CS152
Computer Architecture and Engineering
Lecture 25

Low Power Design,
Advanced Intel Processors

May 3, 2004

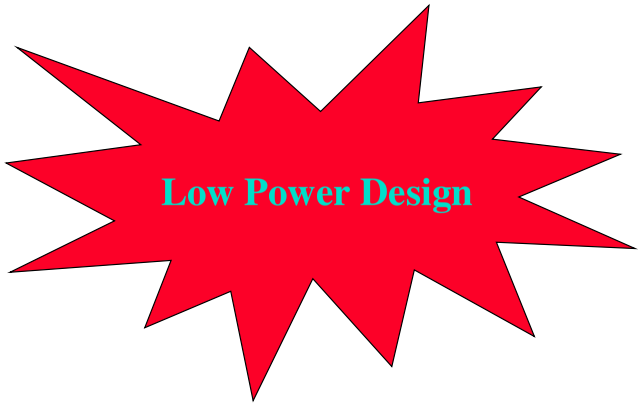
John Kubiatowicz (http.cs.berkeley.edu/~kubitron)

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

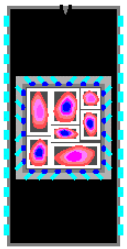
Recap: I/O Summary

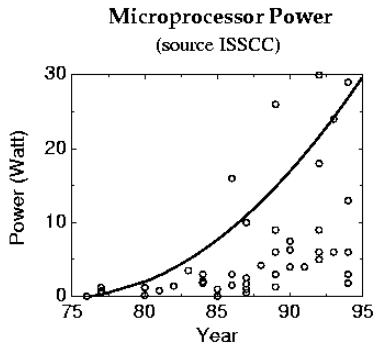
- I/O performance limited by weakest link in chain between OS and device
- Queueing theory is important
  - 100% utilization means very large latency
  - Remember, for M/M/1 queue (exponential source of requests/service)
    - queue size goes as u/(1-u)
    - latency goes as $T_{ser} \times u/(1-u)$
  - For M/G/1 queue (more general server, exponential sources)
    - latency goes as $m1(z) \times u/(1-u) = T_{ser} \times \{1/2 \times (1+C)\} \times u/(1-u)$
- Three Components of Disk Access Time:
  - Seek Time: advertised to be 8 to 12 ms. May be lower in real life.
  - Rotational Latency: 4.1 ms at 7200 RPM and 8.3 ms at 3600 RPM
  - Transfer Time: 2 to 50 MB per second
- I/O device notifying the operating system:
  - Polling: it can waste a lot of processor time
  - I/O interrupt: similar to exception except it is asynchronous
- Delegating I/O responsibility from the CPU: DMA, or even IOP

Slides Borrowed from Bob Broderson



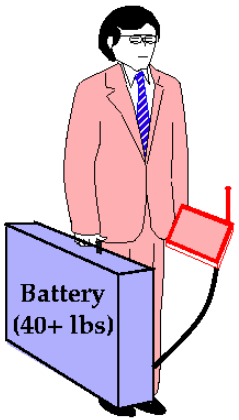
Low Power Design Problem





When supply voltage drops to 1Volt, then 100Watts = 100 Amps

Portable devices



Required Portable Functions

- Multimodal radio
- Protocols, ECC, ...
- Voice I/O compression & decompression
- Handwriting recognition
- Text/Graphics processing
- Video decompression
- Speech recognition
- Java interpreter

How to get 1 month of operation?

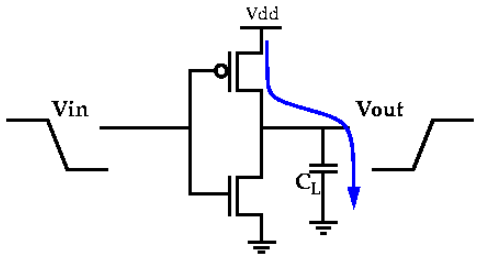
Two Kinds of Computation

- General purpose processing (what you have been studying so far)
  - Bursty - mostly idle with bursts of computation
  - Maximum possible throughput required during active periods
- Signal processing (for multimedia, wireless communications, etc.)
  - Stream based computation
  - No advantage in increasing processing rate above required for real-time requirements

Optimizing for Energy Consumption

- Conventional General Purpose processors (e.g. Pentiums)
  - Performance is everything ... somehow we'll get the power in and back out
  - 10-100 Watts, 100-1000 Mips = .01 Mips/mW
- Energy Optimized but General Purpose
  - Keep the generality, but reduce the energy as much as possible - e.g. StrongArm
  - .5 Watts, 160 Mips = .3 Mips/mW
- Energy Optimized and Dedicated
  - 100 Mops/mW

Switching Energy



Energy/transition = $C_L * V_{dd}^2$

Power = Energy/transition * f = $C_L * V_{dd}^2 * f$

Low Power & Low Energy System Design

System	Design partitioning, Power Down
Algorithm	Complexity, Concurrency, Locality, Regularity, Data representation
Architecture	Voltage scaling, Parallelism, Instruction set, Signal correlations
Circuit/Logic	Transistor Sizing, Logic optimization, Activity Driven Power Down, Low-swing logic, Adiabatic switching
Technology	Threshold Reduction, Multi-thresholds

Energy Reduction in CPU's

- Standard power management helps
  - Sleep modes
  - Power down blocks
- Clock rate reduction doesn't help
  - Number of operations = N_{ops}
  - Energy/operation = CV^2
  - Total Energy = $N_{ops} * CV^2$

Energy is independent of clock rate!

- Reducing the clock rate only degrades throughput, but no savings in battery life - unless the voltage is changed

Node Transition Activity and Power

Switch a CMOS gate for N clock cycles

$$E_N = C_L \bullet V_{dd}^2 \bullet n(N)$$

E_N: the energy consumed for N clock cycles

$n(N)$: the number of 0->1 transition in N clock cycles

$$P_{avg} = \lim_{N \to \infty} \frac{E_N}{N} \bullet f_{clk} = \left(\lim_{N \to \infty} \frac{n(N)}{N} \right) \bullet C_L \bullet V_{dd}^2 \bullet f_{clk}$$

$$\alpha_{0 \to 1} = \lim_{N \to \infty} \frac{n(N)}{N}$$

$$P_{avg} = \alpha_{0 \to 1} \bullet C_L \bullet V_{dd}^2 \bullet f_{clk}$$

Factors Affecting Transition Activity, $\alpha_{0->1}$

- "Static" component (does not account for timing)
  - ➟ Type of Logic Function (NOR vs. XOR)
  - ➟ Type of Logic Style (Static vs. Dynamic)
  - ➟ Signal Statistics
  - ➟ Inter-signal Correlations
- "Dynamic" or timing dependent component
  - ➟ Circuit Topology
  - ➟ Signal Statistics and Correlations

Static 2 Input NOR Gate

Assume:

prob(A=1) = 1/2

prob(B=1) = 1/2

Then:

prob(Out=1) = 1/2

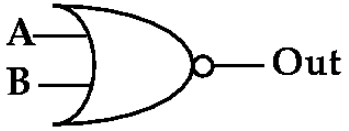
prob(0→1)

= ***prob***(Out=0).***prob***(Out=1)

