Name _____

Staple This Side

Computer Architecture
LSU EE 4720
Midterm Examination

Wednesday, 29 March 2023 9:30-10:20 CDT

Problem 1	_____	(17 pts)
Problem 2	_____	(20 pts)
Problem 3	_____	(16 pts)
Problem 4	_____	(16 pts)
Problem 5	_____	(16 pts)
Problem 6	_____	(15 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Staple This Side

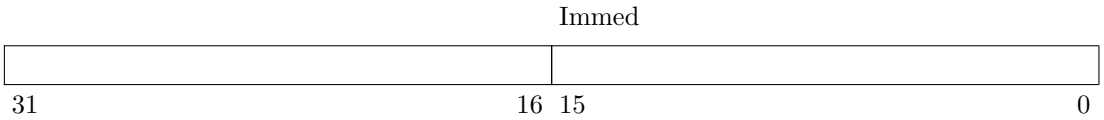
Problem 1: [17 pts] Candidate MIPS instruction `subir r1, 22, r3` is to compute `r1=22-r3`, which can't be done with a single existing MIPS instruction. The `22` is taken from instruction bits 15:0, which is the immediate field of Type-I instructions.

The `subir` instruction is to be encoded so that it can be executed by the implementation to the right **with the ALU computing** $X = A - B$, the same operation used by existing subtract instructions. Notice that in the implementation the **immediate connects to both** ALU inputs.

(*a*) Show how `subir r1, 22, r3` instruction would be encoded for this hardware.

☐ Show encoding of `subir r1, 22, r3`. Be sure to show ☐ the position of the fields and ☐ the field values for the sample instruction.

☐ Be sure that the encoding fits with the illustrated hardware and other MIPS instructions.



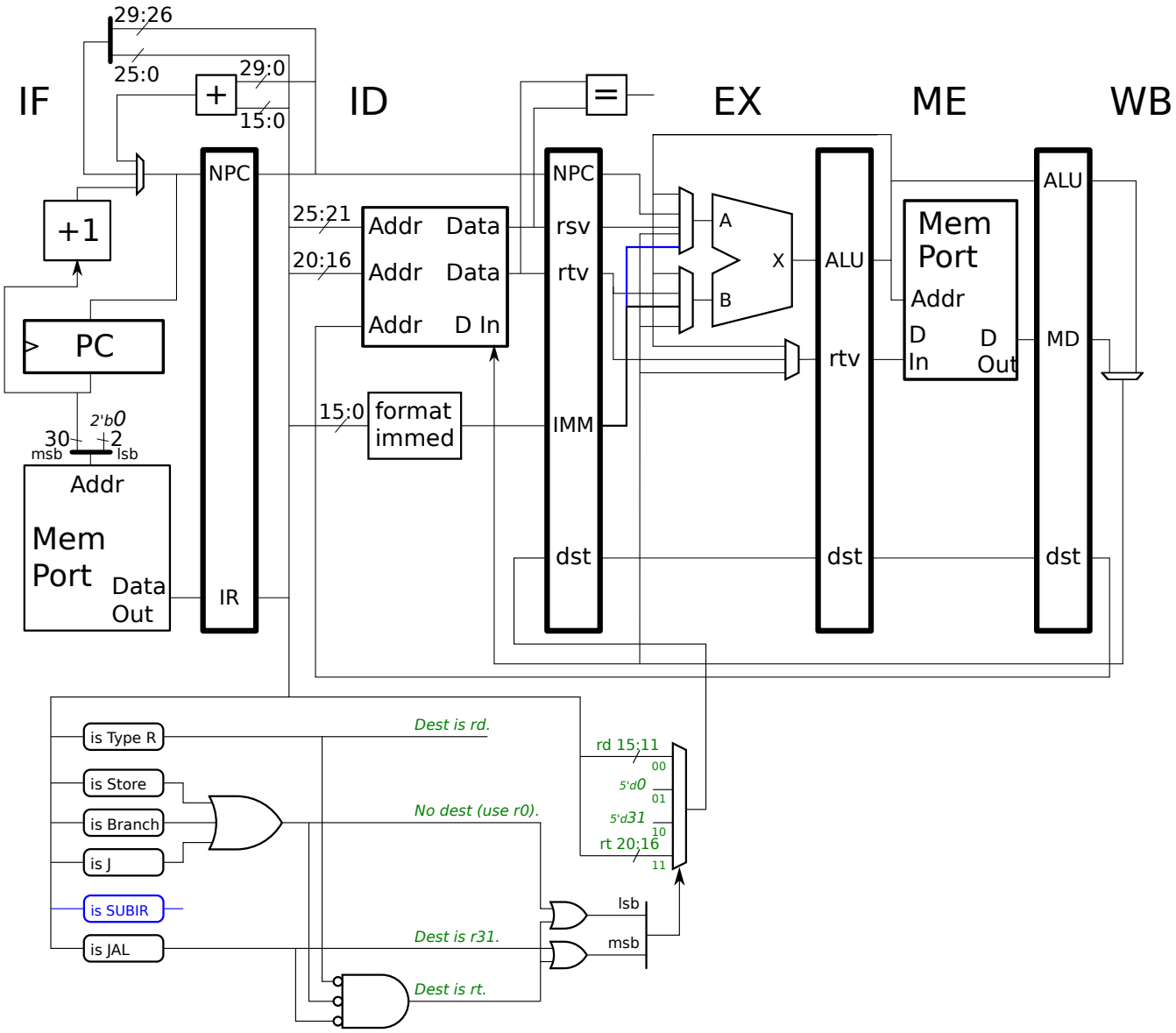
(*b*) Some control logic is shown for the implementation.

