RISC, CISC, and ISA Variations

Anne Bracy
CS 3410
Computer Science
Cornell University

The slides are the product of many rounds of teaching CS 3410 by Professors Weatherspoon, Bala, Bracy, McKee, and Sirer.
Big Picture: Where are we now?

- **Write**: Memory
- **Back**: Instruction
- **Fetch**: Instruction
- **Instruction Decode**: Instruction
- **Compute jump/branch targets**: Control
- **Extend register file**: Multiplexer
- **Detect hazard**: Multiplexer
- **Forward unit**: Multiplexer
- **Memory**: Multiplexer
- **Write-Back**: Multiplexer

**Pipeline Stages**:
- **IF/ID**: Instruction Fetch
- **ID/EX**: Instruction Decode
- **EX/MEM**: Execute
- **MEM/WB**: Write-Back
Big Picture: Where are we going?

- **C compiler**
  - `int x = 10;`  
  - `x = x + 15;`

- **MIPS assembly**
  - `addi r5, r0, 10`
  - `addi r5, r5, 15`

- **Assembler**
  - `op = addi r0 r5 10`
  - `op = addi r5 r5 15`

- **Machine code**
  - Binary representation:
    - `00100000000001010000000000001010`
    - `00100000101001010000000000001111`

- **CPU**
  - **Circuits**
  - **Gates**
  - **Transistors**
  - **Silicon**

- **Diagram**:
  - Logic gate: `V_D` as input, `V_G` and `V_S` as control signals, output `0` when `V_D` is high and both `V_G` and `V_S` are low.
Big Picture: Where are we going?

- **C**
  - compiler
  - MIPS
    - assembly
      - assembler
  - machine code
    - CPU
      - Circuits
        - Gates
          - Transistors
            - Silicon

- **High Level Languages**
  - int x = 10;
    x = 2 * x + 15;
  - addi r5, r0, 10
  - muli r5, r5, 2
  - addi r5, r5, 15

- **MIPS Assembly**
  - Machine code
    - Instruction Set Architecture (ISA)

**Machine Code:**

```
00100000000001010000000000001010
000000000000010100100001000000
00100001010010101000000000000111
```
Goals for Today

Instruction Set Architectures

- ISA Variations, and CISC vs RISC

Next Time

- Program Structure and Calling Conventions
Next Goal

Is MIPS the only possible instruction set architecture (ISA)?

What are the alternatives?
Instruction Set Architecture Variations

ISA defines the permissible instructions

- MIPS: load/store, arithmetic, control flow, ...
- ARMv7: similar to MIPS, but more shift, memory, & conditional ops
- ARMv8 (64-bit): even closer to MIPS, no conditional ops
- VAX: arithmetic on memory or registers, strings, polynomial evaluation, stacks/queues, ...
- Cray: vector operations, ...
- x86: a little of everything
Accumulators

• Early stored-program computers had **one** register!

• One register is two registers short of a MIPS instruction!

• Requires a memory-based operand-addressing mode
  
  – Example Instructions:  add 200
    
    ▪ Add the accumulator to the word in memory at address 200
    ▪ Place the sum back in the accumulator
Brief Historical Perspective on ISAs

Next step, more registers...

• Dedicated registers
  – E.g. indices for array references in data transfer instructions, separate accumulators for multiply or divide instructions, top-of-stack pointer.

![Intel 8086](image)

Intel 8086
“extended accumulator”
Processor for IBM PCs

• Extended Accumulator
  – One operand may be in memory (like previous accumulators).
  – Or, all the operands may be registers (like MIPS).
Brief Historical Perspective on ISAs

Next step, more registers...

