Lecture 07: RISC-V ISA

CSE 564 Computer Architecture Summer 2017

Department of Computer Science and Engineering
Yonghong Yan
yan@oakland.edu
www.secs.oakland.edu/~yan

Contents

1. RISC-V ISA
2. 1 stage implementation
3. Chisel hardware construction language
4. Pipeline implementation

Acknowledgement

- Slides adapted from
  - Computer Science 152: Computer Architecture and Engineering, Spring 2016 by Dr. George Michelogiannakis from UCB

What is RISC-V

- RISC-V (pronounced "risk-five") is a ISA standard (a document)
  - An open source implementation of a reduced instruction set computing (RISC) based instruction set architecture (ISA)
  - There was RISC-I, II, III, IV before
- Most ISAs: X86, ARM, Power, MIPS, SPARC
  - Commercially protected by patents
  - Preventing practical efforts to reproduce the computer systems.
- RISC-V is open
  - Permitting any person or group to construct compatible computers
  - Use associated software
- The project was originated in 2010 by researchers in the Computer Science Division at UC Berkeley, but it now has a large number of contributors. As of 2017 version 2 of the userspace ISA is fixed
  - User-Level ISA Specification v2.2
  - Draft Compressed ISA Specification v1.79
  - Draft Privileged ISA Specification v1.10

Goals in defining RISC-V

- A completely open ISA that is freely available to academia and industry
- A real ISA suitable for direct native hardware implementation, not just simulation or binary translation
- An ISA that **avoids "over-architecting" for**
  - a particular microarchitecture style (e.g., microcoded, in-order, decoupled, out-of-order) or
  - implementation technology (e.g., full-custom, ASIC, FPGA), but which allows efficient implementation in any of these
- RISC-V ISA includes
  - **A small base integer ISA**, usable by itself as a base for customized accelerators or for educational purposes, and
  - **Optional standard extensions**, to support general-purpose software development
  - **Optional customer extensions**
- Support for the revised 2008 IEEE-754 floating-point standard

License for the ISA specification

- It is **a BSD Open Source License**.
  - This is a non-viral license, only asking that if you use it, you acknowledge the authors, in this case UC Berkeley.
  - No patent that would be required to implement a RISC-V-compatible processor
  - They may be many micro-architectural patents that might be infringed by a particular RISC-V implementation.
    - But cannot indemnify users against ISA or implementation patents asserted by others
- The goal of the proposed RISC-V consortium is to maintain and track possible patent issues for RISC-V implementors

- Open Source Software License
  - GPL: Extensions must be open sourced with the same license (kind of)
  - BSD: Use it as you want/like (kind of)

RISC-V ISA Principles

- Generally kept very simple and extendable
- Separated into multiple specifications
  - User-Level ISA spec (compute instructions)
  - Compressed ISA spec (16-bit instructions)
  - Privileged ISA spec (supervisor-mode instructions)
  - More …

- ISA support is given by RV + word-width + extensions supported
  - E.g. RV32I means 32-bit RISC-V with support for the I instruction set

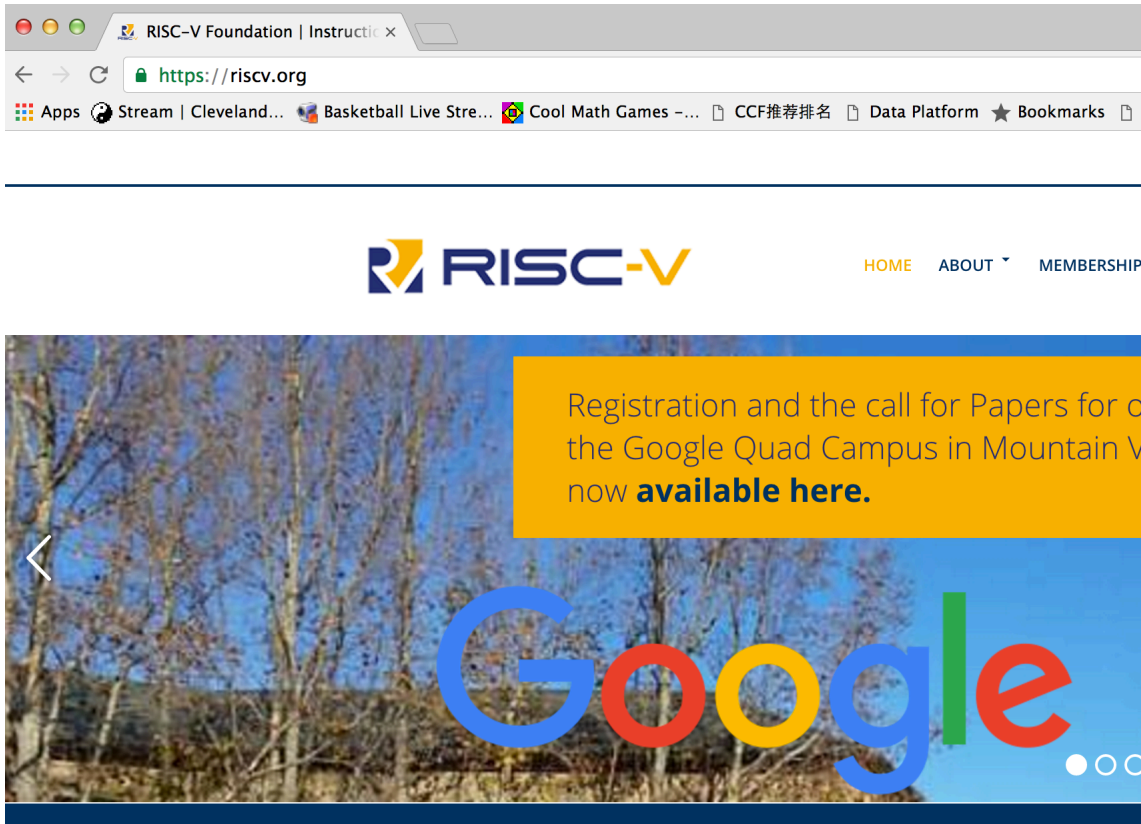
User Level ISA

- Defines the normal instructions needed for computation
  - A mandatory **Base integer ISA**
    - **I: Integer instructions: ALU, branches/jumps, and loads/stores**
    - Support for misaligned memory access is mandatory
  - **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)**

