EE3220 – System-on-Chip Design – 2021/22 Spring

Assignment 2 – Due date: March 15, 2022 (Tue) – 23:59pm

Submission method – Canvas online assignment collection box

Name: ______________________ **Date:** ____________
Student ID: _________________ **Mark:** ____________

Submission guidelines:

- Please prepare your assignment in "PDF" format.
- Please rename your file to "EE3220 – assignment 2 – studentID.pdf".
- For late submission, 10% deduction per day.

Q1: Consider the assembly code, which the compiler generates for a C function. Explain what each assembly instruction does and describe what data is in the register.

C Program:

```c
void fn(int *a, long *b, float *c){
    volatile int a1, a2;
    volatile long b1, b2;
    volatile float c1, c2;

    a1 = 15;
    a2 = -14;
    *a = a1*a2;

    b1 = 15;
    b2 = -14;
    *b = b1*b2;

    c1 = 15;
    c2 = -14;
    *c = c1*c2;
}

int main(void){
    int *a;
    long *b;
    float *c;

    fn(a, b, c);
    return 0;
}
```

Assembly Program:

```
fn(int*, long*, float*):
```

```
        push    {r11, lr}
        mov     r11, sp
        sub     sp, sp, #40
        str     r0, [r11, #-4]
        str     r1, [r11, #-8]
        str     r2, [r11, #-12]
        mov     r0, #15
        str     r0, [r11, #-16]
        mvn     r1, #13
        str     r1, [sp, #20]
        ldr     r2, [r11, #-16]
        ldr     r3, [sp, #20]
        mul     r12, r2, r3
        ldr     r2, [r11, #-4]
        str     r12, [r2]
        str     r0, [sp, #16]
        str     r1, [sp, #12]
        ldr     r0, [sp, #16]
        ldr     r1, [sp, #12]
        mul     r2, r0, r1
        ldr     r0, [r11, #-8]
        str     r2, [r0]
        mov     r0, #24117248
        orr     r0, r0, #1073741824
        str     r0, [sp, #8]
        mov     r0, #23068672
        orr     r0, r0, #-1073741824
        str     r0, [sp, #4]
        ldr     r0, [sp, #8]
        ldr     r1, [sp, #4]
        bl      __aeabi_fmul
        ldr     r1, [r11, #-12]
        str     r0, [r1]
        mov     sp, r11
        pop     {r11, lr}
        bx      lr
main:
        push    {r11, lr}
        mov     r11, sp
        sub     sp, sp, #24
        mov     r0, #0
        str     r0, [r11, #-4]
        ldr     r1, [r11, #-8]
        ldr     r2, [sp, #12]
        ldr     r3, [sp, #8]
        str     r0, [sp, #4]
        mov     r0, r1
        mov     r1, r2
        mov     r2, r3
        bl      fn(int*, long*, float*)
        ldr     r0, [sp, #4]
        mov     sp, r11
        pop     {r11, lr}
        bx      lr
```

You can use an online compiler (https://godbolt.org/) to complete the following questions. (Choose **"armv7-a clang 11.0.1"** or unless specified in the question, and set optimization level using flags **-O0**, **-O1**, **-O2** in the compiler options field).

Q2. Compile the following C code snippet with optimization level **O0**, **O1 and O2**.

```
float foo1(){
   float sum = 0.0;
   unsigned int i = 0;
   for (i=1; i<=15; i++) {
       sum += i*i*i;
   }
   return sum;
}
```

Which instruction(s) computes the return value of the function in **O2**? Compare and explain the difference between the three optimization levels.

Q3. Compile the following C code snippet with different processor architecture.

```
int square(int num) {
   return num * num;
}
```

1) compile using armv7-a clang 11.0, and explain the program.
2) compile using x86-64 gcc 11.2, and explain the program.

Q4. Compile the following C code snippet with optimization level **O1**

```
int foo2(int n){
   if (n == 1){
       return 1;
   } else {
       return n + foo2(n-1);
   }
}
```

- Which register(s) are saved before entering the recursive call? Explain why. (Hint: the registers have their special purposes).
- Next modify line 2 from (n == 1) to (n == 0), Explain what has been changed?

