RISC, CISC, and ISA Variations

Hakim Weatherspoon
CS 3410
Computer Science
Cornell University

[Weatherspoon, Bala, Bracy, McKee, and Sirer]
Announcements

• Prelim *tonight*
  • Tuesday at 7:30pm
  • Go to location based on NetID
    • [a – g]* : HLS110 (Hollister 110)
    • [h – mg]* : HLSB14 (Hollister B14)
    • [mh – z]* : KMBB11 (Kimball B11)
Announcements

• Prelim1:
  • Time: We will start at 7:30pm sharp, so come early
  • Location: on previous slide
  • Closed Book
  • Cannot use electronic device or outside material
  • Practice prelims are online in CMS

• Material covered everything up to end of this week
  • Everything up to and including data hazards
  • Appendix A (logic, gates, FSMs, memory, ALUs)
  • Chapter 4 (pipelined [and non] MIPS processor with hazards)
  • Chapters 2 (Numbers / Arithmetic, simple MIPS instructions)
  • Chapter 1 (Performance)
  • Projects 1 and 2, Lab0-4, C HW1
iClicker Question

Which is **not** considered part of the ISA?

A. There is a control delay slot.
B. The number of inputs each instruction can have.
C. Load-use stalls will **not** be detected by the processor.
D. The number of cycles it takes to execute a multiply.
E. Each instruction is encoded in 32 bits.
iClicker Question

Which is not considered part of the ISA?

A. There is a control delay slot.
B. The number of inputs each instruction can have.
C. Load-use stalls will not be detected by the processor.
D. The number of cycles it takes to execute a multiply.
E. Each instruction is encoded in 32 bits.
Big Picture: Where are we now?

Instruction Fetch

Instruction Decode

Decode

Forward unit

alu

compute jump/branch targets

extend

Detect hazard

control

addr
d_{in} \quad d_{out}

memory

Forward unit

Memory

Write-Back

IF/ID

ID/EX

EX/MEM

MEM/WB
Big Picture: Where are we going?

C
compiler

RISC-V
assembly
assembler

machine
code

CPU

Circuits
Gates
Transistors
Silicon

int x = 10;
x = 2 * x + 15;
x0 = 0
x5 = x0 + 10
addi x5, x0, 10
x5 = x5<<1
#x5 = x5 * 2
muli x5, x5, 2
x5 = x15 + 15
addi x5, x5, 15

0000000001010000000010100010011
00000000101000000100000101000000
00000000111100101000001010010011

10                              r0                  r5
op = addi

15                               r5                 r5
op = addi

00000000111100101000001010010011
00000000010100101000001010000000
00000000010100100000101001001111

r5
x5
shamt=1
x5
func=sll
Big Picture: Where are we going?

```
int x = 10;
x = 2 * x + 15;
addi x5, x0, 10
muli x5, x5, 2
addi x5, x5, 15
```

000000000101000000000001010010011
00000000001000101000001010000000
000000001111001010000010100010011

High Level Languages

Instruction Set Architecture (ISA)
Goals for Today
Instruction Set Architectures
• ISA Variations, and CISC vs RISC
• Peek inside some other ISAs:
  • X86
  • ARM
Next Goal

Is RISC-V the only possible instruction set architecture (ISA)?

What are the alternatives?
Instruction Set Architecture Variations

ISA defines the permissible instructions

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

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

  EDSAC (Electronic Delay Storage Automatic Calculator) in 1949

  Intel 8008 in 1972 was an accumulator

  - One register is two registers short of a RISC-V instruction!
  - Requires a memory-based operand-addressing mode
    - Example Instructions:  
      add 200 // ACC = ACC + Mem[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.

• Extended Accumulator
  - One operand may be in memory (like previous accumulators).
  - Or, all the operands may be registers (like RISC-V).

