NATIONAL UNIVERSITY OF SINGAPORE

CS2100 – COMPUTER ORGANISATION

(Semester 2: AY2024/25)

Time Allowed: 2 Hours

INSTRUCTIONS TO STUDENTS

1. This assessment paper consists of **TWENTY THREE (23)** questions in **TWO (2)** parts and comprises of **FIFTEEN (15)** 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 15 contains the MIPS Data Reference sheet.
7. The maximum mark of this assessment is 100.

Question	Max. mark
MCQs: Q1 – 18	36
Q19	12
Q20	14
Q21	13
Q22	12
Q23	13
Total	**100**

——— END OF INSTRUCTIONS ———

Part A: Multiple-Choice Questions [Total: 18×2=36 marks]

Each multiple-choice question (MCQ) is worth **TWO marks** and has exactly **one** correct answer. Please shade the corresponding bubbles on the Answer Sheets.

1. What is the output of the following C program?

```
#include <stdio.h>

struct Item {
    int x;
    int y;
};

void foo(struct Item *item, int n) {
    for (int i = 0; i < n; i++) {
        if ((item + i)->x % 2 == 0)
            continue;
        if ((item + i)->y + (item + i)->x > 12)
            break;
        (item + i)->y += (item + i)->x;
    }
}

int main() {
    struct Item item[4] = {{1, 4}, {2, 6}, {3, 7}, {5, 2}};
    foo(item, 4);
    for (int i = 0; i < 4; i++)
        printf("%d ", item[i].y);
    return 0;
}
```

A. `5 6 10 7`

B. `5 6 7 2`

C. `4 6 10 7`

D. `5 6 10 2`

E. None of options (A), (B), (C), (D) are correct.

2. What is the output of the following C program?

```
#include <stdio.h>

int foo(int *x){
    return (*x)++;
}

int main() {
    int a = -1, b = 2, c = -1, result = 0;
    if ((a || foo(&b)) && (foo(&c) && !c)) {
        result = a + b + c;
    }
    printf("b = %d, c = %d, result = %d\n", b, c, result);
    return 0;
}
```

A. **b = 3, c = 0, result = 2**

B. **b = 2, c = 0, result = 0**

C. **b = 2, c = 1, result = 0**

D. **b = 2, c = 0, result = 1**

E. None of options (A), (B), (C), (D) are correct.

3. An 8-bit register contains the value 3333 in base-4. A student assumes this is a large positive number. However, the register uses 2's complement format to represent signed integers. What is the correct signed decimal value of this 8-bit 2's complement number?

A. 255

B. –1

C. –127

D. 0

E. None of options (A), (B), (C), (D) are correct.

4. The 32-bit IEEE 754 hexadecimal floating-point number **0x3F838000** corresponds to which of the following decimal values?

   A. 1.015625
   B. 1.02734375
   C. 1.03125
   D. 1.0625
   E. None of options (A), (B), (C), (D) are correct.

5. You are given the following 32-bit MIPS-x instruction in hexadecimal: **0x212A000A**.

   In this MIPS-x system, everything is identical to the MIPS system, except the immediate field which uses excess-2^{15} encoding. Which of the following is the correct decoded MIPS-x assembly instruction?

   A. addi $t1, $t2, 10
   B. addi $t2, $t1, 10
   C. addi $t1, $t2, -32758
   D. addi $t2, $t1, -32758
   E. None of options (A), (B), (C), (D) are correct.

6. A MIPS processor has a fault in one control signal, causing the following behaviour:

   - addi and lw instructions do not work correctly
   - R-type instructions (e.g., add) work correctly
   - sw instruction works correctly

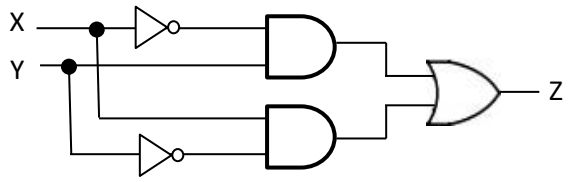
   Which control signal is most likely stuck at the wrong value?

