Microprocessor Design & Organization HCA2102

Computer Evolution and Performance

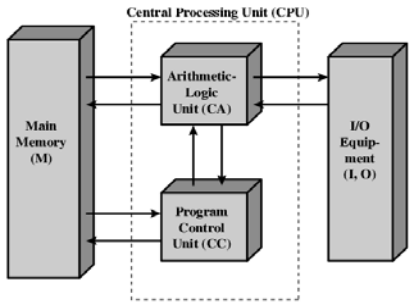
ENIAC - Background

- Electronic Numerical Integrator And Computer
- Eckert and Mauchly - University of Pennsylvania
- Trajectory tables for weapons
- Started 1943 - Finished 1946: Too late for war effort
- Used until 1955
- Decimal (not binary)
- 20 accumulators of 10 digits
- Programmed manually by switches
- 18,000 vacuum tubes - 30 tons - 15,000 square feet
- 140 kW power consumption - 5,000 additions per second

Von Neumann

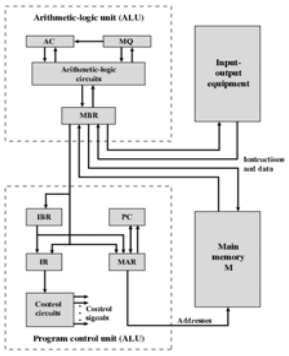
- Stored Program concept
- Main memory storing programs and data
- ALU operating on binary data
- Control unit interpreting instructions from memory and executing
- Input and output equipment operated by control unit
- Princeton Institute for Advanced Studies - IAS
- Completed 1952

Structure of Von Neumann machine



Structure of IAS

- 1000 x 40 bit words
  - Binary number
  - 2 x 20 bit instructions
- Set of registers (storage in CPU)
  - Memory Buffer Register
  - Memory Address Register
  - Instruction Register
  - Instruction Buffer Register
  - Program Counter
  - Accumulator
  - Multiplier Quotient



Commercial Computers

- 1947 - Eckert-Mauchly Computer Corporation
- UNIVAC I (Universal Automatic Computer)
- US Bureau of Census 1950 calculations
- Became part of Sperry-Rand Corporation
- Late 1950s - UNIVAC II -Faster with more memory

IBM

- Punched-card processing equipment
- 1953 - the 701: IBM's first stored program computer for Scientific calculations
- 1955 - the 702: Business applications
- Lead to 700/7000 series

Transistors

- Replaced vacuum tubes: Smaller, Cheaper, Less heat dissipation and Solid State device
- Made from Silicon (sand)
- Invented 1947 at Bell Labs by William Shockley et al.

Transistor Based Computers

- Second generation machines
- NCR & RCA produced small transistor machines
- IBM 7000
- DEC – 1957: Produced PDP-1

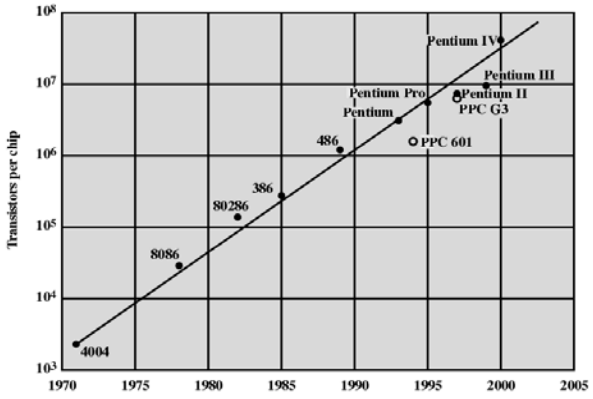
Generations of Computer

- Vacuum tube - 1946-1957
- Transistor - 1958-1964
- Small scale integration - 1965 on
  - Up to 100 devices on a chip
- Medium scale integration - to 1971
  - 100-3,000 devices on a chip
- Large scale integration - 1971-1977
  - 3,000 - 100,000 devices on a chip
- Very large scale integration - 1978 to date
  - 100,000 - 100,000,000 devices on a chip
- Ultra large scale integration
  - Over 100,000,000 devices on a chip

Moore's Law

- Increased density of components on chip
- Gordon Moore - co-founder of Intel
- Number of transistors on a chip will double every year
- Since 1970's development has slowed a little
  - Number of transistors doubles every 18 months
- Cost of a chip has remained almost unchanged
- Higher packing density means shorter electrical paths, giving higher performance
- Smaller size gives increased flexibility
- Reduced power and cooling requirements
- Fewer interconnections increases reliability

