Review: (Write on the board before class)
1) Cost: 1/2 of equation
2) IC cost = f(Area^3)
3) List price vs ASP vs direct costs vs cost

Today Price, Price/Performance, Pitfalls

Business models and people overhead: can vary quite a bit:
For every dollar of increased component cost the average selling price goes up between $2.50 and $3.50, and the list price increases between $3.75 and $4.67.

only 8% to 15% of their income on R&D; business model that won't change (engineering 10% of $)

Impact on supercomputer design: Where invest R&D?
More money to develop => higher margin => poorer price performance +> fewer sales

Elasiticy of price: Bell's hypothesis: cut price in half => 10X size of market (conversely doubling price means 1/10 th market)

Triple whammy vs. WS? more to invest so higher margin, fewer sales so higher margin, WS on faster growth path & longer to develop? Today fastest growing strata of computing

Impact on size of companies? DEC 100,000 employees; fastest machine in 1989 is $12,000 WS; MIPS co design; HW,Compilers, Standard OS; Single board? What going to do?

Price/Performance: keys are reproducibility of results
Figure 2.19 page 69 shows various ways of representing performance

Comparing machines of different price: Figure 2.21

Pitfalls

Fallacy: Hardware-independent metrics predict performance.
Code Size = Speed fallacy (shortcut used by compiler writers of old)

Pitfall: Performance can be summarized by translating throughput into execution time
Motorola's trick(See figure) confusing bandwidth and response time

Fallacy: Peak performance tracks observed performance.
"guranteeed not to exceed" (See figures)
i860 story : 150 MOPS (1 int & 1 float = add & mult)
Common practice in supercomputer industry

Break
Bring wichmann book;
Did you get email? Did you find a partner for HW? Questions?
be prepared to discuss in class next Monday or friday 3.9
Discussion questions?

Chapter 3: Instruction Set Architecture Review, will go over quickly

Implementation vs. Architecture =>
Computer Architecture = Instruction set architecture

Also, hard to figure best order; instruction sets and then fast execution or vice versa; if reread this chapter first, will see difference

static vs. dynamic

First concentrate on register based machines, since dominate and likely to stay dominant; stacks of historical interest

Operand location: register-register or load/store vs. register-memory vs. memory-memory (see figures 3.5 & 3.6)

Memory addressing:
Endian (3.7), Alignment in memory
List of addressing modes (3.11) & frequency (3.12)

Rest of statistics like to see as answer to questions?

What is impact of different sized offsets in instruction set? (3.13)
How important are immediates? (3.14)
SIze of immediates? (3.15) 0 not included here

Next time: Gordon Bell on supercomputers/Titan; see about size of market Monday operations on a machine

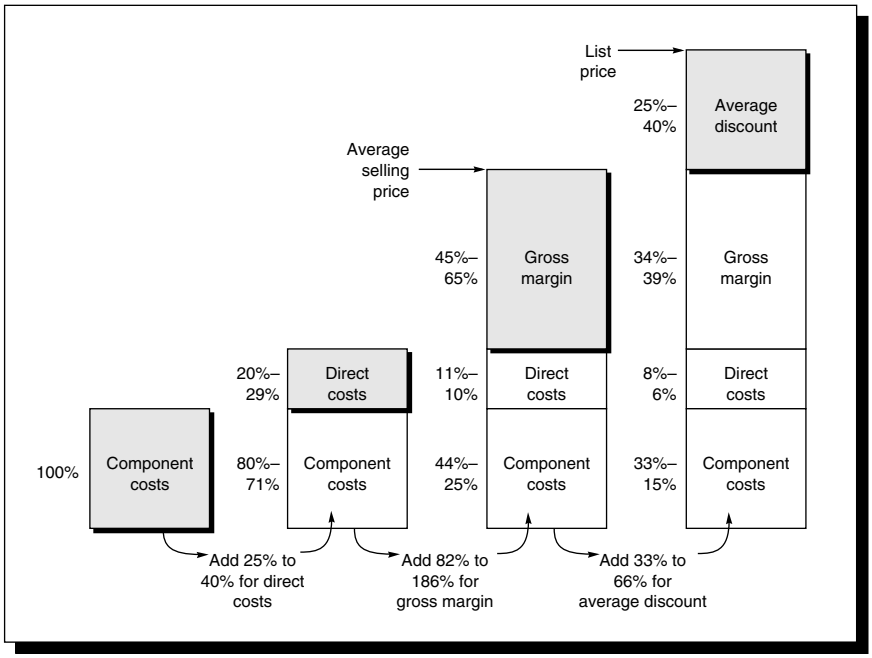

FIGURE 2.15 Starting with component costs, the price increases as we allow for direct costs, gross margin, and average discount, until we arrive at the list price.

	Model A	As % of costs	As % of list price	Model B	As % of costs	As % of list price
Component costs	$2,145	100%	27%	$2,145	100%	21%
Component costs + direct costs	$2,681	125%	33%	$3,003	140%	30%
Average selling price (adds gross margin)	$5,363	250%	67%	$7,508	350%	75%
List price	$8,044	375%	100%	$10,010	467%	100%

FIGURE 2.16 The diskless workstation in Figure 2.13 priced using two different business models.

