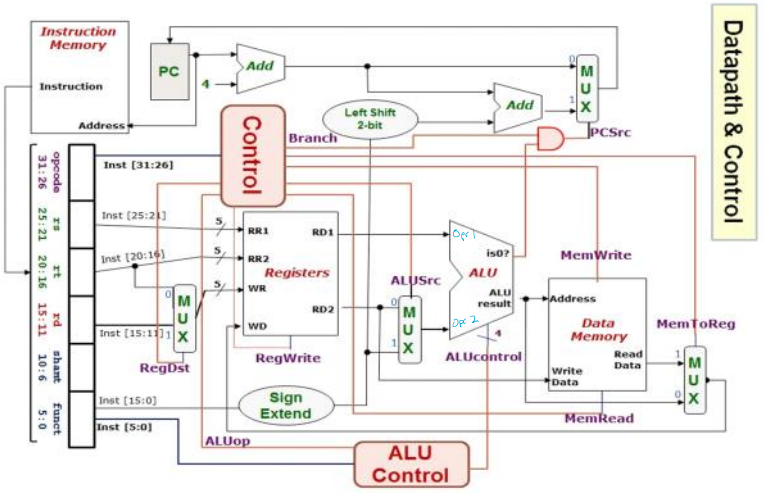
Inst-Mem	Adder	MUX	ALU	Reg-File	Data-Mem	Control/ ALUControl	Left-shift/ Sign-Extend/ AND
400ps	100ps	30ps	120ps	200ps	350ps	100ps	20ps

   Give the estimated latencies for the following MIPS instructions:

   (a) "SUB" instruction (e.g. `sub $25, $20, $5`)
   (b) "LW" instruction (e.g. `lw $24, 0($15)`)

   What do you think the **cycle time** should be for this particular processor implementation?

   Hint: First, you need to find out the **critical path** of an instruction, i.e. the path that takes the longest time to complete. Note that there could be several parallel paths that work more or less simultaneously



tut05qns

Thursday, October 5, 2023 6:37 PM

tut05qns

CS2100 Computer Organization
Tutorial 5: Control Path
(Week 6: 2 Oct – 6 Oct 2023)

For this tutorial, we will keep it simple and (like the textbook) just assume a MIPS processor that can handle only R-type, **`lw`**, **`sw`**, and **`beq`** instructions.

1. So far you have learnt C as well as MIPS programming. You are also starting to see the hardware implementation aspect of things. We will soon be starting on the journey to see how hardware implementations are done but whatever hardware does, it can be *described* in software. First, we need some preliminaries. There is a header file in C, called stdint.h, introduced by the C99 standard, that allows programmer to specify the exact bitwidths of integers. Among other things, it introduces the data type **`int8_t`**, **`int16_t`**, **`int32_t`**, **`int64_t`**, **`uint8_t`**, **`uint16_t`**, **`uint32_t`**, and **`uint64_t`** for signed 8, 16, 32, and 64 bit integers and *unsigned* 8, 16, 32, and 64 bit integers, respectively. In hardware description languages (like Verilog and VHDL – but not C), arbitrarily lengths are also supported. For the purpose of this tutorial, let's assume there is a **`int<N>_t`** and **`uint<N>_t`** type in C that also supports arbitrary length **`N`** signed and unsigned integers. As with **`int8_t`** etc, we shall trust that the compiler will "do the right thing" such as selecting the right assembly instructions to accomplish the operations required. Note that (just as a convention) **`N`** must be greater than 1. If **`N`** = 1, we will use **`bool`** (Boolean) data type instead.

   Going back to what is described earlier, we can use software (specifically, C in our case) to describe hardware and its function. So let's start with some of the components of the CPU. We can described instruction memory as an array:

```
uint32_t instruction_memory[1073741824];
```

   Yeah, the array size is huge but we are only doing a description here. Dealing with this large range in reality requires the help of hardware and the operating system, which you will gradually discover in your journey through SoC. Also, for convenience, even though we know that memory is an array of bytes, we will deal it with 32-bit at a time for simplicity since we don't need to deal with **`lb`** etc.

   Now:

   a) Write a C data structure and a function to describe the register file and its operation.

   b) Write a C data structure and a function to describe the data memory and its operation.

2. Next, let's deal with multiplexing (labelled as "MUX" in the slides). Ideally, we want to use some kind of template feature to describe it but C does not support templates (of course C++ does, but that's another story.) So show how two-way multiplexors ("MUX")

of different bitwidths can be instantiated using the macro expansion facility of C. In other words, each call to the macro should yield a C function that implements a two-way multiplexing function (i.e., "if selection is 0, output is input0, and if selection is 1, output is input1) where the inputs and output are of bitwidth **N**.

In particular, the following call to the macro:

```
MUX(32);
```

will instantiate a function definition:

```
int32_t mux_32(bool ctrl, int32_t in0, int32_t in1)
{
    if (ctrl) return in1;
    else return in0;
}
```

Assume the default is signed integer types. We can have another macro that instantiate unsigned integer types.

To do this question successfully, you will need to read up on C's macro *token pasting* feature: https://www.geeksforgeeks.org/stringizing-and-token-pasting-operators-in-c/.

3. Using C's **struct** data types, define a data type that will hold an instruction (let's call it "**struct insn**") of R-, I-, and J-type along with its sub-fields.
4. Main control is to be implemented by this function (we use pointers to pass multiple values out):

```
void Control(uint6_t opcode,
             bool    *_RegDst,
             bool    *_Branch,
             bool    *_MemRead,
             bool    *_MemtoReg,
             uint2_t *_ALUOp,
             bool    *_MemWrite,
             bool    *_ALUSrc,
             bool    *_RegWrite);
```

Provide the C code for computing the following signals that would be in this function:

a) **_RegDst**

b) **_ALUSrc**

c) **_MemRead**

d) **_ALUOp**

5. Write a C function that will model ALUControl:

```
uint4_t ALUControl(uint2_t _ALUOp,
                   uint6_t _funct);
```

ALUcontrol	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	slt
1100	NOR

Sw t0, 1(t5

6. Write a C function that will model the behavior of the ALU having the following function prototype:

```
int32_t ALU(int32_t in0, int32_t in1,
            uint4_t ALUControl, bool *ALUiszero);
```

where **in0** and **in1** are the 32-bit inputs, **ALUControl** is the 4-bit ALU control signals, and the outputs are the **ALUiszero** bit (passed by pointer) and the 32-bit result.

tut06qns

Thursday, October 12, 2023 10:22 PM

tut06qns

CS2100: Computer Organisation

Tutorial #6: Boolean Algebra, Logic Gates and Simplification

(Week 8: 9 – 13 October 2023)

Discussion Questions:

D1. (a) One common mistake that students make is the following: $A \cdot B + A' \cdot B' = 1$ … (equation 1)

This seems to be erroneously "derived" from the following rule: $X + X' = 1$.
Explain why the rule is wrongly applied here.

(b) Is the following equation correct? Why? $A \cdot B + (A \cdot B)' = 1$ … (equation 2)

D2. Given the following two 3-variable Boolean functions:

$$F(A,B,C) = \Sigma\, m(0, 2, 4, 6, 7)$$
$$G(A,B,C) = \Sigma\, m(1, 2, 3, 6)$$

(a) Write the product-of-maxterms expressions in ΠM notation for F and G.

(b) If $X = F + G$, write the sum-of-minterms expressions in Σm notation for X.

(c) If $Y = F \cdot G$, write the sum-of-minterms expressions in Σm notation for Y.

(d) If $Z = F \oplus G$, write the sum-of-minterms expressions in Σm notation for Z.

Do you know how to generalise the above for any arbitrary Boolean functions F and G?

[If it is not convenient to type symbols like Σ and Π in the forums, you may use Sum-m to mean Σm and Prod-M to mean ΠM. Example: Sum-m(0, 2, 4, 6, 7), Prod-M(2, 3, 5).]

D3. For each part below, how many prime implicants (PIs) and essential prime implicants (EPIs) are there in the K-map? What is/are the simplified **SOP expressions**? List out all alternative answers. [d(...) and D(...) denote don't-cares.]

(a) $F1(A,B,C,D) = \Sigma\, m(5, 8, 10, 12, 13, 14)$

(b) $F2(W,X,Y,Z) = \Pi\, M(0, 1, 2, 8, 9, 10)$

(c) $F3(K,L,M,N) = \Sigma\, m(1, 7, 10, 13, 14) + d(0, 5, 8, 15)$

(d) $F4(A,B,C,D) = \Pi\, M(4, 8, 9, 11, 12) \cdot D(2, 3, 6, 7, 10, 14)$

D4. For each of the functions in D3 above, find the simplified **POS expression**. List out all alternative answers, if any.

You are encouraged to do the above discussion questions and discuss them on Canvas or QnA. These are fundamental concepts that you must know, before you attempt the tutorial questions below.

Tutorial Questions:

1. The **consensus theorem** is given as

$$x \cdot y + x' \cdot z + y \cdot z = x \cdot y + x' \cdot z$$

Can you prove this using the laws and theorems of Boolean algebra given in class?
[Hint: Draw the K-map which will give you an idea which laws/theorems should be used.]

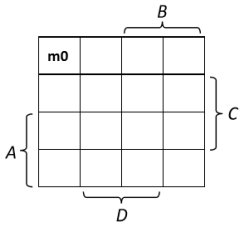
2. Using Boolean algebra, simplify each of the following expressions into simplified **sum-of-products (SOP) expression**. Indicate the law/theorem used at every step.

   (a) $F(x,y,z) = (x + y \cdot z') \cdot (y' + y) + x' \cdot (y \cdot z' + y)$

   (b) $G(p,q,r,s) = \Pi M(5, 9, 13)$

   [Tip: For (b), it is easier to start with the given expression and get done in about 5 steps, rather than to expand it into sum-of-products/sum-of-minterms expression first.]

3. (a) The following K-map layout is used for a 4-variable Boolean function $T(A,B,C,D)$. Fill in the minterm positions m1 to m15 into the respective cells. m0 has been filled for you.



   (b) Given the following 4-variable Boolean function:

$$T(A,B,C,D) = \Pi M(3,7,8,10,12,13) \cdot X(6,11,14,15)$$

   where X's are the don't-cares, write out the Σm notation for $T(A,B,C,D)$.

   (c) Draw the K-map for T using the layout above.

   (d) How many PIs (prime implicants) are there in the K-map? List out all the PIs.

   (e) How many EPIs (essential prime implicants) are there? List out all the EPIs.

   (f) What is the simplified SOP expression for T? List out all alternative solutions.

   (g) What is the simplified POS expression for T? List out all alternative solutions.

   (h) Implement the simplified SOP expression for T using a 2-level AND-OR circuit and a 2-level NAND only circuit, assuming that primed literals are not available.

Note: Always assume that prime literals are not available unless otherwise stated.

4. A circuit takes in four inputs K,L,M,N and generates 3 outputs X,Y,Z as follow:

   $X(K,L,M,N) = 1$ if $KL = MN$, or 0 otherwise,
   where KL and MN are 2-bit unsigned integers.

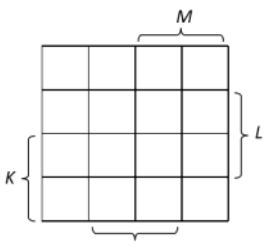
   $Y(K,L,M,N) = 1$ if $KL \leq MN$, or 0 otherwise,
   where KL and MN are 2-bit unsigned integers.

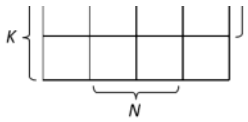
   $Z(K,L,M,N) = 1$ if $KLM < LMN$, or 0 otherwise,
   where KLM and LMN are 3-bit unsigned integers.

For parts (a) – (c) below, you may assume that the input 0000 will not occur.

(a) Fill in the truth table for the circuit. Write 'd' for don't cares.

(b) Fill in the K-maps of X, Y and Z using the layout given below.





(c) Write out the simplified SOP expressions of *X*, *Y* and Z.

(d) After designing the circuit according to the simplified SOP expressions in (c), if you feed the input 0000 into it, what will be the outputs?

midterm_21s1_qns

Friday, October 13, 2023 12:27 AM

midterm_21s1_qns

CS2100 2021/22 Semester I

Backup Midterm Paper

1. This paper is only to be used in the event of a serious LumiNUS failure. Otherwise this paper should be completed online in the LumiNUS Quiz Tool.

2. This is an open-internet paper. You can use any tool (compilers, code converters, online forums, etc.), as long as you **DO NOT COMMUNICATE WITH ANYONE NOR RECEIVE COMMUNICATIONS FROM ANYONE. IF YOU DO SO YOU WILL BE DEEMED TO HAVE CHEATED AND SERIOUS DISCIPLINARY ACTION WILL BE TAKEN AGAINST YOU.**

3. You may ask questions at this link: https://forms.gle/1vqjA8huxLfkh3MQA. Your questions must be phrased in a way that can be answered with a "YES" or a "NO", with no further elaboration. If your question is not appropriately phrased, you may be asked to rephrase, or you may be ignored.

4. This paper consists of FIFTEEN (15) questions on SEVEN (7) printed pages including. Marks are indicated, and total 40 marks, forming 20% of your final course grade.

5. You may complete this paper electronically on your tablet, print it out and write down your answers, or simply write your answers on a blank piece of paper. If you use a blank piece of paper, ensure that your answers are properly labeled so that we can identify which question (or which part of a question) your answer belongs to.

6. This paper is 60 minutes long.

7. **ENSURE THAT YOUR NAME, STUDENT NUMBER AND TUTORIAL GROUP NUMBER ARE CLEARLY WRITTEN ON YOUR ANSWER SHEET. IF YOU FAIL TO WRITE ANY OF THESE YOUR PAPER MAY NOT BE GRADED AND YOU WILL RECEIVE ZERO FOR THIS ASSESSMENT!**

8. At the end of the assessment, submit your answers as a PDF named TYY_Axxxxxx.pdf, where TYY is your tutorial group number, and Axxxxxx is your student number. **IF YOU SUBMIT USING AN INCORRECT NAMING, YOUR SCRIPT MAY NOT BE GRADED AND YOU WILL RECEIVE ZERO FOR THIS ASSESSMENT!**

9. IF LUMINUS IS AVAILABLE, Submit to your personal folder in the MIDTERMS->Txx folder, where Txx is your tutorial group number. If LumiNUS is not available, email your script, PROPERLY NAMED TYY-Axxxxxx.pdf, to cs2100.papers@gmail.com. Here TYY is your tutorial group number and Axxxxx is your student number. **REMINDER: IF YOU SUBMIT USING AN INCORRECT NAMING, YOUR SCRIPT MAY NOT BE GRADED AND YOU WILL RECEIVE ZERO FOR THIS ASSESSMENT!**

10. All rules in the Standard Operating Procedures are to be observed. Failure to do so may result in disciplinary action being taken against you.

NAME: _____ STUDENT NUMBER: _____ TUTORIAL GROUP: _____

1. We are given the following addition in an unknown base x (2 marks):

1231_x
$+ 3231_x$

10012_x

The unknown base x is base ___5___.

2. We are given the following subtraction in an unknown base y (2 marks):

3231_y
$- 1114_y$

2113_y

The unknown base y is base ___6___.

3. Fill in the unknown base w in the following conversion (2 marks):

$3237_w = 1695_{10}$

The unknown base w is base ___8___.

4. We have the following conversion from binary to an unknown base v (2 marks):

$101011101111_2 = 2567_v$

The unknown base v is base ___8___.

5. We consider a 24-bit signed number system in excess-1048576. In this number system, there are ___1048576___ negative numbers and ___15728640___ non-negative numbers (2 marks).

6. Find the 4-digit 3's complement representation in base 4, of the following decimal numbers. 3's complement in base 4 is the diminished radix complement for base 4 (4 marks).

$112_{10} =$ ___1300___ $_{3s}$
$-114_{10} =$ _____ $_{3s}$

7. Fill in the blank below in 3's complement (2 marks):

$112_{10} - 114_{10} =$ _____ $_{3s}$

Aiken has discovered that, for signed integers, the ">>" operator in C actually does an "arithmetic shift right" rather than a "logical shift right" like the srl operator in MIPS. What this means is that rather than filling the left bits with 0, the ">>" operator fills the vacated left bits with the sign bit.

On a 32-bit system, let's take:

```
int x = 0x80000000;
printf("%x\n", x); // Prints out 0x80000000, i.e. 0b1000 0000 .... 0000
printf("%x\n", x >> 1); // Prints out 0xc0000000, i.e. 0b1100 0000 ... 0000
printf("%x\n", x >> 2); // Prints out 0xe0000000, i.e. ob1110 0000 ... 0000
etc.
```

Aiken wants to write a function called "shift_right" which will do a logical right shift of signed integers instead of an arithmetic right shift. His best friend Dueet says that the best way to do this is to use "type-casting".

However, Aiken didn't learn "type-casting" in CS2100, so he wants to use bitwise logical operators like & and |. This is his attempt:

```
int right_shift(int val, int shift) {
// Returns the logical right shift of "val" by "shift" bits
    int mask1 = 0x80000000;
    int mask2 = *Mystery Instruction 1*

    return *Mystery Instruction 2*
}
```

8. What is "Mystery Instruction 1"? (2 marks):

   a. ~(mask1 >> shift);
   b. ~(mask1 << shift);
   c. (mask1 >> (shift-1));
   d. ~(mask1 >> (shift-1));
   e. ~(mask1 << (shift -1));

9. What is "Mystery Instruction 2"? (2 marks):

   a. (val & mask2);
   b. ((val >> shift) & mask2);
   c. ((val >> shift) && mask2);
   d. (val | mask2)
   e. ((val >> shift) | mask2)

We consider the MIPS assembly language program below, which processes two arrays, Array 1 and Array 2, both with the same number of elements. The outer loop reads from Array 1, the inner loop reads from Array 2.

```
        addi $5, $zero, 0
        addi $6, $2, 0
        addi $7, $3, 0
        sll $8, $4, 2
        add $9, $2, $8
        add $10, $3, $8
a:      slt $11, $6, $9
        beq $11, $zero, e
        lw $11, 0($6)
b:      slt $12, $7, $10
        beq $12, $zero, d
        lw $12, 0($7)
        bne $11, $12, c
        addi $5, $5, 1
c:      addi $7, $7, 4
        j b
d:      addi $6, $6, 4
        addi $7, $3, 0
        j a
e:
```

Within each array, numbers are never repeated. So Array 1 might hold 10, 15, 12, 92, 5, while Array 2 might hold 92, 3, 101, 10, 78. However you will not have the case where, for example, Array 1 holds 10, 15, 12, 10, 5, because here 10 is repeated.

10. Choose ALL the statements that are correct about this program (2 marks).

a. Register $2 and $3 contain the base addresses of the two arrays.
b. Register $8 contains the number of elements in Array 1.
c. Register $6 contains the address of the current element being processed from one of the arrays.
d. The inner and outer loops run the same number of times.
e. Register $9 contains the address of the final element of Array 1 to be loaded.

11. What is the MINIMUM number of instructions executed, if this program processes all the elements in two 10-element arrays? Fill your answer here: _____ (2 marks).

12. What is the MAXIMUM number of instructions executed, if this program processes all the elements in two 10-element arrays? Fill your answer here: _____ (2 marks).

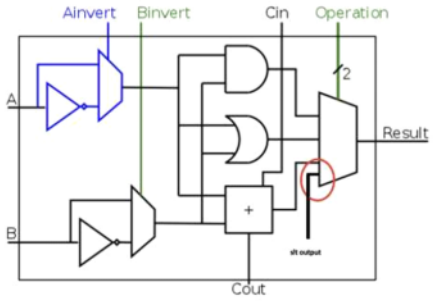
13. What does $5 contain at the end of the execution of this program (2 marks)?

a. $5 contains the number of elements in Array 1 that are also in Array 2.
b. $5 contains the number of elements that are in Array 1 but not in Array 2.
c. $5 contains the number of elements that are in Array 2 but not in Array 1.
d. $5 contains the differences between elements in Array 1 and Array 2.
e. $5 returns the sum of the elements in Array 1 and Array 2.

14. The MIPS slt instruction takes the form slt $rd, $rs, $rt, where [$rd] = 1 if [$rs] < [$rt], and 0 otherwise.

We recall that the full 32-bit ALU is made up of single-bit ALU slices, with the carry-out (Cout) of bit 0 connected to the carry-in (Cin) of 1, the Cout of bit 1 connected to Cin of bit 2, etc.

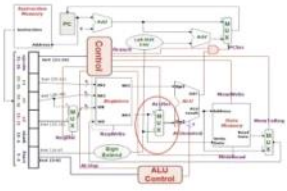
We will now implement slt using the last input (input 0b11) of the output multiplexer of each ALU slice. For brevity we will call this input "Input ob11 of the bitX ALU slice", where X is from 0 to 31. Input 0b11 for one slice is circled below.

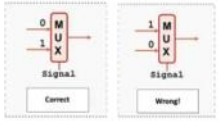


Choose ALL of the options needed to implement the slt instruction (4 marks):

a. Set ALUControl to 0b0011.
b. Set the 0b11 input of the ALU slice for bit31 to the output of the adder of the bit 0 ALU slice.
c. Set ALUControl to 0b0110.
d. Set the 0b11 input of the ALU slice for bit 0 to the output of the adder of the bit31 ALU slice, and the 0b11 inputs of the remaining ALU slices to 0.
e. Set ALUControl to 0b1011
f. Set ALUControl to 0b0111
g. Set the 0b11 input of the ALU slice for bit 0 to 1, and the 0b11 inputs of the remaining ALU slices to 0.
h. Set the 0b11 inputs of all ALU slices to the output of the adder of the bit 31 ALU slice.

15. In one of our tutorial questions, Mr. De Blunder accidentally swapped the inputs of the RegDst multiplexer. That problem has since been fixed. Unfortunately, now Mr. De Blunder accidentally swapped the inputs of the ALUSrc multiplexer!





Each register has been initialized to contain its own register number. Example: register $1 contains 1, register $2 contains 2, register $3 contains 3, etc.

Find the results, in decimal, of the following instructions (2 marks each, total 8 marks):

a. addi $3, $2, 8192

The result 5 will be written to register $3.

b. sub $4, $5, $6

The result -8221 will be written to register $4.

c. andi $7, $8, 0b1111

The result 0 will be written to register $7.

d. add $9, $10, $11

The result 18474 will be written to register $9.

Assignment 2

Saturday, October 14, 2023 4:22 PM

Assignment 2

CS2100 Computer Organization
AY2023/24 Semester I
Assignment 2
Due: Monday, 16 October 2023, 1pm

Instructions (PLEASE READ CAREFULLY!)

1. There are FOUR (4) questions in this assignment, totaling THIRTY-SEVEN (37) marks. Please do all parts of every question. Marks are indicated against each part.

2. An additional 3 marks is awarded for putting in your name, student ID and tutorial group number in your answers and for submitting in PDF format with the correct naming convention (see below). Thus, the total will be FORTY (40) marks. If you fail to do any of these, you will lose the 3 marks.

3. This assignment is due on **Monday, 16 October 2023, 1 pm.** You will be given until 1.15 pm to submit, **after which no submission will be accepted and you will receive ZERO for this assignment, regardless of how hard you've worked on it.**

4. **Plan to submit at least 2 hours early (i.e., by 11 am on 16 October)** in case there are technical issues with submission. If, by 1 pm on 16 October you are unable to submit due to technical reasons, please email your answers to Weng Fai at wongwf@nus.edu.sg before 1.15 pm. **No submission over email will be accepted after 1.15 pm.**

5. Complete your answers on the provided **CS2100Assg2AnsBk.docx** file. Save it as AxxxxxxY.pdf before submitting. For the programming part, you are required to describe the corresponding parts of your code both for documentation as well as assessing your thought process in designing it.

6. **Zip your AxxxxxxxY.pdf file together with your mips.c (your C code must use this name) file from the programming parts into a file called AxxxxxxxY.zip, and submit that on Canvas. Note that while the answer booklet is supplied as a Word document so that you have more flexibility space-wise, you must submit a PDF.** You will forfeit the 3 marks if you do not do all these, or if you fail to fill in your name, student ID and tutorial group number. **In addition if you do not submit your `mips.c` file, you will not receive marks for the programming portion of the Assignment.**

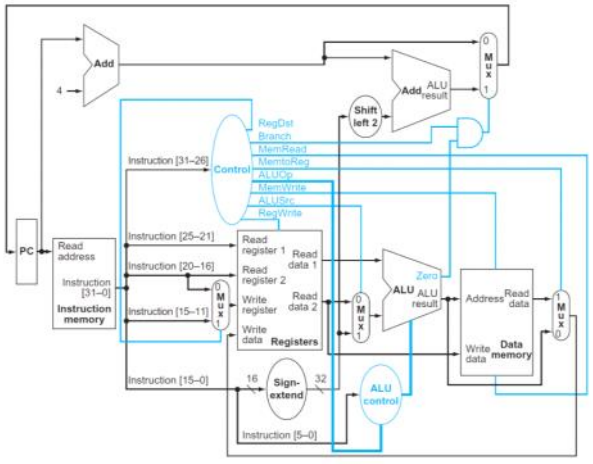
7. **`mips.c`** must be entirely compilable using one single command, namely:
   ```
   gcc -o mips main.c mips.c
   ```
   In particular, **no** additional file or library may be used.

8. You should do these assignments on your own. Do not discuss the assignment questions with others. Note that the NUS/SoC policy on plagiarism shall apply in full. Severe penalty may result in its violation.

9. Please use the Canvas Discussion Forums for clarifications.

Part A – Data and Control Paths

We shall finish what we started in Tutorial 5. In other words, you are to write a C simulator for the simple MIPS data and control path.



In particular, your implementation should conform to the following constraints:

1. Besides this question paper, you will be provided with three C files – **`main.c`**, **`mips.h`**, and **`mips.c`**. Modify/add to only the parts marked out in mips.c. All other parts of the code should not be modified. You will be penalized for touching parts you are not supposed to.
2. Your processor only needs to implement the following instructions:
   `add`, **`and`**, **`beq`**, **`lw`**, **`sw`**, **`nor`**, **`or`**, **`slt`**, **`sub`**.
3. Instead of the arbitrary **`intN_t`** and **`uintN_t`** we used in Tutorial 5, you can only use the bit lengths predefined in **`stdint.h`**, i.e., **`int8_t`**, **`uint8_t`**, **`int16_t`**, **`uint16_t`**, **`int32_t`**, **`uint32_t`**, as well as the **`bool`** type in **`stdbool.h`**.
4. Unlike in Tutorial 5, the data memory (instruction memory is not needed since our simulator for this assignment only works with a single instruction as input) is at most 1024 integers (4096 bytes). Any address that goes beyond is raised as an error.
5. The executable should be called with a single MIPS instruction in hexadecimal, for example:
   `./mips 0x014B4820`
   which is "**`add $t1, $t2, $t3`**".

Question 1. (7 MARKS)

(a) Complete the implementations of the three multiplexor functions found in `mips.c`. They may be needed later. [3 marks]

(b) Complete the implementation of the `void decode(uint32_t in, struct instr *insn)` function found in `mips.c`. This function will take in a 32-bit unsigned integer encoding of a valid MIPS instruction and then fill out the `struct insn` whose address is also passed in as a pointer. Note that here (unlike in Tutorial 5) the purpose of this structure is to capture all the relevant information about instruction. Hence the binary will be decoded by interpreting the encoding in all three types of instructions. [4 marks]

Question 2. (10 MARKS)

(a) Complete the implementation for `Control()` function found in line 95 of the given (original skeleton) `mips.c`.. This should be simple given Tutorial 5. Just want to check that you were paying attention. [3 marks]

(b) Complete the implementation for `ALUControl()` function found in line 105 of the given (original skeleton) `mips.c`. [3 marks]

(c) Complete the implementation for the `ALU()` function found in line 114 of the given (original skeleton) `mips.c`. [4 marks]

Question 3. (10 MARKS)

Complete the implementation of `execute()` function found in line 123 of the given (original) `mips.c` which will complete your simulator. Given an instruction, `execute()` will simulate 4 of the 5 stages of instruction processing, naming decode (and operand fetch), execute, memory, and write-back. At the end of `execute()`, the contents of the register file, memory and PC should be as you would expect as per our lectures. Your code will be tested against a suite of 10 hidden tests consisting of instruction encoding for the instructions mentioned above. Note that not only will the contents of registers, memory and PC be checked against that expected of the test cases but also the structure of `execute()` should mimic the data and control path diagram given above. The idea being that you should be simulating, as faithfully as possible, how instructions are executed in this hardware. Hopefully this will complete your understanding of the various aspects of this (simple) processor.

Note that `execute()` does not return any result directly. Instead, it changes the global state of the program including the register file, memory and the PC. These will be checked – as a whole, not just the parts affected by the test case - for correctness during grading when we execute your code.

[10 marks]

Part B – Boolean Algebra

Question 4. (10 MARKS)

Note that unless otherwise stated, complemented literals are not available.

(a) What is the minimum number of 2-input NAND gates required to implement XOR(a,b), where a and b are Boolean variables? Illustrate your answer with a neatly drawn logic diagram, with inputs and output clearly labelled. Mark will be deducted if the inputs and output are not properly labelled. [2 marks]

(You do not need to provide the proof, but you are advised to trace your logic circuit to check its correctness yourself.)

(b) A chessboard consists of 8 × 8 squares, as shown in the diagram on the right where the squares are labelled from 0 through 63. We may represent the labels as 6-bit binary numbers.

0	1	2	3	4	5	6	7
15	14	13	12	11	10	9	8
16	17	18	19	20	21	22	23
31	30	29	28	27	26	25	24
32	33	34	35	36	37	38	39
47	46	45	44	43	42	41	40
48	49	50	51	52	53	54	55
63	62	61	60	59	58	57	56

In one move, a bishop can move any number of squares diagonally; up to the edge of the board. If it is on a grey square, it may land on any grey square, in the course of a game, but it will never land on a white square. Likewise, a bishop that is on a white square will never land on a grey square.

Assume that the binary numbers $A_5A_4A_3A_2A_1A_0$ and $B_5B_4B_3B_2B_1B_0$ represent two labels. Define a Boolean function $S_k(A_k, B_k)$ that outputs 1 if $A_k = B_k$, or 0 otherwise, where $0 \le k \le 5$. Write out the simplified Sum-of-Products (SOP) expression for S_k. [1 mark]

(Use the symbols we introduced in class: ' for NOT, · for AND, + for OR. Do not use other symbols.)

(c) Given two distinct labels $A_5A_4A_3A_2A_1A_0$ and $B_5B_4B_3B_2B_1B_0$, define a Boolean function $F(A_5A_4A_3A_2A_1A_0, B_5B_4B_3B_2B_1B_0)$ that outputs 1 if a bishop at $A_5A_4A_3A_2A_1A_0$ is able to move to $B_5B_4B_3B_2B_1B_0$ in two or fewer moves, or 0 otherwise.

Write out the simplified SOP expression for F in terms of S_k. [1 mark]

(d) Given the 4-variable Boolean function $G(P,Q,R,S) = \prod M(1,8,9,10,12,14) \cdot \prod X(2,5,6,15)$ where M are the maxterms and X the don't-cares, what are the prime implicants on the K-map of G? Write out all the prime implicants. You do not need to show your K-map. [1 mark]

(e) Given the 4-variable Boolean function $G(P,Q,R,S) = \prod M(1,8,9,10,12,14) \cdot \prod X(2,5,6,15)$ in part (d) above, what are the essential prime implicants on the K-map of G? Write out all the essential prime implicants. You do not need to show your K-map. [1 mark]

(f) Given the 4-variable Boolean function $G(P,Q,R,S) = \prod M(1,8,9,10,12,14) \cdot \prod X(2,5,6,15)$ in part (d) above, write out the simplified Product-of-Sums (POS) expression for G. [2 marks]

(g) Given the 5-variable Boolean function

$$H(A,B,C,D,E) = \sum m(5,8,13,16,18,29) + \sum X(0,7,15,23,24,26,31)$$

where m are the minterms and X the don't-cares, write out the simplified Sum-of-Products (SOP) expression for H. You do not need to show the K-map or working. [2 marks]

=== END OF PAPER ===

5

questio

Tuesday, October 17, 2023 2:30 AM

Question 4

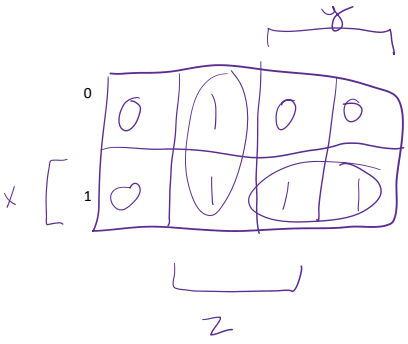
1 / 1 pts

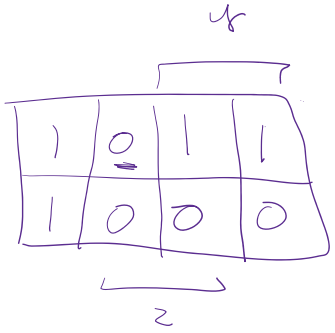
We have the following function:

g(x, y, z) = (x + y').(y+ z)

Which of the following sums are NOT in the canonical product of sums for g?

- [x] (x' + y' + z)
- [ ] (x + y' + z)
- [ ] (x + y + z)
- [ ] (x' + y + z)
- [ ] (x + y' + z')





Invert the sop from the inverted kmap

$z \cdot y' + y \cdot x$

(x*z'*y') + (x'*z'*y') + (x'*z*y) + (x'*z'*y)
(X' + Z + Y) * (X + Z + Y) * (X + Z' + Y') * (X+Z+Y')

Midterms-Check_release

Wednesday, October 18, 2023 1:00 AM

Midterms-Check_rele...

