RISC V Architecture

Mahesh Awati
Department of Electronics and Communication Engg.

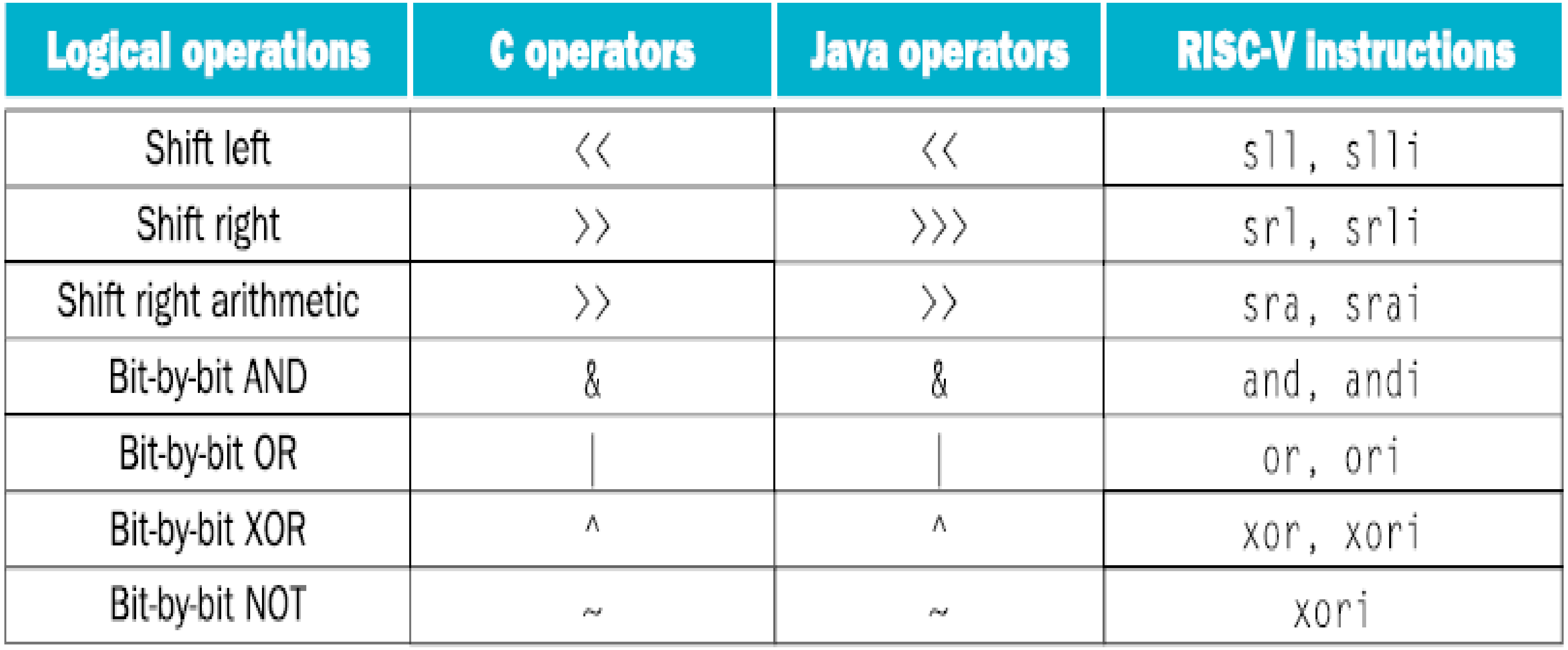
RISC V ARCHITECTURE

UNIT 2 – Instructions: The Language of Computer

Mahesh Awati
Department of Electronics and Communication Engineering

Instructions – Language of Computer

Logical Operations

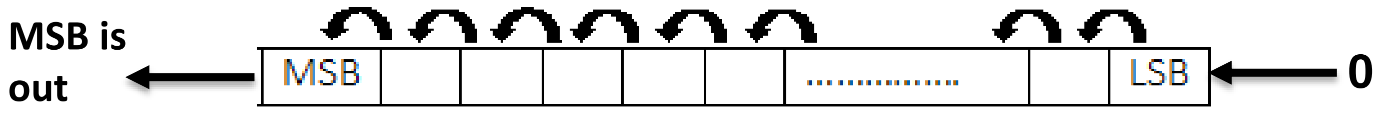
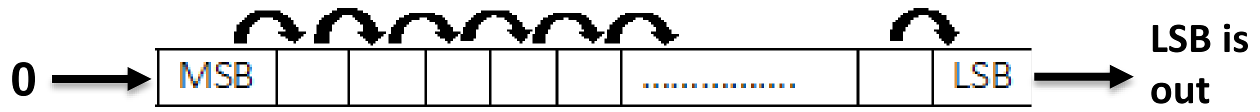
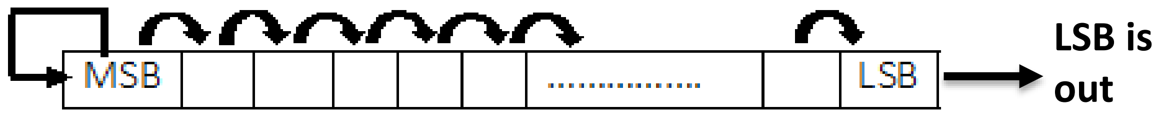


