Computer Architecture and Engineering
Lecture 6: The Design Process & ALU Design

February 3, 1995

Dave Patterson (patterson@cs) and
Shing Kong (shing.kong@eng.sun.com)

Slides available on http://http.cs.berkeley.edu/~patterson

Recap of Last Lecture

- Performance and Technology Trends
  - Keep the design simple to take advantage of the latest technology
  - CMOS inverter and CMOS logic gates
- Delay Modeling and Gate Characterization
  - Delay = Internal Delay + (Load Dependent Delay x Output Load)
- Clocking Methodology and Timing Considerations
  - Simplest clocking methodology
    - All storage elements use the SAME clock edge
  - Cycle Time = CLK-to-Q + Longest Delay Path + Setup + Clock Skew
  - CLK-to-Q + Shortest Delay Path - Clock Skew < Hold Time

Outline of Today's Lecture

- Recap of Last Lecture and Introduction of Today's Lecture (4 min.)
- An Overview of the Design Process (16 min.)
- Questions and Administrative Matters (5 min.)
- An Review of Binary Arithmetics (5 min.)
- Designing a Simple 4-bit ALU (20 min.)
- Questions and Break (5 min.)
- Other ALU Construction Techniques (5 min.)
- Keeping an On-line Design Notebook (20 min.)

The Design Process

"To Design Is To Represent"

Design activity yields description/representation of an object

-- Traditional craftsman does not distinguish between the conceptualization and the artifact

-- Separation comes about because of ***complexity***

-- The concept is captured in one or more ***representation languages***

-- This process IS design

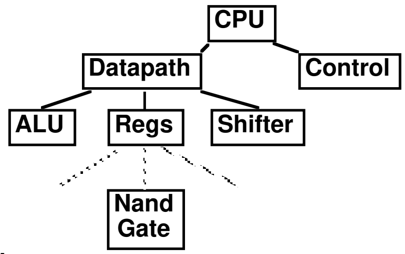
Design Begins With Requirements

-- Functional Capabilities: what it will do

-- Performance Characteristics: Speed, Power, Area, Cost, . . .

Design Process (cont.)

Design Finishes As Assembly



-- Design understood in terms of components and how they have been assembled

-- Top Down *decomposition* of complex functions (behaviors) into more primitive functions

-- bottom-up *composition* of primitive building blocks into more complex assemblies

Design is a "creative process," not a simple method

Design Refinement

Informal System Requirement

Initial Specification

Intermediate Specification

Final Architectural Description

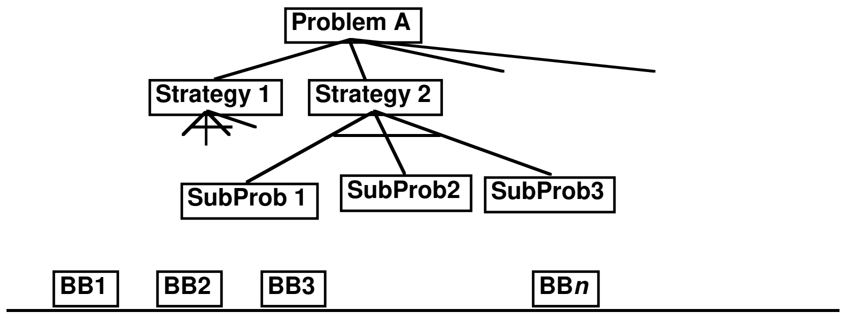
Intermediate Specification of Implementation

Final Internal Specification

Physical Implementation

refinement
increasing level of detail

Design as Search



Design involves educated guesses and verification

-- Given the goals, how should these be prioritized?

-- Given alternative design pieces, which should be selected?

-- Given design space of components & assemblies, which part will yield the best solution?

Feasible (good) choices vs. Optimal choices

Design as Representation (example)

(1) Functional Specification "VHDL Behavior"

Inputs: 2 x 16 bit operands- A, B; 1 bit carry input- Cin.

