Tikrit University
Electrical Engineering Department

EE-317
Computer Engineering
2024-2025

Instructions: Format

Jalal Nazar Abdulbaqi, Ph.D.

jalal.abdulbaqi@tu.edu.iq

Outline

- Instructions Format (*transform Assembly to Machine code*)
  - Registers (R)
  - Immediate (I)
  - Store (S)
- Stored Program Computers
- Logical and Shifts Instructions
- Conditional Operations

Instructions Format

	31 27	26 25	24 20	19 15	14 12	11 7	6 0
R	funct7		rs2	rs1	funct3	rd	Opcode
I	imm[11:0]			rs1	funct3	rd	Opcode
S	imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode
SB	imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]					rd	opcode
UJ	imm[20|10:1|11|19:12]					rd	opcode

Instructions Format

RISC-V Instructions	Name	Format
Add	add	R
Subtract	sub	R
Add immediate	addi	I
Load word	lw	I
Load word, unsigned	lwu	I
Store word	sw	S
Load halfword	lh	I
Load halfword, unsigned	lhu	I
Store halfword	sh	S
Load byte	lb	I
Load byte, unsigned	lbu	I
Store byte	sb	S
Load reserved	lr.d	R
Store conditional	sc.d	R
Load upper immediate	lui	U
And	and	R

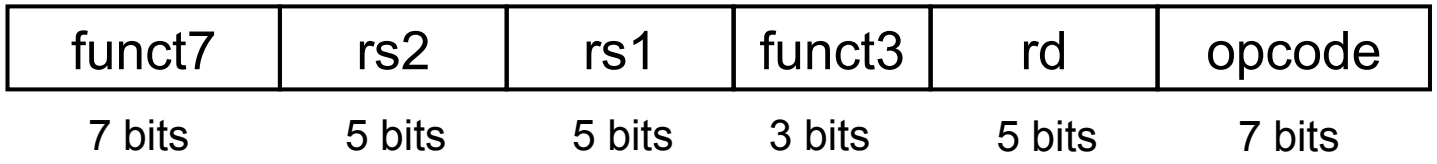
RISC-V Instructions	Name	Format
Inclusive or	or	R
Exclusive or	xor	R
And immediate	andi	I
Inclusive or immediate	ori	I
Exclusive or immediate	xori	I
Shift left logical	sll	R
Shift right logical	srl	R
Shift right arithmetic	sra	R
Shift left logical immediate	slli	I
Shift right logical immediate	srli	I
Shift right arithmetic immediate	srai	I
Branch if equal	beq	SB
Branch if not equal	bne	SB
Branch if less than	blt	SB
Branch if greater or equal	bge	SB
Branch if less, unsigned	bltu	SB
Branch if greatr/eq, unsigned	bgeu	SB
Jump and link	jal	UJ
Jump and link register	jalr	I

RISC-V R-format Instructions

- Instruction fields

```
add rd,rs1,rs2
```

  - **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)



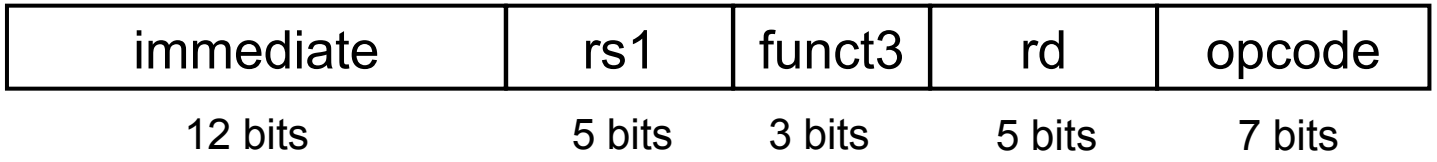
R-format Example

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

funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits
0	21	20	0	9	51
0000000	10101	10100	000	01001	0110011

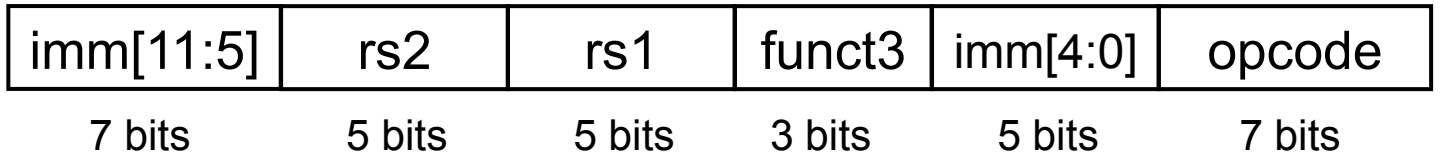
$0000\ 0001\ 0101\ 1010\ 0000\ 0100\ 1011\ 0011_{\mathbf{two}} = 015A04B3_{\mathbf{hex}}$

