# CS 152
# Computer Architecture and Engineering
# Lecture 26

## The Final Chapter
## A whirlwind retrospective on the term

$\aleph_0$

**May 10, 2004**

**John Kubiatowicz (www.cs.berkeley.edu/~kubitron)**

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

## What *IS* Quantum Computing?



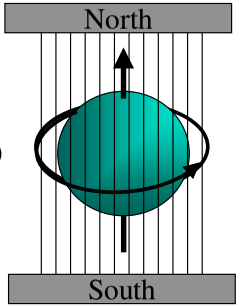
## Can we Use Quantum Mechanics to Compute?

- Weird properties of quantum mechanics?
  - You've already seen one: tunneling of electrons through insulators to make TMJ RAM
  - Quantization: Only certain values or orbits are good
    - Remember orbitals from chemistry???
  - Superposition: Schizophrenic physical elements don't quite know whether they are one thing or another
- All existing digital abstractions try to eliminate QM
  - Transistors/Gates designed with classical behavior
  - Binary abstraction: a "1" is a "1" and a "0" is a "0"
- Quantum Computing:
  Use of Quantization and Superposition to compute.
- Interesting results:
  - Shor's algorithm: factors in polynomial time!
  - Grover's algorithm: Finds items in unsorted database in time proportional to square-root of n.



## Quantization: Use of "Spin"

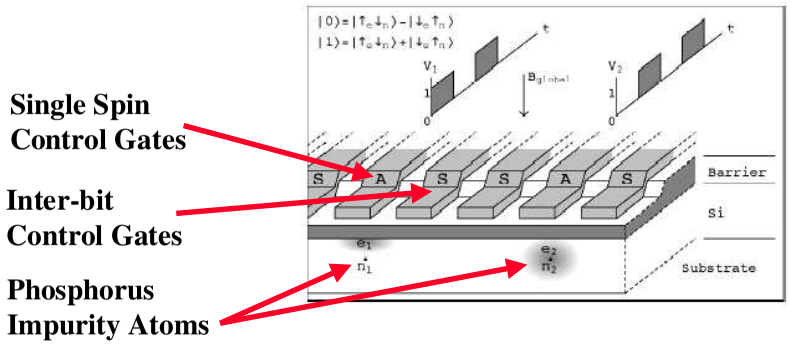
**Spin ½ particle:**
**(Proton/Electron)**




**Representation:**
**|0> or |1>**

- Particles like Protons have an intrinsic "Spin" when defined with respect to an external magnetic field
- Quantum effect gives "1" and "0":
  - Either spin is "UP" or "DOWN" nothing between

## Kane Proposal II (First one didn't quite work)




- Bits Represented by combination of proton/electron spin
- Operations performed by manipulating control gates
  - Complex sequences of pulses perform NMR-like operations
- Temperature < 1° Kelvin!



## Now add Superposition!

- The bit can be in a combination of "1" and "0":
  - Written as: $\Psi = C_0|0> + C_1|1>$
  - The C's are *complex numbers!*
  - Important Constraint: $|C_0|^2 + |C_1|^2 = 1$
- If *measure* bit to see what looks like,
  - With probability $|C_0|^2$ we will find |0> (say "UP")
  - With probability $|C_1|^2$ we will find |1> (say "DOWN")
- Is this a real effect? Options:
  - This is just statistical – given a large number of protons, a fraction of them ($|C_0|^2$ ) are "UP" and the rest are down.
  - This is a real effect, and the proton is really both things until you try to look at it
- Reality: second choice! There are experiments to prove it!



## Implications: A register can have many values

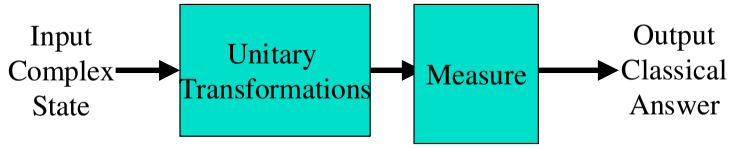
- Implications of superposition:
  - An *n*-bit register can have $2^n$ values simultaneously!
  - 3-bit example:

$$\Psi = C_{000}|000> + C_{001}|001> + C_{010}|010> + C_{011}|011> + C_{100}|100> + C_{101}|101> + C_{110}|110> + C_{111}|111>$$

- Probabilities of measuring all bits are set by coefficients:
  - So, prob of getting |000> is $|C_{000}|^2$, etc.
  - Suppose we measure only one bit (first):
    - We get a "0" with probability: $P_0 = |C_{000}|^2 + |C_{001}|^2 + |C_{010}|^2 + |C_{011}|^2$
      Result: $\Psi = \frac{1}{\sqrt{P_0}}(C_{000}|000> + C_{001}|001> + C_{010}|010> + C_{011}|011>)$
    - We get a "1" with probability: $P_1 = |C_{100}|^2 + |C_{101}|^2 + |C_{110}|^2 + |C_{111}|^2$
      Result: $\Psi = \frac{1}{\sqrt{P_1}}(C_{100}|100> + C_{101}|101> + C_{110}|110> + C_{111}|111>)$
- Problem: Don't want environment to *measure* before ready!
  - Solution: Quantum Error Correction Codes!



## Model? Operations on coefficients + measurements



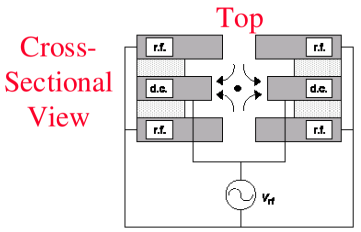

- Basic Computing Paradigm:
  - Input is a register with superposition of many values
    - Possibly all $2^n$ inputs equally probable!
  - Unitary transformations compute on coefficients
    - Must maintain probability property (sum of squares = 1)
    - Looks like doing computation on all $2^n$ inputs simultaneously!
  - Output is one result attained by measurement
- If do this poorly, just like probabilistic computation:
  - If $2^n$ inputs equally probable, may be $2^n$ outputs equally probable.
  - After measure, like picked random input to classical function!
  - All interesting results have some form of "fourier transform" computation being done in unitary transformation

