

CS352 – Fall 2008

Introduction

Herb Schwetman
Adjunct Professor

Preliminaries

- See handouts
 - Introduction
 - How to pass this course
- No class on Monday (Labor Day)
- Will need to use C programming language
 - Get ready
- List server, etc.

So what is this course about?

- Computer architecture
 - Machine instructions (assembly language)
 - Instruction execution
 - Processor(s) and how they work (at a high level)
 - Memory hierarchy
 - Most important – PERFORMANCE
 - Speed of execution of programs

Cost vs. Performance

- Every system represents a compromise between cost and performance
 - Supercomputer – cost is no constraint (maybe)
 - Performance is critical
 - First IBM PC – cost was critical
 - Performance sacrificed
 - Today, picture is more complex
 - Powerful, inexpensive microprocessor chips
 - Large, inexpensive memory and disk drives

Cost/Performance (con't.)

- Estimating cost is relatively straightforward
- How do we measure performance?
- How do we predict performance?
 - If system does not exist yet
 - If system is changed
- The unifying theme of this course is system performance

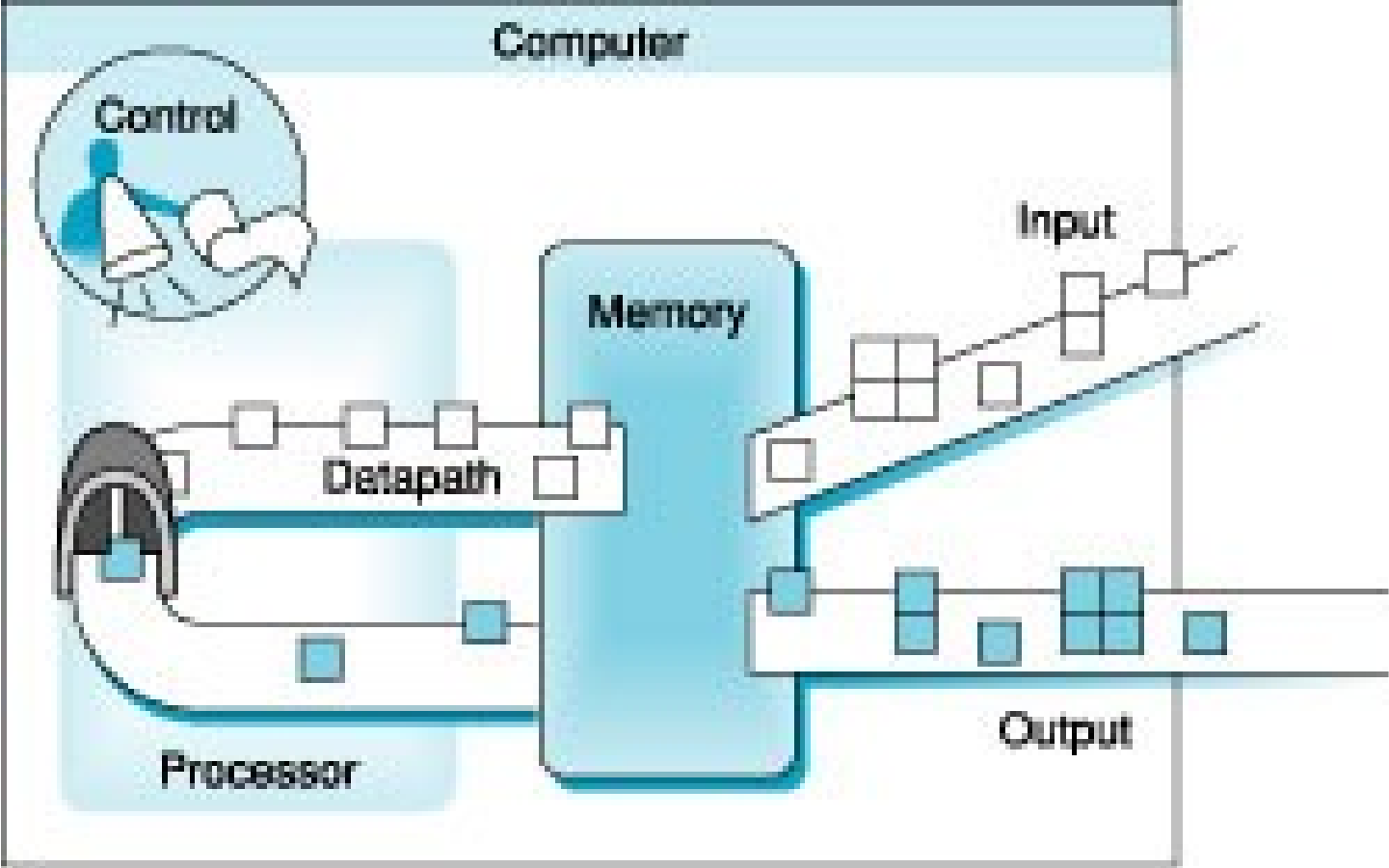
Computer

- Does automatic computation
- Automatic:
 - Without intervention, self-sustaining, “by itself”
- Computation:
 - Arithmetic: add, subtract, multiply, divide
 - Logical: and, or, xor, complement (invert)
- Sequencing (flow of control)
 - Branch (unconditional), jump
 - Conditional branch (test a condition)
- Input/output
- Note: getting the right answer is required (not optional)

