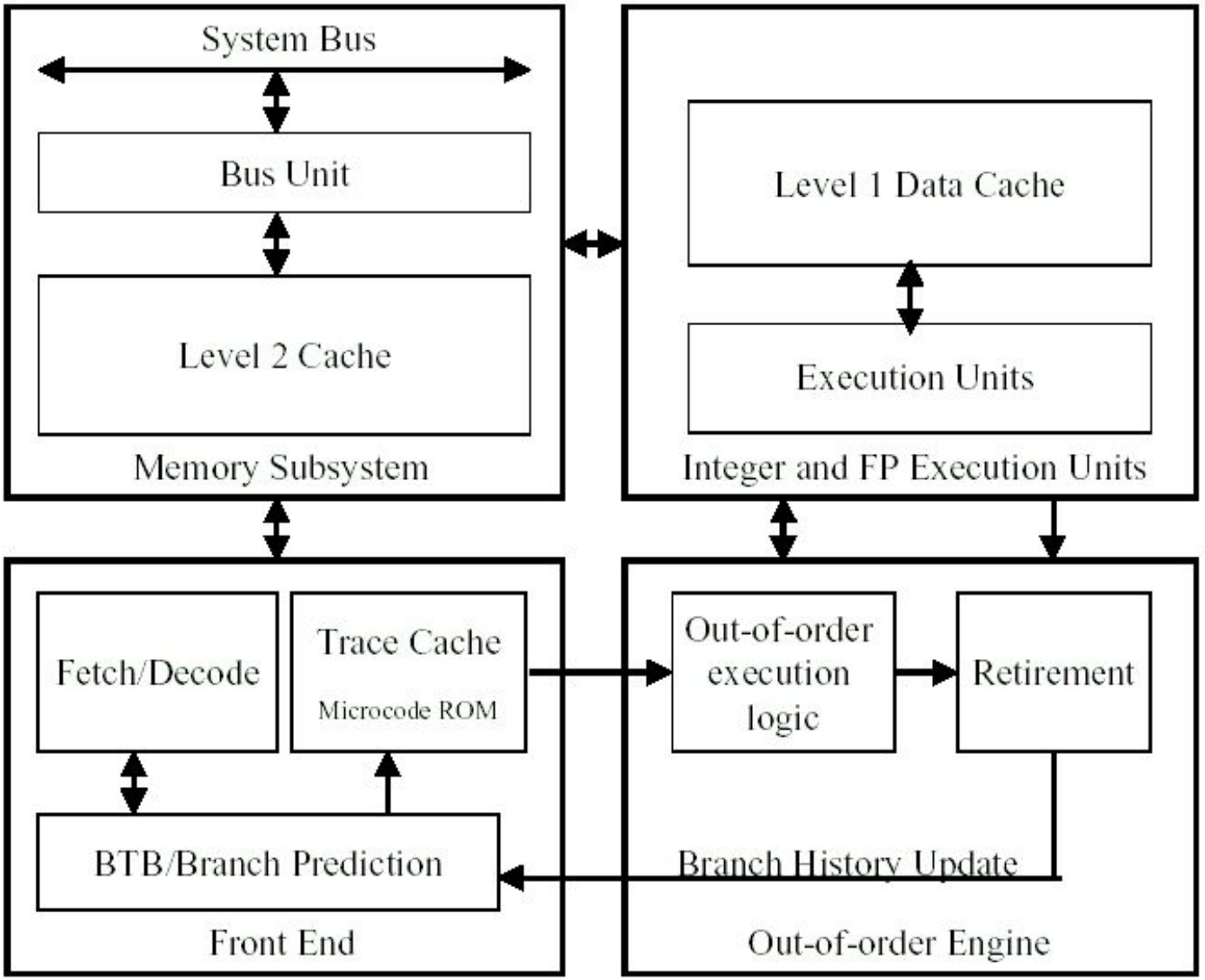
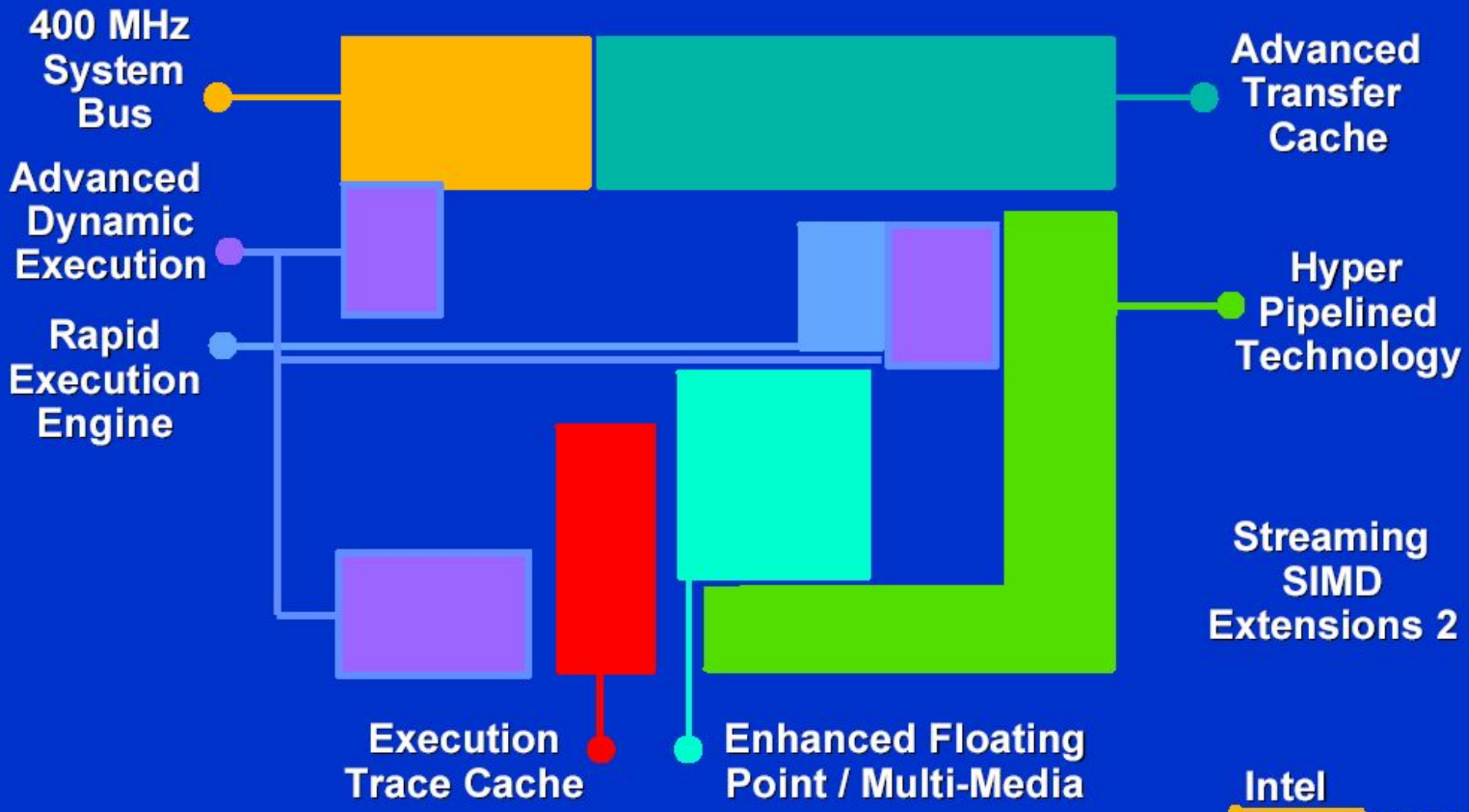
The Microarchitecture of the Pentium 4 Processor

