CS152
Computer Architecture and Engineering
Lecture 4

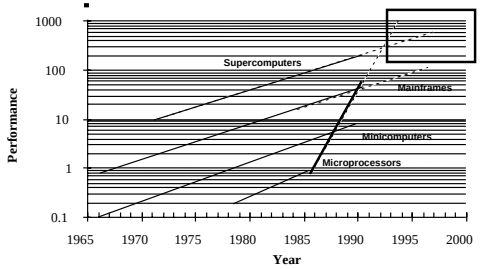
Cost and Design

Feb 3, 1999

John Kubiatowicz (http.cs.berkeley.edu/~kubitron)

lecture slides: http://www-inst.eecs.berkeley.edu/~cs152/

Review: Performance and Technology Trends



° **Technology Power: 1.2 x 1.2 x 1.2 = 1.7 x / year**
- **Feature Size: shrinks 10% / yr. => Switching speed improves 1.2 / yr.**
- **Density: improves 1.2x / yr.**
- **Die Area: 1.2x / yr.**

° **RISC lesson is to keep the ISA as simple as possible:**
- **Shorter design cycle => fully exploit the advancing technology (~3yr)**
- **Advanced branch prediction and pipeline techniques**
- **Bigger and more sophisticated on-chip caches**

Review: Technology, Logic Design and Delay

° **CMOS Technology Trends**
- **Complementary: PMOS and NMOS transistors**
- **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**

Overview: Cost and Design

° **Review from Last Lecture (2 minutes)**

° **Cost and Price (18)**

° **Administrative Matters (3 minutes)**

° **Design process (27 minutes)**

° **Break (5 minutes)**

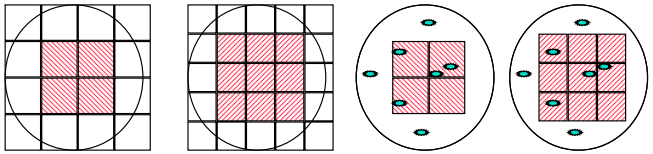
° **More Design process (15 minutes)**

° **Online notebook (10 minutes)**

Integrated Circuit Costs

$$\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per Wafer} * \text{Die yield}}$$

$$\text{Dies per wafer} = \frac{\pi * (\text{Wafer\_diam} / 2)^2}{\text{Die Area}} - \frac{\pi * \text{Wafer\_diam}}{\sqrt{2 * \text{Die Area}}} - \text{Test dies} \approx \frac{\text{Wafer Area}}{\text{Die Area}}$$



$$\text{Die Yield} = \frac{\text{Wafer yield}}{\left\{ 1 + \frac{\text{Defects\_per\_unit\_area} * \text{Die\_Area}}{\alpha} \right\}^{\alpha}}$$

Die Cost is goes roughly with the cube of the area.

Die Yield

Raw Dice Per Wafer

wafer diameter	*die area (mm²)* 100	144	196	256	324	400
6”/15cm	139	90	62	44	32	23
8”/20cm	265	177	124	90	68	52
10”/25cm	431	290	206	153	116	90
die yield	23%	19%	16%	12%	11%	10%

typical CMOS process: α =2, wafer yield=90%, defect density=2/cm2, 4 test sites/wafer

Good Dice Per Wafer (Before Testing!)

6”/15cm	31	16	9	5	3	2
8”/20cm	59	32	19	11	7	5
10”/25cm	96	53	32	20	13	9

typical cost of an 8”, 4 metal layers, 0.5um CMOS wafer: ~$2000

Real World Examples

Chip	*Metal layers*	*Line width*	*Wafer cost*	*Defect /cm²*	*Area mm²*	*Dies/ wafer*	*Yield*	*Die Cost*
386DX	2	0.90	$900	1.0	43	360	71%	$4
486DX2	3	0.80	$1200	1.0	81	181	54%	$12
PowerPC 601	4	0.80	$1700	1.3	121	115	28%	$53
HP PA 7100	3	0.80	$1300	1.0	196	66	27%	$73
DEC Alpha	3	0.70	$1500	1.2	234	53	19%	$149
SuperSPARC	3	0.70	$1700	1.6	256	48	13%	$272
Pentium	3	0.80	$1500	1.5	296	40	9%	$417

From "Estimating IC Manufacturing Costs,” by Linley Gwennap, *Microprocessor Report*, August 2, 1993, p. 15

Other Costs

$$\text{IC cost} = \frac{\text{Die cost} + \text{Testing cost} + \text{Packaging cost}}{\text{Final test yield}}$$

