

LE12.3.1 Stack Detective

13 points possible (ungraded)

You are given the following incomplete listing of a C program and its translation to Beta assembly code:

```
int ones(int x) {
    int lowbit = x & 1;    // low bit of x
    int rest = x >> 1;    // shift other bits right
    if (x == 0) return 0;
    else return ???;
}
```

```
ones:  PUSH(LP)
        PUSH(BP)
        MOVE(SP, BP)
        ALLOCATE(2)
        PUSH(R1)
        LD(BP, -12, R0)
        ANDC(R0, 1, R1)
        ST(R1, 0, BP)
        SHRC(R0, 1, R1)
        ST(R1, 4, BP)

xx:    BEQ(R0, labl)

zz:    LD(BP, 4, R1)
        PUSH(R1)
yy:    BR(ones, LP)
        DEALLOCATE(1)
        LD(BP, 0, R1)
        ADD(R1, R0, R0)

labl:  POP(R1)

        MOVE(BP, SP)
        POP(BP)
        POP(LP)
        JMP(LP)
```

1. Fill in the binary value of the instruction stored at the location tagged **xx** in the above assembly-language program.

opcode (6 bits): 0b

Answer: 011100

Rc (5 bits): 0b

Answer: 11111

Ra (5 bits): 0b

Answer: 00000

literal (16 bits): 0b

Answer: 0000000000000011

Explanation

The instruction tagged **xx** is **BEQ(R0, labl)**.

The opcode for the **BEQ** instruction is 011100.

Register Rc is R31, or 11111 when encoded using 5 bits.

Register Ra is R0, or 00000.

The literal stores the distance from the instruction following the BEQ to the target instruction measured in words. This distance is 7 instruction words when you take into account that PUSH is actually a macro consisting of 2 instructions.

2. What is the missing C source corresponding to the **???** in the above listing?

☐ ones(rest + lowbit)☐ lowbit☐ ones(rest)☒ ones(rest) + lowbit

Explanation

The code between label **zz** and **labl** is executing the else portion of the if statement. It first loads the contents of BP + 4 into R1. The **ones** portion of the code stored two local variables at BP and BP + 4, namely lowbit, and rest. So rest is loaded into R1. This value is then pushed onto the stack as the next argument x, and the code branches to procedure ones again. This means that a recursive call to ones(rest) is being made. When this call returns its answer is in R0. R1 is

then loaded with the contents of BP + 0 which is lowbit. Then R0 and R1 are added together to produce the final answer in R0. So the missing code is ones(rest) + lowbit.

3. Suppose the instruction bearing the tag **zz** were eliminated from the assembly language program. Would the program continue to work?

☒ Yes

☐ No

Explanation

At label **zz**, R1 already contains the contents of location BP + 4, so there is no need to reload that value into R1.

The procedure **ones** is called from an external procedure and its execution is interrupted just prior to the execution of the instruction tagged **xx**. The contents of a region of memory are shown to the left.

NB: All addresses and data values are shown in hex. The contents of **BP** are 0x1C8, and **SP** contains 0x1D4.

Address in Hex Contents in Hex

184:	4
188:	7
18C:	47
190:	C4
194:	D4
198:	1
19C:	23
1A0:	22
1A4:	23
1A8:	4C
1AC:	198
1B0:	1
1B4:	11
1B8:	23
1BC:	11

1C0: **4C**
 1C4: **1B0**
 BP→1C8: **1**
 1CC: **8**
 1D0: **???**
 SP→1D4: **0**

4. What was the argument to the *most recent* call to **ones**?

Most recent argument (HEX): x = 0x

Answer: 11

Explanation

The remaining questions are easier to answer if we label the stack with the stack frame elements that each location represents.

Address in Hex Contents in Hex

184:	4	
188:	7	
18C:	47	x
190:	C4	LP
194:	D4	BP
198:	1	lowbit
19C:	23	rest
1A0:	22	R1
1A4:	23	x
1A8:	4C	LP
1AC:	198	BP
1B0:	1	lowbit
1B4:	11	rest
1B8:	23	R1
1BC:	11	x
1C0:	4C	LP
1C4:	1B0	BP
BP→1C8:	1	lowbit
1CC:	8	rest
1D0:	???	R1
SP→1D4:	0	

The argument for the most recent call to **ones** is located at $BP - 12 = 0 \times 1BC$. So the most recent argument is $x = 0 \times 11$.

5. What is the missing value marked **???** for the contents of location 1D0?

Contents of 1D0 (HEX): 0x

Answer: 11

Explanation

Location $0 \times 1D0$ stores the last value of R1. R1 holds the argument x immediately before making the recursive call to **ones**, so $R1 = 0 \times 11$.

6. What is the hex address of the instruction tagged **xx**?

Address of xx (HEX): xx = 0x

Answer: 38

Explanation

The second and third instances of the stored $LP = 0 \times 4C$ are from the recursive calls to **ones** so $0 \times 4C$ is the address of the DEALLOCATE(1) instruction. Since PUSH is a macro consisting of 2 instructions, label **xx** is 5 words before the DEALLOCATE. The address is 0×38 .

7. What was the argument of the *original* call to **ones**?

Original argument (HEX): x = 0x

Answer: 47

Explanation

The original call to **ones** is the one whose LP value is different, namely $LP = 0 \times C4$. The argument for that call is immediately before the stored LP so it is 0×47 .

8. What is the hex address of the BR instruction that called ones originally?

Address of original call (HEX): 0x

Answer: C0

Explanation

The original stored LP points to the instruction immediately following the original BR instruction. Since that $LP = 0 \times C4$, the branch instruction was at $0 \times C0$.

9. What were the contents of R1 at the time of the original call?

Original R1 contents (HEX): R1 = 0x

Answer: 22

Explanation

The original R1 is stored after the two local variables in the stack frame, so it can be found at BP + 8 in the top stack frame. This is location 0x1A0 so the original contents of R1 = 0x22.

10. What value will be returned to the original caller?

Return value for original call (HEX): 0x

Answer: 4

Explanation

The **ones** procedure is counting the number of ones in the input argument which is $0 \times 47 = 0b01000111$. The number of ones in this number is 4.

Submit

i Answers are displayed within the problem


Discussion


Hide Discussion

Topic: 12. Procedures and Stacks / LE12.3

Add a Post

Show all posts 

by recent activity 

- | | | |
|---|---|---|
| <input checked="" type="checkbox"/> | Question C | 2 |
| | I was going to say that I think the answer to the answer to this question could also be no. But then... | |
|  | A [literal (16bit)]. | 3 |
| | I don't understand the answer of A last part. The way I calculate the literal is by finding the number... | |
| <input checked="" type="checkbox"/> | 12.3.1 Question | 5 |
| | Sorry, I'm a little behind and catching up (too many MOOCs :) In the explanation for this problem, t... | |