Final Exam

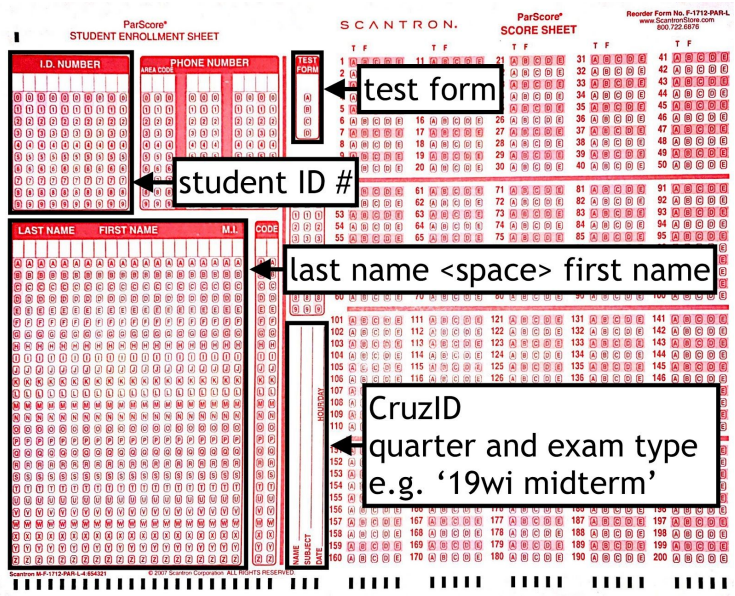
CMPE 012: Computer Systems and Assembly Language

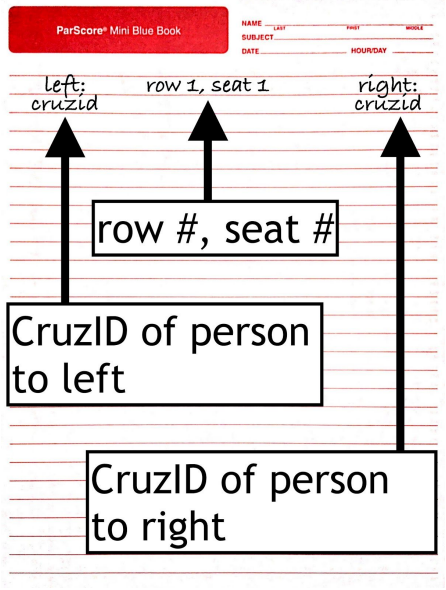
University of California, Santa Cruz

DO NOT BEGIN UNTIL YOU ARE TOLD TO DO SO.

This exam is closed book and closed notes. Only 4-function calculators are permitted. Answers must be marked on the Scantron form to be graded. All work must be written on the exam.

On the Scantron form, bubble in your name, student ID number, and test form (found in the footer of subsequent pages). In the center of the page write your CruzID, quarter, and exam type. On the back of the page, write the CruzIDs of students sitting to your left and right, and your row and seat number. See below.





On this page, write your last name, first name, CruzID, row and seat numbers, and the CruzIDs of the people to your immediate left and right. Once you are permitted to begin, write your CruzID on all subsequent pages of the exam.

You must sit in your assigned seat. Keep your student or government issued ID on your desk. Brimmed hats must be removed or turned around backwards. Only unmarked water bottles are permitted. Backpacks must be placed at the front of the room or along the walls. Your cell phone must be on a setting where it will not make noise or vibrate.

There are 45 questions on this exam; you only need to answer 42 for full points. The additional three questions (of your choosing) will be counted as extra credit. All questions are multiple choice, and some questions have more than one correct answer. **You must mark all correct answers to receive credit for a question.** Some true/false questions might list False as answer A and True as answer B. Follow the answers on the exam, **NOT** the T F notation on the Scantron Form. You will have 120 minutes to complete this exam.

Row # _____ Seat # _____ CruzID _____

Last Name _____ First Name _____

CruzID of person to left _____ CruzID of person to right _____

CruzID: _____ @ucsc.edu

CMPE 12 Final Exam - Version A

Winter 2019

Bits

1. How many bits are needed to encode one ASCII character?
   - ◯ A. 8 bits
   - ◯ B. 10 bits
   - ◯ C. 6 bits
   - ◯ D. 7 bits
   - ◯ E. 9 bits
2. What is the size of a word in MIPS? Select all that apply.
   - ◯ A. 8 bytes
   - ◯ B. 32 bits
   - ◯ C. 8 nybbles
   - ◯ D. 4 bytes
   - ◯ E. 32 bytes

Binary Arithmetic

3. Perform the following 12-bit two's complement addition.

```
    0b  0 1 1 1  1 1 0 0  0 1 0 0
+   0b  0 0 0 0  0 0 0 1  1 1 0 0
-------------------------------------
```

