CS 152
Computer Architecture and Engineering
Lecture 25

The Final Chapter
A whirlwind retrospective on the term

א

May 10, 1999

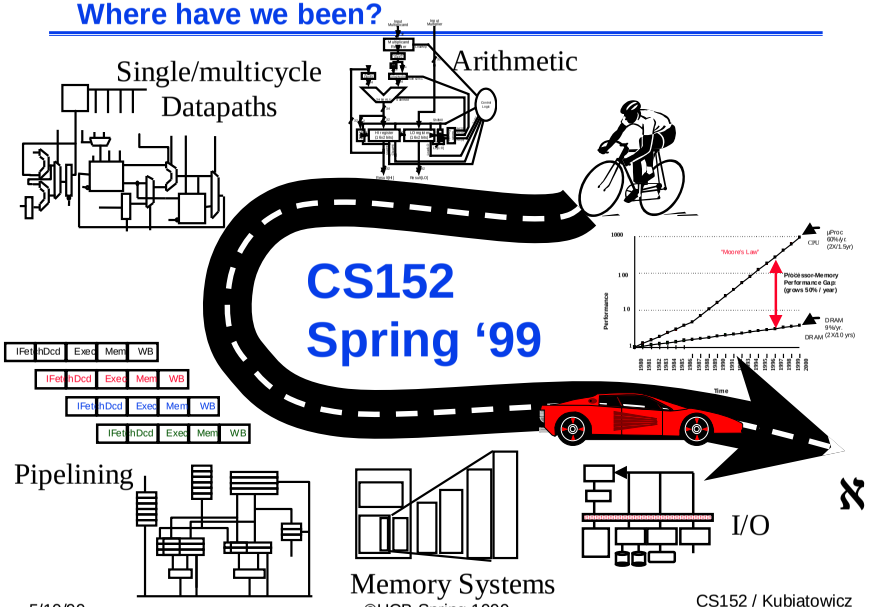
John Kubiatowicz (http.cs.berkeley.edu/~kubitron)

lecture slides: http://www-inst.eecs.berkeley.edu/~cs152/

Outline of Today's Lecture

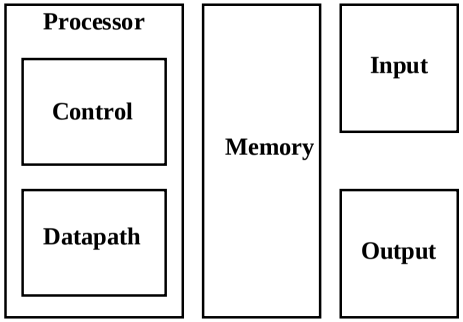
- Recap: What was covered in lectures 45 minutes)
- Questions and Administrative Matters (2 minutes)
- Future of Computer Architecture and Engineering (15 minutes)
- Lessons from CS 152 (10 minutes)
- HKN evaluation of teaching staff (15 minutes)

Where have we been?

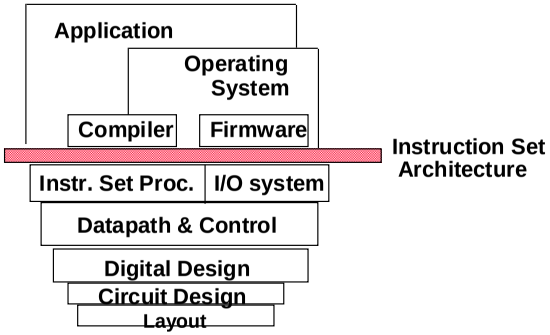


The Big Picture

- Since 1946 all computers have had 5 components

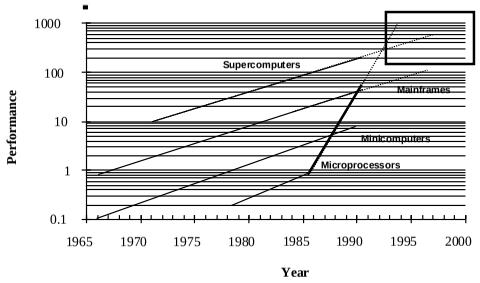


What is "Computer Architecture"?



- Coordination of many *levels of abstraction*
- Under a rapidly changing set of forces
- Design, Measurement, *and* Evaluation

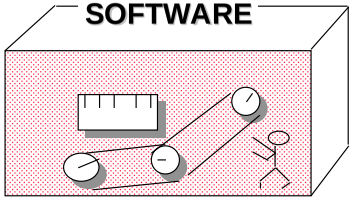
Performance and Technology Trends



- **Technology Power: 1.2 x 1.2 x 1.2 = 1.7 x / year**
  - Feature Size: shrinks 10% / yr. => Switching speed improves 1.2 / yr.
  - Density: improves 1.2x / yr.
  - Die Area: 1.2x / yr.
- **One lesson of RISC is to keep the ISA as simple as possible:**
  - Shorter design cycle => fully exploit the advancing technology (~3yr)
  - Advanced branch prediction and pipeline techniques
  - Bigger and more sophisticated on-chip caches

Instruction Set Architecture (subset of Computer Arch.)

... the attributes of a [computing] system as seen by the programmer, *i.e.* the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation. – Amdahl, Blaaw, and Brooks, 1964



-- Organization of Programmable Storage

-- Data Types & Data Structures: Encodings & Representations

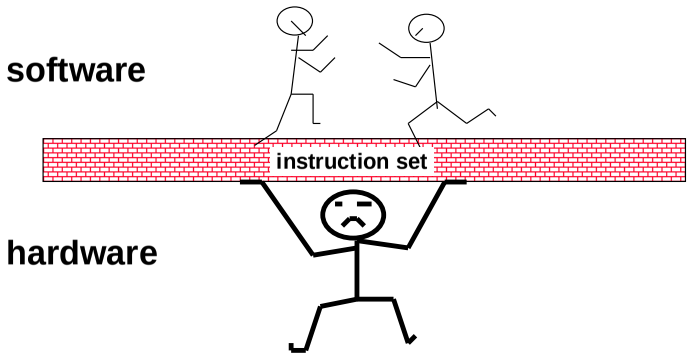
-- Instruction Set

-- Instruction Formats

-- Modes of Addressing and Accessing Data Items and Instructions

-- Exceptional Conditions

Instruction Set Design



Summary of the Design Process

Hierarchical Design to manage complexity

Top Down vs. Bottom Up vs. Successive Refinement

Importance of Design Representations:

- Block Diagrams
- Decomposition into Bit Slices
- Truth Tables, K-Maps
- Circuit Diagrams

top down

bottom up

mux design meets at TT

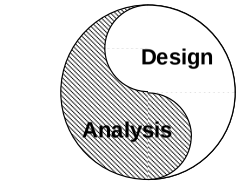
Other Descriptions: state diagrams, timing diagrams, reg xfer, . . .

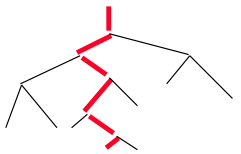
Optimization Criteria:

- Gate Count — *Area*
- [Package Count] — *Pin Out*
- Logic Levels — *Delay*
- Fan-in/Fan-out — *Cost*
- *Power*
- *Design time*

Measurement and Evaluation

Architecture is an iterative process
-- searching the space of possible designs
-- at all levels of computer systems





Creativity

Cost / Performance Analysis

You must be willing to throw out Bad Ideas!!

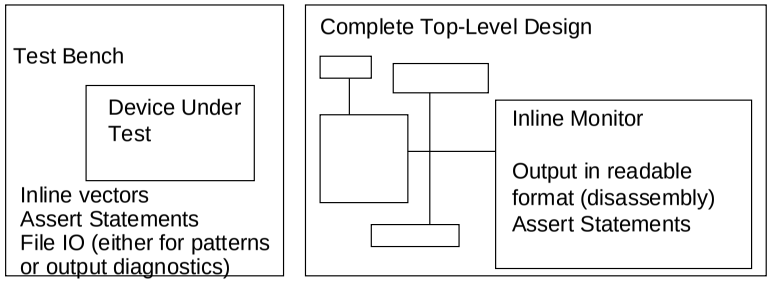
Good Ideas

Mediocre Ideas

Bad Ideas

One of most important aspects of design: TEST

- Think about testing from beginning of design
- Well over 50% of modern teams devoted to testing
- VHDL Test benches: monitoring hardware to aid debugging:
  - Include assert statements to check for “things that should never happen”



Why should you keep an design notebook?

