# CS4100: 計算機結構

# Performance and Cost

國立清華大學資訊工程學系
九十三學年度第一學期





## 買那一支手機比較好?



## 差不多的價錢，你怎麼比？

易利信T68

諾基亞8250

摩托羅拉V66



## 何謂手機的效能？

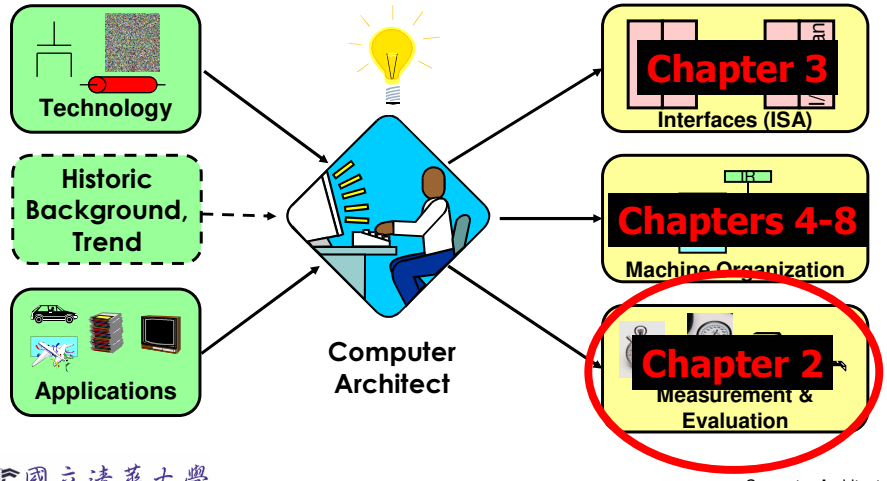
- 比較的基準有那些？
- 有那些值可以量測、評比的？
- 如何量？
- 如何提出客觀的評比報告？
- ...

# Performance

- Purchasing perspective:
  Given a collection of machines, which has the
  - best performance? least cost? best performance/cost?
- Design perspective:
  Faced with design options, which has the
  - best performance improvement? least cost?
  - best performance/cost?
- Both require
  - basis for comparison
  - metric for evaluation
- Goal: understand cost and performance implications of architectural choices



# Tasks of a Computer Architect






# Outline

- Performance
  - Definition
  - CPU performance formula
  - Measuring and evaluating performance
- Cost
  - Cost and price
  - Cost of chips



# 那一架飛機的效能比較好？

Concorde:
- Capacity: 100 persons
- Range: 6667 km
- Cruising speed: 2160 kph (Mach 2) at 60,000 ft

747-400:
- Capacity: 400 persons
- Range: 11,485 km
- Cruising speed: 929 kph at 35,000 ft

# Two Notions of Performance

| Plane | DC to Paris | Speed | Passengers | Throughput (pmph) |
|---|---|---|---|---|
| Boeing 747 | 6.5 hours | 610 mph | 470 | 286,700 |
| Concorde | 3 hours | 1350 mph (Mach 2) | 132 | 178,200 |

- Which has higher performance?
  - Time to delivery 1 passenger? deliver 400 passengers?
  - Time to do the task: execution time, response time, latency
  - Tasks per unit time: throughput, bandwidth

*Response time and throughput often are in opposition*



# Which Is Better?

- Time of Concorde vs. Boeing 747:
  - Concord is 1350 mph / 610 mph
    = 2.2 times faster
    = 6.5 hours / 3 hours
- Throughput of Concorde vs. Boeing 747:
  - Boeing is 286,700 pmph / 178,200 pmph
    = 1.6 times better
- Boeing is 1.6 times (60%) faster in terms of throughput
- Concord is 2.2 times (120%) faster in terms of flying time (response time)

***We will focus on execution time for a single job***



# Performance Definition

- Performance according to time:
  => faster is better

$$\text{performance(X)} = \frac{1}{\text{execution\_time(X)}}$$

- If interested in comparing two things:
  "X is *n* times faster than Y" means

$$n = \frac{\text{performance(X)}}{\text{performance(Y)}}$$



# What is Time?

- Straightforward definition of time:
  - Total time to complete a task, including disk & memory accesses, I/O activities, OS overhead, …
  - May include execution time of other programs in a multiprogramming environment
  - Too many factors involved
- Alternative: the time that the processor (CPU) is working only on your program (since multiple processes running at same time)
  - "CPU execution time" or "CPU time "
  - Often divided into *system CPU time* (in OS) and *user CPU time* (in user program)

