**Computer Architecture: Exciting Times Ahead!**

Prediction is very difficult, especially about the future.  
-- Niels Bohr

The best way to predict the future is to invent it.  
-- Alan Kay

---

**You’ve mastered a lot…**

Fets & voltages

Logic gates

Combinational logic circuits

Sequential logic

Combinational contract:
- discrete-valued inputs
- complete in/out spec.
- static discipline

Acyclic connections

Summary specification

Design:
- sum-of-products
- simplification
- mixes, ROMs, PLAs

Storage & state

Dynamic discipline

Finite-state machines

Metastability

Throughput & latency

Pipelining

---

**… a WHOLE lot …**

Sequential logic

CPU Architecture

Computer Systems

Computing Theory

Instruction Set Architectures

Beta implementation

Pipelined Beta

Software conventions

Memory architectures

Interconnect

Virtual machines

Interprocess communication

Operating Systems

Real time, Interrupts

Parallel Processing

---

**What’s next?**

Some follow-on options…

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.111 (G) Introductory Digital Systems Laboratory</td>
<td>6.823 (G) Computer System Architecture</td>
</tr>
<tr>
<td>6.115 (U) Microcomputer Project Laboratory</td>
<td></td>
</tr>
</tbody>
</table>

LA for 6.004

Special Topics

UROP
Things to look forward to…

6.004 is only an appetizer!

Verilog example: Beta Register File

module regfile(ra1, rd1, ra2, rd2, clk, wef, wa, wd);
input [4:0] ra1; // address for read port 1 (Reg[RA])
output [31:0] rd1; // read data for port 1
input [4:0] ra2; // address for read port 2 (Reg[RB], Reg[RC] for ST)
output [31:0] rd2; // read data for port 2
input clk;
input wef; // write enable, active high
input [4:0] wa; // address for write port (Reg[RC])
input [31:0] wd; // write data
reg [31:0] registers[31:0]; // the register file itself (local)

// read paths are combinational, check for reads from R31
assign rd1 = (ra1 == 31) ? 0 : registers[ra1];
assign rd2 = (ra2 == 31) ? 0 : registers[ra2];

// write port is active only when WERF is asserted
always @(posedge clk)
if (wef) registers[wa] <= wd;
endmodule

The Crystal Ball

some trends in computer evolution

- Technology shrinks
  - 30% linear shrink/generation
  - Cheaper, faster, lower power
- Multicores (SMP, Tiled NUMA, …)
- Superscalar/SMT pipelines
- Power management
- Reconfigurable processing/interconnect
- VLIW, SIMD influences
2010 Architecture?

Giant uniprocessors (maybe with SMT) remain popular in markets where software is the main expense.

- **Embedded DRAM**
- **Tiles**
- **Input/Output (I/O)**
- **Off-chip DRAM**

Tiles (VLIW/reconfigurable/vector) machines become popular in systems with resource constraints (hardware cost, low power, hard real time).

- 10 GHz processor clock
- 5 GHz network clock
- 128 processing tiles
- >5 TFLOPS peak (32b FLOPS)
- >40 TOPS peak (8b OPS)
- 1GB on-chip DRAM
- 100 GB/s off-chip DRAM interface
- 100 GB/s I/O
- 25x25mm² in 0.045μm CMOS

Thinking Outside the Box

Will computers always look and operate the way computers do today?

Some things to question:

- Well-defined system “state”
- Programming
- Silicon-based logic
- Logic at all

Our programming hangup


Is learning an alternative to programming?

Wet Computers

1) The most reliable, sustainable, efficient, and smartest??? machines that we know of are biological
2) Fined tuned through millions of years of evolution
3) The assembly, repair, and operation “instructions” for multi-billion element machines are “digitally” encoded in a single molecule
4) We are just beginning to understand the “gates” and the “machine language”

Is there an engineering discipline for building goal-oriented systems from goal-oriented components?

I wonder if 2289384-1 is prime?
**DNA Chips**
(DNA probes or microarrays)

- Leverages VLSI fabrication techniques (photolithography)
- Use PCRs (polymerase chain reactions) to make an exponential number of DNA copies
- Mechanically bind specific “tagged” gene sequences onto a patterned substrate
- Expose to bath of denatured nucleotides (separated and diced up pieces of DNA)
- Look for phosphorescent markers

Medical applications are obvious, but what does it have to do with computation?

Questions:
What inputs satisfy \( f(x_1, x_2, \ldots, x_N) = 1 \).

---

**Can we Program Microbes?**

DNA = program

Protein synthesis = gates?

Can we “engineer” organisms to perform computations for us?

Can we make a “standard cell library” offering digital building blocks from DNA sequences?

This is alien thinking for biologist, but standard fare for systems designers

\[ F(n) = n \cdot F(n-1); \]
\[ F(0) = 1 \]

---

**Computing at the limit**

At the particle level nature behaves very strangely…

Far separated particles can be entangled
- electron spins
- photon polarizations
- magnetic fields

They can be simultaneously in either state (so long as you don’t look).

The act of looking at them (measuring, or observing them) forces the entangled particle into one of its states.

Strangely enough, it is believed that we can use such entangled particles in computations w/o disturbing them.

---

**Quantum Computing?**

Classic computers perform operations on strings of bits (0s and 1s).

A quantum computer would be able to compute on bits (qubits) that can be simultaneously in either state.

**Classic computer:**
(with a dumb algorithm)
Search through all \( 2^{20} \) permutations

**Quantum computer:**
Insert 20 qubits, select the desired answer, then look back and see what the qubits resolved to...

\[ F(0 < x < 2^{20}) = x \cdot 371 \]

\[ F(?) = 197001 \]
The Dilemma

- We have no clue how to build a practical quantum computer
- Currently, quantum computing is merely a fantasy of theoreticians
- What other problems can a quantum computer solve more efficiently than a classic computer?

A SUBTLE Reminder:
Turing, Church, Post, Kleene, and Markov really “invented” most of modern day computer science long before a “practical” implementation.

6004: The Big Lesson

You’ve built, debugged, understood a complex computer from FETs to OS… what have you learned?

Engineering Abstractions:
- Understanding of their technical underpinnings
- Respect for their value
- Techniques for using them

But, most importantly:
- The self assurance to discard them, in favor of new abstractions!

Good engineers use abstractions;
GREAT engineers create them!

THE END!

Pens, pencils, paper they attempt to solve problems that teachers set forth.
The only problem with Haiku is that you just get started and then