ECE 152
Introduction to Computer Architecture
Arithmetic and ALU Design
Copyright 2012 Daniel J. Sorin
Duke University

Slides are derived from work by
Amir Roth (Penn) and Alvy Lebeck (Duke)
Spring 2012

Where We Are in This Course Right Now

• So far:
  • We know what a computer architecture is
  • We know what kinds of instructions it might execute
• Now:
  • We learn how to perform many of the most important instructions
    • Computers spend lots of time doing arithmetic and logical ops
    • Examples: add, subtract, multiply, divide, shift, rotate, load, store
  • We develop hardware for arithmetic logic unit (ALU)
• Next:
  • We learn how the computer uses and controls the ALU
  • Lots of stuff in computer besides the ALU
    • E.g., Logic to fetch and decode instructions, memory, etc.

This Unit: Arithmetic and ALU Design

• Integer Arithmetic and ALU
  • Binary number representations
  • Addition and subtraction
  • Integer ALU
  • Shifting and rotating
  • Multiplication
  • Division
• Floating Point Arithmetic
  • Binary number representations
  • FP arithmetic
  • Accuracy

Readings

• Patterson and Hennessy textbook
  • Chapter 3

Readings

• Patterson and Hennessy textbook
  • Chapter 3
Review: Fixed Width

- You've seen much of the upcoming material in ECE 52 – if none of this looks familiar, please talk with me ...

- In hardware, integers have **fixed width**
  - N bits: 16, 32, or 64
  - LSB is $2^0$, MSB is $2^{N-1}$
  - **Unsigned number range**: 0 to $2^N - 1$
  - Numbers $>2^N$ represented using multiple fixed-width integers
    - In software
    - **ICQ**: What happens when your C++ code specifies an integer greater than this max? What does compiler do?

Review: Two’s Complement

- What about negative numbers?
  - **Option I: sign/magnitude**
    - Unsigned binary plus one bit for sign
      - $10_{10} = 000001010$, $-10_{10} = 100001010$
    - Two representations for zero (0 and $-0$ are different)
    - Addition in hardware is difficult
    - **Number range**: $-(2^{N-1} - 1)$ to $2^{N-1} - 1$
    - Matches our intuition from "by hand" decimal arithmetic
  - **Option II: two’s complement (TC)**
    - Leading 0s mean positive number, leading 1s negative
      - $10_{10} = 00001010$, $-10_{10} = 11110110$
      - One representation for 0
      - Easy addition in hardware
    - **Number range**: $-(2^{N-1})$ to $2^{N-1} - 1$ → not symmetric

Review: Still More On TC

- What is the interpretation of TC?
  - Same as binary, except **MSB represents $-2^{N-1}$, not $2^{N-1}$**
    - $-10 = 11110110 = -2^7 + 2^6 + 2^5 + 2^4 + 2^2 + 2^1$
    - Works with any width
      - $-10 = 110110 = -2^4 + 2^3 + 2^2 + 2^1$
      - Why? $2^N = 2^1 2^{N-1}$
      - $-2^3 + 2^4 + 2^2 + 2^1 = \ldots$
    - Trick to negating a number quickly: $-B = B' + 1$
      - $-1 = (0001)' + 1 = 1110 + 1 = 1111 = -1$
      - $-(-1) = (1111)' + 1 = 0000 + 1 = 0001 = 1$
      - $-0 = (0000)' + 1 = 1111 + 1 = 0000 = 0$
      - Think about why this works (on your own time)

Review (way back!): Decimal Addition

- Remember decimal addition from 1st grade?
  - 4 + 39 = 43 + 29 = 72
  - Repeat N times
    - Add least significant digits and any overflow from previous add
    - Carry the overflow to next addition
    - **Overflow**: any digit other than least significant of sum
      - Shift two addends and sum one digit to the right
    - Sum of two N-digit numbers can yield an N+1 digit number
Review: Binary Addition

- Binary addition works the same way:
  
  $43 = 00101011$
  
  $+29 = 00011101$
  
  $72 = 01001000$

- Repeat N times:
  - Add least significant bits and any overflow from previous add:
  - Carry the overflow to next addition:
  - Shift two addends and sum one bit to the right:
  - Sum of two N-bit numbers can yield an N+1 bit number:
    - More steps (smaller base)
    - Each one is simpler (adding just 1 and 0)
    - So simple we can do it in hardware

Review: The Half Adder

- How to add two binary integers in hardware?
  - Start with adding two bits:
    - When all else fails... look at truth table:

  $A \oplus B = C$
  $C = A \cdot B$

- This is called a half adder (HA):

Review: The Full Adder

- We could chain half adders together, but to do that...
  - Need to incorporate a carry out from previous add:
  - Let's look at the truth table:

  $C_i \cdot A \cdot B = C_i + C_i \cdot A' + C_i \cdot B'$
  $C_i = C_i \cdot A'B + C_i \cdot AB + C_i = C_i' + AB$

- This is a full adder → ICQ: what is its delay (in #gates)?

