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.

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 <u>not</u> an EPI in the K-map of *F*?

   A. $C \cdot D$
   B. $B \cdot C$
   C. $B' \cdot D$
   D. $A \cdot B' \cdot C'$
   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 <u>cannot</u> 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.

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
   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.

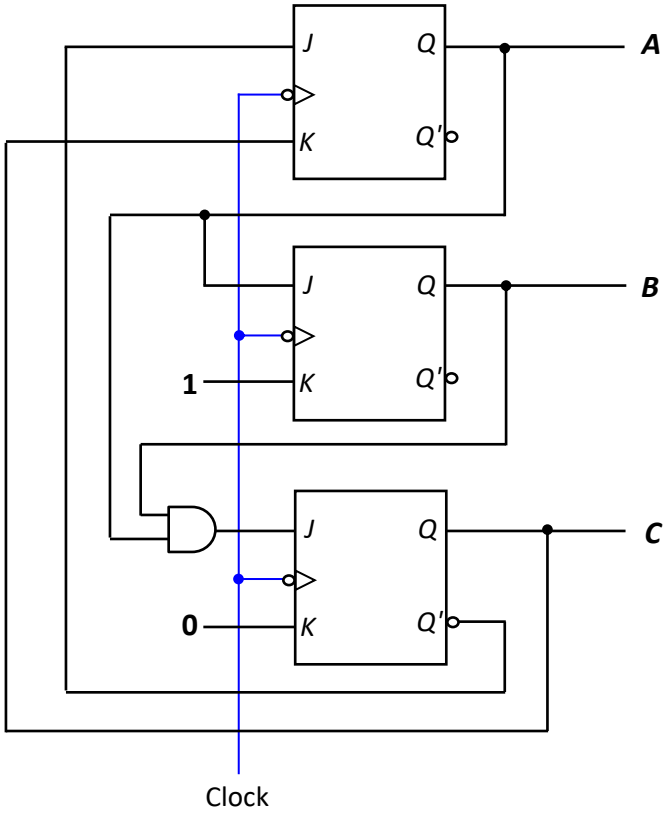
    Assume all bits are utilised and all instructions types exists, each ***address*** holds a **word** instead of a byte, the immediate field is in 2s complement, all addresses are used and all registers can be encoded. Which of the following statements are true about the instruction?

    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]



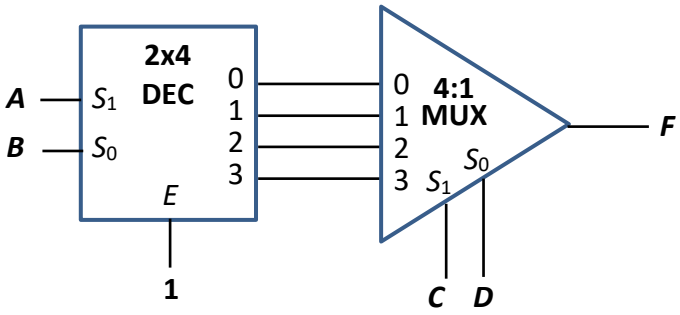
(b) Is the circuit self-correcting? Explain. (Mark will not be awarded if no explanation is given or the explanation given is incomplete/incorrect.) [1 mark]

(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]

Q14. Combinational circuits [16 marks]

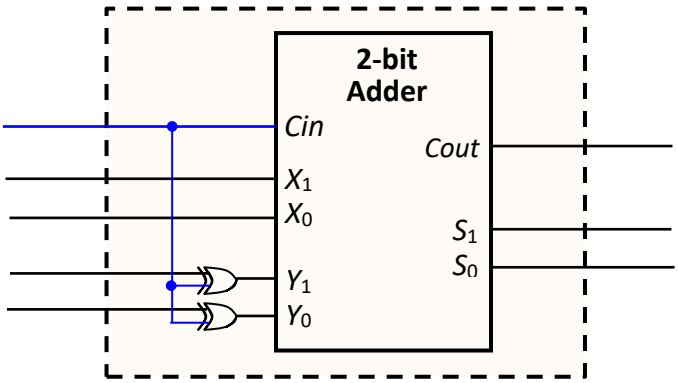
Note that logical constants 0 and 1 are available, but not complemented literals.

(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]

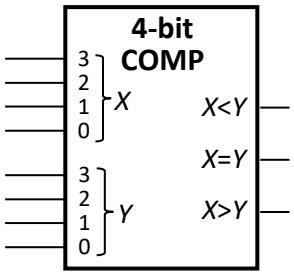


(c) [Total: 8 marks]
Given this Boolean function: $G(A,B,C,D) = \Sigma m(0, 4, 6, 8, 9, 10, 12, 13, 14, 15)$,

(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.

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

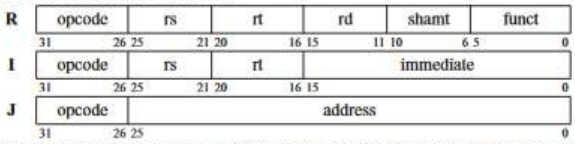
MIPS Reference Data ①

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'b'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

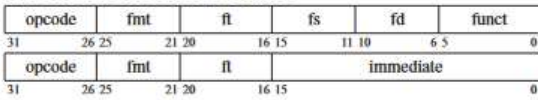
BASIC INSTRUCTION FORMATS



ARITHMETIC CORE INSTRUCTION SET ②

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