Growth in CPU Transistor Count



Intel

- 1971 - 4004
  - First microprocessor
  - All CPU components on a single chip
  - 4 bit
- Followed in 1972 by 8008
  - 8 bit
  - Both designed for specific applications
- 1974 - 8080
  - Intel's first general purpose microprocessor

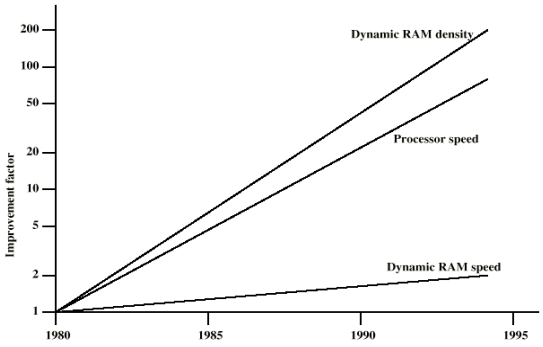
Speeding it up

- Pipelining
- On board L1 & L2 cache
- Branch prediction
- Speculative execution

Performance Mismatch

- Processor speed increased
- Memory capacity increased
- Memory speed lags behind processor speed

DRAM and Processor Characteristics



Solutions

- Increase number of bits retrieved at one time
  - Make DRAM "wider" rather than "deeper"
- Change DRAM interface
  - Cache
- Reduce frequency of memory access
  - More complex cache and cache on chip
- Increase interconnection bandwidth
  - High speed buses
  - Hierarchy of buses

Pentium Evolution (1)

- 8080
  - first general purpose microprocessor
  - 8 bit data path
  - Used in first personal computer – Altair
- 8086
  - much more powerful
  - 16 bit
  - instruction cache, prefetch few instructions
  - 8088 (8 bit external bus) used in first IBM PC
- 80286
  - 16 MByte memory addressable
  - up from 1Mb
- 80386
  - 32 bit
  - Support for multitasking

Pentium Evolution (2)

- 80486
  - sophisticated powerful cache and instruction pipelining
  - built in maths co-processor
- Pentium
  - Superscalar
  - Multiple instructions executed in parallel
- Pentium Pro
  - Increased superscalar organization
  - Aggressive register renaming
  - branch prediction
  - data flow analysis
  - speculative execution

Pentium Evolution (3)

- Pentium II
  - MMX technology
  - graphics, video & audio processing
- Pentium III
  - Additional floating point instructions for 3D graphics
- Pentium 4
  - Note Arabic rather than Roman numerals
  - Further floating point and multimedia enhancements
- Itanium
  - 64 bit
- See Intel web pages for detailed information on processors

Abstraction and Layering

- **Abstraction**: only way of dealing with complex systems
  - Divide world into objects, each with an…
    - **Interface**: knobs, behaviors, knobs -> behaviors
    - **Implementation**: "black box" (ignorance, apathy)
  - Only specialists deal with implementation, rest of us with interface
  - Example: car, only mechanics know how implementation works
- **Layering**: abstraction discipline makes life even simpler
  - Removes need to even know interfaces of most objects
  - Divide objects in system into layers
  - Layer X objects
    - Implemented in terms of interfaces of layer X-1 objects
    - Don't even need to know interfaces of layer X-2 objects
    - But sometimes helps if they do

Abstraction, Layering, and Computers

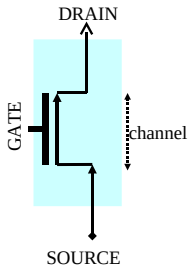
- Computers are complex systems, built in layers
  - Applications
  - O/S, compiler
  - Firmware, device drivers
  - Processor, memory, raw I/O devices
  - Digital circuits, digital/analog converters
  - Gates
  - Transistors
- 99% of users don't know hardware layers implementation
- 90% of users don't know implementation of any layer
- That's OK, world still works just fine
  - But unfortunately, the layers sometimes breakdown
  - Someone needs to understand what's "under the hood"

Abstraction, Layering, and Computers

- Computers are complex systems, built in layers
  - Applications
  - OS, compiler
  - Firmware, device drivers
  - Processor, memory, raw I/O devices
  - Digital circuits, DAC, ADC, MUX
  - Gates
  - Transistors
- 99% of users don't know hardware layers implementation
- 90% of users don't know implementation of any layer
- That's OK, world still works just fine
  - But unfortunately, the layers sometimes breakdown
  - Someone needs to understand what's "under the hood"

