Computer Science 210

Computer Systems 1

2007 Semester 1

Lecture Notes Part 2

1. Floating Point

Lecture 12
18 May 07

James Goodman

Floating Point in the Alpha

- The Alpha has 32 floating-point registers $f0-$f31
  - $f31 is always zero
- Instructions are same format as integer instructions
  - Floating-point registers are implied (mostly)
  - No literals allowed
- Two levels of precision
  - S: "single-precision" 32-bit IEEE format
  - T: "double-precision" 64-bit IEEE format

Arithmetic Instructions

- Arithmetic instructions have separate "operate" opcode
- Instructions are either single (S) or double (T) type
- Operations
  - ADD
  - SUB
  - MUL
  - DIV
  - SQRT

Load/Store Instructions

- Load: LDS, LDT
  - Address comes from integer register
  - Same as ldl, ldq but target is FP register
- Store: STS, STT
  - Address comes from integer register
  - Same as stl, stq but source is FP register

Control Instructions

- Branch on condition
  - Same as integer, but using FP reg (6 cases)
  - Additional problem: NAN
- Compare two FP regs
  - EQ, GE, GT, LE, LT, NE
  - Result set zero/non-zero value in FP reg
  - Also "unordered"
- Conditional move instructions
  - Test FP register *a* against zero: EQ, GE, GT, LE, LT, NE
  - If true, copy register *b* into register *c*

Other Instructions

- Move: copy between I register and FP register
  - FtoIS,FtoIT,ItoFS,ItoFT
  - No format change
- Convert between floating point/Integer
  - convert in-place in FP registers
  - S <=> T; T <=> Q ; Q => S (no S=> Q)
  - Longword <=> Quadword
- CPYS: Copy sign bit to destination (merge)
- CPYSN: Copy and invert sign bit to destination
- CPYSE: Copy sign and exponent to destination

Summary

- Alpha has excellent support for floating point computations
- Fully support for IEEE Floating Point Standard
- Observations
  - Most programs use little or no floating point, don't care about performance
  - Programs that use floating point tend to be dominated by floating point operations, care greatly about performance

Computer Science 210

Computer Systems 1

2007 Semester 1

Lecture Notes Part 2

2. Performance: the Big Picture

Lecture 12
18 May 07

James Goodman

Performance

Question:

"How long does it take to execute an instruction?"

Answer:

"It depends"

Modern Processors

Are pipelined (Assembly-line process)

One stage per clock cycle

- Stage 1: Fetch instruction
- Stage 2: Decode instruction
- Stage 3: Fetch operands
- Stage 4: Perform operation
- Stage 5: Put away result

Problems

- Need to fetch multiple operands in stage 3
- Sometimes operand is not ready
  - Producing instruction has not completed
  - Must *stall* pipeline
- For load instruction, operation is to memory
  - Memory is slow
- What happens on a conditional branch???

Cache Memory

- Small memory runs at CPU speed
- Contains (redundant) subset of what is in memory
- Cache memory has two parts
  - Data
  - Address of the data
- Data is in cache (a hit)
  - Data is supplied in one clock cycle
- Data is not in cache (miss)
  - Must stall for many cycles while data is fetched
- Exploit *temporal & spatial locality*

Conditional Branch

- Problem: what instruction is next?
- Don't even know this is a problem until instruction is decoded
  - Already fetching next instruction (OK, but maybe wasted)
- Perform test
  - If true, must flush pipe, fetch new instruction and restart
- Starting up may be slow because fetching instruction is "surprise"

Two Sources of Disruption

- Cache miss on load (or store)
  - Must wait while data is fetched from memory
- Branch taken; instruction cache miss
  - Must wait while data is fetched from memory
- Observation:

Memory performance is critical!

Tricks to Tolerate Cache Misses

- Further exploit temporal & spatial locality
  - Anticipate memory requests
  - Remember what happened last time
  - Look for patterns
- Execute instructions out of order
  - Bypass instructions waiting for memory
  - Find instructions with available operands and execute
- Speculate!
  - Fetch instructions and "execute" them before it is known that they should be executed, but don't "commit"
  - Guess the value of an operand
- Branches: Predict whether taken
  - Speculate down most likely path

Summary

Performance optimized by making the common case fast

- The uncommon case merely must execute correctly
- This approach has been highly successful, but results in large variance in execution time

Memory plays a critical role in performance

- Many programs spend *more time waiting for memory than executing instructions*

Implications for programmers

- Placement of data is critical
  - Temporal & spatial locality can be exploited
- Branches are expensive
  - Unpredictable branches are especially difficult

Find this topic interesting?

Consider taking CS 313 Computer Organization!

- Preparation needed for 313: Physics 243
- 2008: Physics 243 becomes CS 143 (no prerequisites)