

Computer Architecture Performance

Dr. Arjan Durresi
Louisiana State University
Baton Rouge, LA 70810
Durresi@Csc.LSU.Edu

These slides are available at:

http://www.csc.lsu.edu/~durresi/CSC7080_05/



- Performance Metrics
- CPU time, throughput
- Cycles, clock rate, CPI, MIPS
- Benchmarks

Performance

- Measure, Report, and Summarize
- Make intelligent choices
- See through the marketing hype
- Key to understanding underlying organizational motivation

Why is some hardware better than others for different programs?

What factors of system performance are hardware related?

(e.g., Do we need a new machine, or a new operating system?)

Which of these airplanes has the best performance?



Airplane	Passengers	Range (mi)	Speed (mph)	Passenger throughput
Boeing 737-100	101	630	598	228,750
Boeing 747	470	4150	610	286,700
BAC/Sud Concorde	132	4000	1350	178,200
Douglas DC-8-50	146	8720	544	79,429

- How much faster is the Concorde compared to the 747?
- How much bigger is the 747 than the Douglas DC-8?

Computer Performance: TIME, TIME, TIME

- Response Time (latency)
 - How long does it take for my job to run?
 - How long does it take to execute a job?
 - How long must I wait for the database query?
- Throughput
 - How many jobs can the machine run at once?
 - What is the average execution rate?
 - How much work is getting done?
- *If we upgrade a machine with a new processor what do we increase?*
- *If we add a new machine to the lab what do we increase?*

Execution Time

- Elapsed Time
 - counts everything (*disk and memory accesses, I/O, etc.*)
 - a useful number, but often not good for comparison purposes
- CPU time
 - doesn't count I/O or time spent running other programs
 - can be broken up into system time, and user time
- Our focus: user CPU time
 - time spent executing the lines of code that are "in" our program

Book's Definition of Performance

- For some program running on machine X,

$$\text{Performance}_X = 1 / \text{Execution time}_X$$

- "X is n times faster than Y"

$$\text{Performance}_X / \text{Performance}_Y = n$$

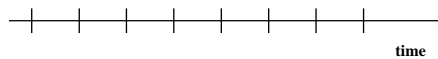
- Problem:

- machine A runs a program in 20 seconds
- machine B runs the same program in 25 seconds

Clock Cycles

- Instead of reporting execution time in seconds, we often use cycles

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$



- Clock "ticks" indicate when to start activities (one abstraction):

$$\frac{1}{4 \times 10^9} \times 10^{12} = 250 \text{ picoseconds (ps)}$$

- cycle time = time between ticks = seconds per cycle
- clock rate (frequency) = cycles per second (1 Hz = 1 cycle/sec)

How to Improve Performance

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

- So, to improve performance (everything else being equal) you can either (increase or decrease?)

_____ the # of required cycles for a program, or
_____ the clock cycle time or, said another way,
_____ the clock rate.

Example

- Our favorite program runs in 10 seconds on computer A, which has a 4 GHz. clock. We are trying to help a computer designer build a new machine B, that will run this program in 6 seconds. The designer can use new (or perhaps more expensive) technology to substantially increase the clock rate, but has informed us that this increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for the same program. What clock rate should we tell the designer to target?"

Example

- ❑ A program runs in 10s on computer A, at 4GHz
- ❑ How to build a computer B to run this program in 6s
- ❑ The designer has determined that if the clock rate will be increased, it will cause computer B to require 1.2 times more clock cycles than A
- ❑ What clock rate should be used in computer B?

