EE234

Microprocessor Systems

Midterm Exam 1

Oct. 13, 2023. (2:10pm – 3pm)

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

Name:

WSU ID:

Problem	Points	
1	10	
2	10	
3	20	
4	30	
5	30	
Total	100	

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. R1 has an input data. The following two instructions perform an arithmetic operation. Explain what it does (i.e., briefly explain the meaning of the data stored in R2 in terms of arithmetic operations) or draw a graph of (R1 vs. R2). Here, "arithmetic" means something like addition, subtraction, multiplication, division (quotient), division (remainder), square root, transcendental functions, etc. Ignore overflow/underflow exceptions in the operations.

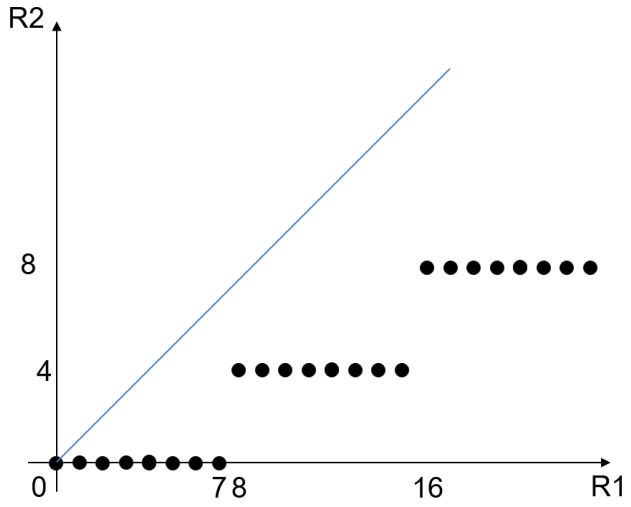
MOV R2, R1, LSR #3

MOV R2, R2, LSL #2

$$R_1 = x_7x_6x_5x_4x_3x_2x_1x_0$$
$$R_2 = 0x_7x_6x_5x_4x_300$$

If we divide R1 by 2, then we obtain $0x_7x_6x_5x_4x_3x_2x_1$. Then, then two LSBs are cleared.

$$R_2 = \left\lfloor \frac{\left\lfloor \frac{R_1}{2} \right\rfloor}{4} \right\rfloor \times 4 = \left\lfloor \frac{R_1}{8} \right\rfloor \times 4$$



Problem #2 (Bit manipulation, 10 points)

Suppose R# is an 8-bit register. R_1 and R_2 are given as follows:

$$R_1 = x_7x_6x_5x_4\ x_3x_2x_1x_0$$

$$R_2 = y_7y_6y_5y_4\ y_3y_2y_1y_0$$

Write an assembly code to generate R_3 from R_1 and R_2. You can use the following instructions only. (&: logical AND. |: logical OR, ^: logical XOR)

- AND, ORR, EOR, MOV (including LSL, LSR)

$$R_3 = \overline{x_7}\ y_6\ \overline{y_5}\ x_4\ x_3\ \overline{x_2}\ y_1\ \overline{y_0}$$

EOR R3, R1, #0b1000 0100

AND R3, R3, #0b 1001 1100

EOR R4, R2, #0b 0010 0001

AND R4, R4, #0b 0110 0011

ORR R3, R3, R4

Problem #3 (ARM assembly, 20 points)

```
main:
  MOV R0, #0
  MOV R1, #10
  MOV R2, #20
loop:
  MOV R3, R1, LSR #2
  MOV R4, R2, LSR #1
  ADD R5, R3, R4
  MOV R6, R5, LSL #1
  CMP R0, #3
  BGE end
  MOV R1, R2
  MOV R2, R6
  ADD R0, R0, #1
  B loop
end:
```

What is the value of the data stored in R6 when the following program ends?

R0	R1	R2	R3	R4	R5	R6
0	10	20				
			2	10	12	24
1	20	24				
			5	12	17	34
2	24	34				
			6	17	23	46
3	34	46				
			8	23	31	62

R6 = 62

Problem #4 (ARM assembly, 30 points)

What is the value of the data stored in R0 when the program ends?

```
main:
  MOV R0, #0
  MOV R1, #0
loop1:
  CMP R1, #100
  BGE loop1_end
  ADD R2, R1, #1
loop2:
  CMP R2, #100
  BGE loop2_end
  AND R3, R1, #3
  CMP R3, #0
  BNE loop2_term
  AND R4, R2, #7
  CMP R4, #0
  BNE loop2_term
  ADD R0, R0, #1
loop2_term:
  ADD R2, R2, #1
  B loop2
loop2_end:
  ADD R1, R1, #1
  B loop1
loop1_end:
```

(Hint: Translate the code into a C code, and then analyze it.)

Let R0=x, R1=a, R2=b, R3=c, R4=d. Then the code becomes

x=0; a=0;

If (a >= 100), then go to the end of the code.

Otherwise (i.e., a < 100), b = a + 1. Then

If (b >= 100) , then go to loop2_end. In this case, a++; then go to the first if.

Otherwise (i.e., b < 100), c = a & 3. If (c != 0), then b++ and go to the second if.

If (c == 0), then d = b & 7. If (d != 0), then b++ and go to the second if.

Otherwise (i.e., c == 0 and d == 0), x++ and then b++ and go to the second if. Thus, the code is like this:

```
int x = 0;

for ( int a = 0 ; a < 100 ; a++ ) {
  for ( int b = a+1 ; b < 100 ; b++ ) {
    int c = a & 3;
    int d = b & 7;
    if ( (c == 0) && (d == 0) )
      x++;
  }
}
```

a&3==0 when a is an integer multiple of 4.

b&7==0 when b is an integer multiple of 8.

