18-640 Foundations of Computer Architecture

Lecture 7:
"Dynamic Scheduling and Out-of-Order Execution"

John Paul Shen
September 16, 2014

- Required Reading Assignment:
  - **Chapter 5 of Shen and Lipasti (SnL).**
- Recommended Reference:
  - Robert Tomasulo, "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," IBM Journal of Research and Development, 11(1):25-33, January 1967.

Electrical & Computer ENGINEERING

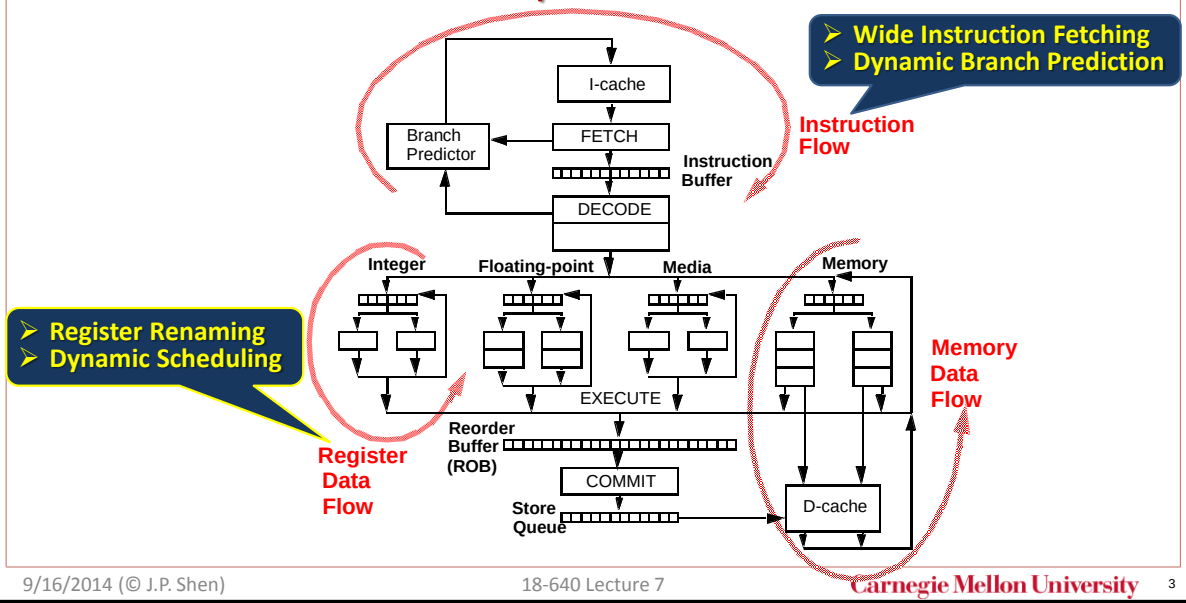
18-640 Foundations of Computer Architecture

Lecture 7:
"Dynamic Scheduling and Out-of-Order Execution"

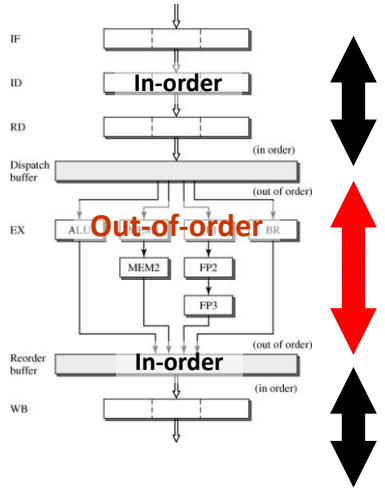
A. OOO Execution Implementation
   a. Instruction Window Implementation
   b. Reorder Buffer Implementation
B. Dynamic Instruction Scheduling
   a. Instruction Wake Up
   b. Instruction Select
   c. Result Forwarding
C. Intel Pentium Pro Case Study

Electrical & Computer ENGINEERING

Three Flow Paths of Superscalar Processors



Register Data Flow Techniques

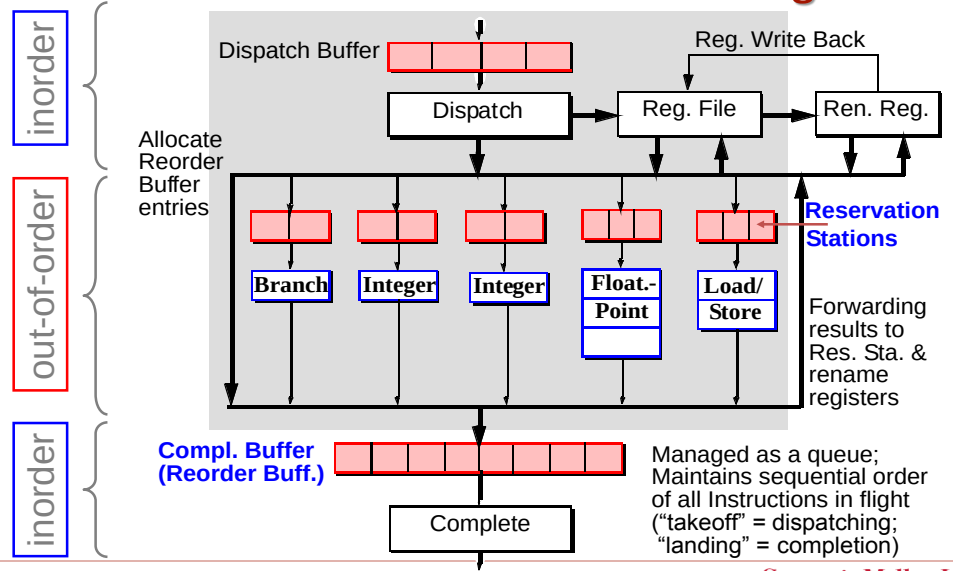


Out-of-Order Execution:

- Register Renaming
  - Address limitation of score-boarding:
    - One pending instruction per destination register
  - Eliminate WAR and WAW dependences without stalling
- Dynamic Scheduling
  - Track and resolve true RAW dependences
  - Scheduling HW: Instruction window, reservation stations, reorder buffer, etc…

A. Out-of-Order Execution Implementation

Elements of Modern Micro-Dataflow Engine



Steps in Dynamic OOO Execution (1)

- FETCH instruction (in-order, speculative)
  - I-cache access, predictions, insert in a fetch buffer
- DISPATCH (in-order, speculative)
  - Read operands from Register File (ARF) and/or Rename Register File (RRF)
    - RRF may return a ready value or a Tag for a physical location
  - Allocate new RRF entry (rename destination register) for destination
  - Allocate Reorder Buffer (ROB) entry
  - Advance instruction to appropriate entry in the scheduling hardware
    - Typical name for centralized: Issue Queue or Instruction Window
    - Typical name for distributed: Reservation Stations

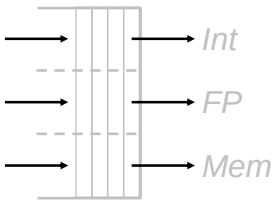
Steps in Dynamic OOO Execution (2)

- ISSUE & EXECUTE (out-of-order, speculative)
  - Scheduler entry monitors result bus for rename register Tag(s) for pending operand(s)
    - Find out if source operand becomes ready; if Tag(s) match, latch in operand(s)
  - When all operands ready, instruction is ready to be issued into FU (wake-up)
  - Issue instruction into FU, deallocate scheduler entry, no further stalling in FU pipe
    - Issuing is subject to structural hazards and scheduling priorities (select)
  - When execution finishes, broadcast result to waiting scheduler entries and RRF entry
- COMMIT/RETIRE/GRADUATE (in-order, non-speculative)
  - When ready to commit result into "in-order" (architectural) state (head of the ROB):
    - Update architectural register from RRF entry, deallocate RRF entry, and if it is a store instruction, advance it to Store Buffer
    - Deallocate ROB entry and instruction is considered architecturally completed
    - Update predictors based on instruction result

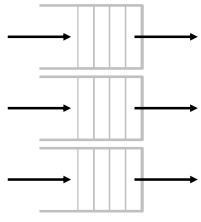
Instruction Window Implementation Alternatives

