CSE2312-002/003, Fall 2014, Homework 4

Due October 7, 2014 in Class (at 2:00pm for 002, 3:30pm for 003)

The following problems are from Chapter 2 of the ARM Edition of the Patterson and Hennessy textbook (available on Blackboard as PDFs under Course Materials). In addition to the resources available in Chapter 2, you may find Appendices B1 and B2 of the ARM Edition useful for these problems (also available as PDFs on Blackboard). Appendix A of the book may also be useful.

The problems below respectively correspond to the book problems 2.19 (2.19.1 through 2.19.4), 2.20 (2.20.4 through 2.20.6), and 2.24 (all), and a Bonus: 2.28 (all).

Exercise 2.19

For the following problems, the table holds C code functions. Assume that the first function listed in the table is called first. You will be asked to translate these C code routines into ARM Assembly.

a.

```
int compare(int a, int b) {
   if (sub(a, b) >= 0)
      return 1;
   else
      return 0;
}
int sub (int a, int b) {
   return a-b;
}
```

b.

```
int fib_iter(int a, int b, int n){
    if(n == 0)
      return b;
   else
      return fib_iter(a+b, a, n-1);
}
```

2.19.1 [15] <2.8> Implement the C code in the table in ARM assembly. What is the total number of ARM instructions needed to execute the function?

2.19.2 [5] <2.8> Functions can often be implemented by compilers “in-line”. An in-line function is when the body of the function is copied into the program space, allowing the overhead of the function call to be eliminated. Implement an “in-line” version of the C code in the table in ARM assembly. What is the reduction in the total number of ARM assembly instructions needed to complete the function? Assume that the C variable n is initialized to 5.

2.19.3 [5] <2.8> For each function call, show the contents of the stack after the function call is made. Assume the stack pointer is originally at address `0x7ffffffc`, and follow the register conventions as specified in Figure 2.11.

The following three problems in this exercise refer to a function f that calls another function func. The code for C function func is already compiled in another module using the ARM calling convention from Figure 2.14. The function declaration for func is "`int func(int a, int b);`". The code for function f is as follows:

a.
```
int f(int a, int b, int c){
    return func(func(a,b),c);
}
```

b.
```
int f(int a, int b, int c){
    return func(a,b)+func(b,c);
}
```

2.19.4 [10] <2.8> Translate function f into ARM assembler, also using the ARM calling convention from Figure 2.14. If you need to use registers r4 through r11, use the lower-numbered registers first.

Exercise 2.20

For the following problems, the table has an assembly code fragment that computes a Fibonacci number. However, the entries in the table have errors, and you will be asked to fix these errors.

a.
```
FIB:        SUB     sp, sp, #12
            STR     lr,[sp,#0]
            STR     r2,[sp,#4]
            STR     r1,[sp,#8]
            CMP     r1,#1
            BGE     L1
            MOV     r0,r1
            B       EXIT

L1:         SUB     r1,r1,#1
            BL      FIB
            MOV     r2,r0
            SUB     r1,r1,#1
            BL      FIB
            ADD     r0,r0,r2

EXIT:       LDR     lr,[sp,#0]
            LDR     r1,[sp,#8]
            LDR     r2,[sp,#4]
            ADD     sp,sp,#12
            MOV     pc,lr
```

b.

```
FIB:        SUB   sp,sp,#12
            STR   lr,[sp,#0]
            STR   r2,[sp,#4]
            STR   r1,[sp,#8]
            CMP   r1,#1
            BGE   L1
            MOV   r0,r1
            B     EXIT

      L1:   SUB   r1,r1,#1
            BL    FIB
            MOV   r2,r0
            SUB   r1,r1,#1
            BL    FIB
            ADD   r0,r0,r2
      EXIT: LDR   lr,[sp,#0]
            LDR   r1,[sp,#8]
            LDR   r2,[sp,#4]
            ADD   sp,sp,#12
            MOV   pc,lr
```

2.20.4 [5] <2.8> The ARM assembly program above computes the Fibonacci of a given input. The integer input is passed through register r1, and the result is returned in register r0. In the assembly code, there are a few errors. Correct the ARM errors.

2.20.5 [10] <2.8> For the recursive Fibonacci ARM program above, assume that the input is 4. Rewrite the Fibonacci program to operate in a nonrecursive manner. What is the total number of instructions used to execute your solution?

2.20.6 [5] <2.8> Show the contents of the stack after each function call, assuming that the input is 4.

Exercise 2.24

Assume that the register r1 contains the address 0x1000 0000 and the register r2 contains the address 0x1000 0010.

a.

```
LDRB  r0, [r1,#0]
STRH  r0, [r2,#0]
```

b.

```
LDRB  r0, [r1,#0]
STRB  r0, [r2,#0]
```

2.24.1 [5] <2.9> Assume that the data (in hexadecimal) at address `0x1000 0000` is:

1000 0000	12	34	56	78

What value is stored at the address pointed to by register r2? Assume that the memory location pointed to r2 is initialized to `0xFFFF FFFF`.

2.24.2 [5] <2.9> Assume that the data (in hexadecimal) at address `0x1000 0000` is:

1000 0000	80	80	80	80

What value is stored at the address pointed to by register r2? Assume that the memory location pointed to r2 is initialized to `0x0000 0000`.

2.24.3 [5] <2.9> Assume that the data (in hexadecimal) at address `0x1000 0000` is:

1000 0000	11	00	00	FF

What value is stored at the address pointed to by register r2? Assume that the memory location pointed to r2 is initialized to `0x5555 5555`.

Exercise 2.28 (Bonus)

The following table contains ARM assembly code for a lock.

```
try:  MOV   R3,#1
      SWP   R2,R3,[R1,#0]
      CMP   R2,#1
      BEQ   try
      LDR   R4,[R2,#0]
      ADD   R3,R4,#1
      STR   R3,[R2,#0]
      SWP   R2,R3,[R1,#0]
```

2.28.1 [5] <2.11> For each test and fail of the “swp”, how many instructions need to be executed?

2.28.2 [5] <2.11> For the swp-based lock-unlock code above, explain why this code may fail.

2.28.3 [15] <2.11> Re-write the code above so that the code may operate correct. Be sure to avoid any race conditions.

Each entry in the following table has code and also shows the contents of various registers. The notation, “(r1)” shows the contents of a memory location pointed to by register r1. The assembly code in each table is executed in the cycle shown on parallel processors with a shared memory space.

a.

Processor 1	Processor 2	Cycle	Processor 1		MEM	Processor 2	
			r2	r3	(r1)	r2	r3
		0	1	2	99	30	40
SWP R2,R3, [R1,#0]		1					
	SWP R2,R3,[R1,#0]	2					

b.

Processor 1	Processor 2	Cycle	Processor 1			MEM	Processor 2		
			r2	r3		(r1)	r2	r3	
		0	2	3		1	10	20	
	try: MOV R3,#1	1							
try: MOV R3,#1	SWP R2,R3,[R1,#0]	2							
SWP R2,R3,[R1,#0]		3							
CMP R2,#1		4							
BEQ try	CMP R2,#1	5							
	BEQ try	6							

2.28.4 [5] <2.11> Fill out the table with the value of the registers for each given cycle.