lab2

实验目的

1. 进一步熟悉LC-3指令集, 能够实现减法、取模等指令集中没有的指令.
2. 学习编写汇编代码, 学会利用控制指令改变程序执行的顺序, 达到期望的`if else`结构等等.
3. 锻炼调试程序的能力.
4. 体会不同实现方法对最终指令数的影响.

实现方法

基本思路

1. 用R0记录v_n的值, 依次从v_1计算到v_N. 初始化为$f(1)=3$.
2. R1记录d_n, 但为方便计算, 直接存储2或-2. 初始化为2. 后面依情况翻转为相反数 (取反加1即可).
   代码如下:

```
AND R0, R0, #0
ADD R0, R0, #3  ; init R0 (v) to 3
AND R1, R1, #0
ADD R1, R1, #2  ; init R1 (d) to 2
...
FLIP NOT R1, R1
     ADD R1, R1, #1  ; R1 = -R1
```

对4096取模

1. 容易看出, 当$n\geq 2$时, v_n为偶数, 因此在模4096的情况下, $v_n\leq 4094$, 那么$2v_n\leq 4094\times 2=8188<8192=4096\times 2$.
2. 由此, 只需要在每次对v_n乘2后减去4096 (若超过了4096), 即可达到模4096的效果.
3. 同时注意到, $(2v_n-4096)+2\leq 8188-4096+2\leq 4094$, 因此下一步即使+2, 也无需再模4096.
   代码如下:

```
LD R7, MOD      ; load -4096 into R7
...
ADD R0, R0, R0       ; R0 *= 2
```

```
ADD R4, R0, R7       ; R4 = R0 - R7
BRn NEXT             ; if R0 < 4096, no need to subtract 4096
ADD R0, R0, R7       ; R0 -= 4096
NEXT ADD R0, R0, R1  ; R0 += d
...
MOD    .FILL #-4096
```

判断是否能被8整除

一个能被8整除的二进制数的最低3位应该为000, 因此可以将其与111 (7) 做按位与, 若结果为0说明可以被8整除. 代码如下:

```
MOD8 AND R4, R0, #7  ; v % 8
```

判断个位是否为8

1. 使用R6记录每一次计算所得v_n的个位.
2. 如果这一轮计算v_n时没有减去4096 (使用R5记录是否减去了4096) , 那么R6就更新为$(R6 \pm 2)\%10$.
3. 否则, R6还需要减去6 (4096的个位).
4. 由于每一轮中, R6最大可能要减去8, 因此其乘2后若小于8, 就先加上10 (相当于借位). 最后得到的R6如果≥10, 还需要减去10, 保证其为一位数.
   代码如下:

```
ADD R6, R6, R6             ; R6 *= 2
ADD R4, R6, #-8            ; R4 = R6 - 8
BRzp TWO                   ; if R6 * 2 >= 8, no need to borrow digit
ADD R6, R6, #10            ; borrow digit
TWO ADD R6, R6, R1         ; R6 += d
ADD R5, R5, #0             ; set cc according to R5
BRz MINUS10                ; if R5 == 0, skip the next line
ADD R6, R6, #-6            ; R6 -= 6 (4096)
MINUS10 ADD R4, R6, #-10   ; R6 - 10
BRn JUDGE                  ; if R6 < 10, no need to substract 10
ADD R6, R6, #-10           ; R6 -= 10
JUDGE ADD R4, R6, #-8      ; R4 = R6 - 8
      BRnp MOD8            ; if R6 != 8, jump to MOD8
      FLIP NOT R1, R1
      ADD R1, R1, #1       ; R1 = -R1
      BR LOOP
```

加载与存储数据

1. 本题要用到的-4096超出了5位立即数能表示的范围, 因此不能利用ADD加载进寄存器, 需要用LD指令, 代码如下:

```
LD R7, MOD      ; load -4096 into R7
...
MOD    .FILL #-4096
```