A 16-bit Adder

- Simple 16-bit adder:
  - 16 1-bit full adders "chained" together:
    - $C_{i-1} = C_i = C_i \cdot B + C_i \cdot A$ etc.
    - $C_{15} = 0, C_{15}$ is carry-out of entire adder:
      - $CO_{16} = 1 \rightarrow "overflow"$
  - Design called ripple-carry: how fast is it?
    - In terms of gate delays (longest gate path):
      - Longest path is to $CO_{16}$ (or $S_{15}$):
        - $d(CO_{16}) = 2 + \max(d(A_{15}),d(B_{15}),d(CI_{15}))$
        - $d(A_{15}) = d(B_{15}) = 0, d(CI_{15}) = d(CO_{14})$
        - $d(CO_{14}) = 2 + d(CO_{13}) = 2 + 2 + d(CO_{13}) ...$
        - $d(CO_{16}) = 32
          - $2N = slow!
A Faster (16-bit) Adder

- One option: carry-select adder
  - Do \( A_{15-8} + B_{15-8} \) twice, once assuming \( C_0 = 0 \), then once = 1
  - Effectively cuts carry chain in half
    - But adds 8b adder and mux

\[
\begin{align*}
\text{CO} & \rightarrow \text{S}_{7-0} \\
\text{S}_{15-8} & \rightarrow \text{S}_{7-0} \\
\text{A}_{15-0} & \rightarrow \text{S}_{7-0} \\
\text{B}_{15-0} & \rightarrow \text{S}_{7-0} \\
\text{CO} & \rightarrow \text{mux}
\end{align*}
\]

How Fast Is the Faster Adder?

- \( d(CO_{15}) = \max\{d(CO_{15-8}), d(CO_{7-0})\} + 2 \) (\(+2\) is for mux)
- \( d(CO_{15}) = \max(2^8, 2^8) + 2 = 18 \) (2N delay for 16bit add)
- For dividing N-bit adder into 2 parts: \( 2^N(N/2) + 2 = N+2 \)

- What if we broke up 16b adder into 4 parts?
  - Would delay be \( 2^4(8/4) + 2 = 10 \)? Not quite!
  - \( d(CO_{15}) = \max\{2^4, \max\{d(CO_{11-8}), d(CO_{7-0})\}\} + 2 \)
  - \( d(CO_{15}) = \max(2^4, \max(2^4, \max(2^4, \max(2^4, 2^4)) + 2) + 2 \)
  - \( d(CO_{15}) = 2^4 + 3*2 = 14 \)

- In general, N-bit adder in M pieces: \( 2^N(N/M) + (M-1)*2 \)
  - 16-bit adder in 8 parts: \( 2^8(16/8) + 7*2 = 18 > 14 \) ???!

Another Option

- Is the piece-wise faster adder as fast as we can go?
  - No!
- Another approach to using additional resources
  - Instead of redundantly computing sums assuming different carries, use redundancy to compute carries more quickly
  - This approach is called carry lookahead addition (CLA)

Review: Carry Lookahead Addition (CLA)

- Let’s look at the carry function
  - \( C_{16} = CO_{15} = A_{16}B_{15} + A_{16}C_{15} + B_{16}C_{15} = (A_{16}B_{15}) + (A_{15} + B_{15})C_{15} \)
- Very important insights into CLA:
  - \( (A_{15}B_{15}) \) generates a carry regardless of \( C_{15} \) → rename to \( g_{15} \)
  - \( (A_{15} + B_{15}) \) propagates \( C_{15} \) → rename to \( p_{15} \)
  - \( C_{16} = g_{15} + p_{15}C_{15} \)
  - \( C_{16} = g_{15} + p_{15} + p_{15}C_{15} \)
  - \( C_{16} = g_{15} + p_{15}g_{14} + p_{15}p_{15}C_{15} \)
  - Important note: can compute \( C_{16} \) in 2 levels of logic!
  - Similar functions for \( C_{15} = (CO_{14}), \text{etc.} \)
  - In general: \( C_i = g_i + p_i C_i \)
Infinite Carry Lookahead

- Previous slide's CLA functions assume "infinite" hardware
  - Performance? Critical path is $d(S_{out}) = ?$
  - $d(p_0, q_0) + d(C_{out} \text{ given } p_0, q_0) + d(S_{out} \text{ given } C_{out}) = 1 + 2 + 2 = 5$ !!
  - Constant delay, i.e., not a function of $N$
  - But not very practical in terms of hardware
- Assume $2N$ gates to compute $p$ and $g_i$ initially (ICQ: why 2N?)
- Computation of a single $C_i$ needs the following hardware:
  - $N$ AND gates + 1 OR gate, and largest gates have $N+1$ inputs
  - Computation of all $C_1$ needs:
    - $N^*(N+1)/2$ AND gates + $N$ OR gates, max $N+1$ inputs
  - Not too bad if $N=16$: 152 gates, max input 17
  - Pretty bad if $N=64$: 2144 gates, max input 65
  - Big circuits are slow and high input gates are slow