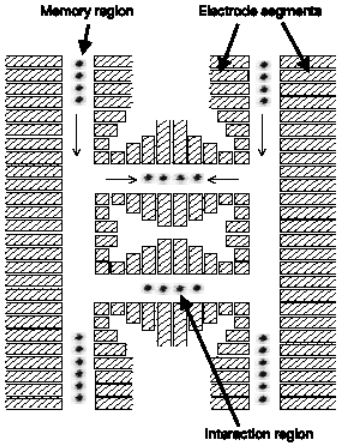
## Some Issues in building quantum computer

- What are the bits and how do we manipulate them?
  - NMR computation: use "cup of liquid".
    - Use nuclear spins (special protons on complex molecules).
    - Manipulate with radio-frequencies
    - IBM Has produced a 7-bit computer
  - Silicon options (more scalable)
    - Impurity Phosphorus in silicon
    - Manipulate through electrons (including measurement)
    - Still serious noise/fabrication issues
  - Other options:
    - Optical (Phases of photons represent bits)
    - Single ions trapped in magnetic fields
- How do we prevent the environment from "Measuring"?
  - Make spins as insulated from environment as possible
  - Quantum Error Correction!
- Where do we get "clean" bits (I.e. unsuperposed |0> or |1>)?
  - Entropy exchange unit:
    - Radiates heat to environment (entropy)
    - Produces clean bits (COLD) to enter into device



## ION Trap Quantum Computer: Promising technology



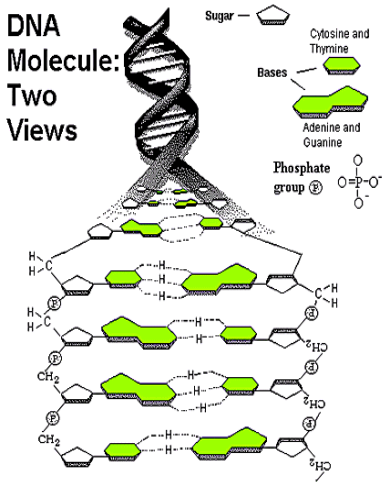

Top View
*Proposal: NIST Group*

- IONS of Be+ trapped in oscillating quadrature field
  - Internal electronic modes of IONS used for quantum bits
  - MEMs technology
  - Target? 50,000 ions
  - ROOM Temperature!
- Ions moved to interaction regions
  - Ions interactions with one another moderated by lasers



## DNA Computing

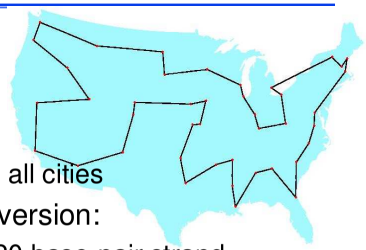
- Can we use DNA to do massive computations?
  - Organisms do it
  - DNA has an *extremely high information density*:
    - 4 different base pairs:
      - Adenine/Thymine
      - Guanine/Cytosine
    - Always paired up on opposite strands=>Energetically favorable
  - Active operations:
    - Copy: Split strands of 1 DNA apart in proper solution, gain 2 copies
    - Concatenate: eg: GTAATCCT will cause XXXXXCATT to combine with AGGAYYYYY
    - Polymerase Chain Reaction (PCR): greatly amplifies region of molecule between two marker molecules




©1999 Access Excellence @ the National Health Museum



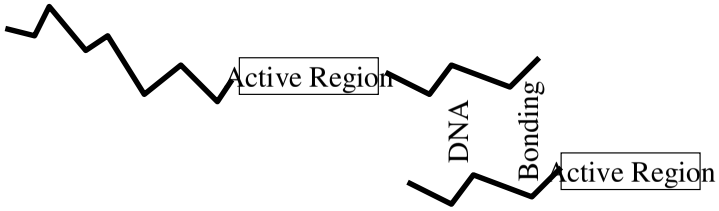
## DNA Computing and the Hamiltonian Path



- Given a set of cities and costs between them (possibly directed paths):
  - Find shortest path
  - Simpler: find single path that visits all cities
- DNA Computing example is latter version:
  - Every city represented by unique 20 base-pair strand
  - Every path between cities represented by complementary pairs: 10 pairs from source city, 10 pairs from destination
  - Shorter example: `AAGT` for city 1, `TTCG` for city 2
    Path 1->2: `CAAA`
    Will build:
    ```
    AAGTTTCG
     ||||
    ..CAAA..
    ```
  - Dump "city molecules" and "path molecules" into testtube. Select and amplify paths of right length. Analyze for result.
  - Been done for 6 cities! (Adleman, ~1998!)

## Even more promising uses of DNA

- Self-assembly of components
  - DNA serves as substrate
  - Attach active elements in middle of components.
  - Final step – metal deposited over DNA




- Other interesting structures could be built

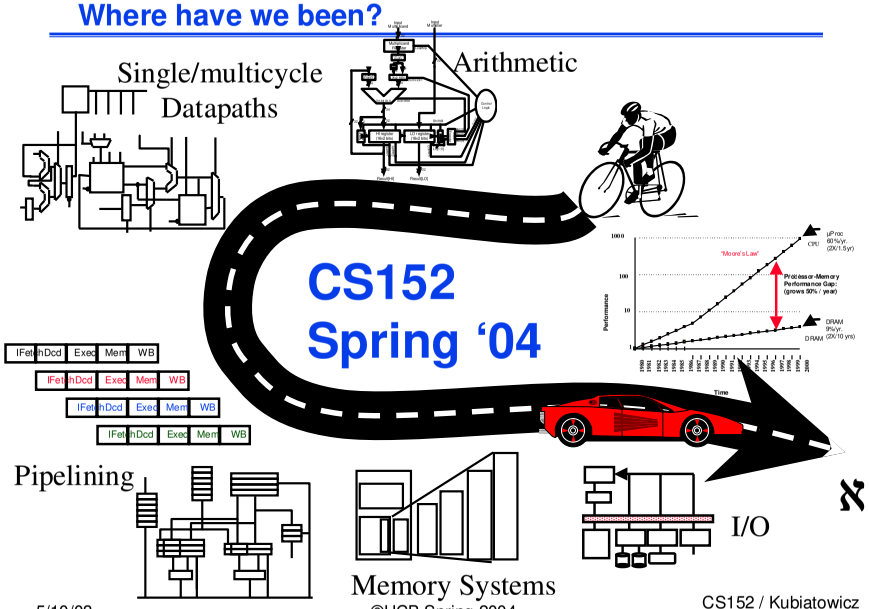


## Administrivia

- **Oral reports Thursday:**
  - **1:00 – 3:00 in 306 Soda**
  - **Remember: talk is 20 minutes + 5 minutes questions**
    - **Don't bring more than 10 slides!!!**
    - **Practice! Your final project grade will depend partially on your oral report.**
    - **Everyone should participate in talk**
  - **Sheet on my door**
  - **Everyone should show up for these 2 hours (short!)**
  - **Plan on cheering on your classmates (ask good questions)**
- **Finally: Final contest when?**
  - **Friday?**
  - **Monday? (My original plan)**



