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]

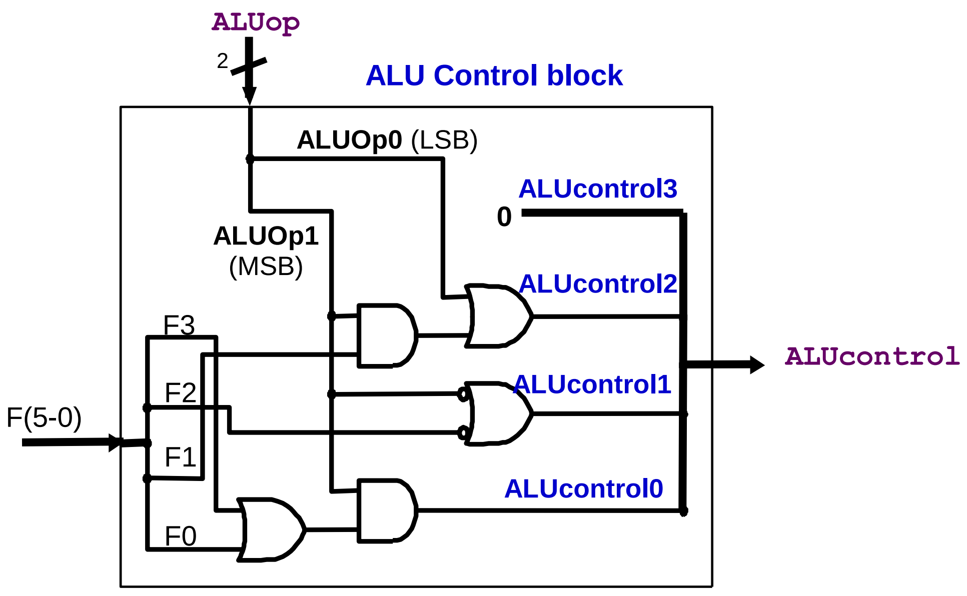
(b) Consider an instruction **`0x0800C840`** at address **`0x2100FFFC`**. What is the next value of PC when the instruction is executed using the incorrect processor above?

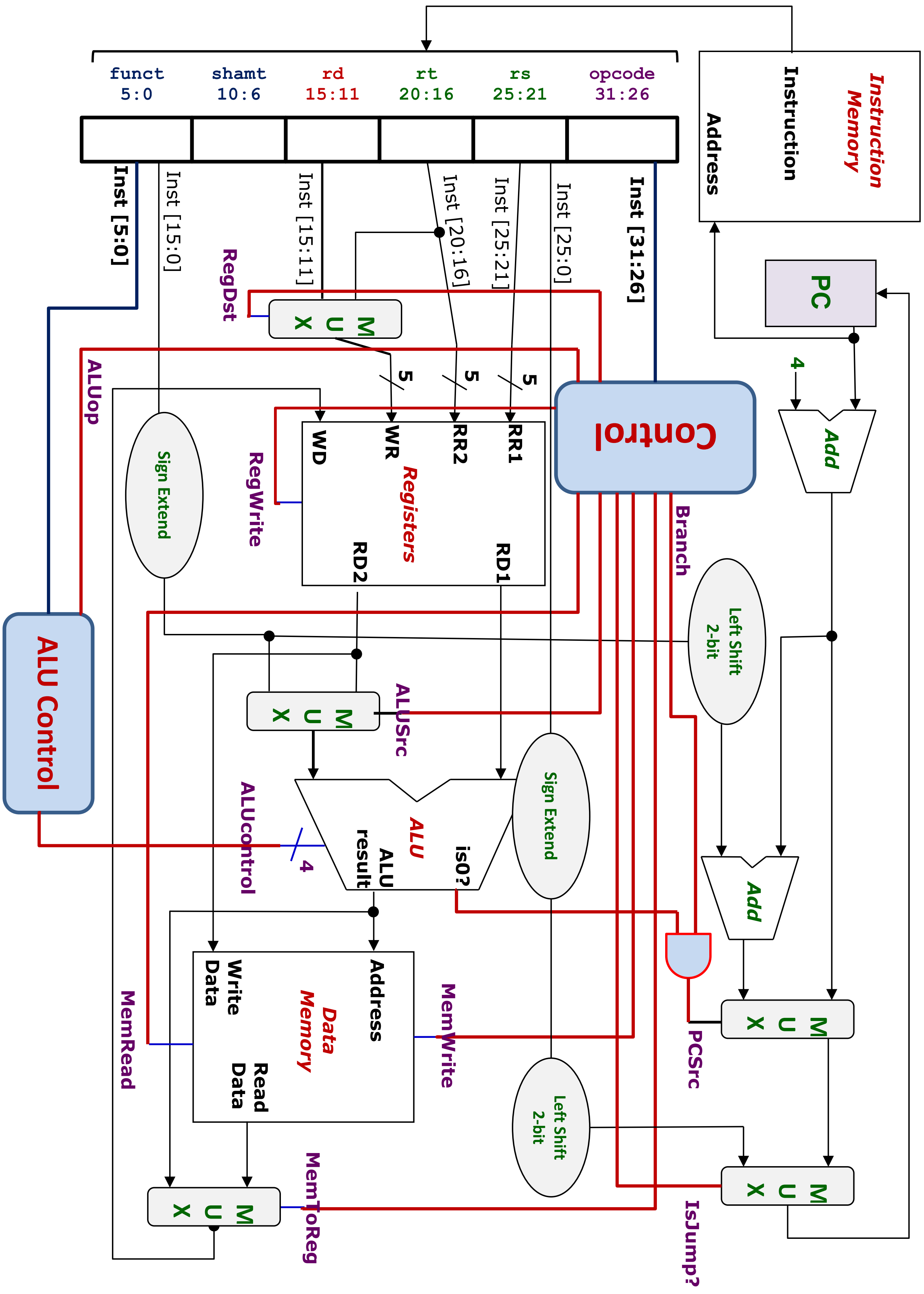
[3 marks]

