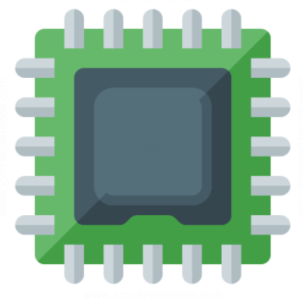
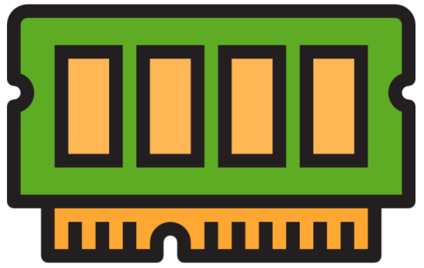
# ECE 222 Digital Computers



## RISC-V Instruction Formats, Part I



Ziqiang Patrick Huang

Electrical and Computer Engineering

University of Waterloo

WATERLOO | ENGINEERING

# Recall: Abstraction

**High Level Language Program (e.g., C)**

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

*Compiler*

**Assembly Language Program (e.g., RISC-V)**

```
lw    x3, 0(x10)
lw    x4, 4(x10)
sw    x4, 0(x10)
sw    x3, 4(x10)
```

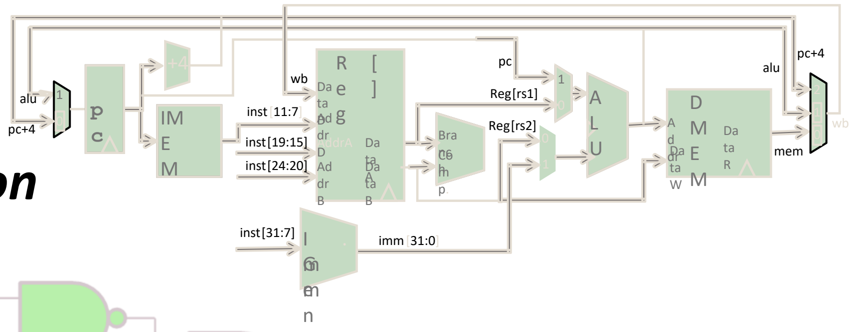
Anything can be represented as a number, i.e., data or instructions

*Assembler*

**Machine Language Program (RISC-V)**

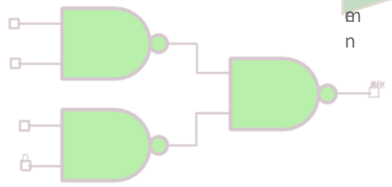
```
1000 1101 1110 0010 0000 0000 0000 0000
1000 1110 0001 0000 0000 0000 0000 0100
1010 1110 0001 0010 0000 0000 0000 0000
1010 1101 1110 0010 0000 0000 0000 0100
```

**Hardware Architecture Description (e.g., block diagrams)**



*Architecture Implementation*

**Logic Circuit Description (Circuit Schematic Diagrams)**



WATERLOO | ENGINEERING

## Instructions Are 32 Bits Wide

- ISA defines instructions for CPU down to the bit level:

`add x18, x19, x10` → `00000001010100110000100100110011`

assembly code — *machine* code

  - Remember: Computer only understands 1s and 0s
    Text like "`add x18,x19,x10`" is meaningless to hardware
- Most data we work with is in words (32-bit chunks)
  - 32-bit registers; `lw`, `sw` access memory one word at a time
- Similarly, RISC-V instructions are fixed size, 32-bit words
  - Simply instruction fetching and decoding
  - Same 32-bit instructions used for RV32, RV64, RV128

