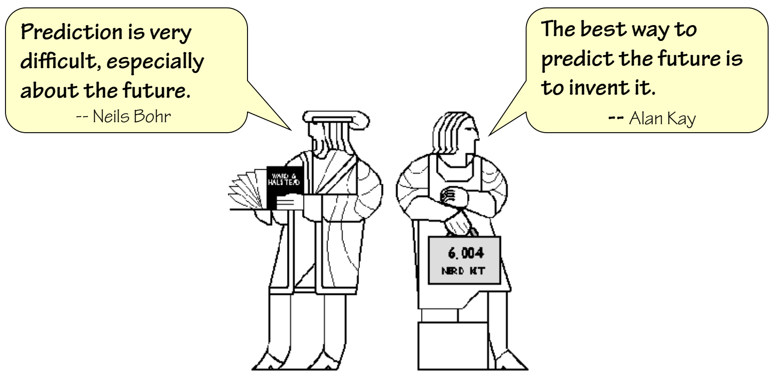
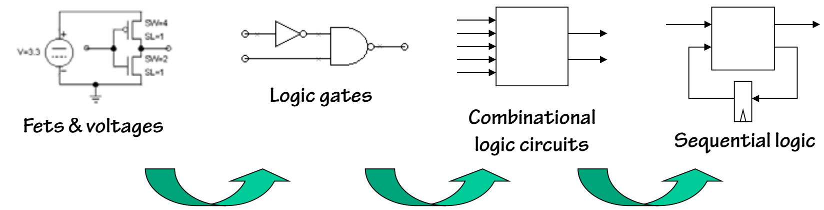
6.004 Computation Structures
Spring 2009

Computer Architecture: Exciting Times Ahead!



You've mastered a lot…

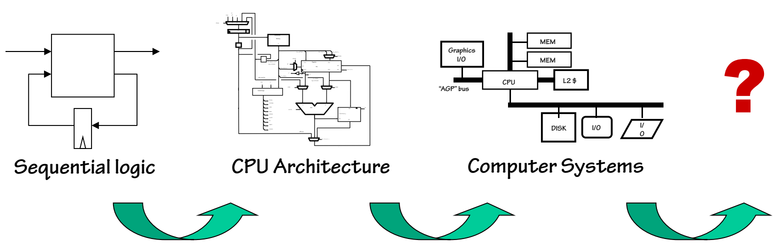


Combinational contract:
- discrete-valued inputs
- complete in/out spec.
- static discipline

Acyclic connections
Summary specification
Design:
- sum-of-products
- simplification
- muxes, ROMs, PLAs

Storage & state
Dynamic discipline
Finite-state machines
Metastability
Throughput & latency
Pipelining

… a WHOLE lot …

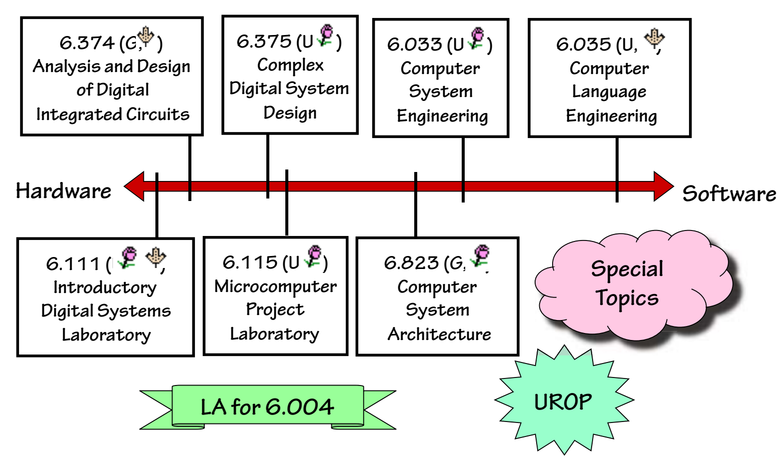


Computing Theory
Instruction Set Architectures
Beta implementation
Pipelined Beta
Software conventions
Memory architectures

Interconnect
Virtual machines
Interprocess communication
Operating Systems
Real time, Interrupts
Parallel Processing

What's next?

Some follow-on options…



Things to look forward to…

6.004 is only an appetizer!