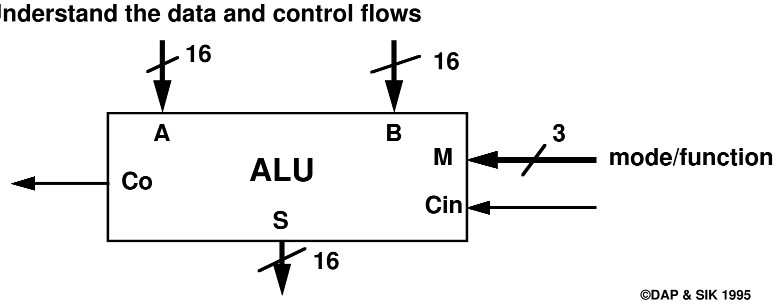
Outputs: 1 x 16 bit result- S; 1 bit carry output- Co.

Operations: PASS, ADD (A plus B plus Cin), SUB (A minus B minus Cin), AND, XOR, OR, COMPARE (equality)

Performance: left unspecified for now!

(2) Block Diagram "VHDL Entity"

Understand the data and control flows



Elements of the Design Process

- Divide and Conquer
  - Formulate a solution in terms of simpler components.
  - Design each of the components (subproblems)
- Generate and Test
  - Given a collection of building blocks, look for ways of putting them together that meets requirement
- Successive Refinement
  - Solve "most" of the problem (i.e., ignore some constraints or special cases), examine and correct shortcomings.
- Formulate High-Level Alternatives
  - Articulate many strategies to "keep in mind" while pursuing any one approach.
- Work on the Things you Know How to Do
  - The unknown will become “obvious” as you make progress.

Summary of the Design Process

Hierarchical Design to manage complexity

Top Down vs. Bottom Up vs. Successive Refinement

Importance of Design Representations:

Block Diagrams

Decomposition into Bit Slices

Truth Tables, K-Maps

Circuit Diagrams

top down ↓

bottom up ↑

mux design meets at TT

Other Descriptions: state diagrams, timing diagrams, reg xfer, . . .

Optimization Criteria:

Gate Count — *Area*

[Package Count]

Pin Out

Logic Levels

Fan-in/Fan-out

Delay

Power

Cost

Design time

Administrative Matters

- A new “tentative” schedule
  - Good News: You will have more time to work on the project after the 2nd mid-term
  - Bad News: The 2nd mid-term is moved up one week.
- After the 2nd mid-terms, we will try to arrange some guest lecturers to talk about some topics that are interesting

Introduction to Binary Numbers

- Consider a 4-bit binary number

Decimal	**Binary**	**Decimal**	**Binary**
0	0000	4	0100
1	0001	5	0101
2	0010	6	0110
3	0011	7	0111

- Examples:
  - 3 + 2 = 5

		1		
	0	0	1	1
+	0	0	1	0
	0	1	0	1

3 + 3 = 6

		1	1	
	0	0	1	1
+	0	0	1	1
	0	1	1	0

Two's Complement Representation

- 2's complement representation of negative numbers
  - Bitwise inverse and add 1
  - The MSB is always "1" for negative number => sign bit
- Biggest 4-bit Binary Number: 7 Smallest 4-bit Binary Number: -8

Decimal	Binary	Decimal	Bitwise Inverse	2's Complement
0	0000	0	1111	0000
1	0001	-1	1110	1111
2	0010	-2	1101	1110
3	0011	-3	1100	1101
4	0100	-4	1011	1100
5	0101	-5	1010	1011
6	0110	-6	1001	1010
7	0111	-7	1000	1001
$\boxed{8}$	$\boxed{1000}$	-8	0111	1000

"Illegal" Positive Number!