Logical operations	C operators	Java operators	RISC-V Instructions
Shift left	<<	<<	sll, slli
Shift right	>>	>>>	srl, srli
Shift right arithmetic	>>	>>	sra, srai
Bit-by-bit AND	&	&	and, andi
Bit-by-bit OR	|	|	or, ori
Bit-by-bit XOR	^	^	xor, xori
Bit-by-bit NOT	~	~	xori

Instructions – Language of Computer

Logical Operations

SHIFT Operations

Mnemonic	Descriptions
Logical Shift Left **sll , slli**	Content of Source Operand is shifted Left by Count times. Count is either an immediate value (#0-31 or Register Rs 
Logical Shift Right **srl , srli**	Content of Source Operand (Unsigned Number) is shifted Right by Count times. Count is either a immediate value (#0-31 or Register Rs. 
Arithmetic Right Shift **sra , srai**	Content of Source Operand (Signed Number) is shifted Right by Count times. It means in this status of MSB bit is shifted to right as well as copied into MSB itself. Count is either a immediate value (#0-31 or Register Rs. 

Instructions – Language of Computer

Logical Operations

Shift Left Logically Left

Syntax	slli rd, rs1,imm
Operation:	It logically shifts left the content of rs1 by immediate time (Note: Immediate value can be max of 32) and stores the shifted result in rd. Vacated bits on the right(lsb) is filled with zeros. i.e., rd=rs1<<imm

Example:	slli x11, x19, 4;
	x11=x19<<4

Before Execution	
x19=9$_{ten}$ =	0000 0000 0000 0000 0000 0000 0000 1001
x11= xxxxxxxx	
Imm =4	
After Execution	
x19=9$_{ten}$ =	0000 0000 0000 0000 0000 0000 0000 1001 b
x11= 144$_{ten}$ =	**0000 0000 0000 0000 0000 0000 1001 0000 b [9x16=144]**
Imm =	**4**

Shifting left by *i* bits gives the identical result as multiplying by 2^i

sll x11,x19,x20

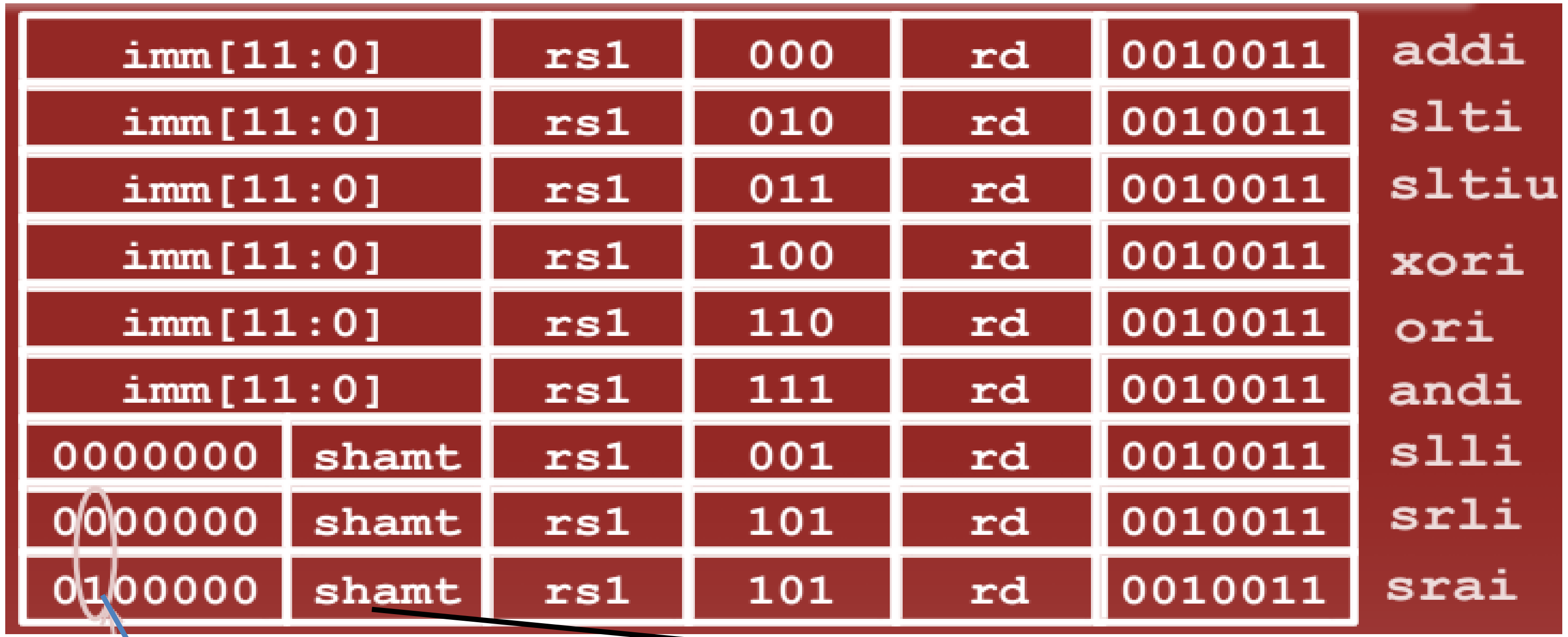
- In this shift count is basically the value held in x20.
- Functionally it does the same operation
- X11=x19<<x20 (count)