Processors
- Superscalars
- Deep pipelines
- Multicores

Languages & Models
- Python/Java/Ruby/…
- Objects/Streams/Aspects
- Networking

Tools
- Design Languages
- FPGA prototyping
- Timing Analyzers

Algorithms
- Arithmetic
- Signal Processing
- Language implementation

Systems Software
- Storage
- Virtual Machines
- Networking

Verilog example: Beta Register File

```
// 2-read, 1-write 32-location register file
module regfile(ra1,rd1,ra2,rd2,clk,werf,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 werf;               // 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 (werf) registers[wa] <= wd;
endmodule
```


PC

```
module pc(clk,reset,pcsel,offset,jump_addr,
          branch_addr,pc,pc_plus_4);
  input clk;
  input reset;              // forces PC to 0x80000000
  input [2:0] pcsel;        // selects source of next PC
  input [15:0] offset;      // inst[15:0]
  input [31:0] jump_addr;   // from Reg[RA], used in JMP instruction
  output [31:0] branch_addr; // send to datapath for LDR instruction
  output [31:0] pc;         // used as address for instruction fetch
  output [31:0] pc_plus_4;  // saved in regfile during branches, JMP, traps

  reg [31:0] pc;
  wire [30:0] pcinc;
  wire [31:0] npc;

  // the Beta PC increments by 4, but won't change supervisor bit
  assign pcinc = pc + 4;
  assign pc_plus_4 = {pc[31],pcinc};

  // branch address = PC + 4 + 4*sxt(offset)
  assign branch_addr = {0,pcinc + {{13{offset[15]}},offset[15:0],2'b00}};

  assign npc = reset ? 32'h80000000 :
               (pcsel == 0) ? {pc[31],pcinc} :                    // normal
               (pcsel == 1) ? {pc[31],branch_addr[30:0]} :        // branch
               (pcsel == 2) ? {pc[31] & jump_addr[31],jump_addr[30:0]} :  // jump
               (pcsel == 3) ? 32'h80000004 : 32'h80000008;        // illop, trap

  // pc register, pc[31] is supervisor bit and gets special treatment
  always @(posedge clk) pc <= npc;
endmodule
```

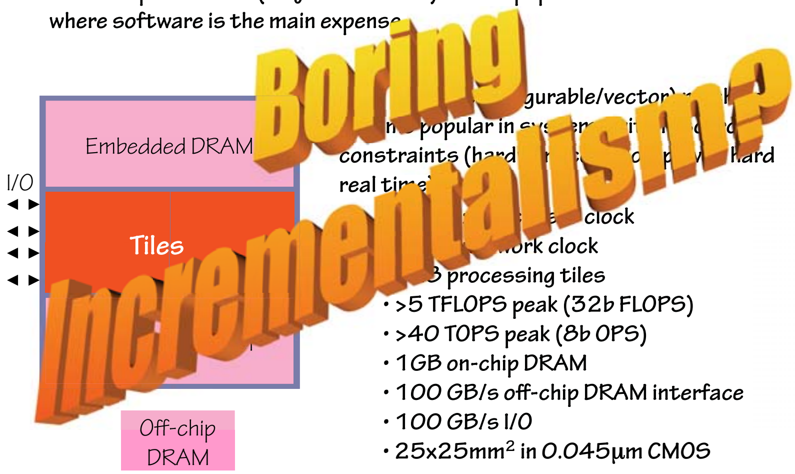

The Crystal Ball

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



...gurable/vector)...
...popular in sys... constraints (hard ... hard real time)
... clock
...work clock
...3 processing tiles

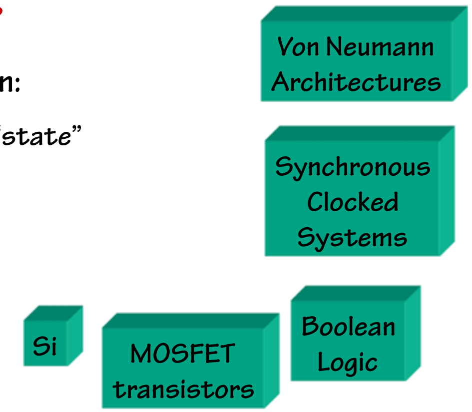
- >5 TFLOPS peak (32b FLOPS)
- >40 TOPS peak (8b OPS)
- 1GB on-chip DRAM
- 100 GB/s off-chip DRAM interface
- 100 GB/s I/O
- 25x25mm^2 in 0.045μm CMOS

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

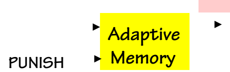


Our programming hangup

Our machines slavishly execute sequences of instructions. Does a cerebellum? A society? A beehive? An MIT student?

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

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”

I wonder if $2^{289384}-1$ 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?

We can reliably reslice and recombine (state machines?)

Questions:
What inputs satisfy
f(x1,x2,…xN) =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 * 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

$F(0 < x < 2^{20}) = x * 371$

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

$F(?) = 197001$

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