CENG 3420
Computer Organization & Design

HW2 Review

ZHAO Wenqian
CSE Department, CUHK
wqzhao@cse.cuhk.edu.hk

Spring 2022

Overview

Overview

Q1 Description

1. What is 5ED4 - 07A4 when these values represent unsigned 16-bit hexadecimal numbers? The result should be written in hexadecimal. Show your work.
2. What is 5ED4 - 07A4 when these values represent signed 16-bit hexadecimal numbers stored in sign-magnitude format? The result should be written in hexadecimal. Show your work.

Solution

1. 5730
2. 5730, if you work with signed numbers, they are represented by the first (most significant) bit being one. That is to say that if you work with a number of bits that is a multiple of four, then a number is negative if the first hexadecimal digit is 8,9,A,B,C,D,E or F.

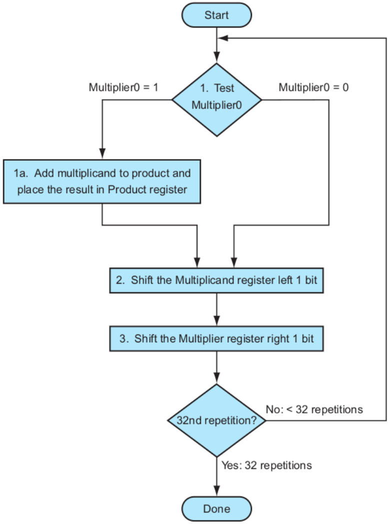
Overview

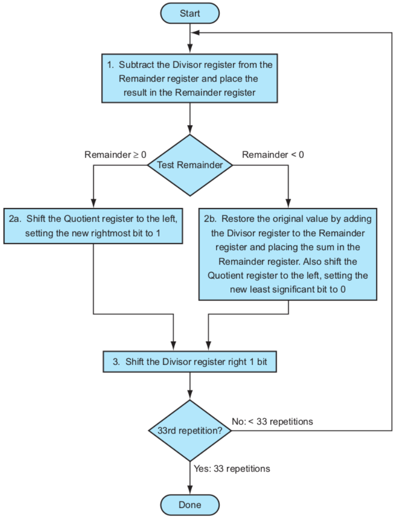
Q2 Description

Read through the multiplication/devision algorithm:

1. Write down the step by step procedure to calculate 7×3 or 0111×0011. Use Multiplier0 to indicate the least significant bit of the multiplier.
2. Write down the step by step procedure to calculate $7 \div 2$ or $0111 \div 0010$.

Q2 Description


Start
1. Test Multiplier0
Multiplier0 = 1
Multiplier0 = 0
1a. Add multiplicand to product and place the result in Product register
2. Shift the Multiplicand register left 1 bit
3. Shift the Multiplier register right 1 bit
32nd repetition?
No: < 32 repetitions
Yes: 32 repetitions
Done


Start
1. Subtract the Divisor register from the Remainder register and place the result in the Remainder register
Test Remainder
Remainder ≥ 0
Remainder < 0
2a. Shift the Quotient register to the left, setting the new rightmost bit to 1
2b. Restore the original value by adding the Divisor register to the Remainder register and placing the sum in the Remainder register. Also shift the Quotient register to the left, setting the new least significant bit to 0
3. Shift the Divisor register right 1 bit
33rd repetition?
No: < 33 repetitions
Yes: 33 repetitions
Done

The left figure is the multiplication algorithm for reference. The right figure is the division algorithm.

