NATIONAL UNIVERSITY OF SINGAPORE

CS2100 – COMPUTER ORGANISATION

(Semester 2: AY2017/18)

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This assessment paper consists of **SEVEN (7)** questions and comprises **FOURTEEN (14)** printed pages.
2. This is a **CLOSED BOOK** assessment. One double-sided A4 reference sheet is allowed.
3. Calculators and computing devices such as laptops and PDAs are not allowed.
4. Answer all questions and write your answers in the **ANSWER BOOKLET** provided.
5. Fill in your Student Number clearly with a pen on your ANSWER BOOKLET.
6. Do NOT write your name on your ANSWER BOOKLET.
7. You may use pencil to write your answers.
8. Page 9 onwards contain a blank page, the MIPS Reference Data Sheet and several blank tables for your rough works.
9. You are to submit only the **ANSWER BOOKLET** and no other document.

1. **[10 marks]**

(a) Write the output of the following C program. [4 marks]

```
#include <stdio.h>

typedef struct {
    int  val;
    char ch[2];
} rec_t;

void process1(rec_t *);
void process2(rec_t);

int main(void) {
    rec_t st[2] = {{11,{'A','B'}}, {22,{'C','D'}}};

    process1(&st[1]);
    process2(st[0]);
    printf("%d %c\n", st[0].val, st[0].ch[0]);
    printf("%d %c\n", st[1].val, st[1].ch[1]);
    return 0;
}

void process1(rec_t *para) {
    para->val = 33;
    para->ch[0] += ('a' - 'A') + 1;
    para->ch[1] += ('a' - 'A') + 2;
}

void process2(rec_t para) {
    para.val = 44;
    para.ch[0] += ('a' - 'A') + 3;
    para.ch[1] += ('a' - 'A') + 4;
}
```

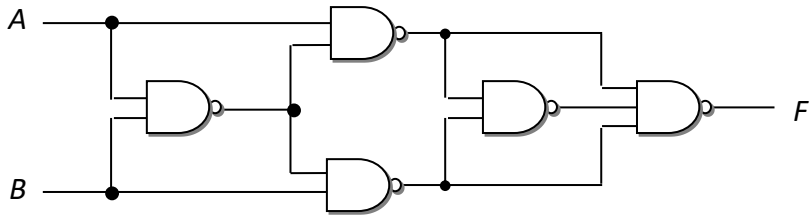
(b) Given the following hexadecimal representation in IEEE 754 single-precision floating-point number system:

42F64000

What is the decimal value it represents? [3 marks]

1. *(continue…)*

(c) Given the logic circuit below:

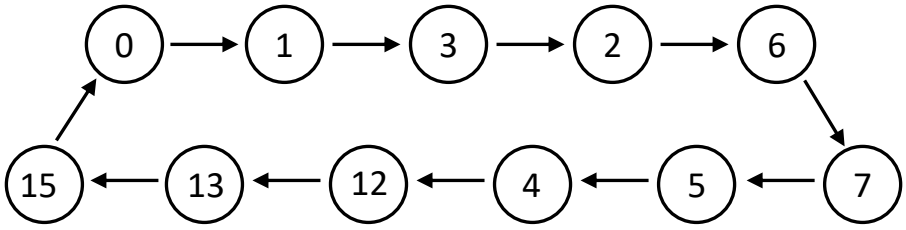


(i) What is *F*? [2 marks]

(ii) What is the circuit propagation delay if the propagation delay of a NAND gate with fan-in of *n* is *nt*? [1 mark]

---

2. **[15 marks]**
A sequential circuit goes through the following states, whose state values are shown in decimal:



