南开大学

Nankai University

南　　开　　大　　学

计　算　机　学　院

语言处理系统的完整工作过程实验报告

语言处理系统探究

杨乔钦　　王泽舜

年级：2023 级

专业：计算机科学与技术

指导教师：王刚

2025 年 9 月 25 日

摘要

关键字：Parallel

目录

一、 原始 SysY 程序

我们编写了一个简单的 SysY 程序，功能是读取一个整数数组，计算其元素和，并根据和的大小进行不同的输出操作如果大于 20，会输出和的两倍，如果小于等于 20，会输出和除以 3 的余数。程序中包含了变量声明、数组操作、算术运算、条件判断和函数调用等基本语法结构。以下是该 SysY 程序的完整代码:

逐列访问平凡算法

```
// 全局变量和常量声明
const int N = 5;
int global_array[5] = {2,2,3,4,5};

// SysY运行时库函数声明 - 必须添加这些声明才能调用库函数
void putint(int);        // 输出一个整数
void putch(int);         // 输出一个字符

// 函数声明
int calculate(int a, int b);

int main() {

    int sum = 0;
    int i = 0;

    // 使用getarray从输入获取数据，替换原来的固定数组赋值
    int count = 5;


    // 算术运算：计算数组元素的和
    while (i < count) {  // 使用实际读取的元素个数
        sum = sum + global_array[i];
        i = i + 1;
    }

    // 条件判断和输出
    if (sum > 20) {
        int result = calculate(sum, 2);
    putint(result);  // 使用putint输出结果而不是直接return
    putch(10);       // 输出换行符
    } else {
        int remainder = sum % 3;
    putint(remainder); // 使用putint输出结果
    putch(10);         // 输出换行符
    }

    return 0;  // main函数返回0表示正常结束
}

```

```c
// 函数定义: 复杂计算
int calculate(int a, int b) {
    int result = 0;
    while (b > 0) {
        result = result + a;
        b = b - 1;
    }
    return result;
}
```

二、 arm 汇编程序

我编写的 arm 汇编程序如下所示，该程序实现了与上述 SysY 程序相同的功能。它读取一个整数数组，计算其元素和，并根据和的大小进行不同的输出操作。如果和大于 20，则输出和的两倍；如果和小于等于 20，则输出和除以 3 的余数。程序中包含了变量声明、数组操作、算术运算、条件判断和函数调用等基本语法结构。以下是该 arm 汇编程序的完整代码:

逐列访问平凡算法

```
        .arch armv8-a ; 指定目标架构为 AArch64
        .file   "test.c" ; 源文件名信息
        .text ; 文本段开始
        .section        .text.startup,"ax",@progbits ; 启动代码段
        .align  2 ; 对齐
        .p2align 4,,11 ; 更高对齐
        .global main ; 导出 main 符号
        .type   main, %function ; 声明 main 为函数
main: ; main 函数入口
.LFB0: ; 函数框架开始标签
        .cfi_startproc ; 调试信息开始
        adrp    x0, .LANCHOR0 ; 将 LAUNCHER 基址高位加载到 x0
        add     x1, x0, :lo12:.LANCHOR0 ; 将基址低位加到 x0 得到完整地址放 x1
        stp     x29, x30, [sp, -16]! ; 保存帧指针和返回地址并调整栈（push）
        .cfi_def_cfa_offset 16 ; 调试: 定义 CFA 偏移
        .cfi_offset 29, -16 ; 调试: x29 保存位置
        .cfi_offset 30, -8 ; 调试: x30 保存位置
        mov     x29, sp ; 设置帧指针 x29 = sp
        ldp     w0, w4, [x1] ; 从内存载入两个 32-bit 值到 w0,w4
        ldp     w3, w2, [x1, 8] ; 继续加载另外两个 32-bit 值到 w3,w2
        add     w0, w0, w4 ; 累加 w0 和 w4
        ldr     w1, [x1, 16] ; 载入第五个 32-bit 值到 w1
        add     w0, w0, w3 ; 累加 w3
        add     w0, w0, w2 ; 累加 w2
        add     w0, w0, w1 ; 累加 w1, w0 保存元素和
        cmp     w0, 20 ; 比较和与 20
        ble     .L2 ; 若小于等于 20 跳到 .L2
        lsl     w0, w0, 1 ; 否则将和乘 2（左移一位）
        bl      putint ; 调用 putint 输出整数
```

```
30          mov     w0, 10 ; 将换行符 ASCII 10 放入 w0
31          bl      putch ; 调用 putch 输出换行
32  .L3: ; 公共返回标签
33          mov     w0, 0 ; 返回值置 0
34          ldp     x29, x30, [sp], 16 ; 恢复 x29,x30 并释放栈空间
35          .cfi_remember_state ; 调试信息: 记住状态
36          .cfi_restore 30 ; 恢复调试寄存器 30 信息
37          .cfi_restore 29 ; 恢复调试寄存器 29 信息
38          .cfi_def_cfa_offset 0 ; 恢复 CFA 偏移
39          ret ; 返回调用者
40  .L2: ; 小于等于20的处理分支
41          .cfi_restore_state ; 恢复调试状态
42          mov     w1, 3 ; 将除数 3 放入 w1
43          sdiv    w1, w0, w1 ; w1 = w0 / 3 (商)
44          add     w1, w1, w1, lsl 1 ; w1 = w1 + (w1 << 1) = 3 * 商
45          sub     w0, w0, w1 ; 计算余数 w0 = w0 - 3*商
46          bl      putint ; 输出余数
47          mov     w0, 10 ; 设置换行字符
48          bl      putch ; 输出换行
49          b       .L3 ; 跳转到公共返回
50          .cfi_endproc ; 结束调试信息
51  .LFE0: ; 函数结束标签
52          .size   main, .-main ; 记录 main 大小
53          .text ; 文本段 (下一个函数)
54          .align  2 ; 对齐
55          .p2align 4,,11 ; 对齐
56          .global calculate ; 导出 calculate 符号
57          .type   calculate, %function ; 声明 calculate
58  calculate: ; calculate 函数入口
59  .LFB1: ; 函数框架开始
60          .cfi_startproc ; 调试开始
61          mul     w0, w1, w0 ; w0 = w0 * w1 (乘法实现)
62          cmp     w1, 0 ; 比较参数 w1 与 0
63          csel    w0, w0, wzr, gt ; 若 w1>0 返回乘积, 否则返回 0
64          ret ; 返回
65          .cfi_endproc ; 结束调试信息
66  .LFE1: ; 结束标签
67          .size   calculate, .-calculate ; 记录大小
68          .global global_array ; 导出全局数组
69          .global N ; 导出常量 N
70          .section        .rodata ; 只读数据段
71          .align  2 ; 对齐
72          .type   N, %object ; 声明 N 为对象
73          .size   N, 4 ; N 的大小
74  N: ; 常量 N 标签
75          .word   5 ; N = 5
76          .data ; 数据段开始
77          .align  4 ; 数据对齐
```

```
78          .set    .LANCHOR0,. + 0 ; 设置锚点符号
79          .type   global_array, %object ; 声明全局数组
80          .size   global_array, 20 ; 数组大小 20 字节
81  global_array: ; 全局数组标签
82          .word   2 ; 元素 0 = 2
83          .word   2 ; 元素 1 = 2
84          .word   3 ; 元素 2 = 3
85          .word   4 ; 元素 3 = 4
86          .word   5 ; 元素 4 = 5
87          .ident  "GCC: (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0" ; 编译器标识
88          .section        .note.GNU-stack,"",@progbits ; 指示不可执行栈
```

