Bitwise Logical Operations

sll, or

EEL 3801

What is the value of $t2 in binary after completing the below instructions:

```
addi  $t0,$zero,0x00000055
addi  $t1,$zero,0x0000007A
sll   $t2,$t0,4
or    $t2,$t2,$t1
```

```
a) 0000 0000 0000 0000 0000 0000 0101 0101
b) 0000 0000 0000 0000 0000 0000 0111 1010
c) 0000 0000 0000 0000 0000 0101 0101 0000
d) 0000 0000 0000 0000 0000 0101 0101 1010
e) 0000 0000 0000 0000 0000 0101 0111 1111
f) 1111 1111 1111 1111 1111 0101 0101 0000
g) 1111 1111 1111 1111 1111 0000 0111 1010
h) 1111 1111 1111 1111 1111 0101 0111 1010
i) none of the choices listed
```

Solution:
$t0 gets 0 + 0x00000055, which is equal to 0000 0000 0000 0000 0000 0000 0101 0101. $t1 gets 0 + 0x0000007A, which is equal to 0000 0000 0000 0000 0000 0000 0111 1010. sll $t2, $t0, 4 shifts $t0 by 4 (1 hex number), and then puts it into $t2 = 0000 0000 0000 0000 0000 0101 0101 0000 = 0x00000550. Or $t2, $t2, $t1 logically ors the bits between $t1 and $t2 and then puts it into $t2 = 0000 0000 0000 0000 0000 0101 0111 1010 = 0x0000057A which is not listed so select i) for none.

Bitwise Logical Operations

writing sll, ori, andi based on task description

Sought:
Find the shortest sequence MIPS instructions, i.e. dynamic instruction count, for the following task:

> Extract bits 16 down to 11 inclusive from register **`$t0`** and use the value of this field to replace bits 31 down to 26 inclusive in register **`$t1.`** Perform these steps without modifying the other 26 bits of register **`$t1`**.

Hint: "Extract bits 16 down to 11" means to mask off all of the other bits except those specified.
Namely, a MIPS register has 32 bits:

```
bit31 bit30 bit29 bit28 ... bit1 bit0
```

so take out the substring:

```
bit16 bit15 bit14 bit13 bit12 bit11
```

then perform the next step requested with just those bits.
Drawing the register contents as above can help to design the operations needed.
After getting the flow, look for optimizations to get the same functionality in fewer instructions.

Solution 1: allowing pseudoinstructions:

```
.text
sll $t2, $t0, 15
andi $t2, $t2, 0xFC000000
andi $t1, $t1, 0x03FFFFFF
or $t1, $t1, $t2
```

*This solution is **not unique** --- there are others possible using different bit operations.*

Hint: this problem aligns the bits first, masks out using AND, then merges the intermediate results using OR.

PLEASE try it in MARS … you can scrape the text and paste it into MARS to step the operation and watch contents of $t1 and $t2 change after each instruction executes.

Bitwise Logical Operations
writing MIPS instruction based C shift

Sought:
For the following C statement:

```
A = C[0] << 4;
```

write a minimal sequence of MIPS instructions that does the identical operation.
Assume `$t1 = A`, `$t2 = B`, and `$s1` is the base address of `C`.

Solution:

```
lw $t3, 0($s1)

sll $t1, $t3, 4
```

ASCII Values 1/2
determine character range

Given that an ASCII character is stored in $s0, what does the following MIPS assembly program do?

```
        addi $t0, $zero, 0x60
        add  $t1, $zero, $zero
        slt  $t7, $t0, $s0
        bne  $t7, $zero, skip
        srl  $t7, $t7, 4
        nor  $t0, $t0, $zero
        j    done
skip:   addi $t0, $zero, 0x7b
        slt  $t1, $s0, $t0
done:   add  $t0,$t7,$t0
```

a) Converts lower case to upper case
b) Converts upper case to lower case
c) Sets $t1 to 1 if the character in $s0 is upper case
d) Sets $t7 to 1 if the character in $s0 is upper case
e) Sets $t1 to 1 if the character in $s0 is lower case
f) Resets $t1 to 0 if the character in $s0 is lower case
g) Multiplies the value in $t7 by 4 if and only if the character in $s0 is lower case
h) Multiplies the value in $t7 by 16 if and only if the character in $s0 is lower case
i) Multiplies the value in $t7 by 4 if and only if the character in $s0 is upper case
j) Multiplies the value in $t7 by 16 if and only if the character in $s0 is upper case
k) Multiplies the value in $t7 by 4 if and only if the character in $s0 is a digit
l) Multiplies the value in $t7 by 16 if and only if the character in $s0 is a digit
m) None of the choices listed

ASCII Values 2/2
determine character range

Solution:

Knowing that an ASCII character resides in $s0, and upon inspection of the code, we realize that the code first checks to see if $s0 is greater than 0x60. It then checks to see if it is less than 0x7b. If it is greater than 0x60 and smaller than 0x7b, then it sets $t1 = 1, if not, then it $t1 is still = 0. Shift right divides by a power of two rather than multiplying. Using the ASCII table, the range 0x61 to 0x7a are all the lower case letters, so answer e)

Bitwise Logical Operations 1/2

determine number of 1's

Given: Given the partial MIPS assembly program below that determines the number of 1's in $s0.

```
         addi    $s1,$zero,0
         addi    $t1,$zero,32
Loop:    <# INSTRUCTION_X>
         add     $s1,$s1,$t2
         <# INSTRUCTION_Y>
         addi    $t1,$t1,-1
         bne     $t1,$zero,Loop
```

Sought:
Select 2 instructions at left to replace <INSTRUCTION_X> and <INSTRUCTION_Y> above such that **`$s1`** contains the number of 1's in **`$s0`** upon completion.

Note: ordering of instructions selected does not impact your score for this question.

Choices (choose 2):
a) andi $s0, $t1, 1
b) andi $t2, $s0, 2
c) andi $t2, $s0, 1
d) andi $t2, $s0, -1
e) andi $so, $t1, -1
f) ori $t2, $s0, 1
g) ori $t2, $s0, 2
h) ori $s0 $t0, -1
i) ori $t2, $s0, -1
j) sll $s0, $s1, 2
k) sll $s0, $s0, 1
l) srl $s0, $s0, 1
m) srl $s0, $s1, 1
n) none of the choices listed

Bitwise Logical Operations 2/2
determine number of 1's

Solution:

The loop body executes 32 times, which corresponds to the number of bits in a MIPS register. The body code needs to check each bit shifted into the LSb and count how many 1's there are. For <INSTRUCTION_X>, we need it to mask the LSb of $s0 then place that bit in $t1 in order for $s1 to be incremented by it in the next instruction; we do this masking by andi $t2, $s0, 1. Once the first digit has been checked and counted, we have to check the next digit, which is where <INSTRUCTION_Y> comes in; we do this by shifting $s0 to the right before the next loop using srl $s0, $s0, 1. So choose answers c) and l).

Completed design is:

```
        addi   $s1,$zero,0
        addi   $t1,$zero,32
Loop:   andi   $t2,$s0,1        # INSTRUCTION_X
        add    $s1,$s1,$t2
        srl    $s0,$s0,1        # INSTRUCTION_Y
        addi   $t1,$t1,-1
        bne    $t1,$zero,Loop
```

Translating Assembly to Machine Code

Understanding bit-field contents and expressing in Bin/Hex

Given: the following MIPS instructions needing to be converted to machine code:

Partial Credit 1) `sll $t2,$t1,1`

Partial Credit 2) `addi $s0,$0,73`

Partial Credit 3) `or $t1,$t2,$t3`

Partial Credit 4) `sw $t1,-8($t2)`

Sought: Corresponding binary and hexadecimal machine code for each instruction.

Solution:

Using the Testing Reference sheet to determine the content and location of each of the 6 instruction fields, the following are obtained:

1) $\texttt{0000 0000 0000 1001 0101 0000 0100 0000}_{\texttt{two}}$ = 0x00095040

2) $\texttt{0010 0000 0001 0000 0000 0000 0100 1001}_{\texttt{two}}$ = 0x20100049

3) $\texttt{0000 0001 0100 1011 0100 1000 0010 0101}_{\texttt{two}}$ = 0x014B4825

4) $\texttt{1010 1101 0100 1001 1111 1111 1111 1000}_{\texttt{two}}$ = 0xAD49FFF8

Translating Machine Code to Assembly
Disassembly from Hex to MIPS

Given: the following MIPS machine code instructions expressed in hexadecimal:

Partial Credit 1) `0x00004820`

Partial Credit 2) `0x308a0001`

Partial Credit 3) `0x11400003`

Sought: Write the corresponding MIPS assembly instruction.

Solution: Using the Testing Reference sheet to determine the content and location of each of the 6 instruction fields, the following are obtained:

Solution 1) `add $t1,$0,$0`

Solution 2) `andi $t2,$a0,1`

Solution 3) `beq $t2,$0,12`

Partial Credit 4) What is the term for representing instructions as bits in memory and who is credited for it?
a) Binary Commutativity by Kilby
b) Binary Commutativity by Shockley
c) Semantic Gap by Kilby
d Semantic Gap by Von Neumann
e) Semantic Gap by Shockley
f) Stored Program Concept from Von Neumann
g) Stored Program Concept from Gauss

Solution 4)

We saw in Module 8 video, Jack Kilby's first crude Integrated Circuit. William Shockley was an inventor of the transistor as mentioned at start of Module 3a video. The semantic gap is term for the difference between what the programmer wants to realize and what the hardware is capable of performing in a single machine instruction. The Stored Program Concept was put forth by John Von Neumann so choose f).