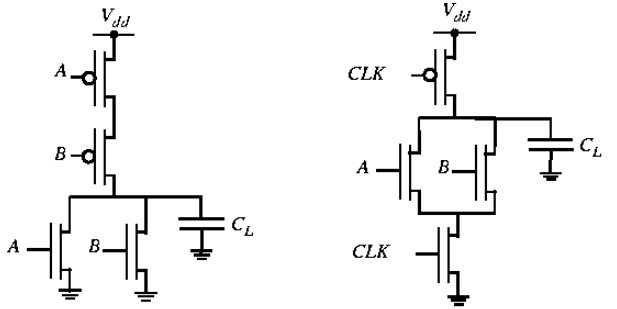
= 3/4 × 1/4 = 3/16

A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

$\alpha_{0->1} = 3/16$



Type of Logic Style: Static vs. Dynamic



Power is dissipated when Out=0

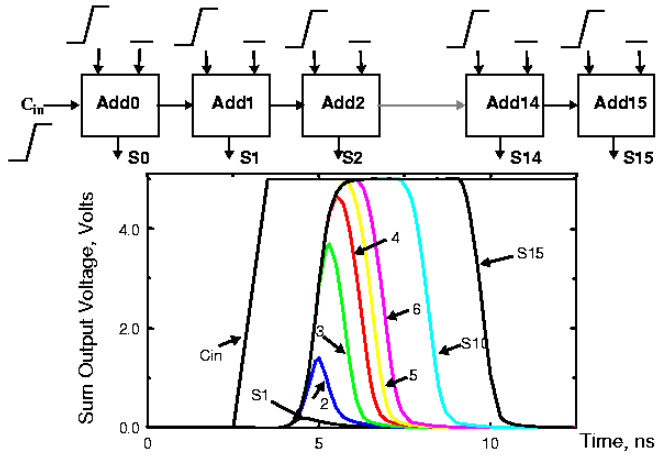
STATIC NOR

$\alpha_{0->1} = 3/16$

DYNAMIC NOR

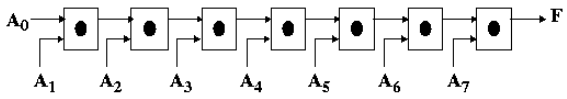
$\alpha_{0\to 1} = \frac{N_0}{2^N} = \frac{3}{4}$

"Dynamic" or Glitching Activity in CMOS

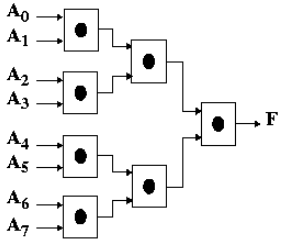


$\alpha_{0->1}$ **can be > 1 due to glitching!**

Glitch Reduction Using Balanced Paths

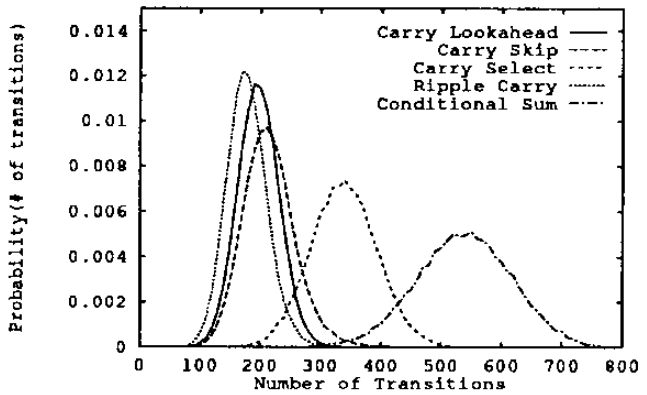


Ripple



Merge

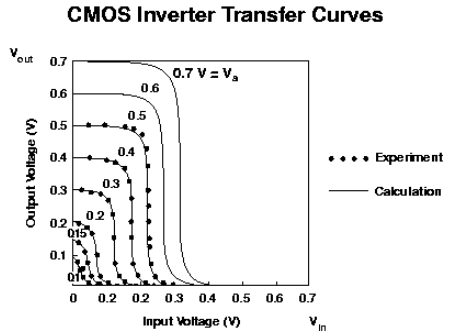
Comparison of Adder Topologies



Switching activity and capacitance minimization

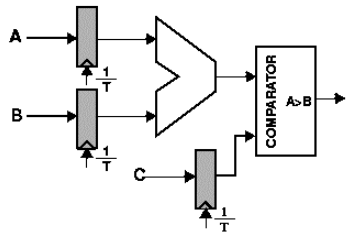
- Gated clocks. (*disable all modules not in use each cycle*)
- Block enables. (*enable only those modules using a bus*)
- Instruction Buffer. (*0th level cache*)
- Add stop and sleep instructions to the instruction set.
- Minimum size busses
- Minimize I/O - on-chip memory

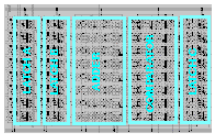
Minimum Supply Voltage



$V_{dd\text{-}min} \geq$ 50-100mV above V_t

Architecture Trade-offs - Reference Datapath



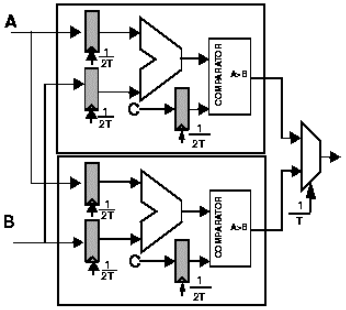


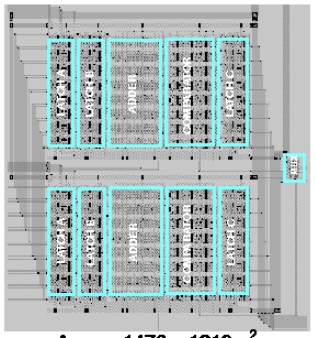
Area = 636 x 833 μ^2

- Critical path delay $\Rightarrow T_{adder} + T_{comparator}$ (= 25ns)
  $\Rightarrow f_{ref}$ = 40Mhz
- Total capacitance being switched = C_{ref}
- $V_{dd} = V_{ref} = 5V$
- Power for reference datapath = $P_{ref} = C_{ref} V_{ref}^2 f_{ref}$

from [Chandrakasan92] (*IEEE JSSC*)

Parallel Datapath

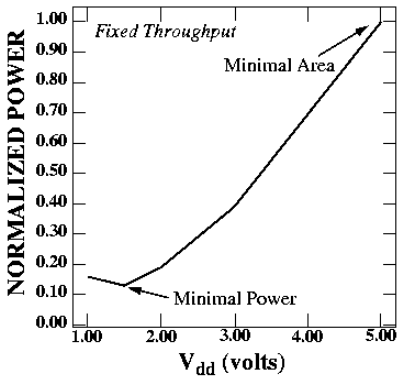




Area = 1476 x 1219 μ^2

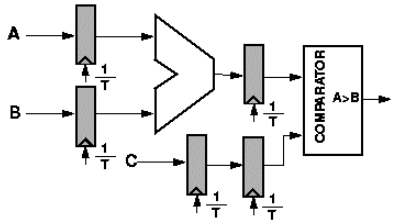
- The clock rate can be reduced by half with the same throughput $\Rightarrow f_{par} = f_{ref} / 2$
- $V_{par} = V_{ref} / 1.7$, $C_{par} = 2.15C_{ref}$
- $P_{par} = (2.15C_{ref})\ (V_{ref}/1.7)^2\ (f_{ref}/2) \approx 0.36\ P_{ref}$

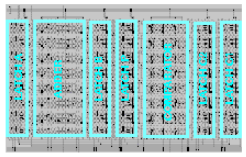
The More Parallel the Better??



