CS2011 (Intro to computer systems)

Homework 2 - Machine-level Representation of programs

Thavishka Gamage
M14304830
10/23/2022

(A)

1) Code Snippet 1

1. Since %ax contains 16 bits & addw deals in 16 bits, or two bytes, %ax should be holding 0xDAA0 at the end of the snippet.
2. Signed Flag is 1, Overflow Flag is 1

2) Code Snippet 2

1. We have 0x0FD7 at the very end of rax because we are shifting the value to the eax, which is the bottom half of rax, and addb in this case adds to the last 8 bits
2. Given that there is no overflow and t>0, none of the flags ought to be set.

3) Code snippet 3

1. The remaining 16 bits of EAX, 3A are subtracted from 60, which results in overflow because the register is signed. OD
2. CF set given the need for carrying.

4) Code Snippet 4

1. -0x79 because of deducting 58 from -0x47 to get a negative numbe
2. Since it was signed, SF is set

5) Code Snippet 5

1. 0x80ff because incb increases bits 8 through 15 to get the number 00
2. Since incb takes this into consideration, none of the flags are set

6) Code snippet 6

1. 0xFF0C65 because imulw is signed multiplication and the original Integer.
2. SF is set

7) Code snippet 7

1. The final 16 bits are right shifted by 15 bits, leaving the value 0xFFFF0000
2. SF since the code starts with an F

8) Code snippet 8

1. Because the lowest 8 bits are right-shifted by 5 and leave 30 in al, the value is 0xFFFF0030.
2. No flags

9) Code snippet 9

1. As the & negates the lowest 16 bits, the value is 0xFF0000.
2. No flags

10) Code Snippet 10

1. 0xFFFFFFFFFFFF00FE because after the right shift and leaw
2. No Flags.

(B)

A 1. Code Snippet 1

1. The -16 constant is moved 5 blocks away from %.rbp in the first line. The following code then stores this value to %eax after padding it with 0s at the beginning. After that, the third line moves this location four blocks distant from %rbp.
2. 0xF
3. Char, -16, int, int since movb only moves 8 bits, the size of char

2. Code Snippet 2

1. We relocate 0x88 to a location three blocks away from rbp. Then this is transferred to the 16-bit value %ax, where it is lengthened to fit. Then another block is used for this.
2. 0x88
3. Char, 0x88, short, short since %ax is 16 bits or 2 bytes big

3. Code snippet 3

1. We transferred 0x7FFF to a block 6 distanced from rbp. The value that was previously stored in the preceding register is subsequently transferred to rax and zero-padded there. Afterward, this is transferred to Block 4 away from RBP.
2. 0x7FFF
3. Short, 0x7FFFF, int, int

4. code snippet 4

1. We relocate 0x9C to a block 9 that is not near rbp. The value that was previously stored in the preceding register is subsequently transferred to rax and zero-padded there. Then, it is transferred to a block 8 distant from the rbp.
2. 0x9C
3. Char, 0x9C, long, long

5. Code Shippet

1. We relocate -2 into a block 8 distance from the rbp. The value kept in the preceding register is then transferred to eax. Afterward, this is transferred to Block 4 away from RBP.
2. -2
3. Int, -2, int, int

C. A

1. Result is rax, x is edx, n is cl, mask is rdx
2. Result = 0, mask = 1
3. Mask != 0
4. Mask <<= n
5. long loop(long x, long n) {
   long result = 0;
   long mask;
   for (mask = 1; mask != 0; mask <<= n) {
   result |= (x & mask);
   }
   return result;