This section describes the available arithmetic **opcodes/mnemonics** and their corresponding operations.

> All arithmetic instructions accept **only a single operand**.
> The **other operand**, as well as the **destination**, is taken from one of the **Link registers**:
> `L0`, `L1`, `L2`, `L3`.

📎 See: Register Reference – Link Registers

🔢 Addition

The following opcodes are used for **addition**:

- `ADDI` — Add Signed Integer
- `ADDU` — Add Unsigned Integer
- `ADDF` — Add Floating Point

??? abstract "ADDI — *Add Signed Integer*"

```
=== "Properties"

    | Property        | Value                       |
    |-----------------|-----------------------------|
    | **Opcode**      | 13                          |
    | **Type**        | Arithmetic                  |
    | **Operand Type**| Signed 64-bit integer       |
    | **Destination** | `L2` (implicit)             |

=== "Algorithm"

    ```
    L2 = L2 + <signed_imm>
    L2 = L2 + <reg_val>
    L2 = L2 + <const>
    ```

=== "Example"

    ```
    ; imm +ve
        ADDI    1
    ; imm -ve
        ADDI    -123
    ; reg val
        ADDI    val(QT)
    ; const
        ADDI    SOME_CONST_VAL
    ```
```

??? abstract "ADDU — *Add Unsigned Integer*"

```
=== "Properties"

    | Property        | Value                       |
    |-----------------|-----------------------------|
    | **Opcode**      | 18                          |
    | **Type**        | Arithmetic                  |
    | **Operand Type**| Unsigned 64-bit value       |
    | **Destination** | `L3` (implicit)             |

=== "Algorithm"

    ```
    L3 = L3 + <unsigned_imm>
    L3 = L3 + <reg_val>
    L3 = L3 + <const>
    ```

=== "Example"

    ```
    ; imm +ve
        ADDU    1
    ; reg val
        ADDU    val(QT)
    ; const
        ADDU    SOME_CONST_VAL

    ```
```

??? abstract "ADDF — *Add Float value*"

```
=== "Properties"

    | Property        | Value                       |
    |-----------------|-----------------------------|
    | **Opcode**      | 23                          |
    | **Type**        | Arithmetic                  |
    | **Operand Type**| 64-bit float value          |
    | **Destination** | `L1` (implicit)             |

=== "Algorithm"

    ```
    L1 = L1 + <float>
    L1 = L1 + <reg_val>
    L1 = L1 + <const>
```
```

```
```

=== "Example"

```
; imm float
    ADDF    3.14
; reg val
    ADDF    val(QT)
; const
    ADDF    SOME_CONST_VAL
```

🔢 Subtraction

The following opcodes are used for **subtraction**:

- SUBI — Sub Signed Integer
- SUBU — Sub Unsigned Integer
- SUBF — Sub Floating Point

??? abstract "SUBI — *Sub Signed Integer*"

=== "Properties"

Property	Value
Opcode	14
Type	Arithmetic
Operand Type	Signed 64-bit integer
Destination	`L2` (implicit)

=== "Algorithm"

```
L2 = L2 - <signed_imm>
L2 = L2 - <reg_val>
L2 = L2 - <const>
```

=== "Example"

```
; imm +ve
    SUBI    1
; imm -ve
    SUBI    -123
```

```
    ; reg val
        SUBI    val(QT)
    ; const
        SUBI    SOME_CONST_VAL
    ```
```

??? abstract "SUBU — *Sub Unsigned Integer*"

```
=== "Properties"

Property	Value
Opcode	19
Type	Arithmetic
Operand Type	Unsigned 64-bit value
Destination	`L3` (implicit)

=== "Algorithm"

    ```
    L3 = L3 - <unsigned_imm>
    L3 = L3 - <reg_val>
    L3 = L3 - <const>
    ```

=== "Example"

    ```
    ; imm +ve
        SUBU    1
    ; reg val
        SUBU    val(QT)
    ; const
        SUBU    SOME_CONST_VAL

    ```
```

??? abstract "SUBF — *Sub Float value*"

```
=== "Properties"

    | Property        | Value                      |
    |-----------------|----------------------------|
    | **Opcode**      | 24                         |
    | **Type**        | Arithmetic                 |
    | **Operand Type**| 64-bit float value         |
    | **Destination** | `L1` (implicit)            |
```
```

```
=== "Algorithm"

    ```
    L1 = L1 - <float>
    L1 = L1 - <reg_val>
    L1 = L1 - <const>
    ```

=== "Example"

    ```
    ; imm float
        SUBF    3.14
    ; reg val
        SUBF    val(QT)
    ; const
        SUBF    SOME_CONST_VAL

    ```
```

🔢 Multiplicaction

The following opcodes are used for **multiplicaction**:

- `MULI` — MUL Signed Integer
- `MULU` — MUL Unsigned Integer
- `MULF` — MUL Floating Point

??? abstract "MULI — *MUL Signed Integer*"

```
=== "Properties"

    | Property        | Value                       |
    |-----------------|-----------------------------|
    | **Opcode**      | 15                         |
    | **Type**        | Arithmetic                  |
    | **Operand Type**| Signed 64-bit integer      |
    | **Destination** | `L2` (implicit)            |

=== "Algorithm"

    ```
    L2 = L2 * <signed_imm>
    L2 = L2 * <reg_val>
    L2 = L2 * <const>
    ```

=== "Example"
```

```
    ; imm +ve
        MULI    1
    ; imm -ve
        MULI    -123
    ; reg val
        MULI    val(QT)
    ; const
        MULI    SOME_CONST_VAL
    ```
```

??? abstract "MULU — *Mul Unsigned Integer*"

```
=== "Properties"

Property	Value
Opcode	20
Type	Arithmetic
Operand Type	Unsigned 64-bit value
Destination	`L3` (implicit)

=== "Algorithm"

    ```
    L3 = L3 * <unsigned_imm>
    L3 = L3 * <reg_val>
    L3 = L3 * <const>
    ```

=== "Example"

    ```
    ; imm +ve
        MULU    1
    ; reg val
        MULU    val(QT)
    ; const
        MULU    SOME_CONST_VAL

    ```
```

??? abstract "MULF — *Mul Float value*"

```
=== "Properties"

    | Property        | Value                       |
```
```

```
    |-----------------|---------------------------|
    | **Opcode**      | 25                        |
    | **Type**        | Arithmetic                |
    | **Operand Type**| 64-bit float value        |
    | **Destination** | `L1` (implicit)           |

=== "Algorithm"

    ```
    L1 = L1 * <float>
    L1 = L1 * <reg_val>
    L1 = L1 * <const>
    ```

=== "Example"

    ```
    ; imm float
        MULF    3.14
    ; reg val
        MULF    val(QT)
    ; const
        MULF    SOME_CONST_VAL

    ```
```

DIVI	
DIVU	
DIVF	
MODI	
MODU	