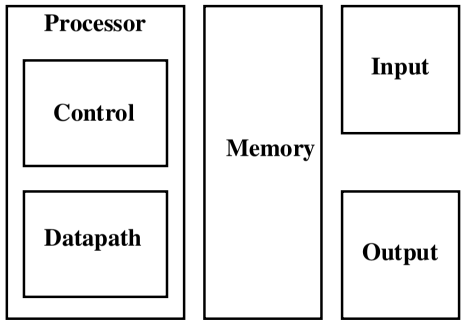
## Where have we been?



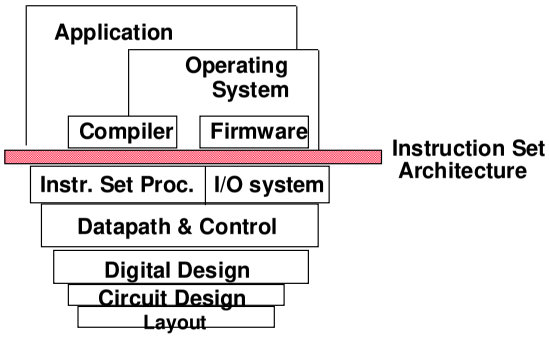



## The Big Picture

° **Since 1946 all computers have had 5 components**



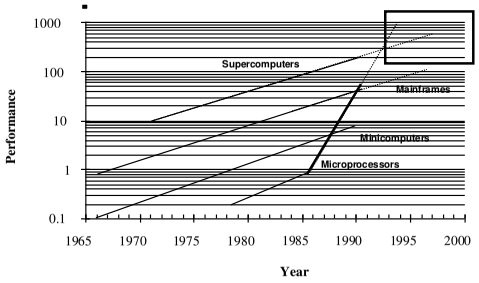
## What is "Computer Architecture"?




- 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



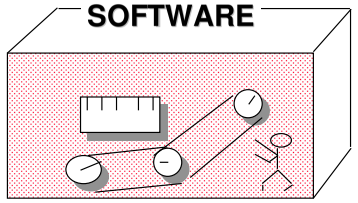
## Examples of "Moore's Law's"

- Processor
  - logic capacity: about 30% per year
  - clock rate: about 20% per year
  - Performance: about 50-60% per year (2x in 18 months)
- Memory
  - DRAM capacity: about 60% per year (4x every 3 years)
  - Memory speed: about 10% per year
  - Cost per bit: improves about 25% per year
- Disk
  - capacity: about 60% per year



## 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**

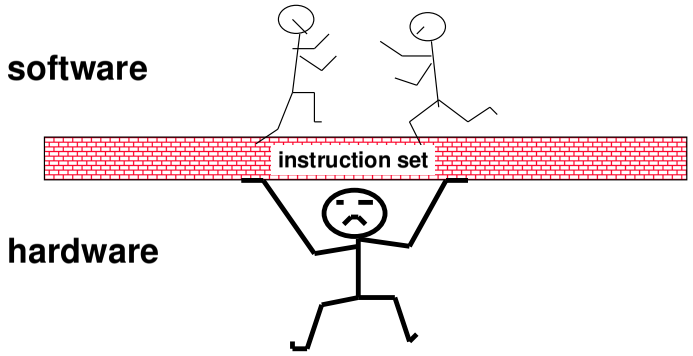
**-- Instruction Set**

**-- Instruction Formats**

**-- Modes of Addressing and Accessing Data Items and Instructions**

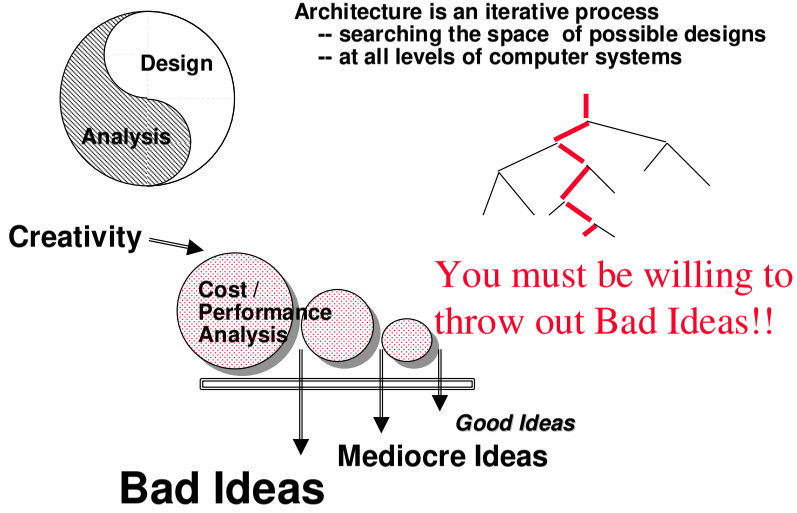
**-- Exceptional Conditions**

## Instruction Set Design





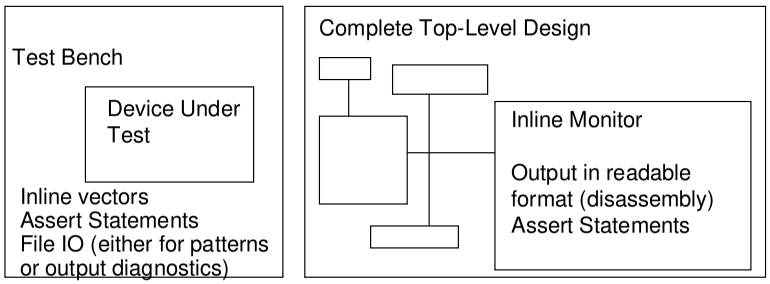

## Measurement and Evaluation






## One of most important aspects of design: TEST

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





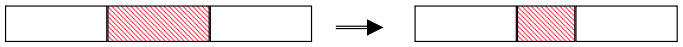

## Basis of Evaluation

| Pros | | Cons |
|---|---|---|
| • representative | Actual Target Workload | • very specific<br>• non-portable<br>• difficult to run, or measure<br>• hard to identify cause |
| • portable<br>• widely used<br>• 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,

