RISC-V 101

KAIST, School of EE

* This slide is from Yujeong Choi's slides.

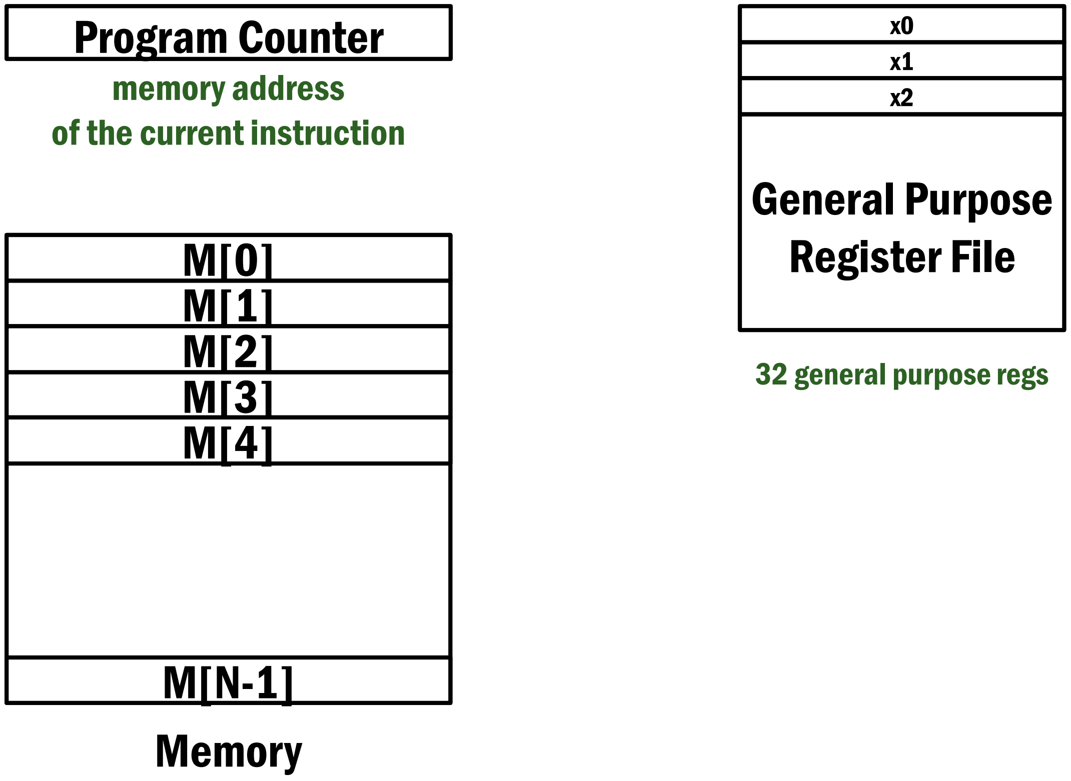
RISC–V

- Popular open-source ISA (Instruction Set Architecture) for research, education, and industrial usage

- RISC–V vs MIPS
  - Similarities
    - Based on the RISC (reduced instruction set computer) ISA philosophy
  - Differences
    - Free (RISC–V) vs commercial (MIPS)
    - Variable-length instructions (RISC–V) vs fixed-length instructions (MIPS)

      // RISC-V has a fixed-length base instruction set and all our lab assignments will assume this fixed-length version (i.e., RV32I)
    - RISC–V is more expandable

RISC-V Programmer Visible State



Base Integer ISA & Extension

- Base integer ISA
  - Integer data type
  - Width of integer registers & user address space specifies instruction set
  - 32-bit (RV32I), 64-bit (RV64I), and 128-bit (RV128I) integer data format
    // A customized RV32I is used for all lab assignments
  - // All data for the lab are in 32-bit integer format
  - Little-endian
- Extension
  - Multiply/divide (M), atomic operations (A), floating-point arithmetic (F), double-precision floating-point arithmetic (D) support

    ex) RV32I (base), RV32IF (base + floating point extension),
    RV32IMD (base + multiplication/division & double precision extension)
  - G for all extensions
    - RV32G (= RV32IMAFD)

Instruction Formats

- Fixed 32-bit instructions for base integer instructions (RV32I, RV64I)
- Variable-length instructions for ISA extensions
  - Multiples of 16-bit length

