CS184b: Computer Architecture [Single Threaded Architecture: abstractions, quantification, and optimizations]

Day16: March 8, 2001

Review and Retrospection

Today

- This Quarter
  - What is Architecture?
  - Why?
  - Optimizations w/in Model
  - Themes
- Next Quarter
  - beyond a single thread of control
- Admin: final

CS184 Sequence

- A - structure and organization
  - raw components, building blocks
  - design space
- B - single threaded architecture
  - emphasis on abstractions and optimizations including quantification
- C - multithreaded architecture

"Architecture"

- "attributes of a system as seen by the programmer"
- "conceptual structure and functional behavior"
- Defines the visible interface between the hardware and software
- Defines the semantics of the program (machine code)

Architecture distinguished from Implementation

- IA32 architecture vs.
  - 80486DX2, AMD K5, Intel Pentium-II-700
- VAX architectures vs.
  - 11/750, 11/780, uVax-II
- PowerPC vs.
  - PPC 601, 604, 630 …
- Alpha vs.
  - EV4, 21164, 21264, …
- Admits to many different implementations of single architecture

Value?

- **Abstraction**
- Effort
  - human brain time is key bottleneck/scarce resource in exploiting modern computing technology
- Economics
- Software Distribution
- capture and package meaning
  - pragmatic of failure of software engineering

Fixed Points

- Must “fix” the interface
- Trick is picking what to expose in the interface and fix, and what to hide
- What are the “fixed points?”
  - how you describe the computation
  - primitive operations the machine understands
  - primitive data types
  - interface to memory, I/O
  - interface to system routines?

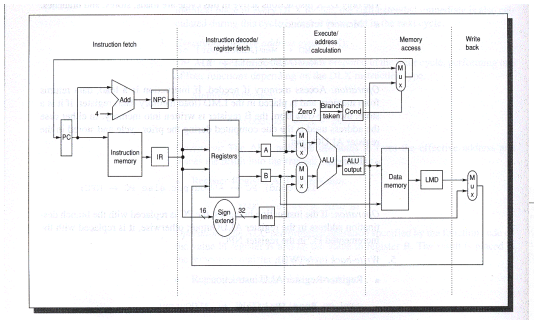
Abstract Away?

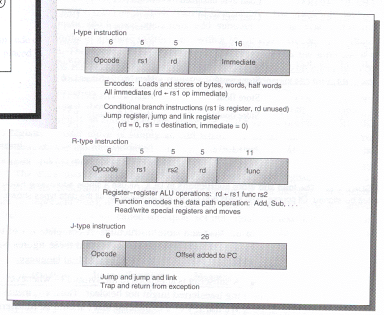
- Specific sizes
  - what fits in on-chip memory
  - available memory (to some extent)
  - number of peripherals
  - 0, 1, infinity
- Timing
  - individual operations
  - resources (e.g. memory)

Optimizations

- Simple Sequential Model
- Pipeline
  - hazards, interlocking
- Multiple Instructions / Cycle
  - out of order completion, issue
  - renaming, scoreboarding
- Branch prediction, predication
- Speculation
- Memory Optimization
- Translation to different underlying org.