- Keep track of the design decisions and the reasons behind them
  - Otherwise, it will be hard to debug and/or refine the design
  - Write it down so that can remember in long project: 2 weeks - >2 yrs
  - Others can review notebook to see what happened
- Record insights you have on certain aspect of the design as they come up
- Record of the different design & debug experiments
  - Memory can fail when very tired
- Industry practice: learn from others mistakes

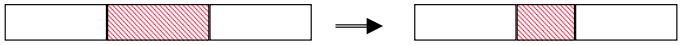
Basis of Evaluation

Pros		Cons
• representative	Actual Target Workload	• very specific • non-portable • difficult to run, or measure • hard to identify cause
• portable • widely used • improvements useful in reality	Full Application Benchmarks	•less representative
• easy to run, early in design cycle	Small "Kernel" Benchmarks	• easy to "fool"
• identify peak capability and potential bottlenecks	Microbenchmarks	• "peak" may be a long way from application performance

Amdahl's Law

Speedup due to enhancement E:

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



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

$$\text{ExTime(with E)} = ((1-F) + F/S) \times \text{ExTime(without E)}$$

$$\text{Speedup(with E)} = \frac{1}{(1-F) + F/S}$$

Performance Evaluation Summary

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

° ***Time is the measure of computer performance!***

° **Remember Amdahl's Law:**
Speedup is limited by unimproved part of program

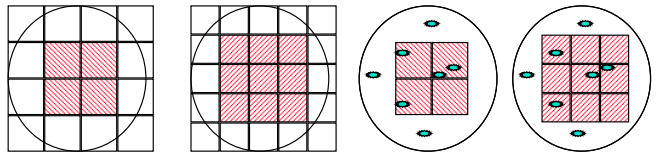
° **Good products created when have:**
- **Good benchmarks**
- **Good ways to summarize performance**

° **If NOT good benchmarks and summary, then choice between**
1) improving product for real programs
2) changing product to get more sales (sales almost always wins)

Integrated Circuit Costs

$$\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per Wafer} * \text{Die yield}}$$

$$\text{Dies per wafer} = \frac{\pi * (\text{Wafer_diam} / 2)^2}{\text{Die Area}} - \frac{\pi * \text{Wafer_diam}}{\sqrt{2 * \text{Die Area}}} - \text{Test dies} \approx \frac{\text{Wafer Area}}{\text{Die Area}}$$



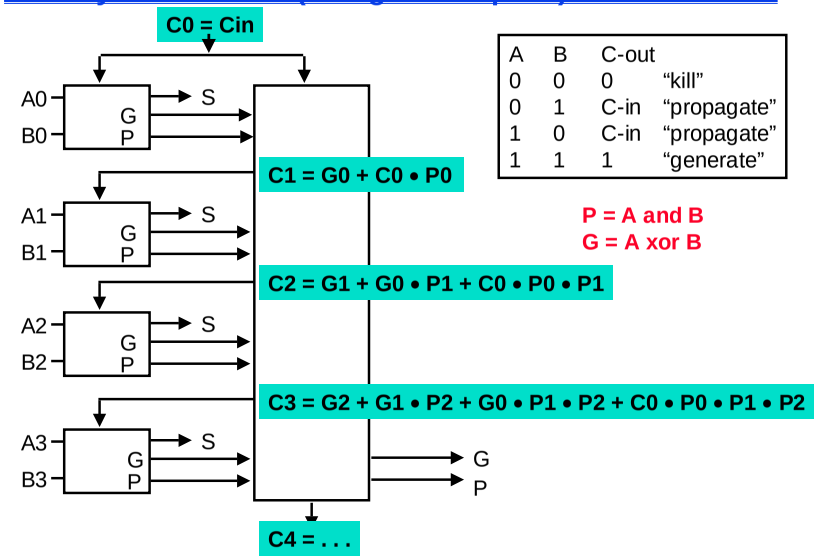
$$\text{Die Yield} = \frac{\text{Wafer yield}}{\left\{1 + \frac{\text{Defects_per_unit_area} * \text{Die_Area}}{\alpha}\right\}^{\alpha}}$$

Die Cost is goes roughly with the cube of the area.

Computer Arithmetic

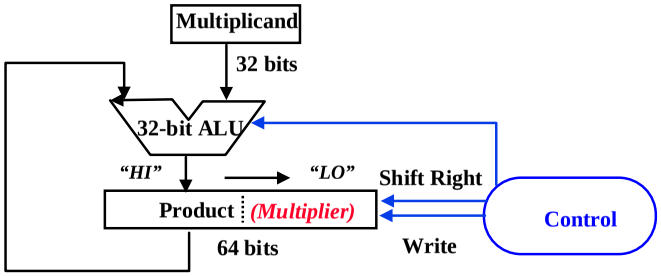
- **Bits have no inherent meaning: operations determine whether really ASCII characters, integers, floating point numbers**
- **Hardware algorithms for arithmetic:**
  - **Carry lookahead/carry save addition**
  - **Multiplication and divide.**
  - **Booth algorithms**
- **Divide uses same hardware as multiply (Hi & Lo registers in MIPS)**
- **Floating point follows paper & pencil method of scientific notation**
  - **using integer algorithms for multiply/divide of significands**

Carry Look Ahead (Design trick: peek)



MULTIPLY HARDWARE Version 3

- 32-bit Multiplicand reg, 32-bit ALU,
  64-bit Product reg (shift right), (0-bit Multiplier reg)



Divide can use same hardware!

Booth's Algorithm Insight

middle of run

end of run **0 1 1 1 1 0** beginning of run

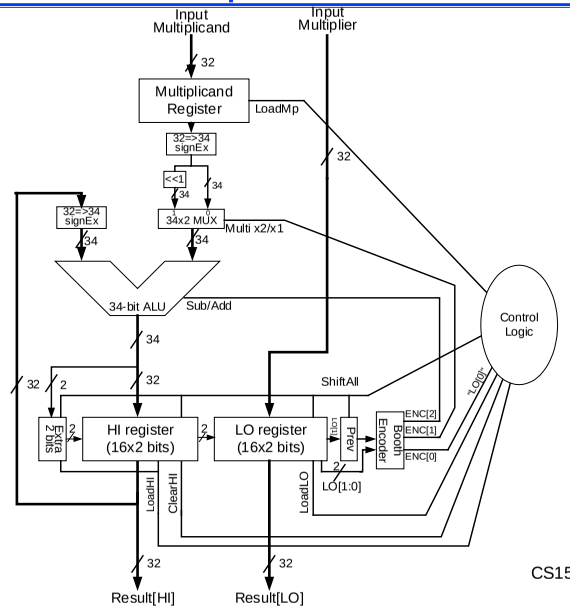
Current Bit	Bit to the Right	Explanation	Example	Op
1	0	Begins run of 1s	0001111000	sub
1	1	Middle of run of 1s	0001111000	none
0	1	End of run of 1s	0001111000	add
0	0	Middle of run of 0s	0001111000	none

Originally for Speed (when shift was faster than add)

- **Replace a string of 1s in multiplier with an initial subtract when we first see a one and then later add for the bit after the last one**

```
  −1
+ 10000
 01111
```

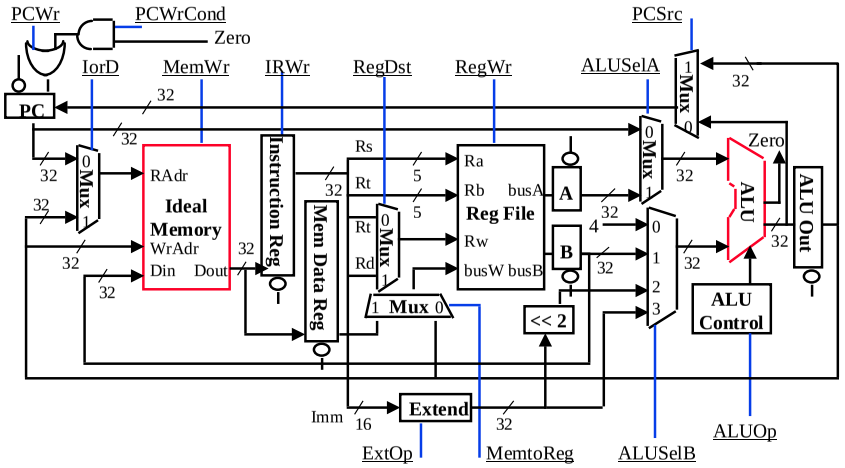
Double Bit Booth Multiplier



Pentium Bug

- Pentium: Difference between bugs that board designers must know about and bugs that potentially affect all users
  - –$200,000 cost in June to repair design
  - –$400,000,000 loss in December in profits to replace bad parts
  - –How much to repair Intel's reputation?
  - –Make public complete description of bugs in later category?
- What is technologist's and company's responsibility to disclose bugs?

