EE234

Microprocessor Systems

Midterm Exam 1

Oct. 8, 2021. (2:10pm – 3pm)

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

Name:

WSU ID:

Problem	Points	
1	10	
2	10	
3	20	
4	20	
5	30	
6	30	
Total	120	

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.

AND R2, R1, #0xFD

ORR R2, R2, #0x01

Input: $x_7x_6x_5x_4x_3x_2x_1x_0$

Output: $x_7x_6x_5x_4x_3x_2 01$

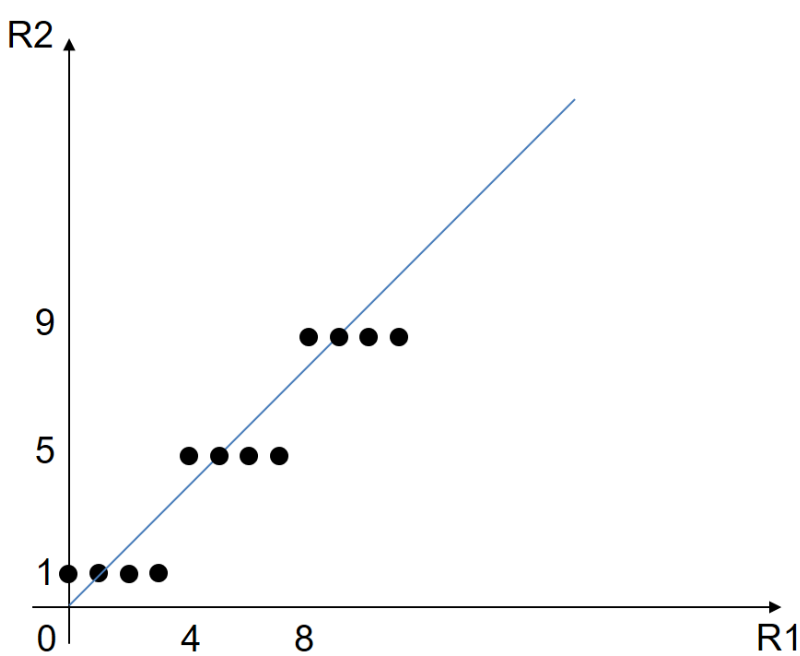
0, 1, 2, 3 → 1

4, 5, 6, 7 → 5

8, 9, 10, 11 → 9

Answer: Suppose X and Y are the values in R1 and R2, respectively. Then,

$$Y = 4 \cdot \left\lfloor \frac{X}{4} \right\rfloor + 1$$



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. R1 has an input data. The following instruction performs 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.

AND R2, R1, #0xBF

Input: $x_7x_6x_5x_4x_3x_2x_1x_0$

Output: $x_70x_5x_4x_3x_2x_1x_0$

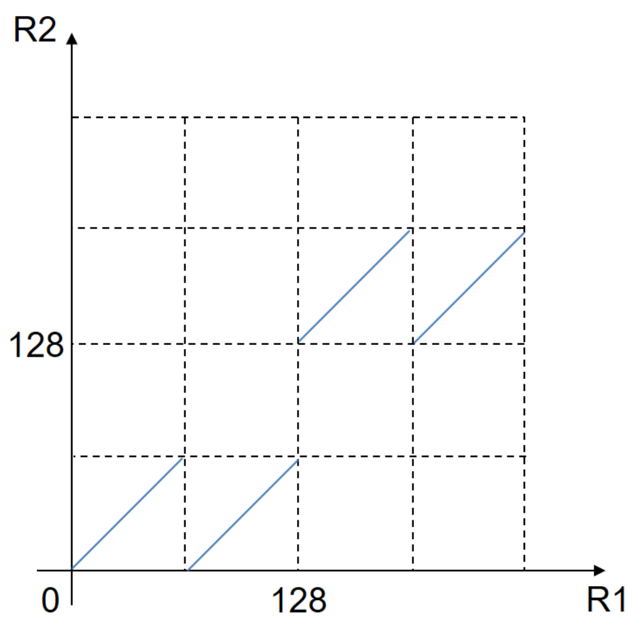
Answer: Suppose X and Y are the values in R1 and R2, respectively.

If $X < 64$ (i.e., $x_{7:6} = 00$), $Y = X$.

If $64 \leq X < 128$ (i.e., $x_{7:6} = 01$), $Y = X - 64$.

If $128 \leq X < 192$ (i.e., $x_{7:6} = 10$), $Y = X$.

If $128 \leq X$ (i.e., $x_{7:6} = 11$), $Y = X - 64$.



Problem #3 (ARM assembly, 20 points)

What is the value of the data stored in R1 when the following program ends?

```
  MOV R1, #0
  MOV R2, #0
loop1:
  CMP R2, #200
  BGE end
  AND R3, R2, #0x07
  CMP R3, #2
  BNE loop1_end
  ADD R1, R1, #1
loop1_end:
  ADD R2, R2, #1
  B loop1
end:
  // end of code
```

R1: 0

R2: 0

if (R2 < 200)

R3 = R2 & #0x07 = $0\ 0\ 0\ 0\ 0\ x_2\ x_1\ x_0$ (for R2 = $x_7\ x_6\ \ldots\ x_0$)

if (R3 == 2)

R1++;

R2++;

go back to the first if statement.

