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

Roger Luis Uy

College of Computer Studies

De La Salle University

Manila, Philippines

'2301

Integer computation instructions

Integer computation instructions (register-immediate)				
ADDI	SLTI	SLTIU	ANDI	ORI
XORI	SLLI	SRLI	SRAI	LUI
AUIPC				

ADDI instruction

ADDI *rd, rs1, imm*

- adds the sign-extended 12-bit immediate to register *rs1*. Arithmetic overflow is ignored, and the low 32-bit of the result is written in *rd*.
- Constant range is from +2047 to -2048 (0x000007FF to 0xFFFFF800)
- ADDI *rd, x0, imm* is the assembler pseudoinstruction of *li rd, imm*.

Example:

```
addi x10, x0, 0x05
addi x11, x0, 0x06
add x12, x10, x11
```

After execution:

```
x10 = 00000005
x11 = 00000006
x12 = 0000000B
```

Example:

```
addi x10, x0, 5
addi x11, x0, -1
add x12, x10, x11
```

After execution:

```
x10 = 00000005
x11 = FFFFFFFF
x12 = 00000004
```

ANDI instruction

ANDI *rd, rs1, imm*

- ANDI is a logical operation that performs bitwise AND on register *rs*1 and the sign-extended 12-bit immediate and place the result in *rd*.

Example:

ADDI x10, x0, 0x05

ANDI x11, x10, 0xFFFFFFFF

After execution:

x10 = 00000005

x11 = 00000005

ORI instruction

ORI *rd, rs1, imm*

- ORI is a logical operation that performs bitwise OR on register *rs*1 and the sign-extended 12-bit immediate and place the result in *rd*.

Example:

ADDI x10, x0, 0x05

ORI x11, x10, 0xFFFFFFFF

After execution:

x10 = 00000005

x11 = FFFFFFFF

XORI instruction

XORI *rd, rs1, imm*

- XORI is a logical operation that performs bitwise XOR on register *rs*1 and the sign-extended 12-bit immediate and place the result in *rd*.

Example:

ADDI x10, x0, 0x05

XORI x11, x10, 0xFFFFFFFF (looks like NOT x11, x10)

After execution:

x10 = 00000005

x11 = FFFFFFFA

SLLI instruction

SLLI *rd, rs1, imm*

- SLLI (shift left logical immediate) - the operand to be shifted is in *rs*1 and the shift amount is encoded in the lower 5 bits of the immediate field and the result is placed in *rd*. Zeros are shifted into the lower bits.

Example:

```
var1: .byte 0x05
la x5, var1
lb x10, 0(x5)
SLLI x12, x10, 0x04
```

After execution:

```
x10 = 00000005
x12 = 00000050
```

Example:

```
var1: .word 0xFFFFFFFF
la x5, var1
lw x10, 0(x5)
SLLI x12, x10, 0x1F
```

After execution:

```
x10 = FFFFFFFF
x12 = 80000000
```

SRLI instruction

SRLI *rd, rs1, imm*

- SRLI (shift right logical immediate)- the operand to be shifted is in *rs*1 and the shift amount is encoded in the lower 5 bits of the immediate field and the result is placed in *rd*. Zeros are shifted into the upper bits.

Example:

```
var1: .byte 0x35
la x5, var1
lb x10, 0(x5)
SRLI x12, x10, x04
```

After execution:

x10 = 00000035

x12 = 00000003

Example:

```
var1: .word 0xFFFFFFFF
la x5, var1
lw x10, 0(x5)
SRLI x12, x10, 0x1F
```

After execution:

x10 = FFFFFFFF

x12 = 00000001

SRAI instruction

SRAI *rd, rs1, imm*

- SRAI (shift right arithmetic)- the operands to be shifted is in *rs*1 and the shift amount is encoded in the lower 5 bits of the immediate field and the result is placed in *rd*. Original sign bit is copied into the vacated upper bits.

Example:

```
var1: .word 0x8000000F


la x5, var1
lw x10, 0(x5)
SRAI x12, x10, 0x04
```

After execution:

x10 = 8000000F

x12 = F8000000

SLTI instruction

SLTI *rd*, *rs1, imm*

- SLTI (set if less than) places the value 1 in register *rd* if register *rs*1 is less than the sign-extended immediate.; else 0 is written to *rd*.
- Both *rs1* and *imm* are treated as signed integers

Example:

```
var1: .byte 0x04


la x5, var1
lb x10, 0(x5)
SLTI x12, x10, 0x03
```

After execution:

x10 = 00000004

x12 = 00000000

Example:

```
var1: .word 0xFFFFFFFF


la x5, var1
lw x10, 0(x5)
SLTI x12, x10, 0x04
```

After execution:

x10 = FFFFFFFF

x12 = 00000001

SLTIU instruction

SLTIU *rd, rs1, imm*

- SLTIU (set if less than unsigned) places the value 1 in register *rd* if register *rs*1 is less than the sign-extended immediate; else 0 is written to *rd*.
- Both *rs1* and *imm* are treated as unsigned integers

Example:

```
var1: .byte 0x03


la x5, var1
lb x10, 0(x5)
SLTIU x12, x10, 0x04
After execution:
x10 = 00000003
x12 = 00000001
```

Example:

```
var1: .word 0xFFFFFFFF


la x5, var1
lw x10, 0(x5)
SLTIU x12, x10, 0x04
After execution:
x10 = FFFFFFFF
x12 = 00000000
```

LUI instruction

LUI *rd*

- LUI (load upper immediate) places the immediate value in the top 20 bits of the destination register *rd*, filling in the lowest 12 bits with zeros.

Example:

LUI x12, 0x12345

After execution:

x12 = 12345000

AUIPC instruction

AUIPC *rd, imm*

- AUIPC (add upper immediate to pc) forms a 32-bit offset from the 20-bit immediate, filling in the lowest 12 bits with zeros, adds this offset to the address of the AUIPC instruction, then places the result in register *rd*.
- The current PC can be obtained by setting the immediate to 0.

Example:

AUIPC x12, 0x0FC10

(assume pc = 0x00400000)

After execution:

x12 = 10010000

(0FC10000 + 00400000)

AUIPC instruction

- The combination of an AUIPC and the 12-bit immediate in a JALR can transfer control to any 32-bit PC-relative address,
- The combination of an AUIPC plus the 12-bit immediate offset in regular load or store instructions can access any 32-bit PC-relative data address.