Branch Address Calculation
beq encoding

Given: `beq $s0, $s1, Label`

The 'beq' instruction resides in memory at address PC=120_{ten}.
The machine code for this instruction is given below:

000100	10000	10001	1111 1111 1111 1100
op	*rs*	*rt*	*offset*

Sought: Find the address of next instruction executed if \$s0 == \$s1. Represent your answer in binary.

Solution:

The branch address is equal to:

Branch address = PC + 4 + (16-bit number sign-extended and shifted left by 2)

The 16-bit number, sign-extended, and shifted left by 2 is:

11111111 11111111 11111111 11110000

To represent this in decimal, invert and add 1, to obtain:

0…00001111+1 = 10000, which is equal to -16 as the offset is represented in 2sc

Therefore, the branch address is:

120 + 4 – 16 = 108

In 32-bit binary it is: 00000000 00000000 00000000 01101100

Branch Address Calculation 1/2
beq encoding

Given: Consider the following MIPS assembly code.

```
40,000 start:   lw      $t0, 0($s0)
40,004          addi    $t0, $t0, 1
40,008          bne     $t0, $t1, start
```

Sought: What value is contained in the bne instruction's 16-bit branch field?

a) 39,996
b) 40,000
c) 40,004
d) 40,008
e) 160,014
f) 160,064
g) 1
h) -1
i) 2
j) -2
k) 3
l) -3
m) Not enough information: depends on the value in $t0
n) Not enough information: depends on the value in $t0 and the value in $t1
o) None of the choices listed

=

Branch Address Calculation 2/2

beq encoding

EEL 3801

Solution:

After the bne instruction is executed, the program counter is incremented by 4, which is 1 instruction. The “start” label is 2 instructions back from the bne instruction, but PC has already been incremented by 4 (or 1 instruction), so the decimal value stored in the bne’s branch field must be -3. So select choice l)

Jump Address Calculation 1/2

j encoding

Given: Consider the following MIPS assembly code.

```
1024   root:   lw      $t0, 0($s0)
1028           addi    $t0, $t0, 1
1032           beq     $t0, $t1, done
1036           j       root
1040   done:    slti   $t7, $t3, $t0
```

Sought: What value contained in the Jump Target field of the {`j root`} instruction?

a) 1020
b) 1024
c) 1028
d) 4096
e) 3
f) -3
g) -12
h) 0xFF
i) 0x10
j) 0x100
k) 0x400
l) 0x000100000000
m) None of the choices listed

Jump Address Calculation 2/2
j encoding

Solution:

The jump target field for jump instructions use absolute addressing. The absolute address of the jump label is 1024, but the jump instruction hard-codes the bottom 2 bits as zeros for word boundary yielding. Because of this, the address 1024 is shifted to the right 2 bits, which is equivalent to dividing by 4. 1024/4 = 256 decimal = 0x100 hex, so choose answer j).

Translating Loop to Machine Code

loop with bne and j

```
80,000  Loop:sll     $t1,$s3,2   # Temp reg $t1 = 4 * i
80,004       add $tl,$tl,$s6     # $tl = address of save[i]
80,008       lw  $t0,0($t1)      # Temp reg $t0 = save[i]
80,012       bne $t0,$s5, Exit   # go to Exit if save[i] ≠ k
80,016       addi $s3,$s3,1      # i = i + 1
80,020       j    Loop           # go to Loop
80,024  Exit:
```

Solution See next page first then come back for detailed explanation →

:

`bne` branches from address 80,012 to 80,024

- branch address = PC+4+offset so offset = 80,024–80,012–4 = 8_{ten} = 1000 $_{two}$
- drop the '00' from the right side LSb's as only multiples of 4 for instr addr
- branch field becomes 10 $_{two}$ so that instruction = (5, 8, 21, 2) $_{ten}$ for its fields

`j loop` jumps to address 80,000

- 80,000 $_{ten}$ = 0000 0000 0000 0001 0011 1000 1000 0000 $_{two}$
- drop the '00' from the right side LSb's as only multiples of 4 for instr addr … which is same as dividing by 4, yielding 20,000 $_{ten}$
- so that instruction = (2, 20000) $_{ten}$ for its fields

Translating Loop to Machine Code

loop with bne and j

<u>Final machine code</u>: expressed on this slide in decimal for readability

				op	rs	rt	rd	sa	funct
`Loop:`	`sll`	`$t1, $s3, 2`	80000	0	0	19	9	2	0
	`add`	`$t1, $t1, $s6`	80004	0	9	22	9	0	32
	`lw`	`$t0, 0($t1)`	80008	35	9	8	0		
	`bne`	`$t0, $s5, Exit`	80012	5	8	21	2 "relative"		
	`addi`	`$s3, $s3, 1`	80016	8	19	19	1		
	`j`	`Loop`	80020	2	20000 "absolute"				
`Exit:`	`…`		80024						

Units for branch/jump are instructions.
(i.e. 32-bit words not Bytes)