Motivation for Multi-Level Carry Lookahead

- Let's look at what we have so far (the two extremes)
  - **Ripple carry**
    - Few small gates: no additional gates used to speed up addition
    - Logic in series: constant latency
  - **Infinite CLA**
    - Many big gates: $N^*(N+3)/2$ additional gates, max $N+1$ inputs
    - Logic in parallel: constant latency of 5 gate delays
  - We'd like something in between
    - Reasonable number of small gates
    - Sub-linear (doesn't have to be constant) latency
  - **Multi-level CLA**
    - Exploits hierarchy to achieve good compromise between the two extremes

Two-Level CLA for 4-bit Adder

- Individual carry equations
  - $C_i = g_i + p_i C_i$
  - $C_1 = g_1 + p_1 C_1$
  - $C_2 = g_2 + p_2 g_1 + p_2 C_2$
  - $C_3 = g_3 + p_3 g_2 + p_3 g_1 + p_3 C_3$
  - $C_4 = g_4 + p_4 g_3 + p_4 g_2 + p_4 g_1 + p_4 p_i C_3$
- Fully expanded (infinite hardware) CLA equations
  - $C_i = g_i + p_i C_i$
  - $C_1 = g_1 + p_1 C_1$
  - $C_2 = g_2 + p_2 g_1 + p_2 g_0 + p_2 p_0 C_0$
  - $C_3 = g_3 + p_3 g_2 + p_3 g_1 + p_3 g_0 + p_3 p_0 p_1 C_0$
  - $C_4 = g_4 + p_4 g_3 + p_4 g_2 + p_4 g_1 + p_4 p_0 p_1 p_2 C_0$
- Hierarchical CLA equations
  - **First level**: expand $C_i$ using $C_j$ and $C_k$ using $C_j$
    - $C_j = g_j + p_j (g_k + p_k C_k) = (g_j + p_j g_k) + (p_j p_k C_k) = G_{jk} + P_{jk} C_k$
    - $C_k = g_k + p_k (g_j + p_j C_j) = (g_k + p_k g_j) + (p_k p_j C_j) = G_{kj} + P_{kj} C_j$
  - **Second level**: expand $C_i$ using expanded $C_j$
    - $C_i = G_{ij} + P_{ij} (G_{jk} + P_{jk} C_j) = (G_{ij} + P_{ij} G_{jk}) + (P_{ij} P_{jk} C_j) C_j$
    - $C_i = G_{ij} + P_{ij} C_j$

Two-Level (2L) CLA for 4-bit Adder

- **Hardware?**
  - First level: block is infinite CLA for $N=2$
    - 5 gates per block, max # gate inputs ($M_{NGI}$)=3
    - 2 of these "blocks"
  - Second level: 1 of these "blocks"
    - Total: 15 gates & 3 MNGI
    - Infinite CLA: 14 & 5 (??)
- **Latency?**
  - Total: 9 (ICQ: why?)
  - Infinite CLA: 5
  - **2 level: bigger and slower??!!**
    - ICQ: what happened?
Two-Level CLA for 16-bit Adder

- 4 G/P inputs per level

Hardware?
- First level: 14&5 * 4 blocks
- Second level: 14&5 * 1 block
- Total: 70&5
  - Infinite: 152&17

Latency?
- Total: 9 (1 + 2 + 2 + 2 + 2)
  - Infinite: 5

That’s more like it!
- CLA for a 64-bit adder?

CLA Tree Signal Timing: d1

- Signals ready after 1 gate delay
  - C0
  - Individual G/P

CLA Tree Signal Timing: d3

- What is ready after 3 gate delays?
  - First level group G/P

A Closer Look at CLA Delay

- CLA block has “individual” G/P inputs
  - Uses them to perform two calculations
  - Group G/P on way up tree
  - Group interior carries on way down tree

- Given group carry-in from level above
  - Group carry-in for outer level (C0) ready at 0
  - Outer level G/P, interior carries in parallel
CLA Tree Signal Timing: d5

- And after 5 gate delays?
  - Outer level "interior" carries
    - $C_4, C_8, C_{12}, C_{16}$

CLA Tree Signal Timing: d7

- And after 7 gate delays?
  - First level "interior" carries
    - $C_1, C_2, C_3$
    - $C_5, C_6, C_7$
    - $C_9, C_{10}, C_{11}$
    - $C_{13}, C_{14}, C_{15}$
  - Essentially, all remaining carries
  - $S_i$ ready 2 gate delays after $C_i$
  - All sum bits ready after 9 delays!