## Instruction Formats

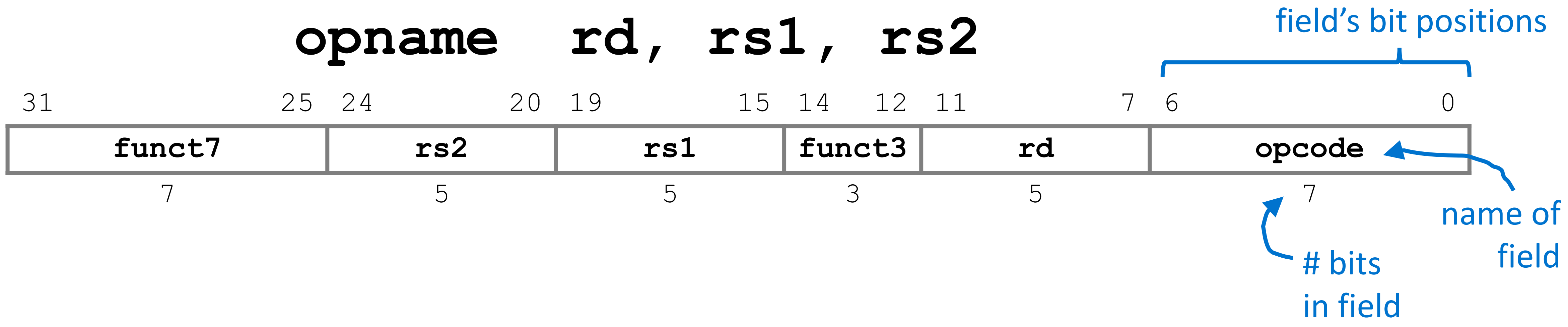
- RISC-V's 32b instruction words are divided into fields
  - Each field tells processor something about the instruction
- The RISC-V ISA defines six basic types of instruction formats
  - RISC-V seeks simplicity: Avoid defining different fields for each instruction; instead, use same format for similar instructions

| | |
|---|---|
| R-Format | Register-register arithmetic operations |
| I-Format | Register-immediate arithmetic operations; Loads |
| S-Format | Stores |
| B-Format | Branches (minor variant of S-format) |
| U-Format | 20-bit upper immediate instructions |
| J-Format | Jumps (minor variant of U-format) |

## R-Format Instruction Layout

- Register-Register Arithmetic Instructions (`add`, `xor`, `sll`, etc.)

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

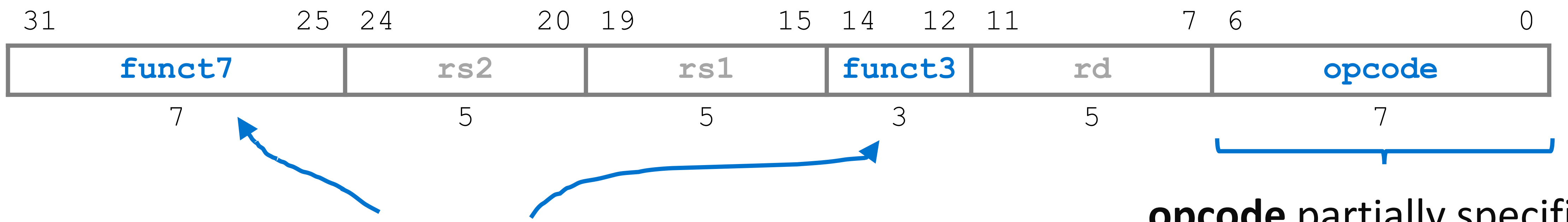


WATERLOO | ENGINEERING

## R-Format Instruction Layout

- Register-Register Arithmetic Instructions (`add`, `xor`, `sll`, etc.)

