HW2 Answer V2

Problem 1

a) 5 points

Method 1

$$CPI_a = \frac{1}{totalIC} \sum_{k \in A,L,B} IC_k \times CPI_{1k} = 4 * 0.5 + 10 * 0.3 + 3 * 0.2 = 5.6$$

$$CPI_b = \frac{1}{totalIC} \sum_{k \in A,L,B} IC_k \times CPI_{1k} = 1 * 0.5 + 20 * 0.3 + 2 * 0.2 = 6.9$$

Method 2

Total instruction count = $5 \times 10^8 + 3 \times 10^8 + 2 \times 10^8 = 10^9$

Implementation a:
Total clock cycles = $5 \times 10^8 \times 4 + 3 \times 10^8 \times 10 + 2 \times 10^8 \times 3 = 5.6 \times 10^9$
Global CPI = $\frac{5.6 \times 10^9}{10^9} = 5.6$

Implementation b:
Total clock cycles = $5 \times 10^8 \times 1 + 3 \times 10^8 \times 20 + 2 \times 10^8 \times 2 = 6.9 \times 10^9$
Global CPI = $\frac{6.9 \times 10^9}{10^9} = 6.9$

Grading Criteria

CPI values for both implementations, 2.5 points each, total 5 points.

b) 5分

Implementation a:

Execution time = $\frac{5.6 \times 10^9}{5 \times 10^9} = 1.12s$ **2.5 points**

Implementation b:

Execution time = $\frac{6.9 \times 10^9}{5 \times 10^9} = 1.38s$ **2.5 points**

Grading Criteria

For both implementations, 2.5 points for a correct execution time, total 5 points.

c)

Assuming a 25% reduction in arithmetic instructions and a 10% increase in clock cycle time (frequency decreases by about 9.09%).

New instruction count
Arithmetic instructions = $5 \times 10^8 \times 0.75 = 3.75 \times 10^8$
Total instruction count = $3.75 \times 10^8 + 3 \times 10^8 + 2 \times 10^8 = 8.75 \times 10^8$

New frequency
Original cycle time $T \rightarrow$ New cycle time $1.1T \rightarrow$ New frequency = $\frac{5GHz}{1.1} \approx 4.545GHz$

Total cycles for implementation a
= $(3.75 \times 10^8 \times 4) + (3 \times 10^8 \times 10) + (2 \times 10^8 \times 3) = 5.1 \times 10^9$
New execution time = $\frac{5.1 \times 10^9}{4.545 \times 10^9} \approx 1.122s$ (originally 1.12 seconds, slower)

Total cycles for implementation b
= $(3.75 \times 10^8 \times 1) + (3 \times 10^8 \times 20) + (2 \times 10^8 \times 2) = 6.775 \times 10^9$
New execution time = $\frac{6.775 \times 10^9}{4.545 \times 10^9} \approx 1.491s$(originally 1.38 seconds, slower)

Conclusion
The changes are detrimental to both implementations, making the design change undesirable.

Grading Criteria

This question can be graded based on the assumptions in part b, 2.5 points for each implementation if calculated correctly, total 5 points.

Problem 2

a) **-40 + 100**

- Binary Two's Complement Representation (5 points)
  - `-40` = `1101_1000`
  - `100` = `0110_0100`
- Binary Addition (5 points)

```
  11011000
+ 01100100
-----------
 (1)0011_1100 (ignore carry)
```

- **Result**: $00111100_2 \rightarrow 60_{10}$。

Grading Criteria

5 points for correctly representing both numbers in two's complement, 5 points for correctly calculating binary addition and ignoring the carry.

b) **-40 - 100**

- Subtraction Converted to Addition (2.5 points)
  - The two's complement of `-100` = `1001_1100`
- Binary Addition (5 points)

```
  11011000
+ 10011100
-----------
 (1)01110100 (ignore carry)
```

- Overflow Detection: (2.5 points)
  - Both operands have a sign bit of `1`, and the result sign bit is `0` → **Positive Overflow**.
  - **Saturated Result**: The minimum value for 8-bit two's complement is **-128**.

Grading Criteria

2.5 points for correctly calculating the two's complement of -100, 5 points for correctly performing binary addition, and 2.5 points for accurate overflow detection and result.

Problem 3

Step	Operation	(Multiplicand)	(Product)
0	Initialization	`00110100`	`00000000` `00011010`
1	0: Shift right Prod	`00110100`	`00000000` `00001101`
2	1: Prod left half accumulate	`00110100`	`00110100` `00001101`
	Shift right Prod	`00110100`	`00011010` `00000110`
3	0: Shift right Prod	`00110100`	`00001101` `00000011`
4	1: Prod left half accumulate	`00110100`	`0100_0001` `0000_0011`
	Shift right Prod	`00110100`	`0010_0000` `1000_0001`
5	1: Prod left half accumulate	`00110100`	`0101_0100` `1000_0001`
	Shift right Prod	`00110100`	`0010_1010` `01000000`
6	0: Shift right Prod	`00110100`	`0001_0101` `0010_0000`
7	0: Shift right Prod	`00110100`	`0000_1010` `1001_0000`
8	0: Shift right Prod	`00110100`	`0000_0101` `0100_1000`

Grading Criteria

- Initialization of registers and correct register states: 5 points
- Intermediate steps total: 15 points (8 steps total, 2 points deducted for each error, maximum of 15 points)
- Result: 5 points
- **If someone omits intermediate iterative steps, deduct 10 points (this refers to omitting major steps; minor steps can be abbreviated).**

Problem 4

Initial State

- Dividend: 54 → `110110` (6 bits).
- Divisor: 17 → `010001` (6 bits).
- Remainder Register: `001101` (high 3 bits of the dividend: `110` → 6, needs adjustment).
- Quotient Register: `000000`.