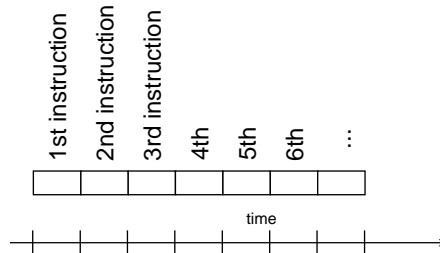
$$\text{CPU time}_A = \frac{\text{CPU clock cycles}_A}{\text{Clock rate}_A} \quad 10\text{s} = \frac{\text{CPU clock cycles}_A}{4 \times 10^9 \frac{\text{cycles}}{\text{seconds}}}$$

$$\text{CPU clock cycles}_A = 40 \times 10^9 \text{ cycles}$$

$$\text{CPU time}_B = \frac{1.2 \times \text{CPU clock cycles}_A}{\text{Clock rate}_B} \quad \text{Clock rate}_B = 8 \text{ GHz}$$

How many cycles are required for a program?

- ❑ Could assume that number of cycles equals number of instructions

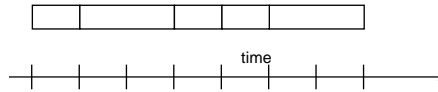


This assumption is incorrect,

different instructions take different amounts of time on different machines.

Why? hint: remember that these are machine instructions, not lines of C code

Different numbers of cycles for different instructions



- ❑ Multiplication takes more time than addition
- ❑ Floating point operations take longer than integer ones
- ❑ Accessing memory takes more time than accessing registers
- ❑ *Important point: changing the cycle time often changes the number of cycles required for various instructions (more later)*

Now that we understand cycles

- ❑ A given program will require
 - some number of instructions (machine instructions)
 - some number of cycles
 - some number of seconds
- ❑ We have a vocabulary that relates these quantities:
 - cycle time (seconds per cycle)
 - clock rate (cycles per second)
 - CPI (cycles per instruction)
 - a floating point intensive application might have a higher CPI*
 - MIPS (millions of instructions per second)
 - this would be higher for a program using simple instructions*

Performance

- ❑ Performance is determined by execution time
- ❑ Do any of the other variables equal performance?
 - # of cycles to execute program?
 - # of instructions in program?
 - # of cycles per second?
 - average # of cycles per instruction?
 - average # of instructions per second?
- ❑ Common pitfall: thinking one of the variables is indicative of performance when it really isn't.

CPI Example

- ❑ Suppose we have two implementations of the same instruction set architecture (ISA).

For some program,

Machine A has a clock cycle time of 250 ps and a CPI of 2.0

Machine B has a clock cycle time of 500 ps and a CPI of 1.2

What machine is faster for this program, and by how much?

CPI Example

CPU clock cycles_A = I x 2.0 ; CPU clock cycles_B = I x 1.2

CPU time_A = CPU clock cycles_A x CPU clock time_A = I x 2.0 x 250ps = I x 500ps

CPU time_B = CPU clock cycles_A x CPU clock time_A = I x 1.2 x 500ps = I x 600ps

$$\frac{\text{CPU time}_B}{\text{CPU time}_A} = 1.2$$

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

CPI

- CPU clock cycles = $\sum(\text{CPI}_i \times C_i)$
- C_i is the count of the number of instructions of class i , CPI_i is the average number per instructions for that class.

of Instructions Example

- A compiler designer is trying to decide between two code sequences for a particular machine. Based on the hardware implementation, there are three different classes of instructions: Class A, Class B, and Class C, and they require one, two, and three cycles (respectively).

The first code sequence has 5 instructions: 2 of A, 1 of B, and 2 of C

The second sequence has 6 instructions: 4 of A, 1 of B, and 1 of C.

Which sequence will be faster? How much?
What is the CPI for each sequence?

of Instructions Example

- CPU clock cycles₁ = $\sum(CPI_i \times C_i) = (2 \times 1) + (1 \times 2) + (2 \times 3) = 10$ cycles
- CPU clock cycles₂ = $\sum(CPI_i \times C_i) = (4 \times 1) + (1 \times 2) + (1 \times 3) = 9$ cycles
- $CPI_1 = 10/2 = 2$
- $CPI_2 = 9/6 = 1.5$
- When comparing, all three factors: clock rate, number of instructions, and CPI should be compared

MIPS

- ❑ One alternative to time is the metric MIPS (Million Instructions per Second)
- ❑
$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$
- ❑ MIPS does not take into account the capabilities of instructions
- ❑ MIPS varies among programs on the same computer
- ❑ MIPS can vary inversely with performance

MIPS example

- ❑ Two different compilers are being tested for a 4 GHz. machine with three different classes of instructions: Class A, Class B, and Class C, which require one, two, and three cycles (respectively). Both compilers are used to produce code for a large piece of software.

