

Pipelining in Real Processors

AMD Athlon

CS 333
Fall 2006

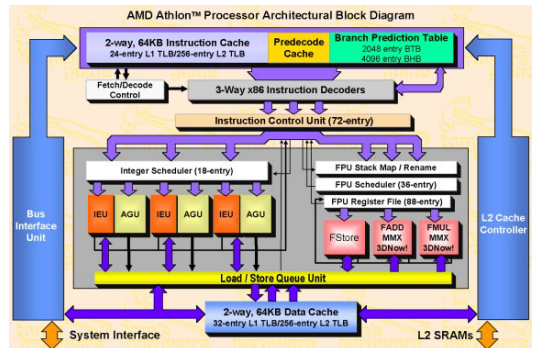
Reminders

- Homework #4, due this Friday
- Midterm, Friday, November 3, 2006, in-class (date is changed)

Main Points

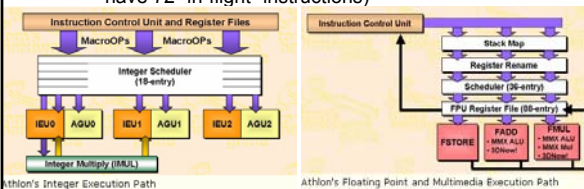
- AMD Athlon pipeline compared to Intel Pentium 4
- Pipeline design process
 - Steps
 - Handling hazards (RAW, WAW, WAR)
 - Register renaming
 - architectural vs. physical registers

Athlon Processor Architecture



Architecture Summary

- Instruction Control Unit
 - Holds 72 MOps Before Assignment (MOp = IA-32 instruction, therefore Athlon can have 72 "in-flight" instructions)

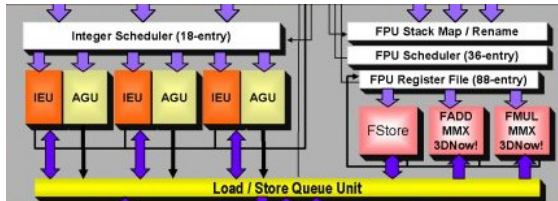


Micro-OPs / Macro-OPs

- Athlon has 3 parallel IA-32 instruction decoders translate into a Macro-Op of 72-entry ICU
 - Uses 2 pipelines (Intel uses 1)
 - Decoding common instructions (direct path)
 - Decoding complex IA-32 instructions (vector path)
 - Integer Scheduler is fed and holds max 15 M-Ops, representing 30 at a time
 - Leads to 3 parallel integer execution units

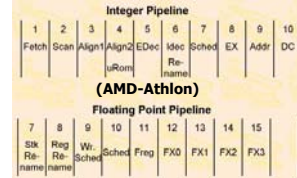
ALU's

- AMD Athlon – 9 functional units (vs. Pentium 4's 5 functional units)



Pipelining Differences

- Determining the length
 - Execution rate of pipeline (ALU)
 - Degree of Parallelism
- AMD Athlon
 - Integer pipeline – 10 stages
 - Floating point pipeline - 15 stages



Branch Prediction

- Approximate correct branch predictions
 - AMD Athlon – 95% correct predictions
 - Misprediction penalty - 9 cycles
 - Intel Pentium 4 – 95% correct predictions
 - Misprediction penalty – at least 19 cycles (shortest P4 pipeline)

Branch Misprediction Penalty

- Example: 1000 instructions, 10% branches, how much penalty paid for Athlon over the course of the program? Pentium 4?
 - 10% of 1000 = 100 instructions are branches
 - 95% correct = 95 correctly predicted branches, 5 incorrectly predicted branches
 - Athlon – $5 * 9$ cycle penalty = 45 cycles
 - Pentium 4 – $5 * 19$ cycle penalty = 145 cycles

Designing a Pipeline

Overview

1. Classify instructions by register transfer behavior
2. Determine the number of pipeline stages
3. Partition instruction behavior into the subtasks from step 2
4. Partition op codes into groups with similar control signals
5. Add hardware and control to support pipelining

Step 1: Classify Instructions

- How and where does data flow during instruction execution?
- SRC
 - load/store
 - ALU
 - branch

Step 2: Determine # Pipeline Stages (SRC)

1. Instruction fetch (IF)
2. Instruction decode and operand fetch (ID)
3. ALU operations (EXE)
4. Data memory access (MEM)
5. Register write (WB)

Step 3: Partition Each Instruction into Pipeline Subtasks

- Example: ld, ldr, la, lar
 - ld ra, c2
 - ld ra, c2(rb)
 - ldr ra, c1

 1. IFetch – $IR \leftarrow PC + 4$
 2. IDecode – decode c1, c2 fields
 3. EXE – Calculate relative or displacement
 4. MEM – Copy data from memory to register
 5. WB – Write memory value to the architectural register file

Go through this process for each instruction class

Step 4: Partition Opcodes into Similar Groups

- See Fig. 5.3
 - Examples:
 - branch := br v brl :
 - load := ld v ldr :
 - loadrel := la v lar :
 - regwrite := load v ldr v brl v alu :
 - dsp := ld v st v la :

Step 5: Add Needed Hardware and Control Support

- Modifications to Memory
 - IF and MEM stage may need to access data at the same time
 - Solution: partition instruction memory and data memory
- Register File.
 - 3-port register file (2 reads, 1 write)
- Buses and Datapath
 - Need to pass intermediate information from stage to stage (pipeline registers)
 - Multiplexers
- Specialized Hardware

Hazard Detection (Software)

- Software (compiler) – compile time
 - Code rearrangement or insert nops (pipeline bubbles)
 - Disadvantages
 - Makes compiler development slower, more expensive
 - Without hardware detection – forwarding not possible

Hazard Detection (Hardware)

- Hardware – run time
 - Stalls
 - With 5-stage pipeline, without data forwarding, dependent instructions must be 4 instructions apart (operand fetch)
 - “Stallee” must be held in ID until hazard is resolved
 - Detection mechanism must detect dependences of distance 1-3 instructions apart
 - “Staller” must make forward progress

Example: Hardware Hazard Detection in RTN

- $alu3 \wedge alu2 \wedge ((ra3 = rb2) \vee ((ra3 = rc2) \wedge \neg imm2)) \rightarrow (pause2: pause1: op3 \leftarrow 0):$
- See Eq. 5.1 – 5.3, Also read hazard detection by hardware, resolved by data forwarding

WAW and WAR Hazard Resolution

- Register renaming
 - Architectural register file
 - Programmer visible registers (ISA)
 - Physical register file
 - May be larger than architectural register file. Allows renaming of registers with WAW and WAR dependences.

Main Points

- AMD Athlon pipeline compared to Intel Pentium 4
- Pipeline design process
 - Steps
 - Handling hazards (RAW, WAW, WAR)
 - Register renaming
 - architectural vs. physical registers

Next Class

- Chapter 7, Memory System Design
- Homework, due Friday in class
 - Problems 5.3, 5.4, 5.8, 5.9, 5.15