`opname rd, rs1, rs2`



**funct7, funct3** combined with opcode describes what operation to perform.

Why not just a 17-bit field for simplicity?
We'll answer this later…

**opcode** partially specifies which instruction it is.

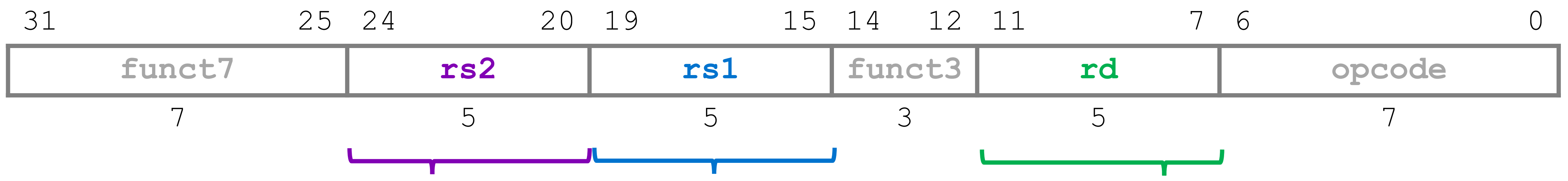
All R-Format instructions have opcode `0110011`.

WATERLOO | ENGINEERING

## R-Format Instruction Layout

- Register-Register Arithmetic Instructions (**`add`**, **`xor`**, **`sll`**, etc.)

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



"Source" Register 2
contains second operand

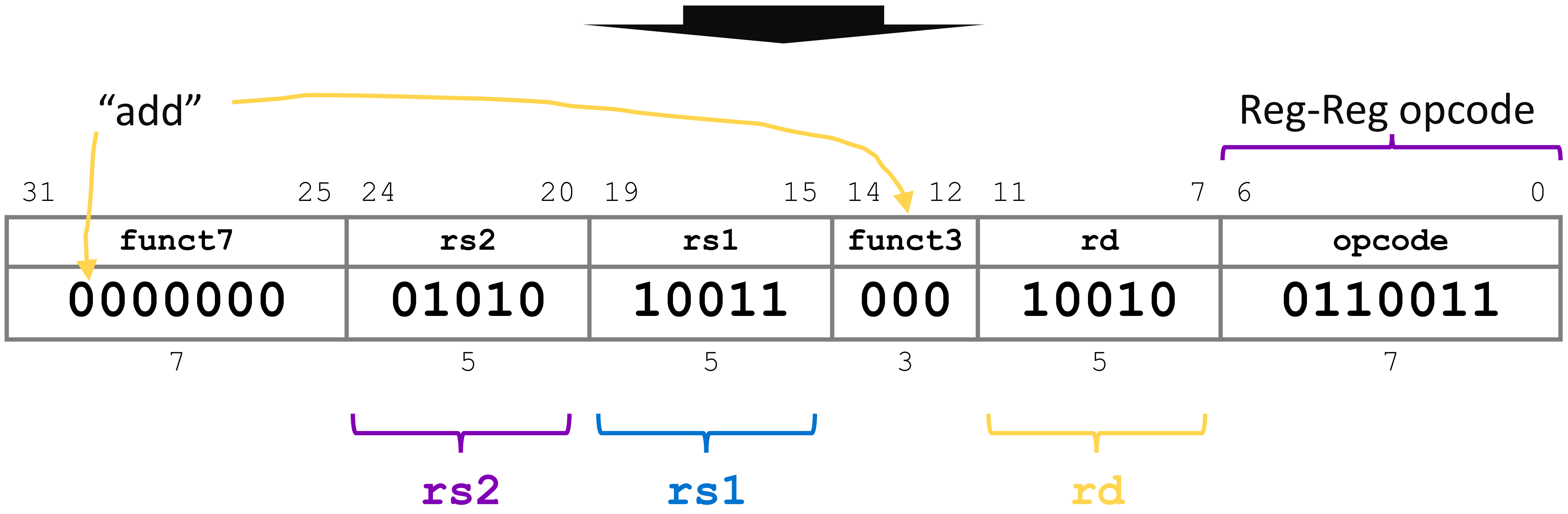
"Source" Register 1
contains first operand

"Destination" Register
gets result of computation

Register field **(`rs1`, `rs2`, `rd`)** holds a 5-bit unsigned integer **[0-31]** corresponding to a register number **(x0-x31)**.