Capacitance overhead starts to dominate at "high" levels of parallelism and results in an optimum voltage

Pipelined Datapath



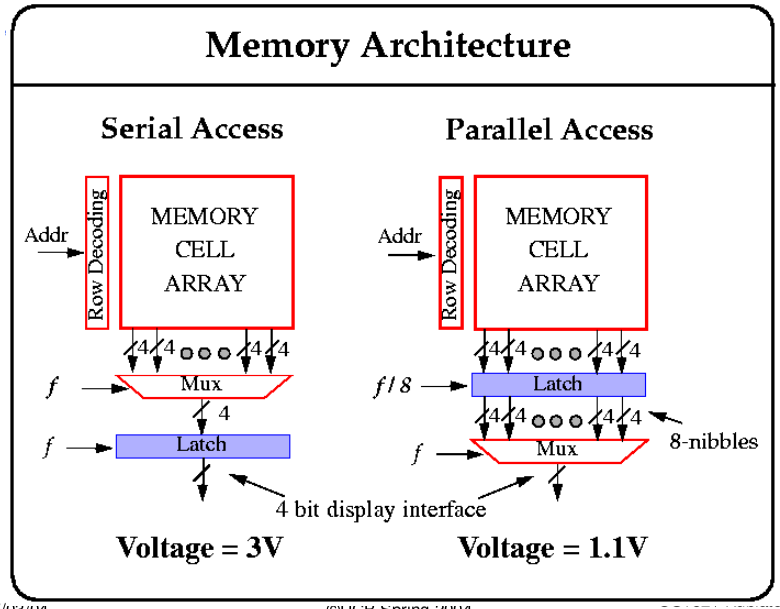


Area = 640 x 1081 μ^2

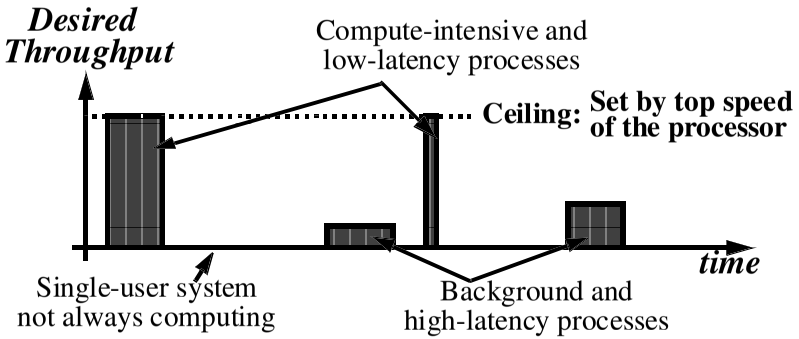
- Critical path delay is less $\Rightarrow$ max [T_{adder}, $T_{comparator}$]
- Keeping clock rate constant: $f_{pipe} = f_{ref}$
  Voltage can be dropped $\Rightarrow V_{pipe} = V_{ref} / 1.7$
- Capacitance slightly higher: $C_{pipe} = 1.15C_{ref}$
- $P_{pipe} = (1.15C_{ref})\ (V_{ref}/1.7)^2\ f_{ref} \approx 0.39\ P_{ref}$

Architecture Summary for a Simple

Architecture type	Voltage	Area	Power
Simple datapath (no pipelining or parallelism)	5V	1	1
Pipelined datapath	2.9V	1.3	0.39
Parallel datapath	2.9V	3.4	0.36
Pipeline-Parallel	2.0V	3.7	0.2



Back to original goal: Processor Usage Model

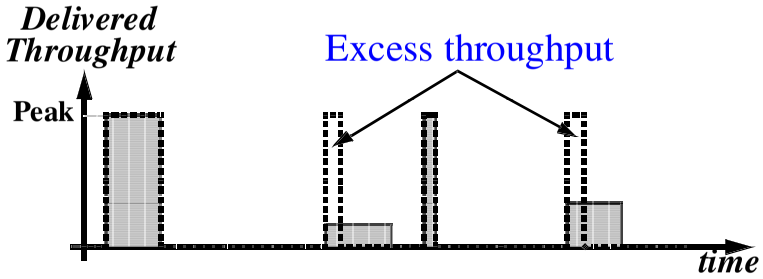


System Optimizations:

- **Maximize Peak Throughput**
- **Minimize Average Energy/operation**

(maximize computation per battery life)

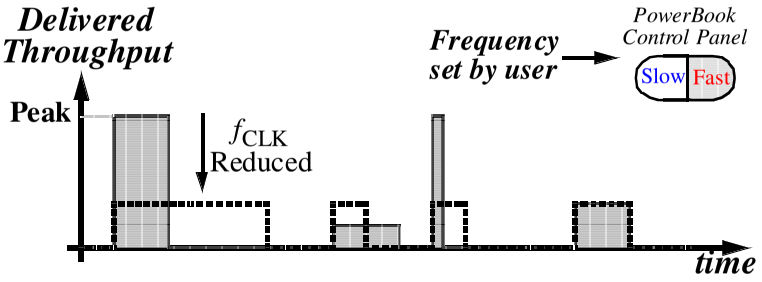
Typical Usage



Wake up → Compute ASAP → Go to idle/sleep mode

Always high throughput
Always high energy/operation

Another approach: Reduce Frequency

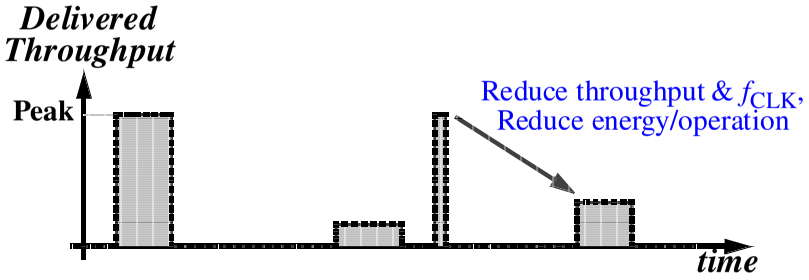


Energy/operation remains unchanged...
while throughput scales down with f_{CLK}

Problems:

- Circuits designed to be fast are now "wasted".
- Demand for peak throughput not met.

Alternative: Dynamic Voltage Scaling

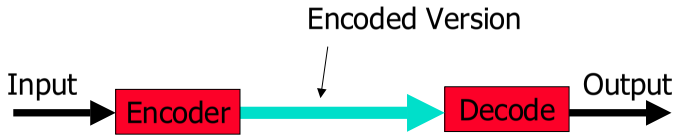


Dynamically scale energy/operation with throughput

Extend battery life by up to 10x with the same hardware!

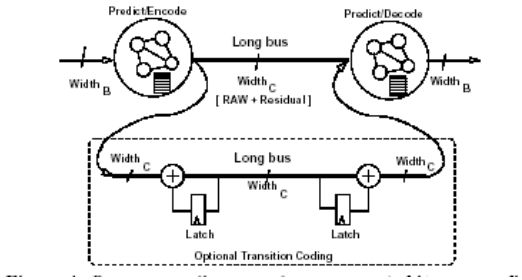
Key: Process scheduler determines operating point.

What about bus transitions?