Thus, whenever R2 is $XXXX\ X010$, it increases R1 by 1. It repeats it as long as R2<200.

How many times? When R2 is 2, 10, 18, 26, ..., 194.

Thus, R1 will have **25**.

Problem #4 (ARM assembly, 20 points)

What is the value of the data stored in R3 when the program ends?

```
  MOV R3, #0
  MOV R1, #0
loop1:
  CMP R1, #5
  BGE loop1_end
  MOV R2, #0
loop2:
  CMP R2, #5
  BGE loop2_end
  AND R4, R1, R2
  ADD R3, R3, R4
  ADD R2, R2, #1
  B loop2
loop2_end:
  ADD R1, R1, #1
  B loop1
loop1_end:
// end of code
```

R1 = 0, R3 = 0

loop1 (R1 = 0 < 5)

  R2 = 0

  loop2 (R2 = 0 < 5)

    R4 = R1 & R2 = 0, R3 += R4, so R3 = 0, R2 = 1

  loop2 (R2 = 1 < 5)

    R4 = R1 & R2 = 0, R3 += R4, so R3 = 0, R2 = 2

  ...

  R1 += 1, so R1 = 1

loop1 (R1 = 1 < 5)

  ...

so the outer loop iterates five times for R1 = 0, 1, 2, 3, 4, and the inner loop iterates five times for R2 = 0, 1, 2, 3, 4 for each R1.

R3 is the sum of R1 & R2. Then,

R1 = 0: R1&R2 = 0 for all R2 = 0 – 4.

R1 = 1: R1&R2 = 1 for R2 = 1, 3

R1 = 2: R1&R2 = 2 for R2 = 2, 3

R1 = 3: R1&R2 = 1 for R2 = 1, 2 for R2 = 2, 3 for R2 = 3

R1 = 4: R1&R2 = 4 for R2 = 4.

R3 = 1 + 1 + 2 + 2 + 1 + 2 + 3 + 4 = 16.

Thus, R3 has **16**.

Problem #5 (ARM assembly, 30 points)

Make an assembly code for the following C code.

```
int a, b, c;  // a in R0, b in R1, c in R2

if ( (a == 0) && (b == 3) ) {
  c++;
}
else if ( (b == 2) || (c == 4) ) {
  a++;
}
else if ( (a == 5) && (c != 6) ) {
  b++;
}
else {
  a--;
}
```

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

```
    CMP R0, #0
    BNE else1
    CMP R1, #3
    BNE else1
    ADD R2, R2, #1
    B end_if
else1:
    CMP R1, #2
    BEQ else2_run
    CMP R2, #4
    BEQ else2_run
    CMP R0, #5
    BNE else3_run
```

```
    CMP R2, #6
    BEQ else3_run
    ADD R1, R1, #1
    B end_if
else2_run:
    ADD R0, R0, #1
    B end_if
else3_run:
    SUB R0, R0, #1
end_if:
```

Problem #6 (ARM assembly, 30 points)

Let's use the 32-bit ARM architecture, i.e., R# is a 32-bit register and the register file has 16 registers (you can use R0~R12 only). R0 has a positive number (given to you). We want to check whether the number in R0 is a square number (i.e., n^2) or not. If it is, we set R1 to 1. If not, we set R1 to 0. Here is an algorithm for that.

1) If R0 is 0, R1 = 0. Done.

2) If R0 is 1, R1 = 1. Done.

3) If R0≥ 2, try to compare 1^2 with R0, 2^2 with R0, ..., n^2 with R0. If n^2==R0, R1 = 1. Done. If $(n-1)^2$<R0 and n^2>R0, R1 = 0. Done.

Simply speaking, suppose 35 is given (in R0). Then, $1^2 = 1 < 35, 2^2 = 4 < 35, 3^2 = 9 < 35, 4^2 = 16 < 35, 5^2 = 25 < 35, 6^2 = 36 > 35$, so we know that 35 is not a square number.

Write an assembly code running the above algorithm. Use only the instructions shown in the instruction sheet. Assume that R0 has a given number. The performance of the code doesn't matter as long as the code works. You can't use multiply instructions, so you should use ADD to compute n^2.

```
  MOV R1, #0  // init R1 = 0
  CMP R0, #0
  BEQ end
  CMP R0, #1
  BEQ set_one

  // compute n^2
  MOV R2, #2
for:  // compute R4 = R2^2
  B square
for_back:  // R4 has R2^2
  CMP R4, R0
  BEQ set_one
  BGT end
  ADD R2, R2, #1
  B for
square:  // compute n^2 (n is in R2)
  MOV R3, #0
  MOV R4, #0  // sum
square_loop:
  CMP R3, R2  // if (R3 < R2)
  BGE for_back
  ADD R4, R4, R2  // sum += R2
  ADD R3, R3, #1  // R3++
  B square_loop
set_one:
  MOV R1, #1
end:
```