• General-purpose registers
  – Registers can be used for any purpose
  – E.g. MIPS, ARM, x86

• Register-memory architectures
  – One operand may be in memory (e.g. accumulators)
  – E.g. x86 (i.e. 80386 processors)

• Register-register architectures (aka load-store)
  – All operands must be in registers
  – E.g. MIPS, ARM
The number of available registers greatly influenced the instruction set architecture (ISA)

<table>
<thead>
<tr>
<th>Machine</th>
<th>Num General Purpose Registers</th>
<th>Architectural Style</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDSAC</td>
<td>1</td>
<td>Accumulator</td>
<td>1949</td>
</tr>
<tr>
<td>IBM 701</td>
<td>1</td>
<td>Accumulator</td>
<td>1953</td>
</tr>
<tr>
<td>CDC 6600</td>
<td>8</td>
<td>Load-Store</td>
<td>1963</td>
</tr>
<tr>
<td>IBM 360</td>
<td>18</td>
<td>Register-Memory</td>
<td>1964</td>
</tr>
<tr>
<td>DEC PDP-8</td>
<td>1</td>
<td>Accumulator</td>
<td>1965</td>
</tr>
<tr>
<td>DEC PDP-11</td>
<td>8</td>
<td>Register-Memory</td>
<td>1970</td>
</tr>
<tr>
<td>Intel 8008</td>
<td>1</td>
<td>Accumulator</td>
<td>1972</td>
</tr>
<tr>
<td>Motorola 6800</td>
<td>2</td>
<td>Accumulator</td>
<td>1974</td>
</tr>
<tr>
<td>DEC VAX</td>
<td>16</td>
<td>Register-Memory, Memory-Memory</td>
<td>1977</td>
</tr>
<tr>
<td>Intel 8086</td>
<td>1</td>
<td>Extended Accumulator</td>
<td>1978</td>
</tr>
<tr>
<td>Motorola 6800</td>
<td>16</td>
<td>Register-Memory</td>
<td>1980</td>
</tr>
<tr>
<td>Intel 80386</td>
<td>8</td>
<td>Register-Memory</td>
<td>1985</td>
</tr>
<tr>
<td>ARM</td>
<td>16</td>
<td>Load-Store</td>
<td>1985</td>
</tr>
<tr>
<td>MIPS</td>
<td>32</td>
<td>Load-Store</td>
<td>1985</td>
</tr>
<tr>
<td>HP PA-RISC</td>
<td>32</td>
<td>Load-Store</td>
<td>1986</td>
</tr>
<tr>
<td>SPARC</td>
<td>32</td>
<td>Load-Store</td>
<td>1987</td>
</tr>
<tr>
<td>PowerPC</td>
<td>32</td>
<td>Load-Store</td>
<td>1992</td>
</tr>
<tr>
<td>DEC Alpha</td>
<td>32</td>
<td>Load-Store</td>
<td>1992</td>
</tr>
<tr>
<td>HP/Intel IA-64</td>
<td>128</td>
<td>Load-Store</td>
<td>2001</td>
</tr>
<tr>
<td>AMD64 (EMT64)</td>
<td>16</td>
<td>Register-Memory</td>
<td>2003</td>
</tr>
</tbody>
</table>
Next Goal

How to compute with limited resources?

i.e. how do you design your ISA if you have limited resources?
In the Beginning...
People programmed in assembly and machine code!

- Needed as many addressing modes as possible
- Memory was (and still is) slow

CPUs had relatively few registers
- Register’s were more “expensive” than external mem
- Large number of registers requires many bits to index

Memories were small
- Encouraged highly encoded microcodes as instructions
- Variable length instructions, load/store, conditions, etc
In the Beginning...
People programmed in assembly and machine code!
E.g. x86
• > 1000 instructions!
  – 1 to 15 bytes each
  – E.g. dozens of add instructions
• operands in dedicated registers, general purpose registers, memory, on stack, ...
  – can be 1, 2, 4, 8 bytes, signed or unsigned
• 10s of addressing modes
  – e.g. Mem[segment + reg + reg*scale + offset]
E.g. VAX
• Like x86, arithmetic on memory or registers, but also on strings, polynomial evaluation, stacks/queues, ...
Complex Instruction Set Computers (CISC)
Takeaway