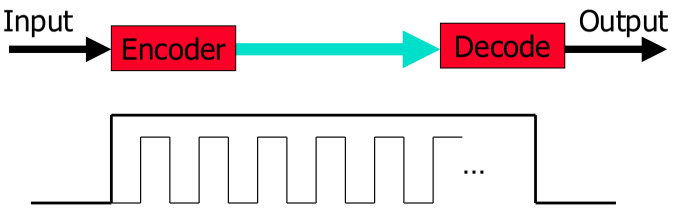
- **Can we reduce total number of transitions on buses by sophisticated bus drivers?**
- **Can we encode information in a way that takes less power?**
  - **Do this on chip?!**
  - **Trying to reduce total *number* of transitions**

Reasoning



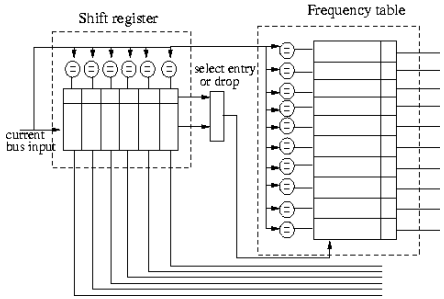
- **Increasing importance of wires relative to transistors**
  - **Spend transistors to drive wires more efficiently?**
  - **Try to reduce transitions over wires**
- **Orthogonal to other power-saving techniques**
  - **I.e. voltage reduction, low-swing drive**
  - **clock gating**
  - **Parallelism (like vectors!)**

Huffman-based Compression



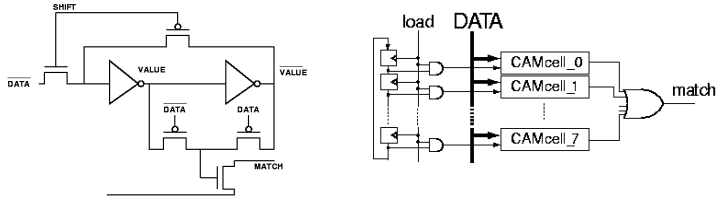
- **Variable bit length – problem!**
- **Possible soln: macro clock**
- **Less bits != less transitions**

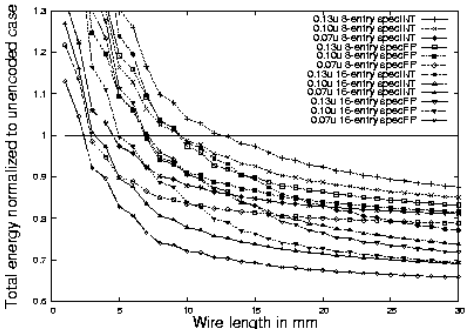
Context-based encoder



° Context-based encoder
- Detecting of repeated values going across bus
- Shift-register finds short-term frequent values
- Frequency table holds long-term values

Just the Shift-register: "window-based"





° Focus on shift-register
- 8 or 16 entries

° Careful Design can break-even!
- Register bus results
- 16 entires break even:
  - 7.0 mm for .13µm
  - 2.7 mm for .07µm

Administrivia

° Pending schedule:
- Wednesday 5/5: Midterm II. 5:30 – 8:30, 306 Soda hall
  - No class that day (I will be having office hours)
  - 1 page of handwritten notes, both sides
  - Fair topics:
    - Pipelining
    - Memory Systems
    - I/O, Disks, Queueing Theory
    - Power
  - Pizza at LaVal's afterwards
- Monday 5/10 (wrap up, evaluations, etc)
- Thursday 5/13: Oral reports: Times TBA
  - Signup sheet will be on my office door *next week*
  - Project reports must be submitted via web by 5pm on 5/10
- Monday 5/17: Final project reports due?

° Oral Report
- Powerpoint
- 20 minute presentation, 5 minutes for questions

7 Talk Commandments for a Bad Talk

I. Thou shalt not illustrate.

II. Thou shalt not covet brevity.

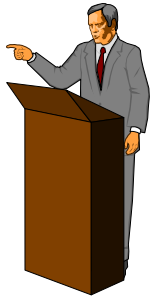
III. Thou shalt not print large.

IV. Thou shalt not use color.

V. Thou shalt not skip slides in a long talk.

VI. Thou shalt cover thy naked slides.

VII. Thou shalt not practice.



Following all the commandments

- We describe the philosophy and design of the control flow machine, and present the results of detailed simulations of the performance of a single processing element. Each factor is compared with the measured performance of an advanced von Neumann computer running equivalent code. It is shown that the control flow processor compares favorablylism in the program.
- We present a denotational semantics for a logic program to construct a control flow for the logic program. The control flow is defined as an algebraic manipulator of idempotent substitutions and it virtually reflects the resolution deductions. We also present a bottom-up compilation of medium grain clusters from a fine grain control flow graph. We compare the basic block and the dependence sets algorithms that partition control flow graphs into clusters.
- Our compiling strategy is to exploit coarse-grain parallelism at function application level: and the function application level parallelism is implemented by fork-join mechanism. The compiler translates source programs into control flow graphs based on analyzing flow of control, and then serializes instructions within graphs according to flow arcs such that function applications, which have no control dependency, are executed in parallel.
- A hierarchical macro-control-flow computation allows them to exploit the coarse grain parallelism inside a macrotask, such as a subroutine or a loop, hierarchically. We use a hierarchical definition of macrotasks, a parallelism extraction scheme among macrotasks defined inside an upper layer macrotask, and a scheduling scheme which assigns hierarchical macrotasks on hierarchical clusters.
- We apply a parallel simulation scheme to a real problem: the simulation of a control flow architecture, and we compare the performance of this simulator with that of a sequential one. Moreover, we investigate the effect of modelling the application on the performance of the simulator. Our study indicates that parallel simulation can reduce the execution time significantly if appropriate modelling is used.
- We have demonstrated that to achieve the best execution time for a control flow program, the number of nodes within the system and the type of mapping scheme used are particularly important. In addition, we observe that a large number of subsystem nodes allows more actors to be fired concurrently, but the communication overhead in passing control tokens to their destination nodes causes the overall execution time to increase substantially.
- The relationship between the mapping scheme employed and locality effect in a program are discussed. The mapping scheme employed has to exhibit a strong locality effect in order to allow efficient execution. We assess the average number of instructions in a cluster and the reduction in matching operations compared with fine grain control flow execution.
- Medium grain execution can benefit from a higher output bandwidth of a processor and finally, a simple superscalar processor with an issue rate of ten is sufficient to exploit the internal parallelism of a cluster. Although the technique does not exhaustively detect all possible errors, it detects nontrivial errors with a worst-case complexity quadratic to the system size. It can be automated and applied to systems with arbitrary loops and nondeterminism.

Alternatives to a Bad Talk

- Practice, Practice, Practice!
  - Use casette tape recorder to listen, practice
  - Try videotaping
  - Seek feedback from friends
- Use phrases, not sentences
  - Notes separate from slides (don't read slide)
- Pick appropriate font, size (~ 24 point to 32 point)
- Estimate talk length
  - - 2 minutes per slide
  - Use extras as backup slides (Question and Answer)
- Use color tastefully (graphs, emphasis)
- Don't cover slides
  - Use overlays or builds in powerpoint
- Go to room early to find out what is WRONG with setup
  - Beware: PC projection + dark rooms after meal!

Include in your final presentation

- Who is on team, and who did what
  - Everyone should say something
- High-level description of what you did and how you combined components together
  - Use block diagrams rather than detailed schematics
  - Assume audience knows Chapters 6 and 7 already
- Include novel aspects of design
  - Did you innovate? How?
  - Why did you choose to do things the way that you did?
- Give Critical Path and Clock cycle time
  - Bring paper copy of schematics in case there are detailed questions.
  - What could be done to improve clock cycle time?
- Description of testing philosophy!
- Mystery program statistics: instructions, clock cycles, CPI, why stalls occur (cache miss, load-use interlocks, branch mispredictions, ...)
- Lessons learned, what might do different next time

Review: Road to Faster Processors

- Time = Instr. Count x CPI x Clock cycle time
- How get a shorter Clock Cycle Time?
- Can we get CPI < 1?
- Can we reduce pipeline stalls for cache misses, hazards, … ?
- IA-32 P6 microarchitecture (μarchitecture): Pentium Pro, Pentium II, Pentium III
- IA-32 "Netburst" μarchitecture (Pentium 4, …
- IA-32 AMD Athlon, Opteron μarchitectures
- IA-64 Itanium I and II microarchitectures

Dynamic Scheduling in Pentium Pro, II, III

° P6 doesn't pipeline 80x86 instructions

° P6 decode unit translates the Intel instructions into 72-bit "micro-operations" (~ MIPS instructions)