   A. MemtoReg stuck at 1.
   B. RegDst stuck at 1.
   C. ALUSrc stuck at 0.
   D. MemRead stuck at 1.
   E. None of options (A), (B), (C), (D) are correct.

7. Which of the following Boolean laws/theorems is/are used to simplify ***P·Q' + Q'·R' + P'·R'*** into ***P·Q' + P'·R'*** ?

   A. Distributive law
   B. Absorption theorem 1 and DeMorgan's theorem
   C. Complement law and idempotency theorem
   D. Consensus theorem
   E. None of options (A), (B), (C), (D) are correct.

8. Name the logic gate that is equivalent to the circuit below.



   A. NAND gate
   B. NOR gate
   C. XOR gate
   D. XNOR gate
   E. None of options (A), (B), (C), (D) are correct.

9. What is the minimum number of 2-input NAND gates required to implement an XOR gate?

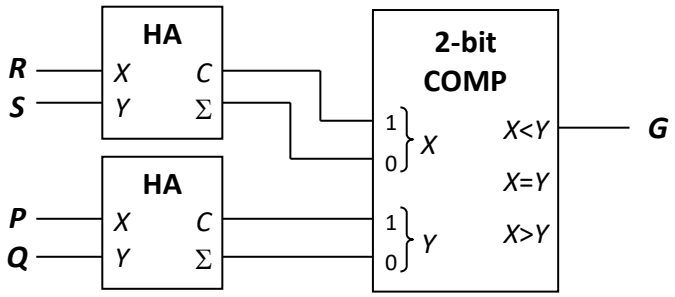
   A. 3
   B. 4
   C. 5
   D. 6
   E. None of options (A), (B), (C), (D) are correct.

10. How many <u>unique minterms</u> does the following 5-variable Boolean function contain?

$$F(A,B,C,D,E) = C'\cdot E' + C\cdot D\cdot E + B\cdot C'\cdot D + A\cdot B\cdot D$$

   A. 12
   B. 15
   C. 18
   D. 20
   E. None of options (A), (B), (C), (D) are correct.

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



A. Σ*m*(1, 2, 3, 6, 7, 11)
B. Σ*m*(4, 8, 9, 12, 13, 14)
C. Σ*m*(1, 2, 3, 7, 11)
D. Σ*m*(4, 8, 12, 13, 14)
E. None of options (A), (B), (C), (D) are correct.

12. Which of the following statements are true?

(i) Every maxterm is a product term.
(ii) A literal is both an SOP expression and a POS expression.
(iii) Every Boolean expression can be rewritten into POS form.

A. Only (i) and (ii).
B. Only (i) and (iii).
C. Only (ii) and (iii).
D. All of (i), (ii) and (iii).
E. None of options (A), (B), (C), (D) are correct.

13. Given the Boolean function: ***Z*(*A,B,C,D*) = Σ*m*(1,2,3,9,11,12,15) + Σ*x*(4,5,6,7,13,14)**

where *x*'s are don't-cares, and an 8-to-1 multiplexer with selector lines S_2 (most significant), S_1 and S_0. Assuming that logical constants are available but not complemented literals, which of the following settings cannot implement the function *Z* if no additional logic gates are allowed?

A. $S_2S_1S_0 = ABC$
B. $S_2S_1S_0 = ABD$
C. $S_2S_1S_0 = BCD$
D. $S_2S_1S_0 = CBA$
E. None of options (A), (B), (C), (D) are correct.

14. Given the Boolean function $F(A,B,C,D) = \Sigma m(0,3,6,9,11) + \Sigma x(2,4,5,7,13,15)$ where x denotes don't-care, how many prime implicants are there in the K-map of F?