Shaping Force: Technology

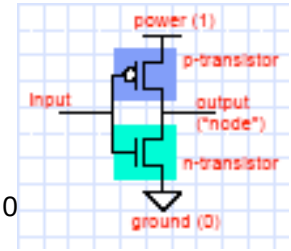
- Basic technology element: **MOSFET**
  - Invention of 20th century
  - **MOS**: metal-oxide-semiconductor
    - Conductor, insulator, semi-conductor
  - **FET**: field-effect transistor
  - Solid-state component acts like electrical switch
  - Channel conducts source to drain when voltage applied to gate



- **Channel length**: parameter (short & fast)
  - aka "feature size" or "technology"
  - Currently: 0.09 µm (0.09 micron), 90 nm
  - Continued miniaturization (scaling) known as "**Moore's Law**"
  - Won't last forever, physical limits approaching

Complementary MOS (CMOS)

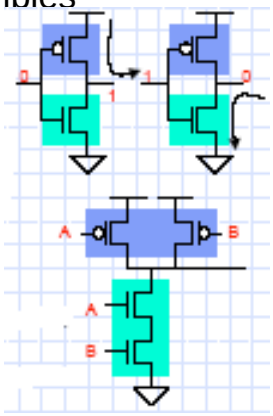
- Voltages as values
  - Power (VDD) = 1, Ground = 0
- Two kinds of MOSFETs
  - **N-transistors**
    - Conduct when gate voltage is 1
    - Good at passing 0s
  - **P-transistors**
    - Conduct when gate voltage is 0
    - Good at passing 1s



- **CMOS**: complementary MOS form Boolean logic

CMOS Examples

- Example 1: inverter
  - Case I: input = 0
    - P-transistor closed, n-transistor open
    - Power charges output (1)
  - Case II: input = 1
    - P-transistor open, n-transistor closed
    - Output discharges to ground (0)
- Example II: look at **truth table**
- 0, 0 = 1 0, 1 = 1
- 1, 0 = 1 1, 1 = 0
- Result: this is a **NAND gate.**
- NAND is universal (can build any logic function)



More About CMOS and Technology

- Two different **CMOS** families
- **SRAM (logic)**: used to make processors
  - Storage implemented as inverter pairs
  - Optimized for speed
- **DRAM (memory)**: used to make memory
  - Storage implemented as capacitors
  - Optimized for density, cost, power
- Disk is also a "technology", but isn't transistor-based.

VLSI + Manufacturing

- **VLSI (very large scale integration)**
  - MOSFET manufacturing process
  - As important as invention of MOSFET itself
  - Multi-step photochemical and electrochemical process
  - Fixed cost per step
  - Cost per transistor shrinks with transistor size
- Other production costs
  - Packaging
  - Test
  - Mask set
  - Design

MOSFET Side View

- **MOS**: three materials needed to make a transistor
  - **Metal** - Aluminum, Tungsten, Copper: conductor
  - **Oxide** - Silicon Dioxide (SiO2): insulator
  - **Semiconductor** - doped Si: conducts under certain conditions
- **FET**: field effect (the mechanism) transistor
  - Voltage on gate: current flows source to drain (transistor on)
  - No voltage on gate: no current (transistor off)

Empirical Evaluation

- Metrics
  - Cost
  - Performance
  - Power
  - Reliability
- Often more important in combination than individually
  - Performance/cost (MIPS/$)
  - Performance/power (MIPS/W)
- Basis for
  - Design decisions
  - Purchasing decisions

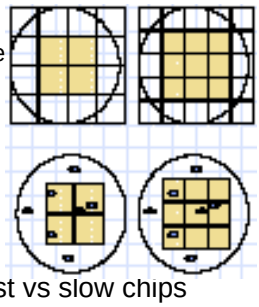
Cost

- Metric: **$**
- Usually CPU accounts for fraction of cost
  - Some of that is profit
- We are concerned about Intel's cost (transfers to you)
  - Unit cost: costs to manufacture individual chips
  - Startup cost: cost to design chip, build the fabrication line, marketing, etc

	Desktop	Laptop	PDA	Phone
$	$100–$300	$150-$350	$50–$100	$10–$20
% of total	10–30%	10–20%	20–30%	20-30%
Other costs	Memory, display, power supply/battery, disk, packaging			

