計算機組織與 RISC-V CPU 設計 · 第 19 章

RISC-V 組合語言程式設計

組譯流程 / 偽指令 / 呼叫慣例與堆疊框架 / 完整範例

摘要

本章把第 18 章的指令「用起來」: 介紹 GNU 工具鏈的組譯與連結流程、常用偽指令與組譯器指示詞; 詳述 RISC-V 呼叫慣例 **(calling convention)**——參數與回傳值的暫存器分配、呼叫者/被呼叫者保存責任、堆疊框架的建立與拆除; 並以迴圈、陣列、字串、遞迴 (階乘與費氏) 等完整範例示範慣用寫法。這些程式將於第 26 章載入我們自製的 Verilog CPU 上實際執行。

目錄

1 工具鏈與組譯流程 **2**

1.1 常用指示詞與偽指令 2

2 呼叫慣例 (ilp32 ABI) **2**

3 基本範例 **2**

3.1 迴圈與陣列: 求和 2

3.2 字串長度 (位元組走訪) 3

4 遞迴與堆疊框架 **3**

5 與 C 的對照 **4**

6 本章重點回顧 **4**

7 練習 **4**

1 工具鏈與組譯流程

```
# 組譯 (RV32I, 小端序)
riscv64-unknown-elf-as   -march=rv32i -mabi=ilp32 prog.s -o prog.o
# 連結 (指定記憶體佈局)
riscv64-unknown-elf-ld   -m elf32lriscv -T link.ld prog.o -o prog.elf
# 反組譯檢查
riscv64-unknown-elf-objdump -d prog.elf
# 轉出純二進位 / 十六進位檔 (給 Verilog $readmemh 用)
riscv64-unknown-elf-objcopy -O binary prog.elf prog.bin
```

程式 1 GNU RISC-V 工具鏈典型流程

沒有安裝工具鏈也可以使用線上模擬器 (如 RARS、Venus) 練習; 第 26 章我們則以「手工組譯 +`$readmemh`」的方式餵程式給自製 CPU。

1.1 常用指示詞與偽指令

指示詞	`.text`(程式段)、`.data`(資料段)、`.word/.byte/.asciz`(配置資料)、`.globl`(輸出符號)、`.align`
偽指令	`li rd,imm32`(lui+addi)、`la rd,symbol`(auipc+addi)、`mv`、`not`、`neg`、`seqz/ snez`、`beqz/ bnez`、`j`、`jr`、`call`、`ret`、`nop`

2 呼叫慣例 (ilp32 ABI)

- 參數:a0–a7(x10–x17) 依序傳遞; 更多參數走堆疊。回傳值:a0(a1 輔助 64 位元)。
- 呼叫者保存 **(caller-saved)**:t0–t6、a0–a7、ra——被呼叫的函式可以隨意破壞; 呼叫者若還需要, 呼叫前自行保存。
- 被呼叫者保存 **(callee-saved)**:s0–s11、sp——函式若要使用, 必須先存後還。
- **sp** 永遠 16 位元組對齊; 堆疊向下成長。

堆疊框架的標準八步

非葉函式的樣板: (1) 開場:`addi sp,sp,-N` 挪出框架;(2) 保存 ra;(3) 保存要用的 s 暫存器;(4) 本體;(5)–(7) 逆序恢復;(8) `ret`。葉函式若只用 t/a 暫存器, 完全不必碰堆疊——這是 ra 走暫存器的紅利 (第 10 章)。

3 基本範例

3.1 迴圈與陣列: 求和

```
sum_array:
    li   t0, 0            # t0 = 累加器
    li   t1, 0            # t1 = i
loop:
    bge  t1, a1, done     # i >= n 則離開
    slli t2, t1, 2        # t2 = i*4
```

```
    add  t2, t2, a0       # t2 = &A[i]
    lw   t3, 0(t2)        # t3 = A[i]
    add  t0, t0, t3       # sum += A[i]
    addi t1, t1, 1        # i++
    j    loop
done:
    mv   a0, t0           # 回傳值
    ret
```

