HW2 - 20161595

Problem 1

2.23 value \$0 is less than \$t0
So \$t2=1 after slt \$t2, \$0, \$t0 statement.
value \$t2 =1 and \$0 is not equal
So we go to ELSE: addi, \$t2 \$t2, 2 statement
After this statement, \$t2 has value 3.

∴ The value of \$t2 after the instructions = 3 (0x00000003)

2.24

0X20000000 = 0010 0000 0000 0000 0000 0000 0000 0000 ← current PC
0x4000 0000 = 0100 0000 0000 0000 0000 0000 0000 0000 ← wants to get

when we use jump, PC's most significant 4bit should be same. but they are different (0010, 0100). So we cannot use jump instruction to set.
And when we use beq instruction, we can get the address by currentPC + 4 + offset x 4. but offset is only 16bits. So it is too far to get 0x40000000 from 0X20000000 by using 16bits offset. So we cannot use beq instruction

∴ NO, NO

Problem2.

2.39

lui \$t1, 8193
ori \$t1, \$t1, 18724

2.41

0000 0000 0000 0000 0000 0110 0000 0000 ← Current PC
0010 0000 0000 0001 0100 1001 0010 0100 ← wants to get

when we use branch instruction, we can get the address by currentPC + 4 + offset x 4. but offset is only 16bits. So it is too far to get 0x20014924 from 0x00000600 by using 16bits offset. So we cannot use branch instruction.

∴ NO

2.42

0001 1111 1111 1111 1111 0000 0000 0000 ← Current PC
0010 0000 0000 0001 0100 1001 0010 0100 ← wants to get

when we use branch instruction, we can get the address by currentpc +4+ offsetx4. but offset is only 16bits. So it is too far to get 0x20014924 from 0x1FFFF000 by using 16bits offset. So we cannot use branch instruction.

$\therefore$ NO

Problem3

2.46.1 CPU time = Clock Cycles × clock cycle time.

$Clock\ Cycles_{org} = 5\times10^8\times1 + 3\times10^8\times10 + 1\times10^8\times3$
$= 3.8\times10^9$

$CPUtime_{org} = 3.8\times10^9\times Clock\ cycletime_{org}$

$Clock\ Cycles_{new} = 0.75\times5\times10^8\times1 + 3\times10^8\times10 + 1\times10^8\times3$
$= 3.675\times10^9$

$CPUtime_{new} = 3.675\times10^9\times1.1\times Clock\ cycle\ time_{org}$
$= 4.0425\times10^9\times clock\ cycle\ time_{org}$

$\therefore$ It is not a good design choice. because cputime is increased.

2.46.2

$CPU\ time_d$ (double the Performance of arithmatic instructions)
$CPU\ time_t$ (improve the Performance of arithmatic instructions by 10 times)

$Clock\ Cycles_d = 5\times10^8\times0.5 + 3\times10^8\times10 + 1\times10^8\times3$
$= 3.55\times10^9$

$CPU\ time_d = 3.55\times10^9\times clock\ cycle\ time_{org}$

$clock\ Cycle_t = 5\times10^8\times0.1 + 3\times10^8\times10 + 1\times10^8\times3 = 3.35\times10^9$

$CPU\ time_t = 3.35\times10^9\times clock\ cycle\ time_{org}$

$\frac{CPU\ time_{org}}{CPU\ time_d} = \frac{3.8}{3.55} \approx 1.08$ $\frac{CPU\ time_{org}}{CPU\ time_t} = \frac{3.8}{3.35} \approx 1.13$

Speed up: 1.08 (doubled)
1.13 (10 times) } relative to original machine.

Ploblem 4.

3.6 10111001_2 $\quad$ 185 $\quad$ → no overflow and no underflow.

$$\begin{array}{r} 10111001_2 \\ -\ 01111010_2 \\ \hline 00111111_2 \end{array} \qquad \begin{array}{r} 185 \\ -122 \\ \hline 63 \end{array}$$

∴ So neither.