    A. 3
    B. 4
    C. 5
    D. 6
    E. None of options (A), (B), (C), (D) are correct.

15. Given the Boolean function $F(A,B,C,D) = \Sigma m(0,3,6,9,11) + \Sigma x(2,4,5,7,13,15)$ where x denotes don't-care (same as the question above), how many essential prime implicants are there in the K-map of F?

    A. 3
    B. 4
    C. 5
    D. 6
    E. None of options (A), (B), (C), (D) are correct.

16. The 4221 code and 8421 code (or BCD code) introduced in the tutorial are two examples of decimal codes. Self-complementing decimal codes are codes such that given any decimal digits x and y which are 9's complements of each other, their corresponding codes are 1's complement of each other. (For example, 0010 and 1101 are the 4221 codes of decimal digits 2 and 7 respectively. The 4221 code is self-complementing but 8421 code is not.

    (Note that there are alternative 4221 codes for decimal digit 2, such as 0010 and 0100. As long as there is a way such that any two complementary decimal digits have their corresponding codes that are complementary in binary, it is considered a self-complementing code.)

    Which of the following codes are self-complementing?

    (i) 84-2-1 code (the weights are 8, 4, -2 and -1)
    (ii) 5211 code
    (iii) 4321 code

    A. Only (i).
    B. Only (ii).
    C. Only (i) and (ii).
    D. Only (ii) and (iii).
    E. None of options (A), (B), (C), (D) are correct.

17. Which of the following symbols is used for logic gate inputs in Logisim?

A. 

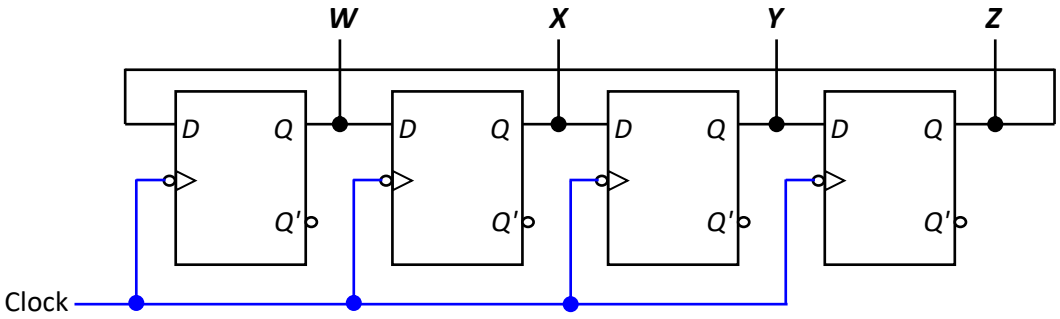
B. 

C. 

D. 

E. None of options (A), (B), (C), (D) are correct.

18. The sequential circuit below is implemented using D flip-flops.



Suppose the circuit is initialised with the value *WXYZ* = 0101, how many unique states does the circuit cycle through?

A. 2

B. 4

C. 8

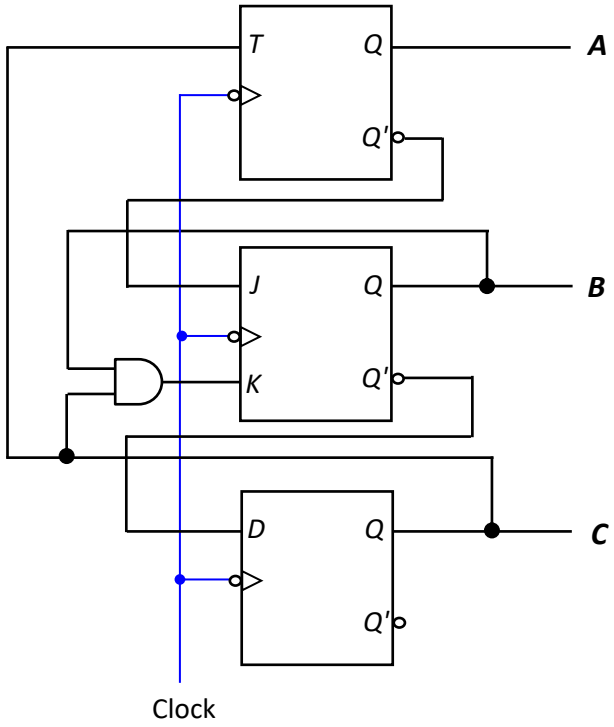
D. 16

E. None of options (A), (B), (C), (D) are correct.

Part B: There are 5 questions in this part [Total: 64 marks]