☐ Modify the control logic that computes `dst` so that `subir` executes correctly. ☐ **Do not** design control logic for the ALU multiplexors.

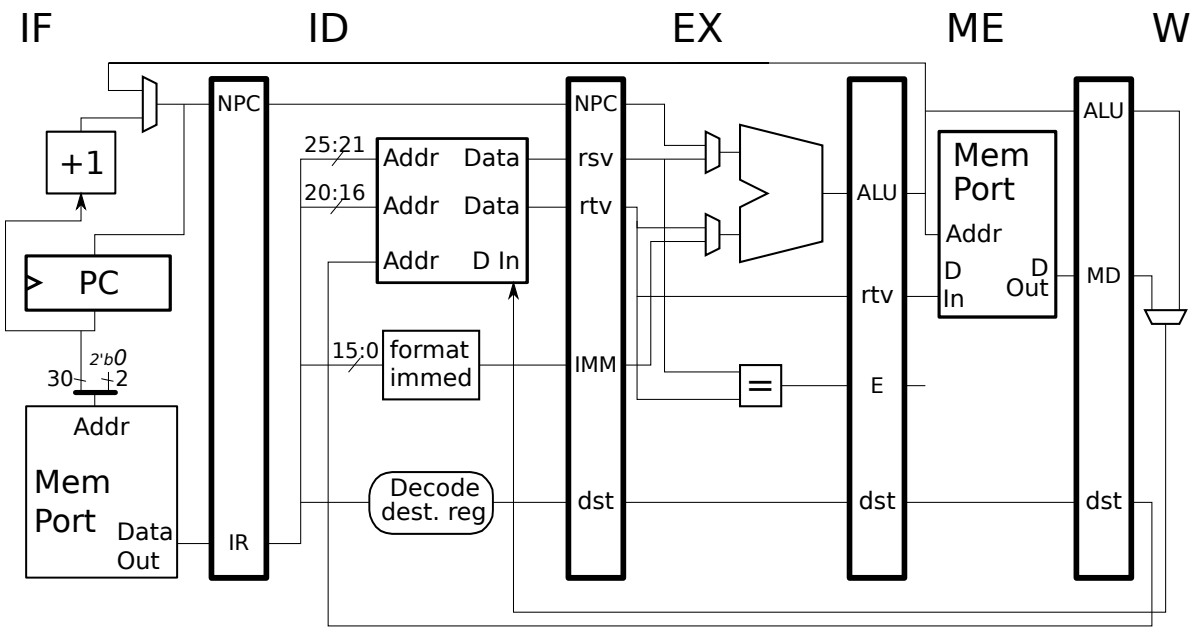
☐ The control logic should not break existing instructions.

☐ The control logic changes should be consistent with your answer to the previous part.