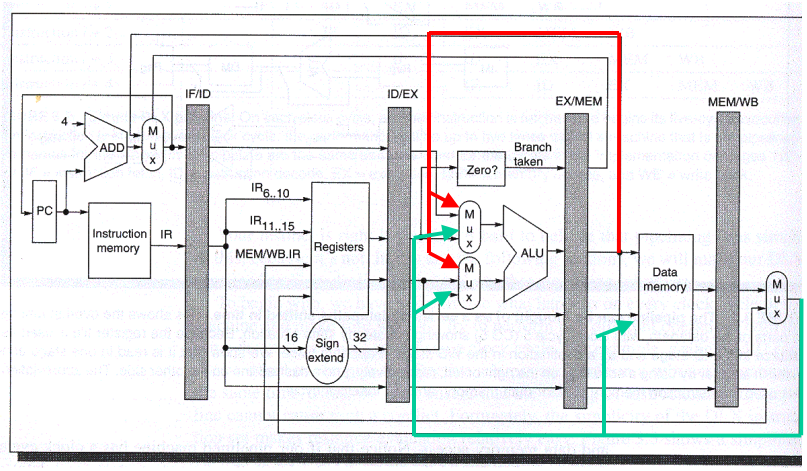
Simple Seq. Model





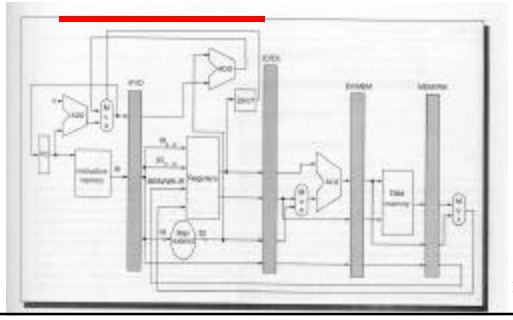
Do one instruction completely; then do next instruction.

Pipeline

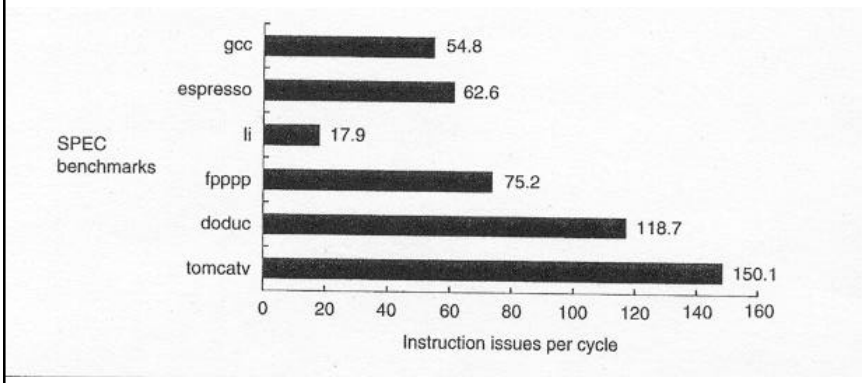


Pipelining

- Watch Data Hazards: bypass stall
- Watch Control Hazards:
  - minimize cycle, predict, flush
- Watch Exceptions: in-order retire to state

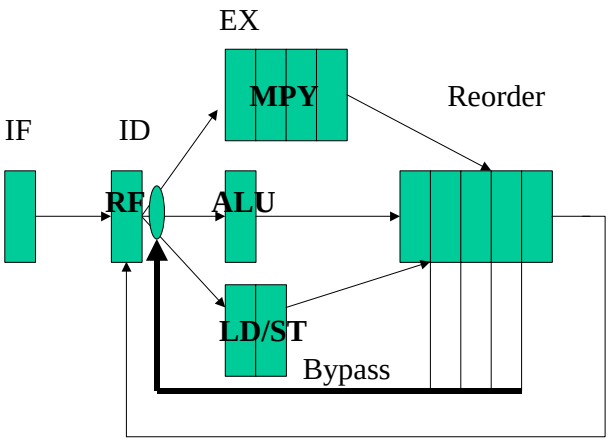


ILP (available)



Hennessy and Patterson 4.38

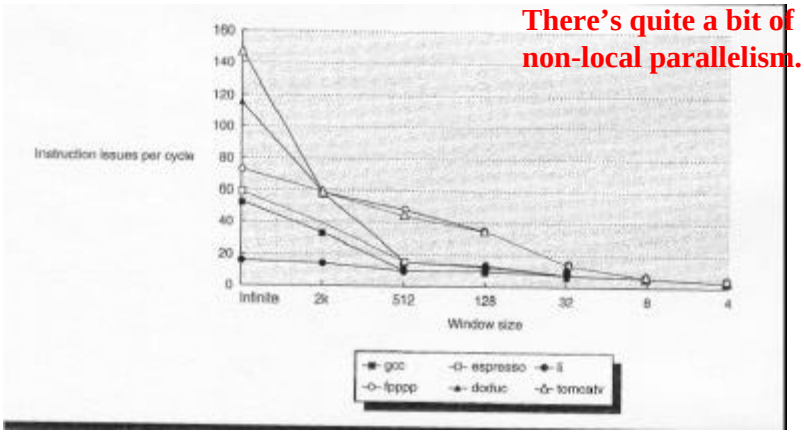
Supporting ILP



- Rename
- Scoreboard
- Reorder

ILP Challenges: *e.g.* Window Size

There's quite a bit of non-local parallelism.