Whenever this occurs, x is increased by 1.

Integer multiples of 4 (for a) in the range of 0, 1, …, 99 => 0, 4, 8, 12, …, 96. 25 cases.

a=0: integer multiples of 8 (for b) in the range of 1, …, 99 => 12 cases.

a=4: integer multiples of 8 (for b) in the range of 5, …, 99 => 12 cases.

a=8: integer multiples of 8 (for b) in the range of 9, …, 99 => 11 cases.

…

a=80: integer multiples of 8 (for b) in the range of 81, …, 99 => 2 cases.

a=84: integer multiples of 8 (for b) in the range of 85, …, 99 => 2 cases.

a=88: integer multiples of 8 (for b) in the range of 89, …, 99 => 1 case.

a=92: integer multiples of 8 (for b) in the range of 93, …, 99 => 1 case.

a=96: integer multiples of 8 (for b) in the range of 97, …, 99 => no case.

Thus, x = 12*2 + 11*2 + … + 1*2 = 78*2=156

Problem #5 (ARM assembly, 30 points)

Translate the following C code into an assembly code.

```
int a, b, c, d;
…
while ( ((a + b) == 10) || ((c – d) == 8) ) {
  a++;

  if ( (c == 20) && (d == 17) )
    break;
  else if ( a == 6 )
    continue;
  else
    b++;
}
```

- Use the assembly instructions listed in the last page only.
- a is in R0, b is in R1, c is in R2, and d is in R3.
- The exit point (the end of the if statement) could be just an address label.

```
loop:
  ADD R4, R0, R1
  CMP R4, #10
  BEQ main_body
  SUB R4, R2, R3
  CMP R4, #8
  BEQ main_body
  B while_end
main_body:
  ADD R0, R0, #1
  CMP R2, #20
  BNE else_if
  CMP R3, #17
  BNE else_if
  B while_end
else_if:
  CMP R0, #6
  BEQ loop
  ADD R1, R1, #1
  B loop
while_end:
```

Assembly Instructions

R# is a register. (# = 0 ~ 12)

<table>
<tr><th>Instruction</th><th>Meaning</th></tr>
<tr><td>MVN Rd, Ra</td><td>Bitwise inversion. (Rd = Bitwise-NOT Ra)
<table><tr><td>Before</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr><tr><td>After</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table></td></tr>
<tr><td>AND Rd, Ra, Rb
AND Rd, Ra, #imm</td><td>Bitwise AND. (Rd = Ra AND Rb), (Rd = Ra AND #imm)
<table><tr><td>Ra</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>Rb</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>Rd</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr></table></td></tr>
<tr><td>ORR Rd, Ra, Rb
ORR Rd, Ra, #imm</td><td>Bitwise OR. (Rd = Ra OR Rb), (Rd = Ra OR #imm)
<table><tr><td>Ra</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr><tr><td>Rb</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>Rd</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr></table></td></tr>
<tr><td>EOR Rd, Ra, Rb
EOR Rd, Ra, #imm</td><td>Bitwise exclusive-OR. (Rd = Ra ⊕ Rb), (Rd = Ra ⊕ #imm)
<table><tr><td>Ra</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>Rb</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>Rd</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr></table></td></tr>
<tr><td>MOV Rd, Ra, LSR #imm</td><td>Logical shift right by (#imm) bits. (Rd = Ra >> #imm)
Ex) #imm = 3
<table><tr><td>Before</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr><tr><td>After</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table></td></tr>
<tr><td>MOV Rd, Ra, LSL #imm</td><td>Logical shift left by (#imm) bits. (Rd = Ra << #imm)
Ex) #imm = 3
<table><tr><td>Before</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr><tr><td>After</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr></table></td></tr>
<tr><td>MOV Rd, Ra
MOV Rd, #imm</td><td>(Rd = Ra)
(Rd = #imm)</td></tr>
<tr><td>ADD Rd, Ra, Rb
ADD Rd, Ra, #imm</td><td>(Rd = Ra + Rb)
(Rd = Ra + #imm)</td></tr>
<tr><td>SUB Rd, Ra, Rb
SUB Rd, Ra, #imm</td><td>(Rd = Ra - Rb)
(Rd = Ra - #imm)</td></tr>
<tr><td>CMP Rd, #imm
CMP Rd, Ra</td><td>Set Z = 1 if Rd == #imm. Otherwise, Z = 0. (Z is the Zero field of the CPSR.)
Set Z = 1 if Rd == Ra. Otherwise, Z = 0.
Notice that N != V is Rd < #imm or Rd < Ra.</td></tr>
<tr><td>B [addr]</td><td>Jump to [addr] unconditionally</td></tr>
<tr><td>BEQ, BNE, BLT, BGT, BGE, BLE [addr]</td><td>Branch to [addr] if (BEQ: R1 == R2), (BNE: R1 != R2), (BLT: R1 < R2), (BGT: R1 > R2), (BGE: R1 >= R2), (BLE: R1 <= R2)</td></tr>
<tr><td>LDR Rd, [Ra, #imm]</td><td>Load the data stored at [Ra + #imm] to Rd.</td></tr>
<tr><td>STR Rd, [Ra, #imm]</td><td>Store the data stored in Rd to [Ra + #imm].</td></tr>
</table>