***CPU performance: user CPU time of a single program***

# Outline

- Performance
  - Definition
  - *CPU performance formula (Sec. 2.3)*
  - Measuring and evaluating performance
- Cost
  - Cost and price
  - Cost of chips



# 如何以公式表達程式執行時間?

- Hint: basic components of a program
- 指令數
- 指令執行時間（平均）



# 何謂程式的指令數？

- 有幾條C指令？

```
for(i=0; i<100; i++)
   a[i] = b[i] * c[i];
```

- 有幾條組合語言指令？

```
        sub  $r1,$r2,$r3
Loop:   beq  $r9,$r0,End
        add  $r8,$r8,$r10
        addi $r9,$r9,-1
        j    Loop
End:
```

10 times => 41 instructions

*Dynamic Instruction Count*



# Instruction Execution Time

- Time unit: from a user's perspective: time = seconds
- CPU Time: computers are constructed using a clock that runs at a constant rate and determines when events take place in the hardware
  - These discrete time intervals called *clock cycles* (or informally clocks or cycles)
  - Length of clock period: *clock cycle time* (e.g., 2 nanoseconds or 2 ns) and *clock rate* (e.g., 500 megahertz, or 500 MHz), which is the inverse of the clock period

指令執行時間以cycle為單位

# Program Execution Time

CPU execution time for program

= Clock Cycles for program x Clock Cycle Time

$$= \frac{\text{Clock Cycles for program}}{\text{Clock Rate}}$$

Clock Cycles for program
= Instructions for program ("Instruction Count")
x Average Clock Cycles per Instruction ("CPI")

- CPI: one way to compare two machines with same instruction set, since Instruction Count is same



# Performance Calculation (1/2)

CPU execution time for program

= Clock Cycles for program x Clock Cycle Time

- Substituting for clock cycles:

CPU execution time for program

= Instruction Count x CPI x Clock Cycle Time

$$\text{CPU time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$



# How to Calculate the 3 Components?

- Clock Cycle Time: in specification of computer (Clock Rate in advertisements)
- Instruction Count:
  - Count instructions in loop of small program
  - Use simulator or emulator to count instructions
  - Debugger or tracing program
  - Execution-based monitoring: insert instrumentation code into binary code, run, and record information
  - Hardware counter in special register (Pentium)
- CPI:
  - Calculate: $\frac{\text{Execution Time / Clock Cycle Time}}{\text{Instruction Count}}$
  - Hardware counter in special register (Pentium)



# Calculating CPI Another Way

- First calculate CPI for each individual instruction (add, sub, and, etc.)
- Next calculate frequency of each individual instruction in the workload
- Finally multiply these two for each instruction and add them up to get final CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \frac{\text{Clock Cycles}_i}{\text{Instruction Count}}$$

$$\text{CPI} = \sum_{i=1}^{n} \text{CPI}_i \times F_i \quad \text{where} \quad F_i = \frac{I_i}{\text{Instruction Count}}$$

="instruction frequency"

# Example (RISC processor)

| Op | $Freq_i$ | $CPI_i$ | Prod | (% Time) |
|---|---|---|---|---|
| ALU | 50% | 1 | .5 | (23%) |
| Load | 20% | 5 | 1.0 | (45%) |
| Store | 10% | 3 | .3 | (14%) |
| Branch | 20% | 2 | .4 | (18%) |
| | | | 2.2 | |

Instruction Mix (Where time spent)

- What if Branch instructions twice as fast?
- What if two ALU instr. could be executed at once?