CVS2100 Computer Organization
2023/24 Semester I
Midterm Assessmetn
ANSWER KEY

1. We are given the following C program fragment:

```
uint16_t x = 77;
uint16_t y, m = 1;

…
y = x;
x = 0;

while(y) {
   if(y & 1) {
        x = x | m;
        m = m << 1;
   }
   y = y >> 1;
}
```

What is the value in x in binary after this code fragment is executed? Leave out leading 0's in your answer, and do not add the 0b prefix. For example, 0b0000000001100000 should be entered as 1100000.

Explanation: Every time the LSB of y is 1, we OR in a 1 into x using mask m initially set to 0b0000000000000001, but shifted left after each OR. So the first 1 bit in y, we will OR in this mask into x setting x = 0b1. Then m is shifted left to give 0b0000000000000010, and y is shifted right again. The next time the LSB is 1, x = x | m = 0b11, and m is shifted left to give 0b0000000000000100, etc. Effectively the number of bits in x set to 1 is equal to the number of 1 bits in y.

Initially y=77, 0b01001101. So there are 4 1-bits in y.

Answer: 0b1111

Crafted and presented a mock launchpad for a portfolio manager scenario using Bloomberg Terminal.

2. Suppose we are given the following C code fragment:

```
uint16_t x=5, y=10;
uint8_t *p;

…
p = (uint8_t *) &x; // Casts the pointer to x into a
                    // pointer to an 8-bit unsigned number
p+=1;
*p=23;
p+=1;
*p=129;
…
```

Assuming that y is stored immediately after x in memory, what are the values of x and y after the code fragment shown above is executed completely on a big-endian computer with byte-addressable memory? Write your answer in decimal.

Explanation: Initially x=5 and y=10. Our system is big endian so this would be stored as (smallest address on the left):

x		y	
0000 0000	0000 0101	0000 0000	0000 1010

Initially p is set to the second byte of x, which is replaced with 23:

x		y	
0000 0000	0001 0111 (p)	0000 0000	0000 1010

p is then advanced by 1 position, putting it at the first byte of y. This is then replaced with 129:

x		y	
0000 0000	0001 0111	1000 0001 (p)	0000 1010

x is thus 0b0000 0000 0001 0111 = 23, and y is 1000 0001 0000 1010 = 33034

x = 23
y = 33034

3. $3125_7 + 4162_7 = 10320_7$

4. $2042_5 - Y_5 = 213_5$. What is Y? Y = 1324_5

5. Aiken encoded a number X in 32-bit IEEE-754 format, but mistakenly used excess-128 instead of excess-127. This is the encoding he obtained with his error:

X = 0x4209A000 0100 0010 0000 1001 1010 0000 0000 0000

What is this number X in base 10? X = 17.203125

Working:
0x4209A000 = 0100 0010 0000 1001 1010 0000 0000 0000
0 10000100 00010011010000000000000
= 0 10000100 00010011010000000000000
Exponent = 132 -128 = 4
Number = 1.0001001101 x 2^4 = 10001.001101
= 17.203125

What is the correct encoding for X in 32-bit IEEE-754 format, in hexadecimal? Leave out leading 0's and do not include the 0x prefix. For example, an answer of 0x0034EF should just be entered as 34EF.

Sign = 0
Mantissa = 00010011010000000000000
Exponent = 4 + 127 = 131 = 10000011

0100 0001 1000 1001 1010 0000 0000 0000
=0x4189A000

X in IEEE-754 format is : 0x 4189A000

6. We are given the following MIPS code. Register $s0 contains the value 2100.

```
     addi $t0,$zero,0x20  # Loaded at address 0x0000FC04
     addi $t1,$zero,0
     addi $t2,$s0,0
     addi $s1,$zero,0
L:   beq  $t1,$t0,E
     andi $t3,$t2,1
     bne  $t3,$zero,S
     addi $s1,$s1,1
S:   addi $t1,$t1,1
     srl  $t2,$t2,1
     j L                  # This j instruction
E:   ...
```

a. At the completion of this MIPS code, how many instructions would have been executed?

Within the loop: 32 x 7 + 1 = 225 instructions
The add $s1, $s1, 1 instruction is skipped every time a '1' occurs. So it is skipped 4 times.
# of instructions executed in the loop = 225 – 4 = 221

Before the loop = 4
Total = 225 instructions

Counter-check: All 7 instructions are executed 28 times, and 6 instructions executed 4 times, and the beq $t1, $t0, E is executed one last time.

Total in the loop = 28 x 7 + 4 x 6 + 1 = 221 instructions. Add in the 4 instructions before the loop giving us 225.

Answer: 225

Code counts # of 0's in 2100. 2100 in binary
= 0b0000 0000 0000 0000 1000 0011 0100
There are 28 bits that are 0's.

b. What is the value in $s1 in decimal at the end of the run? Answer: 28
c. What is the value in $t2 in decimal at the end of the run? Answer: 0

We assume that the above program is loaded at address 0x0000FC04. Answer the following questions:

d. What is the encoding for the beq instruction at label L? Write your answer in hexadecimal, and exclude any leading 0's, as well as the 0x prefix. For example, if your answer is 0x00134A67, write it as 134A67.

beq $t1, $t0, E. Immediate value = 6. Rs=$t1 = $9, rt=$t0 = $8

Encoding = 000100 01001 01000 0000 0000 0000 0110
= 0001 0001 0010 1000 0000 0000 0000 0110
= 0x11280006

e. What is the encoding for the jump instruction marked "# This J instruction"? Write your answer in hexadecimal, and exclude any leading 0's, as well as the 0x prefix. For example, if your answer is 0x00134A67, write it as 134A67.

Address at L = 0xFC04 + 4 x 4 = 0xFC14
= 0000 0000 0000 0000 1111 1100 0001 0100
= 0000 0000 0000 1111 1100 0001 01
Full encoding = 000010 0000 0000 1111 1100 0001 01
= 0000 1000 0000 0011 1111 0000 0101
= 0x08003F05
= 0x8003F05

7. We are given the following MIPS machine code involving register $3 and other registers. Each 32-bit hexadecimal number is a single instruction:

0x20020000 addi $2, $0, 0
0x20030000 addi $3, $0, 0
0x00621820 add $3, $3, $2
0x20420001 addi $2, $2, 1
0x2844000A (See note) slti $4, $2, 10
0x1480FFFC bne $4, $0, L

What is the value in $3 when this program completes execution?

Answer:

	addi $2, $zero, 0	0b0010 0000 0000 0010 0000 0000 0000 0000 = 0x20020000
	addi $3, $zero, 0	0b0010 0000 0000 0011 0000 0000 0000 0000 = 0x20030000
L:	add $3, $3, $2	0b0000 0000 0110 0010 0001 1000 0010 0000 = 0x00621820
	addi $2, $2, 1	0b0010 0000 0100 0010 0000 0000 0000 0001 = 0x20420001
	slti $4, $2, 10	0b0010 1000 0100 0100 0000 0000 0000 1010 = 0x2844000A
	bne $4, $zero, L	0b0001 0100 1000 0000 1111 1111 1111 1100 = 0x1480FFFC

Note: In the original question this statement was set as 0x2848000A, which is slti $8, $2, 10. The result is that the code would not execute as intended.

We thus also accept answers like "Infinite loop" or "Unknown", together with the original intended answers.

$3 contains 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 +8 + 9 = 45
How many instructions would have been executed by the time this program exits? Answer:
10 x 4 + 2 = 42

8. We are given a machine that has two classes of instructions called Class A and Class B. All instructions are 16 bits long. There are 5 registers and a total of 12 memory locations. Class A instructions have one register and one address operands, while class B instructions have one register and two address operands

For technical reasons, instructions cannot begin with 0b011 and 0b0101. This means that any instruction words in the form of 0b011x xxxx xxxx xxxx and 0b0101y yyyy yyyy yyyy are invalid.

Instruction formats:

# of register bits = $\log_2 5 = 3$, # of address bits = $\log_2 12 = 4$
Type A: Opcode length = 16 – 3 – 4 = 9 bits
Type B: Opcode length = 16 – 3 – 4 – 4 = 5 bits

What is the maximum number of instructions on this machine if the encoding space is fully utilized, and there is at least one instruction in each class?

Maximize A:

011x cannot be used. # of opcodes lost = 011x xxxxx = 2^4 = 64
0101y cannot be used, # of lost opcodes = 2^5 = 32
Set aside 11111 for Type B, opcodes lost = 16

Total class A = 2^9 – 64 – 32 – 16 = 400 opcodes
# of class B = 1
Total = 401 opcodes.

What is the minimum number of instructions on this machine if the encoding space is fully utilized, and there is at least one instruction in each class?

Maximize B:
011x cannot be used. # of opcodes lost = 4
0101y cannot be used. # of opcodes lost = 2
Set aside 11111 for class A. # of opcodes = 16

Total number of B opcodes = 2^5 – 4 – 2 – 1 = 25
Total number of opcodes = 25 + 16 = 41.

9. What is the final result on the stack after executing this stacked-based program? State your answers in decimal.

PUSH 7
PUSH 4
SUBTRACT
PUSH 2
PUSH 6
ADD
MULTIPLY
PUSH 3
ADD

Instruction	Stack (Grows left to right)
PUSH 7	7
PUSH 4	7 4
SUBTRACT	3
PUSH 2	3 2
PUSH 6	3 2 6
ADD	3 8
MULTIPLY	24
PUSH 3	24 3
ADD	27

Answer: 27

Six MCQs:

ADD	27
MULTIPLY	
PUSH 3	
ADD	

Answer: 27

Six MCQs:

For the questions below, assume the following register contents. All values are in hexadecimal.

```
R0  (r0) = 0x00000000 | R1  (at) = 0x00002000
R2  (v0) = 0x00000001 | R3  (v1) = 0x0000000a
R4  (a0) = 0x00000005 | R5  (a1) = 0x7ffff000
R6  (a2) = 0x7ffff004 | R7  (a3) = 0x000000b0
R8  (t0) = 0x00000001 | R9  (t1) = 0x00000c00
R10 (t2) = 0x0000c000 | R11 (t3) = 0xfffffff0
R12 (t4) = 0xf0000000 | R13 (t5) = 0x00000fff
R14 (t6) = 0x00006200 | R15 (t7) = 0x00000e00
R16 (s0) = 0x00300000 | R17 (s1) = 0x00000c00
R18 (s2) = 0x00040200 | R19 (s3) = 0x00011000
R20 (s4) = 0x00030200 | R21 (s5) = 0x10000000
R22 (s6) = 0x00055000 | R23 (s7) = 0xf0000000
R24 (t8) = 0x00000005 | R25 (t9) = 0x0000d000
R26 (k0) = 0x00000000 | R27 (k1) = 0x00000000
R28 (gp) = 0x10008000 | R29 (sp) = 0x7fffeff4
R30 (s8) = 0x1000000f | R31 (ra) = 0x00400018
```

10. Consider the following control signals and their values:
    RegDst = 1
    RegWrite = 1
    ALUSrc = 0

    Which of the following MIPS instructions would have these same control signals?
    (i) **add**
    (ii) **addi**
    (iii) **sub**
    (iv) **subi**
    (v) **and**
    (vi) **beq**

    Answer:

(A) (i) only
(B) (i) and (ii) only
(C) (i) and (iii) only
(D) (i), (iii) and (v) only
(E) None of the above

11. Consider the following MIPS instruction:

```
beq $t0, $zero, else
```

    If this instruction is located at the 4 bytes starting at **0x104** and **else** is at the location **0x70**. What is the value of the 16-bit immediate that is the instruction?

    Answer:
    (A) **0x0034**
    (B) **0xff34**
    (C) **0xff60**
    (D) **0xff68**
    (E) **None of the above (A) to (D) are correct.**

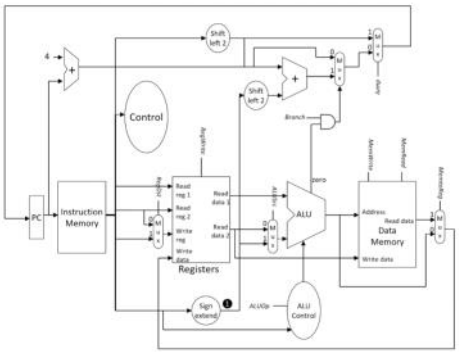
12. Suppose the PC currently has the value of **0x0ffffffc** and is pointing at the following instruction. What is the value of PC after the execution the instruction?

```
0x0bffffff
```

    Answer:
    (A) **0x0ffffffc**
    (B) **0x0fffffff**
    (C) **0x1ffffffc**
    (D) **0xfffffff0**
    (E) **None of the above (A) to (D) are correct.**

13. What is the destination register and its value after the execution of an instruction whose binary encoding is **0x30d5f8f8**.
    Answer:
    (A) **$s5 = 0x0000f000** (andi $21, $6, 0xf8f8)
    (B) **$t0 = 0x10101011**
    (C) **$v1 = 0x00000004**
    (D) **$a1 = 0x0000f8f8**
    (E) **None of the above (A) to (D) are correct.**

14. Assuming the same register setup as before, consider the following datapath diagram:



What is the value at ❶ during the execution of the following instruction:

```
sub $t0, $a1, $s5
```

Answer:
(A) 0x00000000
(B) 0xffffffff
(C) 0x00004022
(D) 0x00008511
(E) None of the above (A) to (D) are correct.

15. For the situation in Question 5, which of the following statement is TRUE?

Answer:
(A) The ALU result will be incorrectly written into the destination register.
(B) It will cause the PC to start fetching from some incorrect address in memory. This will most likely lead to the program eventually crashing.
(C) The ALU will correct the calculation by adding an offset to the instruction.
(D) The control signals of ALUsrc and Branch will cause the corresponding multiplexors to safely ignore it
(E) None of the above (A) to (D) are correct.

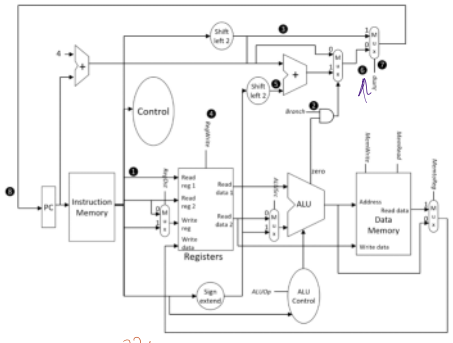
16. For the following MIPS instruction:

```
addi $s0, $t9, -9
```

what are the respective control signals? Give your answers as binary string of the correct length. Note that a string of the wrong length will only get partial marks at best.

Opcode:	001000
rs:	11001
rt:	10000
rd:	11111
immed:	1111 1111 1111 0111
RegDst:	0
RegWrite:	1
ALUSrc:	1

17. Consider the following datapath with markers placed at certain wires.



Now suppose that PC is **0x100** and the instruction is:

0x08000043

Write down as binary strings of the correct length, the values present at the various points in the CPU. Note that a string of the wrong length will only get partial marks at best.

❶	00000
❷	0
❸	0000 0000 0000 0000 0000 0001 0000 1100
❹	0
❺	0000 0000 0000 0000 0000 0001 0000 1100
❻	0000 0000 0000 0000 0000 0001 0000 0100
❼	1
❽	0000 0000 0000 0000 0000 0001 0000 1100

11

tut07qns

Thursday, October 19, 2023 7:06 PM

tut07qns

CS2100: Computer Organisation

Tutorial #7: Combinational Circuits

(Week 9: 16 – 20 October 2023)

Discussion Questions

D1. Design a circuit, without using any logic gate, that takes a 3-bit input *ABC* representing an unsigned integer *x*, and produces a 5-bit output *VWXYZ* which is equivalent to 4*x*+2. What are *V*, *W*, *X*, *Y* and *Z*?

D2. The following algorithm to convert binary to standard Gray code sequence is given in "Digital Logic Design" book, page 35:

1. Retain the MSB.
2. From left to right, add each adjacent pair of binary code bits to get the next Gray code bit, discarding the carry.

The following example shows the conversion of binary number $(10110)_2$ to its corresponding standard Gray code value, $(11101)_{Gray}$.

1	0	1	1	0	Binary
↓					
1					Gray

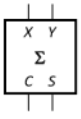
1	+	0	1	1	0	Binary
		↓				
1		**1**				Gray

1	0	+	1	1	0	Binary
			↓			
1	1		**1**			Gray

1	0	1	+	1	0	Binary
				↓		
1	1	1		**0**		Gray

1	0	1	1	+	0	Binary
					↓	
1	1	1	0		**1**	Gray

Given a half-adder as shown on the right where *X* and *Y* are its inputs and *C* (carry) and *S* (sum) its outputs, implement a 5-bit binary to Gray code converter to convert the binary value *ABCDE* to its equivalent Gray code *PQRST* by using the fewest number of half-adders without any additional logic gates.



D3. [Past year's question]

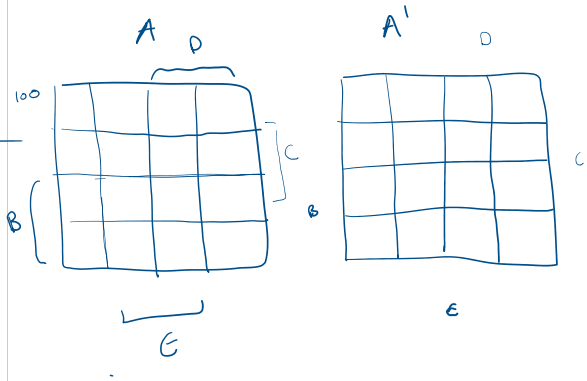
A combinational circuit takes in a 5-bit input *ABCDE* and generates a 2-bit value *PQ* such that *PQ* represents the *distance* between the two closest 1s in the input. The distance is defined to be the number of 0s between the two closest 1s.

For example, if *ABCDE* is 01011, then the distance between the two closest 1s (the two rightmost 1s) is zero, therefore, *PQ*=00. If *ABCDE* is 10010, then the distance between the two closest 1s is 2, therefore, *PQ*=10.

You may assume that the distance is always determinable from the given input. Therefore, inputs such as 00000 and 01000 will not be supplied to this circuit.

Draw the K-maps for *P* and *Q* and write the simplified SOP expressions for *P* and *Q*.

Using the simplified SOP expressions for *P* and *Q* to implement the circuit, what is the output if the circuit is fed with the input *ABCDE*=00100?

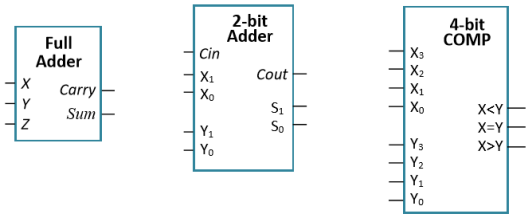


Tutorial questions

Note that for questions on logic design, you may assume that logical constants 0 and 1 are always available. However, complemented literals are not available unless otherwise stated.

1. [Past-year's question]

   You are to design a circuit to implement a function *V*(*A*,*B*,*C*,*D*,*E*) that takes in input *ABCDE* and generates output 1 if *ABCDE* is a valid input for the circuit in question D3 above, or 0 if *ABCDE* is an invalid input. You are allowed to use only the following devices: full adder, 2-bit parallel adder, and 4-bit magnitude comparator. You should use the fewest number of these approved devices, and no other devices or logic gates. The block diagrams for these devices are shown below.



2. [Past year's exam question]

a. You want to construct a circuit that takes in a 4-bit unsigned binary number *ABCD* and outputs a 4-bit unsigned binary number *EFGH* where *EFGH* = (*ABCD* + 1) / 2. Note that the division is an integer division. For example, if *ABCD* = 0110 (or 6 in decimal), then *EFGH* = 0011 (or 3 in decimal). If *ABCD* = 1101 (or 13 in decimal), then *EFGH* = 0111 (or 7 in decimal).

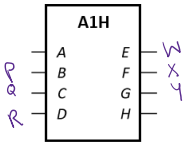
Construct the above circuit using a single **4-bit parallel adder** and at most one logic gate with no restriction on its fan-in.

b. The following table shows the 4221 code and 8421 code (also known as BCD code) for the ten decimal digits 0 through 9.

Digit	4221 code	8421 code
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0110	0100
5	1001	0101
6	1100	0110
7	1101	0111
8	1110	1000
9	1111	1001

You want to construct a 4221-to-8421 decimal code converter, which takes in a 4-bit 4221 decimal code *PQRS* and generates the corresponding 4-bit 8421 decimal code *WXYZ*.

Let's call the circuit you created in part (a) above the A1H (Add-1-then-Half) device, represented by the block diagram below. Implement your 4221-to-8421 decimal code converter using this A1H device with the fewest number of additional logic gates.



3. [Past year's exam question]
The BCD code (also known as 8421 code) values for the ten decimal digits are given below:

Digit:	0	1	2	3	4	5	6	7	8	9
Code:	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

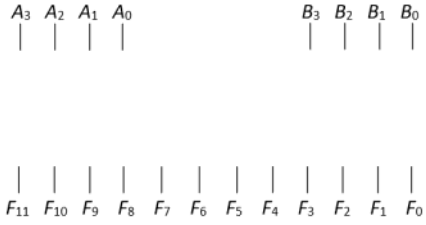
For example, the decimal value 396 is represented as 0011 1001 0110 in BCD code.

Given two decimal digits A and B, represented by their BCD codes $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ respectively, implement a circuit without using any logic gates to calculate the BCD code of the 3-digit output of $(51\times A) + (20\times(B\%2))$, where % is the modulo operator. Name the outputs $F_{11}F_{10}F_9F_8$ $F_7F_6F_5F_4$ $F_3F_2F_1F_0$.

For example, if A=2 (or 0010 in BCD) and B=7 (or 0111 in BCD), then $(51\times A) + (20\times(B\%2)) = 122$ or 0001 0010 0010 in BCD. Hence, the circuit is to produce the output 0001 0010 0010 for the inputs 0010 and 0111.

[*Hint:* Fill in the table below that computes $5\times A$.]

A				$5\times A$							
A_3	A_2	A_1	A_0								
0	0	0	0								
0	0	0	1								
0	0	1	0								
0	0	1	1								
0	1	0	0								
0	1	0	1								
0	1	1	0								
0	1	1	1								
1	0	0	0								
1	0	0	1								



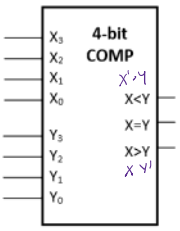
4. Given a 4-bit magnitude comparator as shown on the right, implement the following 4-variable Boolean functions using only this single magnitude comparator with no other logic gates. (Note that there could be multiple answers.)

(a) $F(A,B,C,D) = \Sigma m(12-15)$. ABCD → 1011

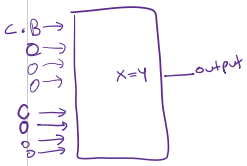
(b) $G(A,B,C,D) = \Sigma m(0, 6, 9, 15)$. *

(c) $H(A,B,C,D) = \Sigma m(0, 1, 6, 7, 8, 9, 14, 15)$. • $B' \cdot C' + B \cdot C$

(d) $Z(A,B,C,D) = \Sigma m(1, 3, 5, 7, 9, 11, 13)$. △

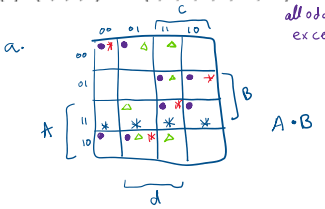


A = B
⇔ $A'B' + A \cdot B$



a.

all odd minterms except 15



midterm_20s2_qns

Friday, October 20, 2023 3:07 PM

midterm_20s2_qns

CS2100 Computer Organization
2020/21 Semester 2
Mid-term Assessment (Backup)

1. This assessment paper is valid only when instructed to be used by the course coordinators, via the proctors.

2. Answer the questions on a piece of paper. Listen to instructions from the proctors on the ending time of the assessment.

3. When the proctors have told you to stop writing, you have 5 minutes to scan your answers using your mobile phone. You can use software like CamScanner, or simply take a picture. In either case ensure that your writing is legible.

4. Answers that cannot be read will not be marked and you will receive 0 for those questions.

5. When you have finished scanning, email your scans to colin.mod.mails@gmail.com.

6. All restrictions and instructions in the midterm SOP continue to apply.

7. There are FIFTEEN (15) questions on EIGHT (8) printed pages, including this one.

MRQ Questions

Choose ALL options that are correct, and none that are wrong. Each question is worth 2 marks. No partial credits.

1. Given the hexadecimal value 0x3F2C, choose all of the possible interpretations of this value below:

   a. A 16-bit 1's complement negation of -16173
   b. The decimal value 16172.
   c. A 16-bit 2's representation of 16173.
   d. The ASCII characters '?' and ','
   e. The C string "?,"

2. We wish to implement the pseudo-instruction BLE $s1, $s2, target, which branches to "target" if $s1<=$s2. Choose all of the sequences below that implement this behaviour correctly (Note: We are only looking for correct behaviour. May not necessarily be the shortest possible implementation.)

   a. slt $t1, $s1, $s2
      bne $t1, $zero, target

   b. sub $t1, $s1, $s2
      slt $t2, $t1, $zero
      bne $t2, $zero, target

   c. +slt $t1, $s2, $s1
      beq $t1, $zero, target

   d. +sub $t1, $s2, $s1
      slt $t2, $t1, $zero
      beq $t2, $zero, target

   e. slt $t1, $s2, $s1
      bne $t1, $zero, target

3. Aiken has written a program to multiply a number stored in register $s0 by 3 and store the result in register $s1. Sadly he lost the assembly code and only has the following fragments of MIPS machine code in hexadecimal. Help Aiken to find which instructions are from his program. The ordering of the instructions is not important, and we further assume that the register $s2 contains the value 3.

	16	18	17	0	24	a.	0x02128818
add	17	16	17	0	32	b.	0x02308820
sll	16	0	17	1	0	c.	0x02008840
and	16	17	17	0	36	d.	0x02118824
sll	0	16	17	1	0	e.	0x00108840

4. Pick all of the statements below that are TRUE about the MIPS datapath.

   a. A single cycle implementation is better than multicycle implementation since every MIPS instruction takes exactly the same amount of time to go through the datapath.
   b. A multicycle implementation is better than single-cycle since not all instructions need to pass through every stage of the datapath.
   c. In an N stage datapath where each stage may take a different amount of time to complete, a multicycle implementation potentially allows up to N times speedup over a single cycle implementation if there are instructions that must pass through only one stage.
   d. In a multicycle implementation where every stage takes the same amount of time to complete, the instruction that passes through the most stages will have the same execution time as in a single-cycle implementation.
   e. In a multicycle implementation where every stage may take a different amount of time to complete, the instruction that takes the longest will NOT have the same execution time as in a single cycle implementation.

5. We wish to implement a no-operation (NOP) instruction in the MIPS datapath that, when executed, effectively does nothing except waste time (this sounds strange but is actually very useful in applications that must meet strict minimum timing requirements). Choose all the signal combinations that would correctly implement NOP (note: X = Don't care.). NOP is implemented as an R-format instruction with a function code of 0x3F. (Note: Here we only consider the specifications for the signal values; we do not consider actual implementation yet).

   a. RegDst=1, RegWrite=0, AluSrc=0, MemWrite=0, MemRead=0, MemToReg=0, PCSrc=0
   b. RegDst=1, RegWrite=X, AluSrc=X, MemWrite=0, MemRead=0, MemToReg=X, PCSrc = 0
   c. RegDst=0, RegWrite=0, ALuSrc=1, MemWrite=0, MemRead=0, MemToReg=1, PcSrc=1
   d. RegDst=X, RegWrite=0, AluSrc=X, MemWrite=X, MemRead=0, MemToReg=X, PCSrc=0
   e. RegDst=X, RegWrite=0, AluSrc=X, MemWrite=0, MemRead=0, MemToReg=1, PCSrc=0

MCQ Questions

Each question has exactly one correct answer and is worth 1 mark.

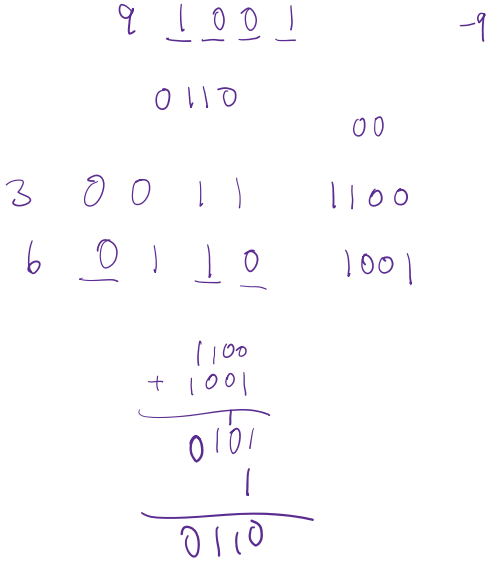
Section 1 – Number Systems

6. To find the 10's complement of an N-digit decimal number Y, we do $10^N - Y$. Which ONE of the following statements is TRUE?

   a. In an N-digit 10's complement number system, the leftmost digit has a weight of 10^N.
   b. In an N-digit 10's complement number system, the leftmost digit has a weight of 10^{N-1}.
   c. In an N-digit 10's complement number system, the leftmost digit has a weight of -10^N (negative 10^N)
   d. In an N-digit 10's complement number system, the leftmost digit has a weight of -10^{N-1} (negative 10^{N-1}).
   e. None of the choices in this question (except this one) are true.

7. In a 4-bit 1's complement number system, doing -3 - 6 (negative 3 minus 6) will give us:

   a. -9
   b. 5
   c. 6
   d. -5
   e. -6



Section 2 – C Programming

8. What does the following C function do? Assume that a long is twice the size of an int.

```
unsigned long mystery(unsigned int x, unsigned int y) {
    unsigned long a = 0, b = x;

    while(y) {
            if(y & 0b1) {
                a = a + b;
            }

            b = b << 1;
            y = y >> 1;
    }

    return a;
}
```

a. Adds x and y.
b. Shifts b to the right by 1 bit and y to the left by 1 bit and adds them together.
c. Produces an even parity for x and y.
d. Multiplies x and y.
e. Divides x by y.

9. What does the following C function do?

```
unsigned char mystery2(unsigned char x, unsigned char y) {
    unsigned char a = x;
    unsigned char b = y;
    unsigned char c = 1;
    unsigned char d = 0;

    while(c) {
        if((a & 1) != (b & 1)) {
            d = d + c;
        }

        a = a >> 1;
        b = b >> 1;
        c = c << 1;
    }

    return d;
}
```

a. Shifts a and b left by 1 bit and c right by 1 bit, then adds c to d.
b. Does a bit-wise XOR of x and y.
c. Checks whether both x and y are equal.
d. The function will never exit.
e. Does a bit-wise division of x and y.

Section 3. Assembly Language

Dueet wrote the following code in MIPS assembly to process some elements of an array whose base address is in $s2.

1		addi $s0, $zero, 0
2		addi $s1, $zero, 0
3		addi $t0, $s2, 0
4		addi $t1, $s2, 40
5	x:	slt $t2, $t0, $t1
6		beq $t2, $zero, z
7		lw $t3, 0($t0)
8		addi $t0, $t0, 4
9		andi $t3, $t3, 1
10		beq $t3, $zero, y
11		addi $s0, $s0, 1
12		j x
13	y:	addi $s1, $s1, 1
14		j x
15	z:	

s0 = 0 ← if 1
s1 = 0 ← if 0
t0 = s2 + 4
t1 = s2 + 40 ← end of arr
t2 = t0 < t1
if t0 < t1 else exit
t3 = 0(t0)

10. What does this code do?

a. It counts the number of array elements divisible by 2 and number of items divisible by 3 and stores the results in $s0 and $s1 respectively.
b. It counts the number of array elements with even parity and odd parity and stores the results in $s0 and $s1 respectively.
c. It counts the number of odd and even array elements and stores the results in $s0 and $s1 respectively.
d. It counts the number of array elements with even parity and odd parity and stores the results in $s1 and $s0 respectively.
e. It counts the number of array elements that are zero and non-zero and stores the results in $s0 and $s1 respectively.

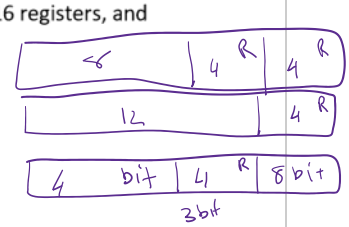
11. Dueet's software license for her assembler has expired, and she decided to "hand-assemble" the code above by consulting the datasheet and translating the assembly code into machine code. However when she ran her code she found that her code always terminates with $s1=0 regardless of the contents of the array, though $s0 is correct. What is the likeliest problem?

    a. The j statement at line 12 was encoded to jump to line 2 instead of line 5. X
    b. She calculated the offset for the branch instruction at line 10 based on the instruction at line 10 instead of line 11.
    c. She calculated the offset for the jump statement at line 14 using the address at line 15 instead of the address at line 13.
    d. She encoded the andi instruction at line 9 to do "andi $t3, $t3, 0" instead of "andi $t3, $t3, 1". X
    e. She calculated the offset for the branch at line 10 based in bytes rather than words.

Section 4. Fill In The Blanks (Expanding Opcodes)

In this section we assume a processor with a fixed 16-bit instruction length, 16 registers, and the following 3 instruction classes:

Class A: Two registers.
Class B: One register.
Class C: One register, one 8-bit immediate value.



