Assembly Language Lecture Series: RV32M instructions: Multiplication and Division

Roger Luis Uy

College of Computer Studies

De La Salle University

Manila, Philippines

‘2112

RV32M instructions

RV32M instructions				
MUL	MULH	MULHU	MULHSU	--
DIV	DIVU	REM	REMU	--

MUL instruction

MUL *rd*, *rs*1, *rs*2

- MUL performs a 32-bit×32-bit multiplication of *rs*1 by *rs*2 and places the lower 32 bits in the destination register.

Example:

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

After execution:

```
x10 = 00000005
x11 = 00000006
x12 = 0000001E
```

Example:

```
addi x10, x0, 0x05
addi x11, x0, 0xFFFFFFFE
mul x12, x10, x11
```

After execution:

```
x10 = 00000005
x11 = FFFFFFFE
x12 = FFFFFFF6
```

*same as:

```
addi x11, x0, -2
```

MULHU instruction

MULHU *rdh*, *rs*1, *rs*2

- MULHU performs multiplication but returns the upper 32 bits of the full 2×32-bit product, for unsigned multiplication.
- If both the high and low bits of the same product are required, then the recommended code sequence is: MULHU *rdh*, *rs1*, *rs2*; MUL *rdl*, *rs1*, *rs2* (source register specifiers must be in the same order and *rdh* cannot be the same as *rs1* or *rs2*).
- Microarchitectures can then fuse these into a single multiply operation instead of performing two separate multiplication.

Example:

```
var1: .word 0xFFFFFFFF
var2: .word 0xFFFFFFFF

la t0, var1
la t1, var2
lw x10, (t0)
lw x11, (t1)
mulhu x12, x10, x11
mul x13, x10, x11
```

After execution:

```
x10 = FFFFFFFF
x11 = FFFFFFFF
x12 = FFFFFFFE
x13 = 00000001
```

MULH instruction

MULH *rdh*, *rs*1*, rs*2

- MULH performs multiplication but return the upper 32 bits of the full 2×32-bit product, for signed multiplication.
- If both the high and low bits of the same product are required, then the recommended code sequence is: MULH *rdh*, *rs1*, *rs2*; MUL *rdl*, *rs1*, *rs2* (source register specifiers must be in same order and *rdh* cannot be the same as *rs1* or *rs2*).
- Microarchitectures can then fuse these into a single multiply operation instead of performing two separate multiplies.

Example:

```
var1: .word 0xFFFFFFFF
var2: .word 0xFFFFFFFF

la t0, var1
la t1, var2
lw x10, (t0)
lw x11, (t1)
mulh x12, x10, x11
mul x13, x10, x11
```

After execution:

x10 = FFFFFFFF

x11 = FFFFFFFF

x12 = 00000000

x13 = 00000001

MULHSU instruction

MULHSU *rdh*, *rs*1*, rs*2

- MULHSU performs multiplication but return the upper 32 bits of the full 2×32-bit product, for signed *rs1* and unsigned *rs2* multiplication.
- If both the high and low bits of the same product are required, then the recommended code sequence is: MULHSU *rdh*, *rs1*, *rs2*; MUL *rdl*, *rs1*, *rs2* (source register specifiers must be in same order and *rdh* cannot be the same as *rs1* or *rs2*).
- Microarchitectures can then fuse these into a single multiply operation instead of performing two separate multiplies.

Example 1:

addi x10, 0, -4 (0xFFFFFFFC view as signed → -4)

addi x11, 0, -5 (0xFFFFFFFB view as unsigned → 4294967291)

mulhsu x12, x10, x11 (-4*4294967291)

mul x13, x10, 11

After execution:

x10 = FFFFFFFC

x11 = FFFFFFFB

x12 = FFFFFFFC

x13 = 00000014

MULHSU instruction

MULHSU *rdh*, *rs*1*, rs*2

- MULHSU performs multiplication but return the upper 32 bits of the full 2×32-bit product, for signed *rs1* and unsigned *rs2* multiplication.
- If both the high and low bits of the same product are required, then the recommended code sequence is: MULHSU *rdh*, *rs1*, *rs2*; MUL *rdl*, *rs1*, *rs2* (source register specifiers must be in same order and *rdh* cannot be the same as *rs1* or *rs2*).
- Microarchitectures can then fuse these into a single multiply operation instead of performing two separate multiplies.

