

## More on Pipelining and Pipelines in Real Machines

CS 333  
Fall 2006

## Main Ideas

- Data Hazards
  - RAW
  - WAR
  - WAW
- More pipeline stall reduction techniques
  - Branch prediction
    - » static
    - » dynamic
      - bimodal branch prediction
- Multiple-issue
  - static
  - dynamic (superscalar)
- Pipelines in real processors - Pentium 4, AMD Athlon (next class)

## Pipelining

- Exploits **parallelism** between instructions
  - **Instruction-level parallelism (ILP)**
  - Gain performance
- Pipelining overlaps more instructions
  - Subdivides instruction execution into subtasks

## Data Hazards

- Read after Write (RAW)
  - add **r0**, r1, r2
  - sub r4, r3, **r0**
- Write after Write (WAW)
  - add **r0**, r1, r2
  - sub **r0**, r4, r5
- Write after Read (WAR)
  - add r2, r1, **r0**
  - sub **r0**, r3, r4

**WAW and WAR**  
Only possible if instructions can be executed in parallel or out-of-order  
register renaming

## Control Hazards

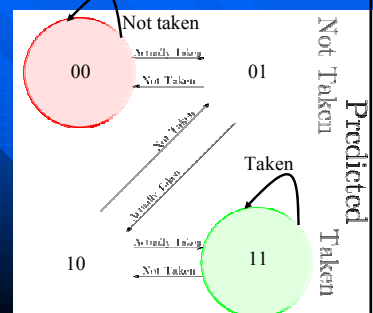
- Resolution of branch condition does not occur until **MEM** stage
  - Causes stalls
- Approaches to handling:
  - Stall (pipeline bubbles)
  - Delayed branch slot
  - Static prediction
    - » Does not rely on runtime information
    - » various approaches:
      - always taken (or, always not taken)
      - backwards taken, forwards not taken (loops)

If branch is mispredicted, need to undo all instructions in the pipeline along the incorrect path (need extra resources)

## Dynamic Branch Prediction

- Use execution behavior to make a prediction

- bimodal branch prediction**
- 2 bit counters
  - use recent branch behavior to predict future



## Multiple Issue

- Another approach to increasing instruction level parallelism
- Launch many instructions in each stage
  - Need more resources (instead of 1 washer and dryer, may need multiple washers and dryers, AND more people to help fold clothes)
  - Disadvantage
    - » Need additional resources to keep all resources busy

## Multiple Issue

- Additional tasks
  - Packaging instructions into **issue slots**
    - » **how many instructions?**
    - » **which instructions?**
  - Handling data and control hazards

## Static Multiple Issue

- Uses **compiler** to determine:
    - **issue packets**
    - handle data and control hazards
      - » **register renaming**
        - removes WAR and WAW dependences
- ```
add r0, r1, r2
sub r0, r4, r5
Rename r0 in sub to r3
```

## Dynamic Multiple Issue

- aka **superscalar**
  - Dynamically chooses instructions to execute next, possibly reorders them (to avoid stalls)
- **Reservation stations**
  - Holds operands and operation information
- **Reorder buffer**
  - Stores results to be written to register file
  - Writes results to register file **in-order**

## Real Processors

Intel Pentium 4 and AMD Athlon

## Main Points

- Difference in approaches
  - Intel Pentium 4
    - » Increase pipeline depth to handle more instructions
    - » Increase clock frequency
  - AMD Athlon
    - » Try to balance performance (IPC) and operating frequency
- Big Picture Similarities
  - Both use
    - » pipelining
    - » dynamic multiple issue (superscalar)
    - » out-of-order execution
    - » branch prediction
  - **Both execute IA-32 instructions**

## Intel Pentium 4 Processor



## Outline

- Pentium 4 – 20 stages
  - Instruction Set Architecture
  - Instruction Stream
- Pentium 4 revisions
  - Northwood (1/2002) – 21 stages
  - Prescott (2/2004) – 31 stages

## Introduction

- Intel Pentium 4 processor
  - Latest IA-32 processor equipped with a full set of IA-32 SIMD (single-instruction multiple data) operations

## IA-32

- Intel architecture 32-bit (IA-32)
  - 80386 instruction set (1985)
  - CISC, 32-bit addresses
- Registers
  - Eight 32-bit registers
  - Eight FP stack registers
  - 6 segment registers

