Floating point: max range: -2^128 to 2^128, max precision: less than 2^-126, larger range and better precision with same number of bits at small numbers. BUT precision changes across the range, very coarse precision at two ends of the range

TUTORIAL #9 Computer Arithmetic and Processor Pipeline (SC1006/Cx1106)

9.1 Computer Arithmetic

Fixed point: max range when radix right of LSB, unsigned = 2^32. Max precision: radix left of MSB 2^-32. Uniform precision across entire range. Smaller range and worse precision with same number of bits

1. Describe and explain the pros and cons of fixed and floating number system, particularly the representable range and precision. Illustrate using example of 32-bit fixed point and the 32-bit floating point number in single precision IEEE754 format.

2. An array consisting of the length of 256 wires is given by L[0], L[1], … , L[255]. Assume that user are not allowed to test for any overflow during computation, describe a scheme to compute the average length of the 256 wires that will yield a result with the highest precision based on the following specifications:

use 13bits for the integer and 3 bits for the fraction. max integer for 13 bits is 8191 which can hold the addition of 8 maximum length wires. The last 3 bits are used to divide the 8 wires to get a precise number for the average length of 8 wires. This forms Set A. 8 of 32 different Sets A are added together and divided by 8. This forms Set B. 4 of such set Bs form and are added together then divided by 4. ✗

- 16-bit registers are used for storing the data and result.
- Only Single-Precision (16-bit) and Fixed-Point arithmetic is used.
- Maximum possible length of each wire is 0x3FF and is an integer. → 12 bits = 1023

Never account for float when doing division????

No coding is required, illustrate your answer in the form of a mathematical expression and justify your answer.

Unsigned arithmetic used to maximise the range since the wire length cannot be negative

16bit->maximum 0xFFFF
Max length of one wire = 0x3FF=>Max number of wire length that can be accumulated before the result will potentially overflow = 64

Math expression =
{(L[0]+...+L[63])/64 + (L[64]+...+L[127])/64 + (L[128]+...+L[191])/64 + (L[192]+...+L[255])/64}/4

Always perform division last to avoid truncating the LSBs too early in the computation. But need to take note of overflow when using addition, subtraction and multiplication

9.2 Pipelines

3. Consider a processor (not ARM processor) with 4 pipeline stages: Fetch Instruction (F), Decode (D), Execute (E) and Store (S). Assume that

- Branch target address is calculated at the execute stage
- Instruction length for every instruction is one word long
- Each pipeline stage take 1 cycle to complete
- Delay Branching is not enabled.

How many cycles does the code in Figure 9.2 take? You can ignore all pipeline conflicts you see in the code, just focus on the pipeline behaviour and execution cycles.

```
        MOV   R6, #0x900  ; I1
        MOV   R5, #0      ; I2
        MOV   R4, #0x800  ; I3
        MOV   R3, #0x300  ; I4
Loop    LDR   R0, [R3]    ; I5
        LDR   R1, [R4]    ; I6
        ADD   R2, R1, R0  ; I7
        ADD   R5, R5, #1  ; I8
        ADD   R4, R4, #1  ; I9
        ADD   R3, R3, #1  ; I10
        CMP   R5, #5      ; I11
        BNE   Loop        ; I12
        ADD   R4, R4, R2  ; I13
        STR   R2, [R6]    ; I14
```

Figure 9.2

7

4(8+1) + 8

2

I1 will take 4 cycles to pass through pipeline ?

~~4~~ + ~~(8x5) - 1~~

~~x~~ +

~~5 = 48x~~

I5 to I12. For the first four iterations, each iteration take 8+2 cycles. This includes 2 cycles discarded in each iteration ?

For the fifth iteration, 8 cycles is needed as the branch is not taken so no instructions are discarded.

Total = 57 cycles

4. Consider a processor (not ARM processor) with 4 pipeline stages: Fetch Instruction (F), Decode (D), Execute (E) and Store (S). Assume that

- Branch target address is calculated at the execute stage
- Instruction length for every instruction is one word long
- Each pipeline stage takes 1 cycle to complete
- No Resource Conflicts
- Delayed Branching is enabled

Status bits are updated in the E stage so no data dependency conflicts between CMP and BNE instructions

Identify and describe ALL pipeline conflicts the code in Figure 9.3 has when executed in the pipeline processor above. Suggest workaround for pipeline conflicts identified.

Data dependency conflicts ✓

The pipeline can be stalled by stalling the decode stage of the next instruction.

Another way is to insert NOP instructions between instructions with data dependencies. ✓

Compare S stage of previous instruction and see if current instruction's E stage is using the same register to determine data dependency

```
     MOV  R6, #0x900  ; I1
     MOV  R5, #0      ; I2
     MOV  R4, #0x800  ; I3
     MOV  R3, #0x300  ; I4
Loop LDR  R0, [R3]    ; I5
     LDR  R1, [R4]    ; I6
     ADD  R2, R1, R0  ; I7
     ADD  R5, R5, #1  ; I8
     CMP  R5, #5      ; I9
     BNE  Loop        ; I10
     ADD  R4, R4, #1  ; I11
     ADD  R3, R3, #1  ; I12
     ADD  R4, R4, R2  ; I13
     STR  R2, [R6]    ; I14
```

← 5 ✓

← 7 ✓

← 9 ✓

I5 & I12

Figure 9.3

No data dependency between I13 and I 14 as I14 is just reading and not writing to R2

No branch conflicts as delayed branching is used

Data dependency conflicts:

1) I4 and I5. Data dependency. R3 register value is used in I5 before I4 completes loading R3 register. Resolved by swapping I3 and I4 or, swapping I5 and I6, OR inserting a NOP instruction between I4 and I5.

2) I6 and I7. Data dependency. R1 register value is used in I7 before I6 completes loading R1 register. Resolved by swapping I7 and I8, OR inserting a NOP instruction between I6 and I7.

3)I8 and I9. Data dependency. R5 register value is used in I9 before I8 completes loading R1 register. Resolved by swapping I7 and I8, OR inserting a NOP instruction between I8 and I9.

4) I5 and I12. Data dependency. I11 and I12 are delay slot instructions. During the first four iterations of the loop, I5 uses R3 before I12 can load the latest increment to R3.
Resolved by swapping I11 and I12, and add a NOP before I13

(Not necessary to be covered during tutorial)
[Optional, but students are encouraged to attempt these questions]

9.3 Rounding Error

5. You have been tasked to write a program that calculates the actual time based on a counter that is incremented once every 0.10 seconds. For example, if the counter value is 3,600,000, you would expect the actual time to be 100 hours ((3,600,000 x 0.1) / (60 x 60)).

   Suppose you have decided to use a 24-bit fixed point representation as shown in Figure 8.4 to store the value of 0.10 seconds ($2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} + \ldots$).

0	▪	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0

-4 -5 -8 -9 -12 -13 -16 -17 -20 -21

Figure 9.3 – Fixed point representation of 0.1010

a. Approximate the round-off error (in decimal) of 0.10_{10} due to the fixed-point representation.
b. What is the effect of this round-off error on the time calculated if the counter value is 3,600,000?

a) 0.000110011001100110011001100 = 2^-4+ 2^-5+ 2^-8+ 2^-9+ 2^-12+ 2^-13+ 2^-16+ 2^-17+ 2^-20+ 2^-21 = 0.9999990463.
Round off error = 0.1-0.9999990463 = 0.00000009536 ✓

The time calculated will be less than expected.

Time calculated will be off by approximately 0.34 s (3600000 x 0.000000095)