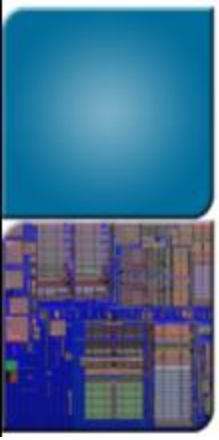
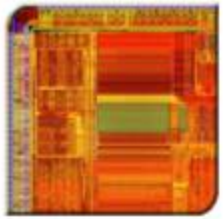


Hardware



A Washing machine

A simple Embedded System only work on a set of rules:

Step 1: Rinse in fresh water mixed with detergent

Step 2: Wash by spinning the motor

Step 3: Rinse in fresh water after draining dirty water

Step 4: Second spin with fresh water

Step 5: Draining out the water completely

Step 6: Dry spin

Step 7: Sound the alarm to signal the wash cycle is complete

In case of an interruption, stop and continue execute only the remaining part of the program

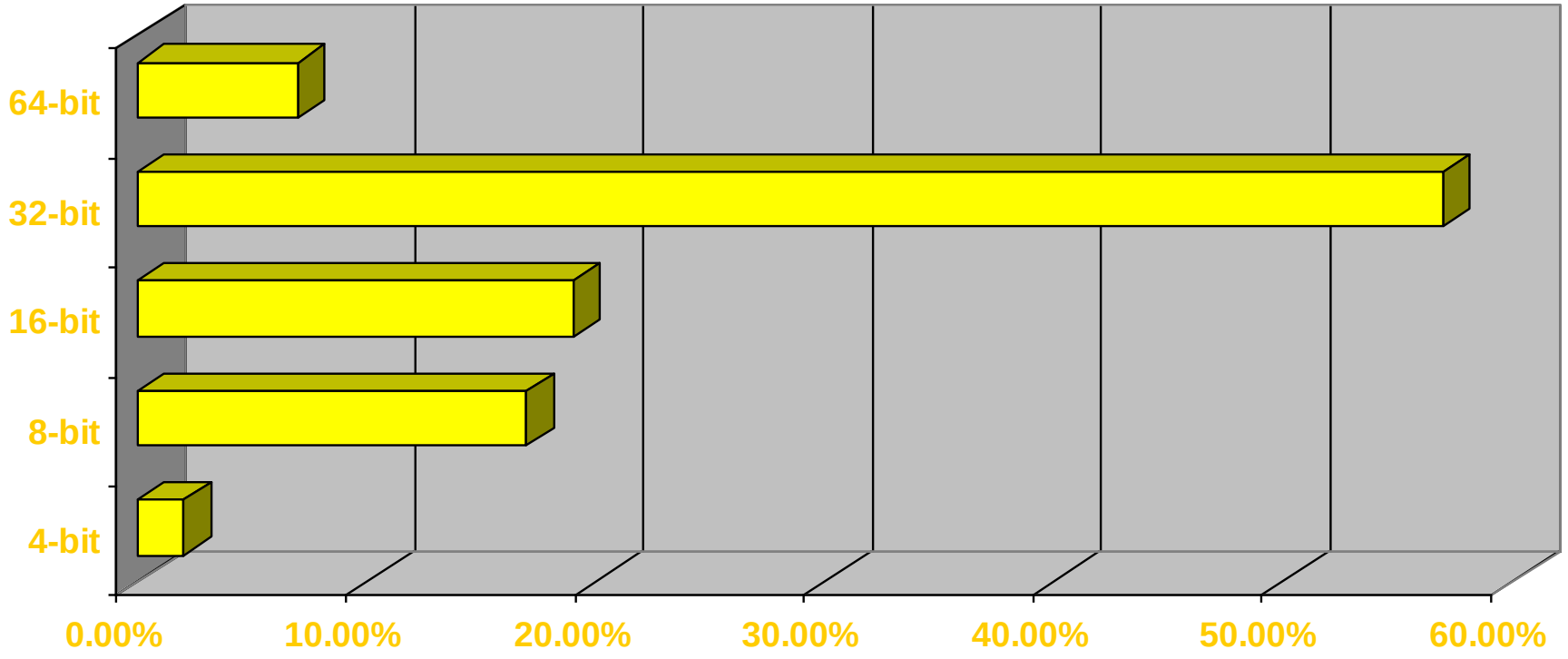
Three major components of Embedded Systems

- Hardware
  - Microprocessor
- Operating System
- Applications

What is actually being used in New Embedded Designs?

- What Types of Processors are used?
- What Operating Systems are used?
- What Programming Languages are used?
- Will examine data from a 2006 Market Survey of design engineers by EETimes and Embedded Systems Design Magazine

Processor Bit Size Used in New Embedded Designs

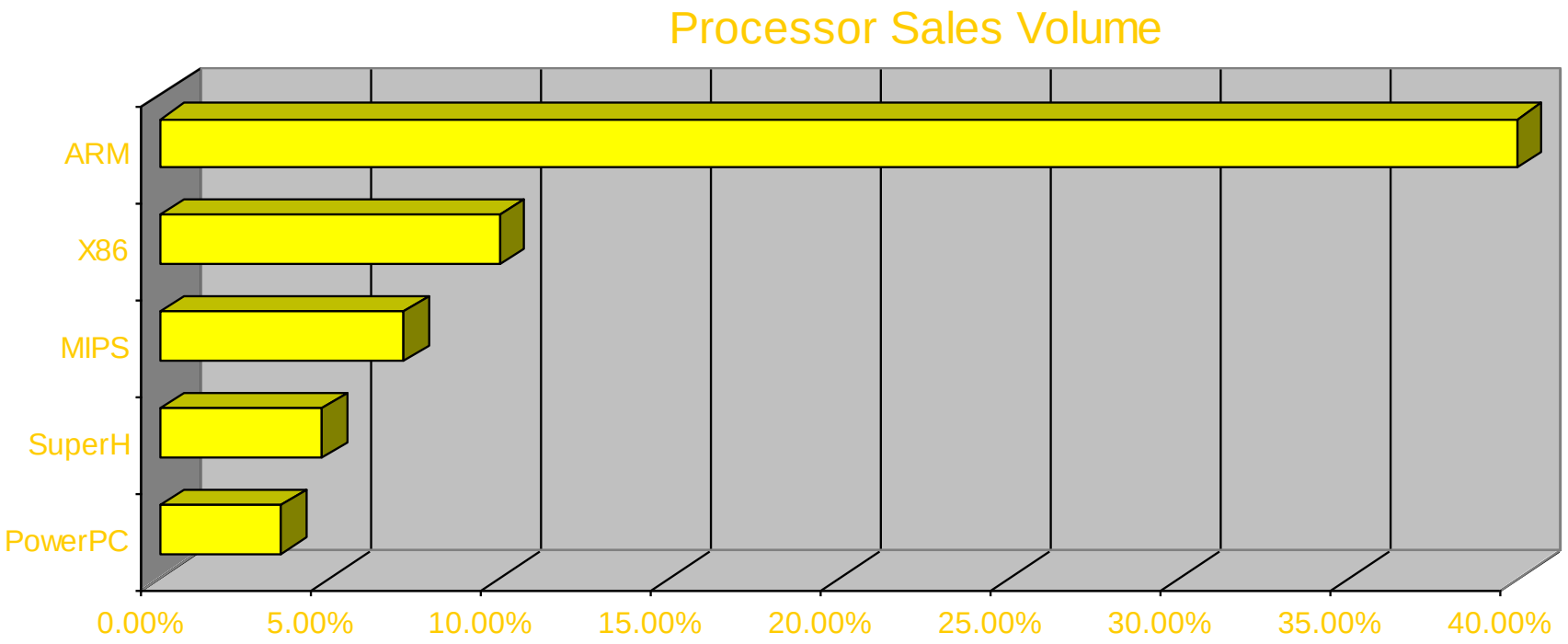


Data was derived from EETimes and Embedded Systems Design Magazine 2006 Embedded Market Survey

Processor Architectures Widely Used in New Embedded Designs

- ARM
- X86
- MIPS
- Xscale (ARM)
- Renesas SuperH
- PowerPC

32 & 64-bit Annual Processor Sales Volume



Based on 2002 sales data

Processor Selection Issues

- Software Support
  - OS, Compilers, Debug Tools, Applications
- Price
- Performance
- Power
  - Battery Life (MIPS/Watt), Cooling (Fan?)
    - Desktop PC 100 W vs. Battery power 200 mw
- Availability
  - Long term availability, Multiple Vendors?

