Chapter 7: Intel's P6 Architecture

Modern Processor Design: Fundamentals of Superscalar Processors

Pentium Pro Case Study

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

Goals of P6 Microarchitecture

IA-32 Compliant

Performance (Frequency - IPC)

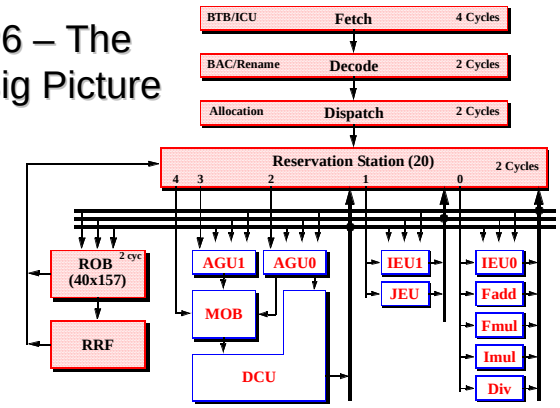
Validation

Die Size

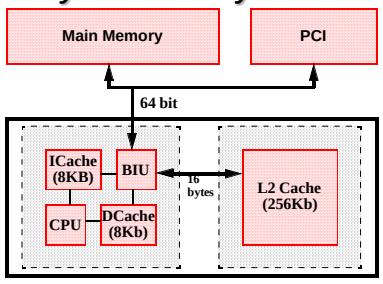
Schedule

Power

P6 – The Big Picture

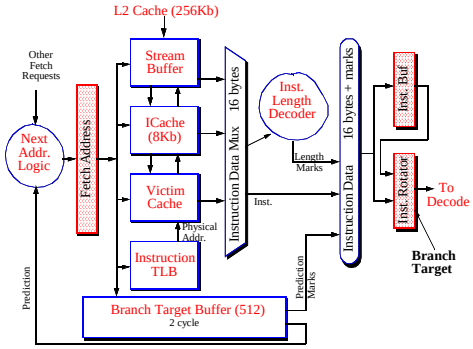


Memory Hierarchy

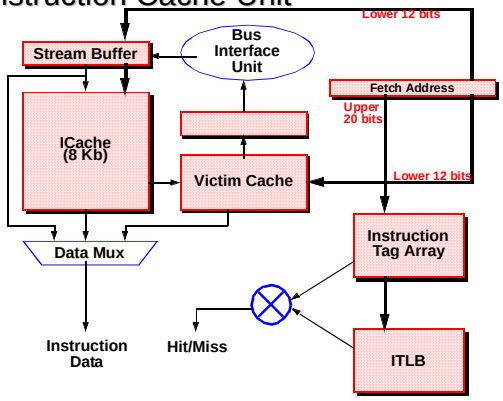


- Level 1 instruction and data caches - 2 cycle access time
- Level 2 unified cache - 6 cycle access time
- Separate level 2 cache and memory address/data bus
- Level 2 cache fill policy - implications

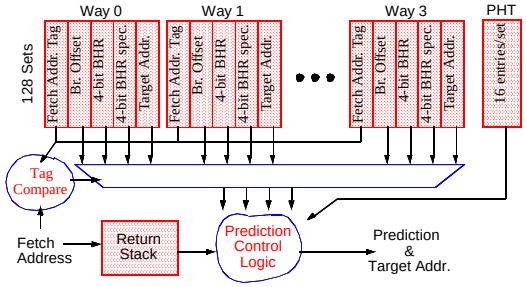
Instruction Fetch



Instruction Cache Unit

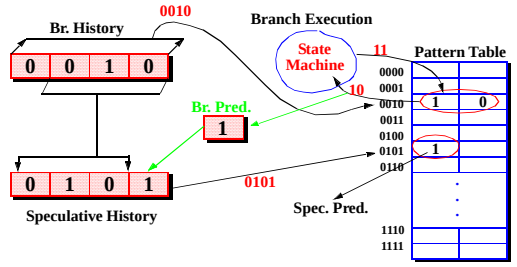


Branch Target Buffer



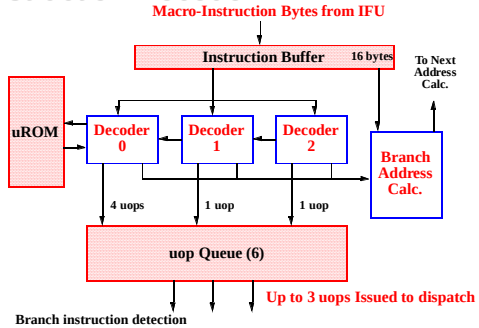
- Pattern History Table (PHT) is not speculatively updated
- A speculative Branch History Register (BHR) and prediction state is maintained
- Uses speculative prediction state if it exist for that branch

