CA homework#2

b07902076
資工三 許世儒

2.8

```
1     addi x30, x10, 8
2     addi x31, x10, 0
3     sd   x31, 0(x30)
4     ld   x30, 0(x30)
5     add  x5 , x30, x31
```

In C:

```
1     A[1] = &A[0];
2     f = A[1] + &A[0];
```

2.9

instruction	op	type	rs1	rs2	rd	imm	funct3	funct7
addi x30, x10, 8	0010011	I-type	x10	-	x30	8	000	-
addi x31, x10, 0	0010011	I-type	x10	-	x31	0	000	-
sd x31, 0(x30)	0100011	S-type	x31	x30	-	-	011	-
ld x30, 0(x30)	0000011	I-type	x30	-	x30	-	011	-
add x5, x30, x31	0110011	R-type	x30	x31	x5	-	000	0000000

2.16

2.16.1

rs1, rs2 and rd will expand from 5 bits(at most 32) to 7 bits(at most 128).
The instructions are 4 times more, so the opcode will shift 2 bits,
that is from 7 bits to 9 bits. Therefore, funct3 and funct7 field can potentially be decreased which depends on ISA designer.

2.16.2

rs1 and rd will expand from 5 bits to 7 bits.
The opcode will expand from 7 bits to 9 bits.
The imm and funct3 field can potentially be decreased depending on ISA designer.

2.16.3

With more registers, it may decrease the size of program because we won't need that much memory, therefore, the numbers of **load** and **store** instructions may decrease.
With more registers, it may increase the size of program because **the length of instructions is larger**.
With larger instruction set, it may decrease the size of program because we may finish the same job with **less instructions** (instructions can be more functional).
With larger instruction set, we have to **expand the length (opcode) of each instruction** so it may increase the size of the program.

Report on matrix multiplication

- It takes about 1.69×10^7 cycles by naive method.

```
root@47f72354622b:~/Problems/matrix# make test
spike pk ./matrix
bbl loader
Took 16892677 cycles
```

- For the naive method, to compute each element in matrix C, we need to load 128 elements from A and 128 from B. That is, we need 2×128 load operations for one element in C. We have to compute 128×128 elements in matrix C, so we need $128^2 \times 2 \times 128 = 2 \times 128^3$ times of load operations. Then, we need to store to matrix C, and there are 128×128 elements in matrix C. Hence, we need 128^2 times of store operations. Totally, we need $2 \times 128^3 + 128^2 \approx 4.2 \times 10^6$ load and store operations.

- The C code is as follows.

```
1   for(int i = 0; i < 128; ++i) {
2       for(int j = 0; j < 128; j+=8) {
3           for(int k = 0; k < 128; ++k) {
4               unsigned short a = A[i][k];
5               C[i][j] = (C[i][j] + a * B[k][j]) % MOD;
6               C[i][j+1] = (C[i][j+1] + a * B[k][j+1]) % MOD;
7               C[i][j+2] = (C[i][j+2] + a * B[k][j+2]) % MOD;
8               C[i][j+3] = (C[i][j+3] + a * B[k][j+3]) % MOD;
9               C[i][j+4] = (C[i][j+4] + a * B[k][j+4]) % MOD;
10              C[i][j+5] = (C[i][j+5] + a * B[k][j+5]) % MOD;
11              C[i][j+6] = (C[i][j+6] + a * B[k][j+6]) % MOD;
12              C[i][j+7] = (C[i][j+7] + a * B[k][j+7]) % MOD;
13          }
14
15      }
16  }
```

  As we can see, we load `A[i][k]` for 8 times of multiplication operations. Therefore, we can keep `A[i][k]` in a register and keep it being used. As a result, the total load operations on A is $\frac{1}{8}$ times compared to the naive method.

- As the code in the previous problem has shown, we use three for-loop controls using this techniques.