# Lec 04 Programming with Stack

## 寄存器保存与恢复

**问题**：假设寄存器R0,R5,R7中存储了重要值，在调用POP子程序后需要保留这些值，如何保存和恢复这些寄存器?

**两种保存方式**：

1. **Caller-Saver(调用者保存)**：
   - 主程序在调用子程序前保存寄存器的值，调用后恢复。
   - 代码示例：

```
.ORIG x3000
ST R0, Save_R0   ; 保存R0
ST R5, Save_R5   ; 保存R5
ST R7, Save_R7   ; 保存R7
JSR POP          ; 调用POP
LD R0, Save_R0   ; 恢复R0
LD R5, Save_R5   ; 恢复R5
LD R7, Save_R7   ; 恢复R7
```

2. **Callee-Saver(被调用者保存)**：
   - 子程序内部保存和恢复它使用的寄存器。
   - 代码实例（POP子程序内部）：

```
POP
    ST R3, POP_SaveR3  ; 保存R3
    ST R6, POP_SaveR6  ; 保存R6
    ; 栈操作逻辑...
    LD R3, POP_SaveR3  ; 恢复R3
    LD R6, POP_SaveR6  ; 恢复R6
    RET
```

## 基于栈的计算器

- **传统方式**：LC-3的ADD指令需要显式制定三个操作数（如`ADD R0,R1,R2`）
- **栈方式**：操作数隐式存储在栈中，操作直接从栈顶取值。
  - 示例：表达式`E=(A+B)*(C+D)`
    - LC-3实现：

```
LD R0, A
LD R1, B
ADD R0, R0, R1
LD R2, C
LD R3, D
ADD R2, R2, R3
MULT R0, R0, R2  ; 假设MULT存在
ST R0, E
```

- 栈实现：

```
PUSH A
PUSH B
ADD
PUSH C
PUSH D
ADD
MULT
POP E
```

- **利用栈方式计算乘法**：
  - 主程序：

```
.ORIG x3000
ADD R1, R0, #5     ; R1 = 5
ADD R2, R0, #7     ; R2 = 7
ST R0, MAIN_SaveR0 ; 保存R0
ADD R0, R1, #0
JSR PUSH           ; 入栈5
ADD R0, R2, #0
JSR PUSH           ; 入栈7
JSR MUL            ; 调用乘法
JSR POP            ; 弹出结果
LD R0, MAIN_SaveR0 ; 恢复R0
HALT
```

  - MUL子程序：

```
MUL
    ST R2, MULT_SaveR2
    ST R7, MULT_SaveR7
    JSR POP          ; 弹出第一个数到R0
    ADD R2, R0, #0   ; 保存到R2
    JSR POP          ; 弹出第二个数到R0
    ADD R1, R0, #0   ; 保存到R1
    AND R0, R0, #0   ; 清零R0
LOOP
```

```
    ADD R0, R0, R1   ; 重复加法
    ADD R2, R2, #-1
    BRp LOOP
    JSR PUSH         ; 结果入栈
    LD R2, MULT_SaveR2
    LD R7, MULT_SaveR7
    RET
```