ECE 6913, Computing Systems Architecture, Spring 2020

Please fill in your name: ____

Quiz 1, March 9th 2020

Maximum time: 2.5 hours : **12:25 PM – 2:55 PM**

Closed Book, Closed Notes, 1 'cheat sheet' allowed (both sides)

Calculators allowed. RISC V Card provided

This Test has 4 problems. Please attempt all of them. Please show **all work**. Please write **legibly**

1. After graduating, you are asked to become the lead computer designer at Hyper Computers, Inc. Your study of usage of high-level language constructs suggests that procedure calls are one of the most expensive operations. You have invented a scheme that reduces the loads and stores normally associated with procedure calls and returns. The first thing you do is run some experiments with and without this optimization. Your experiments use the same state-of-the-art optimizing compiler that will be used with either version of the computer. These experiments reveal the following information:

• The clock rate of the unoptimized version is 5% higher.
• 30% of the instructions in the unoptimized version are loads or stores.
• The optimized version executes 2/3 as many loads and stores as the unoptimized version. For all other instructions the dynamic counts are unchanged.
• All instructions (including load and store) take one clock cycle.

Which is faster? Justify your decision quantitatively

CPU performance equation:
```
Execution Time = IC  * CPI * Tcycle
```

The unoptimized version is 5% faster in clock cycle time
```
Tcycle_unop = 0.95 * Tcycle_op
```

Instruction Count (IC) of load/store instructions are 30% of total IC in the unoptimized version
```
IC_ld/st_unop = 0.3 * IC_unop
```

IC of load store instructions in optimized version is 0.67 of IC of load/store instructions in unoptimized version
```
IC_ld/st_op = 0.67 * IC_ld/st_unop
```

IC of all other (non load/store) instructions in optimized version = IC of all other (non load/store) instructions in unoptimized version
```
IC_others_unop = IC_others_op
```

CPI = 1

```
Execution Time_unop = IC_unop * Tcycle_unop
= 0.95 * IC_unop * Tcycle_op

Execution Time_op = IC_op * Tcycle_op

IC_op = IC_others_op + IC_ld/st_op
IC_op = IC_others_unop + IC_ld/st_op
IC_op = 0.7 * IC_unop + 0.67 * IC_ld/st_unop
IC_op = 0.7 * IC_unop + 0.67 * (0.3 * IC_unop) = 0.7 * IC_unop
+ 0.2*IC_unop
= 0.9 * IC_unop
Execution Time_op = IC_op * Tcycle_op
= 0.9 * IC_unop * 1.05 * Tcycle_unop

= 0.945 * IC_unop * Tcycle_unop
```

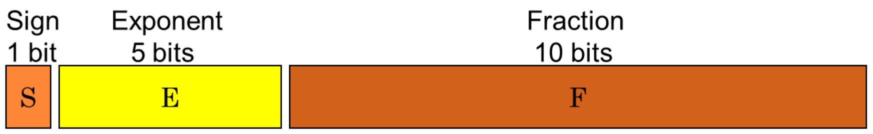
Improvement of 5.5%

2. Find a short sequence of RISC-V instructions that extracts **`bits 23 down to 8`** from register `x7` and uses the bitwise inverted value of this field to replace bits `31 down to 16` in register `x8` without changing any of the bits of registers `x7` or any of the bits in the less significant half word of `x8 (bits 15:0).`

```
lui x5 0x0ffff          // fill most significant 20 bits (5 hex) with
                        // 0ffff char: x5 now has these 8 hex:
                        // 0ffff000
slri x5 8               // x5 now has 000ffff0 with bits 23 to 8
                        // now a'1'

and x28 x5 x7           // apply mask to x7 and place 16 desired

                        // bits in positions 23 → 8 from x7 into x28

                        // all bits in positions other than 23→8

                        // are '0'

xori x29 x28 -1         // invert these bits and update them in x29

and x28 x5 x29          // copy only these inverted bits back to x28

                        // using the original mask identifying

                        // positions 23 → 8. all other bits are '0'

slli x5 x5 8            // create mask for x8 to identify 16 bits in

                        // most significant 16 bit positions

xori x5 x5 -1           // invert this mask placing '0' 16 most

                        // significant positions and '1'

                        // in least significant positions

and x8 x8 x5            // zero out the most significant 16

                        // positions in x8

slli x28 x28 8          // 16 desired bits from positions 23→8 in x7

                        // are now moved into most

                        // significant positions 31 – 16 in x28

or x8 x8 x28            // simply copy 16 desired bits from x7 into

                        // most significant 16 positions in x8
```

3. NVIDIA has a "half" format, which is similar to IEEE 754 except that it is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and the Fraction field is 10 bits long. A hidden 1 is assumed.

REPRESENTATION RANGE OF THE NVIDIA 'HALF FORMAT'



For each of the following, write the *binary value* and *the* ***corresponding*** *decimal value* <u>of the 16-bit floating point number that is the closest available representation of the requested number</u>. If rounding is necess2ary use round-to-nearest. Give the decimal values either as whole numbers or fractions.

Number	Binary	Decimal
0	00000 0000000000	0
Charge of an electron: -1.6 x 10^{-19} (C)	1 00000 0000000000	0
Smallest positive normalized number	0 00001 0000000000	6.103515625 x 10^{-5}
Smallest positive **denormalized** number > 0	0 00000 0000000001	5.960464477 x 10^{-8}
Largest positive **denormalized** number > 0	0 00000 1111111111	~6.103515625 x 10^{-5}
Largest positive number < infinity	0 11110 1111111111	= 65536
Average distance b/w proton and neutron in Hydrogen atom = 0.8751 x 10^{-15} m	0 00000 0000000000	0
Distance between Earth and Neptune in inches = 171,072,000,000,000	0 11111 0000000000	overflow

<u>Smallest normalized number > 0</u>: 0 00001 0000000000; bias = 15; val of exponent = 1-bias = -14, Significand = 1.000…0_2 = 1, Value in decimal ~ 1.00 x 2^{-14} = 6.103515625 x 10^{-5}

<u>Smallest positive denormalized number > 0</u>: 0 00000 0000000001; value of exponent = -14; significand = 0.0000000001 = 1 x 2^{-10}; so smallest denormalized number = 1 x 2^{-10} x 2^{-14} = 2^{-24} = 5.960464477 x 10^{-8}

<u>Largest possible denormalized number > 0:</u> 0 00000 1111111111; value of exponent = 1-15 = -14; significand = 0.1111…1 ~ 1.0; so largest denormalized number = 1 x 2^{-14} ~ 6.103515625 x 10^{-5}

<u>Largest positive number < inf:</u> 0: 0 11110 1111111111; value of exponent = 30 - 15 = 15; Significand = 1.11…1_2 = ~ 2; value in decimal ~ 2 x 2^{15} = 2^{16} = 65536

<u>Overflow:</u> represented by +infinity: 0 11111 0000000000

4a. Write out instruction(s) to **copy contents at one memory location to another:** `B[g+2] = A[i+j+4]`. Assume i, j, g values are in registers `x5, x6, x7`. Assume base address in memory of Array data structures 'A, B' (or address in memory of 'A[0]' and '`B[0]`') are stored in Registers `x28, x29`

4b. The number '12345678' in hex is stored as a 32-bit word at address '100'

Identify the contents of the word with the byte addresses as identified in the Table below for little endian and big endian machines:

Byte address	Little Endian	Big Endian
98		
99		
100	78	12
101	56	34
102	34	56
103	13	78
104		