Packaging Cost: depends on pins, heat dissipation

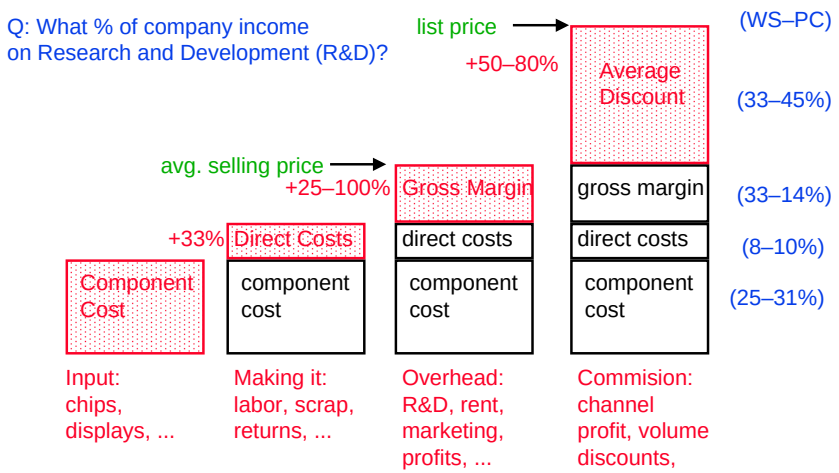
Chip	*Die cost*	*Package pins*	*Package type*	*Package cost*	*Test & Assembly*	*Total*
386DX	$4	132	QFP	$1	$4	$9
486DX2	$12	168	PGA	$11	$12	$35
PowerPC 601	$53	304	QFP	$3	$21	$77
HP PA 7100	$73	504	PGA	$35	$16	$124
DEC Alpha	$149	431	PGA	$30	$23	$202
SuperSPARC	$272	293	PGA	$20	$34	$326
Pentium	$417	273	PGA	$19	$37	$473

System Cost: -1995-96 Workstation

System	Subsystem	% of total cost
Cabinet	Sheet metal, plastic	1%
	Power supply, fans	2%
	Cables, nuts, bolts	1%
	(Subtotal)	(4%)
Motherboard	Processor	6%
	DRAM (64MB)	36%
	Video system	14%
	I/O system	3%
	Printed Circuit board	1%
	(Subtotal)	(60%)
I/O Devices	Keyboard, mouse	1%
	Monitor	22%
	Hard disk (1 GB)	7%
	Tape drive (DAT)	6%
	(Subtotal)	(36%)

Cost vs. Price

Q: What % of company income on Research and Development (R&D)?



Cost Summary

- Integrated circuits driving computer industry
- Die costs goes up with the cube of die area
- Economics ($$$) is the ultimate driver for performance!

Administrative Matters

- Review complete: did ok on prob 1 & 4.
  Problems 2 and 3 more challenging
  Make sure you look at solutions!
- Read Chapter 4: ALU, Multiply, Divide, FP Mult
- Dollars for bugs! First to report bug gets $1 check
  - Send 1 bug/ email to `mkp@mkp.com`
  - Include page number, orginal text, why bug, fixed text

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

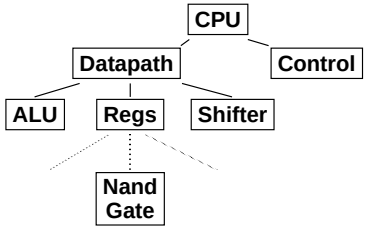
Design Begins With Requirements

- -- Functional Capabilities: what it will do
- -- Performance Characteristics: Speed, Power, Area, Cost, . . .

Design Process (cont.)

