EE234

Microprocessor Systems

Midterm Exam

Oct. 18, 2024. (2:10pm – 3pm)

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

Name:

WSU ID:

Problem	Points	
1	10	
2	20	
3	20	
4	20	
5	20	
Total	90	

Problem #1 (Bit manipulation, 10 points)

R# is an 8-bit register. The data stored in R# is treated as an unsigned binary number. R1 has an input data x in the range of $0 \leq x \leq 63$. 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.

```
AND R2, R1, #0xFC

MOV R2, R2, LSL #2
```

Problem #2 (ARM assembly, 20 points)

```
main:
    MOV R6, #0
    MOV R1, #1
loop1:
    CMP R1, #6
    BGT loop1_end
    ADD R2, R1, #1
loop2:
    CMP R2, #6
    BGT loop2_end
    MOV R3, #0
    MOV R4, #1
loop3:
    CMP R4, R2
    BGT loop3_end
    ADD R3, R3, R1
    ADD R4, R4, #1
    B loop3
loop3_end:
    ADD R6, R6, R3
    ADD R2, R2, #1
    B loop2
loop2_end:
    ADD R1, R1, #1
    B loop1
loop1_end:
```

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

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

Problem #3 (ARM assembly, 20 points)

Translate the following C code into an assembly code.

```
int a, b, c, d;
…

while ( (a != 10) && (b < 5) && (c > 7) ) {
   b++;

   while ( (d > 6) || (d < 12) ) {
      c++;
      d--;
   }

   if ( a <= b ) {
      break;
   }
}
```

- 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.

Problem #4 (ARM assembly, 20 points)

```
main:                   // Addresses
  MOV R0, #0            // 0x00
  MOV R1, #1            // 0x04
label1:
  CMP R1, #11           // 0x08
  BGE end               // 0x0C
  BL sub1               // 0x10
  BL sub2               // 0x14
  ADD R1, R1, #1        // 0x18
  B label1              // 0x1C
sub1:
  MOV R2, #0            // 0x20
  MOV R3, #0            // 0x24
label2:
  CMP R2, R1            // 0x28
  BGE sub1_end          // 0x2C
  ADD R3, R3, R1        // 0x30
  ADD R2, R2, #1        // 0x34
  B label2              // 0x38
sub1_end:
  BX LR                 // 0x3C
sub2:
  ADD R0, R0, R3        // 0x40
  BX LR                 // 0x44
end:
  …                     // 0x48
```

The address column shows the addresses of the instructions.

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

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

2. What is the value of R14 (LR) when the above program ends?

Problem #5 (ARM assembly, 20 points)

Write an assembly code to compute x^4. Assume x is an unsigned integer and stored in R0. Store x^4 in R1. Use the assembly instructions listed in the last page only.

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>