Staple This Side



Problem 2: [20 pts] Show the execution of the code fragments below on their accompanying MIPS implementations.



☐ Show the execution of the code fragment below on ☐ the implementation above. ☐ Be sure to check for dependencies.

```
addi r1, r2, 4

lw   r3, 0(r1)

sw   r1, 4(r3)
```

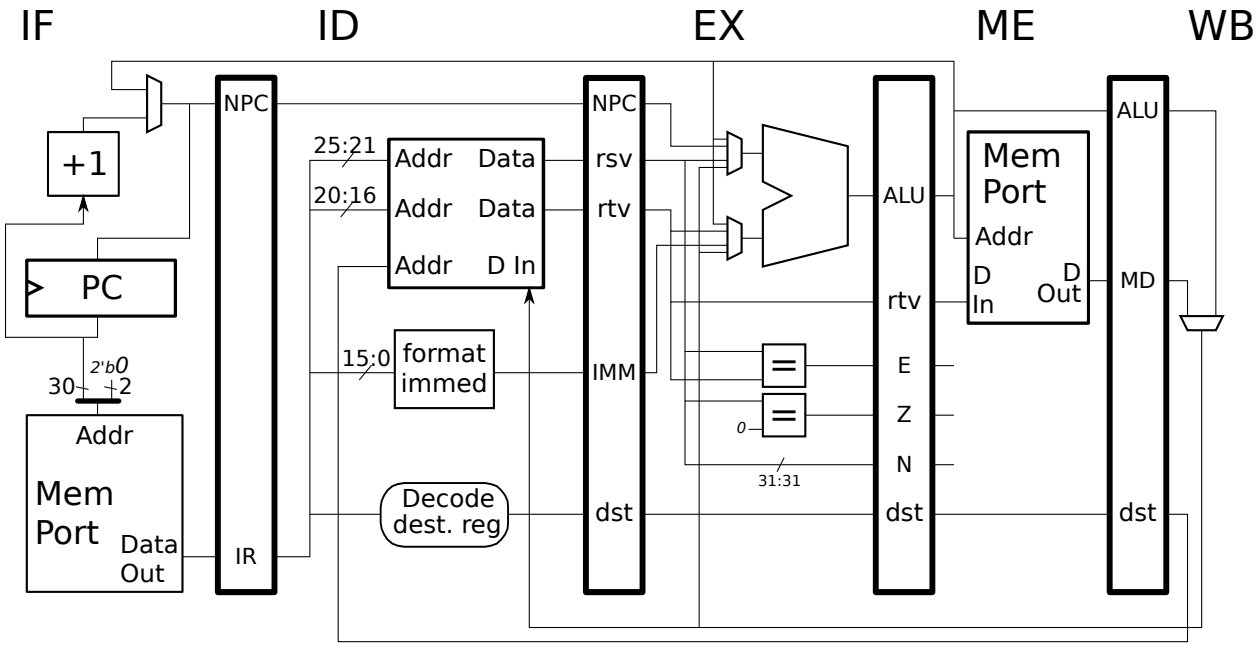
☐ Show the execution of the code fragment below on ☐ the implementation above. ☐ Be sure to check for dependencies.

```
addi r1, r2, 4

sw   r1, 4(r3)

lw   r3, 0(r1)
```


Staple This Side

Staple This Side

Problem 2, continued:



☐ Show the execution of the code fragment below on ☐ **the implementation above**. ☐ Be sure to check for dependencies.