   - ◯ A. `000011011100`
   - ◯ B. `000001010100`
   - ◯ C. `011111100001`
   - ◯ D. `000010100111`
   - ◯ E. `011111100000`
4. Which of these 8-bit two's complement computations has carry out but no overflow? Select all that apply.
   - ◯ A. `0x1E + 0x26 = 0x44`
   - ◯ B. `0xFA + 0xED = 0xE7`
   - ◯ C. `0x0F + 0x85 = 0x94`
   - ◯ D. `0x01 + 0x7F = 0x80`
   - ◯ E. `0xFF + 0x01 = 0x00`
5. A logical right shift and an arithmetic right shift perform the same operation
   - ◯ A. True
   - ◯ B. False

Data Representation

6. Which IEEE 754 single precision floating point number is furthest from zero?
   - ◯ A. `0xC70FFFFF`
   - ◯ B. `0x47700000`
   - ◯ C. `0x1F8FFFFF`
   - ◯ D. `0x380FFFFF`
   - ◯ E. `0xB8700000`

7. What is the following base 9 number in base 5? 106_9
   - ◯ A. 123_5
   - ◯ B. 322_5
   - ◯ C. 742_5
   - ◯ D. 305_5
   - ◯ E. 222_5

8. What is the range of values for an 8-bit two's complement integer?
   - ◯ A. 0 to 255
   - ◯ B. -128 to 127
   - ◯ C. -127 to 128
   - ◯ D. -128 to 128
   - ◯ E. -127 to 127

9. What is the following 8-bit two's complement number in signed magnitude form?

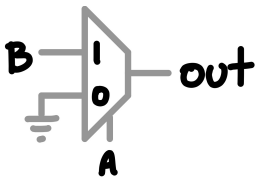
   `11010110`
   - ◯ A. `10110110`
   - ◯ B. `10101010`
   - ◯ C. `01010110`
   - ◯ D. `00101010`
   - ◯ E. `11010110`

10. What is the following base 3 number in base 7? 2101_3
    - ◯ A. 736_7
    - ◯ B. 123_7
    - ◯ C. 121_7
    - ◯ D. 64_7
    - ◯ E. 46_7

11. 6-bit two's complement, signed magnitude, and unsigned all represent the same number of integers, some just have more negative than positive.
    - ◯ A. True
    - ◯ B. False

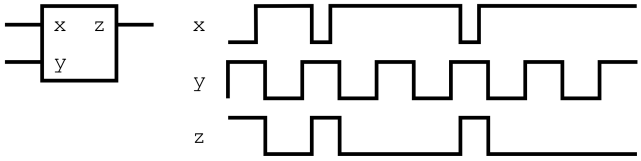
Logic Design

12. This figure is logically equivalent to which circuit?



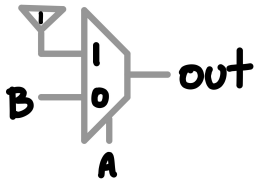
- ◯ A. XNOR gate
- ◯ B. XOR gate
- ◯ C. AND gate
- ◯ D. XOR gate
- ◯ E. Positive D-Latch

13. What device does this timing diagram represent?



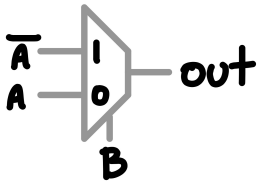
- ◯ A. Negative edge triggered D flip flop
- ◯ B. SR latch active high
- ◯ C. Positive edge triggered D flip flop
- ◯ D. D latch
- ◯ E. SR latch active low

14. This figure is logically equivalent to which circuit?



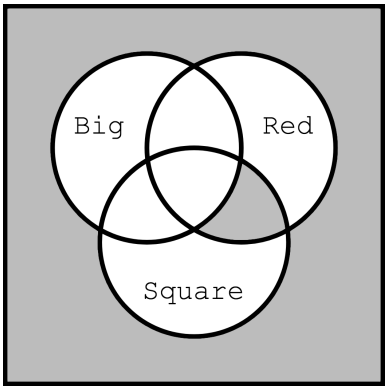
- ◯ A. AND gate
- ◯ B. XOR gate
- ◯ C. Negative D-Flip Flop
- ◯ D. XNOR gate
- ◯ E. OR gate

15. This figure is logically equivalent to which circuit?



- ◯ A. XOR gate
- ◯ B. XOR gate
- ◯ C. Negative D-Latch
- ◯ D. Positive D-latch
- ◯ E. XNOR gate