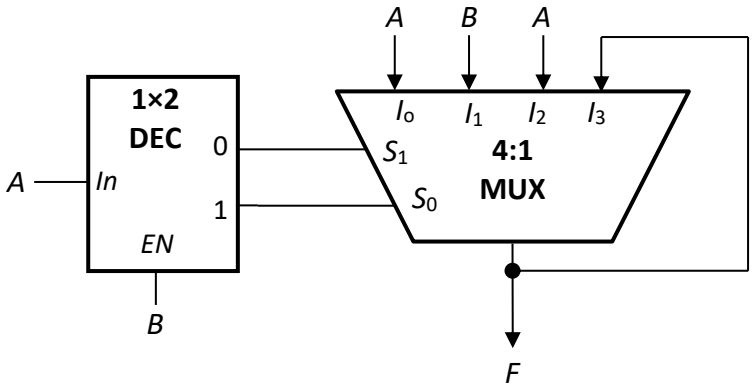
The states are represented by 4-bit values *ABCD*. Implement the sequential circuit using a *D* flip-flop for *A*, a *D* flip-flop for *B*, a *T* flip-flop for *C*, and a *JK* flip-flop for *D*.

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]

3. **[20 marks]**

(a) Given the following circuit, what is *F*? [4 marks]



(b) Given $G(A,B,C,D) = \Pi M(1, 2, 6, 8, 9, 11, 13)$, implement *G* using a single 8:1 multiplexer without any additional logic gates. Complemented literals are not available. [4 marks]

(c) Given $H(A,B,C,D) = \Sigma m(12, 13)$, implement *H* using a single 2×4 active high output decoder with 1-enable, without any additional logic gates. Complemented literals are not available. [4 marks]

(d) 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 in BCD code as 0011 1001 0110.

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 $F_{11}F_{10}F_9F_8\ F_7F_6F_5F_4\ F_3F_2F_1F_0$. You are free to use the logical constants 0 and 1.

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:* To help you, you may fill in the table on the Answer Booklet that computes $5\times A$. This table is worth 2 marks.)

[8 marks]

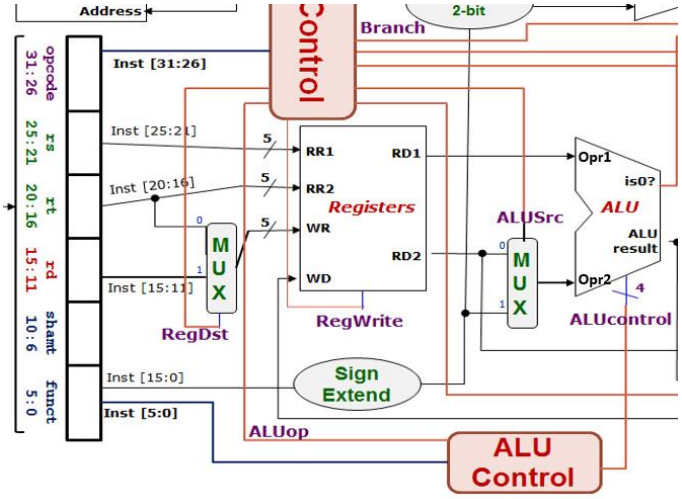
4. **[12 marks]**

(a) Suppose MIPS instructions in R-format must use the following five opcodes (in decimal): 0, 1, 16, 17 and 32, what is the maximum total number of instructions that can be supported in MIPS? [2 marks]

(b) Suppose due to a hardware defect in the datapath circuit, a stuck-at-0 fault occurs at **bit 6** of every MIPS instruction. This means that bit 6 of a MIPS instruction is always 0 regardless of what the instruction is originally. Devise a simple test using a MIPS instruction to discover this error. Explain your test. Keep your explanation clear and short, in no more than 2 sentences. [3 marks]

(c) The diagram on the right shows a portion of the datapth.

Suppose the stuck-at-0 fault occurs at the **ALUSrc** control signal. Assuming that **$t0** and **$t1** contains 12 and 34 respectively, and we are to use the instruction **lw $t1, 0($t0)** to discover the error. Describe what other preparation work needs to be done. You may assume that we can write data into any location in the memory. [3 marks]



(d) The table below shows the ALUcontrol signal of the datapath we discussed in class.