*Must know the limit of architectural enhancement*



# Summary: CPU Time Formula

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

| | Instruction Count | Cycle per Instruction (CPI) | Clock Rate |
|---|---|---|---|
| Program | X | | |
| Compiler | X | X | |
| Instruction Set | X | X | |
| Organization | | X | X |
| Technology | | | X |



# 有關效能的另一個公式

從台北到高雄要多久？


如果改坐飛機，
台北到高雄只要1小時
全程可以加快多少？

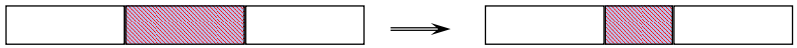
如何導公式？



# Amdahl's Law

- Speedup due to enhancement E:

$$\text{Speedup(E)} = \frac{\text{Execution Time w/o E}}{\text{Execution Time w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}$$



Suppose that enhancement E accelerates a fraction F of the task by a factor S and the remainder of the task is unaffected then,

$$\text{ExecutionTime(w/E)} = \left((1-F) + \frac{F}{S}\right) \times \text{ExecutionTime(w/oE)}$$

$$\text{Speedup(w/E)} = \frac{1}{(1-F) + \frac{F}{S}} \overset{S \to \infty}{\approx} \frac{1}{1-F}$$

# 由台北到高雄

- 不能enhance的部份為在市區的時間: 0.5 + 0.5 = 1小時
- 可以enhance的部份為在高速公路上的4小時
  => 佔總時間的 4/(4+1) = 0.8 = F
- 現在改用飛機, 可以enhance的部份縮短為1小時
  => S = 4/1 = 4
- $$\text{speedup} = \frac{\text{走高速公路所需時間}}{\text{坐飛機所需時間}} = \frac{4+1}{1+1} = 2.5$$
- 另一種算法:
  $$\text{speedup} = \frac{((1-0.8)+0.8)*5}{((1-0.8)+0.8/4)*5} = \frac{1}{(1-0.8)+0.8/4}$$
- When S -> ∞, speedup -> 5



# Outline

- Performance
  - Definition
  - CPU performance formula
  - *Measuring and evaluating performance (Sec. 2.4-2.6)*
    - Benchmark programs
    - Summarizing performance
    - Reporting performance
- Cost
  - Cost and price
  - Cost of chips



# What Programs for Comparison?

- What's wrong with this program as a workload?

```
integer A[][], B[][], C[][];
for (I=0; I<100; I++)
  for (J=0; J<100; J++)
    for (K=0; K<100; K++)
      C[I][J] = C[I][J] + A[I][K]*B[K][J];
```

- What measured? Not measured? What is it good for?
- Ideally run typical programs with typical input before purchase, or before even build machine
  - Called a "workload"; For example:
  - Engineer uses compiler, spreadsheet
  - Author uses word processor, drawing program, compression software



# Choosing Benchmark Programs

| Pros | | Cons |
|---|---|---|
| •representative | Actual Target Workload | •not portable<br>•hard to measure, find cause |
| •portable<br>•widely used<br>•improvements useful in reality | Full Application Benchmarks | •less representative |
| •easy to run, early in design | Small Kernel Benchmarks | •easy to fool |
| •find potential bottleneck & peak capability | Microbenchmarks | •peak does not reflect application performance |

# Benchmarks

- Obviously, apparent speed of processor depends on code used to test it
- Need industry standards so that different processors can be fairly compared => *benchmark programs*
- Companies exist that create these benchmarks: "typical" code used to evaluate systems
- Tricks in benchmarking:
  - different system configurations
  - compiler and libraries optimized (perhaps manually) for benchmarks
  - test specification biased towards one machine
  - very small benchmarks used
- Need to be changed every 2 or 3 years since designers could target these standard benchmarks