16. Select the Boolean expression matching the filled areas of this Venn diagram.



- ◯ A. $(\texttt{Red} + \texttt{Square}) \cdot \overline{(\texttt{Big} \cdot \texttt{Red} \cdot \texttt{Square})} \cdot (\texttt{Big} + \texttt{Red} + \texttt{Square})$
- ◯ B. $\texttt{Red} \cdot \texttt{Square} \cdot \overline{(\texttt{Big} \cdot \texttt{Red} \cdot \texttt{Square})} \cdot \overline{(\texttt{Big} + \texttt{Red} + \texttt{Square})}$
- ◯ C. $(\texttt{Red} + \texttt{Square}) + \overline{(\texttt{Big} \cdot \texttt{Red} \cdot \texttt{Square})} \cdot (\texttt{Big} + \texttt{Red} + \texttt{Square})$
- ◯ D. $\texttt{Red} \cdot \texttt{Square} \cdot \overline{(\texttt{Big} \cdot \texttt{Red} \cdot \texttt{Square})}$
- ◯ E. $\texttt{Red} \cdot \texttt{Square} \cdot \overline{(\texttt{Big} \cdot \texttt{Red} \cdot \texttt{Square})} + \overline{(\texttt{Big} + \texttt{Red} + \texttt{Square})}$

17. How many outputs does a 4-16 decoder have?
- ◯ A. 64
- ◯ B. 4
- ◯ C. 1
- ◯ D. 16
- ◯ E. 32

Memory

18. How many bits are needed to represent a memory location address in a 4TB memory space that is 64-byte addressable?
- ◯ A. 28
- ◯ B. 36
- ◯ C. 64
- ◯ D. 34
- ◯ E. 2^{34}

19. How much memory is allocated with the following line of code?

```
.asciiz "ce_12"
```

- ◯ A. 6 bytes
- ◯ B. 5 words
- ◯ C. 4 bytes
- ◯ D. 2 words
- ◯ E. 5 bytes

For the following two questions, assume a portion of data memory looks like this:

ADDRESS	CONTENTS
0x10011085	0xCD
0x10011084	0xAB
0x10011083	0x87
0x10011082	0x65
0x10011081	0x43
0x10011080	0x21

20. Assuming big endian memory storage, what is in $t7 after the following instructions?

```
ADDI $t0, $zero, 0x10011080
LH   $t7, 2($t0)
SW   $t7,  ($t0)
LW   $t7,  ($t0)
```

- ◯ A. 0x87654321
- ◯ B. 0x00008765
- ◯ C. 0x00006587
- ◯ D. 0xFFFF8765
- ◯ E. 0x00004321

21. Assuming little endian memory storage, what is in $t0 after the following instructions?

```
LI $t3, 0x10011082
LW $t0, ($t3)
```

- ◯ A. 0x00008765
- ◯ B. 0x5678BADC
- ◯ C. 0x6587ABCD
- ◯ D. Undefined. There will be an alignment error.
- ◯ E. 0xCDAB8765

ASCII

22. Decode the following ASCII string. Values are given in hex.

44 69 64 20 79 6f 75 20 65 76 65 72 20 68 65 61 72 20 74 68 65 20 74 72 61 67 65 64 79 20 6f 66 20 44 61 72 74 68 20 50 6c 61 67 75 65 69 73 20 74 68 65 20 57 69 73 65 3f

- ◯ A. Did you ever hear the tragedy of Darth Plagueis the Wise?
- ◯ B. No! Try not. Do. Or do not. There is no try.
- ◯ C. Help me, Obi-Wan Kenobi. You're my only hope.
- ◯ D. I have a bad feeling about this.
- ◯ E. I find your lack of faith disturbing.

23. Say that a user enters a single ASCII character in the range'0'-'9'. Assume that the user input is stored in $v0. Which MIPS instruction would you use to convert their input into an integer in the range 0-9?

- ◯ A. subi $t0, $v0, 49
- ◯ B. subi $t0, $v0, 48
- ◯ C. addi $t0, $v0, 48
- ◯ D. subi $t0, $v0, 30
- ◯ E. subi $t0, $v0, 60

MIPS

24. What is the value of $t0 after the following instructions are executed (represented in hex)?

```
li  $t1, 5
li  $t0, 5
loop:
   sll  $t0, $t0, 1
   addi $t1, $t1, -1
   bgez $t1, loop

   li   $v0, 10
   syscall
```

◯ A. 0x00000140
◯ B. 0x0001FA00
◯ C. 0x000000A0
◯ D. 0x00001400
◯ E. 0x001000FA

25. Which MIPS32 native/basic instruction(s) perform the same function as the following pseudo instruction?

```
ORI $s0 $t5 0xABCDEF00
```

◯ A.
```
ADDIU $1  $0  0xABCD
SRL   $1  $1  16
OR    $16 $13 $1
```
◯ B.
```
ORI   $16 $13 0xABCDEF00
```
◯ C.
```
LI    $1  0xABCDEF00
OR    $16 $13 $1
```
◯ D.
```
LUI   $1  0xEF00
OR    $16 $13 $1
ORI   $16 $16 0xABCD
```
◯ E.
```
LUI   $1  0xABCD
ORI   $1  $1  0xEF00
OR    $16 $13 $1
```

26. Which register(s) in MIPS must the callee preserve?

◯ A. $t0 - $t9
◯ B. $s0 - $s7
◯ C. $sp
◯ D. $v0 - $v1
◯ E. $a0 - $a3

27. What is the value of $t0 after the following instructions are executed?

```
li   $t0, 4
li   $t1, 5
add  $t0, $t1, $t0
addi $t0, $t0, -1
xor  $t0, $t0, $t0
```