The first compiler's code uses 5 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

The second compiler's code uses 10 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

- ❑ Which sequence will be faster according to MIPS?
- ❑ Which sequence will be faster according to execution time?

MIPS example

$$\text{Execution time} = \frac{\text{CPU clock cycles}}{\text{Clock rate}}$$

$$\text{CPU clock cycles}_1 = \sum(\text{CPI}_i \times C_i) = ((5 \times 1) + (1 \times 2) + (1 \times 3)) \times 10^9 = 10 \times 10^9$$

$$\text{CPU clock cycles}_2 = \sum(\text{CPI}_i \times C_i) = ((10 \times 1) + (1 \times 2) + (1 \times 3)) \times 10^9 = 15 \times 10^9$$

$$\text{Execution time}_1 = 2.5 \text{ seconds}$$

$$\text{Execution time}_2 = 3.75 \text{ seconds}$$

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$\text{MIPS}_1 = 2800$$

$$\text{MIPS}_2 = 3200$$

Benchmarks

- ❑ Performance best determined by running a real application
 - Use programs typical of expected workload
 - Or, typical of expected class of applications
e.g., compilers/editors, scientific applications, graphics, etc.
- ❑ Small benchmarks
 - nice for architects and designers
 - easy to standardize
 - can be abused
- ❑ SPEC (System Performance Evaluation Cooperative)
 - companies have agreed on a set of real program and inputs
 - valuable indicator of performance (and compiler technology)
 - can still be abused

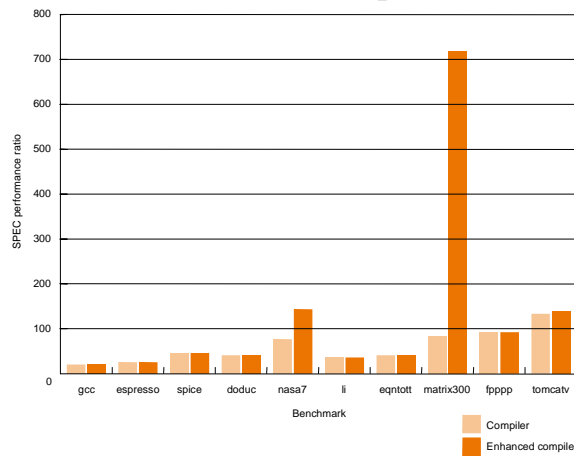
Benchmark Games

- *An embarrassed Intel Corp. acknowledged Friday that a bug in a software program known as a compiler had led the company to overstate the speed of its microprocessor chips on an industry benchmark by 10 percent. However, industry analysts said the coding error...was a sad commentary on a common industry practice of “cheating” on standardized performance tests...The error was pointed out to Intel two days ago by a competitor, Motorola ...came in a test known as SPECint92...Intel acknowledged that it had “optimized” its compiler to improve its test scores. The company had also said that it did not like the practice but felt to compelled to make the optimizations because its competitors were doing the same thing...At the heart of Intel’s problem is the practice of “tuning” compiler programs to recognize certain computing problems in the test and then substituting special handwritten pieces of code...*

