RISC-V ISA

Arithmetic Instructions: Multiply & Divide

•

Sparsh Mittal

•

MUL and MULH instructions

- **MUL (Multiply signed and return lower bits): :** Calculates the product of two XLEN-bit operands, stores less significant XLEN bits in rd and ignores any overflow
- Syntax: mul rd, rs1, rs2
- Example: mul x4, x9, x13   # x4 ← LowerBits [x9 * x13]
- **MULH (Multiply signed and return upper bits):** multiplies two signed numbers and returns upper XLEN bits of the full 2*XLEN product, into rd.
- Syntax: mulh rd, *rs*1, *rs*2
- Example: mulh x5, x9, x13   # x5 ← HigherBits [x9 * x13]
- Use MULH with MUL to get the complete 2*XLEN bits result.

MULHU and MULHSU

- MULHU: Multiplies two unsigned operands in the source registers and returns upper-part of result.
- MULHSU (Multiply Signed-Unsigned and return upper bits): first operand is signed, second is unsigned

- MUL performs an n-bit×n-bit multiplication and places the lower ‘n’ bits in the destination register.
- MULH, MULHU, and MULHSU perform the same multiplication but return the upper ‘n’ bits of the full 2×n-bit product, for signed×signed, unsigned×unsigned, and signed×unsigned multiplication respectively.

Example of MUL and MULH

- We multiply two numbers, both of which are 00010000000000000001_2 or 10001_{16} or 65537_{10}
- The product comes to be 4,29,50,98,369
- In hexadecimal, it is $\underbrace{00000001}_{\text{Upper XLEN bits}}\underbrace{00020001}_{\text{Lower XLEN bits}}$

XLEN=32 bits

Obtained from MULH

Obtained from MUL

Code for Example of MUL and MULH

```
.globl _start
_start:
        li a0, 65537            # a0 <- 0x00010001 = 65537₁₀
        li a1, 65537            # a1 <- 0x00010001 = 65537₁₀
        mul a2, a0, a1          # a2 <- 0x00020001 = 131073
        mulh a3, a0, a1         # a3 <- 0x00000001 = Upper XLEN bits
exit:
```

- 65,537 x 65537 = 4,29,50,98,369
- 4294967295 + 131073 = 4,29,50,98,369

Code for Example of MULHU, MULHSU

```
.globl _start
_start:
        li a0, 2147483648     # a0 <- 0x80000000 = 2^31 (Unsigned)
                              #                  = -2^31 (Signed)
        li a1, 3221225472     # a1 <- 0xC0000000 = 3 * 2^30 (unsigned)
        mul a3, a0, a1        # a3 <- 0x00000000 = Lower XLEN bits are all zero
        mulhu a4, a0, a1           # a4 <- 0x60000000 = Upper XLEN bits of 3 * 2^61
        mulhsu a5, a0, a1          # a5 <- 0xA0000000 = Upper XLEN bits of -3 * 2^61
exit:
```

Optimization

- Even though we require two separate instructions now, micro-architectures can fuse them dynamically.
- They can identify two consecutive multiplication instructions where one instruction computes the lower 32 bits and the next instruction computes the upper 32 bits.
- This sequence can be identified dynamically and a single multiplication will only be required.

Division operation

$$Quotient = (Dividend - Remainder) \div Divisor$$

$$Dividend = Quotient \times Divisor + Remainder$$

$$Remainder = Dividend - (Quotient \times Divisor)$$

DIV/DIVU and REM/REMU

- DIV does the division of operands in source registers and stores quotient in rd.
- Both operands and the result are signed values.
- DIVU: same as DIV, except that both are unsigned values
- REM: stores the remainder in rd. Both are signed values.
- REMU: same as REM, but both are unsigned values.
- First use DIV and then use REM to get quotient and remainder

Example of DIV/DIVU and REM/REMU

li a0, 2147483648 # a0 <- 0x80000000 = 2^{31} (Unsigned), -2^{31} (Signed)

li a1, 1073741824 # a1 <- 0x40000000 = 2^{30}

div a2, a0, a1 # a2 <- 0xFFFFFFFE = -2

divu a3, a0, a1 # a3 <- 0x000000002 = 2

li a0, 2147483649 # a0 <- 0x80000001 = $(2^{31} + 1)$ (Unsigned), $(-2^{31} + 1)$(Signed)