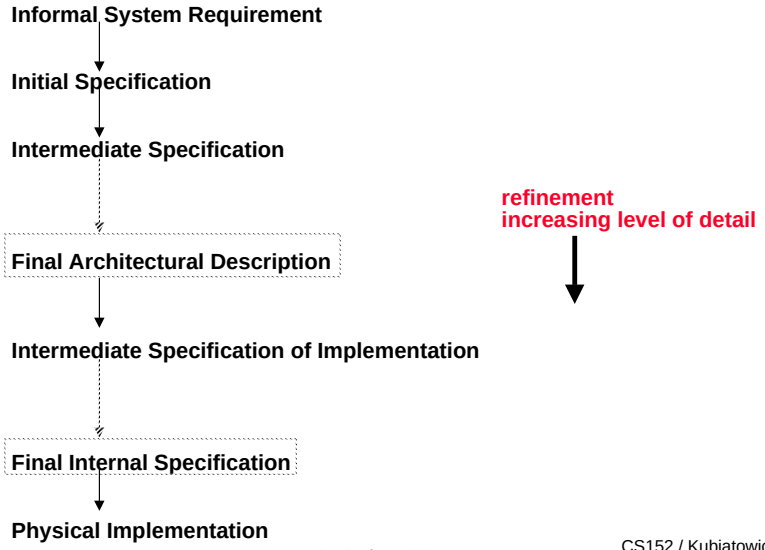
Design Finishes As Assembly

- -- 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

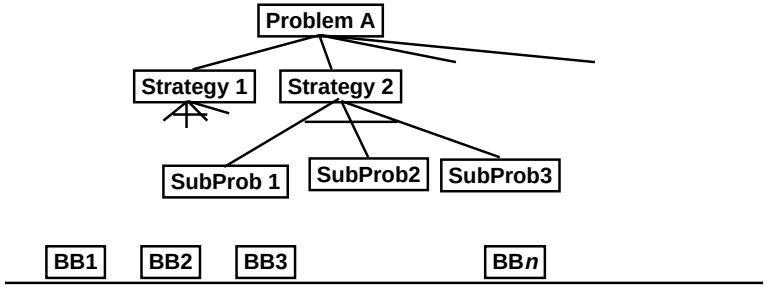


Design is a "creative process," not a simple method

Design Refinement



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

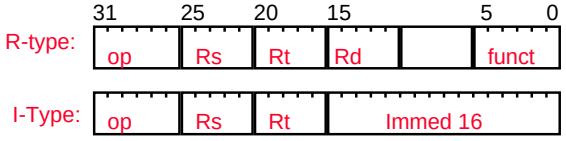
Problem: Design a "fast" ALU for the MIPS ISA

- Requirements?
- Must support the Arithmetic / Logic operations
- Tradeoffs of cost and speed based on frequency of occurrence, hardware budget

MIPS ALU requirements

- Add, AddU, Sub, SubU, AddI, AddIU
  - => 2's complement adder/sub with overflow detection
- And, Or, AndI, OrI, Xor, Xori, Nor
  - => Logical AND, logical OR, XOR, nor
- SLTI, SLTIU (set less than)
  - => 2's complement adder with inverter, check sign bit of result
- ALU from from CS 150 / P&H book chapter 4 supports these ops

MIPS arithmetic instruction format



Type	op	funct
ADDI	10	xx
ADDIU	11	xx
SLTI	12	xx
SLTIU	13	xx
ANDI	14	xx
ORI	15	xx
XORI	16	xx
LUI	17	xx

Type	op	funct
ADD	00	40
ADDU	00	41
SUB	00	42
SUBU	00	43
AND	00	44
OR	00	45
XOR	00	46
NOR	00	47

Type	op	funct
	00	50
	00	51
SLT	00	52
SLTU	00	53

- Signed arith generate overflow, no carry

Design Trick: divide & conquer

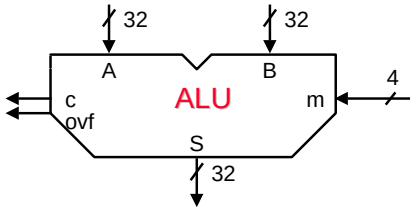
- Break the problem into simpler problems, solve them and glue together the solution
- Example: assume the immediates have been taken care of before the ALU
  - 10 operations (4 bits)

00	add
01	addU
02	sub
03	subU
04	and
05	or
06	xor
07	nor
12	slt
13	sltU

Refined Requirements

(1) Functional Specification
inputs: 2 x 32-bit operands A, B, 4-bit mode
outputs: 32-bit result S, 1-bit carry, 1 bit overflow
operations: add, addu, sub, subu, and, or, xor, nor, slt, sltU

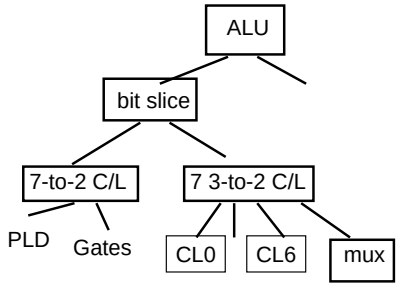
(2) Block Diagram (powerview symbol, VHDL entity)