(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]

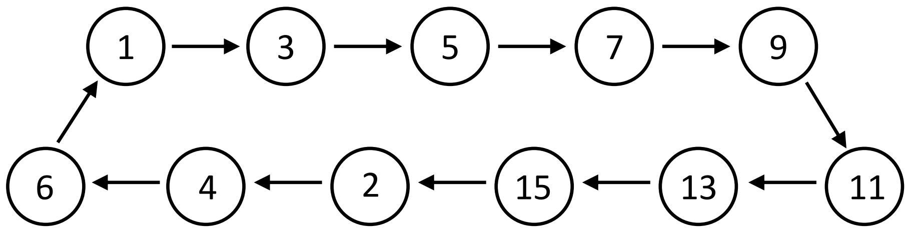
(d) Modify the combinational circuit given in the answer booklet to include **ALUop1**, **ALUop0** and **`IsJump?`** control signals. [4 marks]

(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]




funct 5:0
shamt 10:6
rd 15:11
rt 20:16
rs 25:21
opcode 31:26
Instruction Memory
Instruction
Address
PC
4
Add
Control
Inst [5:0]
Inst [15:0]
Inst [15:11]
Inst [20:16]
Inst [25:21]
Inst [25:0]
Inst [31:26]
RegDst
MUX
5
5
5
ALUop
WD
WR
RR2
RR1
Registers
RD2
RD1
RegWrite
Sign Extend
Branch
Left Shift 2-bit
ALU Control
MUX
ALUSrc
ALU
ALU result
is0?
ALUcontrol
4
Sign Extend
Add
Address
Write Data
Data Memory
Read Data
MemRead
MemWrite
MUX
PCSrc
Left Shift 2-bit
MUX
MUX
MemToReg
IsJump?

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*.

