NATIONAL UNIVERSITY OF SINGAPORE

CS2100 – COMPUTER ORGANISATION

(Semester 2: AY2025/26)

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 SHEET**.
3. This is an **OPEN BOOK** assessment.
4. Write your answers only on the **ANSWER SHEET**. 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 – 17	34
Q18	5
Q19	10
Q20	12
Q21	14
Q22	12
Q23	13
Total	**100**

——— END OF INSTRUCTIONS ———

Part A: Multiple-Choice Questions [Total: 17×2=34 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>

int main() {
    int arr[] = {10, 20, 30, 40};
    int *p = arr;
    printf("%d ", *p++);
    printf("%d ", *p+2);
    printf("%d\n", *(p+2) - *(p+1));
    return 0;
}
```

   A. 10 22 10
   B. 10 20 10
   C. 20 22 10
   D. 10 22 -10
   E. None of options (A), (B), (C), (D) are correct.

2. Two processors implement the same ISA but have different microarchitectures. Which of the following statements must be true?

   (i) Both processors must produce identical outputs for the same program (assuming no undefined behavior).
   (ii) Both processors must execute the same number of clock cycles for each instruction.
   (iii) Both processors must use the same control signals internally.
   (iv) Both processors must have the same instruction encoding in machine code.

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

3. Consider the following C statement:

```
*(p + i) = *(q + i + 1) + 3;
```

Assume the following:

- p and q are pointer variables that point to integer values
- p is mapped to register $s0
- q is mapped to register $s1
- i is an integer variable (int i) and is mapped to register $s2
- Each integer occupies 4 bytes in memory

Which of the following MIPS code sequences correctly implements the C statement?

A.
```
sll  $t0, $s2, 2
add  $t1, $s1, $t0
addi $t1, $t1, 4
lw   $t2, 0($t1)
addi $t2, $t2, 3
add  $t3, $s0, $t0
sw   $t2, 0($t3)
```

B.
```
sll  $t0, $s2, 2
add  $t1, $s1, $t0
lw   $t2, 0($t1)
addi $t2, $t2, 3
add  $t3, $s0, $t0
sw   $t2, 0($t3)
```

C.
```
sll  $t0, $s2, 2
add  $t1, $s1, $t0
lw   $t2, 4($t1)
add  $t2, $t2, $s2
add  $t3, $s0, $t0
sw   $t2, 0($t3)
```

D.
```
add  $t1, $s1, $s2
lw   $t2, 4($t1)
addi $t2, $t2, 3
add  $t3, $s0, $s2
sw   $t2, 0($t3)
```

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

4. Which of the following instructions correctly loads 0x00001234 into $s2?

(i) `lui $s2, 0x1234`

(ii) `ori $s2, 0x1234`

(iii) `addi $s2, $zero, 0x1234`

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.

5. For the instruction `lui $t0, 0x1234`, which of the following statements is correct?

   A. MemToReg must be 1.
   B. ALU performs addition.
   C. The immediate is shifted left by 16 bits before writing.
   D. RegWrite can be don't-care.
   E. None of options (A), (B), (C), (D) are correct.

6. A student mistakenly sets the control signal MemToReg = 1 for **all instructions**, regardless of instruction type. Assume the following:

   - Data memory output is undefined unless MemRead = 1
   - Other control signals are generated correctly
   - No hardware is damaged

   Which of the following statements is TRUE?

   A. Only lw instructions work correctly; all other instructions do not work correctly.
   B. All instructions that write to registers (except lw) produce incorrect results; instructions that do not write to registers are unaffected.
   C. Only R-type instructions (add, sub, etc.) produce incorrect results; all other instructions execute correctly.
   D. lw and R-type instructions write correct values; only store instructions (sw) fail.
   E. None of options (A), (B), (C), (D) are correct.

7. Which of the following statements are **TRUE**?

   (i) Both sign-and-magnitude and 2's complement have two distinct bit-patterns that represent the value zero.
   (ii) Negating an integer in 1's complement is performed using a simple bitwise NOT, whereas negating an integer in 2's complement requires a bitwise NOT followed by adding one.
   (iii) In 1's complement and 2's complement, adding a positive integer and a negative integer can never result in an overflow.
   (iv) In n-bit Excess-2^{n-1} representation, the value zero is represented by a bit pattern where the MSB is 1 and all other bits are zeros.

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

8. Given that a decimal value x is represented as **0x41@80000** in the IEEE 754 single-precision hexadecimal representation, where @ is some hexadecimal digit that is smaller than the value 8, what are the smallest decimal value and the largest decimal value that x can take?

   A. Smallest = 0.5, largest = 7.5
   B. Smallest = 8, largest = 15
   C. Smallest = 8.5, largest = 15.5
   D. Smallest = 8.25, largest = 15.75
   E. None of options (A), (B), (C), (D) are correct.

9. C provides the function **int abs(int x)**, declared in <stdlib.h>, that returns the absolute value of x. Given the following C program, the first line of output is **`n = DEADDEAD`**. What is the second line of output? The format specifier %X is used to print the value in hexadecimal. Integers are represented in two's complement system.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int n = 0xDEADDEAD;
    printf("n = %X\n", n);
    printf("abs(n) = %X\n", abs(n));
    return 0;
}
```