Two's Complement Arithmetic

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

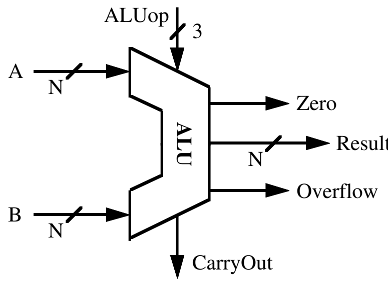
Decimal	2's Complement
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

- Examples: 7 - 6 = 7 + (- 6) = 1 3 - 5 = 3 + (- 5) = -2

	1	1	1		
		0	1	1	1
+		1	0	1	0
		0	0	0	1

		1	1	
	0	0	1	1
+	1	0	1	1
	1	1	1	0

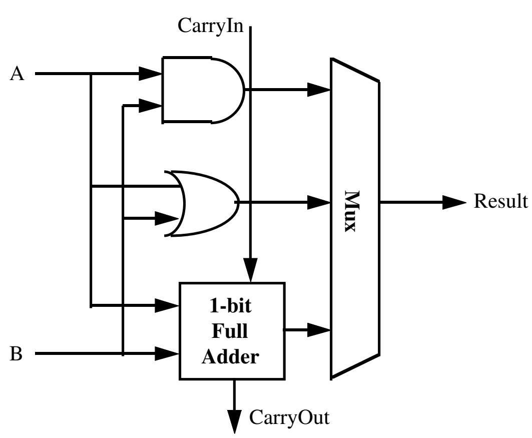
Functional Specification of the ALU



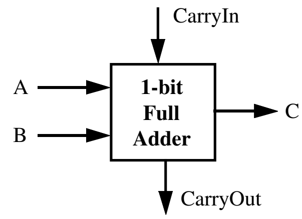
° **ALU Control Lines (ALUop)** — **Function**
- **000** — **And**
- **001** — **Or**
- **010** — **Add**
- **110** — **Subtract**
- **111** — **Set-on-less-than**

A One Bit ALU

° **This 1-bit ALU will perform AND, OR, and ADD**



A One-bit Full Adder



- This is also called a (3, 2) adder
- Half Adder: No CarryIn nor CarryOut
- Truth Table:

Inputs			Outputs		
A	**B**	**CarryIn**	**CarryOut**	**Sum**	**Comments**
0	0	0	0	0	0 + 0 + 0 = 00
0	0	1	0	1	0 + 0 + 1 = 01
0	1	0	0	1	0 + 1 + 0 = 01
0	1	1	1	0	0 + 1 + 1 = 10
1	0	0	0	1	1 + 0 + 0 = 01
1	0	1	1	0	1 + 0 + 1 = 10
1	1	0	1	0	1 + 1 + 0 = 10
1	1	1	1	1	1 + 1 + 1 = 11

Logic Equation for CarryOut

Inputs			Outputs		
A	**B**	**CarryIn**	**CarryOut**	**Sum**	**Comments**
0	0	0	0	0	0 + 0 + 0 = 00
0	0	1	0	1	0 + 0 + 1 = 01
0	1	0	0	1	0 + 1 + 0 = 01
0	1	1	1	0	0 + 1 + 1 = 10
1	0	0	0	1	1 + 0 + 0 = 01
1	0	1	1	0	1 + 0 + 1 = 10
1	1	0	1	0	1 + 1 + 0 = 10
1	1	1	1	1	1 + 1 + 1 = 11

- CarryOut = (!A & B & CarryIn) | (A & !B & CarryIn) | (A & B & !CarryIn) | (A & B & CarryIn)
- CarryOut = B & CarryIn | A & CarryIn | A & B

Logic Equation for Sum

Inputs			Outputs		
A	B	CarryIn	CarryOut	Sum	Comments
0	0	0	0	0	0 + 0 + 0 = 00
0	0	1	0	1	0 + 0 + 1 = 01
0	1	0	0	1	0 + 1 + 0 = 01
0	1	1	1	0	0 + 1 + 1 = 10
1	0	0	0	1	1 + 0 + 0 = 01
1	0	1	1	0	1 + 0 + 1 = 10
1	1	0	1	0	1 + 1 + 0 = 10
1	1	1	1	1	1 + 1 + 1 = 11