Overview


System Bus
Bus Unit
Level 2 Cache
Memory Subsystem
Level 1 Data Cache
Execution Units
Integer and FP Execution Units
Fetch/Decode
Trace Cache
Microcode ROM
BTB/Branch Prediction
Front End
Out-of-order execution logic
Retirement
Branch History Update
Out-of-order Engine

Intel® NetBurst™ Micro-architecture

Developer Forum Fall 2000



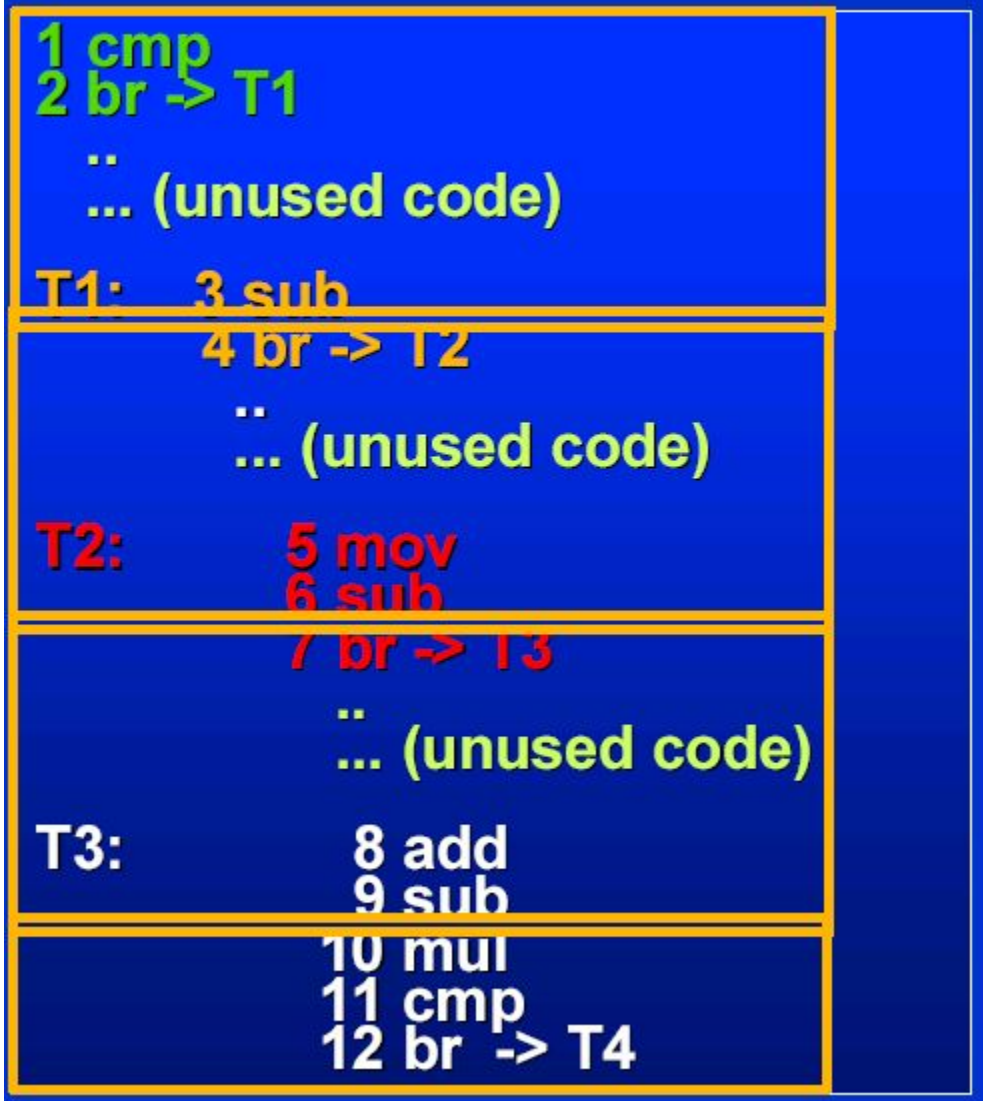
Intel PDX

Execution Trace Cache

- Advanced L1 instruction cache
  - Caches “decoded” IA-32 instructions (uops)
- Removes decoder pipeline latency
- Capacity is ~12K uOps
- Integrates branches into single line
  - Follows predicted path of program execution

Execution Trace Cache feeds fast engine

Execution Trace Cache



Trace Cache Delivery

1	cmp	2	br T1	3 T1:	sub
4	br T2	5	mov	6	sub
7	br T3	8	T3:add	9	sub
10	mul	11	cmp	12	br T4

Branch Prediction

- Accurate branch prediction is key to enabling longer pipelines
- Dramatic improvement over P6 branch predictor:
  - 8x the size (4K)
  - Eliminated 1/3 of the mispredictions
- Proven to be better than *all* other publicly disclosed predictors
  - (g-share, hybrid, etc)