(一) 说明

- 调用约定：AArch64 使用寄存器 x0..x7 传递前八个整型/指针参数，返回值放在 x0（32-bit 时用 w0）。被调用者保存寄存器（如 x19..x28）需在函数入口保存并在返回前恢复。
- 栈帧与栈保护：函数入口处通过 `sub sp, sp, #64` 分配栈空间，编译器加入了栈保护（`_stack_chk_guard` / `_stack_chk_fail`）以检测栈溢出攻击。
- 数组寻址：编译器使用位移指令（例如 `lsl 2`）将索引乘以 4 来计算整数元素的字节偏移，从而高效访问数组元素。
- 循环与分支优化：简单的循环被编译为比较/分支形式；小的数学变换（例如将循环展开或用乘法替代循环）可能由编译器进行优化，如本例中 `calculate` 使用乘法和条件选择实现语义。
- 运行时库依赖：汇编中调用的 `putint`、`putch` 等函数由运行时库（例如 `libsysy_aarch.a`）提供。链接时必须指定对应目标架构的静态库，否则会出现未定义引用错误。
- 链接建议：使用交叉链接器将汇编/目标文件与 AArch64 静态库链接，例如：

  ```
  aarch64-linux-gnu-gcc test_aarch64.s libsysy_aarch.a -o test_arm.out
  ```

  若库放在子目录则传递相对路径：`lib/..../libsysy_aarch.a`。
- 架构匹配：务必确保静态库是为目标架构（AArch64）构建的；x86 构建的库无法用于 ARM 链接。
- 调试与反汇编：若需查看生成的可执行的真实指令或符号信息，可用 `objdump -d` / `readelf` / `nm` 等工具检查符号表与节信息。

(二) 结果展示

我们将上述 SysY 程序编译为 ARM 汇编，并链接运行时库，成功生成了可执行文件。以下是 SysY 程序和 arm 汇编的编译成功控制台信息：

运行成功信息

```
yqq@LAPTOP-B4JFEDK6:~/repository/Copiler/1.编译器了解$ aarch64-linux-gnu-gcc test_aarch64.s lib.tar.gz/libsysy_aarch.a -o test_arm.out
yqq@LAPTOP-B4JFEDK6:~/repository/Copiler/1.编译器了解$ qemu-aarch64 -L /usr/aarch64-linux-gnu ./test_arm.out
1
TOTAL: 0H-0M-0S-0us
```

oprule array 内容	SysY 编译输出	arm 汇编编译输出
1,2,3,4,5	0	0
2,2,3,4,5	1	1
3,2,3,4,5	2	2
4,2,3,4,5	0	0

表 1: 运行结果比较

由此可见，SysY 程序和 arm 汇编程序在相同输入下均能正确输出预期结果，功能实现一致。