◯ A. 0
◯ B. 6
◯ C. 10
◯ D. 8
◯ E. Not enough information given

28. What is the least significant byte stored in $t0 after the following MIPS commands execute?

```
li   $t0, 0x9F
andi $t0, $t0, 0x0F
```

◯ A. 00001111
◯ B. 10011111
◯ C. 11110000
◯ D. 00011111
◯ E. 00000000

29. What is printed to the screen after the following MIPS commands execute?

```
1   .data
2   prompt1: .ascii "I"
3   prompt2: .asciiz " LOVE"
4   prompt3: .asciiz " CE 12 FINAL"
5
6   .text
7   li $v0, 4
8   la $a0, prompt1
9   syscall
```

◯ A. I
LOVE
◯ B. I
◯ C. I LOVE
◯ D. I LOVE CE12 FINAL
◯ E. nothing

30. Processing an instruction requires the following steps

    a. Execute operation/evaluate effective address
    b. Write value to register file
    c. Fetch instruction from memory
    d. Access data from memory
    e. Decode instruction

    What is the correct ordering for these steps?

    ◯ A. ecadb
    ◯ B. ceadb
    ◯ C. caedb
    ◯ D. deacb
    ◯ E. aebdc

31. Which combination of MIPS instructions perform a pop operation of one word from the stack?

    ◯ A.
    ```
    sw   $t0, ($sp)
    subi $sp, $sp, 4
    ```
    ◯ B.
    ```
    addi $sp, $sp, 4
    lw   $t0, ($sp)
    ```
    ◯ C.
    ```
    lw   $t0, ($sp)
    addi $sp, $sp, 4
    ```
    ◯ D.
    ```
    subi $sp, $sp, 4
    sw   $t0, ($sp)
    ```
    ◯ E. none of the above

The next four questions will refer to the following MIPS code:

```
1   .text
2   la    $a0, str1
3   addiu $v0, $zero, 4
4   syscall
5
6   la    $a0, str2
7   syscall
8
9   lbu   $a0, str3
10  addiu $v0, $zero, 11
11  syscall
12
13  addiu $v0, $zero, 1
14  syscall
15
16  addiu $v0, $zero, 10
17  syscall
18
19  .data
20  str1: .ascii "hello"
21  str2: .asciiz "there"
22  str3: .byte 0x21 0x21 0x00
```

32. Assume you changed line 21 in the original program from

```
str2: .asciiz "there"
```

to

```
str2: .ascii "there"
```

What will be printed to the screen after the altered program completes execution?

- ◯ A. hellothere!!33
- ◯ B. hellotherethere!33
- ◯ C. hellothere!33
- ◯ D. hellothere!!there!!!33
- ◯ E. hellothere!!there!!33

33. What will be printed to the screen after the original program completes execution?

- ◯ A. hellotherethere!33
- ◯ B. hellothere!!33
- ◯ C. hellotherethere!!33
- ◯ D. hellothere!33
- ◯ E. hellotherethere!21

34. Assume you changed line 13 in the original program from

```
addiu $v0, $zero, 1
```

to

```
addiu $v0, $zero, 35
```

What will be printed to the screen after the altered program completes execution?

- ◯ A. hellotherethere!00000000000000000000000000100001
- ◯ B. hellothere!00000000000000000000000000100001
- ◯ C. hellothere!!00000000000000000000000000100001
- ◯ D. hellotherethere!0x00000021
- ◯ E. hellotherethere!33

35. Given the branch instruction in machine code

```
000101 00010 01000 1111111111111100
```

Assume the branch target address is 0x2004, what is the address of the branch instruction?

- ◯ A. None of the other answers
- ◯ B. `0x2004`
- ◯ C. `0x2010`
- ◯ D. `0x2018`
- ◯ E. `0x2014`

The addresses of some of the instructions of the following program are listed. Please refer to the program for the next two questions

```
            .text

            main:
0x00400000  jal   getString     #sets v0 to address of string
            move  $a0, $v0
            li    $v0, 4
0x0040000c  syscall

            li    $v0, 10
            syscall

0x00400018  getString:
            la     $v0, string1
0x00400020  jr     $ra


.data
string1:   .asciiz    "Greetings!"
```

36. What is the value of $pc right after the jal is taken?
    - ◯ A. 0x0040000c
    - ◯ B. 0x00400000
    - ◯ C. 0x00400018
    - ◯ D. 0x00400020
    - ◯ E. 0x00400004

37. What is the value of $ra right after the jal is taken?
    - ◯ A. 0x0040000c
    - ◯ B. 0x00400000
    - ◯ C. 0x00400020
    - ◯ D. 0x00400018
    - ◯ E. 0x00400004

Arrays

38. The next question refers to the following MIPS code. Assume all memory locations are initialized to 0x0000.

```
.text
la   $t0, space
li   $t1, 0
li   $t2, 0x39

loop:
sb   $t2, ($t0)
addi $t0, $t0, 1 # increment address
addi $t1, $t1, 1 # incrememt counter
subi $t2, $t2, 2
blt  $t1, 5, loop

la   $a0, space
li   $v0, 4
syscall

li   $v0, 10
syscall

.data
space: .space 10
```

What will be printed to the screen after the program completes execution?