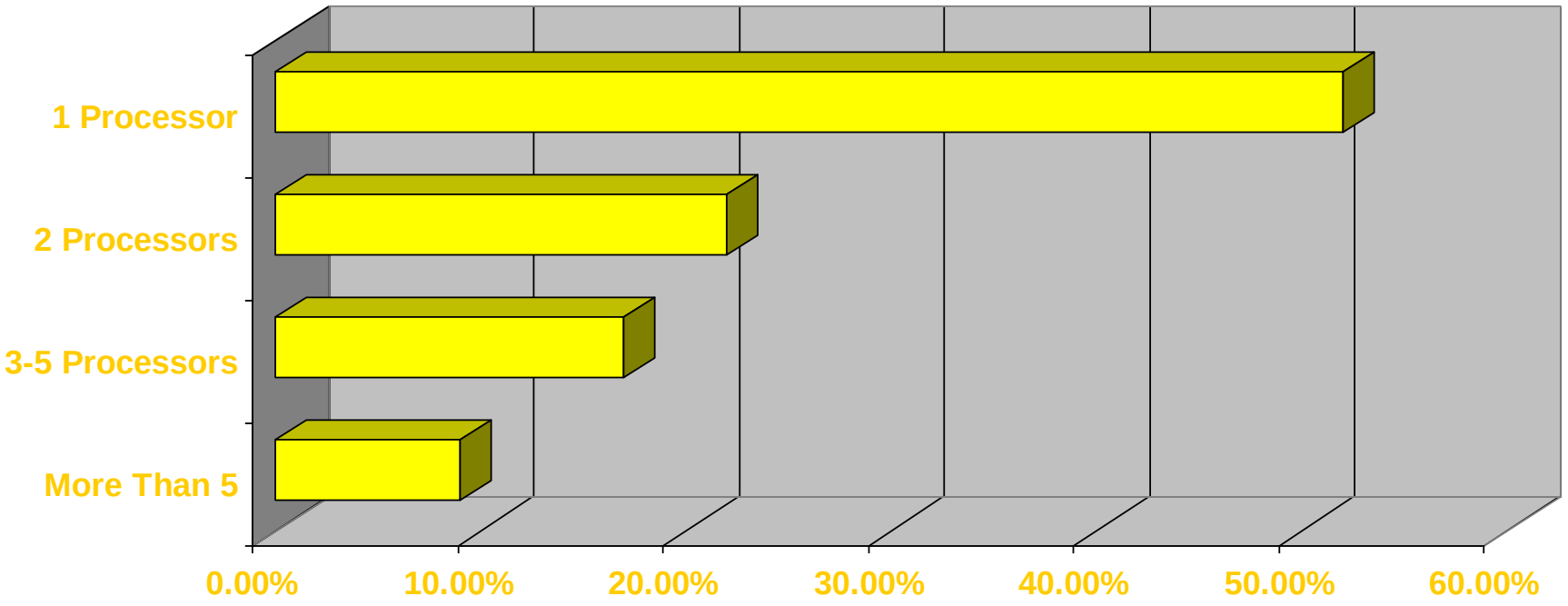
ARM Processors

- 32-bit RISC low-power design from an English IP company, ARM ltd (Advanced RISC Machines) http://www.arm.com/
- ARM's processor designs are licensed to over 100 chip manufacturers. ARM does not make chips.
- Used in many devices such as Cell phones, iPod Nano, Cameras, Handheld Games, HDTVs, and Set-Top boxes. 80% of ARM processors are in phones
- Good performance/power makes it a very popular choice in low power and battery operated devices.
- ARM's thumb instruction subset is coded into 16-bits and decompressed on-the-fly to full 32-bit instructions. Can switch from 16-bit to 32-bit instructions on the sub-routine level.

X86 (IA-32) Processors

- Based on the Intel X86 CISC instruction set used in processors in PCs since the mid 1980s
- Low cost due to widespread use in PC technology
- Processors and support chips are available from multiple vendors
- A wide processor performance range is available
- Most X86 processors for desktop PCs have been optimized for performance and not low power
- The major desktop PC processor vendors (Intel, AMD) are moving on to newer designs and 64-bit architectures – but other manufacturers are making X86 processors for embedded devices

Number of Processors Used in New Embedded Designs



Data was derived from EETimes and Embedded Systems Design Magazine 2006 Embedded Market Survey

Why Operating System(OS)?

Product: TMIO Connect^{IO} Oven

OS: Windows CE

Also refrigerates food before cooking and is controlled over the internet or via phone

Internet Oven: Example Tasks

- Control Oven Temperature (when cooking)
- Control Refrigeration Temperature (when cooling)
- Check for Keypad input and handle input
- Check cooking time to turn oven on/off (timed cooking)
- Update time and temperature display on front panel
- Check for Internet communications and handle messages
- Check for Phone communications and handle messages

Internet Oven: Tasks

- **How would all of these tasks be prioritized and scheduled?**
- **Is synchronization needed anywhere?**
- **What networking and communications support is needed?**
- **Would an Operating System help?**

Why have an OS in an embedded device?

- Support for multitasking, scheduling, and synchronization
- Support for a wide range of I/O devices
- Support for file systems
- Scheduling and buffering of I/O operations
- Support for networking
- Memory management
- Support for graphics displays
- Security and Power Management

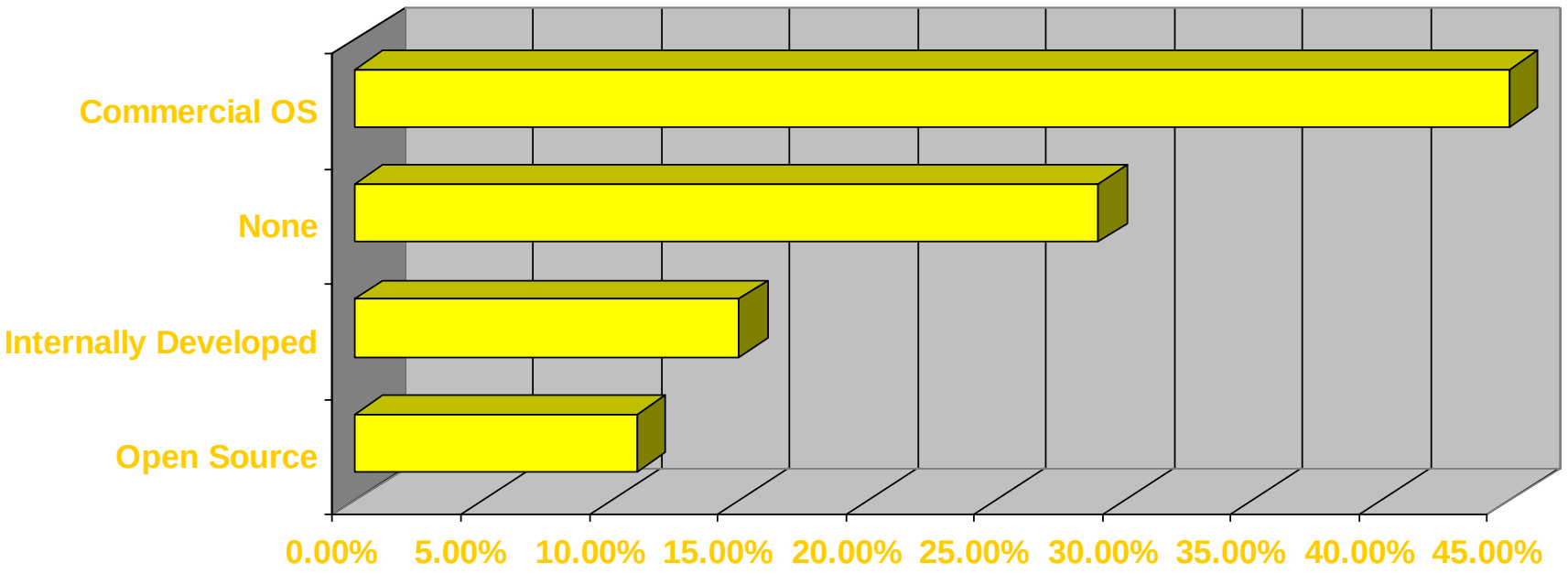
Why have an OS in an embedded device?