Step	Divisor	Remainder	Quotient	Operation
0	0100_0100_0000	0000_0011_0110	00_0000	Initialization
1	0100_0100_0000	1011_1111_0110	000000	Remainder = Remainder - Divisor
	0100_0100_0000	0000_0011_0110	000000	Remainder < 0 + Divisor, add 0 to Quotient
	0010_0010_0000	0000_0011_0110	000000	Shift Divisor Right
2	0010_0010_0000	1110_0001_0110	000000	Remainder = Remainder - Divisor
	0010_0010_0000	0000_0011_0110	000000	Remainder < 0 + Divisor, add 0 to Quotient
	0001_0001_0000	0000_0011_0110	000000`	Shift Divisor Right
3	0001_0001_0000	1111_0010_0110	000000	Remainder = Remainder - Divisor
	0001_0001_0000	0000_0011_0110	000000	Remainder < 0 + Divisor, add 0 to Quotient
	0000_1000_1000	0000_0011_0110	000000	Shift Divisor Right
4	0000_1000_1000	1111_1010_1110	000000	Remainder = Remainder - Divisor
	0000_1000_1000	0000_0011_0110	000000	Remainder < 0 + Divisor, add 0 to Quotient
	0000_0100_0100	0000_0011_0110	000000	Shift Divisor Right
5	0000_0100_0100	1111_1111_0010	000000	Remainder = Remainder - Divisor
	0000_0100_0100	0000_0011_0110	000000	Remainder < 0 + Divisor, add 0 to Quotient
	0000_0010_0010	0000_0011_0110	000000	Shift Divisor Right
6	0000_0010_0010	0000_0001_0100	000000	Remainder = Remainder - Divisor
	0000_0010_0010	0000_0001_0100	000001	Remainder >= 0, add 1 to Quotient
	0000_0001_0001	0000_0001_0100	000001	Shift Divisor Right
7	0000_0001_0001	0000_0000_0011	000001	Remainder = Remainder - Divisor
	0000_0001_0001	0000_0000_0011	000011	Remainder >= 0, add 1 to Quotient

Final Result: Quotient `00_0011` = `3`, Remainder `00_0011` = `3`.

Grading Criteria

- Initialization of registers and correct register states: 5 points
- Intermediate steps total: 15 points (7 steps total, 3 points deducted for each error, maximum of 15 points)
- Result: 5 points
- If someone omits intermediate iterative steps, deduct 10 points (this refers to omitting major steps; minor steps can be abbreviated).

Problem 5

a)

`0x3E800000` = `0_01111101_00000000000000000000000`

- Sign bit: `0` (positive)
- Exponent: `01111101` → Decimal **125** → Actual exponent = 125 - 127 = **-2**
- Mantissa: `00000000000000000000000` → Implicit 1.0

Thus, the final result is $(-1)^0 \times 1.0 \times 2^{-2} = 0.25$

Grading Criteria

- Sign bit: 1 point
- Exponent: 1 point
- Mantissa: 1 point
- Final result: 2 points

b)

$12.5 = 1100.1_2 = 1.1001_2 \times 2^3$

- Sign bit: `0`
- Exponent: 3 + 127 = 130 → `10000010`
- Mantissa: `10010000000000000000000`

`0_10000010_10010000000000000000000` = `0x41480000`

Grading Criteria

- Sign bit: 1 point
- Exponent: 1 point
- Mantissa: 1 point
- Final result: 2 points (hexadecimal representation is not required)

c)

Binary Decomposition::

Number A (`0x3F800000`):

- Sign bit: `0`
- Exponent: `01111111` → 127 → Actual exponent **0**
- Mantissa: `1.00000000000000000000000` (implicit 1) → Value = $1.0_2 \times 2^0 = 1.0$

Number B（`0xBF400000`）：

- Sign bit: `1` (negative)
- Exponent: `01111110` → 126 → Actual exponent **-1**
- Mantissa: `1.10000000000000000000000` → Value = $-1.5 \times 2^{-1} = -1.1_2 \times 2^{-1} = -0.75$

Aligning Exponents:

- The exponent of number B is lower (-1), so it must be aligned to number A's exponent (0):
  - Mantissa right shift by 1 → `0.11000000000000000000000`
  - Exponent adjusted to 0

Adding Mantissas:

- Number A mantissa：`1.00000000000000000000000`
- Number B mantissa: `0.11000000000000000000000`
- **Actual operation**: `1.00000000000000000000000 - 0.11000000000000000000000 = 0.01000000000000000000000`

Normalization:

- Result $0.01_2 \times 2^0 = 1.00_2 \times 2^{-2}$
- Adjust exponent：-2 + 127 = 125 → `01111101`

Final Result:

`0 01111101 00000000000000000000000` = `0x3E800000`

Grading Criteria

- Decomposition of number A: 1 point
- Decomposition of number B: 1 point
- Exponent alignment: 1 point
- Mantissa addition: 1 point
- Final result: 1 point