16 register need 5 bit

In all cases we assume that the encoding space for opcodes is fully utilized.

12. The minimum number of opcodes that can be encoded on this machine is _____.

13. The maximum number of opcodes that can be encoded on this machine is _____.

$1 + 1 \cdot 2^4 + (2^4 - 2) \cdot 2^8 = 3601$

$2^4 - 1 + 2^4 - 1 + 2^4 = 46$ (C)

Section 5. Fill In The Blanks (Floating Point Numbers):

We have a 16-bit floating point number system with the following format:

1 sign bit for the mantissa	6 bit exponent in 2's complement. No reserved bit patterns with special meanings.	9 bit normalized mantissa, no hidden bit, in sign and magnitude representation.

The leftmost bit and the rightmost 9 bits form a sign-and-magnitude mantissa. Answer the following questions, filling in the **exact** numeric answer each time.

14. Within the representation range of this number system (i.e. between the most negative and most positive numbers that can be represented), the number 0 cannot be represented.

15. The most negative number that can be represented in this number system is -63.

midterm_19s2_qns

Friday, October 20, 2023 8:27 PM

midterm_19s2_qns

NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

ONLINE QUIZ
AY2019/2020 Semester 2

CS2100 — COMPUTER ORGANISATION

11 March 2020 Time Allowed: **1 hour 40 minutes**

INSTRUCTIONS

1. This question paper contains **FIVE (5)** questions and comprises **SIX (6)** printed pages. (Question 0 is on the Answer Sheet.)
2. Answer **ALL** questions within the space provided on the Answer Sheet.
3. Please type **ALL** your answers. If you are writing your answers, ensure that your handwriting is legible, or marks may be deducted.
4. Submit only the Answer Sheet.
5. Maximum score of this quiz is **40 marks**.

——— **END OF INSTRUCTIONS** ———

CS2100 Online Quiz

SIMP Language

SIMP is a 16-bit general-purpose register architecture processor with the specification below. Registers `$rs`, `$rt`, and `$rd` are placeholders for actual general-purpose registers `$1`, `$2`, ..., `$6`, each holding a 16-bit value. `const` refers to a constant value and `label` refers to label in the instruction corresponding to a specific line in the code. All **constants** are given as *4 bits 2's complement signed values*. All **labels** will be converted into *actual addresses using direct addressing mode*.

SIMP can only accommodate up to 128 addressable memory where each memory location holds 1 byte. The memory locations are numbered from 0000000_2 to 1111111_2. All memory locations are given as unsigned binary values. Additionally, each word in this processor consists of 2 bytes (16 bits) and every instruction is word-aligned.

We will use the following convention to simplify our discussion:

- Given a register `$r`, the content of the register `$r` is given as `R[$r]`.
- Given a memory location `addr`, the content of the memory location at `addr` is given as `M[addr]`.
- The notation `A + B` is used to denote the arithmetic + operation where the operands are `A` and `B`.
- The notation `A - B` is used to denote the arithmetic - operation where the operands are `A` and `B`.
- The notation `A & B` is used to denote the bitwise AND operation where the operands are `A` and `B`.
- The notation `A ^ B` is used to denote the bitwise XOR operation where the operands are `A` and `B`.
- The notation `~A` is used to denote the bitwise NOT operation where the operand is `A`.
- `PC` is used for the special register holding the address of the current instruction.

Round brackets `()` are used to disambiguate order of operations.

Addressing Architecture:	General-Purpose Register
Number of General-Purpose Registers:	Six (`$1`, `$2`, ..., `$6`)
Special Registers (*addressable*): *These registers cannot be written*	`R[$0]= 0x0000` [i.e., all 0s] `R[$7]= 0xFFFF` [i.e., all 1s]
Special Register (*non-addressable*):	Program Counter (`$pc`)
Instruction Format:	Fixed length 16-bit instructions
Arithmetic Instructions: *These are arithmetic operations and they must*	• `ADD $rd, $rs, $rt, const`

Instruction Format:	Fixed length 16-bit instructions

Arithmetic Instructions:
These are arithmetic operations and they must come with 3 operands
	• `ADD $rd, $rs, $rt, const`
	  ○ `R[$rd] = R[$rs] + R[$rt] + const`
	• `SUB $rd, $rs, $rt, const`
	  ○ `R[$rd] = (R[$rs] - R[$rt]) - const`

Logical Instructions:
These are bitwise operations and they must come with 3 operands
	• `AND $rd, $rs, $rt, const`
	  ○ `R[$rd] = R[$rs] & R[$rt] & const`
	• `XOR $rd, $rs, $rt, const`
	  ○ `R[$rd] = R[$rs] ^ R[$rt] ^ const`

Load/Store Instructions:
All addresses are byte addresses and they are word-aligned
	• `LW  $rt, $rs, const`
	  ○ `R[$rt] = M[R[$rs] + const]`
	• `SW  $rt, $rs, const`
	  ○ `M[R[$rs] + const] = R[$rt]`

Branch Instruction:
This is a compare and branch instruction
	• `BEQ $rs, $rt, label`
	  ○ Branch to label if `R[$rs] == R[$rt]`
	  ○ Otherwise, go to next instruction

CS2100 Online Quiz

Question 1: Warmup Questions ***[8 marks]***

Consider 4 statements written in C below with the following variable-to-register mapping:

x: $1	y: $2	z: $3

```
1) x = y * 2;
2) x = ~y;
3) x = y + z - 3;
4) x = y - 2;
```

Answer the following questions below. Each answer must be a *single SIMP instruction*.

a) How do we perform instruction (1) in SIMP? [2 marks]

b) How do we perform instruction (2) in SIMP? [2 marks]

c) How do we perform instruction (3) in SIMP? [2 marks]

d) How do we perform instruction (4) in SIMP? [2 marks]

Question 2: Compilation ***[8 marks]***

Using the given SIMP instruction set, complete the SIMP equivalent of the C code below on the answer sheet. Each array element is two bytes long. Ensure that your code is properly commented to ease understanding.

```
sum = 0;
```

to ease understanding.

```
sum = 0;
for (i=0; sum==0; i++) {
  switch (x[i]) {
    case  0: x[i] = -1;
    case -1: break;
    default: sum = x[i] + i + 5;
  }
}
```

Assume that you have the following variable-to-register mapping.

`sum`: `$1`	`i`: `$2`	base address of `x`: `$3`

Recap that in C, switch-case statement will continue execution from one case to the next unless there is a `break`. In particular, for the code above, `case 0` will "spill over" to `case -1`. Additionally, any `break` statement within switch-case will exit the switch-case only. You are already given the first 4 lines as well as the last 2 lines of the SIMP code. Your task is to fill in the middle section. You are not to modify the given lines of code on the answer sheet.

Question 3: Encoding ***[14 marks]***

We can categorize the set of instructions in SIMP into three categories, with instruction formats as shown below:

	Bits	3-bit	3-bit	3-bit	3-bit	4-bit
Class A	Fields	opcode	rs	rt	rd	const
Class B	Fields	opcode	rs	rt	funct	const
Class C	Fields	opcode	rs	rt	label	

The following rules summarize how to encode a SIMP instruction to binary.

- Register is encoded as its number. For instance, `$6` is encoded as 110_2.
- Constant is encoded as 2's complement signed value. For instance, `-2` is encoded as 1110_2.

- Register is encoded as its number. For instance, `$6` is encoded as 110_2.
- Constant is encoded as 2's complement signed value. For instance, `-2` is encoded as 1110_2.
- Label is first converted into address and the address is encoded directly. For instance, to branch to address 0011010_2, the label field is specified as 0011010_2.
- `funct` is not used in this encoding, it will always be `0` in question 3a. It is reserved for future use (*see question 3b*).
- Opcode is encoded using the table below where the value is given as *decimal*, then converted to binary. For instance, for `LW` instruction, the opcode is `5`, the encoded value is 101_2.

Instruction	Opcode	Instruction	Opcode
ADD	1	LW	5
SUB	2	SW	6
AND	3	BEQ	7
XOR	4		

a) Consider the following SIMP code which has been partially encoded. Assume that the first instruction is at address 0000000_2. Fill in the missing hexadecimal encodings or the instructions on the answer sheet.

Hexadecimal	SIMP	
0x402F		[2 marks]
	`L: ADD $2, $2, $2, 0`	[2 marks]
0x241F		[2 marks]
0xE08A		[2 marks]
	`BEQ $1, $1, L`	[2 marks]
	`E:`	

b) Suppose besides the 7 instructions given (`ADD/SUB/AND/XOR` in class A, `LW/SW` in class B and `BEQ` in class C), you want to add more instructions into the SIMP instruction set. Assuming the encoding space is completely utilized, *including* `funct`, and note that the opcodes of the given 7 instructions should not be used for other instructions. What is the

(i) maximum total number of instructions (including the 7 given instructions)? [2 marks]

(ii) minimum total number of instructions (including the 7 given instructions)? [2 marks]

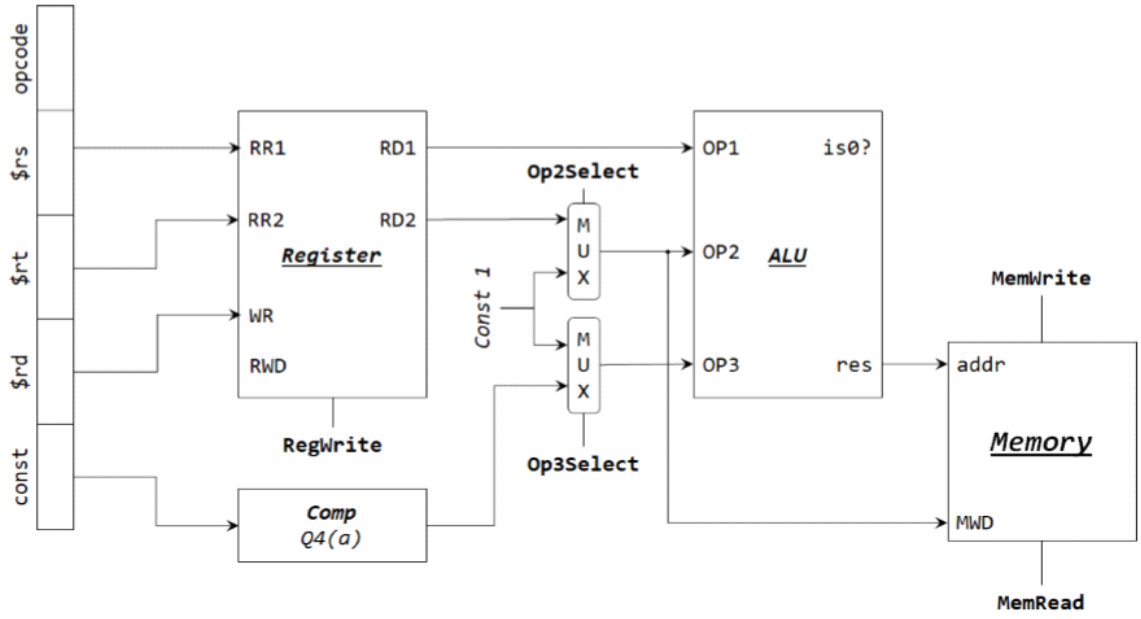
You do not need to show workings for the above.

You do not need to show workings for the above.

Question 4: Datapath and Control ***[9 marks]***

Study the dathapath of the processor below. It partially implements the fetch, decode, and ALU stage for the SIMP language. `Const 1` is to be determined in part (b). The ALU is special as it can accept 3 operands instead of the usual 2. Operations in the ALU require all operands to have 16-bit value. The implementation is currently limited up to the store-to-memory stage.

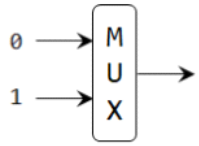
The control signals are summarized in the table below.



Instruction	Op2Select	Op3Select	MemRead	MemWrite	RegWrite
ADD	0	1	0	0	1
SUB	0	1	0	0	1
AND	0	1	0	0	1
XOR	0	1	0	0	1
LW	1	1	1	0	1
SW	1	1	0	1	0
BEQ	0	0	0	0	0

Note that for the multiplexer, the input line is selected as follows:

Note that for the multiplexer, the input line is selected as follows:



Since the inputs to the ALU must be 16-bit values, we will need the component marked as `Comp`. Additionally, for `BEQ` operation, the input to `OP3` of ALU must be chosen from either `label` or `Const 1`. Similarly, for `LW` and `SW` operation, the input to `OP2` of ALU must be chosen from either `$rt` or `Const 1`.

For `BEQ`, the operation performed by the ALU is subtraction (i.e., `OP1 - OP2 - OP3`). For `LW` and `SW`, the operation performed by the ALU is addition (i.e., `OP1 + OP2 + OP3`).

Given the information above, answer the questions below.

a) What is the component marked as `Comp`? [1 mark]

b) What is the value of `Const 1`? Give your answer in ***hexadecimal***. [1 mark]

Consider executing the instruction `BEQ $4, $5, 54`. Note that `54` is already the target address and not a label anymore. The encoding for this instruction is: `0xF2B6` or `1111 0010 1011 0110`$_2$.

Use the following convention in your answer:

- Given a register `$r`, the content of the register `$r` is given as `R[$r]`.
- Given a memory location `addr`, the content of the memory location at `addr` is given as `M[addr]`.
- The notation `A + B` is used to denote the arithmetic + operation where the operands are `A` and `B`.
- The notation `A - B` is used to denote the arithmetic - operation where the operands are `A` and `B`.
- The notation `A & B` is used to denote the bitwise AND operation where the operands are `A` and `B`.
- The notation `A ^ B` is used to denote the bitwise XOR operation where the operands are `A` and `B`.
- The notation `~A` is used to denote the bitwise NOT operation where the operand is `A`.
- `PC` is used for the special register holding the address of the current instruction.
- Use decimal values for constants

c) Fill in the table on the answer sheet for the input/output value of each component in the datapath of the processor given the control signal and encoding above. `RR1` has been filled for you. [7 marks]

RR1	RR2	WR	OP1	OP2	OP3	addr	MWD
$4							

=== END OF PAPER ===

termtest1_18s2_qns

Friday, October 20, 2023 9:42 PM

termtest1_18s2_qns

NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

MID-TERM TEST
AY2018/19 Semester 2

CS2100 — COMPUTER ORGANISATION

13 March 2019 Time Allowed: **1 hour 45 minutes**

INSTRUCTIONS

1. This question paper contains **THIRTEEN (13)** questions and comprises **EIGHT (8)** printed pages.
2. Page 7 contains the **MIPS Reference Data** sheet.
3. Page 8 contains reference tables for **ASCII** and **Powers of Two**.
4. An **Answer Sheet**, comprising **TWO (2)** printed pages, is provided for you.
5. Write your **Student Number** and **Tutorial Group Number** on the Answer Sheet with a **PEN**.
6. Answer **ALL** questions within the space provided on the Answer Sheet.
7. You may write your answers in pencil (at least 2B).
8. You must write legibly or marks may be deducted.
9. Submit only the Answer Sheet at the end of the test. You may keep the question paper.
10. This is a **CLOSED BOOK** test. However, an A4 single-sheet double-sided reference sheet is allowed.
11. Maximum score of this test is **40 marks**.
12. Calculators and computing devices such as laptops and PDAs are not allowed.

——— **END OF INSTRUCTIONS** ———

Questions 1 – 6: Each multiple-choice-question has only <u>one</u> correct answer. Write your answers in the boxes on the **Answer Sheet**. Two (2) marks are awarded for each correct answer and no penalty for wrong answer. **[Total: 12 marks]**

1. You are told that $(32)_b$ is a product of two prime numbers (*i.e.*, $(32)_b = P_1 \times P_2$ *where* P_1 *and* P_2 *are primes*). What is/are the value of base ***b***?

   (i) 8 (ii) 9 (iii) 11

   A. (i) only
   B. (i) and (ii) only
   C. (i) and (iii) only
   D. (i), (ii), and (iii)
   E. None of the above

2. What is the content of **$t2** after executing the following MIPS code?

```
lui  $t0,  0xAAAA
srl  $t0,  $t0  , 16
lui  $t0,  0xA0A0
ori  $t1,  $zero, 0x5555
and  $t2,  $t1  , $t0
```

   A. 0x00000000
   B. 0x0000FFFF
   C. 0xA0A00000
   D. 0xA0A05555
   E. None of the above.

For questions 3 – 4:
You are designing a machine with 6 registers and 64 addresses. You are in the process of creating two (2) classes of 16-bit instructions. The first is instruction class A that has 3 registers. The second is instruction class B that has 1 address and 2 registers. Both instructions exist and the encoding space is completely utilised.

3. What is the ***maximum*** total number of instructions?

   A. 119
   B. 120
   C. 121
   D. 122
   E. None of the above.

4. What is the ***minimum*** total number of instructions?

   A. 22
   B. 21
   C. 20
   D. 19
   E. None of the above.

For questions 5 – 6:
Study the following C programs.

```
#include <stdio.h>

typedef struct {
  int numer[1]; // numerator
  int *denom;   // denominator
} rational;

void multiply(rational, rational*);

int main(void) {
  int val1 = 2, val2 = 5;
  rational num1 = {{1}, &val1}, // 1/2
           num2 = {{2}, &val2}; // 2/5
  multiply(num1, &num2);
  printf("%d %d\n", num1.numer[0], *(num1.denom)); // Question 3
  printf("%d %d\n", num2.numer[0], *(num2.denom)); // Question 4
}

void multiply(rational x, rational *y) {
  int x_num = *(x.numer), y_num = *(y->numer),
      x_den = *(x.denom), y_den = *(y->denom);
  *(x.numer)  = x_num * y_num;
  *(x.denom)  = x_den * y_den;
  *(y->numer) = x_num * y_num;
  *(y->denom) = x_den * y_den;
}
```

5. What is the output of the ***first*** print?
   A. 1 5
   B. 1 10
   C. 2 5
   D. 2 10
   E. None of the above
6. What is the output of the ***second*** print?
   A. 1 5
   B. 1 10
   C. 2 5
   D. 2 10
   E. None of the above

Questions 7 – 10: **C & MIPS (Tracing, Compiling, Encoding)** **[Total: 14 marks]**
For the next *four* (4) questions, refer to the code below. The code has been partially filled in for you. One of the blanks has been filled for you.

C Code	MIPS Code

`int main(void) {`
`  i = 0;`
`  A = 'A';`
`  a = 'a';`
`  Z = 'Z';`
	[addi $s1, $zero, 0]
	[addi $s3, $zero, 65]
	[addi $s4, $zero, 97]
	[addi $s5, $zero, 90]

`  do {`	***loop:***
`  if( str[i] >= 'A'` `  && str[i] <= 'Z' ) {`	add $s7, $s0 , $s1 lb $t2, 0($s7) slt $t1, $t2 , $s3 bne $t1, $zero, else [slt $t4, $s5, $t2] [bne $4, $zero, else]
`  func(str + i);`	j func ***ret:***
`  }`	***else:***
`[ i++ ];`	addi $s1, $s1, 1
`  } while(str[i-1] != 0);`	[bne $t2, $zero, loop]
`  return 0;` `}`	***quit:*** j exit
// Code omitted	***# Code omitted***
`void func(char* str) {`	***func:***

`[ *str = *str + 'a' - 'A' ];`
`  return;`
	lb $t8, 0($s7)
	addi $t8, $t8, 97
	addi $t8, $t8, -65
	sb $t8, 0($s7)
	j ret

`}`	
// Code omitted	***# Code omitted***
// End of Program	***exit:***

We will also use the following variable-to-register mapping within the ***main*** function and not the ***func*** function.

base addr of `str`	`$s0`	`A`	`$s3`
`i`	`$s1`	`a`	`$s4`
`len`	`$s2`	`Z`	`$s5`

Note that in the section marked with "**`Code omitted`**", there can be *any number* of instructions. The ASCII table, if required, is available at the end of the question paper.

7. Fill in the blanks with appropriate C code and MIPS code. Note that you must fill in ***exactly*** the given number of instructions. You CANNOT use *pseudo-instructions* for the MIPS instructions. [8 marks]
8. Consider running the program with the initial value of **`str`** as **"`CS2100 is Easy!`"**. What will be the final value of **`str`**? [2 marks]
9. What is the encoding in ***hexadecimal*** for the 8th MIPS instruction "**`bne $t1, $zero, else`**"? [2 marks]
10. **[CHALLENGING]** What is the ***maximum*** possible number of MIPS instructions that can be inserted into the regions marked with "**`Code omitted`**"? Note that the code can be inserted into any one of the two regions. We are only interested in the total. For simplicity, write your answer in terms of $2^x \pm y$ [2 marks]

Question 11: **Number Systems** **[Total: 6 marks]**
For the next question, recall the IEEE 754 single-precision floating-point number representation.

11. Given the following hexadecimal value in the IEEE 754 single-precision floating-point number representation:

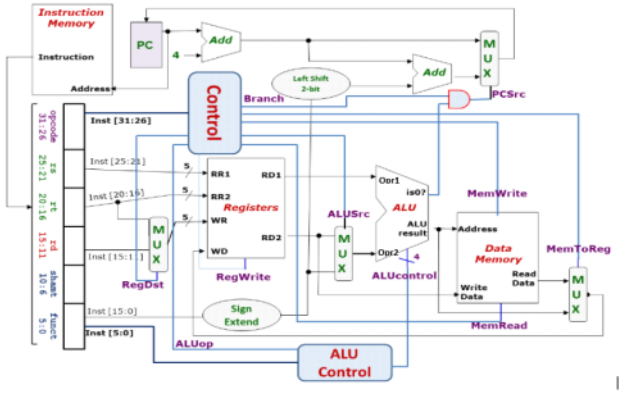
0x C0BB0000

What decimal value does it represent? [6 marks]

Questions 12 – 13: **Datapath & Control** **[Total: 8 marks]**
For the next *two* (2) questions, refer to the diagram of the Datapath and Control below.

Questions 12 – 13: **Datapath & Control** **[Total: 8 marks]**

For the next *two* (2) questions, refer to the diagram of the Datapath and Control below.



However, let us consider a ***different*** ***Control*** unit from the one we have in the lecture notes. Assume that our modified control unit produces the following control signals:

Instr. Type	RegDst	ALUsrc	MemToReg	RegWrite	MemRead	MemWrite	Branch
R-type	1	0	0	1	0	0	0
lw	0	1	1	1	1	0	0
sw	1	1	1	0	0	1	1 (wrong)
beq	1	0	1	0	1	0	1

However, **ALUop** is still the same as before. We will also be using the following notation:

- The value of constants will be given as constants (e.g., **-5**)
- The value of the register number 8 will be represented as **$8**
- The value stored in the register number 8 will be represented as **R[$8]**
- The value of an arithmetic operation (op) between two values A and B will be represented as A op B. For example, A + B in the case of addition.
- The value stored in the memory location L will be represented as **M[L]**

12. You are given an instruction below, fill in the table in the answer sheet. Assume that the label **L1** is converted to the immediate value **-2**.
beq $8, $0, L1 [6 marks]

13. **[CHALLENGING]** The instruction **sw** should not produce the value **1** for Branch control signal (as highlighted above). Give one MIPS instruction using **sw** that is *guaranteed* to cause the processor to perform a branch. [2 marks]

MIPS Reference Data

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		0 / 08_{hex}
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27_{hex}
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25_{hex}
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / $2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00_{hex}
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		0 / 02_{hex}
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0)= R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (atomic) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22_{hex}
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23_{hex}

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1'b0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

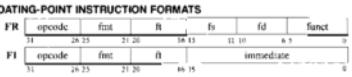
ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION		OPCODE / FMT /FT / FUNCT (Hex)
Branch On FP True	bc1*	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr	(4)	11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/--/--/1b
FP Add Single	add.s	FR	F[fd]= F[fs] + F[ft]		11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}		11/11/--/0
FP Compare Single	c.x.s*	FR	FPcond = (F[fs] *op* F[ft]) ? 1 : 0		11/10/--/y
FP Compare Double	c.x.d*	FR	FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0		11/11/--/y

* (x is eq, lt, or le) (*op* is ==, <, or <=) (*y* is 32, 3c, or 3e)

FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}		11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}		11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--
Move From Hi	mfhi	R	R[rd] = Hi		0 /--/--/10
Move From Lo	mflo	R	R[rd] = Lo		0 /--/--/12
Move From Control	mfc0	R	R[rd] = CR[rs]		10 /0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Shift Right Arith.	sra	R	R[rd] = R[rt] >>> shamt		0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS



PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Positive Power of 2

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

Negative Power of 2

Exp	Val	Exp	Val
2^{-1}	0.5	2^{-9}	0.001953125
2^{-2}	0.25	2^{-10}	0.0009765625
2^{-3}	0.125	2^{-11}	0.00048828125
2^{-4}	0.0625	2^{-12}	0.000244140625
2^{-5}	0.03125	2^{-13}	0.0001220703125
2^{-6}	0.015625	2^{-14}	0.00006103515625
2^{-7}	0.0078125	2^{-15}	0.000030517578125
2^{-8}	0.00390625	2^{-16}	0.0000152587890625

termtest1_17s2_qns

Saturday, October 21, 2023 1:58 AM

termtest1_
17s2_qns

NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

MID-TERM TEST
AY2017/18 Semester 2

CS2100 — COMPUTER ORGANISATION

13 March 2018 Time Allowed: **1 hour 30 minutes**

INSTRUCTIONS

1. This question paper contains **ELEVEN (11)** questions (excluding the bonus question) and comprises **EIGHT (8)** printed pages.
2. Page 8 contains the **MIPS Reference Data** sheet.
3. An **Answer Sheet**, comprising **TWO (2)** printed page, is provided for you.
4. Write your **Student Number** and **Tutorial Group Number** on the Answer Sheet with a **PEN**.
5. Answer **ALL** questions within the space provided on the Answer Sheet.
6. You may write your answers in pencil (at least 2B).
7. Submit only the Answer Sheet at the end of the test. You may keep the question paper.
8. This is a **CLOSED BOOK** test. However, an A4 single-sheet double-sided reference sheet is allowed.
9. Maximum score of this test is **40 marks**.
10. Calculators and computing devices such as laptops and PDAs are not allowed.

——— END OF INSTRUCTIONS ———

Bonus question:

0. [This bonus question is worth **1** mark. The mark of this question will only be added if the total mark scored is less than 40.]

The photo on the right shows Aaron playing the 'magic number' game in class. When did that happen?

A. In the first week of the semester.
B. In the second week of the semester.
C. In the third week of the semester.
D. This is fake news, it has never happened.

Questions 1 – 5: Each multiple-choice-question has only one correct answer. Write your answers in the boxes on the **Answer Sheet**. Two marks are awarded for each correct answer and no penalty for wrong answer.

1. What is the base *x* for this addition operation to hold: $(135)_x + (25)_x = (161)_x$?

A. 6
B. 7
C. 8
D. 9
E. 11

2. What is the content of **$t2** after executing the following MIPS code?

```
lui  $t0, 0xA0A0
ori  $t0, $t0, 0x1234
addi $t1, $0, -8
xor  $t2, $t0, $t1
```

A. 0x5F5FEDC3
B. 0x5F5FEDC8
C. 0x5F5FEDCC
D. 0xA0A0123C
E. None of the above.

3. Which of the following statement(s) regarding the "`j`" (jump) instruction in MIPS is/are TRUE?

i. It is possible that a particular "`j`" instruction can **only** jump backward to instruction earlier in the code.
ii. If the same "`j`" instruction is executed multiple times (e.g. in a loop), it is possible that it jumps to different instructions.
iii. If a "`j`" instruction fails to reach its target due to limitation of the immediate field, it is possible that we can construct a chain of multiple "`j`" instructions to reach the target.

A. Only (i)
B. Only (ii)
C. Only (i) and (iii)
D. Only (ii) and (iii)
E. All of (i), (ii) and (iii).

For questions 4 and 5:
An ISA has three types of instructions: Type *A* instructions have 4-bit opcode, type *B* instructions have 7-bit opcode, and type *C* instructions have 8-bit opcode. All three types of instructions exist and the encoding space is completely utilised.

4. What is the minimum total number of instructions?

A. 3
B. 16
C. 24
D. 28
E. None of the above.

5. What is the maximum total number of instructions?

A. 238
B. 240
C. 247
D. 252
E. None of the above.

6. Given the following hexadecimal value in the IEEE 754 single-precision floating-point number representation:

C4007000

What decimal value does it represent? [3 marks]

7. Assuming that **x** is a positive integer, the following function returns 1 if **x** is a certain type of values or it returns 0 otherwise.

```
int mystery(int x) {
   return !((x-1) & x);
}
```

(a) What does mystery(20) return? [1 mark]
(b) What does mystery(32) return? [1 mark]
(b) What kind of values must **x** be for the function to return 1? [1 mark]

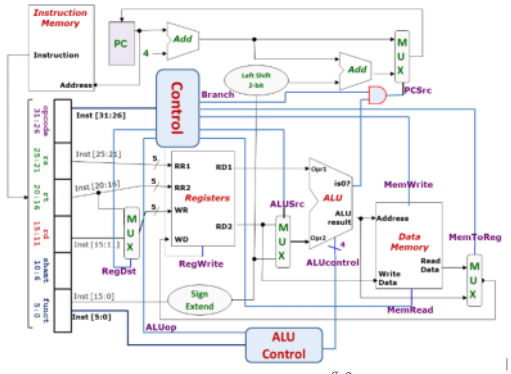
8. Write out the output of the following program. [3 marks]

```
#include <stdio.h>

typedef struct {
    int first, second;
} pair_t;

void g(int *, pair_t);

int main(void) {
    int arr[2] = { 11, 22 };
    pair_t pair = { 33, 44 };

    g(arr, pair);
    printf("%d %d %d\n", arr[0], pair.first, pair.second);
    return 0;
}

void g(int *arr, pair_t pair) {
    *arr = 55;
    pair.first = 66;
    pair.second = 77;
}
```


9. Write out the output of the following program. [6 marks]

```
#include <stdio.h>

int f(int *, int, char *);

int main(void) {
    int a, b, *p, *q;
    char c;

    a = 12;
    b = 50;
    c = 'c';
    p = &a;
    q = &b;
    *q = *q + 2;
    *p = f(&b, a, &c);
    printf("%d %d %c\n", a, b, c);
    return 0;
}

int f(int *p, int b, char *q) {
    int a = *p * 2 + b;
    *p = *p + 10;
    *q = *q + ('a' - 'A') + 1;
    return (a + *p);
}
```

10. Given the following datapath for the MIPS processor, answer the following question.



For this instruction

```
add $8, $9, $10
```

fill in the table in the Answer Sheet. Use the notation $8 to represent register number 8, [$8] to represent the content of register number 8 and Mem(X) to represent the memory data at address X. You are to fill in the data at the datapath element regardless of whether it is used for this instruction. If the data is used, underline it; if the data is not used, do not underline it. [4 marks]

11. [Total: 11 marks]
The base address of an array *A* is stored in register **$s0**, and the number of elements in **$s1**. Each array element is an integer that occupies 4 bytes. Assume that the initial contents of *A* are **{ 3, 2, 5, 1, 8 }** (i.e. *A*[0] is 3, *A*[1] is 2, etc. and **$s1** = 5). Study the following MIPS code.

```
        add  $t0, $s0, $zero   # Address: 0x004000A4
        add  $t1, $s1, $s1
        add  $t1, $t1, $t1
        add  $t1, $s0, $t1
S1:     slt  $t2, $t0, $t1
        beq  $t2, $zero, E1
        addi $t3, $t0, 4
S2:     slt  $t4, $t3, $t1
        beq  $t4, $zero, E2
        lw   $t5, 0($t0)
        lw   $t6, 0($t3)
        add  $t6, $t6, $t5
        sw   $t6, 0($t3)
        addi $t3, $t3, 4
        j    S2
E2:     addi $t0, $t0, 4
        j    S1
E1:
```

(a) Fill in the array elements after the execution of the above code. [3 marks]

(b) Write an equivalent C code for the above code. You may use the variable *n* to denote the size (number of elements) of array *A*. [4 marks]

(c) Convert the first instruction (`add $t0, $s0, $zero`) into its hexadecimal representation. [2 marks]

(d) Convert the last instruction (`j S1`) into its hexadecimal representation, assuming that the first instruction (`add $t0, $s0, $zero`) is at memory address 0x004000A4. [2 marks]

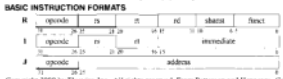
MIPS Reference Data

CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20$_{hex}$
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8$_{hex}$
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9$_{hex}$
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21$_{hex}$
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24$_{hex}$
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c$_{hex}$
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC = PC + 4 + BranchAddr	(4)	4$_{hex}$
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC = PC + 4 + BranchAddr	(4)	5$_{hex}$
Jump	j	J	PC = JumpAddr	(5)	2$_{hex}$
Jump And Link	jal	J	R[31] = PC + 8; PC = JumpAddr	(5)	3$_{hex}$
Jump Register	jr	R	PC = R[rs]		0 / 08$_{hex}$
Load Byte Unsigned	lbu	I	R[rt] = {24'b0, M[R[rs] +SignExtImm](7:0)}	(2)	24$_{hex}$
Load Halfword Unsigned	lhu	I	R[rt] = {16'b0, M[R[rs] +SignExtImm](15:0)}	(2)	25$_{hex}$
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30$_{hex}$
Load Upper Imm	lui	I	R[rt] = {imm, 16'b0}		f$_{hex}$
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23$_{hex}$
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27$_{hex}$
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25$_{hex}$
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d$_{hex}$
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / 2a$_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a$_{hex}$
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b$_{hex}$
Set Less Than Unsig.	sltu	R	R[rd] (R[rs] < R[rt]) ? 1 : 0	(6)	0 / 2b$_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00$_{hex}$
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		0 / 02$_{hex}$
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0)= R[rt](7:0)	(7)	28$_{hex}$
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (atomic) ? 1 : 0	(2,7)	38$_{hex}$
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29$_{hex}$
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	2b$_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22$_{hex}$
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23$_{hex}$

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16'b0, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

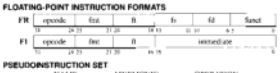
BASIC INSTRUCTION FORMATS



ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC	FOR-MAT	OPERATION	OPCODE / FMT / FT / FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr	(4)	11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/--/--/1b
FP Add Single	add.s	FR	F[fd] = F[fs] + F[ft]		11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}		11/11/--/0
FP Compare Single	c.x.s*	FR	FPcond = ({F[fs],F[fs+1]} op {F[ft],F[ft+1]}) ? 1 : 0		11/10/--/y
FP Compare Double	c.x.d*	FR	FPcond = ({F[fs],F[fs+1]} op {F[ft],F[ft+1]}) ? 1 : 0		11/11/--/y
* (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)					
FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}		11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}		11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--
Move From Hi	mfhi	R	R[rd] = Hi		0/--/--/10
Move From Lo	mflo	R	R[rd] = Lo		0/--/--/12
Move From Control	mfc0	R	R[rd] = CR[rs]		10/0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Shift Right Arith.	sra	R	R[rd] = R[rt] >>> shamt		0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS



PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

tut08qns

Thursday, October 26, 2023 5:59 PM

tut08qns

CS2100 Computer Organisation

Tutorial #8: MSI Components

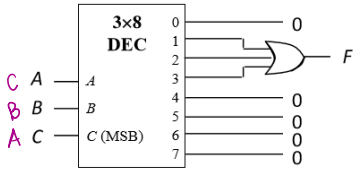
(Week 10: 23 – 27 October 2023)

Discussion Questions:

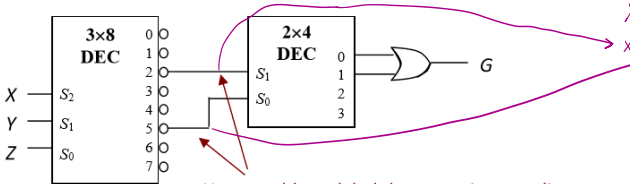
D1. Given this Boolean function:

$$F(A,B,C) = \Sigma\, m(1, 2, 3)$$

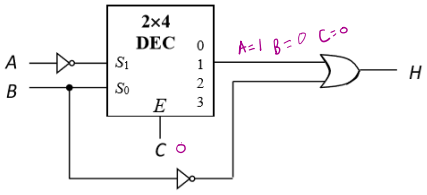
We want to implement this function using a **3×8 decoder with normal outputs** as shown below. Point out the mistakes in the solution below.



