Examination 2

Name: **CS 2318, Spring 2026** **Grade:**

Instructions/directions/notes:

(1) You should attempt to answer all questions.
(2) Closed book and closed notes.
(**NOTE**: Selected reference material is provided on the **last 2 pages** of this exam paper.)
(3) When coding, use only the instructions/pseudoinstructions listed in the reference provided, subject to any additional restrictions explicitly indicated.
(4) No other equipment besides writing equipment such as pencils and eraser is allowed.
(5) 15 questions, 80 total points.
(6) Time allowed (in class): 80 minutes.
(7) If you need more writing space for a question, use the blank page opposite the page the question is on.

Binary <-> Hex conversion table in case it's helpful

Binary	Hex	Binary	Hex
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

1. *(3 points)* For each of the *bitwise* operations listed under ***operation*** column, fill the corresponding blank under ***result*** column with the result of the operation.
(TIP: As the 6 dashed boxes suggests, each operation should give a *6-bit* result.)
(NOTE: The 2 operands involved in each operation have bit patterns that are *identical*.)

operation	***result***
"bitwise-**AND**"ing `101010` with `101010`	
"bitwise-**OR**"ing `101010` with `101010`	
"bitwise-**XOR**"ing `101010` with `101010`	

2. *(6 points)* You have an 8-bit *unsigned whole number* and you are to apply a *bitwise operation* to the number with a suitable *mask* to obtain a *result* that you can use to test if the number is *>=32* or *not*. On the groups below, check the appropriate boxes for the *operation*, *mask* and *test* you can use. ***(Check only one per group.)*** (TIPS: $32 = 2^5$. First scan the *Test* group to see the kinds of test on tap.)

Operation	*Mask*	*Test*
☐ and	☐ 00011111	☐ *>=32* if result is 0, *<32* if result is non-zero
☐ not	☐ 11011111	☐ *>=32* if result is non-zero, *<32* if result is 0
☐ or	☐ 11000000	☐ *>=32* if result is 1, *<32* if result is not 1
☐ xor	☐ 11100000	☐ *>=32* if result is not 1, *<32* if result is 1
	☐ 00100000	
	☐ 00111111	

3. *(24 points)* Indicate which item on the right best matches each of those on the left by filling each of the blanks on the right with the corresponding number. (Each blank should have a different number.)

"fast-but-few" kind of storage ①	___ *load instruction*
"slow-but-plenty" kind of storage ②	___ *store instruction*
"brain" ③	___ *memory (RAM)*
"brawn" ④	___ *registers*
have both instructions and data held/accessed in RAM ⑤	___ *addressing mode*
internal connection for CPU-RAM communication ⑥	___ *control unit*
separate memory used for data and instructions ⑦	___ *datapath*
single memory used for both data and instructions ⑧	___ *bus*
way of specifying location(s) of operand(s) ⑨	___ *stored program concept*
register-to-memory data transfer Ⓐ	___ *Harvard architecture*
memory-to-register data transfer Ⓑ	___ *load-store architecture*
only load and store instructions access memory Ⓒ	___ *von Neumann/Princeton architecture*

4. (*3 points*) Give the correct order for the 3 key phases involved when a processor carries out an instruction by filling in the blanks with 1 (= *first* in order) through 3 (= *last* in order).

___ Execute instruction ___ Fetch instruction ___ Decode instruction

5. *(4 points)* At which of the following addresses may a *4-byte* (32-bit) data item be stored for it to be *aligned* in memory? (***Check <u>all</u> that apply.***) (Assume memory is *byte-addressable*.)

   ☐ 10101100 (binary) ☐ 86 (decimal) ☐ 92 (decimal) ☐ 7A (hex)

6. (6) Indicate which item on the right best matches which item on the left by connecting the corresponding dots with a line. (HINT: Each dot gets connected exactly once.)

global data ●	● *stack* segment
program *code* ●	● *data* segment (static part)
local data ●	● *text* segment

7. (*3 points*) Clearly circle **T**(rue) or **F**(alse) **[T F]**

   The following lines of code (seen appearing in a MIPS32 assembly language program)

```
            .data
iArr:       .word 4
```

   allocates 4 bytes of memory (for a global variable labeled `iArr`) that is used to store an integer.