Multiple Cycle Datapath

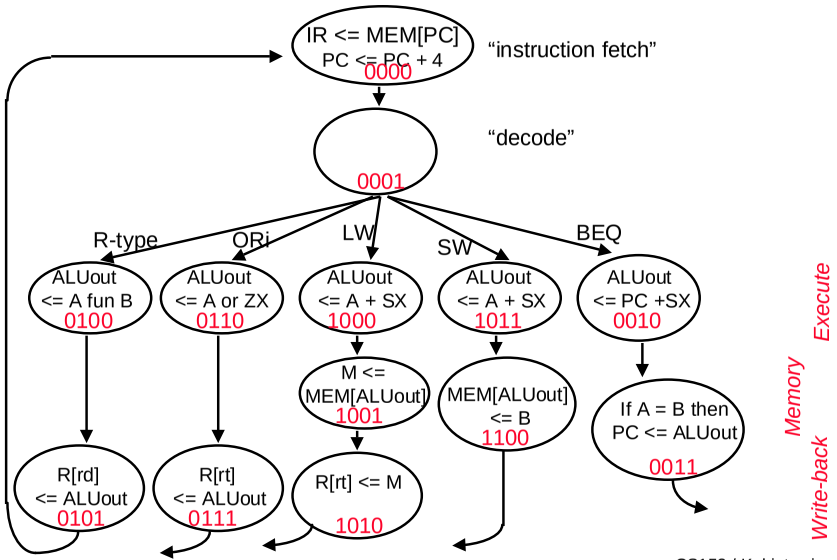


Control: Hardware vs. Microprogrammed

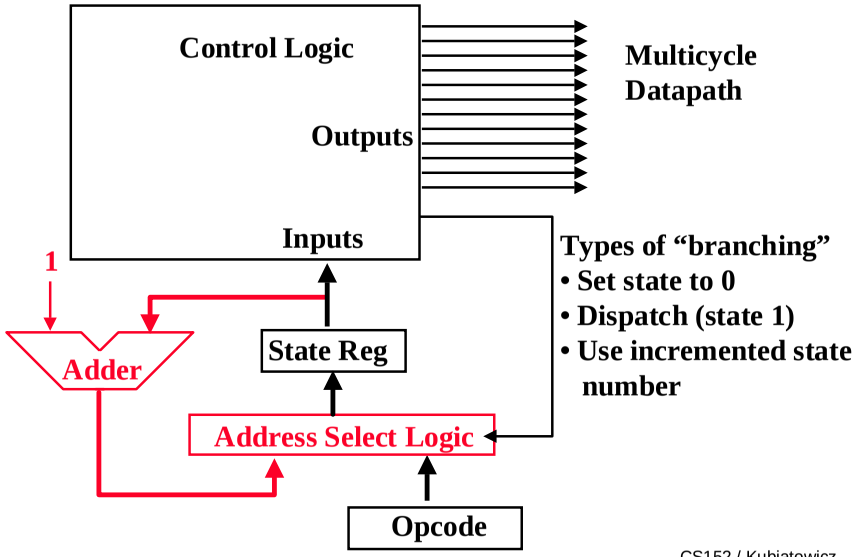
° Control may be designed using one of several initial representations. The choice of sequence control, and how logic is represented, can then be determined independently; the control can then be implemented with one of several methods using a structured logic technique.

Initial Representation	Finite State Diagram	Microprogram
Sequencing Control	Explicit Next State Function	Microprogram counter + Dispatch ROMs
Logic Representation	Logic Equations	Truth Tables
Implementation Technique	PLA	ROM
	"hardwired control"	*"microprogrammed control"*

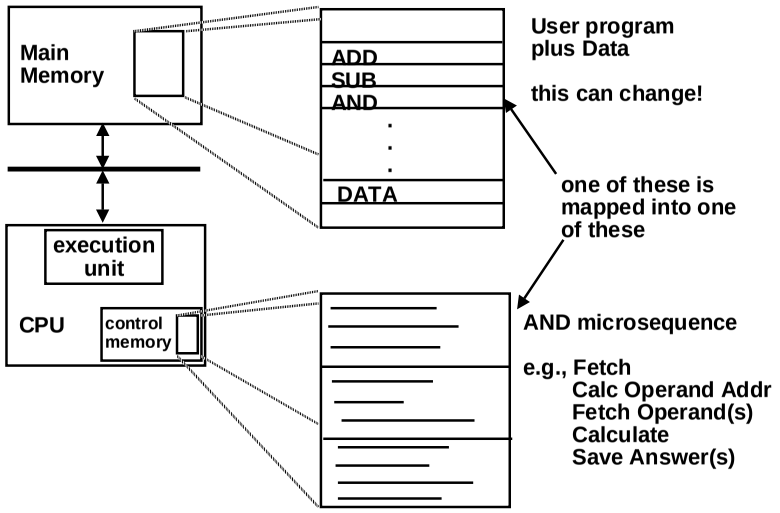
Finite State Machine (FSM) Spec



Sequencer-based control unit



"Macroinstruction" Interpretation



Microprogramming

Label	ALU	SRC1	SRC2	Dest.	Memory	Mem. Reg. PC Write		Sequencing
Fetch:	Add	PC	4		Read PC	IR	ALU	Seq
	Add	PC	Extshft					Dispatch
Rtype:	Func	rs	rt					Seq
				rd ALU				Fetch
Ori:	Or	rs	Extend0					Seq
				rt ALU				Fetch
Lw:	Add	rs	Extend					Seq
					Read ALU			Seq
				rt MEM				Fetch
Sw:	Add	rs	Extend					Seq
					Write ALU			Fetch
Beq:	Subt.	rs	rt			ALUoutCond.		Fetch

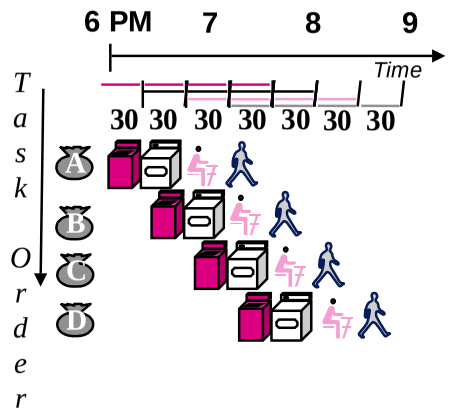
Precise Interrupts

- Precise ⇒ state of the machine is preserved as if program executed up to the offending instruction
  - All previous instructions completed
  - Offending instruction and all following instructions act as if they have not even started
  - Same system code will work on different implementations
  - Position clearly established by IBM
  - Difficult in the presence of pipelining, out-ot-order execution, ...
  - MIPS takes this position
- Imprecise ⇒ system software has to figure out what is where and put it all back together
- Performance goals often lead designers to forsake precise interrupts
  - system software developers, user, markets etc. usually wish they had not done this
- Modern techniques for out-of-order execution and branch prediction help implement precise interrupts