D2. Given the following circuit comprising a **3×8 decoder with negated outputs** and a **2×4 decoder with normal outputs**, what is the Boolean function *G(X,Y,Z)*?



D3. Given the following circuit comprising a **one-enabled 2×4 decoder with normal outputs**, what is the simplified SOP expression of Boolean function *H(A,B,C)*?



Tutorial Questions:

(Make sure you have done the above discussion questions.)

1. Realize the following function with (a) **an 8:1 multiplexer**, and (b) **a 4:1 multiplexer** using the first 2 input variables as the selector inputs.

   $$F(X, Y, Z) = \Pi M(1, 5, 6) \cdot D(4)$$

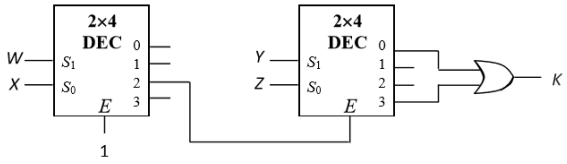
   You may write complemented variables instead of drawing an inverter to derive it. If you have several choices for your answer, choose the simplest one (constant logic values 0 and 1 are simpler than literals). You may write "x" or "d" for "don't-care" values.

   What if we use the last 2 input variables as the selector inputs instead for the 4:1 multiplexer?

2. You are given the following Boolean function: $K(W,X,Y,Z) = \Sigma m(8, 11)$.

   You are to implement this function using the fewest number of one-enabled 2×4 decoder with normal outputs and at most one logic gate? (Logic gates, as you have learned, are NOT, AND, OR, NAND, NOR, XOR, and XNOR.)

   The following is one solution. Is there a simpler circuit using just one decoder and one logic gate?



3. [AY2011/2 Semester 2 Exam question]
   You are to design a converter that takes in 4-bit input *ABCD* and generates a 3-bit output *FGH* as shown in Table 1 below.

Input				Output		
A	***B***	***C***	***D***	***F***	***G***	***H***
0	0	0	0	0	0	0
1	0	0	0	0	0	1
1	1	0	0	0	1	0
1	1	1	0	0	1	1
1	1	1	1	1	0	0
0	1	1	1	1	0	1
0	0	1	1	1	1	0
0	0	0	1	1	1	1

Table 1

S	$Y_3Y_2Y_1Y_0$
0	$J_3J_2J_1J_0$
1	$K_3K_2K_1K_0$

Table 2

   You are given the following components:

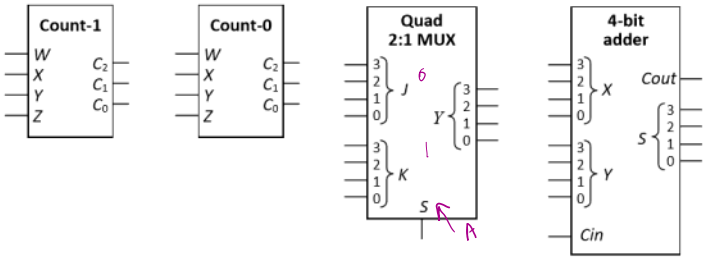
a. A **Count-1** device that takes in a 4-bit input *WXYZ* and generates a 3-bit output $C_2C_1C_0$ which is the number of 1s in the input. For example, if *WXYZ* = 0111, then $C_2C_1C_0$ = 011 (or 3).

b. A **Count-0** device that takes in a 4-bit input *WXYZ* and generates a 3-bit output $C_2C_1C_0$ which is the number of 0s in the input. For example, if *WXYZ* = 0111, then $C_2C_1C_0$ = 001 (or 1).

c. A **quad 2:1 multiplexer** that takes in two 4-bit inputs $J_3J_2J_1J_0$ and $K_3K_2K_1K_0$, and directs one of the inputs to its output $Y_3Y_2Y_1Y_0$ depending on its control signal *S*, as shown in Table 2 above.

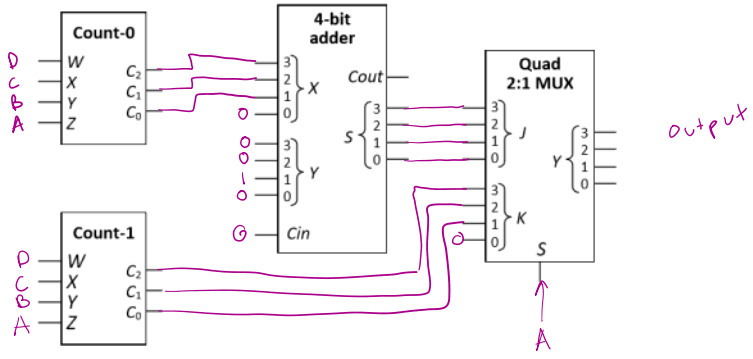
d. A **4-bit parallel adder** that takes in two 4-bit unsigned binary numbers and outputs the sum.

The block diagrams of these components are shown below:



Given the above 4 components, you are to employ block-level design to design the converter, without using any additional logic gate or other devices. You may observe that if *A* = 1, then the output *FGH* is simply the number of 1s in the input *ABCD*. You are to make your own observation for the case when *A* = 0.

[Hint (not given in exam): You need only use one of each of the components. Complete the diagram below.]

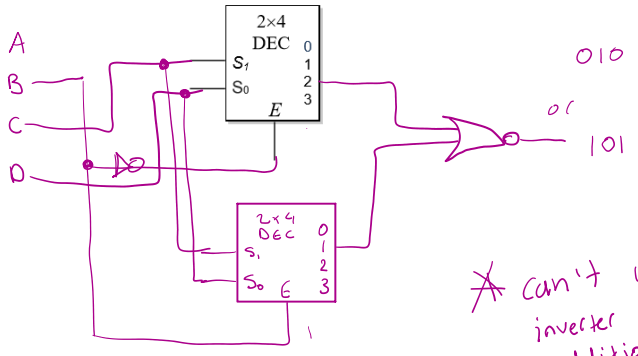


4. Implement the following Boolean function using the fewest number of **2×4 decoder with 1-enable and normal outputs**, and <u>at most two logic gates</u>.

$F(A,B,C,D) = \Sigma m\ (0,1,3,4,6,7,8,9,11,12,14,15)$ 2, 5, 10, 13

(There is a solution with two decoders and one logic gate which is easy to obtain. A more challenging solution uses one decoder and two logic gates. We will discuss the former and leave the latter as an exercise for your own attempt.)

NOR gate



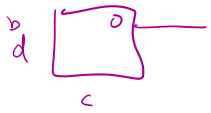
✳ can't use inverter cuz additional logic gate not allowed

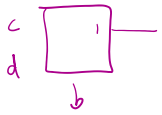
use only 1 values in E

SOP

$c' \cdot d' + b' \cdot d + b \cdot c$

$F' = b' \cdot c \cdot d' + b \cdot c' \cdot d$





cs2100_23s1_assign3qns

Wednesday, November 1, 2023 8:20 PM

cs2100_23s
1_assign3...

CS2100 Computer Organization
AY2023/24 Semester I
Assignment 3
(Deadline: 6 November 2023, Monday, 1pm)

Instructions

1. There are FOUR (4) questions in this assignment, totaling THIRTY-SEVEN (37) marks. Please do all parts of every question. Marks are indicated against each part.

2. An additional 3 marks is awarded for putting in your name, student ID and tutorial group number in your answers and for submitting in PDF format with the correct naming convention (see below). Thus, the total will be FORTY (40) marks. If you fail to do any of these, you will lose the 3 marks.

3. This assignment is due on **Monday, 6 November 2023, 1 pm.** You will be given until 1:15 pm to submit, **after which no submission will be accepted and you will receive ZERO for this assignment, regardless of how hard you've worked on it.**

4. **Plan to submit at least 2 hours early (i.e., by 11 am on 6 November)** in case there are technical issues with submission. If, by 1 pm on 6 November you are unable to submit due to technical reasons, please email your answers to Aaron at tantc@comp.nus.edu.sg before 1:15 pm. **No submission over email will be accepted after 1:15 pm.**

5. It is your responsibility to check that you have submit the correct file, and that it is complete, i.e., no missing pages.

6. Complete your answers on the provided **CS2100Assg3AnsBk.docx** file. Save it as AxxxxxxxY.pdf (where AxxxxxxxY is your Student ID) before submitting.

7. Unlike assignments 1 and 2, **you do NOT need to zip your file this time**. Just submit your pdf file. Only one file needs to be submitted. We will grade your last submitted file.

8. You should do these assignments on your own. Do not discuss the assignment questions with others.

9. Please use the QnA forums for clarifications.

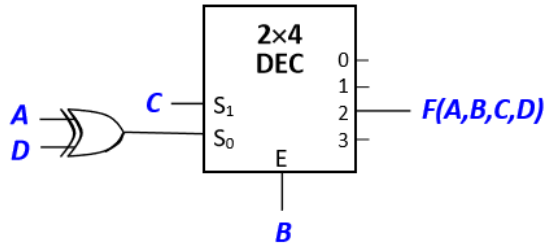
1

Note that unless otherwise stated, complemented literals are not available.

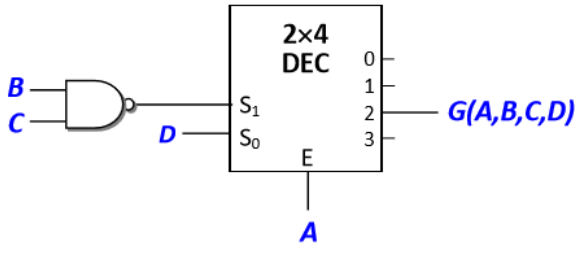
Question 1. (6 MARKS)

For each of the following implementations using 2x4 decoders with one-enable, write out the function in $\boldsymbol{\Sigma m}$ **notation** (sum-of-minterms). Pay attention to the order of the variables in the function.

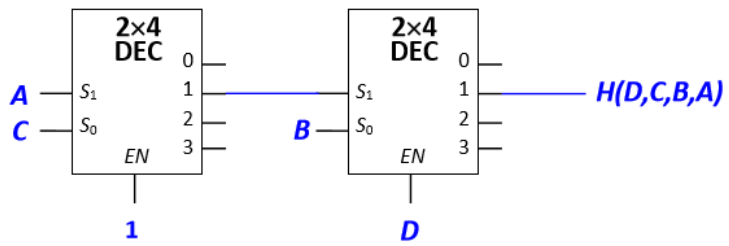
(a) (2 marks)



(b) (2 marks)



(c) (2 marks)



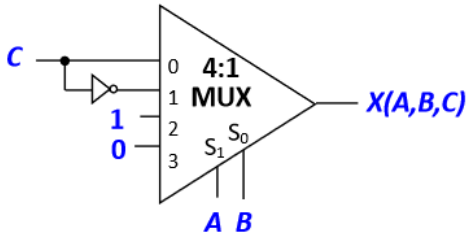
Question 2. (6 MARKS)

1 D

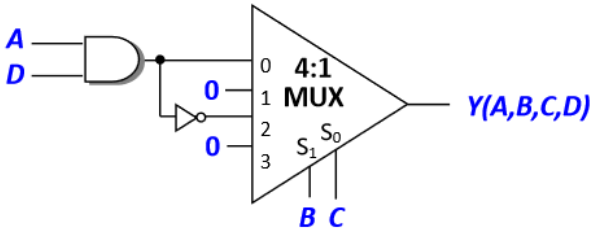
Question 2. (6 MARKS)

For each of the following implementations using 4:1 and 2:1 multiplexers, write out the function in **ΠM notation** (product-of-maxterms). Pay attention to the order of the variables in the function.

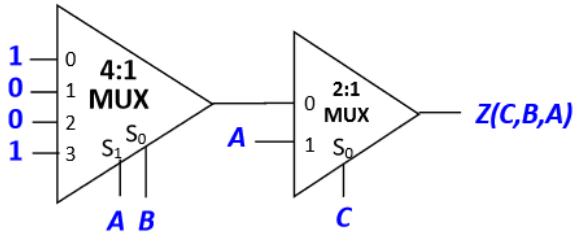
(a) (2 marks)



(b) (2 marks)

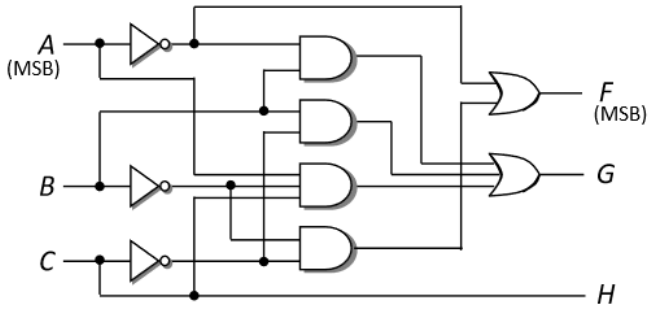


(c) (2 marks)



Question 3. (7 MARKS)

The logic circuit below is a code converter.



(a) Write the simplified SOP expressions for *F*, *G* and *H*. (3 marks)

(b) Given this statement that describes that function of the above circuit, (2 marks)

"The circuit converts a 3-bit to 3-bit "

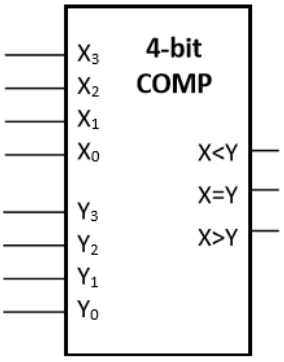
(b) Given this statement that describes that function of the above circuit, (2 marks)

"The circuit converts a 3-bit _____ to 3-bit _____."

fill in the blanks by choosing among the following choices:

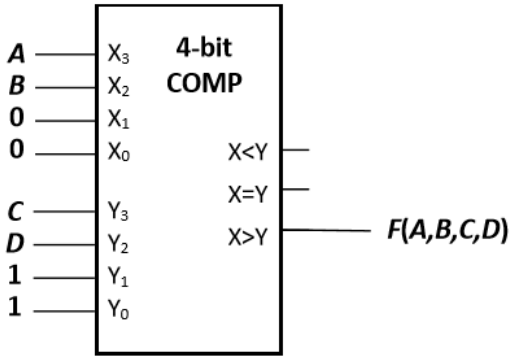
- 1's complement number
- 2's complement number
- excess 4 code
- excess 3 code
- sign-and-magnitude number

(c) Using a single 4-bit magnitude comparator as shown below, implement the function *F* of the above circuit. You are not to use any other devices or logic gates. (2 marks)



Question 4. (7 MARKS)

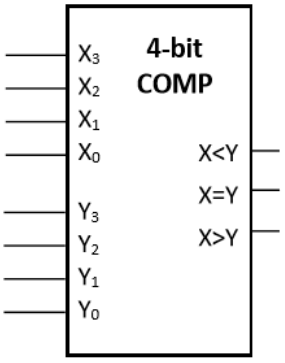
(a) Given the following 4-bit magnitude comparator and inputs, what is the simplified SOP expression of *F(A,B,C,D)*? (2 marks)



(b) Implement the following function using a single 4-bit magnitude comparator and at most one additional logic gate. (Logic gates are inverters (NOT), AND, OR, NAND, NOR, XOR, and XNOR gates. Apart from inverters, the rest are 2-input logic gates.)

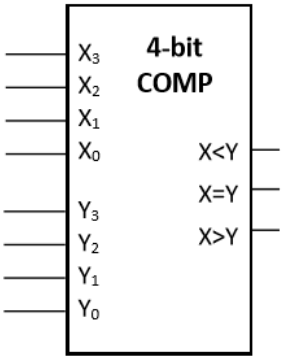
(b) Implement the following function using a single 4-bit magnitude comparator and at most one additional logic gate. (Logic gates are inverters (NOT), AND, OR, NAND, NOR, XOR, and XNOR gates. Apart from inverters, the rest are 2-input logic gates.)

$G(A,B,C,D) = \Pi M(5,13)$ (2 marks)



(c) Implement the following function using a single 4-bit magnitude comparator without any additional logic gate.

$H(A,B,C,D) = \Sigma m(1,2,3,8,9,10,11)$ (3 marks)



Question 5. (5 MARKS)

Below is a partial function table of an **8-to-3 priority encoder**. It consists of 8 inputs (A_7 to A_0) and 3 outputs (F_2 to F_0). A_i has higher priority than A_j if $i>j$. The only invalid input combination occurs when all inputs are zero. 'X' represents don't-care.

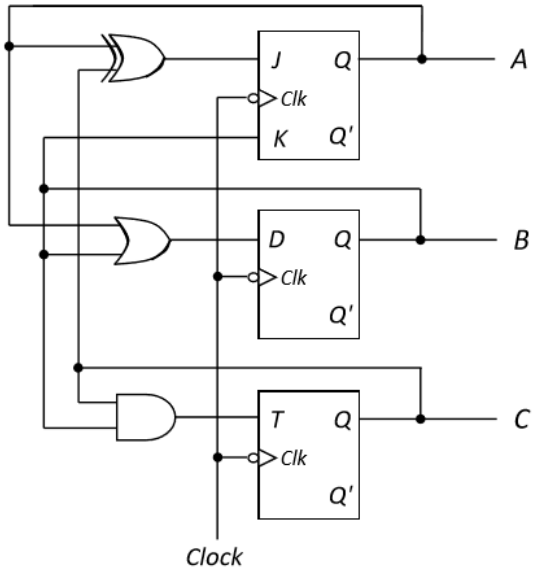
A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	F_2	F_1	F_0
0	0	0	0	0	0	0	0	X	X	X
1	X	X	X	X	X	X	X	1	1	1
0	1	X	X	X	X	X	X	1	1	0
0	0	1	X	X	X	X	X	1	0	1

1	X	X	X	X	X	X	X	1	1	1
0	1	X	X	X	X	X	X	1	1	0
0	0	1	X	X	X	X	X	1	0	1
				:					:	
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	0	0	1	0	0	0

Write out the simplified SOP expressions for F_2, F_1 and F_0.

Question 6. (6 MARKS)

Study the sequential circuit below with state *ABC*.



You may write your answers in decimal, for example, state 5 for state 101_2.

(a) If the initial state is 1 (or 001_2), what state is the circuit in after 2 clock cycles? (2 marks)

(b) If the initial state is 3 (or 011_2), what state is the circuit in after 2 clock cycles? (2 marks)

(c) Identify all the sink states in this circuit. (2 marks)

=== END OF PAPER ===

tut09qns

Thursday, November 2, 2023 10:07 PM

tut09qns

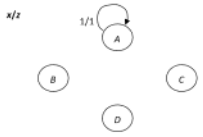
CS2100 Computer Organisation
Tutorial #9: Sequential Circuits
(Week 11: 30 October – 3 November 2023)

Discussion Questions:

D1. The state table on the right describes the state transition of a circuit with 4 states *A*, *B*, *C* and *D*, an input *x*, and an output *z*. For example, if the circuit is in state *A* and its input *x* is 0, then it moves into state *C* and generates the output 0 for *z*.

	x	
	0	1
A	*C*/0	*A*/1
B	*D*/1	*B*/0
C	*B*/1	*D*/0
D	*C*/0	*D*/0

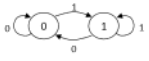
(a) Complete the state diagram below. The label of the arc indicates input/output, hence 1/1 means *x*=1 and *z*=1.



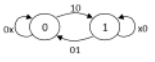
(b) Assuming that the circuit starts in state *A*, find the output sequence and state sequence for the input sequence *x* = 100010 (read from left to right). (*x* = 100010 means that initially *x* is 1, then in the next clock *x* is 0, and so on.)

D2. Match the following state diagrams to the 4 flip-flops: *JK* flip-flop, *D* flip-flop, *RS* flip-flop, and *T* flip-flop. Don't-care value is indicated by "*x*".

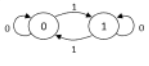
(a)



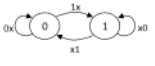
(b)



(c)

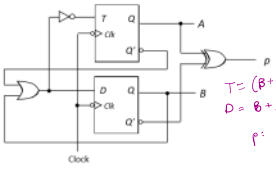


(d)



Tutorial Questions

1. A four-state sequential circuit below consists of a ***T* flip-flop** and a ***D* flip-flop**. Analyze the circuit.



$T = (B+A')' = B' \cdot A$

$D = B + A'$

$p: A \oplus B'$

(a) Complete the state table and hence draw the state diagram.
(b) Assuming that the circuit is initially at state 0, what is the final state and the outputs generated after 3 clock cycles?

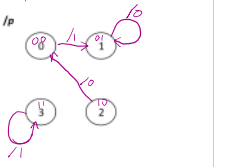
A state is called a ***sink*** if once the circuit enters this state, it never moves out of that state.

(c) How many sinks are there for this circuit?
(d) Which is likely to be an unused state in this circuit?

D flip-flops set to 1/0 / toggle
T " " no change

Present state		Output	Flip-flop inputs		Next state	
A	*B*	*p*	*TA*	*DB*	*A+*	*B+*
0	0	1	0	1	0	1
0	1	0	0	1	0	1
1	0	0	1	0	0	0
1	1	1	0	1	1	1

/p

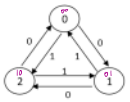


b. at state 1
output p = 1

c. 2 sinks 1, 3

d. 3 cuz no way to get to it

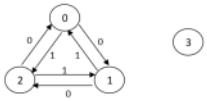
2. Given the state transition diagram on the right with states *AB* and input *x*, implement the circuit using ***JK* flip-flop**s and the fewest number of logic gates.



Fill in the state table below and draw the circuit. You do not need to follow the simplest SOP expression in your implementation as that might not give you a circuit with the fewest logic gates.

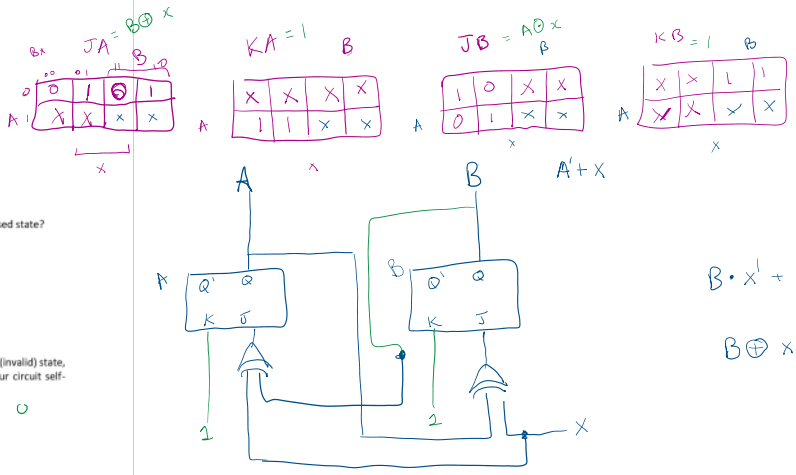
Present state		Input	Next state		Flip-flop *A*		Flip-flop *B*	
A	*B*	*x*	*A*⁺	*B*⁺	*JA*	*KA*	*JB*	*KB*
0	0	0	0	1	0	X	1	X
0	0	1	1	0	1	X	0	X
0	1	0	1	0	1	X	X	1
0	1	1	0	0	0	X	X	1
1	0	0	0	0	X	1	0	X
1	0	1	0	1	X	1	1	X
1	1	0	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X

State 3 is unused. Can you complete the following state diagram with the unused state?

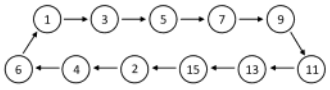


A circuit is **self-correcting** if for some reason the circuit enters into any unused (invalid) state, it is able to transit to a valid state after a finite number of transitions. Is your circuit self-correcting, and why?

yes, state 3 transitions to state 0



3. [AY2018/19 Semester 2 exam]
A sequential circuit goes through the following states, whose state values are shown in decimal:



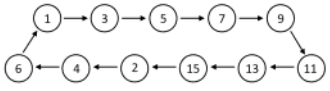
The states are represented by 4-bit values *ABCD*. Implement the sequential circuit using a *D* flip-flop for *A*, *T* flip-flops for *B* and *C*, and a *JK* flip-flop for *D*.

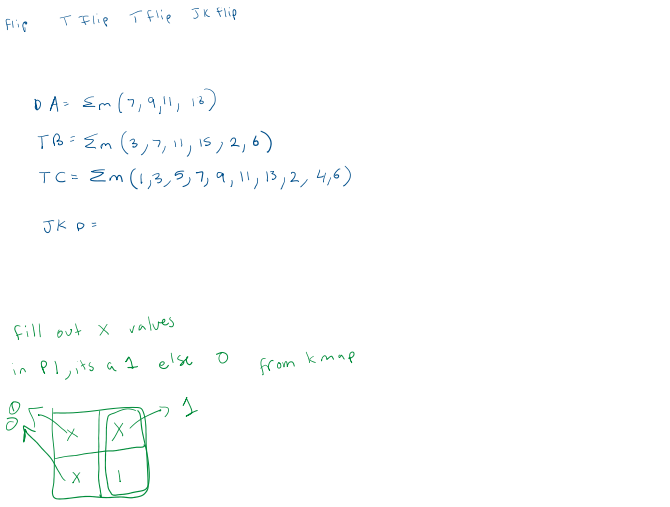
(a) Write out the **simplified SOP expressions** for all the flip-flop inputs.

(b) Implement your circuit according to your simplified SOP expressions obtained in part (a). Complete the given state diagram, by indicating the next state for each of the five unused states.

(c) Is your circuit self-correcting? Why?

yes



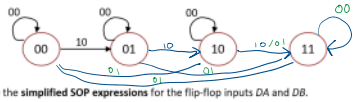


4. Pokemone Theme Park offers locker rental to its visitors. Visitors may purchase two types of token: Pokemoney $1 (P$1) and Pokemoney $2 (P$2). A locker's rental costs P$3. When a visitor deposits P$3 into the locker's token slot, its door will open.

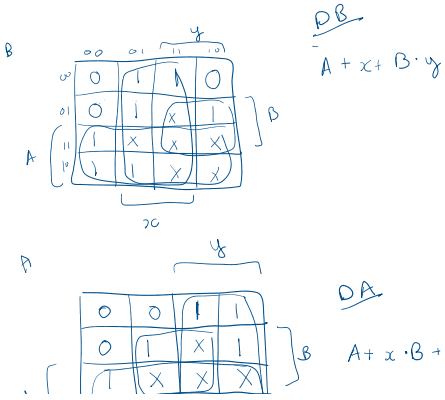
Design a sequential circuit with states *AB* for the locker's door using *D* flip-flops. The circuit consists of 4 states representing the amount a visitor has deposited: 0, 1, 2 and 3 (or, in binary, *AB* = 00, 01, 10 and 11). The visitor can deposit only one token at a time. When the circuit reaches the final state 3, it remains in state 3 even if the visitor continues to put tokens into the slot. When the circuit is in state 2 and the visitor deposits a P$2 token, the circuit goes into state 3.

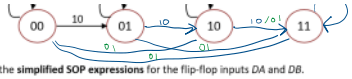
The partial state diagram is shown below. The inputs *x* and *y* represent the P$1 and P$2 tokens respectively. The label on each arrow represents *xy*.

(a) Draw and write the missing arrows and labels.

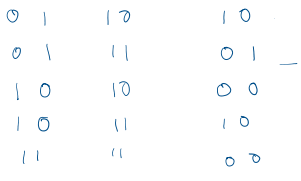


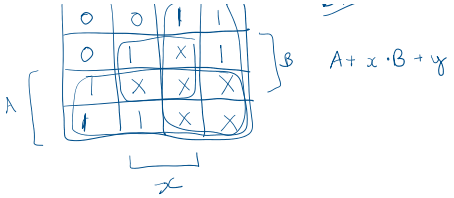
(b) Write the **simplified SOP expressions** for the flip-flop inputs *DA* and *DB*.





(b) Write the **simplified SOP expressions** for the flip-flop inputs *DA* and *DB*.





cs2100_23s1_assign3qns (1)

Friday, November 3, 2023 6:06 PM

cs2100_23s1_assign3...

CS2100 Computer Organization
AY2023/24 Semester I
Assignment 3
(Deadline: 6 November 2023, Monday, 1pm)

Instructions

1. There are FOUR (4) questions in this assignment, totaling THIRTY-SEVEN (37) marks. Please do all parts of every question. Marks are indicated against each part.
2. An additional 3 marks is awarded for putting in your name, student ID and tutorial group number in your answers and for submitting in PDF format with the correct naming convention (see below). Thus, the total will be FORTY (40) marks. If you fail to do any of these, you will lose the 3 marks.
3. This assignment is due on **Monday, 6 November 2023, 1 pm.** You will be given until 1:15 pm to submit, **after which no submission will be accepted and you will receive ZERO for this assignment, regardless of how hard you've worked on it.**
4. **Plan to submit at least 2 hours early (i.e., by 11 am on 6 November)** in case there are technical issues with submission. If, by 1 pm on 6 November you are unable to submit due to technical reasons, please email your answers to Aaron at tantc@comp.nus.edu.sg before 1:15 pm. **No submission over email will be accepted after 1:15 pm.**
5. It is your responsibility to check that you have submit the correct file, and that it is complete, i.e., no missing pages.
6. Complete your answers on the provided **CS2100Assg3AnsBk.docx** file. Save it as AxxxxxxxY.pdf (where AxxxxxxxY is your Student ID) before submitting.
7. Unlike assignments 1 and 2, **you do NOT need to zip your file this time**. Just submit your pdf file. Only one file needs to be submitted. We will grade your last submitted file.
8. You should do these assignments on your own. Do not discuss the assignment questions with others.
9. Please use the QnA forums for clarifications.

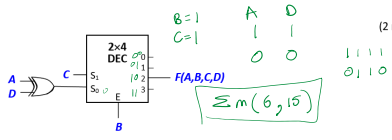
1

Note that unless otherwise stated, complemented literals are not available.

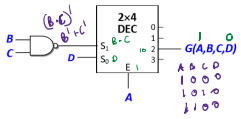
Question 1. (6 MARKS)

For each of the following implementations using 2x4 decoders with one-enable, write out the function in **Σm notation** (sum-of-minterms). Pay attention to the order of the variables in the function.

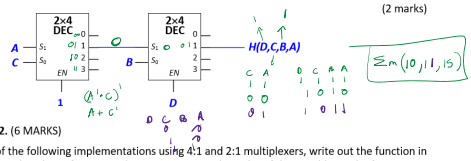
(a) (2 marks)



(b) (2 marks)



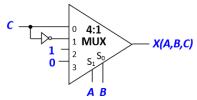
(c) (2 marks)



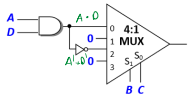
Question 2. (6 MARKS)

For each of the following implementations using 4:1 and 2:1 multiplexers, write out the function in **ΠM notation** (product-of-maxterms). Pay attention to the order of the variables in the function.