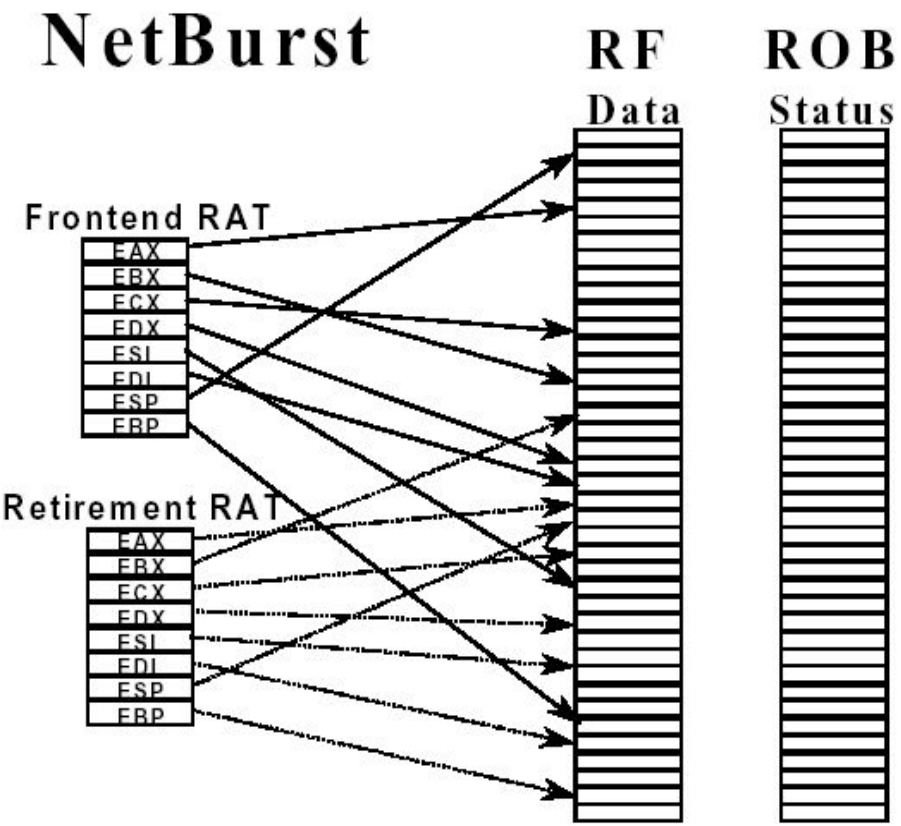
Advanced Dynamic Execution

- Extends basic features found in P6 core
- Very deep speculative execution
  - 126 instructions in flight (3x P6)
  - 48 loads (3x P6) and 24 stores (2x P6)
- Provides larger window of visibility
  - Better use of execution resources

Deep Speculation Improves Parallelism

Register Allocation and Renaming

- The allocator allocates a reorder buffer (ROB) entry
  - Track the completion status of an μop
- Remember the most current version of each register in Register Alias Table (RAT)
  - A new instruction knows where to get the correct current instance
- Allocate the ROB and Register File (RF) separately
- On retirement, no result data values are actually moved from one physical structure to another



µop Scheduling

- Determine when an µop is ready to execute by tracking its input register operand
- Several individual µop schedulers
  - schedules different types of µops for various execution units
  - are tied to four different dispatch ports

