Homework

**Due: Saturday, April 10, 2021, midnight**

Submit through Canvas

This homework is designed to help you study for the 3rd exam. Complete the problems below. Show all work. Your answers must be in RED. Many of these come directly from the slides. However, if you simply copy them, you will not learn anything and will not do well on the exam.

1. Show the steps needed to multiply the following binary numbers:

1002(M = multiplicand) \* 1012(Q = multiplier)

Step 0 Initialize the data M = 100 C = 0 Acc = 0 Q = 101

Step 1:

C ACC MQ

0 000 101

+ 100 (LSB=1)

-------------------------------------------------

0 100 101

>>

0 010 010

-------------------------------------------------

Step 2:

C ACC MQ

0 010 010

+ 000 (LSB=0)

-------------------------------------------------

0 010 010

>>

0 001 001

-------------------------------------------------

Step 3:

C ACC MQ

0 001 001

+ 100 (LSB=1)

-------------------------------------------------

0 101 001

>>

0 010 100 (ANSWER)

-------------------------------------------------

There are three steps because the sizes of the numbers are 3 bits. Thus:

1002(M = multiplicand) \* 1012(Q = multiplier) = 101002

1. With respect to dividing by powers 2’s complement using right shifting, it is common for rounding errors to occur, which can accumulate causing larger problems. In class, we discussed how “biasing” the value before shifting can correct this problem. Given -12340 = 1100111111001100 complete the chart below. This is the chart directly from the slides. Yes, you can simply copy the answers. However, to help you grasp this I strongly suggest you make sure you understand what this is doing. Such as, what is the bias and how was it determined. Why am I adding the bias? Etc.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **K** | **Bias** | **-12340 + bias (binary)** | **>> K (binary)** | **Decimal** | **12,340/2k** |
| 0 | 0 | 1100111111001100 | 1100111111001100 | -12340 | -12340.0 |
| 1 | 1 | 1100111111001101 | 1110011111100110 | -6170 | -6170.0 |
| 4 | 15 | 1100111111011011 | 1111110011111101 | -771 | -771.25 |
| 8 | 255 | 1101000011001011 | 1111111111010000 | -48 | -48.203125 |

1. Determine the fractional representation of the following positional binary representations. These are not IEEE.

Show your work.

Again, this is directly off of the slides, you must show your work. If you do not you will not get credit.

|  |  |  |
| --- | --- | --- |
| **Binary representation** | **Value** | **Decimal** |
| 0.0 | 0/2 | 0.0 |
| 0.01 | 1/4 | 0.25 |
| 0.010 | 2/8 | 0.25 |
| 0.0011  Accuracy = 2^4 = 16  Fraction value = 0011 = 3 | 3/16 | 0.1875 |
| 0.00110  Accuracy = 2^5 = 32  Fraction value = 00110 = 6 | 6/32 | 0.1875 |
| 0.001101  Accuracy = 2^6 = 64  Fraction value = 001101 = 13 | 13/64 | 0.203125 |
| 0.0011010  Accuracy = 2^7 = 128  Fraction value = 0011010 = 26 | 26/128 | 0.203125 |
| 0.00110011 | 51/256 | 0.19921875 |

1. Using positional notation for fractional binary numbers convert the following binary number to decimal. You must show your work.

1010.101 This is not IEEE. There are examples of this on the slides.

10102 = 10

.1012 = 5/8 = .625

1010.1012 = 10102 + 0.1012 = 10 + 0.625 = 10.625

1. For both single and double precision, name the three sets of bit(s) that make up the IEEE 754 floating point number. Also list the number of bits required for each of the three parts that make up the single and double precision.

Single precision: s = 1 (sign set), k = 8 (exponent set), n = 23 (fraction/mantissa set)

Double precision: s = 1 (sign set), k = 11 (exponent set), n = 52 (fraction/mantissa set)

1. An IEEE Floating-Point value encoded by a given bit representation can be divided into three different cases with the third having two variants depending on the value of the mantissa (frac). Name and describe each case.

Normalized: This is the normal case where the exponents are neither all 0 nor all 1. It describes numbers not close to 0 and not infinity, and the M = 1 + *f* (implied leading 1).