## IA-32 (cont'd)

- **Addressing modes**
  - Register indirect ( $\text{mem}[\text{reg}]$ )
  - Base + displacement ( $\text{mem}[\text{reg} + \text{const}]$ )
  - Base + scaled index ( $\text{mem}[\text{reg} + (2^{\text{scale}} \times \text{index})]$ )
  - Base + scaled index + displacement ( $\text{mem}[\text{reg} + (2^{\text{scale}} \times \text{index}) + \text{displacement}]$ )
- SIMD instruction sets
  - MMX (Pentium II)
    - » Eight 64-bit MMX registers, integer ops only
  - SSE (Streaming SIMD Extension, Pentium III)
    - » Eight 128-bit registers

## Pentium III vs. Pentium 4 Pipeline

| Basic 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 |

| Basic Pentium 4 Processor Misprediction Pipeline |         |    |       |       |       |        |     |     |     |     |      |      |    |    |    |      |       |       |    |
|--------------------------------------------------|---------|----|-------|-------|-------|--------|-----|-----|-----|-----|------|------|----|----|----|------|-------|-------|----|
| 1                                                | 2       | 3  | 4     | 5     | 6     | 7      | 8   | 9   | 10  | 11  | 12   | 13   | 14 | 15 | 16 | 17   | 18    | 19    | 20 |
| TC                                               | Next IP | TC | Fetch | Drive | Alloc | Rename | Que | Sch | Sch | Sch | Disp | Disp | RF | RF | Ex | Flgs | Br Ck | Drive |    |



## What is a Trace Cache?

I1 ...  
I2 br r2, L1  
I3 ...  
I4 ...  
I5 ...  
L1: I6  
I7 ...

- Traditional instruction cache

|    |    |    |    |
|----|----|----|----|
| I1 | I2 | I3 | I4 |
|----|----|----|----|

- Trace cache

|    |    |    |    |
|----|----|----|----|
| I1 | I2 | I6 | I7 |
|----|----|----|----|

## Pentium 4 Trace Cache

- Has its own branch predictor that directs where instruction fetching needs to go next in the Trace Cache
- Removes
  - Decoding costs on frequently decoded instructions
  - Extra latency to decode instructions upon branch mispredictions

## Microcode ROM

- Used for complex IA-32 instructions (> 4  $\mu$ ops), such as string move, and for fault and interrupt handling
- When a complex instruction is encountered, the Trace Cache jumps into the microcode ROM which then issues the  $\mu$ ops
- After the microcode ROM finishes, the front end of the machine resumes fetching  $\mu$ ops from the Trace Cache

## Branch Prediction

- Predicts ALL near branches
  - Includes conditional branches, unconditional calls and returns, and indirect branches
- Does not predict far transfers
  - Includes far calls, irets, and software interrupts

## Branch Prediction

- Dynamically predict the direction and target of branches based on PC using **branch target buffer (BTB)**
- If no dynamic prediction is available, **statically predict**
  - Taken for backwards looping branches
  - Not taken for forward branches
- Traces are built across predicted branches to avoid branch penalties

## Branch Target Buffer

- Stores branch target addresses
- Uses a branch history table and a branch target buffer to predict
- Updating occurs when branch is retired

## Return Address Stack

- 16 entries
- Predicts return addresses for procedure calls
- Allows branches and their targets to coexist in a single cache line
  - Increases parallelism since decode bandwidth is not wasted

## Branch Hints

- P4 permits software to provide hints to the branch prediction and trace formation hardware to enhance performance
- Take the forms of prefixes to conditional branch instructions
- Used only at trace build time and have no effect on already built traces

## Out-of-Order Execution

- Designed to optimize performance by handling the most common operations in the most common context as fast as possible
- 126  $\mu$ ops can in flight at once
  - Up to 48 loads / 24 stores

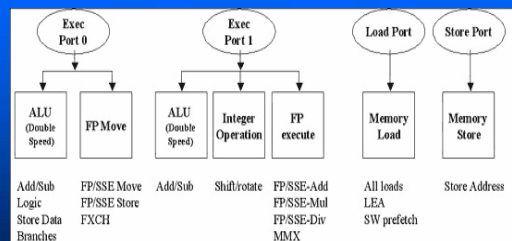
## Issue

- Instructions are fetched and decoded by translation engine
- Translation engine builds instructions into sequences of  $\mu$ ops
- Stores  $\mu$ ops to trace cache
- Trace cache can issue 3  $\mu$ ops per cycle