Saturday, January 6, 1996 New York Times

SPEC '89

- **Compiler “enhancements” and performance**



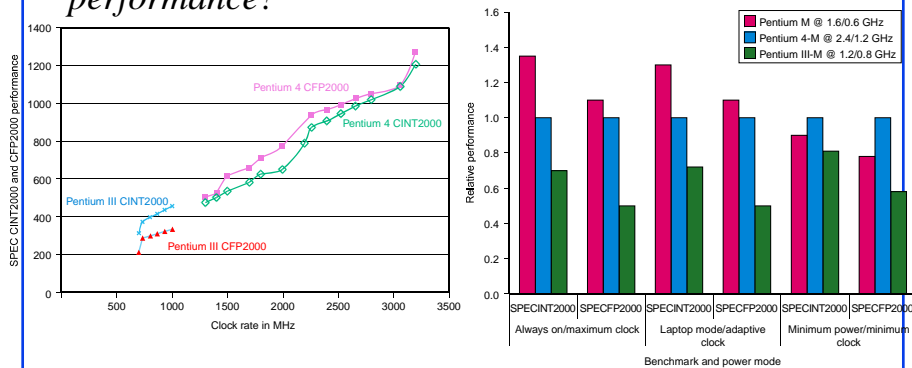
SPEC CPU2000

Integer benchmarks		FP benchmarks	
Name	Description	Name	Type
gzip	Compression	wupwise	Quantum chromodynamics
vpr	FPGA circuit placement and routing	swim	Shallow water model
gcc	The Gnu C compiler	mgrid	Multigrid solver in 3-D potential field
mcf	Combinatorial optimization	applu	Parabolic/elliptic partial differential equation
crafty	Chess program	mesa	Three-dimensional graphics library
parser	Word processing program	galgel	Computational fluid dynamics
scn	Computer visualization	art	Image recognition using neural networks
perlmmk	perl application	equale	Seismic wave propagation simulation
gap	Group theory, interpreter	facerec	Image recognition of faces
vortex	Object-oriented database	ammp	Computational chemistry
bbp2	Compression	lucas	Primality testing
twolf	Place and rote simulator	fma3d	Crash simulation using finite-element method
		sttrack	High-energy nuclear physics accelerator design
		apst	Metacology: pollutant distribution

FIGURE 4.5 The SPEC CPU2000 benchmarks. The 12 integer benchmarks in the left half of the table are written in C and C++, while the floating-point benchmarks in the right half are written in Fortran (77 or 90) and C. For more information on SPEC and on the SPEC benchmarks, see www.spec.org. The SPEC CPU benchmarks use wall clock time as the metric, but because there is little I/O, they measure CPU performance.

SPEC 2000

*Does doubling the clock rate double the performance?
Can a machine with a slower clock rate have better performance?*



Pentium 4 provides a new set of instructions (Streaming SIMD)
So both CPI and instruction count are different

Experiment

- ❑ Phone a major computer retailer and tell them you are having trouble deciding between two different computers, specifically you are confused about the processors strengths and weaknesses

(e.g., Pentium 4 at 2Ghz vs. Celeron M at 1.4 Ghz)
- ❑ What kind of response are you likely to get?
- ❑ What kind of response could you give a friend with the same question?

Amdahl's Law

Execution Time After Improvement =

Execution Time Unaffected +(Execution Time Affected / Amount of Improvement)

- ❑ Example:

"Suppose a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?"

How about making it 5 times faster?

- ❑ *Principle: Make the common case fast*

Example

- ❑ Suppose we enhance a machine making all floating-point instructions run five times faster. If the execution time of some benchmark before the floating-point enhancement is 10 seconds, what will the speedup be if half of the 10 seconds is spent executing floating-point instructions?

- ❑ We are looking for a benchmark to show off the new floating-point unit described above, and want the overall benchmark to show a speedup of 3. One benchmark we are considering runs for 100 seconds with the old floating-point hardware. How much of the execution time would floating-point instructions have to account for in this program in order to yield our desired speedup on this benchmark?

