Lab3 ARM Assembly II

0516039 羅文笙

1. Lab objectives 實驗目的

Familiar with the ARMv7 assembly language programming.
熟悉基本 ARMv7 組合語言語法使用。

2. Steps 實驗步驟

(1) Postfix arithmetic

(1)-1 將在 text 內的　postfix_expr 變數轉成一個一個的由 space 分出的 token，存到記憶體的某個連續的位置

```
37 atoi:
38     //TODO: implement a "convert string to integer" function
39
40     ldrb r1,[r0]//load lowest 8 bit(1 byte)of values in r0 into r1
41     add r0,r0,#1//to get next char
42     ldrb r2,[r0]//load next char
43     sub r0,r0,#1//get back to right now string pointer
44
45     //check_minus_or_negative
46     cmp r1, #45
47     beq check_minus_or_negative
48
49     //IT block for  pushing "+" into stack
50     cmp r1, #43
51     ITTT eq
52     pusheq {r1}
53     addeq r0,r0,#1
54     beq atoi
55
56     //branch to push_int_space to push integer into stack
57     cmp r1, #32
58     beq push_int_space
59
60     //branch to push_int_end to push integer into stack & end for loop
61     cmp r1, #0
62     beq push_int_end
63
64     //
65     cmp r1,#47 //'0'~'9' = 48~57
66     ITTTT gt
67     subgt r1,r1,#48 //make char num to int num
68     mulgt r5,r5,r3
69     addgt r5,r5,r1
70     movgt r7,#1
71
72     //for loop doing i++
73     add r0,r0,#1 //go to next char's address
74     b atoi //if r1 != '\0',then keep doing atoi
75
76
```

```
106 check_minus_or_negative:
107     cmp r2, #32
108     ITE ne
109     movne r4,#1
110     pusheq {r1}
111
112     cmp r2, #0
113     IT eq
114     pusheq {r1}
115
116     add r0,r0,#1
117     b atoi
```

```
 77 push_int_space:
 78     cmp r4, #1 //test if temp is a negative number
 79     IT eq
 80     subeq r5,r6,r5
 81
 82     cmp r7, #1 //test if temp is empty or not
 83     ITTTT eq
 84     pusheq {r5} //push temp into stack
 85     moveq r4, #0 //negative = false
 86     moveq r5, #0 //temp = 0
 87     moveq r7, #0 //empty = 0
 88
 89     add r0,r0,#1
 90     b atoi
 91 push_int_end:
 92     cmp r4, #1 //test if temp is a negative number
 93     IT eq
 94     subeq r5,r6,r5
 95
 96     cmp r7, #1 //test if temp is empty or not
 97     ITTTT eq
 98     pusheq {r5} //push temp into stack
 99     moveq r4, #0 //negative = false
100     moveq r5, #0 //temp = 0
101     moveq r7, #0 //empty = 0
102
103     //add r0,r0,#1
104     push {r6}
105     bx lr
```

(1)-2 照順序讀取剛剛的連續記憶體位置，如果遇到數字就 push 到 stack 裡面，如果遇到加減符號就從 stack pop 兩個數字出來做相對應的運算，再將答案 push 進去 stack。

```
118 calculate_expr_result:
119     //use r4, r5,r6 to do stackt calculation,
120     //r4~r5 is used to store pop value, r6 is the answer
121     sub r1,r1,#4
122     ldr r2,[r1] //load the value in r1 to r2
123
124     cmp r2,#43 //if it is a '+', do addition
125     ITTTT eq
126     popeq {r4,r5}
127     addeq r6, r4,r5
128     pusheq {r6}
129     beq calculate_expr_result
130
131     cmp r2,#45 ////if it is a '-', do subtraction
132     ITTTT eq
133     popeq {r4,r5}
134     subeq r6, r5,r4
135     pusheq {r6}
136     beq calculate_expr_result
137
138     cmp r2,#0 //if it  is '\0', end the function
139     ITTTT eq
140     popeq {r6}
141     ldreq r0,=expr_result
142     streq r6,[r0]
143     beq program_end
144
145     push {r2} //if it is just a number, push it onto stack
146     b calculate_expr_result //loop
```

