ELE 475 / COS 475
Computer Architecture
Lecture 1: Introduction, Instruction Set Architectures, and Microcode
David Wentzlaff
Department of Electrical Engineering
Princeton University
What is Computer Architecture?
What is Computer Architecture?

Application

Physics
What is Computer Architecture?

Gap too large to bridge in one step
What is Computer Architecture?

In its broadest definition, computer architecture is the design of the abstraction/implementation layers that allow us to execute information processing applications efficiently using manufacturing technologies.

Gap too large to bridge in one step
What is Computer Architecture?

In its broadest definition, computer architecture is the design of the abstraction/implementation layers that allow us to execute information processing applications efficiently using manufacturing technologies.

Gap too large to bridge in one step
Abstractions in Modern Computing Systems

- Application
- Algorithm
- Programming Language
- Operating System/Virtual Machines
- Instruction Set Architecture
- Microarchitecture
- Register-Transfer Level
- Gates
- Circuits
- Devices
- Physics
Abstractions in Modern Computing Systems

- Application
- Algorithm
- Programming Language
- Operating System/Virtual Machines
- Instruction Set Architecture
- Microarchitecture
- Register-Transfer Level
- Gates
- Circuits
- Devices
- Physics

Computer Architecture
ELE 475
Computer Architecture is Constantly Changing

Application Requirements:
• Suggest how to improve architecture
• Provide revenue to fund development

Technology Constraints:
• Restrict what can be done efficiently
• New technologies make new arch possible
Computer Architecture is Constantly Changing

**Application Requirements:**
- Suggest how to improve architecture
- Provide revenue to fund development

**Technology Constraints:**
- Restrict what can be done efficiently
- New technologies make new arch possible
Computers Then...

Computers Now

- Sensor Nets
- Cameras
- Set-top boxes
- Media Players
- Laptops
- Games
- Servers
- Routers
- Robots
- Supercomputers
- Automobiles
- Smartphones
Moore’s Law
The Fifth Paradigm

Major Technology Generations

Electromechanical
Relays
Vacuum Tubes
Bipolar
CMOS
nMOS
pMOS

Calculations per Second per $1,000


[from Kurzweil]
Sequential Processor Performance

Move to multi-processor

From Hennessy and Patterson Ed. 5
Course Administration

Instructor:  Prof. David Wentzlaff (wentzlaf@princeton.edu)
Office: EQuad B228
Office Hours: Mon. & Wed. 3-4pm B228

TA:  Dan Lustig (dlustig@princeton.edu)
Office Hours: Tues. 2-3pm & Thurs. 11am-noon

Lectures:  Monday & Wednesday 1:30-2:50pm EQuad B205

Text:  Computer Architecture: A Quantitative Approach

Hennessey and Patterson, 5th Edition (2012)


John P. Shen and Mikko H. Lipasti

Prerequisite:  ELE 375 & ELE 206

Course Webpage:
http://parallel.princeton.edu/classes/ele475/spring_2012
Course Structure

• Midterm (20%)
• Final Exam (35%)
• Labs (20%)
  – 1 Optional Warm-up lab (ungraded)
  – 2 Design labs (Verilog)
  – 1 Architecture simulation lab
• Design Project (20%)
  – Open ended
  – In small groups
• Class Participation (5%)
• Ungraded Problem Sets (0%)
  – Very useful for exam preparation
Course Content ELE 375

ELE 375

• Basic Pipelined Processor

~100,000 Transistors

Photo of MIPS R2000, Courtesy of MIPS
Course Content ELE 475

Intel Nehalem Processor, Original Core i7, Image courtesy of Intel
Course Content ELE 475

~700,000,000 Transistors

Intel Nehalem Processor, Original Core i7, Image courtesy of Intel
Course Content ELE 475

The Intel Nehalem Processor, Original Core i7, Image courtesy of Intel

~700,000,000 Transistors
Course Content ELE 475

- Instruction Level Parallelism
  - Superscalar
  - Very Long Instruction Word (VLIW)
- Long Pipelines (Pipeline Parallelism)
- Advanced Memory and Caches
- Data Level Parallelism
  - Vector
  - GPU
- Thread Level Parallelism
  - Multithreading
  - Multiprocessor
  - Multicore
  - Manycore

~700,000,000 Transistors

Intel Nehalem Processor, Original Core i7, Image courtesy of Intel
Architecture vs. Microarchitecture

“Architecture”/Instruction Set Architecture:
• Programmer visible state (Memory & Register)
• Operations (Instructions and how they work)
• Execution Semantics (interrupts)
• Input / Output
• Data Types/Sizes

Microarchitecture/Organization:
• Tradeoffs on how to implement ISA for some metric (Speed, Energy, Cost)
• Examples: Pipeline depth, number of pipelines, cache size, silicon area, peak power, execution ordering, bus widths, ALU widths
Software Developments

up to 1955 Libraries of numerical routines
- Floating point operations
- Transcendental functions
- Matrix manipulation, equation solvers, . . .

