

Lab2

实验目的

- 本次实验我们需要实现如下的特殊的斐波那契数列:

$$F(N) = F(N - 2) \% p + F(N - 1) \% q$$

$$p = 2^k (2 \leq k \leq 10), 10 \leq q \leq 1024$$

- 本次实验我们希望能熟悉汇编指令
- 熟悉LC3 TOOL的使用方法

实验原理

- 因为LC3没有原生的减法指令，所以我们采用了如下的方式实现 $R2 = R1 - R0$

```
NOT R3, R0
ADD R3, R3, #1
ADD R2, R1, R3
```

- 同理LC3没有原生的mod指令，我们采用如下的方式实现 $R2 = R1 \bmod R0$

```
NOT R3, R0
ADD R3, R3, #1
LOOP ADD R1, R1, R3 ;先执行减法操作
BRzp LOOP ;如果得到的数是大于等于零的，继续执行减法操作，直到得到小于等于零的结果
ADD R2, R1, R0 ;得到最后的取模的值
```

- 我们自定义了两个地址 `a_n` 和 `a_{n+1}` 用于存储每次计算后的结果
我们将 `a_n` 和 `a_{n+1}` 在每次计算出结果后更新
- 需要初始化R2的值，因为我们不是从 $N=0$ 开始算起的

实验过程

本次实验较为简单，仅有两处debug的过程

- 第一处在于设置 `LABEL` 的时候，未充分考虑到 `offset` 的大小限制，导致了编译失败，重新更改 `LD` 的位置即可解决！
- 第二处在于未充分考虑到 `N` 是从2开始的，不需要循环N次，需要对 `N` 进行初始化

实验结果

$$1. N = 100, p = 256, q = 123$$

LC3Tools v2.2.2 Application Edit View

Registers

Register	Address	Value
R0	0000	0
R1	0001	32767
R2	0002	0
R3	0003	256
R4	0004	0
R5	0005	123
R6	0006	12288
R7	0007	0
PC	0000	2
MCR	0000	0

Memory

Address	Value
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0
29	0
30	0
31	0
32	0
33	0
34	0
35	0
36	0
37	0
38	0
39	0
40	0
41	0
42	0
43	0
44	0
45	0
46	0
47	0
48	0
49	0
50	0
51	0
52	0
53	0
54	0
55	0
56	0
57	0
58	0
59	0
60	0
61	0
62	0
63	0
64	0
65	0
66	0
67	0
68	0
69	0
70	0
71	0
72	0
73	0
74	0
75	0
76	0
77	0
78	0
79	0
80	0
81	0
82	0
83	0
84	0
85	0
86	0
87	0
88	0
89	0
90	0
91	0
92	0
93	0
94	0
95	0
96	0
97	0
98	0
99	0

Console (click to focus)

```

--- Waiting the I/O ---

--- Waiting the I/O ---


```

PC: 0

$$2. N = 200, p = 512, q = 456$$

LC3Tools v2.2.2 Application Edit View

Registers

Register	Address	Value
R0	0000	0
R1	0001	32767
R2	0002	0
R3	0003	512
R4	0004	0
R5	0005	456
R6	0006	12288
R7	0007	0
PC	0000	2
MCR	0000	0

Memory

Address	Value
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0
29	0
30	0
31	0
32	0
33	0
34	0
35	0
36	0
37	0
38	0
39	0
40	0
41	0
42	0
43	0
44	0
45	0
46	0
47	0
48	0
49	0
50	0
51	0
52	0
53	0
54	0
55	0
56	0
57	0
58	0
59	0
60	0
61	0
62	0
63	0
64	0
65	0
66	0
67	0
68	0
69	0
70	0
71	0
72	0
73	0
74	0
75	0
76	0
77	0
78	0
79	0
80	0
81	0
82	0
83	0
84	0
85	0
86	0
87	0
88	0
89	0
90	0
91	0
92	0
93	0
94	0
95	0
96	0
97	0
98	0
99	0

Console (click to focus)

```

--- Waiting the I/O ---

--- Waiting the I/O ---

warning: 16730: Skipping 'Updating Keyboard' scheduled for 16730
warning: 16730: Skipping 'Updating Display' scheduled for 16730
warning: 16730: Skipping 'No interrupt of higher priority pending' scheduled for 16730

--- Waiting the I/O ---


```

PC: 0

$$3. N = 300, p = 1024, q = 789$$

LC3Tools v2.2.2 Application Edit View

Registers

Register	Address	Value
R0	0000	0
R1	0001	32767
R2	0002	0
R3	0003	1024
R4	0004	0
R5	0005	789
R6	0006	12288
R7	0007	0
PC	0000	2
MCR	0000	0

Memory

Address	Value
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0
29	0
30	0
31	0
32	0
33	0
34	0
35	0
36	0
37	0
38	0
39	0
40	0
41	0
42	0
43	0
44	0
45	0
46	0
47	0
48	0
49	0
50	0
51	0
52	0
53	0
54	0
55	0
56	0
57	0
58	0
59	0
60	0
61	0
62	0
63	0
64	0
65	0
66	0
67	0
68	0
69	0
70	0
71	0
72	0
73	0
74	0
75	0
76	0
77	0
78	0
79	0
80	0
81	0
82	0
83	0
84	0
85	0
86	0
87	0
88	0
89	0
90	0
91	0
92	0
93	0
94	0
95	0
96	0
97	0
98	0
99	0

Console (click to focus)

```

--- Waiting the I/O ---

--- Waiting the I/O ---

warning: 16730: Skipping 'Updating Keyboard' scheduled for 16730
warning: 16730: Skipping 'Updating Display' scheduled for 16730
warning: 16730: Skipping 'No interrupt of higher priority pending' scheduled for 16730

--- Waiting the I/O ---

warning: 24970: Skipping 'Updating Keyboard' scheduled for 24970
warning: 24970: Skipping 'Updating Display' scheduled for 24970
warning: 24970: Skipping 'No interrupt of higher priority pending' scheduled for 24970

--- Waiting the I/O ---


```

PC: 0

Answer to the question:

How can you improve the efficiency of loop structure in your program?

- 我们使用了两个地址来存取 a_n 和 a_{n+1} 的值，避免了过多寄存器的使用，以及方便程序的编写和理解
- 我们在每次循环中只需正确维护 a_n 和 a_{n+1} 的值即可，其中 a_n 的值很好维护，只需要将其等于上一个 a_{n+1} 即可
- 我们对 a_n 和 a_{n+1} 分别取模，相加即可得到最终的值