° **Sum = (!A & !B & CarryIn) | (!A & B & !CarryIn) | (A & !B & !CarryIn) | (A & B & CarryIn)**

Logic Equation for Sum (continue)

° **Sum = (!A & !B & CarryIn) | (!A & B & !CarryIn) | (A & !B & !CarryIn) | (A & B & CarryIn)**

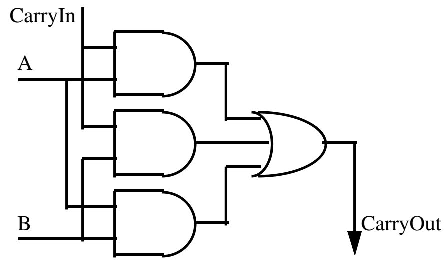
° **Sum = A XOR B XOR CarryIn**

° **Truth Table for XOR:**

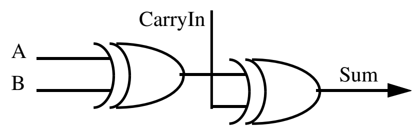
X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

Logic Diagrams for CarryOut and Sum

° **CarryOut = B & CarryIn | A & CarryIn | A & B**

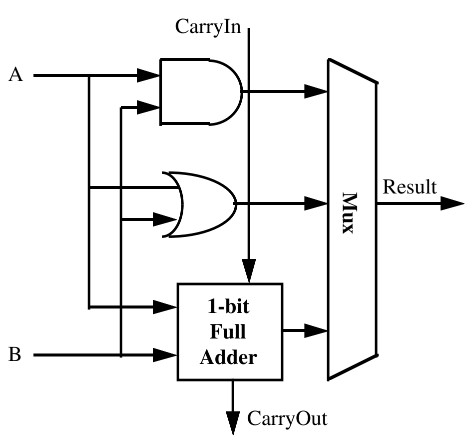


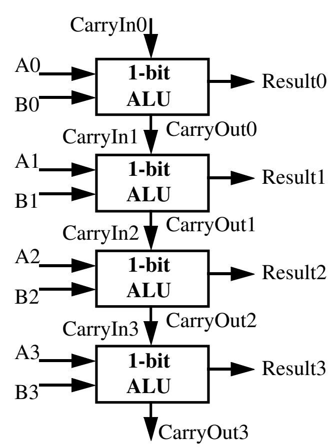
° **Sum = A XOR B XOR CarryIn**



A 4-bit ALU

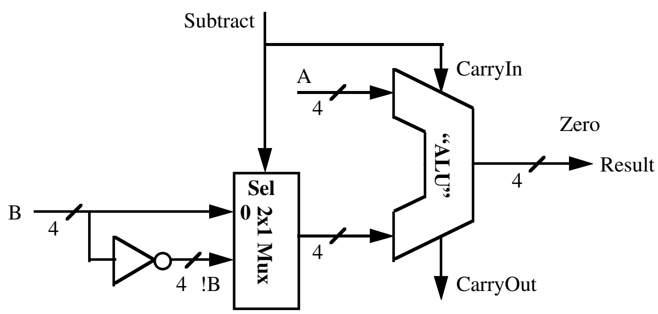
° **1-bit ALU** **4-bit ALU**





How About Subtraction?