    - **G = IMAFD: Integer base + four standard extensions**
  - Optional extensions

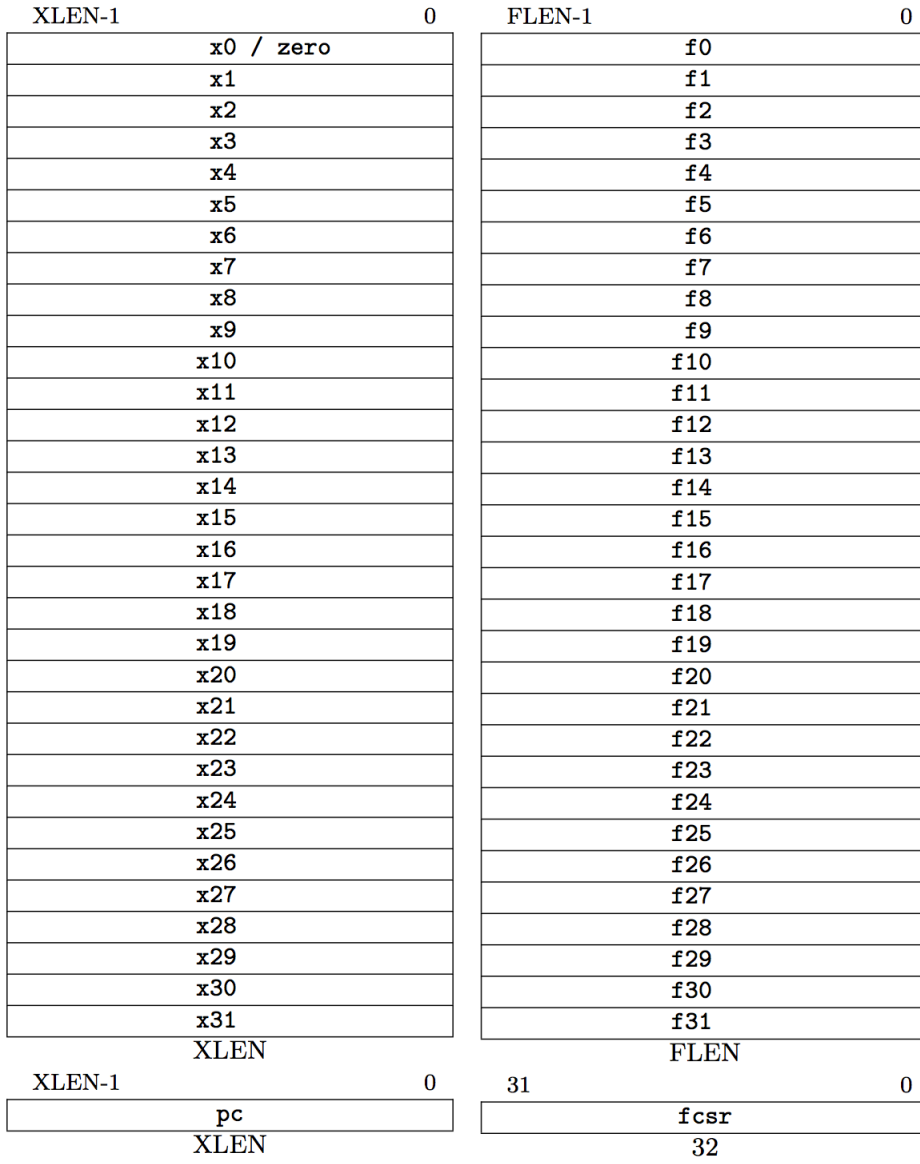
RISC-V ISA

- Both 32-bit and 64-bit address space variants
  - RV32 and RV64
- Easy to subset/extend for education/research
  - RV32IM, RV32IMA, RV32IMAFD, RV32G
- SPEC on the website
  - www.riscv.org



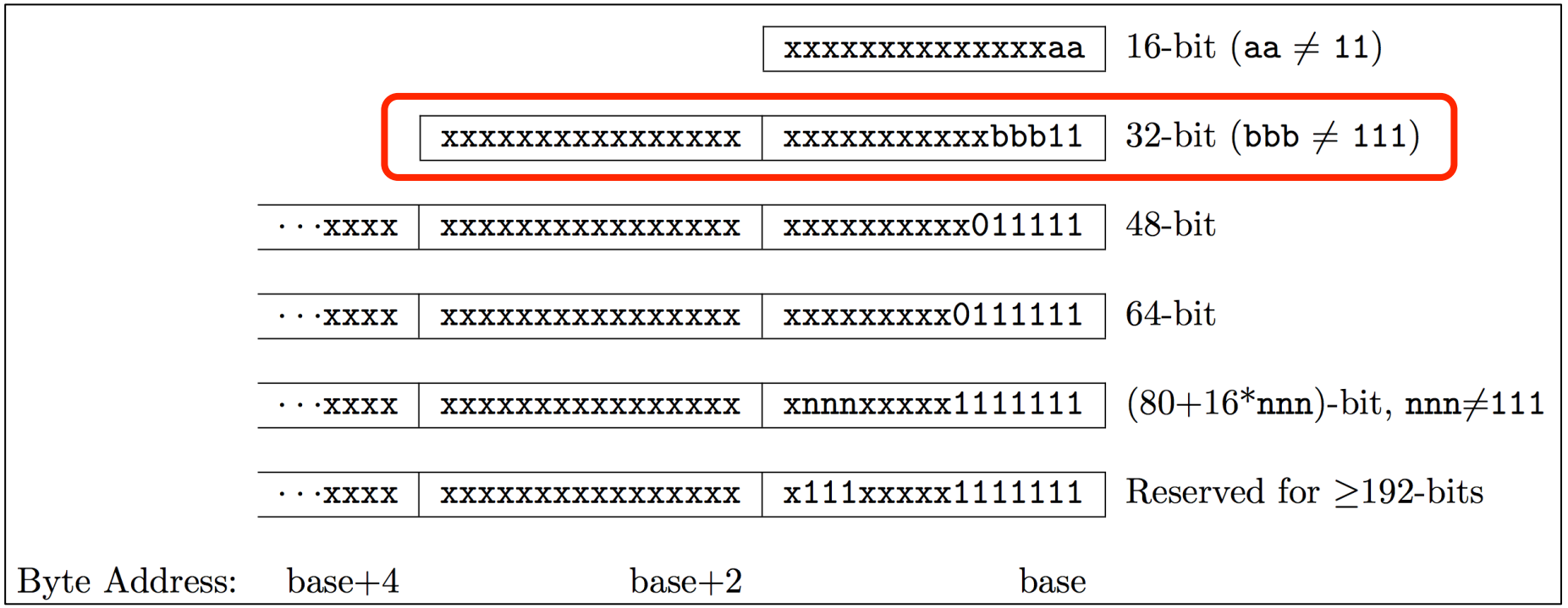
RV32 Processor State

- Program counter (**pc**)
- 32x32-bit integer registers (**x0-x31**)
  - x0 always contains a 0
  - x1 to hold the return address on a call.
- 32 floating-point (FP) registers (**f0-f31**)
  - Each can contain a single- or double-precision FP value (32-bit or 64-bit IEEE FP)
  - Is an extension
- FP status register (**fsr**), used for FP rounding mode & exception reporting