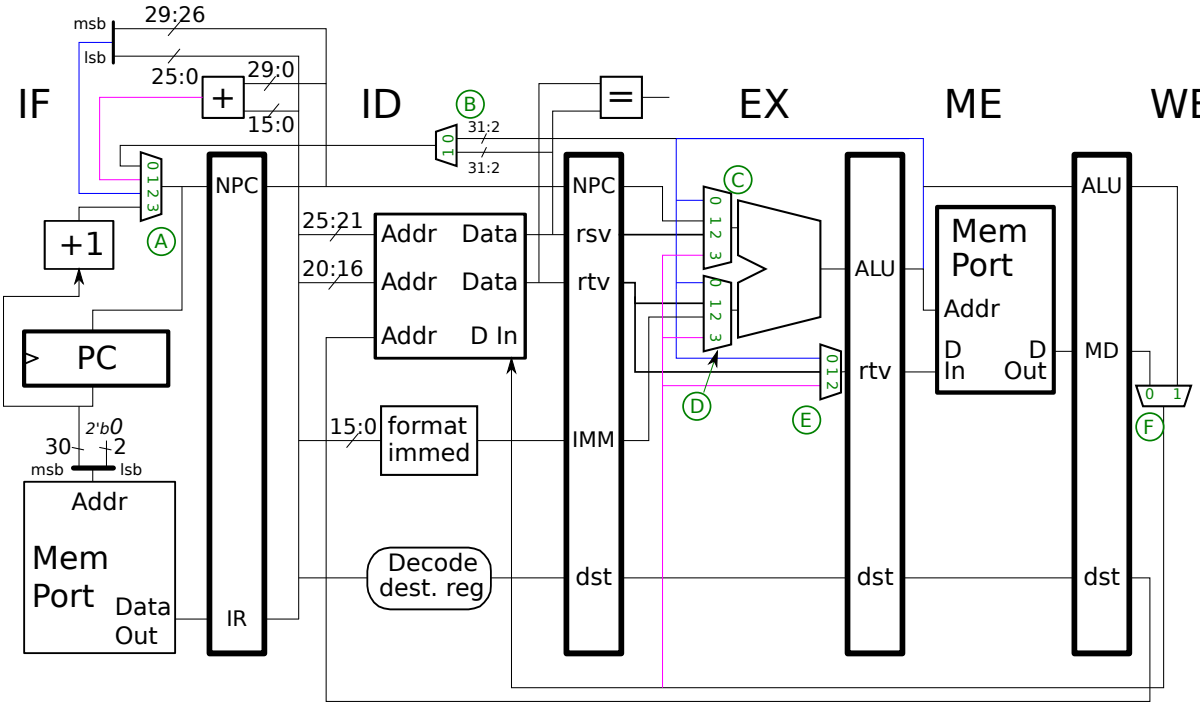
```
addi r1, r2, 4
lw   r3, 0(r1)
sw   r1, 4(r3)
```

☐ Show the execution of the code fragment below on ☐ the implementation above. ☐ Be sure to check for dependencies.

```
addi r1, r2, 4
sw   r1, 4(r3)
lw   r3, 0(r1)
```

Staple This Side

Problem 3: [16 pts] Appearing below is the MIPS implementation with labeled multiplexor select signals from Homework 3. Following that is an execution diagram along with a row showing select signal values for the D multiplexor. The first instruction, `add`, is shown.



☐ Complete the code fragment so that it produces the values shown for D.

# Cycle	0	1	2	3	4	5	6	7	8
add r1, r2, r3	IF	ID	EX	ME	WB				
		IF	ID	EX	ME	WB			
			IF	ID	EX	ME	WB		
				IF	ID	EX	ME	WB	
					IF	ID	EX	ME	WB

# Cycle	0	1	2	3	4	5	6	7	8
D:			1	0	1	2	3		

Problem 4: [16 pts] Rewrite each code fragment below so that it uses fewer instructions.

☐ Simplify code fragment.

```
addi r1, r0, 123
add  r1, r1, r2
```

☐ Simplify code fragment.