- Keep in mind the followings:
  - (A - B) is the that as: A + (-B)
  - 2's Complement: Take the inverse of every bit and add 1
- Bit-wise inverse of B is !B:
  - A + !B + 1 = A + (!B + 1) = A + (-B) = A - B



Overflow

Decimal	Binary	Decimal	2's Complement
0	0000	0	0000
1	0001	-1	1111
2	0010	-2	1110
3	0011	-3	1101
4	0100	-4	1100
5	0101	-5	1011
6	0110	-6	1010
7	0111	-7	1001
		-8	1000

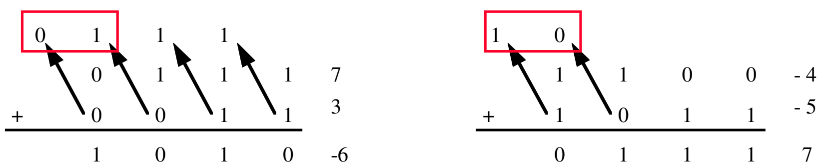
- Examples: 7 + 3 = 10 but ... - 4 - 5 = - 9 but ...

	0	1	1	1	
	0	1	1	1	7
+	0	0	1	1	3
	1	0	1	0	- 6

	1				
	1	1	0	0	- 4
+	1	0	1	1	- 5
	0	1	1	1	7

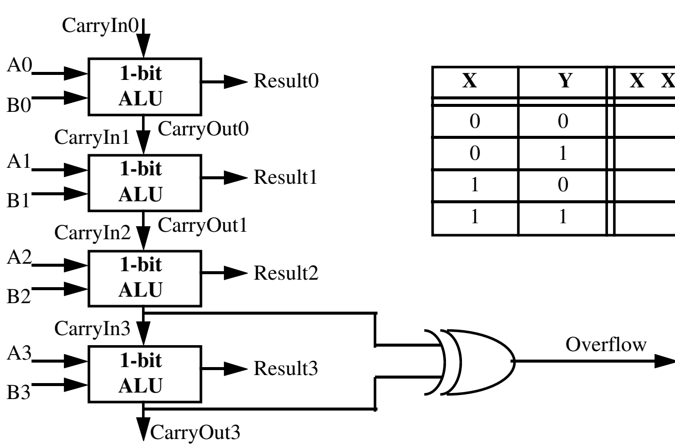
Overflow Detection

- Overflow: the result is too large (or too small) to represent properly
  - Example: - 8 < = 4-bit binary number <= 7
- When adding operands with different signs, overflow cannot occur!
- Overflow occurs when adding:
  - 2 positive numbers and the sum is negative
  - 2 negative numbers and the sum is positive
- Homework exercise: Prove you can detect overflow by:
  - Carry into MSB ! = Carry out of MSB



Overflow Detection Logic

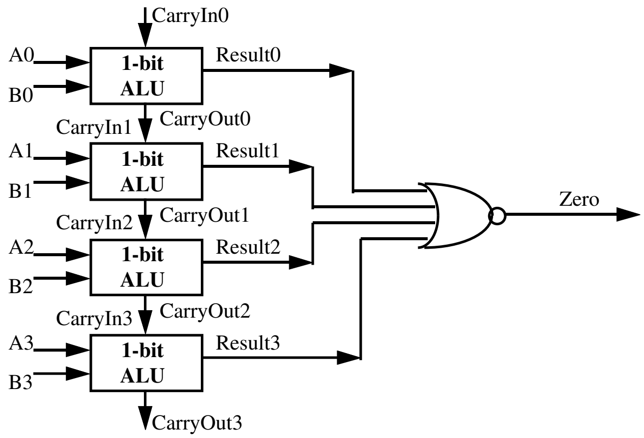
- Carry into MSB ! = Carry out of MSB
  - For a N-bit ALU: Overflow = CarryIn[N - 1] XOR CarryOut[N - 1]