Remember

- ❑ Performance is specific to a particular program/s
 - Total execution time is a consistent summary of performance

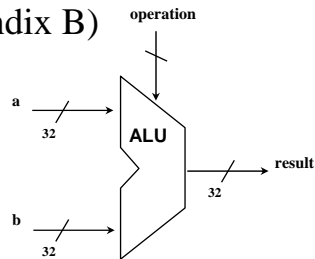
- ❑ For a given architecture performance increases come from:
 - increases in clock rate (without adverse CPI affects)
 - improvements in processor organization that lower CPI
 - compiler enhancements that lower CPI and/or instruction count
 - Algorithm/Language choices that affect instruction count

- ❑ Pitfall: expecting improvement in one aspect of a machine's performance to affect the total performance

Lets Build a Processor

- ❑ Almost ready to move into chapter 5 and start building a processor
- ❑ First, let's review Boolean Logic and build the ALU we'll need

(Material from Appendix B)



Review: Boolean Algebra & Gates

- ❑ Problem: Consider a logic function with three inputs: A, B, and C.

Output D is true if at least one input is true

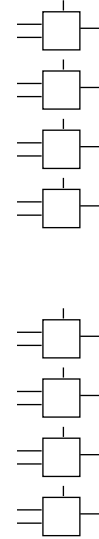
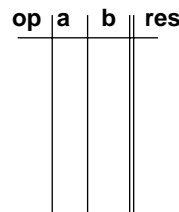
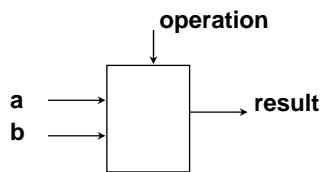
Output E is true if exactly two inputs are true

Output F is true only if all three inputs are true

- ❑ Show the truth table for these three functions.
- ❑ Show the Boolean equations for these three functions.
- ❑ Show an implementation consisting of inverters, AND, and OR gates.

An ALU (arithmetic logic unit)

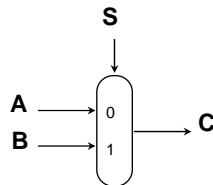
- Let's build an ALU to support the `andi` and `ori` instructions
 - we'll just build a 1 bit ALU, and use 32 of them



Review: The Multiplexor

- Selects one of the inputs to be the output, based on a control input

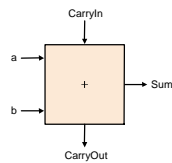
note: we call this a 2-input mux even though it has 3 inputs!



- Lets build our ALU using a MUX:

Different Implementations

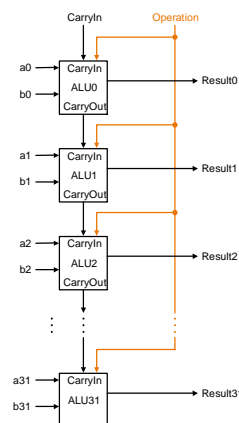
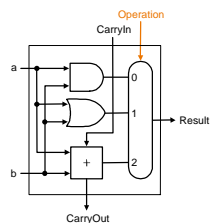
- ❑ Not easy to decide the “best” way to build something
 - Don't want too many inputs to a single gate
 - Don't want to have to go through too many gates
 - for our purposes, ease of comprehension is important
- ❑ Let's look at a 1-bit ALU for addition:



$$c_{out} = a b + a c_{in} + b c_{in}$$

$$sum = a \text{ xor } b \text{ xor } c_{in}$$

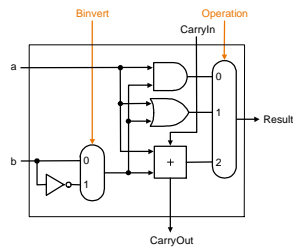
Building a 32 bit ALU



What about subtraction ($a - b$) ?

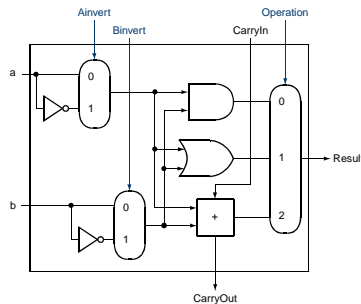
- ❑ Two's complement approach: just negate b and add.
- ❑ How do we negate?