Instructions – Language of Computer

Representing Instructions in the Computer

How I-Format Instruction are Encoded ?

Syntax: mnemonics rd,rs1,imm



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

These shift instructions use the I-type format. Since it isn't useful to shift a 32-bit register by more than 31 bits, only the lower bits of the I-type format's 12-bit immediate are actually used. The remaining bits are repurposed as an additional opcode field, funct7.

One of the higher-order immediate bits is used to distinguish "shift right logical" (SRLI) from "shift right arithmetic" (SRAI)

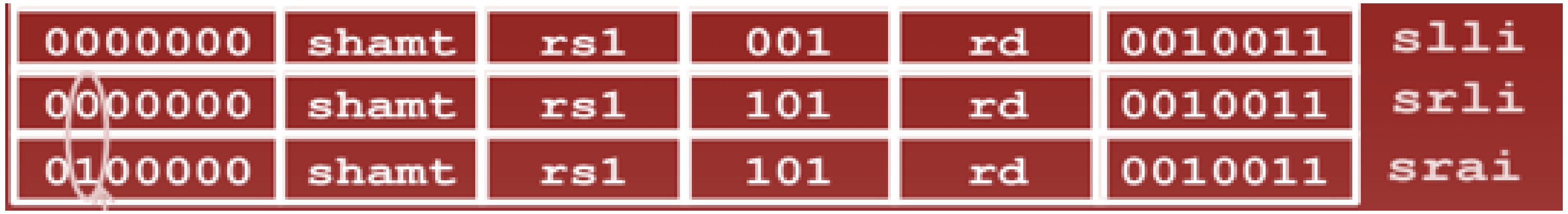
"Shift-by-immediate" instructions only use lower 5 bits of the immediate value for shift amount (can only shift by 0-31 bit positions)

Instructions – Language of Computer

Representing Instructions in the Computer

I-type Instruction Format

Syntax: shift rd,rs1,imm



0000000	shamt	rs1	001	rd	0010011	slli
0000000	shamt	rs1	101	rd	0010011	srli
0100000	shamt	rs1	101	rd	0010011	srai

slli x11, x19, 4

19d = 13h

Funct7	shamt	rs1	funct3	rd	001 0011
0000 000	0 0100	1001 1	001	0101 1	0010011

Hexadecimal Representation: 0x00499593

Instructions – Language of Computer

Logical Operations

Shift Left Logically Left

Syntax	sll rd, rs1,rs2
Operation:	It logically shifts left the content of rs1 by rs2 times (Note: rs2 should be holding the count for shifting) and stores the shifted result in rd. Vacated bits on the right (lsb) is filled with zeros. i.e., rd=rs1<<rs2

Example:	sll x11, x19,x20;
	x11=x19<<x20

Before Execution	
x19 = 9$_{ten}$	= 0000 0000 0000 0000 0000 0000 0000 1001
x11	**= xxxxxxxx**
x20	**= 4$_{ten}$**
After Execution	
x19=9$_{ten}$	= 0000 0000 0000 0000 0000 0000 0000 1001 b
x11= 144$_{ten}$	**= 0000 0000 0000 0000 0000 0000 1001 0000 b [9x16=144]**
x20	**= 4$_{ten}$**