Unit Cost: Integrated Circuit

- Chips built in multi-step chemical processes on **wafers**
  - Cost / wafer is constant, f(wafer size, number of steps)
- Chip (die) cost is proportional to **area**
  - Larger chips means fewer of them
  - Larger chips means fewer working one
  - Why? Uniform defect density
  - Chip cost $\propto$ chip area
  - α = 2 to 3
- **Wafer yield**: % wafer that is chips
- **Die yield**: % chips that work
- Yield is increasingly non-binary - fast vs slow chips



Yield/Cost Examples

- Parameters
  wafer yield = 90%, α = 2, defect density = 2/cm^2

Die size (mm^2)	100	144	196	256	324	400
Die yield	23%	19%	16%	12%	11%	10%
6" Wafer	139(31)	90(16)	62(9)	44(5)	32(3)	23(2)
8" Wafer	256(59)	177(32)	124(19)	90(11)	68(7)	52(5)
10" Wafer	431(96)	290(53)	206(32)	153(20)	116(13)	90(9)

	Wafer Cost	Defect (/cm^2)	Area (mm^2)	Dies	Yield	Die Cost	Package Cost (pins)	Test Cost	Total
Intel 486DX2	$1200	1.0	81	181	54%	$12	$11(168)	$12	$35
IBM PPC601	$1700	1.3	196	66	27%	$95	$3(304)	$21	$119
DEC Alpha	$1500	1.2	234	53	19%	$149	$30(431)	$23	$202
Intel Pentium	$1500	1.5	296	40	9%	$417	$19(273)	$37	$473

Startup Costs

Startup costs: must be amortized over chips sold
- Research and development: ~$100M per chip
  - 500 person-years @ $200K per person
- Fabrication facilities: ~$2B per new line
- Clean rooms (bunny suits), lithography, testing equipment

- If you sell 10M chips, startup adds ~$200 to cost of each
  - Companies (e.g., Intel) don't make money on new chips
  - They make money on proliferations (shrinks and frequency)
  - No startup cost for these

Moore's Effect on Cost

- Scaling has opposite effects on unit and startup costs
  + Reduces unit integrated circuit cost
    - Either lower cost for same functionality…
    - Or same cost for more functionality
  – Increases startup cost
    - More expensive fabrication equipment
    - Takes longer to design, verify, and test chips

Performance

- Two definitions
  - **Latency (execution time)**: time to finish a fixed task
  - **Throughput (bandwidth)**: number of tasks in fixed time
  - Very different: throughput can exploit parallelism, latency cannot
    - Baking bread analogy
  - Often contradictory
  - Choose definition that matches goals (most frequently throughput)
- Example: move people from A to B: 10 miles
  - Car: capacity = 5, speed = 60 miles/hour
  - Bus: capacity = 60, speed = 20 miles/hour
  - Latency: **car = 10 min**, bus = 30 min
  - Throughput: car = 15 PPH (count return trip), **bus = 60 PPH**

Performance Improvement

- Processor A is X times faster than processor B if
  - Latency(P,A) = Latency(P,B) / X
  - Throughput(P,A) = Throughput(P,B) * X
- Processor A is X% faster than processor B if
  - Latency(P,A) = Latency(P,B) / (1+X/100)
  - Throughput(P,A) = Throughput(P,B) * (1+X/100)

Car/bus example
- Latency? Car is 3 times (and 200%) faster than bus
- Throughput? Bus is 4 times (and 300%) faster than car

What Is 'P' in Latency (P,A)? *Program*

SPEC Benchmarks

- SPEC (Standard Performance Evaluation Corporation)
  - http://www.spec.org/
  - Consortium of companies that collects, standardizes, and distributes benchmark programs
  - Post **SPECmark** results for different processors
    - 1 number that represents performance for entire suite
  - Benchmark suites for CPU, Java, I/O, Web, Mail, etc.
  - Updated every few years: so companies don't target benchmarks
- SPEC CPU 2000
  - 12 "integer": gzip, gcc, perl, crafty (chess), vortex (DB), etc.
  - 14 "floating point": mesa (openGL), equake, facerec, etc.
  - Written in C and Fortran (a few in C++)

