

Computation Structures 2: Computer Architecture







<u>Course</u> <u>Progress</u> <u>Dates</u> <u>Discussion</u>

☆ Course / 9. Designing an Instruction Set / Lecture Videos (52:28)





□ Bookmark this page

■ Calculator

LE9.5.1: Branch Instructions

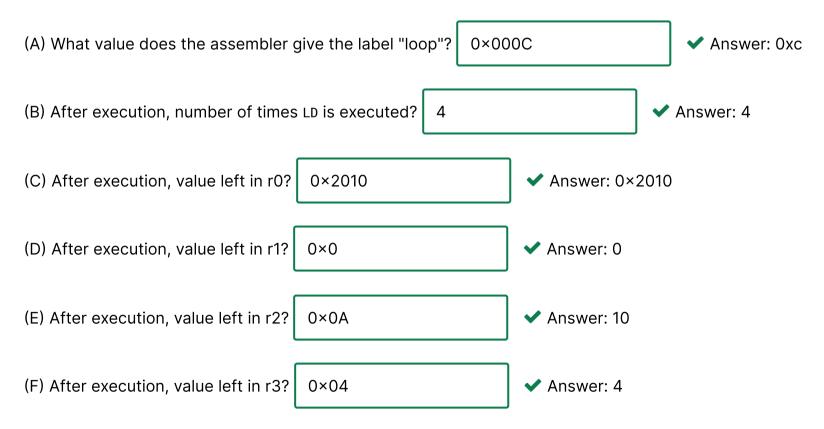
1/1 point (ungraded)

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

Consider the execution of a short program that loops to sum the elements of an array with 4 elements. The first element of the array is stored at location 0×2000.

```
\cdot = 0
                       // first instruction is at location 0
                       // r0 = pointer to next array element
  ADDC(r31,array,r0)
  ADDC(r31,4,r1)
                       // r1 = number of array elements remaining
  ADDC(r31,0,r2)
                       // r2 = accumulated sum
loop:
  LD(r0,0,r3)
                       // load next value from array
  ADD(r3,r2,r2)
                       // add to sum
  ADDC(r0,4,r0)
                       // increment pointer to next word
  SUBC(r1,1,r1)
                      // decrement counter
  BNE(r1,loop,r31)
                       // loop if more elements to go
  ST(r2, result, R31) // write result to memory
  // execution stops here
   = 0 \times 2000 
array:
                       // array[0] = 1
  LONG(1)
  LONG(2)
                       // array[1] = 2
  LONG(3)
                       // array[2] = 3
                       // array[3] = 4
  LONG(4)
result:
  LONG(0)
                       // where result will be stored
```

Program execution starts with the first instruction and halts after execution of the ST instruction.



Explanation

"loop:" labels the fourth instruction of the program and the first instruction is at location 0. So the value of the symbol "loop" is 0xC.

At the end of the loop, the loop counter (R1) is decremented and if it's still non-zero, we branch back for another loop iteration. The initial value for R1 was 4, so the LD instruction is executed 4 times.

The array pointer (R0) is incremented by 4 on each loop iteration. The initial value of the pointer was 0×2000 and there were four loop iterations, so the value left in R0 is 0×2010.

The loop only terminates when the value of the loop counter (R1) is 0. So at the end of execution, the value left in R1 is 0.

R2 is the accumulated sum of the array values = 1 + 2 + 3 + 4 = 10.

R3 is used to hold the value of the array element accessed during each iteration. The last array element accessed had the value 4.

■ Calculator

instructions. Figuring out the value for the 16-bit constant field takes a little more work. The offset value is the number of words between the instruction following the branch (ST in this example) to target instruction (LD in this example). Positive values indicate a forward branch to a subsequent location with a higher address; negative offset values indicate a backward branch to a location with a lower address.

In this example, we'd start counting instructions backwards from the store instruction until we reached the LD instruction. Since it's a backwards branch, we'd encode the count as a negative number in the 16-bit constant field of the BNE instruction.

(G) What is the binary encoding for BNE(r1, loop, r31)?

0b011101111110000111111⁷

✓ Answer: 0×77E1FFFB

Explanation

Here's a diagram showing how the offset of -5 is calculated:

```
instruction
offset
            \cdot = 0
                                 // first instruction is at location 0
            ADDC(r31,array,r0) // r0 = pointer to next array element
                                // r1 = number of array elements remaining
            ADDC(r31,4,r1)
            ADDC(r31,0,r2)
                                 // r2 = accumulated sum
         loop:
-5
            LD(r0,0,r3)
                                // load next value from array
            ADD(r3,r2,r2)
-4
                                // add to sum
-3
            ADDC(r0,4,r0)
                                 // increment pointer to next word
-2
            SUBC(r1,1,r1)
                                // decrement counter
            BNE(r1,loop,r31)
-1
                                // loop if more elements to go
0
            ST(r2, result, R31) // write result to memory
```

The 32-bit binary encoding of the BNE instruction is

```
opcode RC RA constant (= -5)
011101 11111 00001 1111 1111 1011
```

Submit

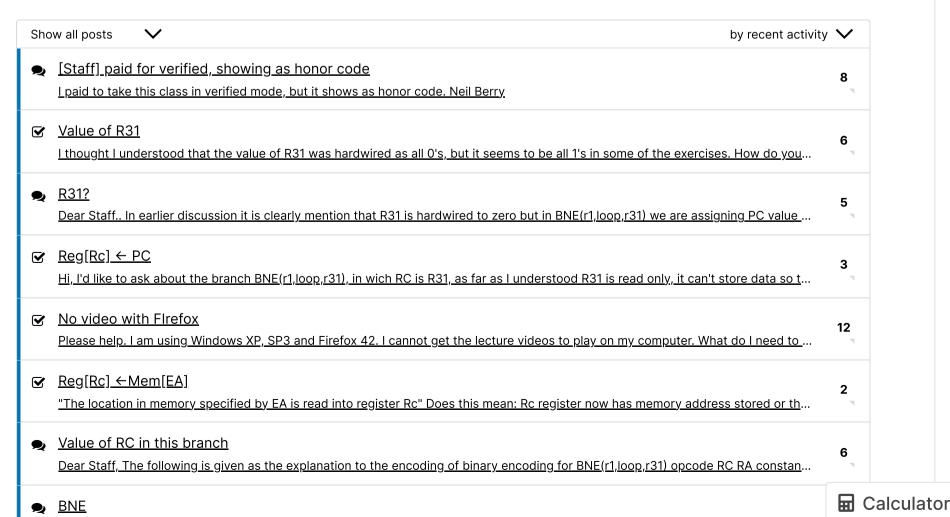
1 Answers are displayed within the problem

Discussion

Hide Discussion

Topic: 9. Designing an Instruction Set / LE9.5

Add a Post



If like me you find the video lectures for *Programmable Machines* too fast to cope with at the first pass, and find the 0.5x slow-do...

Next >

© All Rights Reserved

1



Hey

Hows everyone doing my name is Tim from San Diego California

Previous

edX

About

Affiliates

edX for Business

Open edX

Careers

<u>News</u>

Legal

Terms of Service & Honor Code

Privacy Policy

Accessibility Policy

Trademark Policy

<u>Sitemap</u>

Cookie Policy

Your Privacy Choices

Connect

<u>Idea Hub</u>

Contact Us

Help Center

<u>Security</u>

Media Kit

















© 2024 edX LLC. All rights reserved.

深圳市恒宇博科技有限公司 <u>粤ICP备17044299号-2</u>