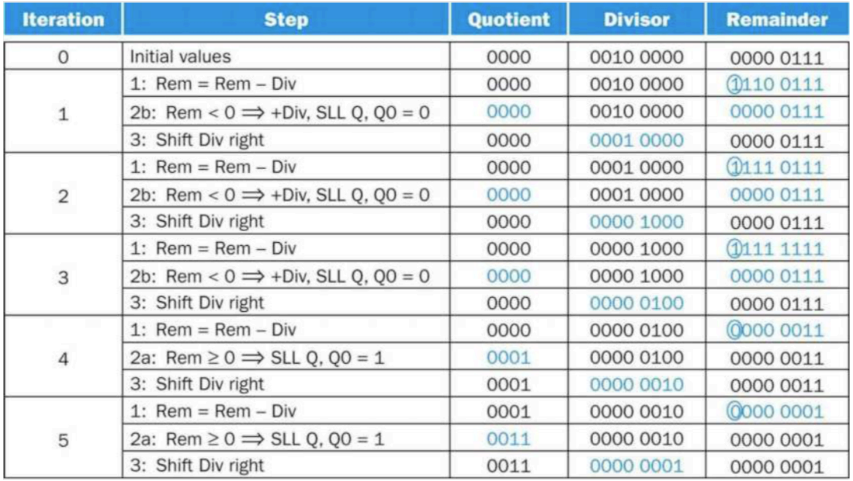
Solution 2.1

Steps for multiplication

Iteration	Step	Multiplier	Multiplier0	Multiplicand	Product
0	Initial values	0011	1	0000 0111	0000 0000
1	1a: 1⇒Prod=Prod+Mcand	0011	1	0000 0111	0000 0111
	2: Shift left Multiplicand	0011	1	0000 1110	0000 0111
	3: Shift right Multiplier	0001	1	0000 1110	0000 0111
2	1a: 1⇒Prod=Prod+Mcand	0001	1	0000 1110	0001 0101
	2: Shift left Multiplicand	0001	1	0001 1100	0001 0101
	3: Shift right Multiplier	0000	0	0001 1100	0001 0101
3	1a: 1⇒Prod=Prod+Mcand	0000	0	0001 1100	0001 0101
	2: Shift left Multiplicand	0000	0	0011 1000	0001 0101
	3: Shift right Multiplier	0000	0	0011 1000	0001 0101
4	1a: 1⇒Prod=Prod+Mcand	0000	0	0011 1000	0001 0101
	2: Shift left Multiplicand	0000	0	0111 0000	0001 0101
	3: Shift right Multiplier	0000	0	0111 0000	0001 0101

Solution 2.2

Steps for division



Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	①110 0111
	2b: Rem < 0 $\Longrightarrow$ +Div, SLL Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	①111 0111
	2b: Rem < 0 $\Longrightarrow$ +Div, SLL Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	①111 1111
	2b: Rem < 0 $\Longrightarrow$ +Div, SLL Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	⓪000 0011
	2a: Rem ≥ 0 $\Longrightarrow$ SLL Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	⓪000 0001
	2a: Rem ≥ 0 $\Longrightarrow$ SLL Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

Overview

Q3 Description

EEE 754-2008 contains a half precision that is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and has a bias of 15, and the mantissa is 10 bits long. A hidden 1 is assumed. Write down the bit pattern to represent $-1.5625 \times 10 - 1$ assuming a version of this format, which uses an excess-16 format to store the exponent.

Solution of Q3

1. $-1.5625 \times 10^{-1} = -0.15625 \times 10^{0}$
2. $= -0.00101 \times 2^{0}$
3. move the binary point three to the right, $= -1.01 \times 2^{-3}$
4. exponent $= -3 = -3 + 15 = 12$, fraction -0.0100000000
5. answer: 1011000100000000

Overview

Q4 Description

Calculate the sum of 2.6125×10^{1} and $4.150390625 \times 10^{-1}$ by hand, assuming A and B are stored in the 16-bit half precision described in Q3. Assume 1 guard, 1 round bit, and 1 sticky bit, and round to the nearest even. Show all the steps. (To perform addition of 2 number, firstly shift mantissa right to make sure the exponent are matched)

Solution of Q4

```
mantissa format plus extra bits:

    1.XXXXXXXXXX  0   0   0

    ^       ^          ^   ^   ^
    |       |          |   |   |
    |       |          |   |   -   sticky bit (s)
    |       |          |   -   round bit (r)
    |       |          -   guard bit (g)
    |       -   10 bit mantissa from a representation
    -   hidden bit
```

Solution of Q4

```
     Example:
mantissa from representation,   1100000100
must be shifted by 8 places (to align radix points)
                                          g r s
Before first shift:   1.1100000100  0 0 0
After 1 shift:        0.1110000010  0 0 0
After 2 shifts:       0.0111000001  0 0 0
After 3 shifts:       0.0011100000  1 0 0
After 4 shifts:       0.0001110000  0 1 0
After 5 shifts:       0.0000111000  0 0 1
After 6 shifts:       0.0000011100  0 0 1
After 7 shifts:       0.0000001110  0 0 1
After 8 shifts:       0.0000000111  0 0 1
```