Shifting left by *i* bits gives the identical result as multiplying by 2^i

Instructions – Language of Computer

Logical Operations

Shift Left Logically Left

Syntax	srli rd, rs1,imm
Operation:	It logically shifts right the content of rs1 by immediate time (Note: Immediate value can be max of 32) and stores the shifted result in rd. Vacated bits on the left(msb) is filled with zeros i.e., rd=rs1>>imm

Example:	srli x11, x19, 4;
	x11=x19>>4

Before Execution	
x19=9$_{ten}$ = **x11= xxxxxxxx** **Imm =4**	0000 0000 0000 0000 0000 1000 0000 0000
After Execution	
x19=2048$_{ten}$ = **x11= 128$_{ten}$ =** **Imm =**	0000 0000 0000 0000 0000 1000 0000 0000 b **0000 0000 0000 0000 0000 0000 1000 0000 b [2048/16=128]** **4**

Shifting left by *i* bits gives the identical result as dividing by 2^i

srl x11,x19,x20

- In this shift count is basically the value held in x20.
- Functionally it does the same operation as that of srli but count is value stored in rs2
- X11=x19<<x20 (count)

Instructions – Language of Computer

Logical Operations

Shift Left Logically Left

Syntax	srai rd, rs1,imm
Operation:	It logically shifts right the content of rs1 by immediate time (Note: Immediate value can be max of 32) and stores the shifted result in rd. Vacated bits on the left is filled them with copies of the old sign bit.

Example:	srli x11, x19, 4;
	x11=x19>>4

Before Execution	
x19=9_{ten} = **x11= xxxxxxxx** **Imm =4**	1000 0000 0000 0000 0000 1000 0000 0000
After Execution	
x19=2048_{ten} = **x11= 128_{ten}** **=** **Imm** **=**	1000 0000 0000 0000 0000 1000 0000 0000 b **1111 1000 0000 0000 0000 0000 1000 0000 b** **4**

Shifting left by *i* bits gives the identical result as signed division by 2^i

Instructions – Language of Computer

Logical Operations

Shift Left Logically Left

Examples:

1) Write the assembly code to **load the value of 17 in x6**, and then to **multiply it by 4** and store the result in x5. Use sll/slli
2) Write the assembly code to calculate a) (88 / 8) use srl/srli b) (-88/3) use srl/srli c) (-88/3) use sra/srai. Comment on result obtained for b) and c)
3) Shift instructions are often used for extracting subsequences of bits. Write the assembly code to move bits [7:4] of the value in x6 to the 4 least significant bits [3:0] of x5 nullifying all its other bits. Use addi and slli,srli
4) Store the value of 0x0000123400000000 in x5 using only addi and slli instructions.

Instructions – Language of Computer

Examples

Shift Left Logically Left

Examples:

1) Extract bits [7:4] of the value in x6 by directly masking out all other bits using the andi (and immediate) instruction and then move the bits to the LSB part.
2) Write a assembly code to combine the middle hexadecimal digit of the value 0x123 with the 1st and the 3rd hexadecimal digits of the value 0x456 and store in result 0x423 in x5.
3) Rotate right by 4 bits the value of 0x0000000000000123. The expected result is 0x3000000000000012, i.e. all hexadecimal digits move right by one position while the rightmost one moves to the front.

Instructions – Language of Computer

Examples

For the following C statement, write the corresponding RISC-V assembly code. Assume that the variables f, g, h, i, and j are assigned to registers x5, x6, x7, x28, and x29, respectively. Assume that the base address of the arrays A and B are in registers x10 and x11, respectively.

B[8] = A[i–j];

```
sub x30, x28, x29 // compute i-j  (x28 – x29)
slli x30, x30, 2       // 32 bit data at [i-j]th element of Array A is at offset 4x[i-j]
add x3, x30, x10  // physical address of A[i-j] = Base Address + 4 x [i-j]
lw x30, 0(x3)         // load A[i-j]
sw x30, 32(x11)  // store in the content of x30 into B[8] =B[BA + 4x8] = B[x11 + 32]
```

Variables	Registers
f	x5
g	x6
h	x7
i	x28
j	x29
Base Address of A	x10
Base Address of B	x11

Memory	0x0000	0x0004	0x0008	……………….	(i-j)x4	[(i-j)+1]x4	[(i-j)+2]
Element	0	1	2	……………….	(i-j)	[(i-j)+1]	[(i-j)+2]
A[]	A[0]	A[1]	A[2]	……………….	A[i-j]	A[(i-j)+1]	A[(i-j)+2]

PES UNIVERSITY ONLINE

THANK YOU

Mahesh Awati
Department of Electronics and Communication

mahesha@pes.edu

+91 9741172822