8. *(10 points)* Make **5** corrections to MIPS code below so that it is free of bugs and can be assembled and work as intended, which is to divide **789** by **10** and report the resulting truncated *quotient*.
   <u>TIP</u>: **`789/10 has a truncated quotient of 78.`** is what output should be as intended.

```
            .data
outputLabel: .ascii  "789/10 has a truncated quotient of "
period:     .byte   '.'

            .text
            .globl  main
main:
            li $t1, 789
            li $v1, 10
            li $v0, 4
            lw $a0, outputLabel
            syscall
            div $t1, $v1
            li $v0, 1
            mfhi $a0
            syscall
            li $v0, 11
            move $t0, period
            sb $a0, 0($t0)
            syscall

            li $v0, 10
            syscall
```

   <u>Note</u>: These intend to display the character `'.'` in memory labeled **`period`**.

9. *(2 points)* **[T F]** MIPS assembly code that has • `<some'thing>` in it (like the first 5 lines in the code segment of ***Question 8***) will not translate into some machine instructions.

10. *(2 points)* **[T F]** A *pseudoinstruction* does not directly correspond to machine code but is instead translated by an assembler into one or more hardware-level instructions (each of assembler translated hardware-level instructions does correspond to machine code).

11. *(3 points)* Which of the following **`syscall`** input services returns result in *RAM*? (***Check only one.***)

- ☐ service # 5
- ☐ service # 8
- ☐ service # 12
- ☐ none of the above (all **`syscall`** input services return result in *register*)

12. *(3 points)* Which of the following best describes what effect **`andi $t0, $t0, 0xDF`** will have on the (ASCII) alphabet originally contained in **`$t0`**? (***Check only one.***)

☐ force lowercase ☐ force upercase ☐ toggle the case ☐ **bold** it

13. *(3 points)* Which of the following RISC design principles best explains why the MIPS general purpose register **`$0`** is hardwired to **`0`**? (***Check only one.***)

- ☐ simplicity favors regularity
- ☐ smaller is faster
- ☐ make the common case fast
- ☐ good design demands compromise

14. *(3 points)* (***Check only one.***) Which of the following does not apply to MIPS studied in this class?

- ☐ There are 32 general-purpose registers.
- ☐ Each and every general-purpose register is 32-bit wide.
- ☐ **`lo`** and **`hi`** refer to two of the 32 general-purpose registers.
- ☐ Four bytes (32 bits) is required to store a word.
- ☐ Four bytes (32 bits) is required to store an integer.
- ☐ Four bytes (32 bits) is used to represent an address.

15. *(5 points)* (***Check only one.***) Consider the following partial MIPS assembly language program:

```
        .data
intArr: .word 12, 23, 34, 45, 56, 67, 78

        .text
        .globl main
main:
        < some instructions here >   # to swap (in memory) contents of 3rd and 6th
                                     # elements (i.e., intArr[2] & intArr[5])
        < more instructions here >   # not relevant to this exercise
        li $v0, 10
        syscall
```

Which of the following will do the job (as commented) of **`< some instructions here >`** ?

☐

```
la $a0, intArr
lw $t3, 2($a0)
lw $t6, 5($a0)
sw $t3, 5($a0)
sw $t6, 2($a0)
```

☐

```
la $a0, intArr
sw $t3, 2($a0)
sw $t6, 5($a0)
lw $t6, 2($a0)
lw $t3, 5($a0)
```

☐

```
la $a0, intArr
lw $t3, 8($a0)
lw $t6, 20($a0)
sw $t3, 20($a0)
sw $t6, 8($a0)
```

☐

```
la $a0, intArr
lw $t3, 8($a0)
lw $t6, 20($a0)
move $a0, $t3
move $t3, $t6
move $t6, $a0
```