The number of available registers greatly influenced the instruction set architecture (ISA)

*Complex Instruction Set Computers* were very complex

- Necessary to reduce the number of instructions required to fit a program into memory.
- However, also greatly increased the complexity of the ISA as well.
Next Goal

How do we reduce the complexity of the ISA while maintaining or increasing performance?
Reduced Instruction Set Computer (RISC)

John Cock

- IBM 801, 1980 (started in 1975)
- Name 801 came from the bldg that housed the project
- Idea: Possible to make a very small and very fast core
- Known as “the father of RISC Architecture”
- Turing Award Recipient and National Medal of Science
Reduced Instruction Set Computer (RISC)

Dave Patterson
- RISC Project, 1982
- UC Berkeley
- RISC-I: ½ transistors & 3x faster
- Influences: Sun SPARC, namesake of industry

John L. Hennessy
- MIPS, 1981
- Stanford
- Simple pipelining, keep full
- Influences: MIPS computer system, PlayStation, Nintendo
CS Colloquium

Thursdays @ 4:15 in Gates, G01

Thursday, October 15: David Patterson
Reduced Instruction Set Computer (RISC)

MIPS Design Principles

Simplicity favors regularity
  • 32 bit instructions

Smaller is faster
  • Small register file

Make the common case fast
  • Include support for constants

Good design demands good compromises
  • Support for different type of interpretations/classes
Reduced Instruction Set Computer

MIPS = Reduced Instruction Set Computer (RISC)
- ≈ 200 instructions, 32 bits each, 3 formats
- all operands in registers
  - almost all are 32 bits each
- ≈ 1 addressing mode: Mem[reg + imm]

x86 = Complex Instruction Set Computer (CISC)
- > 1000 instructions, 1 to 15 bytes each
- operands in dedicated registers, general purpose registers, memory, on stack, ...
  - can be 1, 2, 4, 8 bytes, signed or unsigned
- 10s of addressing modes
  - e.g. Mem[segment + reg + reg*scale + offset]
<table>
<thead>
<tr>
<th>RISC vs CISC</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RISC Philosophy</td>
<td>CISC Rebuttal</td>
</tr>
<tr>
<td>Regularity &amp; simplicity</td>
<td>Compilers can be smart</td>
</tr>
<tr>
<td>Leaner means faster</td>
<td>Transistors are plentiful</td>
</tr>
<tr>
<td>Optimize the common case</td>
<td>Legacy is important</td>
</tr>
<tr>
<td>Code size counts</td>
<td>Micro-code!</td>
</tr>
<tr>
<td>Micro-code!</td>
<td>“RISC Inside”</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy efficiency</td>
<td>Desktops/Servers</td>
</tr>
<tr>
<td>Embedded Systems</td>
<td></td>
</tr>
<tr>
<td>Phones/Tablets</td>
<td></td>
</tr>
</tbody>
</table>
ARMDroid vs WinTel

• Android OS on ARM processor
• Windows OS on Intel (x86) processor
The number of available registers greatly influenced the instruction set architecture (ISA).

Complex Instruction Set Computers were very complex.
- Necessary to reduce the number of instructions required to fit a program into memory.
- However, also greatly increased the complexity of the ISA as well.

Back in the day... CISC was necessary because everybody programmed in assembly and machine code! Today, CISC ISA’s are still dominant due to the prevalence of x86 ISA processors. However, RISC ISA’s today such as ARM have an ever increasing market share (of our everyday life!). ARM borrows a bit from both RISC and CISC.
Next Goal

How does MIPS and ARM compare to each other?
MIPS instruction formats

All MIPS instructions are 32 bits long, has 3 formats

R-type

op  rs  rt  rd  shamt  func

6 bits  5 bits  5 bits  5 bits  5 bits  6 bits

I-type

op  rs  rt  immediate

6 bits  5 bits  5 bits  16 bits

J-type

op  immediate (target address)

