

Lecture 2a:

Instructions: Language of the Computer (1/3)

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

John Owens

Introduction to Computer Architecture

UC Davis EEC 170, Winter 2021

Big picture for today

- ***Stored program* computer: both programs and information are treated as data**
 - **“Information”: text, pictures, videos, simulation data, etc.**
- **All data is stored in the “memory” of the machine**
- **We wish to manipulate this data:**
 - **Some data is instructions; we will treat that data as instructions and execute/evaluate them (but also treat them as data! e.g., linking against other code)**
 - **Some data is information; our instructions will operate on this information**
- **Today’s topic: *What are these instructions?***

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

**The *instruction set* is the most
fundamental abstraction in this course.**

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

**It is the boundary between software and
hardware.**

**What factors do we want to consider
when we are designing our instructions?**

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Instruction Set

- The repertoire of instructions of a computer
- Different computers have different instruction sets
 - But with many aspects in common
- Early computers had very simple instruction sets
 - Simplified implementation
- Many modern computers also have simple instruction sets
- Since the 1980s, the dominant philosophy has been toward simpler (“reduced”, “RISC”) instruction sets as opposed to complex (“CISC”)
 - x86 is an exception, but x86 CPUs are RISC underneath

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

The RISC-V Instruction Set

- Used as the example throughout the book
- Developed at UC Berkeley as open ISA
- Now managed by the RISC-V Foundation (riscv.org)
- Typical of many modern ISAs
 - See RISC-V Reference Data tear-out card
- Increasing market!
 - It's a good ISA
 - It's also ~free
- Similar ISAs have a large share of embedded core market
 - Applications in consumer electronics, network/storage equipment, cameras, printers, ...

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Throughout this discussion:

- *What decisions* did the architects make about the RISC-V instruction set?
- *Why* did they make these decisions?

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Arithmetic Operations

- Add and subtract, three operands
 - Two sources and one destination
- `add a, b, c // a gets b + c`
 - What are a, b, and c?
 - add only operates on integer data (what's an integer?)
 - Note: RISC V instructions put the destination first
- All arithmetic operations have this form
- *Design Principle 1: Simplicity favors regularity*
 - Regularity makes implementation simpler
 - Simplicity enables higher performance at lower cost

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Arithmetic Example

- C code:

`f = (g + h) - (i + j);`

- Compiled RISC-V code:

`add t0, g, h // temp t0 = g + h`

`add t1, i, j // temp t1 = i + j`

`sub f, t0, t1 // f = t0 - t1`

- We're not done yet though—where are t0, g, etc.?

Assignment Project Exam Help

<https://tutores.com>

WeChat: cstutores

RISC-V Reference Card

- Google it to find it

Free & Open  **RISC-V** Reference Card

- It'll be attached to your exams

Arithmetic	ADD	R	ADD	rd,rs1,rs2
	ADD Immediate	I	ADDI	rd,rs1,imm
	SUBtract	R	SUB	rd,rs1,rs2
	Load Upper Imm	U	LUI	rd,imm
	Add Upper Imm to PC	U	AUIPC	rd,imm

Assignment Project Exam Help

<https://tutorcs.com>

- What this says:

WeChat: cstutorcs

- ADD is an “R” type instruction (you don’t know this yet)
- ADD’s operands are rd (destination), rs1 (source 1), rs2 (source 2). The “r” means “register”.
- SUB is ~the same as ADD

Register Operands

- Where is the data stored?
- Arithmetic instructions use *register* operands
 - Important! In RISC-V, all arithmetic instructions **ONLY** use register operands
- RISC-V has a **32 × 64-bit register file**
 - Use for frequently accessed data
 - 64-bit data (in RISC-V) is called a “doubleword”
 - 32 × 64-bit general purpose registers x0 to x31
 - x0 is hardwired to 0
 - 32-bit data is called a “word”
- *Design Principle 2: Smaller is faster*
 - c.f. main memory: millions of locations

RISC-V Registers

- **x0: the constant value 0**
- **x1: return address**
- **x2: stack pointer**
- **x3: global pointer**
- **x4: thread pointer**
- **x5 – x7, x28 – x31: temporaries**
- **x8: frame pointer**
- **x9, x18 – x27: saved registers**
- **x10 – x11: function arguments/results**
- **x12 – x17: function arguments**

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: estutorcs

Register Operand Example

■ C code:

```
f = (g + h) - (i + j);  
- f, ..., j in x19, x20, ..., x23
```

■ Compiled RISC-V code:

```
add x5, x20, x21  
add x6, x22, x23  
sub x19, x5, x6
```

