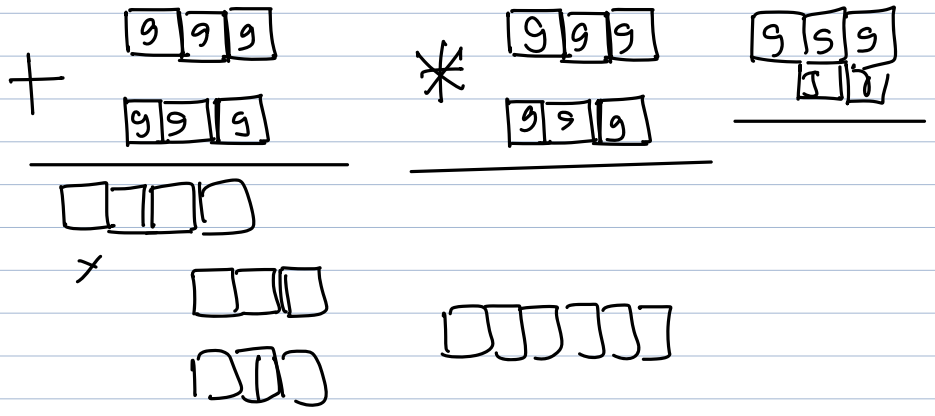
mul src, dest # dest ← dest * src

add src, dest # dest ← dest + src

sub src, dest # dest ← dest − src

div src, dest → q, r



+ 8 bit – n1
  8 bit – n2
  max – 9 bit

* 8 bit – n1
  8 bit – n2
  max – 16 bit

+ 16 bit – n1
  16 bit – n2
  17 bit

* 16 bit – n1
  16 bit – n2
  32 bit

+ 32 bit – n1
  32 bit – n2
  33 bit

* 32 bit – n1
  32 bit – n2
  64 bit

mul src, dest X

div src, dest X

---

5 * ? =

div number	register eax
mul number	eax = Extended accumulator register
	eax * // implied operand

---

n1 * n2.

R0 ← n1

mul n2

R0 * n2

n1 * n2

R0 ← n2

mul n1

n1 / n2

eax ← Dividend

div divisor

n1 / n2

r0 ← n1

div n2

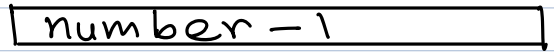
r0 ← n2

div n1

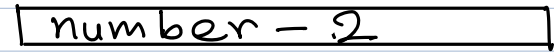
number-1 * number-2.

Step-I : Mov either number-1 or number2 in r_0

Step-II : Give the other number to mul.

1) 

r_0 2) mul number_2

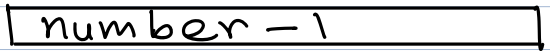
1) 

r_0 2) mul number_1

---

number-1 / number-2. | Dividend = number-1
Divisor = number 2

1) $r_0 \leftarrow$ number-1 2) div number-2

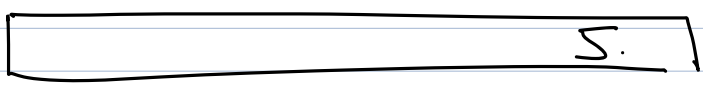


r_0

Result:

1) Multiplication can produce result upto 64-bits.

∴ whatever is the result of multiplication, it is converted into 64-bit number.



3. n2 *

60 0's 1111

8 byte.

multiplication stored in 8 byte

(8 byte [illegible]
[illegible]
[illegible] 8 byte
[illegible] 64 [illegible]
[illegible],
[illegible] zero taka)

—— 64 bits ——

32-bit upper.
32-bit lower

$r_1 \leftarrow$ upper 32 $r_0 \leftarrow$ lower.

bits of result.

32 bits of result

---

mull multiplicand.

↓

constant / register / memory

✓ ✓ ✗

mull constant

mull register.

---

divl divisor

↓

constant / register / memory

✓ ✓ ✗

divl constant

divl register

In case of divison.

Step-1: load dividend (= number to be divided) in r_0.

$r_0 \leftarrow$ dividend.

Step II: Write divl instruction and write divisor as an operand to divl instruction. The operand must be constant or register (but not memory).

divl divisor (as constant or in register).

Result: after divl instruction

r_0 == Quotient

r_1 == Remander.