De-normalized: The case when all of the bits in the exponent are 0. This is for numbers that are close to 0, including positive and negative 0 themselves. M = *f* (without leading)

Special case: If all the bits in the exponent are 1, then there are two special cases. If the rest of the bits in the fraction are all 0s, then the number represents either positive or negative infinity (depending on the sign bit). Else the number is NaN – undefined.

C – Review

1. What is the purpose of the C function memcpy?

Copies a set number of bytes from source to destination.

1. What is the purpose of the C function memset?

Fills memory byte-by-byte with a particular (int) value.

1. What is the purpose of the C function memcmp?

Compares two values byte-by-byte to determine if they are equal, less than, or greater.

Chapter 3

1. Describe what the compile flag -Og does.

“Tells the compiler to apply a level of optimization that yields machine code that follows the overall structure of the original C code (slide 10).” This would be useful to view hidden aspects of the code, visible only via machine code representation (such as the program counter, integer register files, etc.)

1. Describe the significance of the compile flag -S.

Tells the compiler to not assemble; stop after compilation proper.

1. Describe the significance of the compile flag -c.

This tells the compiler to assemble the source files, but to not link. The significance is that the files become object files (.o) without the executable. This might be useful to inspect machine-code before the linking phase.

1. If I want to be able to understand the content of an “.o” file I can use a disassembler. We discussed two disassemblers in class. Only one of them can be used with linux and Mac OS. What command would I use to view the content of a “.o” file on a linux architecture.

objdump -d ---.o, using the program objdump to disassemble

1. Much of the success of computers in the last 60+ years has been due to the use of transistors.
2. How many general-purpose registers does x86-64 have? 16
3. X86-64, also have 4 special-purpose registers. What are they? How many of these are also considered general-purpose registers? %rsp – stack pointer, %rbp – base pointer, %rip – instruction pointer (program counter), %rflags – flags and condition codes. The %rbp is also considered a general-purpose register.
4. What is the register used for the return value of a function?

The return value is placed in %rax

1. How many of the general-purpose registers are used for function parameters? List these registers. The six registers %rdi, %rsi, %rdx, %rcx, and %r8 are used for the first six function parameters. More are passed to the stack.
2. 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. The 7th argument is stored at %rsp (the top of the stack), and the stack frame for the call will contain the extra arguments in address order.
3. What general purpose-register is use as the stack pointer? %rsp
4. What special-purpose register is use as the program counter also known as the instruction pointer? %rip
5. We discussed three type of operand specifiers used as source values when performing many assembly operations. Name the three and give an example of each.

Immediate - $100 ($ followed by an integer or integer-representing number)

Register- R[ra] / %rax (sort of like an array, accessing R dereferences a specific register)

Memory- 0x104 (use a computed address, like with pointers)

1. Similar to what we discussed in class, fill in the table showing the values for the indicated operands.

|  |  |  |  |
| --- | --- | --- | --- |
| **Address** | **Value** | **Register** | **Value** |
| 0x100 | 0xFF | %rax | 0x100 |
| 0x104 | 0xAB | %rcx | 0x1 |
| 0x108 | 0x13 | %rdx | 0x2 |
| 0x10C | 0x11 |  |  |

|  |  |
| --- | --- |
| **Operand** | **Value** |
| %rdx | 0x100 |
| 0x108 | 0xAB |
| $0x10C | 0x10C |
| (%rax) | 0xFF |
| 8(%rax) | 0x13 |
| 10(%rax,%rdx) | 0x11 |
| 257(%rcx,%rdx) | 0xAB |
| 0xFC(,%rcx,8) | 0xAB |
| (%rax,%rdx,4) | 0x13 |

1. Move is one of the most used instruction for x86-64.
2. 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 precision | s | 4 |
| double | Double precision | l | 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. Immediate—Register movl $100, %rax
   2. Register—Register movw %rax, %rip
   3. Memory—Register movb (%rdi, %rcx), %rbp
   4. Immediate—Memory movb $0x2030, (%rax)
   5. Register—Memory movq %bp, (%rbp)

Cannot do Memory—Memory