Example:

A recent cell phone design contained over five million lines of code!

- Few, if any projects will have the time and funding needed to develop all of this code on their own!
- Typical Embedded OS license fees are a few dollars per device – less than a desktop OS
- Some very simple low-end devices might not need an OS – but new devices are getting more complex

Use of Real-Time OS Kernels in New Embedded Designs

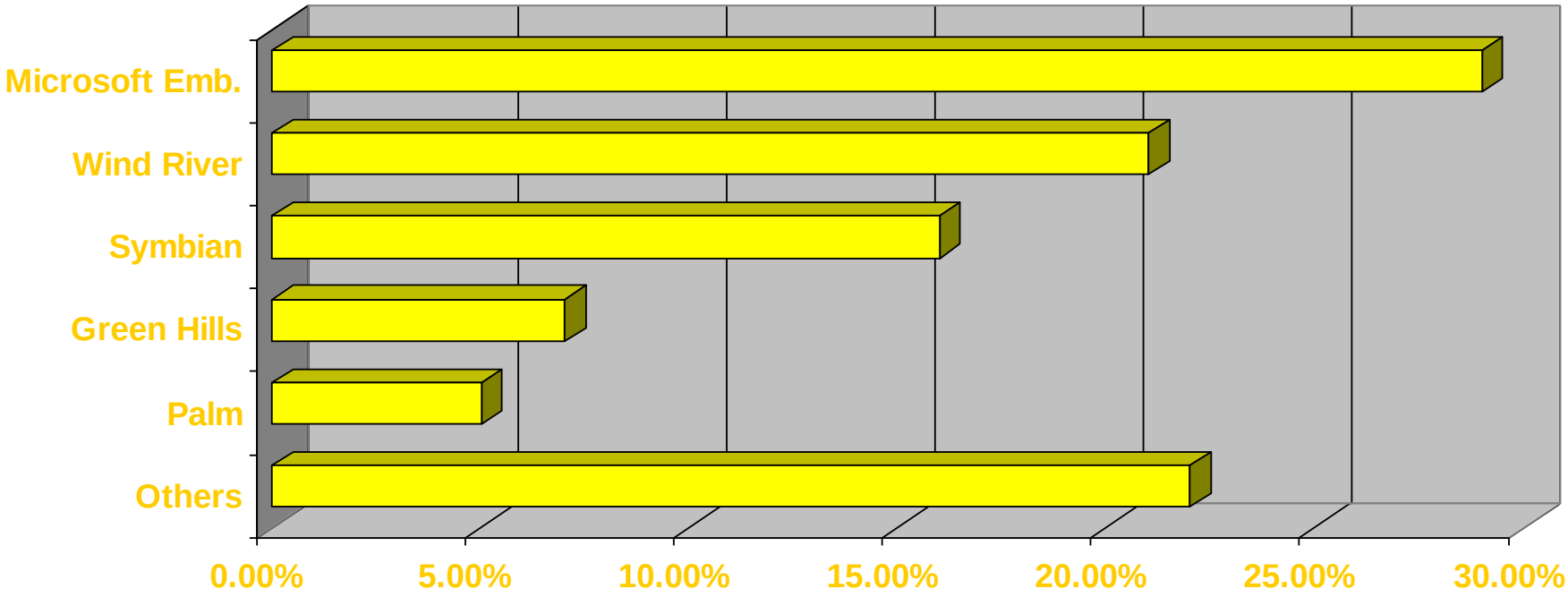


Data was derived from EETimes and Embedded Systems Design Magazine 2006 Embedded Market Survey

Open Source OS?

- “Free” OS can cost more for product development
- More time to develop kernel, device drivers & product can increase labor cost more than the commercial OS license fees saved
- Some other licenses still required for many devices in addition to free OS (real-time kernel, browser, encoders & decoders, encryption, media player)
- Open source license model may require that you publish your device’s source code
- Some Studies even show a recent decline in Open Source OS use:
  - http://www.embedded-forecast.com/EMFTCD2003v3.pdf
  - http://www.embedded.com/showArticle.jhtml?articleID=187203732

Commercial Operating Systems used in New Embedded Designs



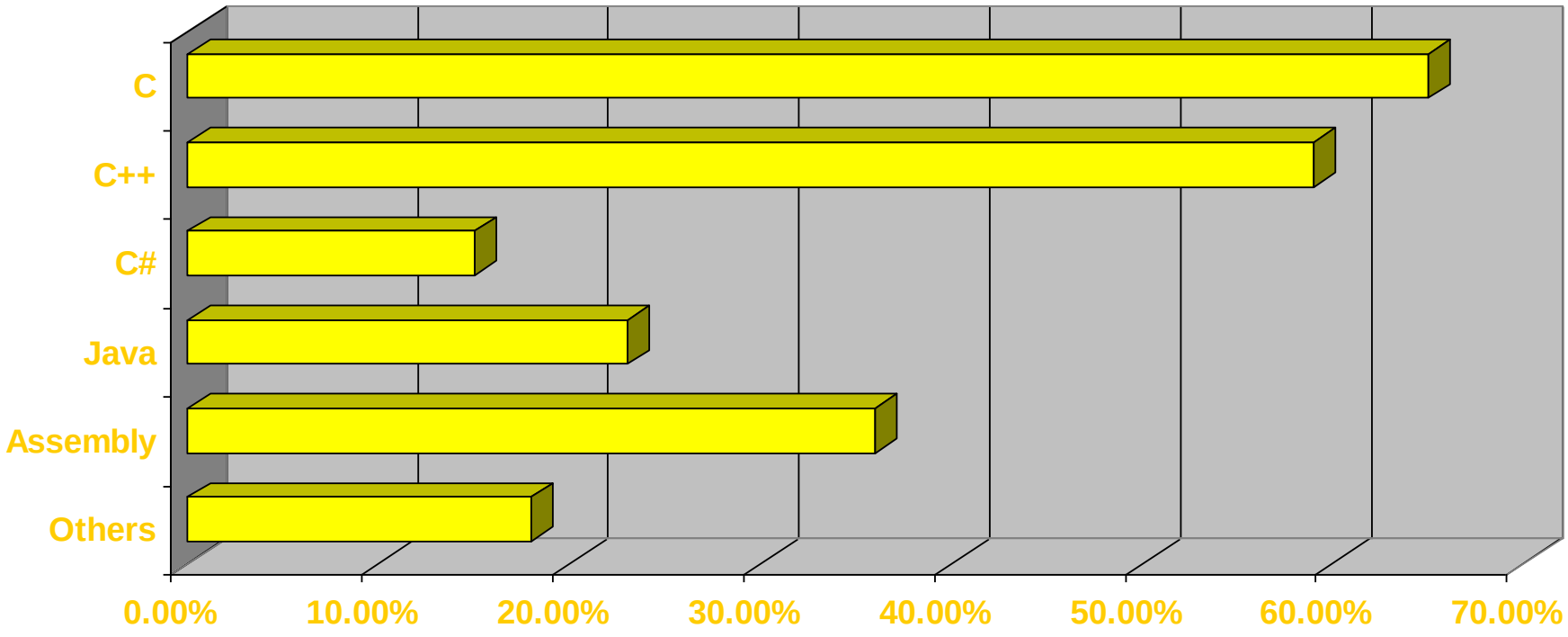
Data was derived from EETimes and Embedded Systems Design Magazine 2006 Embedded Market Survey

Applications

- What language should we use?

