Data Movement Instruction:

The mov instruction.

Opcode Mneumonic: mov

Suffix: b for 1 byte data movement, w for 2 bytes data movement, l for 4 bytes data movement.

Possible Opcode Mneumonics: movb, movw, movl

General format: mov(b/w/l) src8/src16/src32, dest8/dest16/dest32

Operation: destination <- source

Arithmetic Instructions:

1) The add instruction

Opcode Mneumonic: add

Suffix: b for 1 byte addition, w for two bytes addition, l for 4 bytes addition

Possible Mneumonics: addb, addw, addl

General Format: add(b/w/l) src8/src16/src32, dest8/dest16/dest32

Operation: destination <- destination + source

2) The sub instruction

Opcode Mneumonic: sub

Suffix: b for 1 byte subtraction, w for two bytes subtraction, l for 4 bytes subtraction

Possible Mneumonics: subb, subw, subl

General Format: sub(b/w/l) src8/src16/src32, dest8/dest16/dest32

Operation: destination <- destination - source

3) The mul instruction

Opcode Mneumonic: mul

Suffix: l for 4 bytes multiplication (under MSTC, we are considering multiplication of 4 bytes numbers only. Although its not very difficult to incorporate multiplication of 1 byte and two bytes numbers. But I have decided to cover that topic directly in the context of Intel microprocessor's assembly language. Same can be said about division)

Possible Instruction Mneumonics: mull

General Format: mull constant 32/register 32

Operation: r0 <- LOWER_32_BITS_OF(constant 32 / register 32 * r0)
r1 <- UPPER_32_BITS_OF(constant 32 / registger 32 * r0)

Special Notes: Although multiplication is a binary operator, the mul instruction accepts only one operand. That is because the second operand is implied operand r0. Therefore, multiplication is taken of number stored in r0 and the operand to mul instruction and stored in two 32 bits registers viz. r0 and r1. Out of which r0 stores lower 32 bits of the multiplication and r1 stores the upper 32 bits of the result.

4) The div instruction:

Opcode Menumonic: div

Suffix: l for four bytes division.

Possible Mneumonics: divl

General Format: divl constant32/register32

Operation: r0 <- QUOTIENT_OF(r0 divided by constant32 / register32)
r1 <- REMAINDER_OF(r0 divided by constant32 / register32)

Special Notes: Although Division is a binary operator, the div instruction accepts only one operand. That is because the second operator is implied which register 0 or r0.

The DIVIDEND number in the division is expected in r0

The DIVISOR number in the division is expected as operand to div instruction.

After the execution of the div instruction,
The QUOTIENT OF THE DIVISION is stored in r0
The REMAINDER OF THE DIVISION is stored in r1

Examples:

Problem: Allocate three integers, each four bytes long. Set first integer to 100, second to 200 and the third to the addition of numbers stored in the first two.

STAGE 1: Pseucode level:

Let the name of first integer be num1, that of second be num2 and that of third be summation.

S1: num1 <- 100
S2: num2 <- 200
S3: r0 <- num1
S4: r1 <- num2
S5: r0 <- r0 + r1
S6: summation <- r0

STAGE 2: Using assembly syntax

```
.section .data
    num1:
    .int     0

    num2:
    .int     0

    summation:
    .int     0

.section .text
    movl  $100, num1
    movl  $200, num2
    movl  num1, %r0
    movl  num2, %r1
    addl  %r1, %r0
    movl  %r0, summation
```

Program 2: Reserve storage for three integers, each 4 bytes long. Set 1523899 in the first integer and 845671 in the second. Subtract the number stored in the second integer location from the number first integer location and store the result in the third integer location,

Solution Stage 1 : Pseudo Code Level

Let the name of first integer be num1, that of second be num2 and that of third be subtraction.

S1: num1 <- 1523899
S2: num2 <- 845671
S3: r1 <- num1
S4: r0 <- num2
S5: r0 <- r0 - r1
S6: subtraction <- r0

Stage 2: Assembly code

```
.section .data

    num1:
    .int     0

    num2:
    .int     0

    subtraction:
    .int     0

.section .text
    movl  $1523899, num1
    movl  $845671, num2
    movl  num1, %r1
    movl  num2, %r0
    subl  %r1, %r0
    movl  %r0, subtraction
```