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

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)

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

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?

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.)

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.

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>