ExTime(with E) = ((1-F) + F/S) X 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)

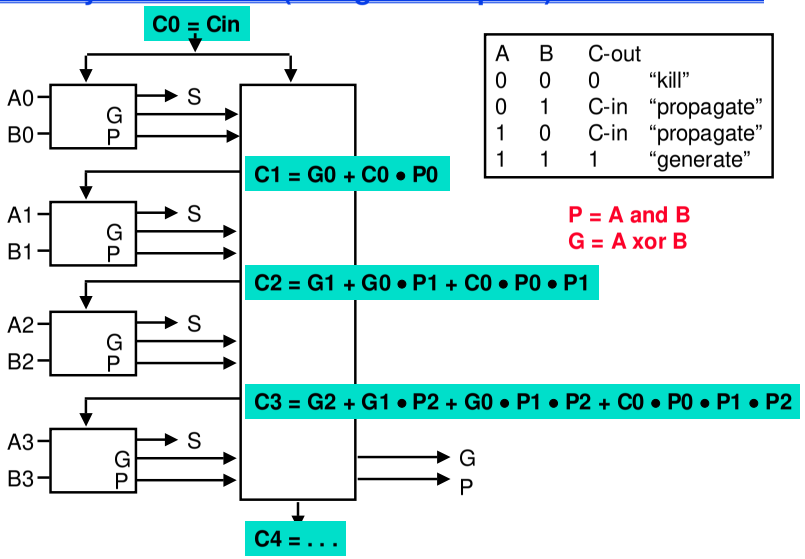


## Computer Arithmetic

- 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

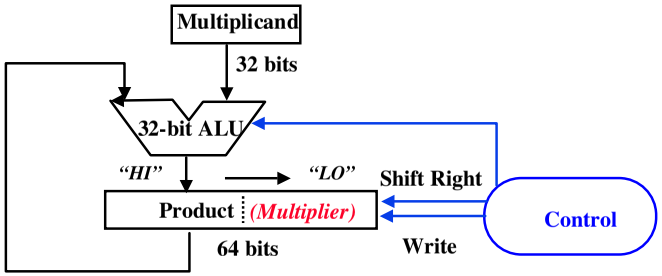


## Carry Look Ahead (Design trick: peek)



## 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 |

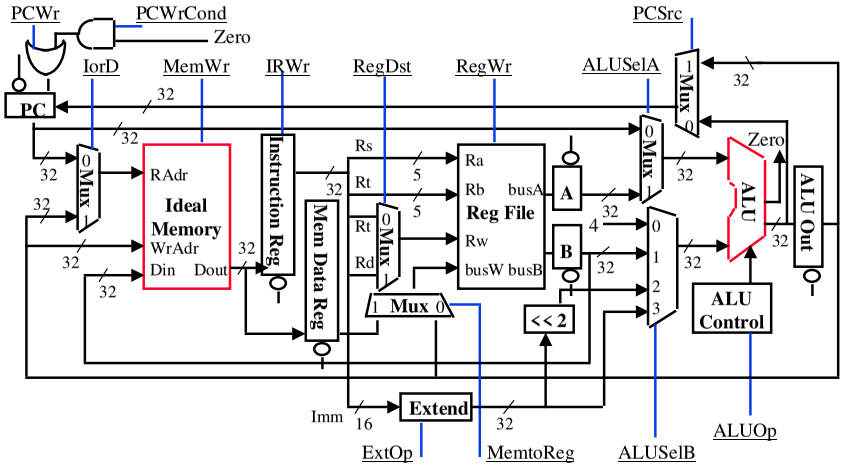
Originally for Speed (when shift was faster than add)