1955-60 High level Languages - Fortran 1956
Operating Systems -
- Assemblers, Loaders, Linkers, Compilers
- Accounting programs to keep track of usage and charges

Machines required experienced operators

• Most users could not be expected to understand these programs, much less write them
• Machines had to be sold with a lot of resident software
Compatibility Problem at IBM

By early 1960’s, IBM had 4 incompatible lines of computers!

701 ⇒ 7094
650 ⇒ 7074
702 ⇒ 7080
1401 ⇒ 7010

Each system had its own

- Instruction set
- I/O system and Secondary Storage:
  magnetic tapes, drums and disks
- assemblers, compilers, libraries,...
- market niche business, scientific, real time, ...

⇒ IBM 360
IBM 360 : Design Premises

*Amdahl, Blaauw and Brooks, 1964*

- The design must lend itself to *growth and successor machines*
- General method for connecting I/O devices
- Total performance - answers per month rather than bits per microsecond $\Rightarrow$ *programming aids*
- Machine must be capable of *supervising itself* without manual intervention
- Built-in *hardware fault checking* and locating aids to reduce down time
- Simple to assemble systems with redundant I/O devices, memories etc. for *fault tolerance*
- Some problems required floating-point larger than 36 bits
IBM 360: A General-Purpose Register (GPR) Machine

• Processor State
  – 16 General-Purpose 32-bit Registers
    • *may be used as index and base register*
    • *Register 0 has some special properties*
  – 4 Floating Point 64-bit Registers
  – A Program Status Word (PSW)
    • *PC, Condition codes, Control flags*

• A 32-bit machine with 24-bit addresses
  – But no instruction contains a 24-bit address!

• Data Formats
  – 8-bit bytes, 16-bit half-words, 32-bit words, 64-bit double-words

*The IBM 360 is why bytes are 8-bits long today!*
# IBM 360: Initial Implementations

<table>
<thead>
<tr>
<th></th>
<th>Model 30</th>
<th>Model 70</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Storage</strong></td>
<td>8K - 64 KB</td>
<td>256K - 512 KB</td>
</tr>
<tr>
<td><strong>Datapath</strong></td>
<td>8-bit</td>
<td>64-bit</td>
</tr>
<tr>
<td><strong>Circuit Delay</strong></td>
<td>30 nsec/level</td>
<td>5 nsec/level</td>
</tr>
<tr>
<td><strong>Local Store</strong></td>
<td>Main Store</td>
<td>Transistor Registers</td>
</tr>
<tr>
<td><strong>Control Store</strong></td>
<td>Read only 1μsec</td>
<td>Conventional circuits</td>
</tr>
</tbody>
</table>

IBM 360 instruction set architecture (ISA) completely hid the underlying technological differences between various models.

**Milestone:** The first true ISA designed as portable hardware-software interface!

*With minor modifications it still survives today!*
IBM 360: 47 years later...

The zSeries z11 Microprocessor

- 5.2 GHz in IBM 45nm PD-SOI CMOS technology
- 1.4 billion transistors in 512 mm²
- 64-bit virtual addressing
  - original S/360 was 24-bit, and S/370 was 31-bit extension
- Quad-core design
- Three-issue out-of-order superscalar pipeline
- Out-of-order memory accesses
- Redundant datapaths
  - every instruction performed in two parallel datapaths and results compared
- 64KB L1 I-cache, 128KB L1 D-cache on-chip
- 1.5MB private L2 unified cache per core, on-chip
- On-Chip 24MB eDRAM L3 cache
- Scales to 96-core multiprocessor with 768MB of shared L4 eDRAM

[IBM, HotChips, 2010]
Same Architecture
Different Microarchitecture

**AMD Phenom X4**
- X86 Instruction Set
- Quad Core
- 125W
- Decode 3 Instructions/Cycle/Core
- 64KB L1 I Cache, 64KB L1 D Cache
- 512KB L2 Cache
- Out-of-order
- 2.6GHz