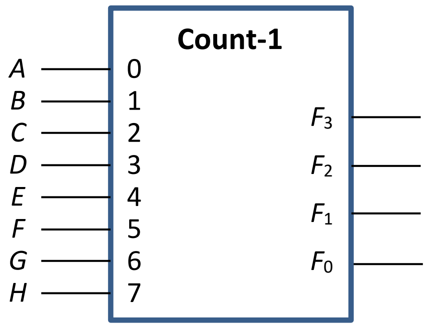
a. Write out the **simplified SOP expressions** for all the flip-flop inputs. [10 marks]

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]

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…)

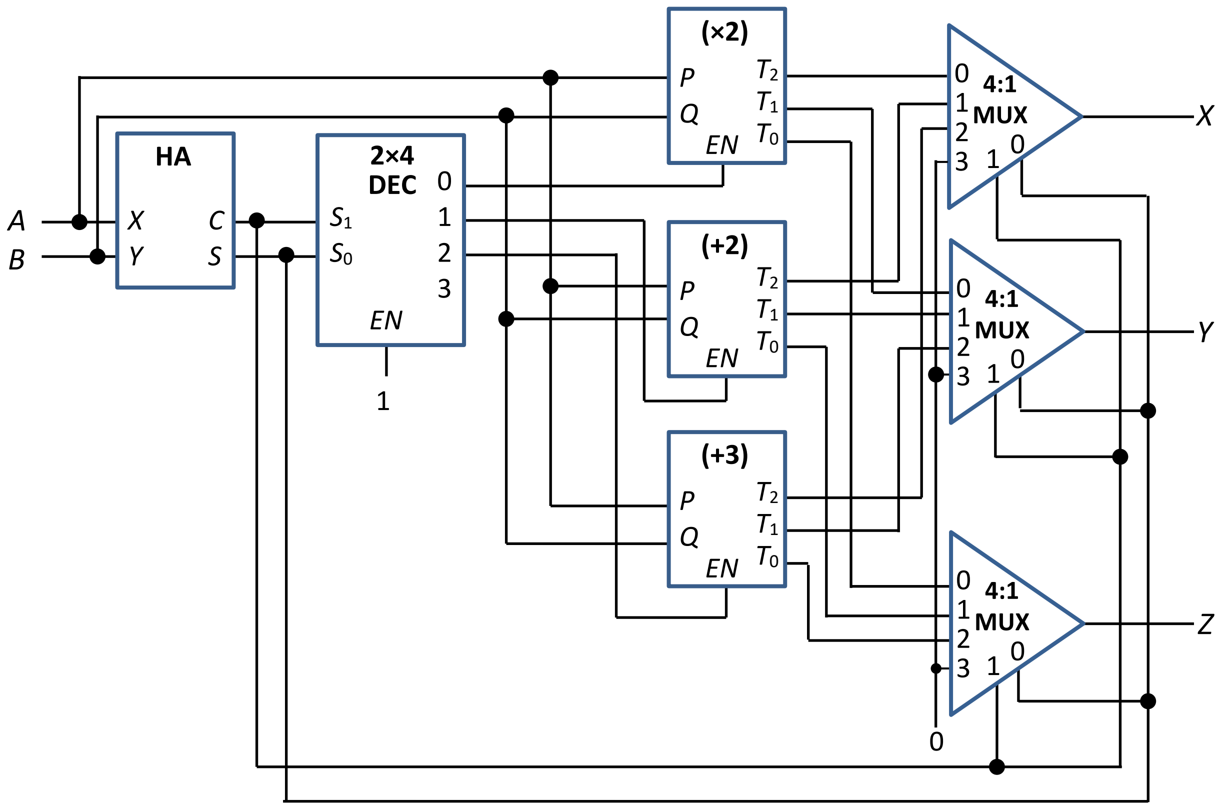
(d) Implement the following four-variable function $R(A,B,C,D)$ using a <u>single</u> **4:1 multiplexer** <u>without any additional logic gates</u>. [6 marks]

$$R(A,B,C,D) = \Sigma m(0, 2, 3, 4, 6, 7, 12, 14)$$

(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 $(P+Q)\times 2$.
- 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:

$$A[k] = A[k] + B[k/2] \quad \text{if } k \text{ is even}$$
$$A[k] = A[k] + C[(k-1)/2] \quad \text{if } k \text{ is odd}$$

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:
```

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 0x**00000080**, 0x**00100000** and 0x**00108040** 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]

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 <u>with forwarding and early branching</u>, 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 <u>with forwarding</u>, 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

①

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{1b'0}, 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

②

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.)