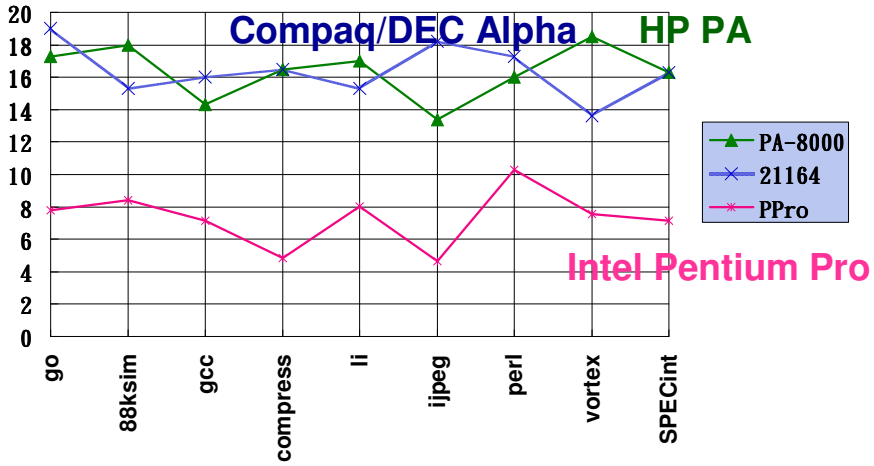


# Example Standardized Workload Benchmarks

- Workstations: Standard Performance Evaluation Corporation (SPEC)
- SPEC95:18 application benchmarks (with inputs) reflecting a technical workload (Fig. 2.6)
  - Eight integer:
    - go, m88ksim, gcc, compress, li, ijpeg, perl, vortex
  - Ten floating-point intensive:
    - tomcatv, swim, su2cor, hydro2d, mgrid, applu, turb3d, apsi, fppp, wave5
  - Separate average for integer (CINT95) and FP (CFP95) relative to a base machine
  - Benchmarks distributed in source code
  - Company representatives select workload
  - Compiler, machine designers target benchmarks, so try to change every 3 years

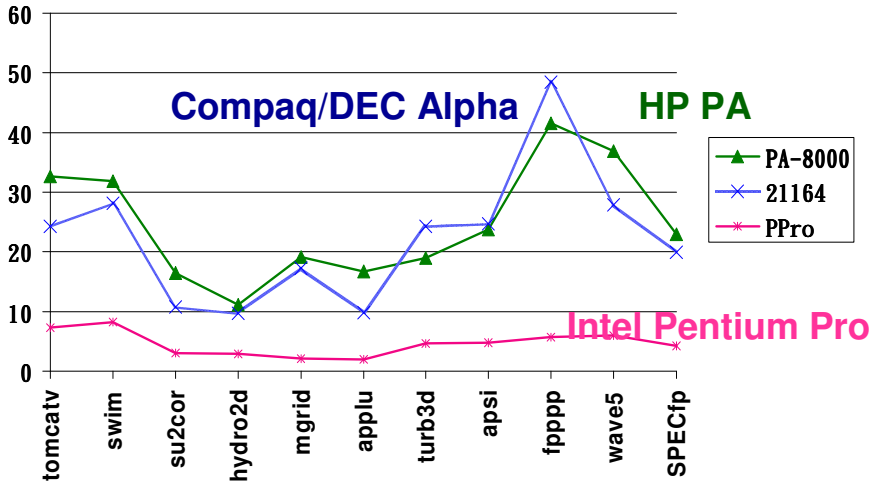


# SPECint95base Performance (10/1997)






# SPECfp95base Performance (10/1997)



# SPEC2000 (CINT)

| Benchmark | Language | Category |
|---|---|---|
| 164.gzip | C | Compression |
| 175.vpr | C | FPGA Placement/Route |
| 176.gcc | C | C Compiler |
| 181.mcf | C | Combinatorial Opt. |
| 186.crafty | C | Chess |
| 197.parser | C | Word Processing |
| 252.eon | C++ | Computer Visualization |
| 253.perlbmk | C | PERL |
| 254.gap | C | Group Theory, Interpreter |
| 255.vortex | C | OO Database |
| 256.bzip2 | C | Compression |
| 300.twolf | C | Place/Route Simulator |

(http://www.spec.org/cpu2000)



# SPEC2000 (CFP)

| Benchmark | Lang. | Category |
|---|---|---|
| 168.wupwise | F77 | Quantum Chromodynamics |
| 171.swim | F77 | Shallow Water Modeling |
| 172.mgrid | F77 | Multi-grid Solver |
| 173.applu | F77 | Parabolic/Elliptic PDE |
| 177.mesa | C | 3-D Graphics Library |
| 178.galgel | F90 | Computational Fluid Dynamics |
| 179.art | C | Image Recognition/Neural Net |
| 183.equake | C | Seismic Wave Propagation |
| 187.facerec | F90 | Image Processing |
| 188.ammp | C | Computational Chemistry |
| 189.lucas | F90 | Number Theory |
| 191.fma3d | F90 | Finite-element Crash Simulation |
| 200.sixtrack | F77 | Nuclear Accelerator Designs |
| 301.apsi | F77 | Pollutant Distribution |



# Example PC Workload Benchmark

- PCs: Ziff Davis WinStone 99 Benchmark
  - A system-level, application-based benchmark that measures a PC's overall performance when running today's top-selling Windows-based 32-bit applications
  - Works through a series of scripted activities and uses the time a PC takes to complete those activities to produce its performance scores
  - Winstone's tests don't mimic what these programs do; they run actual application code
  - www1.zdnet.com/zdbop/winstone/winstone.html



