Advanced Computer Architecture

Lecture No. 36

Reading Material

Vincent P. Heuring & Harry F. Jordan — Chapter 6
Computer Systems Design and Architecture — 6.3.2, 6.4, 6.4.1, 6.4.2, 6.4.3

Summary

- NxN Crossbar Design for Barrel Rotator
- Barrel Shifter with Logarithmic Number of Stages
- ALU Design
- Floating-Point Representations
- IEEE Floating-Point Standard
- Floating-Point Addition and Subtraction
- Floating-Point Multiplication
- Floating-Point Division

NxN Crossbar Design for Barrel Rotator

Figure 6.11 of the text book
The figure shows an NxN crossbar design for barrel rotator. x indicates the input. So x0,x1,…,xn-1 are applied to the rows. The vertical lines are indicated by y1, y2,…yn-1 where y shows the output. So this forms a cross of x and y and the number of cross points are NxN. There is also a connection between each input and output using a tri-state buffer. At the input, we have a decoder which is used to select the shift count. Each output from the decoder is connected diagonally to the tri-state buffers. This arrangement requires N2 gates.

Barrel Shifter with Logarithmic Number of Stages

Another alternate to an NxN crossbar barrel rotator is a logarithmic barrel shifter. This design is time-space trade-off. In this case, the number of shifts required is eight, and then there will be three stages for this purpose. Now a word is passed as input to the shifter. There are two possibilities. First the input word is passed to the next stage without any shift. This process is called bypass and second option is shift. The word is passed to the next stage after shift.
For the first stage, we have 1-bit right shift, for second stage, 2-bit right shift and so on. There is also a shift count unit which controls the number of shifts. For example, if 1-bit shift is required then only s0 will be one and other signals from shift count will be zero. If we want a 3-bit shift, then s0 and s1 will be 1 and all other signals will be zero.
The figure also shows one shift/bypass cell which is a combinational logic circuit. A shift/bypass signal decides whether the input word should be shifted or bypassed. This

design requires only O (NlogN) switches but propagation delay has increased i.e. from O(1) to O(logN).

Figure 6.12 of the text book

ALU Design

ALU is a combination of arithmetic, logic and shifter unit along with some multiplexers and control unit. The idea is that based on the op-code of an instruction, appropriate control signals are activated to perform required ALU operation.
Figure 6.13 of the text book
The diagram shows two inputs x and y and one output z. All these are of n-bits. The inputs x and y are simultaneously provided to arithmetic, logic and shifter unit. There is a control unit which accepts op-code as input. Based on the op-code, it provides control signals to arithmetic, logic and shifter unit. The control unit also provides control signals to the two multiplexers. One mux has three inputs; each from arithmetic, logic and shifter unit and its output is z. The second mux provides status output corresponding to condition codes.

Floating Point Representations

Example

-0.5 × 10-3
Sign = -1
Significand= 0.5
Exponent= -3
Base = 10= fixed for given type of representation

Significand is also called mantissa.
In computers, floating-point representation uses binary numbers to encode significant, exponent and their sign in a single word.
The diagram on Page 293 of the text shows an m-bit floating point number where s represents the sign of the floating point number. If s = 1 then the floating-point number will be a positive number; if s= 0 then it will be a negative number. The e field shows the value of exponent. To represent the exponent, a biased representation is used. So we represent e^ instead of e to show biased representation. In this technique, a number is added to the exponent so that the result is always positive. In general floating point numbers are of the form.
(-1)s × f × 2e

Normalization

A normalized, non zero floating point number has a significand whose left-most digit is non- zero and is a single number.
Example

0.56 × 10-3……….. (Not normalized)
5.6 × 10-3……….. (Normalized form)

Same is the case for binary.

IEEE Floating-Point Standard

IEEE floating -point standard has the following features.

Single-Precision Binary Floating Point Representation

- 1-bit sign
- 8-bit exponent
- 23-bit fraction
- A bias of 127 is used.

Figure 6.15 of the text book

Double precision Binary Floating Point Representation

- 1-bit sign
- 11-bit exponent
- 52-bit fraction
- Exponent bias is 1023

Figure 6.16 of the text book.

Overflow

In table 6.7 of the text book, e^= 255, denotes numbers with no numeric value including + ∞ and - ∞ and called Not-a-Number or NaN. In computers, a floating-point number ranges from 1.2 × 10-38 ≤ x ≤ 3.4 × 1038 can be represented. If a number does not lie in this range, then overflow can occur.
Overflow occurs when the exponent is too large and can not be represented in the exponent field.

Floating –Point Addition and Subtraction

The following are the steps for floating-point addition and subtraction.

- Unpack sign , exponent and fraction fields
- Shift the significand
- Perform addition
- Normalize the sum
- Round off the result
- Check for overflow

Figure 6.17 of the text book.

Example 1

Perform addition of the following floating-point numbers.
0.510 , -0.437510
Binary:
$0.5_{10} = 1/2_{10} = 0.1_2 = 1.000$ x 2-1
$-0.4375_{10} = -7/16_{10} = -7/24 = -0.0111_2 = -$ 1.110 x 2-2

Align: -1.110 x 2-2 → -0.111 x 2-1

Addition: 1.000 x 2-1 + (-0.111 x 2-1) = 0.001 x 2-1

Normalization of Sum:

0.001_2 x 2-1= 0.010_2 x 2-2
= 1.000_2 x 2-4

Hardware Structure for Floating-Point Add and Subtract
Figure 6.17 of the text book.

Floating-Point Multiplication

The floating-point multiplication uses the following steps:

- Unpack sign, exponent and significands
- Apply exclusive-or operation to signs, add exponents and then multiply significands.
- Normalize, round and shift the result.
- Check the result for overflow.
- Pack the result and report exceptions.

Floating-Point Division

The floating-point division uses the following steps:

- Unpack sign, exponent and significands
- Apply exclusive-or operation to signs, subtract the exponents and then divide the significands.
- Normalize, round and shift the result.
- Check the result for overflow.
- Pack the result and report exceptions.