Intel 8086
“extended accumulator”
Processor for IBM PCs
 Brief Historical Perspective on ISAs

Next step, more registers…

• General-purpose registers
  - Registers can be used for any purpose
  - E.g. RISC-V, 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. RISC-V, MIPS, ARM
Takeaway

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
- Influences: 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: $\frac{1}{2}$ transistors & 3x faster
- Influences: Sun SPARC, namesake of industry

John L. Hennessy
- MIPS, 1981
- Stanford
- Simple, *full* pipeline
- Influences: MIPS computer system, PlayStation, Nintendo
Reduced Instruction Set Computer (RISC)  
RISC-V Design Principles

Simplicity favors regularity
• 32 bit instructions
• Same instruction format works at 16- or 64-bit formats

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

RISC-V = Reduced Instruction Set Computer (RISC)
- ≈ 200 instructions, 32 bits each, 4 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]
The RISC Tenets

**RISC**
- Single-cycle execution
- Hardwired control
- Load/store architecture
- Few memory addressing modes
- Fixed-length insn format
- Reliance on compiler optimizations
- Many registers (compilers are better at using them)

**CISC**
- many multicycle operations
- microcoded multi-cycle operations
- register-mem and mem-mem
- many modes
- many formats and lengths
- hand assemble to get good performance
- few registers
RISC vs CISC

RISC Philosophy
Regularity & simplicity
Leaner means faster
Optimize the common case

Energy efficiency
Embedded Systems
Phones/Tablets

CISC Rebuttal
Compilers can be smart
Transistors are plentiful
Legacy is important
Code size counts
Micro-code!

Desktops/Servers
ARMDroid vs WinTel

Android OS on ARM processor

Windows OS on Intel (x86) processor
What is one advantage of a CISC ISA?

A. It naturally supports a faster clock.
B. Instructions are easier to decode.
C. The static footprint of the code will be smaller.
D. The code is easier for a compiler to optimize.
E. You have a lot of registers to use.
What is one advantage of a CISC ISA?

A. It naturally supports a faster clock.
B. Instructions are easier to decode.
C. The static footprint of the code will be smaller.
D. The code is easier for a compiler to optimize.
E. You have a lot of registers to use.
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.

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 RISC-V and ARM compare to each other?
RISC-V instruction formats

All RISC-V instructions are 32 bits long, have 4 formats

• **R-type**
  
  
  funct7  rs2  rs1  funct3  rd  op
  
  7 bits  5 bits  5 bits  3 bits  5 bits  7 bits

• **I-type**
  
  
  imm  rs1  funct3  rd  op
  
  12 bits  5 bits  3 bits  5 bits  7 bits

• **S-type**
  
  
  imm  rs2  rs1  funct3  imm  op
  
  7 bits  5 bits  5 bits  3 bits  5 bits  7 bits

• **U-type**
  
  
  imm  rd  op
  
  20 bits  5 bits  7 bits
ARMv7 instruction formats

All ARMv7 instructions are 32 bits long, has 3 formats

<table>
<thead>
<tr>
<th>Format</th>
<th>opx</th>
<th>op</th>
<th>rs</th>
<th>rd</th>
<th>opx</th>
<th>rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<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>
<tr>
<td>I-type</td>
<td>4 bits</td>
<td>8 bits</td>
<td>4 bits</td>
<td>4 bits</td>
<td>12 bits</td>
<td></td>
</tr>
<tr>
<td>J-type</td>
<td>4 bits</td>
<td>4 bits</td>
<td>immediate (target address)</td>
<td>24 bits</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ARMv7 Conditional Instructions

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

In RISC-V, performance will be slow if code has a lot of branches.

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:
ARMv7 Conditional Instructions

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

LOOP: CMP Ri, Rj // set condition "NE" if (i != j)
     // "GT" if (i > j),
     // or "LT" if (i < j)
     SUBGT Ri, Ri, Rj // 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
```

In ARM, can avoid delay due to Branches with conditional instructions
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
- **RISC-V**: load/store, arithmetic, control flow, …
- ARMv7: similar to RISC-V, but more shift, memory, & conditional ops
- ARMv8 (64-bit): even closer to RISC-V, no conditional ops
- VAX: arithmetic on memory or registers, strings, polynomial evaluation, stacks/queues, …
- Cray: vector operations, …
- x86: a little of everything
ISA Takeaways

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

Complex Instruction Set Computers were very complex
+ Small # of insns necessary to fit program into memory.
- 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.