EE3220 Assignment 2 Reference Solutions:

Please note that this file is only a sample answer, and some questions do not have the fixed answer. Make sure you have solved with your own logic and you will get full marks.

Q1. Please refer to our lecture note about assembly and explain each line with the instructions.

Q2.

```
mov         r0, #6356992
orr         r0, r0, #1174405120
```

there are less loop iteration in o1 than o0. O2 use orr instruction.

Q3. As an example:

1)

```
square(int):
        sub     sp, sp, #4      ;
        str     r0, [sp]        ;
        ldr     r0, [sp]        ;
        mul     r1, r0, r0      ;
        mov     r0, r1          ;
        add     sp, sp, #4      ;
        bx      lr              ;
```

2)

```
square(int):
        push    rbp             ;
        mov     rbp, rsp        ;
        mov     DWORD PTR [rbp-4], edi    ;
[rbp-4]
        mov     eax, DWORD PTR [rbp-4]    ;
to eax
        imul    eax, eax        ;
        pop     rbp             ;
        ret                     ;
```

Q4. As an example:

1)

```
foo2:
        push    {r4, lr}
        mov     r4, r0
        mov     r0, #1
        cmp     r4, #1
        beq     .LBB0_2
        sub     r0, r4, #1
        bl      foo2
        add     r0, r0, r4
```

```
.LBB0_2:
            pop        {r4, lr}
            bx         lr
```

- R4 (preserved register) and link register (return address) are saved.

2)

```
foo2(int):
            cmp         r0, #0
            moveq       r0, #1
            bxeq        lr
            push        {r4, lr}
            mov         r4, r0
            sub         r0, r0, #1    ; n-1
            bl          foo2(int)
            add         r0, r0, r4    ; r0 = r0 + r4(r0)
            pop         {r4, lr}
            bx          lr
```

- There is no .LBB0_2 label because no push{r4, lr} is needed if n == 0

Q5.

1) 2 register r0 and r1
2) t.x and t.y is assessed with lsl and asr instructions.

```
t.x:        lsl       r1, r1, #16
            asr       r1, r1, #16
t.y:        ldr       r2, [r0, -r3, lsl #2]
```

3) cmp r0, #19 is updated to

   cmp r0, #4

It will exit the loop if i > 4, so it compares to 4 rather than 19

4) 20 times and 200 times relatively.

Q6.

You can explain it in your own word such as Reset handler is nothing but a normal function written in assembly or C language, which you want to get called whenever processor resets. . And offer relative examples.

Q7.

In the table: -12 -16 8 -22 -24

```
.L__const.main.list:
            .long     32                          @ 0x20
            .long     43                          @ 0x2b
            .long     54                          @ 0x36
            .long     65                          @ 0x41
            .long     91                          @ 0x5b
            .long     76                          @ 0x4c
```

```
          .long     32                              @ 0x20
          .long     29                              @ 0x1d
          .long     13                              @ 0xd
          .long     78                              @ 0x4e
```

Q8.

As an example:

```
Int unknown (int a)   {
              Return a * unknown (a-1);
                       }
Else {
               Return 1;
    }
```

Q9

Please explain the procedure in your own words.

Q10

Here is an example and you can finish in your own logic:

```
push      {r4, lr}     ; save r4 and lr on stake
          ldr       r4, .LCPI0_0   ; load the string to be print to r4
          mov        r1, #0       ; i = 0
          mov        r2, #0       ; j = 0
          mov        r3, #1       ; array[0][0] = 1
          mov        r0, r4       ; string move to r4
          bl        printf       ; print
          mov        r0, r4       ; string move to r4
          mov        r1, #0       ; i = 0
          mov        r2, #1       ; j = 1
          mov        r3, #2       ; array[0][1] = 2
          bl        printf       ; print
          mov        r0, r4       ; string move to r4
          mov        r1, #1       ; i = 1
          mov        r2, #0       ; j = 0
          mov        r3, #3       ; array[1][0] = 3
          bl        printf       ; print
          mov        r0, r4       ; string move to r4
          mov        r1, #1       ; i = 1
          mov        r2, #1       ; j = 1
          mov        r3, #4       ; array[1][1] = 3
          bl        printf       ; print
          mov        r0, #0       ; return 0 (r0)
          pop        {r4, lr}     ; pop r4 and lr
          bx         lr
```

```
.LCPI0_0:
            .long      .L.str        ; point to string
.L.str:
            .asciz    "array[%d] [%d] = %d \n"    ; string
```