   A. **`abs(n) = DEADDEAD`**
   B. **`abs(n) = DAEDDAED`**
   C. **`abs(n) = 21522152`**
   D. **`abs(n) = 21522153`**
   E. None of options (A), (B), (C), (D) are correct.

10. Given the Boolean function $F(A,B,C,D) = \Sigma m(2,4,7,8,10,14) + \Sigma x(0,6,11,13,15)$ where x denotes don't-care, how many PIs and EPIs are there in the K-map of F?

    A. 6 PIs; 3 EPIs
    B. 5 PIs, 3 EPIs
    C. 5 PIs, 2 EPIs
    D. 4 PIs, 2 EPIs
    E. None of options (A), (B), (C), (D) are correct.

11. Given the Boolean function $F(A,B,C,D) = \Sigma m(2,4,7,8,10,14) + \Sigma x(0,6,11,13,15)$ where x denotes don't-care (same as the question above), what is the simplified POS expression for F?

    A. $(C+D') \cdot (B+D') \cdot (A'+B'+C)$
    B. $(C'+D) \cdot (B'+D) \cdot (A+B+C')$
    C. $(B+D) \cdot (B'+C') \cdot (A+D)$
    D. $(B'+D') \cdot (B+C) \cdot (A'+D')$
    E. None of options (A), (B), (C), (D) are correct.

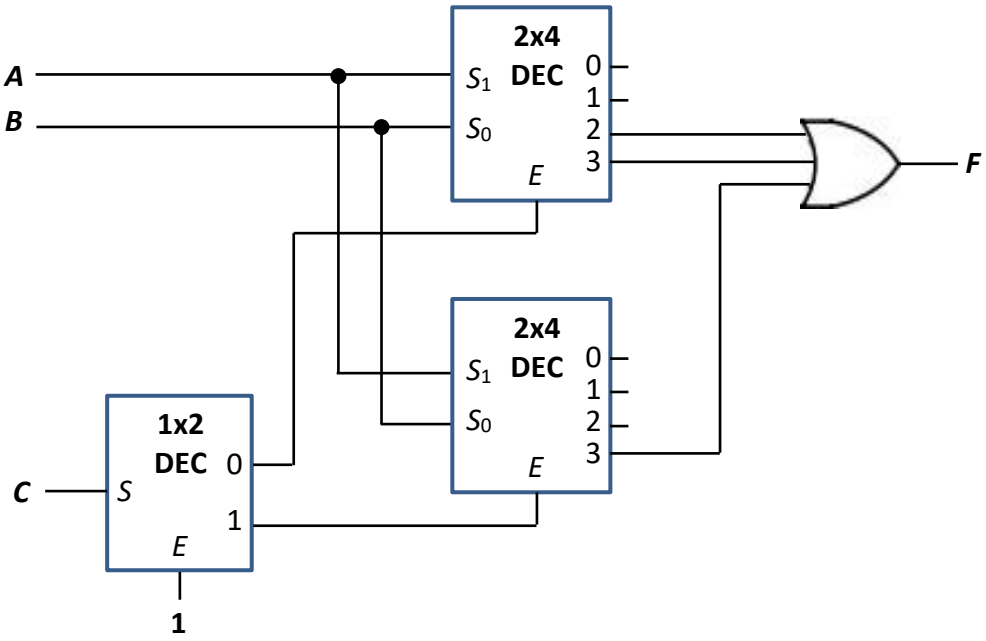
12. A 4-to-1 multiplexer has inputs I_0, I_1, I_2, I_3 and select lines S_1, S_0. Given the following inputs:

$$I_0 = 1, \quad I_1 = 0, \quad I_2 = 0, \quad I_3 = 1,$$

which Boolean function F does the multiplexer implement?

A. $F = S_1 + S_0$
B. $F = (S_1 + S_0)'$
C. $F = S_1 \oplus S_0$
D. $F = 1$
E. None of options (A), (B), (C), (D) are correct.

13. Given the following circuit using 1-enabled decoders with active high outputs, what is the function *F*(*A*,*B*,*C*)?



A. $B \cdot (A' + C)$
B. $A \cdot (B + C')$
C. $A \cdot (B + C)$
D. $C \cdot (A' + B')$
E. None of options (A), (B), (C), (D) are correct.

14. Consider the following statements on Boolean algebra:

    (i) If $A + B = A$, then $B = 0$.

    (ii) If $A + B = A + C$, then $B = C$.

    (iii) If $A + B = 1$, then $A' \cdot B' = 0$.

    Which of the above statements are true?

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

15. In a 5-stage MIPS pipeline, an instruction cache miss occurs during the IF stage. What is the effect?

    A. The whole program terminates.
    B. The instruction is fetched from main memory while the pipeline continues executing subsequent instructions.
    C. The entire pipeline stalls until the instruction is fetched.
    D. The instruction causing the cache miss is skipped and the next instruction is fetched.
    E. None of options (A), (B), (C), (D) are correct.

16. Which forwarding path is used for the following situation?

```
add $t0, $t1, $t2
sub $t3, $t0, $t4
```

    A. ID → EX
    B. EX → WB
    C. EX → MEM
    D. EX → EX
    E. None of options (A), (B), (C), (D) are correct.

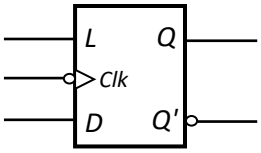
17. A processor has a cache hit time of 1 cycle and a miss rate of 4%. If there is a cache miss, it takes 50 cycles to bring the data from the main memory to the cache. What is the average memory access time?

    A. 2.6 cycles
    B. 2.96 cycles
    C. 3 cycles
    D. 3.2 cycles
    E. None of options (A), (B), (C), (D) are correct.

Part B: There are 5 questions in this part [Total: 66 marks]

Q18. Sequential circuit [5 marks]

In assignment 3 we introduced the *LD* flip-flop (as shown below) consisting of two inputs *L* and *D*: when *L*=0 (load command), the next state Q^+ is set to *D*; when *L*=1 (toggle command), Q^+ is set to *Q'*.



