Lab 10

Due Friday, November 5, 2021, Midnight

Submission: Canvas

Answer the following questions. Your answer must be in RED. YOU MAY WRITE YOUR ANSWER. IF YOUR ANSWER IS WRITTEN THE ANSWERS MUST BE READABLE OR THE TA WILL MARK THEM INCORRECT.

ROB HUGHES C11411208

1. What size data does the following assembly instruction referring to:
   1. pushq \_\_\_\_8 byte\_\_\_\_\_\_
   2. movl \_\_\_\_\_2 bytes \_\_\_\_\_
   3. movw \_\_\_\_\_1 byte \_\_\_\_\_
   4. movb \_\_\_1 byte \_\_\_\_\_
2. Based on the chart in Figure 3.2 in the notes, what are the sizes of the following registers? You can find a chart in the notes to help you with this.
   1. %edi \_\_\_\_ 4 bytes \_\_\_\_\_
   2. %rcx \_\_\_\_ 8 bytes \_\_\_\_\_
   3. %di \_\_\_\_2 bytes \_\_\_\_\_
   4. %sil \_\_\_\_ 1 byte \_\_\_\_\_
   5. %bp\_\_\_\_ 2 bytes \_\_\_\_\_
   6. %al \_\_\_\_\_ 1 byte \_\_\_\_
   7. %rbx \_\_\_ 8 bytes \_\_\_\_\_
3. Complete the following

|  |  |  |  |
| --- | --- | --- | --- |
| Address | Value | Register | Value |
| 0x100 | 0xAB | %rbx | 0x104 |
| 0x104 | 0xFA | %rcx | 0x3 |
| 0x108 | 0x1B | %rdx | 0x2 |
| 0x10F | 0xC1 | %rax | 0x1 |

Fill in the following table showing the values for the indicated operands. Explain how you came up with the value:

|  |  |  |
| --- | --- | --- |
| Operand | Value | Explain how you arrived at your answer |
| %rbx | 0x104 | Value at register %rbx |
| 0x108 | 0x1B | Value at address 0x108 |
| $0x100 | 0x100 | Constant 0x100 |
| (%rbx) | 0xFA | value of the address stored at the address %rbx |
| 4(%rbx) | 0xBB | value at address at %rbx +4 |
| 9(%rbx, %rdx) | 0xC1 | value at address %rbx + 9 + %rdx |
| 260(%rcx, %rax) | 0x1B | value at address %rcx + 260 + %rax |
| 0xFC(,%rcx,4) | 0x1B | value at address %rcx\*4 + 0xFC |
| (%rbx, %rax, 4) | 0x1B | value at address %rax\*4 + %rbx |

1. For each of the following values of K, find ways to express x\*K using only the specified number of operations (shifts and add/subs).

|  |  |  |  |
| --- | --- | --- | --- |
| K | Shifts | Add/Subs | Expression |
| 18 | 2 | 1 | (x<<4) + (x<<1) |
| 42 | 3 | 2 | (x<<5) + (x<<3) + (x<<1) |
| 125 | 2 | 2 | (x<<7) –(x<<1) - x |
| -14 | 2 | 1 | (x<<1)- (x<<4) |

1. Assume variables **sp** and **dp** are declared with types:

src\_t \*sp;

dest\_t \*dp;

Where src\_t and dest\_t are data types declared with typedef. We wish to use the appropriate pair of data movement instructions to implement the operation

\*dp = (dest\_t) \*sp;

Assume that the values of **sp** and **dp** are stored in registers **%rdi** and **%rsi**, respectively. For each entry in the table, show the two instructions that implement the specified data movement. The first instruction in the sequence should read from memory, do the appropriate conversion, and set the appropriate portion of register %rax. The second instruction should then write the appropriate portion of %rax to memory. In both cases, the portions may be %rax, %eax, %ax, or %al, and they may differ from one another.