- 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
```

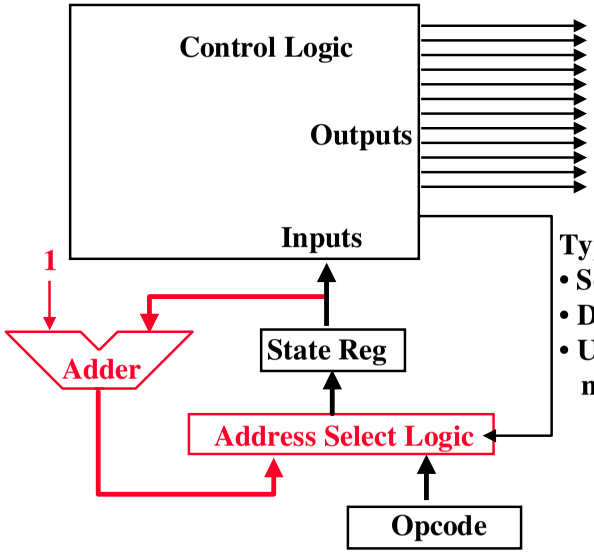


## Multiple Cycle Datapath






## Sequencer-based control unit




Types of "branching"
- Set state to 0
- Dispatch (state 1)
- Use incremented state number

## 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 |

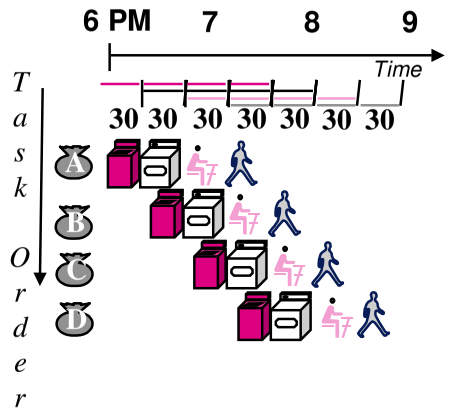


## Precise Interrupts

- 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



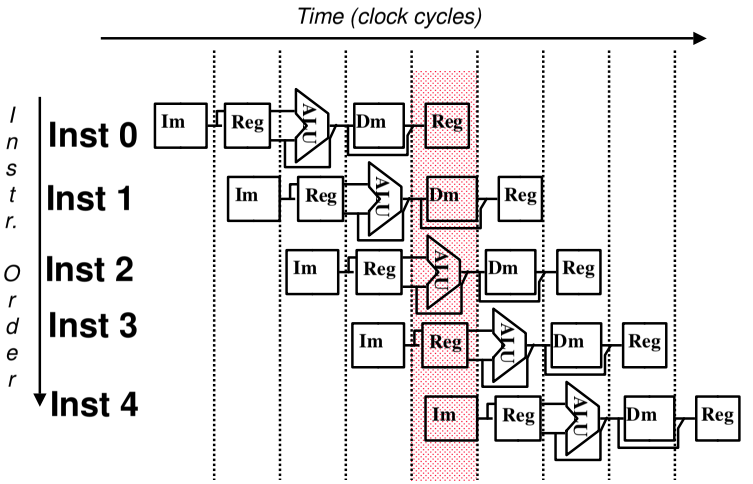
## Recap: Pipelining Lessons (its intuitive!)




- 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!

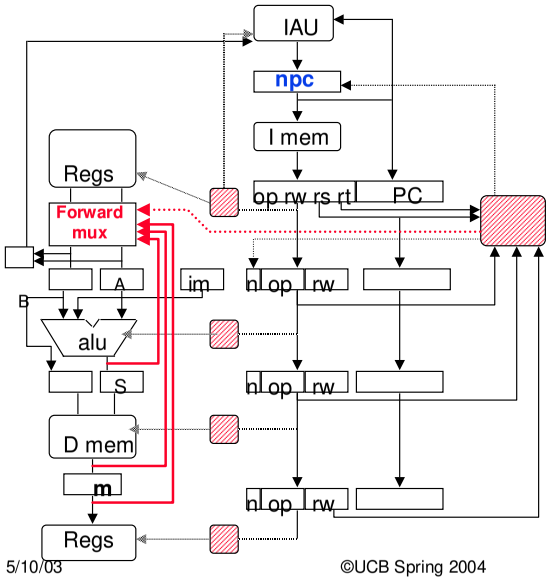


## Can pipelining get us into trouble?

- 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**



## Resolve RAW by "forwarding" (or bypassing)

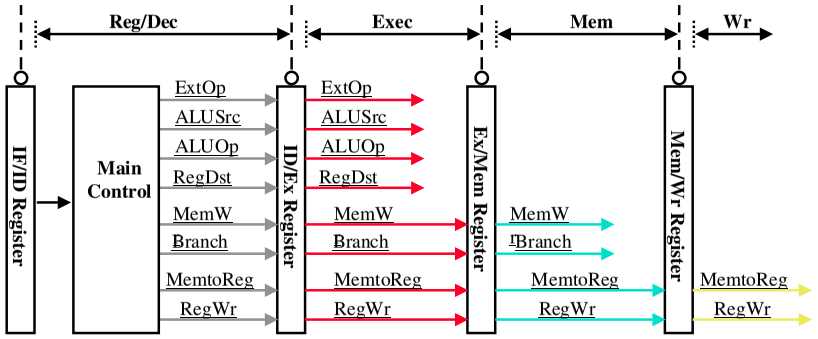



- Detect nearest valid write op operand register and forward into op latches, bypassing remainder of the pipe
- Increase muxes to add paths from pipeline registers
- Data Forwarding = Data Bypassing



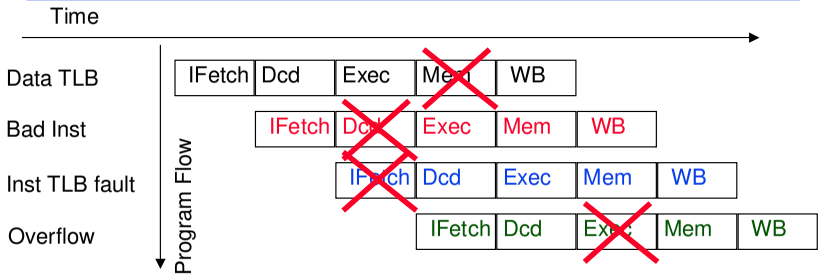
## Data Stationary Control

- 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






## 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!

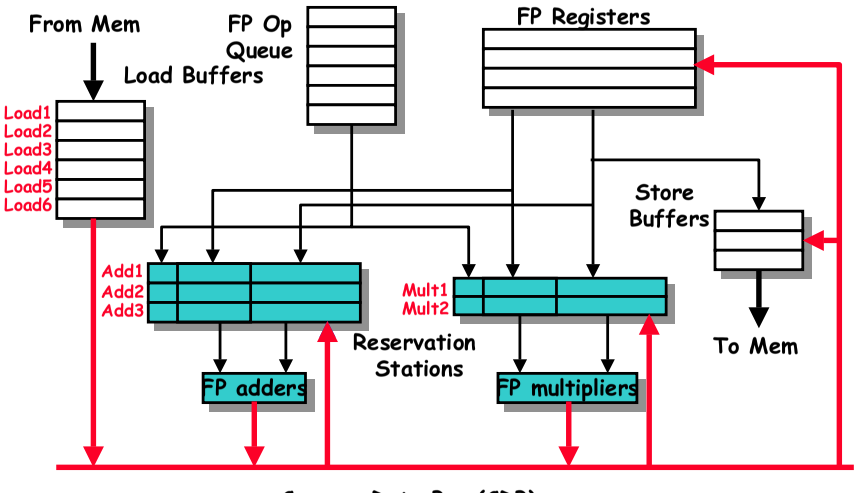
## You Did It!

- You all built a working 5-stage pipeline!

(Give yourselves a hand!)

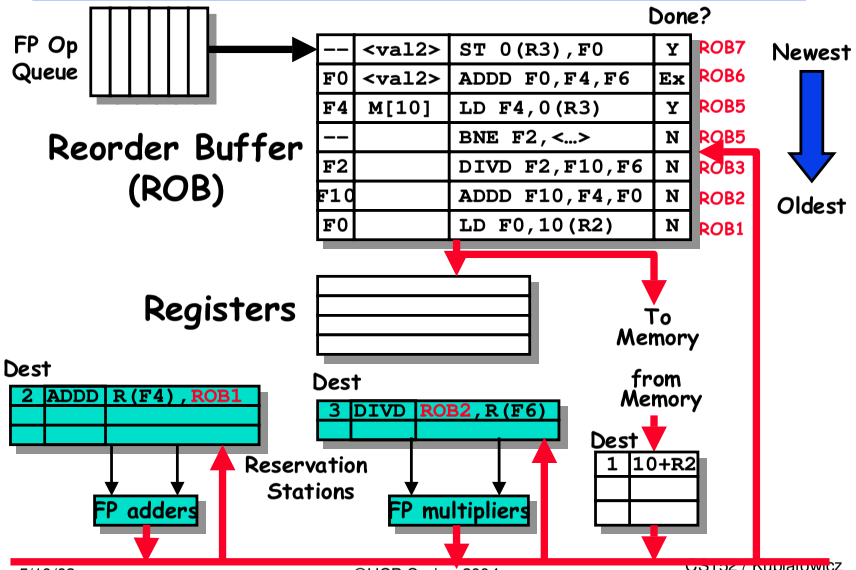


## Out of order execution: Tomasulo Organization






## Tomasulo With Reorder buffer (ROB)

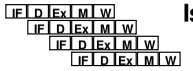
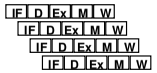
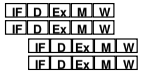
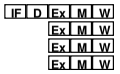





## Prediction: Branches, Dependencies, Data

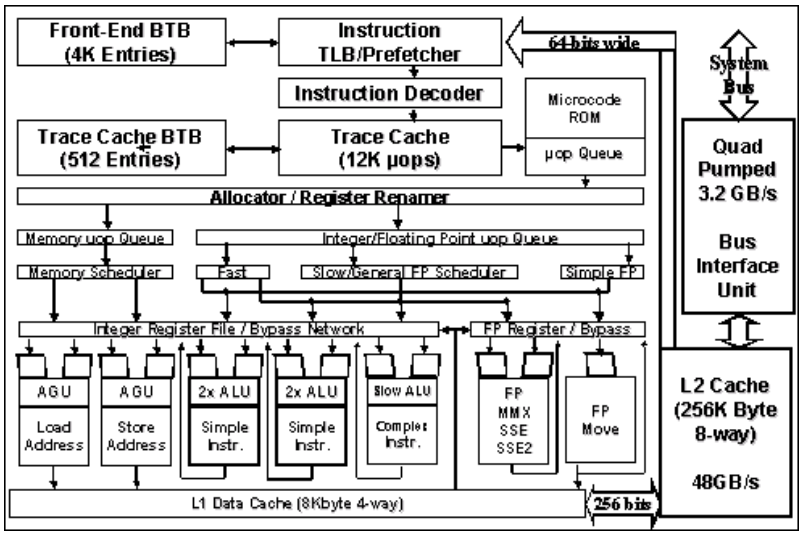
- Prediction has become essential to getting good performance from scalar instruction streams.
