

Pentium Pro Case Study

Prof. Mikko H. Lipasti
University of Wisconsin-Madison

Lecture notes based on notes by John P. Shen
Updated by Mikko Lipasti

Pentium Pro Case Study

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

Goals of P6 Microarchitecture

IA-32 Compliant

Performance (Frequency - **IPC**)

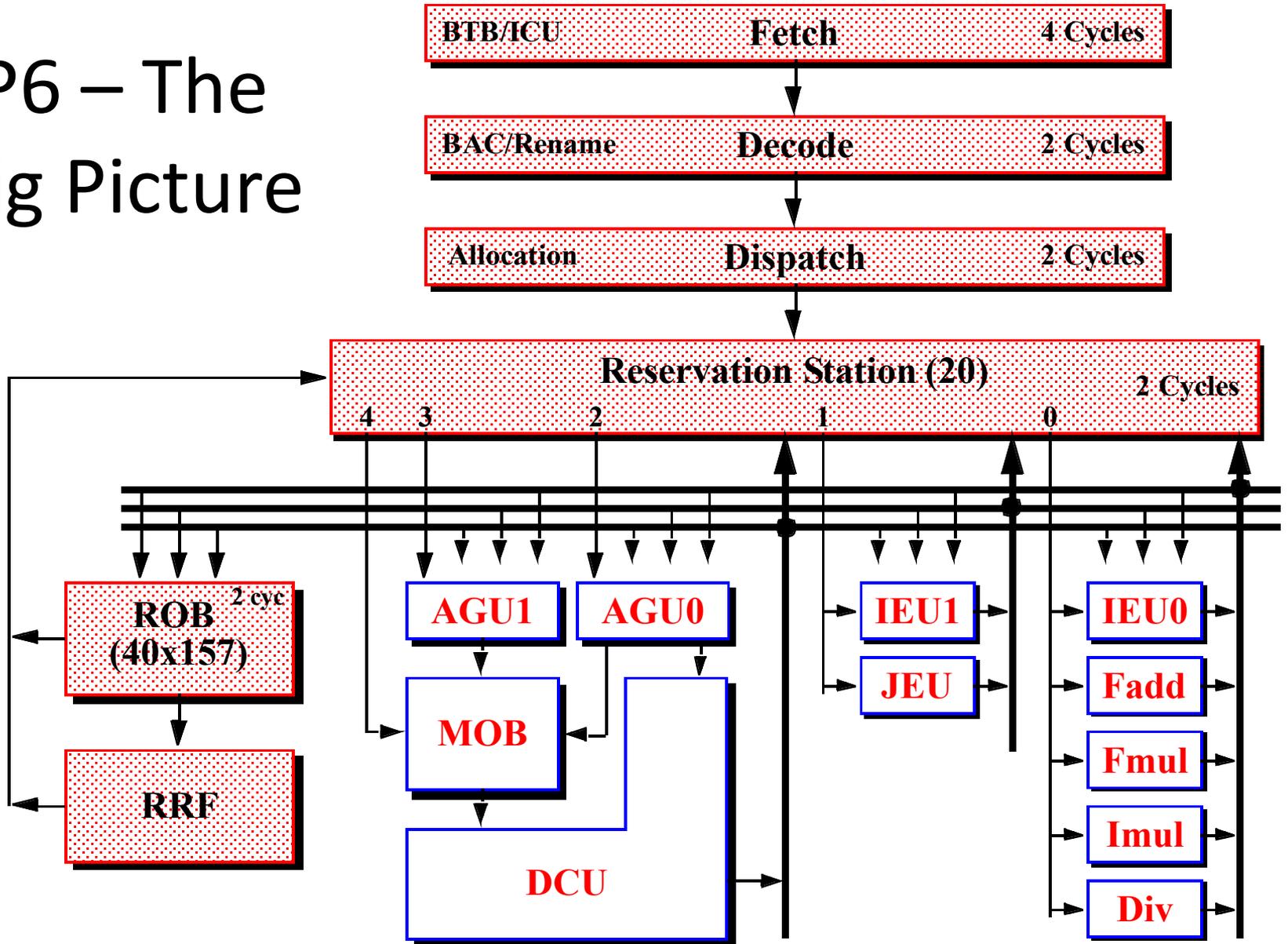
Validation

Die Size

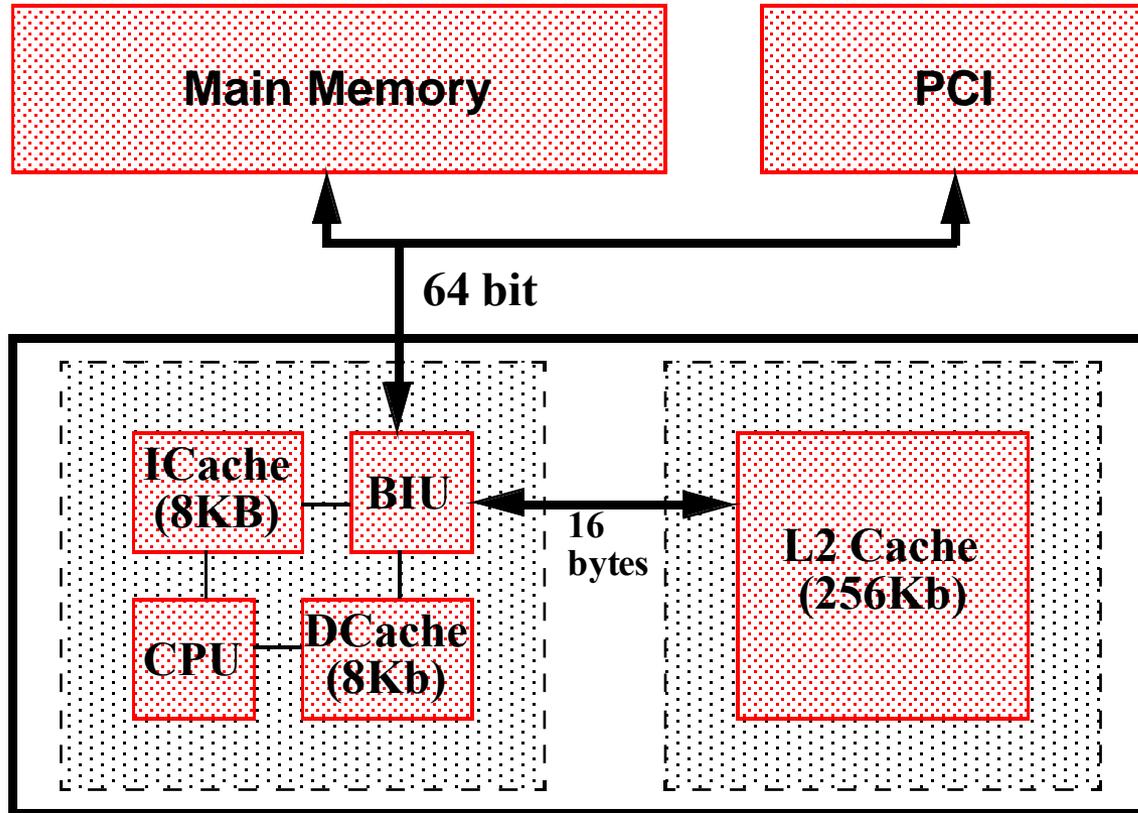
Schedule

Power

P6 – The Big Picture

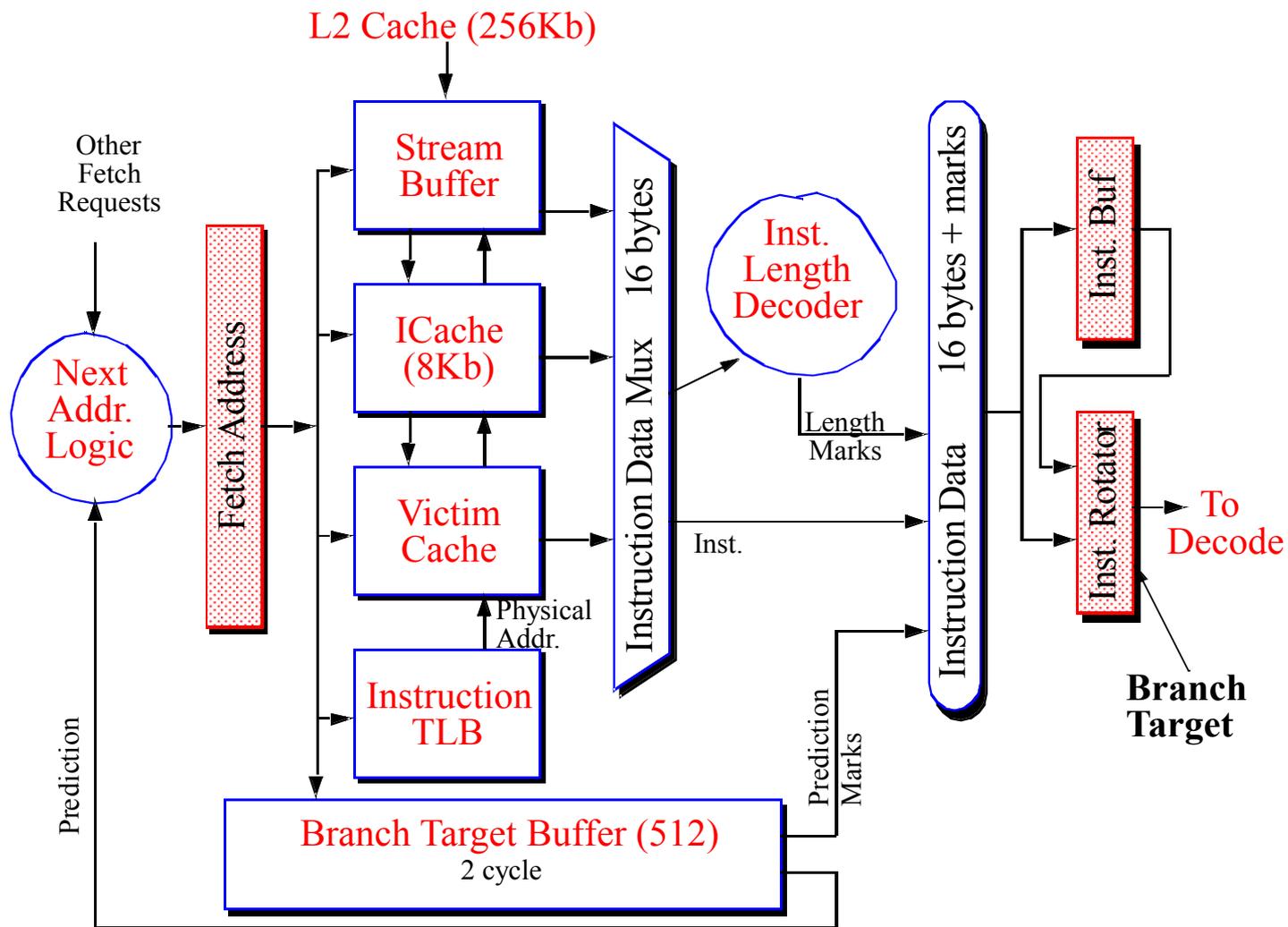


Memory Hierarchy

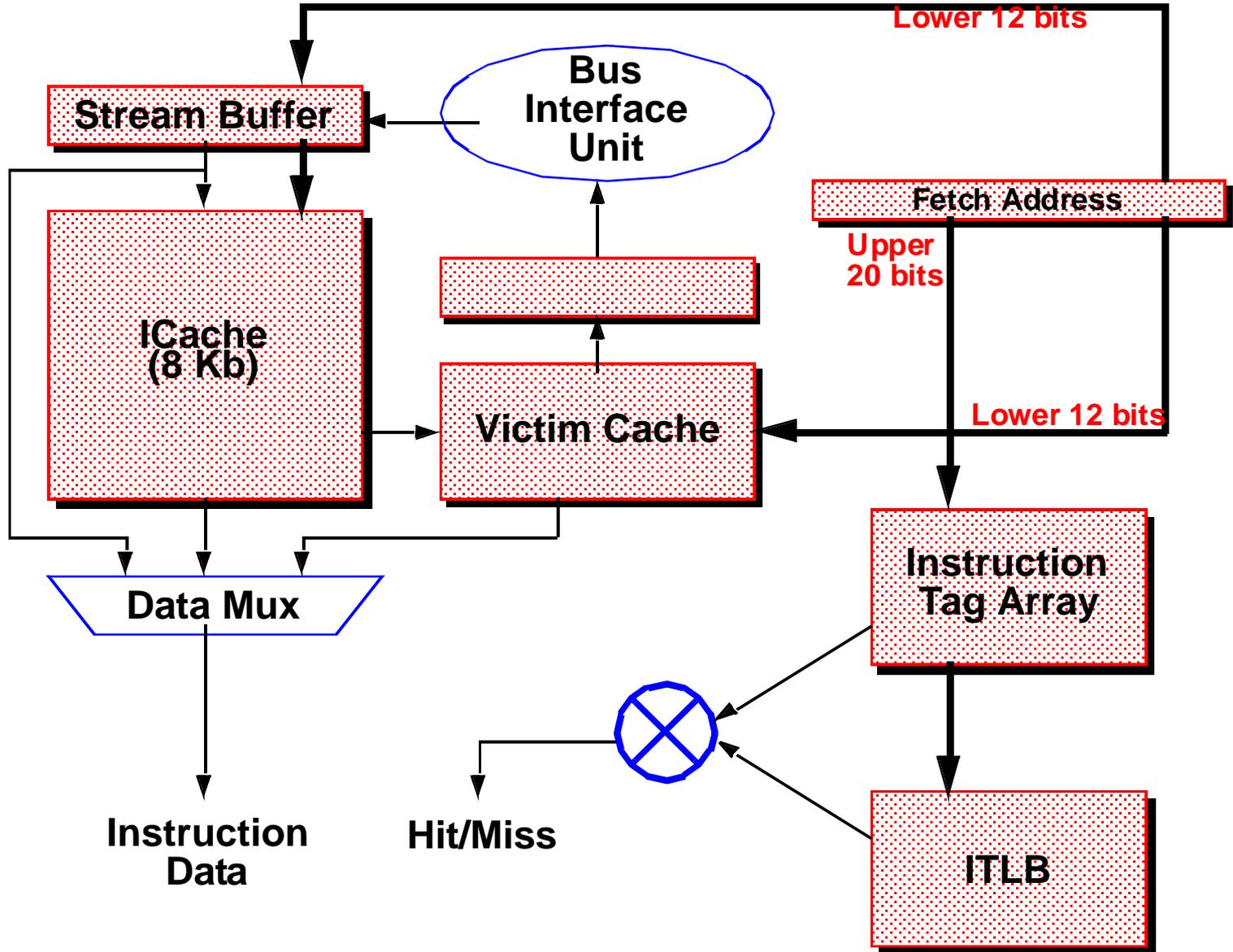


- Level 1 instruction and data caches - 2 cycle access time
- Level 2 unified cache - 6 cycle access time
- Separate level 2 cache and memory address/data bus

