```
# Leggere due numeri naturali, a e b, da input
# Controllare che siano tra 0 e 9, e a >= b
# Calcolare e stampare in output il coeff. binomiale ( a  b ) = a! / ( b! * (a - b)!
)

# a, b => 8 bit
# a!, b!, (a - b)! => 32 bit
# denom. =>
        # (a-b) in N
        # a! >= denom.
        # => denom. su 32 bit

# divisore 32 bit => dividendo 64 bit

#   a_fatt = fattoriale(a)
#   b_fatt = fattoriale(b)
#   ab_fatt = fattoriale(a - b)
#   denom = b_fatt * ab_fatt
#   risultato = a_fatt / denom

.GLOBAL _main
.INCLUDE "C:/amb_GAS/utility"

.DATA
a:          .BYTE 0
b:          .BYTE 0
a_fatt:     .LONG 0
b_fatt:     .LONG 0
ab_fatt:    .LONG 0
denom:      .LONG 0
risultato:  .LONG 0

msg_1: .ASCII "Inserire i due naturali a e b, da 0 a 9:\r"
msg_2: .ASCII "Il coefficiente binomale (a b) e':\r"
msg_err: .ASCII "Input invalidi\r"

.TEXT

_main:
            NOP

            # lettura a e b
            LEA msg_1, %EBX
            MOV $80, %ECX
            CALL outline

            CALL indecimal_byte
            CALL newline
            MOV %AL, a

            CALL indecimal_byte
            CALL newline
            MOV %AL, b

            # controllo validita' a e b
            CMPB $9, a
            JA wrong_input
            CMPB $9, b
            JA wrong_input
```

- INPUT IN 8 BIT
- FATTORIALI IN 32BIT (QUINDI ANCHE NUMERATORE)
- DENOMINATORE IN 32BIT

STESSA STRATEGIA DEL PROBLEMA PRECEDENTE (GUARDO UN SOLO REGISTRO)

* - SO CHE IL NUM. STA IN 32 bit
- SO CHE IL NUM È MAGGIORE O UGUALE AL DENOMINATORE
- SEGUE CHE IL DENOMINATORE STA IN 32 BIT =

INPUT

SALVO I DATI CALCOLATI NEL TEMPO

OUTPUT

STAMPO MSG_1

CHIEDO a

CHIEDO b

CONTROLLO CHE LE CONDIZIONI RICHIESTE SIANO RISPETTATE

```
60          MOVB a, %AL
61          MOVB b, %BL
62          CMP %AL, %BL
63          JA wrong_input
64
65          # calcolo dei fattoriali
66          MOV $0, %ECX
67          MOVB a, %CL
68          CALL factorial_inc
69          MOV %EAX, a_fatt
70
71          MOV $0, %ECX
72          MOVB b, %CL
73          CALL factorial_inc
74          MOV %EAX, b_fatt
75
76          MOV $0, %ECX
77          MOVB a, %CL
78          SUB b, %CL # cl -= b => cl = a - b
79          CALL factorial_inc
80          MOV %EAX, ab_fatt
81
82          # calcolo del denominatore
83          MOV b_fatt, %EAX
84          MOV ab_fatt, %EBX
85          MUL %EBX    # edx_eax = eax * ebx
86          MOV %EAX, denom
87
88          # divisione
89          MOV $0, %EDX
90          MOV a_fatt, %EAX
91          MOV denom, %EBX
92          DIV %EBX    # EAX = EDX_EAX / EBX
93          MOV %EAX, risultato
94
95          # stampa del risultato
96          LEA msg_2, %EBX
97          MOV $80, %ECX
98          CALL outline
99
100         MOV risultato, %EAX
101         CALL outdecimal_long
102         RET
103
104 wrong_input:
105         LEA msg_err, %EBX
106         MOV $80, %ECX
107         CALL outline
108         RET
109
110
111
112 # sottoprogramma fattoriale, da 2 a n
113 # input: ECX naturale da 0 a 9
114 # output: EAX fattoriale del numero (1 se invalido)
115 # sporca: EDX, BX
116 factorial_inc:
117             MOV $1, %AX    # fara' da risultato e moltiplicando
118             MOV $0, %DX
119
```

a!

b!

(a-b)!

b!(a-b)!

a! / b!(a-b)!

SE QUALCOSA NON È RISPETTATO FACCIO JUMP.

CALCOLIAMO I FATTORIALI COL CODICE DEL PROBLEMA PRECEDENTE

NUMERATORE: a!

MUL A 32 BIT

ISTRUZIONE DIV

DIV A 32 BIT (DIVISORE A 32 BIT)

STAMPO MSG_2

STAMPO IL RISULTATO

STAMPO MSG_ERR

CI SI FERMA

```
                # controllo validita'
                CMP $2, %ECX
                JB  fine_factorial_inc
                CMP $9, %ECX
                JA  fine_factorial_inc

                MOV $2, %BX

ciclo_factorial_inc:
                # do {

                MUL %BX    # dx_ax = ax * bx

                CMP %BX, %CX
                JE fine_factorial_inc

                INC %BX
                JMP ciclo_factorial_inc  # } while( cl > 1 )

fine_factorial_inc:
                # edx = ?_rh
                # eax = ?_rl
                SHL $16, %EDX    # edx = rh_0
                MOV %AX, %DX    # edx = rh_rl

                MOV %EDX, %EAX  # eax = rh_rl
                RET

```