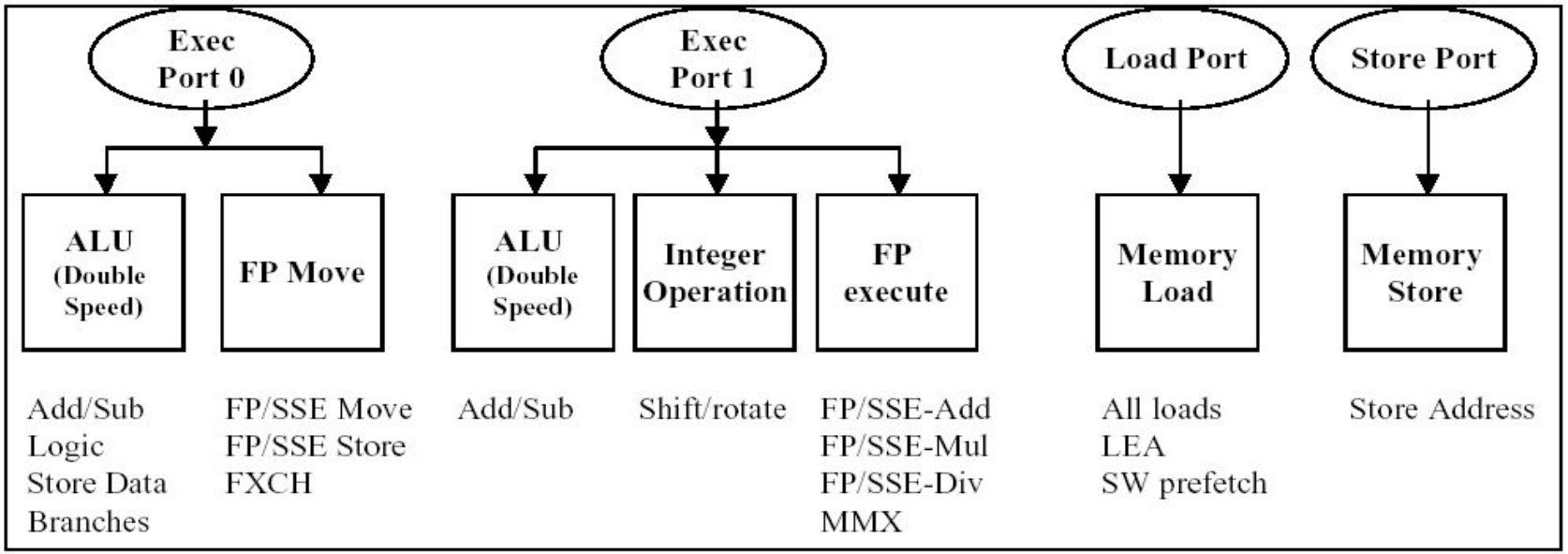


Figure 6: Dispatch ports in the Pentium® 4 processor

Pipeline

- Long pipeline with short steps
  - 20 stages
  - Higher clock rate and less logic per stage
  - Lots of in-flight instructions
  - Long store queue
  - Misprediction cost increases
  - 50% frequency gives 30% performance

Execution Core

- Different clock speeds
  - High speed ALU (2x main clock, 3GHz)
  - Main clock speed, ½ speed, bus speed
- Different ALU types
  - Fast for common cases (60-70% of µops)
  - Slower for more complex operations
- Fast ALU uses staggered computation
- Each RF has bypass network to forward computed data without write first

Caching

- Trace cache as L1 instruction cache
- Small 'n' fast L1 data cache (2-clock latency), large L2 cache with prefetching for high bandwidth streaming
- Long pipeline and early dispatch
  - Assume hit in L1 cache
  - Dependent instructions might use bad data
  - Detect and replay incorrect ones

Instruction Set

- IA-32 instructions decoded into µopts
  - µopts are cached in trace cache to avoid repeated decoding
- Microcode ROM for complex IA-32 ins.
  - string move, fault and interrupt handling
- ISA extensions (MMX/SSE/SSE2)
  - SIMD instructions
  - SSE2: 128-bit packed data of types:
    - byte, word, double word, quad word, double precision floating point

Pros and Cons

- \+ High performance
- \+ High accuracy of branch predictor
- – Long pipeline = large penalty for branch mispredictions
- – Overlapped store-to-load forwarding not possible
- – Optimizations for older cores can give negative effect

Questions

- What is the reason to separate ROB and RF entries in the P4?
- Is a long pipeline and higher clock a clear win?
- Do we know anything about the secret branch predictor and its strategies?
- How much overhead comes with the IA-32 decoder?
- Architecture optimized code defeats one SW distribution for all. Cost for SW companies?
- How efficient is it to double or quad-clock buses? Any problems doing so?
- In what ways is the Trace Cache BTB different from the Front-End BTB?
- Does the deep pipeline approach make the high clock rates somewhat artificial?