Administrivia

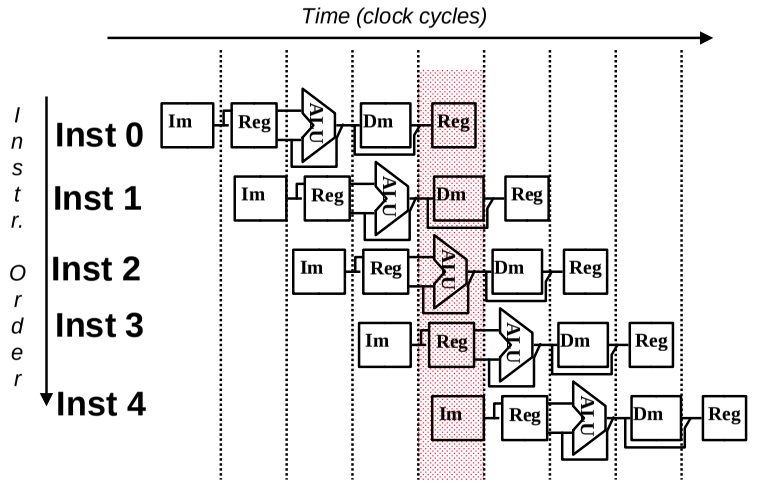
- **Oral reports Tomorrow:**
  - **10am - 12pm, 2pm-4pm, 306 Soda**
  - **5pm go over to lab to run mystery programs**
  - **Still 3 empty slots. You *MUST* sign up today.**
  - **Reports due at 5pm in Lab (not in BOX downstairs)**
  - **TAs have handed out a list of requirements**
  - **Remember: talk is 15 minutes + 5 minutes questions**
    - **Don't bring more than 8 slides!!!**
    - **Practice! Your final project grade will depend partially on your oral report.**
- **Grades posted by Friday**
  - **Give us a random 8 digit number for final grades! We will use this to post final grades on the web site.**

Recap: Pipelining Lessons (its intuitive!)



- **Pipelining doesn't help latency of single task, it helps throughput of entire workload**
- **Multiple tasks operating simultaneously using different resources**
- **Potential speedup = Number pipe stages**
- **Pipeline rate limited by slowest pipeline stage**
- **Unbalanced lengths of pipe stages reduces speedup**
- **Time to "fill" pipeline and time to "drain" it reduces speedup**
- **Stall for Dependences**

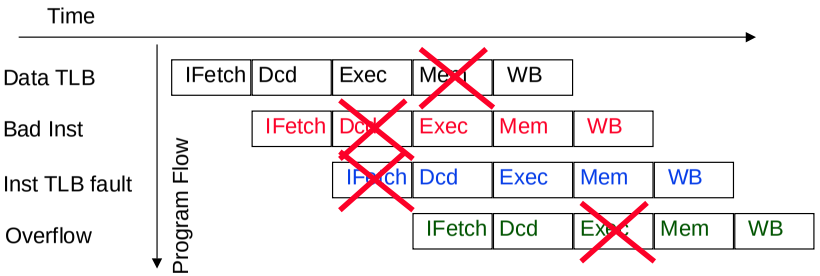
Why Pipeline? Because the resources are there!



Can pipelining get us into trouble?

- Yes: Pipeline Hazards
  - **structural hazards: attempt to use the same resource two different ways at the same time**
    - **E.g., combined washer/dryer would be a structural hazard or folder busy doing something else (watching TV)**
  - **data hazards: attempt to use item before it is ready**
    - **E.g., one sock of pair in dryer and one in washer; can't fold until get sock from washer through dryer**
    - **instruction depends on result of prior instruction still in the pipeline**
  - **control hazards: attempt to make a decision before condition is evaulated**
    - **E.g., washing football uniforms and need to get proper detergent level; need to see after dryer before next load in**
    - **branch instructions**
- Can always resolve hazards by waiting
  - **pipeline control must detect the hazard**
  - **take action (or delay action) to resolve hazards**

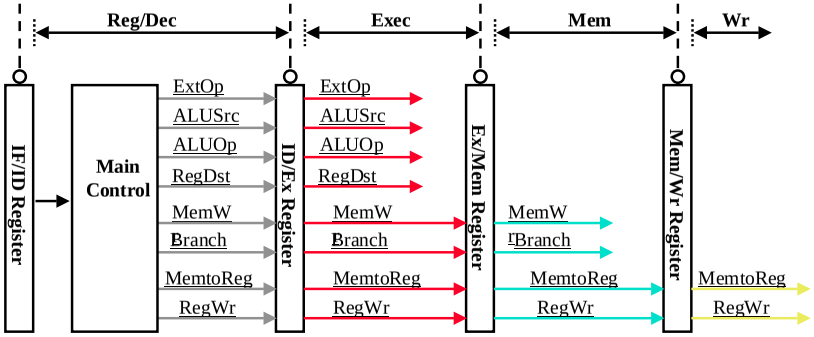
Exceptions in a 5 stage pipeline



- Use pipeline to sort this out!
  - Pass exception status along with instruction.
  - Keep track of PCs for every instruction in pipeline.
  - Don't act on exception until it reache WB stage
- Handle interrupts through "faulting noop" in IF stage
- When instruction reaches WB stage:
  - Save PC ⇒ EPC, Interrupt vector addr ⇒ PC
  - Turn all instructions in earlier stages into noops!

Data Stationary Control

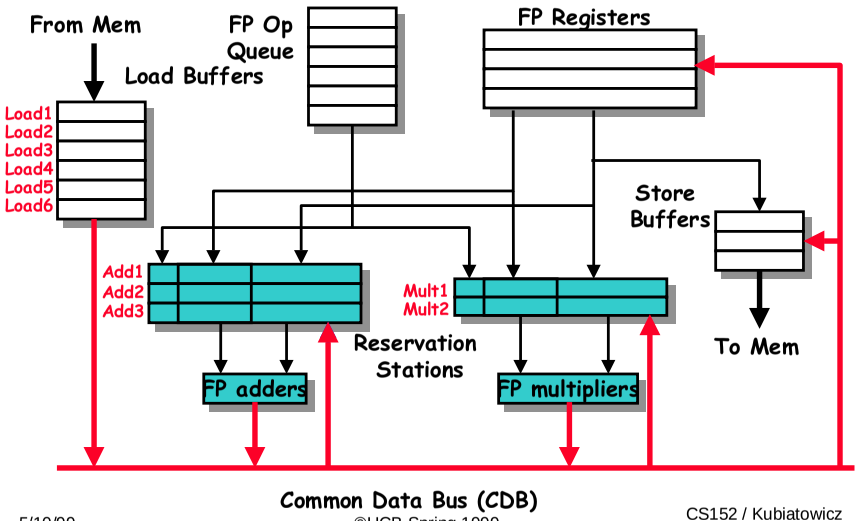
- The Main Control generates the control signals during Reg/Dec
  - Control signals for Exec (ExtOp, ALUSrc, ...) are used 1 cycle later
  - Control signals for Mem (MemWr Branch) are used 2 cycles later
  - Control signals for Wr (MemtoReg MemWr) are used 3 cycles later



Pipeline Summary

- **Simple 5-stage pipeline: F D E M W**
- **Pipelines pass control information down the pipe just as data moves down pipe**
- **Resolve data hazards through forwarding.**
- **Forwarding/Stalls handled by local control**
- **Exceptions stop the pipeline**
- **MIPS I instruction set architecture made pipeline visible (delayed branch, delayed load)**
- **More performance from deeper pipelines, parallelism**
- **You built a complete 5-stage pipeline in the lab!**

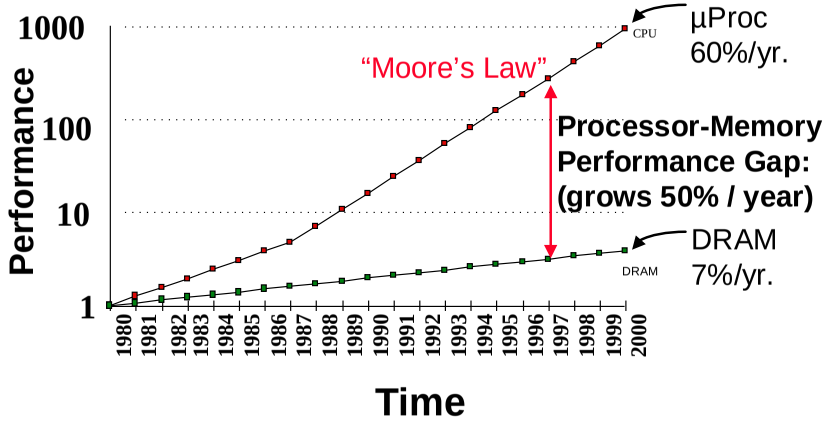
Out of order execution: Tomasulo Organization