(a) Write out the simplified SOP expression for $Q^+(L,D,Q)$. [1]

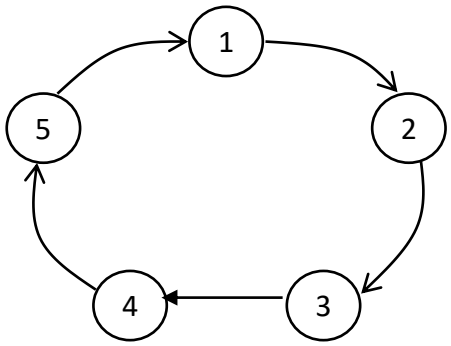
(b) A sequential circuit with states *AB* is implemented using two *LD* flip-flops with the following flip-flop input functions:

$$LA = B'; \quad DA = A'; \qquad LB = A; \quad DB = B$$

Write out the transition of the states, that is, the next state of each state in the circuit. State numbers are to be written in decimal. [4]

Q19. Sequential circuit [10 marks]

The state diagram on the right shows a sequential circuit with five states: states 1 ($ABC = 001_2$) through state 5 ($ABC = 101_2$). States 0, 6 and 7 are unused. Design this circuit using two *JK* flip-flops for *A* and *B*, and a *T* flip-flop for *C*.



(a) Write out the simplified SOP expressions for the flip-flop inputs. [5]

(b) What is the minimum number of additional logic gates required to implement this sequential circuit? [1]

(c) Write out the next state of each of the unused states. State numbers are to be written in decimal. [3]

(d) Is the circuit self-correcting? Answer "Yes" or "No". [1]

Q20. Combinational circuits [12 marks]

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

(a) Let $F(A, B, C, D) = \Pi M(0,6,7,8)$. Implement *F* using a single 4:1 multiplexer (as shown in Figure 20(a)) without any additional logic gate. Fill in the inputs for $I_0I_1I_2I_3$ and S_1S_0. [4]

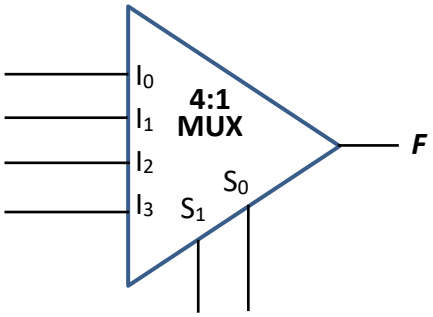


Figure 20(a)

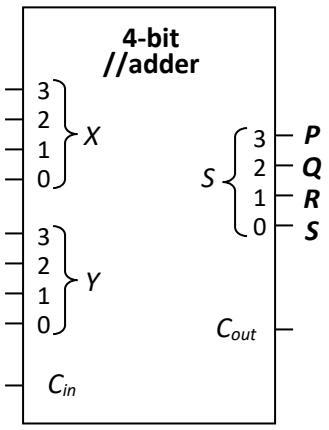


Figure 20(b)

(b) The following table shows the 8,4,-2,-1 code and 8421 (BCD) code for the decimal digits:

	0	1	2	3	4	5	6	7	8	9
8,4,-2,-1 code	0000	0111	0110	0101	0100	1011	1010	1001	1000	1111
8421 code	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

You are to implement an **8,4,-2,-1 to 8421 code converter** to convert input *ABCD* in 8,4,-2,-1 code to output *PQRS* in 8421 code using a single 4-bit parallel adder (as shown in Figure 20(b)) and the fewest number of logic gates. Fill in the inputs for $X_3X_2X_1X_0$, $Y_3Y_2Y_1Y_0$ and C_{in}. The output *PQRS* has been filled for you and you are not to change it. It is understood that if an input is not an uncomplemented literal, then some logic gates are required – for example, *A'* would require an inverter, *B*+*C'* would require an OR gate and an inverter. [4]

(c) Given a 4-variable Boolean function $G(A, B, C, D) = \Sigma m(1,3,6,9,14) + \Sigma x(2,5,7,11,15)$ where x's are don't-cares, answer the following parts:

(i) Write out the simplified POS expression for *G*. [2]

(ii) Static-1 and static-0 hazards were introduced in the assignment. The simplified POS expression for G has static-0 hazards. Add a sum term to the simplified POS expression for *G* to eliminate static-0 hazards. You need to write only this sum term in your answer. [2]

