HW1 REPORT

王柏舜, 0716089

***Abstract*—修改 libc 中的 strcpy() 以及 strcmp() 來達到提升 Dhrymips 的目的，報告中會提到如何修改 strcpy() 以及 strcmp() 以及在模擬的過程中以及使用邏輯分析儀的時候有什麼發現**

Keywords—component; formatting; style; styling; insert (key words)

I. 介紹

這份作業讓我們學習如何利用 uart 輸入我們需要跑的程式碼以及對於 vivado 的ＩＬＡ的基本操作。對於 Dhrystone Benchmarks 在執行的時候會需要呼叫次的 strcpy() 以及 strcmp() 但是原始的程式碼並沒有經過優化這是作業的目標就是優化這兩個函式以提升 performance 可以使用的工具有 vivado 內建的 ILA

II. OPTIMIZE STRCPY()

A. 觀察

首先我先對於原本的 strcpy() 觀察其 objdump 所產生 strcpy()的部分除了產生的指令非常多以外指令間也存

```
00001ec0 <strncpy>:
    1ec0:   0005c703            lbu a4,0(a1)
    1ec4:   04070463            beqz    a4,1f0c <strncpy+0x4c>
    1ec8:   02060e63            beqz    a2,1f04 <strncpy+0x44>
    1ecc:   00050793            mv  a5,a0
    1ed0:   0080006f            j   1ed8 <strncpy+0x18>
    1ed4:   02060a63            beqz    a2,1f08 <strncpy+0x48>
    1ed8:   00178793            addi    a5,a5,1
    1edc:   fee78fa3            sb  a4,-1(a5)
    1ee0:   00158593            addi    a1,a1,1
    1ee4:   0005c703            lbu a4,0(a1)
    1ee8:   fff60613            addi    a2,a2,-1
    1eec:   fe0714e3            bnez    a4,1ed4 <strncpy+0x14>
    1ef0:   00c78733            add a4,a5,a2
    1ef4:   02060063            beqz    a2,1f14 <strncpy+0x54>
    1ef8:   00178793            addi    a5,a5,1
    1efc:   fe078fa3            sb  zero,-1(a5)
    1f00:   fee79ce3            bne a5,a4,1ef8 <strncpy+0x38>
    1f04:   00008067            ret
    1f08:   00008067            ret
    1f0c:   00050793            mv  a5,a0
    1f10:   fe1ff06f            j   1ef0 <strncpy+0x30>
    1f14:   00008067            ret
```

在著非常多的 data hazard ，像是 a4 、a5 這兩個 register 常常在執行 load byte 和 store byte 的指令接著執行 beqz 或 bne 等指令但因為 load 和 store 的關西使得至少需要經過兩的 stall beqz 和 bne 才能正確地執行這使得 performance 變得很低。

B. 修改策略以及如何利用 ILA

修改過程中我總共改了三個版本的 strcpy()出來，這三個版本是依序繼成前一個版本繼續優化，最後使用效率最好的版本三，接下來三張圖分別為版本一至三

```
// version 1
char *save = dst;
for(;(*dst=*src)!='\0';++src,++dst);
return save;
```

```
// version 2
char *tmp = dst;
--src;
--dst;
do{
    src++;
    dst++;
    if((*dst=*src)=='\0') break;

} while(1);
return tmp;
```

```
// vsrsion 3
char *d = dst;
char c;
do{
    c = *src;
    asm volatile("addi a0,a0,1");
    asm volatile("addi a1,a1,1");
    //must a nop produced because the value load in lbu can be used after three instructions
    asm volatile("SB a5,-1(a0)");
    // c=*src++;
    // *d++=c;

}while(c-'\0');
return d;
```

我在版本一和二透過調整 operation 在 loop 中的執行順序先降低部分的 data hazard 但在版本二中產生的 objdump file 我觀察到還是有某些 lub ，sb 指令沒辦法相距至少兩的 clk cycle 來避免 data hazard ，因此我決定以版本二為基礎在不改變邏輯順序的前提下自行排序 assembly code 已達成降低 data hazard 的產生，經過調整 assembly code 後所產生的 objdump file 如下

```
00001e98 <strcpy>:
    1e98:   0005c783            lbu a5,0(a1)
    1e9c:   00150513            addi    a0,a0,1
    1ea0:   00158593            addi    a1,a1,1
    1ea4:   fef50fa3            sb  a5,-1(a0)
    1ea8:   fe0798e3            bnez    a5,1e98 <strcpy>
    1eac:   00008067            ret
```