# R-Format Example

```
add  x18, x19, x10
```



WATERLOO | ENGINEERING

# All ten RV32 R-Format Instructions

Eight funct3 fields for ten instructions

2's comp negation of rs2

sign extend

| funct7 | | | funct3 | | opcode | |
|---|---|---|---|---|---|---|
| 0000000 | rs2 | rs1 | 000 | rd | 0110011 | add |
| 0100000 | rs2 | rs1 | 000 | rd | 0110011 | sub |
| 0000000 | rs2 | rs1 | 001 | rd | 0110011 | sll |
| 0000000 | rs2 | rs1 | 010 | rd | 0110011 | slt |
| 0000000 | rs2 | rs1 | 011 | rd | 0110011 | sltu |
| 0000000 | rs2 | rs1 | 100 | rd | 0110011 | xor |
| 0000000 | rs2 | rs1 | 101 | rd | 0110011 | srl |
| 0100000 | rs2 | rs1 | 101 | rd | 0110011 | sra |
| 0000000 | rs2 | rs1 | 110 | rd | 0110011 | or |
| 0000000 | rs2 | rs1 | 111 | rd | 0110011 | and |

set less than (see lab manual)

Different funct7 & funct3 encodings select different operations.

# In-class Exercise

How do we encode `add x4, x3, x2` ?

Lookup table:

| | | | | | | |
|---|---|---|---|---|---|---|
| 0000000 | rs2 | rs1 | 000 | rd | 0110011 | add |
| 0100000 | rs2 | rs1 | 000 | rd | 0110011 | sub |
| 0000000 | rs2 | rs1 | 100 | rd | 0110011 | xor |
| 0000000 | rs2 | rs1 | 111 | rd | 0110011 | and |

A. $4021\ 8233_{hex}$
B. $0021\ 82b3_{hex}$
C. $4021\ 82b3_{hex}$
D. $0021\ 8233_{hex}$
E. $0021\ 8234_{hex}$
F. Something else

| 31 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|---|
| funct7 | rs2 | rs1 | funct3 | rd | opcode |
| | | | | | |

WATERLOO | ENGINEERING

# Immediate Fields Need to be Wider

- R-Format: **`add rd, rs1, rs2`**

| 31 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|---|
| **funct7** | **rs2** | **rs1** | **funct3** | **rd** | **opcode** |
| 7 | 5 | 5 | 3 | 5 | 7 |

- What about instructions with immediates: **`add rd, rs1, imm`** ?
  - Use R-format ? (represent **`imm`** in the 5-bit **`rs2`** field?)
  - Immediates may be (and are often) much bigger!
- Introduce the I-Format:

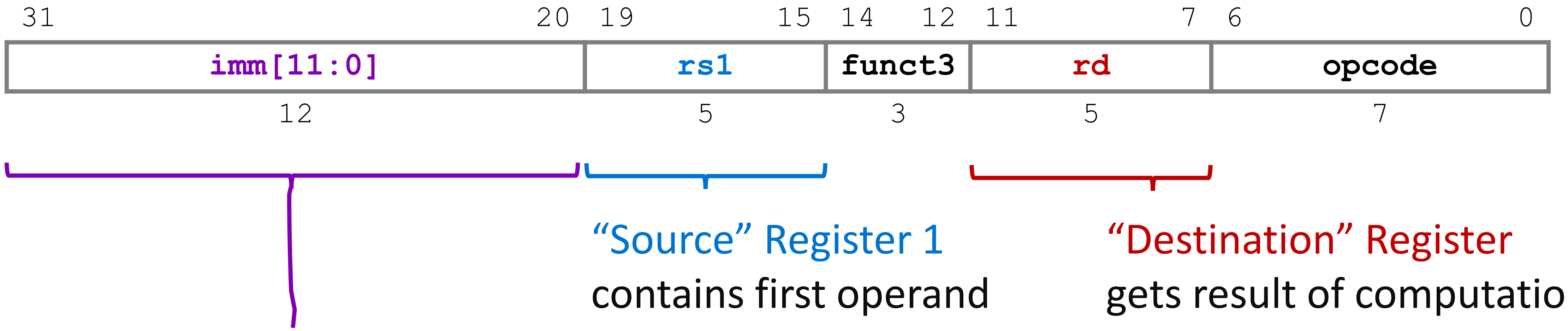
| 31 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|
| **imm[11:0]** | **rs1** | **funct3** | **rd** | **opcode** |
| 12 | 5 | 3 | 5 | 7 |

I-Format is *mostly* consistent with R-Format. Simplify how the CPU processes instructions!

# I-Format Instruction Layout

- Register-Immediate Arithmetic Instructions (**`addi`**, **`xori`**, etc.)

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



“Source” Register 1
contains first operand

“Destination” Register
gets result of computation

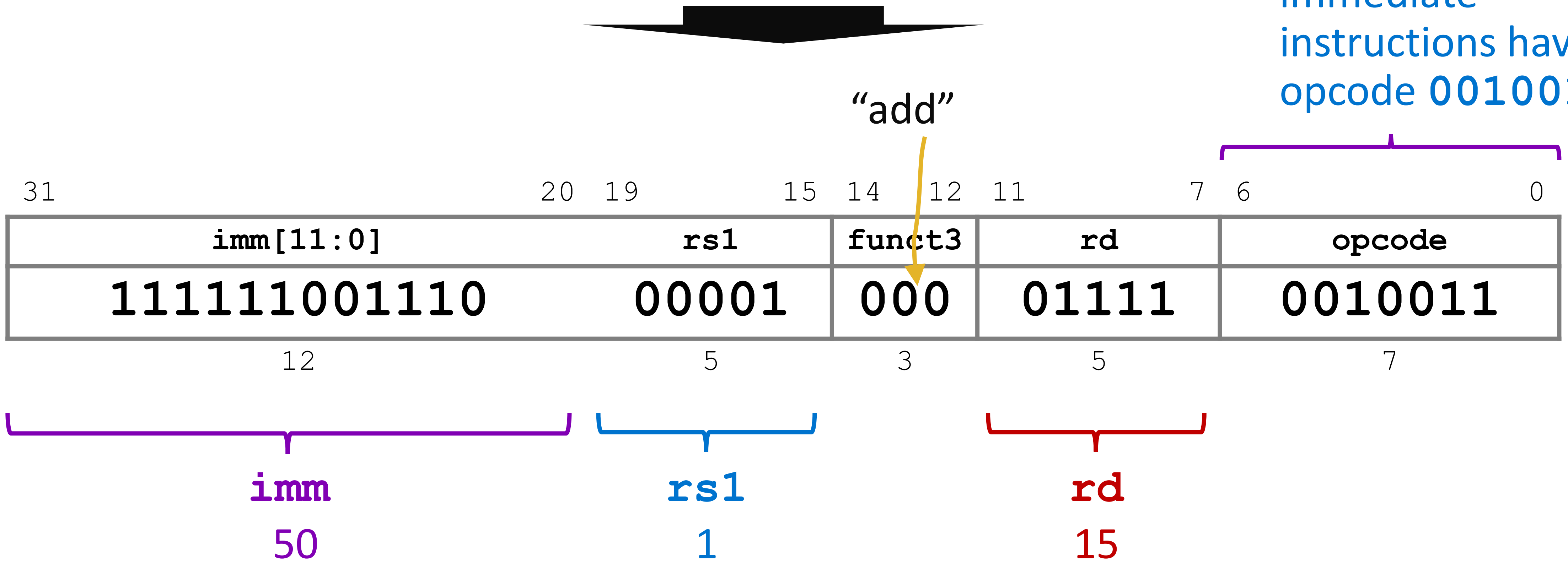
**imm[11:0]** holds 12-bit-wide immediate values:

- Values in range [-2048$_{ten}$ , +2047$_{ten}$]
- CPU sign-extends to 32 bits before use in an arithmetic operation
- How to handle immediates > 12 bits? (more next time)

# I-Format Example

```
addi x15, x1, -50
```

All arithmetic immediate instructions have opcode **0010011**.



| imm[11:0] | rs1 | funct3 | rd | opcode |
|---|---|---|---|---|
| 111111001110 | 00001 | 000 | 01111 | 0010011 |

# All nine RV32 I-Format Arithmetic Instructions

Same funct3 fields as corresponding R-format operation (remember, no **subi**)

| | | | funct3 | | opcode | |
|---|---|---|---|---|---|---|
| imm[11:0] | | rs1 | 000 | rd | 0010011 | addi |
| imm[11:0] | | rs1 | 010 | rd | 0010011 | slti |
| imm[11:0] | | rs1 | 011 | rd | 0010011 | sltiu |
| imm[11:0] | | rs1 | 100 | rd | 0010011 | xori |
| imm[11:0] | | rs1 | 110 | rd | 0010011 | ori |
| imm[11:0] | | rs1 | 111 | rd | 0010011 | andi |
| 0000000 | shamt | rs1 | 001 | rd | 0010011 | slli |
| 0000000 | shamt | rs1 | 101 | rd | 0010011 | srli |
| 0100000 | shamt | rs1 | 101 | rd | 0010011 | srai |

"Shift by Immediate" instructions encode the **shift amount** in the lower-order 5 bits of the imm.

- We can only (meaningfully) shift 32-b word by 0-31 positions.
- One higher-order immediate bit used for sign extend (**srli** vs. **srai**). Same bit position as in R-Format!

# Load Instructions Are Also I-Format: Example

```
lw x14, 8(x2)
```

"load word"

Load opcode

| | | | | |
|---|---|---|---|---|
| 31 … 20 | 19 … 15 | 14 … 12 | 11 … 7 | 6 … 0 |
| imm[11:0] | rs1 | funct3 | rd | opcode |
| 000000001000 | 00010 | 010 | 01110 | 0000011 |
| 12 | 5 | 3 | 5 | 7 |

**imm ("offset")**
+8

**rs1 ("base")**

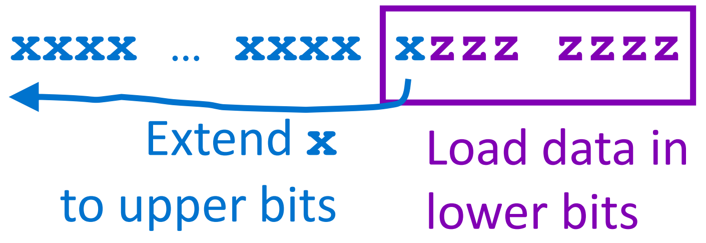
**rd**
14

Load address = (Base Register) + (Immediate Offset)

# All five RV32 Load Instructions

Encodes data size and "signedness" of load operation

| | | funct3 | | opcode | |
|---|---|---|---|---|---|
| imm[11:0] | rs1 | 000 | rd | 0000011 | lb |
| imm[11:0] | rs1 | 001 | rd | 0000011 | lh |
| imm[11:0] | rs1 | 010 | rd | 0000011 | lw |
| imm[11:0] | rs1 | 100 | rd | 0000011 | lbu |
| imm[11:0] | rs1 | 101 | rd | 0000011 | lhu |



`lb`: "load byte," `lh` "load halfword" (16 bits)

*Sign extend* to fill upper bits of destination 32-bit register.

`lbu`: "load unsigned byte," `lhu` "load unsigned halfword"

*Zero extend* to fill upper bits of destination 32-bit register.

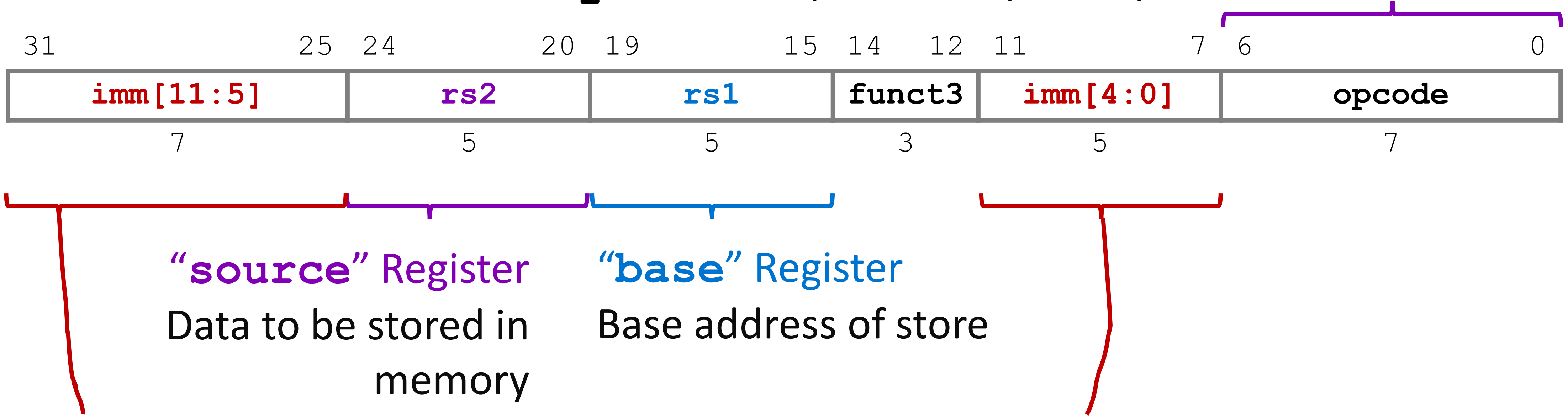
Note no `lwu` instruction in RISC-V.

# S-Format Instruction Layout

- Store instructions have their S-Format.

**`storeop  rs2, imm(rs1)`**

All store instructions have opcode **`0100011`**.



"**source**" Register
Data to be stored in memory

"**base**" Register
Base address of store

Immediate "**offset**" added to base address to form memory address.
Store address = (Base Register) + (Immediate Offset)

- The immediate's higher 7 bits and lower 5 bits are in separate fields!

# S-Format Simplifies Hardware Design

- Why split up the immediate field? `storeop rs2, imm(rs1)`

| 31 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|---|
| imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode |
| 7 | 5 | 5 | 3 | 5 | 7 |

RISC-V design prioritizes keeping *register fields* in the same places. Immediates are less critical to hardware.

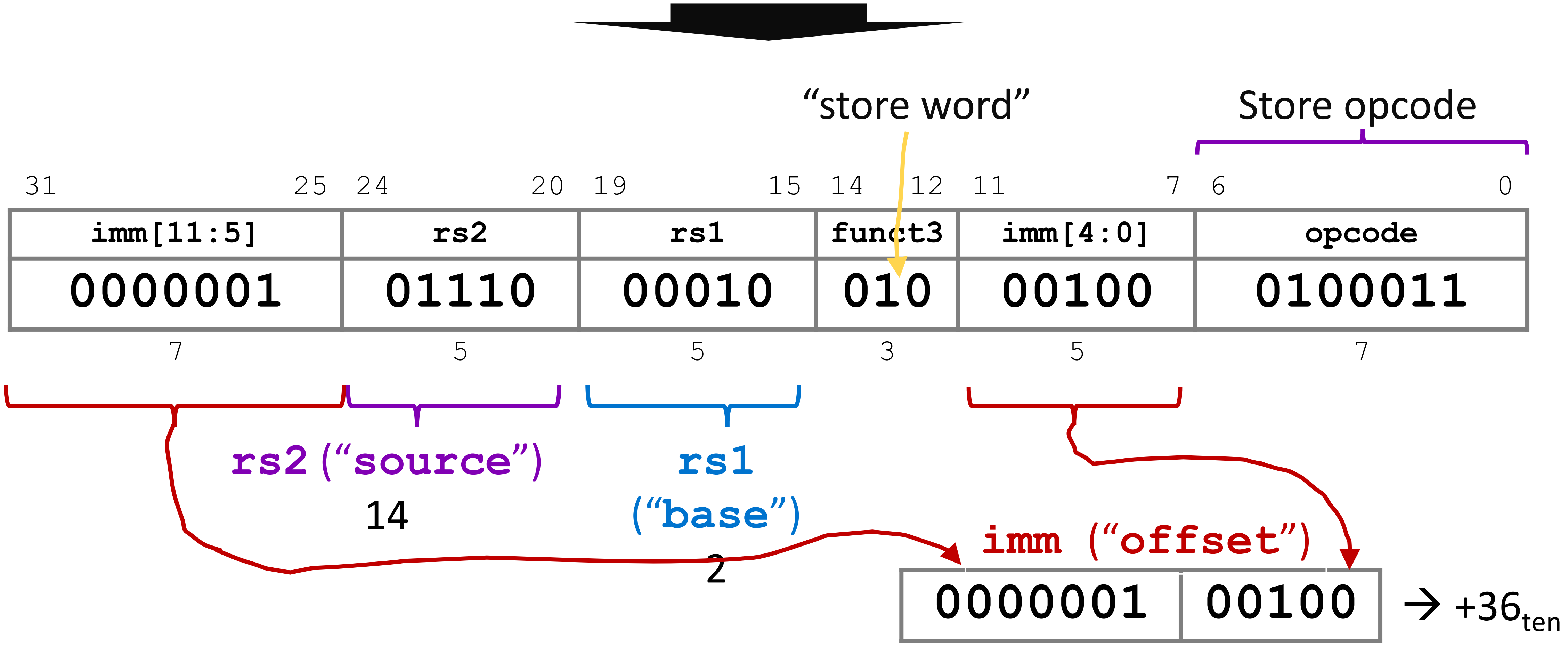
- Store needs immediate, two read registers, no destination register!
- Move lower 5 bits of immediate to **rd** field location in other formats:

|  | 31 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 |
|---|---|---|---|---|---|---|
| I-type | imm[11:5] | imm[4:0] | rs1 | funct3 | ~~rd~~ | opcode |
|  | 12 |  | 5 | 3 | 5 | 7 |
| R-type | ~~funct7~~ | rs2 | rs1 | funct3 | ~~rd~~ | opcode |
|  | 7 | 5 | 5 | 3 | 5 | 7 |

WATERLOO | ENGINEERING

# S-Format Example

**`sw x14, 36(x2)`**



→ +$36_{ten}$

# All three RV Store Instructions

Encodes data size of store operation

| | | | funct3 | | opcode | |
|---|---|---|---|---|---|---|
| imm[11:5] | rs2 | rs1 | 000 | imm[4:0] | 0100011 | sb |
| imm[11:5] | rs2 | rs1 | 001 | imm[4:0] | 0100011 | sh |
| imm[11:5] | rs2 | rs1 | 010 | imm[4:0] | 0100011 | sw |

No sign/zero extending for store!
We only write to memory
the data width specified.

Same data width fields as
load instructions.

## Summary

- In this lecture we talked about
  - 3 RISC-V instruction format
    - R-format, I-format and S-format

- Next up
  - The other 3 RISC-V instruction format

## Acknowledgement

- This slide builds on the hard work of the following amazing instructors:
  - Alvin Lebeck, Daniel Sorin, Andrew Hilton (Duke)
  - Dan Garcia, Bora Nikolic (Berkeley)