Assignment-5 Assembly Programming Problems

Full Credit: 20 pts | Due: Sunday 3/20

In this assignment you are asked to translate C code to ARM64/ARMv8 assembly code, or trace assembly code. C code variables should be mapped to allocated stack frame space or ARMv8 registers in assembly. You must add comment to each line of assembly code. Test your code with Dr. Shuqun Zhang's AMR64 Online Simulator.

Q1. Convert the following C functions into ARMv8 assembly language. Again, comment each line of assembly code on what it does. Note that local variables should be kept in function's stack frame.

1) Testing arithmetic operations

```
int arithOps( int a, int b ) {
    int u, v, w;
    u = a << 2;
    v = b >> 2;
    w = u + v;
    return w;
}
sub sp, sp, #32
str w0, [sp, #28]
str w1, [sp, #24]
ldr w8, [sp, #28]
lsl w8, w8, #2
str w8, [sp, #20]
ldr w8, [sp, #24]
asr w8, w8, #2
str w8, [sp, #16]
ldr w8, [sp, #20]
ldr w9, [sp, #16]
add w8, w8, w9
str w8, [sp, #12]
ldr w0, [sp, #12]
add sp, sp, #32
ret
```

2) Doubles the input value

```
int doubleMe( int a ) {
    int tmp;
    tmp = 2 * a;
    // tmp = a + a;
    return tmp;
}
sub sp, sp, #16

str w0, [sp, #12]

ldr w9, [sp, #12]

mov w8, #2
```

```
mul w8, w8, w9

str w8, [sp, #8]

ldr w0, [sp, #8]

add sp, sp, #16

ret
```

3) Main function to test the above two functions

```
int main( ) {
    int x = -3;
    int y = 28;
    int z1, z2;

    z1 = arithOps(x, y);
    z2 = doubleMe(x);

    return 0;
}
  ; int main()

         push    rbp

        mov     rbp, rsp

        sub     rsp, 16
   ; int x = -3;

        mov     DWORD PTR [rbp-4], -3
   ; int y = 28;

        mov     DWORD PTR [rbp-8], 28
   ;z1 = arithOps(x,y);

        mov     edx, DWORD PTR [rbp-8]

        mov     eax, DWORD PTR [rbp-4]

        mov     esi, edx

        mov     edi, eax

        call    arithOps(int, int)

        mov     DWORD PTR [rbp-12], eax
   ;z2 = doubleme(x);

        mov     eax, DWORD PTR [rbp-4]

        mov     edi, eax

        call    doubleme(int)

        mov     DWORD PTR [rbp-16], eax
   ;return 0;
        mov     eax, 0
        leave
        ret
```

Q2. [Code-tracing] Given the following ARM64 assembly code, add comments to each line, trace the code execution to figure out what it does, and finally convert it back to its corresponding C-code. Notice that local variables are stored in the main function's stack frame, drawing the stack frame would help you understand the code better.

```
main:
        sub     sp, sp, #32         stack pointer increase to 32 memory
        str     wzr, [sp, 28] // a  store at sp+28
        mov     w0, 1               copy value
        str     w0, [sp, 24] // b   store at sp+24
        str     wzr, [sp, 16] // c  store at sp+16
        mov     w0, 8               copy the data
        str     w0, [sp, 12] // n   store at sp+12
        mov     w0, 1               copy the data
        str     w0, [sp, 20] // i   store at sp+20
        b       Checking
Loop:
        ldr     w1, [sp, 28]        load into register
        ldr     w0, [sp, 24]        load into register
        add     w0, w1, w0
        str     w0, [sp, 16]        store at sp+16
        ldr     w0, [sp, 24]        load into register
        str     w0, [sp, 28]        store at sp+28
        ldr     w0, [sp, 16]        load into register
        str     w0, [sp, 24]        store at sp+24
        ldr     w0, [sp, 20]        load into register
        add     w0, w0, 1           addition
        str     w0, [sp, 20]        store at sp+20
Checking:
        ldr     w1, [sp, 20]        load into register
        ldr     w0, [sp, 12]        load into register
        cmp     w1, w0              compare
        b.lt    Loop
        mov     w0, 0               copy
        add     sp, sp, 32          addition
        ret                         return address
```

```
int main(int r, int t)
{
int a,b, c;
a = r<<2;
b = t>>2;
c = a+b;
return c ;
}
int doublelooping(int r){
```

```
int res;
res= 2*a;
return res;
}
```

Q3. [Code-tracing] Trace the following program, comment every line of code on what it does, draw call stack diagram to keep track on stack growing and shrinking movements. Remember that each function call needs a stack frame for saving return address and local variables. Explain what the overall program does. When function calls finish and back to main, what's the result stored at [sp, 20]?

```
main:
        stp     x29, x30, [sp, -32]!       stack at the address -32
        mov     x29, sp                    copy v in 29
        mov     w0, 3                      w0 has value 3
        str     w0, [sp, 28]               at address 28 stack pointer
        mov     w0, 5                      now 5 moving to w0
        str     w0, [sp, 24]               address 24 will store
        ldr     w1, [sp, 24]               load address
        ldr     w0, [sp, 28]               w0 has value of 3
        bl      findMaxIncr                branch to link calling function 6
        str     w0, [sp, 20]               store at stack pointer 20
        mov     w0, 0                      value 0
        ldp     x29, x30, [sp], 32         loading pointer
        ret

findMaxIncr:
        stp     x29, x30, [sp, -48]!       stack at the address -48
        mov     x29, sp                    copy to register
        str     w0, [sp, 28]               at address 28 stack pointer
        str     w1, [sp, 24]               at address 24 stack pointer
        ldr     w1, [sp, 28]               load address
        ldr     w0, [sp, 24]               load address
        cmp     w1, w0                     Both comparing
        b.ge    greatTE                    w1>w0
        ldr     w0, [sp, 24]               adding 24
        str     w0, [sp, 44]               store at 44
        b       commonPart                 calling function
greatTE:
        ldr     w0, [sp, 28]               load data
        str     w0, [sp, 44]               moving to stack pointer 44
commonPart:
        ldr     w0, [sp, 44]               having value of 5
        bl      incrOne                    function call
        str     w0, [sp, 40]               at address 40 stack pointer
        ldr     w0, [sp, 40]               loading pointer
        ldp     x29, x30, [sp], 48         7 is in sp 48
        ret

incrOne:
        sub     sp, sp, #16                incrementing value
        str     w0, [sp, 12]               store at 12
        ldr     w0, [sp, 12]               load at 12
        add     w0, w0, 1                  adding
        add     sp, sp, 16                 adding
        ret                                return which store in stack pointer
```

```
Taking two values into two registers and comparing them, and finding greater
value and later it is incrementing the value which come from function and
returning to the main.
```

Submission Instructions: Place all your source code answers either in ARM64 assembly or C into a single file, name it as **LastFirstInitial_a5.c**, which must include all required comments. Put all explanations and drawings of call stack movements into a separate pdf file, name it as LastFirstInitial_a5.pdf, and then upload them through your blackboard account by the due day.

Note: If you cannot write into the pdf directly, then you can work on your assignment in Microsoft Word and then save it as pdf before submission.