Assembly Language Lecture Series: RV32I: Integer Computation (Reg-Reg) instructions

Roger Luis Uy
College of Computer Studies
De La Salle University
Manila, Philippines

Integer computation instructions

Integer computation instructions (register-register)				
ADD	SLT	SLTU	AND	OR
XOR	SLL	SRL	SRA	SUB
NOP				

ADD instruction

ADD *rd*, *rs1, rs2*

- performs addition of *rs1* and *rs2*. Arithmetic overflow is ignored, and the low 32 bits of the result is written in *rd*.
- ADD *rd*, x0, *rs* is the assembler pseudoinstruction of MV *rd*, *rs*.

Example:

```
var1: .byte 0x05
var2: .byte 0x06
la x5, var1
la x6, var2
lb x10, 0(x5)
lb x11, 0(x6)
add x12, x10, x11
```

After execution:

x10 = 00000005

x11 = 00000006

x12 = 0000000B

SUB instruction

SUB *rd*, *rs1, rs2*

- subtraction of rs2 from rs1. Arithmetic overflow is ignored, and the low 32 bits of the result is written in *rd*.

Example:

var1: .byte 0x05

var2: .byte 0x06

la x5, var1

la x6, var2

lb x10, 0(x5)

lb x11, 0(x6)

sub x12, x10, x11

After execution:

x10 = 00000005

x11 = 00000006

x12 = FFFFFFFF

AND instruction

AND *rd*, *rs1, rs2*

- AND is a logical operation that performs bitwise AND on registers *rs*1 and *rs*2 place the result in *rd*.

Example:

var1: .byte 0x05

var2: .byte 0x05

la x5, var1

la x6, var2

lb x10, 0(x5)

lb x11, 0(x6)

AND x12, x10, x11

After execution:

x10 = 00000005

x11 = 00000005

x12 = 00000005

OR instruction

OR *rd*, *rs1, rs2*

- OR is a logical operation that performs bitwise OR on registers *rs*1 and *rs*2 place the result in *rd*.

Example:

var1: .byte 0x05

var2: .byte 0xFF

la x5, var1

la x6, var2

lb x10, 0(x5)

lb x11, 0(x6)

OR x12, x10, x11

After execution:

x10 = 00000005

x11 = FFFFFFFF

x12 = FFFFFFFF

XOR instruction

XOR *rd*, *rs1, rs2*

- XOR is a logical operation that performs bitwise XOR on registers *rs*1 and *rs*2 place the result in *rd*.

Example:

```
var1: .byte 0x05
var2: .byte 0x05
la x5, var1
la x6, var2
lb x10, 0(x5)
lb x11, 0(x6)
XOR x12, x10, x11
```

After execution:

x10 = 00000005

x11 = 00000005

x12 = 00000000

SLL instruction

SLL *rd*, *rs1, rs2*

- SLL (shift left logical)- the operand to be shifted is in *rs*1 by the shift amount held in the lower **5 bits** of register *rs*2 and the result is placed in *rd*. Zeros are shifted into the lower bits.

Example:

```
var1: .byte 0x05
var2: .byte 0x04
la x5, var1
la x6, var2
lb x10, 0(x5)
lb x11, 0(x6)
SLL x12, x10, x11
```

After execution:

x10 = 00000005

x11 = 00000004

x12 = 00000050

Example:

```
var1: .word 0xFFFFFFFF
var2: .byte 0x1f
la x5, var1
la x6, var2
lw x10, 0(x5)
lb x11, 0(x6)
SLL x12, x10, x11
```

After execution:

x10 = FFFFFFFF

x11 = 0000001F

x12 = 80000000

SRL instruction

SRL *rd*, *rs1, rs2*

- SRL (shift right logical)- the operand to be shifted is in *rs*1 by the shift amount held in the lower **5 bits** of register *rs*2 and the result is placed in *rd*. Zeros are shifted into the upper bits.

Example:

var1: .byte 0x35

var2: .byte 0x04

la x5, var1

la x6, var2

lb x10, 0(x5)

lb x11, 0(x6)

SRL x12, x10, x11

After execution:

x10 = 00000035

x11 = 00000004

x12 = 00000003

Example:

var1: .word 0xFFFFFFFF

var2: .byte 0x1f

la x5, var1

la x6, var2

lw x10, 0(x5)

lb x11, 0(x6)

SRL x12, x10, x11

After execution:

x10 = FFFFFFFF

x11 = 0000001F

x12 = 00000001

SRA instruction

SRA *rd*, *rs1, rs2*

- SRA (shift right arithmetic)- the operand to be shifted is in *rs*1 by the shift amount held in the lower **5 bits** of register *rs*2 and the result is placed in *rd*. Original sign bit is copied into the vacated upper bits.

Example:

```
var1: .word 0x8000000F
var2: .byte 0x04
la x5, var1
la x6, var2
lw x10, 0(x5)
lb x11, 0(x6)
SRA x12, x10, x11
```

After execution:

x10 = 8000000F

x11 = 00000004

x12 = F8000000

SLT instruction

SLT *rd*, *rs1, rs2*

- SLT (set if less than) performs signed compare, places the value 1 in register *rd* if register *rs*1 < *rs*2, 0 otherwise.

Example:

var1: .byte 0x04

var2: .byte 0x03

la x5, var1

la x6, var2

lb x10, 0(x5)

lb x11, 0(x6)

SLT x12, x10, x11

After execution:

x10 = 00000004

x11 = 00000004

x12 = 00000000

Example:

var1: .word 0xFFFFFFFF

var2: .byte 0x04

la x5, var1

la x6, var2

lw x10, 0(x5)

lb x11, 0(x6)

SLT x12, x10, x11

After execution:

x10 = FFFFFFFF

x11 = 00000004

x12 = 00000001

SLTU instruction

SLTU *rd*, *rs1, rs2*

- SLTU (set if less than unsigned) performs unsigned compare, places the value 1 in register *rd* if register *rs*1 < *rs*2, 0 otherwise.

Example:

var1: .byte 0x03

var2: .byte 0x04

la x5, var1

la x6, var2

lb x10, 0(x5)

lb x11, 0(x6)

SLT x12, x10, x11

After execution:

x10 = 00000003

x11 = 00000004

x12 = 00000001

Example:

var1: .word 0xFFFFFFFF

var2: .byte 0x04

la x5, var1

la x6, var2

lw x10, 0(x5)

lb x11, 0(x6)

SLTU x12, x10, x11

After execution:

x10 = FFFFFFFF

x11 = 00000004

x12 = 00000000

NOP instruction

- NOP performs nothing except for advancing the *pc*. NOP is encoded as ADDI x0, x0, 0.