° Takes 1 clock cycle to determine length of 80x86 instructions + 2 more to create the micro-operations

°Most instructions translate to 1 to 4 micro-operations

° Sends micro-operations to reorder buffer & reservation stations

Dynamic Scheduling in P6 (Pentium Pro, II, III)

° Complex 80x86 instructions are executed by a conventional microprogram (8K x 72 bits) that issues long sequences of micro-operations

° 10 stage pipeline for micro-operations

° 14 clocks in total pipeline

P6 Pipeline

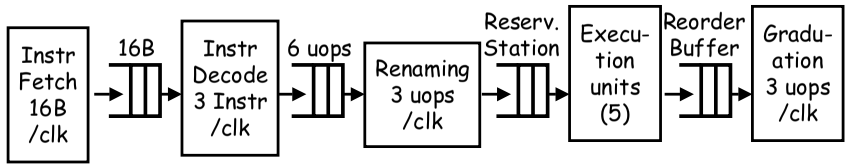
° 14 clocks in total (~3 state machines)

° 8 stages are used for in-order instruction fetch, decode, and issue

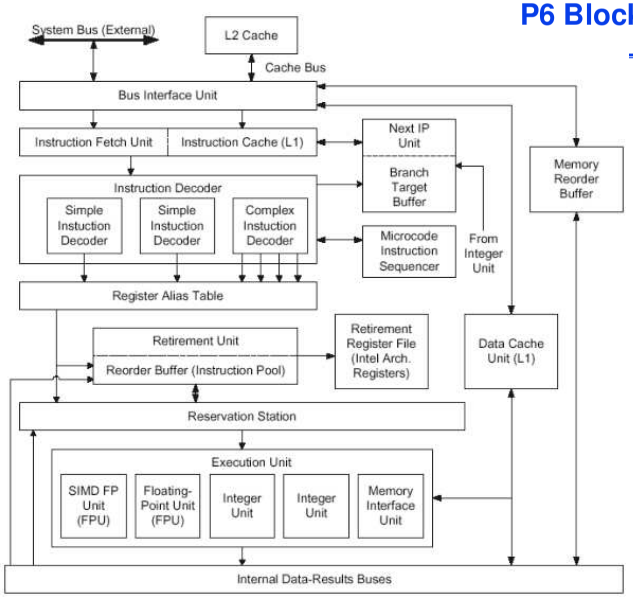
- Takes 1 clock cycle to determine length of 80x86 instructions + 2 more to create the micro-operations (uops)

° 3 stages are used for out-of-order execution in one of 5 separate functional units

° 3 stages are used for instruction commit



P6 Block Diagram



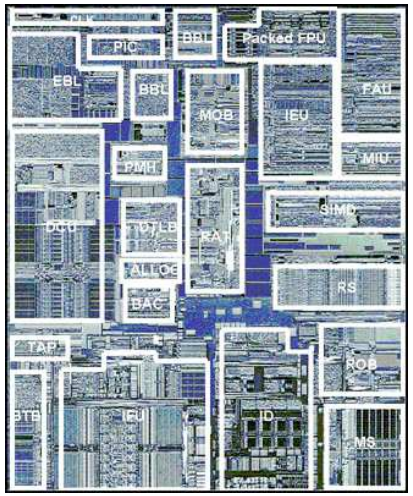
- IP = PC
- Simple Decoder:
  - 1 µop
- Complex Decoder:
  - 1-4 µops
- Rename: 40 regs

From: http://www.digit-life.com/articles/pentium4/

Dynamic Scheduling in P6

Parameter	80x86	microops
Max. instructions issued/clock	3	6
Max. instr. complete exec./clock		5
Max. instr. commited/clock		3
Window (Instrs in reorder buffer)		40
Number of reservations stations		20
Number of rename registers		40
No. integer functional units (FUs)	2	
No. floating point FUs	1	
No. SIMD Fl. Pt. FUs	1	
No. memory Fus	1 load + 1 store	

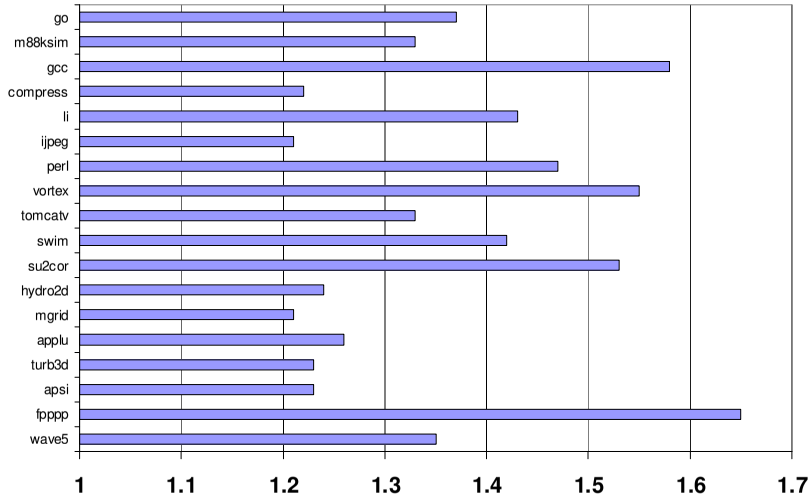
Pentium III Die Photo



1st Pentium III, Katmai: 9.5 M transistors, 128 mm**2 in 0.25-micron

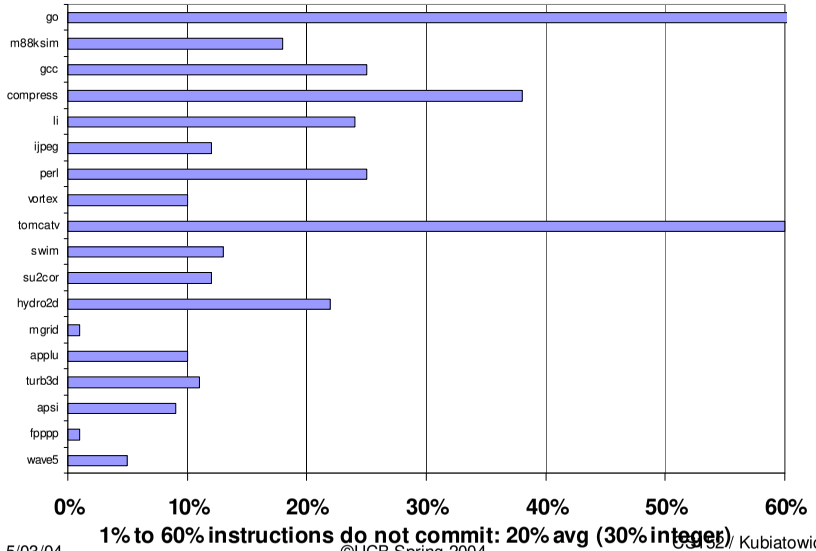
- EBL/BBL - Bus logic, Front, Back
- MOB - Memory Order Buffer
- Packed FPU - MMX Fl. Pt. (SSE)
- IEU - Integer Execution Unit
- FAU - Fl. Pt. Arithmetic Unit
- MIU - Memory Interface Unit
- DCU - Data Cache Unit
- PMH - Page Miss Handler
- DTLB - Data TLB
- BAC - Branch Address Calculator
- RAT - Register Alias Table
- SIMD - Packed Fl. Pt.
- RS - Reservation Station
- BTB - Branch Target Buffer
- IFU - Instruction Fetch Unit (+I$)
- ID - Instruction Decode
- ROB - Reorder Buffer
- MS - Micro-instruction Sequencer