- Single vs. multiple buffers (trade-offs?)
  - Single centralized window
  - Single centralized window with static alignment for different FUs
  - Separate integer – FP – LSU windows
  - Separate buffers for each FU
    - Aka, Reservation Stations (a la Tomasulo algorithm)
- Management policies to keep in mind
  - Random access or FIFO
    - In-order vs out-of-order within each queue
  - Age-prioritized or criticality based
  - Value vs. Tag only
  - When to deallocate
    - Reservation Stations for Ld/St units are more complicated

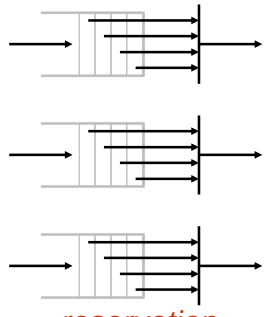
Instruction Window Organizations



statically aligned (inorder)

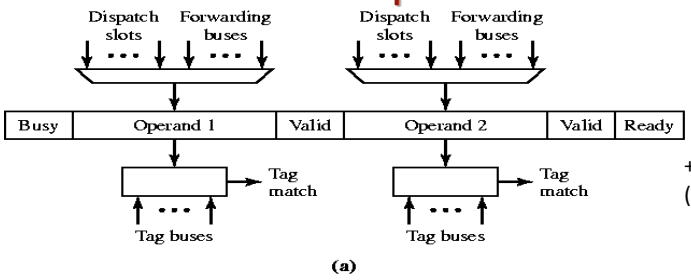


distributed queues

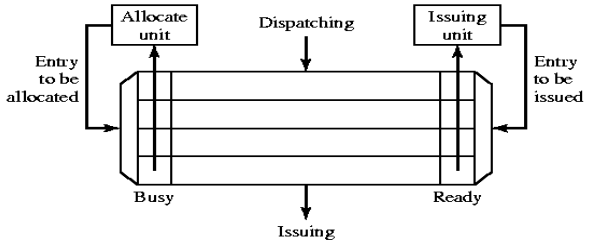


reservation stations

Reservation Station Implementation



(a)

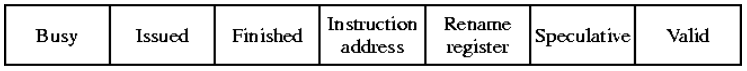


(b)

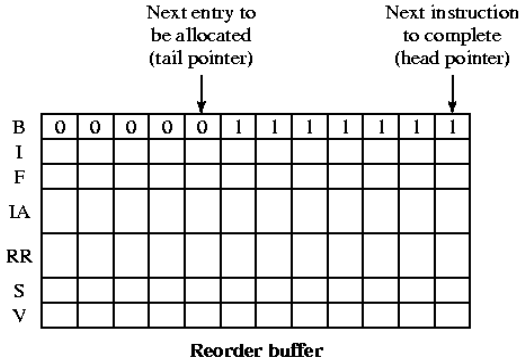
+ info for executing instruction (opcode, ROB entry, RRF entry…)

- Reservation Stations: distributed vs. centralized
  - Wakeup: benefit to partition across data types
  - Select: much easier with partitioned scheme
    - Select 1 of n/4 vs. 4 of n

Reorder Buffer Implementation



(a)

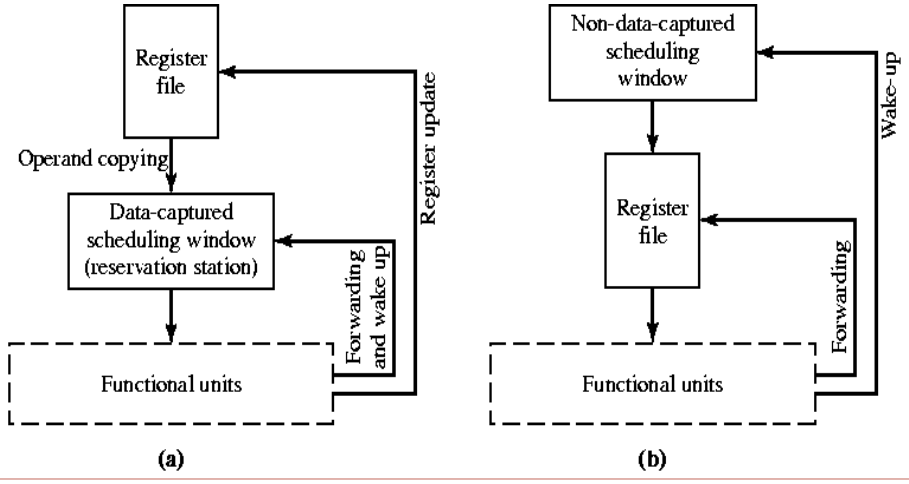


(b)

- Reorder Buffer
  - "Bookkeeping"
  - Can be instruction-grained, or block-grained (4-5 ops)

Dynamic Instruction Scheduler Options

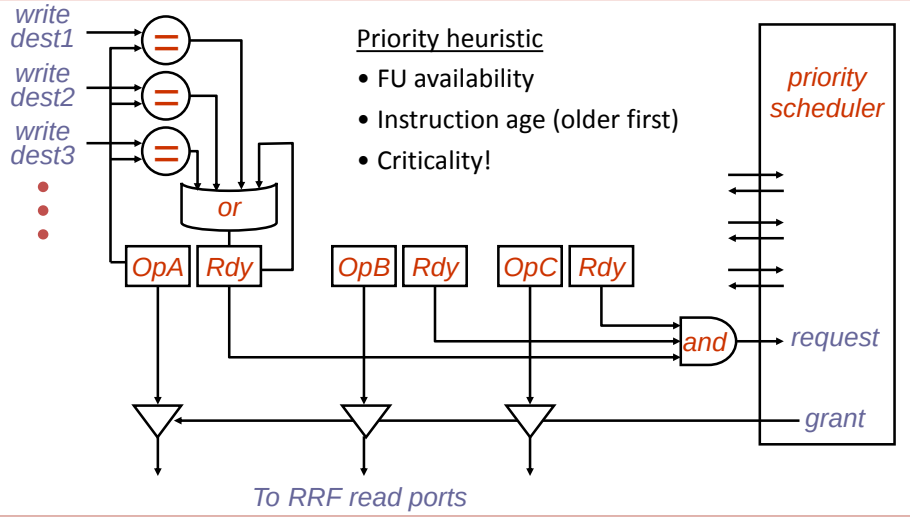
With Data Capture vs. Without Data Capture



B. Dynamic Instruction Scheduling

Electrical & Computer ENGINEERING

Instruction Window Scheduling



Priority heuristic

- FU availability
- Instruction age (older first)
- Criticality!

Priority/Select Logic

- Tree of arbiters that works in 2 phases
- First phase
  - Request signals are propagated up the tree. Only ready instructions send requests
  - This in turn raises the ready signal of its parent arbiter cell. At the root cell one or more of the input request signals will be high if there are one or more instructions that are ready.
  - The root cell grants the functional unit to one of its children by raising one of its grant outputs.
- Second phase
  - Grant signal is propagated down the tree to the instruction that is selected
  - The enable signal to the root cell is high whenever the functional unit is ready to execute an instruction.

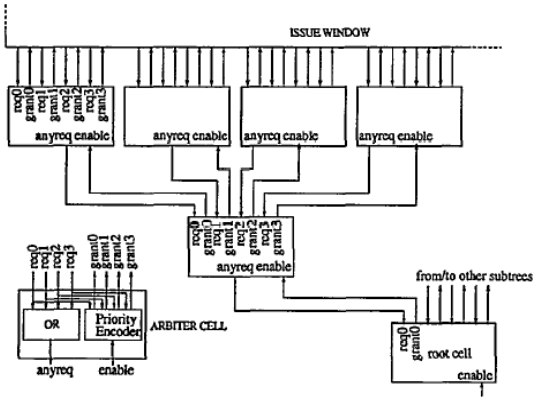


Figure 7: Selection logic.