- ◯ A. 9753
- ◯ B. `  '%#!`
- ◯ C. 97531
- ◯ D. 0x39 0x37 0x35 0x33 0x31
- ◯ E. 97531/

Instruction Decoding

39. Assume an ISA with 8 general purpose registers and the following 16-bit instruction format:
| opcode | RD | RS | RT |
How many unique instructions can this ISA have?

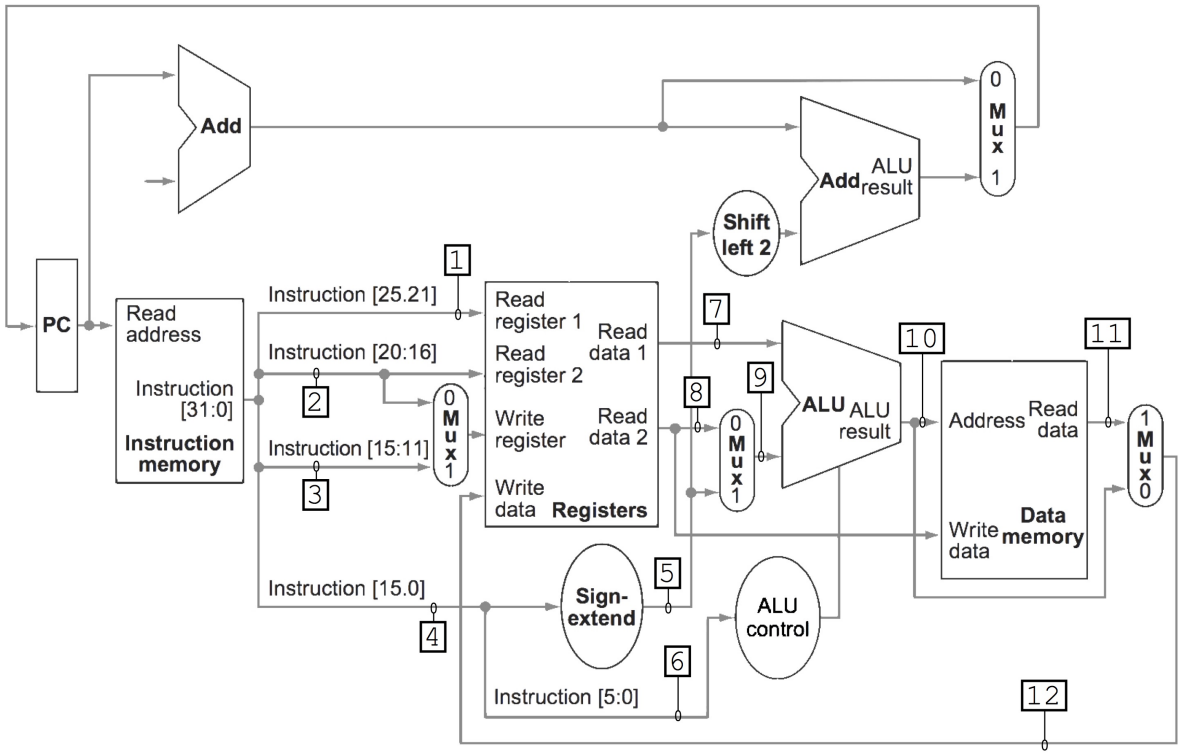
- ◯ A. 16
- ◯ B. 9
- ◯ C. 7
- ◯ D. 128
- ◯ E. 8

40. Decode the following MIPS32 instruction: `0x8D4C3210`

- ◯ A. `SW   $t2 0x0101 ($t3)`
- ◯ B. `AND  $t2 0x0123 $t4`
- ◯ C. `ANDI $t2 $t4 0x0123`
- ◯ D. `LW   $t4 0x3210 ($t2)`
- ◯ E. `LW   $t2 0x3210 ($t4)`

41. Decode the following MIPS32 instruction: `0x01097820`. Select all that apply.
    - ◯ A. `ADD $t0 $t1 $t7`
    - ◯ B. `AND $8 $9 $15`
    - ◯ C. `ADD $8 $9 $15`
    - ◯ D. `ADD $t7 $t0 $t1`
    - ◯ E. `ADD $15 $8 $9`

Data Path



42. Assume $t0 = 5 and `LB $t0 4($t0)` is executed. The programmer has access to all memory locations. What is the value on wire 9?
    - ◯ A. 5
    - ◯ B. 9
    - ◯ C. Not enough information given
    - ◯ D. 4
    - ◯ E. 8