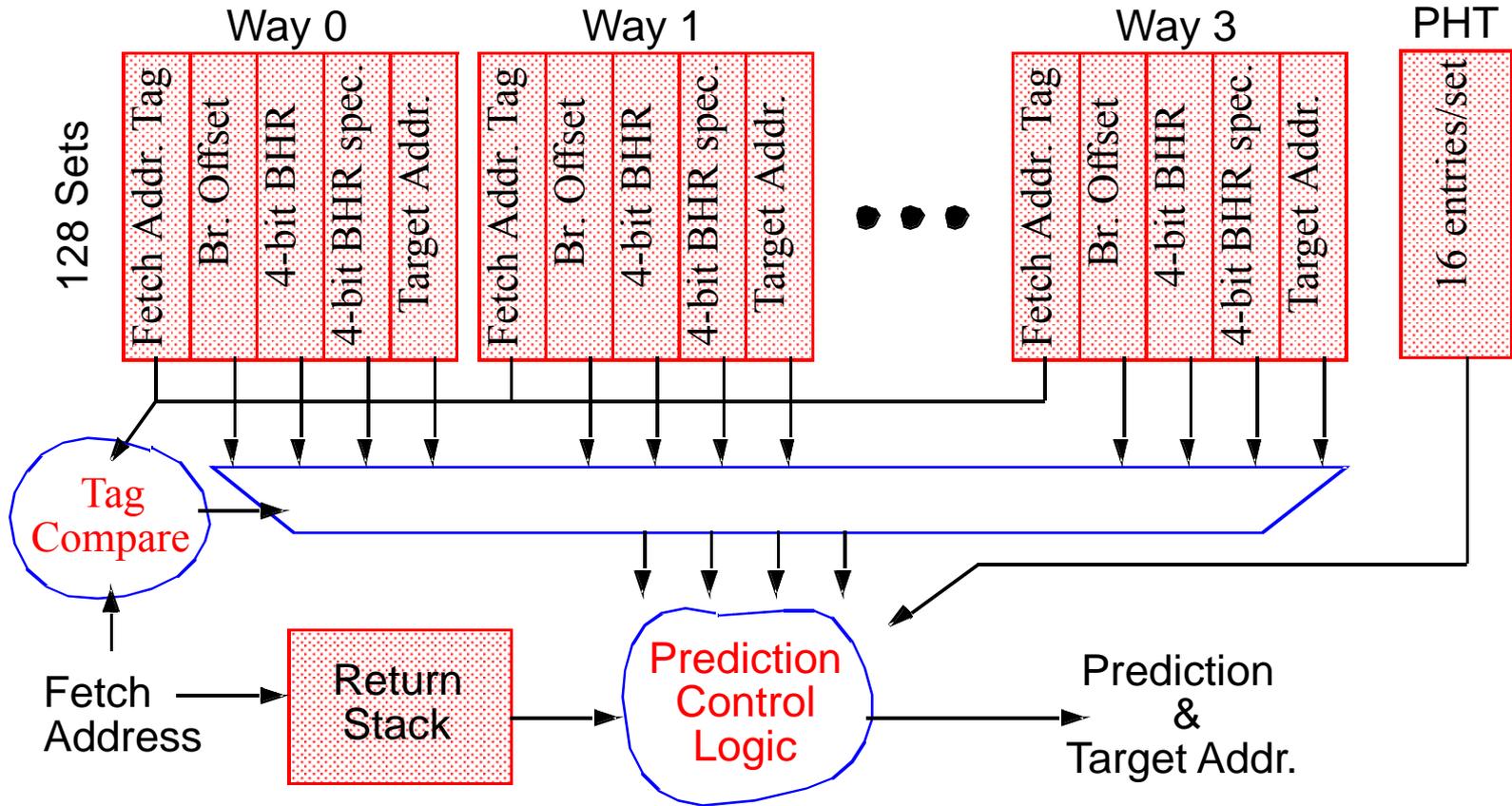
Instruction Fetch



Instruction Cache Unit

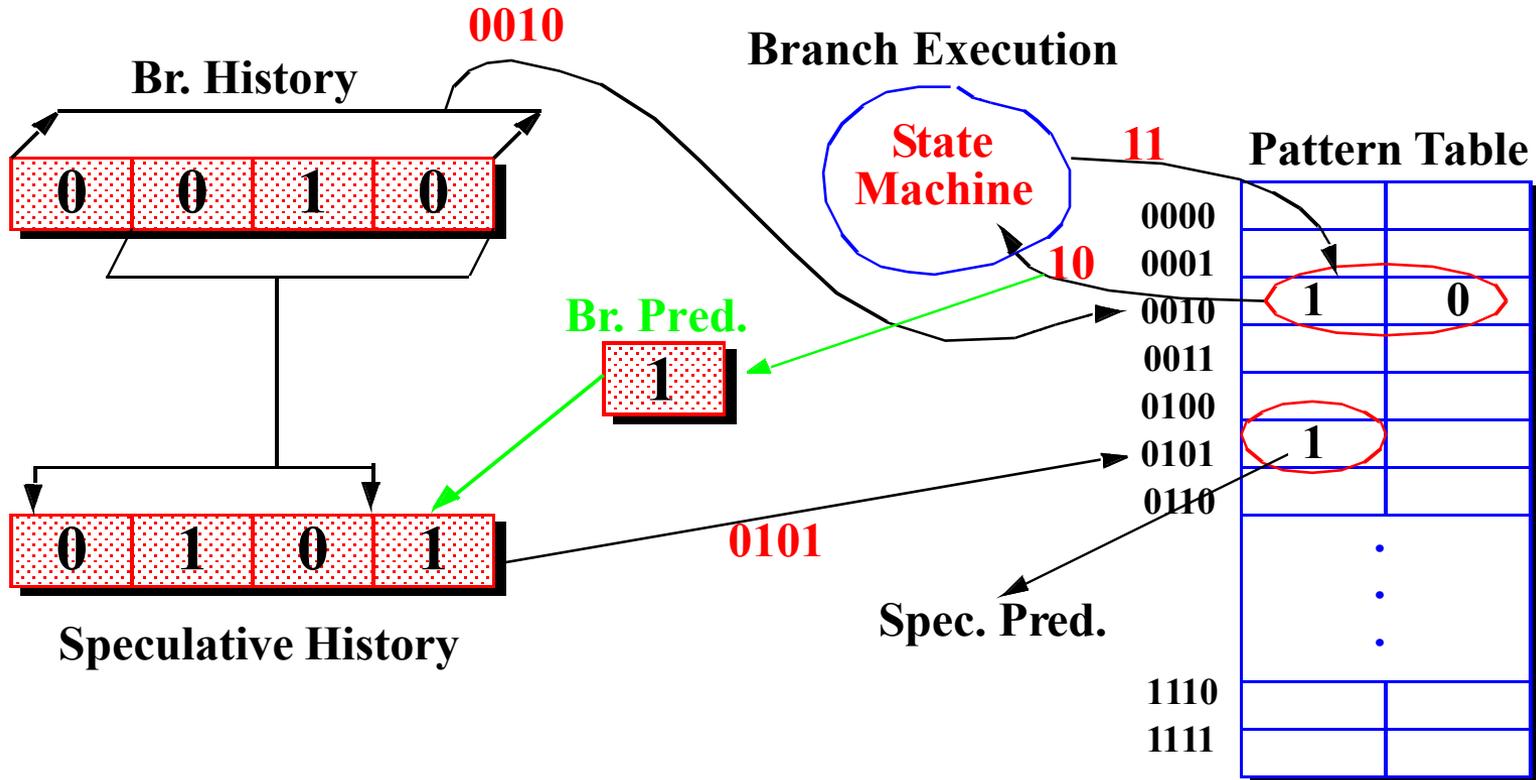


Branch Target Buffer



- 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

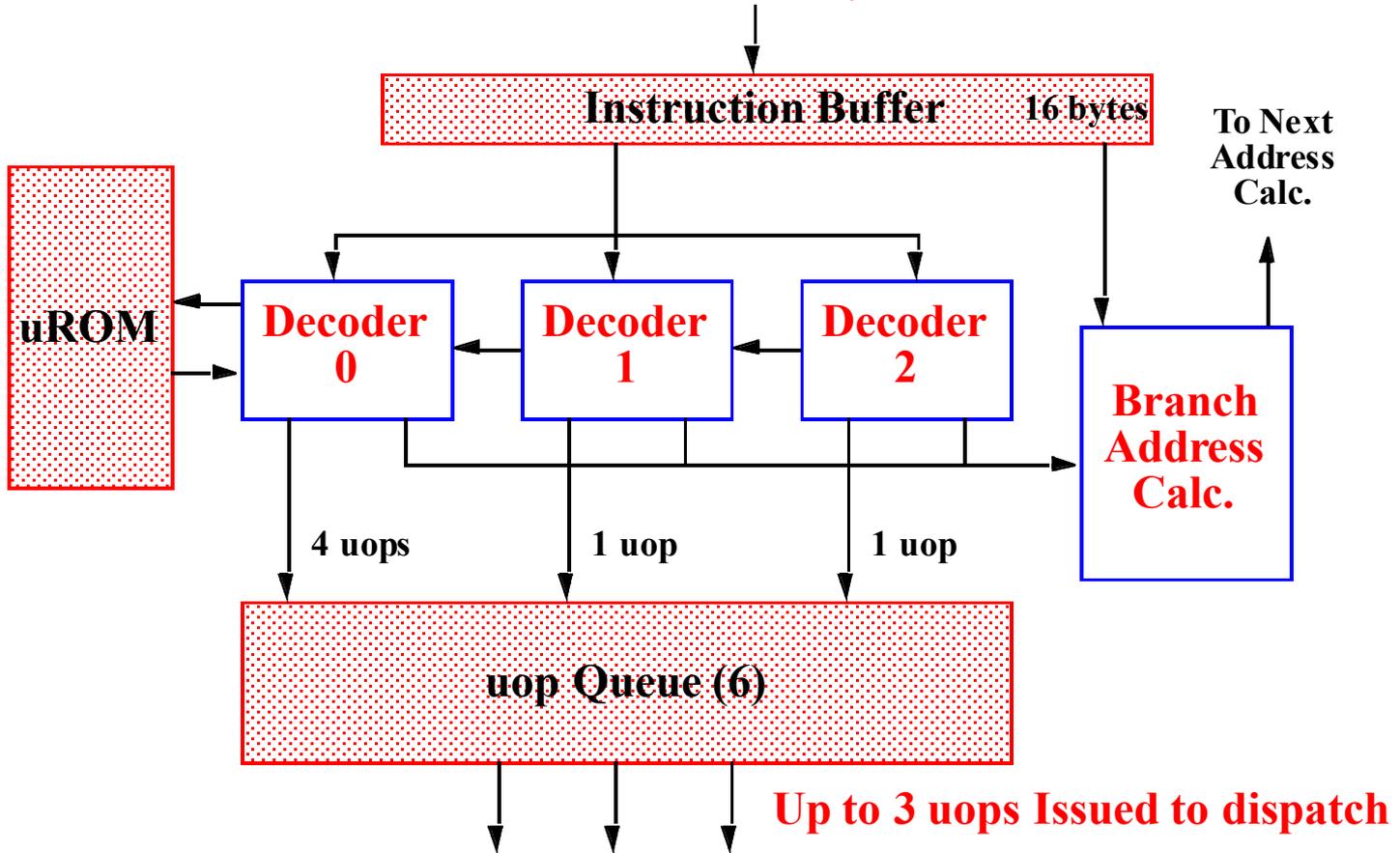
Branch Prediction Algorithm



- 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