Priority/Select Logic Issues

- Selection is easier if priority depends on instruction location
  - Older instructions are at the bottom of window and receive priority
- This creates an issue of compacting/collapsing
  - As instructions depart, compress remaining towards the bottom
  - Younger instructions will be inserted towards the top (lower priority)
- Compacting the window can be quite complex
  - Its complexity can affect performance (clock frequency)
  - Often implemented in some restricted form
    - E.g. split window into multiple groups, allow compaction of groups
    - Trade-off between window utilization and compaction simplicity

Wake-up Latency

- Assume a result becomes available in cycle i
  - When can you start executing an instruction that is waiting for it?
- Ideal solution: in cycle i+1
  - Back to back executing, just like with 5-stage pipeline
  - Requirement: the following have to work in one cycle
    - Distribute result tag to the window and detect that instruction becomes ready
    - Select instruction for execution and forward its info/operands to FU
  - May stress clock cycle in wide processors
- Alternative: split wake-up and select in separate cycles
  - Simpler hardware, faster clock cycle
  - Lower IPC (dependencies cost one extra cycle)

Result Forwarding

- Common data bus: used to broadcast results of FUs
- Broadcast destinations
  - ARF or RRF or ROB, depending on the renaming scheme
  - Instruction window
    - May need result or tag for the result
- Number of CDBs
  - Best case, 1 per functional unit
  - Can have less, but now we may have structural hazard
- Notes:
  - CDBs can be slow as the routing goes across large chip area
  - Broadcast tag early

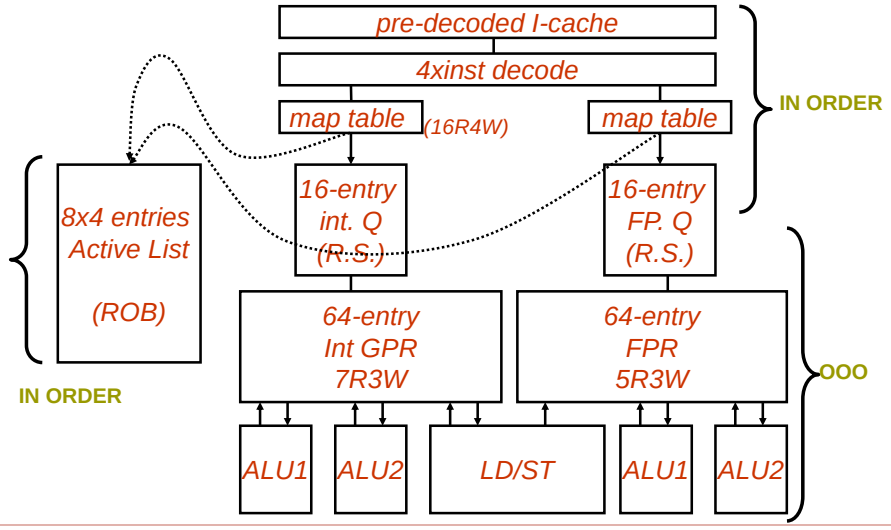
Dynamic Scheduling Implementation Cost

- To support N-way dispatch into IW per cycle
  - Nx2 simultaneous lookups into the rename map (or associative search)
  - N simultaneous write ports into the IW and the ROB
- To support N-way issue per cycle (assuming read at issue)
  - 1 prioritized associative lookup of N entries
  - N read ports into the IW
  - Nx2 read ports into the RF
- To support N-way complete per cycle
  - N write ports into the RF and the ROB
  - Nx2 associative lookup and write in IW
- To support N-way retire per cycle
  - N read ports in the ROB
  - N ports into the RF (potentially)

Example: MIPS R10000 circa 1996

- 4-way superscalar
  - fetch, decode, dispatch, and complete
- 5 execution pipelines
  - 2 Int, FP Add, FP Mult, Ld/St
- Micro-dataflow instruction scheduling
  - 16+16 instruction window
- Register renaming + memory renaming
  - 64 physical integer registers to hold 33 logical registers + renamed registers
- Speculative execution pass 4 unresolved branches
- Precise Interrupts

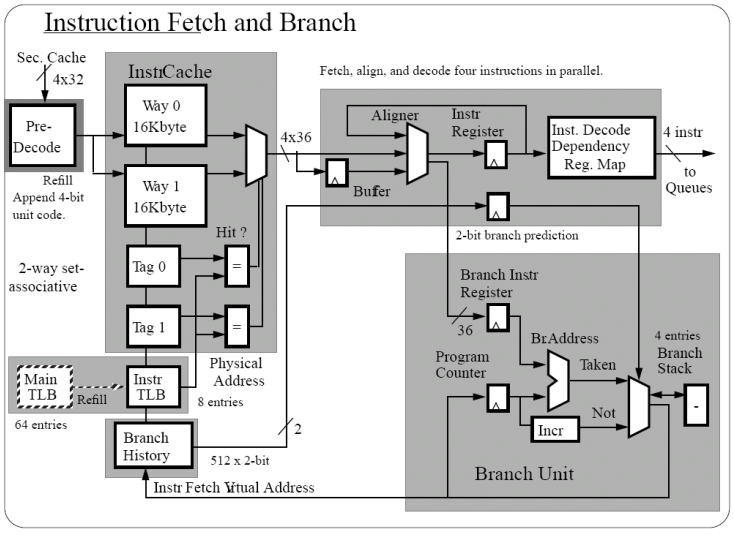
Dynamic Scheduling in the OOO MIPS R10000



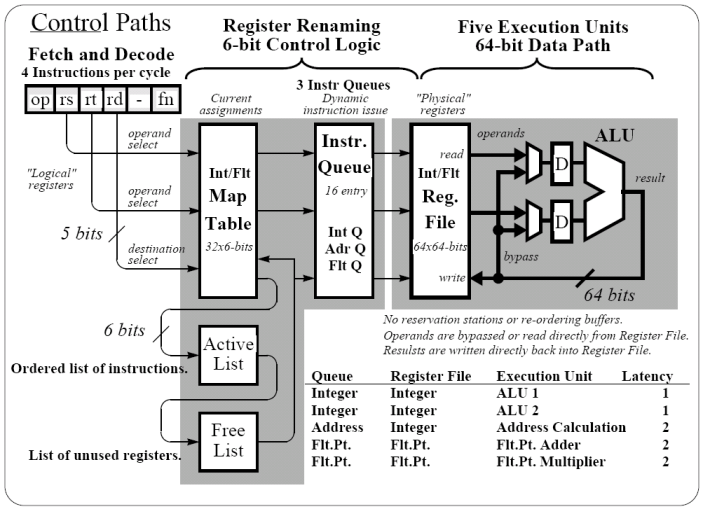
Design Choices

- Register Renaming
  - Map table lookup + dependency check on simultaneous dispatches
  - Unified physical register file
  - 4-deep branch stack to backup the map table on branch predictions
  - Sequential (4-at-a-time) back-tracking to recover from exceptions
- Instruction Queues
  - Separate 16-entry floating point and integer instruction queues
  - Prioritized, dataflow-ordered scheduling
- Reorder Buffer
  - One per outstanding instruction, FIFO ordered
  - Stores PC, logical destination number, old physical destination number
    Why not current physical destination number?

