CS152
Computer Architecture and Engineering
Lecture 6
Multiply, Divide, Shift

February 12, 2003

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

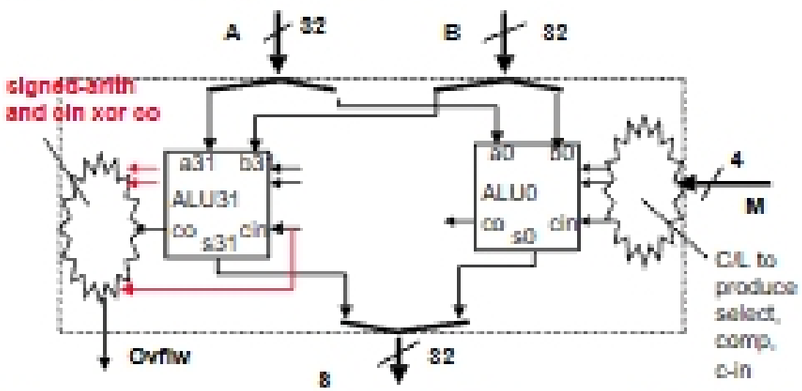
lecture slides: http://www-inst.eecs.berkeley.edu/~cs152/

Review: Elements of the Design Process

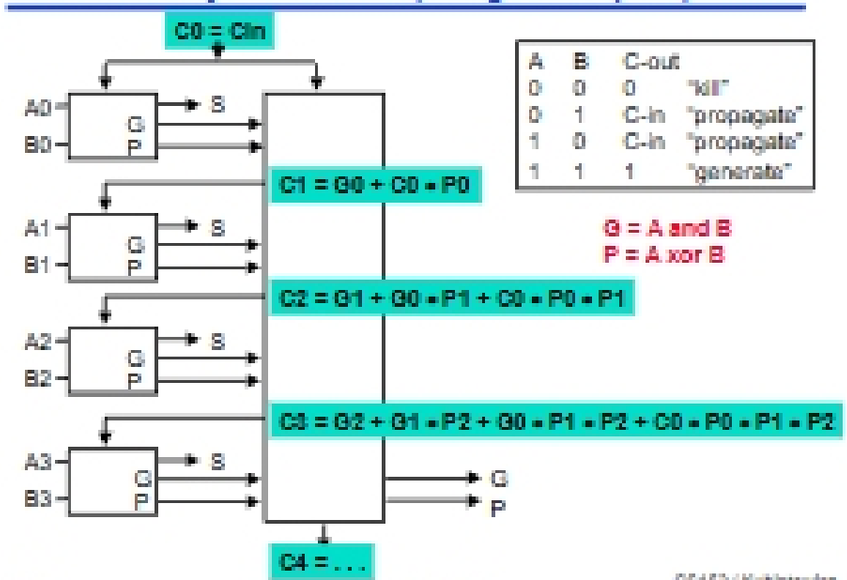
- Divide and Conquer (e.g., ALU)
  - Formulate a solution in terms of simpler components.
  - Design each of the components (subproblems)
- Generate and Test (e.g., ALU)
  - Given a collection of building blocks, look for ways of putting them together that meets requirement
- Successive Refinement (e.g.,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

Review: ALU Design

- Bit-slice plus extra on the two ends
- Overflow means number too large for the representation
- Carry-look ahead and other adder tricks

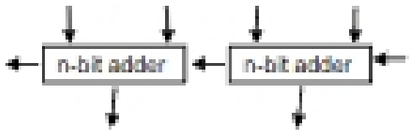


Review: Carry Look Ahead (Design trick: peek)

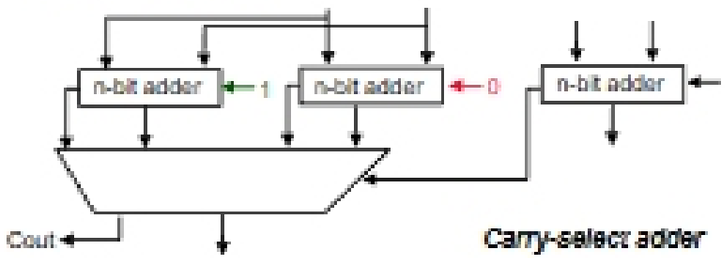


Review: Design Trick: Guess (or "Precompute")

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

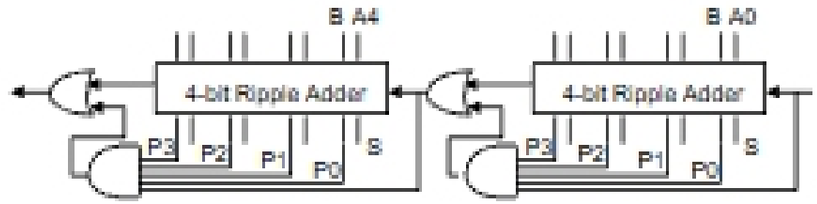


CP(2n) = CP(n) + CP(mux)