- What’s about the support?

Programming Languages Used in New Embedded Designs

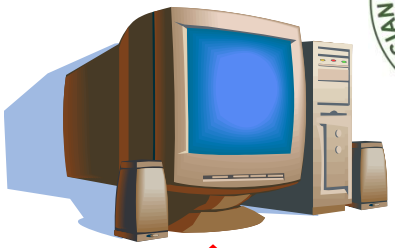


Data was derived from EETimes and Embedded Systems Design Magazine 2006 Embedded Market Survey

Typical Software Development Tools Used for Embedded Applications

- **Compiler** - compile C/C++ and in-line assembly language
- **Linker** – links compiled application code, OS, and runtime libraries
- **Memory Image tools** – tools to place code in non-volatile memory at a given physical memory address
- **Debugger** – tool to debug OS and applications programs
- **Loader** – load OS at power on and applications. Also a tool to download new code from the development system is typically provided.

Abstraction



High Level Language

```
main() {
   int  i,b,c,a[10];
   for (i=0; i<10; i++)...
      a[2] = b + c*i;
}
```

Memory Image tool

Compiler

Complete Machine code

ISA

```
...
lw    r2, mem[r7]
add   r3, r4, r2
st    r3, mem[r8]
```

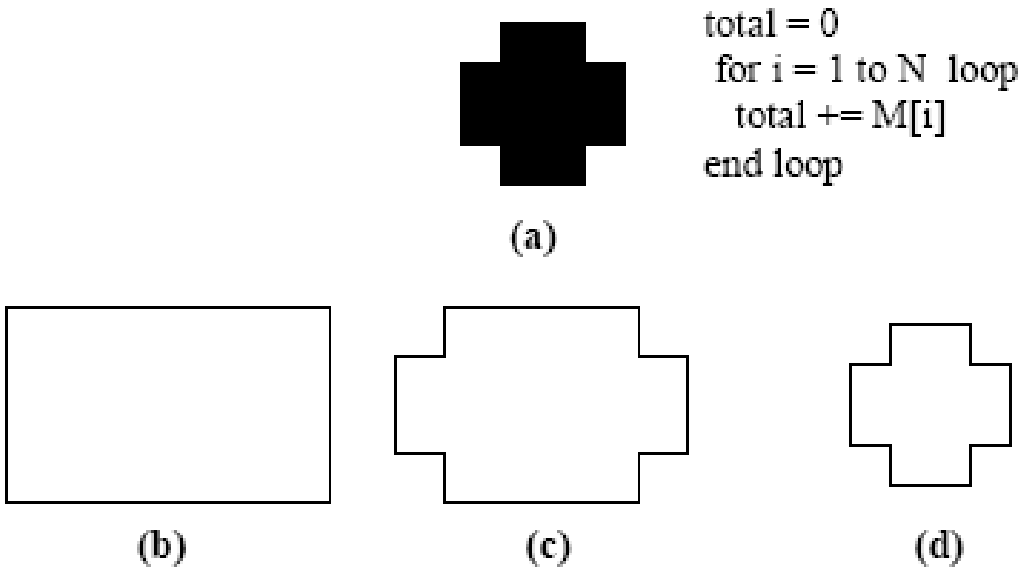
Machine code

Linker

Assembler

Other Library

Embedded processor technology

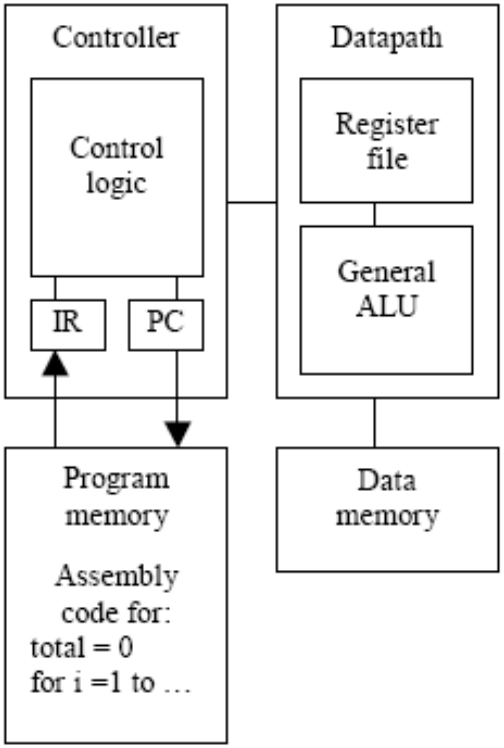


General purpose Processor (GP)

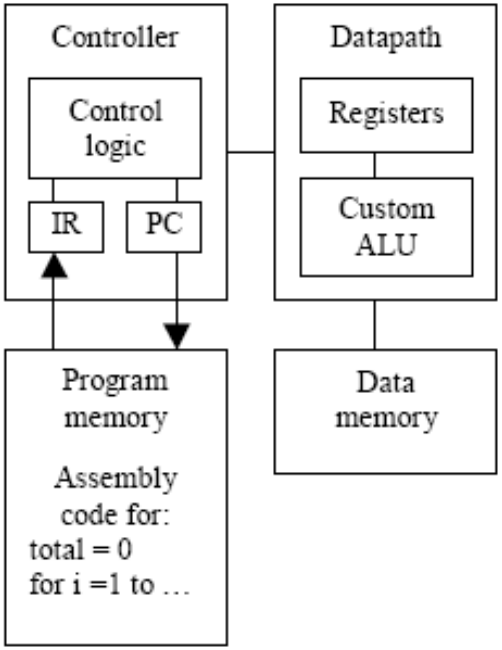
Application Specific processor

Single purpose processor

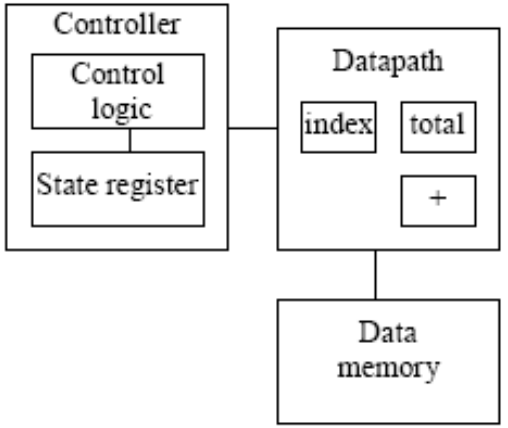
Implementation on different system



General purpose Processor (GPP)



Application Specific Processor (ASP)



Single purpose processor (SPP)

