Homework 11

Question 1:

Express the following base 10 number in IEEE 754 single-precision floating-point format. Express your answer in Hexadecimal.

```
-13.5625 ⇒ (?)
```

solution

Integer conversion:

```
⇒ (13)/2 ⇒ 6 r 1
⇒ (6)/2 ⇒ 3 r 0
⇒ (3)/2 ⇒ 1 r 1
⇒ (1)/2 ⇒ 0 r 1

Result: 1101
```

Decimal conversion:

```
⇒ (.5625) × 2 = 1.125
⇒ (.125) × 2 = 0.25
⇒ (.25) × 2 = 0.5
⇒ (.5) × 2 = 1.0

Result: .1001
```

Combining:

$\Rightarrow$ **1101.1001**

$$\Rightarrow 1101.1001 = 1.1011001 \times 2^3$$

We can create a generalized representation:

```
(* Sign bit for negative number *)
let Sign = 1;
let Bias = 127;
let Exponent = 3;
(* Biased exponent = 130 = 10000010 *)
let BiasedExp = Bias + Exponent;
(* Normalized mantissa (after 1.) = 1011001 *)
let Normalized = 1011001;
let zeroPadding = 0000000000000000;
(* Mantissa = normalized + padding (23 bits total) *)
let Mantissa = Normalized + zeroPadding;

(* Combine: {Sign | BiasedExp | Mantissa}
   ⇒ 1 10000010 10110010000000000000000 *)
let Result = Sign + BiasedExp + Mantissa;
```

Q.E.D.

$\Rightarrow$ Convert binary to Hexadecimal:

Binary (4 bits)	Hexadecimal
1100	C

Binary (4 bits)	Hexadecimal
0001	1
0101	5
1001	9
0000 (×4)	0000

$\therefore$ **−13.5625** $\Longrightarrow$ **`0xC1590000`**

Question 2:

Convert the following IEEE 754 single-precision floating-point number to decimal format.

0x40980000

solution

Starting with: `0x40980000`

Hex	Binary
4	0100
0	0000
9	1001
8	1000
0 (×4)	0000 (×4)

$\Longrightarrow$ Binary representation: `0100 0000 1001 1000 0000 0000 0000 0000`

```
let Sign = 0;               (* Positive number *)
let BiasedExp = 10000001;   (* 129 in decimal *)
let Mantissa = 00110000000000000000000;

(* Exponent = BiasedExp - 127 = 129 - 127 = 2 *)
let Exponent = 2;
```

Reconstructing the value:

Mantissa in decimal:

$$1.00110000... = 1 + \frac{0}{2} + \frac{0}{4} + \frac{1}{8} + \frac{1}{16}$$

$$= 1 + 0.125 + 0.0625 = 1.1875$$

Final calculation:

$$\text{value} = (+1) \times 1.1875 \times 2^2$$

$$= 1.1875 \times 4 = 4.75$$

$\therefore$ **`0x40980000` = 4.75**

Question 3:

Translate this C++ code into RISC-V assembly language with correct use of Floating-Point instructions where necessary. Submit your code and screenshot of the outputs.

```
int main() {
    float value1 = 3.5;
    float result = 0;

    if (value1 < 7)
        result = 7 + value1;
    else
        result = value1 / 7;

    cout << result << endl;
}
```

solution

Integer conversion:

Step-by-step translation:

1. Load `value1 = 3.5` into FP register `f0`
2. Load constant `7.0` into `f1` (IEEE-754)
3. Compare: `flt.s` checks if `f0 < f1`
4. Branch: If false, jump to ELSE
5. THEN: `fadd.s` computes `7 + 3.5 = 10.5`
6. ELSE: `fdiv.s` computes `3.5 / 7 = 0.5`
7. Print result using syscall

Expected outputs:

- `value1 = 3.5`: Result = **10.5**
- `value1 = 35.0`: Result = **5.0**

Decimal conversion:

```
    .data
value1: .float 3.5
result: .float 0.0
    .text
    .globl main
main:
    flw    f0, value1, t0
    li     t1, 0x40E00000
    fmv.w.x f1, t1
    flt.s  t2, f0, f1
    beq    t2, x0, ELSE
THEN:
    fadd.s f2, f1, f0
    fsw    f2, result, t0
    j      END
ELSE:
    fdiv.s f2, f0, f1
    fsw    f2, result, t0
END:
    flw    f10, result, t0
    li     a7, 2
    ecall
    li     a7, 10
    ecall
```

Question 4:

Part (i): Suppose one of the following control signals in the multi-cycle RISC-V processor has a stuck-at-0 fault, meaning the signal is always 0. What instructions (R-Type, lw, sw, beq) would malfunction? Why?

Part (ii): Repeat assuming the signal has a stuck-at-1 fault.

Control signals: (a) RegWrite (b) PCUpdate (c) Branch (d) AdrSrc (e) MemWrite (f) IRWrite

Multi-cycle RISC-V Control Signals

Solution

Part (i): Stuck-at-0 Faults

Signal	Instructions Affected	Reason
(a) RegWrite	R-type, lw	Cannot write results to register file. R-type ALU results and lw loaded data are lost. sw/beq unaffected (don't write registers).
(b) PCUpdate	All instructions	PC never advances - processor keeps fetching same instruction repeatedly. Program flow frozen.
(c) Branch	beq	PCWriteCond = Branch & Zero always 0, so conditional branches never taken. Other instructions unaffected.
(d) AdrSrc	lw, sw	Memory address forced to use PC instead of register base. lw/sw access wrong memory addresses.
(e) MemWrite	sw	Store operations do nothing - memory never updated. lw and fetch reads unaffected.
(f) IRWrite	All after first	Instruction register never updated with new instructions. CPU repeats old instruction.

Part (ii): Stuck-at-1 Faults

Signal	Instructions Affected	Reason
(a) RegWrite	All (corruption)	Register file written at wrong times with incorrect/partial data. sw/beq may spuriously overwrite registers.

Signal	Instructions Affected	Reason
(b) PCUpdate	All	PC updated in wrong cycles (during memory access, write-back). Causes skipped/repeated instructions or jumps to garbage addresses.
(c) Branch	beq + others	PCWriteCond = Zero (Branch gating lost). Any ALU operation producing Zero=1 causes spurious branch.
(d) AdrSrc	All (fetch broken)	Instruction fetch uses register address instead of PC. Fetches wrong instructions, corrupting entire program.
(e) MemWrite	All memory ops	Memory written during lw reads and instruction fetches, corrupting both instruction and data memory.
(f) IRWrite	All	IR overwritten at wrong times (e.g., with lw data). Control unit decodes wrong instruction, breaking flow.