Tags: #cec32x, #hw, cec32x_HW_38–HW9_flow_control, 130 points

HW 9. Flow Control

We have learned in the class how to use conditional branching to implement various flow-control constructs in C. We have also learned how to use conditional execution to avoid the branches to improve the performance (pipeline can be better maintained). Here, we practice these techniques by converting the C code to assembly code. The C code for this problem is given below:

```
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>

extern void func_if_then_impl_1(void);
extern void func_if_then_impl_2(void);
extern void func_if_then_or_impl_1(void);
extern void func_if_then_or_impl_2(void);
extern void func_if_then_else_impl_1(void);
extern void func_if_then_else_impl_2(void);
extern void func_for_loop(void);
extern void func_while_loop(void);

#define INITIALIZE_vals \
                a = -4;\
                x = 1;

int32_t a, x, total_sum;
int32_t a1 = -4;
int32_t x1 = 1;

int main(void) {
// Simple if-then statement:
    INITIALIZE_vals;
    if (a < 0) {
        a = 0 - a;
    }
    x += 1;

    func_if_then_impl_1();
    func_if_then_impl_2();

// Simple if-then statement with compound logic OR expression:
    INITIALIZE_vals;
    if (x <= 20 || x >= 25) {
        a = 1;
    }

    func_if_then_or_impl_1();
```

```
    func_if_then_or_impl_2();


// Simple if-then-else statement:
    INITIALIZE_vals;
    if (a == 1) {
        x = 3;
    } else {
        x = 4;
    }

    func_if_then_else_impl_1();
    func_if_then_else_impl_2();

// The for loop---a simple example
    total_sum = 0;
    for (int i = 0; i < 10; i++) {
        total_sum += i;
    }

    func_for_loop();

// The while loop---a simple example
    total_sum = 0;
    int i = 15;
    while (i > 0) {
        total_sum += i;
        i--;
    }

    func_while_loop();

    while (1);
}
```

We need to implement the three `if` statements using two versions. Version 1 is based on conditional branch, and Version 2 is based on conditional execution. Each of the `if` statement implementations has 15 points, leading to a total of 90 points.

For the `for` loop and `while` loop, we only need one implementation based on conditional branch. Each has 20 points, leading to a total of 40 points.

Note that before running each C or assembly block code, we have an initialization code to reset the values of the variables. For cleanliness, we used Macros to do that.

Note that the assembly functions are the translation of the C code with hardcoded variables. They are not the conventional functions that take arguments. We just illustrate the principles here.

```
        AREA my_fancy_asm_code, CODE, READONLY   ; Define the program area

        ; Export functions defined in this file.
        ; in the file calling them.
        EXPORT func_if_then_impl_1
        EXPORT func_if_then_impl_2
        EXPORT func_if_then_or_impl_1
        EXPORT func_if_then_or_impl_2
        EXPORT func_if_then_else_impl_1
        EXPORT func_if_then_else_impl_2
        EXPORT func_for_loop
        EXPORT func_while_loop

        ALIGN                   ; Align the data in the boundary of 4 bytes.

myMC    PROC
        MACRO                   ; start macro definition
        initialize_r1_r2
        MOV  r1, #-4            ; variable a in the C code
        MOV  r2, #1             ; variable x in the assembly code
        MEND                    ; end macro definition
        ENDP


; Simple if-then statements:
; a. Implementation 1:
func_if_then_impl_1 PROC
        initialize_r1_r2
; Put your code here. Note that you have to use BX  lr to return to the caller.




        ENDP

; b. Implementation 2:
func_if_then_impl_2 PROC
        initialize_r1_r2
; Put your code here. Note that you have to use BX  lr to return to the caller.




        ENDP

; Simple if-then statement with compound logic OR expression:
```

```
; a. Implementation 1:
func_if_then_or_impl_1 PROC
        initialize_r1_r2
; Put your code here. Note that you have to use BX  lr to return to the caller.




        ENDP

; b. Implementation 2:
func_if_then_or_impl_2 PROC
        initialize_r1_r2
; Put your code here. Note that you have to use BX  lr to return to the caller.




        ENDP

; Simple if-then-else statement:
; a. Implementation 1:
func_if_then_else_impl_1 PROC
        initialize_r1_r2
; Put your code here. Note that you have to use BX  lr to return to the caller.




        ENDP

; b. Implementation 2:
func_if_then_else_impl_2 PROC
        initialize_r1_r2
; Put your code here. Note that you have to use BX  lr to return to the caller.




        ENDP

; The for loop---a simple example
func_for_loop PROC
; Put your code here. Note that you have to use BX  lr to return to the caller.
```

```
        ENDP

; The while loop---a simple example
func_while_loop PROC
; Put your code here. Note that you have to use BX  lr to return to the caller.




        ENDP

        END                             ; End of the entire file
```