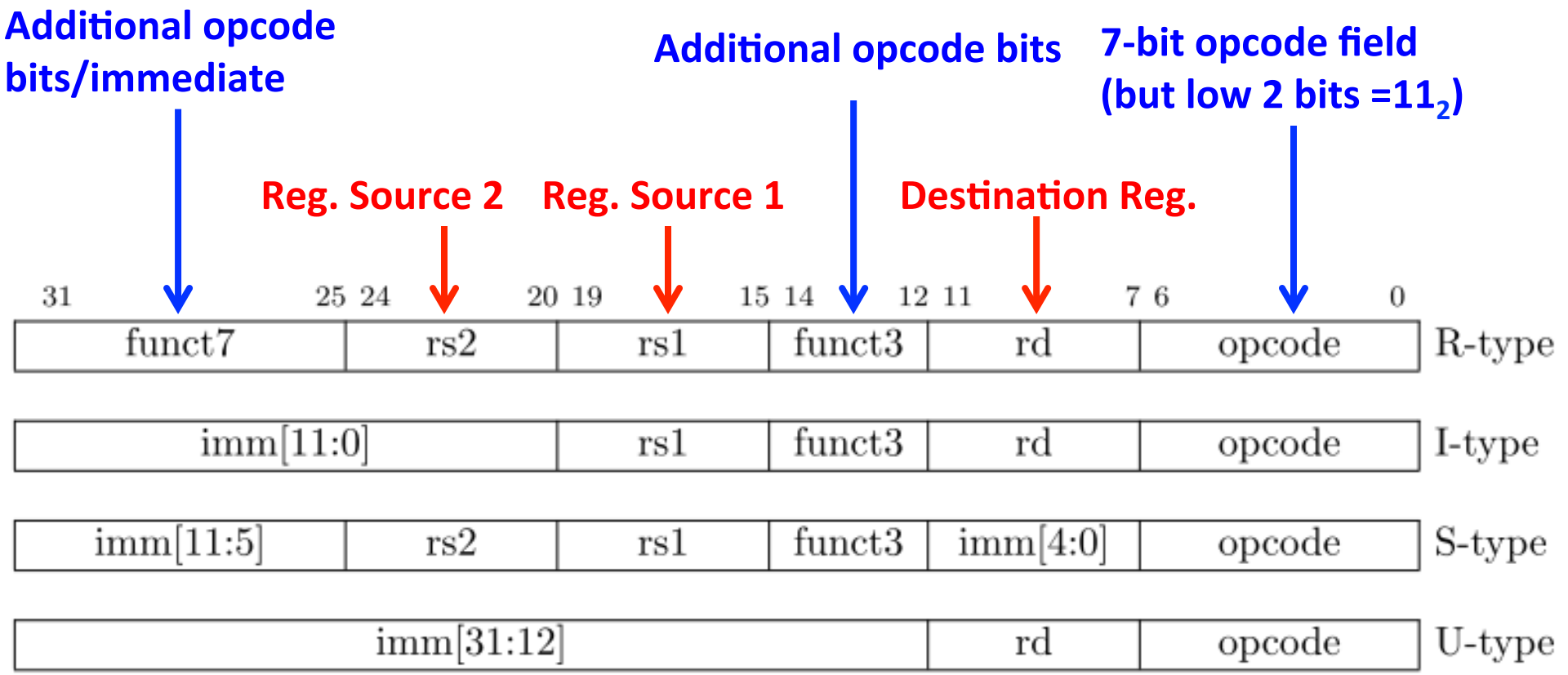
RISC-V Hybrid Instruction Encoding

- 16, 32, 48, 64 … bits length encoding
- Base instruction set (RV32) always has fixed 32-bit instructions lowest two bits = 11_2
- All branches and jumps have targets at 16-bit granularity (even in base ISA where all instructions are fixed 32 bits



Four Core RISC-V Instruction Formats

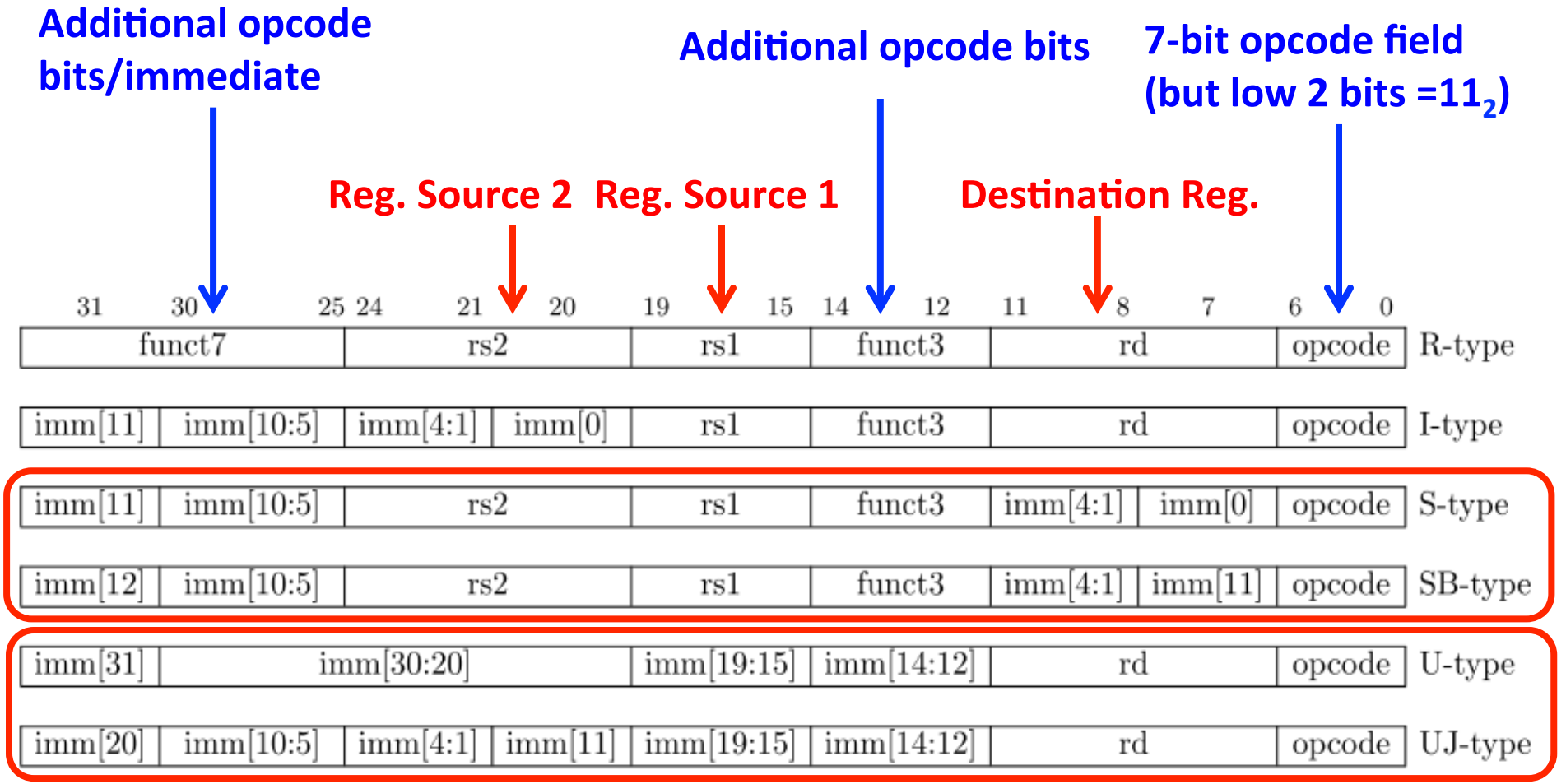
https://github.com/riscv/riscv-opcodes/blob/master/opcodes



Aligned on a four-byte boundary in memory. There are variants!

