Lab02 Report

崔士强 PB22151743

November 22, 2023

1 Purpose

The purpose of the program is to compute the *Pingpong* Sequence under the following rules:

1. $f(n) = \{(v_n, d_n) \,|v_n \in \mathbb{Z}, d_n \in \{+, -\}, n \geq 1\}$
2. $f(1) = (3, +)$
3. $v_{n+1} = 2v_n \ d_n \ 2$
4. After computing v_{n+1}, if v_{n+1} is divisible by 8 or if the last digit of its decimal representation is '8', then d_{n+1} changes to another, else $d_{n+1} = d_n$

The programs computes $f\,(N)$. N is loaded from memory location x3102 and the result is saved in x3103. Every time the program determines a term of the sequence, arithmetic operations are performed modulo 4096.

Anticipated outcomes:

N	1	2	3	4	5	6	7	8	9	10	11
$f(N)$	3	8	14	26	50	98	198	394	786	1570	3138
operand	+	−	−	−	−	+	−	−	−	−	+

N	12	13	14	15	16	17	18	19	20	> 20
$f(N)$	2182	270	542	1086	2174	254	510	1022	2046	4094
operand	+	+	+	+	+	+	+	+	+	+

2 Principles

2.1 Steps

1. Read N into R0. Set R1 as the initial value of the sequence. Clear relevant registers.
2. Check whether the operand needs changing. If yes, flip the indicator.

3. Compute the next term of the sequence according to the indicator.

4. Repeat steps 2 and 3 for $N-1$ times.

2.2 Performing modulus

During calculation, arithmetic operations are performed modulo 4096. Given that $4096 = 2^{12}$, we simply needs to mask out bits[16:13]. R3 is loaded with x0FFF to obtain bits[12:0] by AND operation.

Initialization:

```
        AND R3, R3, #0    ; Clear R3 to store x0FFF to mask the result
        LD R3, MASK
MASK    .FILL x0FFF
```

Performing modulus:

```
        AND R1, R1, R3    ; Mask out the first 4 bits of R1
```

2.3 Indicator

R2 acts as the indicator for the operand. If R2 is x0000, the operand is $+$, otherwise it is $-$. When operand needs to be changed, flip all bits of R2.

Relevant code:

```
FLIP    NOT R2, R2        ; Change the indicator
        BRnzp CHOOSE

CHOOSE ADD R2, R2, #0     ; Get the value of R2
        BRn SUBS          ; If the indicator is negative, perform substraction
```

2.4 Checking

In this part we need to check for two things:

1. Divisibility by 8: Repeatedly substract 8, until the result is zero(divisible) or negative(indivisible).

2. The last digit: Repeatedly substract 10, until the result is -2(the last digit is 8) or other negative numbers(the last digit is not 8).

Relevant code:

```
CKD     AND R5, R5, #0    ; CheckK for Divisibility
        AND R5, R1, R5
DIVE    ADD R5, R5, #-8   ; DIVide by Eight
        BRp DIVE
        BRz FLIP          ; Divisible by 8
```

```
6
7  CKLD   AND R5, R5, #0    ; CheckK for the Last Digit
8         ADD R5, R1, R5
9  DIVT   ADD R5, R5, #-10  ; DIVide by Ten
10        BRp DIVT
11
12        ADD R5, R5, #2    ; If the last digit is 8, R5 should be #-2 now
13        BRnp CHOOSE       ; The last digit is not 8
```

3 Procedure

3.1 Bugs encountered

1. When using LD instruction to load N to R0, R0 is loaded with x3102, not the content of memory location x3102:

```
1         LD R0, N
2  N      .FILL x3102
```

   Reason & Solution: The memory location to be visited stores x3102, which means we should use LDI instruction.

2. When computing a certain term of the sequence $f(N)$, the program gives $f(N+1)$. Reason: Every time the program finishes one loop, R0 substracts 1 and the program determines where to go:

```
1  TERM   ADD R0, R0, #-1
2         BRp CKD           ; Check the conditions of changing the operand
3         STI R1, DEST      ; Computation completed, store the result
4         TRAP x25
```

   This part was placed wrongly at the end of a loop, causing the program to iterate 1 more time.

   Solution: Move the part labeled "TERM" before the loop.

3.2 Chanllenges

The major chanllenge I encountered was examining the last digit.

One trivial approach, which is adopted in the program, is to perform division and observe the remainder. But this approach has an obvious flaw: when N becomes larger, numbers of operations performed would skyrocket.

Due to the fact that when $N > 20$, $f(N) \equiv (4094, +)$, One possible solution is to directly give the result if $N > 20$. (Result Oriented Programming, lol)

4 Results

Results are shown below:

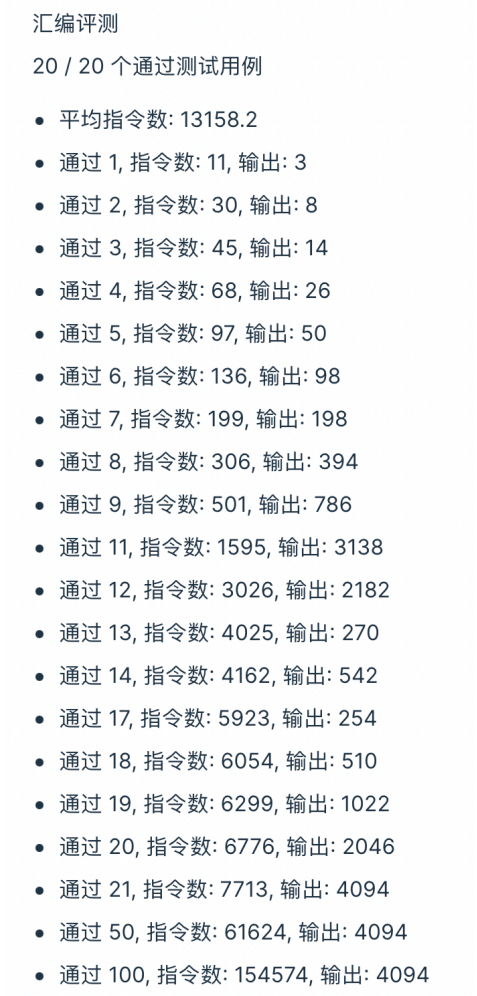

Figure 1: Result

5 Improvements

We can improve efficiency by optimizing the checking process. As is mentioned above, checking for the last digit requires lots of operations when N is big. So a better algorithm for division might significantly improve efficiency.

One possible approach is to enumerate all possible values of a 16-bit binary number from MSB. In this way we can obtain the result within 16 operations.

This approach is not adopted in the program due to limited time and its complexity.