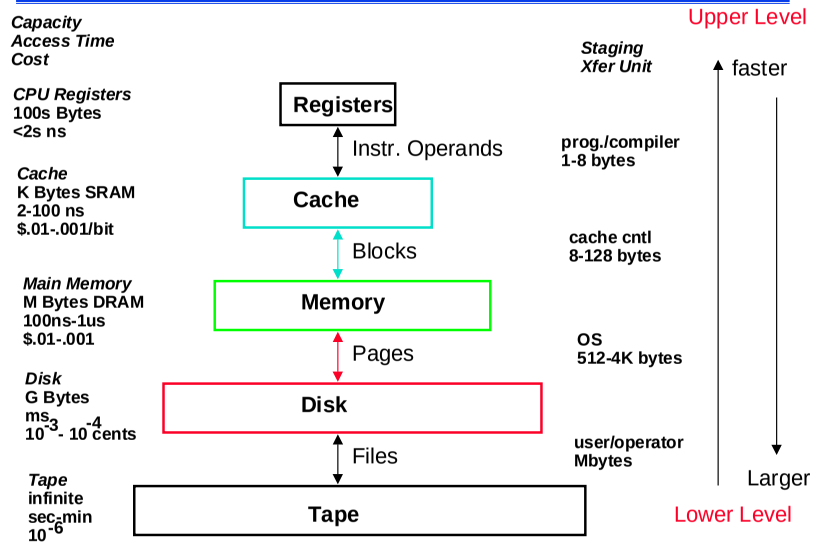
How can the machine exploit available ILP?

Technique	Limitation
° Pipelining	Issue rate, FU stalls, FU depth
° Super-pipeline - Issue 1 instr. / (fast) cycle - IF takes multiple cycles	Clock skew, FU stalls, FU depth
° Super-scalar - Issue multiple scalar instructions per cycle	Hazard resolution
° VLIW - Each instruction specifies multiple scalar operations	Packing, Compiler

Processor-DRAM Gap (latency)



Levels of the Memory Hierarchy



Memory Hierarchy

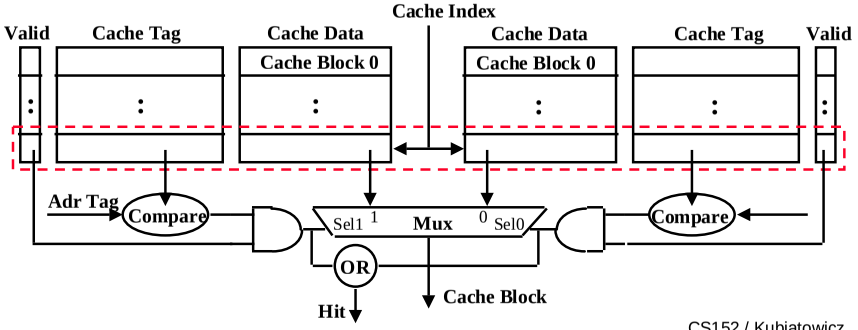
° The Principle of Locality:
- Program access a relatively small portion of the address space at any instant of time.
  - Temporal Locality: Locality in Time
  - Spatial Locality: Locality in Space

° Three Major Categories of Cache Misses:
- Compulsory Misses: sad facts of life. Example: cold start misses.
- Conflict Misses: increase cache size and/or associativity.
- Capacity Misses: increase cache size

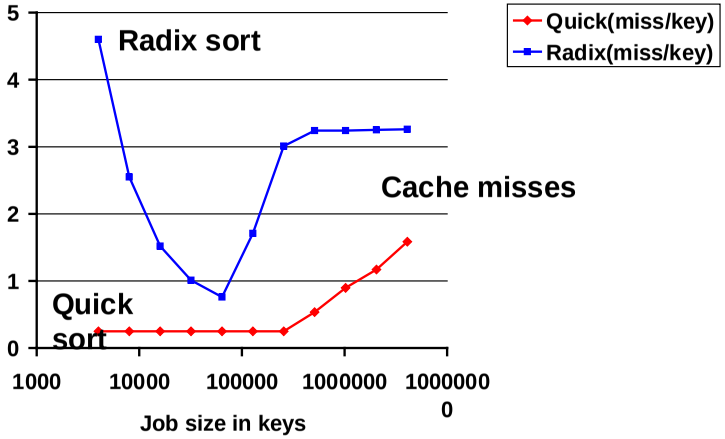
° Virtual Memory invented as another level of the hierarchy
- Today VM allows many processes to share single memory without having to swap all processes to disk, protection more important
- TLBs are important for fast translation/checking

Set Associative Cache

- N-way set associative: N entries for each Cache Index
  - N direct mapped caches operates in parallel
- Example: Two-way set associative cache
  - Cache Index selects a "set" from the cache
  - The two tags in the set are compared to the input in parallel
  - Data is selected based on the tag result

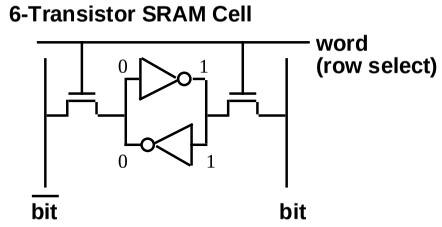


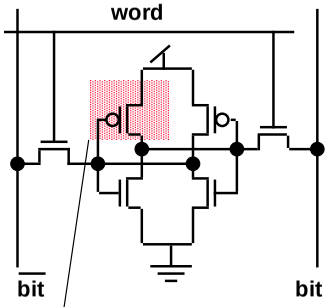
Quicksort vs. Radix as vary number keys: Cache misses



What is proper approach to fast algorithms?

Static RAM Cell

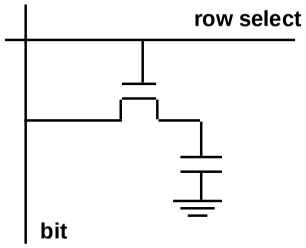




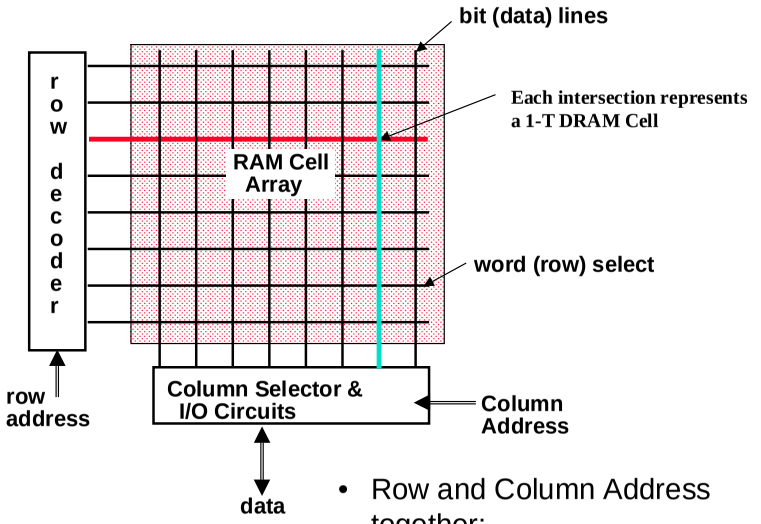
- Write:
  1. Drive bit lines (bit=1, bit=0)
  2.. Select row
- Read:
  1. Precharge bit and bit to Vdd or Vdd/2 => make sure equal!
  2.. Select row
  3. Cell pulls one line low
  4. Sense amp on column detects difference between bit and bit

1-Transistor Memory Cell (DRAM)

- Write:
  - 1. Drive bit line
  - 2.. Select row
- Read:
  - 1. Precharge bit line to Vdd
  - 2.. Select row
  - 3. Cell and bit line share charges
    - Very small voltage changes on the bit line
  - 4. Sense (fancy sense amp)
    - Can detect changes of ~1 million electrons
  - 5. Write: restore the value
- Refresh
  - 1. Just do a dummy read to every cell.