P6 Performance: uops/x86 instr
200 MHz, 8KI$/8KD$/256KL2$, 66 MHz bus

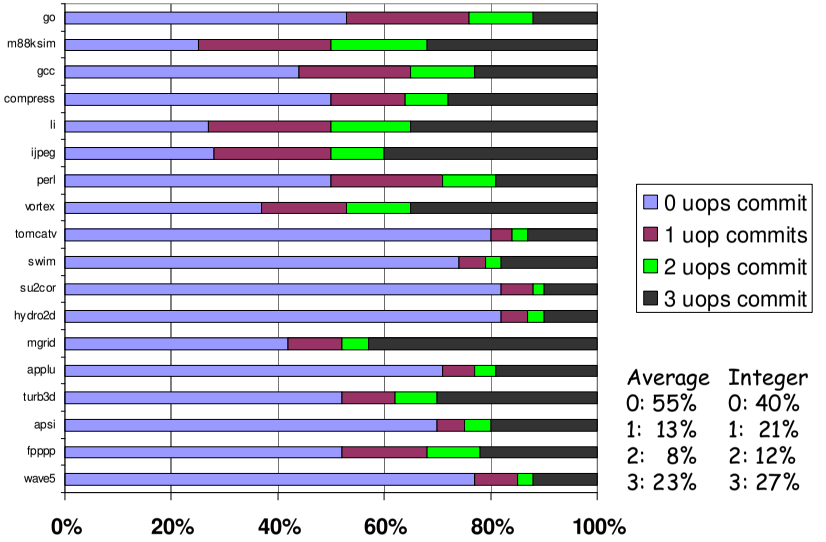


1.2 to 1.6 uops per IA-32 instruction: 1.36 avg. (1.37 integer)

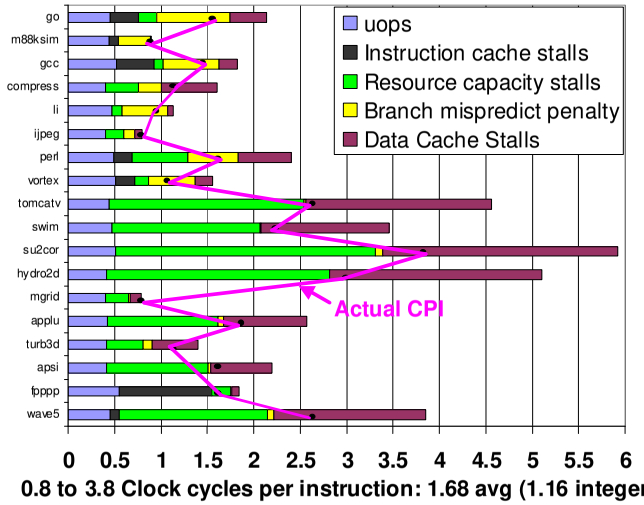
P6 Performance: Speculation rate (% instructions issued that do not commit)



P6 Performance: µops commit/clock



P6 Dynamic Benefit? Sum of parts CPI vs. Actual CPI



Ratio of sum of parts vs. actual CPI: 1.38X avg. (1.29X integer)

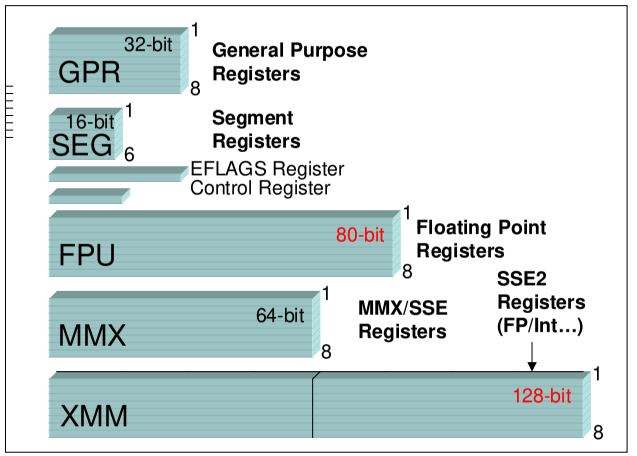
Pentium 4 features

- Called "NetBurst" Microarchitecture
  - Still translate from 80x86 to micro-ops
  - Instruction Cache (Execution Trace Cache)
  - Out-of-Order (OOO) execution engine
  - Double-pumped Arithmetic Logic Unit
  - Memory Subsystem (L1 access in 2 CP)
- Floating Point/Multi-Media performance
  - Multimedia instructions 128 bits wide vs. 64 bits wide for P6
    - ⇒ 144 new instructions
  - When used by programs??
  - Faster Floating Point: execute 2 64-bit Fl. Pt. Per clock
  - Memory FU: 1 128-bit load, 1 128-store /clock to MMX regs
- P4 has better branch predictor
  - BTB: 4096 vs 512 (Intel: 1/3 misprediction improvement)
- Instruction Cache holds micro-operations vs. 80x86 instructions
  - no decode stages of 80x86 on cache hit
  - called "trace cache" (TC)

Pentium 4 features (Continued)

- Faster memory bus: initially 400 MHz v. 133 MHz
- Caches
  - Pentium III: L1I 16KB, L1D 16KB, L2 256 KB
  - Pentium 4: L1I 12K µops, L1D 8 KB, L2 256 KB
  - Block size: PIII 32B v. P4 128B; 128 v. 256 bits/clock
- Initial P4 Clock rates:
  - Pentium III 1 GHz v. Pentium IV 1.5 GHz
  - 14 stage pipeline vs. 24 stage pipeline
- Using RAMBUS DRAM
  - Bandwidth faster, latency same as SDRAM
  - Later changed to support DDR SDRAM
- Rename registers: 128 vs 40 (although 8 for architectural state)

Registers



SIMD: Single Instruction Multiple Data

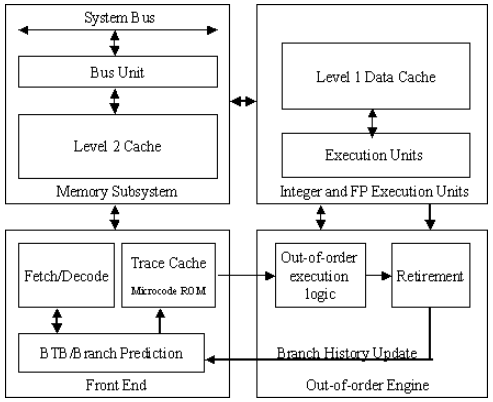
- Beginning with Pentium II, "SIMD" instructions added
- "Partitions" ALU to do multiple narrow data operations in 1 clock cycle by breaking carry chain:
  - 64 bits => 2 32-bit int ops OR 4 16-bit ops OR 8 8-bit ops
- SSE2 added in Pentium 4
  - 128 bits => 2 64-bit Fl. Pt. OR 4 32-bit Fl. Pt. OR ...