Adding/Averaging Performance Numbers

- You can add latencies, but not throughput
  - Latency(P1+P2, A) = Latency(P1,A) + Latency(P2,A)
  - Throughput(P1+P2,A) != Throughput(P1,A) + Throughput(P2,A)
    - 1 mile @ 30 miles/hour + 1 mile @ 90 miles/hour
      - Average is **not** 60 miles/hour
    - 0.033 hours at 30 miles/hour + 0.01 hours at 90 miles/hour
      - Average is only 47 miles/hour! (2 miles / (0.033 + 0.01 hours))
  - Throughput(P1+P2,A) = 1 / [(1/ Throughput(P1,A)) + (1/ Throughput(P2,A))]
- Same goes for means (averages)
  - **Arithmetic**: (1/N) * !P=1..N Latency(P)
    - For units that are proportional to time (e.g., latency)
  - **Harmonic**: N / !P=1..N 1/Throughput(P)
    - For units that are inversely proportional to time (e.g., throughput)
  - **Geometric**: N"#P=1..N Speedup(P)
    - For unitless quantities (e.g., speedups)

SPECmark

- Reference machine: Sun SPARC 10
- Latency SPECmark
  - For each benchmark
    - Take odd number of samples: on both machines
    - Choose median
    - Take latency ratio (Sun SPARC 10 / your machine)
  - Take GMEAN of ratios over all benchmarks
- Throughput SPECmark
  - Run multiple benchmarks in parallel on multiple-processor system
- Recent (latency) leaders
  - SPECint: Intel 3.4 GHz Pentium4 (1705)
  - SPECfp: IBM 1.9 GHz Power5 (2702)

CPU Performance Equation

- Multiple aspects to performance: helps to isolate them
- Latency (P,A) = seconds / program =
  - (instructions / program) * (cycles / instruction) * (seconds / cycle)
- **Instructions / program**: dynamic instruction count
  - Function of program, compiler, instruction set architecture (ISA)
- **Cycles / instruction**: CPI
  - Function of program, compiler, ISA, micro-architecture
- **Seconds / cycle**: clock period
  - Function of micro-architecture, technology parameters
- For low latency (better performance) minimize all three
  - Hard: often pull against the other

Danger: Partial Performance Metrics

- Micro-architects often ignore dynamic instruction count
  - Typically work in one ISA/one compiler ! treat it as fixed
- CPU performance equation becomes
  - seconds / instruction = (cycles / instruction) * (seconds / cycle)
  - This is a latency measure, if we care about throughput …
  - **Instructions / second** = (instructions / cycle) * (cycles / second)
- **MIPS** (millions of instructions per second)
  - Instructions / second * 10-6
  - **Cycles / second**: clock frequency (in MHz)
  - Example: CPI = 2, clock = 500 MHz, what is MIPS?
    - 0.5 * 500 MHz * 10-6 = 250 MIPS
- Example problem situation:
  - compiler removes instructions, program faster
  - However, “MIPS” goes down (misleading)

MIPS and MFLOPS (MegaFLOPS)

- Problem: MIPS may vary inversely with performance
  - Some optimizations actually add instructions
  - Work per instruction varies (e.g., FP mult vs. integer add)
  - ISAs are not equivalent
- **MFLOPS**: like MIPS, but counts only FP ops, because…
  - + FP ops can’t be optimized away
  - + FP ops have longest latencies anyway
  - + FP ops are same across machines
- May have been valid in 1980, but today…
  - Most programs are “integer”, i.e., light on FP
  - Loads from memory take much longer than FP divide
  - Even FP instructions sets are not equivalent
- Upshot: MIPS not perfect, but more useful than MFLOPS

Danger: Partial Performance Metrics II

- Micro-architects often ignore dynamic instruction count…
- … but general public (mostly) also ignores CPI
  - Equates clock frequency with performance!!
- Which processor would you buy?
  - Processor A: CPI = 2, clock = 500 MHz
  - Processor B: CPI = 1, clock = 300 MHz
  - Probably A, but B is faster (assuming same ISA/compiler)
- Classic example
  - 800 MHz PentiumIII faster than 1 GHz Pentium4
  - Same ISA and compiler

Cycles per Instruction (CPI)

- **Improving CPI**
  - Cycle/instruction for **average instruction**
  - **IPC** = 1/CPI
    - Used more frequently than CPI, but harder to compute with
  - Different instructions have different cycle costs
    - E.g., integer add typically takes 1 cycle, FP divide takes > 10
  - Assumes you know something about instruction frequencies
- CPI example
  - A program executes equal integer, FP, and memory operations
  - Cycles per instruction type: integer = 1, memory = 2, FP = 3
  - What is the CPI? (0.33 * 1) + (0.33 * 2) + (0.33 * 3) = 2