Example 2:

addi x10, 0, +4 (0x00000004 view as signed → +4)

addi x11, 0, -5 (0xFFFFFFFB view as unsigned → 4294967291)

mulhsu x12, x10, x11 (+4*4294967291)

mul x13, x10, 11

After execution:

x10 = 00000004

x11 = FFFFFFFB

x12 = 00000003

x13 = FFFFFFEC

MULHSU instruction

MULHSU *rdh*, *rs*1*, rs*2

- MULHSU performs multiplication but return the upper 32 bits of the full 2×32-bit product, for signed *rs1* and unsigned *rs2* multiplication.
- If both the high and low bits of the same product are required, then the recommended code sequence is: MULHSU *rdh*, *rs1*, *rs2*; MUL *rdl*, *rs1*, *rs2* (source register specifiers must be in same order and *rdh* cannot be the same as *rs1* or *rs2*).
- Microarchitectures can then fuse these into a single multiply operation instead of performing two separate multiplies.

Example 3:

addi x10, 0, -4 (0xFFFFFFFC view as signed → -4)

addi x11, 0, +5 (0x00000005 view as unsigned → +5)

mulhsu x12, x10, x11 (-4*+5)

mul x13, x10, 11

After execution:

x10 = FFFFFFFC

x11 = 00000005

x12 = FFFFFFFF

x13 = FFFFFFEC

MULHSU instruction

MULHSU *rdh*, *rs*1*, rs*2

- MULHSU performs multiplication but return the upper 32 bits of the full 2×32-bit product, for signed *rs1* and unsigned *rs2* multiplication.
- If both the high and low bits of the same product are required, then the recommended code sequence is: MULHSU *rdh*, *rs1*, *rs2*; MUL *rdl*, *rs1*, *rs2* (source register specifiers must be in same order and *rdh* cannot be the same as *rs1* or *rs2*).
- Microarchitectures can then fuse these into a single multiply operation instead of performing two separate multiplies.

Example 4:

addi x10, 0, +4 (0x00000004 view as signed → +4)

addi x11, 0, +5 (0x00000005 view as unsigned → +5)

mulhsu x12, x10, x11 (+4*+5)

mul x13, x10, 11

After execution:

x10 = 00000004

x11 = 00000005

x12 = 00000000

x13 = 00000014

DIV/REM instruction

DIV *rd*, *rs*1*, rs*2 */* REM *rd*, *rs*1*, rs*2

- DIV performs a 32-bit by 32-bit signed integer division of *rs1* by *rs2*, rounding towards zero (*rs1*/*rs2*).
- REM provides the remainder of the corresponding division operation.
- For REM, the sign of the remainder follows the sign of the dividend.
- If both the quotient and remainder are required from the same division, the recommended code sequence is: DIV *rdq*, *rs1*, *rs2*; REM *rdr*, *rs1*, *rs2* (*rdq* cannot be the same as *rs1* or *rs2*).
- Microarchitectures can then fuse these into a single divide operation instead of performing two separate division

Example:

```
addi x10, x0, +23
addi x11, x0, -5
div x12, x10, x11
rem x13, x10, x11
```

After execution:

```
x10 = 00000017
x11 = FFFFFFFB
x12 = FFFFFFFC
x13 = 00000003
```

Example:

```
addi x10, x0, -23
addi x11, x0, +5
div x12, x10, x11
rem x13, x10, x11
```

After execution:

```
x10 = FFFFFFE9
x11 = 00000005
x12 = FFFFFFFC
x13 = FFFFFFFD
```

DIVU/REMU instruction

DIVU *rd*, *rs*1*, rs*2 */* REMU *rd*, *rs*1*, rs*2

- DIVU performs a 32-bit by 32-bit unsigned integer division of *rs1* by *rs2*, rounding towards zero (*rs1*/*rs2*).
- REMU provides the remainder of the corresponding division operation.
- If both the quotient and remainder are required from the same division, the recommended code sequence is: DIVU *rdq*, *rs1*, *rs2*; REMU *rdr*, *rs1*, *rs2* (*rdq* cannot be the same as *rs1* or *rs2*).
- Microarchitectures can then fuse these into a single divide operation instead of performing two separate division

Example:

```
addi x10, x0, 23
addi x11, x0, 5
divu x12, x10, x11
remu x13, x10, x11
```

After execution:

```
x10 = 00000017
x11 = 00000005
x12 = 00000004
x13 = 00000003
```

Division instruction

- Division by zero (both signed and unsigned):
  - quotient: all bits set
  - remainder: dividend
- Signed division overflow:
  - quotient: dividend
  - remainder: zero