EE4039: Computer Architecture
Midterm Exam (14:20-17:20, 2024/10/29)

Name: ________________ ID: ________________

Answer 1. The average CPI for P1 is $1 \times 0.1 + 2 \times 0.2 + 3 \times 0.5 + 3 \times 0.2 = 2.6$. The average CPI for P2 is 2. Thus there are 5.2 and 4 million instructions respectively.

1. The global CPIs are 2.6 and 2 respectively.
2. The number of clock cycles are $5.2M$ and $4M$ respectively.
3. The running time for P1 is $5.2M/2.5G = 2.08ms$. The running time for P2 is $4M/3G = 1.33ms$.

Answer 2.

1. `0x28`.

2. The immediate `imm = 0x10`. We include each part: `imm[12|10:5] = 00000000`, `rs2 = 00000`, `rs1 = 00101`, `func3 = 101`, `imm[4:1|11] = 10000`, and `opcode = 1100011`. Combining these together we have

```
0000000 00000 00101 101 10000 1100011 = 0x0002d863
```

3. `0x63, 0xd8, 0x02, 0x00`.

Answer 3.

1. $13_{10} = 1101_2$ and $6_{10} = 0110_2$. See below for each update.

```
0000 0110 (initial value)
0000 0011 (right shift)
0110 1001 (add, shift right)
1001 1100 (add, shift right)
0100 1110 (shift right)
```

   The product is $01001110_2 = 78_{10}$.

2. $123_{10} = 01111011_2$ and $8_{10} = 1000_2$. See below for each update.

```
0111 1011 (initial value)
1111 0110 (shift left)
1110 1101 (sub, shift left)
1101 1011 (sub, shift left)
1011 0111 (sub, shift left)
0110 1111 (sub, shift left)
```

The quotient is $1111_2 = 15_{10}$ and the remainder is $0011_2 = 3_{10}$. Since the remainder gets left-shifted five times, we neglect the lsb of the left part.

Answer 4.

1. `23.171875 = 0 10011 0111001011 = 0100 1101 1100 1011 = 0x4DCB`.
   binary: 1 0111.001011

2. With the leading 1 back,

```
  0 10000 1.0100000000
+ 1 11011 1.1000000000
--------------------------------
  0 11011 0.00000000001010000...
+ 1 11011 1.10000000000000000...
--------------------------------
  1 11011 1.01111111110110000...
--------------------------------
  1 11011 1.0111111111
```

   The result is `1 11011 0111111111 = 1110 1101 1111 1111 = 0xEDFF`.

3. With the leading 1 back,

```
  0 10000 1.0100000000
+ 1 11011 1.1000000000
--------------------------------
  0 11011 0.000000000010
+ 1 11011 1.100000000000
--------------------------------
  1 11011 1.011111111110
--------------------------------
  1 11011 1.1000000000
```

   The result is `1 11011 1000000000 = 1110 1110 0000 0000 = 0xEE00`.

4. With the leading 1 back,

```
  0 10000 1.0100000000
+ 1 11011 1.1000000000
--------------------------------
  0 11011 0.0000000000101
+ 1 11011 1.1000000000000
--------------------------------
  1 11011 1.0111111111011
--------------------------------
  1 11011 1.0111111111
```

The result is `1 11011 0111111111 = 0xEDFF`.

Answer 5.

1. Let x be any n-bit binary representation of integer a, i.e., $a = -x_{n-1}2^{n-1} + b$, where $b = x_{n-2}2^{n-2} + \ldots + 2x_1 + x_0$. The sign extension to m bits is the representation of number

$$\begin{aligned} a' &= -x_{n-1}2^{m-1} + x_{n-1}2^{m-2} + \ldots + x_{n-1}2^{n-1} + b \\ &= -x_{n-1}(2^{m-1} - 2^{m-2} - \ldots - 2^{n-1}) + b \\ &= -x_{n-1}2^{n-1} + b = a. \end{aligned}$$

2. Since the most significant t bits are all the same, we may view $(x)_2$ as a sign extension of an $(m-t+1)$-bit binary representation to m bits. This implies that $-2^{m-t} \leq a < 2^{m-t}$. Thus a left shift by $k < t$ bits does not cause an overflow.

   Let the sign bit of x be s. Since the most significant t bits are all the same, we write $x = s...sx_{m-t-1}...x_1x_0$. A left shift by k bits yields a binary number $x' = s...sx_{m-t-1}...x_1x_00...0$, where s is repeated for $t-k$ times.