// We assume the fixed-length (32-bit) ISA in our labs (RV32I)

Instruction Formats – RV32I

- 4 simple formats
  - R-type, 3 register operands

funct7	rs2	rs1	funct3	rd	opcode	R-type
7-bit	5-bit	5-bit	3-bit	5-bit	7-bit	

  - I-type, 2 register operands and 12-bit immediate operand

imm[11:0]	rs1	funct3	rd	opcode	I-type
12-bit	5-bit	3-bit	5-bit	7-bit	

  - S-type, 2 register operands and 12-bit immediate operand

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
7-bit	5-bit	5-bit	3-bit	5-bit	7-bit	

  - U-type, 1 register operand and 20-bit immediate operand

imm[31:12]	rd	opcode	U-type
20-bit	5-bit	7-bit	

Instruction Formats – RV32I

- **4 simple formats**
  - **R-type, 3 register operands**

funct7	rs2	rs1	funct3	rd	opcode	R-type
7-bit	5-bit	5-bit	3-bit	5-bit	7-bit	

  - **I-type, 2 register operands and 12-bit immediate operand**

imm[11:0]	rs1	funct3	rd	opcode	I-type
12-bit	5-bit	3-bit	5-bit	7-bit	

  - **S-type, 2 register operands and 12-bit immediate operand**

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
7-bit	5-bit	5-bit	3-bit	5-bit	7-bit	

  - **U-type, 1 register operand and 20-bit immediate operand**

imm[31:12]	rd	opcode	U-type
20-bit	5-bit	7-bit	

All opcodes are aligned to the right

Instruction Formats – RV32I

- 4 simple formats
  - R-type, 3 register operands

funct7	rs2	rs1	funct3	rd	opcode	R-type
7-bit	5-bit	5-bit	3-bit	5-bit	7-bit	

  - I-type, 2 register operands and 12-bit immediate operand

imm[11:0]	rs1	funct3	rd	opcode	I-type
12-bit	5-bit	3-bit	5-bit	7-bit	

  - S-type, 2 register operands and 12-bit immediate operand

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
7-bit	5-bit	5-bit	3-bit	5-bit	7-bit	

  - U-type, 1 register operand and 20-bit immediate operand

imm[31:12]	rd	opcode	U-type
20-bit	5-bit	7-bit	

All registers (rs1, rs2, rd) are located in the same position

Instruction Formats – RV32I

- 4 simple formats

Each immediate subfield is labeled with position (imm[x]) in the immediate value being "produced", rather than the bit position within the instruction's immediate field as is usually done (which was the case in our MIPS examples)

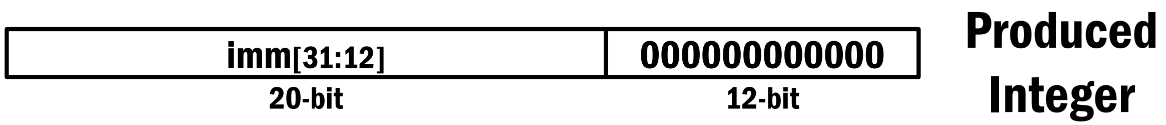
- U-type, 1 register oper nd and 20-bit immediate operand

imm[31:12]	rd	opcode	U-type
20-bit	5-bit	7-bit	

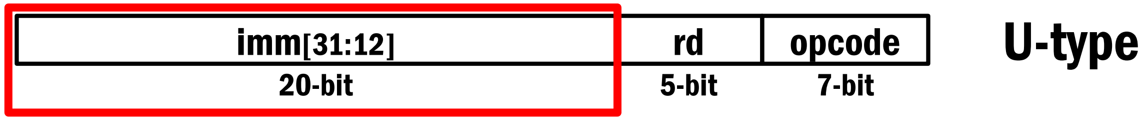
Instruction Formats – RV32I

- 4 simple formats

Each immediate subfield is labeled with position (imm[x]) in the immediate value being "produced", rather than the bit position within the instruction's immediate field as is usually done (which was the case in our MIPS examples)



- U-type, 1 register operand and 20-bit immediate operand



Instruction Formats – RV32I

- **2 variants**
  - **B-type, 2 register operands and 12-bit immediate operand**

imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
1-bit	6-bit	5-bit	5-bit	3-bit	4-bit	1-bit	7-bit