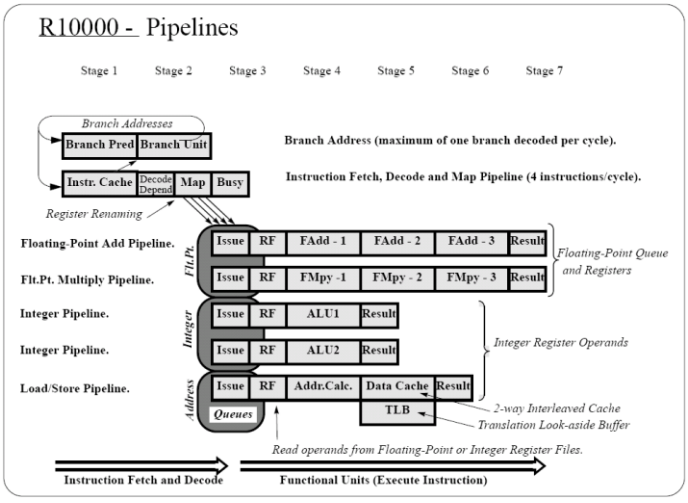
R10000 Instruction Fetch and Branch



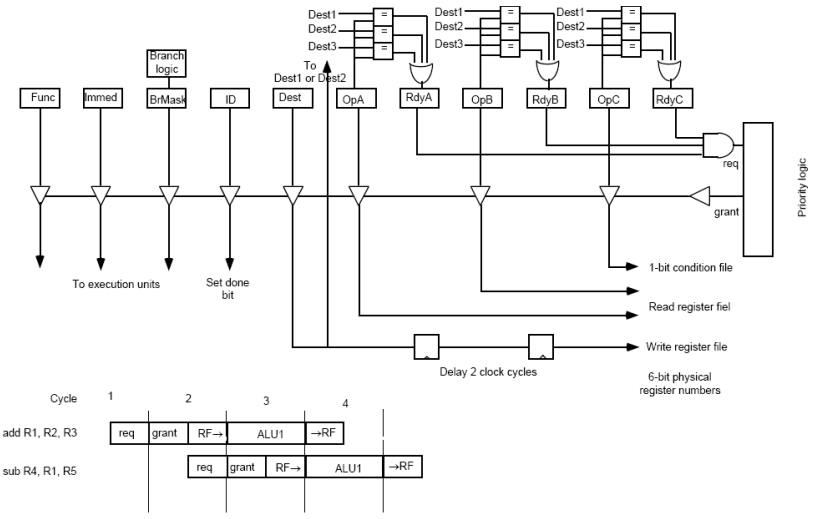
R10000 Register Renaming



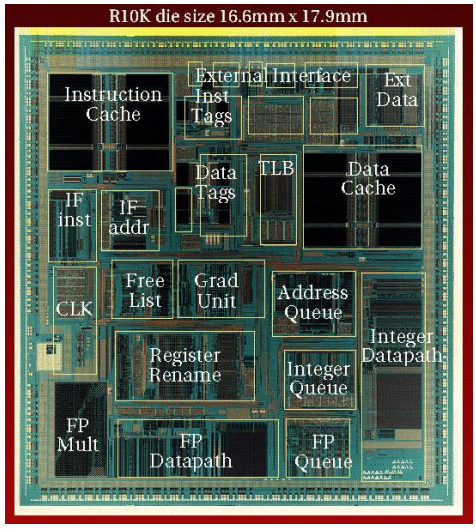
R10000 Pipelines



R10000 Integer Queue



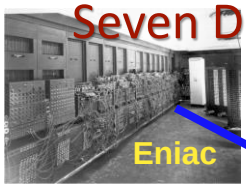
MIPS R10000 Die Photo



C. Intel Pentium Pro (P6) Case Study

Electrical & Computer ENGINEERING

Seven Decades of Modern Computing . . .



Mainframes

Minicomputers

Workstations

Personal Computers

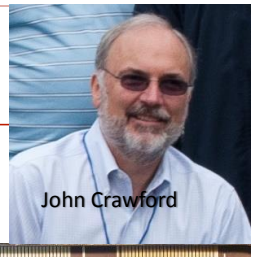
Laptop Computers

Mobile Computers

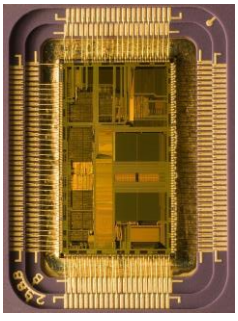
When Workstations Roamed on the Earth…

- Early Days (1970's)
  - Xerox Alto (1973)
  - Three Rivers PERQ (1979)
- The "3M Computer" (1980's)
  - Megabyte of memory, Megapixel display, MegaFLOPS performance, (MegaPenny cost)
  - 32-bit microprocessors, Unix OS, RISC microprocessors, GUI, WYSIWYG, laser printers
  - Apollo Computer, Sun Microsystems (SPARC), HP (HP-UX), SGI, DEC, IBM, Next, DELL
- Lucrative Market (1990's)
  - High margin computers ($10K-$50K)
  - Co-existed with low margin PC's
- Gradually Became Extinct (2000's)

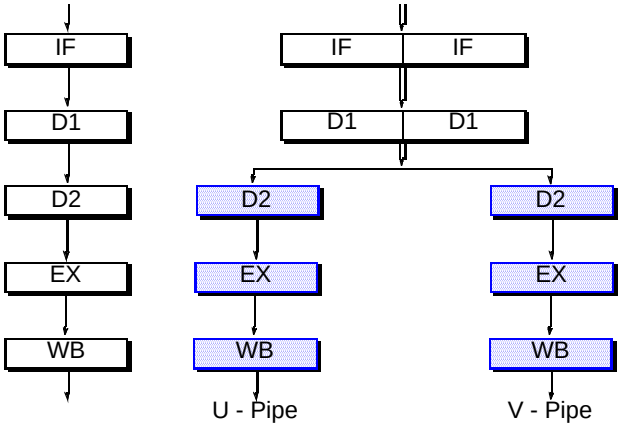
Intel 80486 and Pentium (P5) Processors

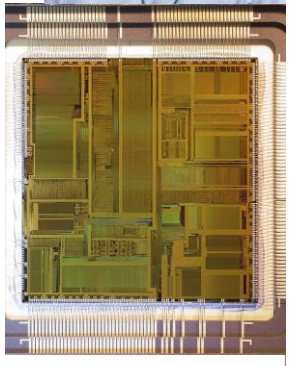


John Crawford



Introduced in 1989

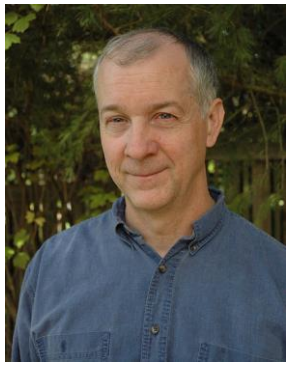




Introduced in 1993

Intel Pentium Pro (P6) Case Study

- **Microarchitecture**
  - Order-3 Superscalar
  - Out-of-Order execution
  - Speculative execution
  - In-order completion
- **Design Methodology**
- **Performance Analysis**
- **Retrospective**



Introduced in 1995

Electrical & Computer ENGINEERING

Goals of P6 Microarchitecture

IA-32 Compliant

Performance (Frequency - IPC)