2. 使用LDI指令将N从x3102处加载进寄存器R2, 代码如下:

```
LDI R2, N       ; load N into R2
...
N      .FILL x3102
```

3. 使用STI指令将结果从R0存储到x3103, 代码如下:

```
DONE STI R0, RESULT  ; store result
...
RESULT .FILL x3103
```

完整代码

```
.ORIG x3000

LDI R2, N       ; load N into R2
LD R7, MOD      ; load -4096 into R7

AND R0, R0, #0
ADD R0, R0, #3  ; init R0 (v) to 3
AND R1, R1, #0
ADD R1, R1, #2  ; init R1 (d) to 2
AND R6, R6, #0
ADD R6, R6, #3  ; init R6 (last digit of v) to 3

LOOP ADD R2, R2, #-1      ; decrement R2
     BRz DONE             ; R2 == 0 -> done
     ADD R0, R0, R0       ; R0 *= 2
     AND R5, R5, #0       ; clear R5 (a flag indicates whether v subtracts 4096)
     ADD R4, R0, R7       ; R4 = R0 - R7
     BRn NEXT             ; if R0 < 4096, no need to subtract 4096
     ADD R0, R0, R7       ; R0 -= 4096
     ADD R5, R5, #1       ; R5 (flag) = 1
```

```
     NEXT ADD R0, R0, R1  ; R0 += d

     ADD R6, R6, R6             ; R6 *= 2
     ADD R4, R6, #-8            ; R4 = R6 - 8
     BRzp TWO                   ; if R6 * 2 >= 8, no need to borrow digit
     ADD R6, R6, #10            ; borrow digit
     TWO ADD R6, R6, R1         ; R6 += d
     ADD R5, R5, #0             ; set cc according to R5
     BRz MINUS10                ; if R5 == 0, skip the next line
     ADD R6, R6, #-6            ; R6 -= 6 (4096)
     MINUS10 ADD R4, R6, #-10   ; R6 - 10
     BRn JUDGE                  ; if R6 < 10, no need to minus 10
     ADD R6, R6, #-10           ; R6 -= 10
     JUDGE ADD R4, R6, #-8      ; R4 = R6 - 8
           BRnp MOD8            ; if R6 != 8, jump to MOD8
           FLIP NOT R1, R1
                ADD R1, R1, #1  ; R1 = -R1
                BR LOOP

     MOD8 AND R4, R0, #7  ; v % 8
          BRz FLIP        ; if v % 8 == 0, flip d
          BR LOOP

DONE STI R0, RESULT  ; store result
TRAP x25             ; halt

N      .FILL x3102
RESULT .FILL x3103
MOD    .FILL #-4096

.END
```

遇到的困难与解决方案

判断个位是否是8

我原先的想法非常朴素, 将求得的v_n逐次减去10, 直至一位数, 从而判断它个位是否为10:

```
LASTDIGIT ADD R4, R5, #-8   ; R5 - 8
          BRz FLIP          ; last digit is 8
          BRn LOOP          ; last digit is not 8
```

```
            ADD R5, R5, #-10  ; R5 -= 10
            BR LASTDIGIT
```

然而进行评测时发现, 当N较大的时候程序非常慢. 其实可想而知, 后面v_n稳定在4094, 那么这一步就要运行400多遍, 显然效率是很低的.
由于我暂时想不到快速对10取模的方法, 因此采取了"实现方法"部分描述的, 通过前一次的个位数推断出这一次的个位数.

推断个位数时没有考虑到对4096的取模

我的初版程序在$N = 1 \sim 12$时都运行正确, 然而到$N = 13$时就出现了错误, $f(13)$应当为270, 然而我的程序输出的是266. 经过单步调试, 我发现我的程序在$N = 13$时推断出的个位数是错的, 导致方向计算出现了错误. 进一步考察, 发现$f(11) = 3138$, 因此计算$f(12)$时是进行了MOD 4096的, 而我在推断个位数的时候并没有考虑到这一点. 由此我增加了对"是否进行了MOD 4096"的判断, 得到了如"实现方法"部分所描述的正确程序.

