Data Science PB22051087 YufengPeng 2023.11.21

1 Purpose

1.1 Objective

Calculate the given function

$$f\left(N\right) = 2 \times v_{N-1}\, d_{N-1}\, 2$$

Where $f\left(N\right)$ includes two elements, one is an oprand (v_N) and another is either " $+$ " or " $-$ " oprator (d_N).

1.2 Anticipated outcomes

N	1	2	3	4	5	6	7	8	9
$f\left(N\right)$	3	8	14	26	50	98	198	394	786
Oprator	+	-	-	-	-	+	-	-	-

2 Principles

In order to compute a given function, there are several technical difficulties to overcome.

2.1 Remainder operation

In this task, we need to implement the remainder operation for 8 and 10. If $v_{n+1} \mod 8 \equiv 0$ or $v_{n+1} \mod 10 \equiv 8$. d_{n+1} will change to another. To implement this function, I did these steps:

(1) Load v_{n+1} into R0 (or any other register).

(2) Store the result of R0 $-$ 10 to R1.

(3) If the condition code p is TRUE, then update the data in R0 to R0 $-$ 10 and go back to step (2).

(4) Otherwise the data stored in R0 will be the remainder of $v_{n+1} \mod 10$.

After we get the remainder of $v_{n+1} \mod 10$, we only need to subtract R0 by 8 to know that if d_{n+1} need to change. If the result of R0 $-$ 8 is 0, we can infer that the last digit of v_{n+1} is 8 ($v_{n+1} \mod 10 \equiv 8$).

For the remainder of 8, we can simply execute the instruction AND R1, R0, #7, if the result is 0, then R0 $\mod 8 \equiv 0$.

2.2 Load the oprand 4096

To represent the number 4096, we need 13 bits, which is cannot be load by ADD or any other instruction directely. So I did the follow steps:

(1) Put the instruction ADD R0, R0, R0 at an address that after the instruction TRAP x25.

(2) Using LD instruction to load this instruction into register.

Because the instruction ADD R0, R0, R0 is 0001 0000 0000 0000, which is exactly the representation of 4096, so if I load this instruction to a register, then I will get a register storing 4096. We can use this register to do the remainder operation.

3 Procedure

3.1 Algorithm

After we have solved the above two problems in section 2, we can successfully calculate the given function by using follow algorithm.

(1) Load $v_1 = 3$ to a register (R1 as an example). (When I say "load" means using any mathod to let the DR store the right data.)

(2) Load 0 into a register (R2 as an example) to record d_n. 0 for " $+$ " oprator, 1 for " $-$ " oprator.

(3) Load N from x3102 into a register (R0 as an example).

(4) Calculate $N - 1(\text{R0} = \text{R0} - 1)$, if the result is 0, jump to step (13).

(5) Detect the data stored in R2, if it's 0, do step (6). Otherwise (1 is stored in R2) jump to step (7).

(6) $\text{R1} = \text{R1} + \text{R1} + 2$, jump to step (8).

(7) $\text{R1} = \text{R1} + \text{R1} - 2$, jump to step (8).

(8) Calculate $\text{R1} \mod 4096$ ($\text{R1} = \text{R1} \mod 4096$).

(9) Calculate $N - 1$, if the result is 0, jump to step (13).

(10) Calculate $\text{R1} \mod 8$, if the remainder is 0, jump back to step (12).

(11) Calculate $\text{R1} \mod 10$, if the remainder is not 8, jump to step (5).

(12) Change the data stored in R2 ($\text{R2} = 1 - \text{R2}$), jump back to step (5).

(13) Store the data in R1 to x3103.

When I said " Calculate $A \mod B$, " there are actually many steps required to complete the calculation, but in order to briefly describe the algorithm, I have omitted some simple intermediate steps.

3.2 Bugs and Solutions

When I built my algorithm, a bug appeared, that is, starting from N equal to 13, the program operation result was different from the expected result. After checking, I found that I did not make the result remainder to 4096 after each operation, but It is only at the end of the last loop that the result is subjected to a remainder operation of 4096 before storing the result.

Therefore, when N is less than 12, the result is less than 4096. Whether to calculate the remainder of 4096 has no impact on the result. However, when N is equal to 12, whether to calculate the remainder will have an impact on the result of N equal to 13, which makes the subsequent data error.