- ❑ A very clever solution:



Adding a NOR function

- ❑ Can also choose to invert a.
- ❑ How do we get "a NOR b" ?

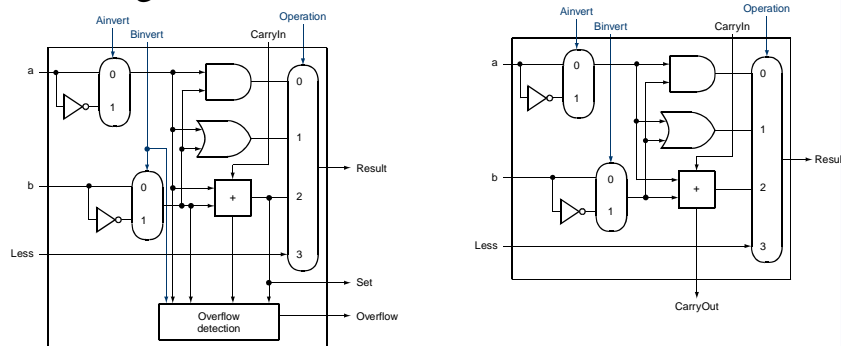


Tailoring the ALU to the MIPS

- Need to support the set-on-less-than instruction (slt)
 - remember: slt is an arithmetic instruction
 - produces a 1 if $rs < rt$ and 0 otherwise
 - use subtraction: $(a-b) < 0$ implies $a < b$
- Need to support test for equality (beq \$t5, \$t6, \$t7)
 - use subtraction: $(a-b) = 0$ implies $a = b$

Supporting slt

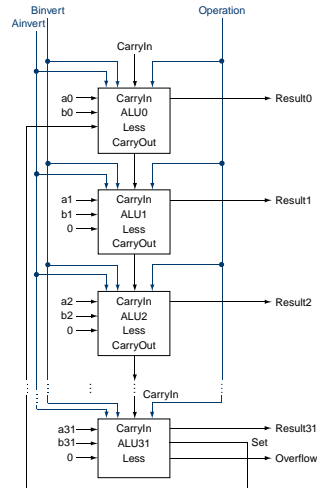
- Can we figure out the idea?



Use this ALU for most significant bit

all other bits

Supporting slt

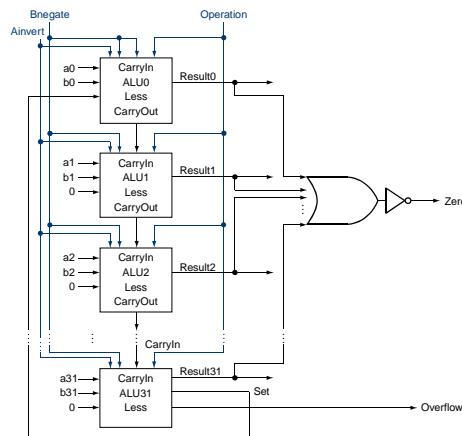


Test for equality

Notice control lines:

- 0000 = and
- 0001 = or
- 0010 = add
- 0110 = subtract
- 0111 = slt
- 1100 = NOR

Note: zero is a 1 when the result is zero!



Conclusion

- ❑ We can build an ALU to support the MIPS instruction set
 - key idea: use multiplexor to select the output we want
 - we can efficiently perform subtraction using two's complement
 - we can replicate a 1-bit ALU to produce a 32-bit ALU
- ❑ Important points about hardware
 - all of the gates are always working
 - the speed of a gate is affected by the number of inputs to the gate
 - the speed of a circuit is affected by the number of gates in series (on the “critical path” or the “deepest level of logic”)
- ❑ Our primary focus: comprehension, however,
 - Clever changes to organization can improve performance (similar to using better algorithms in software)
 - We saw this in multiplication, let's look at addition now

