CS2100 Computer Organisation Cheatsheet
for Final of AY 19/20 Semester 2, by Howard Liu

1 IEEE 754 Floating Point

Given a 32-bit single precision IEEE 754 floating point number A, we have the three parts:

- A[31:31] - Sign bit, 0 for +ve, 1 for -ve;
- A[30:23] - Exponent bits (8 bits)
- A[22:0] - Mantissa bits (24 bits)

From a decimal form, we can calculate the IEEE 754 representation in the following steps (taking number **85.125** as an example):

1. Covert integer part and decimal part into binary
   (a) $85 = (1010101)_2$
   (b) $.125 = (.001)_2$
2. We have 1010101.001 now. Make it only have one bit in the integer part, hence we do right shift for 6 bits. $(1010101.001 = 1.010101001 * 2^6)$
3. We calculate the exponent bit. For single precision, exponent = 127 + power of 2 . Hence we have 133 for this example. $(133 = (10000101)_2)$
4. Put everything together with 0 fills at the back, we have the binary representation in IEEE 754 format:
   $(0\ \ 1000101\ \ 010101001\ \ 00000000000000)_2$ = 0x42AA4000
5. Sometimes numbers cannot be represented precisely in binary (e.g. **0.3**), we do the binary conversion until we have the full 23-bit Mantissa.

2 Binary convertion

To convert a decimal point to binary, taking **0.125** as an example, we do this:

Decimal Number	Result	Binary
$0.125 * 2$	**0**.25	0
$0.25 * 2$	**0**.50	0
$0.50 * 2$	**1**.00	1
$0.00 * 2$	**0**.00	0

* Binary number is read from top to bottom. In this case, $.125 = (.001)_2$.

3 Negating Numbers

To negate a binary number, we have

- Sign-and-Magnitude: just flip MSB to negate a number $(0001_2 \rightarrow 1001_2)$
- 1s Complement: flip every bits to negate a number $(0001_2 \rightarrow 1110_2)$
- 2s Complement: add 1 to **LSB** of 1s Complement $(0001.01_2 \rightarrow 1110.10_2 \rightarrow 1110.11_2)$

We can detect overflow if either happens:

- +ve add +ve become -ve
- -ve add -ve become +ve

When performing binary addition on **1s-complement** numbers, if there is a carry out of the MSB, add 1 to the result and dispose the MSB. We do **NOT** need to do so for addition on **2s-complement** numbers.

4 Boolean algebra laws and theorems

Duality: if the AND/OR operators and identity elements 0/1 in a Boolean equation are interchanged, it remains valid (but brackets are to be added when necessary). (e.g. $X + X \cdot Y = X \rightarrow X \cdot (X + Y) = X$)

Laws:

- Identity: $A + 0 = 0 + A = A$
- Inverse/Complement: $A + A' = 1$
- Commutative: $A + B = B + A$
- Associative: $A + (B + C) = (A + B) + C$
- Distributive: $A + (B \cdot C) = A \cdot B + A \cdot C$

Theorems:

- Idempotency: $X + X = X$
- One element/Zero element: $X + 1 = 1$
- Involution: $(X')' = X$
- Absorption 1: $X + X \cdot Y = X$
- Absorption 2: $X + X' \cdot Y = X + Y$
- DeMorgans': $(X + Y)' = X' \cdot Y'$
- Consensus: $X \cdot Y + X' \cdot Z + Y \cdot Z = X \cdot Y + X' \cdot Z$

5 Flip-flop reference

Characteristic Tables:

J	K	Q^+
0	0	Q
0	1	0
1	0	1
1	1	Q(t)'

S	R	Q^+
0	0	Q
0	1	0
1	0	1
1	1	Unpredictable

D	Q^+
0	0
1	1

T	Q^+
0	Q
1	Q'

Excitation Tables:

Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Q	Q^+	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Q	Q^+	D
0	0	0
0	1	1
1	0	0
1	1	1

$D = Q^+$

Q	Q^+	T
0	0	0
0	1	1
1	0	1
1	1	0

$T = Q \oplus Q^+$

6 Notes about pipeline

Considering instruction A and B. B is just the instruction executed after A.

- A pipeline stage of B will always behind the same stage for A, even if the resource is not occupied and there is no data dependency / control hazard. (e.g. D stage of instruction B cannot be executed at cycle 3, even D is vacant)

Inst	1	2	3	4	5	6	7	8
A	F			D	E	M	W	
B		F			D	E	M	W

- For N instructions, ideal 5-stage pipeline takes $N + 2$ cycles to execute.
- Delays introduced when no forwarding, no early branching and no branch prediction.
  - Data dependency: **2 cycles**
  - Control hazard (beq/bne, j): **3 cycles**
- Delays introduced when there is forwarding: **1 cycle** for data dependency after **lw**.
- Data dependency for **beq/bne** instruction can lead to delay **when forwarding and early branching is implemented**. This is because data is required in stage 2 (D), not stage 3 (E), of the branching instruction.
- When branch prediction is wrong, flush everything including the correct ones to be executed. This is especially true when there is **no early branching**. Considering the following example: assuming not taken is used, but actually A branched to C. C and D is executed once again after everything flushed.

Inst	1	2	3	4	5	6	7	8
A (beq)	F	D	E	M	W			
B		F	D	E	X	X		
C			F	D	X	X	X	
D				F	X	X	X	X
C (again)					F	D	E	M

7 Cache

Average Access Time formula:

$$\text{Hit rate} \times \text{Hit time} + (1 - \text{Hit rate}) \times \text{Miss penality}$$

8 Quick reference

if (a < b) {...} else {...}

```
slt $t0, $a, $b
beq $t0, $zero, else
```

if (a >= b) {...} else {...} : flip a and b in the above example.

9 Miscellaneous notes

- Zero-extension and Sign-extension is different especially in the case that the number starts with 1 in binary.
- MIPS does not have **not** instruction. We use **nor $1, $1, $zero** to do bitwise not.