Q19. Sequential circuit [12 marks]

A sequential circuit with states *ABC* is implemented using a *T* flip-flop, a *JK* flip-flop and a *D* flip-flop as shown below.



(a) Complete the state diagram on the Answer Sheets. Two of the transition arrows have been drawn for you. [6]

(b) List out all the sink states in <u>decimal values</u> (that is, if *ABC*=101 is a sink state, write the value 5 instead of 101). If there are no sink states, write "none" instead of leaving a blank answer. [1]

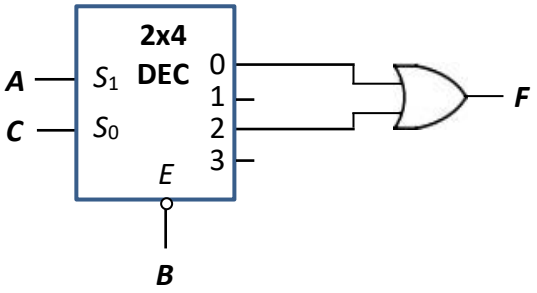
(c) Redesign the circuit using only ***T* flip-flops**. Write out the flip-flop input functions *TB* and *TC* in the simplest SOP expressions. [4]

(d) Assuming that part (c) is correctly answered, if *TB* and *TC* are to be implemented using the fewest number of logic gates, at least how many logic gates in total are required to implement *TB* and *TC*? Logic gates available are inverters, AND, OR, NAND, NOR, XOR and XNOR gates. [1]

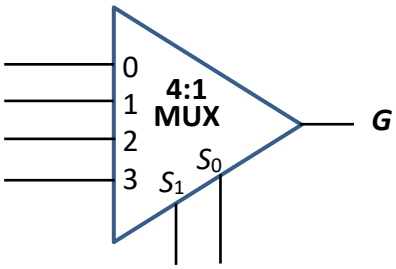
Q20. Combinational circuits [14 marks]

Note that logical constants 0 and 1 are available, but not complemented literals.

(a) Given a 0-enable active-high output 2×4 decoder as shown below, what is the simplified SOP expression for *F* ? [4]



(b) Let $G(A,B,C,D) = \Pi M(1,3,4,12)$. Implement *G* using a single 4:1 multiplexer without any additional logic gate. Complete the diagram on the Answer Sheets. [4]



(c) In class, we have discussed the design of the ALU Control Unit for the following set of instructions: lw, sw, beq, add, sub, and, or, slt. Suppose the funct values of all R-format instructions are changed. The new funct values (in base 10) for the 5 R-format instructions are given below:

R-format instruction	add	sub	and	or	slt
funct value in base 10	27	19	11	43	51

Write out the simplified SOP expressions for ALUcontrol2, ALUcontrol1 and ALUcontrol0.

(Use ALUop1 and ALUop0 for the 2-bit ALUop, with ALUop1 being the MSB. Use F5, F4, F3, F2, F1 and F0 for the 6-bit funct field, with F5 being the MSB and F0 the LSB.) [6]

Q21. MIPS [13 marks]

Study the following MIPS code on integer arrays *A* and *B*, with both arrays having the same number of elements. The variable mappings are given below:

- $a0 = *size* (number of elements in array *A*)
- $a1 = base address of array *A*
- $a2 = base address of array *B*

```
.data
size: .word 9
A: .word 23, 13, 16, 20, 100, 17, 82, 12, 80  # contents of array A
B: .word 11, 22, 33, 44, 55, 66, 77, 88,99    # contents of array B
.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    # Inst1
      addi $t1, $a1, 0    # Inst2
      addi $t2, $a2, 0    # Inst3

loop: slt  $t9, $t0, $a0  # Inst4
      beq  $t9, $0, exit  # Inst5
      lw   $s1, 0($t1)    # Inst6
      lw   $s2, 0($t2)    # Inst7
      andi $s3, $s1, 3    # Inst8
      bne  $s3, $0, skip  # Inst9
      addi $s2, $s2, -1   # Inst10
      sw   $s2, 0($t2)    # Inst11
skip: addi $t0, $t0, 2    # Inst12
      addi $t1, $t1, 8    # Inst13
      addi $t2, $t2, 8    # Inst14
      j    loop           # Inst15
# ------------------------------------------------------------------------------
exit: li   $v0, 10        # system call code for exit
      syscall
```

(a) Write out the contents of array *B* after the execution of the above MIPS code. [2]

(b) Using the variable names (*size*, *A*, *B*) given in the variable mappings, write an equivalent C code that corresponds to instructions Inst1 to Inst15, using a 'for' loop. You may use additional variable(s) if necessary. You do not need to declare the variables in your C code. Do not do a line-by-line direct translation of the MIPS code. Marks may be deducted if your code is convoluted or unnecessarily long. [3]

(c) With the given contents of arrays *A* and *B*, and considering only instructions Inst1 to Inst15, how many instructions are executed in total by the code? [2]

(d) Write the encoding of Inst6 (`lw $s1, 0($t1)`) in hexadecimal. [2]

(e) Write the encoding of Inst9 (`bne $s3, $0, skip`) in hexadecimal. [2]

(f) Assuming that Inst15 (`j loop`) is stored at address **0x0020 0000**, write the encoding of Inst15 (`j loop`) in hexadecimal. [2]

Q22. Pipelining [12 marks]

Study the MIPS code below, which is the same code in Q21, with only Inst1 to Inst15 shown. The variable mappings are given below:

- $a0 = *size* (number of elements in array *A*)
- $a1 = base address of array *A*
- $a2 = base address of array *B*

```
      add  $t0, $0, $0    # Inst1
      addi $t1, $a1, 0    # Inst2
      addi $t2, $a2, 0    # Inst3
loop: slt  $t9, $t0, $a0  # Inst4
      beq  $t9, $0, exit  # Inst5
      lw   $s1, 0($t1)    # Inst6
      lw   $s2, 0($t2)    # Inst7
      andi $s3, $s1, 3    # Inst8
      bne  $s3, $0, skip  # Inst9
      addi $s2, $s2, -1   # Inst10
      sw   $s2, 0($t2)    # Inst11
skip: addi $t0, $t0, 2    # Inst12
      addi $t1, $t1, 8    # Inst13
      addi $t2, $t2, 8    # Inst14
      j    loop           # Inst15
exit:
```

Assuming a 5-stage MIPS pipeline, you need to count until the last stage (WB stage) of Inst15. Assume that no delayed branching is used and the jump instruction (j) computes the target address to jump to in its ID stage (stage 2). Assume that the contents of arrays *A* and *B* are the same as in Q21; in particular, *A*[0] = 23.

(a) How many cycles does the code segment Inst1 to Inst15 take to complete its execution in the first iteration in an ideal pipeline, that is, one with no delays? [1 mark]

For parts (b) to (d) below, given the assumption for each part, how many <u>additional cycles</u> does the code segment Inst1 to Inst15 take to complete its execution in the first iteration as compared to an ideal pipeline computed in (a)? For example, if part (a) takes 30 cycles and part (b) takes 39 cycles, you should write **9** as the answer for part (b).

