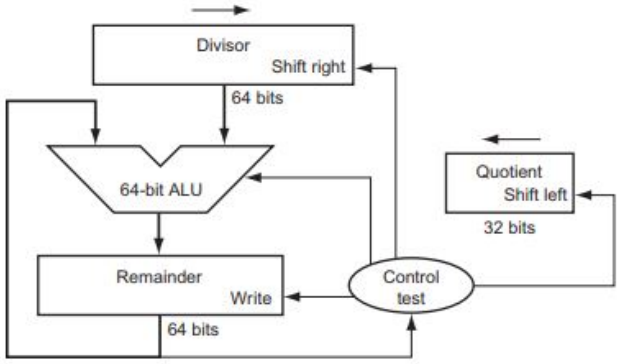
National Chung Cheng University
Department of Electrical Engineering
Computer Organization and Design, Spring 2018
TAs: Jun-Jie Zhao , Tzu-Wei Wang , Wei-Da Wang , Wei-Liang Wu , Yan-Ting Guo , Zhang-Yao Liou

May 17, 2018
Instructor: Mark P.-H. Lin

Quiz #2

1. Calculate 74 divided by 21 using the hardware described bellow. You should show the contents of each register on each step. Assume both inputs are unsigned 6-bit integers. (15 points)



sol.
OCT.

		Quotient	Divisor	Remainder
0	Initial	000 000	010 001 000 000	000 000 111 100
1	Rem=Rem-Div	000 000	010 001 000 000	101 111 111 100
	Rem<0,+Div,sll Q,Q0=0	000 000	010 001 000 000	000 000 111 100
	Shift div right	000 000	001 000 100 000	000 000 111 100
2	Rem=Rem-Div	000 000	001 000 100 000	111 000 011 100
	Rem<0,+Div,sll Q,Q0=0	000 000	001 000 100 000	000 000 111 100
	Shift div right	000 000	000 100 010 000	000 000 111 100
3	Rem=Rem-Div	000 000	000 100 010 000	111 100 101 100
	Rem<0,+Div,sll Q,Q0=0	000 000	000 100 010 000	000 000 111 100
	Shift div right	000 000	000 010 001 000	000 000 111 100
4	Rem=Rem-Div	000 000	000 010 001 000	111 110 110 100
	Rem<0,+Div,sll Q,Q0=0	000 000	000 010 001 000	000 000 111 100
	Shift div right	000 000	000 001 000 100	000 000 111 100
5	Rem=Rem-Div	000 000	000 001 000 100	111 111 111 000
	Rem<0,+Div,sll Q,Q0=0	000 000	000 001 000 100	000 000 111 100
	Shift div right	000 000	000 000 100 010	000 000 111 100
6	Rem=Rem-Div	000 000	000 000 100 010	000 000 011 010
	Rem>0,sll Q,Q0=1	000 001	000 000 100 010	000 000 011 010
	Shift div right	000 001	000 000 010 001	000 000 011 010
7	Rem=Rem-Div	000 001	000 000 010 001	000 000 001 001
	Rem>0,sll Q,Q0=1	000 011	000 000 010 001	000 000 001 001
	Shift div right	000 011	000 000 001 000	000 000 001 001

DEC.

Table 1: Add caption

		Quotient	Divisor	Remainder
0	Initial	0000 0000	00010101 00000000	00000000 01001010
	Rem=RemDiv	0000 0000	00010101 00000000	11101011 01001010
1	Rem¡0,+Div,sll Q,Q0=0	0000 0000	00010101 00000000	00000000 01001010
	Shift div right	0000 0000	00001010 10000000	00000000 01001010
	Rem=RemDiv	0000 0000	00001010 10000000	11110101 11001010
2	Rem¡0,+Div,sll Q,Q0=0	0000 0000	00001010 10000000	00000000 01001010
	Shift div right	0000 0000	00000101 01000000	00000000 01001010
	Rem=RemDiv	0000 0000	00000101 01000000	11111011 00001010
3	Rem¡0,+Div,sll Q,Q0=0	0000 0000	00000101 01000000	00000000 01001010
	Shift div right	0000 0000	00000010 10100000	00000000 01001010
	Rem=RemDiv	0000 0000	00000010 10100000	11111101 10101010
4	Rem¡0,+Div,sll Q,Q0=0	0000 0000	00000010 10100000	00000000 01001010
	Shift div right	0000 0000	00000001 01010000	00000000 01001010
	Rem=RemDiv	0000 0000	00000001 01010000	11111110 11111010
5	Rem¡0,+Div,sll Q,Q0=0	0000 0000	00000001 01010000	00000000 01001010
	Shift div right	0000 0000	00000000 10101000	00000000 01001010
	Rem=RemDiv	0000 0000	00000000 10101000	11111111 10100010
6	Rem¡0,+Div,sll Q,Q0=0	0000 0000	00000000 10101000	00000000 01001010
	Shift div right	0000 0000	00000000 01010100	00000000 01001010
	Rem=RemDiv	0000 0000	00000000 01010100	11111111 11110110
7	Rem¡0,+Div,sll Q,Q0=0	0000 0000	00000000 01010100	00000000 01001010
	Shift div right	0000 0000	00000000 00101010	00000000 01001010
	Rem=RemDiv	0000 0000	00000000 00101010	00000000 00100000
8	Rem¿0,sll Q,Q0=1	0000 0001	00000000 00101010	00000000 00100000
	Shift div right	0000 0001	00000000 00010101	00000000 00100000
	Rem=RemDiv	0000 0001	00000000 00010101	00000000 00001011
9	Rem¿0,sll Q,Q0=1	0000 0011	00000000 00010101	00000000 00001011
	Shift div right	0000 0011	00000000 00001010	00000000 00001011

