CSC 411

Computer Organization (Fall 2024)
Lecture 10: RISC-V basics

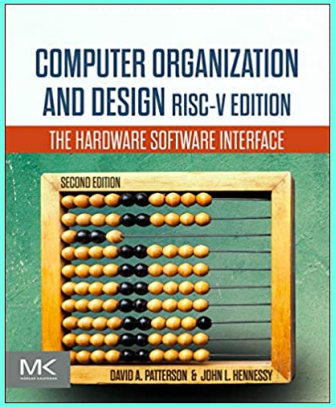
Prof. Marco Alvarez, University of Rhode Island

Disclaimer

Some figures and slides are adapted from:

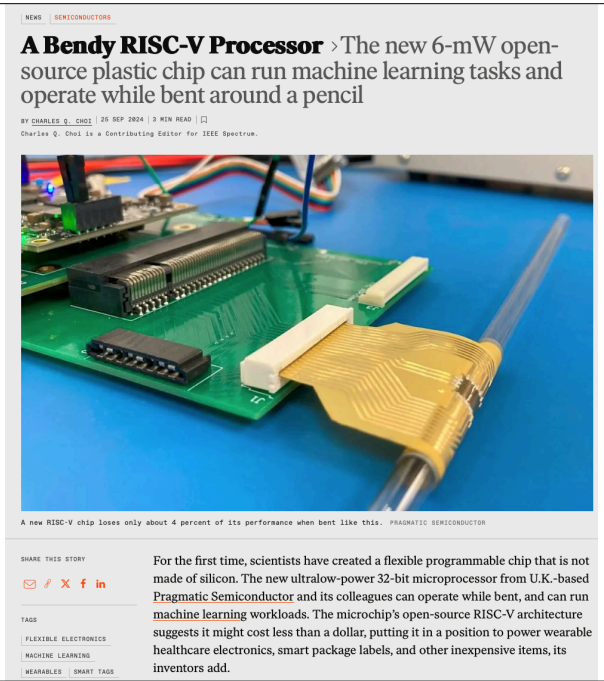
Computer Organization and Design (Patterson and Hennessy)

The Hardware/Software Interface



The RISC-V ISA

- RISC-V: modern and full-featured RISC ISA
  - open-source, free, simple, and extensible
  - supports heterogeneous and parallel systems
  - supports 32-bit, 64-bit, and 128-bit variants
  - supports (but does not require) IEEE 754 floating-point standard
- Developed at UC Berkeley as an open ISA (~2010)
  - by 2024, over 350 members in RISC-V International (https://riscv.org/members/)
- Growing market share in embedded and high-performance computing
  - IoT devices, AI accelerators, automotive systems, data centers



NEWS | SEMICONDUCTORS

A Bendy RISC-V Processor ›The new 6-mW open-source plastic chip can run machine learning tasks and operate while bent around a pencil

BY CHARLES Q. CHOI | 25 SEP 2024 | 3 MIN READ |

Charles Q. Choi is a Contributing Editor for IEEE Spectrum.

A new RISC-V chip loses only about 4 percent of its performance when bent like this. PRAGMATIC SEMICONDUCTOR

SHARE THIS STORY

TAGS

FLEXIBLE ELECTRONICS | MACHINE LEARNING | WEARABLES | SMART TAGS

For the first time, scientists have created a flexible programmable chip that is not made of silicon. The new ultralow-power 32-bit microprocessor from U.K.-based Pragmatic Semiconductor and its colleagues can operate while bent, and can run machine learning workloads. The microchip's open-source RISC-V architecture suggests it might cost less than a dollar, putting it in a position to power wearable healthcare electronics, smart package labels, and other inexpensive items, its inventors add.

RISC-V base and extensions

- Modular design allows for customization and specialization
- Base Integer Instruction Set
  - RV32I (32-bit), RV64I (64-bit)
  - includes basic integer arithmetic, logical operations, and control flow instructions
- Standard Extensions
  - M: Integer Multiplication and Division
  - A: Atomic Instructions
  - F: Single-Precision Floating-Point
  - D: Double-Precision Floating-Point
  - C: Compressed Instructions (16-bit encoded)
  - V: Vector Operations
- Custom Extensions
  - allows for domain-specific instructions (e.g., cryptography, machine learning)

RISC-V base and extensions

Name	Description	Version	Status[A]	Instruction count
Base				
RVWMO	Weak Memory Ordering	2.0	Ratified	
RV32I	Base Integer Instruction Set, 32-bit	2.1	Ratified	40
RV32E	Base Integer Instruction Set (embedded), 32-bit, 16 registers	2.0	Ratified	40
RV64I	Base Integer Instruction Set, 64-bit	2.1	Ratified	15
RV64E	Base Integer Instruction Set (embedded), 64-bit	2.0	Ratified	
RV128I	Base Integer Instruction Set, 128-bit	1.7	Open	15
Extension				
M	Standard Extension for Integer Multiplication and Division	2.0	Ratified	8 (RV32) 13 (RV64)
A	Standard Extension for Atomic Instructions	2.1	Ratified	11 (RV32) 22 (RV64)
F	Standard Extension for Single-Precision Floating-Point	2.2	Ratified	26 (RV32) 30 (RV64)
D	Standard Extension for Double-Precision Floating-Point	2.2	Ratified	26 (RV32) 32 (RV64)

