# LE9.5

Bookmark this page

## LE9.5.1: Branch Instructions

1.0/1.0 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.

```
  . = 0                 // first instruction is at location 0
  ADDC(r31,array,r0)    // r0 = pointer to next array element
  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

  . = 0x2000
array:
  LONG(1)               // array[0] = 1
  LONG(2)               // array[1] = 2
  LONG(3)               // array[2] = 3
  LONG(4)               // array[3] = 4
result:
  LONG(0)               // where result will be stored
```

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

(A) What value does the assembler give the label "loop"? 0×000C ✔

(B) After execution, number of times `LD` is executed? 4 ✔

(C) After execution, value left in r0? 0×2010 ✔

(D) After execution, value left in r1? 0×0 ✔

(E) After execution, value left in r2? 0×0A ✔

(F) After execution, value left in r3? 0×04 ✔

The encoding for the OPCODE, RC and RA fields of a branch instruction is just like the encodings for other Beta 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)`? 0b01110111111000011111 ✔

Submit

Calculator

## Discussion

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

Hide Discussion

Add a Post

All Posts

### BNE

discussion posted 8 years ago by **yasser-5**

BNE instruction is supposed to work like that

NPC = 0×1c+4 = 0×20

if (REG[R1]!=0)

pc = NPC + 4*offset

if the condition is true, the next address to be executed is to be 0×20 + 4*loop = 0×50

how does it work in this example ?

This post is visible to everyone.

Add a Response

1 response

**rhodesd**

8 years ago - endorsed 8 years ago by **ryan_berg** (Staff)

Here's the instruction docs:

```
Operation:
literal = ((OFFSET(label) –
            OFFSET(current instruction)) ÷ 4) – 1
PC ← PC + 4
EA ← PC + 4*SXT(literal)
TEMP ← Reg[Ra]
Reg[Rc] ← PC
if TEMP ≠ 0 then PC ← EA
```

> The **displacement literal** is treated as a signed word offset. This means it is multiplied by 4 to convert it to a byte offset, sign extended to 32 bits, and added to the updated PC to form the target address.

The intended offset is not from 0 to loop, but from loop to the current instruction

So, I think the calculation is instead:

```
literal = (offset(label) – offset(current instruction)) / 4 – 1
        = (0xc – 0x1c) / 4 – 1
        =  –5
PC ← PC + 4
EA ← PC + 4 * SXT(–5) = 0x20 + 4 * –5 = 12 = 0xc
```

I was having same doubts as @yasser-5, because I was only looking at the slides. When looking at "L09: Programmable Machines, Slide #30" I got the impression that the actual parameter "offset" was passed in as the required offset (in this example the value in 'loop' symbol).

Then I turned to the "BETA Documentation" and realized the difference, which you stated very clearly. Thanks!

posted 8 years ago by **abelinux**

I just wanted to clarify that the literal is actually measuring the distance from the instruction **immediately**

Calculator

**following** the current instruction to loop, not from the actual current instruction.

That's why in this example:

```
          loop:
-5           LD(r0,0,r3)
-4           ADD(r3,r2,r2)
-3           ADDC(r0,4,r0)
-2           SUBC(r1,1,r1)
-1           BNE(r1,loop,r31)
 0           ST(r2,result,R31)
```

the literal = -5 because you begin counting from the ST operation back to loop.

The -1 in this equation:

```
literal = ((OFFSET(label) -
           OFFSET(current instruction)) ÷ 4) - 1
```

is what accounts for that.

posted 8 years ago by **silvinahw** (Staff)

---

```
According to the hint of this problem, the definition of offset is like:
 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

then if offset(label)=0xc, shouldn't offset(current instruction(i.e. BNE ins))=0x10
since there are 4 words in between? I don't quite understand this although I can get the
answer right. Could someone please point out what's wrong?
```

posted 8 years ago by **chi_shawn**

---

Each instruction is 4 bytes wide (1 byte = 8 bits). Addresses are in bytes, so we add 4 bytes between consecutive instructions.

```
addr      instr
0x0C      LD(r0, 0, r3)
0x10      ADD(r3, r2, r2)
0x14      ADDC(r0, 4, r0)
0x18      SUBC(r1, 1, r1)
0x1C      BNE(r1, loop, r31)
0x20      ST(r2, result, r31)
```

the BNE instruction is at address 0×1C.

posted 8 years ago by **silvinahw** (Staff)

---

Add a comment

Showing all responses

Add a response:

Preview

Submit

‹ Previous Next ›

Calculator

## edX

About
Affiliates
edX for Business
Open edX
Careers
News

## 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