评测结果

使用数据`1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 54, 100, 9999`在自动评测机上进行评测, 结果如下:

汇编评测

26 / 26 个通过测试用例

- 平均指令数: 9195.384615384615
- 通过 1, 指令数: 11, 输出: 3
- 通过 2, 指令数: 33, 输出: 8
- 通过 3, 指令数: 54, 输出: 14
- 通过 4, 指令数: 74, 输出: 26
- 通过 5, 指令数: 95, 输出: 50
- 通过 6, 指令数: 116, 输出: 98
- 通过 7, 指令数: 137, 输出: 198
- 通过 8, 指令数: 158, 输出: 394
- 通过 9, 指令数: 178, 输出: 786
- 通过 10, 指令数: 199, 输出: 1570
- 通过 11, 指令数: 220, 输出: 3138
- 通过 12, 指令数: 244, 输出: 2182
- 通过 13, 指令数: 269, 输出: 270
- 通过 14, 指令数: 291, 输出: 542
- 通过 15, 指令数: 313, 输出: 1086
- 通过 16, 指令数: 334, 输出: 2174
- 通过 17, 指令数: 357, 输出: 254
- 通过 18, 指令数: 378, 输出: 510
- 通过 19, 指令数: 400, 输出: 1022
- 通过 20, 指令数: 422, 输出: 2046
- 通过 21, 指令数: 443, 输出: 4094
- 通过 22, 指令数: 466, 输出: 4094
- 通过 23, 指令数: 489, 输出: 4094
- 通过 54, 指令数: 1202, 输出: 4094
- 通过 100, 指令数: 2260, 输出: 4094
- 通过 9999, 指令数: 229937, 输出: 4094

改进

- 目前, 判断最后一位数是不是8的部分仍旧比较复杂, 指令数较多, 因为在维护个位数的过程中涉及到借位与进位等操作, 可以针对此对某些情况进行合并, 精简指令数量. 此外, 如果有更好的快速求个位数的方法, 也无需如此麻烦的操作.

- 此外, pingpong序列在N较大的时候稳定在4094 (见下节) , 所以对于较大的N其实根本无需循环计算, 可以直接得出结果.

Pingpong序列的规律

编写Python程序, 计算$N = 1 \sim 999$时的$f(N)$值:

```
def pingpong(v, d):
    if d:
        v_next = (2 * v + 2) % 4096
    else:
        v_next = (2 * v - 2) % 4096

    if v_next % 8 == 0 or v_next % 10 == 8:
        d_next = not d
    else:
        d_next = d

    return v_next, d_next

v = 3
d = True

with open("result.txt", "w") as f:
    for i in range(1, 1000):
        f.write("f(" + str(i) + ")=" + str(v) + " direction=" + str(d) + "\n")
        v, d = pingpong(v, d)
```

结果如下:

```
f(1)=3 direction=True
f(2)=8 direction=False
f(3)=14 direction=False
f(4)=26 direction=False
f(5)=50 direction=False
f(6)=98 direction=True
f(7)=198 direction=False
f(8)=394 direction=False
f(9)=786 direction=False
f(10)=1570 direction=False
f(11)=3138 direction=True
f(12)=2182 direction=True
f(13)=270 direction=True
f(14)=542 direction=True
f(15)=1086 direction=True
```

```
f(16)=2174 direction=True
f(17)=254 direction=True
f(18)=510 direction=True
f(19)=1022 direction=True
f(20)=2046 direction=True
f(21)=4094 direction=True
f(22)=4094 direction=True
f(23)=4094 direction=True
f(24)=4094 direction=True
...
```

发现从$N=21$开始, $f(N)$稳定在4094. 这是因为$(2\times 4094+2)\%4096=4094$, 而4094既不是8的倍数, 个位也不是8, 因此方向保持为↑不变.