B-type

  - **J-type, 1 register operand and 20-bit immediate operand**

imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode
1-bit	10-bit	1-bit	8-bit	5-bit	7-bit

J-type

Instruction Formats – RV32I

- 2 variants
  - B-type, 2 register operands and 12-bit immediate operand

imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	B-type
1-bit	6-bit	5-bit	5-bit	3-bit	4-bit	1-bit	7-bit	

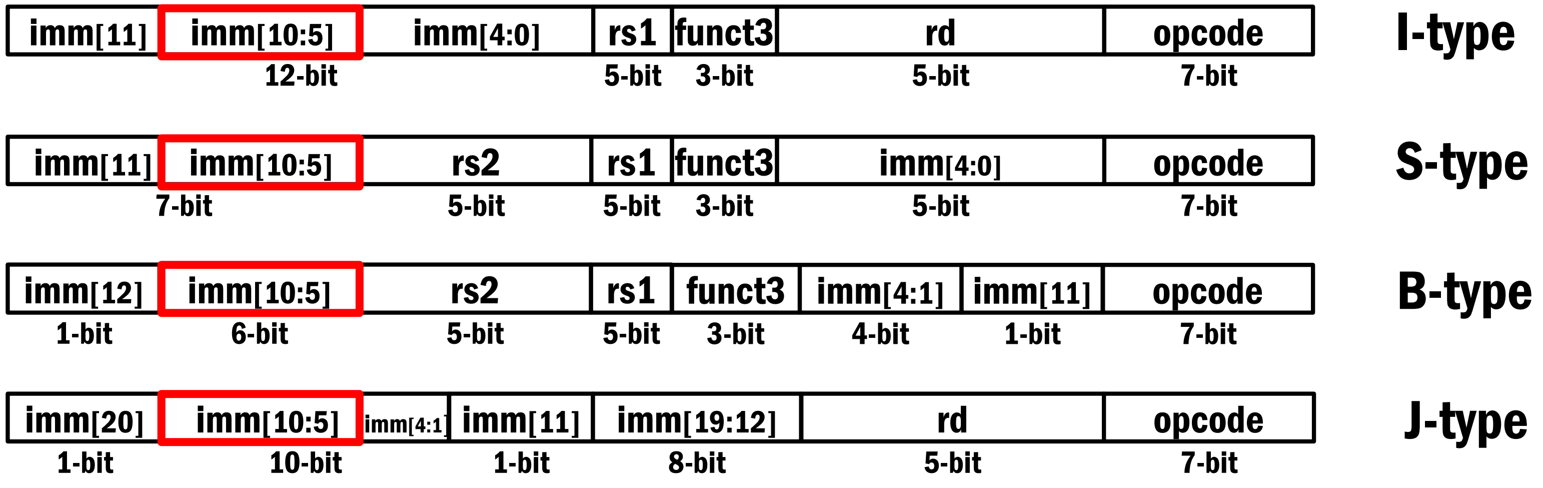
  - J-type, 1 register operand and 20-bit immediate operand

imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode	J-type
1-bit	10-bit	1-bit	8-bit	5-bit	7-bit	

Each immediate subfield is labeled with position (imm[x]) in the immediate value being "produced", rather than the bit position within the instruction's immediate field as is usually done (which was the case in our MIPS examples)

Instruction Formats – RV32I

- Reason for sliced & spread immediate position
  - MSB (Most Significant Bit) can be easily extended for sign extension
  - Each immediate's bit position overlaps across the types
    - → reducing hardware complexity
    - ex) imm[10:5] of I, S, B, and J are in the same position


imm[11] imm[10:5] imm[4:0] rs1 funct3 rd opcode
12-bit 5-bit 3-bit 5-bit 7-bit
I-type
imm[11] imm[10:5] rs2 rs1 funct3 imm[4:0] opcode
7-bit 5-bit 5-bit 3-bit 5-bit 7-bit
S-type
imm[12] imm[10:5] rs2 rs1 funct3 imm[4:1] imm[11] opcode
1-bit 6-bit 5-bit 5-bit 3-bit 4-bit 1-bit 7-bit
B-type
imm[20] imm[10:5] imm[4:1] imm[11] imm[19:12] rd opcode
1-bit 10-bit 1-bit 8-bit 5-bit 7-bit
J-type