Q21. MIPS [14 marks]

Study the following MIPS code on integer arrays *A* and *B*, both arrays having the same number of elements. The code computes some answers in $s6 and $s7. 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*
- $s6 = *answer1*
- $s7 = *answer2*

You may assume that the **`sll`** instruction does not result in overflow.

```
.data
size: .word 4
A: .word 9, -5, 17, 12      # contents of array A
B: .word 1,  4, -6,  5      # 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: $t0 = loop index
       add  $t1, $a1, $0    # Inst2: $t1 = &A[0]
       add  $t2, $a2, $0    # Inst3: $t2 = &B[0]
       add  $s6,  $0, $0    # Inst4: Initialize answer1 to 0
       add  $s7,  $0, $0    # Inst5: Initialize answer2 to 0

loop:  slt  $t9, $t0, $a0   # Inst6
       beq  $t9,  $0, exit  # Inst7
       lw   $s1, 0($t1)     # Inst8
       lw   $s2, 0($t2)     # Inst9
       sll  $s3, $s2, 2     # Inst10: sll does not cause overflow
       add  $s5, $s1, $s3   # Inst11
       add  $s6, $s6, $s5   # Inst12
       sub  $t7, $s1, $s2   # Inst13
       slt  $t8, $t7, $0    # Inst14
       beq  $t8,  $0, skip1 # Inst15
       sub  $t7,  $0, $t7   # Inst16

skip1: slt  $t8, $t7, $s7   # Inst17
       bne  $t8,  $0, skip2 # Inst18
       add  $s7, $t7, $0    # Inst19

skip2: addi $t0, $t0, 1     # Inst20
       addi $t1, $t1, 4     # Inst21
       addi $t2, $t2, 4     # Inst22
       j    loop            # Inst23
# ------------------------------------------------------------------------------
exit:  li   $v0, 10         # system call code for exit
       syscall
```

Q21. (continue…)

(a) Write out the decimal value of $s6 (*answer1*) after the execution of the above MIPS code. [1]

(b) Write out the decimal value of $s7 (*answer2*) after the execution of the above MIPS code. [1]

(c) Using the variable names (*size*, *A*, *B*, *answer1*, *answer2*) given in the variable mappings, write an equivalent C code that corresponds to instructions Inst1 to Inst23, 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. Complete the code fragment in the Answer Sheet, without changing the parts that are given. You may use suitable math function if necessary. [4]

(d) With the given contents of arrays *A* and *B*, and considering only instructions Inst1 to Inst23, how many instructions are executed in total? [2]

(e) Write the encoding of Inst7 (**`beq $t9, $0, exit`**) in hexadecimal. [2]

(f) Write the encoding of Inst13 (**`sub $t7, $s1, $s2`**) in hexadecimal. [2]

(g) Assuming that Inst1 (**`add $t0, $0, $0`**) is stored at address **0x0040 003C**, write the encoding of Inst23 (**`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 Inst23 shown.

```
        add  $t0,  $0, $0    # Inst1: $t0 = loop index
        add  $t1, $a1, $0    # Inst2: $t1 = &A[0]
        add  $t2, $a2, $0    # Inst3: $t2 = &B[0]
        add  $s6,  $0, $0    # Inst4: Initialize answer1 to 0
        add  $s7,  $0, $0    # Inst5: Initialize answer2 to 0

loop:   slt  $t9, $t0, $a0   # Inst6
        beq  $t9,  $0, exit  # Inst7
        lw   $s1, 0($t1)     # Inst8
        lw   $s2, 0($t2)     # Inst9
        sll  $s3, $s2, 2     # Inst10: sll does not cause overflow
        add  $s5, $s1, $s3   # Inst11
        add  $s6, $s6, $s5   # Inst12
        sub  $t7, $s1, $s2   # Inst13
        slt  $t8, $t7, $0    # Inst14
        beq  $t8,  $0, skip1 # Inst15
        sub  $t7,  $0, $t7   # Inst16

skip1:  slt  $t8, $t7, $s7   # Inst17
        bne  $t8,  $0, skip2 # Inst18
        add  $s7, $t7, $0    # Inst19

skip2:  addi $t0, $t0, 1     # Inst20
        addi $t1, $t1, 4     # Inst21
        addi $t2, $t2, 4     # Inst22
        j    loop            # Inst23
```