X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

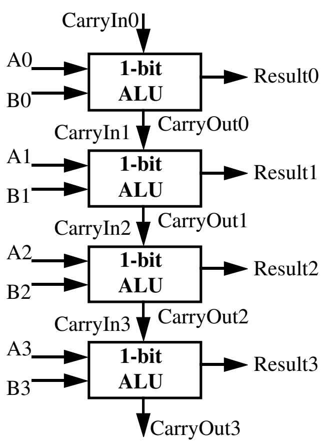
Zero Detection Logic

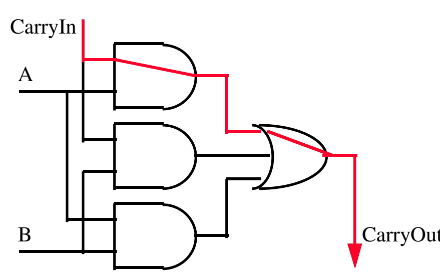
- Zero Detection Logic is just a one BIG NOT gate
  - Any non-zero input to the NOR gate will cause its output to be zero



The Disadvantage of Ripple Carry

- The adder we just built is called a "Ripple Carry Adder"
  - The carry bit may have to propagate from LSB to MSB
  - Worst case delay for a N-bit adder: 2N-gate delay





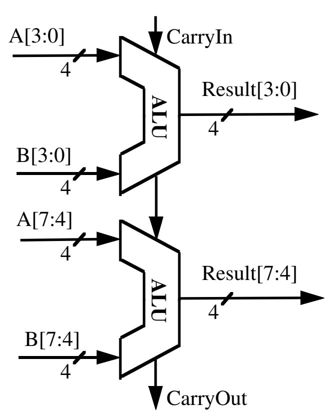
Break

° **5-minute Break**

Carry Select Header

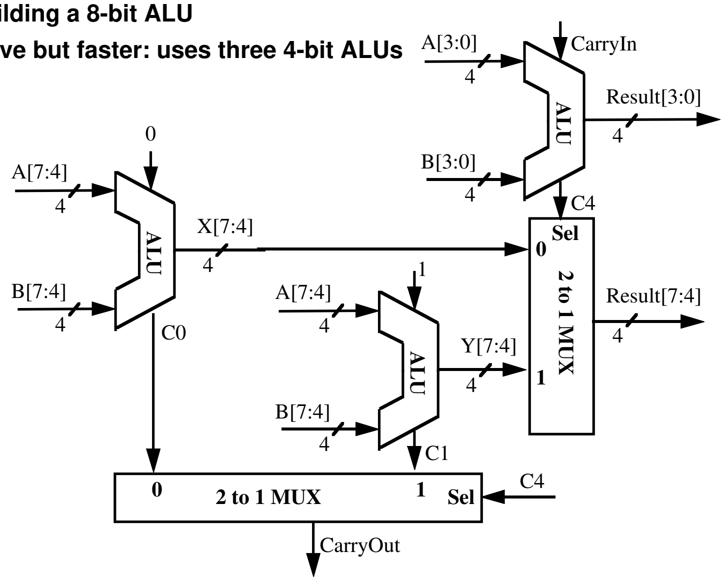
° **Consider building a 8-bit ALU**

- **Simple: connects two 4-bit ALUs in series**

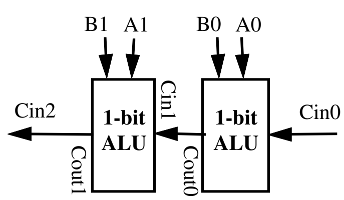


Carry Select Header (Continue)