   From the first part of the problem, write $a = -s \cdot 2^{m-t} + b$, where $b = x_{m-t-1}2^{m-t-1} + \ldots + x_1 2 + x_0$. By simple calculation, $a' = -s \cdot 2^{m-t+k} + b \cdot 2^k = a \cdot 2^k$.

Answer 6.

1. The program copies a number from location `a` to location `b`.

2. The following code does the job:

```
sbn temp, temp, .+1
sbn temp, b, .+1
sbn a, temp, .+1
```

3. See the following code:

```
sbn c, c, .+1
LOOP: sbn b, one, EXIT        // b = b - 1
                              // if ( b < 0 ) go to EXIT
      sbn temp, temp, .+1     // c = c + a
      sbn temp, a, .+1
      sbn c, temp, .+1
      sbn temp, temp, .+1     // always go to loop
      sbn temp, one, LOOP
EXIT:
```

Answer 7.

1. The multiplication

$$\begin{aligned} 2A_{16} \times 77_{16} &= 2A_{16} \times (80_{16} - 10_{16} + 08_{16} - 01_{16}) \\ &= 1500_{16} - 02A0_{16} + 0150_{16} - 002A_{16} \\ &= 1260_{16} + 126_{16} \\ &= 1386_{16} \end{aligned}$$

2. The partial product is initially set to zero. Scan the multiplier $y = y_{b-1}y_{b-2}y_{b-1}\dots y_1 y_0$ from right to left. If $y_0 = 1$, subtract x from the partial product. For $i = 1 \dots b-1$, when $y_i y_{i-1}$ is equal to

   - 00, do nothing;
   - 01, add x left shifted by i bits to the partial product;
   - 10, subtract x left shifted by i bits from the partial product;
   - 11, do nothing.

   Each addition is sign extended to $a+b$ bits.

3. Without loss of generality, we write the multiplier as an alternation of all-1 strings and all-0 strings, i.e.,

$$y = \dots 00\dots 0 \mid 11\dots 1 \mid 00\dots 0 \mid \dots \tag{1}$$

   where $|$ is the concantenation symbol. Operations act on the bit left to each concatenation symbol.

   We prove the correctness of the algorithm by induction on the number k of all-1 strings. For any string x, let $[x]$ denote the number it represents.

   For $k = 0$, the multiplier is either an empty string or a zero string, and the algorithm trivially outputs 0.

   Suppose that for $k \leq t$, the algorithm is correct. Then for the induction step, let y be any multiplier with $t+1$ all-1 strings. We write $y = z|y'$, where y' has k all-1 strings, and the lsb of z is 1. Now we consider the following two cases:

   - The msb of z is 1. This means that z is an all-1 string. Let m be the size of y'. By the induction hypothesis, the algorithm additionally subtracts x left shifted by m bits, i.e., it outputs the 2s complement representation of

$$[x] \cdot [y'] + [x] \cdot (-1) \cdot 2^m = [x] \cdot [1|y'] = [x] \cdot [z|y'] = [x] \cdot [y].$$

     where the second inequality holds by the first item of Problem 5.

- The msb of z is 0. Let the number of 1s in z be n. This implies that the algorithm, in addition to the operation for y', performs a subtraction of x shifted by m bits and an addition of x shifted by $m+n$ bits. By the induction hypothesis, the algorithm outputs the 2s complement representation of

$$\begin{aligned}[x]\cdot[y'] + [x]\cdot(-1)\cdot 2^m + [x]\cdot 2^{m+n} &= [x]\cdot([y'] + 2^m(2^{n-1} + \ldots + 2 + 1)) \\ &= [x]\cdot[z|y'] = [x]\cdot[y].\end{aligned}$$

Answer 8.

1. See the code below:

```
        addi x11, x0, 1
FACT:   beq x10, x0, EXIT
        mul x11, x11, x10
        addi x10, x10, -1
        jal x0, FACT
EXIT:   ...
```

2. See the following C code:

```
int fib2(int n, int a, int b){
    if (n == 0) return a;
    else return fib2(n - 1, b, a + b);
}
```

The call `fib2(n, 0, 1)` returns the nth Fabonacci number.

The following RISC-V code will also work as a valid answer if students choose to use the language. Let `x10`, `x11`, `x12` be the registers holding n, a, b respectively. The final answer will be stored in `x11`.

```
addi x11, x0, 0
addi x12, x0, 1
FIB:    beq x10, x0, EXIT
        addi x10, x10, -1
        add x13, x0, x12
        add x12, x12, x11
        add x12, x11, x12
        add x11, x0, x13
        jal x0, FIB
EXIT:   ...
```