**Intel Atom**
- X86 Instruction Set
- Single Core
- 2W
- Decode 2 Instructions/Cycle/Core
- 32KB L1 I Cache, 24KB L1 D Cache
- 512KB L2 Cache
- In-order
- 1.6GHz

Image courtesy of AMD and Intel
Different Architecture
Different Microarchitecture

**AMD Phenom X4**
- X86 Instruction Set
- Quad Core
- 125W
- Decode 3 Instructions/Cycle/Core
- 64KB L1 I Cache, 64KB L1 D Cache
- 512KB L2 Cache
- Out-of-order
- 2.6GHz

**IBM POWER7**
- Power Instruction Set
- Eight Core
- 200W
- Decode 6 Instructions/Cycle/Core
- 32KB L1 I Cache, 32KB L1 D Cache
- 256KB L2 Cache
- Out-of-order
- 4.25GHz

*Image courtesy of AMD and IBM*
Where Do Operands Come from And Where Do Results Go?
Where Do Operands Come from And Where Do Results Go?

Stack

Accumulator

Register-Memory

Register-Register

Number Explicitly Named Operands:

0

1

2 or 3

2 or 3
Stack-Based Instruction Set Architecture (ISA)

- Burrough’s B5000 (1960)
- Burrough’s B6700
- HP 3000
- ICL 2900
- Symbolics 3600

Modern
- Inmos Transputer
- Forth machines
- Java Virtual Machine
- Intel x87 Floating Point Unit
Evaluation of Expressions

\[(a + b \times c) \div (a + d \times c - e)\]

Reverse Polish

```
apush a
push b
push c
multiply
push a
d push c
multiply
push e
subtract
push a
divide
```

Evaluation Stack

```
b \times c
a
```
Evaluation of Expressions

\[(a + b \times c) / (a + d \times c - e)\]

Reverse Polish

\[a \ b \ c \times + \ a \ d \ c \times + \ e \ - \ /\]

Evaluation Stack
Hardware organization of the stack

• Stack is part of the processor state
  \[ \Rightarrow \text{stack must be bounded and small} \]
  \[ \approx \text{number of Registers, not the size of main memory} \]

• Conceptually stack is unbounded
  \[ \Rightarrow \text{a part of the stack is included in the processor state; the rest is kept in the main memory} \]
Stack Operations and Implicit Memory References

• Suppose the top 2 elements of the stack are kept in registers and the rest is kept in the memory.

  Each \textit{push} operation \implies 1 \text{ memory reference}
  \textit{pop} operation \implies 1 \text{ memory reference}

  \textit{No Good!}

• Better performance by keeping the top \textit{N} elements in registers, and memory references are made only when register stack overflows or underflows.

  \textit{Issue - when to Load/Unload registers?}
# Stack Size and Memory References

$$\begin{align*}
\text{program} & \quad \text{stack (size = 2)} & \quad \text{memory refs} \\
push a & \quad R0 & \quad a \\
push b & \quad R0 \ R1 & \quad b \\
push c & \quad R0 \ R1 \ R2 & \quad c, \ ss(a) \\
* & \quad R0 \ R1 & \quad sf(a) \\
+ & \quad R0 & \quad \text{sf(a+b*c)} \\
push a & \quad R0 \ R1 & \quad a \\
push d & \quad R0 \ R1 \ R2 & \quad d, \ ss(a+b*c) \\
push c & \quad R0 \ R1 \ R2 \ R3 & \quad c, \ ss(a) \\
* & \quad R0 \ R1 \ R2 & \quad sf(a) \\
+ & \quad R0 \ R1 & \quad sf(a+b*c) \\
push e & \quad R0 \ R1 \ R2 & \quad e, ss(a+b*c) \\
- & \quad R0 \ R1 & \quad \text{sf(a+b*c)} \\
/ & \quad R0 & \quad \text{sf(a+b*c)}
\end{align*}$$

