

# Lab02: The PingPong Sequence

PB22151796-莫环欣

## 实验目的

- 本次实验提到了一种名为“PingPong Sequence”的序列
  - 这个序列的特点是，当序列的索引是 8 的倍数或者包含数字 8 时，序列的增减方向会发生改变。这类序列常常用于教授递归的概念。
- 在实验中，我们需要使用 LC-3 汇编语言来计算简单情况下的乒乓序列某位置的值

## 实现原理

- 教程中给出的实现原理如下方的 Python 代码

```
def calculate_next_term(v_n: int, d_n: bool):  
    if d_n:  
        v_next = 2 * v_n + 2  
    else:  
        v_next = 2 * v_n - 2  
    if v_next % 8 == 0 or last_digit(v_next) == 8:  
        d_next = not d_n  
    else:  
        d_next = d_n  
    return v_next, d_next
```

- 在我们的第一版代码中。通过以下方法处理特定的运算
  - i.  $v\_next = 2 * v\_n + 2$ 
    - 先使用 ADD R1, R1, R1 指令将存有值的寄存器（这里是 R1）倍增
    - 再使用 ADD R1, R1, #2 指令将倍增后的寄存器值加 2
  - ii.  $v\_next = 2 * v\_n - 2$ 
    - 先使用 ADD R1, R1, R1 指令将存有值的寄存器（这里是 R1）倍增
    - 再使用 ADD R1, R1, #-2 指令将倍增后的寄存器值减 2
  - iii. 判断是否需要翻转（是否为 8 的倍数或十进制表示尾数是否为 8）
    - 使用了 CHECK、MODULO、D\_MODULO、MODULO\_EXTRA 四个标签
    - 通过模拟 模运算 进行判断
    - 将当前等待判断的值从 R1 复制到 R3，避免修改原值
      - 先清空( AND R3, R3, #0 )再复制( ADD R3, R1, #0 )

- 接下来跳转到进行模运算的标签 MODULO
- 在 MODULO 中,
  - 循环让 R4 减去 8
  - 循环让 R3 减去 10
  - 直到  $R3 \leq 0$ , 跳转 MODULO\_EXTRA 标签
    - 模 10 的速度显然比模 8 快
- 在 MODULO\_EXTRA 中,
  - 我们继续循环减 8
  - 直到  $R4 \leq 0$ , 跳转 D\_MODULO 标签
- 在 D\_MODULO 标签中,
  - $R4 = 0$  时, 待判断的值才可能是 8 的倍数
    - 是则跳转 FLIP 进行翻转
  - 上一步得到的 R3 并非余数, 加上 10 才是余数
    - 将其加上 2 之后进行判断 ( $+10 - 8 = 2$ )
      - 若  $R3 = 0$ , 说明余数为 8, 跳转 FLIP 进行翻转

iv. 使用 ADD R2, R2, #0 判断 R2 的正负并跳转到对应标签

v. 对于“ $f(N) \leq 4096$ ”的约束

- 在结尾存储了 MAX\_VALUE 和 MAX\_VALUE\_N 两个值
  - 前者为最大值 #4096, 后者是其负数 #-4096
  - 起始时分别存入 R7 和 R6
- 使用了 CHECK\_MAX 和 MAXX\_ADD 两个标签
  - 前者使用 ADD R1, R1, R6 判断当前计算结果是否超出最大值
    - 若超出则循环减去 4096
    - 当  $R1 = 0$  时, 进入正常操作范围, 跳转到 CHECK
    - 当指令使得  $R1 < 0$  时, 进入 MAX\_ADD
  - 后者只能从前者跳转进入
    - 作用是为 R1 加上 4096 使得其回到正常操作范围
    - 接着跳转到 CHECK 正常操作

vi. 对于计算结束时机的判断

- 直接对 R1 中存储的 N 的值进行递减
  - 此时将 N 视为我们剩下还需要操作的次数
- 初始化时减一次
- 之后每次调用 CHECK 时 都递减一次
- 当  $R1 = 0$  时, 说明我们接下来不需要再向后进行计算, 结束操作