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



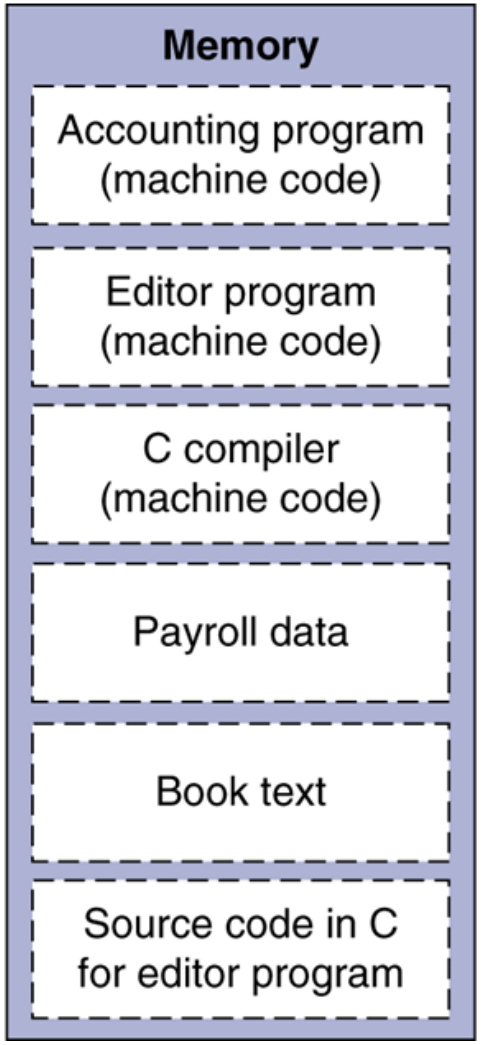
- Different immediate format for store instructions
  - **rs1**: base address register number
  - **rs2**: source operand register number
  - **immediate**: offset added to base address
    - Split so that **rs1** and **rs2** fields always in the same place

Stored Program Computers

1) **Instructions** represented in **binary**, just like **data**

2) **Instructions** and **data** stored in **memory**

- Programs can operate on programs
  - e.g., compilers, linkers, …
- Binary compatibility allows compiled programs to work on different computers
  - Standardized ISAs



Logical Operations

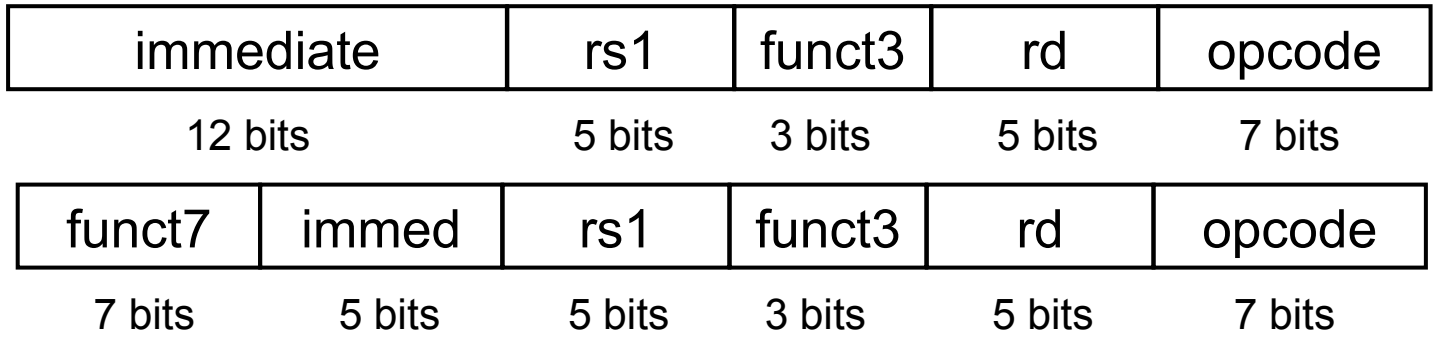
- Instructions for bitwise manipulation

Operation	C	Java	RISC-V
Shift left	<<	<<	`sll, slli`
Shift right	>>	>>>	`srl, srli`
Bit-by-bit AND	&	&	`and, andi`
Bit-by-bit OR	|	|	`or, ori`
Bit-by-bit XOR	^	^	`xor, xori`
Bit-by-bit NOT	~	~	

