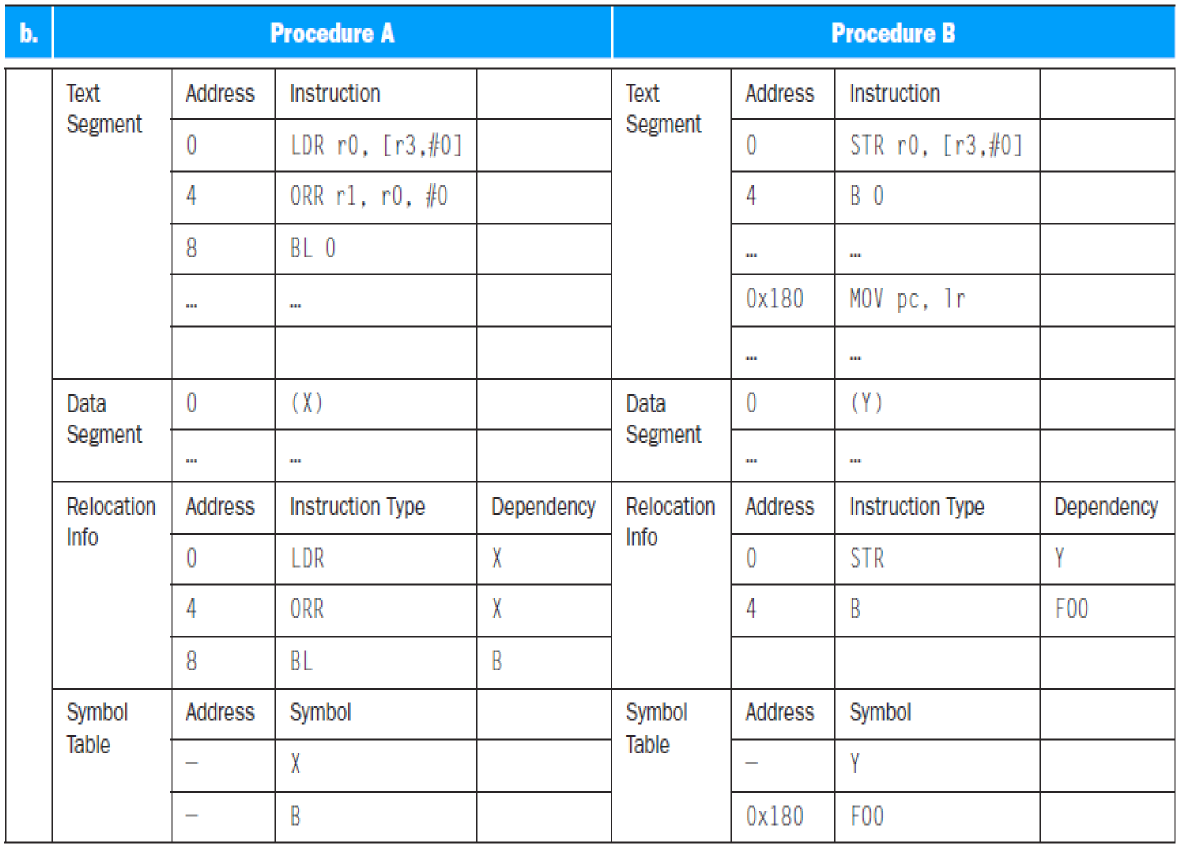
CISC 260 Machine Organization and Assembly Language (Spring 2018)

Assignment # 6
(Due: May 3, 2018)

Problem 1 [20 points]:



b.	Procedure A				Procedure B			
	Text Segment	Address	Instruction		Text Segment	Address	Instruction	
		0	LDR r0, [r3,#0]			0	STR r0, [r3,#0]	
		4	ORR r1, r0, #0			4	B 0	
		8	BL 0			…	…	
		…	…			0x180	MOV pc, lr	
						…	…	
	Data Segment	0	(X)		Data Segment	0	(Y)	
		…	…			…	…	
	Relocation Info	Address	Instruction Type	Dependency	Relocation Info	Address	Instruction Type	Dependency
		0	LDR	X		0	STR	Y
		4	ORR	X		4	B	FOO
		8	BL	B				
	Symbol Table	Address	Symbol		Symbol Table	Address	Symbol	
		–	X			–	Y	
		–	B			0x180	FOO	

