

< Previous



Next >

Tutorial: Beta Assembly

 Bookmark this page

 Calculator

For all Beta related questions, you should make use of the [Beta documentation](#), the [Beta Instruction Summary](#), and the [Beta Diagram](#).

Beta Assembly: 1

0.0/1.0 point (ungraded)

1. A beta program contains the line

```
X: LDR(X, R0)
```

After executing this instruction, what is the high-order bit of the value loaded into R0?

High order R0 bit?

Select an option ▾

Answer: 0

Explanation

The `LDR(X, R0)` instruction loads into R0, the contents stored at address X. What is stored at address X is the LDR instruction itself. The most significant bit of the encoding of the LDR instruction is a 0.

2. Memory address **0** contains a **BR** instruction that branches to location **0×1004**. If the 32- bit binary instruction word at 0 is copied unchanged to address 0×5000, what location will the BR at 0×5000 branch to?


BR at 0×5000 will branch to location (HEX): 0x

Answer: 6004

Explanation

A **BR** instruction that branches from address 0 to address 0×1004 would be encoded with a literal that specifies how many words away from PC+4 the destination address is. So if you take that encoded instruction and move it to address 0×5000, that means that the same offset will now be added to the new PC value, so the resulting destination is 0×6004.

Submit

 Answers are displayed within the problem

Beta Assembly: 2

0.0/1.0 point (ungraded)

```
. = 0x100
BEQ(R31, target, R31)
target: ADDC(R31, 0, R31)
```


16-bit offset portion (literal) of above BEQ instruction: 0x

Answer: 0

Explanation

The BEQ instruction stores in its 16-bit literal the distance, in words, from the instruction immediately following the BEQ instruction to the target instruction. In this case, the instruction after the BEQ is also the target of the branch instruction so that distance is 0 words.

Submit

 Answers are displayed within the problem

Beta Assembly: 3

0.0/1.0 point (ungraded)

For each of the Beta instruction sequences shown below, indicate the values of the specified quantities after

 Calculator

sequence has been executed. Consider each sequence separately and assume that execution begins at the beginning of the sequence and halts when the HALT() instruction is about to be executed. Also assume that all registers have been initialized to 0 before execution begins. Remember that even though the Beta loads and stores 32-bit words from memory, all addresses are *byte addresses*, i.e., the addresses of successive words in memory differ by 4. Fill in requested values left after execution of each segment.

1.

```
. = 0x100
BR( .+4 , R0 )
HALT( )
```

Address of BR instruction (HEX): 0x

Answer: 100

Value left in R0 (HEX): 0x

Answer: 104

Explanation

`. = 0x100` specifies that the following instruction is at the specified address. The value stored in R0 is the return address of the branch which is the address of the instruction following the branch instruction, 0x104.

2.

```
. = 0
LD(R31 , c , R0 )
LDR( c , R1 )
ADDC( R0 , b , R0 )
HALT( )

. = 0x200
a:  LONG( 0x100 )
b:  LONG( 0x200 )
c:  LONG( 0x300 )
```

Value left in R0 (HEX): 0x

Answer: 504

Value left in R1 (HEX): 0x

Answer: 300

Value assembler assigns to symbol “c”: 0x

Answer: 208

Explanation

The **LD** instruction loads the contents of address c, 0x300, into register R0. The ADDC then increments R0 by the value b. Since label a is the address 0x200, then label b is 0x204, and label c is 0x208. This means that R0 becomes 0x300 + 0x204 = 0x504.

The **LDR** instruction loads the contents of address c, 0x300, into register R1.

The difference between the **LD** and **LDR** instructions is the way in which they are encoded. In the case of the LD the literal = the constant c which is 0x208. In the case of the LDR the literal = the offset measured in words from the next instruction to the source address of the LD = (c - 8)/4 = 0x200/4 = 0x80.

3.

```
. = 0
LD(R31 , x , R0 )
CMOVE( 0 , r1 )

loop: ANDC( r0 , 1 , r3 )
      ADD( r3 , r1 , r1 )
      SHRC( r0 , 1 , r0 )
      BNE( r0 , loop )
      HALT( )

x:    LONG( 0x123456 )
```

Value left in R0 (HEX): 0x

Answer: 0

Value left in R1 (HEX): 0x

Answer: 9

Value left in R3 (HEX): 0x

Answer: 1

Explanation

This loop counts the number of 1's in the data at address x and stores the result in R1. $0 \times 123456 = 0b000100100011010001010110$, so $R1 = 9$. R0 is originally loaded with the value at address $x = 0 \times 123456$. Each time through the loop, R0 is shifted to the right by 1 thus losing its least significant bit. The program halts once $R0 = 0$. R3 stores the current bit that is being evaluated. The last time through this loop $R0 = 0 \times 1$, so $R3 = 1$, and then after shifting R0 to the right by 1, $R0 = 0$ and the program halts.