6 bits  26 bits
# ARMv7 Instruction Formats

All ARMv7 instructions are 32 bits long, has 3 formats:

**R-type**

<table>
<thead>
<tr>
<th>opx</th>
<th>op</th>
<th>rs</th>
<th>rd</th>
<th>opx</th>
<th>rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 bits</td>
<td>8 bits</td>
<td>4 bits</td>
<td>4 bits</td>
<td>8 bits</td>
<td>4 bits</td>
</tr>
</tbody>
</table>

**I-type**

<table>
<thead>
<tr>
<th>opx</th>
<th>op</th>
<th>rs</th>
<th>rd</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 bits</td>
<td>8 bits</td>
<td>4 bits</td>
<td>4 bits</td>
<td>12 bits</td>
</tr>
</tbody>
</table>

**J-type**

<table>
<thead>
<tr>
<th>opx</th>
<th>op</th>
<th>immediate (target address)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 bits</td>
<td>4 bits</td>
<td>24 bits</td>
</tr>
</tbody>
</table>
MIPS Control Dependence

while(i != j) {
    if (i > j) {
        i -= j;
    } else {
        j -= i;
    }
}

Loop: BEQ Ri, Rj, End // if "NE" (not equal), then stay in loop
    SLT Rd, Rj, Ri    // "GT" if (i > j),
    BNE Rd, R0, Else // ...
    SUB Ri, Ri, Rj    // if "GT" (greater than), i = i-j;
    J Loop
Else: SUB Rj, Rj, Ri // or "LT" if (i < j)
    J Loop            // if "LT" (less than), j = j-i;
End:
while (i != j) {
    if (i > j)
        i -= j;
    else
        j -= i;
}

ARM: avoid delay due to branches with conditional instructions

Control Independence!

ARMv7 Conditional Instructions

Loop: CMP Ri, Rj 0 1 0 0 // set condition "NE" if (i != j)

// "GT" if (i > j),

// or "LT" if (i < j)

SUBGT Ri, Ri, Rj 1 0 1 0 // if "GT" (greater than), i = i-j;

SUBLE Rj, Rj, Ri // if "LE" (less than or equal), j = j-i;

BNE loop // if "NE" (not equal), then loop
ARMv7: Other Cool operations

Shift one register (e.g. Rc) any amount
Add to another register (e.g. Rb)
Store result in a different register (e.g. Ra)

ADD Ra, Rb, Rc LSL #4
Ra = Rb + Rc<<4
Ra = Rb + Rc x 16
ARMv7 Instruction Set Architecture

All ARMv7 instructions are 32 bits long, has 3 formats

Reduced Instruction Set Computer (RISC) properties

- Only Load/Store instructions access memory
- Instructions operate on operands in processor registers
- 16 registers

Complex Instruction Set Computer (CISC) properties

- Autoincrement, autodecrement, PC-relative addressing
- Conditional execution
- Multiple words can be accessed from memory with a single instruction (SIMD: single instr multiple data)
ARMv8 (64-bit) Instruction Set Architecture
All ARMv8 instructions are 64 bits long, has 3 formats
Reduced Instruction Set Computer (RISC) properties
  • Only Load/Store instructions access memory
  • Instructions operate on operands in processor registers
  • 32 registers and r0 is always 0

NO MORE Complex Instruction Set Computer (CISC) properties
  • NO Conditional execution
  • NO Multiple words can be accessed from memory with a single instruction (SIMD: single instr multiple data)
Instruction Set Architecture Variations

ISA defines the permissible instructions

- MIPS: load/store, arithmetic, control flow, ...
- ARMv7: similar to MIPS, but more shift, memory, & conditional ops
- ARMv8 (64-bit): even closer to MIPS, no conditional ops
- VAX: arithmetic on memory or registers, strings, polynomial evaluation, stacks/queues, ...
- Cray: vector operations, ...
- x86: a little of everything