LE9.3

Bookmark this page

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?	0×1234	✔ Answer: 0×1234
Value left in R2?	0×1235	✔ Answer: 0×1235
Value left in R3?	0×12	✔ Answer: 0×12
Value left in R4?	0×1235	✔ Answer: 0×1235
Value left in R5?	0xFFFFEDCA	✔ Answer: 0xFFFFEDCA
Value left in R6?	0×0D0A	✔ Answer: 0×00000D0A

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?

1/1 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?

Calculator

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

✔

Explanation
No, there is a SUBC instruction with no ADDC equivalent:

```
SUBC(Rx,0x8000,Ry)
```

The difficulty with the proposed transformation is that when executing an ALUC instruction, the 16-bit constant field 0×8000 will be sign-extended to 0xFFFF8000. The negative of this value is 0×00008000, which, because of the ALUC sign extension, cannot be represented in the 16-bit constant field in the proposed ADDC instruction. So `ADDC(Rx,-0x8000,Ry)` adds 0xFFFF8000 to the value in register Rx, but `SUBC(Rx,0x8000,Ry)` adds 0×00008000.

Submit

ℹ Answers are displayed within the problem

LE9.3.3: ALUC Instruction Encoding?

1/1 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)`?

0b11000000000111111000 ✔ Answer: 0xC03F1234

Explanation
The 32-bit binary encoding is

```
opcode  RC    RA   constant
110000 00001 11111 0001 0010 0011 0100
```

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

0b11101000000000001111 ✔ Answer: 0xE801FFEF

Explanation
17 = 0×00000011. So -17 = ~0×00000011 + 1, where "~" is the bit-wise complement operator. ~0×00000011 + 1 = 0xFFFFFFEE + 1 = 0xFFFFFFEF.
The 32-bit binary encoding is

```
opcode  RC    RA   constant
111010 00000 00001 1111 1111 1110 1111
```

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

0b11011000000001100100 ✔ Answer: 0xD80C9C40

Explanation

Calculator

Explanation
This is a bit of a trick question since the constant decimal 40000 = 0×9C40 is too large to be correctly represented in the 16-bit two's complement constant field. But the assembler will nevertheless place 0×9C40 in the field. But when this instruction is executed, 0×9C40 gets sign-extended to 0xFFFF9c40, which is -25536 in decimal. Oops, probably not what the programmer intended!
The 32-bit binary encoding is

```
opcode  RC    RA   constant
110110 00000 01100 1001 1100 0100 0000
```

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

0b110000000110000100 ✔ Answer: 0xC0610002

Explanation
This is a common typo: using the ADDC mnemonic instead of ADD. Again our assembler dumbly treats the "R2" second operand as shorthand for the number 2, as described in the note above. So it's assembling `ADDC(R1,2,R3)`.
The 32-bit binary encoding is

```
opcode  RC    RA    constant
110000 00011 00001 0000 0000 0000 0010
```

Submit

Answers are displayed within the problem

Discussion

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

Hide Discussion

Add a Post

Show all posts by recent activity

- Google failed me the second time in many years! — 2
  Interesting issue: I use Google search box as a calculator a lot. I was trying to picture SRAC(R2,8,R3) in binary. According to google ...
- Sign Extension — 5
  Why does the 0xFFFF get extended to 0xFFFFFFFF in the first problem for the XORC problem? Is it because the original value store...
- LE 9.3.2

Previous Next

edX

About
Affiliates
edX for Business
Open edX
Careers
News

Calculator

Legal

Terms of Service & Honor Code
Privacy Policy
Accessibility Policy
Trademark Policy
Sitemap
Cookie Policy
Your Privacy Choices

Connect

Idea Hub
Contact Us
Help Center
Security
Media Kit

Calculator