- Useful for extracting and inserting groups of bits in a word

Shift Operations

I-Format

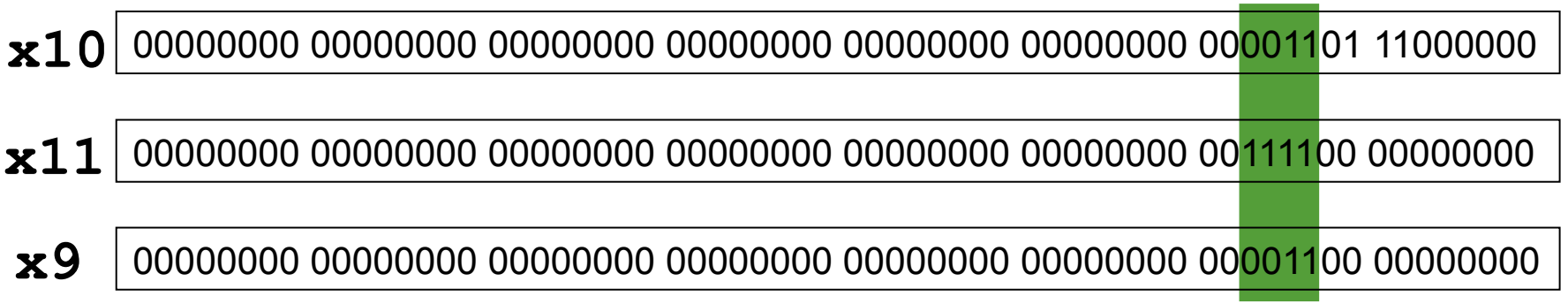


- **immed**: how many positions to shift?
- Shift left logical
  - Shift left and fill with **0** bits
  - **`slli`** by ***i*** bits multiplies by $\mathbf{2}^{i}$
- Shift right logical
  - Shift right and fill with **0** bits
  - **`srli`** by ***i*** bits divides by $\mathbf{2}^{i}$ (unsigned only)
- Shift right arithmetic
  - Shift right and fill with **Sign** bits
  - **`srai`** by ***i*** bits divides by $\mathbf{2}^{i}$ (signed only)

AND Operations

- Useful to mask bits in a word
  - Select some bits, clear others to **0**

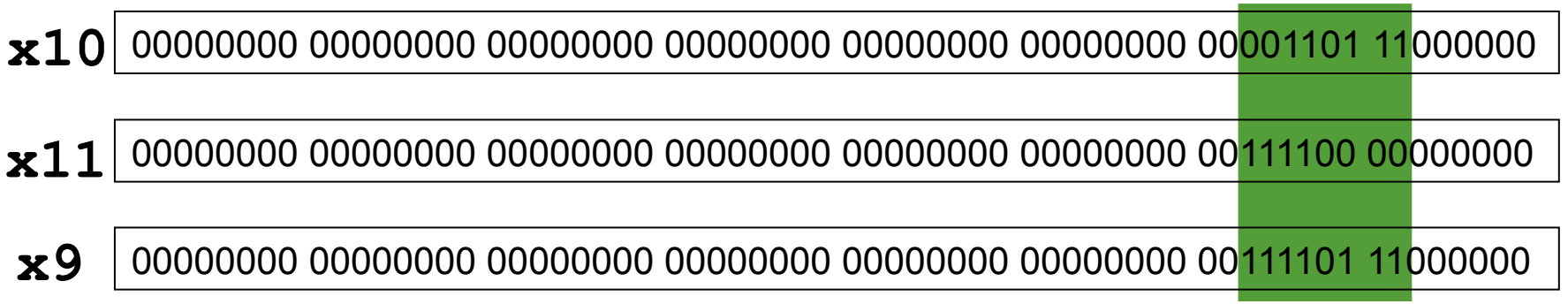
```
and x9,x10,x11
```



OR Operations

- Useful to include bits in a word
  - Set some bits to **1**, leave others unchanged