(a) (2 marks)



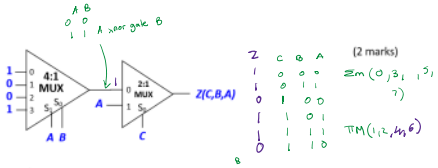
(b) (2 marks)



2

1a. m(6,15)
1b. m(14)
1c. m(10,14,15)

(c)



(2 marks)

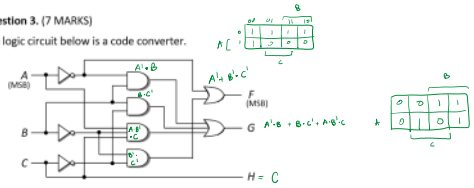
C	B	A	Z
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

$\Sigma m(0,3,5,7)$

$\Pi M(1,2,4,6)$

Question 3. (7 MARKS)

The logic circuit below is a code converter.



(a) Write the simplified SOP expressions for *F*, *G* and *H*. (3 marks)

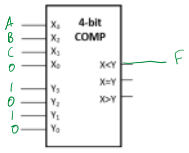
(b) Given this statement that describes that function of the above circuit, (2 marks)

"The circuit converts a 3-bit sign? mag? to 3-bit excess 4."

fill in the blanks by choosing among the following choices:

- 1's complement number
- 2's complement number
- excess 4 code
- excess 3 code
- sign-and-magnitude number X

(c) Using a single 4-bit magnitude comparator as shown below, implement the function *F* of the above circuit. You are not to use any other devices or logic gates. (2 marks)



000	1
001	1
010	1
011	1
100	1
101	0
110	0
111	0

$F = A' + B' \cdot C'$

$G = A' \cdot B + B \cdot C' + A \cdot B' \cdot C$

$H = C$

a. $F = A' + B' \cdot C'$

$G = A' \cdot B + B \cdot C' + A \cdot B' \cdot C$

A B C

1 1 1 1 0 0

1

0

ABC ABC

000 +4 100 0

001 14 101 1

0 100 +4 100

-1 101 011 011

-2 110 010

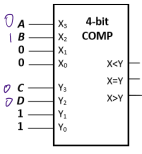
-3 111 001

011 111

$A \cdot C + B \cdot A' \cdot C' \cdot D'$

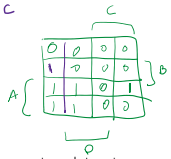
Question 4. (7 MARKS)

(a) Given the following 4-bit magnitude comparator and inputs, what is the simplified SOP expression of *F(A,B,C,D)*? (2 marks)



A > C

B > D and A = C



$B \cdot C' \cdot D'$

1100

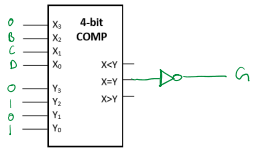
$A \cdot C'$

$+ B \cdot C' \cdot D'$

$+ A \cdot D' \cdot B$

(b) Implement the following function using a single 4-bit magnitude comparator and at most one additional logic gate. (Logic gates are inverters (NOT), AND, OR, NAND, NOR, XOR, and XNOR gates. Apart from inverters, the rest are 2-input logic gates.)

$G(A,B,C,D) = \Pi M(5,13)$ (2 marks)

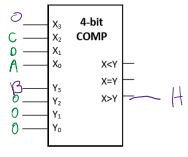


0101

1101

(c) Implement the following function using a single 4-bit magnitude comparator without any additional logic gate.

$H(A,B,C,D) = \Sigma m(1,2,3,8,9,10,11)$ (3 marks)



B C D A

0001

0010

0100

0011

0110

1000

1001

1010

1011

0001

0011

0101

0111

B < D A

$A_7 + A_6 + A_5 A_4' A_3 + A_5 A_4' A_2 = F_1$

5.)

Question 5. (5 MARKS)

Below is a partial function table of an **8-to-3 priority encoder**. It consists of 8 inputs (A_7 to A_0) and 3 outputs (F_2 to F_0). A_i has higher priority than A_j if $i>j$. The only invalid input combination occurs when all inputs are zero. 'X' represents don't-care.

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	F_2	F_1	F_0
0	0	0	0	0	0	0	0	X	X	X
1	X	X	X	X	X	X	X	1	1	1
0	1	X	X	X	X	X	X	1	1	0
0	0	1	X	X	X	X	X	1	0	1
					⋮				⋮	
0	0	0	0	0	0	1	X	0	0	1

6 5

$F_2 = A_4 + A_5 + A_6 + A_7$

$F_1 = A_2 + A_3 + A_6 + A_7$

$F_0 = A_1 + A_3 + A_5 + A_7$

$A_7 + A_5 A_6' A_7' + A_3 A_4' A_5' A_6' A_7' + A_1 A_2' A_3' A_4' A_5' A_6' A_7'$

$A_7 + A_5 A_6' A_7' + A_3 A_4' A_5' A_6' + A_1 A_2' A_3' A_4' A_5' A_6'$

$A_7 + A_5 A_6' A_7' + A_3 A_4' A_5' A_6' + A_1 A_2' A_3' A_4' A_6'$

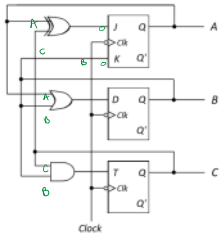
$A_7 + A_5 A_6' + A_3 A_4' A_6' + A_1 A_2' A_4' A_6'$

0	0	0	0	0	0	0	0	X	X	X
1	X	X	X	X	X	X	X	1	1	1
0	1	X	X	X	X	X	X	1	1	0
0	0	1	X	X	X	X	X	1	0	1
				:					:	
0	0	0	0	0	0	1	X	0	0	1
0	0	0	0	0	0	0	1	0	0	0

Write out the simplified SOP expressions for F_2, F_1 and F_0.

Question 6. (6 MARKS)

Study the sequential circuit below with state *ABC*.



You may write your answers in decimal, for example, state 5 for state 101_2.

(a) If the initial state is 1 (or 001_2), what state is the circuit in after 2 clock cycles? (2 marks)

(b) If the initial state is 3 (or 011_2), what state is the circuit in after 2 clock cycles? (2 marks)

(c) Identify all the sink states in this circuit. (2 marks)

=== END OF PAPER ===

5

$A_7 + A_5A_6' + A_3A_4'A_5'A_6' + A_1A_2'A_3'A_4'A_5'A_6'$

$A_7 + A_5A_6' + A_3A_4'A_6' + A_1A_2'A_3'A_4'A_6'$

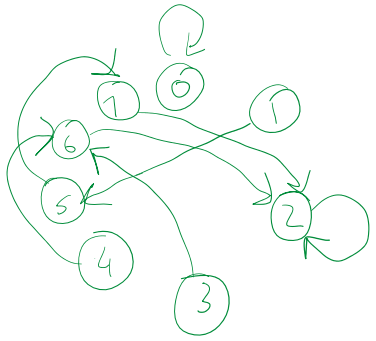
$A_7 + A_5A_6' + A_3A_4'A_6' + A_1A_2'A_4'A_6'$

a.

001		0
101		1
111		2

b.

JK		
11	011	0
11	110	1
11	010	2



	initial	JA	KA	D	T	output A B C
0	000	0	0	0	0	0 0 0
1	001	1	0	0	0	1 0 1
2	010	0	1	1	0	0 1 0
4	100	1	0	1	0	1 1 0
3	011	1	1	1	1	1 1 0
6	110	1	1	1	0	0 1 0
5	101	0	0	1	0	1 1 1
7	111	0	1	1	1	0 1 0

tut10qns

Wednesday, November 8, 2023 11:03 AM

tut10qns

CS2100 Computer Organization
Tutorial 10: Pipelining
(Week 12: 6 Nov – 10 Nov 2023)

1. Let's finish what we started... design a Boolean circuit that will decode the following control bits for the MIPS **add, and, beq, lw, sw, nor, or, slt, sub** instructions. You may use AND, OR, NAND, NOR, XOR gates with up to 4 inputs and 'NOT' must be explicitly performed. Also the opcode should be labelled $op_5op_4op_3op_2op_1op_0$ – where op_i is the i-th bit.

   a) RegDst looking for r-type Nor gate on all 6
   b) MemtoReg only lw is 1 I can invert 1000N zeros and AND
   c) MemWrite only sw is 1, 0x2B
   d) ALUOp 10 for r-type / 01 for beq / 00 for lw sw first bit same as regdst
   e) ALU Control

2. Using Powerpoint animation (or just a number of slides), give the full details of the signals and data bits (highlighting those lines that are active) of the following 3 instructions are executed together in a pipelined MIPS (tut10.ppt, slide 2):

   a) `sub $1, $2, $3`
   b) `lw $6, 4($7)`
   c) `beq $4, $5, L2`

   Assume that L2 at PC=0x100 instruction while the beq is at PC=0x1000.

3. Consider the following sequence of MIPS instructions:

   RAW $1

```
add $1, $2, $3 # PC = 0x100
add $1, $1, $3 # PC = 0x104
add $1, $1, $1 # PC = 0x108
```

   In the datapath with forwarding (tut10.ppt, slide 3 – and you may need to refer to slide 4 for more detailed marking of the lines, which unfortunately are messy and may not quite line up), trace the execution of these 3 instructions as they pass through the pipeline with attention paid especially to the forwarding of operands. Clearly identify which rule(s) of the forwarding (or hazard detection) was fired, if any.

4. Repeat Q3 for the following instruction sequence:

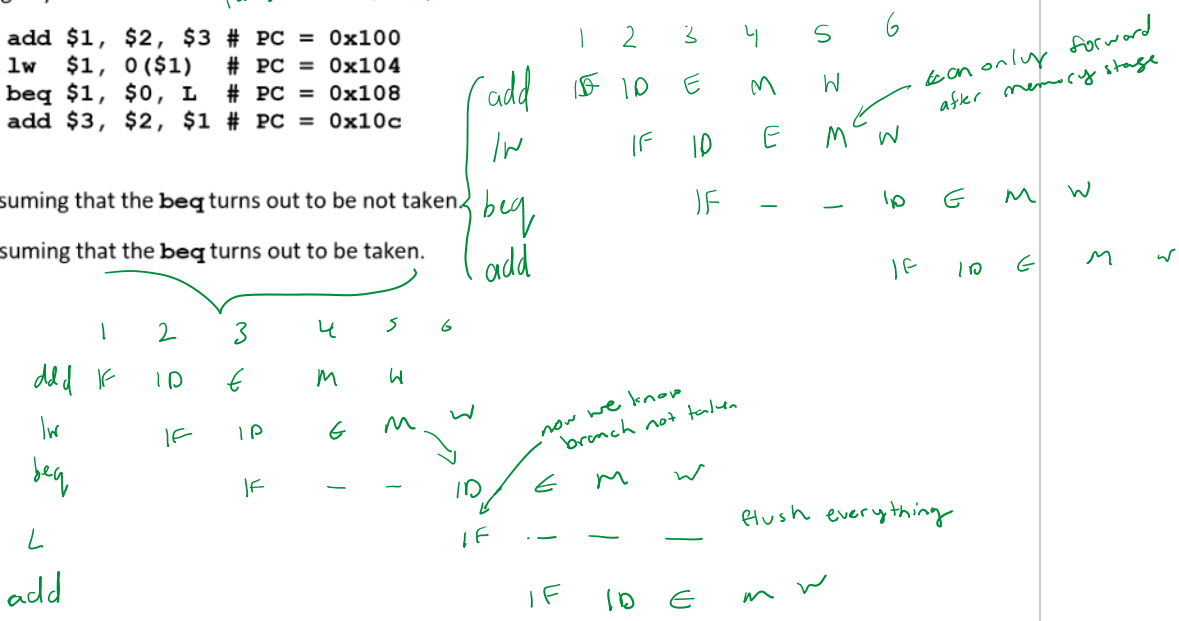
```
add $1, $2, $3 # PC = 0x100
lw  $1, 0($1)  # PC = 0x104
add $1, $1, $1 # PC = 0x108
add $3, $2, $1 # PC = 0x10c
```

5. Using the graphical notation for pipeline introduced in class, show how the following sequence of instructions would be exeucted in a datapath with forwarding, and branch resolution at the ID stage by

```
add $1, $2, $3 # PC = 0x100
lw  $1, 0($1)  # PC = 0x104
beq $1, $0, L  # PC = 0x108
add $3, $2, $1 # PC = 0x10c
```

(a) assuming that the **beq** turns out to be not taken.

(b) assuming that the **beq** turns out to be taken.



tut11qns

Friday, November 17, 2023 7:57 AM

tut11qns

CS2100 Computer Organization
Tutorial 11: Caching
(Week 13: 13 – 17 Nov 2023)

1. A byte-addressed machine with a word size of 16 bits and address width of 32 bits has a direct-mapped cache with 16 blocks and a block size of 2 words.

   (a) Given a sequence of memory references as shown below, where each reference is given as a byte address in both decimal and hexadecimal forms, indicate whether the reference is a hit (H) or a miss (M). For a miss, also identify the type of miss it is (i.e., cold, conflict or capacity.) Assume that the cache is initially empty.

Memory address		Hit (H) or Miss (M)?
(in decimal)	(in hexadecimal)	
4	4 0000 0100	M (cold)
92	5C 0101 1100	M (cold)
7	7 0000 0111	~~M (conflict)~~ Hit
146	92 1001 0010	M (cold)
30	1E 0001 1110	M (cold)
95	5F 0101 1111	Hit
176	B0 1011 0000	M (cold)
93	5D 0101 1101	Hit
145	91 1001 0001	~~Hit~~ M (conflict)
264	108 0001 0000 1000	M (cold)
6	6 0000 0110	Hit

   (b) Given the above sequence of memory references, fill in the final contents of the cache. Use the notation M[*i*] to denote the word at memory address *i*.

Index	Tag value	Word 0	Word 1
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

   (c) Repeat (a) and (b) if the cache is instead a two-way set associative cache while having the same block and overall size.

   (d) Repeat (a) and (b) if the cache is instead a fully associative cache while having the same block and overall size.

2. A machine with byte addresses and a word size of 32 bits and address width of 32 bits has a direct-mapped data cache with 4 blocks each consisting of 2 words.

   (a) Given the MIPS program below, and assuming that array A starts at memory location 0x1000 while array B starts at memory location 0x4010. Fill in the first 10 address requests seen at the data cache and indicate whether the reference is a hit (H) or a miss (M). Assume that the cache is initially empty.

```
start:
      la   $s0, A                 #PC=0x100
      la   $s1, B                 #PC=0x104
      li   $t0, 1                 #PC=0x108
loop:
      slt  $t1, $t0, 1000         #PC=0x10c
      beq  $t1, $zero, end_loop   #PC=0x110
      sll  $t2, $t0, 2            #PC=0x114
      add  $t3, $s0, $t2          #PC=0x118
      lw   $a0, 0($t3)            #PC=0x11c
      add  $t4, $s1, $t2          #PC=0x120
      lw   $a1, 0($t4)            #PC=0x124
      add  $v0, $a0, $a1          #PC=0x128
      sw   $v0, -4($t3)           #PC=0x12c
      addi $t0, $t0, 1            #PC=0x130
      j    loop                   #PC=0x134
end_loop:
      .    .    .
```

   (b) Given the above program, fill in the final contents of the cache. Use the notation M[*i*] to denote the word at memory address *i*. (*i* may be in hexadecimal.)

   (c) What is the total number of data cache memory references, hits and misses after the execution of the above MIPS program?

3. Suppose a (32 bit) MIPS processor has an *instruction* cache that is also direct mapped with 4 blocks each consisting of 2 words. Using the same program in Question 2, answer the following:

   (a) Fill in the final contents of this cache assuming that **start** is at location 0x100 and each of the assembler pseudo-instructions (like **la** and **li**) also occupy 32 bits. You can use "PC=..." to indicate the word content of the cache.

   (b) What is the miss rate in this case?

4. Using your knowledge of Boolean circuits, produce a circuit (the simpler the better) that will check the Valid and Tag bits of a cache block to determine if it is a hit (1) or a miss (0). For simplicity, assume that tags are 20 bits long. Limit each gate to at most 4 inputs. Do not use adders. Suggestion: Use Logisim for easier replications.

5 comparators ? ==

exam21s2_qns

Tuesday, November 21, 2023 3:22 PM

exam21s2_qns

NATIONAL UNIVERSITY OF SINGAPORE

CS2100 – COMPUTER ORGANISATION

(Semester 2: AY2021/22)

Time Allowed: 2 Hours

INSTRUCTIONS TO STUDENTS

1. This assessment paper consists of **SEVENTEEN (17)** questions in **THREE (3)** parts and comprises of **THIRTEEN (13)** printed pages.
2. Answer ALL questions on the **ANSWER SHEETS**.
3. This is an **OPEN BOOK** assessment.
4. Write your answers only on the **ANSWER SHEETS**. You may write in pen or pencil. You are to write within the space provided. No extra pages should be submitted.
5. Printed/written materials are allowed. Apart from calculators, electronic devices are not allowed.
6. Page 13 contains the MIPS Data Reference sheet.
7. The maximum mark of this assessment is 100.

Question	Max. mark
Part A: Q1 – 6	12
Part B: Q7 – 12	18
Part C: Q13	12
Part C: Q14	16
Part C: Q15	13
Part C: Q16	13
Part C: Q17	16
Total	**100**

——— END OF INSTRUCTIONS ———

Part A: Multiple-Choice Questions [Total: 6×2=12 marks]

Each multiple-choice question (MCQ) is worth **TWO marks** and has exactly **one** correct answer. Please write your answers in **CAPITAL LETTERS**.

1. What is $(20.22)_4$ in decimal?
   A. 8.75
   B. 8.625
   C. 8.5
   D. 8.25
   E. None of the above.

2. Consider the following IEEE 754 single-precision floating point number written in hexadecimal: **0x42022000**. What is the number in decimal?
   A. 130.125
   B. 65.0625
   C. 32.53125
   D. 16.265625
   E. None of the above

3. Given that the first print (*i.e.*, ***printf #1***) is "123 321" (*without the quotes*) and the second print (*i.e.*, ***printf #2***) is "123 654" (*without the quotes*). What is the correct definition of the struct **data_t** assuming that **data_t data** variable is correctly declared and initialised at the start of main function?

```
#include <stdio.h>
typedef struct { /* blank for question */ } data_t;
void foo(data_t);
int main() {
  /* data_t data is declared correctly here */
  printf("%d %d\n", data.x[0], data.y[0]);  /* printf #1 */
  foo(data);
  printf("%d %d\n", data.x[0], data.y[0]);  /* printf #2 */
  return 0;
}
void foo(data_t data) {
  data.x[0] = 456;
  data.y[0] = 654;
}
```

   A. `typedef struct { int x[1]; int y[1]; } data_t;`
   B. `typedef struct { int *x  ; int y[1]; } data_t;`
   C. `typedef struct { int *x  ; int *y  ; } data_t;`
   D. `typedef struct { int x[1]; int *y  ; } data_t;`
   E. None of the above.

4. If we were to add "**NAND $rd, $rs, $rt**" operation into our MIPS instructions, given our processor implementation, what will be the ALUControl signal needed?
   - (A) 1101
   - B. 1110
   - C. 1100
   - D. 1111
   - E. None of the above.

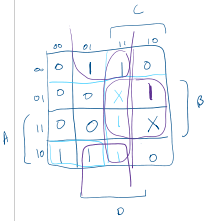
$(A \cdot B)' = A' + B'$

A invert 1

B invert 1

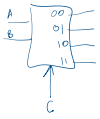
or is 01

5. Given the Boolean function $F(A,B,C,D) = \Sigma m(1,3,6,8,9,11,15) + \Sigma x(7,14)$ where x denotes don't-care, which of the following is not an EPI in the K-map of F?
   - (A) $C \cdot D$
   - B. $B \cdot C$
   - C. $B' \cdot D$
   - D. $A \cdot B' \cdot C'$ ✓EPI
   - E. None of the above.



6. You are given a single 2×4 decoder with 1-enable and active high output, and no other devices or logic gates. Which of the following expressions cannot be implemented with this single decoder, where A, B and C are Boolean variables?
   - A. $A \cdot B \cdot C$ ✓
   - B. $A' \cdot B \cdot C$
   - C. $A \cdot B' \cdot C$
   - (D) $A' \cdot B' \cdot C'$
   - E. None of the above.

cuz we don't have negator and we need 1 in enable



Part B: Multiple-Response Questions [Total: 6×3=18 marks]

Each multiple-response question (MRQ) is worth **THREE marks** and may have one answer or multiple answers. Write out **all** correct answers. For example, if you think that A, B, C are the correct answers, write A, B, C. Please write in **CAPITAL LETTERS**.

Only if you get all the answers correct will you be awarded three marks. **No partial credit will be given for partially correct answers.**

7. Which of the following is **equivalent** to the length of the string `s` as returned by `strlen(s)`?
   - (A) The index of the first null character in the string `s`.
   - B. The number specified during the definition of variable `s` (*e.g.*, `10` in `char s[10]`).
   - C. The amount of memory allocated to the string `s`.
   - (D) The difference of the address of the first null character in the string `s` and value of string `s`.
   - E. Twice the average of the address of the first null character in the string `s` and the value of string `s`.

8. Aiken is running the following MIPS program. What are the *possible* number of instructions executed **in total** assuming that the program does not result in an error, if you can start with any starting value of $s1 and $s2 and with any possible memory content?

```
      addi $t0, $s1  , 0
      addi $t1, $zero, 1
      addi $t2, $s2  , 0
loop: lb   $t3, 0($s2)
      andi $t4, $t3  , 0x1
      beq  $t4, $zero, else
      sll  $t1, $t1  , 1
else: addi $t0, $t0  , 1
      addi $s2, $s2  , 1
      bne  $t3, $zero, loop
```

   - (A) 9
   - B. 14 X
   - (C) 15
   - (D) 43
   - (E) 63

9. Consider the instruction "**lb $rt, imm($rs)**" in general. What is true about this instruction?
   - A. The value specified by **imm** must be a multiple of 4.
   - B. The value in the register specified by **$rs** must be a multiple of 4.
   - C. The sum of **imm** and the value in the register specified by **$rs** must be a multiple of 4.
   - D. The sum of **imm** and the value in the register specified by **$rs** can be any value.
   - (E) The value in the register specified by **$rs** can be any bit pattern.

10. A "*no-operation*" typically written as **nop** is an instruction that tells the processor to do nothing. However, we can simulate this using other instructions. Instead of telling the processor to do nothing, what we want is simply to have the processor produce "*no effect*". In other words, no registers or memory addresses can have their value *changed* (*i.e.*, if the value before the operation is n, then the value after the operation is also n).

    Which of the following operations are valid and produce no effect? You may assume that any label is valid.

    A. `bne $0, $0, label`
    B. `addi $0, $0, 0`
    C. `sll $2, $2, 0`
    D. `add $0, $0, $0` ✗
    E. `srl $4, $4, 0`

11. Which of the following statements are true about the control signal in our processor implementation and our supported MIPS instructions in Lecture 12?

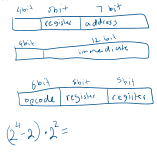
    A. If `RegWrite = 0`, it is actually *possible* to have a different implementation where there are 3 other control signals having the value of "don't care" for at least 1 instruction.
    B. It is possible to have the value of `MemRead` equals to `MemWrite` in some instructions.
    C. If the value of `MemRead` is not equal to `MemWrite`, then the value of `ALUSrc` is always `1`.
    D. There is an instruction where Branch is the only control signal that is equal to `1`.
    E. `sw` instruction may have the value of `MemRead` changed to X since the result is not going to be written into a register.

12. Consider a machine with word size of 4 bytes and 16-bit fixed-length instructions with three types of instructions as shown below. Note that although Type A and Type B have the same number of bits for opcode, they are considered different instruction types because they have different kinds of operands.

    - **Type A**: 4-bit opcode, 1 address and 1 register;
    - **Type B**: 4-bit opcode and 1 immediate;
    - **Type C**: 6-bit opcode and 2 registers.

    Assume all bits are utilised and all instructions types exists, each *address* holds a **word** instead of a byte, the immediate field is in 2s complement, all addresses are used and all registers can be encoded. Which of the following statements are true about the instruction?

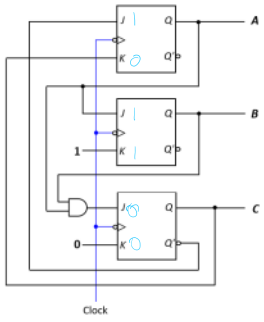
    A. The maximum number of Type A instructions is 15.
    B. The maximum number of Type C instructions is 56.
    C. There are a total of 512 bytes of memory.
    D. The maximum value of the immediate field is 2047. ✗
    E. The maximum number of registers is 16. ✗



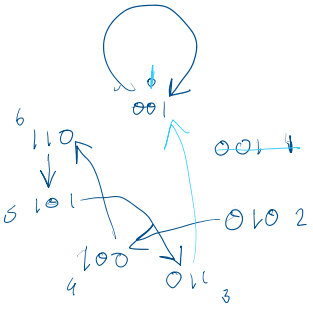
Part C: There are 5 questions in this part [Total: 70 marks]

Q13. Sequential circuits [12 marks]

(a) A sequential circuit with 6 states: state 1 (ABC=001$_2$) through state 6 (ABC=110$_2$) is implemented using three *JK* flip-flops as shown below. Complete the state diagram on the Answer Sheets. One of the transitions on the state diagram has been drawn for you. [5 marks]



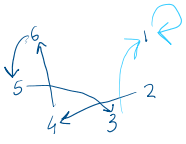
old							New	
A B C	$\bar{J}A$	KA	$\bar{J}B$	KB	$\bar{J}C$	KC	A B C	
0 0 1	0	1	0	1	0	0	0 0 1	
0 1 0	1	0	0	1	0	0	1 0 0	
1 0 0	1	0	1	1	0	0	1 1 0	
0 1 1	0	1	0	1	0	0	0 0 1	
1 1 0	1	0	1	1	1	0	1 0 1	
1 0 1	0	1	1	1	0	0	0 1 1	
0 0 0	1	0	0	1	0	0	1 0 0	
1 1 1	0	1	1	1	1	0	0 0 1	

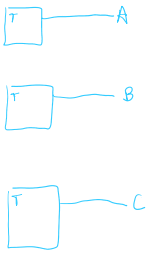


(b) Is the circuit self-correcting? Explain. (Mark will not be awarded if no explanation is given or the explanation given is incomplete/incorrect.) [1 mark]

yes, the unused states 0, 7 change to state 4 / 1 respectively

(c) Redesign the circuit using only ***T* flip-flops**. You do not have to follow where the unused states transit to in the given circuit above. That is, you only need to make sure that the transitions from the used states follow the above circuit. You do not need to draw your new design. Write out the flip-flop input functions *TA*, *TB* and *TC* so that your new design can be implemented with the fewest number of logic gates other than the flip-flops. [6 marks]



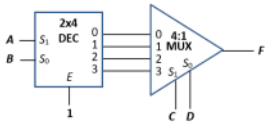


old	A⊕C	A⊕B	A·B	New
A B C	TA	TB	TC	A B C
0 0 1	0	0	0	0 0 1
0 1 0	1	1	0	1 0 0
1 0 1	1	1	0	0 1 1
1 1 0	0	1	1	1 0 1
1 0 0	0	1	0	1 1 0
0 1 1	0	1	0	0 0 1

Q14. Combinational circuits [16 marks]

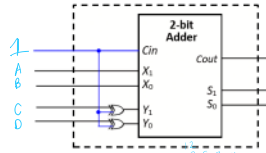
Note that logical constants 0 and 1 are available, but not complemented literals.

(a) Given the following circuit which comprises a 2×4 decoder with 1-enable and active high outputs and a 4:1 multiplexer, write out the sum-of-minterms expression of $F(A,B,C,D)$ in the Σm notation. [4 marks]



(b) The circuit below comprises a 2-bit parallel adder and 2 XOR gates. The circuit is housed inside a chip so the only connections available are those that extend out of the dotted box.

Using this circuit and one additional logic gate (inverter, AND, OR, NAND, NOR, XOR, or XNOR), implement the function F in part (a) above. [4 marks]



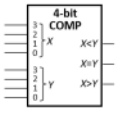
(c) [Total: 8 marks]

Given this Boolean function: $G(A,B,C,D) = \Sigma m(0, 4, 6, 8, 9, 10, 12, 13, 14, 15)$.

(i) How many PIs and EPIs are there in the K-map of G? [2 marks]

(ii) Write the simplified SOP expression for G. [2 marks]

(iii) Implement G using at most two 4-bit magnitude comparators and a 2-input OR gate. The block diagram of a 4-bit magnitude comparator is shown below. [4 marks]



Q15. MIPS [13 marks]

Study the following MIPS code on integer arrays A and B, with array B containing twice as many elements as array A. The following are the variable mappings:

- $a0 = *size* (number of elements in array A)
- $a1 = base address of array A
- $a2 = base address of array B

```
.data
size: .word 5
A: .word 1, 2, 3, 4, 5
B: .word 5, 3, 4, 5, 3, 8, 2, 5, 9, 4

.text
main: la   $t0, size       # $t0 is the address of size
      lw   $a0, 0($t0)     # $a0 is the content of size
      la   $a1, A          # $a1 is the base address of array A
      la   $a2, B          # $a2 is the base address of array B
# ------------------------------------------------------------
      add  $t0, $0, $0     # I1; i = 0
      addi $t1, $a1, 0     # I2; $t1 = &A[0]
      addi $t2, $a2, 0     # I3; $t2 = &B[0]
      sll  $t3, $a0, 2     # I4
loop: slt  $t4, $t0, $t3   # I5
      beq  $t4, $0, end    # I6
      lw   $s1, 0($t1)     # I7
      lw   $s2, 0($t2)     # I8
      slt  $t4, $s1, $s2   # I9
      beq  $t4, $0, skip   # I10
      add  $t9, $s1, $0    # I11
      add  $s1, $s2, $0    # I12
      add  $s2, $t9, $0    # I13
skip: sw   $s1, 0($t1)     # I14
      sw   $s2, 0($t2)     # I15
      addi $t0, $t0, 4     # I16
      addi $t1, $t1, 4     # I17
      addi $t2, $t2, 8     # I18
      j    loop            # I19
# ------------------------------------------------------------
end:  li   $v0, 10         # system call code for exit
      syscall
```


Q15. (continue...)

(a) Write out the contents of array B after the execution of the MIPS code. [2]

(b) Using the variable names (*size*, *A*, *B*) shown in the variable mappings above, write an equivalent C code corresponding to instructions I1 to I19 in the above MIPS code. You may use additional variable(s) if needed.

Q15. (continue...)

(a) Write out the contents of array *B* after the execution of the MIPS code. [2]

(b) Using the variable names (*size*, *A*, *B*) shown in the variable mappings above, write an equivalent C code corresponding to instructions I1 to I19 in the above MIPS code. You may use additional variable(s) if needed.

Do not do a line-by-line direct translation of the MIPS code. You do not need to declare the variables in your C code. [4]

(c) Write the instruction encoding of instruction I5 (`slt $t4, $t0, $t3`) in hexadecimal. The value in **shamt** for `slt` instructions is zero. [2]

(d) Assuming that I19 (`j loop`) is stored at address **0x0040 0084**, (i) calculate the address of I5 (`slt $t4, $t0, $t3`) and (ii) write the instruction encoding of I19 (`j loop`) in hexadecimal. [2]

(e) Change the code from I11 to I15 to make the code more efficient. Make the changes on the Answer Sheets. Except for moving the labels if necessary, you are not to change the code outside of I11 to I15. [3]

Q16. Pipelining [13 marks]

Refer to the following MIPS code which is the same as the one in question 15. Here, we look only at instructions I1 to I19. Pay attention to the assumptions (underlined) given below.

```
       add  $t0, $0, $0     # I1; i = 0
       addi $t1, $a1, 0     # I2; $t1 = &A[0]
       addi $t2, $a2, 0     # I3; $t2 = &B[0]
       sll  $t3, $a0, 2     # I4
loop:  slt  $t4, $t0, $t3   # I5
       beq  $t4, $0, end    # I6
       lw   $s1, 0($t1)     # I7
       lw   $s2, 0($t2)     # I8
       slt  $t4, $s1, $s2   # I9
       beq  $t4, $0, skip   # I10
       add  $t9, $s1, $0    # I11
       add  $s1, $s2, $0    # I12
       add  $s2, $t9, $0    # I13
skip:  sw   $s1, 0($t1)     # I14
       sw   $s2, 0($t2)     # I15
       addi $t0, $t0, 4     # I16
       addi $t1, $t1, 4     # I17
       addi $t2, $t2, 8     # I18
       j    loop            # I19
end:
```

Assuming a 5-stage MIPS pipeline, and all elements in array *A* are smaller than all elements in array *B*, answer the parts below. You need to count until the last stage of instruction I19.

(a) How many cycles does this code segment take to complete its execution in the first iteration (I1 to I19) in an ideal pipeline, that is, one with no delays? [2]

For parts (b) to (d) below, given the assumption for each part, how many additional cycles does this code segment (I1 to I19) take to complete its execution in the first iteration as compared to an ideal pipeline computed in (a)? Note that the jump instruction (j) computes the target address to jump to in its ID stage (stage 2). No delayed branching is used.

Write the total number of additional delay cycles for each of the parts (b) to (d). For example, if part (a) takes 10 cycles and part (b) takes 30 cycles, then you should write +20 for part (b).