Behavioral Representation: VHDL

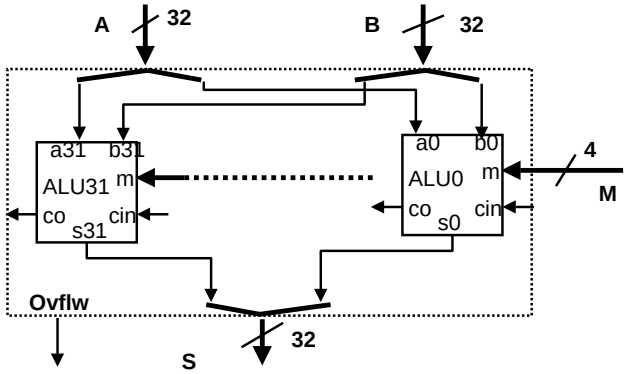
```
Entity ALU is
   generic (c_delay: integer := 20 ns;
            S_delay: integer := 20 ns);

   port ( signal A, B:  in  vlbit_vector (0 to 31);
          signal    m:  in  vlbit_vector (0 to 3);
          signal    S: out  vlbit_vector (0 to 31);
          signal    c: out  vlbit;
          signal  ovf: out  vlbit)
end ALU;

...

      S <= A + B;
```


Design Decisions



- **Simple bit-slice**
  - **big combinational problem**
  - **many little combinational problems**
  - **partition into 2-step problem**
- **Bit slice with carry look-ahead**
- **. . .**

Refined Diagram: bit-slice ALU



7-to-2 Combinational Logic

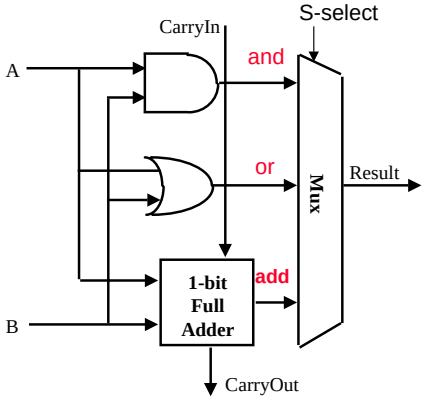
° start turning the crank . . .

	Function	Inputs	Outputs	K-Map
		M0 M1 M2 M3 A B Cin	S Cout	
0	add	0 0 0 0 0 0 0	0 0	
127				

Seven plus a MUX ?

° Design trick 2: take pieces you know (or can imagine) and try to put them together

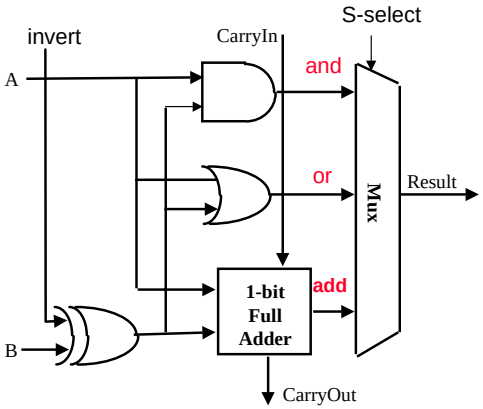
° Design trick 3: solve part of the problem and extend



Additional operations

° $A - B = A + (-B) = A + \overline{B} + 1$

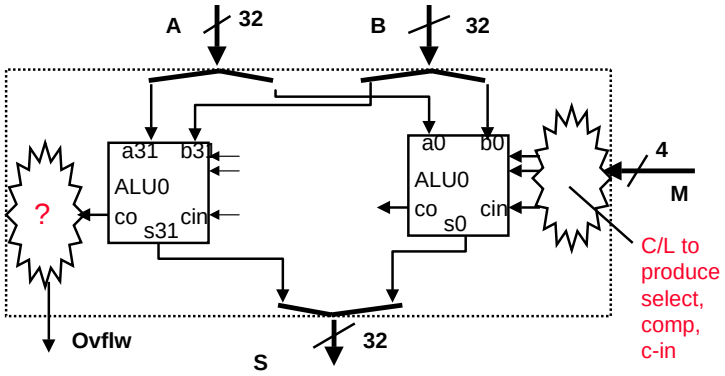
- form two complement by invert and add one