Another CPI Example

- Assume a processor with instruction frequencies and costs
  - Integer ALU: 50%, 1 cycle
  - Load: 20%, 5 cycle
  - Store: 10%, 1 cycle
  - Branch: 20%, 2 cycle
- Which change would improve performance more?
  - A. Branch prediction to reduce branch cost to 1 cycle?
  - B. A bigger data cache to reduce load cost to 3 cycles?
- Compute CPI
  - Base = 0.5*1 + 0.2*5 + 0.1*1 + 0.2*2 = 2
  - A = 0.5*1 + 0.2*5 + 0.1*1 + 0.2*1 = 1.8
  - B = 0.5*1 + 0.2*3 + 0.1*1 + 0.2*2 = 1.6 (**winner**)

Increasing Clock Frequency: Pipelining

- Reduce pipeline stage delay
  - Reduce logic levels and wire lengths (better design)
  - Complementary to technology efforts
  - Increase number of pipeline stages (multi-stage operations)
  - – Often causes CPI to increase
  - – At some point, actually causes performance to decrease
  - "Optimal" pipeline depth is program and technology specific
- Remember example
  - Pentium III: 12 stage pipeline, 800 MHz faster than
  - Pentium4: 22 stage pipeline, 1 GHz
  - Next Intel design: more like PentiumII (more later)

CPI and Clock Frequency

- System components "clocked" independently
  - E.g., Increasing processor clock frequency doesn't improve memory performance
- Example
  - Processor A: CPICPU = 1, CPIMEM = 1, clock = 500 MHz
  - What is the speedup if we double clock frequency?
  - Base: CPI = 2 -> IPC = 0.5 -> MIPS = 250
  - New: CPI = **3 ->** IPC = 0.33-> MIPS = 333
    - Clock *= 2 -> CPIMEM *= 2
  - Speedup = 333/250 = 1.33 << 2
- What about an infinite clock frequency?
  - Only a x2 speedup (Example of Amdahl's Law)

Improving CPI

- Historically, clock accounts for 70%+ of performance improvement
  - Achieved via deeper pipelines
- That will (have to) change
  - Deep pipelining is not power efficient
  - Physical speed limits are approaching
  - 1GHz: 1999, 2GHz: 2001, 3GHz: 2002, 4GHz? almost 2006
- Techniques we will look at
  - Caching, speculation, multiple issue, out-of-order issue
  - Vectors, multiprocessing, more…
- Moore helps because CPI reduction requires transistors
  - The definition of parallelism is "more transistors"
  - But best example is caches

Moore's Effect on Performance

- **Moore's Curve**: common interpretation of Moore's Law
- "CPU performance doubles every 18 months"
- Self fulfilling prophecy
  - 2X every 18 months is ~1% per week
  - Q: Would you add a feature that improved performance 20% if it took 8 months to design and test?
- Processors under Moore's Curve (arrive too late) fail spectacularly
  - E.g., Intel's Itanium, Sun's Millennium

Performance Rules of Thumb

- Make common case fast
  - Sometimes called "**Amdahl's Law**"
  - Corollary: don't optimize 1% to the detriment of other 99%
- Build a balanced system
  - Don't over-engineer capabilities that cannot be utilized
- Design for actual, not peak, performance
  - For actual performance X, machine capability must be > X

Transistor Speed, Power, and Reliability

Transistor characteristics and scaling impact:

- Switching speed
- Power
- Reliability

- "Undergrad" gate delay model for **architecture**
  - Each Not, NAND, NOR, AND, OR gate has delay of "1"
  - Reality is not so simple

Simple RC Delay Model

Switching time is a RC circuit (charge or discharge)

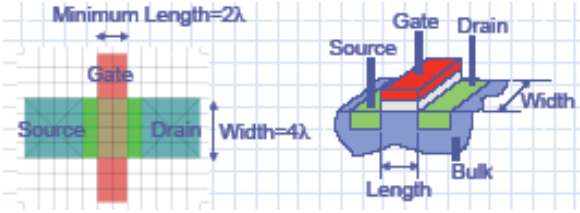
- R - Resistance: slows rate of current flow
  - Depends on material, length, cross-section area
- C - Capacitance: electrical charge storage
  - Depends on material, area, distance
- Voltage affects speed, too
- Transistor channel resistance
  - function of Vg (gate voltage)
  - Wire resistance (negligible for short wires)
- Source/Drain capacitance
  - Gate capacitance
  - Wire capacitance (negligible for short wires)