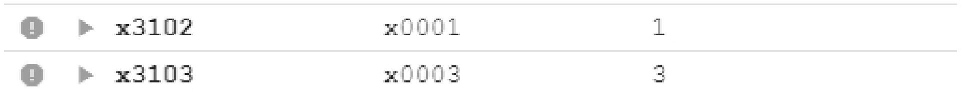
In order to deal with this bug, I adjusted the running order of the program so that after each operation, the result is modulus to 4096.

Besides, there are also some other bugs appeared when I was coding, but all of them are not the problems with the algorithm, so I won't write them down in detail here. To solve these problems, I wiil use the "Step in" function to check if all used register are loaded with expected results, if any one of them gose wrong, I'll check the code to fix it.

Sometimes, I need to run a lot of steps to get to where I want to be with the step in function, which will spend a lot of times. For reducing those unexpected spending of tim, I'll use the breakpoint function to quickly get the program to run where I want it to be, and automatically pause so that I can check the data in registers and memoty.

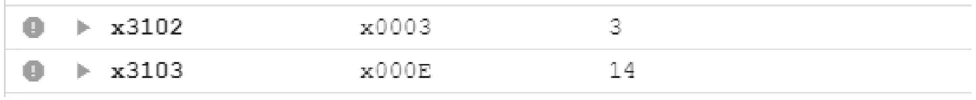
4 Results

I'll attach some screenshots of some typical values of N and their $f\left(N\right)$ below.



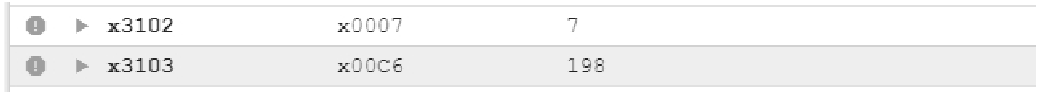
```
x3102    x0001    1
x3103    x0003    3
```

Figure 1: $N = 1$



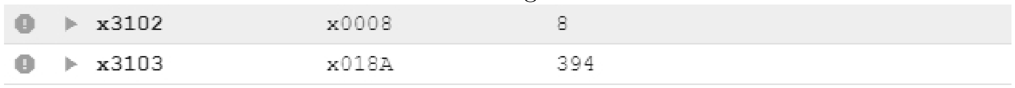
```
x3102    x0003    3
x3103    x000E    14
```

Figure 2: $N = 3$



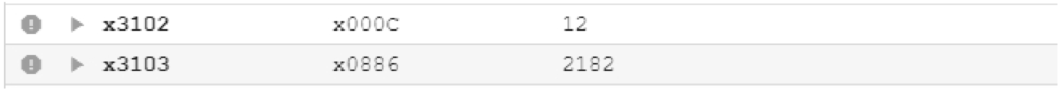
```
x3102    x0007    7
x3103    x00C6    198
```

Figure 3: $N = 7$



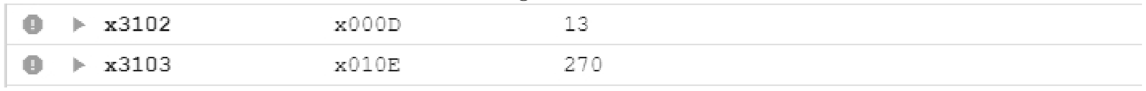
```
x3102    x0008    8
x3103    x018A    394
```

Figure 4: $N = 8$



```
x3102    x000C    12
x3103    x0886    2182
```

Figure 5: $N = 12$



```
x3102    x000D    13
x3103    x010E    270
```

Figure 6: $N = 13$

Obviusly the results of my program are as expected.

5 Improvements

In order to optimize the efficiency of the loop structure in my program, I did the follow things:

1. Try my best to run the same operation in different steps only once.
2. Adjust the position of the program block to reduce unnecessary BR instructions.
3. Place all instructions that only need to be executed once outside the loop (some initialization operations), thereby reducing the number of instruction loops during the loop.
4. Enforcing loop termination conditions at the appropriate locations can greatly reduce the need for unnecessary instructions to run.
5. Using different methods to deal with remainder operations for 8 and 10. For module 8, I did n ADD #7 instead of subtracting n by 8 so many times, which improved the efficiency so much.

Point 4 above has significantly improved the running efficiency of the program. In my program, I placed the loop termination condition after each **complete** calculation (including mod 8 , 10 and change the d_n) at the beginning. After adjustment, I placed the loop termination condition right after the remainder operation of 4096 , which improves program running efficiency by about 30%.