(b) Assuming without forwarding and branch decision is made at MEM stage (stage 4). No branch prediction is made. [3]

(c) Assuming with forwarding and branch decision is made at MEM stage (stage 4). No branch prediction is made. [3]

(d) Assuming with forwarding and branch decision is made at ID stage (stage 2). Branch is predicted not taken. [3]

(e) Assuming the setting in part (b) above (without forwarding and branch decision at MEM stage), without affecting the correctness of the code, is it possible to move one instruction to somewhere else to reduce the number of delay cycles? If so, indicate which instruction to move, where to move it to, and how many delay cycles are reduced by moving it. If it is not possible, explain. [2]

Q17. Cache [16 marks]

Refer to the following MIPS code which is the same as the one in question 15. Here, we look only at instructions I1 to I19. The data segment in the MIPS code in question 15 no longer applies here as the arrays contain a lot more elements in this question.

```
      add  $t0, $0, $0     # I1; i = 0
      addi $t1, $a1, 0     # I2; $t1 = &A[0]
      addi $t2, $a2, 0     # I3; $t2 = &B[0]
      sll  $t3, $a0, 2     # I4
loop: slt  $t4, $t0, $t3   # I5
      beq  $t4, $0, end    # I6
      lw   $s1, 0($t1)     # I7
      lw   $s2, 0($t2)     # I8
      slt  $t4, $s1, $s2   # I9
      beq  $t4, $0, skip   # I10
      add  $t9, $s1, $0    # I11
      add  $s1, $s2, $0    # I12
      add  $s2, $t9, $0    # I13
skip: sw   $s1, 0($t1)     # I14
      sw   $s2, 0($t2)     # I15
      addi $t0, $t0, 4     # I16
      addi $t1, $t1, 4     # I17
      addi $t2, $t2, 8     # I18
      j    loop            # I19
end:
```

For parts (a) and (b): You are given a **2-way set-associative instruction cache** with 16 words in total. The replacement policy is **LRU** (least recently used). You may assume the following:

- There are **100 elements** in array *A* and twice as many elements in array *B*.
- All elements in array *A* are smaller than all elements in array *B*.
- Instruction I1 is stored at address **0x4488 FFFC**.
- Consider only instructions I1 to I19 in the execution of the code.

(a) Assume that each block contains 2 words.

(i) How many bits are there in the set index field? In the byte offset field? [2]

(ii) How many misses in total are there in the execution of the code? [2]

(b) Assume that each block contains 4 words.
How many misses in total are there in the execution of the code? [2]

Q17. Cache (continue...)

For parts (c) to (e): You are given a **direct-mapped data cache** with 1024 words in total and each block contains 16 words. Recall that array *B* contains twice as many elements as array *A*. You may assume the following:

- Array *A* starts at address **0xFFFF 0000**.
- Array *B* follows immediately after array *A* in the memory. That is, if the last element of array *A* is at address *x*, then the first element of array *B* is at address (*x* + 4).
- Only **lw** instructions are considered for the calculation of hits and misses; **sw** instructions are to be excluded from the calculation.

(c) How many bits are there in the index field? In the byte offset field? [2]

(d) Assuming that array *A* contains **512** elements, how many data access hits in total are in the data cache in the execution of the code (i) for array *A* and (ii) for array *B*? [4]

(e) Assuming that array *A* contains **1024** elements, how many data access hits in total are in the data cache in the execution of the code (i) for array *A* and (ii) for array *B*? [4]

=== END OF PAPER ===

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

MIPS Reference Data

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)	NOTE	OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20hex
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8hex
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9hex
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21hex
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24hex
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	chex
Branch On Equal	beq	I	if(R[rs]== R[rt]) PC=PC+4+BranchAddr	(4)	4hex
Branch On Not Equal	bne	I	if(R[rs]!= R[rt]) PC=PC+4+BranchAddr	(4)	5hex
Jump	j	J	PC=JumpAddr	(5)	2hex
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3hex
Jump Register	jr	R	PC=R[rs]		0 / 08hex
Load Byte Unsigned	lbu	I	R[rt] = {24'b0,M[R[rs]+SignExtImm](7:0)}	(2)	24hex
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2)	25hex
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30hex
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		fhex
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23hex
Nor	nor	R	R[rd] = ~(R[rs] | R[rt])		0 / 27hex
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25hex
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	dhex
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / 2ahex
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	ahex
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	bhex
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / 2bhex
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00hex
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		0 / 02hex
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28hex
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (atomic) ? 1 : 0	(2,7)	38hex
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29hex
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	2bhex
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22hex
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23hex

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1'b0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION	OPCODE / FMT /FT / FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4) 11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr	(4) 11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6) 0/--/--/1b
FP Add Single	add.s	FR	F[fd] = F[fs] + F[ft]	11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}	11/11/--/0
FP Compare Single	c.x.s*	FR	FPcond = (F[fs] op F[ft]) ? 1 : 0	11/10/--/y
FP Compare Double	c.x.d*	FR	FPcond = ({F[fs],F[fs+1]} op {F[ft],F[ft+1]}) ? 1 : 0	11/11/--/y
* (x is eq, lt, or le) (op is <, >, <=, or >=) (y is 32, 3c, or 3e)				
FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]	11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}	11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]	11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}	11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]	11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}	11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2) 31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2) 35/--/--/--
Move From Hi	mfhi	R	R[rd] = Hi	0/--/--/10
Move From Lo	mflo	R	R[rd] = Lo	0/--/--/12
Move From Control	mfc0	R	R[rd] = C0[rs]	10 /0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]	0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6) 0/--/--/19
Shift Right Arith.	sra	R	R[rd] = R[rt] >>> shamt	0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2) 39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2) 3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS

PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = Immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	No

exam19s2_qns

Thursday, November 23, 2023 10:31 AM

exam19s2_
qns

NATIONAL UNIVERSITY OF SINGAPORE

CS2100 – COMPUTER ORGANISATION

(Semester 2: AY2019/20)

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. Please write your Student Number on the **ANSWER BOOKLET**. Do not write your name.
2. This assessment paper consists of **EIGHT (8)** questions and comprises **NINE (9)** printed pages.
3. This is an **OPEN BOOK** assessment.
4. Answer all questions and write your answers in the **ANSWER BOOKLET** provided.
5. Make sure your answers are clearly written/typed.
6. You are to submit only the **ANSWER BOOKLET** and no other document.

1. **[16 marks]**
Study the following MIPS program:

```
    addi $s1, $zero, 0
L:  andi $t0, $s0  , 0xF  # $s0 initialized as 0xBEBEC0C0
    add  $s1, $s1  , $t0
    srl  $s0, $s0  , 4
    bne  $s0, $zero, L
```

(a) What is the value of register `$s1` after we finish running the program with `$s0` initialized as `0xBEBEC0C0`? Write your answer in ***hexadecimal***. [2 marks]

(b) What is the maximum possible value of `$s1`? What is the initial value of `$s0` that can give this maximum possible value of `$s1`? Write your answers in ***hexadecimal***. [4 marks]

(c) Write the equivalent C code for the MIPS code above. You should use registers as your variable names (e.g., `$t1 = 5;`) [6 marks]

(d) Give the encoding of the instruction `srl $s0, $s0, 4` in ***hexadecimal***. [2 marks]

(e) Give the encoding of the instruction `bne $s0, $zero, L` in ***hexadecimal***. [2 marks]

2. **[4 marks]**
Convert -1.875 into IEEE-754 single-precision floating-point representation.
Write your answer in ***hexadecimal***. [4 marks]

3. **[10 marks]**
von Neumann architecture stores both data and code in the same memory. Therefore, we can have a program (such as a compiler) to process other programs. Here, we want to test your understanding of MIPS by writing MIPS instructions to reason about MIPS instructions.

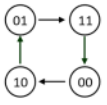
(a) Consider any MIPS instruction stored in `$s0`. Using the minimum number of MIPS instructions, set the value of register `$t0` to 1 if the instruction is an R-format instruction, otherwise set the value to 0. [3 marks]

(b) Consider an encoding of an R-format MIPS instruction stored in `$s1`. Using the minimum number of MIPS instructions, extract the field `rs` into register `$t1`. [3 marks]

(c) Consider an opcode stored in `$s2`. Write a sequence of MIPS instructions to generate the value of ALUop control signal and store it in `$t2`. The table below shows the value of ALUop for each instruction type. You may assume that the opcode will only be for `lw`, `sw`, `beq`, or any R-format instructions. [4 marks]

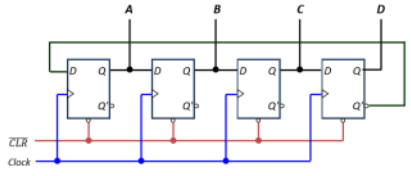
Instruction Type	ALUop
lw / sw	00
beq	01
R-type	10

4. **[12 marks]**
(a) Implement a sequential logic circuit that cycles through the following states using only *JK* flip-flops. The states are represented by 2-bit values *AB*.



Write out the **simplified SOP expressions** for all the flip-flop inputs. [4 marks]

(b) Study the sequential circuit below, which uses four *D* flip-flops. The $\overline{CLR}$ input is an asynchronous input that is used to clear the value of a flip-flop. Note that *Q'* of the right-most flip-flop is connected to the *D* input of the left-most flip-flop.



The circuit is initialised to *ABCD* = 0000 by clearing all flip-flops to zero (i.e. the *Q* output for every flip-flop is cleared to zero). This sequential circuit cycles through a number of states. The first state (call it state 0) is 0000, the second state (call it state 1) is 1000.

(i) How many states are there altogether? List out the states in sequence, i.e. 0000 → 1000 → ... [4 marks]

(ii) Suppose the function State(0) is true when the circuit is in state 0, and false otherwise. The simplest Boolean expression for State(0) is *A'·D'*.

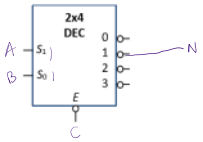
Similarly, State(1) is true when the circuit is in state 1 or false otherwise; and State(2) is true when the circuit is in state 2 or false otherwise. What are the simplest Boolean expressions for State(1) and State(2)? [4 marks]

5. **[12 marks]**
For the parts below, you are to assume that logical constants 0 and 1 are available but complemented literals are not available.

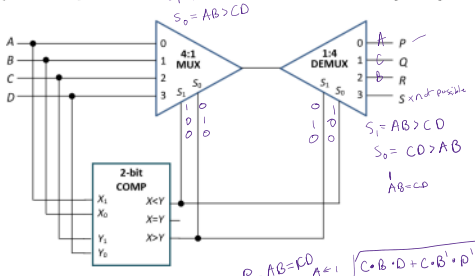
(a) A device **NonZero** takes in a 3-bit unsigned number *ABC*. Its output, *N*, is 0 if the value represented by *ABC* is 0, or 1 if the value represented by *ABC* is not zero.

(i) Write the simplified SOP expression for *N*. [2 marks]

(ii) Implement *N* using the following single 2x4 decoder with zero-enable and negated outputs, with no other logic gates and devices. [4 marks]



(b) The circuit below uses a 2-bit magnitude comparator, a 4:1 multiplexer and a 1:4 demultiplexer. Inputs are *A*, *B*, *C*, *D* and outputs are *P*, *Q*, *R*, *S*. Write the simplified SOP expressions for *P*, *Q* and *R*. [6 marks]



6. **[12 marks]**
For the parts below, you are to assume that logical constants 0 and 1 are available but complemented literals are not available. Note also that a circuit that is correct but uses more logic gates or devices than necessary will be given only partial credit.

The following block diagrams are referred to and explained in the question.

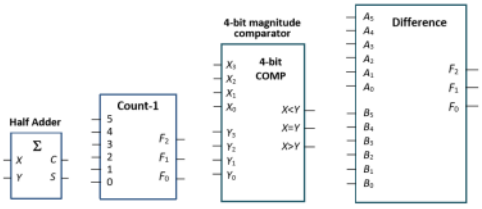


Figure 6(a) Figure 6(b) Figure 6(c) Figure 6(d)

(a) Design a circuit to take in two 6-bit codes $P=P_5P_4P_3P_2P_1P_0$ and $Q=Q_5Q_4Q_3Q_2Q_1Q_0$, and generate as output $D_2D_1D_0$ the number of pairwise bits in *P* and *Q* that are different. For example, P=001001 and Q=010100 differ by 4 bits (P_4≠Q_4, P_3≠Q_3, P_2=Q_2, P_0≠Q_0), so $D_2D_1D_0$=100; P=111110 and Q=011111 differ by 2 bits (P_5≠Q_5, P_0≠Q_0), so $D_2D_1D_0$=010; and 101010 and 010101 differ by 6 bits, so $D_2D_1D_0$=110.

Recall that we have designed a circuit in class to count the number of ones in a 6-bit number. The block diagram of the circuit, called a **Count-1 device**, is shown in Figure 6(b). If the input 001001 is fed into the Count-1 device, the output would be 010 (two).

Using only half adders (Figure 6(a)) and one **Count-1 device** with no other logic gates or devices, design a circuit to count the number of pairwise bit differences of two 6-bit codes *P* and *Q*. [3 marks]

(b) A certain 6-bit code $A_5A_4A_3A_2A_1A_0$ contains only eight valid code values shown in the table below.

$A_5A_4A_3A_2A_1A_0$
0 0 0 1 1 0
0 1 1 0 0 0
0 1 1 0 0 1
0 0 0 1 1 1
1 0 0 1 1 1
1 1 1 0 0 1
1 1 1 0 0 0
1 0 0 1 1 0

You are to design a logic circuit for Boolean function $V(A_5,A_4,A_3,A_2,A_1,A_0)$ that returns 1 if the input $A_5A_4A_3A_2A_1A_0$ is one of these eight valid values, or returns 0 otherwise.

Your circuit should contain the fewest number of half adders and at most one 4-bit magnitude comparator (Figure 6(c)), and no other logic gates or devices. [5 marks]

(c) Referring to the same code given in part (b) above.

You are to pick two code values from this table and send them to some device for processing, but before that, you want to check that these two code values indeed come from this table. The two code values you pick could be the same value.

Design a logic circuit for Boolean function $E(A_5,A_4,A_3,A_2,A_1,A_0,B_5,B_4,B_3,B_2,B_1,B_0)$ where $A=A_5A_4A_3A_2A_1A_0$ and $B=B_5B_4B_3B_2B_1B_0$ are two 6-bit values. The function E returns 1 to indicate that values A and B certainly cannot be from the table. However, if E returns 0, it is non-conclusive and further check is required.

You are to design this circuit using the device designed in part (a), which is called the **Difference** device, whose block diagram is shown in Figure 6(d), and a single 4-bit magnitude comparator, and no other logic gates or devices. [4 marks]

7. **[14 marks]**

An image consisting of 1024×1024 pixels are stored in a one-dimension integer array with 2^{20} elements in **row-major order**. Each element is a 32-bit integer, consisting of 4 parts: *r*, *g*, *b*, *t*, representing red, green, blue and transparency. The values of red, green, blue and transparency occupy 8 bits and are unsigned numbers in the range 0 through 255.

Given two images stored in integer arrays *A* and *B*, the following algorithm creates a new image in array *C* such that the RGB (red, green, blue) components of every pixel in image *C* is the exclusive-or value of the RGB components of the corresponding pixels in images *A* and *B*, and the transparency of the pixel in image *C* is the average of the transparency of the corresponding pixels in images *A* and *B*.

```
for (int r = 0; r < 1024; r++)
   for (int c = 0; c < 1024; c++)
      Cimage[r][c] = merge(Aimage[r][c], Bimage[r][c]);
```

For example, if the first pixel of image *A* is pure red with transparency value of 125 (or 0xFF00007D), and the first pixel of image *B* is pure blue with transparency value of 60 (or 0x0000FF3C), then the first pixel of image C will be purple with transparency of 92 (or 0xFF00FF5C).

The MIPS code fragment is given on the next page. Assuming a 5-stage MIPS pipeline we studied in class, and considering only the code segment from Inst1 (`add $t1,$0,$0`) to Inst23 (`addi $t2,$t2,1`) inclusive, answer the following questions. You need to count until the last stage of Inst23.

a. How many cycles does this code segment take to complete its execution in the first iteration in an ideal pipeline (that is, with no delays)? [1 mark]

For parts (b)–(d) below, given the assumption for each part, how many additional cycles does this code segment (Inst1 to Inst23 inclusive) take to complete its execution in the first iteration compared to an ideal pipeline? (For example, if part (a) takes *X* cycles and part (b) takes *Y* cycles, you are to answer part (b) with the value *Y* − *X*.)

b. Assuming without forwarding and branch decision is made at MEM stage (stage 4). No branch prediction is made and no delayed branching is used. [3 marks]

c. Assuming without forwarding and branch decision is made at ID stage (stage 2). No branch prediction is made and no delayed branching is used. [3 marks]

d. Assuming with forwarding and branch decision is made at ID stage (stage 2). Branch prediction is made where the branch is predicted not taken, and no delayed branching is used. [3 marks]

e. Assuming the condition in part (d) but now we want to implement delayed branching as well. Is it possible to fill in the branch-delay slot for each of the two **beq** instructions? If possible, indicate which instructions can be used to fill in the slots; if not possible, explain why. [2 marks]

f. For branch prediction, we may predict either the branch is taken or not taken. Which of the two choices is easier to implement, and why? [2 marks]

You may assume that $s0, $s1, $s2 and $s3 have been initialised.

```
        # $s0 = 1024
        # $s1 = base address of array A
        # $s2 = base address of array B
        # $s3 = base address of array C
        add  $t1, $0,  $0       # Inst1 : $t1 = r = 0
        add  $t2, $0,  $0       # Inst2 : $t2 = c = 0
        add  $t3, $0,  $0       # Inst3 : $t3 = element index

L1:     slt  $t0, $t1, $s0      # Inst4 : is r < 1024?
        beq  $t0, $0,  E1       # Inst5 : done with rows

L2:     slt  $t0, $t2, $s0      # Inst6 : is c < 1024?
        beq  $t0, $0,  E2       # Inst7 : done with columns

        add  $t4, $s1, $t3      # Inst8 : $t4 = addr of A[r][c]
        lw   $t5, 0($t4)        # Inst9 : $t5 = A[r][c]
        add  $t6, $s2, $t3      # Inst10: $t6 = addr of B[r][c]
        lw   $t7, 0($t6)        # Inst11: $t7 = B[r][c]

        # Inst12-19 to create image C from A and B
        xor  $t9, $t5, $t7      # Inst12
        andi $t5, $t5, 0xFF     # Inst13
        andi $t7, $t7, 0xFF     # Inst14
        add  $t0, $t5, $t7      # Inst15
        srl  $t0, $t0, 1        # Inst16
        srl  $t9, $t9, 8        # Inst17
        sll  $t9, $t9, 8        # Inst18
        or   $t9, $t9, $t0      # Inst19

        add  $t8, $s3, $t3      # Inst20: $t8 = addr of C[r][c]
        sw   $t9, 0($t8)        # Inst21: C[r][c] = $t9

        addi $t3, $t3, 4        # Inst22: next element
Inst23  addi $t2, $t2, 1        # Inst23: increment c
        j    L2                 # Inst24

E2:     add  $t2, $0,  $0       # Inst25: reset c to 0
        addi $t1, $t1, 1        # Inst26: increment r
        j    L1                 # Inst27
E1:
```


8. **[20 marks]**
Refer to the same MIPS code in the previous question.

We assume that the cache is used for **lw** instructions but not for **sw** instructions so you may ignore array *C* for this question.

Integer arrays *A* and *B* are stored starting at memory addresses 0x0040CCC0, and 0x000002C0 respectively. Each integer occupies one word which is 4 bytes long. Recall in the previous question that the arrays are stored in **row-major order** of the images they represent.

For parts (a), (b), (c): Given a **direct-mapped data cache** with 64 words in total and each block contains 4 words.

a. How many bits are there in the index field? In the byte offset field? [2 marks]

b. Which cache block is *A*[0] mapped to? Which block is *B*[60] mapped to? Write your answer in decimal. [2 marks]

c. What is the cache hit rate for array *A*? For array *B*? [2 marks]

For parts (d), (e), (f): Given a **two-way set associative data cache** with 64 words in total and each block containing 4 words. LRU (least recently used) algorithm is used for replacement.

d. Which cache set is *A*[0] mapped to? Which set is *B*[60] mapped to? Write your answer in decimal. [2 marks]

e. What is the cache hit rate for array *A*? For array *B*? [2 marks]

f. Suppose the MIPS code is unchanged but the images are stored in **column-major order** for their corresponding arrays, what would be the cache hit rates for arrays *A* and *B*? Explain. [2 marks]

For parts (g), (h), (i): Given a **direct-mapped instruction cache** with 16 words in total and each block contains 4 instructions (words). The first instruction (**add $t1, $0, $0**) is at memory address 0x04FFFFF8.

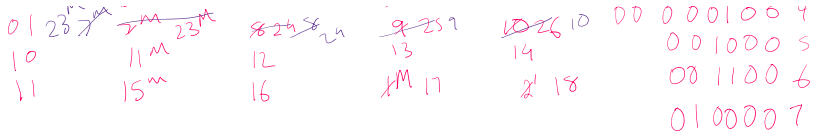
We consider only the inner loop "`for (int c=0; c<1024; c++)`" here.

g. On the instruction cache diagram on the Answer Booklet, fill in the location where the first instruction (call it Inst1) will be loaded into. [2 marks]

h. How many misses are there in the 1st iteration (Inst1 to Inst24 inclusive)? [3 marks]

i. How many misses are there in the 2nd iteration (Inst6 to Inst24 inclusive)? [3 marks]

~~~~ END OF PAPER ~~~~


0 1 23
10
11
15
23
12
13
14
10
00 0 0 0 1 0 0 4
0 0 1 0 0 0 5
0 0 1 1 0 0 6
0 1 0 0 0 0 7

exam18s2_qns (1)

Tuesday, November 28, 2023 10:10 AM

exam18s2_ qns (1)

NATIONAL UNIVERSITY OF SINGAPORE

CS2100 – COMPUTER ORGANISATION

(Semester 2: AY2018/19)

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. Please write your Student Number with a pen (to prevent accidental erasure) only on the **ANSWER BOOKLET**. Do not write your name.
2. This assessment paper consists of **SEVEN (7)** questions and comprises **EIGHTEEN (18)** printed pages.
3. This is a **CLOSED BOOK** assessment. One double-sided A4 reference sheet is allowed.
4. Calculators and computing devices such as laptops and PDAs are not allowed.
5. Answer all questions and write your answers in the **ANSWER BOOKLET** provided.
6. You may use pencil to write your answers.
7. Page 12 onwards contain the MIPS Reference Data Sheet and several blank tables for your rough works.
8. You are to submit only the **ANSWER BOOKLET** and no other document.

1. **[12 marks]**
Study the following MIPS code, which has one input **$s0** and two outputs **$t0** and **$t1**.

```
    addi $t0, $zero, 32
    addi $t1, $zero, 32
L:  beq  $s0, $zero, N
    andi $t2, $s0  , 0x0001
    beq  $t2, $zero, E
    addi $t1, $t1  , -1
E:  addi $t0, $t0  , -1
    srl  $s0, $s0  , 1
    j    L
N:
```

(a) What are the values of **$t0** and **$t1** at the end of the execution if the value of **$s0** at the start is 32? [2 marks]

(b) If the value of **$s0** is **43** at the start of execution, what is the total number of times both **beq** instructions branch? That is, when both "**beq $s0, $zero, N**" branches to **N** and "**beq $t2, $zero, E**" branches to **E**. [2 marks]

(c) Give a value of **$s0** at the start such that the values of **$t0** and **$t1** at the end of the execution are both 0. [2 marks]

(d) What is the encoding of the only **R-format** instruction above in *hexadecimal*? [2 marks]

(e) Write the relationship between **$t0** and **$s0** as well as between **$t1** and **$s0** in a *single sentence each*. [2 marks]

(f) Our current MIPS instruction set does not have **load half-word** since **lhw** is a pseudo-instruction. **lhw** loads 16 bits from memory to the *lower half* of the register and sets the *upper half* of the register to all zeros. The pseudo-instruction **lhw $t0, 80($zero)** will be translated into actual MIPS instructions before being run. Write down the equivalent actual instructions to perform **load half-word** in the fewest number of MIPS instructions possible. [2 marks]

2. **[4 marks]**
As the number -0.3_{10} cannot be represented precisely in binary, it also cannot be represented precisely in the IEEE 754 standard single precision floating point format. However, we can *approximate* the value by *truncating the bits* to the nearest representation.

Write the approximation of -0.3_{10} in IEEE 754 standard single precision floating point format. Give your answer in hexadecimal. [4 marks]

3. **[14 marks]**
You are given the implementation of MIPS processor on the next page with partially incorrect modification to include the Jump instruction (**j**). Note that for the added multiplexer (the one with control signal **IsJump?**), if **IsJump?** is 0, the top input is chosen; otherwise the bottom input is chosen.

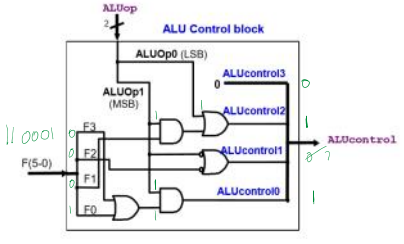
(a) Describe what is wrong with the implementation in one sentence. [2 marks]

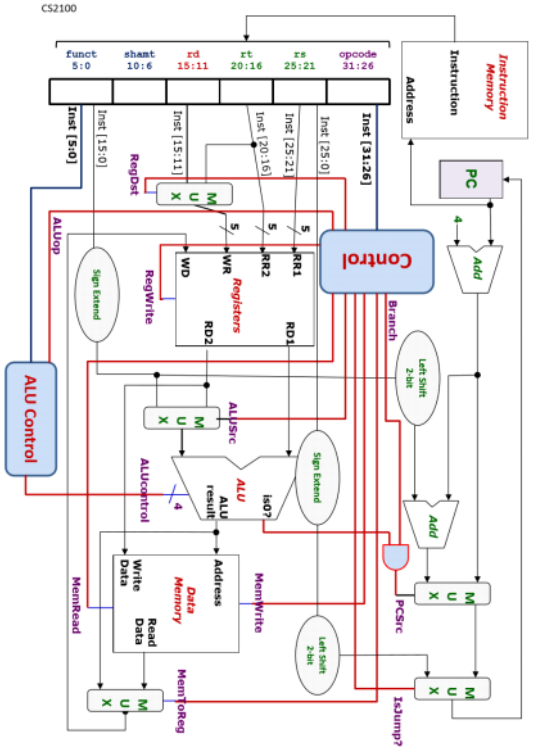
(b) Consider an instruction **0x0800C840** at address **0x2100FFFC**. What is the next value of PC when the instruction is executed using the incorrect processor above? [3 marks]

(c) Since we are using the intermediate signal **ALUop**, we specify that the **ALUop** for **j** instruction is **11**. The rest of the **ALUop** does not change. Fill in the missing values in the control signal table in the answer booklet. [3 marks]

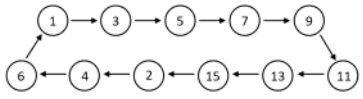
(d) Modify the combinational circuit given in the answer booklet to include **ALUop1**, **ALUop0** and **IsJump?** control signals. [4 marks]

(e) Given that there is no change to the ALU Control unit shown below for your convenience, what will be the value of **ALUcontrol** when the instruction **0x08000031** is executed? Give your answer in 4 bits binary. [2 marks]





4. **[16 marks]**
A sequential circuit goes through the following states, whose state values are shown in decimal:



The states are represented by 4-bit values *ABCD*. Implement the sequential circuit using a *D* flip-flop for *A*, *T* flip-flops for *B* and *C*, and a *JK* flip-flop for *D*.

a. Write out the **simplified SOP expressions** for all the flip-flop inputs. [10 marks]

b. Implement your circuit according to your simplified SOP expressions obtained in part (a). Complete the given state diagram on the Answer Booklet, by indicating the next state for each of the five unused states. [5 marks]

c. Is your circuit self-correcting? Why? (Answer without reason will not be given mark.) [1 mark]

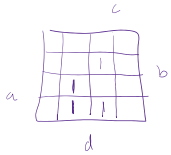
8, 10, 12, 14

yes self correction

because 8 → 10 → 12 → 14 → 3

C $A'+B'+C'$ $B\cdot C$ $A\cdot B\cdot C$

old ABCD	D_A	T_B	T_C	J_D	K_D	new ABCD
0001	0	0	1	X	0	0011
0011	0	1	1	X	0	0101
0101	0	0	1	X	0	0111
0111	1	1	1	X	0	1001
1001	1	0	1	X	0	1011
1011	1	1	1	X	0	1101
1101	1	0	1	X	0	1111
1111	0	1	0	X	1	0010
0010	0	1	1	0	X	0100
0100	0	0	1	0	X	0110
0110	0	1	1	1	X	0001
1000	1	0	1	0	0	1010
1010	1	1	1	0	0	1100
1100	1	0	1	0	0	1110
1110	0	1	0	1	1	0011

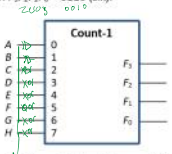


$A\cdot D\cdot C' + A\cdot B'\cdot D + A'\cdot B\cdot C\cdot D$

$A\cdot B' + A\cdot C' + A'\cdot B\cdot C\cdot D$

5. **[22 marks]**
For the parts below, you are to assume that complemented literals are not available. Note also that circuit that is correct but uses more logic gates than necessary will be given partial credit.

(a) The **8-bit count-1** device, whose block diagram is shown below, takes in an 8-bit input *ABCDEFGH* and outputs $F_3F_2F_1F_0$ which is the number of 1s in the input. For example, if *ABCDEFGH* = 11101101, then $F_3F_2F_1F_0$ = 0110 (six).



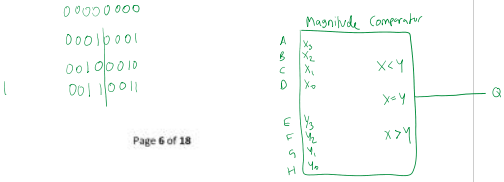
How would you implement an **8-bit count-0** device to count the number of 0s in the input using the above 8-bit count-1 device and XOR gates? No other gates or devices besides the count-1 device and XOR gates are allowed. [3 marks]

(b) Assuming that the 8-bit input *ABCDEFGH* is an unsigned binary number. Let *P*(*A*,*B*,*C*,*D*,*E*,*F*,*G*,*H*) be a Boolean function that returns 1 if *ABCDEFGH* contains an odd number of 1s and *ABCDEFGH* is an even number, or returns 0 otherwise. For example, the function *P* returns 1 for the following inputs: 01110000, 10111010, 00010000, but returns 0 for the following inputs: 00111001, 10100001, 11110000.

Implement *P* using the **Count-1 device** as shown in part (a) above, with the fewest number of additional logic gates. [3 marks]

(c) Assuming that the 8-bit input *ABCDEFGH* is an unsigned binary number. Let *Q*(*A*,*B*,*C*,*D*,*E*,*F*,*G*,*H*) be a Boolean function that returns 1 if *ABCDEFGH* is a multiple of 17 (eg: 0, 17, 34, 51, etc.), or returns 0 otherwise.

Given a **parallel adder**, a **magnitude comparator**, a **decoder**, an **encoder**, and a **demultiplexer**, implement *Q* using only ONE of these devices, without any additional logic gates. Your device should be the smallest possible (for example, if an 8-bit parallel adder is sufficient, you should not use a 16-bit parallel adder). [4 marks]



5. (continue...)

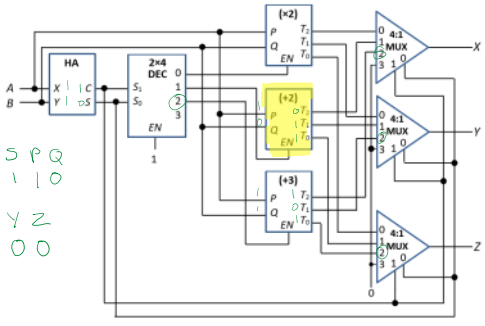
(d) Implement the following four-variable function *R*(*A*,*B*,*C*,*D*) using a single **4:1 multiplexer** without any additional logic gates. [6 marks]

$$R(A,B,C,D) = \Sigma m(0, 2, 3, 4, 6, 7, 12, 14)$$

(e) Study the following circuit which uses a **half adder (HA)**, a **2×4 decoder** with 1-enable and active high outputs, three **4:1 multiplexers** and three devices each with a 1-enable control (*EN*):

- A **(×2)-device**: it takes in two inputs *P* and *Q* and produces 3-bit output with value (*P*+*Q*)×2.
- A **(+2)-device**: it takes in two inputs *P* and *Q* and produces 3-bit output with value *P*+*Q*+2.
- A **(+3)-device**: it takes in two inputs *P* and *Q* and produces 3-bit output with value *P*+*Q*+3.

For the three devices above, if a device is not enabled, its outputs are all zeroes.



Redesign the above circuit so that it can be implemented using the fewest logic gates. Write your expressions for *X*, *Y* and *Z*. You do not need to draw your circuit. [6 marks]

6. **[18 marks]**
Given three integer arrays *A*, *B*, *C*, where arrays *B* and *C* each contains *n* elements and array *A* contains 2*n* elements, a MIPS code is written to update the elements in *A* with the elements in *B* and *C* as follows:

$A[k] = A[k] + B[k/2]$ if *k* is even
$A[k] = A[k] + C[(k-1)/2]$ if *k* is odd

For example, suppose *A* = {1, 2, 3, 4, ...}, *B* = {101, 102, ...} and *C* = {201, 202, ...}, then the final values in *A* are {102, 203, 105, 206, ...}.

The MIPS code fragment is shown below.