- Architects are now predicting everything: *data dependencies, actual data,* and *results of groups of instructions:*
  - At what point does computation become a probabilistic operation + verification?
  - We are pretty close with control hazards already…
- Why does prediction work?
  - Underlying algorithm has regularities.
  - Data that is being operated on has regularities.
  - Instruction sequence has redundancies that are artifacts of way that humans/compilers think about problems.
- Prediction ⇒ Compressible information streams?

## How can the machine exploit available ILP?

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

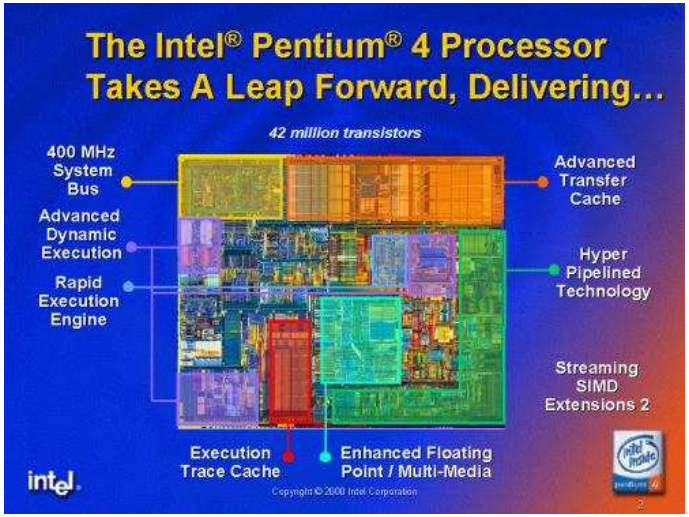


## Pentium 4 Block Diagram





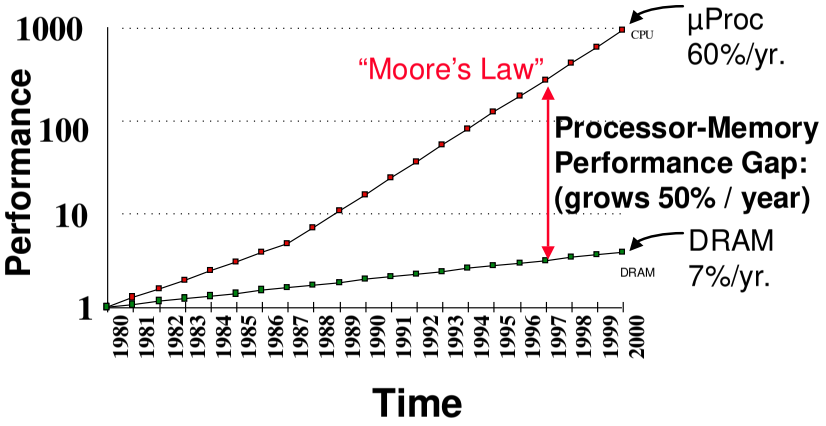

## Pentium-4 die floor plan



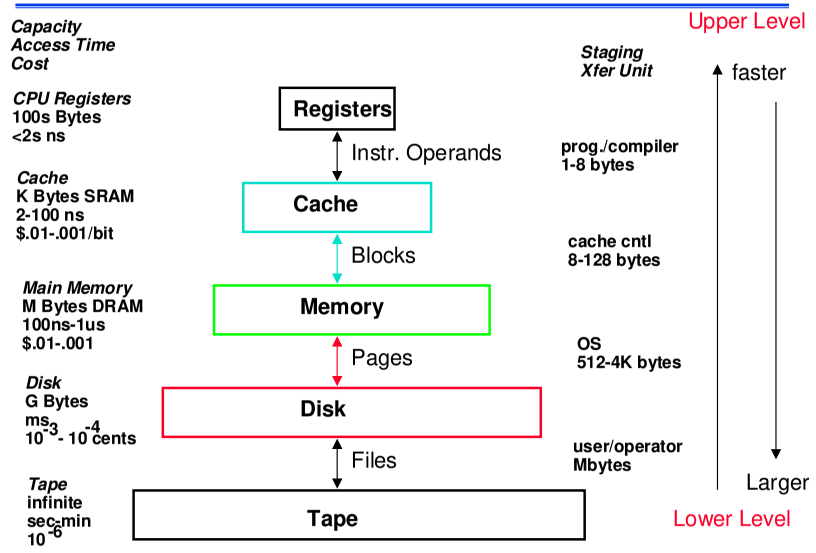

L1 Dcache
L2 cache



## Processor-DRAM Gap (latency)



## Levels of the Memory Hierarchy





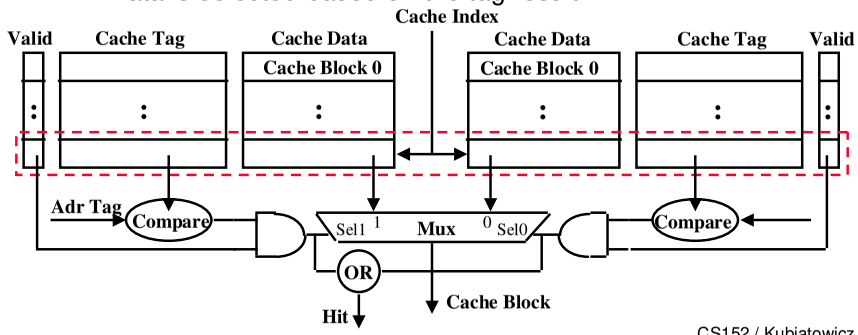

## Memory Hierarchy

- 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



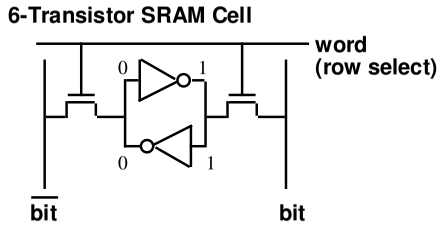
## Set Associative Cache

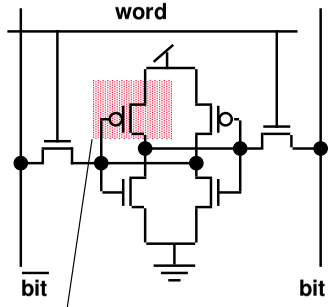
- N-way set associative: N entries for each Cache Index
  - N direct mapped caches operates in parallel
- Example: Two-way set associative cache
  - Cache Index selects a "set" from the cache
  - The two tags in the set are compared to the input in parallel
  - Data is selected based on the tag result






## Static RAM Cell

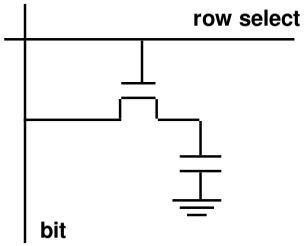



- Write:
  1. Drive bit lines (bit=1, bit=0)
  2.. Select row
- Read:
  1. Precharge bit and $\overline{\text{bit}}$ to Vdd or Vdd/2 => make sure equal!
  2.. Select row
  3. Cell pulls one line low
  4. Sense amp on column detects difference between bit and $\overline{\text{bit}}$

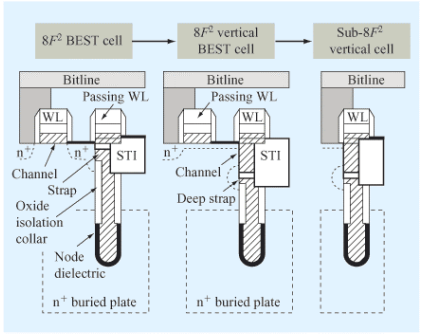