Branch Prediction Algorithm



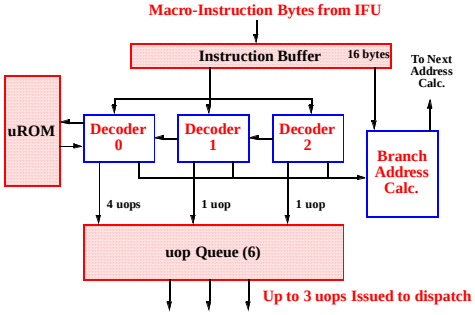
- Current prediction updates the speculative history prior to the next instance of the branch instruction
- Branch History Register (BHR) is updated during branch execution
- Branch recovery flushes front-end and drains the execution core
- Branch mis-prediction resets the speculative branch history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

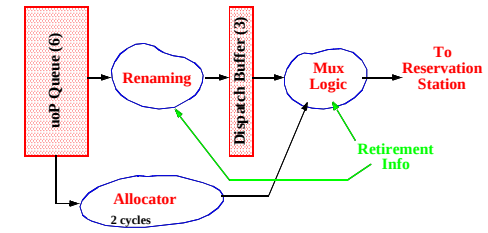
IA-32 instruction

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



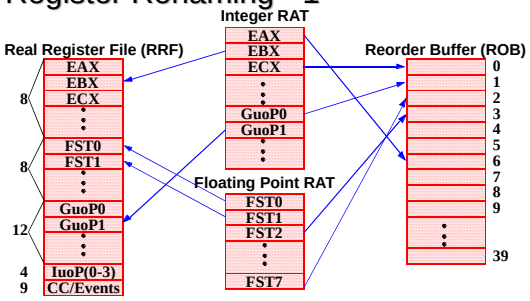
Register Renaming

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

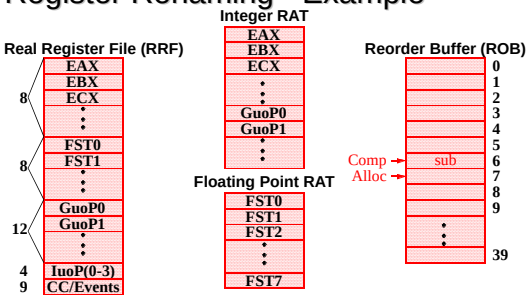


Similar to Tomasulo's Algorithm - Uses ROB entry number as tags

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



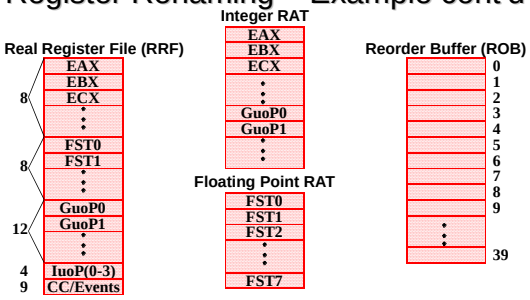
Dispatching:

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

Completing:

```
sub     eax, ecx
```

Register Renaming – Example cont'd



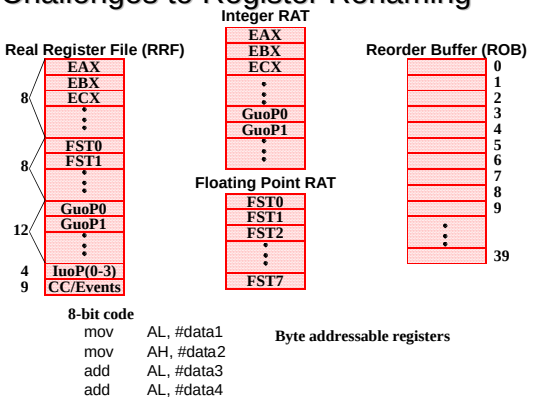
Dispatching:

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

Completing:

```
sub     eax, ecx
```

Challenges to Register Renaming

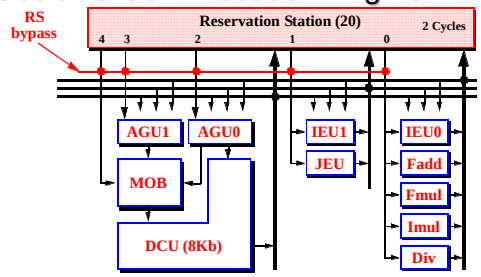


8-bit code

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

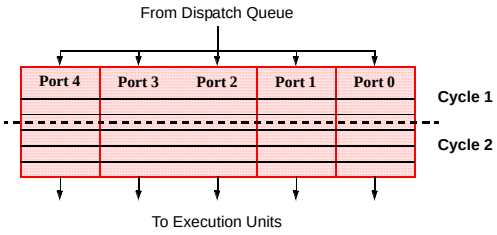
Byte addressable registers

Out-of-Order Execution Engine



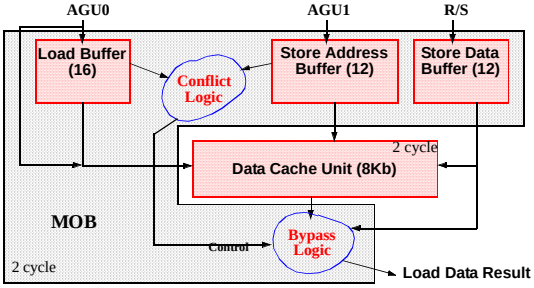
- **In-order branch issue and execution**
- **In-order load/store issue to address generation units**
- **Instruction execution and result bus scheduling**
- **Is the reservation station *"truly"* centralized & what is *"binding"*?**