Carry-select adder

Review: Carry Skip Adder: reduce worst case delay



Just speed up the slowest case for each block

Exercise: optimal design uses variable block sizes

MIPS arithmetic instructions

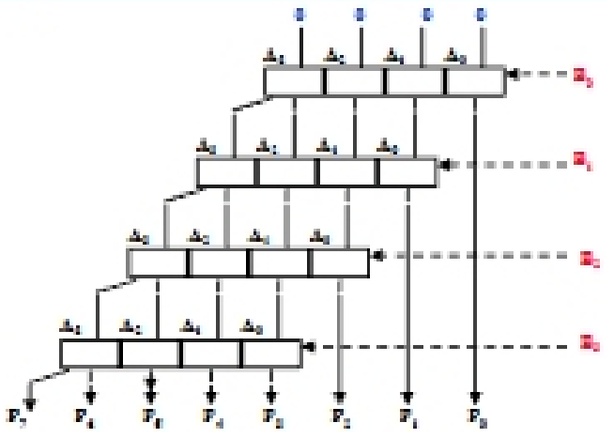
Instruction	*Example*	*Meaning*	*Comments*
add	add $1,$2,$3	$1 = $2 + $3	3 operands; exception possible
subtract	sub $1,$2,$3	$1 = $2 – $3	3 operands; exception possible
add immediate	addi $1,$2,100	$1 = $2 + 100	+ constant; exception possible
add unsigned	addu $1,$2,$3	$1 = $2 + $3	3 operands; no exceptions
subtract unsigned	subu $1,$2,$3	$1 = $2 – $3	3 operands; no exceptions
add imm. unsign.	addiu $1,$2,100	$1 = $2 + 100	+ constant; no exceptions
multiply	mult $2,$3	Hi, Lo = $2 x $3	64-bit signed product
multiply unsigned	multu$2,$3	Hi, Lo = $2 x $3	64-bit unsigned product
divide	div $2,$3	Lo = $2 – $3, Hi = $2 mod $3	Lo = quotient, Hi = remainder
divide unsigned	divu $2,$3	Lo = $2 – $3, Hi = $2 mod $3	Unsigned quotient & remainder
Move from Hi	mfhi $1	$1 = Hi	Used to get copy of Hi
Move from Lo	mflo $1	$1 = Lo	Used to get copy of Lo

MULTIPLY (unsigned)

- **Paper and pencil example (unsigned):**

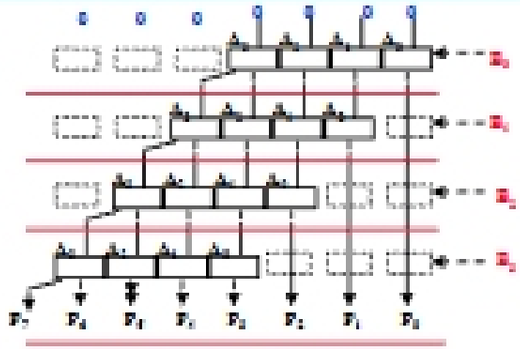
Multiplicand

Unsigned Combinational Multiplier



- Stage i accumulates A * 2^i if B_i == 1
- Q: How much hardware for 32 bit multiplier? Critical path?

How does it work?



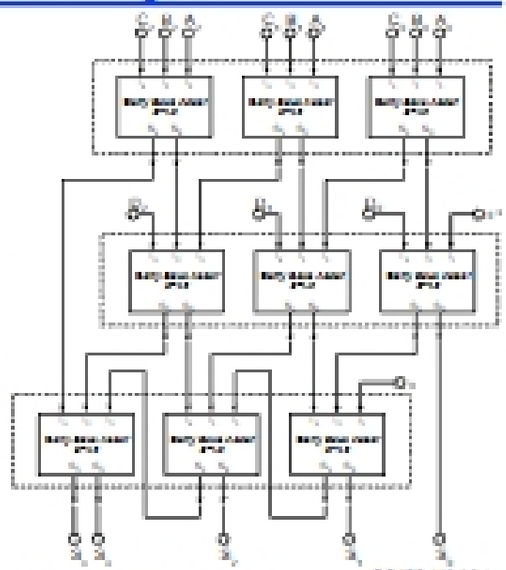
- At each stage shift A left (x 2)
- Use next bit of B to determine whether to add in shifted multiplicand
- Accumulate 2n bit partial product at each stage

Carry Save addition of 4 integers

- Adding:

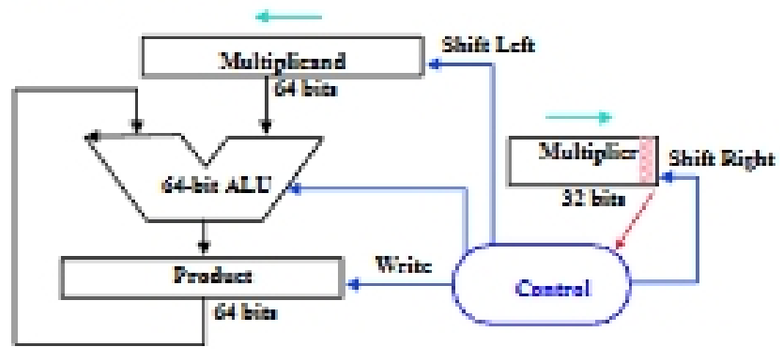
$$\begin{array}{r} A_3A_2A_1A_0 \\ + \; B_3B_2B_1B_0 \\ + \; C_3C_2C_1C_0 \\ + \; D_3D_2D_1D_0 \\ \hline S_4S_3S_2S_1S_0 \end{array}$$

- Add Columns first, then rows!
- Full Adder = 3⇒2 element
- Can be used to reduce critical path of multiply
- Example: 68 bit multiply (for floating point):
  - At least 68 levels with naïve technique
  - Only 9 with Carry save addition!



Unisigned shift-add multiplier (version 1)

- 64-bit Multiplicand reg, 64-bit ALU, 64-bit Product reg, 32-bit multiplier reg



Multiplier = datapath + control