# Winstone 99 (W99) Results

| Company | Processor | Price | Clock | W99 |
|---|---|---|---|---|
| emachines | Cyrix MII | $ 653 | 250 | 14.5 |
| CompUSA | Intel Celeron | $ 764 | 400 | 18.0 |
| Compaq | AMD K6-2 | $ 902 | 350 | 15.4 |
| HP | Intel Celeron | $1,070 | 366 | 17.6 |
| Compaq | AMD K6-2 | $1,453 | 450 | 17.9 |
| Compaq | AMD K6-3 | $1,479 | 400 | 22.3 |
| HP | Intel Pentium II | $1,483 | 400 | 18.9 |
| NEC | Intel Pentium III | $1,680 | 400 | 22.0 |

- Note: 2 Compaq Machines using K6-2 v. 6-3:
  K6-2 Clock Rate is 1.125 times faster, but
  K6-3 Winstone 99 rating is 1.25 times faster!

# Summarizing Performance

| | Machine A | Machine B |
|---|---|---|
| Program 1 | 1 s | 10 s |
| Program 2 | 1000 s | 100 s |
| Total | 1001 s | 110 s |

- A is 10 times faster than B for program 1
- B is 10 times faster than A for program 2
- ➢ What is relative performance of A & B?
- Arithmetic mean (tracking total time):

$$\frac{\text{Perform(B)}}{\text{Perform(A)}} = \frac{1001}{110} = 9.1$$

- Weighted arithmetic mean



# Early Lessons from SPEC

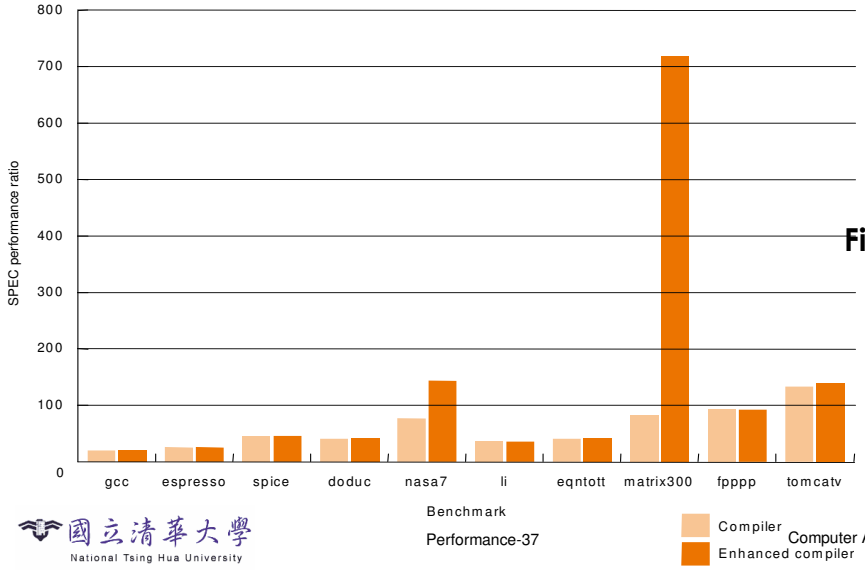



Fig. 2.3



# Summarizing Performance

- Problem with arithmetic mean using ratios (Fig. 2.10)
- Could combine normalized results with the *geometric* mean
  - Independent of the data for normalization

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio i}}$$

- Each SPECrate is a ratio of execution time
  - SPECrate(A,go) = time (sun SS10/40,go) / time(A,go)
  - Summary is geometric mean of these ratios



# Reporting Performance

- Guiding principle: reproducible
  - List everything another experimenter would need to duplicate the results
  - Fig. 2.4

| | |
|---|---|
| **Hardware** | |
| CPU | 41.67-MHz POWER 4164 |
| Cache size | 64K data/8K instruction |
| Memory | 64MB |
| Disk subsystem | 2 400-MB SCSI |
| **Software** | |
| OS | AIX v3.1.5 |
| Compiler | AIX XL C/6000 v1.1.5 |