```
      # $s0 = base address of array A
      # $s1 = base address of array B
      # $s2 = base address of array C
      # $s3 = n, the number of elements in array B
      add  $t0, $s0, $0       # Inst1, Address: 0x00FFFF18
      add  $t1, $s1, $0       # Inst2
      add  $t2, $s2, $0       # Inst3
      add  $t3, $s3, $s3      # Inst4: $t3 = 2n
      add  $t4, $0,  $0       # Inst5: $t4 = k (loop variable)

Loop: slt  $t5, $t4, $t3      # Inst6: k < 2n?
      beq  $t5, $0,  End      # Inst7

      lw   $t6, 0($t0)        # Inst8
      lw   $t7, 0($t1)        # Inst9
      add  $t6, $t6, $t7      # Inst10
      sw   $t6, 0($t0)        # Inst11

      lw   $t8, 4($t0)        # Inst12
      lw   $t9, 0($t2)        # Inst13
      add  $t8, $t8, $t9      # Inst14
      sw   $t8, 4($t0)        # Inst15

      addi $t0, $t0, 8        # Inst16
      addi $t1, $t1, 4        # Inst17
      addi $t2, $t2, 4        # Inst18
      addi $t4, $t4, 2        # Inst19

      j    Loop               # Inst20
End:
```

A 2^{11}
B 2^{10}
C 2^{10}

For parts (a), (b), (c): Given a **two-way set associative data cache** with 64 words in total, and each block containing 4 words with each word being 4 bytes long. LRU (least recently used) algorithm is used for replacement. Each integer occupies one word.

Assuming that the integer arrays *B* and *C* each contains 2^{10} **elements**. Arrays *A*, *B* and *C* are stored starting at memory addresses 0x00000080, 0x00100000 and 0x00108040 respectively.

The data cache is involved when memory is accessed (that is, when **lw** and **sw** instructions are executed).

a. How many bits are there in the set index field? In the byte offset field? [2 marks]

b. Which set is *A*[0] mapped to? Which set is *B*[60] mapped to? Which set is *C*[1032] mapped to? You may write your answer in decimal or binary. [3 marks]

c. What is the cache hit rate for array *A*? For array *B*? For array *C*? Write your answer as a fraction. [6 marks]

For parts (e), (f), (g): Given a **direct-mapped instruction cache** with 16 words in total and each block contains 4 instructions (words). The first instruction (**add $t0, $s0, $0**) is at memory address 0x**00FFFF18**. Recall that the integer arrays *B* and *C* each contains 2^{10} **elements**.

d. How many misses are there in the 1st iteration (Inst1 to Inst20 inclusive)? [2 marks]

e. How many misses are there in the 2nd iteration (Inst6 to Inst20 inclusive)? [2 marks]

f. How many misses are there in the execution of the whole code? [3 marks]

00FFFF18 /// 1000 1000
/// 0011100
100000 3
100100 4
101000 5
101100 6
110000 7
0 000000 11

offset - within block
block = 4 word = 16 bytes
4 bits 2^4 offset

a. $\frac{64}{4} = 16$ blocks
8 blk / set = 2^3
3 bit set index

b. 1000|0000
A[0] → set 0000
B[60] 60·4 = 240 set - 7
1111|0000
C[1032] 1032·4 = 4128 → 8040×
1020×
9060
0111|0000

c. A: 7/8
B: 3/4
C: 3/4

d. 6
e. 2
f. 6+2+1
$(2^{10}-1)$
= 2053

e - tag	index	W1 0-3	W2 4-7	W3 8-11	W4 12-16
	00	11	12	13	14
	01	15	16	17	18
	10	19	20	21	22
	11	7	8	9	10

each blk = 16 bytes
offset 4
4 blks 2^2 2 bit index

7. **[14 marks]**
Refer to the same MIPS code in the previous question:

```
        # $s0 = base address of array A
        # $s1 = base address of array B
        # $s2 = base address of array C
        # $s3 = n, the number of elements in array B
        add  $t0, $s0, $0       # Inst1, Address: 0x00FFFF18
        add  $t1, $s1, $0       # Inst2
        add  $t2, $s2, $0       # Inst3
        add  $t3, $s3, $s3      # Inst4: $t3 = 2n
        add  $t4, $0,  $0       # Inst5: $t4 = k (loop variable)

Loop:   slt  $t5, $t4, $t3      # Inst6: k < 2n?
        beq  $t5, $0,  End      # Inst7

        lw   $t6, 0($t0)        # Inst8
        lw   $t7, 0($t1)        # Inst9
        add  $t6, $t6, $t7      # Inst10
        sw   $t6, 0($t0)        # Inst11

        lw   $t8, 4($t0)        # Inst12
        lw   $t9, 0($t2)        # Inst13
        add  $t8, $t8, $t9      # Inst14
        sw   $t8, 4($t0)        # Inst15

        addi $t0, $t0, 8        # Inst16
        addi $t1, $t1, 4        # Inst17
        addi $t2, $t2, 4        # Inst18
        addi $t4, $t4, 2        # Inst19

        j    Loop               # Inst20
End:
```

We assume a 5-stage MIPS pipeline system, and the first instruction (**add $t0, $s0, $0**) begins at cycle 1.

a. The jump (j) instruction causes a control hazard. What is the minimum number of stall cycles that a jump instruction would incur and how can that be achieved? [2 marks]

b. Assuming without forwarding and branch decision is made at MEM stage (stage 4). No branch prediction is made and no delayed branching is used. How many cycles does the code from instructions 1 through 19 (leaving out the jump instruction) take? You need to count until the last stage of instruction 19. [3 marks]

c. Assuming with forwarding and early branching, that is, the branch decision is made at ID stage (stage 2). No branch prediction is made and no delayed branching is used. How many cycles does the code from instructions 1 through 19 (leaving out the jump instruction) take? You need to count until the last stage of instruction 19. [3 marks]

d. Assuming with forwarding and early branching, that is, the branch decision is made at ID stage (stage 2). Branch prediction is used, where the branch is predicted not taken. How many cycles does the code from instructions 1 through 19 (leaving out the jump instruction) take? You need to count until the last stage of instruction 19. [3 marks]

e. Assuming with forwarding, how would you rearrange the instructions to reduce the number of stall cycles, and how many stall cycles is reduced as a result of this? You do not need to rewrite the full code. Just describe the changes or show the portion that is changed. Your changes should be as minimal as possible. [3 marks]

""" END OF PAPER """

(The next few pages contain the MIPS Reference Data sheet, blank truth tables, K-maps and pipeline charts.)

MIPS Reference Data

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	$0/20_{hex}$
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		$0/21_{hex}$
And	and	R	R[rd] = R[rs] & R[rt]		$0/24_{hex}$
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		$0/08_{hex}$
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		$0/27_{hex}$
Or	or	R	R[rd] = R[rs] | R[rt]		$0/25_{hex}$
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		$0/2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	$0/2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		$0/00_{hex}$
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		$0/03_{hex}$
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (atomic) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	$0/22_{hex}$
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		$0/23_{hex}$

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1'b0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct
	31 26 25 21 20 16 15 11 10 6 5 0					
I	opcode	rs	rt	immediate		
	31 26 25 21 20 16 15 0					
J	opcode	address				
	31 26 25 0					

ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION	OPCODE / FMT /FT / FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr (4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr(4)	11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt] (6)	0/--/--/1b
FP Add Single	add.s	FR	F[fd] = F[fs] + F[ft]	11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}	11/11/--/0
FP Compare Single	c.x.s*	FR	FPcond = (F[fs] op F[ft]) ? 1 : 0	11/10/--/y
FP Compare Double	c.x.d*	FR	FPcond = ({F[fs],F[fs+1]} op {F[ft],F[ft+1]}) ? 1 : 0	11/11/--/y
* (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e)				
FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]	11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}	11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]	11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}	11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]	11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}	11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm] (2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4] (2)	35/--/--/--
Move From Hi	mfhi	R	R[rd] = Hi	0/--/--/10
Move From Lo	mflo	R	R[rd] = Lo	0/--/--/12
Move From Control	mfc0	R	R[rd] = CR[rs]	10/0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]	0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt] (6)	0/--/--/19
Shift Right Arith.	sra	R	R[rd] = R[rt] >>> shamt	0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]; (2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1] (2)	3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS

FR	opcode	fmt	ft	fs	fd	funct
	31 26 25 21 20 16 15 11 10 6 5 0					
FI	opcode	fmt	ft	immediate		
	31 26 25 21 20 16 15 0					

PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	No

(This page is for your rough work.)

A	B	C	D	A*	B*	C*	D*					

DA

TB

TC

JD

KD

(This page is for your rough work.)

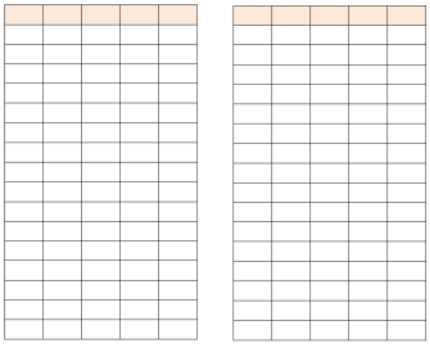
Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
I1 add																														
I2 add																														
I3 add																														
I4 add																														
I5 add																														
I6 slt																														
I7 beq																														
I8 lw																														
I9 lw																														
I10 add																														
I11 sw																														

Cycle																														
I12 lw																														
I13 lw																														
I14 add																														
I15 sw																														
I16 addi																														
I17 addi																														
I18 addi																														
I19 addi																														

(This page is for your rough work.)

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
I1 add																														
I2 add																														
I3 add																														
I4 add																														
I5 add																														
I6 slt																														
I7 beq																														
I8 lw																														
I9 lw																														
I10 add																														
I11 sw																														
I12 lw																														
I13 lw																														
I14 add																														
I15 sw																														
I16 addi																														
I17 addi																														
I18 addi																														
I19 addi																														

(This page is for your rough work.)



(This page is intentionally left blank.)

(This page is intentionally left blank.)

exam20s2_qns

Tuesday, November 28, 2023 4:06 PM

exam20s2_qns

NATIONAL UNIVERSITY OF SINGAPORE

CS2100 – COMPUTER ORGANISATION

(Semester 2: AY2020/21)

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This assessment paper consists of **SIXTEEN (16)** questions (excluding question 0) in **THREE (3)** parts and comprises **ELEVEN (11)** printed pages.
2. Answer ALL questions.
3. This is an **OPEN BOOK** assessment.
4. The maximum mark of this assessment is 100.

Question	Max. mark
Q0	3
Part A: Q1 – 6	12
Part B: Q7 – 11	15
Part C: Q12	12
Part C: Q13	13
Part C: Q14	13
Part C: Q15	18
Part C: Q16	14
Total	**100**

5. You are to submit a single pdf file (size $\leq$ 20MB) to your submission folder on LumiNUS.
6. Your submitted file should be named after your Student Number (eg: A1234567X.pdf) and your Student Number should be written on the first page of your file.
7. Do NOT write your name anywhere in your submitted file.

——— END OF INSTRUCTIONS ———

You do not need to write any answer for question 0.

0. (a) Submit a **single pdf file** into the submission folder. [1 mark]

   (b) Name your pdf file with your Student Number (eg: A1234567X.pdf). [1 mark]

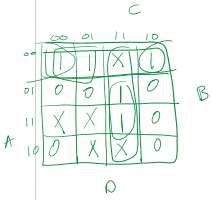
   (c) Write your Student Number (not your name!) on the first page of your file. [1 mark]

Part A: Multiple-Choice Questions [Total: 6×2=12 marks]

Each multiple-choice question (MCQ) is worth **TWO marks** and has exactly **one** correct answer. You may write your answers into the boxes in the Answer Sheets provided or if you are using your own paper, write your answers on a **single line** to conserve space. For example:

1. A 2. B 3. C 4. D ...

Please write in **CAPITAL LETTERS**.



1. Given the Boolean function $F(A,B,C,D) = \Sigma m(0,1,2,7,15) + \Sigma x(3,9,11,12,13)$ where x denotes don't-care, how many prime implicants (PIs) are there on the K-map of F?

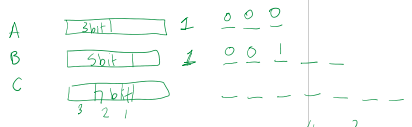
   A. 3
   B. 4
   C. 5
   D. 6
   E. None of the above.

2. Given the same Boolean function in question 1 above, how many essential prime implicants (EPIs) are there on the K-map of F?

   A. 0
   B. 1
   C. 2
   D. 3
   E. None of the above.

3. A certain machine has 3 types of instructions: *A*, *B* and *C*, which have opcode of 3 bits, 5 bits, and 7 bits respectively. Assuming that each type must have at least one instruction, and the encoding space for opcode is completely utilized, what is the maximum total number of instructions using an expanding opcode scheme?

   A. 98
   B. 108
   C. 110
   D. 120
   E. 128



$1 + 1 + 2^7 - 2^4 - 2^2$

4. Study the MIPS code below. $v0 contains a 7-bit value in its least significant bits; the rest of its bits are zeroes.

```
        addi $s0, $zero, 0
        addi $t0, $v0, 0
        addi $t1, $zero, 0
Loop:   andi $t2, $t0, 1
        xor  $t1, $t1, $t2
        addi $s0, $s0, 1
        slti $t3, $s0, 7
        beq  $t3, $zero, Out
        srl  $t0, $t0, 1
        j    Loop
Out:    sll  $t1, $t1, 7
        or   $v0, $v0, $t1
```

What does the code do?

A. Writes the number of 1's in the value in $v0 to bit 7 of $v0.

B. Writes 0 to bit 0 of $v0 if there are odd numbers of 1's in the value in $v0, otherwise writes 1 to bit 0 of $v0.

C. Writes 0 to bit 0 of $v0 if there are even numbers of 1's in the value in $v0, otherwise writes 1 to bit 0 of $v0.

D. Writes 0 to bit 7 of $v0 if there are odd numbers of 1's in the value in $v0, otherwise writes 1 to bit 7 of $v0.

E. Writes 0 to bit 7 of $v0 if there are even numbers of 1's in the value in $v0, otherwise writes 1 to bit 7 of $v0.

5. How many instructions are executed by the code in question 4 when it completes?

A. 12

B. 47

C. 52

D. 54

E. 59

6. Given the Boolean expression below:

$$A \cdot B' \cdot (C + D' + E)' \cdot F + (C' + F')' + D \cdot F$$

Which of the following is equivalent to the above expression?

A. F
B. $D \cdot F$
C. $(C' + D') \cdot F$
D. $(C + D) \cdot F$
E. None of the above.

Part B: Multiple-Response Questions [Total: 5×3=15 marks]

Each multiple-response question (MRQ) is worth **THREE marks** and may have one answer or multiple answers. Write out **all** correct answers. For example, if you think that A, B, C are the correct answers, write A, B, C.

Only if you get all the answers correct will you be awarded three marks. **No partial credit will be given for partially correct answers.**

You may write your answers into the boxes in the Answer Sheets provided or if you are using your own paper, write your answers on a **single line** to conserve space. For example:

7. A,B 8. B,D 9. C 10. A,B,C,D 11. B,D,C

Please write in **CAPITAL LETTERS**.

7. Yucan wrote the value **66512** in his CS2100 exam, but did not specify the base. Choose from the list below all possible interpretations of this value.

A. The 5-digit 7's complement of 155_7.
B. The 5-digit 6's complement of 33156_6.
C. 66512_{10}.
D. -33488_{10} in 5-digit 10's complement of base 10.
E. -22377_{10} in 5-digit 9's complement of base 9.

8. Which of the following statements are true about the MIPS datapath?

A. The register file can only read one register at a time.
B. The register file can only write to one register at a time.
C. A jump instruction (j) may not always be possible to jump to another instruction that is 16 instructions away from it.
D. The target address of a branch instruction is given by PC + Immed x 4.
E. In the pipelined system, the PC contains the address of the instruction that is currently in the WB stage.

9. Which of the following MIPS instructions may be used to set the value of register **$t2** to zero?
   A. `and  $t2, $zero, $zero`
   B. `andi $t2, $t2, 0`
   C. `xor  $t2, $t2, $t2`
   D. `xori $t2, $t2, 0`
   E. `nor  $t2, $zero, $zero`

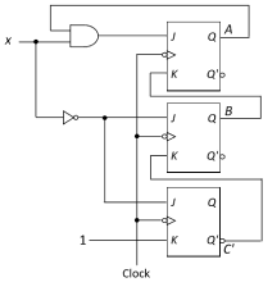
10. If the datapath control generates the wrong signal for **MemToReg**, that is, it generates 0 when it should be 1, and 1 when it should be 0, which of the following instructions will be wrongly executed?
   A. bne
   B. sw
   C. lw
   D. add
   E. ori

11. Assuming that logical constants 0 and 1 are available but complemented literals are not, which of the following functions can be implemented using a single **4-bit magnitude comparator** with no additional logic gates?
   A. $F1(A,B,C,D) = A \cdot B \cdot C$
   B. $F2(A,B,C,D) = \Sigma m(8,10,12,14)$
   C. $F3(A,B,C,D) = \Sigma m(0,15)$
   D. $F4(A,B,C,D) = \Sigma m(1,2,3,4)$

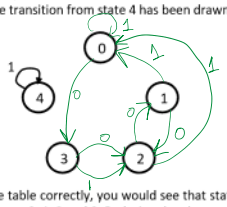
Part C: There are 5 questions in this part [Total: 70 marks]

Q12. Sequential circuits [12 marks]

Study the following sequential circuit with three JK flip-flops and an external input *x*. The states are represented by *ABC*. This circuit implements a machine with 5 valid states *ABC* = 000 to 100 (or 0 to 4 in decimal). The other 3 states (*ABC* = 101 to 111, or 5 to 7 in decimal) are unused.



(a) Complete the state diagram below by drawing the arrows and labelling each arrow with the value of *x*. States are shown in decimal values. You need only fill in the transitions to the used states 0, 1, 2, 3 and 4. You do not need to include the unused states 5, 6 and 7 in your diagram. The transition from state 4 has been drawn for you. [8]



(b) If you did your state table correctly, you would see that state 4 is unreachable from any of the other used states 0, 1, 2 and 3. Redesign the above sequential circuit by removing state 4, using two D flip-flops to replace the 3 JK flip-flops. Call you flip flops *F* and *G*. Your circuit now has 4 used states *FG* = 00 to 11 (or 0 to 3 in decimal). What are your flip-flop inputs ***DF*** and ***DG***? [4]

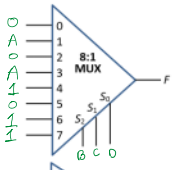
Q13. Combinational circuits [13 marks]

Note that logical constants 0 and 1 are available, but not complemented literals.

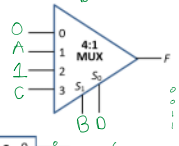
(a) Given the following Boolean function

$$F(A,B,C,D) = \Sigma m(4, 6, 7, 9, 11, 12, 14, 15),$$

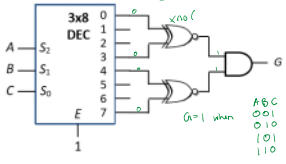
implement the function using a single 8:1 multiplexer as shown on the right without any additional logic gates. Once you have used a variable on a selector line, you should not use the same variable on the multiplexer inputs. [2]



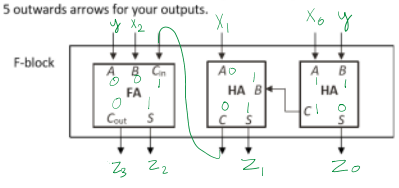
(b) Given the same Boolean function in part (a) above, implement the function using a single 4:1 multiplexer as shown on the right without any additional logic gates. Once you have used a variable on a selector line, you should not use the same variable on the multiplexer inputs. [3]



(c) A Boolean function *G*(*A*,*B*,*C*) is implemented using a 3×8 decoder with 1-enable and active high outputs, 2 XNOR gates and an AND gate as shown on the right. Replace this circuit with a single logic gate. [4]



(d) The F-block contains two half adders and a full adder as shown below. Using a single F-block, without any additional logic gates, design a circuit that takes in a 3-bit unsigned binary number $X=X_2X_1X_0$ and a one-bit value *y* to generate the output $Z_3Z_2Z_1Z_0$ which is a binary number representing the value $X+5y$. You may only connect inputs to the 6 inwards arrows and use the 5 outwards arrows for your outputs. [4 marks]



Q14. MIPS [13 marks]

Study the following MIPS code on integer arrays *A* and *B* which contain the same number of elements. The following are the variable mappings:

- $s0 = base address of array *A*
- $s1 = base address of array *B*
- $s2 = *size* (number of elements in array *A*)
- $s5 = *count*

```
.data
A: .word 10,21,12,17,9,1,20,33
B: .word 100,3,20,15,2,2,65,11
size: .word 8

.text
main: la   $s0, A          # $s0 is the base address of array A
      la   $s1, B          # $s1 is the base address of array B
      [ la $s2, size ]     # Box 1

# -----------------------------------------------------------------
      add  $s5, $0, $0     # I1
      add  $t0, $0, $0     # I2
loop: slt  $t8, $t0, $s2   # I3
      beq  $t8, $0, end    # I4
      sll  $t1, $t0, 2     # I5
      add  $t3, $t1, $s0   # I6
      lw   $s3, 0($t3)     # I7
      andi $t9, $s3, 1     # I8
      beq  $t9, $0, skip   # I9
      add  $t4, $t1, $s1   # I10
      lw   $s4, 0($t4)     # I11
      sub  $s3, $s3, $s4   # I12
      sw   $s3, 0($t3)     # I13
      addi $s5, $s5, 1     # I14
skip: addi $t0, $t0, 1     # I15
      j    loop            # I16
# -----------------------------------------------------------------
end:  [ lw $s2, 0($t0) ]   # Box 2
      syscall              # system call to print
      li   $v0, 10
      syscall              # system call to exit
```

Q14. (continue...)

Note that pseudo-instructions **la** and **li** are allowed, but not other pseudo-instructions.

(a) Fill in Box 1 with MIPS instruction(s) to load the value of *size* into **$s2**. [2]

(b) Fill in Box 2 with MIPS instruction(s) to prepare for the printing of the value of *count* (**$s5**). [2]

(c) Using the variable names (*A*, *B*, *size*, *count*) shown in the variable mappings above, write an equivalent C code that corresponds to instructions I1 to I16 in the above MIPS code. You may use additional variable(s) if needed. You do not need to declare the variables in your C code. [3]

(d) Write the instruction encoding of instruction I3 (`slt $t8, $t0, $s2`). Write your answer in hexadecimal. [2]

(e) Write the instruction encoding of instruction I9 (`beq $t9, $0, skip`). Write your answer in hexadecimal. [2]

(f) Write the instruction encoding of instruction I16 (`j loop`), assuming that instruction I1 (`add $s5, $0, $0`) is stored at address 0x10000C50. Write your answer in hexadecimal. [2]

Q15. Cache [18 marks]

Refer to the following MIPS code which is the same as the one in question 14. Here, we only look at instructions I1 to I16.

```
      add  $s5, $0, $0    # I1
      add  $t0, $0, $0    # I2
loop: slt  $t8, $t0, $s2  # I3
      beq  $t8, $0, end   # I4
      sll  $t1, $t0, 2    # I5
      add  $t3, $t1, $s0  # I6
      lw   $s3, 0($t3)    # I7
      andi $t9, $s3, 1    # I8
      beq  $t9, $0, skip  # I9
      add  $t4, $t1, $s1  # I10
      lw   $s4, 0($t4)    # I11
      sub  $s3, $s3, $s4  # I12
      sw   $s3, 0($t3)    # I13
      addi $s5, $s5, 1    # I14
skip: addi $t0, $t0, 1    # I15
      j    loop           # I16
end:
```

Q15. (continue...)

For parts (a) to (d): You are given a **direct-mapped data cache** with 128 words in total and each block contains 4 words. You may assume the following:

- The data cache is used for **lw** instructions but not for **sw** instructions.
- Array *A* starts at address **0xFF000C00** and all elements in array *A* are positive odd integers.
- Array *B* follows immediately after array *A* in the memory. That is, if the last element of array *A* is at address *x*, then the first element of array *B* is at address *x* + 4.

(a) How many bits are there in the index field? In the byte offset field? [2]

(b) Given that array *A* contains **3200** elements and array *B* contains the same number of elements, what is the hit rate of the data cache (i) for array *A* and (ii) for array *B*? Note that you need to consider only the **lw** instructions but not the **sw** instructions. [2]

(c) Given that array *A* contains **3216** elements and array *B* contains the same number of elements, what is the hit rate of the data cache (i) for array *A* and (ii) for array *B*? Note that you need to consider only the **lw** instructions but not the **sw** instructions. [4]

(d) Given that array *A* contains **3210** elements and array *B* contains the same number of elements, what is the hit rate of the data cache (i) for array *A* and (ii) for array *B*? Note that you need to consider only the **lw** instructions but not the **sw** instructions. [4]

For parts (e) to (g): You are given a **2-way set-associative instruction cache** with 16 words in total and each block contains 2 words, with LRU replacement policy. You may assume the following:

- There are 100 elements in array *A* and the same number of elements in array *B*.
- All elements in array *A* are positive odd integers.

(e) How many bits are there in the set index field? In the byte offset field? [2]

(f) Assuming that instruction I1 (`add $s5, $0, $0`) is stored at address **0x10000C50,** how many hits in total are there in the instruction cache in the execution of the code? Consider only instructions I1 to I16. [2]

(g) Assuming that instruction I1 (`add $s5, $0, $0`) is stored at address **0x10000C54,** how many hits in total are there in the instruction cache in the execution of the code? Consider only instructions I1 to I16. [2]

tag index offset

a. # blks = $\frac{128}{4}$ = 32 2^5 5 bits index

each blk = 4 words x4 = 16 byte 2^4 4 bits offset

b. i 00000000

c.

Q16. Pipelining [14 marks]

Refer to the following MIPS code which is the same as the one in question 14. Here, we only look at instructions I1 to I16.

```
       add  $s5, $0, $0    # I1
       add  $t0, $0, $0    # I2
loop:  slt  $t8, $t0, $s2  # I3
       beq  $t8, $0, end   # I4
       sll  $t1, $t0, 2    # I5
       add  $t3, $t1, $s0  # I6
       lw   $s3, 0($t3)    # I7
       andi $t9, $s3, 1    # I8
       beq  $t9, $0, skip  # I9
       add  $t4, $t1, $s1  # I10
       lw   $s4, 0($t4)    # I11
       sub  $s3, $s3, $s4  # I12
       sw   $s3, 0($t3)    # I13
       addi $s5, $s5, 1    # I14
skip:  addi $t0, $t0, 1    # I15
       j    loop           # I16
end:
```

Assuming a 5-stage MIPS pipeline and all elements in array *A* are positive odd integers, answer the following questions. You need to count until the last stage of instruction I16.

(a) How many cycles does this code segment take to complete its execution in the first iteration (I1 to I16) in an ideal pipeline, that is, one with no delays? [2]

For parts (b) to (d) below, given the assumption for each part, how many additional cycles does this code segment (I1 to I16) take to complete its execution in the first iteration as compared to an ideal pipeline? (For example, if part (a) takes 12 cycles and part (b) takes 20 cycles, you are to answer part (b) with the value 8 and not 20.)

(b) Assuming without forwarding and branch decision is made at MEM stage (stage 4). No branch prediction is made and no delayed branching is used. [3]

(c) Assuming without forwarding and branch decision is made at ID stage (stage 2). No branch prediction is made and no delayed branching is used. [3]

(d) Assuming with forwarding and branch decision is made at ID stage (stage 2). Branch prediction is made where the branch is predicted not taken, and no delayed branching is used. [3]

(e) Assuming the setting in part (d) above and you are not allowed to modify any of the instructions, is it possible to reduce the additional delay cycles in part (d) by rearranging some instructions, and if possible, by how many cycles? Explain your answer. (Answer with no explanation will not be awarded any mark.) [3]

=== END OF PAPER ===

exam21s1_qns

Wednesday, November 29, 2023 8:34 PM

exam21s1_qns

NATIONAL UNIVERSITY OF SINGAPORE

CS2100 – COMPUTER ORGANISATION

(Semester 1: AY2021/22)

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This assessment paper consists of **SIXTEEN (16)** questions (excluding question 0) in **THREE (3)** parts and comprises **FOURTEEN (14)** printed pages.
2. Answer ALL questions.
3. This is an **OPEN BOOK** assessment.
4. The maximum mark of this assessment is 100.

Question	Max. mark
Q0	3
Part A: Q1 – 6	12
Part B: Q7 – 11	15
Part C: Q12	12
Part C: Q13	13
Part C: Q14	13
Part C: Q15	14
Part C: Q16	18
Total	**100**

5. You are to submit a single pdf file (size $\leq$ 20MB) to your submission folder on LumiNUS.
6. Your submitted file should be named after your Student Number (eg: A1234567X.pdf) and your Student Number should be written on the first page of your file.
7. Do NOT write your name anywhere in your submitted file.

——— END OF INSTRUCTIONS ———

You do not need to write any answer for question 0.

0. (a) Submit a **single pdf file** into the submission folder. [1 mark]

   (b) Name your pdf file with your Student Number (eg: A1234567X.pdf). [1 mark]

   (c) Write your Student Number (not your name!) on the first page of your file. [1 mark]

Part A: Multiple-Choice Questions [Total: 6×2=12 marks]

Each multiple-choice question (MCQ) is worth **TWO marks** and has exactly **one** correct answer. You may write your answers into the boxes in the Answer Sheets provided or if you are using your own paper, write your answers on a **single line** to conserve space. For example:

1. A 2. B 3. C 4. D ...

Please write in **CAPITAL LETTERS**.

1. If $(1022)_3 = (50)_x$, what is x?

   A. 4 20 2
   B. 5 25
   C. 6 30
   D. 7 35
   E. None of the above.

2. What is the output of the following C program?

```
#include <stdio.h>
void foo(int *i){
    (*i)++;
}

int main(){
    int i = 20;
    int *ptr = &i;
    foo(ptr);
    printf("i = %d\n", i);
    return 0;
}
```

   A. `i = 20`
   B. `i = 21`
   C. `i = 22`
   D. No output; runtime error.
   E. No output; compilation error.

3. What is the output of the following C program?

```
#include <stdio.h>
int main(){
   int i = 100;
   int *ptr = &i;
   *ptr += 1;
   printf("%d %d\n", *ptr, i);
   return 0;
}
```

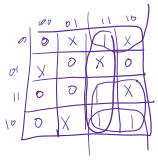
A. 101 101
B. 100 101
C. 101 100
D. 100 100
E. None of the above.

4. For the **lw $20, 100($10)** instruction, which one of the following is true?

A. Data stored in the RR1 from register file and Write Data from Data Memory are not used.
B. Data stored in the RR2 and WD from register files are not used.
C. Data stored in the RR2 from register file and Write Data from Data Memory are not used.
D. Data stored in the RR2 and WR from register file are not used.
E. None of the above.

5. Given the Boolean function $F(A,B,C,D) = \Sigma m(3,10,11,15) + \Sigma x(1,2,4,7,9,12,14)$ where *x* denotes don't-care, how many EPIs (essential prime implicants) are there on the K-map of *F*?

A. 0
B. 1
C. 2
D. 3
E. 4.



6. For the following MIPS code, what is the equivalent C code? Use the following mapping of variables and registers: f, g, h, i, and j are mapped to registers $s0, $s1, $s2, $s3, and $s4 respectively.

f g h i j

```
Loop1:  bne $s3, $s4, Loop2
        add $s0, $s1, $s2
        j   Loop3
Loop2:  sub $s0, $s1, $s2
Loop3:
```

if s3 ≠ s4 go loop2
s0 = s1 + s2
j loop3
s0 = s1 − s2

What does the code do?

A.
```
if (i==j) f = g+h;
f = g-h;
```

if i == j
f = g − h
f = g + h

B.
```
if (i==j) f = g-h;
f = g+h;
```

C.
```
if (i!=j) f = g+h;
else      f = g-h;
```

D.
```
if (i!=j) f = g-h;
else      f = g+h;
```

E. Both (A) and (D).

F. Both (B) and (C).

Part B: Multiple-Response Questions [Total: 5×3=15 marks]

Each multiple-response question (MRQ) is worth **THREE marks** and may have one answer or multiple answers. Write out **all** correct answers. For example, if you think that A, B, C are the correct answers, write A, B, C.

Only if you get all the answers correct will you be awarded three marks. **No partial credit will be given for partially correct answers.**

You may write your answers into the boxes in the Answer Sheets provided or if you are using your own paper, write your answers on a **single line** to conserve space. For example:

7. A,B 8. B,D 9. C 10. A,B,C,D 11. B,D,C

Please write in **CAPITAL LETTERS**.

7. On a particular 32-bit machine with byte-addressable memory, we have the following four bytes in memory (the starting address 0x3E8 is divisible by 4):

Address	Value
0x3E8	0x43
0x3E9	0x61
0x3EA	0x74
0x3EB	0x73

67 1000011
97 1100001
116
115

Which of the following statements are true?

A. If the system is big endian, the four bytes form the word "Cats".
B. If the system is little endian, the four bytes form the word "Cats".
C. If the system is big endian, the four bytes form the integer value 1130460273 in decimal.
D. If the system is little endian, the four bytes form the integer value 1937006915 in decimal.

8. Dueet has written the following piece of MIPS assembly code. The first instruction (addi $3, $zero, 0) is at memory location 0x0.