- Consider building a 8-bit ALU
  - Expensive but faster: uses three 4-bit ALUs



The Theory Behind Carry Lookahead



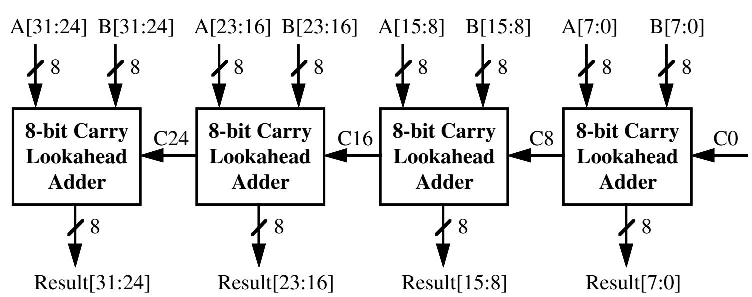
- Recalled: CarryOut = (B & CarryIn) | (A & CarryIn) | (A & B)
  - Cin2 = Cout1 = (B1 & Cin1) | (A1 & Cin1) | (A1 & B1)
  - Cin1 = Cout0 = (B0 & Cin0) | (A0 & Cin0) | (A0 & B0)
- Substituting Cin1 into Cin2:
  - Cin2 = (A1 & A0 & B0) | (A1 & A0 & Cin0) | (A1 & B0 & Cin0) | (B1 & A0 & B0) | (B1 & A0 & Cin0) | (B1 & A0 & Cin0) | (A1 & B1)
- Now define two new terms:
  - Generate Carry at Bit i gi = Ai & Bi
  - Propagate Carry via Bit i

The Theory Behind Carry Lookahead (Continue)

- Using the two new terms we just defined:
  - Generate Carry at Bit i   gi = Ai & Bi
  - Propagate Carry via Bit i   pi = Ai or Bi
- We can rewrite:
  - Cin1 = g0 | (p0 & Cin0)
  - Cin2 = g1 | (p1 & g0) | (p1 & p0 & Cin0)
  - Cin3 = g2 | (p2 & g1) | (p2 & p1 & g0) | (p2 & p1 & p0 & Cin0)
- Carry going into bit 3 is 1 if
  - We generate a carry at bit 2 (g2)
  - Or we generate a carry at bit 1 (g1) and bit 2 allows it to propagate (p2 & g1)
  - Or we generate a carry at bit 0 (g0) and bit 1 as well as bit 2 allows it to propagate (p2 & p1 & g0)
  - Or we have a carry input at bit 0 (Cin0) and bit 0, 1, and 2 all allow it to propagate (p2 & p1 & p0 & Cin0)

A Partial Carry Lookahead Adder

- It is very expensive to build a "full" carry lookahead adder
  - Just imagine the length of the equation for Cin31
- Common practices:
  - Connects several N-bit Lookahead Adders to form a big adder
  - Example: connects four 8-bit carry lookahead adders to form a 32-bit partial carry lookahead adder



Why should you keep an design notebook?

- Keep track of the design decisions and the reasons behind them
  - Otherwise, it will be hard to debug and/or refine the design
- Insights you have on certain aspect of the design
- Results of the different design & debug experiments

Why do we keep it on-line?

- You need to force yourself to take notes
  - Open an widow and leave an editor running while you work
- Take advantage of the window system's "cut and paste" features
- It is much easier to read your typing than your writing

How should you do it?

- Keep it simple
- Separate the entries by dates
- Index: write a one-line summary of what you did each day

On-line Notebook Example

- Refer to the handout

Summary

- An Overview of the Design Process
  - Design is an iterative process-- successive refinement
  - Do NOT wait until you know everything before you start
- An Introduction to Binary Arithmetics
  - If you use 2's complement representation, subtract is easy.
- ALU Design
  - Designing a Simple 4-bit ALU
  - Other ALU Construction Techniques
- On-line Design Notebook
  - Open a window and keep an editor running while you work
  - Refer to the handout as an example

To Get More Information

- Chapter 4 of your text book:
  - David Patterson & John Hennessy, "Computer Organization & Design," Morgan Kaufmann Publishers, 1994.
- A book I really like:
  - David Winkel & Franklin Prosser, "The Art of Digital Design: An Introduction to Top-Down Design," Prentice-Hall, Inc., 1980.
- My master thesis has a chapter on carry lookahead adder design:
  - Shing Kong, "Some Design Techniques for High Performance MOS Circuits," Master Report, EECS Department, UC Berkeley, 1985.