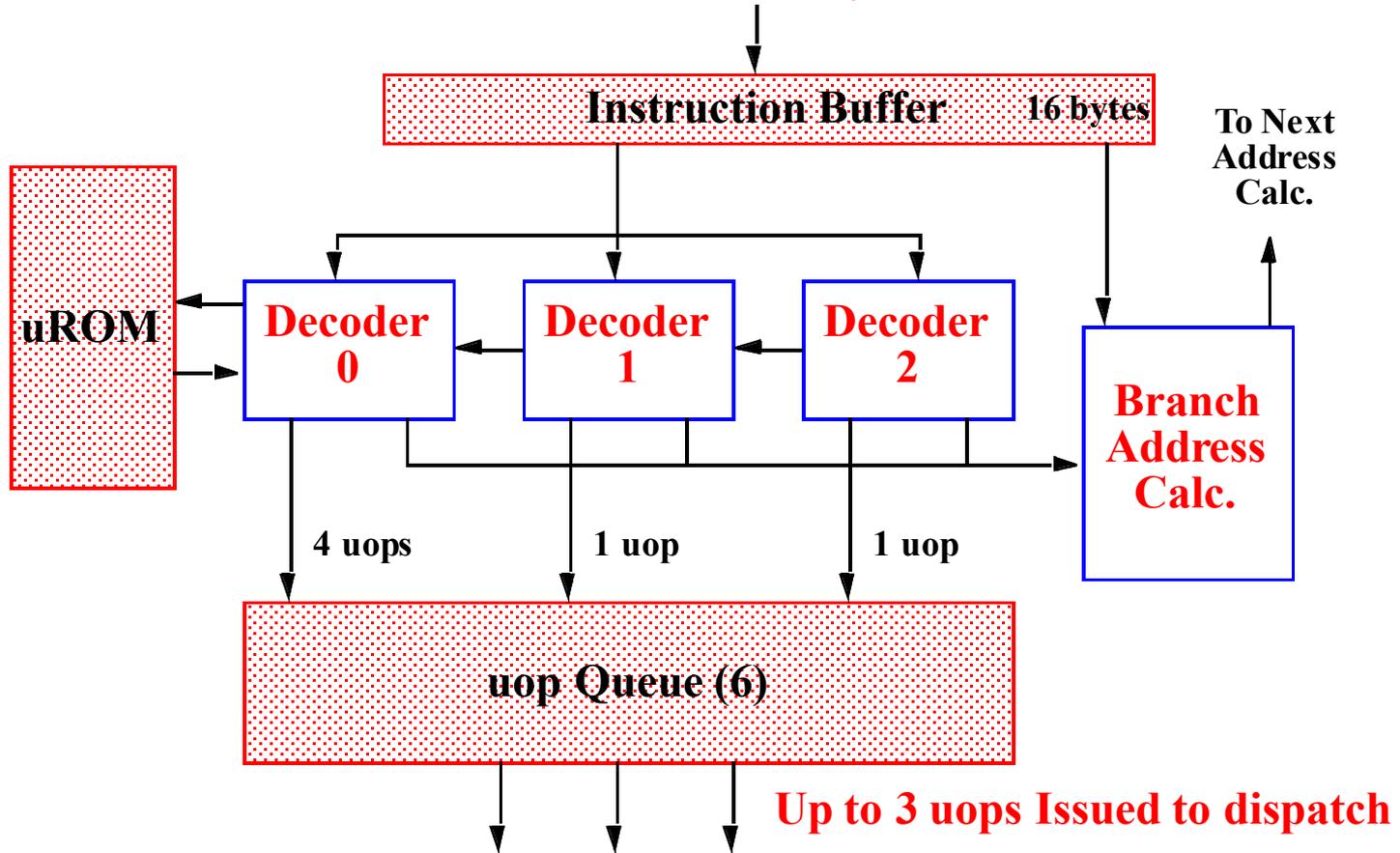
Macro-Instruction Bytes from IFU



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

Instruction Decode - 2

Macro-Instruction Bytes from IFU



- 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) $\text{MEM}(\text{eax}) \leftarrow \text{MEM}(\text{eax}) + \text{MEM}(\text{ebx})$