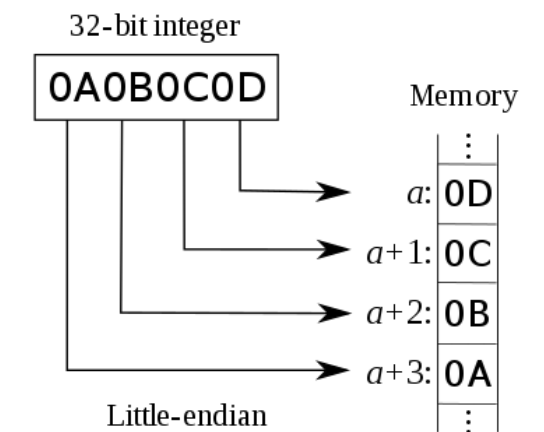
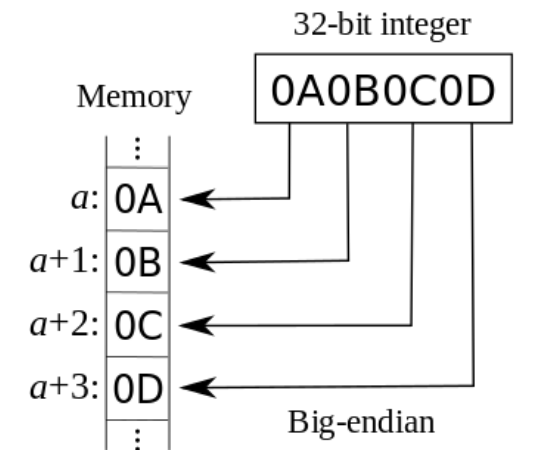
Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

Memory Operands (architectural decisions)

- Main memory used for composite data
 - Arrays, structures, dynamic data
- To apply arithmetic operations:
 - Load values from memory into registers
 - Store result from register to memory
- Memory is byte addressed
 - Each address identifies an 8-bit byte
 - Support for byte [8b], halfword [16b], word [32b], doubleword [64b]
- RISC-V is Little Endian
 - Least-significant byte at least address of a word [processors]
 - c.f. Big Endian: most-significant byte at least address [network]
- RISC-V does not require words to be aligned in memory
 - Unlike some other ISAs



<https://en.wikipedia.org/wiki/Endianness>

Memory Operand Example

■ C code:

```
A[12] = h + A[8];
```

- h in x21, base address of A in x22

■ Compiled RISC-V code:

- Index 8 requires offset of 64
- 8 bytes per doubleword

■ ld dest, imm(src):

- dest: destination register
- imm: immediate (integer)
- src: base memory address, in register

Loads	Load Byte	I	LB	rd,rs1,imm		
	Load Halfword	I	LH	rd,rs1,imm		
	Load Word	I	LW	rd,rs1,imm	L{D Q}	rd,rs1,imm
	Load Byte Unsigned	I	LBU	rd,rs1,imm		
	Load Half Unsigned	I	LHU	rd,rs1,imm	L{W D}U	rd,rs1,imm
Stores	Store Byte	S	SB	rs1,rs2,imm		
	Store Halfword	S	SH	rs1,rs2,imm		
	Store Word	S	SW	rs1,rs2,imm	S{D Q}	rs1,rs2,imm

- ```
ld x9, 64(x22) // x9 <- Mem[x22 + 64]
```
- ```
add   x9, x21, x9
```
- ```
sd x9, 96(x22) // Mem[x22 + 96] <- x9
```

# Registers vs. Memory

- Registers are (much) faster to access than memory
- Operating on memory data requires loads and stores
  - More instructions to be executed
- Compiler must use registers for variables as much as possible
  - Only spill to memory for less frequently used variables
  - Register optimization is important!

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs



# Immediate Operands

- Constant data specified in an instruction

```
addi x22, x22, 4 // add the number 4 to
 // x22, store back in x22
```

Assignment Project Exam Help

- Make the common case fast <https://tutorcs.com>

- Small constants are common [WeChat: cstutorcs](#)
- Immediate operand avoids a load instruction

|                   |                     |   |       |            |
|-------------------|---------------------|---|-------|------------|
| <b>Arithmetic</b> | ADD                 | R | ADD   | rd,rs1,rs2 |
|                   | ADD Immediate       | I | ADDI  | rd,rs1,imm |
|                   | SUBtract            | R | SUB   | rd,rs1,rs2 |
|                   | Load Upper Imm      | U | LUI   | rd,imm     |
|                   | Add Upper Imm to PC | U | AUIPC | rd,imm     |

# Unsigned Binary Integers

- Given an n-bit number

$$X = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \dots + x_12^1 + x_02^0$$