Set-less-than? – left as an exercise

Revised Diagram

° LSB and MSB need to do a little extra



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**

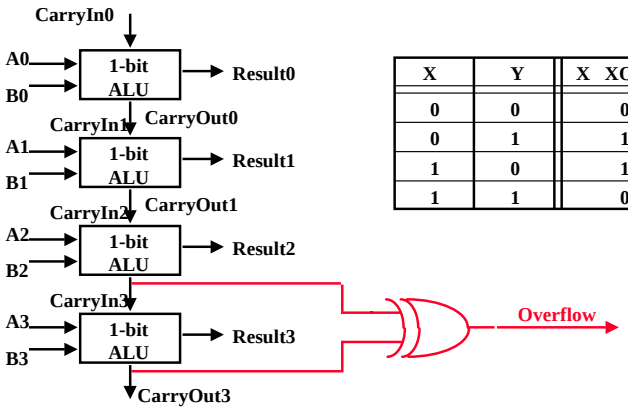
° **On your own: Prove you can detect overflow by:**
- **Carry into MSB ≠ Carry out of MSB**

```
  0 1 1 1
    0 1 1 1   7
+   0 0 1 1   3
  -----------
    1 0 1 0   – 6
```

```
  1 0
    1 1 0 0   –4
+   1 0 1 1   – 5
  -----------
    0 1 1 1   7
```


Overflow Detection Logic

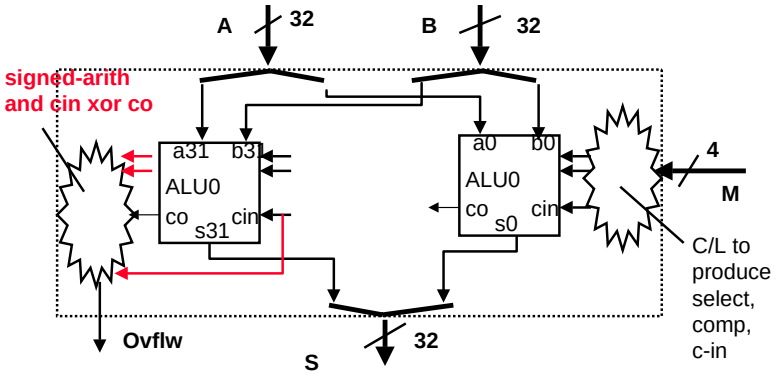
° **Carry into MSB ≠ Carry out of MSB**
- **For a N-bit ALU: Overflow = CarryIn[N - 1] XOR CarryOut[N - 1]**



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

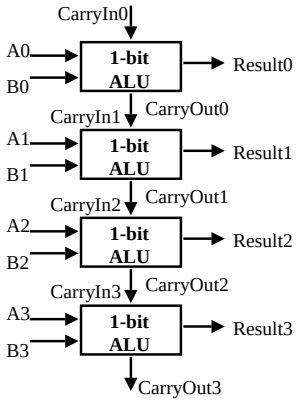
More Revised Diagram

° **LSB and MSB need to do a little extra**



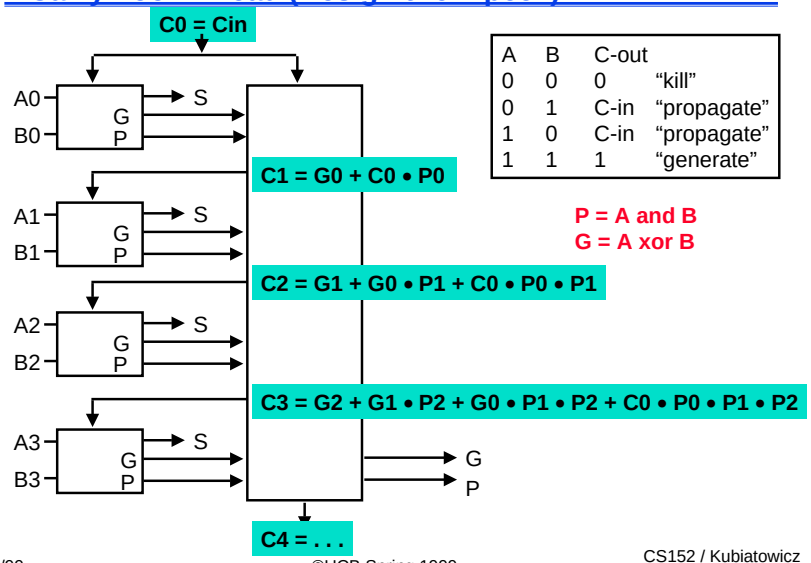
But What about Performance?