4 stores, 4 fetches (implicit)
Stack Size and Expression Evaluation

\[ a \times b + c \times d + e - / \]

\[ \text{program} \]
- push a
- push b
- push c
- *
- +
- push a
- push d
- push c
- *
- +
- push e
- -
- /

\[ \text{stack (size = 4)} \]
- R0
- R0 R1
- R0 R1 R2
- R0 R1
- R0
- R0 R1
- R0 R1 R2
- R0 R1 R2 R3
- R0 R1 R2
- R0 R1
- R0 R1
- R0

\[ a \text{ and } c \text{ are “loaded” twice} \]
⇒
not the best use of registers!
Machine Model Summary

C = A + B

- Push A
- Push B
- Add
- Pop C
- Load A
- Add B
- Store C
- Load R1, A
- Add R3, R1, B
- Store R3, C
- Load R1, A
- Load R2, B
- Add R3, R1, R2
- Store R3, C
Classes of Instructions

• Data Transfer
  – LD, ST, MFC1, MTC1, MFC0, MTC0
• ALU
  – ADD, SUB, AND, OR, XOR, MUL, DIV, SLT, LUI
• Control Flow
  – BEQZ, JR, JAL, TRAP, ERET
• Floating Point
  – ADD.D, SUB.S, MUL.D, C.LT.D, CVT.S.W,
• Multimedia (SIMD)
  – ADD.PS, SUB.PS, MUL.PS, C.LT.PS
• String
  – REP MOVSB (x86)
# Addressing Modes: How to Get Operands from Memory

<table>
<thead>
<tr>
<th>Addressing Mode</th>
<th>Instruction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>Add R4, R3, R2</td>
<td>Regs[R4] &lt;- Regs[R3] + Regs[R2]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>**</td>
</tr>
<tr>
<td>Immediate</td>
<td>Add R4, R3, #5</td>
<td>Regs[R4] &lt;- Regs[R3] + 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>**</td>
</tr>
<tr>
<td>Displacement</td>
<td>Add R4, R3, 100(R1)</td>
<td>Regs[R4] &lt;- Regs[R3] + Mem[100 + Regs[R1]]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Absolute</td>
<td>Add R4, R3, (0x475)</td>
<td>Regs[R4] &lt;- Regs[R3] + Mem[0x475]</td>
</tr>
<tr>
<td>PC relative</td>
<td>Add R4, R3, 100(PC)</td>
<td>Regs[R4] &lt;- Regs[R3] + Mem[100 + PC]</td>
</tr>
</tbody>
</table>

** May not actually access memory!
Data Types and Sizes

• Types
  – Binary Integer
  – Binary Coded Decimal (BCD)
  – Floating Point
    • IEEE 754
    • Cray Floating Point
    • Intel Extended Precision (80-bit)
  – Packed Vector Data
  – Addresses

• Width
  – Binary Integer (8-bit, 16-bit, 32-bit, 64-bit)
  – Floating Point (32-bit, 40-bit, 64-bit, 80-bit)
  – Addresses (16-bit, 24-bit, 32-bit, 48-bit, 64-bit)
ISA Encoding

**Fixed Width:** Every Instruction has same width
- Easy to decode
  (RISC Architectures: MIPS, PowerPC, SPARC, ARM...)
Ex: MIPS, every instruction 4-bytes

**Variable Length:** Instructions can vary in width
- Takes less space in memory and caches
  (CISC Architectures: IBM 360, x86, Motorola 68k, VAX...)
Ex: x86, instructions 1-byte up to 17-bytes

**Mostly Fixed or Compressed:**
- Ex: MIPS16, THUMB (only two formats 2 and 4 bytes)
- PowerPC and some VLIWs (Store instructions compressed, decompress into Instruction Cache)

**(Very) Long Instruction Word:**
- Multiple instructions in a fixed width bundle
- Ex: Multiflow, HP/ST Lx, TI C6000
x86 (IA-32) Instruction Encoding

Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format

From Intel Processor Manual
MIPS Instruction Encoding

I-Type (Immediate).

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>25</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>rs</td>
<td>rt</td>
<td>offset</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

J-Type (Jump).

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>25</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>instr_index</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>26</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R-Type (Register).