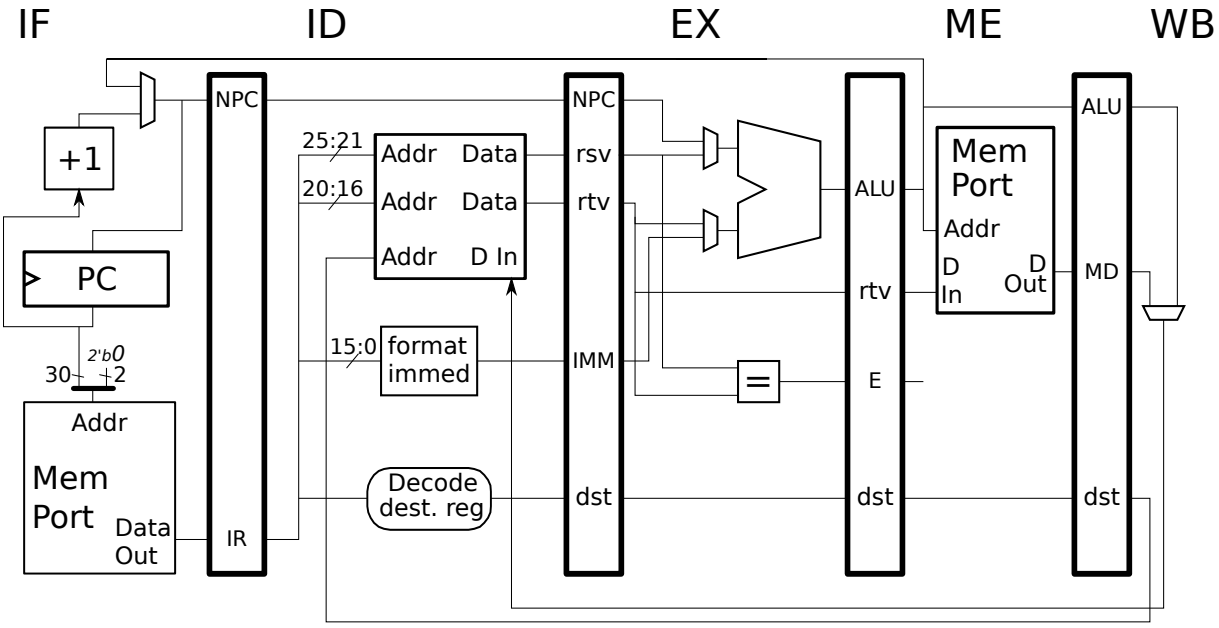
```
lw r1, 0(r2)
addi r2, r2, 4
lw r2, 0(r2)
```

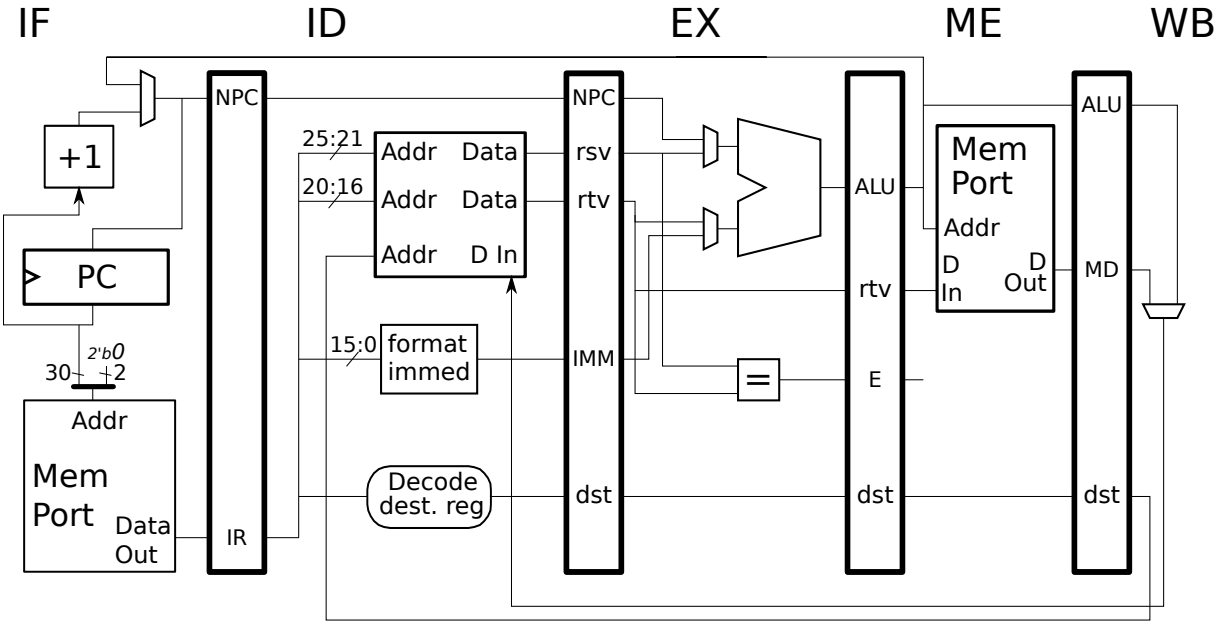
☐ Simplify code fragment.

```
sub r1, r2, r3
beq r1, r0,  TARG
lw r1, 0(r4)
```

Problem 5: [16 pts] Appearing below are two identical illustrations of one of our MIPS implementations. To the right are three executions of a code fragment, only one of which is possible on the implementation.

