Video explanation of solution is provided below the problem.

Procedures and Stacks

11 points possible (ungraded)

You've been commissioned by a government agency to reverse-engineer a mysterious procedure found on the disk of a Beta system used by a cyber-terrorist cell. You've given an incomplete copy of the C source language for the function **f** (shown below), as well as its complete translation to Beta assembly code:

```
// Mystery function:
int f(int x) {
  int a = x&5;  // bitwise AND

  if (x == 0) return 0;
  else return ?????;

}
```

```
f:    PUSH(LP)
      PUSH(BP)
      MOVE(SP, BP)
      ALLOCATE(1)
      PUSH(R1)

      LD(BP, -12, R0)
      ANDC(R0, 5, R1)
      ST(R1, 0, BP)

xx:   BEQ(R0, bye)

      SUBC(R0, 1, R0)
      PUSH(R0)
yy:   BR(f, LP)
      DEALLOCATE(1)

      LD(BP, 0, R1)
      ADD(R1, R0, R0)

bye:  POP(R1)
      MOVE(BP,SP)
      POP(BP)
      POP(LP)
      JMP(LP)
```

1. What is the missing expression shown in the C code as **"?????"**

- ◯ f(x)
- ◯ f(x-1)
- ◯ a + f(x+1)
- ◯ a + f(x)
- ◯ a + f(x-1) ✔

Explanation

Just above label **xx**, x is loaded into R0, a is calculated in R1 and stored locally on the stack. Just below label **xx** R0 is decremented by 1 and then pushed on the stack as the argument to the recursive call to function **f**. So we are calling f(x-1). Then just before label **bye** the value of a is loaded into R1 and added to R0 which contains the result of our call to f(x-1). So the missing code is a + f(x-1).

2. Is the value of the local variable **a** stored in the stack frame of the Beta program? If so, give its offset relative to the contents of **BP**; otherwise, write "None":

Offset of a, or None:

Answer: 0

Explanation

Local variable **a** is stored at location BP + 0 by the ST instruction.

3. Give the 32-bit binary translation of the BR instruction tagged **yy**

opcode (6 bits): 0b

Answer: 011100

Rc (5 bits): 0b

Answer: 11100

Ra (5 bits): 0b

Answer: 11111

literal (16 bits): 0b

Answer: 1111111111110000

Explanation

The opcode for the **BR(f,LP)** instruction which is equivalent to a **BEQ(R31,f,LP)** instruction is 011100.

Rc = LP = R28 = 0b11100.

Ra = R31 = 0b11111.

The literal in the branch instruction encodes the distance between PC + 4 and the destination address in words. Here PC + 4 is the address of the DEALLOCATE(1) instruction, and the destination address is that of label f. Recall that the PUSH and POP are macros, each of which actually represents 2 instructions. Taking this into consideration, we see that the instruction at label f is 16 instructions before the instruction at label **yy**, so the literal = -16 = 0b1111111111110000.

The function **f** is called from an external main program, and the machine is halted when a recursive call to **f** is about to execute the **BEQ** instruction tagged **xx**. The BP register of the halted machine contains **0×174**, and the hex contents of a region of memory location are shown below.

13C:	7
140:	7
144:	$5C$
148:	$D4$
14C:	5
150:	3
154:	6
158:	$A4$
15C	$14C$
160	4
164	5
168	5
16C	$A4$
170	160
$BP$174: $\rightarrow$	5
178	4

4. What is the value in **SP**?

HEX contents of SP: 0x

17C

Answer: 17C

Explanation

When the code reaches label **xx**, two words have been added to the stack after the BP was set. So the SP is BP + 8 = 0×17C.

5. What is the value stored in the local variable **a** in the current stack frame?

HEX value of a: 0x

5

Answer: 5

Explanation

The easiest way to answer the following questions is to label the stack trace.

13C:	7	
140:	7	x
144:	$5C$	LP
148:	$D4$	BP
14C:	5	a
150:	3	R1
154:	6	x
158:	$A4$	LP
15C	$14C$	BP
160	4	a
164	5	R1
168	5	x
16C	$A4$	LP
170	160	BP
$BP$174: $\rightarrow$	5	a
178	4	R1

From the labeled stack frame, we see that the value of local variable **a** in the current stack frame is 5.

6. What is the address of the **BR** instruction that made the original call to **f** from the external main program?

Address of BR for original call: 0x

Answer: 58

Explanation

The stack trace show 2 different values of LP, 0xA4 and 0×5C. 0xA4 corresponds to the recursive calls to **f** whereas 0×5C is the return address of the original call to **f**. This means that the actual **BR** instruction was at address 0×58.

7. What value is currently in the PC?

HEX contents of PC: 0x

90

Answer: 90

Explanation

The PC is at label **xx**. We know that the DEALLOCATE(1) instruction corresponds to the 0xA4 saved LP value. Since the PUSH macro counts as 2 instruction words, that means that label **xx** is 5 words before this, so the PC = 0×90.

The summer intern working for you, after looking at the assembly code for f, argues that the cyber- terrorist group that wrote this code isn't very clever. He argues that one could simply delete four instructions from the assembly-language program -- a **LD**, a **ST**, an **ALLOCATE**, and a **MOVE** -- and the program would continue to work as before (but faster and using less space).