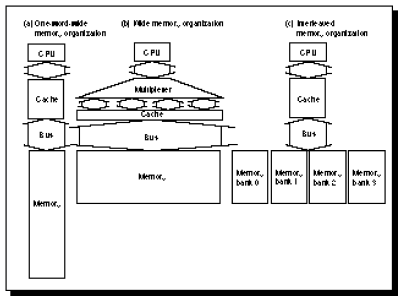
Classical DRAM Organization (square)



- Row and Column Address together:
  - Select 1 bit a time

Main Memory Performance

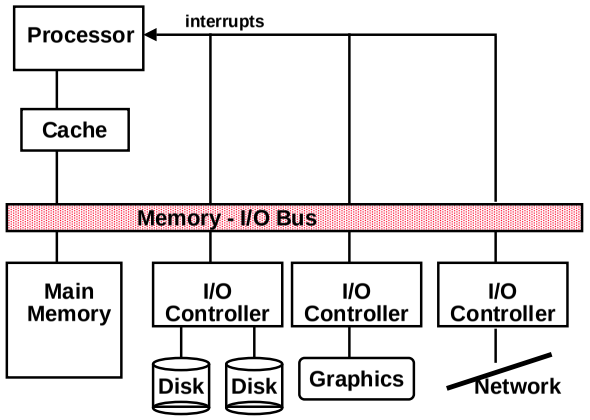
- *Simple*: CPU, Cache, Bus, Memory same width (32 bits)
- *Wide*: CPU/Mux 1 word; Mux/Cache, Bus, Memory N words (Alpha: 64 bits & 256 bits)
- *Interleaved*: CPU, Cache, Bus 1 word: Memory N Modules (4 Modules); example is *word interleaved*



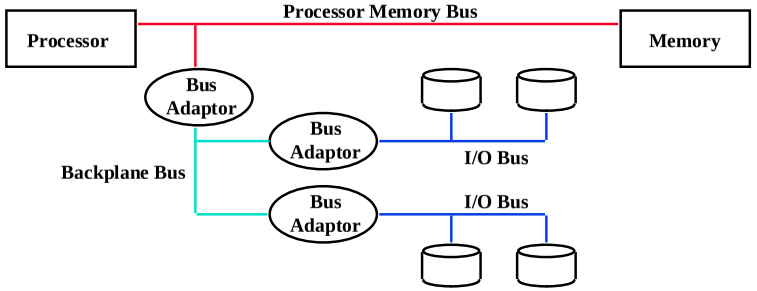
Timing model: 1 to send address, 6 access time, 1 to send data
Cache Block is 4 words
Simple M.P. = 4 x (1+6+1) = 32
Wide M.P. = 1 + 6 + 1 = 8
Interleaved M.P. = 1 + 6 + 4x1 = 11

I/O System Design Issues

- Systems have a hierarchy of busses as well (PC: memory,PCI,ESA)

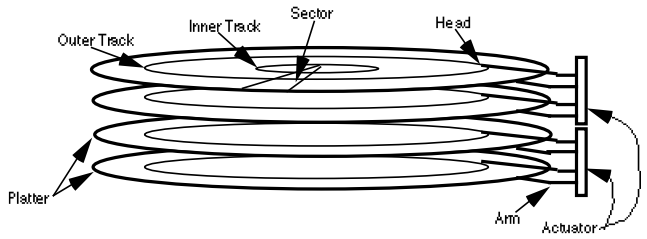


A Three-Bus System



- A small number of backplane buses tap into the processor-memory bus
  - Processor-memory bus is only used for processor-memory traffic
  - I/O buses are connected to the backplane bus
- Advantage: loading on the processor bus is greatly reduced

Disk Device Terminology



Disk Latency = Queueing Time + Controller time + Seek Time + Rotation Time + Xfer Time

Order of magnitude times for 4K byte transfers:

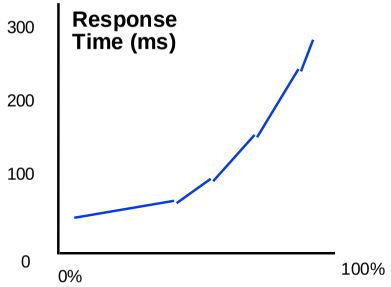
Average Seek: 8 ms or less

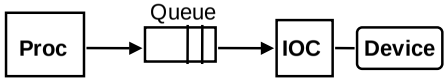
Rotate: 4.2 ms @ 7200 rpm

Xfer: 1 ms @ 7200 rpm

Disk I/O Performance

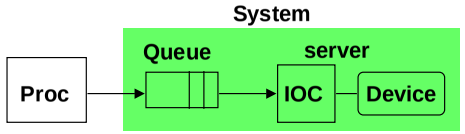
Metrics:
Response Time
Throughput





Response time = Queue + Device Service time

A Little Queuing Theory: M/M/1 queues



- Described "memoryless" or Markovian request arrival (M for C=1 exponentially random), 1 server: *M/M/1 queue*
- When Service times have C = 1, *M/M/1 queue*

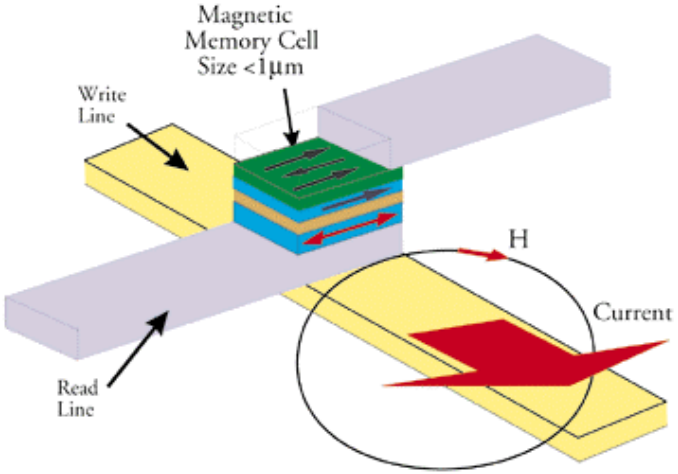
$$T_q = T_{ser} \times u / (1 - u)$$

T_{ser} average time to service a customer

u server utilization (0..1): $u = \lambda \times T_{ser}$

T_q average time/customer in queue

Computers in the news: Tunneling Magnetic Junction



Computers in the News: Sony Playstation 2000

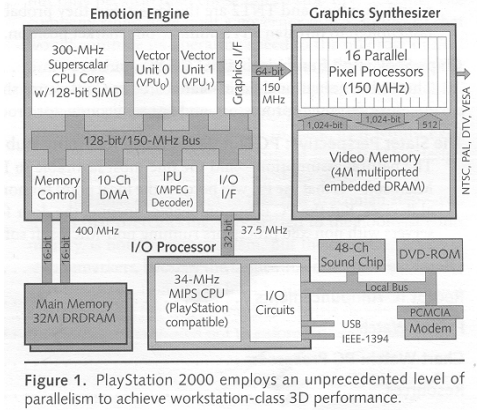


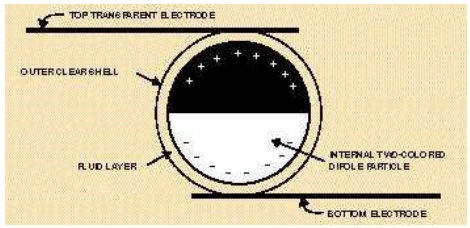
Figure 1. PlayStation 2000 employs an unprecedented level of parallelism to achieve workstation-class 3D performance.

Figure 2. PlayStation 2000 screenshot. (Source: Namco)

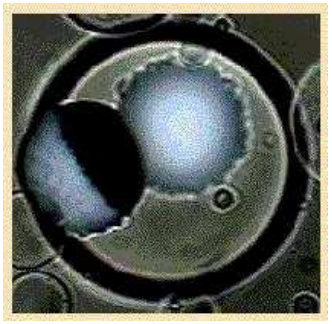
- **(as reported in Microprocessor Report, Vol 13, No. 5)**
  - **Emotion Engine: 6.2 GFLOPS, 75 million polygons per second**
  - **Graphics Synthesizer: 2.4 Billion pixels per second**
  - **Claim: *Toy Story* realism brought to games!**

Computers in the News: Electronic Ink

- Electronic Ink:
  - Little capsules with charged balls that are half black/half white
  - Placing an electronic charge of one polarity makes dot black and the other polarity makes it white.
  - Flexible, cheap, paper-like displays!