Identify the execution that is possible. For each of the executions that is not possible modify one of the illustrations below so that it is. The modification is very simple, just consider the target address. A few well chosen lines will suffice. No logic gates.





Staple This Side

Staple This Side

Staple This Side

☐ Is the execution below consistent with the unmodified implementation? ◯ *Yes* or ◯ *No*.

☐ If not, modify the implementation so that it is and ☐ label your modifications *A*.

```
LOOP: # Cycle      0  1  2  3  4  5  6  7  8  9  EXECUTION A
 bne r1, r2, TARG  IF ID EX ME WB
 add r1, r1, r3       IF ID EX ME WB
 sw r1, 0(r4)            IFx
 lui r5, 0x1234
 ori r5, r5, 0x6789
TARG:
 xor r8,r9,r10              IF ID EX ME WB
LOOP: # Cycle      0  1  2  3  4  5  6  7  8  9  EXECUTION A
```

☐ Is the execution below consistent with the unmodified implementation? ◯ *Yes* or ◯ *No*.

☐ If not, modify the implementation so that it is and ☐ label your modifications *B*.

```
LOOP: # Cycle      0  1  2  3  4  5  6  7  8  9  EXECUTION B
 bne r1, r2, TARG  IF ID EX ME WB
 add r1, r1, r3       IF ID EX ME WB
 sw r1, 0(r4)            IF IDx
 lui r5, 0x1234             IFx
 ori r5, r5, 0x6789
TARG:
 xor r8,r9,r10                 IF ID EX ME WB
LOOP: # Cycle      0  1  2  3  4  5  6  7  8  9  EXECUTION B
```

☐ Is the execution below consistent with the unmodified implementation? ◯ *Yes* or ◯ *No*.

☐ If not, modify the implementation so that it is and ☐ label your modifications *C*.

```
LOOP: # Cycle      0  1  2  3  4  5  6  7  8  9  EXECUTION C
 bne r1, r2, TARG  IF ID EX ME WB
 add r1, r1, r3       IF ID EX ME WB
 sw r1, 0(r4)            IF ID EXx
 lui r5, 0x1234             IF IDx
 ori r5, r5, 0x6789            IFx
TARG:
 xor r8,r9,r10                    IF ID EX ME WB
LOOP: # Cycle      0  1  2  3  4  5  6  7  8  9  EXECUTION C
```

Staple This Side

Problem 6: [15 pts] Answer each question below.

(*a*) Company *A* and *B* both come out with a new computer each year. Company *A* changes both the ISA and implementation each year. Company *B* changes only the implementation each year but uses the same ISA.

☐ Which company is following accepted practice?

☐ Which company's customers are more likely to stay with the company when it is time to upgrade to a new model? ☐ Explain.

(*b*) In MIPS `nop` is a pseudo instruction.

☐ What is a pseudo instruction?

☐ Does having too many pseudo instructions make implementations too expensive? ☐ Explain.

Staple This Side

(*c*) The first code fragment below, from code presented in the course, loads element `i` of an array of integers. (Here integers are four bytes.) Complete the second code fragment so that it loads element `i` from an array of shorts (A short is two bytes.).

```
# C CODE                        #  ASM REGISTER = C VARIABLE NAME
# int *a; ...          # $s1 = a;  $t0 = i    sizeof(int) = 4 chars.
# x = a[i];

sll $t5, $t0, 2       # $t5 -> i * 4;  Each element is four characters.
add $t5, $s1, $t5     # $t5 -> &a[i]  (Address of a[i].)
lw $t1, 0($t5)        # x = a[i];   $t1 -> a[i]
```

☐ Complete code below so that it loads a short.

```
# C CODE                        #  ASM REGISTER = C VARIABLE NAME
# short *a; ...          # $s1 = a;  $t0 = i    sizeof(short) = 2 chars.
# x = a[i];
```