## 1-Transistor Memory Cell (DRAM)

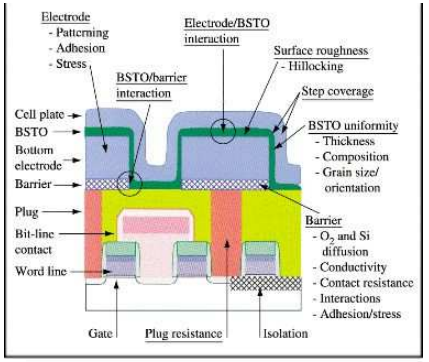
- Write:
  - 1. Drive bit line
  - 2.. Select row
- Read:
  - 1. Precharge bit line to Vdd
  - 2.. Select row
  - 3. Cell and bit line share charges
    - Very small voltage changes on the bit line
  - 4. Sense (fancy sense amp)
    - Can detect changes of ~1 million electrons
  - 5. Write: restore the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






## DRAM Capacitors: more capacitance in a small area

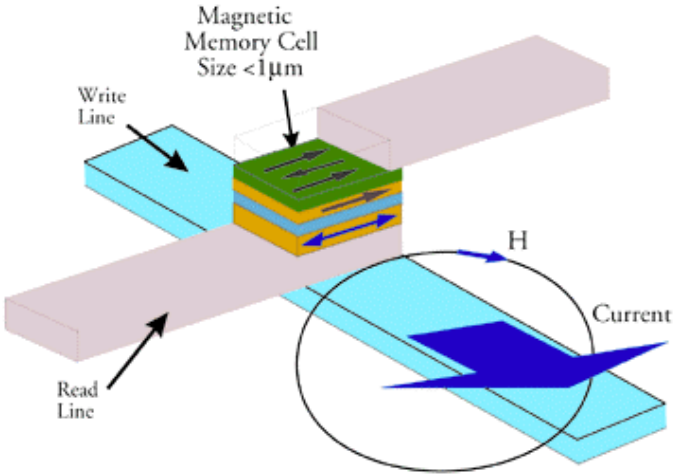



- Trench capacitors:
  - Logic ABOVE capacitor
  - Gain in surface area of capacitor
  - Better Scaling properties
  - Better Planarization
- Stacked capacitors
  - Logic BELOW capacitor
  - Gain in surface area of capacitor
  - 2-dim cross-section quite small

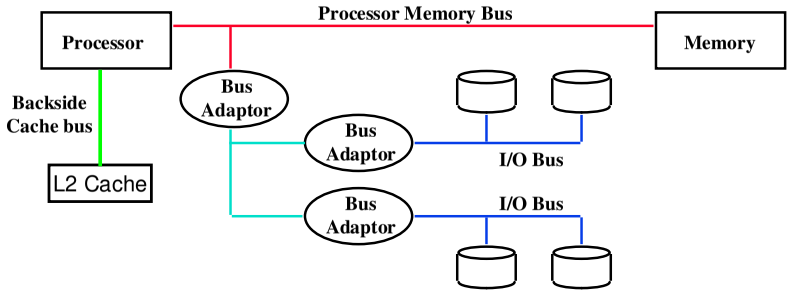


## Computers in the news: Tunneling Magnetic Junction





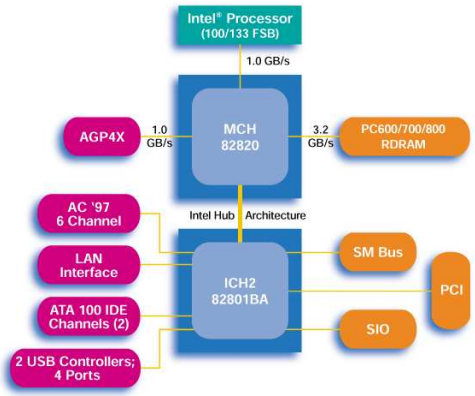

## A Three-Bus System (+ backside cache)




- 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

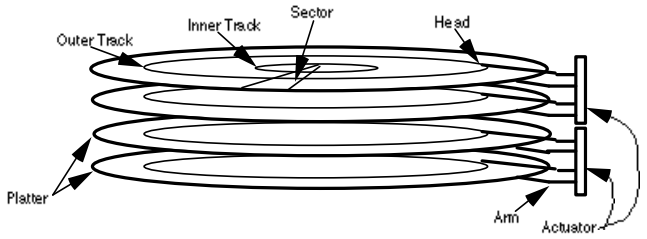
## Main componenets of Intel Chipset: Pentium II/III

- Northbridge:
  - Handles memory
  - Graphics
- Southbridge: I/O
  - PCI bus
  - Disk controllers
  - USB controlers
  - Audio
  - Serial I/O
  - Interrupt controller
  - Timers






## 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**

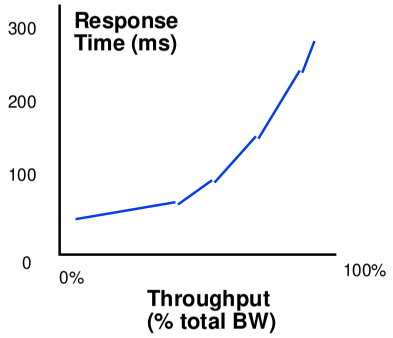
**Rotate: 4.2 ms @ 7200 rpm**

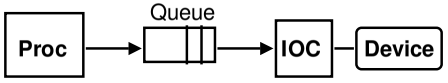
**Xfer: 1 ms @ 7200 rpm**



## Disk I/O Performance

**Metrics:**
**Response Time**
**Throughput**

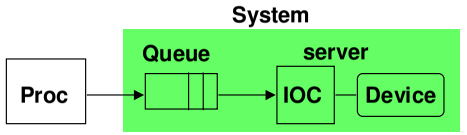



**Response time = Queue + Device Service time**



## 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

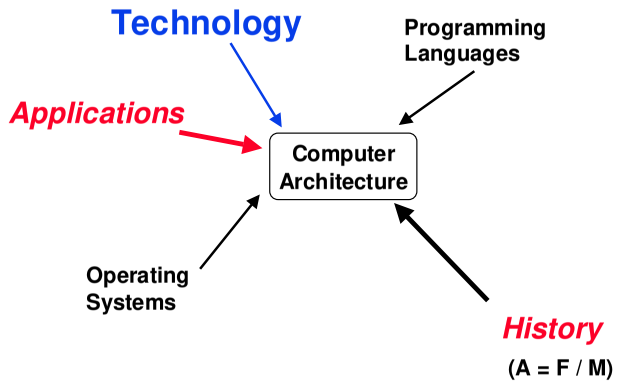
$u$ server utilization (0..1): $u = \lambda \times T_{ser}$