## Execution

- Can dispatch up to 6  $\mu$ ops per cycle
- Exceeds trace cache and retirement  $\mu$ op bandwidth
  - Allows for greater flexibility in issuing  $\mu$ ops to different execution units

## Execution Units



## Double-pumped ALUs

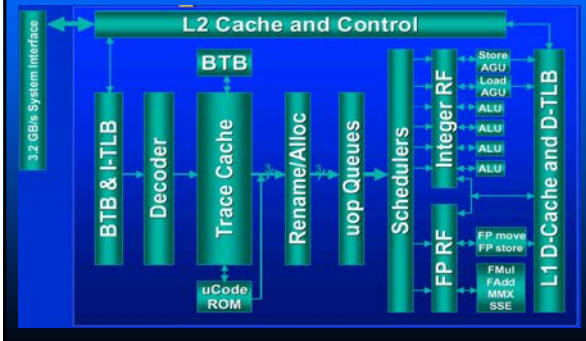
- ALU executes an operation on both rising and falling edges of clock cycle



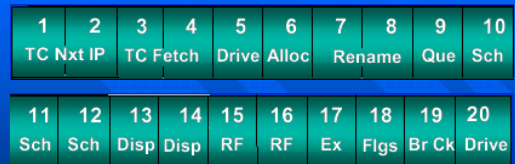
## Retirement

- Can retire 3  $\mu$ ops per cycle
- Precise exceptions
- Reorder buffer to organize completed  $\mu$ ops
- Also keeps track of branches and sends updated branch information to the BTB

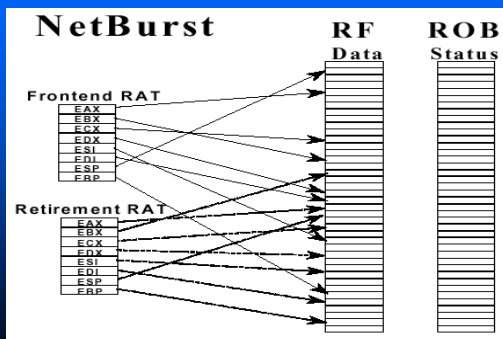
## Execution Pipeline



## Execution Pipeline



## Register Renaming

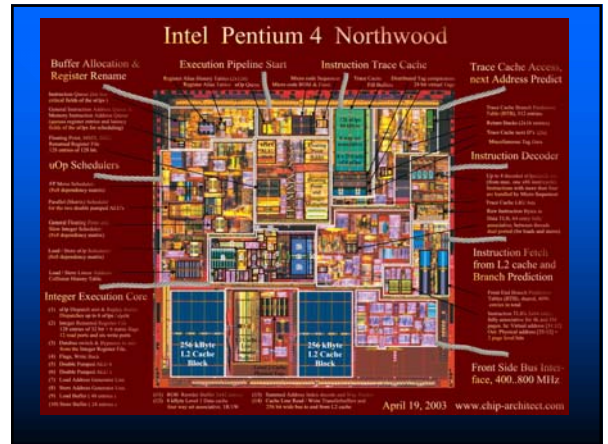


## Register Renaming (2)

- 8-entry architectural register file
- 128-entry physical register file
- 2 RAT
  - Frontend RAT and Retirement RAT
- Data does not need to be copied between register files when the instruction retires

## Loses consistently to AMD

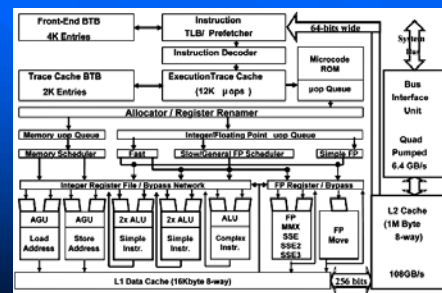
- In terms of performance, the Pentium 4 is as slow or slower than existing Pentium III and AMD Athlon processors
- In terms of price, an entry level Pentium 4 sells for about double the cost of a similar Pentium III or AMD Athlon-based system
- 1.5GHz clock rate is more hype than substance



## Northwood

- 1/2002
- Differences from Willamette
  - Socket 478
  - **21 stage pipeline**
  - 512 KB L2 cache
  - 2.0 GHz, 2.2 GHz clock frequency
  - 0.13μ fabrication process (130 nm)
    - » 55 million transistors

## Prescott



## Prescott

- 2/2004
- Differences
  - **31 stage pipeline!**
  - 1MB L2 cache
  - 3.8 GHz clock frequency
  - 0.9μ fabrication process
  - SSE3