8. Is he right? Can one in fact delete four lines of code as described and still have a working program?

Can one optimize by deleting 4 such lines?

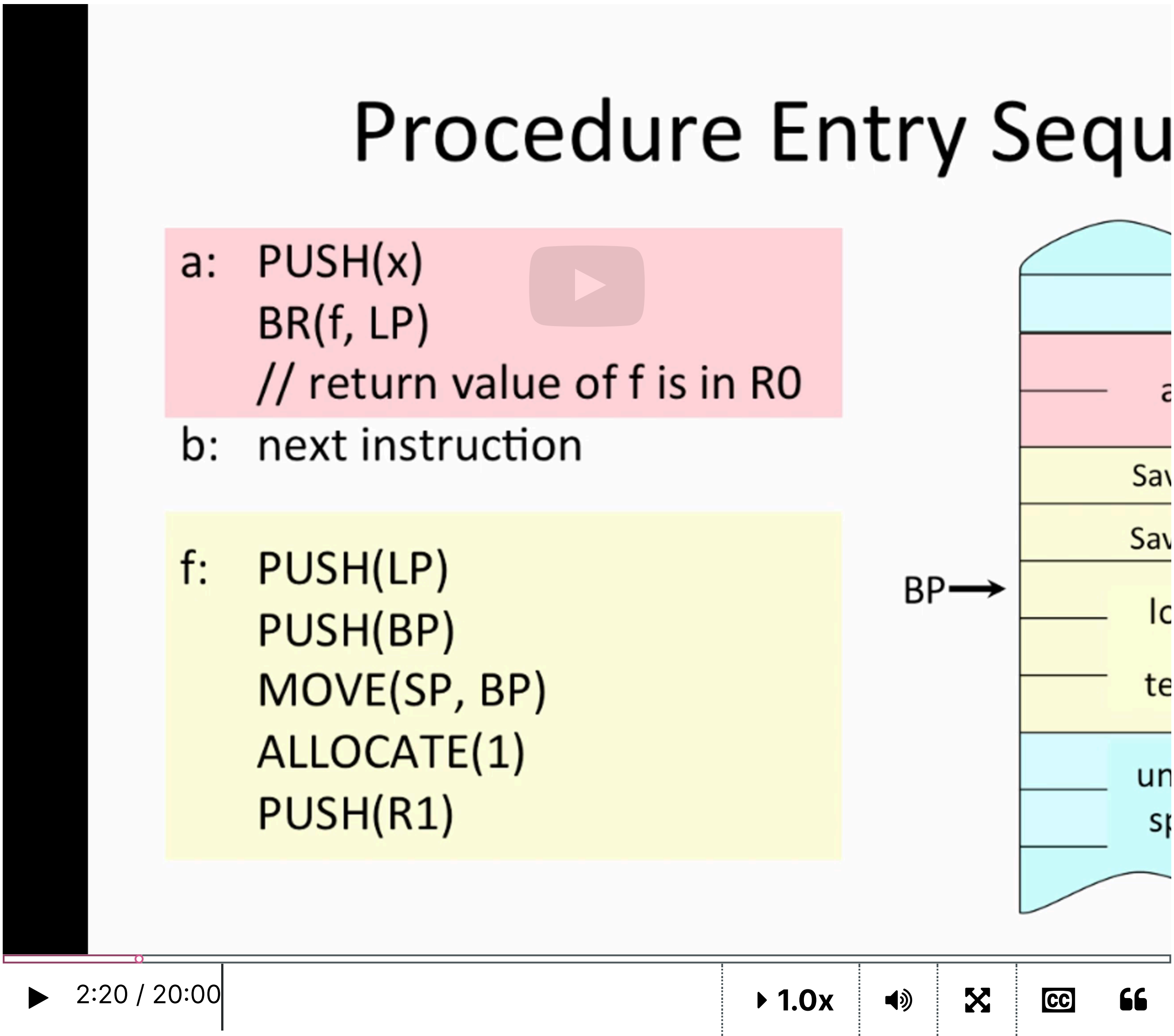
- (selected) YES ✔
- NO

Explanation

Register R1 is assigned the value of variable **a**, and since R1 is saved on the stack, there is no need to also save **a** as a separate local variable. This would allow the ALLOCATE(1) which allocates space for **a** to be removed, then the ST(R1, 0, BP) can be removed since **a** no longer needs to be stored. Similarly the LD(BP, 0, R1) is no longer needed to load R1 with variable **a** since the value is now restored into R1 by popping R1. Finally, the MOVE(BP,SP) instruction can be removed because once there is no local variable, the POP(R1) instruction already makes SP = BP.

Submit

Procedures and Stacks



Video

Download video file

Transcripts

Download SubRip (.srt) file

Download Text (.txt) file

Discussion

Hide Discussion

Topic: 12. Procedures and Stacks / WE12.1

Add a Post

Show all posts by recent activity

- given stack trace? 4
- WE12 - Question H - "Should" one optimize by deleting 4 instructions?

On question H we get to realize that using all steps from procedures "rituals" leads to sub-opti... 3

Bp n Sp Pointers

To correct recursion problem overwriting our reserved registers in procedure linkage, we use a... 5