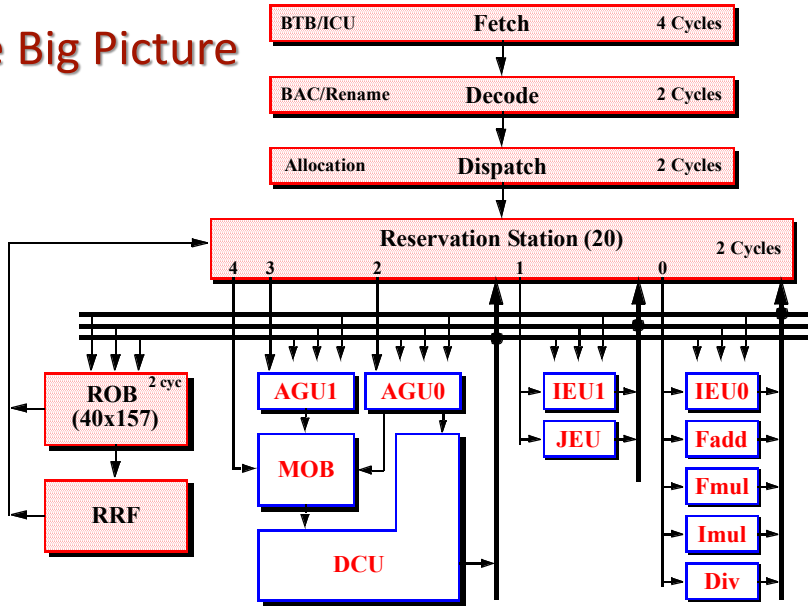
Validation

Die Size

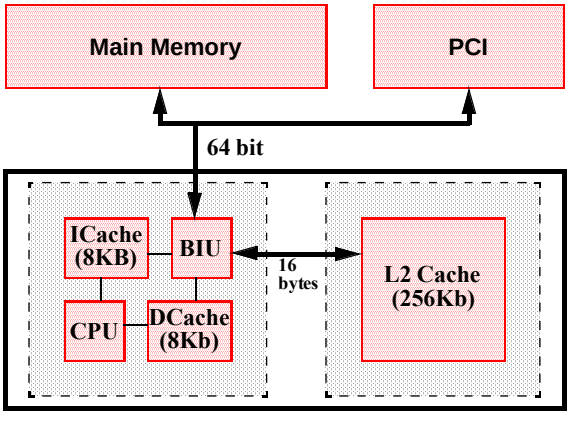
Schedule

Power

P6 – The Big Picture

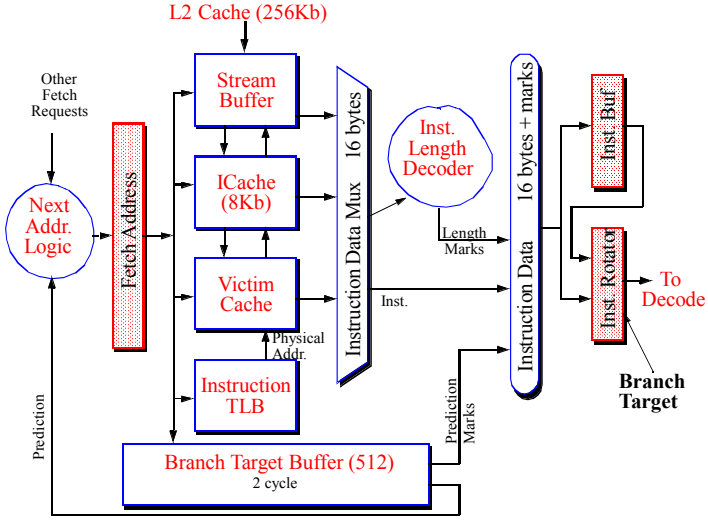


Memory Hierarchy

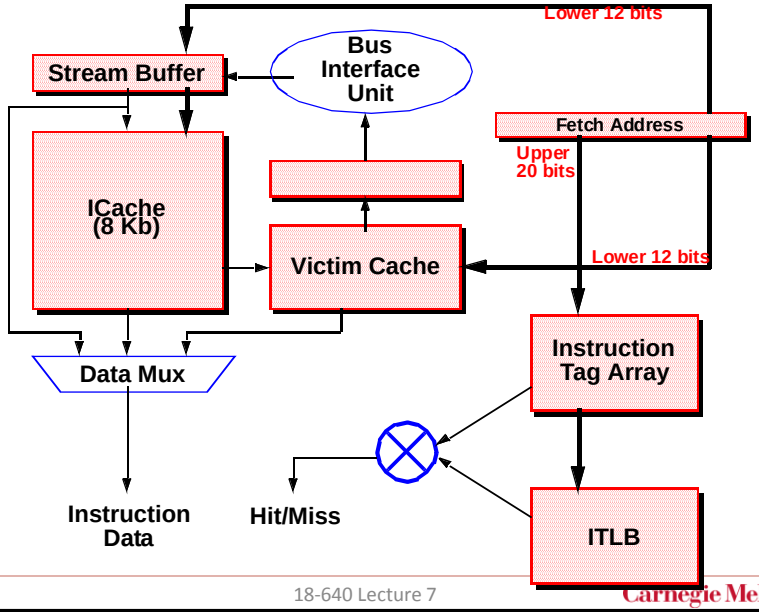


- Level 1 instruction and data caches - 2 cycle access time
- Level 2 unified cache - 6 cycle access time
- Separate level 2 cache and memory address/data bus

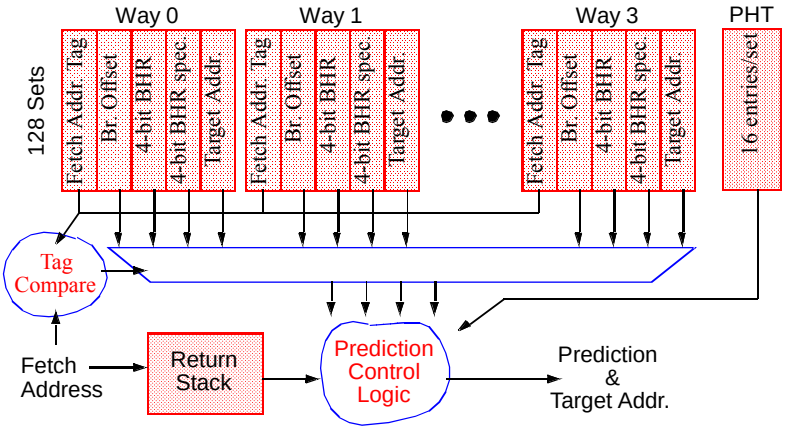
Instruction Fetch



Instruction Cache Unit

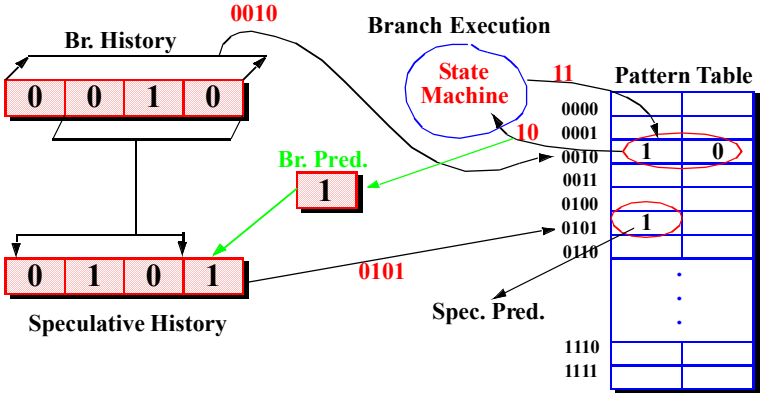


Branch Target Buffer



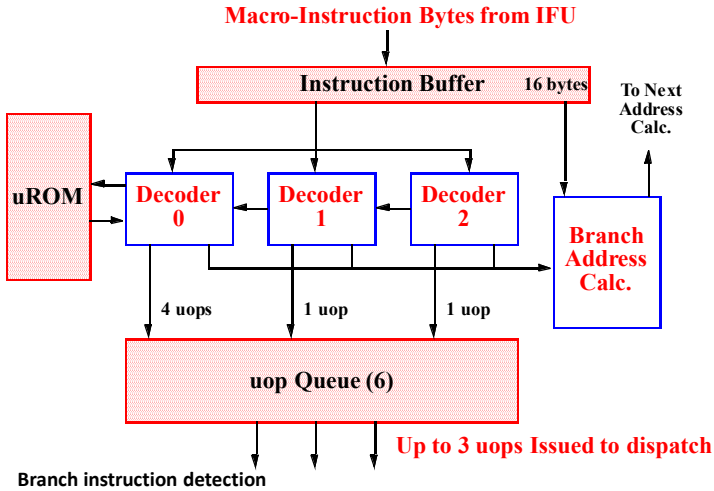
- Pattern History Table (PHT) is not speculatively updated
- A speculative Branch History Register (BHR) and prediction state is maintained
- Uses speculative prediction state if it exist for that branch

Branch Prediction Algorithm



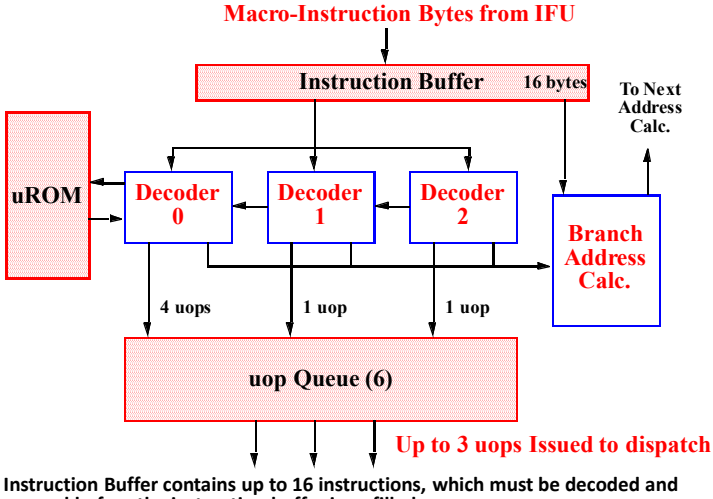
- Current prediction updates the speculative history prior to the next instance of the branch instruction
- Branch History Register (BHR) is updated during branch execution
- Branch recovery flushes front-end and drains the execution core
- Branch mis-prediction resets the speculative branch history state to match BHR

Instruction Decode - 1



- Branch instruction detection
- Branch address calculation - Static prediction and branch always execution
- One branch decode per cycle (break on branch)

Instruction Decode - 2



- Instruction Buffer contains up to 16 instructions, which must be decoded and queued before the instruction buffer is re-filled
- Macro-instructions must shift from decoder 2 to decoder 1 to decoder 0

What is a uop?

Small two-operand instruction - Very RISC like.