(1)-3 最後讀到結尾的話就將 stack 內唯一的個數字 pop 出來到 register，再利用這個 register 將答案存進變數

```
cmp r2,#0 //if it  is '\0', end the function
ITTTT eq
popeq {r6}
ldreq r0,=expr_result
streq r6,[r0]
beq program_end
```

(2) GCD & Max Stack Size

(2)-1 利用 Stein's algorithm 將 gcd 由 c program 轉成 arm

```
26 GCD:
27     //TODO: Implement your GCD function
28     pop {r0,r1} //get a,b
29     mov r4,#0 //initilize r4, which is an odd-even indicator
30     //if(a<b) exchange a b
31     cmp r0,r1
32     ittt lt
33     movlt r2,r0
34     movlt r0,r1
35     movlt r1,r2
36
37     //test a,b are odd or even numbers
38     //test a,if r2 = 1, means a is an odd number,
39     //otherwise it is an even number
40     mov r2,r0
41     lsl r2,#31
42     lsr r2,#31
43     //test b,if r3 = 1, means b is an odd number,
44     //otherwise it is an even number
45     mov r3,r1
46     lsl r3,#31
47     lsr r3,#31
48
49     //if a is odd, add 10 to r4. if b is odd, add 1 to r4
50     cmp r2, #1
51     it eq
52     addeq r4, #10
53
54     cmp r3, #1
55     it eq
56     addeq r4, #1
57
58     //if((!(a&1))&&(!(b&1))) return stein(a>>1,b>>1)<<1;
59     cmp r4, #0
60     ITTT eq
61     lsreq r0,#1
62     lsreq r1,#1
63     addeq r5,r5,#1
```

```
64
65
66      //if((a&1)&&(!(b&1)))return stein(a,b>>1);
67      cmp r4, #10
68      it eq
69      lsreq r1,#1
70
71      //if((!(a&1))&&(b&1))return stein(a>>1,b);
72      cmp r4, #1
73      it eq
74      lsreq r0,#1
75
76      //else return stein(a-b,b);
77      cmp r4, #11
78      it eq
79      subeq r0,r0,r1
80
81      //if(b==0)return a;
82      cmp r1,#0
83      ittt ne
84      pushne {r0,r1,lr}
85      addne r7,r7,#1
86      blne GCD
87
88
89      pop {pc}
90      BX LR
```

(2)-2 在做遞迴的時候，紀錄 max stack size

```
105 get_max_size:
106     add r7,r7,#3 //幾層的recursive 加上a,b和第一層的link register
107     mul r7,r7,r9 //將幾個stack 算成多少bit
108     ldr r8,=max_size//將答案存進max_size
109     str r7,[r8]//將答案存進max_size
110     b program_end //branch to program end
```

3. Results and analysis 實驗結果與分析

(1) Postfix arithmetic

分析:在 demo 的時候發現原來 postfix_expr 因為跟程式碼一起存在.text 裡面，所以假如 postfix_expr 是一個奇數長度(含結尾'\0')的字串的話，會讓編譯錯誤，出現 unalignment 的情況，此時就要在 main 的第一行加上 .align 5 這行，就可使資料對齊，並通過編譯了。

(2) GCD & Max Stack Size

分析:GCD 的遞迴式要小心操作，並且在要進入下一層之前先將 link register 的值 push 到 stack 裡存起來，不然會回不去。 另外，max stack size 就是有幾層的 link register 加上兩個變數的值

4. Conclusions and ideas 心得討論與應用聯想

因為上個作業是我的同伴做的，我沒有參與很多，所以這個作業算是一個新的挑戰，要將自己寫出來的 c code 改來改去進而變成組語的形式，才發現很多原來自己在高階語言習以為常的程式設計方式，在組合語言的層面來說，是不合適的，一定要用另外一種想法才能寫出來。寫完這作業和 demo 完後，很慶幸自己能有這門課可以將之前所有學到的東西都連結在一起用，好像越來越期待接下來的課程了。