Opcode	ALUop	Instruction operation	Funct field	ALU action	ALU control
lw	00	load word	xxxxxx	add	0010
sw	00	store word	xxxxxx	add	0010
beq	01	branch equal	xxxxxx	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

You want to add the **bne** instruction into the datapath, which already includes the required hardware for the instruction. Write out the ALUop for **bne** and how you can determine whether the **bne** results in the branch to be taken. [4 marks]

5. **[15 marks]**
Study the MIPS program below. *A* and *B* are integer arrays whose base addresses are in **$s0** and **$s1** respectively. The arrays are of the same size *n* (number of elements). **$s2** contains the value *n*. The address of the first **beq** instruction is 0x0040003c.

```
# Q5.asm
.data
A: .word 11, 9, 31, 2, 9, 1, 6, 10
B: .word 3, 7, 2, 12, 11, 41, 19, 35
n: .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   $t0, n      # $t0 is the addr of n (size of array)
      [                ] # $s2 is the content of n

      beq  $s2, $zero, End   # Address: 0x0040003c
      addi $t8, $s2, -1
      sll  $t8, $t8, 2
Loop: add  $t0, $s0, $t8
      add  $t1, $s1, $t8
      lw   $t2, 0($t0)
      lw   $t3, 0($t1)
      andi $t4, $t3, 3
      addi $t4, $t4, -3
      beq  $t4, $zero, A1
      add  $t2, $t2, $t3
      j    A2
A1:   addi $t2, $t2, 1
A2:   sw   $t2, 0($t0)
      addi $t8, $t8, -8
      slt  $t7, $t8, $zero
      beq  $t7, $zero, Loop
End:  li   $v0, 10           # system call code for exit
      syscall
```

a. Fill in the missing instruction (the fourth line in the program text) to store the value of *n* into **$s2**. Do not use any pseudo-instruction. [1 mark]

b. Fill in the values of array *A* after the execution of the code. [4 marks]

c. Write an equivalent C code that does the same work. Use variables *A* and *B* for the arrays, and *n* for the size of the array. You do not need to declare *A*, *B* and *n*. [4 marks]

Give the instruction encoding in hexadecimal for the following 3 instructions:

d. `sll  $t8, $t8, 2` (Note: rs = 0) [2 marks]

e. `j    A2` [2 marks]

f. `slt  $t7, $t8, $zero` [2 marks]

6. **[14 marks]**
Refer to the same MIPS code in the previous question, except that now we focus only on a section of the code which is reproduced below:

```
      beq  $s2, $zero, End   # Inst1
      addi $t8, $s2, -1      # Inst2
      sll  $t8, $t8, 2       # Inst3
Loop: add  $t0, $s0, $t8     # Inst4
      add  $t1, $s1, $t8     # Inst5
      lw   $t2, 0($t0)       # Inst6
      lw   $t3, 0($t1)       # Inst7
      andi $t4, $t3, 3       # Inst8
      addi $t4, $t4, -3      # Inst9
      beq  $t4, $zero, A1    # Inst10
      add  $t2, $t2, $t3     # Inst11
      j    A2                # Inst12
A1:   addi $t2, $t2, 1       # Inst13
A2:   sw   $t2, 0($t0)       # Inst14
      addi $t8, $t8, -8      # Inst15
      slt  $t7, $t8, $zero   # Inst16
      beq  $t7, $zero, Loop  # Inst17
End:
```

Assuming a 5-stage MIPS pipeline system with forwarding and early branching, that is, the branch decision is made at the ID stage. No branch prediction is made and no delayed branching is used. For the jump (j) instruction, the computation of the target address to jump to is done at the ID stage as well.

Assume also that the first **beq** instruction begins at cycle 1.

a. Suppose arrays *A* and *B* now each contains 200 positive integers. What is the minimum number and maximum number of instructions executed? (Consider only the above code segment from Inst1 to Inst17.) [2 marks]

b. List out the instructions where some stall cycle(s) are inserted in executing that instruction in the pipeline. These include delay caused by data dependency and control hazard. You may write the instruction number InstX instead of writing out the instruction in full. [6 marks]

c. How many cycles does one iteration of the loop (from Inst1 to Inst17) take if the **beq** instruction at Inst10 branches to *A1*? You have to count until the WB stage of Inst17. [3 marks]

d. How many cycles does one iteration of the loop (from Inst1 to Inst17) take if the **beq** instruction at Inst10 does not branch to *A1*? You have to count until the WB stage of Inst17. [3 marks]

