The Fibonacci sequence refers to such a sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ... The mathematical definition of the Fibonacci sequence: F(0)=0, F(1)=1,

F(n)=F(n-1)+F(n-2) (n>=2, n ∈ N*)

Part1_1: Realize Fibonacci sequence with iteration

1. A brief description of each part completed

a. Initialize index and parameter n

b. Read the initial value: the first and second value first

c. Enter for loop

d. Jump out of the loop and load the output value into the destination register

2. The approach taken

There is no special method. Basic instructions are constantly used, such as branch instructions, store and load instructions, etc.

3. The challenge faced

There is no problem with this part.

4. Possible improvement to the programs

a. The coding style is not clear enough.

b. The number of registers is limited, while I used too many registers, increasing the amount of calculation.

c. I was not familiar enough with the effective address when I started writing this part. So I used the most elementary and repetitive method in offset, which could has be written in only one high-level instruction.

d. After searching on the Internet, I found that two temporary variables and one counter can be implemented to realize iteration, which can be much more simple than array.

Part1_2: Realize Fibonacci sequence with recursive subroutine

1. A brief description of each part completed

a. Initialization

b. Save return address for fib(n), push fib(n-1), fib(n-2) into stack...until fib(1), fib(0)

c. We calculate and pop fib(2) back to stack by ...until fib(n)

2. The approach taken

subroutine, stack

3. The challenge faced

There is no problem with this part.

4. Possible improvement to the programs

It can be solved directly by the general formula of the Fibonacci sequence.

The definition of convolution is an ambiguous mathematical formula: $f * g(n) = \sum_{\tau=-\infty}^{+\infty} f(\tau)g(n-\tau)$. A two-dimensional convolution is actually a kind of multiplication between two matrices.

Part2: 2D convolution

1. A brief description of each part completed

a. Initialization, define 2-dimentional array fx, gx, kx

b. Enter y-loop, x-loop, i-loop, j-loop respectively

c. The movement range of the kernel needs to be restricted

d. Jump out of the loop and load the value into the destination register

2. The approach taken

There is no special method. Basic instructions are constantly used, such as branch instructions, compare instructions, etc.

3. The challenge faced

a. Memory is one-dimensional, while array is two-dimensional. Thus, one-to-one correspondence is a bit troublesome.

b. There are many loops and variables, which need to be handled carefully.

c. When I checked the C code, I didn’t figure out the padding and stride at the beginning. I didn't understand why the output is a 10*10 array. After I printed the intermediate value and found the data flow, I figured out what was going on.

4. Possible improvement to the programs

There are the same problems as part1: too many registers are used, and the code style is not clear.

The principle of bubble sort is to constantly compare two adjacent elements, and swap the elements with a higher value to the right, until it forms an order from smallest to largest.

Part3: bubble sort

1. A brief description of each part completed

a. Initialization

b. Enter for loop

c. Swap elements under some conditions

d. Jump out of the loop and load the value into the destination register

2. The approach taken

I used array instead of pointer given in the C code. The corresponding C is as follows:

```
#include <studio.h>
void bubblingSort(int arr[], int n) {
        int i, j, temp;
        for (i = 0; i < n; i++) {
                for (j = 0; j < n - i - 1; j++) {
                        if (arr[j] > arr[j + 1]) {
                                temp = arr[j];
                                arr[j] = arr[j + 1];
                                arr[j + 1] = temp;
                        }
                }
        }
}

int main() {
        int i;
        int arr[10] = {5, 2, 3, 8, 1, 2, 6, 9, 3, 7};
        bubblingSort(arr, 10);
        for (i = 0; i < 10; ++i) {
                printf("%d ", arr[i]);
        }
        return 0;
}
```

3. The challenge faced

Initially I forgot to set 4 byte alignment, leading to the incorrect output.

4. Possible improvement to the programs

a. There are the same problems as part1: too many registers are used, and the code style is not clear.

b. I shouldn’t avoid pointer method, which is one of the most important instruction in assembly.