Recall that when performing a cast that involves both a size change and a change of “signedness” in C, the operation should change the size first. Section 2.2.6 in the book.

src\_t dest\_t Instruction

long long movq (%rdi), %rax

movq %rax, (%rsi)

char int movsbl (%rdi), %eax

movl %eax, (%rsi)

char unsigned int movsbl (%rdi), %eax

movl %eax, (%rsi)

unsigned char long mobzbq (%rdi), %rax

movl %rax, (%rsi)

int char movl (%rdi), %eax

movb %al, (%rsi)

unsigned int unsigned char movl (%rdi), %eax

movb %al, (%rsi)

char short movsbw (%rdi), %ax

movw %ax, (%rsi)

1. Fill in the chart below. This chart represents the sizes of C data types in x86-64.

|  |  |  |  |
| --- | --- | --- | --- |
| C Declaration | Intel Data Type | Assembly-code suffix | Size (bytes) |
| char | byte | b | 1 |
| short | word | w | 2 |
| int | double word | l | 4 |
| long | quad word | q | 8 |
| char\* | quad word | q | 8 |
| float | single prec | s | 4 |
| double | double prec | d | 8 |

1. There are 5 possible combinations of source and destination types for the move instruction set. List the five and give an example of each.
   1. Imm-reg: movl $0x4050, %eax
   2. Reg-reg: movw %bp, %sp
   3. Mem-reg: movb (%rdi, %rcx), %al
   4. Imm-mem: movb $-17, (%esp)
   5. Reg-mem: movq %rax, -12(%rbp)
2. \_\_\_Move\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ is one of the most used instruction for x86-64.
3. X86-64, has 4 special-purpose registers. What are they? How many of these are also considered general-purpose registers? What are they used for?

%rsp- gen purpose as well- used to call stack pointer

%rbp- gen purpose as well- used to call base pointer

%rip- used to call instruction program

%rflags- used to call flags and condition codes

1. What is the register used for the return value of a function?

%rax

1. How many of the general-purpose registers are used for function parameters? List these registers.

6. %rdi, %rsi, %rdx, %rcx, %r8, %r9

1. What happens when a function has more parameters than the registers set aside for them (where are the excess parameters stored)?

They are passed to the stack.

1. In class we discussed 4 things that are visible to machine code but hidden from the programmer. Using a couple sentences each, discuss the 4 things.

PC- Referred to as %rip. This hold the address of the instruction that is next up to be executed.

Int Register File- These are 16 locations that hold 64 bit values. They keep track of the state of the program, hold temp data, and are used for things such as C pointers.

Condition code registers- these registers hold data referring to the most recent arithmetic or logical instructions, such as a while loop’s condition.

Vector Registers- These can hold multiple integer values as well as floating point values.

1. In the notes we discussed the assembly for the following code. Write the assembly and explain what each line of the assembly is doing. (Hint: This is in the slide.) Make sure you convince the TA you understand what the lines of code are doing.

**void multstore(long x, long y, long\* dest)**

**x is stored in %rdi, y in %rsi, dest in %rdx**

**void multstore(long x, long y, long \*dest){**

**long t = mult2(x,y);**

**\*dest = t;**

**}**

multstore

pushq %rbx Saves %rbx

movq %rdx, %rbx Copies long \* dest to %rbx

call mult2 Calls function mult2(x,y)

movq %rax, (%rbx) Stores result at %rbx(\*dest)

popq %rbx Pops stack of %rbx back to pre call value

ret Return statement

1. What is a calling convention?

A calling convention manages how functions on a particular architecture and operating system act.  They make sure functions compiled by different compilers can interoperate, and they ensure that operating systems can run code from different programming languages and compilers.

1. With respect to a calling convention:
2. What is a caller?

A function that calls on another one.

1. What is a Callee?

The function called.

1. Steps required to call a function are sometimes called entry sequence.
2. Steps required to return a from a function are called exit sequence.
3. With respect to the entry and exit sequences the callee places the return value in the appropriate register.
4. What is the following assembly code doing? xorl %eax, %eax:

Making %eax zero.

1. What is the following assembly code doing? movl $10, -4(%rbp)

Taking the immediate 10 and moving it to the address of %rbp-4.

1. X86-64 has three operand specifiers use when specifying the source values to use in performing an operation and destination location into which to place the result. Name and describe the three:
2. Immediate: constants expressed by a leading $ using an integer or anything that can be equaled to one.
3. Register: enforces the content of the register ra - “a” making an arbitrary register, R[ra] indicates its value.
4. Memory: accesses memory according to what address is computed.