Computer System Components

- Datapath: components of CPU that hold information and do computation
- Control: tells datapath what to do and when
 - Guided by machine instructions
- Memory: holds information (external to CPU) (read and write)
- Input: provides external data to CPU (read)
- Output: accepts data from CPU (write)

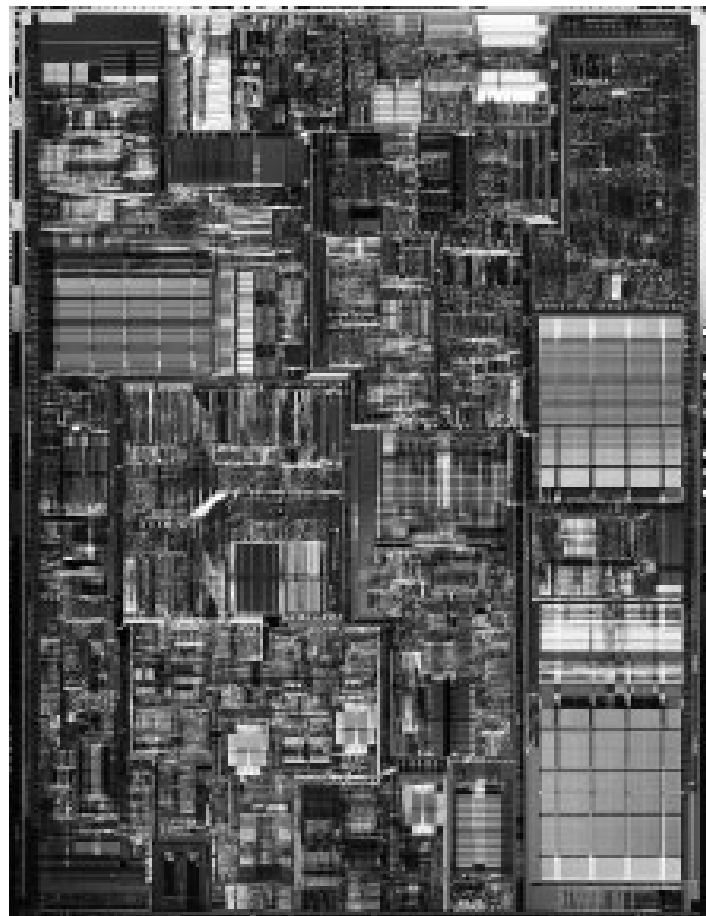
Components



Microprocessors

- Single chip (computer on a chip – almost)
 - Millions of transistors (switching elements)
 - Logic (instruction execution)
 - Memory: registers and level 1 cache
 - Special functional units (video, floating point)
 - Off chip communications (L2 cache and/or main memory, I/O, etc.)
- High frequency
 - 2.0 – 3.5 Ghz (billion clock cycles/second)

