EE234

Microprocessor Systems

Final Exam

Dec. 15, 2020. (11am – 1:30pm)

Instructor: Dae Hyun Kim (daehyun@eecs.wsu.edu)

Name:

WSU ID:

Problem	Points	
1	20	
2	20	
3	30	
4	20	
5-1	20	
5-2	30	
6	20	
Total	160	

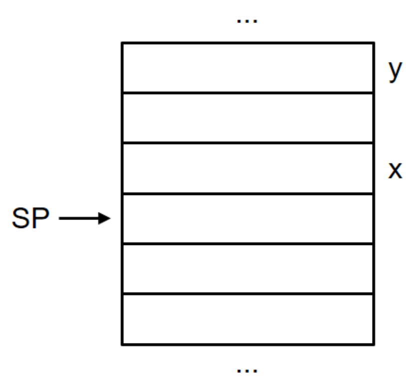
Problem 1 (1-D Array, 20 points)

All the registers R# are 32-bit registers. “int” is a 32-bit signed integer data type. Write an assembly code for the “for” loop in the following C code. The memory figure shows the stack pointer (SP) and the locations of the variables x and y.

```
int y[10];
int* x = new int[10];

...

for ( int k = 0 ; k < 10 ; k++ )
  x[k] = y[k];
```




```
        MOV R0, #0  // k
loop1:
        CMP R0, #10
        BGE finish
        MUL R1, R0, #4       // 4*k
        ADD R2, R1, SP
        LDR R2, [R2, #12]    // R2 = y[k]
        LDR R3, [SP, #4]     // x
        ADD R3, R3, R1       // &(x[k])
        STR R2, [R3]         // x[k] = R2 = y[k]
        ADD R0, R0, #1
        B loop1
finish:
```

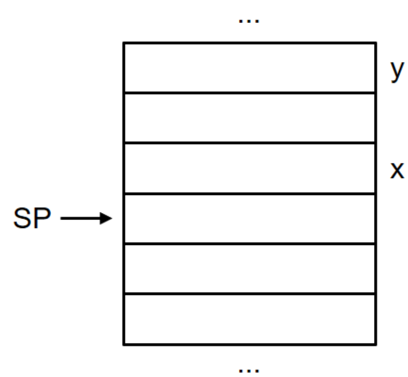
Problem 2 (1-D Array, 20 points)

All the registers R# are 32-bit registers. “long” is a 64-bit signed integer data type. Write an assembly code for the “for” loop in the following C code. The memory figure shows the stack pointer (SP) and the locations of the variables x and y.

```
long y[10];
long* x = new long[10];

...

for ( int k = 0 ; k < 10 ; k++ )
  x[k] = y[k];
```




```
        MOV R0, #0  // k
loop1:
        CMP R0, #10
        BGE finish
        MUL R1, R0, #8      // 8*k
        ADD R2, R1, SP
        LDR R2, [R2, #12]   // R2 = LO(y[k])  LO is the lower 4B.
        LDR R3, [R2, #16]   // R3 = HI(y[k])  HO is the upper 4B.
        LDR R4, [SP, #4]    // x
        ADD R4, R4, R1      // &(LO(x[k]))
        STR R2, [R4]        // LO(x[k]) = R2 = LO(y[k])
        STR R3, [R4, #4]    // HI(x[k]) = R3 = HI(y[k])
        ADD R0, R0, #1
        B loop1
finish:
```

Problem 3 (2-D Array, 20 points)

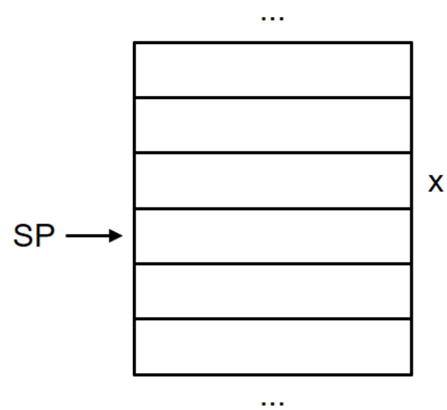
All the registers R# are 32-bit registers. “int” is a 32-bit signed integer data type. Write an assembly code for the nested “for” loop in the following C code. The memory figure shows the stack pointer (SP) and the locations of the variables x and y.

```
int** x = new int*[2];

for ( int k = 0 ; k < 2 ; k++ )
  x[k] = new int[4];

...

for ( int i = 0 ; i < 2 ; i++ ) {
  for ( int k = 0 ; k < 4 ; k++ ) {
    x[i][k] = 0;
  }
}
```



Note: Try to minimize the number of memory access instructions (LDR, STR) executed. However, you should strictly follow the flow of the program (i.e., you should have two nested loops, etc.).

```
        MOV R0, #0    // int i = 0
        MOV R12, #0  // constant 0
        LDR R3, [SP, #4]        // x
loop1:
        CMP R0, #2
        BGE finish
        MUL R2, R0, #4          // 4*i
        ADD R4, R2, R3          // &(x[i])
        LDR R4, [R4]            // x[i]
        MOV R1, #0    // int k = 0
loop2:
        CMP R1, #4
        BGE loop3
        MUL R2, R2, #4          // 4*k
        ADD R4, R4, R2          // &(x[i][k])
        STR R12, [R4]           // x[i][k] = 0
        ADD R1, R1, #1
        B loop2
loop3:
        ADD R0, R0, #1          // i++
        B loop1
finish:
```

Problem 4 (Estimation of Memory Consumption, 20 points)

Estimate how many bytes are used for the array x in the following C code. You should include the memory space used for variable x itself. “int” is a 32-bit signed integer data type.

```
int**** x = new int***[3];

for ( int k = 0 ; k < 3 ; k++ )
  x[k] = new int**[4];

for ( int i = 0 ; i < 3 ; i++ ) {
  for ( int k = 0 ; k < 4 ; k++ ) {
    if ( k % 2 == 0 ) {
      x[i][k] = new int*[5];
      for ( int m = 0 ; m < 3 ; m++ )
        x[i][k][m] = new int[6];
    }
  }
}
```

x needs 4B.

The first “new” operation reserves 3 int*** spaces → 3*4B=12B.

The first “for” loop reserves 4 int** spaces for each x[k] → 3*(4*4B) = 48B.

In the nested “for” loop, the “if” statement is executed six times. In the “if” statement, the “new” statement reserves 5 int* spaces → 6*(5*4B) = 120B.

In the “for” loop in the “if” statement, the “new” operation reserves 6 int spaces. → 6*(3*(6*4B)) = 432B.

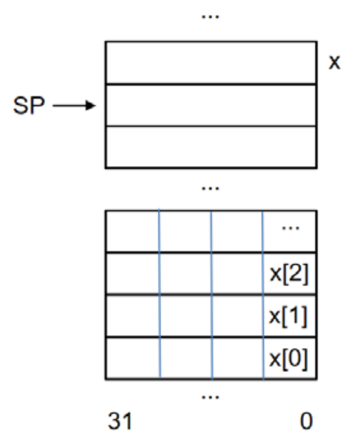
Thus, total 616 Bytes.

Problem 5-1 (Array Manipulation I, 20 points)

The “char” data type in C is used to represent 1 byte. If you need an array of M char-type variables, you will ideally need M bytes. However, all the memory addresses for LDR and STR instructions should be integer multiples of 4 in the 32-bit ARM architecture (so, for example, you cannot use 0x0001 for a target memory address). Now, let’s take a look at the following C code. It reserves memory space for 80 characters, so ideally it should reserve 80 Bytes in the heap memory. However, it requires some bit manipulations. Thus, a compiler can reserve 320 Bytes in the heap memory and use only the lease significant 1B in each word for each x[k] as follows.

```
char* x = new char[80];

for ( int k = 0 ; k < 80 ; k++ )
  x[k] = k;
```



Write an assembly code for the “for” loop in the C code shown above. The memory management should be the same as the compiler above.

