LE9.3.1: ALUC Instructions

1/1 point (ungraded)

- Summary of Instruction Formats (PDF)
- Beta Documentation (PDF)

For the Beta instruction sequence shown below, indicate the 32-bit two's complement values of the specified registers after the sequence has been executed by the Beta. The effect of the instructions is cummulative, later instructions use the values stored by earlier instructions.

You can find detailed descriptions of each Beta instruction in the "Beta Documentation" handout -- see link above. Remember that register values and the ALU use a 32-bit two's complement representation.

Hint: You can enter answers in hex by specifying a "`0x`" prefix, *e.g.*, 17 could be entered as "`0x11`". Usually one would enter addresses, values in memory, etc. using hex. You can also use a "`0b`" prefix to enter a binary value, e.g., "0b10001".

```
ADDC(R31,0x1234,R1)
SUBC(R1,-1,R2)
SRAC(R2,8,R3)
SHLC(R2,32,R4)
XORC(R2,0xFFFF,R5)
ANDC(R5,0x0F0F,R6)
```

Value left in R1? 0x1234 ✓

Answer: 0x1234

Value left in R2? 0x1235 ✓

Answer: 0x1235

Value left in R3? 0x12 ✓

Answer: 0x12

Value left in R4? 0x1235 ✓

Answer: 0x1235

Value left in R5? 0xFFFFEDCA ✓

Answer: 0xFFFFEDCA

Value left in R6? 0x0D0A ✓

Answer: 0x00000D0A

Explanation

`ADDC(R31,0x1234,R1)` adds 0 to 0×1234 = 0×1234

`SUBC(R1,-1,R2)` 0×1234 - (-1) = 0×1235. Subtracting -1 is the same as adding 1.

`SRAC(R2,8,R3)` shifting 0×1235 right by 8 bits gives 0×12. Since the A operand was positive, the arithmetic right shift used 0 to fill in the vacated bits.

`SHLC(R2,32,R4)` The answer is 0×1235, i.e., the value from R2 is unchanged. The insight here is that the shift instructions only use the bottom 5-bits of the B operand to determine the shift count. 32 = 0b100000, so the shift amount is 0.

`XORC(R2,0xFFFF,R5)` the 0xFFFF second operand will be sign-extended to 0xFFFFFFFF. XORing with all 1's gives the bit-wise complement of 0×00001235 = 0xFFFFEDCA

`ANDC(R5,0x0F0F,R6)` ANDing 0xFFFFEDCA with 0×00000F0F gives 0×00000D0A. ANDC is often used to select particular groups of bits from the A operand, an operation called "masking".

Submit

Answers are displayed within the problem

LE9.3.2: Do we need SUBC?

0.0/1.0 point (ungraded)

A 6.004 student here at MIT has proposed that all SUBC instructions of the form

```
    SUBC(Rx,const,Ry)
```

can be replaced by an ADDC that uses the negative of the constant:

```
    ADDC(Rx,-const,Ry)
```

Is the student on to something? Will this transformation work for all SUBC instructions?

- ◯ Yes, the transformation will work for all SUBC instructions
- ◯ No, there are SUBC instructions with no ADDC equivalent

Submit

LE9.3.3: ALUC Instruction Encoding?

0.0/1.0 point (ungraded)

Please give the 32-bit binary encoding for each of the ALUC instruction shown below.

Note that when translating the symbolic form of the ALUC instruction into binary, the program (called the "assembler") we run will only use the low-order 16 bits of the value provided by the second operand, even if that value requires more than 16 bits to represent correctly. Our assembler is dumb :(

One other note: we've been using the symbol R0 to represent register 0. The assembler knows to treat the Rx symbols as shorthand for the decimal constant x, e.g., R17 is a more readable alternative to specifying the constant 17.

(A) 32-bit encoding for `ADDC(R31,0x1234,R1)`?

(B) 32-bit encoding for `XORC(R1,−17,R0)`?

(C) 32-bit encoding for `CMPLEC(R12,40000,R0)`?

(D) 32-bit encoding for `ADDC(R1,R2,R3)`?

Submit

Discussion

Hide Discussion

Topic: 9. Designing an Instruction Set / LE9.3

Add a Post

Show all posts by recent activity

Google failed me the second time in many years!
Interesting issue: I use Google search box as a calculator a lot. I was trying to picture SRAC(R2,8,R...
2

Sign Extension
Why does the 0xFFFF get extended to 0xFFFFFFFF in the first problem for the XORC problem? Is it...
5

LE 9.3.2
I believe it is this Dumb Assembler which leads to the incorrect "correct" answer that SUBC has a u...
8