`and many others … https://en.wikipedia.org/wiki/RISC-V`

Instruction set

- Assembly language
  - bridge between human-readable code and machine code
  - each assembly language is specific to a particular ISA, reflecting its instruction set and architecture
  - one line of assembly code typically represents one machine instruction
- RISC-V assembly operands are primarily **registers**
  - RISC-V operations can only be performed on registers
  - **`load-store`** architecture: separate instructions for memory access

```
add x1, x2, x3    # x1 <- x2 + x3
```

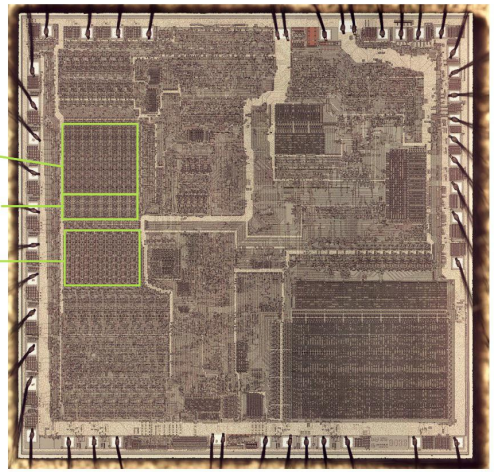
Registers

- Built directly into the hardware
  - limited number of registers available (**register file**)
    - assembly code must be very carefully put together to efficiently use them (mostly done by **compilers**)
  - very fast access, located within the CPU (0-1 cycles)
    - on a 3 GHz CPU, access time is approximately 0.33ns
- Registers are typeless
  - contents are just bits, the applied **“operation”** determines how the bits are interpreted
- Registers are extremely fast with a very limited capacity

Intel 8086's registers

One of the most influential chips ever created; it led to the x86 architecture.

The upper 16-bit registers are used by the Bus Interface Unit for memory accesses, while the general-purpose lower 16-bit registers are used by the Execution Unit. The instruction buffer is a 6-byte queue of prefetched instructions.



How many bytes for all registers?

https://www.righto.com/2020/07/the-intel-8086-processors-registers.html

RISC-V registers

- RISC-V has 32 integer registers (16 in the embedded variant)
  - numbered from **x0 to x31** and can be referenced by **number** or **name**
  - additional 32 floating-point registers f**0 to f31** when the FP extension is implemented
  - each register is 32-bit wide in RV32 variant and 64-bit wide in RV64
- 32-bit sequences are called a **word** and 64-bits sequences a **doubleword**

RISC-V registers

Register name	Symbolic name	Description	Saved by
32 integer registers			
x0	Zero	Always zero	
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	
x4	tp	Thread pointer	
x5	t0	Temporary / alternate return address	Caller
x6–7	t1–2	Temporary	Caller
x8	s0/fp	Saved register / frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function argument / return value	Caller
x12–17	a2–7	Function argument	Caller
x18–27	s2–11	Saved register	Callee
x28–31	t3–6	Temporary	Caller

https://en.wikipedia.org/wiki/RISC-V

RISC-V registers

32 floating-point extension registers			
f0–7	ft0–7	Floating-point temporaries	Caller
f8–9	fs0–1	Floating-point saved registers	Callee
f10–11	fa0–1	Floating-point arguments/return values	Caller
f12–17	fa2–7	Floating-point arguments	Caller
f18–27	fs2–11	Floating-point saved registers	Callee
f28–31	ft8–11	Floating-point temporaries	Caller

https://en.wikipedia.org/wiki/RISC-V

Arithmetic operations

- Require three register **operands**
  - rigid form: opname, destination, source1, source2

```
opname rd, rs1, rs2
```

  - basic integer arithmetic instructions
    - add, sub
  - logical operations
    - and, or, xor, sll (shift left logical), srl (shift right logical), sra (shift right arithmetic)

	Addition	Subtraction
C	`a = b + c`	`a = b - c`
RISC-V	`add x1, x2, x3`	`sub x1, x2, x3`

Arithmetic operations

- How would you translate the following C code?
  - single line of C may convert into multiple lines in assembly
  - some sequences are shorter than others or may use less "temporary" registers

```
// assume these variables are mapped to
// x5, x1, x2, x3, x4 respectively
a = b + c + d + e;
```

Practice

- Translate the following C code into assembly
  - if needed, use temporary registers x6, x7

```
// assume these variables are mapped to
// x5, x1, x2, x3, x4 respectively
f = (g + h) - (i + j);
```

  - optimize the C code to minimize register usage
    - good compilers do it all the time

In fact, if the variables are floating-point values, different sequences of instructions may produce slightly different results. Floating-point operations are not necessarily associative or commutative …

Immediate operations

- Allow use of constant values
  - immediate field (numerical constant) typically 12 bits wide

```
opname rd, rs1, imm
```

- Common immediate instructions
  - addi, andi, ori, xori, slli, srli, srai
  - no separate `subi` instruction, use `addi` with negative constant
  - e.g.

```
addi x5, x6, 5
addi x5, x6, -10
```

Register zero

- Register **x0** is hardwired to constant 0
  - **cannot be overwritten**
  - read always provides 0
- Useful for common operations
  - move between registers

```
add x5, x0, x6
```

  - assigning constants to registers

```
addi x5, x0, 42
```