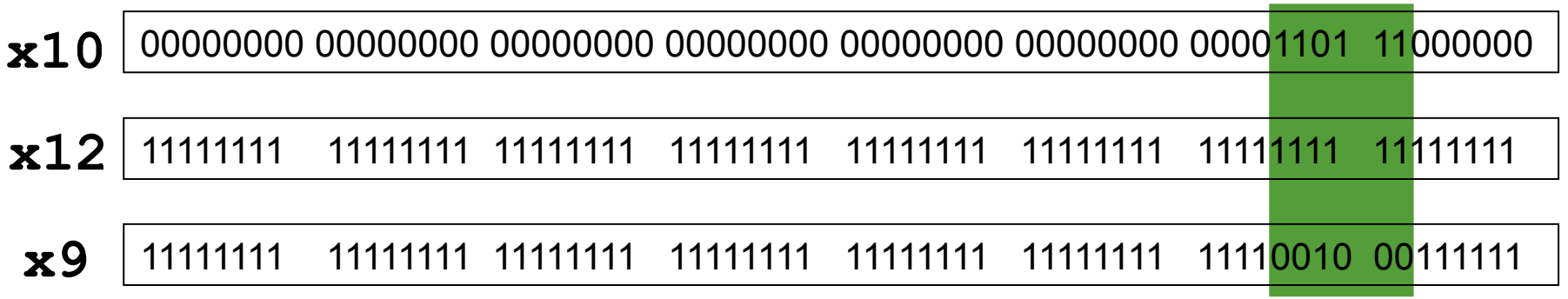
```
or x9,x10,x11
```



XOR Operations

- Differencing operation
- Set some bits to **1**, leave others unchanged

```
xor x9,x10,x12      # NOT operation
xori x9,x10,-1
```



Conditional Operations

- Branch to a labeled instruction if a condition is true
  - Otherwise, continue sequentially

- **`beq rs1, rs2, L1`**
  - if (rs1 == rs2) branch to instruction labeled **`L1`**

- **`bne rs1, rs2, L1`**
  - if (rs1 != rs2) branch to instruction labeled **`L1`**

Compiling If Statements

- C code:

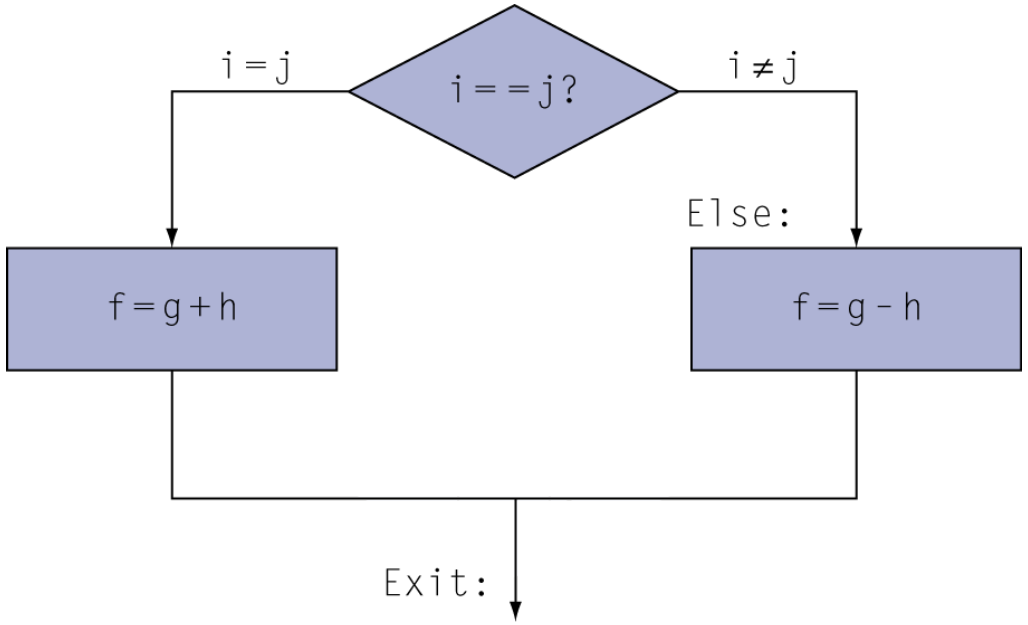
```
if (i==j) f = g+h;
else f = g-h;
```

  - f, g, … in x19, x20, …

- Compiled RISC-V code:

```
      bne x22, x23, Else
      add x19, x20, x21
      beq x0, x0, Exit      # unconditional
Else: sub x19, x20, x21
Exit: …
```