3.7 185 → 10111001_2 → 57 (cause it's sign-magnitude)

$-57+122=65$

It can be stored as 8bit signed integer.

So neither.

3.18

74 → 1001010_2

21 → 0010101_2

∴ Starting Divisor: 0010101 0000000

Starting Remainder: 0000000 1001010

next Page →

Iteration	Step	Quotient	Divisor	Remainder
0	initial values	000 0000	0010101 000 0000	0 000000 1001010
1	1: Rem = Rem − div	0000000	0010101 000 0000	1101011 1001010
	2b: Rem<0 ⇒ +Div, sll Q, Q0=0	000 0000	0010101 0000000	0000000 1001010
	3: Shift Div right	0000000	0001010 1000000	0000000 1001010
2	1: Rem = Rem − div	000 0000	0001010 100 0000	1110110 0001010
	2b: Rem<0 ⇒ +Div, sll Q, Q0=0	000 0000	0001010 1000000	0000000 1001010
	3: shift Div right	000 0000	0000101 0100000	000 0000 1001010
3	1: Rem = Rem − div	0000000	000 0101 0100000	1111011 0101010
	2b: Rem<0 ⇒ +Div, sll Q, Q0=0	0000000	000 0101 0100000	0000000 1001010
	3: shift Div right	0000000	0000010 1010000	0000000 1001010
4	1: Rem = Rem − div	0000000	0000010 1010000	1111101 1111010
	2b: Rem<0 ⇒ +Div, sll Q, Q0=0	0000000	0000010 1010000	0000000 1001010
	3: shift Div right	0000000	0000001 0101000	0000000 1001010
5	1: Rem = Rem − div	000 0000	000 0001 0101000	1111111 0100010
	2b: Rem<0 ⇒ +Div, sll Q, Q0=0	000 0 000	0000001 0101 000	0000000 1001010
	3: shift Div right	000 0000	0000000 1010100	0000000 1001010
6	1: Rem = Rem − div	000 0000	0000000 1010100	1111111 1110110
	2b: Rem<0 ⇒ +Div, sll Q, Q0=0	000 0 000	0000000 1010100	0000000 1001010
	3: shift Div right	000 0000	000 0000 0101010	000 0000 1001010
7	1: Rem = Rem − div	000 0000	0000000 0101010	0000000 0100000
	2a: Rem≥0 ⇒ sll Q, Q0=1	000 0001	0000000 0101010	0000000 0100000
	3: shift Div right	000 0001	000 0000 0010101	0000000 0100000
8	1: Rem = Rem − div	0000001	0000000 0010101	0000000 0001011
	2a: Rem≥0 ⇒ sll Q, Q0=1	0000011	0000000 0010101	0000000 0001011
	3: shift Div right	000 0011	0000000 0001010	0000000 0001011

∴ Quotient $= 0000011_2 = 3$

Remainder $= 0001011_2 = 11$

Problems.

3.22

$$0X0C000000 = \underbrace{0}_{\text{sign bit is}}\overbrace{000\ 1100\ 0}^{\text{exponent}}\underbrace{000\ 0000\ 0000\ 0000\ 0000\ 0000}_{\text{fraction}}$$

Sign bit = 0

exponent = $00011000_2 = 24$

fraction = 0

$$(-1)^s \times (1+\text{fraction}) \times 2^{(\text{exponent}-\overbrace{\text{bias}}^{127})} = 1 \times 2^{24-127} = 2^{-103}$$

$\therefore 2^{-103}$

3.24

$$63.25 = 11111\ 1.01_2 = (1.1111101) \times 2^5$$

Since we use IEEE754 double Precision format, bias is 1023.

$S = 0$

exponent $= 5 + 1023 = 100\ 0000\ 0100_2$

fraction $= 1111\ 1010\ 0000\ 0000\ \cdots\ 0000_2$

$\therefore$ 0100 0000 0100 1111 1010 0000 0000 0000