**Range: 0 to  $+2^n - 1$**

Assignment Project Exam Help

- Example

<https://tutorcs.com>

WeChat: cstutorcs

$$\begin{aligned} & - 0000\ 0000 \dots 0000\ 1011_2 \\ & = 0 \times 2^{31} + \dots + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ & = 0 + \dots + 8 + 0 + 2 + 1 = 11_{10} \end{aligned}$$

- Using 64 bits: 0 to +18,446,774,073,709,551,615

# Numbers

- Bits are just bits (no inherent meaning)
  - conventions define relationship between bits and numbers
- Binary numbers (base 2)
  - 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001...
  - decimal:  $0 \dots 2^n - 1$
- Of course it gets more complicated:
  - numbers are finite (overflow)
  - fractions and real numbers
  - negative numbers
  - RISC-V restrictions (e.g., no RISC-V subi instruction; addi can add a negative number)
- How do we represent negative numbers?
  - i.e., which bit patterns will represent which numbers?

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

# Possible Representations

| ■ Sign Magnitude | One's Complement | Two's Complement |
|------------------|------------------|------------------|
| 000 = +0         | 000 = +0         | 000 = +0         |
| 001 = +1         | 001 = +1         | 001 = +1         |
| 010 = +2         | 010 = +2         | 010 = +2         |
| 011 = +3         | 011 = +3         | 011 = +3         |
| 100 = -0         | 100 = -3         | 100 = -4         |
| 101 = -1         | 101 = -2         | 101 = -3         |
| 110 = -2         | 110 = -1         | 110 = -2         |
| 111 = -3         | 111 = -0         | 111 = -1         |

Assignment Project Exam Help  
<https://tutorcs.com>  
WeChat: cstutorcs

- In all these, the most significant bit is the “sign bit”
- Issues: balance, number of zeros, ease of operations

# 2s-Complement Signed Integers

- Given an n-bit number

$$X = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \dots + x_12^1 + x_02^0$$

Assignment Project Exam Help

Range:  $-2^{n-1}$  to  $+2^{n-1} - 1$   
<https://tutorcs.com>

- Example

WeChat: cstutorcs

$$\begin{aligned} & - 1111\ 1111 \dots 1111\ 1100_2 \\ & = -1 \times 2^{31} + 1 \times 2^{30} + \dots + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 \\ & = -2,147,483,648 + 2,147,483,644 = -4_{10} \end{aligned}$$

- Using 64 bits:  $-9,223,372,036,854,775,808$   
to  $9,223,372,036,854,775,807$

# 2s-Complement Signed Integers

- Bit 63 (the *most significant* bit, MSB) is sign bit
  - 1 for negative numbers
  - 0 for non-negative numbers
- $-(-2^n - 1)$  can't be represented
- Non-negative numbers have the same unsigned and 2s-complement representation
- Some specific numbers
  - 0: 0000 0000 ... 0000
  - -1: 1111 1111 ... 1111
  - Most-negative: 1000 0000 ... 0000
  - Most-positive: 0111 1111 ... 1111

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

# Signed Negation

## ■ Complement and add 1

- Complement means  $1 \rightarrow 0, 0 \rightarrow 1$
- Complement means “flip all the bits”

$$X + \bar{X} = 1111 \dots 1111_2 = -1$$

- $\bar{X} + 1 = -X$

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

## ■ Example: negate +2

- $+2 = 0000\ 0000 \dots 0010_{\text{two}}$
- $-2 = 1111\ 1111 \dots 1101_{\text{two}} + 1$   
 $= 1111\ 1111 \dots 1110_{\text{two}}$

# Sign Extension

- Representing a number using more bits

- Preserve the numeric value

- Replicate the sign bit to the left

- c.f. unsigned values: extend with 0s

- Examples: 8-bit to 16-bit

- +2: 0000 0010 => 0000 0000 0000 0010
- -2: 1111 1110 => 1111 1111 1111 1110

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

- In RISC-V instruction set

- lb: sign-extend loaded byte
- lbu: zero-extend loaded byte

| Loads |                    |   |     |            |                    |
|-------|--------------------|---|-----|------------|--------------------|
|       | Load Byte          | I | LB  | rd,rs1,imm |                    |
|       | Load Halfword      | I | LH  | rd,rs1,imm |                    |
|       | Load Word          | I | LW  | rd,rs1,imm | L{D Q} rd,rs1,imm  |
|       | Load Byte Unsigned | I | LBU | rd,rs1,imm |                    |
|       | Load Half Unsigned | I | LHU | rd,rs1,imm | L{W D}U rd,rs1,imm |



