EE234

Microprocessor Systems

Final Exam

Dec. 12, 2019. (3:10pm – 5:10pm)

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

Name:

WSU ID:

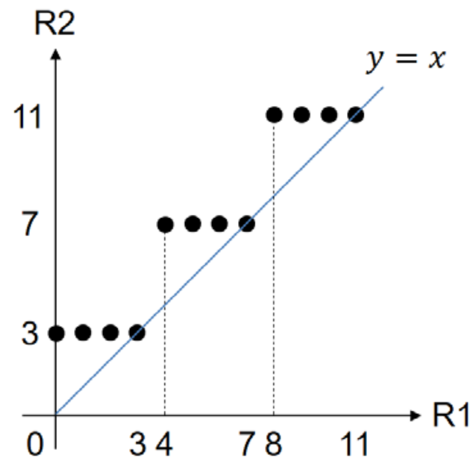
Problem	Points	
1	10	
2	10	
3	10	
4	10	
5	10	
6	10	
7	20	
8	20	
9	10	
Total	110	

Problem #1 (Bit manipulation, 10 points)

Suppose R# is an 8-bit register. The data stored in R# is treated as an unsigned binary number. Draw a graph for the following instruction. The x-axis should be the value stored in R1 and the y-axis should be the value stored in R2.

OR R2, R1, #0x03

The two LSBs are always 1. Thus, $x_7 \ldots x_2 x_1 x_0$ is mapped to $x_7 \ldots x_2 11$.



Problem #2 (Bit manipulation, 10 points)

Suppose R# is an 8-bit register. The data stored in R# is treated as an unsigned binary number. We want to calculate the following for given input R_1 (% is the MOD operation):

$$R_2 = 240 + 2 * (R_1\%16) - R_1$$

R_1 is stored in register R1 (input) and R_2 is the result that will be stored in register R2. The above function can be implemented by a single assembly instruction with a certain constant C as follows:

[ ] R2, R1, #C

Find the instruction and the constant. Notice that the instruction is one of the instructions shown in the last page. Hint: Express R2=$y_7 \dots y_0$ with respect to R1=$x_7 \dots x_0$. Then, find the relationship between R2 and R1.

Suppose $R_1 = x_7x_6 \dots x_1x_0$ and $R_2 = y_7y_6 \dots y_1y_0$.

$$R_1\%16 = 0000x_3x_2x_1x_0$$

$$R_1 + R_2 = 240 + 2 * (R_1\%16) = 0\text{x}F0 + x_3x_2x_1x_0 + x_3x_2x_1x_0$$

$$\begin{array}{rcccccccc}
 & x_7 & x_6 & x_5 & x_4 & x_3 & x_2 & x_1 & x_0 \\
+ & y_7 & y_6 & y_5 & y_4 & y_3 & y_2 & y_1 & y_0 \\
\hline
 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
+ & 0 & 0 & 0 & 0 & x_3 & x_2 & x_1 & x_0 \\
+ & 0 & 0 & 0 & 0 & x_3 & x_2 & x_1 & x_0
\end{array}$$

Thus, R2 should be $\overline{x_7x_6x_5x_4}x_3x_2x_1x_0$. To obtain this from R1, we need $R_1 \oplus 0\text{x}F0$.

Answer: instruction = EOR, constant C = 0xF0

EOR R2, R1, #0xF0

Problem #3 (Assembly, 10 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.

```
int x[30];  // given
int y[30];

// write an assembly code for the following for loop.
for ( int i = 0 ; i < 30 ; i++ )
  y[i] = x[i];
```

- &(x[0]): 0x4000
- &(y[0]): 0x5000

```
LDR R1, =0x4000
LDR R2, =0x5000
MOV R4, #0

loop:
  LDR R3, [R1]
  STR R3, [R2]
  ADD R1, R1, #4
  ADD R2, R2, #4
  ADD R4, R4, #1
  CMP R4, 30
  BNE loop
```

Problem #4 (Assembly, 10 points)

All the registers R# are 32-bit registers. The following (left) shows an array x of 32-bit data and an array y of 32-bit data. Each of them has 20 elements. Now, we want to copy the data as shown in the figure.