Sign bit of immediates always on bit 31 of instruction. Register fields never move

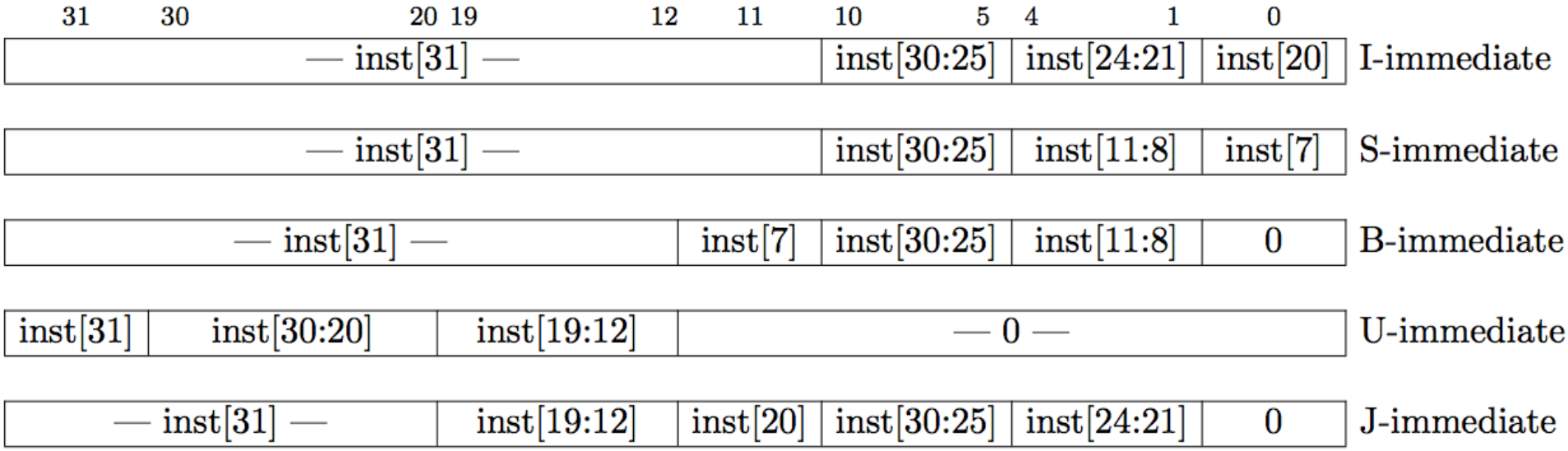
With Variants



Based on the handling of the immediates

Immediate Encoding Variants

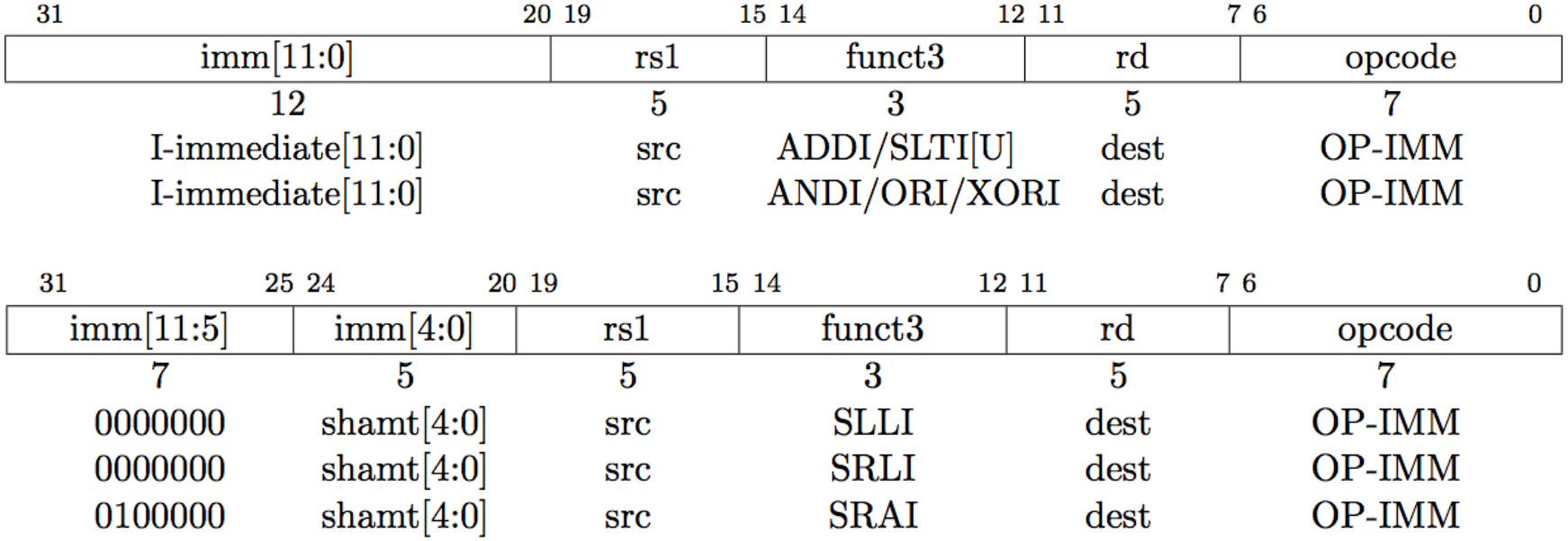
- Immediate produced by each base instruction format
  - Instruction bit (inst[y])



Integer Computational Instructions (ALU)

- **I-type (Immediate), all immediates in all instructions are sign extended**
  - **ADDI: adds sign extended 12-bit immediate to rs1**
  - **SLTI(U): set less than immediate**
  - **ANDI/ORI/XORI: Logical operations**
  - **SLLI/SRLI/SRAI: Shifts by constants**

I-type instructions end with I



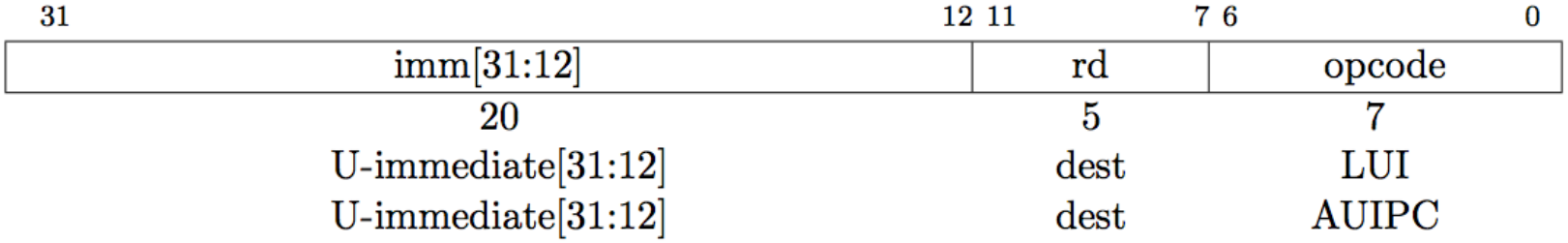
31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
I-immediate[11:0]	src	ADDI/SLTI[U]	dest	OP-IMM
I-immediate[11:0]	src	ANDI/ORI/XORI	dest	OP-IMM

31 25	24 20	19 15	14 12	11 7	6 0
imm[11:5]	imm[4:0]	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	shamt[4:0]	src	SLLI	dest	OP-IMM
0000000	shamt[4:0]	src	SRLI	dest	OP-IMM
0100000	shamt[4:0]	src	SRAI	dest	OP-IMM

Integer Computational Instructions (ALU)

- **I-type (Immediate), all immediates in all instructions are sign extended**
  - **LUI/AUIPC: load upper immediate/add upper immediate to pc**

I-type instructions end with I



31 — 12	11 — 7	6 — 0
imm[31:12]	rd	opcode
20	5	7
U-immediate[31:12]	dest	LUI
U-immediate[31:12]	dest	AUIPC

- Writes 20-bit immediate to top of destination register.
- Used to build large immediates.
- 12-bit immediates are signed, so have to account for sign when building 32-bit immediates in 2-instruction sequence (LUI high-20b, ADDI low-12b)