Assembler calculates addresses



Compiling Loop Statements

- C code:

```
while (save[i] == k) i+= 1;
```

  - **i** in **x22**, **k** in **x24**, address of save in **x25**

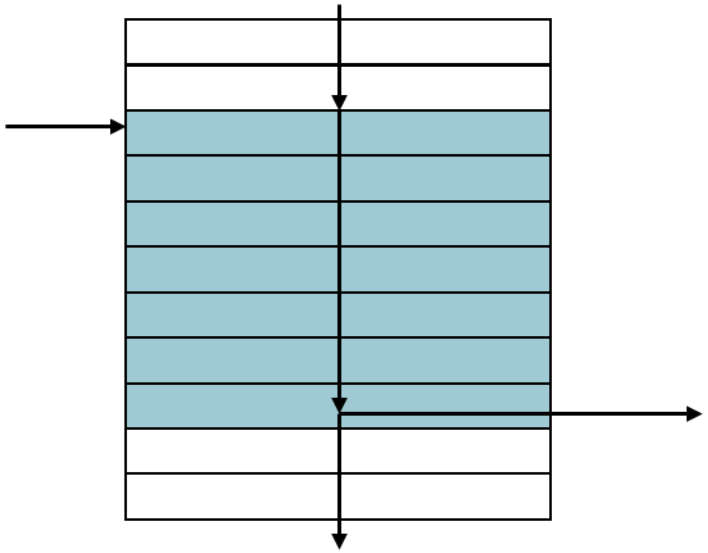
- Compiled RISC-V code:

```
Loop: slli x10, x22, 2
      add  x10, x10, x25
      lw   x9, 0(x10)
      bne  x9, x24, Exit
      addi x22, x22, 1
      beq  x0, x0, Loop
Exit: …
```

Basic Blocks

- A basic block is a sequence of instructions with
  - No embedded branches (except at end)
  - No branch targets (except at beginning)

- A compiler identifies basic blocks for optimization
- An advanced processor can accelerate execution of basic blocks



More Conditional Operations

- **`blt rs1, rs2, L1`**
  - if (rs1 < rs2) branch to instruction labeled **`L1`**
- **`bge rs1, rs2, L1`**
  - if (rs1 >= rs2) branch to instruction labeled **`L1`**
- Example

```
if (a < b) a += 1;
```

a in **x22**, **b** in **x23**

```
 bge  x23, x22, Exit        # branch if b >= a
addi x22, x22, 1
Exit:
```

Signed vs. Unsigned

- Signed comparison: **`blt, bge`**
- Unsigned comparison: **`bltu, bgeu`**
- Example
  - **`x22`** = 1111 1111 1111 1111 1111 1111 1111 1111
  - **`x23`** = 0000 0000 0000 0000 0000 0000 0000 0001
  - **`x22 < x23  //signed`**
    - –1 < +1
  - **`x22 > x23  //unsigned`**
    - +4,294,967,295 > +1

Instruction Summary

Instruction class	RISC-V examples	HLL correspondence	Frequency	
			Integer	Fl. Pt.
Arithmetic	`add, sub, addi`	Operations in assignment statements	16%	48%
Data transfer	`lw, sw, lh, sh, lb, sb, lui`	References to data structures in memory	35%	36%
Logical	`and, or, xor, sll, srl, sra`	Operations in assignment statements	12%	4%
Branch	`beq, bne, blt, bge, bltu, bgeu`	*If* statements; loops	34%	8%
Jump	`jal, jalr`	Procedure calls & returns; *switch* statements	2%	0%

Instruction Summary

Format	Instruction	Opcode	Funct3	Funct6/7
R-type	add	0110011	000	0000000
	sub	0110011	000	0100000
	sll	0110011	001	0000000
	xor	0110011	100	0000000
	srl	0110011	101	0000000
	sra	0110011	101	0000000
	or	0110011	110	0000000
	and	0110011	111	0000000
	lr.d	0110011	011	0001000
	sc.d	0110011	011	0001100
I-type	lb	0000011	000	n.a.
	lh	0000011	001	n.a.
	lw	0000011	010	n.a.
	lbu	0000011	100	n.a.
	lhu	0000011	101	n.a.
	addi	0010011	000	n.a.
	slli	0010011	001	000000
	xori	0010011	100	n.a.
	srli	0010011	101	000000
	srai	0010011	101	010000
	ori	0010011	110	n.a.
	andi	0010011	111	n.a.
	jalr	1100111	000	n.a.
S-type	sb	0100011	000	n.a.
	sh	0100011	001	n.a.
	sw	0100011	010	n.a.
SB-type	beq	1100111	000	n.a.
	bne	1100111	001	n.a.
	blt	1100111	100	n.a.
	bge	1100111	101	n.a.
	bltu	1100111	110	n.a.
	bgeu	1100111	111	n.a.
U-type	lui	0110111	n.a.	n.a.
UJ-type	jal	1101111	n.a.	n.a.

Summary

- In RISC-V, **all instructions** encoded in **32-bit** with **six** different format. In this lecture, we discussed **R, I, and S format**.
- Both **instructions** and **data** represented in **binary** and stored in **memory** (**Stored-Program Concept**)
- In this lecture, we discuss three basic **logic** instructions (**and**, **or**, & **xor**) and two **bit-wise** instructions(**sll** & **srl**).
- In **RV32I**, there are four base comparison instructions: **beq**, **bne**, **blt**, **bge**. (**Conditional Branch**)