Example of different systems

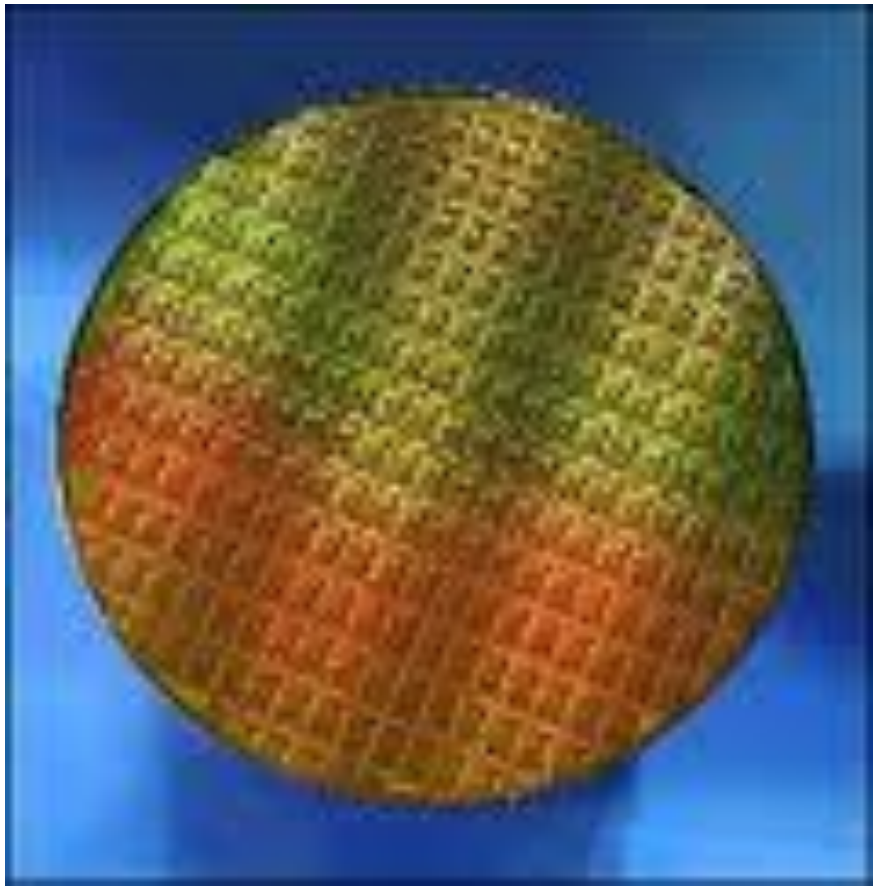
- General Purpose Processor (GPP)
  - Microprocessor
- Application Specific Processor (ASP/ASIP)
  - Microcontroller
  - Embedded microprocessor
  - Digital Signal Processor (DSP)
- Single Purpose Processor (SPP)
  - Controllers
  - Co-processor

Microprocessor vs. microcontroller

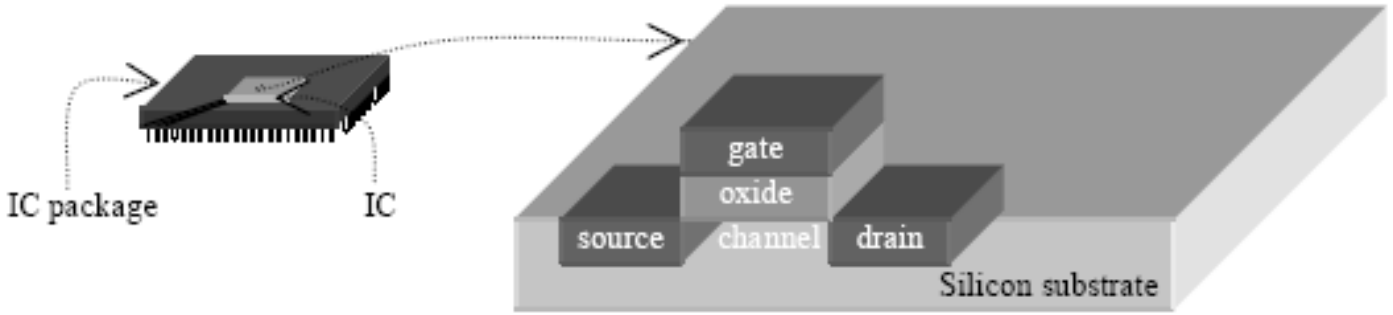
- Microprocessors generally require external components such as memory, ram, and input/output

- Microcontrollers incorporate program memory, ram, and input/output resources internal to the chip