Uop decomposition:

ld guop0, (eax)

guop0 \leftarrow MEM(eax)

ld guop1, (ebx)

guop1 \leftarrow MEM(ebx)

add guop0,guop1

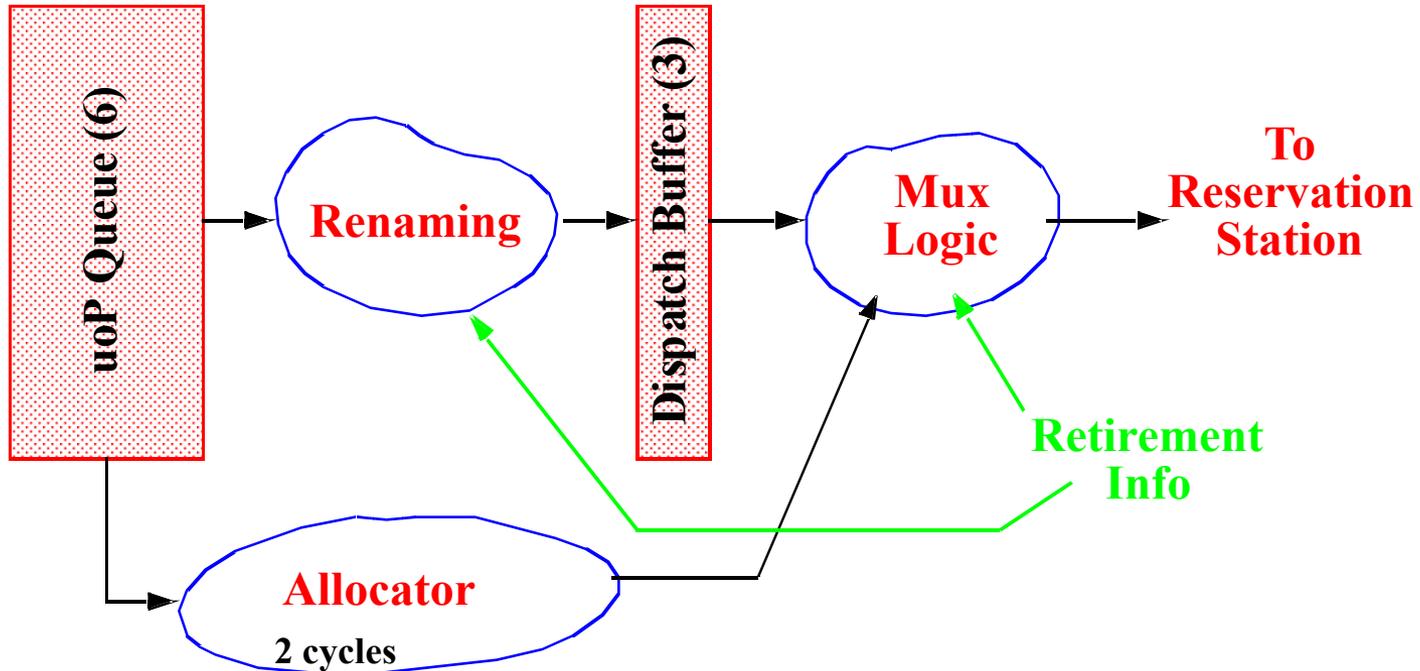
guop0 \leftarrow guop0 + guop1

sta eax

std guop0

$\text{MEM}(\text{eax}) \leftarrow$ guop0

Instruction Dispatch



Register Renaming

Allocation requirements

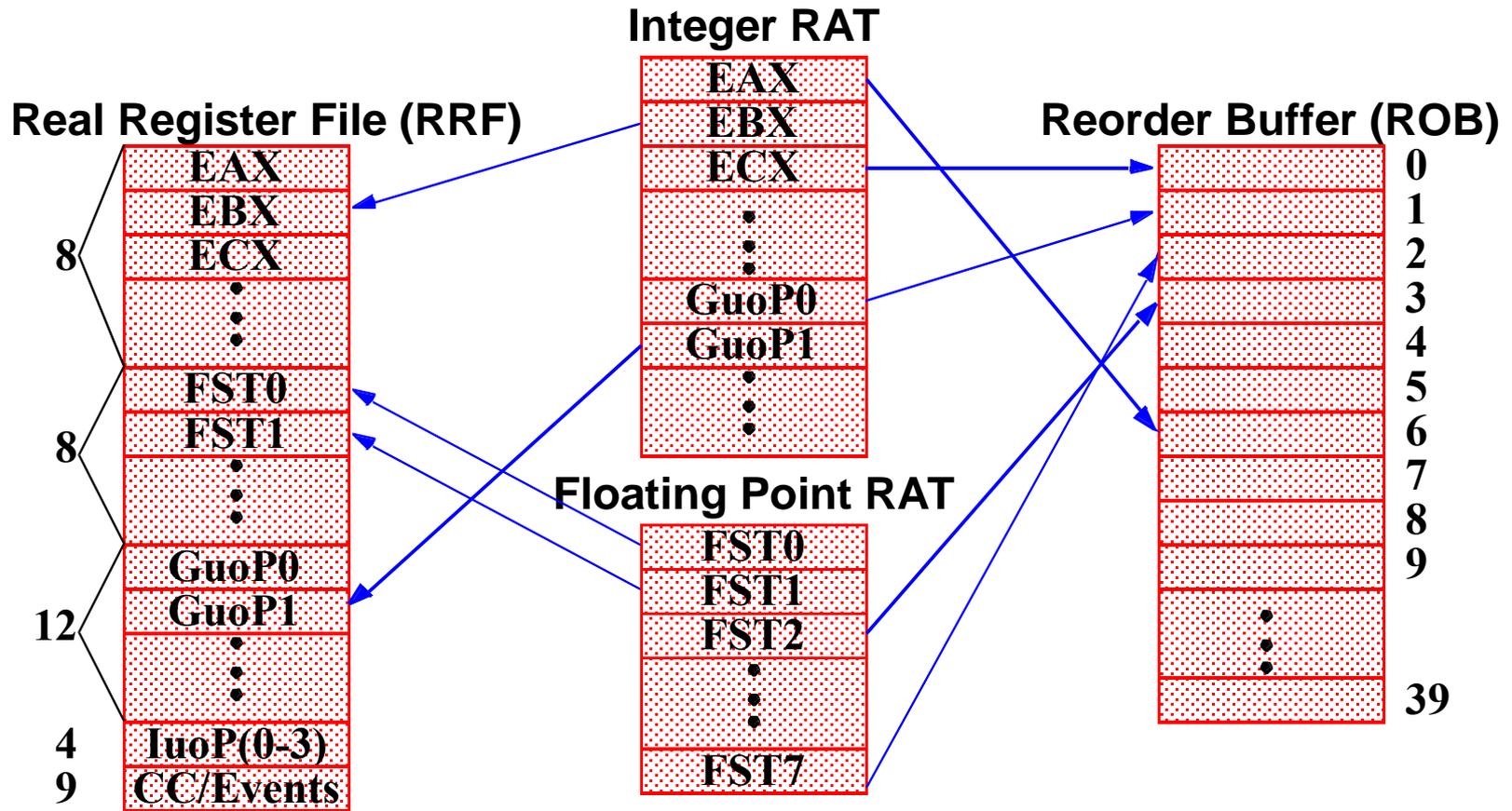
“3-or-none” Reorder buffer entries