RC Delay Model Ramifications

- Want to reduce resistance
  - "wide" drive transistors (width specified per device)
  - Short wires
- Want to reduce capacitance
  - **Number of connected devices**
  - Less-wide transistors (gate capacitance of next stage)
  - Short wires

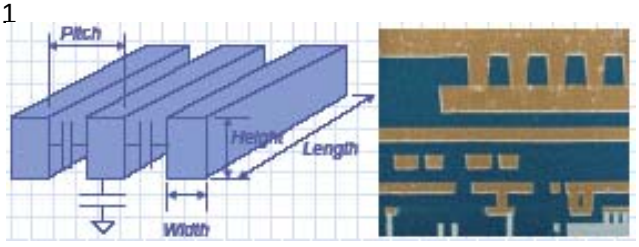
Transistor Scaling

- Transistor length is key property of a "process generation"
  - 90nm refers to the transistor gate length, same for all transistors
- Shrink transistor length:
  - Lower resistance of channel (shorter)
  - Lower gate/source/drain capacitance
- Result: transistor drive strength linear as gate length shrinks



Wires

- Resistance fixed by (length*resistivity) / (height*width)
  - bulk aluminum 2.8 μΩ/cm, bulk copper 1.7 μΩ/cm
- Capacitance depends on geometry of surrounding wires and relative permittivity, εr, of dielectric
- silicon dioxide εr = 3.9, new low-k dielectrics in range 1.2-3.1



Wire Delay

- RC Delay of wires
  - Resistance proportional to length
  - Capacitance proportional to length
- Result: delay of a wire is quadratic in length
  - Insert "inverter" repeaters for long wires to
  - Bring it back to linear delay

Moore's Effect on RC Delay

- Scaling helps reduce wire and gate delays
  - In some ways, hurts in others
  - + Wires become shorter (Length(! Resistance()
  - + Wire "surface areas" become smaller (Capacitance()
  - + Transistors become shorter (Resistance()
  - + Transistors become narrower (Capacitance(, Resistance*)
  - – Gate insulator thickness becomes smaller (Capacitance*)
  - – Distance between wires becomes smaller (Capacitance*)

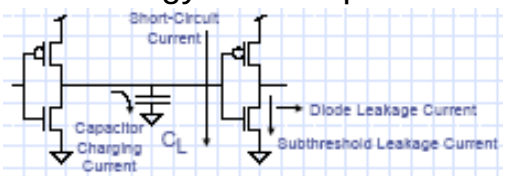
Improving RC Delay

- Exploit good effects of scaling
- Fabrication technology improvements
  - + Use copper instead of aluminum for wires ('(! Resistance()
  - + Use lower-dielectric insulators ()(! Capacitance()
  - + Increase Voltage
- + Design implications
  - + Use bigger cross-section wires (Area* ! Resistance()
    - Typically means taller, otherwise fewer of them
    - – Increases "surface area" and capacitance (Capacitance*)
- + Use wider transistors (Area* ! Resistance()
  - – Increases capacitance (not for you, for upstream transistors)
  - – Use selectively

Another Constraint: Power and Energy

- **Power** (Watt or Joule/Second): short-term (peak, max)
  - Mostly a **dissipation** (heat) concern
    - Power-density (Watt/cm2): important related metric
  - – Thermal cycle: power dissipation* -> power density* -> temperature* -> resistance* -> power dissipation*…
  - Cost (and form factor): packaging, heat sink, fan, etc.
- **Energy** (Joule): long-term
  - Mostly a **consumption** concern
  - Primary issue is battery life (cost, weight of battery, too)
  - Low-power implies low-energy, but not the other way around

Sources of Energy Consumption



Dynamic power

- Capacitor Charging (85-90% of active power)
  - Energy is ½CV^2 per transition
- Short-Circuit Current (10-15% of active power)
  - When both p and n transistors turn on during signal transition

Static power:

- Sub-threshold Leakage (dominates when inactive)
  - Transistors don't turn off completely
- Diode Leakage (negligible)
  - Parasitic source and drain diodes leak to substrate