- v2 版本代码较 v1 的主要改动为模运算的循环部分, 通过设置梯度来提高计算速率, 详见后文

# 汇编代码

最终用于评价的代码请取`Vertion 2`  
由于`Vertion 1`的表述比较清晰，且两版本的实现方式类似，  
故这里把`ver 1`的汇编码也给出

# Version 1 未优化循环的算法

```
.ORIG x3000
    LDI R0, N_ADRESS ; Load the value of N from memory location x3102 into R0
    LD R7, MAX_VALUE ; Load the MAX value
    LD R6, MAX_VALUE_N ; Load the Negative MAX value

    AND R1, R1, #0 ; Clear R1
    AND R2, R2, #0 ; Clear R2
    ADD R1, R1, #3 ; Set the value of v_1 into R1
    ADD R2, R2, #1 ; Set the value of d_1 into R2

    ADD R0, R0, #-1 ; When N = 1
    BRz END ; Finish
    BRp POSITIVE ; N > 1

NEGATIVE
    ADD R1, R1, R1 ; v_next = 2 * v_n
    ADD R1, R1, #-2 ; v_next = v_next - 2
    BRnzp CHECK_MAX ; Branch to CHECK_MAX

POSITIVE
    ADD R1, R1, R1 ; v_next = 2 * v_n
    ADD R1, R1, #2 ; v_next = v_next + 2
    BRnzp CHECK_MAX ; Check if R1 is in normal range

; -----
CHECK
    ADD R0, R0, #-1 ; N = N - 1 ; An operation was done
    BRz END ; Finish

    AND R3, R3, #0 ; Clear R3
    AND R4, R4, #0 ; Clear R4
    ADD R3, R1, #0 ; Copy v_next to R3
    ADD R4, R1, #0 ; Copy v_next to R4
    BRnzp MODULO

; R3 mod 10 And mod 8
MODULO
    ADD R4, R4, #-8 ; R4 = R4 - 8
    ADD R3, R3, #-10 ; R3 = R3 - 10
    BRnz MODULO_EXTRA ; Mod 10 is faster than 8
    BRp MODULO ; Branch back to MODULO Loop

; Deal with mod 8
MODULO_EXTRA
```

```

    ADD R4, R4, #-8    ; Subtract 8 from R4
    BRnz D_MODULO      ; If R4 <= 0, branch to D_MODULO
    BRp MODULO_EXTRA

; Check the remainder
D_MODULO
    ADD R4, R4, #0     ; Check if result_n is 8 times
    BRz FLIP

    ADD R3, R3, #2     ; R3 stores the remainder now
                        ; Check if the remainder is 8
    BRz FLIP

    ADD R2, R2, #0     ; Check the flip flag d_n
    BRp POSITIVE
    BRn NEGATIVE

; -----
; Check if R1 is in normal range
CHECK_MAX
    ADD R1, R1, R6
    BRp CHECK_MAX
    BRz CHECK
    BRn MAX_ADD

; Add R1 to normal range
MAX_ADD
    ADD R1, R1, R7
    BRp CHECK
; -----

FLIP
    NOT R2, R2         ; Flip the bits of d_n
    ADD R2, R2, #1     ; Add 1 to get the logical NOT of d_n
    BRp POSITIVE
    BRn NEGATIVE

END

    STI R1, RESULT     ; Store the value of v_next in memory location x3103
    TRAP x25           ; HALT

MAX_VALUE .FILL #4096
MAX_VALUE_N .FILL #-4096
N_ADRESS .FILL x3102 ; Memory location for N
RESULT .FILL x3103   ; Memory location for result