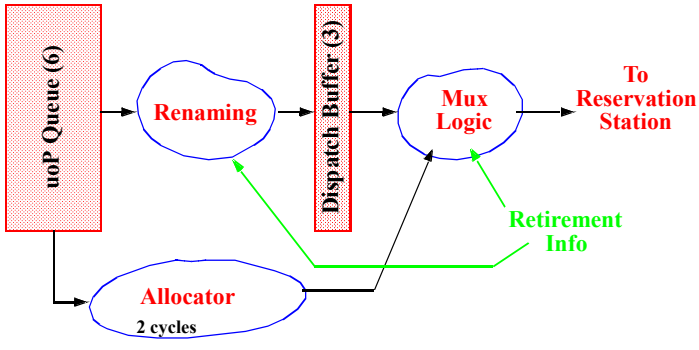
IA-32 instruction

add (eax),(ebx) MEM(eax) <- MEM(eax) + MEM(ebx)

Uop decomposition:

ld guop0,	(eax)	guop0 <- MEM(eax)
ld guop1,	(ebx)	guop1 <- MEM(ebx)
add guop0,guop1		guop0 <- guop0 + guop1
sta eax		
std guop0		MEM(eax) <- guop0

Instruction Dispatch



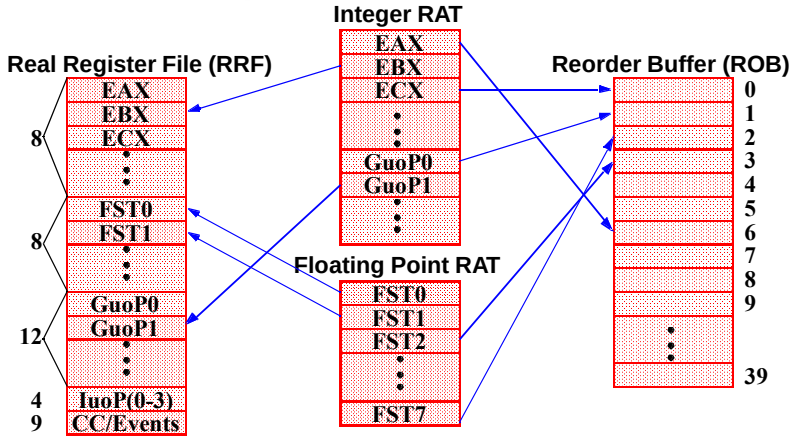
Register Renaming

Allocation requirements

- "3-or-none" Reorder buffer entries
- Reservation station entry
- Load buffer or store buffer entry

Dispatch buffer *"probably"* dispatches all 3 uops before re-fill

Register Renaming - 1

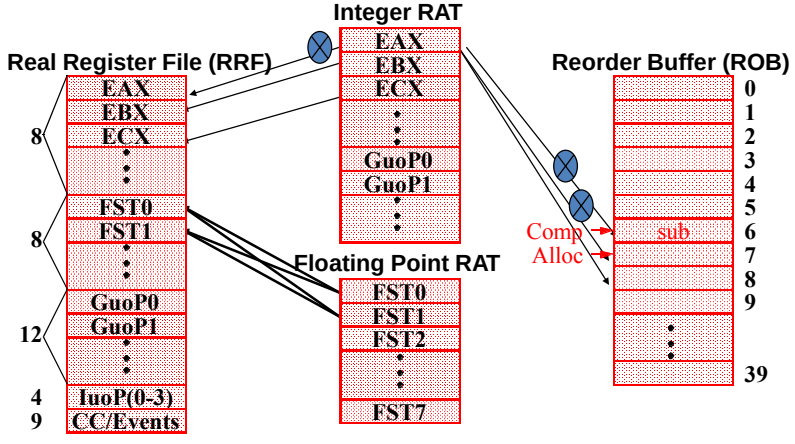


Similar to Tomasulo's Algorithm - Uses ROB entry number as tags

The register alias tables (RAT) maintain a pointer to the most recent data for the renamed register

Execution results are stored in the ROB

Register Renaming - Example



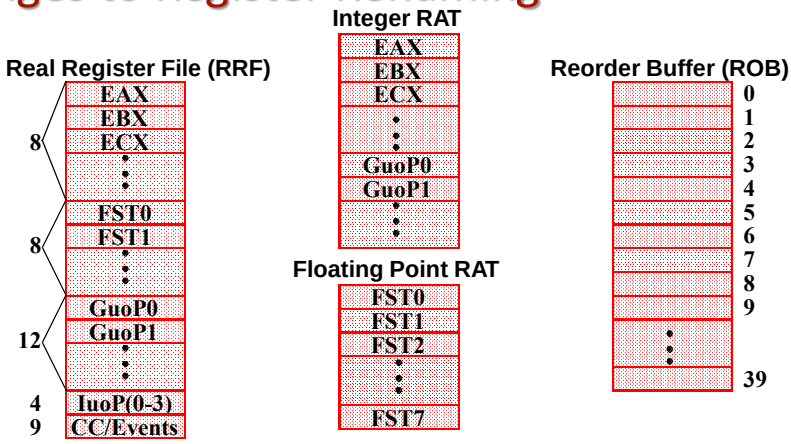
Dispatching:

```
add     eax, ebx
add     eax, ecx
fxch    f0, f1
```

Completing:

```
sub     eax, ecx
```

Challenges to Register Renaming

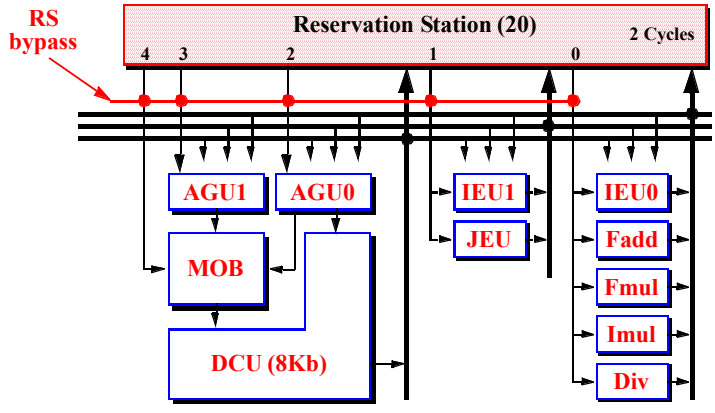


8-bit code

```
mov   AL, #data1
mov   AH, #data2
add   AL, #data3
add   AL, #data4
```

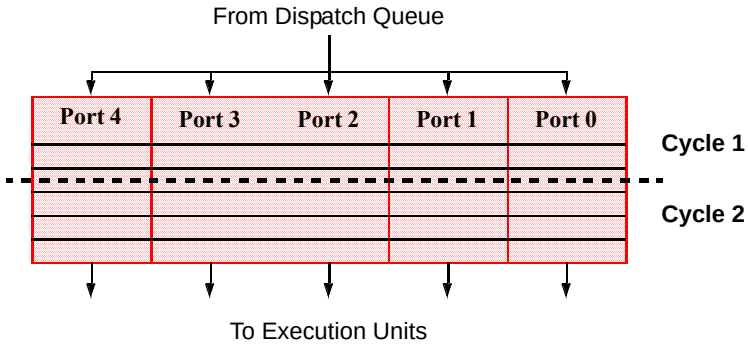
Byte addressable registers

Out-of-Order Execution Engine



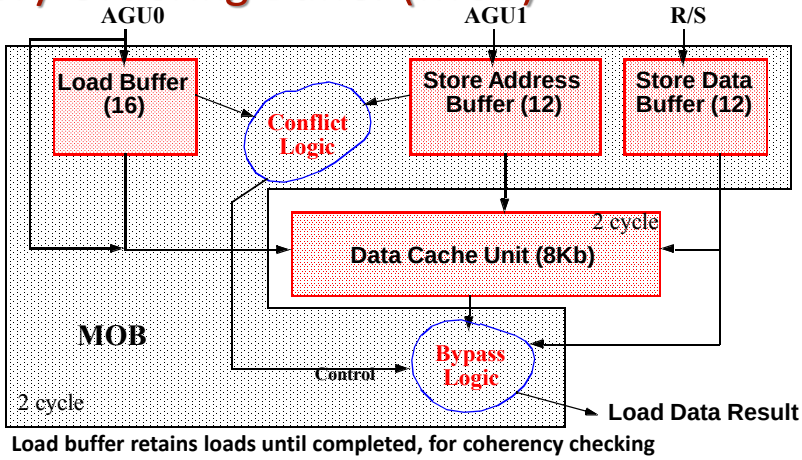
- In-order branch issue and execution
- In-order load/store issue to address generation units
- Instruction execution and result bus scheduling
- Is the reservation station *"truly"* centralized & what is *"binding"*?