Reservation Station



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

Memory Ordering Buffer (MOB)

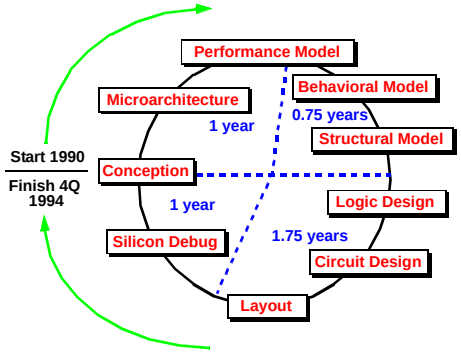


- **Load buffer retains loads until completed, for coherency checking**
- **Store forwarding out of store buffers**
- **2 cycle latency through MOB**
- **"Store Coloring" - Load instructions are tagged by the last store**

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

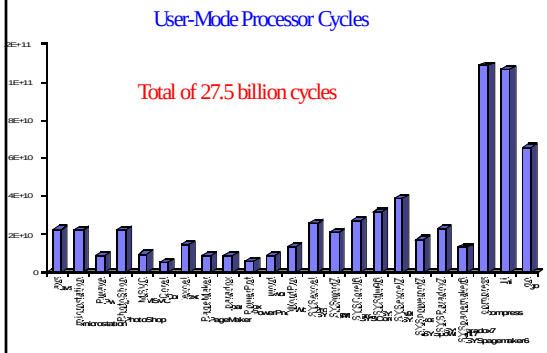
Pentium Pro Design Methodology - 1



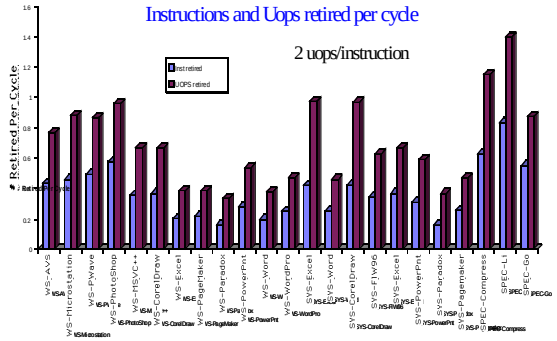
Pentium Pro Performance Analysis

- **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

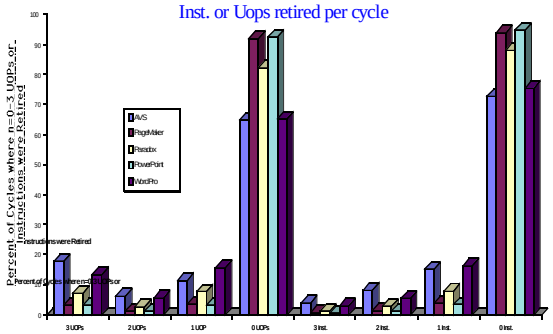
Performance – Run Times



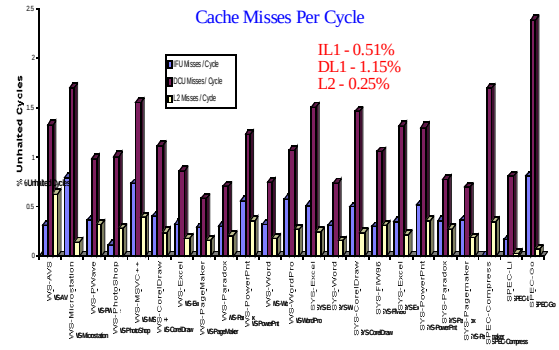
Performance – IPC vs. uPC



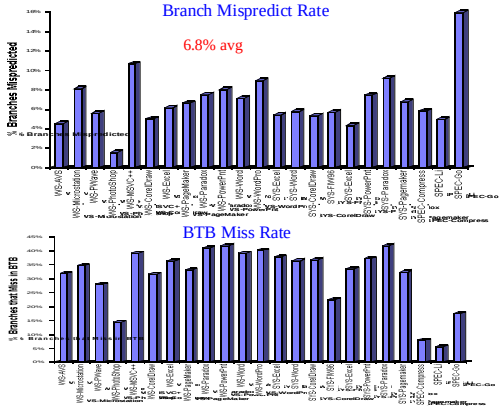
Performance – IPC vs. uPC



Performance – Cache Misses



Performance – Branch Prediction



Conclusions

IA-32 Compliant

Performance (Frequency - IPC)

- 366.0 ISpec92
- 283.2 FSpec92
- 8.09 SPECint95
- 6.70 SPECfp95

Validation

Die Size - Fabable

Schedule - 1 year late

Power -