程式 2 ,a0= 陣列基址、a1= 長度, 回傳 a0]sum = $\sum$ A[i],a0= 陣列基址、a1= 長度, 回傳 a0

慣用最佳化: 以「移動指標」取代「索引計算」, 內圈少兩條指令:

```
    slli t1, a1, 2
    add  t1, t1, a0       # t1 = 尾後位址
    li   t0, 0
loop:
    bgeu a0, t1, done
    lw   t2, 0(a0)
    add  t0, t0, t2
    addi a0, a0, 4        # 指標前進
    j    loop
```

3.2 字串長度 (位元組走訪)

```
strlen:
    mv   t0, a0
1:  lbu  t1, 0(t0)        # 讀一個位元組 (零延伸)
    beqz t1, 2f           # 遇 '\0' 結束
    addi t0, t0, 1
    j    1b
2:  sub  a0, t0, a0       # 長度 = 尾 − 頭
    ret
```

程式 3 strlen:a0= 字串位址, 回傳 a0

4 遞迴與堆疊框架

```
fact:
    li   t0, 2
    blt  a0, t0, base     # n < 2 -> 回傳 1
    addi sp, sp, -16      # 開框架 (16B 對齊)
    sw   ra, 12(sp)       # 保存返回位址
    sw   s0, 8(sp)        # 保存 s0
    mv   s0, a0           # s0 = n (跨呼叫存活)
    addi a0, a0, -1
    call fact             # a0 = fact(n-1)
    mul  a0, s0, a0       # n * fact(n-1)  (RV32M)
    lw   s0, 8(sp)        # 恢復
    lw   ra, 12(sp)
    addi sp, sp, 16       # 拆框架
    ret
base:
    li   a0, 1
    ret
```

程式 4 遞迴階乘:fact(n),a0=n

框架剖析

為什麼 n 要搬進 s0? 因為 `call fact` 可能破壞所有 t/a 暫存器 (呼叫者保存); 跨越呼叫仍需存活的值必須放 s 暫存器, 而使用 s 暫存器就必須先保存到堆疊。ra 同理: 內層 call 會覆寫 ra, 不存就回不了家。純 RV32I(無 M 擴展) 時, 乘法改為呼叫軟體乘法函式, 結構完全相同。

5 與 C 的對照

```c
int max(int a, int b) { return a > b ? a : b; }
```

程式 5 C 原始碼

```asm
max:
    bge  a1, a0, 1f      # b >= a 則回傳 b... 注意編譯器反轉了條件
    ret                  # 回傳 a (已在 a0)
1:  mv   a0, a1
    ret
```

程式 6 gcc -O2 -march=rv32i 產出 (示意)

閱讀編譯器輸出 (`gcc -S` 或 objdump) 是學習慣用法的最快途徑: 你會看到本章所有慣例被嚴格遵守——這正是二進位相容的基礎。

6 本章重點回顧

本章要點

- 工具鏈:as → ld → objcopy; 偽指令讓組語貼近人類 (li/la/call/ret...)。
- 呼叫慣例:a0–a7 傳參、a0 回傳;t/a/ra 呼叫者保存、s/sp 被呼叫者保存。
- 堆疊框架八步樣板; 葉函式可零堆疊開銷。
- 跨呼叫存活的值放 s 暫存器; 用 s 就要先存後還。

7 練習

1. 寫出 `memcpy(dst,src,n)`(位元組版), 再改寫為字 (4B) 對齊快速版。
2. 寫出遞迴費氏 fib(n), 畫出 fib(3) 執行時堆疊框架的變化。
3. 寫一個函式呼叫 strlen 兩次比較兩字串長短: 哪些暫存器必須保存? 為什麼?
4. 把 sum_array 改為每輪處理 4 個元素的迴圈展開版, 數一數指令數差異。
5. 用 `lbu/sb` 實作 `toupper` 字串就地轉大寫。

參考資料

[1] RISC-V International, *RISC-V ELF psABI Specification*. https://github.com/riscv-non-isa/riscv-elf-psabi-doc

[2] D. Patterson and A. Waterman, *The RISC-V Reader*, 2017, Ch. 3.

[3] RARS – RISC-V Assembler and Runtime Simulator. https://github.com/TheThirdOne/rars