Reservation station entry

Load buffer or store buffer entry

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

Register Renaming - 1

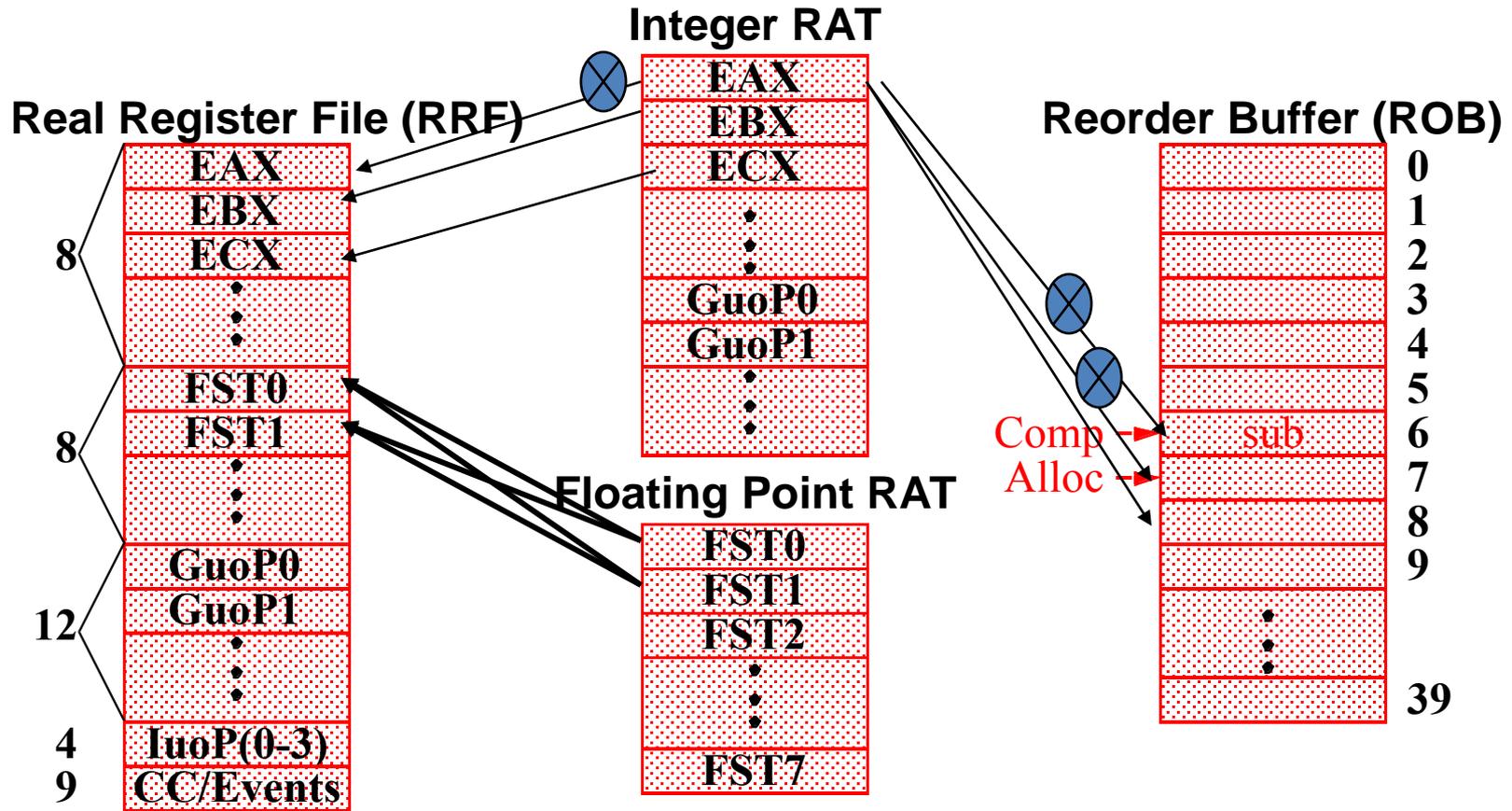


Similar to Tomasulo's Algorithm - Uses ROB entry number as tags

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



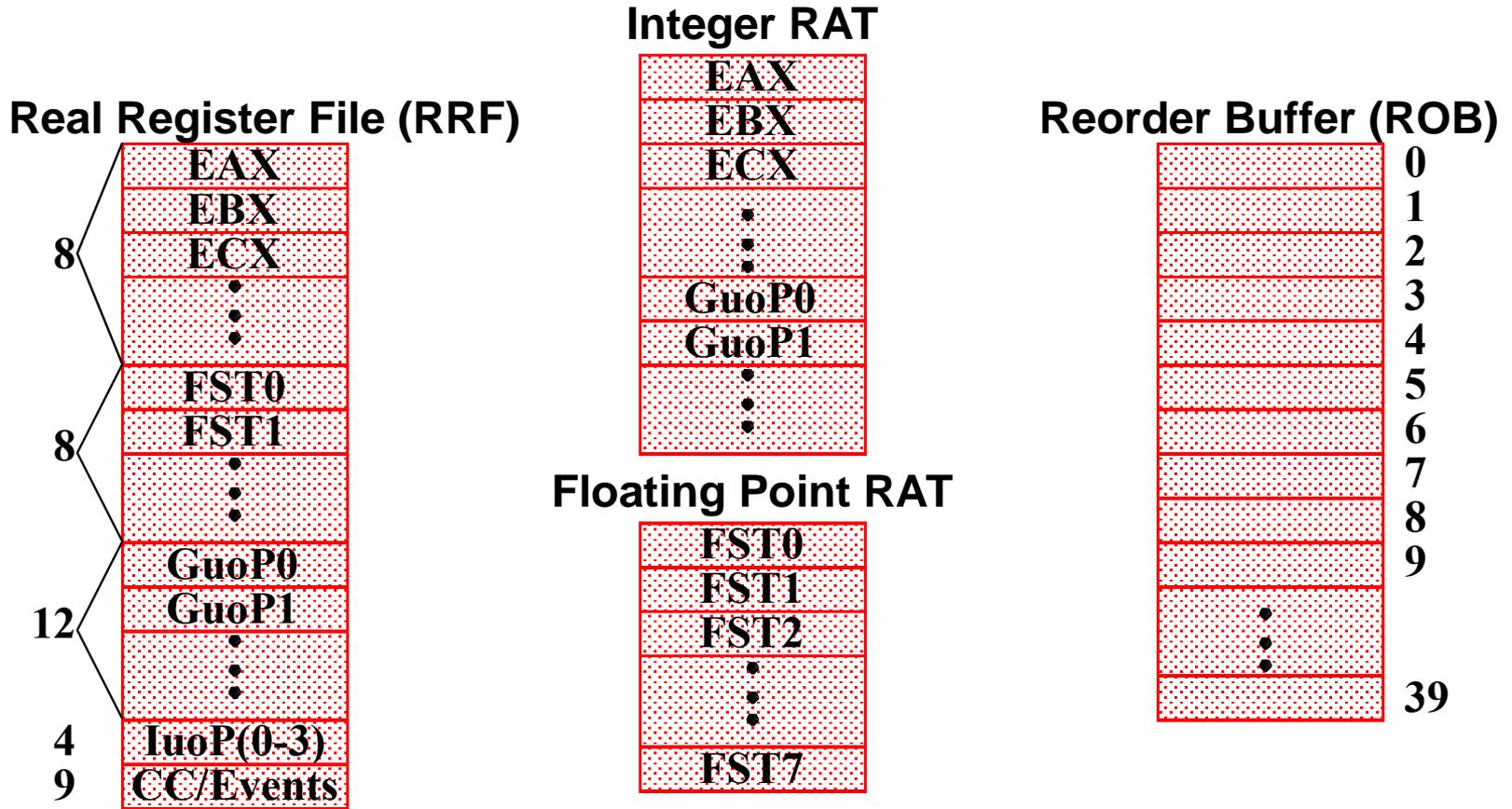
Dispatching:

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

Completing:

sub eax, ecx

Challenges to Register Renaming



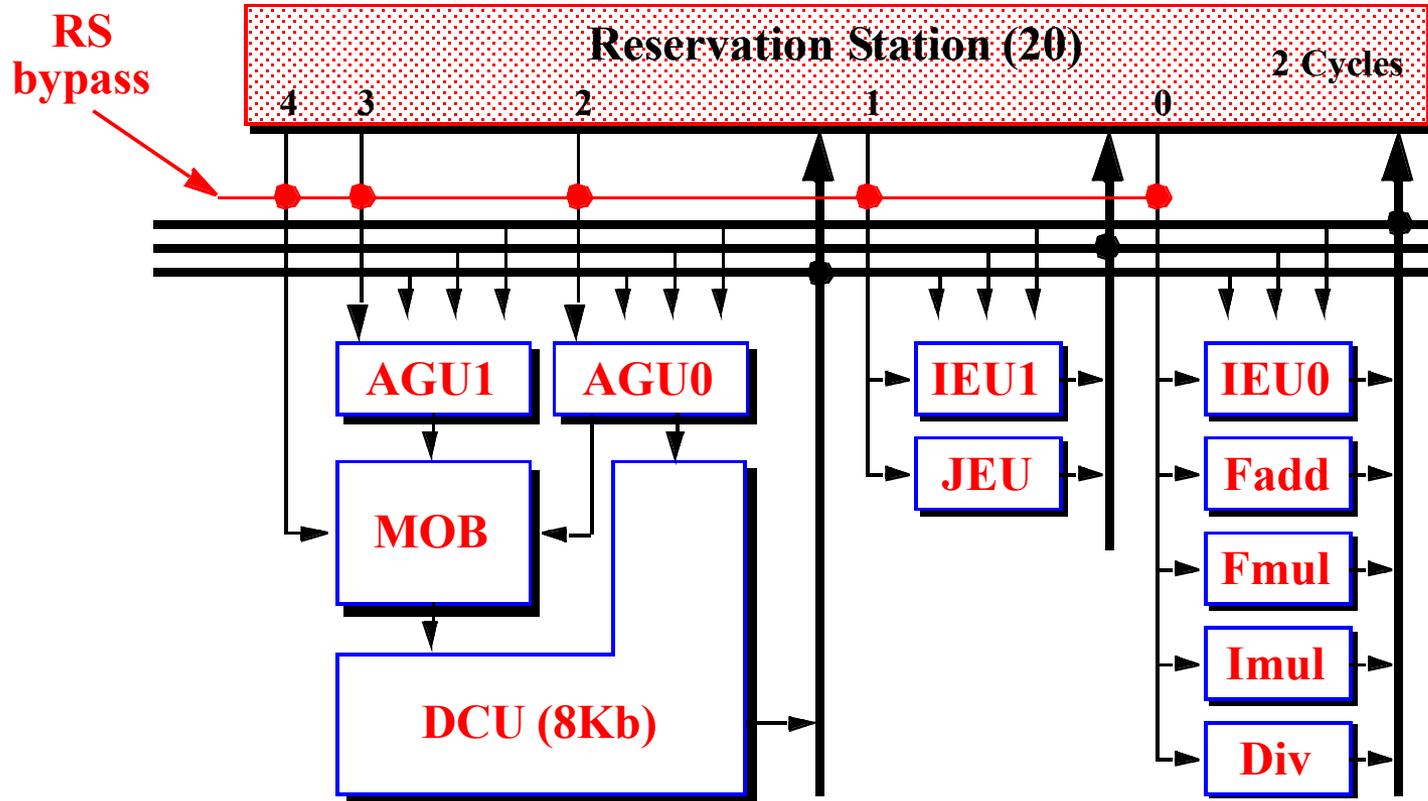
8-bit code

```

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

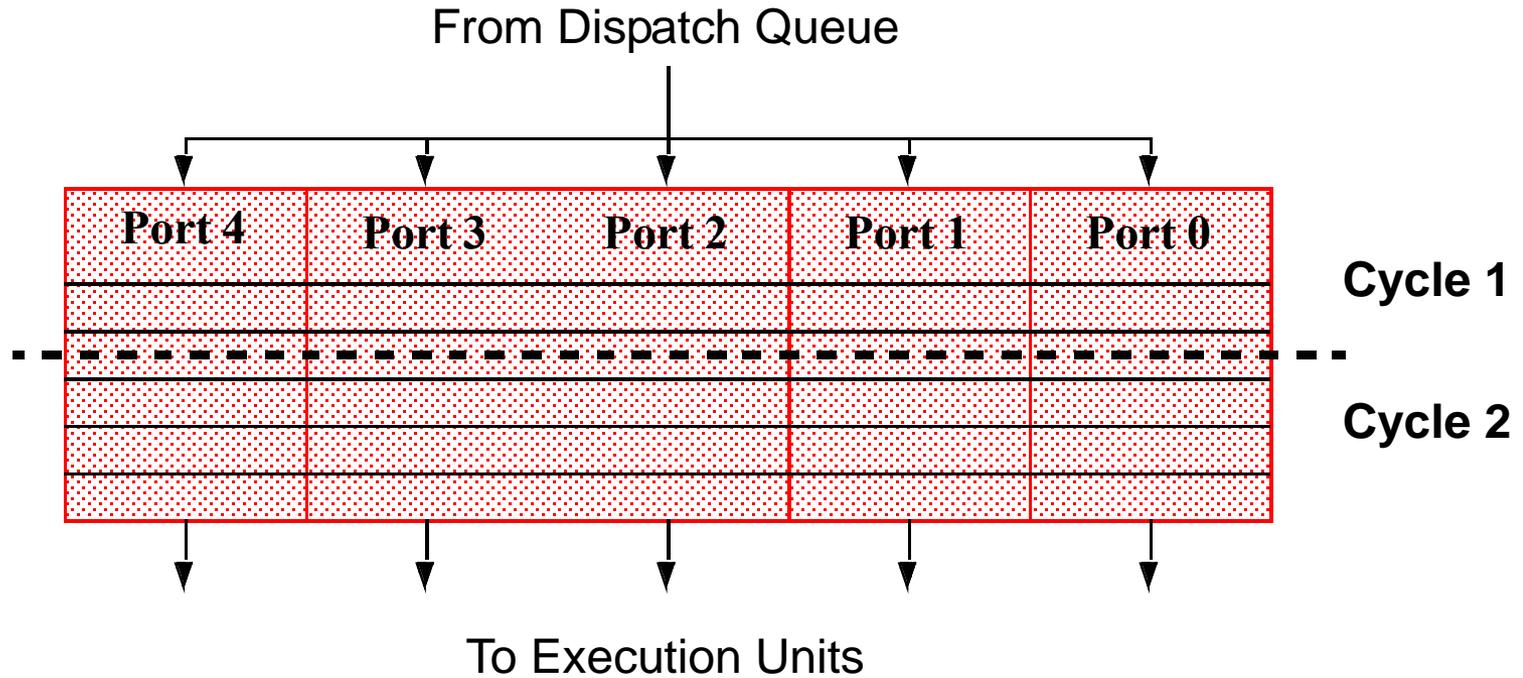
Byte addressable registers

Out-of-Order Execution Engine



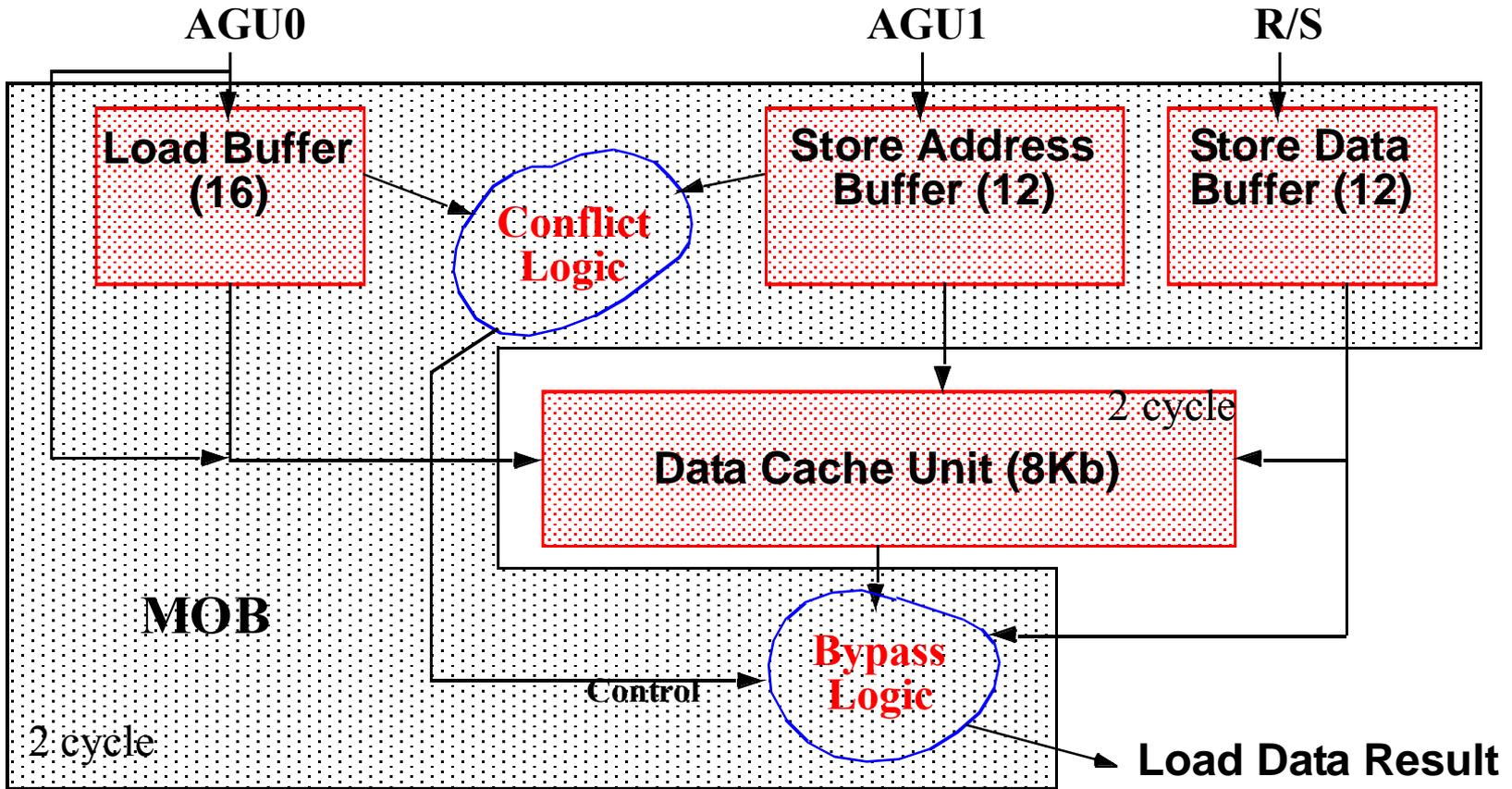
- In-order branch issue and execution
- In-order load/store issue to address generation units
- Instruction execution and result bus scheduling
- Is the reservation station *“truly”* centralized & what is *“binding”*?

Reservation Station



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

Memory Ordering Buffer (MOB)

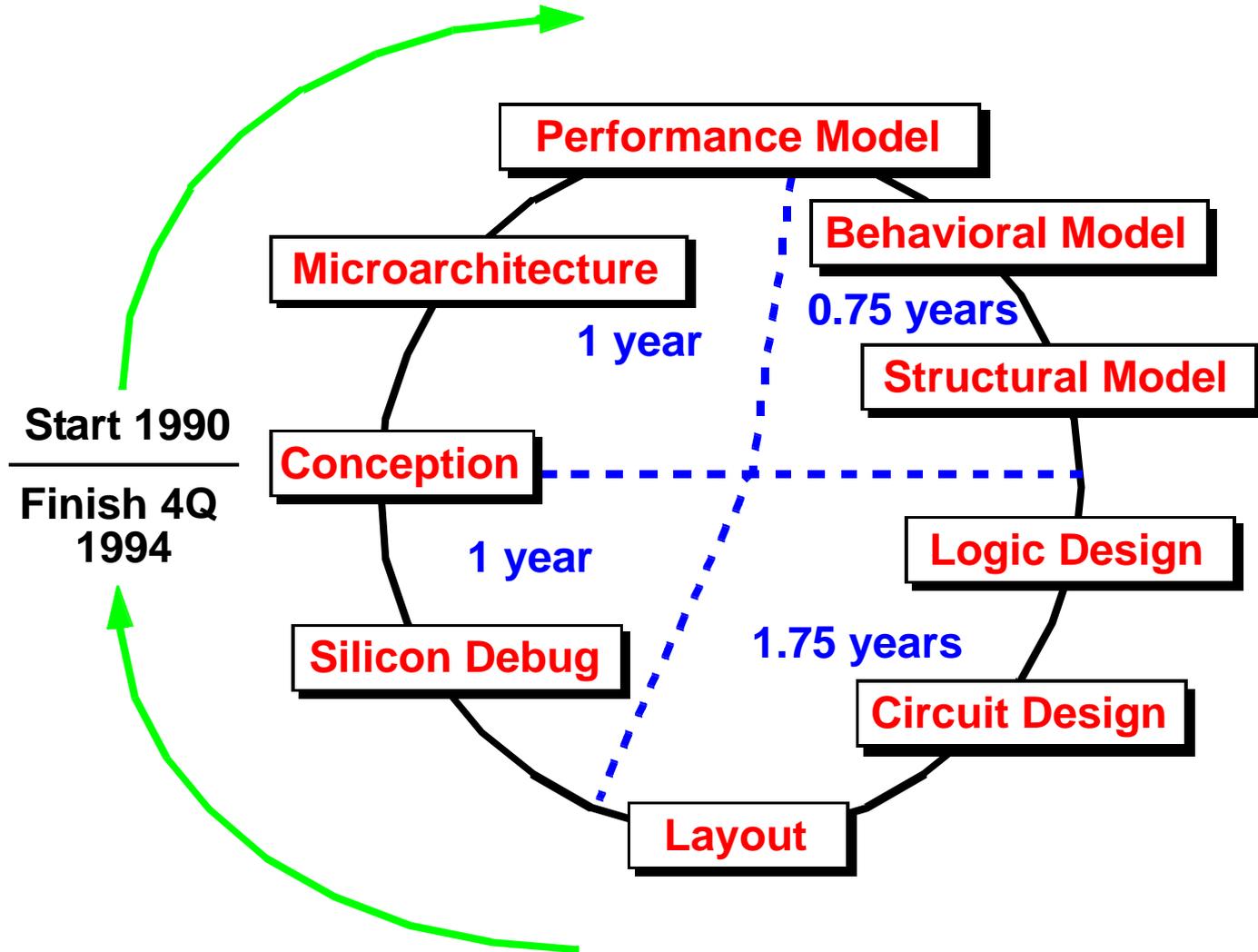