Reservation Station



- Cycle 1
  - Order checking
  - Operand availability
- Cycle 2
  - Writeback bus scheduling

Memory Ordering Buffer (MOB)

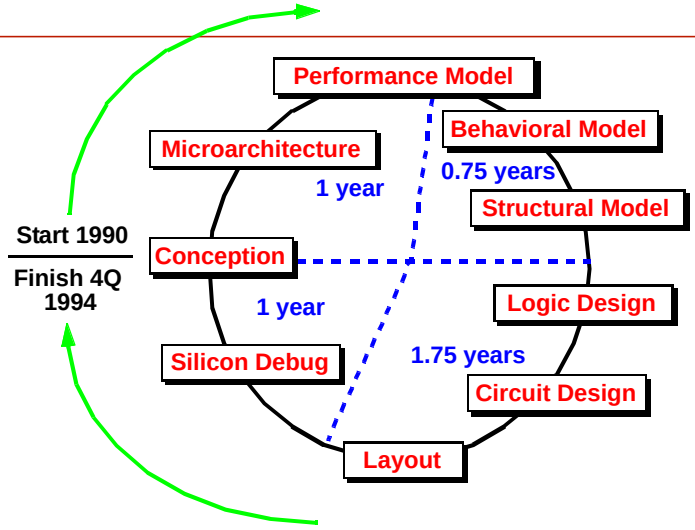


- **Load buffer retains loads until completed, for coherency checking**
- **Store forwarding out of store buffers**
- **2 cycle latency through MOB**
- **"Store Coloring" - Load instructions are tagged by the last store**

Instruction Completion

- Handles all exception/interrupt/trap conditions
- Handles branch recovery
  - OOO core drains out right-path instructions, commits to RRF
  - In parallel, front end starts fetching from target/fall-through
  - However, no renaming is allowed until OOO core is drained
  - After draining is done, RAT is reset to point to RRF
  - Avoids checkpointing RAT, recovering to intermediate RAT state
- Commits execution results to the architectural state in-order
  - Retirement Register File (RRF)
  - Must handle hazards to RRF (writes/reads in same cycle)
  - Must handle hazards to RAT (writes/reads in same cycle)
- "Atomic" IA-32 instruction completion
  - uops are marked as 1st or last in sequence
  - exception/interrupt/trap boundary
- 2 cycle retirement

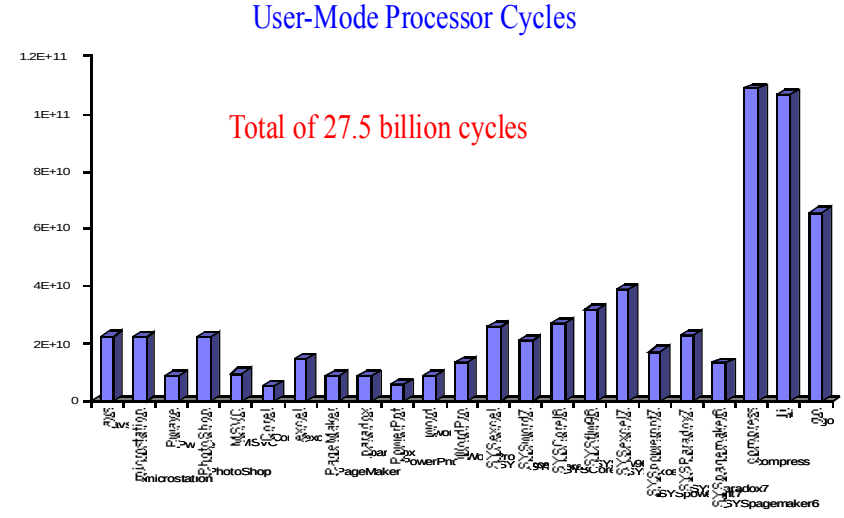
Pentium Pro Design Methodology - 1



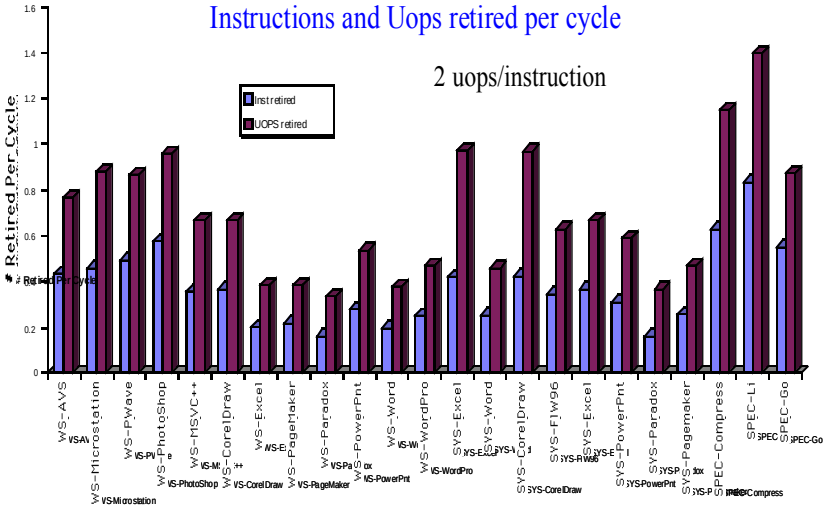
Pentium Pro Performance Analysis

- **Observability**
  - On-chip event counters
  - Dynamic analysis
- **Benchmark Suite**
  - BAPco Sysmark32 - 32-bit Windows NT applications
  - Winstone97 - 32-bit Windows NT applications
  - Some SPEC95 benchmarks