Integer Computational Instructions

- **R-type (Register)**
  - rs1 and rs2 are the source registers. rd the destination
  - ADD/SUB:
  - SLT, SLTU: set less than
  - SRL, SLL, SRA: shift logical or arithmetic left or right

31 25	24 20	19 15	14 12	11 7	6 0
funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	src2	src1	ADD/SLT/SLTU	dest	OP
0000000	src2	src1	AND/OR/XOR	dest	OP
0000000	src2	src1	SLL/SRL	dest	OP
0100000	src2	src1	SUB/SRA	dest	OP

NOP Instruction

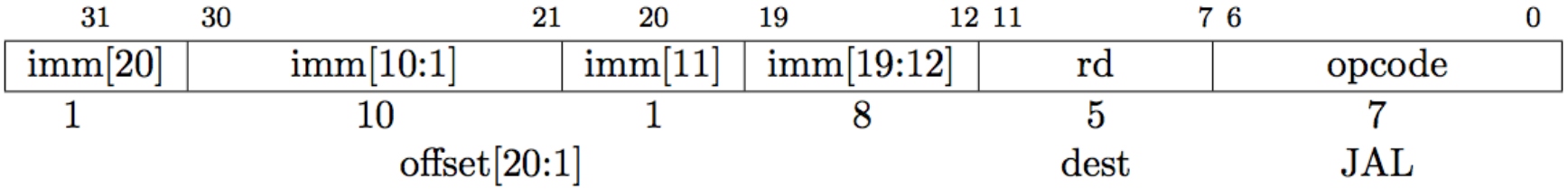
ADDI x0, x0, 0

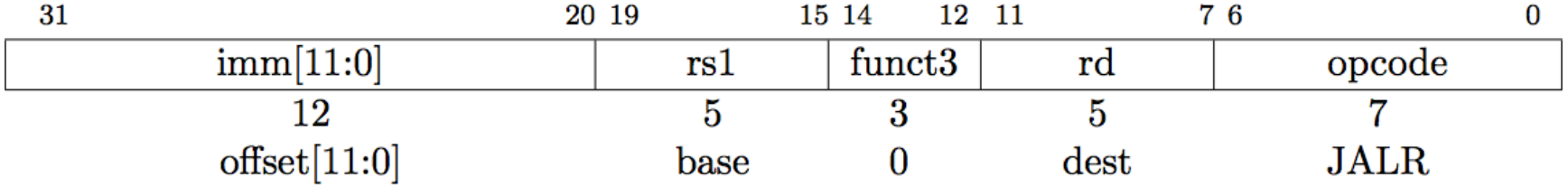
31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
0	0	ADDI	0	OP-IMM

Control Transfer Instructions

NO architecturally visible delay slots

- Unconditional Jumps: PC+offset target
  - JAL: Jump and link, also writes PC+4 to x1, UJ-type
    - Offset scaled by 1-bit left shift – can jump to 16-bit instruction boundary (Same for branches)
  - JALR: Jump and link register where Imm (12 bits) + rd1 = target

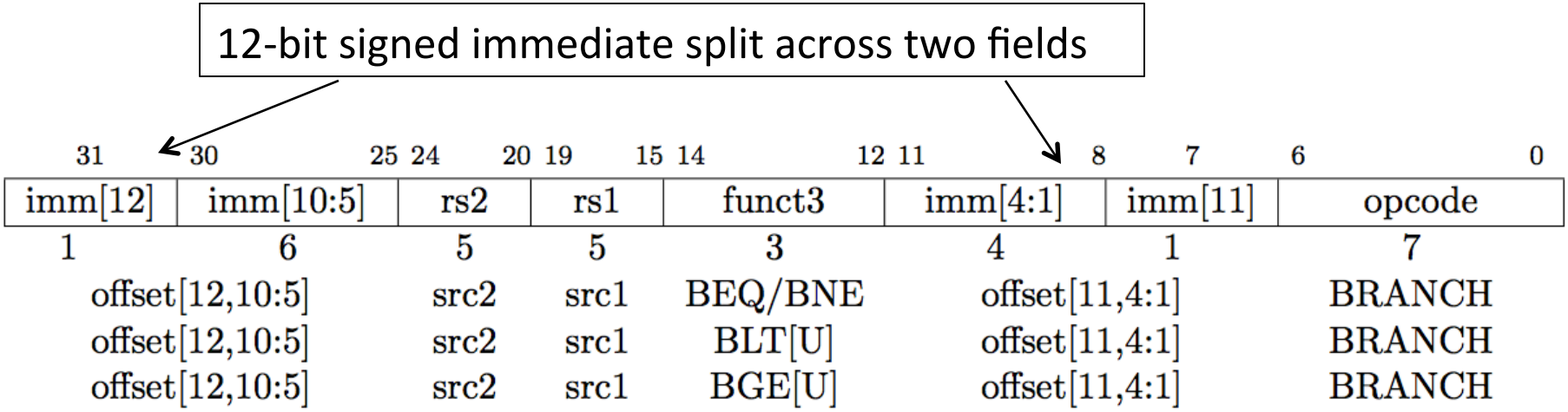




Control Transfer Instructions

NO architecturally visible delay slots

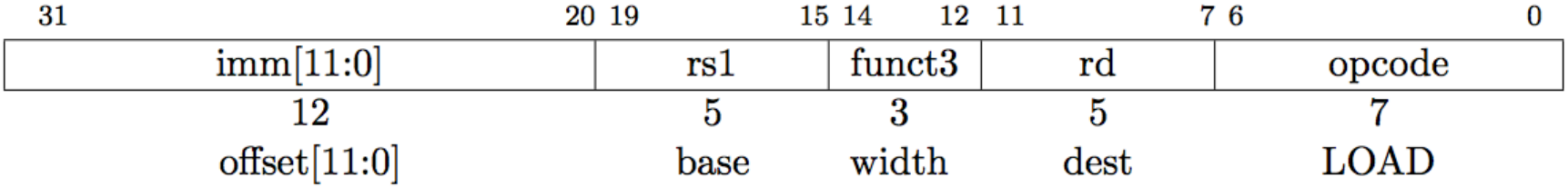
- Conditional Branches: SB-type and PC+offset target

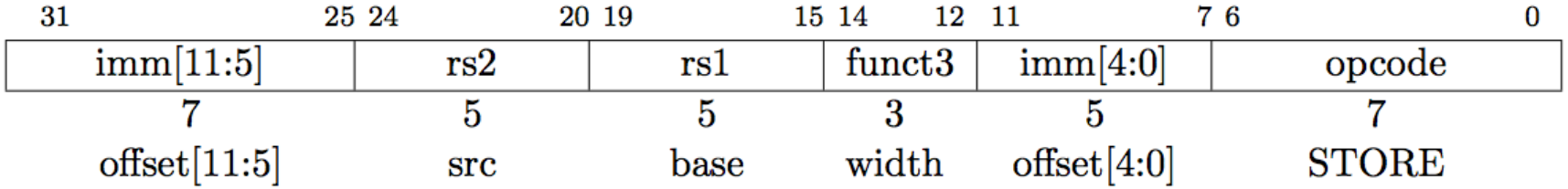


Branches, compare two registers, PC+(immediate<<1) target
*(Signed offset in multiples of two).*Branches do not have delay slot