43. The instruction `SUBI $t7 $t7 -1` is executed. What is the value on wire 4?
    - ◯ A. None of the other answers
    - ◯ B. 0xFFFF
    - ◯ C. 0xF
    - ◯ D. 0xFFFFFFFF
    - ◯ E. Not enough information given

44. Assume the values on wires 5, 7, 10, 11, and 12 are 0x08, 0x12, 0x1A, 0x1B and 0x1B respectively. Which instruction could correspond to these values?
    - ◯ A. Not enough information given
    - ◯ B. `ADDI $12 $8 18`
    - ◯ C. `ADDI $s1 $s2 8`
    - ◯ D. `LW  $t0 12($t1)`
    - ◯ E. `LH  $t8 8($t9)`

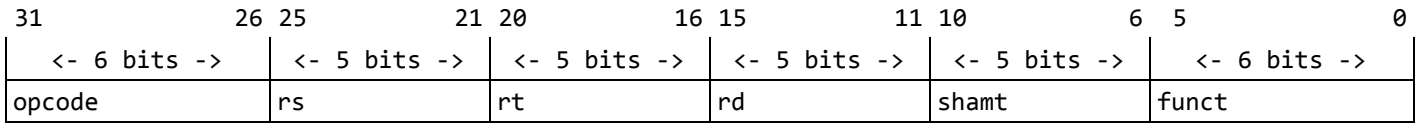
45. Assume $s0 = 0xAB, $s1 = 0xF4 and `SW $s1 8($s0)` is executed. What is the value on wire 8?
    - ◯ A. 0xF4
    - ◯ B. 0x08
    - ◯ C. Not enough information given
    - ◯ D. 0x10
    - ◯ E. 0xAB

REG NAME	REG #
$zero	0
$at	1
$v0	2
$v1	3
$a0	4
$a1	5
$a2	6
$a3	7
$t0	8
$t1	9
$t2	10
$t3	11
$t4	12
$t5	13
$t6	14
$t7	15
$s0	16
$s1	17
$s2	18
$s3	19
$s4	20
$s5	21
$s6	22
$s7	23
$t8	24
$t9	25
$k0	26
$k1	27
$gp	28
$sp	29

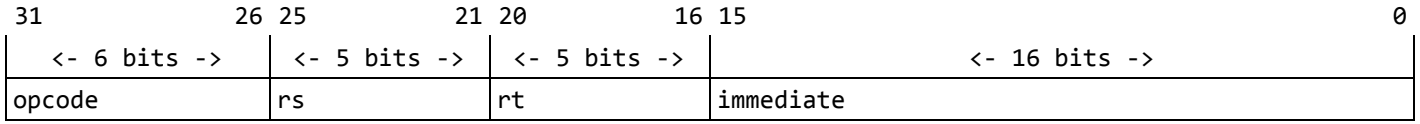
MNEMONIC	MEANING	TYPE	OPCODE	FUNCT
sll	Logical Shift Left	R	0x00	0x00
srl	Logical Shift Right (0-extended)	R	0x00	0x02
sra	Arithmetic Shift Right (sign-extended)	R	0x00	0x03
jr	Jump to Address in Register	R	0x00	0x08
mfhi	Move from HI Register	R	0x00	0x10
mflo	Move from LO Register	R	0x00	0x12
mult	Multiply	R	0x00	0x18
multu	Unsigned Multiply	R	0x00	0x19
div	Divide	R	0x00	0x1A
divu	Unsigned Divide	R	0x00	0x1B
add	Add	R	0x00	0x20
addu	Add Unsigned	R	0x00	0x21
sub	Subtract	R	0x00	0x22
subu	Unsigned Subtract	R	0x00	0x23
and	Bitwise AND	R	0x00	0x24
or	Bitwise OR	R	0x00	0x25
xor	Bitwise XOR (Exclusive-OR)	R	0x00	0x26
nor	Bitwise NOR (NOT-OR)	R	0x00	0x27
slt	Set to 1 if Less Than	R	0x00	0x2A
sltu	Set to 1 if Less Than Unsigned	R	0x00	0x2B
j	Jump to Address	J	0x02	NA
jal	Jump and Link	J	0x03	NA
beq	Branch if Equal	I	0x04	NA
bne	Branch if Not Equal	I	0x05	NA
blez	Branch if Less Than or Equal to Zero	I	0x06	NA
addi	Add Immediate	I	0x08	NA
addiu	Add Unsigned Immediate	I	0x09	NA
slti	Set to 1 if Less Than Immediate	I	0x0A	NA
sltiu	Set to 1 if Less Than Unsigned Immediate	I	0x0B	NA
andi	Bitwise AND Immediate	I	0x0C	NA
ori	Bitwise OR Immediate	I	0x0D	NA
xori	Bitwise XOR (Exclusive-OR) Immediate	I	0x0E	NA
lui	Load Upper Immediate	I	0x0F	NA
mfc0	Move from Coprocessor 0	R	0x10	NA
lb	Load Byte	I	0x20	NA
lh	Load Halfword	I	0x21	NA
lw	Load Word	I	0x23	NA
lbu	Load Byte Unsigned	I	0x24	NA
lhu	Load Halfword Unsigned	I	0x25	NA
sb	Store Byte	I	0x28	NA
sh	Store Halfword	I	0x29	NA
sw	Store Word	I	0x2B	NA