° Critical Path of n-bit Rippled-carry adder is n*CP



Design Trick:
Throw hardware at it

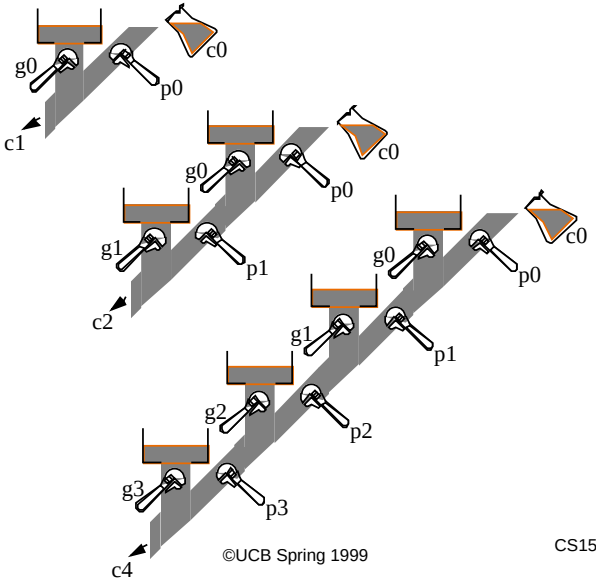
Carry Look Ahead (Design trick: peek)



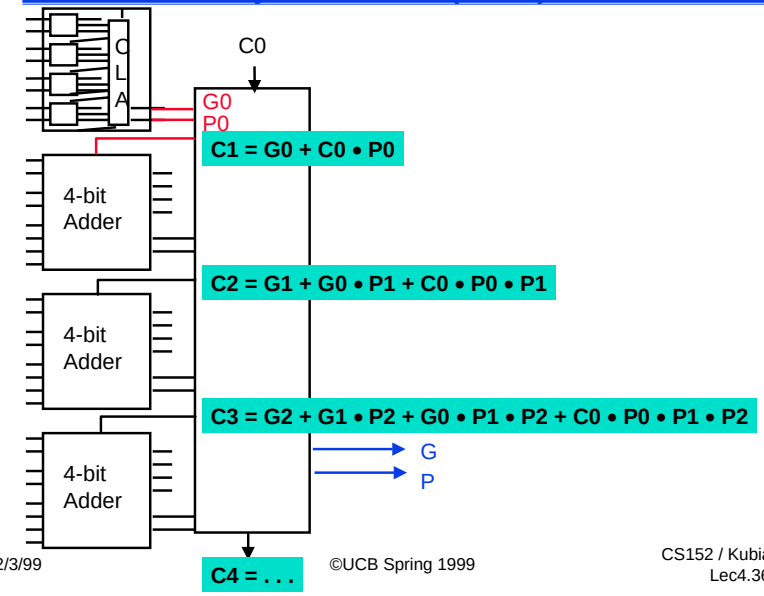
A	B	C-out	
0	0	0	"kill"
0	1	C-in	"propagate"
1	0	C-in	"propagate"
1	1	1	"generate"

P = A and B
G = A xor B

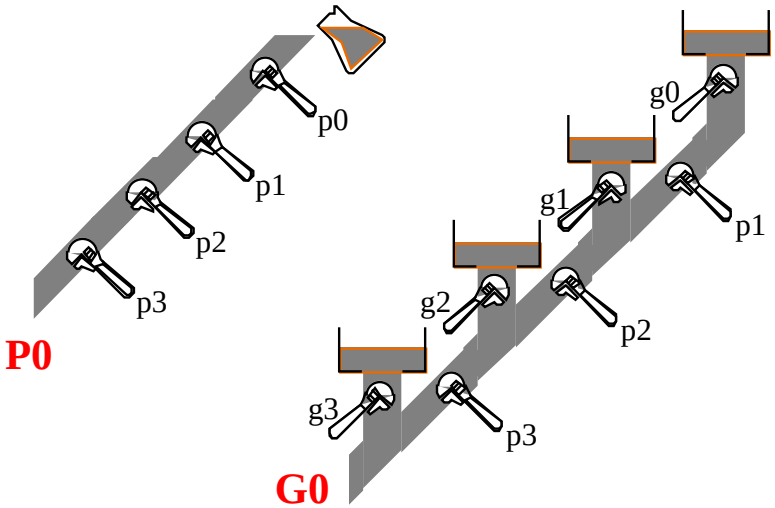
Plumbing as Carry Lookahead Analogy



Cascaded Carry Look-ahead (16-bit): Abstraction

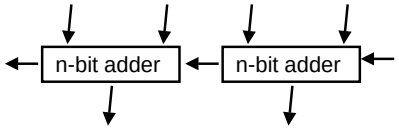


2nd level Carry, Propagate as Plumbing

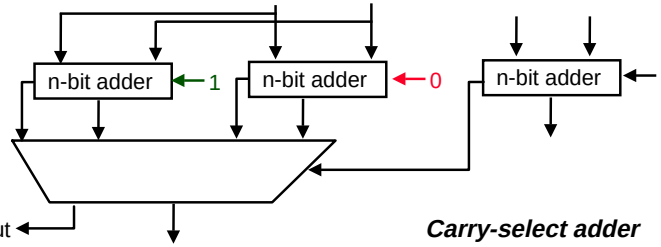


Design Trick: Guess (or "Precompute")