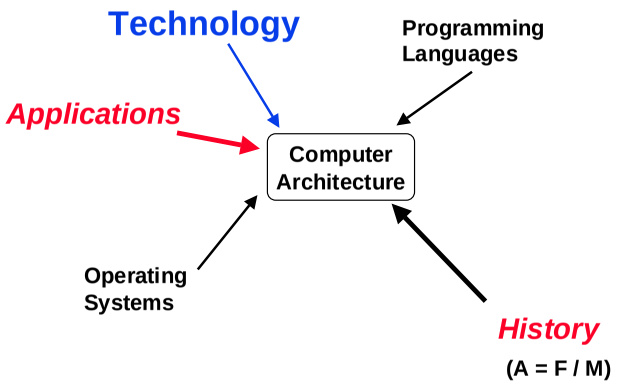
Schematic Diagram



Electron Micrograph

What does the future hold?

Forces on Computer Architecture



Key Technologies

° Fast, cheap, highly integrated “computers-on-a-chip”
  • IDT R4640, NEC VR4300, StrongARM, Superchips

° Affordable access to fast networks -> Network is everywhere!
  • ISDN, Cable Modems, ATM, . . .

° Platform independent programming languages
  • Java, JavaScript, Visual Basic Script

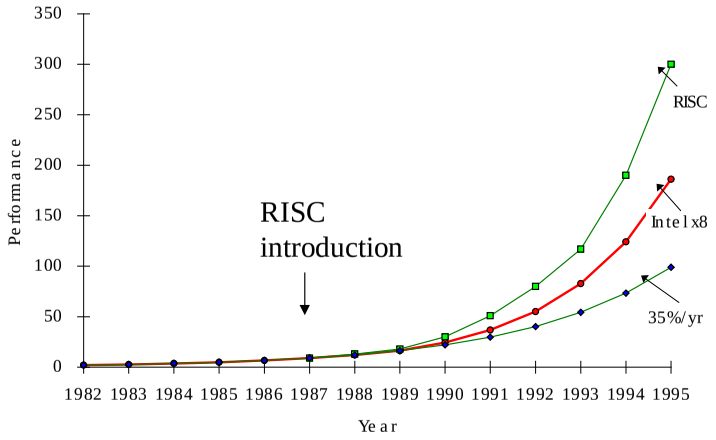
° Lightweight Operating Systems
  • GEOS, NCOS, RISCOS

° ???

Future of Computer Architecture and Engineering

- Performance
- High Level Computer Architecture
- Multiprocessors
- “IRAM”
- “Introspective Computing”
- AetherStore

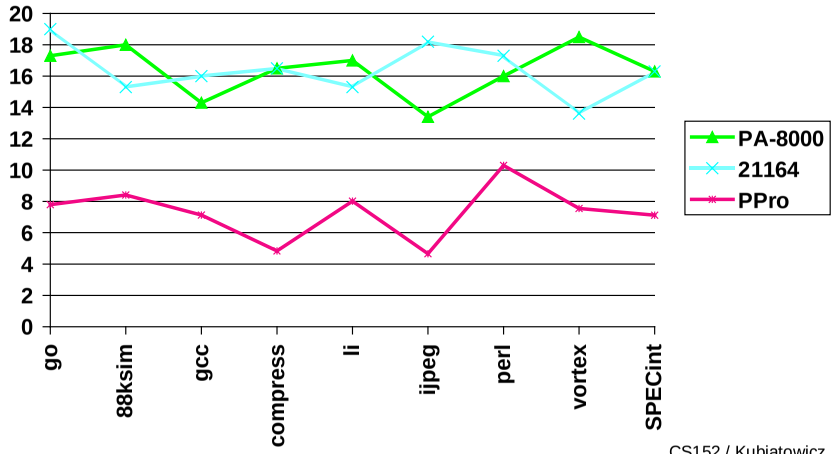
Processor Performance



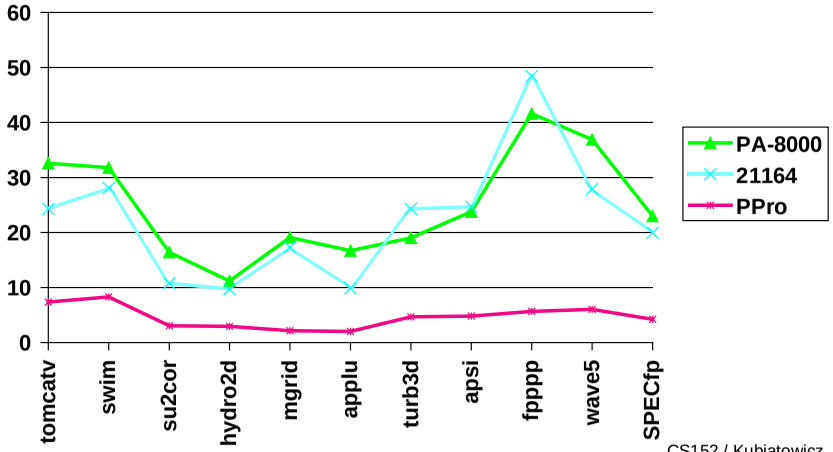
3 Recent Machines

	"Speed Demon"		"Braniac"
	Alpha 21164	Pentium II	HP PA-8000
Year	1995	1996	1996
Clock	600 MHz ('97)	300 MHz ('97)	236 MHz ('97)
Cache	8K/8K/96K/2M	16K/16K/0.5M	0/0/4M
Issue rate	2int+2FP	3 instr (x86)	4 instr
Pipe stages	7-9	12-14	7-9
Out-of-Order	6 loads	40 instr (µop)	56 instr
Rename regs	none	40	56

SPECint95base Performance (Oct. 1997)



SPECint95base Performance (Oct. 1997)

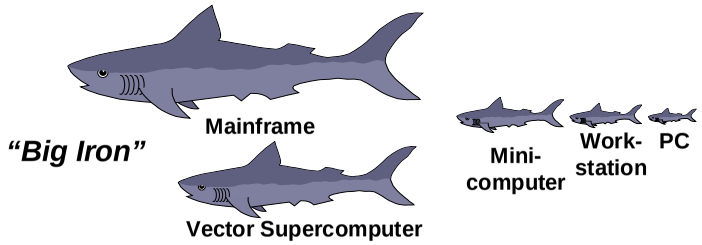


Performance Retrospective

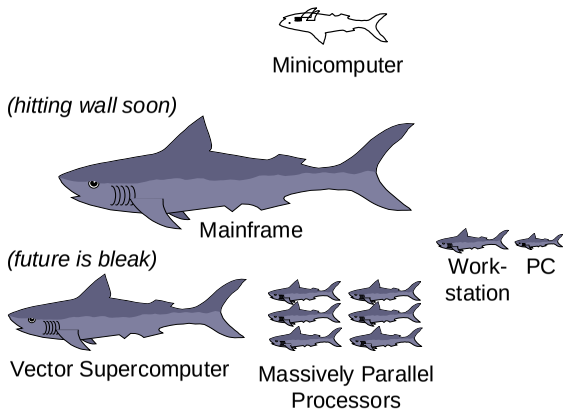
- Theory of Algorithms & Compilers based on number of operations
- Compiler remove operations and "simplify" ops:
  Integer adds << Integer multiplies << FP adds << FP multiplies
  - Advanced pipelines => these operations take similar time (FP multiply faster than integer multiply)
- As Clock rates get higher and pipelines are longer, instructions take less time but DRAMs only slightly faster (although much larger)
- Today time is a function of (ops, cache misses);

- How do you tune performance on Pentium Pro? Random?
- Given importance of caches, what does this mean to:
  - Compilers?
  - Data structures?
  - Algorithms?