Integer Computational Instructions

- Assembly (e.g., register-immediate signed addition)

  ADDI rd_{reg} , $rs1_{reg}$, immediate

- Machine encoding

imm[11:0]	rs1	funct3	rd	opcode	I-type
12-bit	5-bit	3-bit	5-bit	7-bit	
		ADDI		0010011	

- Semantics
  - GPR[rd] $\leftarrow$ GPR[rs1] OP sign-extend (immediate$_{12}$)
  - PC $\leftarrow$ PC + 4
- Overflow is ignored (no exceptions raised)
- Variations
  - Arithmetic:{ADDI}
  - Logical: {ANDI, ORI, XORI}
  - Conditional: {SLTI(U)}

Integer Computational Instructions

- Assembly (e.g., register-register signed addition)

  ADD rd_{reg} , $rs1_{reg}$, $rs2_{reg}$

- Machine encoding

funct7	rs2	rs1	funct3	rd	opcode
7-bit	5-bit	5-bit	3-bit	5-bit	7-bit
0000000			ADD		0110011
(0100000 for SUB/SRA)					

R-type

- Semantics
  - GPR[rd] $\leftarrow$ GPR[rs1] OP GPR[rs2]
  - PC $\leftarrow$ PC + 4
- Overflow is ignored (no exceptions raised)
- Variations
  - Arithmetic: {signed, unsigned} x {ADD, SUB}
  - Logical: {AND, OR, XOR}
  - Shift: {SLL,SRL, SRA}
  - Conditional: {SLT[U]}

Control Transfer Inst. (Unconditional Jump)

- **Assembly**

  JAL x1, immediate (JAL (Jump And Link), x1 : return address)

- **Machine encoding**

imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode	J-type
1-bit	10-bit	1-bit	8-bit	5-bit	7-bit	
					1101111	

sign-bit	imm[20]	imm[19:12]	imm[11]	imm[10:1]	0	Produced Immediate
11-bit	1-bit	8-bit	1-bit	10-bit	1-bit	

21-bit

- **Semantics** // PC-relative jump
  - target $\leftarrow$ PC + sign-extend(immediate$_{21}$)
  - GPR [x1] $\leftarrow$ PC + 4
  - PC $\leftarrow$ target
- How far can you jump (in both direction)? $\pm$1 MiB

Control Transfer Inst. (Unconditional Jump)

- **Assembly**

  JALR x1, rs1, immediate (x1 : return address)

- **Machine encoding**

imm[11:0]	rs1	funct3	rd	opcode	I-type
12-bit	5-bit	3-bit	5-bit	7-bit	
		0		1100111	

- **Semantics**
  - target $\leftarrow$ (GPR[rs1] + sign-extend(immediate$_{12}$)) & 0xfffffffe
  - GPR [x1] $\leftarrow$ PC + 4
  - PC $\leftarrow$ target

Control Transfer Inst. (Conditional Jump)

- Assembly (e.g., branch if equal)

  BEQ rs1_{reg} , rs2_{reg} , immediate

- Machine encoding

imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
1-bit	6-bit	5-bit	5-bit	3-bit	4-bit	1-bit	7-bit
				BEQ			1100011

B-type

- Semantics // PC-relative jump
  - target $\leftarrow$ PC + sign-extend(immediate_{13})
  - if GPR[rs1]==GPR[rs2] then PC $\leftarrow$ target

    else PC $\leftarrow$ PC + 4
- How far can you jump? $\pm$ 4 KiB
- Variations
  - BEQ, BNE, BLT, BLTU, BGT, BGTU
- No branch delay slots

  Direct comparison between two regs unlike MIPS

Load Instructions

- **Assembly (e.g., load 4-byte word)**

  LD rd_{reg} offset ($rs1_{reg}$)

- **Machine encoding**

imm[11:0]	rs1	funct3	rd	opcode	
12-bit	5-bit	3-bit width	5-bit	7-bit 0000011	I-type

- **Semantics**
  - effective_address $\leftarrow$ sign-extend($offset_{12}$) + GPR[rs1]
  - GPR[rd] $\leftarrow$ MEM[translate(effective_address)]
  - PC $\leftarrow$ PC + 4
- **Byte-addressed**
  - Misaligned accesses are allowed (but slow)