Address	Data (31 … 0)	
0x604C		y[19]
...		
0x6004		y[1]
0x6000		y[0]
...	...	
0x404C	0xAAC9	x[19]
0x4040	0xFF74	x[18]
...	...	
0x4004	0x3522	x[1]
0x4000	0x1827	x[0]

→

Address	Data (31 … 0)	
0x604C	0x1827	y[19]
...	...	
0x6004	0xFF74	y[1]
0x6000	0xAAC9	y[0]
...	...	
0x404C	0xAAC9	x[19]
0x4040	0xFF74	x[18]
...	...	
0x4004	0x3522	x[1]
0x4000	0x1827	x[0]

Write an assembly code for the data copy. Basically it does the following:

```
for ( int i = 0 ; i < 20 ; i++ )
   y[19-i] = x[i];
```

```
LDR R1, =#0x4000  // MOV R1, #0x4000 is acceptable.
LDR R2, =#0x604C

loop:
  LDR R3, [R1]
  STR R3, [R2]
  ADD R1, R1, #4
  SUB R2, R2, #4
  CMP R1, #4050
  BNE loop
```

Problem #5 (Assembly, 10 points)

The “unsigned char” data type is used for an 8-bit (one-byte) data. However, we cannot access them individually unless they are word-aligned. See the following example.

Address	Data 31–24	23–16	15–8	7–0
...				
0x4008		...	0xAA x[9]	0x00 x[8]
0x4004	0x12 x[7]	0x34 x[6]	0x56 x[5]	0x78 x[4]
0x4000	0xF0 x[3]	0x11 x[2]	0xEE x[1]	0xBC x[0]

→

Address	Data 31–24	23–16	15–8	7–0
...				
0x4008	0xAA	0x00 ...	x[11]	x[10]
0x4004	0x56 x[5]	0x78 x[4]	0x12 x[7]	0x34 x[6]
0x4000	0xEE x[1]	0xBC x[0]	0xF0 x[3]	0x11 x[2]

Suppose you declare “unsigned char x[20];” as shown above (left). Then, the address of x[0] is 0x4000, that of x[1] is 0x4001, that of x[2] is 0x4002, etc. However, the addresses like 0x4001 and 0x4002 are not word-aligned (i.e., not integer multiples of 4), so you cannot access them using something like “LDR R1, =#0x4001” and “LDR R2, [R1]”. All the addresses must be word-aligned (integer multiples of 4).

Write an assembly code to rearrange the given data “unsigned char x[20]” as shown above (right). Notice that it is just rearranging the data. It does not change the address of the array, i.e., &(x[0]) is still 0x4000, &(x[1]) is still 0x4001, etc. after the rearrangement.

```
    LDR R1, =#0x4000
loop:
    LDR R2, [R1]
    MOV R3, R2
    LSL R2, #16  // R2 = x[1] x[0] 0x0000
    LSR R3, #16  // R3 = 0x0000 x[3] x[2]
    OR R2, R2, R3  // R2 = x[1] x[0] x[3] x[2]
    STR R2, [R1]
    ADD R1, R1, #4
    CMP R1, #0x4014
    BNE loop
end:
    // end
```

Problem #6 (C, 10 points)

All the registers R# are 32-bit registers and everything is based on the 32-bit ARM architecture. How many <u>bytes</u> will C actually use for the following code (including the memory space for x in the stack)?

