CS1216 - Monsoon 2022 - Homework 4

Jivansh Sharma
UG 24 1020211193

22/10/2022

Collaborators: None

1. Show how the value 0xabcdef12 would be arranged in memory of a little-endian and a big-endian machine. Assume the data is stored starting at address 0.

Big Endian Machines store the most significant byte at the lowest address. The first byte read goes in the big end of the register.
So, the value 0xabcdef12 would be stored as

```
most significant bit | ab   cd  ef  12 | least significant bit
```

Little Endian Machines store the least significant byte at the lowest address. The first byte read goes in the least significant bit of register.
So, the value 0xabcdef12 would be stored as

```
most significant bit | 12   ef  cd  ab | least significant bit
```

2. Provide a minimal set of MIPS instructions that may be used to implement the following pseudoinstruction

```
not $s0, $s2      // bit-wise invert

# we know that not (s2) is to be stored in s0
nor $s0, $s2, $zero # $s0 = not ($s2)
```

3.The IEEE 754 2008 version contains a half precision format that is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and has a bias of 15, and the mantissa is 10 bits long. A hidden 1 is assumed. Write down the bit pattern to represent -1.5635 x 10-2 assuming a version of this format, which uses an excess-16 format to store the exponent. Comment on how the range and accuracy of this 16-bit floating point format compares to the single precision IEEE 754 standard.

Converting to base 2 scientific notation, we get $-1.00000000001010011111001 * 2^{-}6$.

So, sign = 1 (negative number, 1 Bit)
exponent = 9 (-6 = 9 - 15; 15 being bias, 5 Bits)
mantissa = 0.00000000001010011111001 (10 Bits)

Therfore the 16 Bit representation is as follows:

```
1    10010   00000000001
Sign Exponent Mantissa
```

The range and accuracy of this 16-bit floating point format are less than the single precision IEEE 754 standard. The range is less because the exponent is only 5 bits long as compared to 8 bits in the single precision IEEE 754 standard. The accuracy is less because the mantissa is only 10 bits long as compared to 23 bits in the single precision IEEE 754 standard.

4. Let us assume that the program counter (PC), while executing a program is currently set to 0x2000 0000. Explain your answer.

We know, PC is set to 0x20000000. Converting this hex values to binary, we get `0010 0000 0000 0000 0000 0000 0000 0000`.

(a) Is it possible to use the jump (j) MIPS assembly instruction to set the PC to the address to a different value, say 0x5000 0000?

We know the jump instruction is of the form `j target`. The instruction takes a 6-bit OPCODE and a 26-bit target address / offset.

In order to convert the 26-bit target address to a 32-bit address, we need to append the target address by 2 bits to the left. Then 4 most significant bits of PC are added the right side of the target address. This forms a 32-bit address.

We already know the MSB of 0x20000000 would be `0010`, making the target address of the from 0x2XXXXXXX, making it impossible for us to jump to 0x5XXXXXXX.

So, it is not possible to use the jump instruction to set the PC from 0x20000000 to 0x50000000.

(b) Is it possible to use the branch (beq) MIPS assembly instruction to set the PC to the address to a different value, say 0x5000 0000?

We know the beq instruction is of the form `beq rs, rt, offset`. The instruction takes a 6-bit OPCODE, 5-bit rs, 5-bit rt and a 16-bit offset.

The target address 0x50000000 is 32 bits long, while permitted offset is 16 bits long. Because it can not be converted to fit in 16 bits, it is not possible to use the beq instruction to set the PC from 0x20000000 to 0x50000000.

5.Write down the binary representation of the decimal number 64.35 assuming the IEEE 754 single precision format.

64.35 is represented as 0x40.58 in IEEE 754 single precision format.

Whole number portion of 64.35 is 64. And, decimal portion is 0.35.

Whole Number Division	Result	Remainder
64/2	32	0
32/2	16	0
16/2	8	0
8/2	4	0
4/2	2	0
2/2	1	0
1/2	0	1

So, the binary representation of 64 is 1000000.

Converting the Decimal portion to binary, we get.

Decimal Number Multiplication	Result	Number Infront of Decimal
0.35 * 2	0.7	0
0.7 * 2	1.4	1
0.4 * 2	0.8	0
0.8 * 2	1.6	1
0.6 * 2	1.2	1
0.2 * 2	0.4	0
0.4 * 2	0.8	0
0.8 * 2	1.6	1

So, the binary representation of 0.35 is $0.010\overline{1100}$ (approx).

Now, combining the whole number and decimal portion, we get $1000000.010\overline{1100}$.

Converting to base 2 scientific notation, we get $1.0000000101\overline{1100} * 2^6$.

So, sign = 0

exponent = 6

mantissa = $0.010\overline{1100}$

1st bit sign = 0

2nd to 9th bit exponent (8 Bits) = 6 + 127 = 133 = $(10000101)_2$

10th to 32nd bit mantissa (23 Bits) = $0.010\overline{1100}$

So, the 32 Bit representation is as follows:

```
0    10000101   01011001100110011001101
Sign Exponent   Mantissa
```

6.Using the IEEE 754 floating point format, write down the bit pattern that would represent -1/4. Now, add -1/4 to itself 4 times. Do you get the equivalent representation for -1.0? Why or why not? Show your work