2. IEEE 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. Write down the binary representation of the decimal number 63.25 assuming the IEEE 754-2008 format. (10 points)

   sol.
   63.25 (decimal) = 111111.01 (binary)

   Let's normalise now. First we bring the first '1' to the left of the '.' by multiplying by a non-zero exponent:

   $111111.01 = 1.1111101 \times 2^5$

   Next we get rid of the '1' before the '.' and we add 15 to the exponent:

   0.1111101×2^{20}

   We convert the exponent to binary as well:

   $0.1111101 \times 2^{10100}$

   Finally we need to set the sign bit to '0' because the number is positive, so the result is:

   0 10100 1111101000

   where the first bit is the sign, the next 5 bits are the exponent, and the remaining 10 are the mantissa.

3. Assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows:

add	addi	not	beq	lw	sw
20%	20%	0%	25%	25%	10%

   a. In what fraction of all cycles is the data memory used? (5 points)

   b. In what fraction of all cycles is the input of sign-extend circuit needed? (5 points)

   sol.
   a. 25(lw)+10(sw)=35
   b. 25(beq)+25(lw)+20(addi)+10(sw)=80

4. Examine how pipeline affects the clock cycle time of the processor and assume that individual stages of the datapath have the following latencies:

IF	ID	EX	MEM	WB
250ps	350ps	150ps	300ps	200ps

Also, assume that instructions executed by processor are broken down as follows:

alu	beq	lw	sw
45%	20%	20%	15%

a. What is the clock cycle time in a pipelined and non-pipelined processor? (10 points)

b. What is the total latency of an lw instruction in a pipelined and non-pipelined processor? (10 points)

c. If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor? (10 points)

sol.
a.
Nonpipline:250+350+150+300+200=1250ps
Pipline:350ps
b.
Nonpipline:250+350+150+300+200=1250ps
Pipline : 350*5=1750ps
c.
Spilt ID stage
Nonpipline:250+350+150+300+200=1250ps
Pipline : 300ps

5. This problem refer to the following sequence of instructions, and assume that it is executed on a 5-stage pipelined datapath:

```
add   r5,  r2,  r1
lw    r3,  4(r5)
lw    r2,  0(r2)
or    r3,  r5,  r3
sw    r3,  0(r5)
```

   a. If there is no forwarding or hazard detection, insert nops to ensure correct execution. (5 points)
   b. Repeat (a) but now use nops only when a hazard cannot be avoided by changing or rearranging these instructions. You can assume register R7 can be used to hold temporary values in your modified code. (15 points)
   c. If there processor has forwarding, but we forgot to implement the hazard detection unit, what happens when this code executes? (10 points)

sol.
a.

```
ADD   R5,  R2,  R1
NOP
NOP
LW    R3,  4(R5)
LW    R2,  0(R2)
NOP
OR    R3,  R5,  R3
NOP
NOP
SW    R3,  0(R5)
```

b.

We can move up an instruction by swapping its place with another instruction that has no dependences with it, so we can try to fill some NOP slots with such instructions. We can also use R7 to eliminate WAW or WAR dependences so we can have more instructions to move up.

```
I1:   ADD   R5,  R2,  R1
I3:   LW    R2,  0(R2)      Move up to fill NOP slot
NOP
I2:   LW    R3,  4(R5)
NOP                         Had to add another NOP here,
NOP                         so there is no performance gain
I4:   OR    R3,  R5,  R3
NOP
NOP
I5:   SW    R3,  0(R5)
```

c.
If the processor with forwarding doesnt have a hazard detection unit implemented, there will be undesired register valuesat different instructions resulting in unintended results as the program executes there will be no stalling/bubble/NOPs taking place and hence the code wont work as expected.