Intel wafer



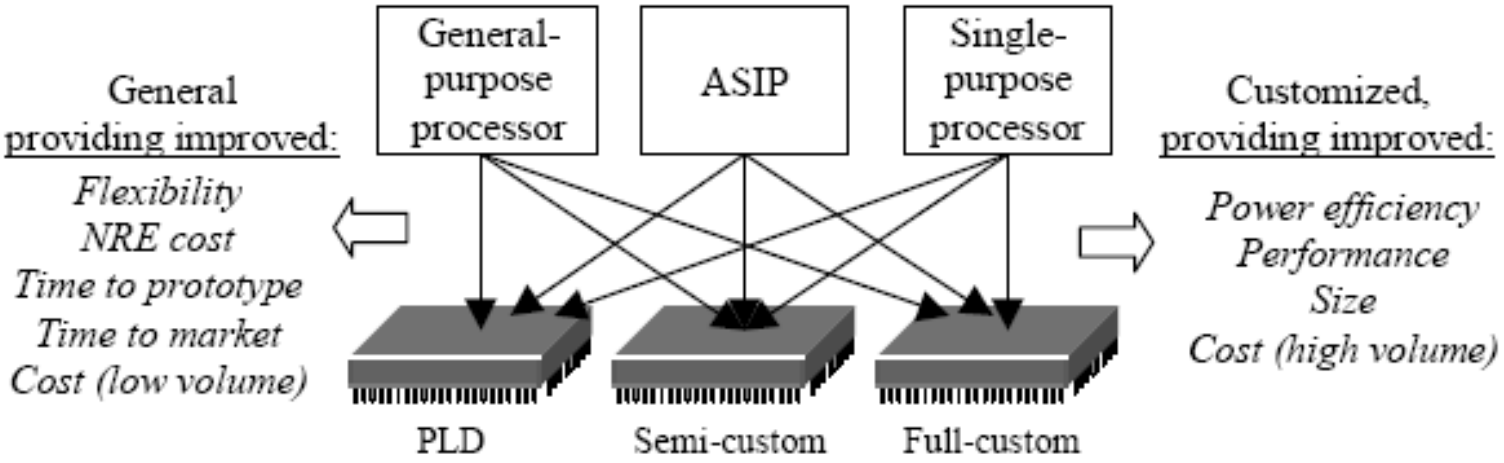
IC chip


IC package
IC
gate
oxide
source
channel
drain
Silicon substrate

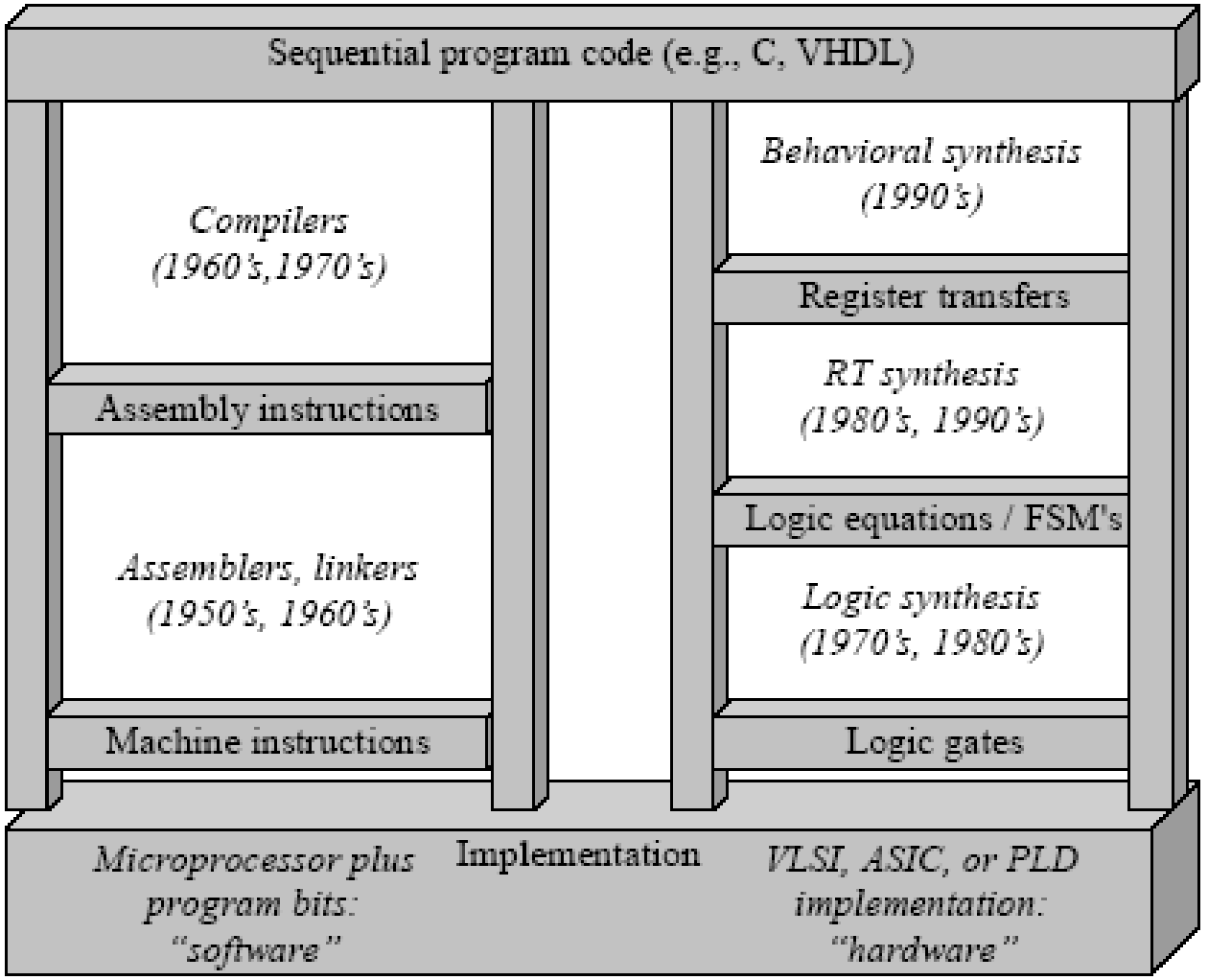
IC technology

- Full-custom/VLSI design
  - New mask design, excellent performance
- Semi-custom ASIC
  - Masks for transistor and gate levels are already built.
  - Main task is to connect gates together. Most popular IC technology
- Programmable Logic Device (PLD) such as FPGA or PLA
  - Programmable, good for rapid prototyping

Processor technology mapping with IC technology



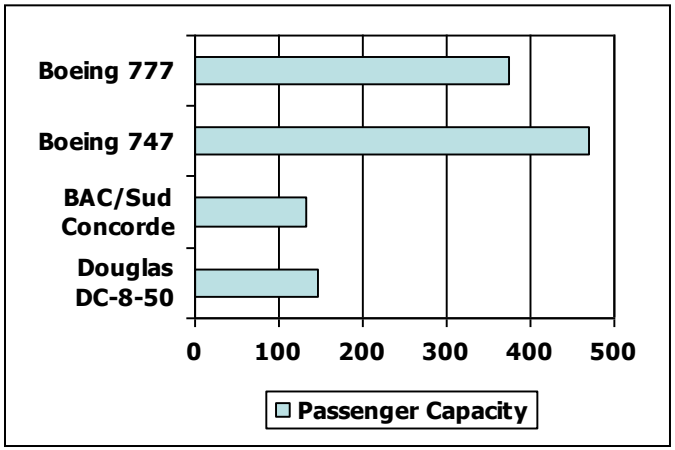
Co-design ladder

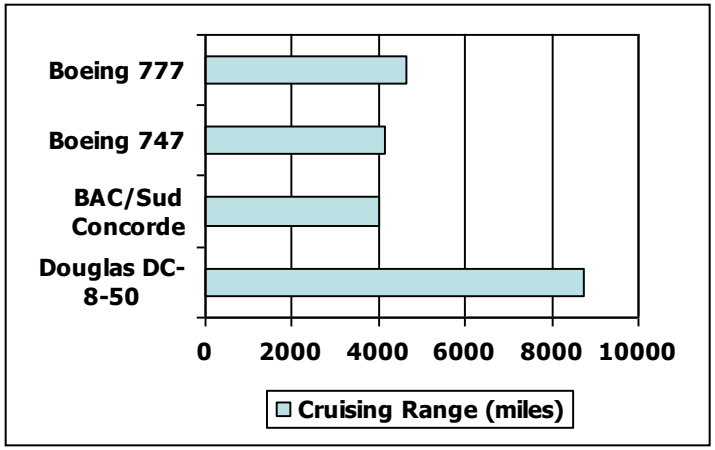


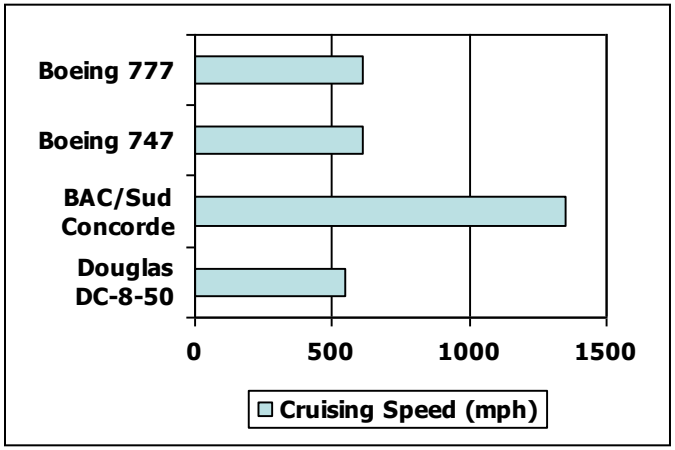
Performance

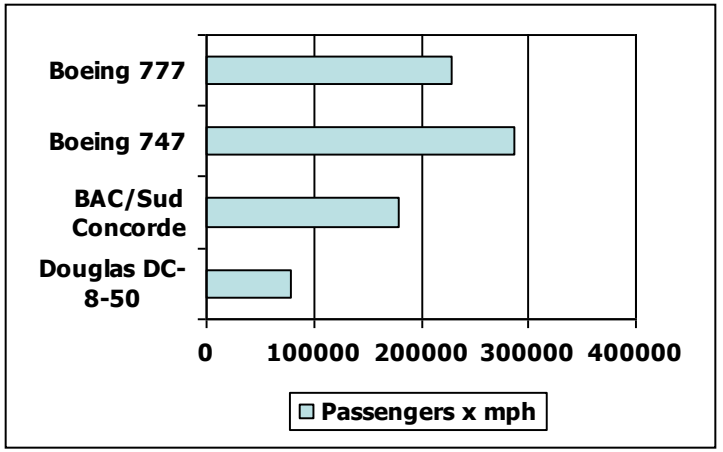
What is Performance?

- Which airplane has the best performance?