MNEMONIC	MEANING	TYPE	OPCODE	FUNCT
add	Add	R	0x00	0x20
addi	Add Immediate	I	0x08	NA
addiu	Add Unsigned Immediate	I	0x09	NA
addu	Add Unsigned	R	0x00	0x21
and	Bitwise AND	R	0x00	0x24
andi	Bitwise AND Immediate	I	0x0C	NA
beq	Branch if Equal	I	0x04	NA
blez	Branch if Less Than or Equal to Zero	I	0x06	NA
bne	Branch if Not Equal	I	0x05	NA
div	Divide	R	0x00	0x1A
divu	Unsigned Divide	R	0x00	0x1B
j	Jump to Address	J	0x02	NA
jal	Jump and Link	J	0x03	NA
jr	Jump to Address in Register	R	0x00	0x08
lb	Load Byte	I	0x20	NA
lbu	Load Byte Unsigned	I	0x24	NA
lh	Load Halfword	I	0x21	NA
lhu	Load Halfword Unsigned	I	0x25	NA
lui	Load Upper Immediate	I	0x0F	NA
lw	Load Word	I	0x23	NA
mfc0	Move from Coprocessor 0	R	0x10	NA
mfhi	Move from HI Register	R	0x00	0x10
mflo	Move from LO Register	R	0x00	0x12
mult	Multiply	R	0x00	0x18
multu	Unsigned Multiply	R	0x00	0x19
nor	Bitwise NOR (NOT-OR)	R	0x00	0x27
or	Bitwise OR	R	0x00	0x25
ori	Bitwise OR Immediate	I	0x0D	NA
sb	Store Byte	I	0x28	NA
sh	Store Halfword	I	0x29	NA
sll	Logical Shift Left	R	0x00	0x00
slt	Set to 1 if Less Than	R	0x00	0x2A
slti	Set to 1 if Less Than Immediate	I	0x0A	NA
sltiu	Set to 1 if Less Than Unsigned Immediate	I	0x0B	NA
sltu	Set to 1 if Less Than Unsigned	R	0x00	0x2B
sra	Arithmetic Shift Right (sign-extended)	R	0x00	0x03
srl	Logical Shift Right (0-extended)	R	0x00	0x02
sub	Subtract	R	0x00	0x22
subu	Unsigned Subtract	R	0x00	0x23
sw	Store Word	I	0x2B	NA
xor	Bitwise XOR (Exclusive-OR)	R	0x00	0x26
xori	Bitwise XOR (Exclusive-OR) Immediate	I	0x0E	NA

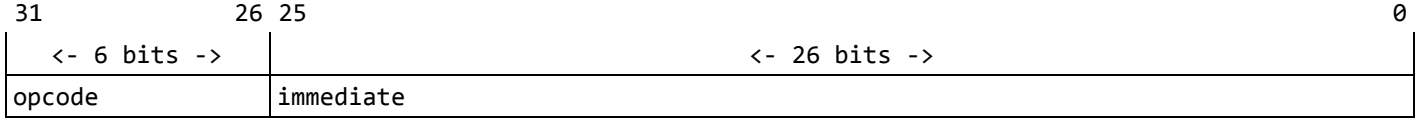
R Type: instr rd rs rt (arithmetic, logical)
instr rd rt shamt (shifts)



I Type: instr rt rs immediate (arithmetic, logical)
branch rs rt immediate (branches)
instr rt immediate(rs) (loads, stores)



J Type: j immediate (jumps)