- Load buffer retains loads until completed, for coherency checking
- Store forwarding out of store buffers
- 2 cycle latency through MOB
- “Store Coloring” - Load instructions are tagged by the last store

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

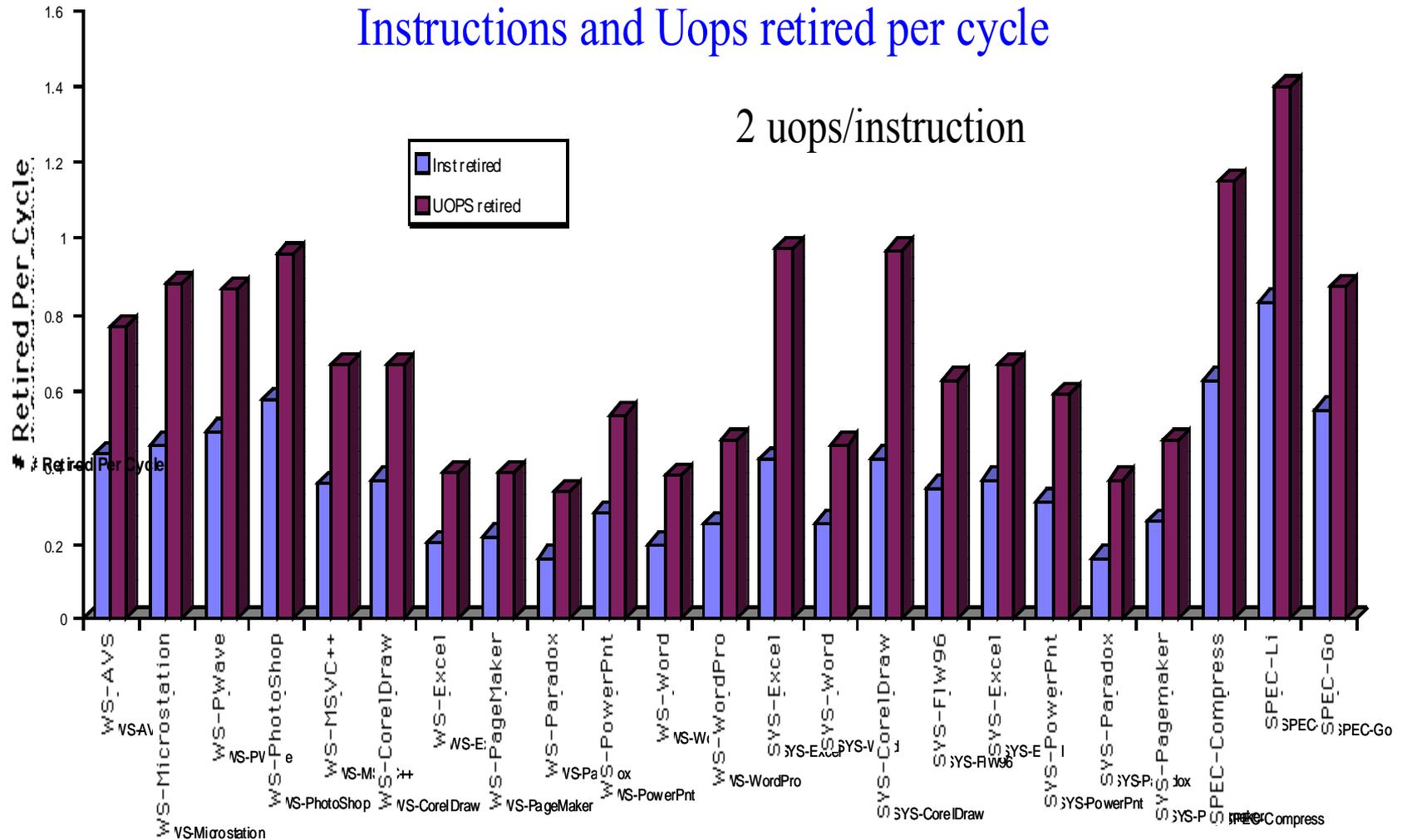
- **Benchmark Suite**

- BAPco Sysmark32 - 32-bit Windows NT applications
- Winstone97 - 32-bit Windows NT applications
- Some SPEC95 benchmarks

Performance – IPC vs. uPC

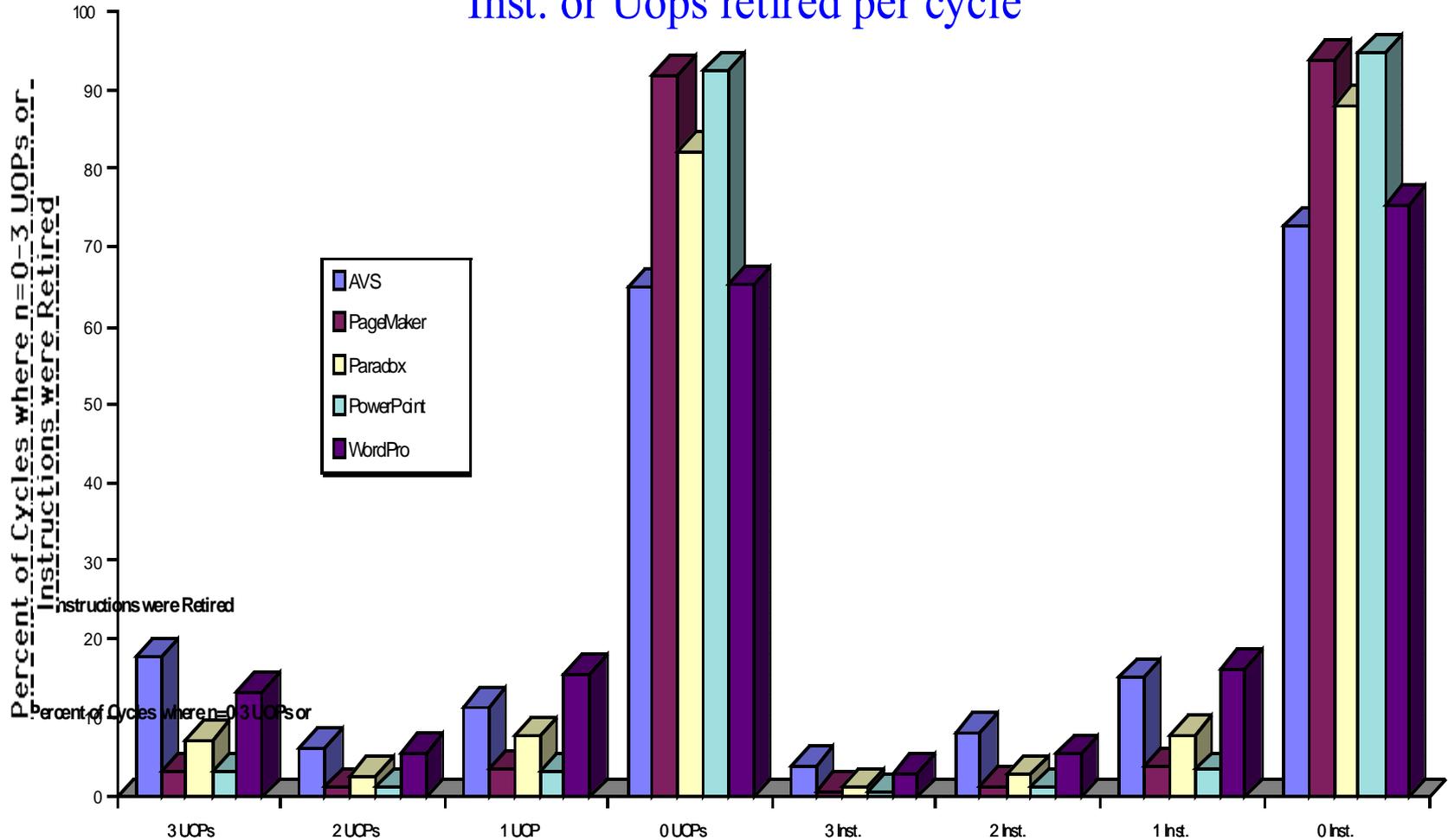
Instructions and Uops retired per cycle

2 uops/instruction

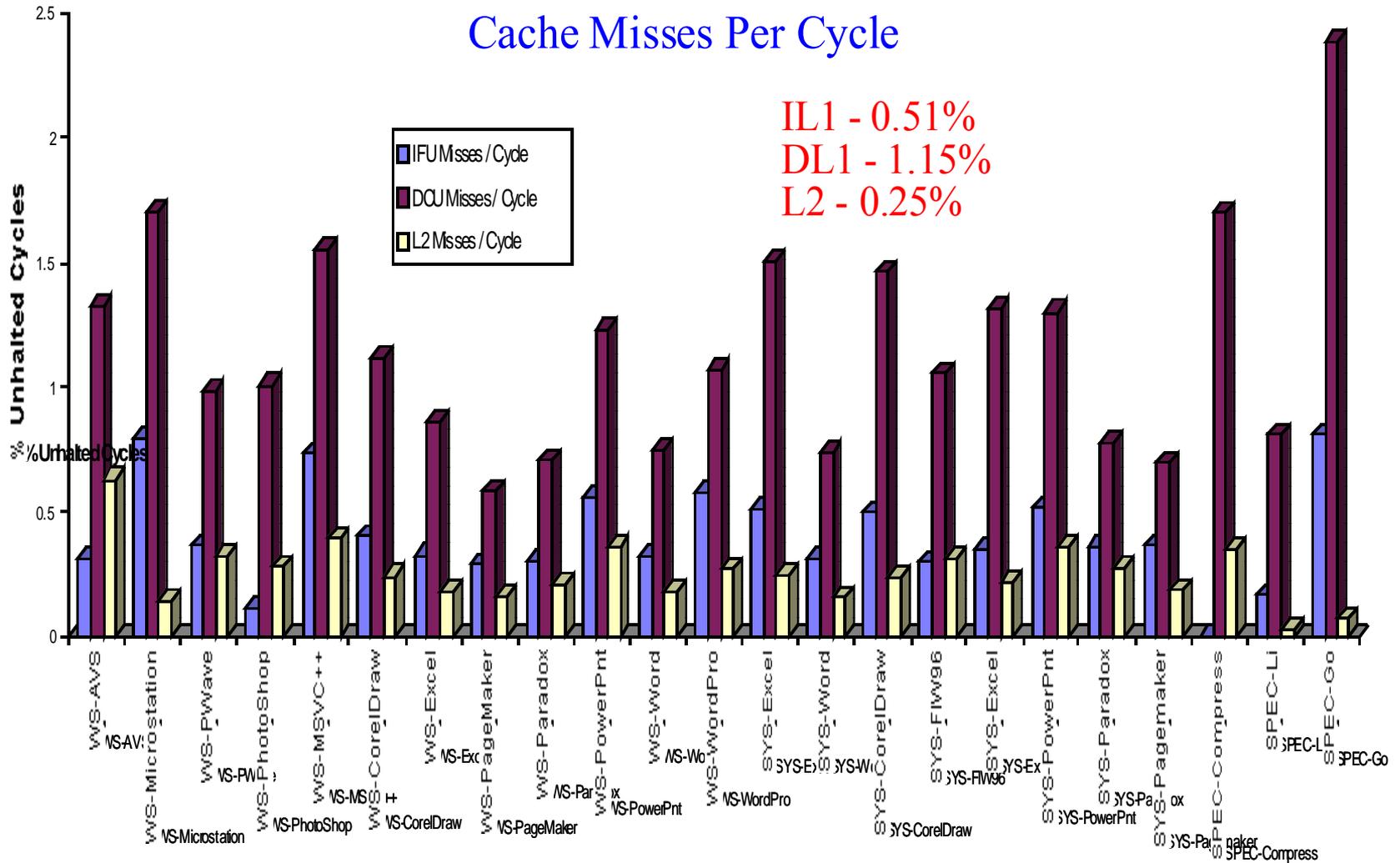


Performance – IPC vs. uPC

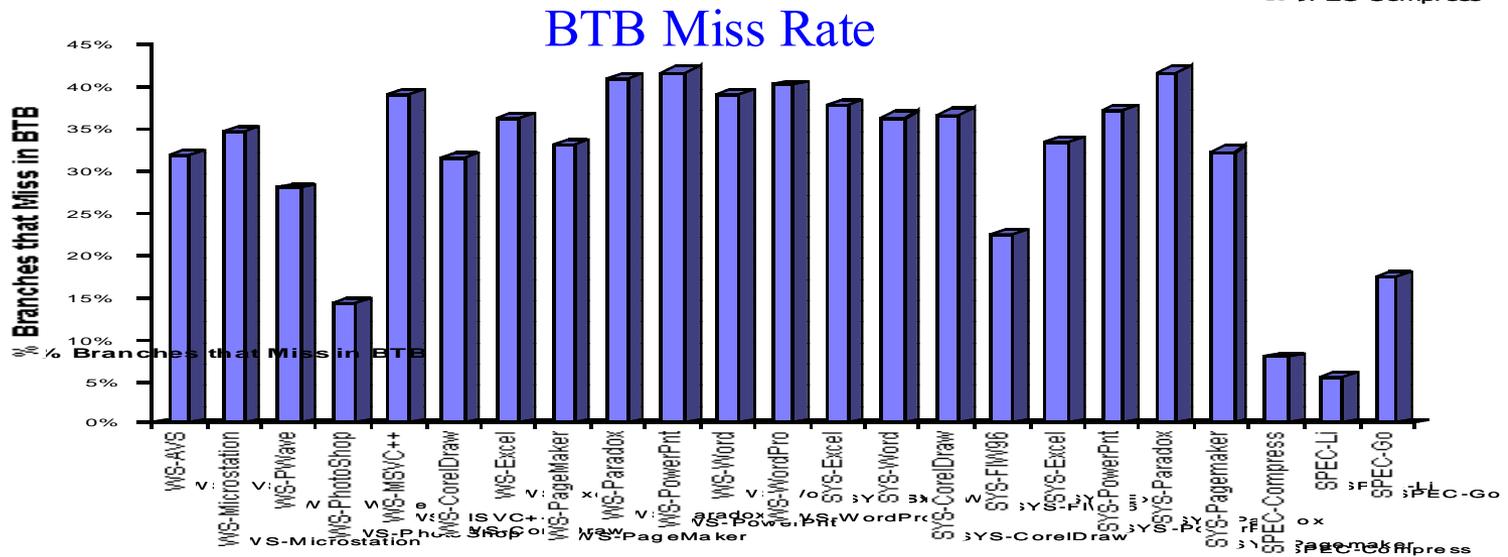
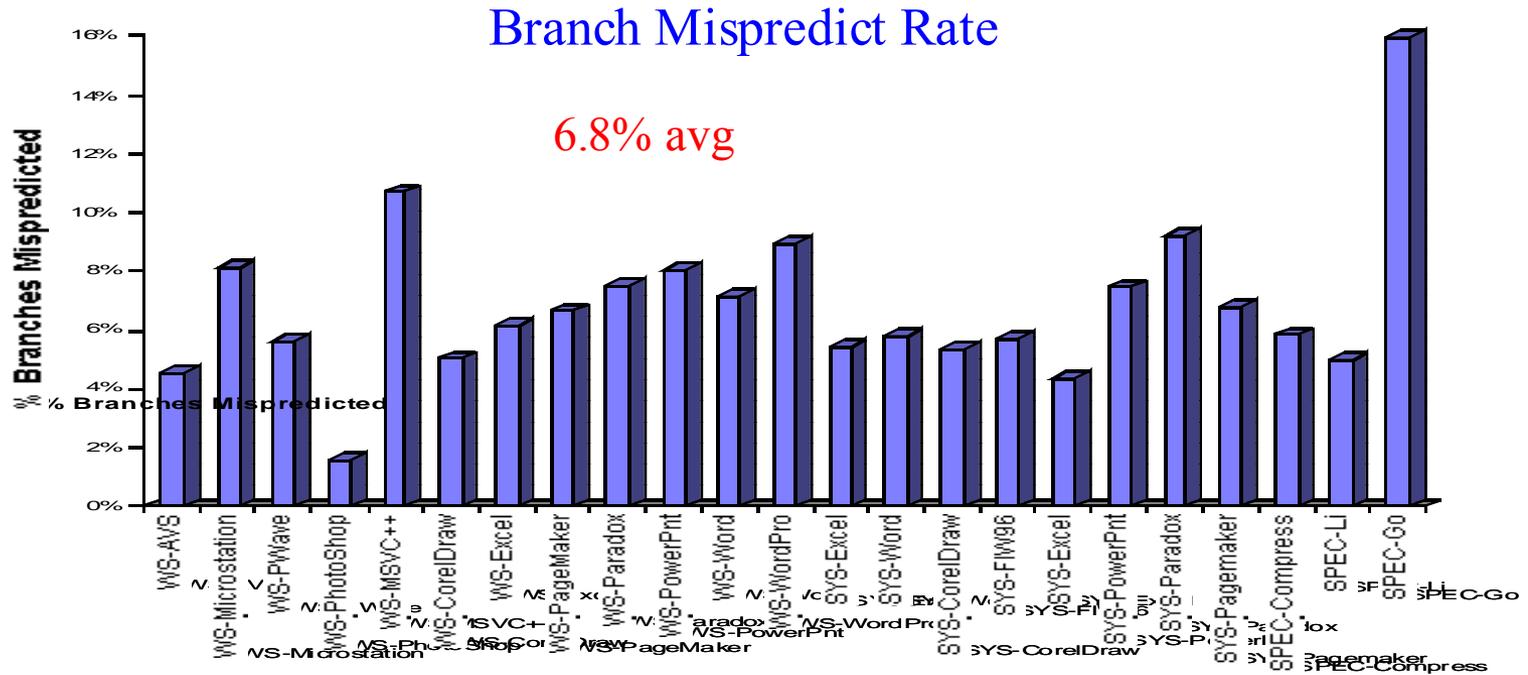
Inst. or Uops retired per cycle



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 -

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 (Banas), 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/Ivy Bridge (2nd-3rd 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\$

Microarchitectural Updates

- Haswell/Broadwell/Skylake: wider & deeper
 - 8-wide issue (up from 6 wide)
 - 4th integer ALU, third AGU, second branch unit
 - 60-entry RS, 192-entry ROB
 - 72-entry load queue/42-entry store queue
 - Physical register file: 168 FP, 168 integer
 - Doubled FP throughput (32 SP/16 DP)
 - Load/store bandwidth to L1 doubled (64B/32B)
 - TSX (transactional memory)
 - Integrated voltage regulator