

EE3220 assignment 2 sample solutions:

Instructions: In this assignment, for the questions asking for explanations, instructions or codes, the answers are not fixed, and you can get full marks if you answer the questions in your own words and comprehension with details and reasons.

Q1:

Please describe the functions of the assemble codes. For example, push and move represents the function of get or save certain value to the registers or stack pointers. Str or ltr means save or load the value into certain memory. Bn fl and Bx lr are used for link to functions or loops. Finally, the other arithmetic calculations or logical calculations represent certain calculation in the total function. And the connection between assembly and C programming can be shown by this process.

When describing the assembly code, you can mention that the values are saved in which registers or memory. For example, str r0, [r11, #-4] means that the machine received 1st argument at memory location r11-4.

For one difference between assembly and C programming is that you need to set the total process by yourself including sending the value to which register, but in C programming, you only need to design the logic with proper words and the computer will allocate the registers automatically.

For the whole question, you can describe in your own words. If you have answered each question with detailed instructions, you will get full marks.

Q2.

You can answer the question in your comprehension, for example, you can compare the complexity, process, efficiency, and other features of different loops in C and assembly in your own words.

Here is a sample solution of the comparison in process:

WHILE loop:

In C, the while loop is operated by execute and check whether the condition equals to true every iteration. In assembly, the while loop is compiled into 3 subroutines, the conditioning subroutine, the true subroutine, and the false subroutine.

IF:

Like the comparison above of a while loop, only the true subroutine does not manipulate the iterator and not branch back to the conditioner, but also breaks out of the conditional and jumps to the further expression.

Q3.

Flag: -O0.

foo1:

```
push    {r11, lr}
mov     r11, sp
sub     sp, sp, #16
mov     r0, #0
str     r0, [r11, #-4]
str     r0, [sp, #8]
```

```

        mov     r0, #1
        str     r0, [sp, #8]
        b       .LBB0_1
.LBB0_1:                                     @ =>This Inner Loop Header:
Depth=1
        ldr     r0, [sp, #8]
        cmp     r0, #8
        bhi     .LBB0_4
        b       .LBB0_2
.LBB0_2:                                     @   in Loop: Header=BB0_1
Depth=1
        ldr     r0, [sp, #8]
        bl      __aeabi_ui2f
        ldr     r1, [r11, #-4]
        str     r0, [sp, #4]                @ 4-byte Spill
        mov     r0, r1
        ldr     r1, [sp, #4]                @ 4-byte Reload
        bl      __aeabi_fadd
        str     r0, [r11, #-4]
        b       .LBB0_3
.LBB0_3:                                     @   in Loop: Header=BB0_1
Depth=1
        ldr     r0, [sp, #8]
        add     r0, r0, #1
        str     r0, [sp, #8]
        b       .LBB0_1
.LBB0_4:
        ldr     r0, [r11, #-4]
        mov     sp, r11
        pop     {r11, lr}
        bx      lr

-O1:
foo1:
        push    {r4, r5, r11, lr}
        mov     r5, #0
        mov     r4, #1
.LBB0_1:                                     @ =>This Inner Loop Header:
Depth=1
        mov     r0, r4
        bl      __aeabi_ui2f
        mov     r1, r0
        mov     r0, r5
        bl      __aeabi_fadd

```

```

mov    r5, r0
add    r4, r4, #1
cmp    r4, #9
bne    .LBB0_1
mov    r0, r5
pop    {r4, r5, r11, lr}
bx     lr

```

-O2:

```

foo1:
mov    r0, #34603008
orr    r0, r0, #1073741824
bx     lr

```

Q4.

armv7-a clang 11.0.1 without setting optimization level

```

foo2:
sub    sp, sp, #4
str    r0, [sp]
ldr    r0, [sp]
mul    r1, r0, r0
mov    r0, r1
add    sp, sp, #4
bx     lr

```

The data is saved in allocated registers.

x86-64 gcc 12.2 without setting optimization level

```

foo2:
push   rbp
mov     rbp, rsp
mov     DWORD PTR [rbp-4], edi
mov     eax, DWORD PTR [rbp-4]
imul    eax, eax
pop     rbp
ret

```

In this compile, the code saves rbp to stack. Rbp=rsp. Select the memory by dereferencing from [rbp-4]

Q5.

a) Two registers

b) ffoo3:

```

asr    r12, r1, #16
lsl    r1, r1, #16
asr    r1, r1, #16
mov    r3, #0

```

.LBB0_1:

@ =>This Inner Loop Header:

Depth=1

```
    ldr    r2, [r0, r3, lsl #2]
    add    r1, r1, r12
    add    r1, r1, r2
    add    r3, r3, #1
    cmp    r3, #8
    bne    .LBB0_1
    mov    r0, r1
    bx     lr
```

with O1 optimization, x is stored in r12 and y is stored in r1. The value is 0x000022640000267a.

c)

foo3:

```
    push   {r4, lr}
    ldm    r0, {r3, r12, lr}
    lsl    r2, r1, #16
    ldr    r4, [r0, #12]
    asr    r2, r2, #16
    add    r2, r2, r1, asr #16
    add    r2, r2, r3
    ldr    r3, [r0, #16]
    add    r2, r2, r1, asr #16
    add    r2, r2, r12
    add    r2, r2, r1, asr #16
    add    r2, r2, lr
    add    r2, r2, r1, asr #16
    add    r2, r2, r4
    add    r2, r2, r1, asr #16
    add    r2, r2, r3
    ldr    r3, [r0, #20]
    add    r2, r2, r1, asr #16
    add    r2, r2, r3
    ldr    r3, [r0, #24]
    ldr    r0, [r0, #28]
    add    r2, r2, r1, asr #16
    add    r2, r2, r3
    add    r1, r2, r1, asr #16
    add    r0, r1, r0
    pop    {r4, lr}
    bx     lr
```

It will have 8 iterations.

Q6.

Local Variables	Offset from Stack Pointer (SP)
s1	sp+52
s2	sp+48
s3	sp+44
t.x	sp+40
t.y	sp+42

Q7.

The assembly code in this question is generated by C code with the function of adding 2 arrays. You can design your C code from the assembly code.

Q8.

As an example:

```
int bar(int n) {  
    if (n == 0 | n == 1)  
        {return 1;}  
    else {  
return bar(n-1) * bar(n-2) + 3;  
        }  
    }  
}
```

Q9.

You can search for the information and materials for this question in your own words and comprehension. You will get full marks as long as answer the question with details or examples.