$T_q$ average time/customer in queue

## What does the future hold?



## Forces on Computer Architecture






## Today: building materials prevalent

- Originally: worried about squeezing the last ounce of performance from limited resources
- Today: worried about an abundance (embarrassment) of riches?
  - Billions of transistors on a chip (17nm Yeah!)
  - Microprocessor Report articles wondering if all the lessons of RISC are now irrelevant
- Moore's laws: exponential growth of everything
  - Transistors, Performance, Disk Space, Memory Size
- So, what matters any more????



## Key Technologies

° **Fast, cheap, highly integrated "computers-on-a-chip"**
  - **IDT R4640, NEC VR4300, StrongARM, Superchips**
  - **Devices everywhere!**

° **Micromechanical Devices (MEMS)**
  - **Integrated sensors everywhere**

° **Ubiquituous access to fast networks**
  - **Network is everywhere!**
  - **ISDN, Cable Modems, ATM, Metrocom (wireless) . .**

° **Platform independent programming languages**
  - **Java, JavaScript, Visual Basic Script**

° **Lightweight Operating Systems**
  - **GEOS, NCOS, RISCOS**

° **???**

## Issues for Research

- Complexity:
  - more than 50% of design teams now for verification
- Power
  - Processor designs hampered in performance to keep from melting
  - Why 3 or 4 orders of magnitude difference in power consumption between custom hardware and general Von Neuman architectures?
- Energy
  - Portable devices
- Scalability, Reliability, Maintainability
  - How to keep services up 24x7?
- Performance ("Cost conscious")
  - how to get performance without a lot of power, complexity, etc.
- Security?
  - What are the consequences of distributed info to privacy?
  - What are moral implications of ubiquitous sensors???



## 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")**



The End!