Intel Pentium 4 Chip



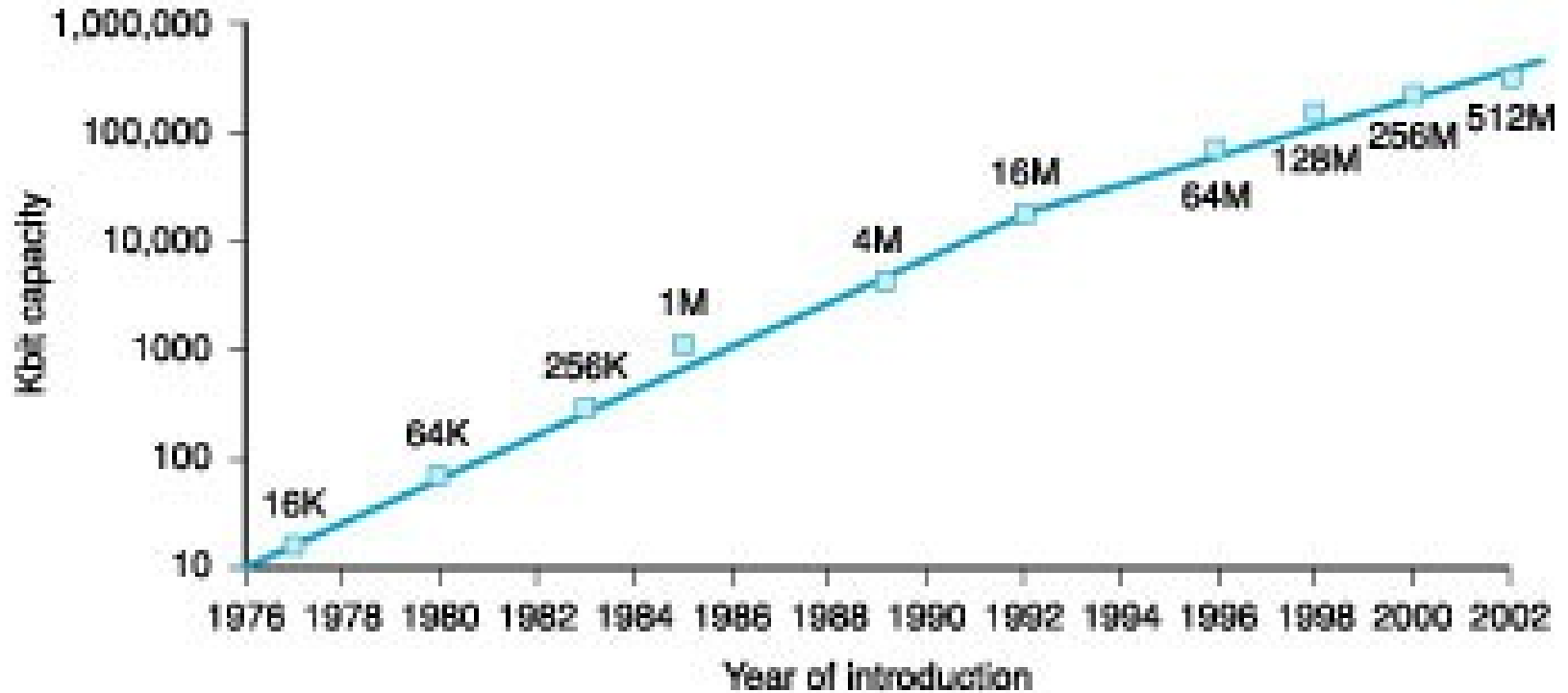
Trends

- Since 1980
 - Dramatic improvements
 - transistors/chip
 - switching rates (clock frequencies)
 - DRAM (memory chip) capacities
 - Disk drive capacities
 - Costs: dramatically decreased
 - Today
 - parallel operation (multi-core, multi-thread, etc.)

Trends

Year	Technology used	Rel.Perf/unit cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit	900
1995	Very LSI	2,400,000
2005	Ultra LSI	6,200,000,000

DRAM Chip Growth



The MIPS Processor

- Introduced in the 1980's
 - Designed by Hennessy (Stanford Univ.)
 - An example of a RISC processor
 - RISC: reduced instruction set computer
- 32-bit and 64-bit versions
- 32 general purpose registers plus floating point registers
- Load/store architecture

MIPS Processor

- Used throughout the textbook and this course
- Will be covered in subsequent lectures
- Used by SGI systems (formerly)
- Widely used in embedded systems

MIPS Instructions

- All instructions are 32 bits
- All arithmetic instructions specify 3 operands
 - Two inputs, one output: $\$s0 \leftarrow \$s1 + \$s2$
 - Immediate value: $\$s0 \leftarrow \$s1 + 0x1234$
- Only memory accesses are loads or stores
 - lw $\$s0, 0x1234(\$s1) \# \$s0 \leftarrow M[0x1234 + \$s1]$
 - sw $\$s0, 0x1234(\$s1) \# M[0x1234 + \$s1] \leftarrow \$s0$

For-Loops and Arrays

- Loops are of great importance in accessing computer architectures (performance)
- Why?
 - most computational work expressed in for-loops manipulating data stored in arrays
 - many hardware optimizations target for-loops
 - many compiler optimizations target for-loops

For-Loops

- `for(i = 0; i < 100; i++) {`
 - loop body `}`
- **Parts**
 - `i = 0`; initializer: executed once at the beginning of the loop
 - `i < 100`; termination condition: executed **BEFORE** execution of loop body; if condition is true, execute loop body; if false, exit loop
 - `i++`; increment: executed at end of loop body

Arrays

- `int a[100];`
- allocate $100 * \text{sizeof}(\text{int})$ bytes
 - 100 elements, each of length $\text{sizeof}(\text{int}) - 4$ bytes
 - total length is 400 bytes
- address of first byte = address of `a[0]`
- address of `a[i]` = $\text{address}(\text{a}[0]) + i * \text{sizeof}(\text{int})$
= $\text{address}(\text{a}[0]) + 4 * i$

Example (in C)

- `#define N 10000`
- `int a[N];`
- `for(i = 0; i < N; i++) {`
 - `a[i] = i;`
 - `}`

Sort of like

- unsigned char *addr;
- unsigned char *a;
- a = malloc(sizeof(int)*N);
- i = 0;
- startLoop:
 - if(i >= N) goto done;
 - addr = a + sizeof(int)*i;
 - *addr = i;
 - goto startLoop;
- done:

Optimizations

- reduce number of instructions and computations
- unwind loop (assume N is multiple of 2)
 - for($i = 0; i < N; i += 2$) {
 - $a[i] = i;$
 - $a[i+1] = i+1;$
 - }
 - do more work per iteration and fewer iterations

Parallelization

- (this is unrealistic – need to do more work per iteration):
 - do “n” assignments per iteration
- for($i = 0; i < N; i += n$)
 - invokeProcess($i, i + n - 1$);
- for($i = 0; i < N; i += n$)
 - waitProcess(i);

Next Meeting

- Read Chapter 1 in the text book
- Get on a system; create, compile and execute a C program (use example from class)
- Next class
 - C programming
 - Instruction architectures
 - MIPS instructions with examples