Moore's Effect on Power

- Scaling has largely **good** effects on local power
  - + Shorter wires/smaller transistors (Length(! Capacitance()
  - – Shorter transistor length (Resistance(, Capacitance()
  - – Global effects largely undone by increased transistor counts
- Scaling has a largely negative effect on **power density**
  - + Transistor/wire power decreases linearly
  - – Transistor/wire density decreases quadratically
  - – Power-density increases linearly
    - Thermal cycle
  - Controlled somewhat by reduced VDD (5->3.3->1.6->1.3->1.1)
  - Reduced VDD sacrifices some switching speed

Reducing Power

Reduce supply voltage (VDD)

- + Reduces dynamic power quadratically and static power linearly
- But poses a tough choice regarding VT
- – Constant VT slows circuit speed -> clock frequency -> performance
- – Reduced VT increases static power **exponentially**

- Reduce clock frequency (f)
  - + Reduces dynamic power linearly
  - – Doesn't reduce static power
  - – Reduces performance linearly
  - Generally doesn't make sense without also reduced VDD …
    - Except that frequency can be adjusted cycle-to-cycle and locally

Dynamic Voltage Scaling (DVS)

- **Dynamic voltage scaling (DVS)**
  - OS reduces voltage/frequency when peak performance not needed

	Mobile PentiumIII "SpeedStep"	TM5400 "LongRun"	Intel X-Scale (StrongARM2)
Frequency	300–1000MHz (50MHz steps)	200–700MHz (33MHz steps)	50–800MHz (50MHz steps)
Voltage	0.9–1.7V (0.1V steps)	1.1–1.6V (continuous)	0.7–1.65V (continuous)
High-speed	3400MIPS @ 34W	1600MIPS @ 2W	800MIPS @ 0.9W
Low-power	1100MIPS @ 4.5W	300MIPS @ 0.25W	62MIPS @ 0.01W

  ± X-Scale is power efficient (6200 MIPS/W), but not IA32 compatible

Reducing Power: Processor Modes

- Modern electrical components have **low-power modes**
  - Note: no low-power disk mode, magnetic (non-volatile)
- "Standby" mode
  - Turn off internal clock
  - Leave external signal controller and pins on
  - Restart clock on interrupt
  - ± Cuts dynamic power linearly, doesn't effect static power
  - Laptops go into this mode between keystrokes
- "Sleep" mode
  - Flush caches, OS may also flush DRAM to disk
  - Turn off processor power plane
  - – Needs a "hard" restart
  - + Cuts dynamic and static power
  - Laptops go into this mode after ~10 idle minutes

Reliability

- **Mean Time Between Failures (MTBF)**
  - How long before you have to reboot or buy a new one
  - Not very quantitative yet, people just starting to think about this
- CPU reliability small in grand scheme
  - Software most unreliable component in a system
    - Much more difficult to specify & test
    - Much more of it
  - Most unreliable hardware component … disk
    - Subject to mechanical wear

Moore's Bad Effect on Reliability

CMOS devices: CPU and memory
- Historically almost perfectly reliable
- Moore has made them less reliable over time

- Two sources of electrical faults
  - Energetic particle strikes (from sun)
    - Randomly charge nodes, cause bits to flip, **transient**
  - Electro-migration: change in electrical interfaces/properties
    - Temperature-driven, happens gradually, **permanent**
- Large, high-energy transistors are immune to these effects
  - – Scaling makes node energy closer to particle energy
  - – Scaling increases power-density which increases temperature
  - Memory (DRAM) was hit first: denser, smaller devices than SRAM

Moore's Good Effect on Reliability

- The key to providing reliability is **redundancy**
  - The same scaling that makes devices less reliable…
  - Also increase device density to enable redundancy
- Classic example
  - Error correcting code (ECC) for DRAM
  - ECC also starting to appear for caches
  - More reliability techniques later
- Today's big open questions
  - Can we protect logic?
  - Can architectural techniques help hardware reliability?
  - Can architectural techniques help with software reliability?

Summary: A Global Look at Moore

- Device scaling (Moore's Law)
  - + Increases performance
    - Reduces transistor/wire delay
    - Gives us more transistors with which to reduce CPI
  - + Reduces local power consumption
    - – Which is quickly undone by increased integration
    - – Aggravates power-density and temperature problems
  - – Aggravates reliability problem
    - +But gives us the transistors to solve it via redundancy
  - + Reduces unit cost
    - – But increases startup cost
- Will we fall off Moore's Cliff? (for real, this time?)
  - What's next: nanotubes, quantum-dots, optical, spin-tronics, DNA?