Assume that Procedure A has a text size of 0x140, data size of 0x40 and Procedure B has a text size of 0x300 and data size of 0x50. Also assume r3 = 0x1000 0000, and the memory allocation strategy as shown in Figure 2.13 (Chapter 02_COD 4e ARM.pdf). Link the object files above to form the executable file, by merging A with B (A goes first). Give explicitly the addresses for instructions and data in the executable and update the symbol table.

Problem 2. [20 points] (P&H ARM Edition) Exercise 2.41
Assume for a given processor the CPI of arithmetic instructions is 1, the CPI of load/store instructions is 10, and the CPI of branch instructions is 3. Assume a program has the following instruction breakdowns: 500 million arithmetic instructions 300 million load/store instructions, 100 million branch instructions.

2.41.1 Suppose that new, more powerful arithmetic instructions are added to the instruction set. On average, through the use of these more powerful arithmetic instructions, we can reduce the number of arithmetic instructions needed to execute a program by 25%, while increasing the clock cycle time by only 10%. Is this a good design choice? Why?

2.41.2 Suppose that we find a way to double the performance of arithmetic instructions. What is the overall speedup of our machine? What if we find a way to improve the performance of arithmetic instructions by 10 times?

Problem 3. [20 points] (P&H ARM Edition) Exercise 2.42
Assume that for a given program 70% of the executed instructions are arithmetic, 10% are load/store, and 20% are branch.

2.42.1 Given this instruction mix and the assumption that an arithmetic instruction requires two cycles, a load/store instruction takes six cycles, and a branch instruction takes three cycles, find the average CPI.

2.42.2 For a 25% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?

2.42.3 For a 50% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?

Problem 4. [40 points]

a) [15 points] Modify the following C code so that the recursion is tail-recursion.

b) [25 points] Translate the tail recursion version into ARM assembly code.

```
funct(int x) {

  if (x <= 0) return 0;
  else if (x & 0x1) {
     return x + funct(x-1);
  } else {
     return x - funct(x-1);
  }
}
```

Bonus Problem [30 points] In the following assembly code, the main function reads integers from an array, calls subroutine "fact" to compute the factorial of each integer N, and prints the result fact(N) to the screen. You are asked to revise the code such that for each integer N, if N is larger than 10, it modifies the subroutine "fact" on-the-fly to compute 1+2 + …+N, instead of 1 x 2 x … x N.

self-modifying code

```
main:
        ldr r2, =myarray
        mov r3, #0
        mov r4, #4
LOOP:   cmp r3, r4
        bgt Stop

        ldr r0, [r2, r3, LSL #2]
        bl fact

        mov r1, r0
        MOV r0, #1     @ Load 1 into register r0 (stdout handle)
        SWI 0x6b      @ Print integer in register r1 to stdout

        @ print a space
        mov    r0,     #1
        ldr    r1,     =Space
        swi    0x69   @ write string to stdout

        add r3, r3, #1
        b LOOP
Stop:   SWI 0x11      @ Stop program execution

fact: sub sp, sp, #8
        str lr, [sp,#0]
        str r0, [sp,#4]
        cmp r0,#1
        bge L1
        mov r0, #1
        add sp, sp, #8
        mov pc, lr
L1: sub r0, r0, #1
        BL fact
        mov r1, r0
        ldr r0, [sp, #4]
        ldr lr, [sp, #0]
        add sp, sp, #8
        mul r0, r1, r0
        mov pc, lr
```

factorial

want to check the integer, so that the code is capable of modifying the integer on the fly?

Not allowed to write 2 editions of the subroutine—both add and mul

on the fly—modify the mul with add

```
.data
myarray: .word 2, 3, 14, 5, 6
Space: .ascii " "
```