Instructions	Packed Data	Registers MXM 64-bit	Registers XMM 128bit	APPS	
MMX (57) Pentium II	INT B,W,Q	Yes	---	Imaging, MM, comm.	
SSE (70) Pentium III	SP Float	Yes	---	3-D geo/rendering video en/decode	
SSE2 (144) Pentium 4	INT, SP/DP Float	Yes	Yes	4-D graphics Scientific Comp	

Pentium 4 Cache

Level	Capacity	Assoc-iativity	Line Size (bytes)	Latency int/float (clocks)	Write Update Policy
First Data	8KB	4	64	2/9	write through
Trace Cache	12K µops	8	N/A	N/A	N/A
Second	256KB, 512KB	8	128 read 64 write	7/7	write back
Third	0, 512KB or 1MB or 2 MB	8	128 read 64 write	14/14	write back

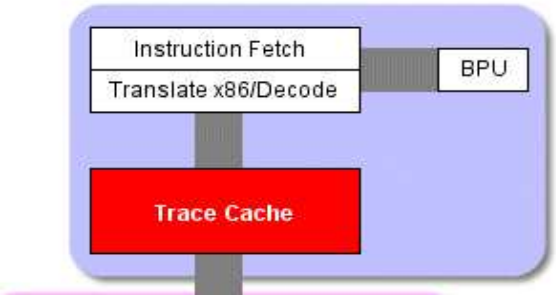
Pentium 4 basic block diagram



° Source: "The Microarchitecture of the Pentium 4 Processor"
- Intel Technology Journal Q1, 2001

Pentium 4 Trace Cache 1/4

- Trace Cache:
  - L1 instruction cache after the Instruction Fetch.
  - Arranges decoded instructions (µops) into some mini-programs that are ready to be used whenever there is a L1 Cache Hit.
  - The trace cache can send up to 3 µops directly to execution



Trace Cache Example

Example: Code in memory

```
      cmp
      br T1
      ...
T1:   sub
      br T2
      ...
T2:   mov
      sub
      br T3
      ...
T3:   add
      sub
      mov
      br T4
      ...
T4:   add
```

Decoded into 6 µop traces In Trace Cache

cmp	br T1	sub
br T2	mov	sub

Br T3	Add	Sub
Mov	Br T4	add

Pentium 4 Trace Cache 2/4

- What happens when there is a Trace Miss?
  - Trace Miss happens when L1 Cache misses, therefore, it needs to go to L2 cache, and fetch it from there. This results in 8 pipeline stages in order to translate and decode the instructions.
  - Trace cache operates in two modes :

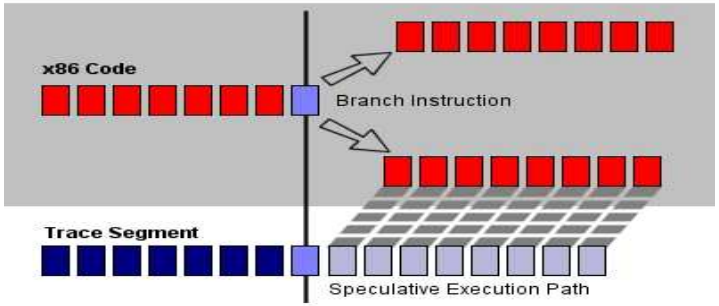
1) Execute mode : trace cache -> execution logic->executed. This is the mode Trace cache normally runs on when there is no Cache miss
2) Trace segment build mode: Happens when L1 cache miss. Fetch code from L2 cache, translate to µops, build trace segment, load segment to trace cache.

Pentium 4 Trace Cache 3/4

Trace cache applies Branch Prediction when building a trace.

It gets the code from the branch that it thinks the program will run on behind the code that it knows the program will take.

x86 code with branch: Trace cache build a trace from instructions up to including branch instruction, then pick a branch.



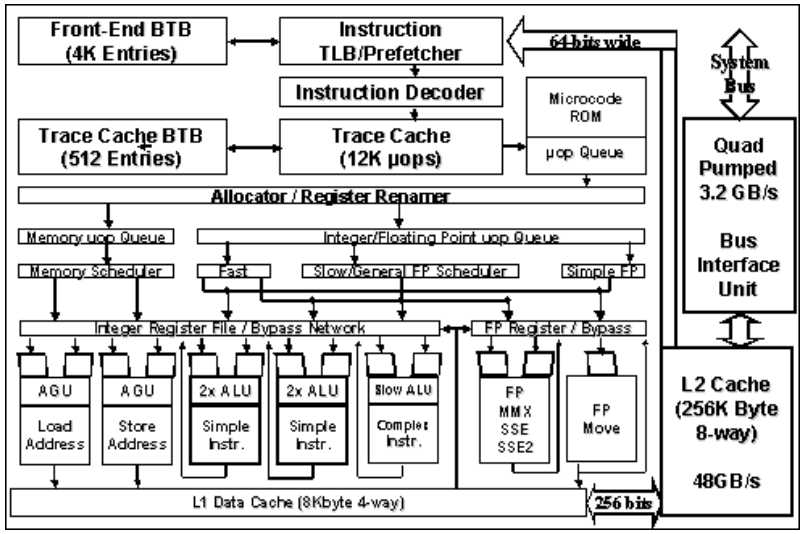
Pentium 4 Trace Cache 4/4

Conventional way:
Branch predictor figure outs branch to speculatively execute, then load a branch. takes up to 1 cycle of delay after every conditional branch instruction

With Trace cache:
the branch code is within the trace segment so there is no delay associated with bringing in the branch code.

- **Most x86 instructions decode into 2 or 3 µops**
- **Rare long instructions, which could decode into 100s of µops. PIII and P4 use microcode ROM which process these instructions so the regular decoder can do decoding on normal smaller instructions.**
- **Trace cache put a tag in trace segment when sees long instruction, Tag points to section of microcode ROM contains the µop sequence.**
- **When trace cache encounters the flag in execute mode, it lets microcode ROM stream proper sequence of µops into instruction stream for execution engine**

Full Block diagram (Intel)



Out-of-Order Execution -- Pipeline

Pentium III processor misprediction pipeline

1	2	3	4	5	6	7	8	9	10
Fetch	Fetch	Decode	Decode	Decode	Rename	ROB Rd	Rdy/Sch	Dispatch	Exec

1 2	3 4	5 6	7 8	9 10	11 12	13 14	15 16	17 18	19 20
TC Fetch	TC Fetch	DriveAlloc	Rename	Que Sch	Sch Sch	Disp Disp	FR FR	Ex Flgs	BrCkDrive

Pentium 4 processor misprediction pipeline

Comparison of two architectures

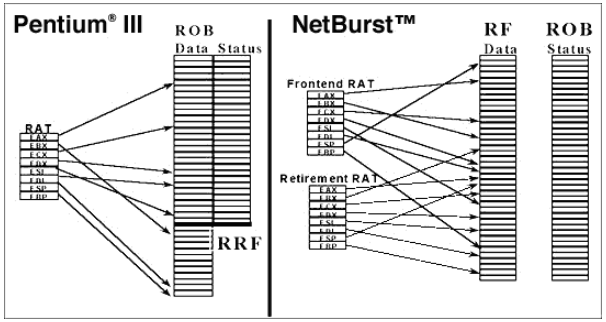
Pentium 4pipeline stages

Stage	Work
1	Trace Cache next instruction pointer
2	Trace Cache next instruction pointer
3	Trace Cache fetch
4	Trace Cache fetch
5	Drive (Wire latency)
6	Allocation
7	Rename
8	Rename
9	Queue
10	Schedule
11	Schedule
12	Schedule
13	Dispatch
14	Dispatch
15	Register Files
16	Register Files
17	Execute
18	Flags
19	Branch Check
20	Drive (Wire Latency)

Pentium 3 pipeline stages

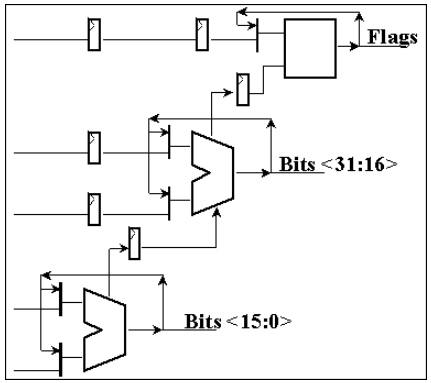
1	Fetch
2	Fetch
3	Decode
4	Decode
5	Decode
6	Rename
7	ROB Rd
8	Rdy/Sch
9	Dispatch
10	Exec