[Hennessy and Patterson 4.39]

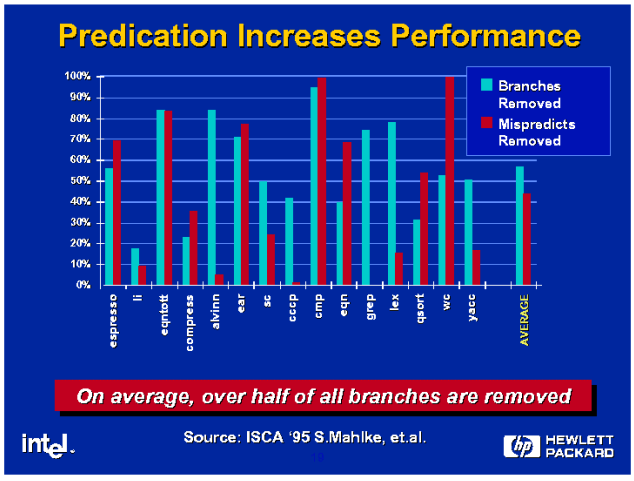
Branching

- Makes stalls expensive
  - potential ILP limiter
  - e.g.
    - with 7 instructions / branch
    - issue 7 instructions, hit branch, stall for instructions to complete...
- Fisher: Instructions/mispredict: 40-160
  - even with different data sets
- Predication: avoid losing trace on small, unpredictable branches
  - can be better to do both than branch wrong

Two Control Options

- Local control = Predication
  - unify choices
    - build all options into spatial compute structure and select operation
- Instruction selection = Branching
  - provide a different instruction (instruction sequence) for each option
  - selection occurs when chose which instruction(s) to issue

Predication: Quantification



Memory System

- Motivation for Caching
  - fast memories small
  - large memories slow
  - need large memories
  - speed of small w/ capacity/density of large
- Programs need frequent memory access
  - e.g. 20% load operations
  - fetch required for every instruction
- Memory is the performance bottleneck?
  - Programs run slow?

Multi-Level Numbers

- L1, 1ns, 4KB, 10% miss
- L2, 5ns, 128KB, 1% miss
- Main, 50ns
- No Cache CPI=Base+0.3*50=Base+15
- L1 only CPI=Base+0.3*0.1*50=Base +1.5
- L2 only CPI=Base+0.3*(0.99*4+0.01*50) =Base+1.7
- L1/L2=Base+(0.3*0.1*5 + 0.01*50) =Base+0.65

Themes for Quarter

- Recurring
  - "cached" answers and change
  - merit analysis (cost/performance)
  - dominant/bottleneck resource requirements
  - structure/common case
    - common case fast
    - fast case common
    - correct in every case
  - exploit freedom in application
  - virtualization

Themes for Quarter

- New/new focus
  - measurement
  - abstractions/semantics
  - abstractions 0, 1, infinity
  - dynamic data/event handling (vs. static)
  - binding times
    - compile-time vs. run-time
    - …now load time (JIT), during execution
  - predictability (avg. vs. worst case)
    - feedback
  - translation

More Themes

- Primitives
- Simplicity
  - of model
  - of implementation

Model and Quantitative

- Have a model which defines the semantics
  - correct behavior/operation
- Any implementation which provides same semantics is acceptable
- Creates freedom to optimize and quantify
  - make changes / hypothesized optimization
  - measure results
  - benchmarks relative to model
  - simple to change implementation below visible fixed point

Equations for Opt. And Understanding

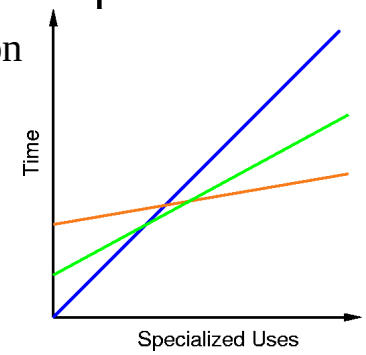
- Time= (Instructions)(Cycles/Instruction)(Cycle Time)
- CPI = 1 + P_{stall} (Stall Cycles) + $P_{br\text{-}mispredict}$ (Branch Penalty)
- CPI = Base CPI + Refs/Instr (Miss Rate)(Miss Latency)