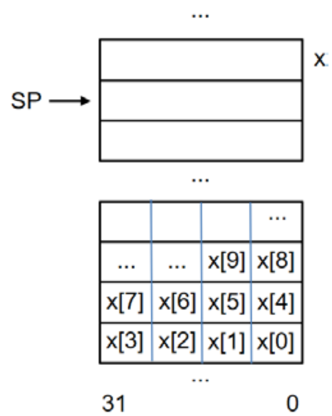
```
        LDR R0, [SP, #4]        // R0 = x
        MOV R1, #0    // k = 0
loop1:
        CMP R1, #80
        BGE finish
        MUL R2, R1, #4          // 4*k
        ADD R2, R0, R2          // &(x[k])
        STR R1, [R2]            // x[k] = k
        ADD R1, R1, #1
        B loop1
finish:
```

Problem 5-2 (Array Manipulation II, 30 points)

For the C code in Problem 5-1, a different compiler reserves exactly 80 Bytes in the heap space as follows.

```c
char* x = new char[80];

for ( int k = 0 ; k < 80 ; k++ )
  x[k] = k;
```



Write an assembly code for the “for” loop in the C code shown above. The memory management should be the same as the new compiler explained above.

```
        LDR R0, [SP, #4]        // R0 = x
        MOV R1, #0    // k = 0
loop1:
        CMP R1, #80
        BGE finish
        MOV R2, R1, LSR #2 // k/4
        MUL R2, R2, #4          // 4*(k/4)
        ADD R2, R0, R2
        LDR R3, [R2]            // load x[a+3:a]
        AND R4, R1, #0x3        // k%4
        MUL R4, R4, #8          // # bits to shift to the left
        MOV R5, R1, LSL R4 // shift k to the left
        MOV R6, #0xFF
        MOV R6, R6, LSL R4 // shift 000...011111111 to the left
        LDR R7, =#0xFFFFFFFF
        EOR R6, R6, R7          // inversion of R6
        AND R6, R3, R6          // finally we get x[a+3] x[a+2] 00000000 x[a] (assuming x[a+1]=k)
        ORR R6, R6, R5          // x[a+3] x[a+2] k x[a]
        STR R6, [R2]
        ADD R1, R1, #1          // k++
        B loop1
finish:
```

Problem 6 (C, 20 points)

All the registers R# are 32-bit registers. "int" is a 32-bit signed integer data type and "long" is a 64-bit signed integer data type. The following table shows the main memory.

```
int** x = new int*[a];

for ( int i = 0 ; i < a ; i++ )
  x[i] = new int[b];
```

"a" and "b" are some constants. Currently, the value of x is 0x4000 as shown in the figure.

Address	Data (31 … 0)	
0x8000	0x4000	x
...		
0x402C	0x4000	
0x4028	0x402C	
0x4024	0x4028	
0x4020	0x4024	
0x401C	0x4020	
0x4018	0x401C	
0x4014	0x4018	
0x4010	0x4014	
0x400C	0x4010	
0x4008	0x4024	
0x4004	0x4014	
0x4000	0x400C	

(a) What is the value of *((int*) x)? 0x400C

(b) What is the value of *((long*) x)? 0x4014 400C

(c) What is the value of x[2]? 0x4024

(d) What is the value of x + 3? 0x400C

(e) What is the value of (x[0]+2)? 0x4014

(f) What is the value of x[1][2]? 0x4020

(g) int* y = x[1]. What is the value of y[3]? 0x4024

(h) long** y = (long**) x. What is the value of y[1]? 0x4014

(i) What is the value of &(x[2])? 0x4008

(j) long* y = (long*) x[0]. What is the value of y[1]? 0x401C 4018