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>25</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>6</th>
<th>5</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>rs</td>
<td>rt</td>
<td>rd</td>
<td>sa</td>
<td>function</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Real World Instruction Sets

<table>
<thead>
<tr>
<th>Arch</th>
<th>Type</th>
<th># Oper</th>
<th># Mem</th>
<th>Data Size</th>
<th># Regs</th>
<th>Addr Size</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alpha</td>
<td>Reg-Reg</td>
<td>3</td>
<td>0</td>
<td>64-bit</td>
<td>32</td>
<td>64-bit</td>
<td>Workstation</td>
</tr>
<tr>
<td>ARM</td>
<td>Reg-Reg</td>
<td>3</td>
<td>0</td>
<td>32/64-bit</td>
<td>16</td>
<td>32/64-bit</td>
<td>Cell Phones, Embedded</td>
</tr>
<tr>
<td>MIPS</td>
<td>Reg-Reg</td>
<td>3</td>
<td>0</td>
<td>32/64-bit</td>
<td>32</td>
<td>32/64-bit</td>
<td>Workstation, Embedded</td>
</tr>
<tr>
<td>SPARC</td>
<td>Reg-Reg</td>
<td>3</td>
<td>0</td>
<td>32/64-bit</td>
<td>24-32</td>
<td>32/64-bit</td>
<td>Workstation</td>
</tr>
<tr>
<td>TI C6000</td>
<td>Reg-Reg</td>
<td>3</td>
<td>0</td>
<td>32-bit</td>
<td>32</td>
<td>32-bit</td>
<td>DSP</td>
</tr>
<tr>
<td>IBM 360</td>
<td>Reg-Mem</td>
<td>2</td>
<td>1</td>
<td>32-bit</td>
<td>16</td>
<td>24/31/64</td>
<td>Mainframe</td>
</tr>
<tr>
<td>x86</td>
<td>Reg-Mem</td>
<td>2</td>
<td>1</td>
<td>8/16/32/64-bit</td>
<td>4/8/24</td>
<td>16/32/64</td>
<td>Personal Computers</td>
</tr>
<tr>
<td>VAX</td>
<td>Mem-Mem</td>
<td>3</td>
<td>3</td>
<td>32-bit</td>
<td>16</td>
<td>32-bit</td>
<td>Minicomputer</td>
</tr>
<tr>
<td>Mot. 6800</td>
<td>Accum.</td>
<td>1</td>
<td>1/2</td>
<td>8-bit</td>
<td>0</td>
<td>16-bit</td>
<td>Microcontroller</td>
</tr>
</tbody>
</table>
Why the Diversity in ISAs?

Technology Influenced ISA
- Storage is expensive, tight encoding important
- Reduced Instruction Set Computer
  - Remove instructions until whole computer fits on die
- Multicore/Manycore
  - Transistors not turning into sequential performance

Application Influenced ISA
- Instructions for Applications
  - DSP instructions
- Compiler Technology has improved
  - SPARC Register Windows no longer needed
  - Compiler can register allocate effectively
What Happens When the Processor is Too Large?

- Time Multiplex Resources!
Microcontrol Unit

*Maurice Wilkes, 1954*

First used in EDSAC-2, completed 1958

Embed the control logic state table in a memory array

op code conditional flip-flop

Memory

Decoder

Matrix A  Matrix B

Next state

Control lines to ALU, MUXs, Registers
Microcoded Microarchitecture

μcontroller (ROM)

Datapath

Memory (RAM)

holds fixed microcode instructions

holds user program written in macrocode instructions (e.g., x86, MIPS, etc.)

busy? zero? opcode

Data Addr

enMem MemWrt
A Bus-based Datapath for RISC

Microinstruction: register to register transfer (17 control signals)
Recap

Computer Architecture
ELE 475

- Application
- Algorithm
- Programming Language
- Operating System/Virtual Machines
- Instruction Set Architecture
- Microarchitecture
- Register-Transfer Level
- Gates
- Circuits
- Devices
- Physics
Recap

- ISA vs Microarchitecture
- ISA Characteristics
  - Machine Models
  - Encoding
  - Data Types
  - Instructions
  - Addressing Modes
- Microcode
  - Enables small Processors
ELE 475 Lecture 1

Lab 0 is Out (Not collected, Review of Verilog)

Class Wed. Feb. 15 Rescheduled Fri. Feb. 17
1:30-2:50pm

Next Class: Review of Pipelining
Acknowledgements

• These slides contain material developed and copyright by:
  – Arvind (MIT)
  – Krste Asanovic (MIT/UCB)
  – Joel Emer (Intel/MIT)
  – James Hoe (CMU)
  – John Kubiatowicz (UCB)
  – David Patterson (UCB)
  – Christopher Batten (Cornell)

• MIT material derived from course 6.823
• UCB material derived from course CS252 & CS152
• Cornell material derived from course ECE 4750