Binding Time

- Hoist code out of heavy use region if at all possible to do earlier
  - loop invariants out of loops
  - instruction decoding/interpretation out of commonly run regions of code
  - scheduling decisions from runtime
    - to compile time
    - to one-time runtime translation

Binding Time Optimization Prospects

- Translation vs. Emulation
  - $T_{trun} = T_{trans} + nT_{op}$
  - $T_{trns} > T_{em\_op} > T_{op}$



- If compute long enough
  - $nT_{op} >> T_{trans}$
  - $\rightarrow$ amortize out load

Common Case (Structure)

- Simple Instructions (fast)
- Non-conflicting/interlocking Instructions
- Fast/Small memory
  - temporal, spatial locality, TLBs
- Predictable control flow
  - branch predict, exceptions
- Speculation on probable properties
  - trace direction, no aliasing, …
- Compiled/optimized code
  - frequently executed regions

Bottlenecks?

- ALU/functional units?
- Feeding data to operators
  - bandwidth
  - latency
- Parallelism
  - that can expose cheaply
- Figuring out where to go next
  - accuracy
  - decision latency

Freedom in Applications

- DAG Scheduling of operations
  - linearization, trace scheduling
  - increase parallelism
    - hide latency, overlap operations
  - promote locality
- Assignment of data to
  - registers, addresses, pages
  - increase locality, decrease conflicts
- Assign operations to ALUs
  - increase locality, reduce communication

0, 1, Infinity

- Virtual Memory
  - abstract out physical capacity
- Traditional RISC/CISC
  - single operator per cycle (model)
- ILP/EPIC operator exploitation
  - arbitrary number of functional units

- Registers **not** have this property

Feedback

- Discover the common case
  - the common case for this application
  - ...this run of this application
- Branch predictability/control flow
- commonly run pieces of code
  - hotspots
- typical aliasing
- latencies/capacities...

Computer Architecture Parallel to Parthenon Critique

- Are we making:
  - copies in submicron CMOS
  - of copies in early NMOS
  - of copies in discrete TTL
  - of vacuum tube computers?

Should we still build computers the way we did in 1967?

In 1983?

Yesterday's solution becomes today's historical curiosity.

-- Goldratt

Old vs. New?

- Sequential ISA
- virtual memory
- caches
- Multiple functional units, ILP
- Register Renaming

- date back to 60's

- Predication
- Feedback
- EPIC/VLIW
- Speculation
- Binary Translation

- last 10 years

EPIC

- New model
  - not strictly sequential instructions
  - still have sequential semantics
    - control flow
    - memory access

Compiler

- Increasing sophistication
- Increasing reliance upon
  - RISC
    - code motion, scheduling, register assignment
  - VLIW/EPIC
    - trace scheduling, parallelism and likelihood management
  - Speculation
    - more aggressive transformation
  - JIT/Binary Translation
    - takes over as means to provide model

CS184 Sequence

- A - structure and organization
  - raw components, building blocks
  - design space
- B - single threaded architecture
  - emphasis on abstractions and optimizations including quantification
- C - multithreaded architecture

Spring Quarter

- Alternate models
  - single threaded, single memory model
  - ...was a big limitation for us
- Greater parallelism (beyond ILP)
  - data
  - coarse-grained

Next Quarter

- Multithreaded Abstractions, Optimization, and Structures
  - dataflow
  - multithreaded
  - message passing
  - shared memory
  - vector/SIMD (could be single threaded)
  - multiprocessor interconnect
  - defect and fault tolerance (also single thread)

Admin

- Final
  - out Sunday evening
  - due Friday (3/16) 5pm
  - similar to last time
    - open book, notes…
    - work alone
    - no time restrictions beyond get done by F5pm

Big Ideas

- Architectural abstraction
  - define the fixed points
  - stable abstraction to programmer
  - admit to variety of implementation
  - ease adoption/exploitation of new hardware
  - reduce human effort

Big Ideas

- Optimize beneath abstraction
  - exploit freedom of implementation
  - exploit binding time
  - exploit structure and common case
- Identify bottlenecks
- Cost/benefits analysis
  - quantify tradeoffs and options

End of Line

(MCP)