1985 Computer Food Chain



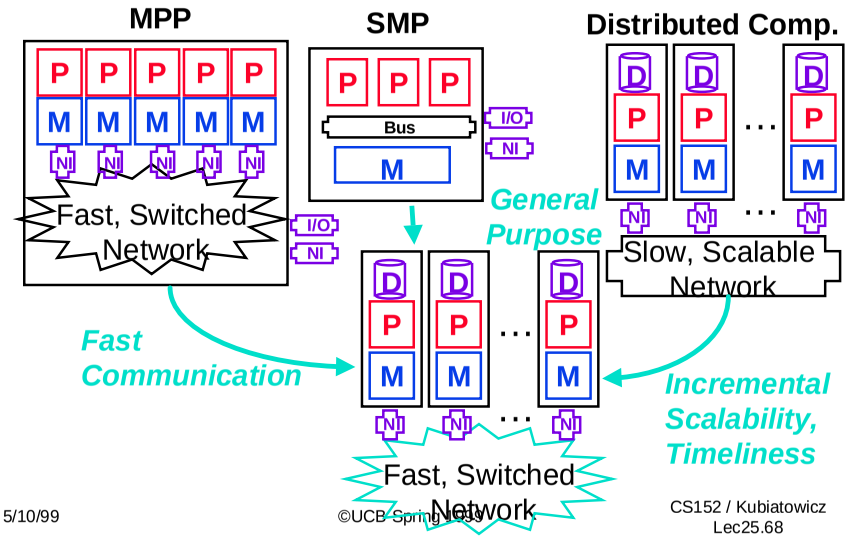
1995 Computer Food Chain

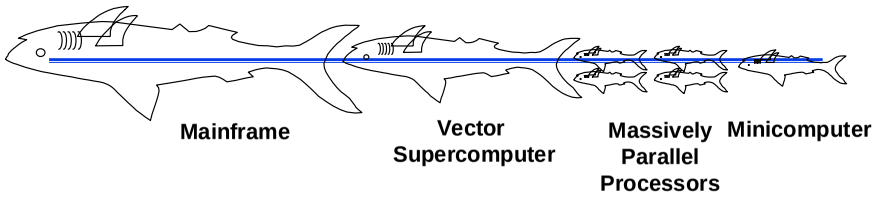


Interconnection Networks

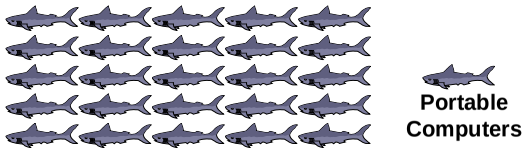
° **Switched vs. Shared Media: pairs communicate at same time:**
"point-to-point" connections

Cluster/Network of Workstations (NOW)





2005 Computer Food Chain?



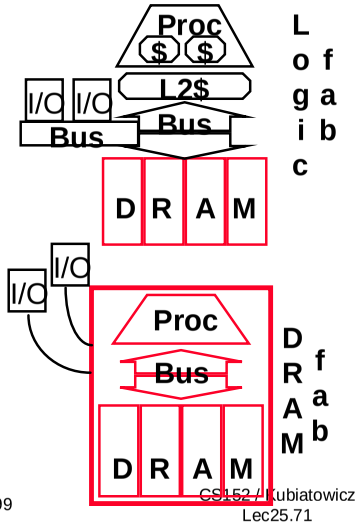
Intelligent DRAM (IRAM)

- IRAM motivation (-2000 to 2005)
  - 256 Mbit/1Gbit DRAMs in near future (128 MByte)
  - Current CPUs starved for memory BW
  - On chip memory BW = SQRT(Size)/RAS or 80 GB/sec
  - 1% of Gbit DRAM = 10M transistors for µprocessor
  - Even in DRAM process, a 10M trans. CPU is attractive
  - Package could be network interface vs. Addr./Data pins
  - Embedded computers are increasingly important
- Why not re-examine computer design based on separation of memory and processor?
  - Compact code & data?
  - Vector instructions?
  - Operating systems? Compilers? Data Structures?

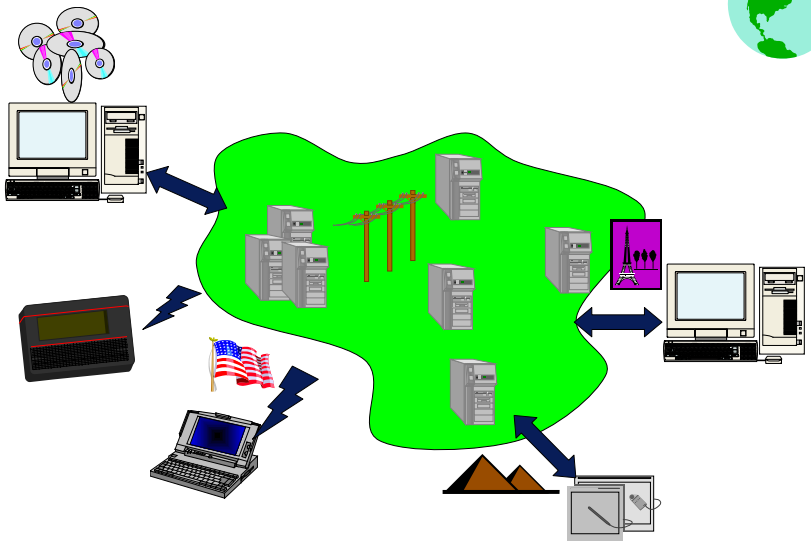
IRAM Vision Statement

Microprocessor & DRAM on a single chip:

- on-chip memory latency 5-10X, bandwidth 50-100X
- improve energy efficiency 2X-4X (no off-chip bus)
- serial I/O 5-10X v. buses
- smaller board area/volume
- adjustable memory size/width

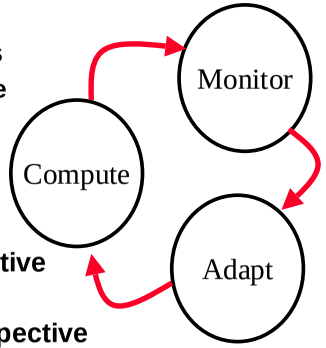


The ÆtherStore View: Oceans of Data



Introspective Computing

- Biological Analogs for computer systems:
  - Continuous adaptation
  - Insensitivity to design flaws
    - Both hardware and software
    - Necessary if can never be sure that all components are working properly…
- Examples:
  - ISTORE -- applies introspective computing to disk storage
  - DynaComp -- applies introspective computing at chip level
    - Compiler always running and part of execution!



and why not

- multiprocessors on a chip?
- complete systems on a chip?
  - memory + processor + I/O
- computers in your credit card?
- networking in your kitchen? car?
- eye tracking input devices?

Learned from Cal/CS152?

CS152: So what's in it for me? (from 1st lecture)

- In-depth understanding of the inner-workings of modern computers, their evolution, and trade-offs present at the hardware/software boundary.
  - Insight into fast/slow operations that are easy/hard to implementation hardware
- Experience with the *design process* in the context of a large complex (hardware) design.
  - Functional Spec --> Control & Datapath --> Physical implementation
  - Modern CAD tools
- Designer's "Intellectual" toolbox.

Simulate Industrial Environment (from 1st lecture)

° Project teams must have at least 4 members
  - Managers have value

° Communicate with colleagues (team members)
  - What have you done?
  - What answers you need from others?
  - You must document your work!!!
  - Everyone must keep an on-line notebook

° Communicate with supervisor (TAs)
  - How is the team's plan?
  - Short progress reports are required:
    - What is the team's game plan?
    - What is each member's responsibility?

So let's thanks those TAs

Summary: Things we Hope You Learned from 152

° Keep it simple and make it work:
  - Fully test everything individually & then together; break when together
  - Retest everything whenever you make any changes
  - Last minute changes are big "no nos"

° Group dynamics. Communication is the key to success:
  - Be open with others of your expectations & your problems (e.g., trip)
  - Everybody should be there on design meetings when key decisions are made and jobs are assigned

° Planning is very important ("plan your life; live your plan"):
  - Promise what you can deliver; deliver more than you promise
  - Murphy's Law: things ***DO*** break at the last minute
    - ***DON'T*** make your plan based on the best case scenarios
    - Freeze your design and don't make last minute changes

° Never give up! It is not over until you give up ("Bear won't die")