```

. END

## Version 2 优化循环后的算法

最终用于评估的代码

具体优化方法见后文“改进”小节

.ORIG x3000

LDI R0, N\_ADRESS ; Load the value of N from memory location x3102 into R0

LD R7, MAX\_VALUE ; Load the MAX value

LD R6, MAX\_VALUE\_N ; Load the Negative MAX value

AND R1, R1, #0 ; Clear R1

AND R2, R2, #0 ; Clear R2

ADD R1, R1, #3 ; Set the value of v\_1 into R1

ADD R2, R2, #1 ; Set the value of d\_1 into R2

ADD R0, R0, #-1 ; When N = 1

BRz END ; Finish

BRp POSITIVE ; N > 1

NEGATIVE

ADD R1, R1, R1 ; v\_next = 2 \* v\_n

ADD R1, R1, #-2 ; v\_next = v\_next - 2

BRnzp CHECK\_MAX ; Branch to CHECK\_MAX

POSITIVE

ADD R1, R1, R1 ; v\_next = 2 \* v\_n

ADD R1, R1, #2 ; v\_next = v\_next + 2

BRnzp CHECK\_MAX ; Check if R1 is in normal range

; -----

CHECK

ADD R0, R0, #-1 ; N = N - 1      An operation was done

BRz END ; Finish

AND R3, R3, #0 ; Clear R3

ADD R3, R1, #0 ; Copy v\_next to R3

ADD R4, R2, #0 ; Copy d\_next to R4

BRnzp MODULO

; -----

; Faster R3 mod operation

MODULO

LD R5, A\_THOUSAND ; R5 <- -1000

BRn MODULO\_AT ; while(R3 - 1000)

; R3 mod 1000

MODULO\_AT

ADD R3, R3, R5 ; R3 = R3 - 1000

BRn AT\_ADD

BRp MODULO\_AT ; Branch back to MODULO\_AT      Loop

BRz D\_MODULO\_DIGIT ; R3 = #1000

; R3 + 1000

AT\_ADD

```
NOT R5, R5      ; Flip the bits of R5
ADD R5, R5, #1   ; Add 1 to get the logical NOT of R5
ADD R3, R3, R5    ; R3 = R3 + 1000
LD R5, A_HUNDRED  ; R5 <- -100
BRp MODULO_AH
```

; R3 mod 100

MODULO\_AH

```
ADD R3, R3, R5    ; R3 = R3 - 100
BRn AH_ADD
BRp MODULO_AH      ; Branch back to MODULO_AH    Loop
BRz D_MODULO_DIGIT ; R3 = #100
```

; R3 + 100

AH\_ADD

```
NOT R5, R5      ; Flip the bits of R5
ADD R5, R5, #1   ; Add 1 to get the logical NOT of R5
ADD R3, R3, R5    ; R3 = R3 + 100
BRp MODULO_TEN
```

; R3 mod 10

MODULO\_TEN

```
ADD R3, R3, #-10   ; R3 = R3 - 10
BRp MODULO_TEN      ; Branch back to MODULO_TEN    Loop
BRnz D_MODULO_DIGIT ; R3 = #8 || #10
```

; Check the remainder

D\_MODULO\_DIGIT

```
ADD R3, R3, #2     ; R3 stores the remainder now
                   ; Check if the remainder is 8
BRz FLIP
```

```
AND R3, R3, #0      ; Clear R3
ADD R3, R1, #0       ; R3 = R1
LD R5, CODE_MONKEY   ; R5 <- -1024
BRn MODULO_CM        ; while(R3 - 1024)
```

; -----

; R3 mod 1024

MODULO\_CM

```
ADD R3, R3, R5      ; R3 = R3 - 1024
BRn CM_ADD
BRp MODULO_CM        ; Branch back to MODULO_CM    Loop
```



```

BRz D_MODULO      ; R3 = #1024

; R3 + 1024
CM_ADD
    NOT R5, R5      ; Flip the bits of R5
    ADD R5, R5, #1   ; Add 1 to get the logical NOT of R5
    ADD R3, R3, R5    ; R3 = R3 + 1024
    LD R5, YAO_ER_BA  ; R5 <- -128
    BRp MODULO_YEB

; R3 mod 128
MODULO_YEB
    ADD R3, R3, R5    ; R3 = R3 - 128
    BRn YEB_ADD
    BRp MODULO_YEB    ; Branch back to MODULO_YEB    Loop
    BRz D_MODULO      ; R3 = #128

; R3 + 128
YEB_ADD
    NOT R5, R5      ; Flip the bits of R5
    ADD R5, R5, #1   ; Add 1 to get the logical NOT of R5
    ADD R3, R3, R5    ; R3 = R3 + 128
    BRp MODULO_EIGHT

; R3 mod 8
MODULO_EIGHT
    ADD R3, R3, #-8   ; R3 = R3 - 8
    BRp MODULO_EIGHT ; Branch back to MODULO_EIGHT Loop
    ADD R3, R3, #-2   ; R3 = R3 - 2    When R3 is 0, Set it to -2
    BRnz D_MODULO     ; R3 <= #8

; -----

; Check the remainder
D_MODULO
    ADD R3, R3, #2    ; R3 stores the remainder now
                    ; Check if the remainder is 8
    BRz FLIP

    ADD R2, R2, #0    ; Check the flip flag d_n
    BRp POSITIVE
    BRn NEGATIVE

; -----

; Check if R1 is in normal range