Assuming a 5-stage MIPS pipeline, you need to count until the last stage (WB stage) of Inst23. 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 as given in Q21.

(a) How many cycles does the code segment Inst1 to Inst23 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 additional cycles does the code segment Inst1 to Inst23 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 (not 39) for part (b).

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

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

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

(e) Assume the configuration in part (d). Suppose we swap the two **lw** instructions. Would this help to reduce the answer for part (d)? If so, write down the instruction number of the instruction that has its delay reduced and the resulting number of delay cycles for that instruction after the reduction.(For example, if the instruction is Inst6 and the resulting number of delay cycles is 2 after reduction, write “Inst6; 2”.) If there is no such instruction, write “none”. [2 marks]

Q23. Cache [13 marks]

Study the MIPS code below, which is the same code in Q21, with only Inst1 to Inst23 shown.

```
        add  $t0,  $0, $0     # Inst1: $t0 = loop index
        add  $t1, $a1, $0     # Inst2: $t1 = &A[0]
        add  $t2, $a2, $0     # Inst3: $t2 = &B[0]
        add  $s6,  $0, $0     # Inst4: Initialize answer1 to 0
        add  $s7,  $0, $0     # Inst5: Initialize answer2 to 0

loop:   slt  $t9, $t0, $a0    # Inst6
        beq  $t9,  $0, exit   # Inst7
        lw   $s1, 0($t1)      # Inst8
        lw   $s2, 0($t2)      # Inst9
        sll  $s3, $s2, 2      # Inst10: sll does not cause overflow
        add  $s5, $s1, $s3    # Inst11
        add  $s6, $s6, $s5    # Inst12
        sub  $t7, $s1, $s2    # Inst13
        slt  $t8, $t7, $0     # Inst14
        beq  $t8,  $0, skip1  # Inst15
        sub  $t7,  $0, $t7    # Inst16

skip1:  slt  $t8, $t7, $s7    # Inst17
        bne  $t8,  $0, skip2  # Inst18
        add  $s7, $t7, $0     # Inst19

skip2:  addi $t0, $t0, 1      # Inst20
        addi $t1, $t1, 4      # Inst21
        addi $t2, $t2, 4      # Inst22
        j    loop             # Inst23
```

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 $x + 4$.

Assume that array *A* has **1026** elements with its base address at **0x80082108**, and array *B* has the same number of elements.

Parts (a) to (e) are on **data cache** with **each block containing 4 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, 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, 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 16 words with **each block containing 4 words**. Assume that Inst1 is at address **0x0040 003C.**

You are to assume that all instructions Inst6 to Inst23 are executed in every iteration (that is, the contents of the arrays are no longer the ones given in Q21).

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

(g) Assuming that array *A* has **1026** elements and array *B* has the same number of elements. 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

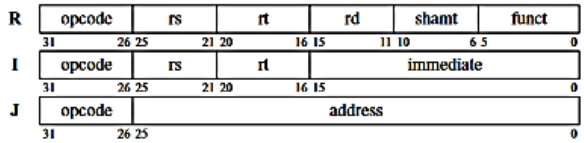
MIPS Reference Data ①

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

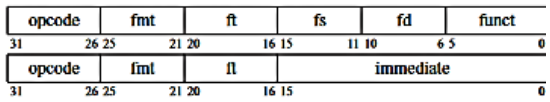
BASIC INSTRUCTION FORMATS



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