	VAXstation 2000		VAXstation 3100		DECstation 3100	
	CPU time	Elapsed time	CPU time	Elapsed time	CPU time	Elapsed time
Gnu C Compiler for 68000	985	1108	291	327	90	159
Common TeX	1264	1304	449	479	95	137
Spice	958	973	352	395	94	132
Arithmetic mean	1069	1128	364	400	93	143
Weighted AM—integer only (50% GCC, 50% TeX, 0% Spice)	1125	1206	370	403	93	148
Weighted AM—floating point (0% GCC, 0% TeX, 100% Spice)	958	973	352	395	94	132
Weighted AM—equal CPU time on V2000 (35.6% GCC, 27.8% TeX, 36.6% Spice)	1053	1113	357	394	93	143
Weighted AM—equal CPU time on V3100 (40.4% GCC, 26.2% TeX, 33.4% Spice)	1049	1114	353	390	93	144
Weighted AM—equal CPU time on D3100 (34.4% GCC, 32.6% TeX, 33.0% Spice)	1067	1127	363	399	93	143

FIGURE 2.19 Performance of the programs in Figure 2.17

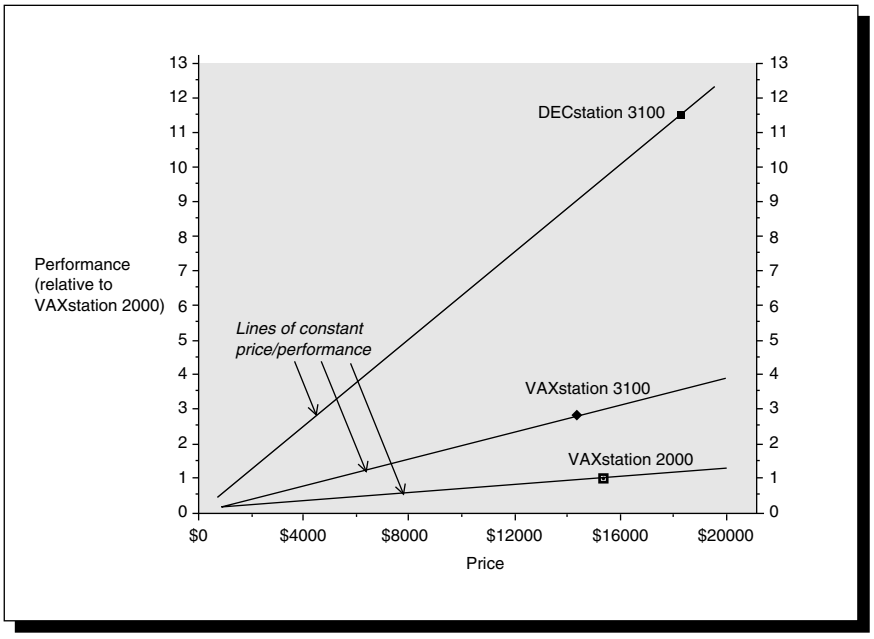

FIGURE 2.21 Price versus performance of VAXstation 2000, VAXstation 3100, and DECstation 3100

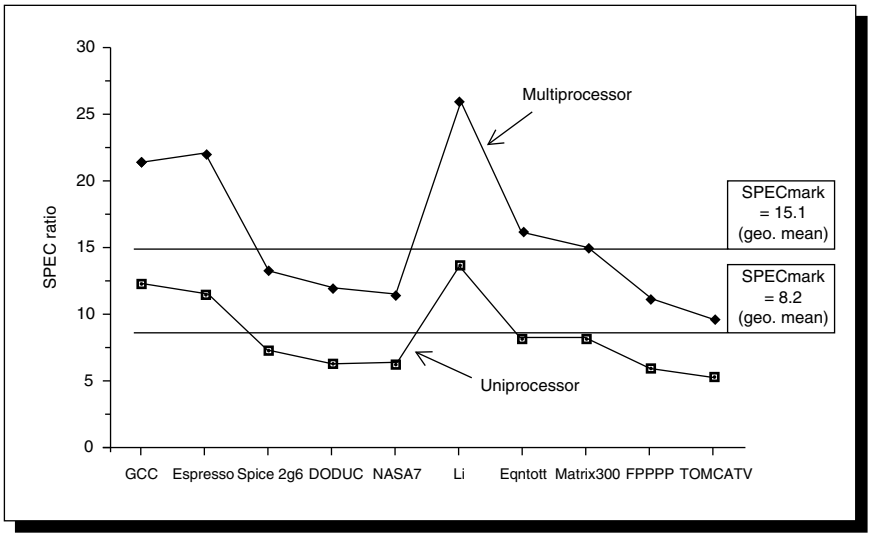

FIGURE 2.23 Performance of uniprocessor and multiprocessor

	CRAY X-MP	Hitachi S810/20	Perfor-mance
A(i)=B(i)*C(i)+D(i)*E(i) (vector length 1000 done 100,000 times)	2.6 secs	1.3 secs	Hitachi 105% faster
Vectorized FFT (vector lengths 64, 32,...,2)	3.9 secs	7.7 secs	CRAY 97% faster

FIGURE 2.24 Measurements of peak performance and actual performance for Hitachi S810/20 and CRAY X-MP.