```

```

CHECK_MAX
    ADD R1, R1, R6
    BRp CHECK_MAX
    BRz CHECK
    BRn MAX_ADD

; Add R1 to normal range
MAX_ADD
    ADD R1, R1, R7
    BRp CHECK
; -----

FLIP
    NOT R2, R2      ; Flip the bits of d_n
    ADD R2, R2, #1   ; Add 1 to get the logical NOT of d_n
    BRp POSITIVE
    BRn NEGATIVE

END
    STI R1, RESULT   ; Store the value of v_next in memory location x3103
    TRAP x25         ; HALT

MAX_VALUE    .FILL #4096
MAX_VALUE_N  .FILL #-4096
A_THOUSAND   .FILL #-1000
A_HUNDRED    .FILL #-100
CODE_MONKEY  .FILL #-1024
YAO_ER_BA    .FILL #-128
N_ADRESS     .FILL x3102 ; Memory location for N
RESULT       .FILL x3103 ; Memory location for result

.END

```

## 结果

在线进行了提交测试，样例全部通过，且优化后的算法 ver 2 提交后的响应速度明显快于 ver 1

# Lab2 提交结果

home.ustc.edu.cn 显示  
All test cases passed!

确定

Memory

Jump to address or label

Manage Labels

0x	Label	Hex	Instruction
<input checked="" type="checkbox"/>	x3000	xA05D	LDI R0, N_ADDRESS
<input type="checkbox"/>	x3001	x2E56	LD R7, MAX_VALUE
<input type="checkbox"/>	x3002	x2C56	LD R6, MAX_VALUE_N
<input type="checkbox"/>	x3003	x5260	AND R1, R1, #0
<input type="checkbox"/>	x3004	x54A0	AND R2, R2, #0
<input type="checkbox"/>	x3005	x1263	ADD R1, R1, #3
<input type="checkbox"/>	x3006	x14A1	ADD R2, R2, #1
<input type="checkbox"/>	x3007	x103F	ADD R0, R0, #-1
<input type="checkbox"/>	x3008	x044D	BRz END
<input type="checkbox"/>	x3009	x0203	BRp POSITIVE
<input type="checkbox"/>	x300A NEGATIVE	x1241	ADD R1, R1, R1
<input type="checkbox"/>	x300B	x127E	ADD R1, R1, #-2
<input type="checkbox"/>	x300C	x0E3F	BRnzp CHECK_MAX
<input type="checkbox"/>	x300D POSITIVE	x1241	ADD R1, R1, R1

Status

Registers

R0: x0000R4: x0000PC: x3000

R1: x0000R5: x0000IR: x0000

R2: x0000R6: x0000PSR: x8002

R3: x0000R7: x0000CC: Z

Clear R0-R7Reset all registers

StepNextFinishRunPauseContinueUnhalt

☒ Follow PC

Lab 2Submit

Console

# Lab2 light 提交结果

home.ustc.edu.cn 显示  
All test cases passed!

确定

Memory

Jump to address or label

Manage Labels

0x	Label	Hex	Instruction
<input type="checkbox"/>	x3052 FLIP	x94BF	NOT R2, R2
<input type="checkbox"/>	x3053	x14A1	ADD R2, R2, #1
<input type="checkbox"/>	x3054	x03B8	BRp POSITIVE
<input type="checkbox"/>	x3055	x09B4	BRn NEGATIVE
<input type="checkbox"/>	x3056 END	xB208	STI R1, RESULT
<input checked="" type="checkbox"/>	x3057	xF025	HALT
<input type="checkbox"/>	x3058 MAX_VALUE	x1000	ADD R0, R0, R0
<input type="checkbox"/>	x3059 MAX_VALUE_N	xF000	TRAP x00
<input type="checkbox"/>	x305A A_THOUSAND	xFC18	.FILL xFC18
<input type="checkbox"/>	x305B A_HUNDRED	xFF9C	.FILL xFF9C
<input type="checkbox"/>	x305C CODE_MONKEY	xFC00	.FILL xFC00
<input type="checkbox"/>	x305D YAO_ER_BA	xFF80	.FILL xFF80
<input type="checkbox"/>	x305E N_ADDRESS	x3102	ST R0, x2F61
<input type="checkbox"/>	x305F RESULT	x3103	ST R0, x2F63