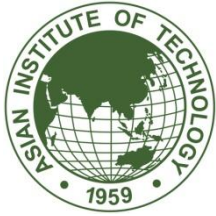








Throughput vs. Response Time



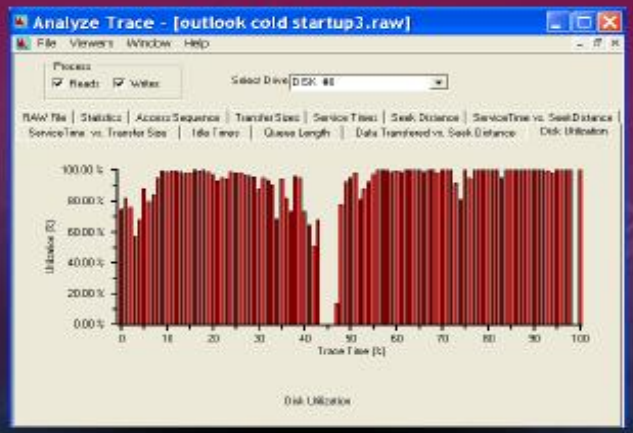
- Response time (execution time) – the time between the start and the completion of a task
  - Important to individual users
- Throughput (bandwidth) – the total amount of work done in a given time
  - Important to data center managers

Will need different performance metrics as well as a different set of applications to benchmark embedded and desktop computers, which are more focused on response time, versus servers, which are more focused on throughput

Response Time Matters

It's the Hard Disk, Stupid!

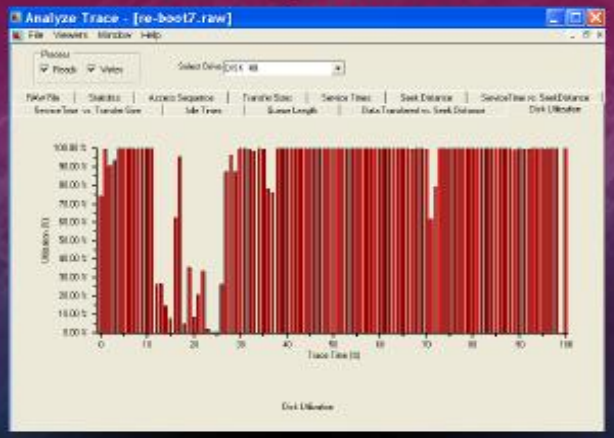
Re-Boot/Startup on Home PC



Elapsed Time	105.213536, s
Disk Busy Time	91.368480, s
Average Data Rate	6.60669, MB/s

86% BUSY

Starting Outlook



Elapsed Time	45.700667, s
Disk Busy Time	41.056997, s
Average Data Rate	1.37389, MB/s

89% BUSY

intel

Justin Rattner's ISCA'08 Keynote (VP and CTO of Intel)

Defining (Speed) Performance

- To maximize performance, need to minimize execution time

$$\text{performance}_X = 1 / \text{execution\_time}_X$$

If X is n times faster than Y, then

$$\frac{\text{performance}_X}{\text{performance}_Y} = \frac{\text{execution\_time}_Y}{\text{execution\_time}_X} = n$$

- **Decreasing response time almost always improves throughput**

Relative Performance Example

- If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?

We know that A is n times faster than B if

$$\frac{\text{performance}_A}{\text{performance}_B} = \frac{\text{execution\_time}_B}{\text{execution\_time}_A} = n$$

The performance ratio is

$$\frac{15}{10} = 1.5$$

So A is 1.5 times faster than B

Performance Factors

- CPU execution time (CPU time) – time the CPU spends working on a task
  - Does not include time waiting for I/O or running other programs

CPU execution time for a program = **# CPU clock for a program** x **clock cycle time**

or

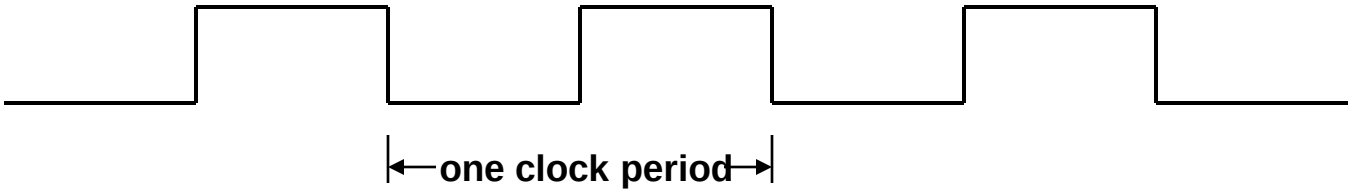
$$\textbf{CPU execution time for a program} = \frac{\textbf{\# CPU clock cycles for a program}}{\textbf{clock rate}}$$

- **Can improve performance by reducing either the length of the clock cycle or the number of clock cycles required for a program**

Review: Machine Clock Rate

- Clock rate (clock cycles per second in MHz or GHz) is inverse of clock cycle time (clock period)

$$CC = 1 / CR$$



10 nsec clock cycle => 100 MHz clock rate

5 nsec clock cycle => 200 MHz clock rate

2 nsec clock cycle => 500 MHz clock rate

1 nsec (10^{-9}) clock cycle => 1 GHz (10^{9}) clock rate

500 psec clock cycle => 2 GHz clock rate

250 psec clock cycle => 4 GHz clock rate

200 psec clock cycle => 5 GHz clock rate

Improving Performance Example

- A program runs on computer A with a 2 GHz clock in 10 seconds. What clock rate must computer B run at to run this program in 6 seconds? Unfortunately, to accomplish this, computer B will require 1.2 times as many clock cycles as computer A to run the program.

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

$$\text{Total CPU clock cycles}_A = 10 \text{ sec} \times 2 \times 10^9 \text{ cycles/sec} = 20 \times 10^9 \text{ cycles}$$

$$\text{CPU time}_B = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{\text{clock rate}_B}$$

$$\text{clock rate}_B = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{6 \text{ seconds}} = 4 \text{ GHz}$$

Clock Cycles per Instruction

- Not all instructions take the same amount of time to execute
  - One way to think about execution time is that it equals the number of instructions executed multiplied by the average time per instruction

 # CPU clock cycles for a program = # Instructions for a program x Average clock cycles per instruction

- **Clock cycles per instruction (CPI) – the average number of clock cycles each instruction takes to execute**
  - **A way to compare two different implementations of the same ISA**

	CPI for this instruction class		
	A	B	C
CPI	1	2	3

Using the Performance Equation

- Computers A and B implement the same ISA. Computer A has a clock cycle time of 250 ps and an effective CPI of 2.0 for some program and computer B has a clock cycle time of 500 ps and an effective CPI of 1.2 for the same program. Which computer is faster and by how much?

Each computer executes the same number of instructions, *I*, so

$$\text{CPU time}_A = I \times 2.0 \times 250 \text{ ps} = 500 \times I \text{ ps}$$

$$\text{CPU time}_B = I \times 1.2 \times 500 \text{ ps} = 600 \times I \text{ ps}$$

Clearly, A is faster … by the ratio of execution times

$$\frac{\text{performance}_A}{\text{performance}_B} = \frac{\text{execution\_time}_B}{\text{execution\_time}_A} = \frac{600 \times I \text{ ps}}{500 \times I \text{ ps}} = 1.2$$

Effective (Average) CPI

- Computing the overall effective CPI is done by looking at the different types of instructions and their individual cycle counts and averaging

$$\textbf{Overall effective CPI} = \sum_{i=1}^{n} (CPI_i \times IC_i)$$

  - **Where IC_i is the count (percentage) of the number of instructions of class i executed**
  - **CPI_i is the (average) number of clock cycles per instruction for that instruction class**
  - **n is the number of instruction classes**

- **The overall effective CPI varies by instruction mix – a measure of the dynamic frequency of instructions across one or many programs**

THE Performance Equation

- Our basic performance equation is then

$$\text{CPU time} = \text{Instruction\_count} \times \text{CPI} \times \text{clock\_cycle}$$

or

$$\text{CPU time} = \frac{\text{Instruction\_count} \times \text{CPI}}{\text{clock\_rate}}$$

- **These equations separate the three key factors that affect performance**
  - **Can measure the CPU execution time by running the program**
  - **The clock rate is usually given**
  - **Can measure overall instruction count by using profilers/ simulators without knowing all of the implementation details**
  - **CPI varies by instruction type and ISA implementation for which we must know the implementation details**

Instruction Count

What is the instruction count of this program?

```
                mov r2, #0
Label1:         add r1, r2, r3
                add r4, r5, r6
                mul r7, r1, r2
                add r2, r2, 1
                add r5, r5, 4
                bne r2, 10, Label1
                halt
```

Determinates of CPU Performance

CPU time = Instruction_count x CPI x clock_cycle

	Instruction_ count	CPI	clock_cycle
Algorithm	X	X	
Programming language	X	X	
Compiler	X	X	
ISA	X	X	X
Core organization		X	X
Technology			X

A Simple Example

Op	Freq	CPI_i	Freq x CPI_i			
ALU	50%	1	.5	.5	.5	.25
Load	20%	5	1.0	.4	1.0	1.0
Store	10%	3	.3	.3	.3	.3
Branch	20%	2	.4	.4	.2	.4
			Σ = 2.2	1.6	2.0	1.95

- How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?