Machine	Peak MFLOPS rating	Harmonic mean MFLOPS of perfect benchmarks	Percent of peak MFLOPS
CRAY X-MP/416	940	14.8	1%
IBM 3090-600S	800	8.3	1%
NEC SX/2	1300	16.6	1%

FIGURE 2.25 Peak performance and harmonic mean of actual performance for the Perfect Benchmarks.

Number of memory addresses per typical ALU instruction	Maximum number of operands allowed for a typical ALU instruction	Examples
0	2 3	IBM RT-PC SPARC, MIPS, HP Precision Architecture
1	2 3	PDP-10, Motorola 68000, IBM 360 Part of IBM 360 (RS instructions)
2	2 3	PDP-11, National 32x32, part of IBM 360 (SS instructions)
3	3	VAX (also has two operand formats)

FIGURE 3.5 Combinations of memory and total operands

Type	Advantages	Disadvantages
Register–register (0,3)	Simple, fixed-length instruction encoding. Simple code generation model. Instructions take similar numbers of clocks to execute (see Chapter 6).	Higher instruction count than architectures with memory references in instructions. Some instructions are short and bit encoding may be wasteful.
Register–memory (1,2)	Data can be accessed without loading first. Instruction format tends to be easy to encode and yields good density.	Operands are not equivalent since a source operand in a binary operation is destroyed. Encoding a register number and a memory address in each instruction may restrict the number of registers. CPI varies by operand location.
Memory–memory (3,3)	Most compact. Doesn't waste registers for temporaries.	Large variation in instruction size, especially for three-operand instructions. Also, large variation in work per instruction. Memory accesses create memory bottleneck.

FIGURE 3.6 Advantages and disadvantages of the three most common types of general-purpose register machines. The notation (*m, n*) means *m* memory operands and *n* total operands.

Addressing mode	Example instruction	Meaning	When used
Register	`Add R4,R3`	R4←R4+R3	When a value is in a register.
Immediate or literal	`Add R4,#3`	R4←R4+3	For constants. In some machines, literal and immediate are two different addressing modes.
Displacement or based	`Add R4,100(R1)`	R4←R4+M[100+R1]	Accessing local variables.
Register defer-red or indirect	`Add R4,(R1)`	R4←R4+M[R1]	Accessing using a pointer or a computed address.
Indexed	`Add R3,(R1 + R2)`	R3←R3+M[R1+R2]	Sometimes useful in array addressing—`R1`=base of array; `R2`=index amount.
Direct or absolute	`Add R1,(1001)`	R1←R1+M[1001]	Sometimes useful for accessing static data; address constant may need to be large.
Memory indirect or memory deferred	`Add R1,@(R3)`	R1←R1+M[M[R3]]	If `R3` is the address of a pointer *p*, then mode yields **p*.
Auto-increment	`Add R1,(R2)+`	R1←R1+M[R2] R2←R2+*d*	Useful for stepping through arrays within a loop. `R2` points to start of array; each reference increments `R2` by size of an element, *d*.
Auto-decrement	`Add R1,-(R2)`	R2←R2-*d* R1←R1+M[R2]	Same use as autoincrement. Autoincrement/decrement can also be used to implement a stack as push and pop.
Scaled or index	`Add R1,100(R2)[R3]`	R1← R1+M[100+R2+R3*d]	Used to index arrays. May be applied to any base addressing mode in some machines.

FIGURE 3.11 Selection of addressing modes with examples, meaning, and usage.

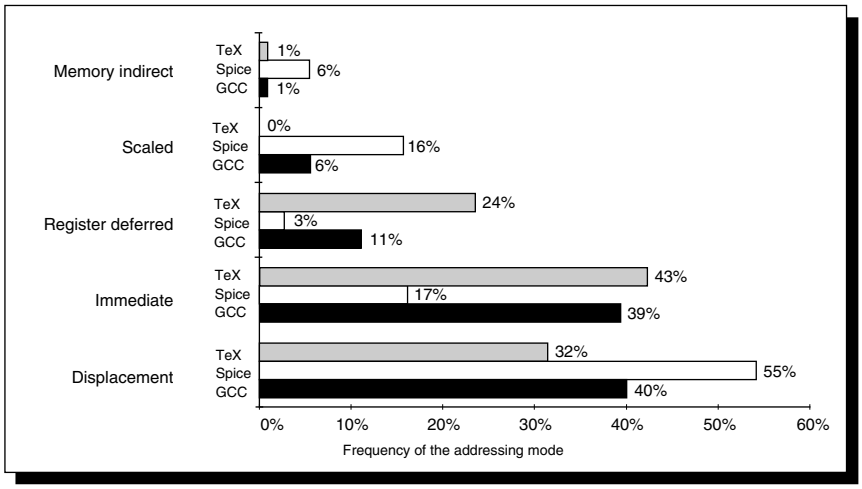

FIGURE 3.12 Summary of use of memory addressing modes (including immediates).