Register Renaming: Pentium III vs NetBurst



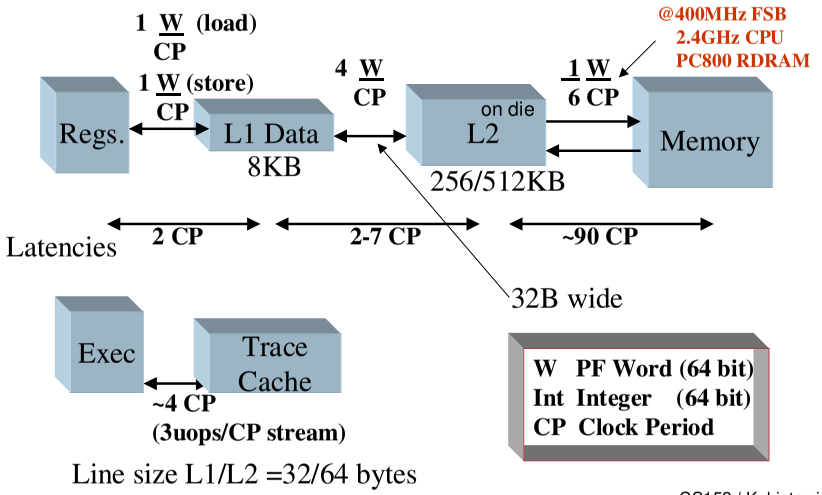
- Pentium III ties names of registers to reorder buffer slots (Implicit?)
  - Values copies to architectural register file (RRF) on commit
- Pentium IV performs explicit register renaming
  - 128 physical registers
  - Commit of translations

Staggered ALU Add



- Add pipeline can operate at 2X clock speed
- Split into 3 fast cycles with forwarding (16bits x 2 + flags)
- L1 data cache access starts with bottom 16 bits

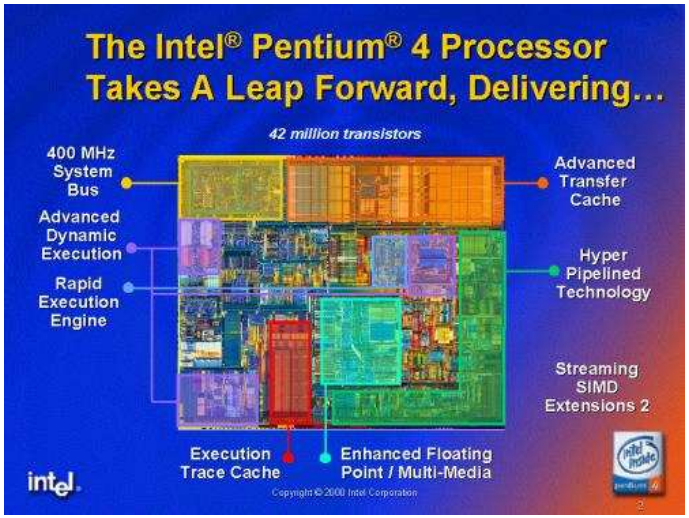
Pentium 4 Speeds & Feeds



Pentium 4 Basic Features

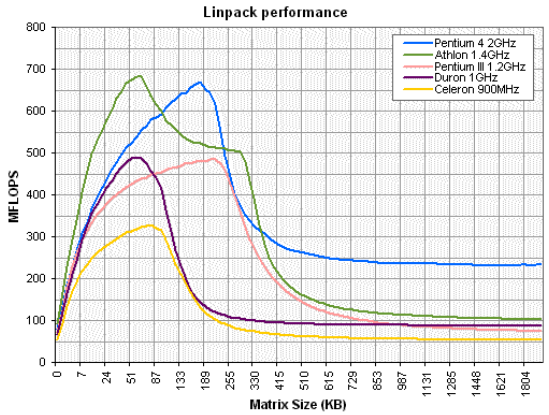
- 42 million transistors (256 KB L2 cache) 55 watts @1.5GHz, 217 mm**2 (0.18u)
- 55 million transistors (512 KB L2 cache)
  82 watts @ 3.0 GHz, 131 mm**2 (0.13u)
- Xeon (server): 160 million transistors
  (512 KB L2 cache + 2048 KB L3)
  65 watts @ 2.0 GHz, 211 mm**2 (0.13u)
- 400/533/800 MHz Front Side Bus
  - Bus to Memory Hub, which connects to DRAM, AGP graphics bus, and I/O Hub

Pentium-4 die floor plan



L1 Dcache
L2 cache

Performance Comparison



100 x 100 x 8 = 80 KB

Scott Wasson "Intel's Pentium 4 Processor, Radical Chic"
www.tech-report.com/reviews/2001q3/pentium4-2ghz/

SPEC 2000 Performance 3/2001 *Source: Microprocessor Report,*

Processor	Alpha 21264B	AMD Athlon	HP PA-8600	IBM Power 3-II	Intel PIII	Intel P4	MIPS R12000	Sun Ultra-II	Sun Ultra-III
System or Motherboard	Alpha ES40 Model 6	AMD GA-7ZM	HP9000 j6000	RS/6000 44P-170	Dell Prec. 420	Intel D850GB	SGI 2200	Sun Enterprs 450	Sun Blade 1000
Clock Rate	833MHz	1.2GHz	552MHz	450MHz	1GHz	1.5GHz	400MHz	480MHz	900MHz
External Cache	8MB	None	None	8MB	None	None	8MB	8MB	8MB
164.gzip	392	n/a	376	230	545	553	226	165	349
175.vpr	452	n/a	421	285	354	298	384	212	383
176.gcc	617	n/a	577	350	401	588	313	232	500
181.mcf	441	n/a	384	498	276	473	563	356	474
186.crafty	694	n/a	472	304	523	497	334	175	439
197.parser	360	n/a	361	171	362	472	283	211	412
252.eon	645	n/a	395	280	615	650	360	209	465
253.perlbmk	526	n/a	406	215	614	703	246	247	457
254.gap	365	n/a	229	256	443	708	204	171	300
255.vortex	673	n/a	764	312	717	735	294	304	581
256.bzip2	560	n/a	349	258	396	420	334	237	500
300.twolf	658	n/a	479	414	394	403	451	243	473
SPECint_base2000	518	n/a	417	286	454	524	320	225	438
168.wupside	529	360	340	360	416	759	280	284	497
171.swim	1,156	506	761	279	493	1,244	300	285	752
172.mgrid	580	272	462	319	274	558	231	226	377
173.applu	424	298	563	327	280	641	237	150	221
177.mesa	713	302	300	330	541	553	289	273	469
178.galgel	558	468	569	429	335	537	989	735	1,266
179.art	1,540	213	419	969	410	514	995	920	990
183.equake	231	236	347	560	249	739	222	149	211
187.facerec	822	411	258	257	307	451	411	459	718
188.ammp	488	221	376	326	294	366	373	313	421
189.lucas	731	237	370	284	349	764	259	205	204
191.fma3d	528	365	302	340	297	427	192	207	302
200.sixtrack	340	256	286	234	170	257	199	159	273
301.aspi	553	278	523	349	371	427	252	189	340
SPECfp_base2000	590	304	400	356	329	549	319	274	427

1.5X

1.2X

1.7X

Conclusion: Power

- Best way to say power or energy: do nothing!
- Most Important equations to remember:
  - Energy = CV^2
  - Power = CV^2f
- Slowing clock rate does not reduce energy for fixed operation!
- Ways of reducing energy:
  - Pipelining with reduced voltage
  - Parallelism with reduced voltage

Conclusion: Intel

- OOO processors
  - HW translation to RISC operations
  - Superpipelined P4 with 22-24 stages vs. 12 stage Opteron
  - Trace cache in P4
  - SSE2 increasing floating point performance