Submit

i Answers are displayed within the problem

Beta Assembly: 4

0.0/1.0 point (ungraded)

The Beta executes the assembly programs below starting at location 0 and stopping when it reaches the HALT() instruction. Please give the values requested below after the Beta stops. **Write the values in hex** or write "CAN'T TELL" if the values cannot be determined.

1.

```
. = 0
ADDC(r31, N, r0)
LD(r0, 8, r1)
SRAC(r1, 4, r2)
ST(r2, 4, r0)
HALT()

. = 0x2000
N: LONG(0x12345678)
   LONG(0xDEADBEEF)
   LONG(0xEDEDEDED)
   LONG(0x00000004)
```

Value in R0: 0x

Answer: 2000

Value in R1: 0x

Answer: EDEDEDED

Value in R2: 0x

Answer: FEDEDEDE

Address of location written by ST: 0x

Answer: 2004

Explanation

In this code, label N is address 0×2000 , so the first ADDC instruction sets $R0 = 0 \times 2000$. Since R0 is not written to again, this is the final value of R0.


The LD instruction then loads the contents of address 0×2008 into R1, this value is $0 \times \text{EDEDEDED}$, and R1 is not written to again.

A SRAC (shift right arithmetic) of R1 by 4 positions is then performed and the result is stored in R2. Since the MSB of R1 is a 1, this will shift four 1's into the highest 4 bits, and shift everything else to the right by 4. This results in $R2 = 0 \times \text{FEDEDEDE}$.

Finally, the ST instruction writes the value of R2 into the address determined by adding $R0 + 4 = 0 \times 2004$.

2. Computing the value in R0 is a bit tricky so you might save that until last. But determining the other requested values can be done without figuring out the detailed computation performed by the program.

```
. = 0
ADDC(R31, 0, R0)
LD(R31, test, R1)
loop: CMPL(R31, R1, R2)
      ADD(R2, R0, R0)
```

 Calculator

```
ADD(R2,R0,R0)
SHLC(R1,1,R1)
BNE(R1,loop,R2)
HALT()

. = 0x6004
test: LONG(0xFACE0FF0)
```

Value in R0: 0x Answer: 9

Value in R1: 0x Answer: 0

Value in R2: 0x Answer: 18

Value in location 0x4: 0x Answer: 603F6004

Explanation

This piece of code begins by running ADDC of $R31 + 0 = 0$, so R0 is initially set to 0. Next, a LD of the contents at address test are loaded into R1, so $R1 = 0xFACE0FF0$. Now the loop begins. It first checks if $0 \leq R1$. Since the MSB of R1 is a 1, this returns false, so $R2 = 0$. Next it adds $R2 + R0 = 0 + 0 = 0$ and stores this result back into R0. Next the value that was loaded from memory into R1 is shifted left by 1 position thus making the previous bit 30 now be the MSB. As long as this shift did not make $R1 = 0$, the loop is repeated, and the address immediately following the BNE is stored in R2. Since the code begins at address 0, the address of the HALT() instruction is 0×18 , so R2 is set to 0×18 and then the loop is repeated. The result of repeating this loop is that while $R1 \neq 0$, it counts the number of 0s in the number loaded from memory and that result ends up in R0. R1 ends up shifting all its bits out until it = 0. Each time through the loop, R2 ends up being set to the address of the HALT instruction which is 0×18 . Since the original number that came from memory was $0xFACE0FF0 = 0b1111\ 1010\ 1100\ 1110\ 0000\ 1111\ 1111\ 0000$, after counting the first 9 zeroes, the loop continues shifting out the 8 1's and then it halts. Note that it does not count those last 4 0's because at that point R1 is already equal to 0. So R0 ends up = 9. Finally, the value stored at address 4 is the LD instruction. The opcode for the LD is 011000. This is followed by $Rc = 1$ or 00001 using 5 bits. Then comes $Ra = 31 = 11111$. Finally, the 16 bit constant is the address of test which is 0×6004 . So the value at address 4 is $0 \times 603F6004$.

Submit

Answers are displayed within the problem

Discussion

Hide Discussion

Topic: 10. Assembly Language, Models of Computation / Tutorial: Beta Assembly

Add a Post

Show all posts	▼	by recent activity	▼
<input checked="" type="checkbox"/>	BA 4: correct answer?	I ran the 2nd part of Beta Assembly: 4 in the BSim Sandbox, and found the value of r2 to be different than stated as the correct ans...	7
<input checked="" type="checkbox"/>	B3: ... r3 should be 0 not 1???	0x123456 = 0b **0** 001 0010 0011 0100 0101 0110 **ANDC** does a **bitwise** comparison —→ last bit is **0** not 0x1 Also **...	4

< 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](#)

Connect

- [Blog](#)
- [Contact Us](#)
- [Help Center](#)
- [Media Kit](#)
- [Donate](#)



© 2021 edX Inc. All rights reserved.
深圳市恒宇博科技有限公司 [粤ICP备17044299号-2](#)