(b) Assuming <u>without forwarding and branch decision is made at MEM stage (stage 4)</u>.
No branch prediction is made. [3 marks]

(c) Assuming <u>with forwarding and branch decision is made at MEM stage (stage 4)</u>.
No branch prediction is made. [3 marks]

(d) Assuming <u>with forwarding and branch decision is made at ID stage (stage 2)</u>.
<u>Branch is predicted not taken</u>. [3 marks]

(e) Assume the configuration in part (b). Let's call the given code X. Suppose we swap Inst7 and Inst8 and call the new code Y. Would X be better than Y (X takes fewer cycles than Y to complete), or would X be worse than Y (X takes more cycles than Y to complete), or the same (X and Y take the same number of cycles to complete)? Write "Better", "Worse" or "Same" on the Answer Sheets. [2 marks]

Q23. Cache [13 marks]

Study the MIPS code below, which is the same code in Q21, with only Inst1 to Inst15 shown. The variable mappings are given below:

- $a0 = *size* (number of elements in array *A*)
- $a1 = base address of array *A*
- $a2 = base address of array *B*

```
      add  $t0, $0, $0     # Inst1
      addi $t1, $a1, 0     # Inst2
      addi $t2, $a2, 0     # Inst3
loop: slt  $t9, $t0, $a0   # Inst4
      beq  $t9, $0, exit   # Inst5
      lw   $s1, 0($t1)     # Inst6
      lw   $s2, 0($t2)     # Inst7
      andi $s3, $s1, 3     # Inst8
      bne  $s3, $0, skip   # Inst9
      addi $s2, $s2, -1    # Inst10
      sw   $s2, 0($t2)     # Inst11
skip: addi $t0, $t0, 2     # Inst12
      addi $t1, $t1, 8     # Inst13
      addi $t2, $t2, 8     # Inst14
      j    loop            # Inst15
exit:
```

In this question, we consider only memory read accesses (lw) and ignore memory write accesses (sw).

You are to assume that array *B* follows immediately after array *A* in the memory, that is, if the last element of array *A* is at address x, then *B*[0] is at address $x + 4$.

Parts (a) to (e) are on **data cache** with **each block containing 8 words**.

(a) How many bits are there in the byte offset field? [1 mark]

(b) Given a direct-mapped data cache with a capacity of 512 words, how many bits are there in the index field? [1 mark]

(c) Given a direct-mapped data cache with a capacity of 512 words, arrays *A* has **1028** elements, array *B* has the same number of elements, and the base address of array *A* is at 0x10010004, how many memory read access cache hits in total are there? [3 marks]

(d) Given a 2-way set-associative data cache with a capacity of 512 words, how many bits are there in the set index field? [1 mark]

(e) Given a 2-way set-associative data cache with a capacity of 512 words, arrays *A* has **1028** elements, array *B* has the same number of elements, and the base address of array *A* is at 0x10010004, how many memory read access cache hits in total are there? [3 marks]

Parts (f) and (g) on the next page…

Parts (f) and (g) are on a direct-mapped **instruction cache** with a capacity of 8 words with **each block containing 4 words**. Assume that Inst1 is at address **0x001F FFC4**.

(f) How many bits are there in the index field? [1 mark]

(g) Assuming that array *A* has **100** elements and array *B* has the same number of elements. All elements in array *A* have the value 99. How many cache hits in the instruction cache in total are there? [3 marks]

=== 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

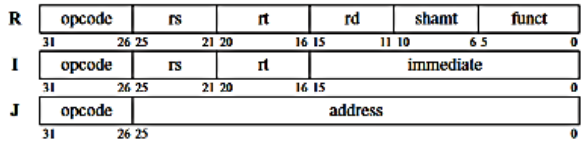
MIPS Reference Data ①

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)	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

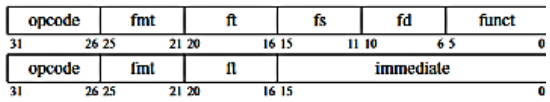
BASIC INSTRUCTION FORMATS



ARITHMETIC CORE INSTRUCTION SET ②

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	No