Loads and Stores

- Store instructions (S-type)
  - MEM(rs1+imm) = rs2
- Loads (I-type)
  - Rd = MEM(rs1 + imm)

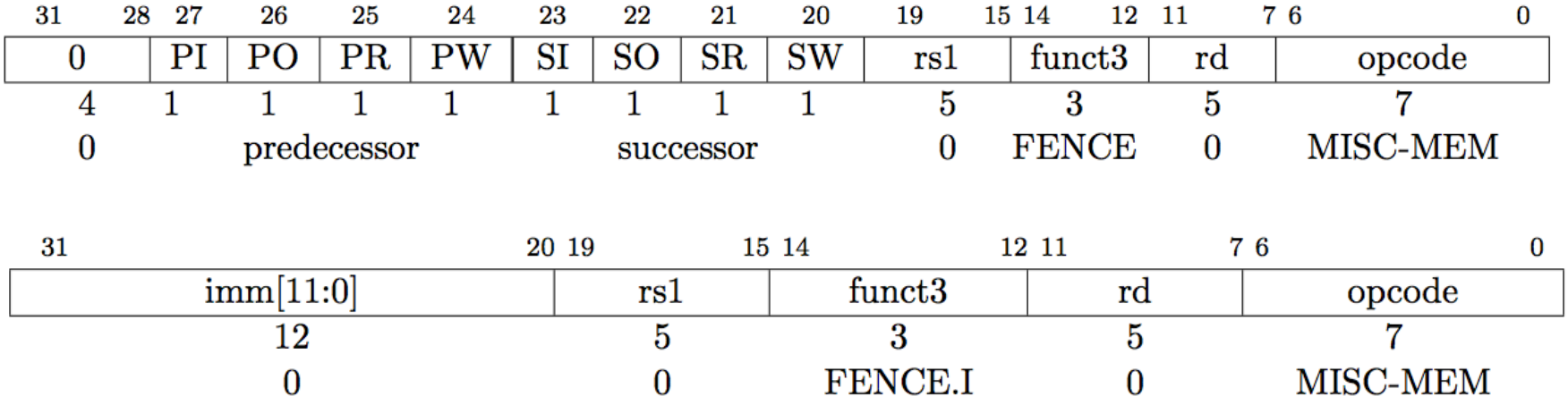




More

Memory Model

- RISC-V: Relaxed memory model



Control and Status Register (CSR) Instructions

- CSR Instructions

31 20	19 15	14 12	11 7	6 0
csr	rs1	funct3	rd	opcode
12	5	3	5	7
source/dest	source	CSRRW	dest	SYSTEM
source/dest	source	CSRRS	dest	SYSTEM
source/dest	source	CSRRC	dest	SYSTEM
source/dest	zimm[4:0]	CSRRWI	dest	SYSTEM
source/dest	zimm[4:0]	CSRRSI	dest	SYSTEM
source/dest	zimm[4:0]	CSRRCI	dest	SYSTEM

- Timer and counters

31 20	19 15	14 12	11 7	6 0
csr	rs1	funct3	rd	opcode
12	5	3	5	7
RDCYCLE[H]	0	CSRRS	dest	SYSTEM
RDTIME[H]	0	CSRRS	dest	SYSTEM
RDINSTRET[H]	0	CSRRS	dest	SYSTEM

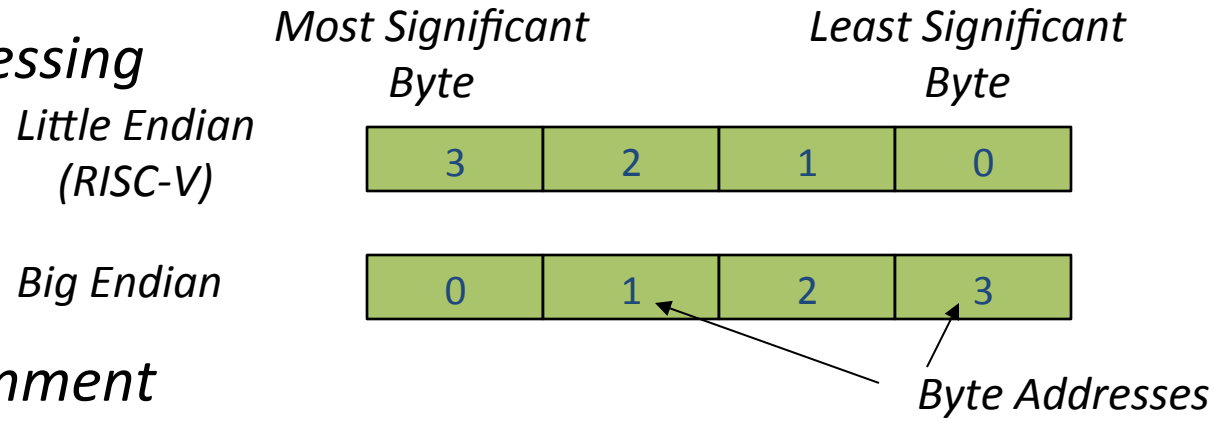
Data Formats and Memory Addresses

Data formats:

8-b Bytes, 16-b Half words, 32-b words and 64-b double words

Some issues

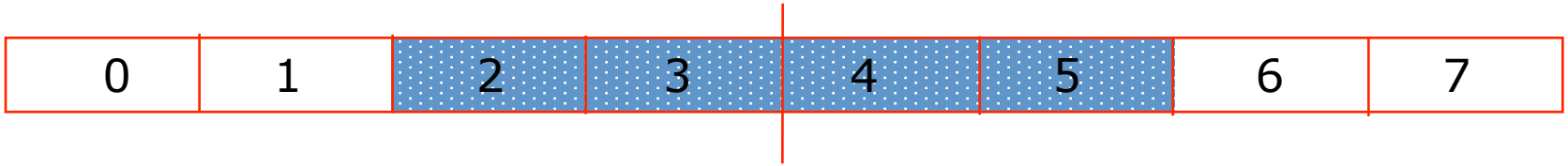
- *Byte addressing*



- *Word alignment*

  Suppose the memory is organized in 32-bit words.

  Can a word address begin only at 0, 4, 8, ... ?