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

x[0] = new int*[3];
x[1] = new int*[4];

x[0][0] = new int[2];
x[0][1] = new int[3];
x[0][2] = new int[2];
x[1][0] = new int[2];
x[1][3] = new int[5];
```

x: 4B.

x[0], x[1]: 4B*2 = 8B

x[0][0], x[0][1], x[0][2]: 4B*3 = 12B

x[1][0], x[1][1], x[1][2], x[1][3]: 4B*4 = 16B

x[0][0], x[0][1], x[0][2]: 4B*(2+3+2) = 28B

y[1][0], x[1][3]: 4B*(2+5) = 28B

Total: 96 Bytes

Problem #7 (C, 20 points)

All the registers R# are 32-bit registers and everything is based on the 32-bit ARM architecture. The following map shows a part of the main memory. The data type of variable “x” is int**. “x” is declared by

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

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

for given constants “a” and “b”. 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 *x? 0x400C

(b) What is the value of &x? 0x8000

(c) What is the value of x+2? 0x4008

(d) What is the value of *(x+1)? 0x4014

(e) What is the value of **x?

0x4010

(f) What is the value of *((*x) + ((int*) 0x10))?

*(0x400C + 0x0010) = *(0x401C) = 0x4020

(g) What is the value of **(x+2)?

*(*0x4008) = *(0x4024) = 0x4028

(h) What is the value of x[0][0]?

0x4010

(i) What is the value of x[0][1]?

0x4014

(j) What is the value of x[1][1]?

0x401C

Problem #8 (C, 20 points)

All the registers R# are 32-bit registers and everything is based on the 32-bit ARM architecture. “int” is a 32-bit data type.

```
struct MyData {
  int x[4];
  int y[2];
};

MyData src[3][3];  // given
MyData** des = new MyData*[3];

for ( int i = 0 ; i < 3 ; i++ )
  des[i] = new MyData[3];

for ( int a = 0 ; a < 3 ; a++ ) {
  for ( int b = 0 ; b < 3 ; b++ ) {
    des[a][b].x[1] = src[a][b].x[1];
  }
}
```

Write an assembly code for the above C code.

- &(src[0][0].x[0]): 0x8000
- The value stored in “des”: 0x4000

```
// Size of MyData: 24 bytes (6 * 4 bytes)
LDR R1, =#0x8000  // src
LDR R2, =#0x4000  // des
MOV R3, #0  // a

loop_a:
  MOV R4, #0  // b
loop_b:
  MUL R5, R3, #72  // src[a]
  ADD R5, R5, R1
  MUL R6, R4, #24
  ADD R5, R5, R6  // src[a][b]
  ADD R5, R5, #4  // src[a][b].x[1]

  MUL R7, R3, #4  // 4*a
  ADD R6, R2, R7  // des + 4*a
  LDR R7, [R6]  // des[a]
  MUL R6, R4, #24
  ADD R7, R7, R6  // des[a][b]
  ADD R7, R7, #4  // des[a][b].x[1]

  LDR R8, [R5]
  STR R8, [R7]

  ADD R4, R4, #1  // b++
  CMP R4, #3
  BNE loop_b

  ADD R3, R3, #1  // a++
  CMP R3, #3
  BNE loop_a
```

Problem #9 (Interrupts, 10 points)

An ARM C source handles keyboard inputs using interrupts and an interrupt handler function H. Whenever a key input is received, the system generates an interrupt and H is called to process the input. The runtime of executing H for a key input is 1,000 clock cycles. The system clock frequency is 100MHz (period: 10ns). If two key inputs k_2 and k_3 are received while H is being executed to process a key input k_1, the CPU stores k_2 and k_3 in its input buffer (queue). When H finishes processing k_1, H is immediately executed again to process k_2, then executed again to process k_3. The size of the keyboard buffer is 20, i.e., it can store maximum 20 key inputs while H is being executed. If the buffer is full, any additional key inputs are ignored (discarded).

Suppose you press 100 keys periodically (i.e., you press a key at time 0, a key at time T, a key at time $2T$, ..., a key at time $99T$). Calculate the minimum T that does not cause discarded keystrokes for the 100 keyboard inputs (this is finding the maximum keystroke speed).

When the 100^{th} key is pressed, H must be processing the 80^{th} key so that the buffer has 19 inputs and the 100^{th} input is inserted into the buffer. Processing a key takes 1000*10ns = 10μs. Processing 79 keys takes 790us. This must be smaller than the time when the 100^{th} key is pressed. The 100^{th} key is pressed at time $99T$.

$$99T > 790\mu s$$

Thus, the minimum value of T that does not discard any keys is approximately 7.98μs.