  CPU time new = 1.6 x IC x CC so 2.2/1.6 means 37.5% faster

- How does this compare with using branch prediction to shave a cycle off the branch time?

  CPU time new = 2.0 x IC x CC so 2.2/2.0 means 10% faster

- What if two ALU instructions could be executed at once?

  CPU time new = 1.95 x IC x CC so 2.2/1.95 means 12.8% faster

CPU Performance

- Two common measures
  - Latency (how long to do X)
    - Also called response time and execution time
  - Throughput (how often can it do X)
- Example of car assembly line
  - Takes 6 hours to make a car (latency is 6 hours)
  - A car leaves every 30 minutes (throughput is 2 cars per hour)
  - Overlap results in Throughput > 1/Latency

Measuring Performance

- Peak (MIPS, MFLOPS)
  - Often not useful
    - unachievable in practice, or unsustainable
- Benchmarks
  - Real applications and application suites
    - E.g. SPEC95, SPEC CPU2000, TPC-C, TPC-H
  - Kernels
    - “Representative” parts of real applications
    - Easier and quicker to set up and run
    - Often not really representative of the entire app
  - Toy programs, synthetic benchmarks, etc.
    - Not very useful for reporting
    - Sometimes used to test/stress specific functions/features

CPU Performance Equation (1)

$$\text{CPU time} = \text{CPU Clock Cycles} \times \text{Clock cycle time}$$

$$\text{CPU time} = \text{Instruction Count} \times \text{Cycles Per Instruction} \times \text{Clock cycle time}$$

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

ISA, Compiler Technology

Organization , ISA

Hardware Technology, Organization

A.K.A. The "iron law" of performance

Car Analogy

- Need to drive from AIT to Victory Monument
  - "Clock Speed" = 3500 RPM
  - "CPI" = 0.994 rotations/ft *or* 1.006 ft/rot
  - "Insts" = 25 miles

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

$$\textbf{25 miles} \times \frac{\textbf{1 rotation}}{\textbf{1.006 feet}} \times \frac{\textbf{1 minute}}{\textbf{3500 rotations}}$$

= 0.625 hours or 37.5 minutes

CPU Version

- Program takes 33 billion instructions to run
- CPU processes insts at 2 cycles per inst
- Clock speed of 3GHz

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

Sometimes clock cycle time given instead (ex. cycle = 333 ps)

IPC sometimes used instead of CPI

= 22 seconds

CPU Performance Equation (2)

$$\text{CPU time} = \text{CPU Clock Cycles} \times \text{Clock cycle time}$$

$$\text{CPU time} = \left(\sum_{i=1}^{n} IC_i \times CPI_i \right) \times \text{Clock cycle time}$$

For each kind of instruction

How many instructions of this kind are there in the program

How many cycles it takes to execute an instruction of this kind

Car Analogy Again

Gear	Distance (miles)	Feet/ Rotation
1st	1.0	0.5
2nd	3.5	0.7
3rd	1.2	1.0
4th	1.8	1.2

$$\text{CPU time} = \left(\sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i \right) \times \text{Clock cycle time}$$

1.0 miles $\times$ $\frac{\textbf{1 rotation}}{\textbf{0.5 feet}}$ **10,560 rotations**

3.5 miles $\times$ $\frac{\textbf{1 rotation}}{\textbf{0.7 feet}}$ **26,400 rotations**

1.2 miles $\times$ $\frac{\textbf{1 rotation}}{\textbf{1.0 feet}}$ **6,336 rotations**

1.8 miles $\times$ $\frac{\textbf{1 rotation}}{\textbf{1.2 feet}}$ **7,920 rotations**

Σ = 51,216 rotations

divide by 3500 RPM

14.63 minutes

CPU Version

Instruction Type	Frequency	CPI
Integer	40%	1.0
Branch	20%	4.0
Load	20%	2.0
Store	20%	3.0

$$\text{CPU time} = \left(\sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i \right) \times \text{Clock cycle time}$$

Total Insts = 50B, Clock speed = 2 GHz

What is CPU time?

Comparing Performance

- "X is n times faster than Y"

$$\frac{\text{Execution time}_{Y}}{\text{Execution time}_{X}} = n$$

- "Throughput of X is n times that of Y"

$$\frac{\text{Tasks per unit time}_{X}}{\text{Tasks per unit time}_{Y}} = n$$

If Only it Were That Simple

- "X is n times faster than Y *on A*"

$$\frac{\text{Execution time of app A on machine Y}}{\text{Execution time of app A on machine X}} = n$$

- But what about different applications (or even parts of the same application)
  - X is 10 times faster than Y on A, and 1.5 times on B, but Y is 2 times faster than X on C, and 2 times on D, and…

Which would you buy?

So does X have better performance than Y?

Workloads and Benchmarks

- Benchmarks – a set of programs that form a “workload” specifically chosen to measure performance
- SPEC (System Performance Evaluation Cooperative) creates standard sets of benchmarks starting with SPEC89. The latest is SPEC CPU2006 which consists of 12 integer benchmarks (CINT2006) and 17 floating-point benchmarks (CFP2006).

www.spec.org

- There are also benchmark collections for power workloads (SPECpower_ssj2008), for mail workloads (SPECmail2008), for multimedia workloads (mediabench), …

Why SPEC?

SPEC CINT2006 on Barcelona (CC = 0.4 x 10^{-9})

Name	ICx10^9	CPI	ExTime	RefTime	SPEC ratio
perl	2,1118	0.75	637	9,770	15.3
bzip2	2,389	0.85	817	9,650	11.8
gcc	1,050	1.72	724	8,050	11.1
mcf	336	10.00	1,345	9,120	6.8
go	1,658	1.09	721	10,490	14.6
hmmer	2,783	0.80	890	9,330	10.5
sjeng	2,176	0.96	837	12,100	14.5
libquantum	1,623	1.61	1,047	20,720	19.8
h264avc	3,102	0.80	993	22,130	22.3
omnetpp	587	2.94	690	6,250	9.1
astar	1,082	1.79	773	7,020	9.1
xalancbmk	1,058	2.70	1,143	6,900	6.0
Geometric Mean					**11.7**

TPC Benchmarks

- Measure transaction-processing throughput
- Benchmarks for different scenarios
  - TPC-C: warehouses and sales transactions
  - TPC-H: ad-hoc decision support
  - TPC-W: web-based business transactions
- Difficult to set up and run on a simulator
  - Requires full OS support, a working DBMS
  - Long simulations to get stable results

What's about Embedded System bench?

Mibench: auto,consumer,office,network,security, telcomm applications

EEMBC Coremark: Autobench, Consumerbench, Energybench

ParMibench: Multi-processor based embedded systems

mobilemark: laptop computing including document reader, flash, movie player

Summarizing Performance

- Arithmetic mean
  - Average execution time
  - Gives more weight to longer-running programs
- Weighted arithmetic mean
  - More important programs can be emphasized
  - But what do we use as weights?
  - Different weight will make different machines look better (see Figure 1.16)

Questions?