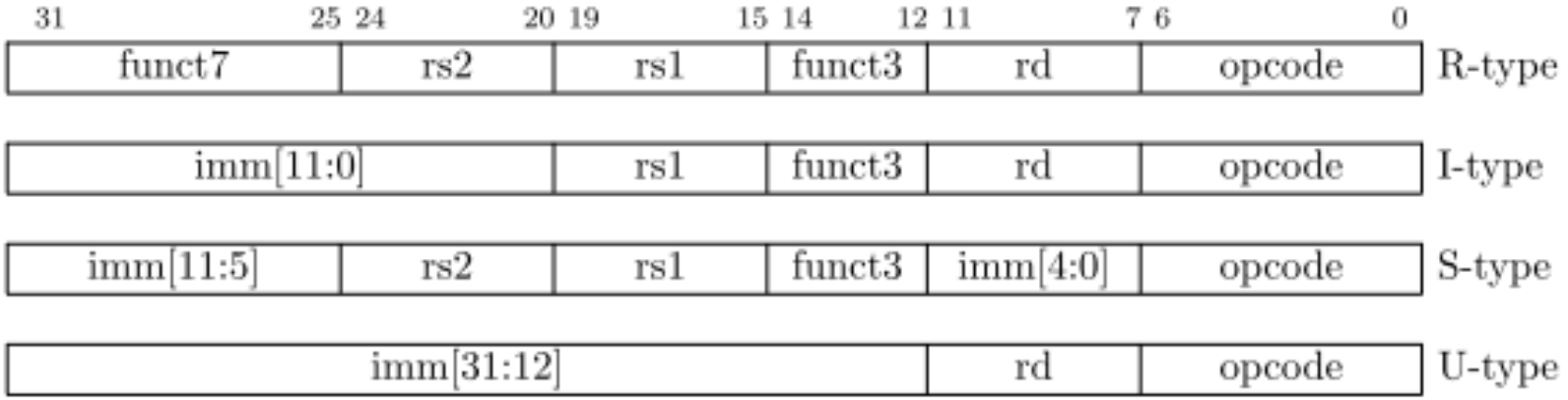
ISA Design

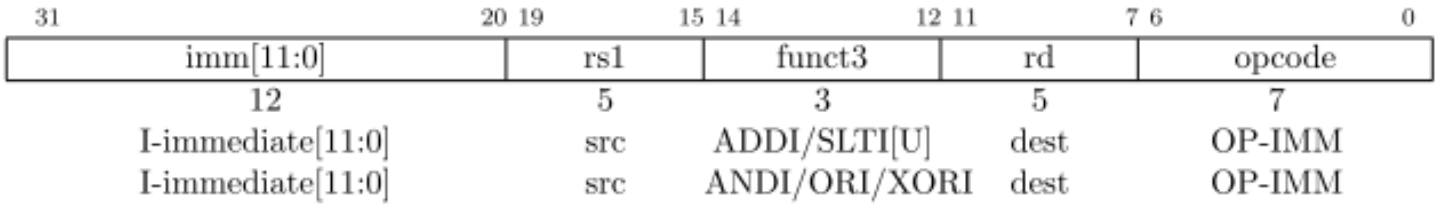
- RISC-V has 32 integer registers and can have 32 floating-point registers
  - Register number 0 is a constant 0
  - Register number 1 is the return address (link register)
- The memory is addressed by 8-bit bytes
- The instructions must be aligned to 32-bit addresses
- Like many RISC designs, it is a "load-store" machine
  - The only instructions that access main memory are loads and stores
  - All arithmetic and logic operations occur between registers
- RISC-V can load and store 8 and 16-bit items, but it lacks 8 and 16-bit arithmetic, including comparison-and-branch instructions
- The 64-bit instruction set includes 32-bit arithmetic

inst[4:2] / inst[6:5]	000	001	010	011	100	101	110	111 (> 32b)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	48b
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	64b
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2/rv128*	48b
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3/rv128*	≥ 80b

ISA Design for Performance

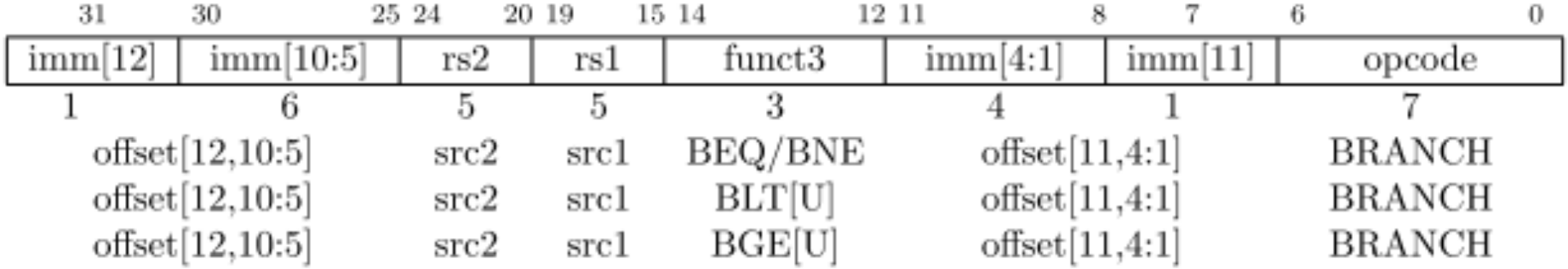
- Features to increase a computer's speed, while reducing its cost and power usage
  - placing most-significant bits at a fixed location to speed sign-extension, and a bit-arrangement designed to reduce the number of multiplexers in a CPU





ISA Design

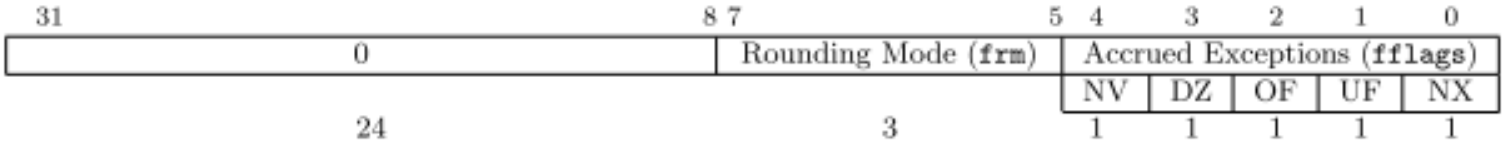
- Intentionally lacks condition codes, and even lacks a carry bit
  - To simplify CPU designs by minimizing interactions between instructions
- Builds comparison operations into its conditional-jumps



31	30 25	24 20	19 15	14 12	11 8	7	6 0
imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
1	6	5	5	3	4	1	7
offset[12,10:5]		src2	src1	BEQ/BNE	offset[11,4:1]		BRANCH
offset[12,10:5]		src2	src1	BLT[U]	offset[11,4:1]		BRANCH
offset[12,10:5]		src2	src1	BGE[U]	offset[11,4:1]		BRANCH

ISA Design

- The lack of a carry bit complicates multiple-precision arithmetic
  - GMP, MPFR
- does not detect or flag most arithmetic errors, including overflow, underflow and divide by zero
  - No special instruction set support for overflow checks on integer arithmetic operations.
    - Most popular programming languages do not support checks for integer overflow, partly because most architectures impose a significant runtime penalty to check for overflow on integer arithmetic and partly because modulo arithmetic is sometimes the desired behavior
  - Floating-Point Control and Status Register



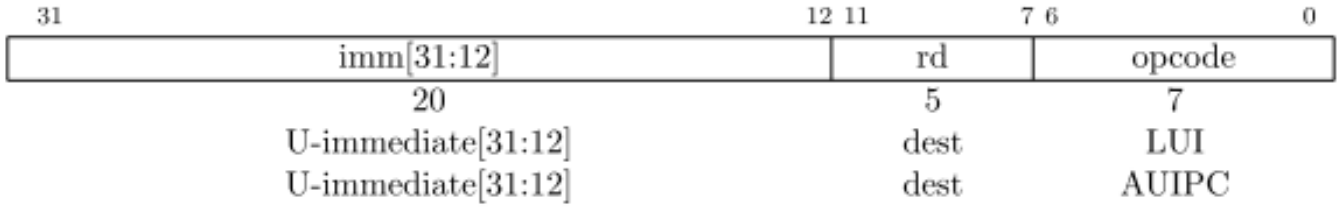
Flag Mnemonic	Flag Meaning
NV	Invalid Operation
DZ	Divide by Zero
OF	Overflow
UF	Underflow
NX	Inexact

