Lab3 Better Angels

2021.12 赖永凡 PB20061210

Task

task 1: read and understand

In lab 3, you will get a piece of machine code in 'foo.txt'. Your first task is to translate machine code into assembly code. Store your program in **'translate.txt'**.

task 2: guess

This code is some other's program in lab2. Guess the owner of the program by the last 4 lines of the program. Write down the owner's student id in **'id.txt'**.

task 3: optimize

The code in lab2 is a L-version program. Of course it's performance is not very well. In this part, you need to optimize other's program. (Rewriting is also a legitimate optimization method) Store the optimized code in **'optimized.txt'**.

Part I & II

The initial machine code and the assembly code translated are shown below:

Tokens from left to right are: **machine code ; address assembly code comment**

```
0011000000000000        ;Suppose that the program begins at x3000
0010 101 000010100      ;x3000 LD R5,x3015
0001 001 001 1 00001    ;x3001 ADD R1,R1,#1
0001 010 010 1 00001    ;x3002 ADD R2,R2,#1
0001 011 011 1 00010    ;x3003 ADD R3,R3,#2
1001 100 101 1 11111    ;x3004 NOT R4,R5
0001 100 100 1 00001    ;x3005 ADD R4,R4,#1     Yeild x03FF
0001 111 111 1 00001    ;x3006 ADD R7,R7,#1     Initialize register
0001 000 000 1 11110    ;x3007 ADD R0,R0,#-2
0000 010 000001010      ;x3008 BRz x3013
0000 100 000001010      ;x3009 BRn x3014        Check for base cases
0001 111 011 0 00 001   ;x300A ADD R7,R3,R1
0001 111 111 0 00 001   ;x300B ADD R7,R7,R1     R7 = 2*R1 + R3
0001 001 010 1 00000    ;x300C ADD R1,R2,#0     R2 -> R1
0001 010 011 1 00000    ;x300D ADD R2,R3,#0     R3 -> R2
0001 111 111 0 00 100   ;x300E ADD R7,R7,R4
0000 011 111111110      ;x300F BRzp x300E
0001 011 111 0 00 101   ;x3010 ADD R3,R7,R5     Acquire remainer
0001 000 000 1 11111    ;x3011 ADD R0,R0,#-1    Sentiel
0000 101 111110111      ;x3012 BRnp x300A
0001 111 011 1 00000    ;x3013 ADD R7,R3,#0     Get final value
1111000000100101        ;x3014 HALT
0000 0100 0000 0000     ;x3015 x0400
0000 0010 1101 0010     ;x3016 x02D2    18
0000 0000 0001 1110     ;x3017 x001E    07
```

```
0000 0011 1111 0110     ;x3018 x03F6    15
0000 0000 0101 0010     ;x3019 x0052    66
```

After using the C program to get a table of $F(a), 0 \le a \le 99$, aided with CTRL+F, we can easily find out the owner's student number, that is $PB18071566$. luckily enough, there is a guy suffixing his name with the persicely number in the QQ group.

Part III

Now is the time to optimize.

Reading through the translated code, I found that the logic behind is so similar to the code I wrote in Lab2, and even more efficient in base cases check part, **except one thing**, that is, dealing with the remainder after divided by 1024. Maybe it is his carelessness, or maybe it is just the kindness he wanted to convey.

Anyway, optimization is easy.

```
        .ORIG x3000
        LD R5,MOD
        ADD R1,R1,#1
        ADD R2,R2,#1
        ADD R3,R3,#2
        ADD R7,R7,#1
        ADD R0,R0,#-2
        BRz ALDONE
        BRn DONE
LOOP    ADD R7,R3,R1
        ADD R7,R7,R1
        ADD R1,R2,#0
        ADD R2,R3,#0
        AND R3,R7,R5
        ADD R0,R0,#-1
        BRnp LOOP
ALDONE  ADD R7,R3,#0
DONE    HALT
MOD     .FILL x03FF
F18     .FILL x02D2
F07     .FILL x001E
F15     .FILL x03F6
F66     .FILL x0052
        .END
```

In the previous version, the way to get the remainder is **interesting**.

First, we load R5 with x0400 (#1024) , and get its opposite number in R4. When we need to calculate $P \mod 1024$, we just keep doing $P - 1024$, which is ADD R7,R7,R4 in the initial code. if the result becomes negative, then we add a 1024 back, in this manner, we can get the $P \mod 1024$.
In one word, the algorithm means to find the smallest positive number having the same remainder.

As for my optimization, just replace all the codes concerning with one operation AND N with x03FF.

Last let's analyses the performance difference. Suppose that we need to calculate $F(n)$ in some case.

In previous version, you need to do the preparation for 3 instructions, and while calculation, you have to do the check process for at least $3 \times n$ times. Compired to the optimized version where only$1 + n$ instructions are needed to get the remainder, the optimization is obvious.