# Break

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

# Administrative stuff

- I'm going to aim for a W 8:30–9:30 office hour (Coffee House)
- HW1 is out. Questions? Issues? Post on Slack #hw1
- Looking at the RARS MIPS-V simulator.
  - Do you want to run on personal computers or in lab?
  - Can you run Java?
- Slides ran a bit slow this morning!
- Word vs. Doubleword (sorry)

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

# Representing Instructions

- **Instructions are encoded in binary**
  - Called “machine code”
  - How do we get from `add x5, x20, x21` to binary?
- **RISC-V instructions**
  - Encoded as 32-bit instruction words
  - Big picture: We divide the 32-bit instruction word into “fields”, each of a few bits, and encode different pieces of information from the instruction into each field
  - Small number of formats encoding operation code (opcode), register numbers, ...
  - Regularity!

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: estutorcs

# Hexadecimal

## ■ Base 16

- Compact representation of bit strings
- 4 bits (“nibble”) per hex digit
- 0x means “I’m hexadecimal”

Assignment Project Exam Help

|   |      |   |      |   |      |   |      |
|---|------|---|------|---|------|---|------|
| 0 | 0000 | 4 | 0100 | 8 | 1000 | c | 1100 |
| 1 | 0001 | 5 | 0101 | 9 | 1001 | d | 1101 |
| 2 | 0010 | 6 | 0110 | a | 1010 | e | 1110 |
| 3 | 0011 | 7 | 0111 | b | 1011 | f | 1111 |

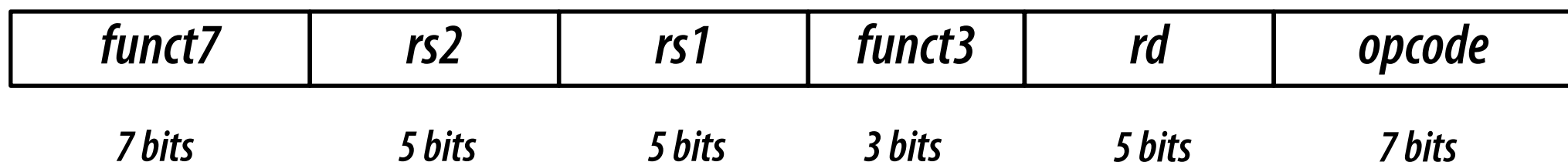
## ■ Example: 0x eca8 6420

- 1110 1100 1010 1000 0110 0100 0010 0000

# RISC-V R-format Instructions

## ■ Instruction fields

- ***opcode***: operation code
- ***rd***: destination register number
- ***funct3***: 3-bit function code (additional opcode)
- ***rs1***: the first source register number
- ***rs2***: the second source register number
- ***funct7***: 7-bit function code (additional opcode)



# R-format Example

```
add x9, x20, x21
```

|               |               |               |               |               |               |
|---------------|---------------|---------------|---------------|---------------|---------------|
| <i>funct7</i> | <i>rs2</i>    | <i>rs1</i>    | <i>funct3</i> | <i>rd</i>     | <i>opcode</i> |
| <i>7 bits</i> | <i>5 bits</i> | <i>5 bits</i> | <i>3 bits</i> | <i>5 bits</i> | <i>7 bits</i> |

Assignment Project Exam Help

<https://tutorcs.com>

|                |              |              |            |              |                |
|----------------|--------------|--------------|------------|--------------|----------------|
| <i>0</i>       | <i>21</i>    | <i>20</i>    | <i>0</i>   | <i>9</i>     | <i>51</i>      |
| <i>0000000</i> | <i>10101</i> | <i>10100</i> | <i>000</i> | <i>01001</i> | <i>0110011</i> |

$$0000\ 0001\ 0101\ 1010\ 0000\ 0100\ 1011\ 0011_{two} = 015A04B3_{16}$$

# Opcode Map

RV32I Base Instruction Set

|                     |       |     |     |             |         |       |
|---------------------|-------|-----|-----|-------------|---------|-------|
| imm[31:12]          |       |     |     | rd          | 0110111 | LUI   |
| imm[31:12]          |       |     |     | rd          | 0010111 | AUIPC |
| imm[20:10:11:19:12] |       |     |     | rd          | 1101111 | JAL   |
| imm[11:0]           |       | rs1 | 000 | rd          | 1100111 | JALR  |
| imm[12:10:5]        | rs2   | rs1 | 000 | imm[4:1:11] | 1100011 | BEQ   |
| imm[12:10:5]        | rs2   | rs1 | 001 | imm[4:1:11] | 1100011 | BNE   |
| imm[12:10:5]        | rs2   | rs1 | 100 | imm[4:1:11] | 1100011 | BLT   |
| imm[12:10:5]        | rs2   | rs1 | 101 | imm[4:1:11] | 1100011 | BGE   |
| imm[12:10:5]        | rs2   | rs1 | 110 | imm[4:1:11] | 1100011 | BLTU  |
| imm[12:10:5]        | rs2   | rs1 | 111 | imm[4:1:11] | 1100011 | BGEU  |
| 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   |
| 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    |
| 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  |
| 0000000             | rs2   | rs1 | 000 | rd          | 0110011 | ADD   |
| 0100000             | rs2   | rs1 | 000 | rd          | 0110011 | SUB   |
| 0000000             | rs2   | rs1 | 001 | rd          | 0110011 | ST R  |