CP(2n) = 2*CP(n)

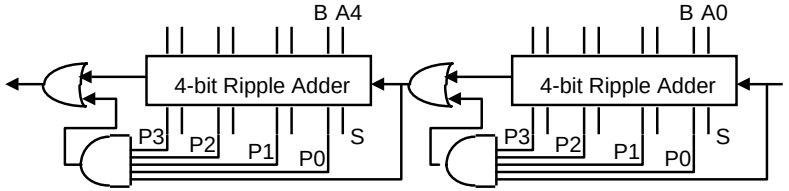


CP(2n) = CP(n) + CP(mux)



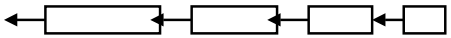
Carry-select adder

Carry Skip Adder: reduce worst case delay



Just speed up the slowest case for each block

Exercise: optimal design uses variable block sizes



Additional MIPS ALU requirements

- **Mult, MultU, Div, DivU (next lecture)**
  => Need 32-bit multiply and divide, signed and unsigned
- **Sll, Srl, Sra (next lecture)**
  => Need left shift, right shift, right shift arithmetic by 0 to 31 bits
- **Nor (leave as exercise to reader)**
  => logical NOR or use 2 steps: (A OR B) XOR 1111....1111

Elements of the Design Process

- Divide and Conquer (e.g., ALU)
  - Formulate a solution in terms of simpler components.
  - Design each of the components (subproblems)
- Generate and Test (e.g., ALU)
  - Given a collection of building blocks, look for ways of putting them together that meets requirement
- Successive Refinement (e.g., carry lookahead)
  - Solve "most" of the problem (i.e., ignore some constraints or special cases), examine and correct shortcomings.
- Formulate High-Level Alternatives (e.g., carry select)
  - 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

Why should you keep an design notebook?

- Keep track of the design decisions <u>and the reasons behind them</u>
  - Otherwise, it will be hard to debug and/or refine the design
  - Write it down so that can remember in long project: 2 weeks ->2 yrs
  - Others can review notebook to see what happened
- Record insights you have on certain aspect of the design as they come up
- Record of the different design & debug experiments
  - Memory can fail when very tired
- Industry practice: learn from others mistakes

Why do we keep it on-line?

- You need to force yourself to take notes
  - Open a window and leave an editor running while you work
    - 1) Acts as reminder to take notes
    - 2) Makes it easy to take notes
  - 1) + 2) => will actually do it
- <u>Take advantage of the window system’s “cut and paste” features</u>
- It is much easier to read your typing than your writing
- Also, paper log books have problems
  - Limited capacity => end up with many books
  - May not have right book with you at time vs. networked screens
  - Can use computer to search files/index files to find what looking for

How should you do it?

- Keep it simple
  - DON'T make it so elaborate that you won't use (fonts, layout, ...)
- Separate the entries by dates
  - type "`date`" command in another window and cut&paste
- Start day with problems going to work on today
- Record output of simulation into log with cut&paste; add date
  - May help sort out which version of simulation did what
- Record key email with cut&paste
- Record of what works & doesn't helps team decide what went wrong after you left
- Index: write a one-line summary of what you did at end of each day

On-line Notebook Example

- Refer to the handout "Example of On-Line Log Book" on cs 152 home page

1st page of On-line notebook (Index + Wed. 9/6/95)

```
* Index ==============================================================

Wed Sep  6 00:47:28 PDT 1995 - Created the 32-bit comparator component
Thu Sep  7 14:02:21 PDT 1995 - Tested the comparator
Mon Sep 11 12:01:45 PDT 1995 - Investigated bug found by Bart in
                               comp32 and fixed it
+ ====================================================================
Wed Sep  6 00:47:28 PDT 1995

  Goal: Layout the schematic for a 32-bit comparator

  I've layed out the schemtatics and made a symbol for the comparator.
  I named it comp32.  The files are
      ~/wv/proj1/sch/comp32.sch
      ~/wv/proj1/sch/comp32.sym

Wed Sep  6 02:29:22 PDT 1995
- ====================================================================
```

- Add 1 line index at front of log file at end of each session: date+summary
- Start with date, time of day + goal
- Make comments during day, summary of work
- End with date, time of day (and add 1 line summary at front of file)