經過這樣的修改後 DMIPS/Mhz 上升了 0.2 。對於 ILA 的

使用與觀察我抓了 top/riscv_core0/fetch 下的 pc_o 來觀察他是你 fetch 到指令的 program counter 我透過它來看哪個指令跑比較慢或是哪幾個指令間有 hazard 的發生。

III. OPTIMIZE STRCMP()

A. 觀察

```
00001fa0 <strcmp>:
    1fa0:   fff50513            addi    a0,a0,-1
    1fa4:   fff58593            addi    a1,a1,-1
    1fa8:   0080006f            j   1fb0 <strcmp+0x10>
    1fac:   02078663            beqz    a5,1fd8 <strcmp+0x38>
    1fb0:   00150513            addi    a0,a0,1
    1fb4:   00158593            addi    a1,a1,1
    1fb8:   00054783            lbu a5,0(a0)
    1fbc:   0005c703            lbu a4,0(a1)
    1fc0:   fee786e3            beq a5,a4,1fac <strcmp+0xc>
    1fc4:   fff00513            li  a0,-1
    1fc8:   00e7f463            bleu    a4,a5,1fd0 <strcmp+0x30>
    1fcc:   00008067            ret
    1fd0:   00100513            li  a0,1
    1fd4:   00008067            ret
    1fd8:   00000513            li  a0,0
    1fdc:   00008067            ret
```

首先對於原始的 strcmp() objdump file 進行觀察，會發現在 lbu 和 beq 間會有 data hazard 的產生這會讓 pipeline 有 stall 的產生因而造成 performance 的下降。

B. 修改策略以及如何使用 ILA

對於 strcmp() 我寫出了兩個版本第二個版本是基於第一個版本來稍作修改，修改的想法和 strcpy()類似，一樣是透過條整迴圈類 operation 的順序並且觀察產生的 objdump file 和 ILA 的結果來將 register a5 的 data hazard 降低以下為版本一和版本二的原始碼

```
//version 1
s1--;
s2--;
do{
    s1++;
    s2++;
    if(*s1 == '\0')
        return 0;

}while(*s1==*s2)
if(*s1-*s2<0) return -1;
else return 1;
```

```
// version 2
while(*s1){
    if(*s1 != *s2){
        break;
    }
    s1++;
    s2++;
}
int a = *s1-*s2;
if(a < 0) return -1;
else  if (a>0) return 1;
else return 0;
```

在寫出版本一的時候我發現 register a5 在 lbu 和 bnez 這兩個指令之間還是相距不夠遠雖然 performance 已經比原始碼好了但是還是無法更有效的加速真正使得 perfromance 變慢的地方，所以我基於版本一在稍作修改得到版本二的結果版本二的 objdump file 如下

```
00001fa0 <strcmp>:
    1fa0:   00054783            lbu a5,0(a0)
    1fa4:   0005c703            lbu a4,0(a1)
    1fa8:   02078863            beqz    a5,1fd8 <strcmp+0x38>
    1fac:   00e79c63            bne a5,a4,1fc4 <strcmp+0x24>
    1fb0:   00150513            addi    a0,a0,1
    1fb4:   00054783            lbu a5,0(a0)
    1fb8:   00158593            addi    a1,a1,1
    1fbc:   0005c703            lbu a4,0(a1)
    1fc0:   fe0796e3            bnez    a5,1fac <strcmp+0xc>
    1fc4:   40e787b3            sub a5,a5,a4
    1fc8:   fff00513            li  a0,-1
    1fcc:   0007c463            bltz    a5,1fd4 <strcmp+0x34>
    1fd0:   00f03533            snez    a0,a5
    1fd4:   00008067            ret
    1fd8:   00000793            li  a5,0
    1fdc:   fe9ff06f            j   1fc4 <strcmp+0x24>
```

可以發現 register a5 在 lbu 和 bnez 相距足夠遠可以避免在 pipeline 中產生 stall 得到最好的 perfromance 最終提升的 DMIPS/Mhz 為 0.3。對於 ILA 的使用我一樣是以 top/riscv_core0/fetch 下的 pc_o 來進行觀察，觀察哪個指令跑比較多 clock cycle。

IV. 最終結果

```
Microseconds for one run through Dhrystone: 14.820006
Dhrystones per Second:                      67476.4
VAX MIPS:                                   38.4
DMIPS/Mhz:                                   0.77
```

最終提升的 DMIPS/Mhz 為 0.05 ，DMIPS/Mhz 由 0.72 提升到 0.7。