ASCII CODE					ASCII CODE					ASCII CODE					- ASCII CODE				
BIN	OCT	DEC	HEX	CHARACTER	BIN	OCT	DEC	HEX	CHARACTER	BIN	OCT	DEC	HEX	CHARACTER	BIN	OCT	DEC	HEX	CHARACTER
010 0000	40	32	20	space	011 1000	70	56	38	8	101 0000	120	80	50	P	110 1000	150	104	68	h
010 0001	41	33	21	!	011 1001	71	57	39	9	101 0001	121	81	51	Q	110 1001	151	105	69	i
010 0010	42	34	22	"	011 1010	72	58	3A	:	101 0010	122	82	52	R	110 1010	152	106	6A	j
010 0011	43	35	23	#	011 1011	73	59	3B	;	101 0011	123	83	53	S	110 1011	153	107	6B	k
010 0100	44	36	24	$	011 1100	74	60	3C	<	101 0100	124	84	54	T	110 1100	154	108	6C	l
010 0101	45	37	25	%	011 1101	75	61	3D	=	101 0101	125	85	55	U	110 1101	155	109	6D	m
010 0110	46	38	26	&	011 1110	76	62	3E	>	101 0110	126	86	56	V	110 1110	156	110	6E	n
010 0111	47	39	27	'	011 1111	77	63	3F	?	101 0111	127	87	57	W	110 1111	157	111	6F	o
010 1000	50	40	28	(	100 0000	100	64	40	@	101 1000	130	88	58	X	111 0000	160	112	70	p
010 1001	51	41	29	)	100 0001	101	65	41	A	101 1001	131	89	59	Y	111 0001	161	113	71	q
010 1010	52	42	2A	*	100 0010	102	66	42	B	101 1010	132	90	5A	Z	111 0010	162	114	72	r
010 1011	53	43	2B	+	100 0011	103	67	43	C	101 1011	133	91	5B	[	111 0011	163	115	73	s
010 1100	54	44	2C	,	100 0100	104	68	44	D	101 1100	134	92	5C	\	111 0100	164	116	74	t
010 1101	55	45	2D	-	100 0101	105	69	45	E	101 1101	135	93	5D	]	111 0101	165	117	75	u
010 1110	56	46	2E	.	100 0110	106	70	46	F	101 1110	136	94	5E	^	111 0110	166	118	76	v
010 1111	57	47	2F	/	100 0111	107	71	47	G	101 1111	137	95	5F	_	111 0111	167	119	77	w
011 0000	60	48	30	0	100 1000	110	72	48	H	110 0000	140	96	60	`	111 1000	170	120	78	x
011 0001	61	49	31	1	100 1001	111	73	49	I	110 0001	141	97	61	a	111 1001	171	121	79	y
011 0010	62	50	32	2	100 1010	112	74	4A	J	110 0010	142	98	62	b	111 1010	172	122	7A	z
011 0011	63	51	33	3	100 1011	113	75	4B	K	110 0011	143	99	63	c	111 1011	173	123	7B	{
011 0100	64	52	34	4	100 1100	114	76	4C	L	110 0100	144	100	64	d	111 1100	174	124	7C	|
011 0101	65	53	35	5	100 1101	115	77	4D	M	110 0101	145	101	65	e	111 1101	175	125	7D	}
011 0110	66	54	36	6	100 1110	116	78	4E	N	110 0110	146	102	66	f	111 1110	178	126	7E	~
011 0111	67	55	37	7	100 1111	117	79	4F	O	110 0111	147	103	67	g	111 1111	177	127	7F	DEL

SERVICE	CODE IN $v0	ARGUMENTS	RESULT
print integer	1	$a0 = integer to print	
print float	2	$f12 = float to print	
print double	3	$f12 = double to print	
print string	4	$a0 = address of null-terminated string to print	
read integer	5		$v0 contains integer read
read float	6		$f0 contains float read
read double	7		$f0 contains double read
read string	8	$a0 = address of input buffer $a1 = maximum number of characters to read	*See note below table*
sbrk (allocate heap memory)	9	$a0 = number of bytes to allocate	$v0 contains address of allocated memory
exit (terminate execution)	10		
print character	11	$a0 = character to print	*See note below table*
read character	12		$v0 contains character read
open file	13	$a0 = address of null-terminated string containing filename $a1 = flags $a2 = mode	$v0 contains file descriptor (negative if error). See note below table
read from file	14	$a0 = file descriptor $a1 = address of input buffer $a2 = maximum number of characters to read	$v0 contains number of characters read (0 if end-of-file, negative if error). See note below table
write to file	15	$a0 = file descriptor $a1 = address of output buffer $a2 = number of characters to write	$v0 contains number of characters written (negative if error). See note below table
close file	16	$a0 = file descriptor	
exit2 (terminate with value)	17	$a0 = termination result	*See note below table*
Services 1 through 17 are compatible with the SPIM simulator, other than Open File (13) as described in the Notes below the table. Services 30 and higher are exclusive to MARS.			
time (system time)	30		$a0 = low order 32 bits of system time $a1 = high order 32 bits of system time. See note below table
MIDI out	31	$a0 = pitch (0-127) $a1 = duration in milliseconds $a2 = instrument (0-127) $a3 = volume (0-127)	Generate tone and return immediately. See note below table
sleep	32	$a0 = the length of time to sleep in milliseconds.	Causes the MARS Java thread to sleep for (at least) the specified number of milliseconds. This timing will not be precise, as the Java implementation will add some overhead.
MIDI out synchronous	33	$a0 = pitch (0-127) $a1 = duration in milliseconds $a2 = instrument (0-127) $a3 = volume (0-127)	Generate tone and return upon tone completion. See note below table
print integer in hexadecimal	34	$a0 = integer to print	Displayed value is 8 hexadecimal digits, left-padding with zeroes if necessary.
print integer in binary	35	$a0 = integer to print	Displayed value is 32 bits, left-padding with zeroes if necessary.
print integer as unsigned	36	$a0 = integer to print	Displayed as unsigned decimal value.

MIPS Address Space (Not to Scale)