# Summary: Performance

- Latency v. Throughput
  - CPU Time: time spent executing a single program: depends solely on design of processor (datapath, pipelining effectiveness, caches, etc.)
- Performance doesn't depend on any single factor: need to know Instruction Count, Clocks Per Instruction and Clock Rate to get valid estimations
- Performance evaluation needs to consider:
  - Benchmark programs
  - Summarizing performance
  - Reporting performance results



# Outline

- Performance
  - Definition
  - CPU performance formula
  - Measuring and evaluating performance
- *Cost*
  - *Cost and price*
  - *Cost of chips*



# Chip Cost: Manufacturing Process

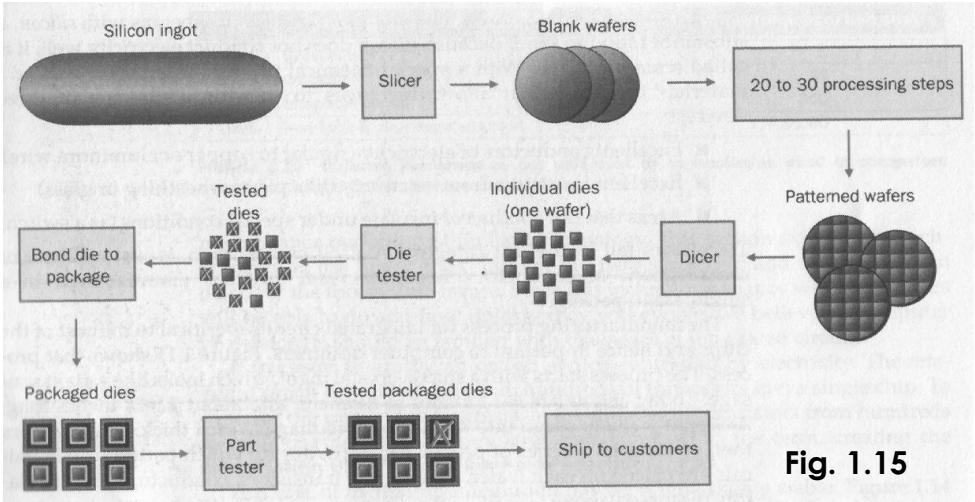



Fig. 1.15



# Cost of a Chip Includes ...

- Die cost: affected by wafer cost, number of dies per wafer, and die yield
  - goes roughly with the cube of the die area
  - An 8" wafer can contain 196 Pentium dies, but only 78 Pentium Pro (Fig. 1.16 and 1.17)
- Testing cost
- Packaging cost: depends on pins, heat dissipation, ...

# Real World Examples

| Chip | Metal layers | Line width | Wafer cost | Defect /cm$^2$ | Area mm$^2$ | 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 6014 | | 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



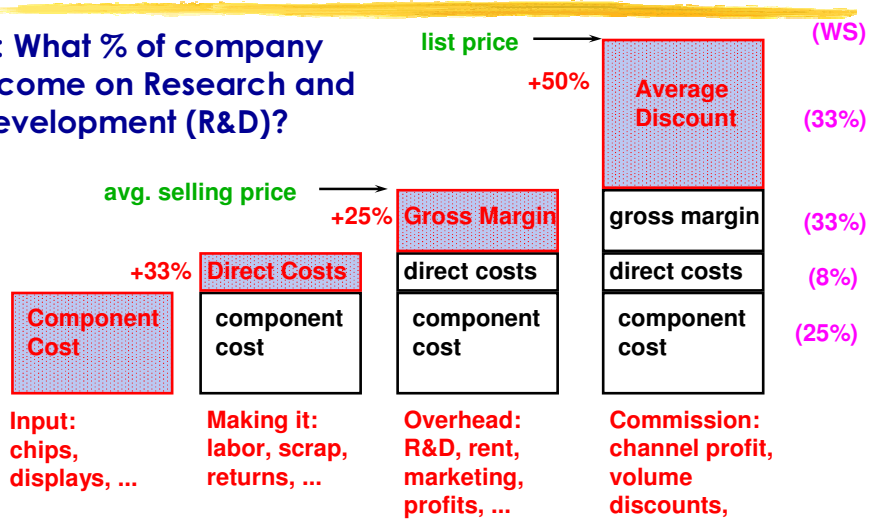
# System Cost: 1995 Workstation

| 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 versus Price

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






# Summary: Cost

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