li a1, 2 # a1 <- 0x000000002 = 2

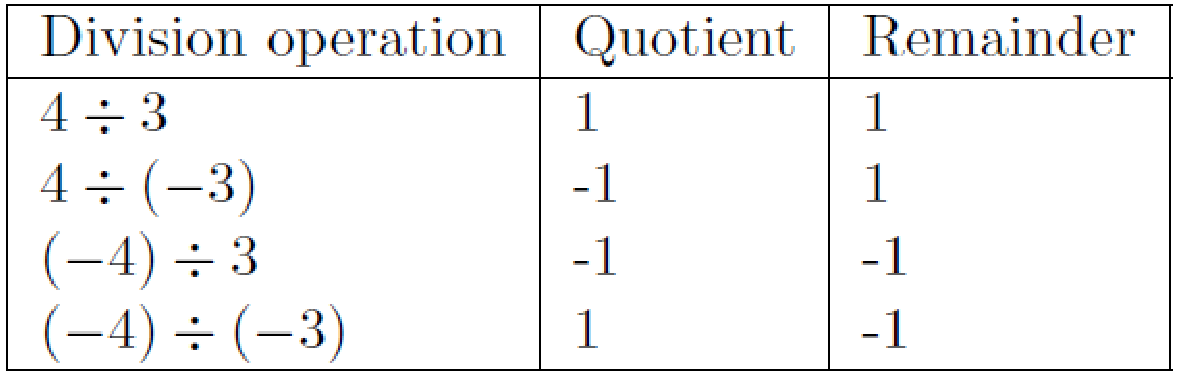
rem a4, a0, a1 # a4 <- 0xFFFFFFFF = -1

remu a5, a0, a1 # a4 <- 0x00000001 = 1

0x80000001 interpreted as signed is -2147483647
0x80000001 interpreted as unsigned is 2147483649

Point to note

- Rounding happens towards zero.
- ➔ the sign of the remainder is the same as the sign of the dividend.



Division operation	Quotient	Remainder
$4 \div 3$	1	1
$4 \div (-3)$	-1	1
$(-4) \div 3$	-1	-1
$(-4) \div (-3)$	1	-1

Example program using MUL and DIV

- RISC-V code to compute volume of a sphere (4/3* pi* r*r*r). ‘r’ is in a0. Take pi=3. (Two different ways are shown below)

Code:	**Comments:**
li a2 4	# load 4 into a2
li a3 3	# load 3 into a3
li a4 3	# load pi into a4
mul a1 a0 a0	# compute r*r
mul a1 a1 a0	# compute r*r*r
mul a1 a1 a4	# compute pi*r*r*r
mul a1 a1 a2	# compute 4*pi*r*r*r
div a1 a1 a3	# compute 4/3*pi*r*r*r

Code:	**Comments:**
li a1, 1	
li x1, 3	
loop:	# loop to calculate r*r*r
mul a1, a1, a0	
addi x1, x1, -1	
bnez x1, loop	
li x2, 4	
li x3, 3	
div a2, x2, x3	
li a3, 3	
mul a1, a1, a2	
mul a1, a1, a3	

What happens if you divide by zero

- RV32I doesn’t trap on divide by zero because few programs want that behavior, and the ones that do can easily check for zero in software by using beqz instruction.

Summary

Format	Name	Pseudocode
MUL rd,rs1,rs2	Multiply	rd ← ux(rs1) × ux(rs2)
MULH rd,rs1,rs2	Multiply High Signed Signed	rd ← (sx(rs1) × sx(rs2)) » xlen
MULHSU rd,rs1,rs2	Multiply High Signed Unsigned	rd ← (sx(rs1) × ux(rs2)) » xlen
MULHU rd,rs1,rs2	Multiply High Unsigned Unsigned	rd ← (ux(rs1) × ux(rs2)) » xlen
DIV rd,rs1,rs2	Divide Signed	rd ← sx(rs1) ÷ sx(rs2)
DIVU rd,rs1,rs2	Divide Unsigned	rd ← ux(rs1) ÷ ux(rs2)
REM rd,rs1,rs2	Remainder Signed	rd ← sx(rs1) mod sx(rs2)
REMU rd,rs1,rs2	Remainder Unsigned	rd ← ux(rs1) mod ux(rs2)

Summary

RV32M

$$\underline{\texttt{mul}}\text{tiply}$$

$$\underline{\texttt{mul}}\text{tiply }\underline{\texttt{h}}\text{igh}\begin{Bmatrix}\underline{\ }\\ \underline{\texttt{u}}\text{nsigned}\\ \underline{\texttt{s}}\text{igned }\underline{\texttt{u}}\text{nsigned}\end{Bmatrix}$$

$$\begin{Bmatrix}\underline{\texttt{div}}\text{ide}\\ \underline{\texttt{rem}}\text{ainder}\end{Bmatrix}\begin{Bmatrix}\underline{\ }\\ \underline{\texttt{u}}\text{nsigned}\end{Bmatrix}$$