Problem: ripple carry adder is slow

- ❑ Is a 32-bit ALU as fast as a 1-bit ALU?
- ❑ Is there more than one way to do addition?
 - two extremes: ripple carry and sum-of-products

Can you see the ripple? How could you get rid of it?

$$c_1 = b_0c_0 + a_0c_0 + a_0b_0$$

$$c_2 = b_1c_1 + a_1c_1 + a_1b_1c_2 =$$

$$c_3 = b_2c_2 + a_2c_2 + a_2b_2 \quad c_3 =$$

$$c_4 = b_3c_3 + a_3c_3 + a_3b_3 \quad c_4 =$$

Not feasible! Why?

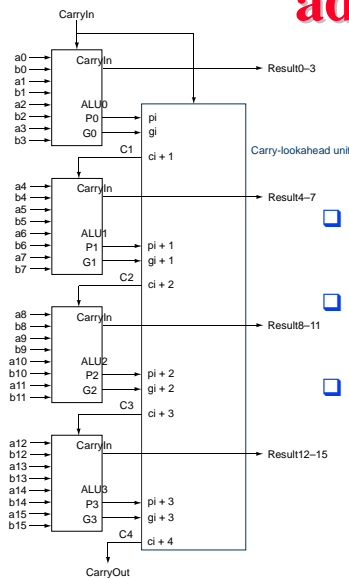
Carry-lookahead adder

- An approach in-between our two extremes
- Motivation:
 - If we didn't know the value of carry-in, what could we do?
 - When would we always generate a carry? $g_i = a_i b_i$
 - When would we propagate the carry? $p_i = a_i + b_i$
- Did we get rid of the ripple?

$$\begin{aligned}
 c_1 &= g_0 + p_0 c_0 \\
 c_2 &= g_1 + p_1 c_1 & c_2 &= \\
 c_3 &= g_2 + p_2 c_2 & c_3 &= \\
 c_4 &= g_3 + p_3 c_3 & c_4 &=
 \end{aligned}$$

Feasible! Why?

Use principle to build bigger adders



- Can't build a 16 bit adder this way... (too big)
- Could use ripple carry of 4-bit CLA adders
- Better: use the CLA principle again!

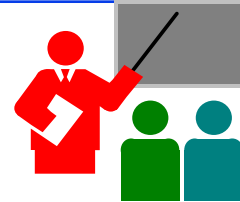
ALU Summary

- ❑ We can build an ALU to support MIPS addition
- ❑ Our focus is on comprehension, not performance
- ❑ Real processors use more sophisticated techniques for arithmetic
- ❑ Where performance is not critical, hardware description languages allow designers to completely automate the creation of hardware!

```
module MIPSALU (ALUctl, A, B, ALUOut, Zero):
  input [3:0] ALUctl;
  input [31:0] A,B;
  output reg [31:0] ALUOut;
  output Zero;
  assign Zero = (ALUOut==0); //Zero is true if ALUOut is 0; goes anywhere
  always @(ALUctl, A, B) //reevaluate if these change
  case (ALUctl)
    0: ALUOut <= A & B;
    1: ALUOut <= A | B;
    2: ALUOut <= A + B;
    6: ALUOut <= A - B;
    7: ALUOut <= A < B ? 1:0;
    12: ALUOut <= ~(A | B); // result is nor
    default: ALUOut <= 0; //default to 0, should not happen:
  endcase
endmodule
```

FIGURE B.4.3 A Verilog behavioral definition of a MIPS ALU. This could be synthesized using a module library containing basic arithmetic and logical operations.

Summary



- ❑ Instruction complexity is only one variable
 - lower instruction count vs. higher CPI / lower clock rate
- ❑ Design Principles:
 - simplicity favors regularity
 - smaller is faster
 - good design demands compromise
 - make the common case fast
- ❑ Instruction set architecture
 - a very important abstraction indeed!