7. **[14 marks]**
Refer to the same MIPS code in the previous two questions:

```
      beq  $s2, $zero, End   # Inst1, Address: 0x0040003c
      addi $t8, $s2, -1      # Inst2
      sll  $t8, $t8, 2       # Inst3
Loop: add  $t0, $s0, $t8     # Inst4
      add  $t1, $s1, $t8     # Inst5
      lw   $t2, 0($t0)       # Inst6
      lw   $t3, 0($t1)       # Inst7
      andi $t4, $t3, 3       # Inst8
      addi $t4, $t4, -3      # Inst9
      beq  $t4, $zero, A1    # Inst10
      add  $t2, $t2, $t3     # Inst11
      j    A2                # Inst12
A1:   addi $t2, $t2, 1       # Inst13
A2:   sw   $t2, 0($t0)       # Inst14
      addi $t8, $t8, -8      # Inst15
      slt  $t7, $t8, $zero   # Inst16
      beq  $t7, $zero, Loop  # Inst17
End:
```

Assuming that arrays *A* and *B* now each contains 1024 positive integers. Given a **direct-mapped data cache** with 128 words in total, each block containing 4 words with each word being 4 bytes long, arrays *A* and *B* are stored starting at memory addresses 0x10001000 and 0x1003F100 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 index field? In the byte offset field? [2 marks]

b. Which index is *A*[1023] mapped to? Which index is *B*[1023] mapped to? [4 marks]

c. How many memory accesses in total are made for array *A*? For array *B*? [2 marks]

d. What is the cache hit rate for array *A*? For array *B*? [2 marks]

e. Given a **direct-mapped instruction cache** with 16 words in total, each block containing 2 instructions (words), and the first **beq** instruction is at memory address 0x0040003c. How many cache hits and misses are there in total during the execution of the code, assuming that the **beq** instruction at Inst10 always branches to *A1*? You may consider only the instructions in the given code segment, that is, Inst1 through Inst17. [4 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

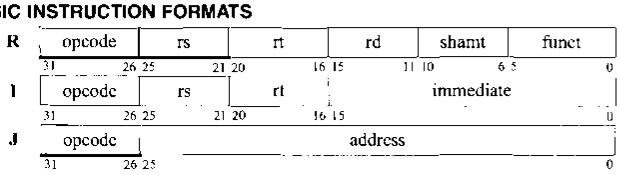
①

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

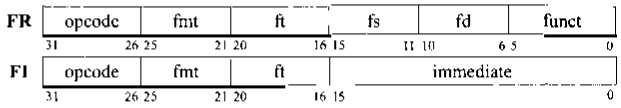


ARITHMETIC CORE INSTRUCTION SET

② OPCODE / FMT /FT / FUNCT (Hex)

NAME, MNEMONIC		FOR-MAT	OPERATION		
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

Copyright 2009 by Elsevier, Inc., All rights reserved. From Patterson and Hennessy, *Computer Organization and Design*, 4th ed.

(This page is for your rough work.)

A	*B*	*C*	*D*	A^+	B^+	C^+	D^+					

DA

DB

TC

JD

KD

(This page is for your rough work.)

(This page is for your rough work.)

	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 beq																														
I2 addi																														
I3 sll																														
I4 add																														
I5 add																														
I6 lw																														
I7 lw																														
I8 andi																														
I9 addi																														
I10 beq A1																														
I11 add																														
I12 J A2																														
I13 A1: addi																														
I14 A2: sw																														
I15 addi																														
I16 slt																														
I17 beq																														

(This page is for your rough work.)

	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 beq																														
I2 addi																														
I3 sll																														
I4 add																														
I5 add																														
I6 lw																														
I7 lw																														
I8 andi																														
I9 addi																														
I10 beq A1																														
I11 add																														
I12 J A2																														
I13 A1: addi																														
I14 A2: sw																														
I15 addi																														
I16 slt																														
I17 beq																														