ISA Design

- Lacks the "count leading zero" and bit-field operations normally used to speed software floating-point in a pure-integer processor
- No branch delay slot, a position after a branch instruction that can be filled with an instruction which is executed regardless of whether the branch is taken or not
  - This feature can improve performance of pipelined processors,
  - Omitted in RISC-V because it complicates both multicycle CPUs and superscalar CPUs
- Lacks address-modes that "write back" to the registers
  - For example, it does not do auto-incrementing

ISA Design

- A load or store can add a twelve-bit signed offset to a register that contains an address. A further 20 bits (yielding a 32-bit address) can be generated at an absolute address
  - RISC-V was designed to permit position-independent code. It has a special instruction to generate 20 upper address bits that are relative to the program counter. The lower twelve bits are provided by normal loads, stores and jumps



  - *LUI (load upper immediate) places the U-immediate value in the top 20 bits of the destination register rd, filling in the lowest 12 bits with zeros*
  - *AUIPC (add upper immediate to pc) is used to build pc-relative addresses, forms a 32-bit offset from the 20-bit U-immediate, filling in the lowest 12 bits with zeros, adds this offset to the pc, then places the result in register rd*

ISA Design

- The RISC-V instruction set was designed for research, and therefore includes extra space for new instructions
  - Planned instruction subsets include system instructions, atomic access, integer multiplication, floating-point arithmetic, bit-manipulation, decimal floating-point, multimedia and vector processing

  - It includes instructions for 32-bit, 64-bit and 128-bit integer and floating-point
  - It was designed for 32-bit, 64-bit and 128-bit memory systems, with 32-bit models designed for lower power, 64-bit for higher performance, and 128-bit for future requirements
  - It's designed to operate with hypervisors, supporting virtualization
  - It was designed to conform to recent floating-point standards

Subset	Name
Standard General-Purpose ISA	
Integer	I
Integer Multiplication and Division	M
Atomics	A
Single-Precision Floating-Point	F
Double-Precision Floating-Point	D
General	G = IMAFD
Standard User-Level Extensions	
Quad-Precision Floating-Point	Q
Decimal Floating-Point	L
16-bit Compressed Instructions	C
Bit Manipulation	B
Transactional Memory	T
Packed-SIMD Extensions	P
Non-Standard User-Level Extensions	
Non-standard extension "abc"	Xabc
Standard Supervisor-Level ISA	
Supervisor extension "def"	Sdef
Non-Standard Supervisor-Level Extensions	
Supervisor extension "ghi"	SXghi

Calling Convention

- C Datatypes and Alignment
  - RV32 employs an ILP32 integer model, while RV64 is LP64
  - Floating-point types are IEEE 754-2008 compatible
  - All of the data types are keeped naturally aligned when stored in memory
  - char is implicitly unsigned
  - In RV64, 32-bit types, such as int, are stored in integer registers as proper sign extensions of their 32-bit values; that is, bits 63..31 are all equal
    - This restriction holds even for unsigned 32-bit types

C type	Description	Bytes in RV32	Bytes in RV64
`char`	Character value/byte	1	1
`short`	Short integer	2	2
`int`	Integer	4	4
`long`	Long integer	4	8
`long long`	Long long integer	8	8
`void*`	Pointer	4	8
`float`	Single-precision float	4	4
`double`	Double-precision float	8	8
`long double`	Extended-precision float	16	16

Calling Convention

- RVG Calling Convention
  - If the arguments to a function are conceptualized as fields of a C struct, each with pointer alignment, the argument registers are a shadow of the first eight pointer-words of that struct
    - Floating-point arguments that are part of unions or array fields of structures are passed in integer registers
    - Floating-point arguments to variadic functions (except those that are explicitly named in the parameter list) are passed in integer registers
  - The portion of the conceptual struct that is not passed in argument registers is passed on the stack
    - The stack pointer sp points to the first argument not passed in a register
  - Arguments smaller than a pointer-word are passed in the least-significant bits of argument registers
  - When primitive arguments twice the size of a pointer-word are passed on the stack, they are naturally aligned
    - When they are passed in the integer registers, they reside in an aligned even-odd register pair, with the even register holding the least-significant bits
  - Arguments more than twice the size of a pointer-word are passed by reference

Calling Convention

- The stack grows downward and the stack pointer is always kept 16-byte aligned
- Values are returned from functions in integer registers v0 and v1 and floating-point registers fv0 and fv1
  - Floating-point values are returned in floating-point registers only if they are primitives or members of a struct consisting of only one or two floating-point values
  - Other return values that fit into two pointer-words are returned in v0 and v1
  - Larger return values are passed entirely in memory; the caller allocates this memory region and passes a pointer to it as an implicit first parameter to the callee

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	s0/fp	Saved register/frame pointer	Callee
x3–13	s1–11	Saved registers	Callee
x14	sp	Stack pointer	Callee
x15	tp	Thread pointer	Callee
x16–17	v0–1	Return values	Caller
x18–25	a0–7	Function arguments	Caller
x26–30	t0–4	Temporaries	Caller
x31	gp	Global pointer	—
f0–15	fs0–15	FP saved registers	Callee
f16–17	fv0–1	FP return values	Caller
f18–25	fa0–7	FP arguments	Caller
f26–31	ft0–5	FP temporaries	Caller

Software

- The RISC-V website has a specification for user-mode instructions, a draft for privileged ISA specification and a draft for compressed ISA specification
- It also includes the files of six CPU designs, the 64-bit superscalar "Rocket" and five "Sodor" CPUs
- The software includes a design compiler, Chisel, which is able to reduce the designs to Verilog for use in devices
- The website includes verification data for testing core implementations
- Available RISC-V software includes a GNU Compiler Collection (GCC) toolchain (with GDB, the debugger), an LLVM toolchain, a simulator ("Spike") and the standard simulator QEMU
- Operating systems support exists for Linux, but the supervisor-mode instructions are not standardized at this time
- There is also a JavaScript ISA simulator to run a RISC-V Linux system on a web browser

Implementations

- The RISC-V ISA has been designed to result in faster, less-expensive, smaller, and less-power-hungry electronics.
- It is carefully designed not to make assumptions about the structure of the computers on which it runs.
  - Validating this, the UCB Sodor cores were implemented as different types of computers
- RISC-V is designed to be extensible from a 32-bit bare bones integer core suitable for a small embedded processor to 64 or 128-bit super and cloud computers with standard and special purpose extensions.
  - It has been tested in a fast pipelined silicon design with the open Rocket SoC.
- The UCB processor designs that implement RISC-V are implemented using Chisel, an open-source hardware construction language that is a specialized dialect of Scala.
  - 'Chisel' is an abbreviation: Constructing Hardware In a Scala Embedded Language

Implementations

- The Indian Institute of Technology Madras is developing six RISC-V open-source CPU designs (SHAKTI) for six distinct usages, from a small 32-bit CPU for the Internet of Things to large, 64-bit CPUs designed for warehouse-scale computers based on RapidIO and Hybrid Memory Cube technologies.
- Bluespec, Inc., a semiconductor tools company, is exploring RISC-V as a possible product.
- lowRISC is a non profit project that aims to implement a fully open-source SoC based on the 64-Bit RISC-V ISA.
- The planned multimedia set may include a general-purpose mixed-precision vector processor similar to the research project "Hwacha."

Resources

- http://riscv.org/workshop-jan2015.html
- http://riscv.org/tutorial-hpca2015.html

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

- Check the class resource page