2nd page of On-line notebook (Thursday 9/7/95)

```
+ ====================================================================
Thu Sep  7 14:02:21 PDT 1995

  Goal: Test the comparator component

  I've written a command file to test comp32.  I've placed it
  in ~/wv/proj1/diagnostics/comp32.cmd.

  I ran the command file in viewsim and it looks like the comparator
  is working fine.  I saved the output into a log file called
  ~/wv/proj1/diagnostics/comp32.log

  Notified the rest of the group that the comparator
  is done.

Thu Sep  7 16:15:32 PDT 1995
- ====================================================================
```

3rd page of On-line notebook (Monday 9/11/95)

```
+ ==================================================================
Mon Sep 11 12:01:45 PDT 1995

  Goal: Investigate bug discovered in comp32 and hopefully fix it

  Bart found a bug in my comparator component. He left the following
  e-mail.

-------------------
From bart@simpsons.residence Sun Sep 10 01:47:02 1995
Received: by wayne.manor (NX5.67e/NX3.0S)
        id AA00334; Sun, 10 Sep 95 01:47:01 -0800
Date: Wed, 10 Sep 95 01:47:01 -0800
From: Bart Simpson <bart@simpsons.residence>
To: bruce@wanye.manor, old_man@gokuraku, hojo@sanctuary
Subject: [cs152] bug in comp32
Status: R

Hey Bruce,
I think there's a bug in your comparator.
The comparator seems to think that ffffffff and fffffff7 are equal.

Can you take a look at this?
Bart
----------------
```


4th page of On-line notebook (9/11/95 contd)

```
I verified the bug. here's a viewsim of the bug as it appeared..
  (equal should be 0 instead of 1)
-------------------
SIM>stepsize 10ns
SIM>v a_in A[31:0]
SIM>v b_in B[31:0]
SIM>w a_in b_in equal
SIM>a a_in ffffffff\h
SIM>a b_in fffffff7\h
SIM>sim
time =    10.0ns  A_IN=FFFFFFFF\H B_IN=FFFFFFF7\H EQUAL=1
Simulation stopped at 10.0ns.
-------------------

  Ah. I've discovered the bug. I mislabeled the 4th net in
  the comp32 schematic.

  I corrected the mistake and re-checked all the other
  labels, just in case.

  I re-ran the old diagnostic test file and tested it against
  the bug Bart found.  It seems to be working fine. hopefully
  there aren't any more bugs:)
```

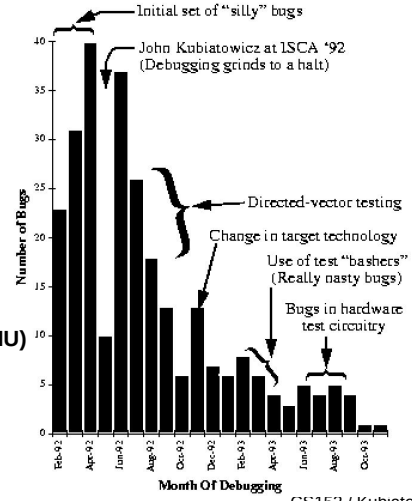

5th page of On-line notebook (9/11/95 contd)

```
  On second inspectation of the whole layout, I think I can
  remove one level of gates in the design and make it go faster.
  But who cares! the comparator is not in the critical path
  right now. the delay through the ALU is dominating the critical
  path.  so unless the ALU gets a lot faster, we can live with
  a less than optimal comparator.

  I e-mailed the group that the bug has been fixed

Mon Sep 11 14:03:41 PDT 1995
- ==================================================================
```

- Perhaps later critical path changes;
what was idea to make compartor faster? Check log book!

Added benefit: cool post-design statistics

Sample graph from the Alewife project:



- **For the Communications and Memory Management Unit (CMMU)**
- **These statistics came from on-line record of bugs**

Lecture Summary

- Cost and Price
  - Die size determines chip cost: cost - die size$^{(\alpha+1)}$
  - Cost v. Price: business model of company, pay for engineers
  - R&D must return $8 to $14 for every $1 invester
- An Overview of the Design Process
  - Design is an iterative process, multiple approaches to get started
  - Do NOT wait until you know everything before you start
- Example: Instruction Set drives the ALU design
- On-line Design Notebook
  - Open a window and keep an editor running while you work;cut&paste
  - Refer to the handout as an example
  - Former CS 152 students (and TAs) say they use on-line notebook for programming as well as hardware design; one of most valuable skills