Q5. Compile the following C code snippet.

```c
struct test_t {
   short int x;
   short int y;
};

int foo3(int *list, struct test_t t){
   int ret;
   ret = t.x;
   unsigned int i;
   for (int i=0; i<20; i++){
       ret = ret + list[i] + t.y;
   }
   return ret;
}
```

a) How many registers are used for the function arguments?

b) If t.x = 0x1234 and t.y = 0x5678, how is the argument t stored in the parameter register(s)? (Hint: Compile with O1 and examine how t is accessed)

c) If (i<20) is now updated to (i<5) in the for-loop, explain which assembly instruction is updated and why?

d) In O2, how many loop iterations do the assembly code perform for 1) (i<20), and 2) (i<100), and why?

Q6. Define what is the reset handler in the ARM processor, and give an example of the code segment in this question.

Q7. Combine all functions above with the following main function, then compile it with optimization level **00**.

```
int main(void){
    volatile int s1;
    volatile int s2;
    volatile int s3;
    struct test_t t;
    int list[10] = {32, 43, 54, 65, 91, 76, 32, 29, 13, 78};

    s1 = foo1();
    s2 = foo2(56);
    t.x = square(s1 + s2);
    t.y = square(s1 - s2);
    s3 = foo3(list,t);
}
```

The local variables are stored in the memory and they are accessed by an offset to the stack pointer **sp**. Using the **sp** after instruction "**sub sp, sp, #72**" as the reference point, fill in missing content in the following table.

Hint: Checks all **str** and **ldr**

Local Variables	Offset from Stack Pointer (SP)
s1	
s2	
list	
t.y	
t.y	

The Integer array with 10 elements, describes how these data are represented in the ARM assembly program.

Q8. Consider the following assembly code. Write an equivalent C program.

```
unknown(int):
        push    {r11, lr}
        mov     r11, sp
        sub     sp, sp, #16
        str     r0, [sp, #8]
        ldr     r0, [sp, #8]
        cmp     r0, #1
        blt     .LBB0_2
        b       .LBB0_1
.LBB0_1:
        ldr     r0, [sp, #8]
        sub     r1, r0, #1
        str     r0, [sp, #4]
        mov     r0, r1
        bl      unknown(int)
        ldr     r2, [sp, #4]
        mul     r1, r2, r0
        str     r1, [r11, #-4]
        b       .LBB0_3
.LBB0_2:
        mov     r0, #1
        str     r0, [r11, #-4]
        b       .LBB0_3
.LBB0_3:
        ldr     r0, [r11, #-4]
        mov     sp, r11
        pop     {r11, lr}
        bx      lr
```

Hint

- bl: Saves PC+4 in link register and jumps to a function

Q9. In the tutorial, we learn how to compile a C-program into an ARM assembly program, and also learn how to disassemble a binary into assembly code.

```
ee3220@ee3220-virtual-machine:~$ arm-linux-gnueabihf-objdump -d helloworld32dyh

helloworld32dyh:     file format elf32-littlearm


Disassembly of section .init:

000003a0 <_init>:
 3a0:   e92d4008        push    {r3, lr}
 3a4:   eb000026        bl      444 <call_weak_fn>
 3a8:   e8bd8008        pop     {r3, pc}

Disassembly of section .plt:

000003ac <.plt>:
 3ac:   e52de004        push    {lr}            ; (str lr, [sp, #-4]!)
 3b0:   e59fe004        ldr     lr, [pc, #4]    ; 3bc <.plt+0x10>
 3b4:   e08fe00e        add     lr, pc, lr
 3b8:   e5bef008        ldr     pc, [lr, #8]!
 3bc:   00010c08        .word   0x00010c08

000003c0 <__cxa_finalize@plt>:
 3c0:   e28fc600        add     ip, pc, #0, 12
 3c4:   e28cca10        add     ip, ip, #16, 20 ; 0x10000
 3c8:   e5bcfc08        ldr     pc, [ip, #3080]!        ; 0xc08
```

- Describe the procedure to complete this compilation.
- Describe the meaning of the above terms: elf32-littlearm, .init, objdump, and push {lr}.

10. Consider the following C code, write and explain an equivalent assembly program with the highest optimization..

```c
int main(){
    int i=0, j=0;
    int array[2][2]={{1,2},{3,4}};

    for(i=0; i<2; i++){
        for(j=0; j<2; j++){
            printf("array[%d] [%d] = %d \n", i, j, array[i][j]);
        }
    }
    return 0;
}
```

~ END ~