Status

Registers

R0: x0000R4: xFFFFPC: x3057

R1: x0032R5: x0080IR: x0000

R2: xFFFFR6: xF000PSR: x8002

R3: xFFFA R7: x1000CC: Z

Clear R0-R7Reset all registers

StepNextFinishRunPauseContinueUnhalt

☒ Follow PC

Lab 2 lightSubmit

Console

# 改进

## 1. 在进行模运算循环时，以模十举例

- 考虑到我们的乒乓序列每次操作都会把原数倍增，数值增长得很快
- 故我们可以在 MODULO 模块中增大减数的步长，上调到其的  $n$  次方
- 设置为其它倍也可以
- 例如，下面我们尝试计算  $128 \bmod 10$ 
  - 当步长为  $n$  次方 时
    - 假设步长设为 100，
    - 一次减得 28，步长改为 10，之后的处理与直接  $\bmod 10$  一样，但速度更快
  - 当步长为某倍时
    - 假设步长为 30
    - 减五次得 -22
    - 加回步长得 8，接下来回到正常  $\bmod 10$  的操作
      - 循环减 10 直到小于零，再加回正数即得余数

## 2. 经过上面的分析后我们发现，还可以进一步加速计算

- 将模 8 与模 10 操作分开，我们就能拥有两个空闲寄存器
  - 同时，这样处理之后，当满足其中一种翻转情况时，可以减少不必要的循环操作
- 再将 R6 删除，R7 存储 #4096，并借鉴 FLIP 标签的方法求 -R7 的值
- 这样我们就有了三个空闲寄存器 R4、R5、R6 来存储值（其实一个即可）
- 之后我们可以为模运算设置梯度减数
  - 例如设置 1000、100、20 三个梯度，在三个标签中进行减法运算
  - 三个梯度的值（下称“梯度模数”）存储在地址中
  - 需要进行操作时从地址中将对应的“梯度模数”读入
  - 例如，我们想求 #3659 的余数
    - 先进入 模1000 标签，
      - 从地址中读取 1000 到寄存器中
      - 循环减去这个值
      - 直到得到负数
    - 再进入 模100 标签，
      - 先利用寄存器内的值（现在还是 1000）与待操作数相加，得到 659
      - 再从地址中读取 100 到寄存器中
      - 循环减去这个值
      - 直到得到负数
    - 再进入 模20 标签，
      - 先利用寄存器内的值（现在还是 100）与待操作数相加，得到 59
      - 再从地址中读取 20 到寄存器中
      - 循环减去这个值
      - 直到得到负数
    - 进入我们上面所说的 ADD\_TO\_P 标签

- 将待操作数与 10 循环相加，直到得到正数
    - 此时就得到了余数（这里是 1）
  - 分析，在上面的改进算法中，对于加减法运算
    - 第一个步骤进行了 4 步
    - 第二个步骤进行了  $1 + 7$  步
    - 第三个步骤进行了  $1 + 3$  步
    - 第四个步骤进行了 1 步
    - 总共需要进行  $4 + 8 + 4 + 1 = 17$  次加减法运算
  - 而对于原来的 mod 10 算法
    - 每次减去 10，我们总共需要减去 366 次
    - 最后再把 10 加回来，需要进行 1 次
    - 总共需要进行  $366 + 1 = 367$  次加减法运算
  - 显然，从这个例子来看，仅考虑加减法运算次数的情况下，设置梯度后的算法比原来的算法快了 20 余倍
3. 综上所述，我们可以通过增大模运算时的 减数 或者设置 减数梯度 来提高循环的效率，加快运算速度。其中，后者对于效率的提升尤为显著
4. 改进后的程序见 汇编代码 小节的 version 2
5. 实际在 LC3 Simulator 网站上进行测试时，发现只考虑余数是否为 8 而不考虑是否为 8 的倍数同样可以用通过测试，如果确实没有这方面需求，可以把相应功能删除，程序的速度和效率会再快一倍