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 <u>one</u> 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: $\mathbf{(135)_x + (25)_x = (161)_x}$?

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:

   C 4 0 0 7 0 0 0

   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]

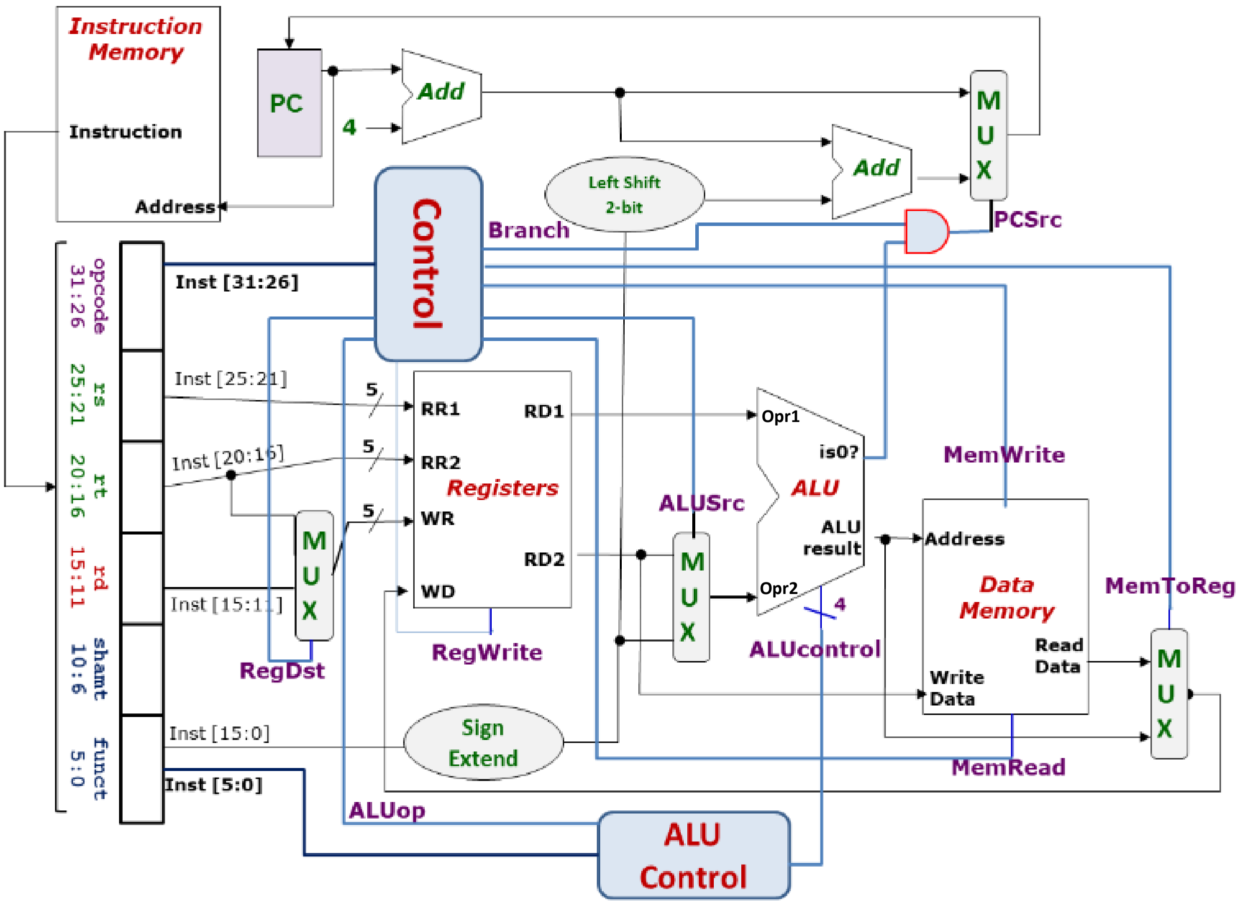
8. Write out the output of the following program. [3 marks]

```
#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 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 (31–26)	rs (25–21)	rt (20–16)	rd (15–11)	shamt (10–6)	funct (5–0)
I	opcode (31–26)	rs (25–21)	rt (20–16)	immediate (15–0)		
J	opcode (31–26)	address (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 (31–26)	fmt (25–21)	ft (20–16)	fs (15–11)	fd (10–6)	funct (5–0)
FI	opcode (31–26)	fmt (25–21)	ft (20–16)	immediate (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	Yes