Solution of Q4

1. $2.6125 \times 10^1 + 4.150390625 \times 10^{-1}$
2. $2.6125 \times 10^1 = 26.125 = 11010.001 = 1.1010001000 \times 2^4$
3. $4.150390625 \times 10^{-1} = 0.4150390625 = 0.011010100111 = 1.1010100111 \times 2^{-2}$
4. Shift binary point six to the left to align exponents,

```
             GR
1.1010001000 00
1.0000011010 10 0111 (Guard 5 1, Round 5 0,
Sticky 5 1)
-------------------
1.1010100010 10
```

5.

6. In this case the extra bit (G,R,S) is more than half of the least signifi cant bit (0).
7. Thus, the value is rounded up.
8. $1.1010100011 \times 2^4 = 11010.100011 \times 2^0 = 26.546875 = 2.6546875 \times 10^1$

Overview

Q5 Description

```
a = b + e;
c = b + f;
```

Here is the generated RISC-V code for this segment, assuming all variables are in memory and are addressable as offsets from x31:

```
ld x1, 0(x31) // Load b
ld x2, 8(x31) // Load e
add x3,x1,x2 // b + e
sd x3, 24(x31) // Store a
ld x4, 16(x31) // Load f
add x5,x1,x4 // b + f
sd x5, 32(x31) // Store c
```

Find the hazards in the preceding code segment and reorder the instructions to avoid any pipeline stalls.

Solution of Q5

Both add instructions have a hazard because of their respective dependence on the previous ld instruction. Notice that forwarding eliminates several other potential hazards, including the dependence of the first add on the first ld and any hazards for store instructions. Moving up the third ld instruction to become the third instruction eliminates both hazards:

1. ld x1, 0(x31) // Load b
2. ld x2, 8(x31) // Load e
3. ld x4, 16(x31) // Load f
4. add x3,x1,x2 // b + e
5. sd x3, 24(x31) // Store a
6. add x5,x1,x4 // b + f
7. sd x5, 32(x31) // Store c

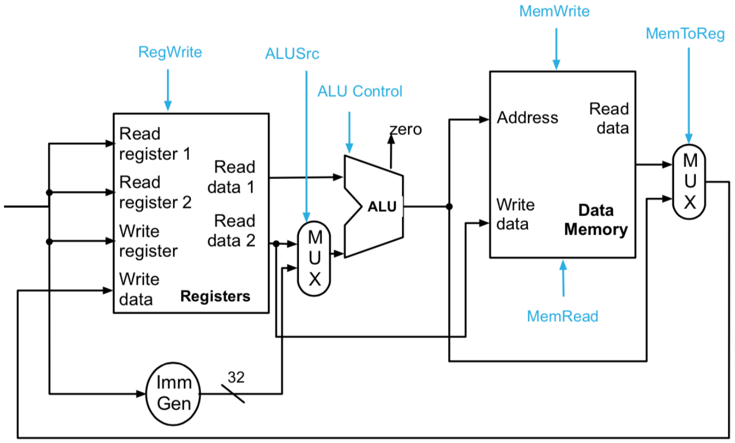
Overview

Q6 Description

Consider the following instruction:
Instruction: `and rd, rs1, rs2`
Interpretation: `Reg[rd] = Reg[rs1] AND Reg[rs2]`

1. What are the values of control signals generated by the control in figure 2 for this instruction?
2. Which resources (blocks) produce no output for this instruction? Which resources produce output that is not used?

Q6 Description



The datapath for the memory instructions and the R-type instructions.

Solution of Q6

1. Mathematically, the MemRead control wire is a “don’t care": the instruction will run correctly regardless of the chosen value. Practically, however, MemRead should be set to false to prevent causing a segment fault or cache miss.

RegWrite	ALUSrc	ALUoperation	MemWrite	MemRead	MemToReg
true	0	"and"	false	false	0

2. Data memory; MemToReg mux.