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

Solutions

## 9.1 Computer Arithmetic

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.

[Suggested Solution]

- Floating Point (IEEE754)
  - Max Range ≈ $2*2^{128}$ (~$2^{128}$ to $-2^{128}$ ).
  - Max Precision (near to zero) less than $2^{-126}$
- Fixed Point
  - Max Range (Radix right of LSB, unsigned) ≈ $2^{32}$
  - Max Precision (Radix left of MSB) $2^{-32}$
- Floating point yield a larger range and better precision at small numbers with the same number of bits representation.
- One usually needs the best precision when the numbers are small.
- However, Fixed point number has the advantage of having uniform precision across entire range.
- Floating point number's precision changes across the range and the very coarse precision at the two end of the range may not be desirable to the intended algorithm.

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:

   - 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.

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

[Suggested Solution]

Solutions

- Unsigned arithmetic used to maximise the range since the wire length cannot be negative
- 16-bit => 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 data dependencies 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**

[Suggested Solution]

- I1 to I4: 7 cycles (I1 will take 4 cycles to pass through the pipeline)
- I5 to I12. For the first four iterations, each iteration takes (8+2) cycles. This includes 2 cycles discarded in each iteration.
- I5 to I12. Fifth iteration takes 8 cycles since the branch is not taken so no instructions are discarded.
- I13 to I14: 2 cycles.
- Total = 7 + (8+2)*4 + 8 + 2 = 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

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.

```
      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
```

**Figure 9.3**

**[Suggested Solution]**

- 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 ad I6, OR
    - inserting a NOP instruction between I4 and I5
- 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
- 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

- 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.
- No branch conflicts since delayed branching is used.
- No data dependency between I13 and I14 since I14 is just reading and not writing to R2
- Assumption not stated in the question (should be stated) is that Status Bits are updated in the E stage, that why there are no data dependency conflicts between CMP and BNE instructions.

---

(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 9.3 to store the value of 0.10 seconds ($2^{-4} + 2^{-5} + 2^{-8} + 2^{-9} + 2^{-12} + 2^{-13} + \dots$).

| | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | ▪ | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |

**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?

[Suggested Solution]

a. The decimal value of the fixed point representation in Figure 9.3 is $0.09999990463256 84_{10}$.

   Hence, the round-off error is approximately **$0.000000095_{10}$** ($0.10_{10}$ - $0.0999999046325684_{10}$).

b. The time calculated will be off by approximately **0.34 seconds** ($3600000_{10}$ x $0.000000095_{10}$).

Note: This question is based on an actual example of disasters caused by bad numeric. The inaccurate calculation of time has contributed to fatalities when it was used for intercepting enemy missiles during Gulf War. While an error of 0.34 seconds may not seem like much, a missile travelling at 1,676 meters per second would have travelled approximately 500 meters in that time.