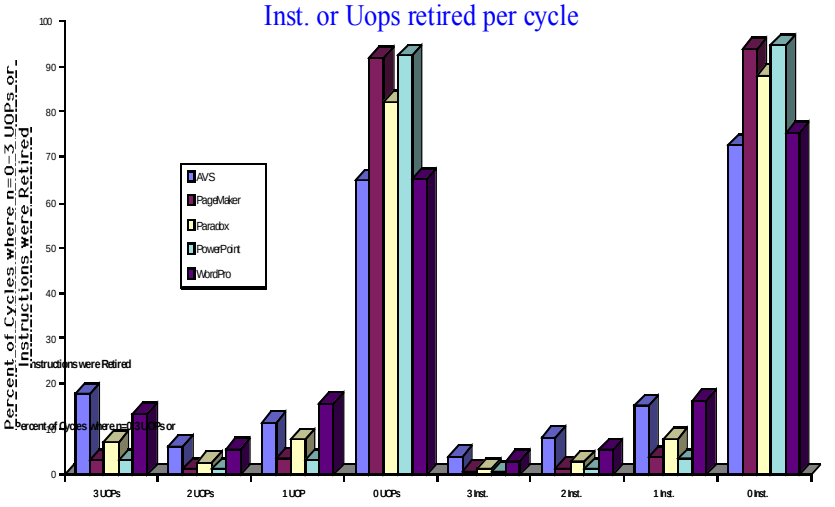
Performance – Run Times



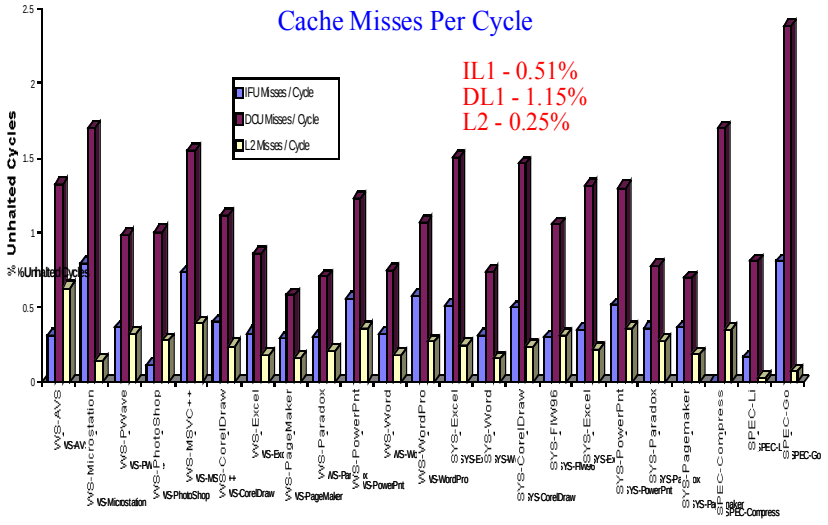
Performance – IPC vs. uPC



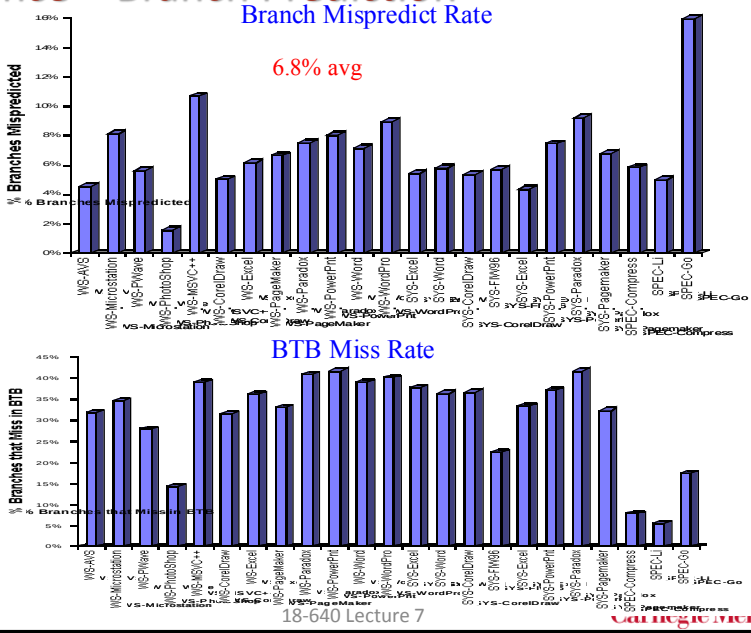
Performance – IPC vs. uPC



Performance – Cache Misses



Performance – Branch Prediction



Retrospective

- Most commercially successful microarchitecture in history
- Evolution
  - Pentium II/III, Xeon, etc.
    - Derivatives with on-chip L2, ISA extensions, etc.
  - Replaced by Pentium 4 as flagship in 2001
    - High frequency, deep pipeline, extreme speculation
  - Resurfaced as Pentium M in 2003
    - Initially a response to Transmeta in laptop market
    - Pentium 4 derivative (90nm Prescott) delayed, slow, hot
  - Core Duo, Core 2 Duo, Core i7 replaced Pentium 4

Microarchitectural Updates

- Pentium M (Banias), Core Duo (Yonah)
  - Micro-op fusion (also in AMD K7/K8)
    - Multiple uops in one: (add eax,[mem] => ld/alu), sta/std
    - These uops decode/dispatch/commit once, issue twice
  - Better branch prediction
    - Loop count predictor
    - Indirect branch predictor
  - Slightly deeper pipeline (12 stages)
    - Extra decode stage for micro-op fusion
    - Extra stage between issue and execute (for RS/PLRAM read)
  - Data-capture reservation station (payload RAM)
    - Clock gated for 32 (int) , 64 (fp), and 128 (SSE) operands

Microarchitectural Updates

- Core 2 Duo (Merom)
  - 64-bit ISA from AMD K8
  - Macro-op fusion
    - Merge uops from two x86 ops
    - E.g. cmp, jne => cmpjne
  - 4-wide decoder (Complex + 3x Simple)
    - Peak x86 decode throughput is 5 due to macro-op fusion
  - Loop buffer
    - Loops that fit in 18-entry instruction queue avoid fetch/decode overhead
  - Even deeper pipeline (14 stages)
  - Larger reservation station (32), instruction window (96)
  - Memory dependence prediction

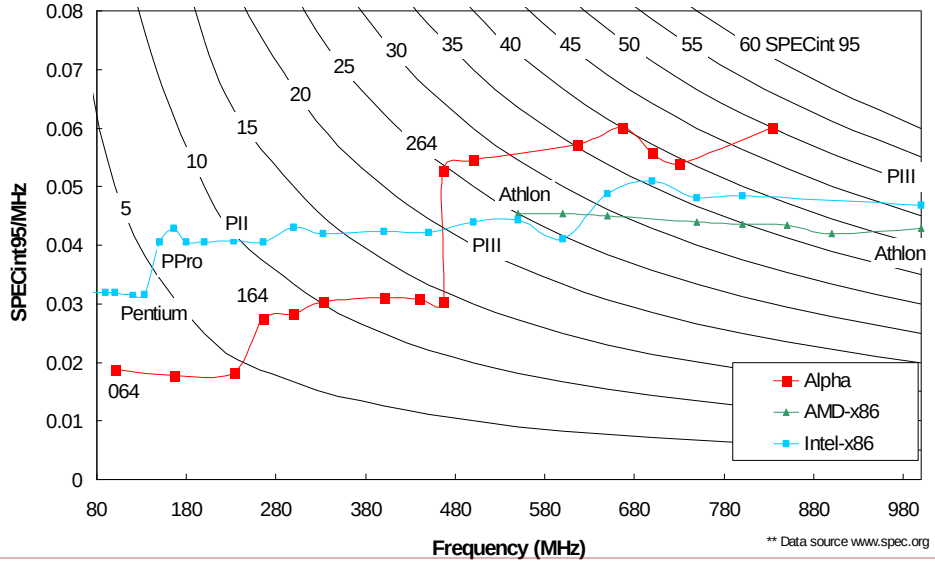
Microarchitectural Updates

- Nehalem (Core i7/i5/i3)
  - RS size 36, ROB 128
  - Loop cache up to 28 uops
  - L2 branch predictor
  - L2 TLB
  - I$ and D$ now 32K, L2 back to 256K, inclusive L3 up to 8M
  - Simultaneous multithreading
  - RAS now renamed (repaired)
  - 6 issue, 48 load buffers, 32 store buffers
  - New system interface (QPI) – finally dropped front-side bus
  - Integrated memory controller (up to 3 channels)
  - New STTNI instructions for string/text handling

Microarchitectural Updates

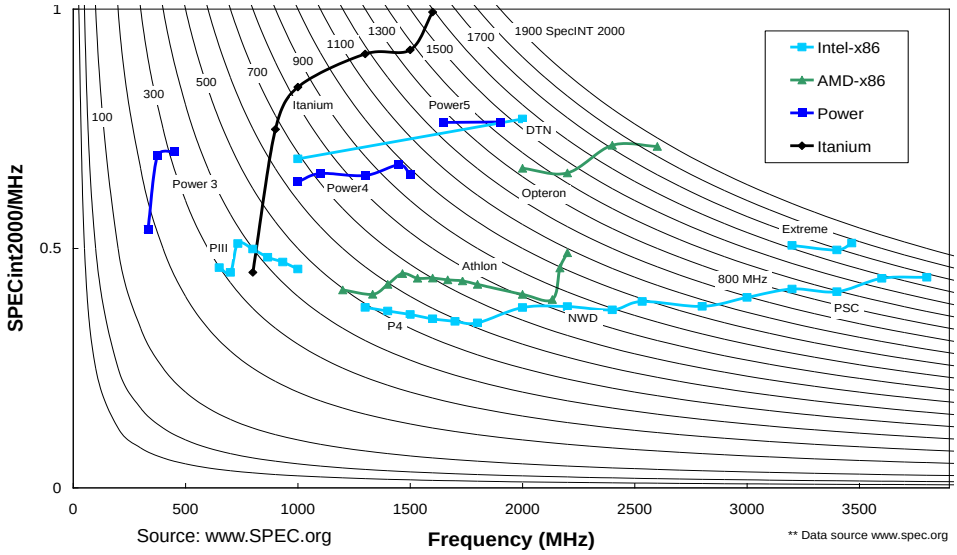
- Sandybridge (2nd generation Core i7)
  - On-chip integrated graphics (GPU)
  - Decoded uop cache up to 1.5K uops, handles loops, but more general
  - 54-entry RS, 168-entry ROB
  - Physical register file: 144 FP, 160 integer
  - 256-bit AVX units: 8 DPFLOP/cycle, 16 SPFLOP/cycle
  - 2 general AGUs enable 2 ld/cycle, 2 st/cycle or any combination, 2x128-bit load path from L1 D$

Landscape of Processor Families [SPECint95]



[John DeVale & Bryan Black, 2005]

Landscape of Processor Families [SPECint2000]



[John DeVale & Bryan Black, 2005]

$$Power = EPI \times IPC \times Frequency$$