```
      addi $3, $zero, 0     # instruction 1
      addi $5, $zero, 0     #
loop: addi $3, $3, 1        # instruction 3
      andi $4, $3, 1        # instruction 4
      bne  $4, $zero, loop  # instruction 5
      addi $5, $5, 1        #
      j    loop             # instruction 7
```

Since Dueet does not have an assembler, he is assembling this code by hand. Unfortunately, he has made some mistakes. Select the instructions that were **wrongly** assembled.

A. 0x20030000 for instruction 1
B. 0x20630001 for instruction 3
C. 0x30830001 for instruction 4
D. 0x1480FFFE for instruction 5
E. 0x08000002 for instruction 7

9. Which of the following statements are **true** about number systems?

A. In an *n*-bit excess *E* signed number system where $0 < E < 2^n$, the integers that can be represented range from $-E$ to $(2^n - E - 1)$.
B. In an *n*-digit diminished radix complement number system in base R, the integers that can be represented range from $-(R^n - 1)$ to $(R^n - 1)$.
C. Sign-extension works with all signed number systems.
D. Ignoring any special bit patterns, the smallest positive number that can be represented in a 32-bit IEEE-754 format is 1×2^{-127}.
E. The decimal number 23 is represented by 10111 in BCD (Binary Coded Decimal) code.

10. Choose the statements below that are **true** about the MIPS processor.

A. The processor sign-extends constants for bitwise operations.
B. Using two j statements, it is possible to jump forward to more than 2^{26} instructions away.
C. When branching forward, the displacement is always calculated relative to the instruction after the branch, but when branching backwards, the displacement is always calculated relative to the branch itself.
D. We can use a 6-to-64 decoder to decode the opcode.
E. The processor instruction format allows for up to 127 different instructions.

11. The code below adds up a series of integers in an array whose base address is in $12 and length in $13. There are three instructions missing, marked by <instr1>, <instr2> and <instr3>. The array is assumed to have at least one element.

```
       addi $2, $zero, 0
       addi $3, $zero, 0
loop:  sll  $4, $3, 2
       add  $5, $4, $12
       lw   $5, 0($5)
       add  $2, $2, $5
       addi $3, $3, 1
       <instr1>
       <instr2>
       <instr3>
exit:
```

Choose all instruction groups that fit into <instr1>, <instr2> and <instr3>. Some groups may not have an <instr3> and this is indicated in the comment for that group.

A.
```
slt $5, $13, $3      # <instr1>
beq $5, $zero, loop  # <instr2>
                     # <instr3> – Nothing
```

B.
```
slt $5, $3, $13      # <instr1>
bne $5, $zero, loop  # <instr2>
                     # <instr3> – Nothing
```

C.
```
slt $5, $3, $13      # <instr1>
bne $5, $zero, exit  # <instr2>
j       loop         # <instr3>
```

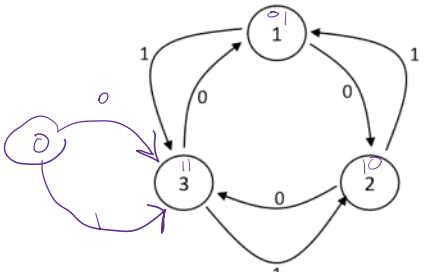
D.
```
slt $5, $13, $3      # <instr1>
bne $5, $zero, exit  # <instr2>
j       loop         # <instr3>
```

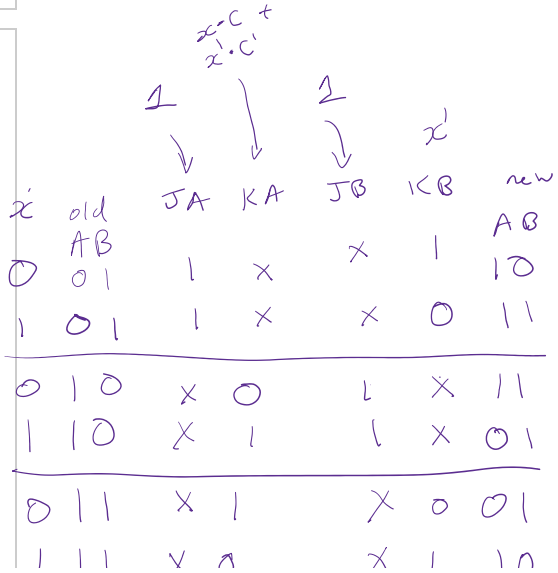
E.
```
slt $5, $3, $13      # <instr1>
beq $5, $zero, exit  # <instr2>
j       loop         # <instr3>
```

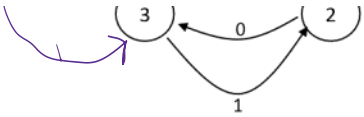

Part C: There are 5 questions in this part [Total: 70 marks]

Q12. Sequential circuits [12 marks]

(a) Given the following state diagram on 3 states with external input *x*, where the states are shown in decimal, design the sequential circuit using two *JK* flip-flops called *A* and *B*. You do not need to produce the logic diagram of your circuit.



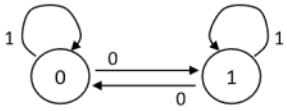




(i) Write out the flip-flop input functions for flip-flops *A* and *B* in SOP expressions. [4 marks]

(ii) Complete the state diagram on the Answer Sheets by drawing the transitions from unused state 0. [2 marks]

(b) Polfpilf wants to design his own *F* flip-flop with an external input *F*. The state diagram of his flip-flop is shown below.



(i) Design this *F* flip-flop using a *T* flip-flop and one logic gate. [3 marks]

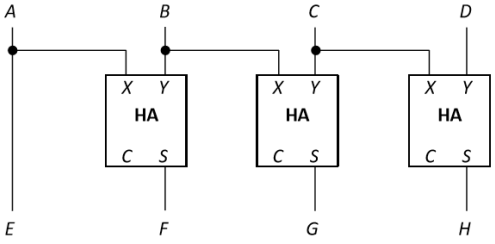
(ii) Design this *F* flip-flop using a *D* flip-flop and one logic gate. [3 marks]

Q13. Combinational circuits [13 marks]

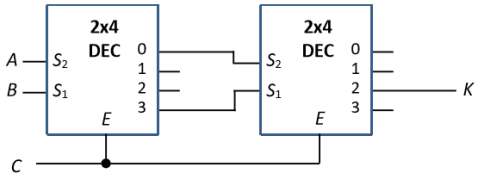
Note that logical constants 0 and 1 are available, but not complemented literals.

(a) The circuit below shows 3 half-adders. Each half adder takes inputs *X* and *Y* and produces outputs *C* (carry) and *S* (sum). The circuit takes in a 4-bit input *ABCD* and generates a 4-bit output *EFGH*.

Write out the Σm notation for the functions *E*(*A*,*B*,*C*,*D*), *F*(*A*,*B*,*C*,*D*), *G*(*A*,*B*,*C*,*D*), and *H*(*A*,*B*,*C*,*D*). [4 marks]



(b) The circuit below shows two 2×4 decoders with 1-enable and active high outputs. What is the simplified SOP expression for *K*? [4 marks]

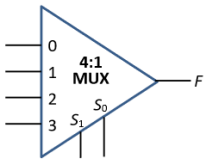


(c) Given the following Boolean function

(c) Given the following Boolean function

$F(A,B,C,D) = \Sigma m(7, 10, 14) + \Sigma d(2,6)$,

implement *F* using a single 4:1 multiplexer as shown on the right with at most one logic gate. Once you have used a variable on a selector line, you should not use the same variable on the multiplexer inputs. [5]



Q14. MIPS [13 marks]

Study the following MIPS code on integer arrays *A* and *B* which contain the same number of elements. The following are the variable mappings:

- $a0 = *size* (number of elements in array *A*)
- $a1 = base address of array *A*
- $a2 = base address of array *B*
- $t9 = *answer*

```
.data
A: .word 10, 21, 12, 17, 9,  1, 20, 33
B: .word  7,  3, 20, 15, 2, 18, 35, 11
size: .word 8

.text
main: la   $t0, size      # $t0 is the address of size
      lw   $a0, 0($t0)    # $a0 is the content of size
      la   $a1, A         # $a1 is the base address of array A
      la   $a2, B         # $a2 is the base address of array B
# ------------------------------------------------------------------
      add  $t0, $0, $0    # I1; i = 0
      add  $t9, $0, $0    # I2; ans = 0
      addi $t1, $a1, 0    # I3; $t1 = &A[0]
      addi $t2, $a2, 0    # I4; $t2 = &B[0]
      sll  $t8, $a0, 2    # I5; $t8 = size * 4
loop: slt  $t3, $t0, $t8  # I6; i < size * 4?
      beq  $t3, $0, end   # I7
      lw   $s1, 0($t1)    # I8; $s1 = A[i/4]
      lw   $s2, 0($t2)    # I9; $s2 = B[i/4]
      slt  $t3, $s1, $s2  # I10
      bne  $t3, $0, else  # I11
      add  $t9, $t9, $s1  # I12; ans = ans + A[i/4]
      j    cont           # I13
else: sub  $t9, $t9, $s2  # I14; ans = ans - B[i/4]
cont: addi $t0, $t0, 4    # I15
      addi $t1, $t1, 4    # I16
      addi $t2, $t2, 4    # I17
      j    loop           # I18
# ------------------------------------------------------------------
end:  li   $v0, 1         # system call code for print_int
      add  $a0, $t9, $0   # transfer $t9 to $a0 for printing
      syscall             # print $t9
      li   $v0, 10        # system call code for exit
      syscall
```

Q14. (continue...)

(a) What output does the above MIPS code produce? [2]

(b) Using the variable names (*A*, *B*, *size*, *answer*) shown in the variable mappings above, write an equivalent C code corresponding to instructions I1 to I18 in the above MIPS code. You may use additional variable(s) if needed.

Do not do a line-by-line direct translation of the MIPS code. You do not need to declare the variables in your C code. [4]

(c) Write the instruction encoding of instruction I7 (`beq $t3, $0, end`) in hexadecimal. [2]

(d) Write the instruction encoding of instruction I8 (`lw $s1, 0($t1)`) in hexadecimal. [2]

(e) Write the instruction encoding of instruction I18 (`j loop`) in hexadecimal, assuming that instruction I1 (`add $t0, $0, $0`) is stored at address **0x2B00 0FFC**. [3]

Q15. Pipelining [14 marks]

Refer to the following MIPS code which is the same as the one in question 14. Here, we look only at instructions I1 to I18.

```
      add  $t0, $0, $0    # I1
      add  $t9, $0, $0    # I2
      addi $t1, $a1, 0    # I3
      addi $t2, $a2, 0    # I4
      sll  $t8, $a0, 2    # I5
loop: slt  $t3, $t0, $t8  # I6
      beq  $t3, $0, end   # I7
      lw   $s1, 0($t1)    # I8
      lw   $s2, 0($t2)    # I9
      slt  $t3, $s1, $s2  # I10
      bne  $t3, $0, else  # I11
      add  $t9, $t9, $s1  # I12
      j    cont           # I13
else: sub  $t9, $t9, $s2  # I14
cont: addi $t0, $t0, 4    # I15
      addi $t1, $t1, 4    # I16
      addi $t2, $t2, 4    # I17
      j    loop           # I18
end:
```

Assuming a 5-stage MIPS pipeline and *A*[0] is larger than *B*[0], answer the parts below. You need to count until the last stage of instruction I18.

(a) How many cycles does this code segment take to complete its execution in the first iteration (I1 to I18) in an ideal pipeline, that is, one with no delays? [2]

For parts (b) to (d) below, given the assumption for each part, how many additional cycles does this code segment (I1 to I18) take to complete its execution in the first iteration as compared to an ideal pipeline computed in (a)? Note that the jump instruction (j) computes the target address to jump to in its ID stage (stage 2). No delayed branching is used.

Write the total number of additional delay cycles for each of the parts (b) to (d). For example, if part (b) takes 30 cycles and part (a) takes 10 cycles, then you should write 20 for part (b).

(b) Assuming without forwarding and branch decision is made at MEM stage (stage 4). No branch prediction is made. [3]

(c) Assuming with forwarding and branch decision is made at MEM stage (stage 4). No branch prediction is made. [3]

(d) Assuming with forwarding and branch decision is made at ID stage (stage 2). Branch is predicted not taken. [3]

(e) Assuming the setting in part (b) above (without forwarding and branch decision at MEM stage), without affecting the correctness of the code, is it possible to move one instruction to somewhere else to reduce the number of delay cycles? If so, indicate which instruction to move, where to move it to, and how many delay cycles are reduced by moving it. If it is not possible, explain. [3]

Q16. Cache [18 marks]

Refer to the following MIPS code which is the same as the one in question 14. Here, we look only at instructions I1 to I18. The data segment in the MIPS code in question 14 no longer applies here as the arrays contain a lot more elements in this question.

```
      add  $t0, $0, $0    # I1
      add  $t9, $0, $0    # I2
      addi $t1, $a1, 0    # I3
      addi $t2, $a2, 0    # I4
      sll  $t8, $a0, 2    # I5
loop: slt  $t3, $t0, $t8  # I6
      beq  $t3, $0, end   # I7
      lw   $s1, 0($t1)    # I8
      lw   $s2, 0($t2)    # I9
      slt  $t3, $s1, $s2  # I10
      bne  $t3, $0, else  # I11
      add  $t9, $t9, $s1  # I12
      j    cont           # I13
else: sub  $t9, $t9, $s2  # I14
cont: addi $t0, $t0, 4    # I15
      addi $t1, $t1, 4    # I16
      addi $t2, $t2, 4    # I17
      j    loop           # I18
end:
```

For parts (a) to (d): You are given a **direct-mapped data cache** with 512 words in total and each block contains 8 words. You may assume the following:

- Array *A* starts at address **0xCDEF 0400**.
- Array *B* follows immediately after array *A* in the memory. That is, if the last element of array *A* is at address *x*, then the first element of array *B* is at address (*x* + 4).

(a) How many bits are there in the index field? In the byte offset field? [2]

(b) Assuming that array *A* contains **1032** elements and array *B* contains the same number of elements, what is the hit rate of the data cache (i) for array *A* and (ii) for array *B*? Write your answers in fractions if they are not equal to zero. [2]

(c) Assuming that array *A* contains **1028** elements and array *B* contains the same number of elements, what is the hit rate of the data cache (i) for array *A* and (ii) for array *B*? Write your answers in fractions if they are not equal to zero. [4]

(d) Assuming that arrays *A* and *B* have at least 100 elements each, (i) what is the lowest hit rate possible for array *A*? Write your answer in fraction if it is not zero; (ii) how many elements in array *A* would result in this lowest hit rate? [2]

Q16. Cache (continue...)

For parts (e) and (f): You are given a **2-way set-associative instruction cache** with **LRU** replacement policy. You may assume the following:

- There are **100 elements** in array *A* and the same number of elements in array *B*.
- All elements in array *A* are positive integers and all elements in array *B* are negative integers.
- Instruction I1 is stored at address **0x2B00 0FFC**.
- Consider only instructions I1 to I18 in the execution of the code. (Make sure the execution processes all the elements in the arrays.)

(e) The instruction cache contains 16 words in total and each block contains 4 words.

(i) How many bits are there in the set index field? In the byte offset field? [2]

(ii) How many misses are there in the execution of the code? [3]

(f) The instruction cache contains 16 words in total and each block contains 2 words. How many misses are there in the execution of the code? [3]

=== END OF PAPER ===

exam16s2_qns

Wednesday, November 29, 2023 10:29 PM

exam16s2_qns

NATIONAL UNIVERSITY OF SINGAPORE

CS2100 – COMPUTER ORGANISATION

(Semester 2: AY2016/17)

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This assessment paper consists of **SIX (6)** questions and comprises **TWELVE (12)** printed pages.
2. This is a **CLOSED BOOK** assessment. One handwritten double-sided A4 reference sheet is allowed. Calculators are not allowed.
3. Answer all questions and write your answers in the **ANSWER BOOKLET** provided.
4. Fill in your Student Number with a pen clearly on your ANSWER BOOKLET.
5. You may use pencil to write your answers.
6. Page 9 contains the MIPS Reference Data sheet and page 10 contains the MIPS Datapath.
7. Pages 11 and 12 are for your rough work.
8. You are to submit only the **ANSWER BOOKLET** and no other document.

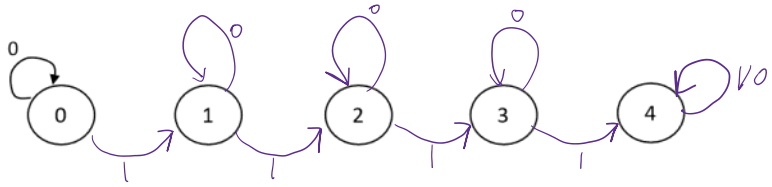
1. **[10 marks]**
A theme park offers locker rental to its visitors. To use a locker, a visitor deposits 4 tokens, one at a time, into the locker's token slot.

1. **[10 marks]**
A theme park offers locker rental to its visitors. To use a locker, a visitor deposits 4 tokens, one at a time, into the locker's token slot.

Design a sequential circuit with states *ABC* for the locker's door using a *D* flip-flop for *A*, a *T* flip-flop for *B*, and a *JK* flip-flop for *C*. The circuit consists of 5 states representing the number of tokens a visitor has deposited: 0, 1, 2, 3 and 4 (or, in binary, *ABC* = 000, 001, 010, 011 and 100). The visitor can deposit only one token at a time. When the circuit reaches the final state 4, it remains in state 4 even if the visitor continues to put tokens into the slot.

Let the external input *t* denotes a token.

a. Complete the given state diagram on the Answer Booklet. The state values are shown in decimal. The value on the arrow represents *t*. [2 marks]



b. Write the **simplified SOP expressions** for the flip-flop inputs. [8 marks]

2. **[10 marks]**
a. Given the following Boolean function:

$$F(A,B,C,D) = \Sigma m(1, 4, 5, 6, 7, 13)$$

You are to implement *F* using a single **2-bit magnitude comparator** with no additional logic gates. Note that complemented literals are not available. [5 marks]

b. Given the following Boolean function:

$$G(A,B,C,D) = \Sigma m(2, 11)$$

You are to implement *G* using a single **2×4 decoder** with one-enable and active high outputs, and one 2-input exclusive-OR gate. Note that complemented literals are not available. [5 marks]

3. **[10 marks]**
Study the following MIPS program. Arrays *A* and *B* are integer arrays.

```
     # $s0 contains the starting address of array A
     # $s1 contains the starting address of array B
     add  $s2, $s0, $zero      #inst 1
     add  $s3, $s1, $zero      #inst 2
L1:  lw   $t0, 0($s2)          #inst 3
     lw   $t1, 0($s3)          #inst 4
     bne  $t0, $zero, L2       #inst 5
     beq  $t1, $zero, done     #inst 6
L2:  slt  $t2, $t0, $t1        #inst 7
     beq  $t2, $zero, L3       #inst 8
     sw   $t0, 0($s3)          #inst 9
     sw   $t1, 0($s2)          #inst 10
L3:  addi $s2, $s2, 4          #inst 11
     addi $s3, $s3, 4          #inst 12
     j    L1                   #inst 13
done:
```

a. Give the instruction encoding in hexadecimal for instruction 1 (**add $s2, $s0, $zero**). The opcode for **add** is 0 and the funct value for **add** is 0x20. [2 marks]

b. Give the instruction encoding in hexadecimal for instruction 6 (**beq $t1, $zero, done**). The opcode for **beq** is 0x04. [2 marks]

c. If instruction 13 (**j L1**) is at memory address **0xE0480030**, give the instruction encoding in hexadecimal for instruction 13. The opcode for jump is 0x02. [2 marks]

d. The following are the initial values of the array elements in arrays *A* and *B*.

Array *A*:	3	-1	7	0	2	-5	9	0	-9	1
Array *B*:	2	5	7	3	6	0	8	0	-3	2

Fill in the final values of the elements. [4 marks]

4. **[35 marks]**

Zephyr is a 32-bit stack-based processor with the specifications shown below. In this table, registers "x" and "y" serve as placeholders for actual general purpose registers $1, $2, …, $8, the capital letter "V" refers to a single variable, the capital letter "A" refers to the first element of an array, and the small letter "c" refers to a constant. The capital letters "X" and "Y" refer to the top-most and second elements of the stack respectively. All constants and displacements are 2's complement signed values.

Addressing Architecture:	Stack based.
Number of General Purpose Registers:	Eight ($1, $2, …, $8)
Special registers:	Stack pointer ($sp) Program counter ($pc)
Instruction formats:	Fixed length 32-bit instructions
Arithmetic instructions: Arithmetic instructions. X is the top-most operand on the stack, Y is the next operand in the stack.	**ADD**: X and Y are popped off the stack, X+Y are pushed back onto the stack. **SUB**: X and Y are popped off the stack, X-Y is pushed back onto the stack. **MUL**: X and Y are popped off the stack, X*Y is pushed back onto the stack. **DIV**: X and Y are popped off the stack. X/Y is pushed back onto the stack.
Stack instructions: Stack manipulation instructions. Special register $sp points to the top-most element of the stack. Stacks are assumed to be arbitrarily large, while popping an empty stack will cause an error, but we WILL NOT consider that here. We will assume that the stack is never empty nor full.	**PUSHI c:** Push immediate value c onto the stack. **PUSH $y**: Push register y onto the stack. **POP $y**: Pop topmost item on the stack into register y. **ZERO:** Reset $sp to bottom of stack.

4. (continued)

Load/Store Instructions: These are load and store instructions that get data from memory to registers and vice versa. All addresses are byte addresses.	**LW $y, $x**: Load 32-bit word stored in address pointed to by register x into register y. **SW $y, $x**: Store 32-bit word in register y, into the address pointed to by register x. **LB $y, $x:** Load a single byte stored in the address indicated by register x, into the lowest (bits 7-0) bits of register y. **SB $y, $x:** Store the lowest 8-bits (bits 7-0) of register y into the byte address indicated by register x. **LDI $y, c**: Store immediate constant c into register y. **LDI $y, V**: Store address of variable V into register y. **LDI $y, A**: Store base address of array A into register y. **INCW $y:** Register y is incremented by 4. **DECW $y:** Register y is decremented by 4. **INC $y:** Register y is incremented by 1. **DEC $y:** Register y is decremented by 1.
B-type instructions: These are compare and branch instructions.	**BEQ $x, $y, displ:** Jump to address ($pc+4) + 4*displ if register x == register y. **BNE $x, $y, displ:** Jump to address ($pc+4) + 4*displ if register x != register y **BLT $x, $y, displ:** Jump to address ($pc+4) + 4*displ if x < y **BGT $x, $y, displ:** Jump to address ($pc+4) + 4*displ if x>y

4. (continued)

a. Using the instruction set given above, write the Zephyr assembly language equivalent of this program. Ensure that your code is properly commented. [5 marks]

```
for (i=0; i<5; i++)
   if (x[i] < 3)
      x[i] = x[i] + 5;
```

All offsets in Zephyr are expressed as 16-bit word addresses, while registers are expressed as 3-bit register numbers (000_2=\$1, 001_2=\$2, ..., 111_2=\$8). Similarly, all constants in Zephyr are 16-bit long.

There are six classes of instructions:

A: No operands (e.g. ADD)

B: One register operand (e.g. PUSH \$1)

C: One constant operand (e.g. PUSHI c)

D: One register and one constant operand (e.g. LDI \$1, c)

E: Two registers (e.g. LW \$1, \$2)

F: Two registers and a displacement (e.g. BEQ \$1, \$2, displ)

b. Sketch the instruction formats for all 6 classes, assuming that all 32 bits of a Zephyr instruction word are utilized fully, and that we maximize the number of opcode bits possible each time. [12 marks]

c. If we utilize an expanding opcode scheme for Zephyr, what is the maximum number of opcodes possible, assuming that there are at least one instruction in each class? Show your working and reasoning process. Where convenient you may leave your answers in terms of powers of 2. [5 marks]

d. What is the minimum number of opcodes possible, assuming that there are at least one instruction in each class? Show your working and reasoning process. Again, where convenient you may leave your answers in terms of powers of 2. [5 marks]

e. What is the furthest forward distance that you can branch to, in the BEQ, BNE, BLT and BGT instructions? Express your answer in number of instructions. [2 marks]

f. Suppose that we have an array *A* of words (i.e. we access elements of *A* one word at a time). What is the maximum size of *A*, expressed in words? [3 marks]

g. Suppose again that we have an array *B* of bytes (i.e. we access elements of *B* one byte at a time). What is the maximum size of *B*, expressed in bytes? [3 marks]

5. **[15 marks]**
In this question we want to modify the (non-pipelined) MIPS datapath to support two new instructions: BLT and BGT – "branch on less than" and "branch on greater than".

The BLT and BGT instructions are shown below:

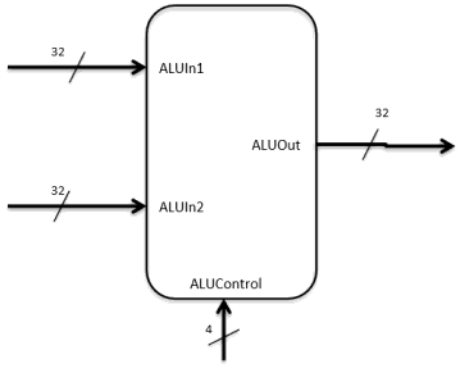
BLT:

0x08	rs	rt	displ

BGT:

0x12	rs	rt	displ

a. The ALU for the MIPS processor is shown below as a single block with two 32-bit inputs, and one 32-bit output. Show, by adding AT MOST ONE 32-input logic gate and any additional wires, how to generate the **IsZero** and **IsNegative** signals. The IsZero signal is 1 when ALUIn1 – ALUIn2 is zero, and the IsNegative signal is 1 when ALUIn1 – ALUIn2 is negative. [4 marks]



b. The CONTROL unit in the datapath must now generate **BranchLess** and **BranchGreater** signals from the instruction bits. Sketch the combinational circuits to generate these signals. [5 marks]

c. For your convenience the MIPS datapath is shown in page 10. Sketch the combinational logic circuit needed to generate the **PCSrc** control signal to support the BEQ, BLT and BGT instructions. [6 marks]

6. **[20 marks]**
Suppose we have a cache that has an access time of 5ns, and a main memory with an access time of 80ns.

a. What is the memory access time when you have a cache hit? [2 marks]

b. What is the memory access time when you have a cache miss? [3 marks]

c. You run some benchmarks on your system and find that 10,000 accesses take a total of 70 microseconds (1 microsecond = 1000 nanoseconds). What is the miss rate of your cache? [5 marks]

Your cache is implemented as a 4-way set associative write-back cache totaling 64KB. Each cache block holds 8 words of 4 bytes each. CPU addresses are 32 bits long.

d. How many bits per set do you require to store the tags? [4 marks]

e. Assuming that the 64KB of cache refers purely to "usable cache" – i.e. cache that is used only to store data or instructions, and not overheads like tag bits, what is the total amount of static RAM that you require to implement this cache? [6 marks]

~~ END OF PAPER ~~~

(The next two pages contain the MIPS Reference Data sheet and the MIPS Datapath.)

MIPS Reference Data

CORE INSTRUCTION SET

NAME, MNEMONIC		FORMAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	7_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		0 / 08_{hex}
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27_{hex}
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25_{hex}
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / $2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00_{hex}
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		0 / 02_{hex}
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (atomic) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22_{hex}
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23_{hex}

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1'b0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

BASIC INSTRUCTION FORMATS

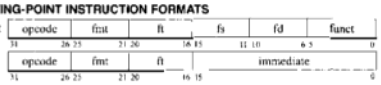
ARITHMETIC CORE INSTRUCTION SET

NAME, MNEMONIC		FORMAT	OPERATION		OPCODE / FMT /FT / FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr	(4)	11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/--/--/1b
FP Add Single	add.s	FR	F[fd]= F[fs] + F[ft]		11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}		11/11/--/0
FP Compare Single	c.x.s*	FR	FPcond = (F[fs] *op* F[ft]) ? 1 : 0		11/10/--/*y*
FP Compare Double	c.x.d*	FR	FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0		11/11/--/*y*

* (x is eq, lt, or le) (*op* is ==, <, or <=) (*y* is 32, 3c, or 3e)

FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}		11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}		11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--
Move From Hi	mfhi	R	R[rd] = Hi		0 /--/--/10
Move From Lo	mflo	R	R[rd] = Lo		0 /--/--/12
Move From Control	mfc0	R	R[rd] = CR[rs]		10 /0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Shift Right Arith.	sra	R	R[rd] = R[rt] >>> shamt		0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS

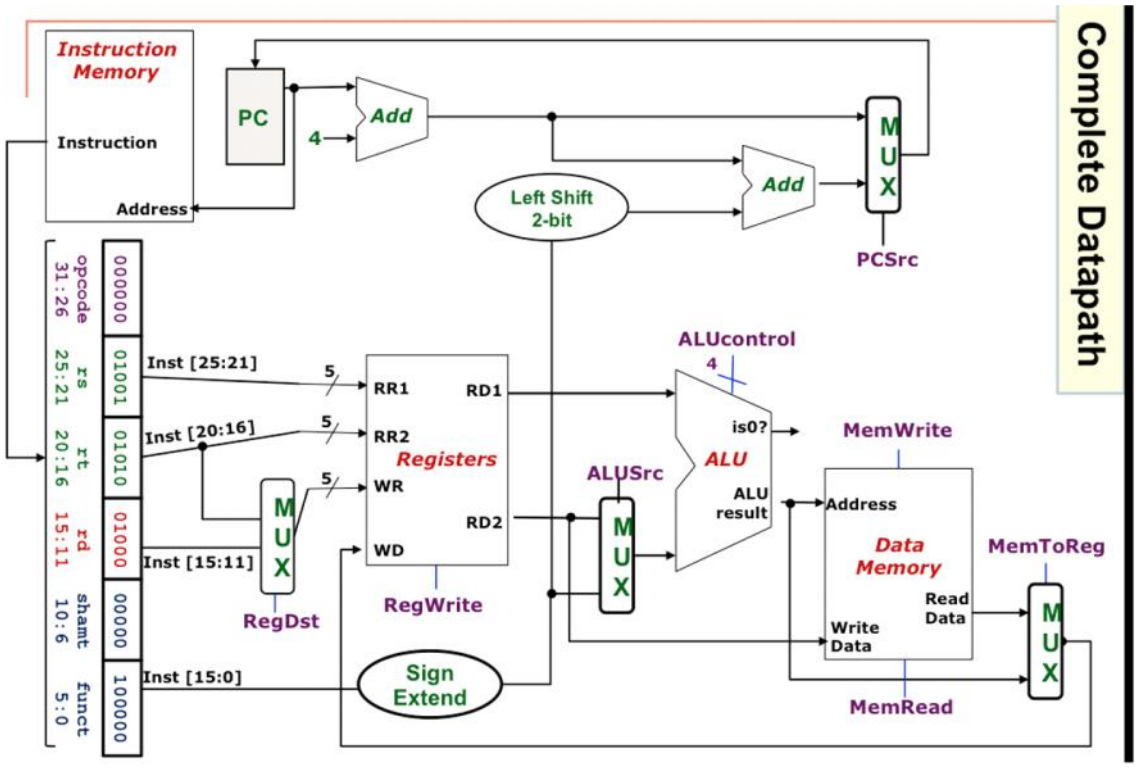


PSEUDOINSTRUCTION SET

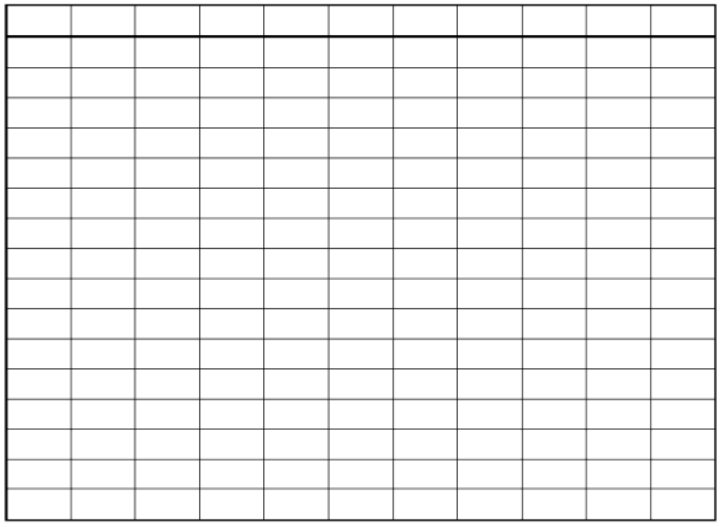
NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes


Complete Datapath
Instruction Memory
Instruction
Address
PC
4
Add
Add
Left Shift 2-bit
M U X
PCSrc
opcode 31:26
rs 25:21
rt 20:16
rd 15:11
shamt 10:6
funct 5:0
000000
01001
01010
01000
00000
100000
Inst [25:21]
Inst [20:16]
Inst [15:11]
Inst [15:0]
5
RR1
RR2
WR
WD
RD1
RD2
Registers
RegDst
RegWrite
Sign Extend
ALUcontrol
4
ALUSrc
is0?
ALU
ALU result
MemWrite
Address
Data Memory
Read Data
Write Data
MemRead
MemToReg

(This page is for your rough work.)



(This page is for your rough work.)