  // Only implement aligned accesses for the lab assignments

Store Instructions

- Assembly (e.g., store 4-byte word)

  SW rs2_{reg} offset (rs1_{reg})

- Machine encoding

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
7-bit	5-bit	5-bit	3-bit	5-bit	7-bit	
			width		0100011	

- Semantics
  - effective_address $\leftarrow$ sign-extend(offset_{12}) + GPR[rs1]
  - MEM[translate(effective_address)] $\leftarrow$ GPR[rs2]
  - PC $\leftarrow$ PC + 4
- Byte-addressed
  - Misaligned accesses are allowed (but slow)

  // Only implement aligned accesses for the lab assignments

RISC-V Register Usage Convention

Register	Description	ABI Name	Saver
x0	Always 0	$zero	-
x1	Return address	$ra	Caller
x2	Stack pointer	$sp	Callee
x3	Global pointer	$gp	-
x4	Thread pointer	$tp	-
x5	Temporary / Alternate link	$t0	Caller
x6~x7	Caller-saved temporaries	$t1~$t2	Caller
x8	Saved register / Frame pointer	$s0 / $fp	Callee
x9	Saved register	$s1	Callee
x10~x11	Function call arguments / Return values	$a0 ~ $a1	Caller
x12~x17	Function call arguments	$a2 ~ $a7	Caller
x18~x27	Saved registers	$s2 ~ $s11	Caller
x28~x31	Caller saved temporaries	$t3 ~ $t6	caller

RV32I Base Instruction Set - 1

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|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
31 25	**24 20**	**19 15**	**14 12**	**11 7**	**6 0**	

RV32I Base Instruction Set - 1

B Type

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	
imm[11:0]		rs1	000	rd	1100111	
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
31 25	**24 20**	**19 15**	**14 12**	**11 7**	**6 0**	

RV32I Base Instruction Set - 1

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|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	
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	
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
31 25	**24 20**	**19 15**	**14 12**	**11 7**	**6 0**	

I Type Load

RV32I Base Instruction Set - 2

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	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
31 25	**24 20**	**19 15**	**14 12**	**11 7**	**6 0**	

RV32I Base Instruction Set - 2

I Type

imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	
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	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
31 25	**24 20**	**19 15**	**14 12**	**11 7**	**6 0**	

RV32I Base Instruction Set - 3

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
~~0000 pred~~	~~succ~~	~~00000~~	~~000~~	~~00000~~	~~0001111~~	FENCE
~~0000 0000~~	~~0000~~	~~00000~~	~~001~~	~~00000~~	~~0001111~~	FENCE.I
~~000000000000~~		~~00000~~	~~000~~	~~00000~~	~~1110011~~	ECALL
~~000000000001~~		~~00000~~	~~000~~	~~00000~~	~~1110011~~	EBREAK
~~csr~~		[illegible]	[illegible]	[illegible]	[illegible]	CSRRW
~~csr~~		[illegible]	[illegible]	[illegible]	[illegible]	CSRRS
~~csr~~		~~rs1~~	~~011~~	~~rd~~	~~1110011~~	CSRRC
~~csr~~		~~zimm~~	~~101~~	~~rd~~	~~1110011~~	CSRRWI
~~csr~~		~~zimm~~	~~110~~	~~rd~~	~~1110011~~	CSRRSI
~~csr~~		~~zimm~~	~~111~~	~~rd~~	~~1110011~~	CSRRCI
31 25	**24 20**	**19 15**	**14 12**	**11 7**	**6 0**	

We don’t cover these instructions

Customized RV32I Instruction Set

Three customized instructions

31 ... 25	24 ... 20	19 ... 15	14 ... 12	11 ... 7	6 ... 0	
0000000	rs2	rs1	111	rd	0001011	MULT
0000001	rs2	rs1	111	rd	0001011	MODULO
0000010	-	rs1	110	rd	0001011	IS_EVEN

Read the RISC-V Instruction Set Manual!!

- Section 1. Introduction
- Section 2. RC32I Base Integer Instruction Set
  - Section 2.1 to Section 2.6
    - No need to read Section 2.7 to 2.9
  - Page 9 – 19
- Chapter 19
  - Page 103 – 104
- Chapter 20
  - Page 109 – 110

Questions?