# RISC-V I-format Instructions

## ■ Immediate arithmetic and load instructions

- **rs1: source or base address register number**
- **immediate: constant operand, or offset added to base address**

Assignment Project Exam Help

- **2s-complement, sign extended**

<https://tutorcs.com>

- **How big can this immediate be?**

WeChat: cstutorcs

- **Why did they pick this size?**

- **Advantages/disadvantages of making it bigger/smaller?**



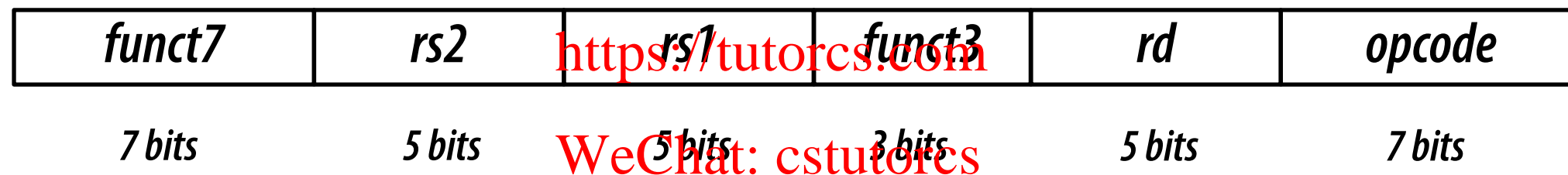


# RISC-V I-format vs. R-format

## ■ I-format:



## ■ R-format:



Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

- ***Design Principle 3: Good design demands good compromises***
  - Different formats complicate decoding, but allow 32-bit instructions uniformly
  - Keep formats as similar as possible

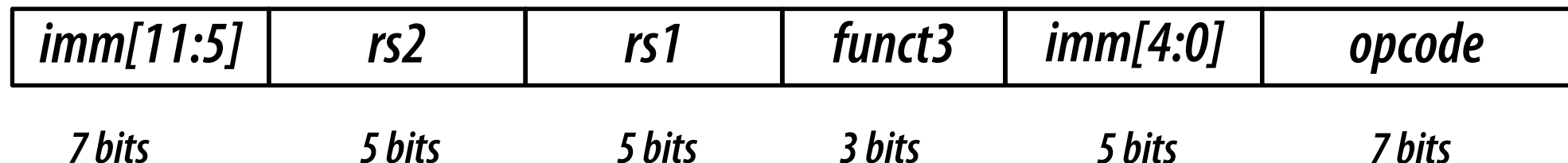
# RISC-V S-format Instructions

- Different immediate format for store instructions
  - **rs1: base address register number**
  - **rs2: source operand register number**
  - **immediate: offset added to base address**
    - **Split so that rs1 and rs2 fields always in the same place**

Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs

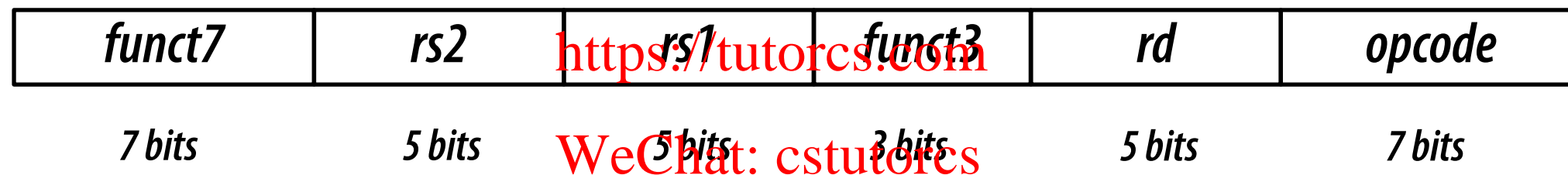


# RISC-V I-format vs. R-format vs. S-format

## ■ I-format:



## ■ R-format:



## ■ S-format:



Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs