```
# Leggere numero naturale, sia n, da input
# Controllare che sia tra 0 e 9
# Calcolare e stampare in output il fattoriale n!
#
# Extra: organizzare il codice di calcolo del fattoriale come sottoprogramma

# n! = n * n - 1 * .... * 1

# n = indecimal_byte()
# # if( n > 9 )
    # return;
#
# fattoriale = 1;
#
# for( int i = 2; i <= n; i++)
     # fattoriale = fattoriale * i;
#
# 8 bit: da 0 a 255
# 16 bit: da 0 a 65535
#
# 2       x   1       =   2       (8 x 8 -> 16)
# 3       x   2       =   6       (8 x 8 -> 16)
# 4       x   6       =   24      (8 x 8 -> 16)
# 5       x   24      =   120     (8 x 8 -> 16)
# 6       x   120     =   720     (8 x 8 -> 16)
# 7       x   720     =   5040    (16 x 16 -> 32)
# 8       x   5040    =   40320   (16 x 16 -> 32)
# 9       x   40320   =   362880  (16 x 16 -> 32)
#                                 32 BIT
# for( int i = n; i >= 2; i--)
    # fattoriale = fattoriale * i;
#
# 9       x   1       =   9       (8 x 8 -> 16)
# 8       x   9       =   72      (8 x 8 -> 16)
# 7       x   72      =   504     (8 x 8 -> 16)
# 6       x   504     =   3024    (16 x 16 -> 32)
# 5       x   3024    =   15120   (16 x 16 -> 32)
# 4       x   15120   =   60480   (16 x 16 -> 32)
# 3       x   60480   =   181440  (16 x 16 -> 32)
# 2       x   181440  =   362880  (32 x 32 -> 64)
#                                 32 BIT
# outdecimal(fattoriale);
#

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

.DATA

n:  .BYTE   0
risultato:  .LONG   1

msg_1:          .ASCII "Inserire naturale n da tra 0 e 9:\r"
msg_2:          .ASCII "Il fattoriale di n (n!) e':\r"

.TEXT

_main:          NOP

                LEA msg_1, %EBX
```

```
                MOV $80, %ECX
                CALL outline

                CALL indecimal_byte
                CALL newline
                MOV %AL, n

                MOV $0, %ECX
                MOVB n, %CL

                CALL factorial_inc
                MOV %EAX, risultato

fine:           LEA msg_2, %EBX
                MOV $80, %ECX
                CALL outline

                MOV risultato, %EAX
                CALL outdecimal_long
                RET

# sottoprogramma fattoriale, da n a 2
# input: ECX naturale da 0 a 9
# output: EAX fattoriale del numero (1 se invalido)
# sporca: EDX
factorial_dec:
                MOV $1, %EAX    # fara' da risultato e moltiplicando

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

ciclo_factorial_dec:
                CMP $1, %CL     # while( cl > 1) {
                JE fine_factorial_dec

                MUL %ECX        # edx_eax = eax * ecx
                DEC %CL
                JMP ciclo_factorial_dec   # }

fine_factorial_dec:
                RET


# sottoprogramma fattoriale, da 2 a n
# input: ECX naturale da 0 a 9
# output: EAX fattoriale del numero (1 se invalido)
# sporca: EDX, BX
factorial_inc:
                MOV $1, %AX    # fara' da risultato e moltiplicando
                MOV $0, %DX

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

STAMPA MSG_1

RICHIEDO INPUT

PULISCO (VITALE PER QUANTO FAREMO DOPO)

PONGO L'INPUT E USO PIÙ AVANTI PER CONFRONTO

SOTTOPROGR.

STAMPO MSG_2

STAMPO IL RISULTATO

NON MI SERVE CALCOLARE IL FATTORIALE DI 1.

ESCO SE ECX $\notin [2,9]$

- RIMANGO NEL CICLO FINCHÉ CL NON SARÀ UGUALE AD 1.
- OGNI VOLTA APPLICO LA MUL A 32 BIT. PERCHÉ?
  - OGNI VOLTA IL RISULTATO VA IN EAX
  - NON HO BISOGNO DI INTERPELLARE EDX

STESSI CONFRONTI DI PRIMA

- USO ECX CHE HA LO STESSO VALORE DI CL (ECX È STATO PULITO ALL'INIZIO)

```
121                 MOV $2, %BX
122
123 ciclo_factorial_inc:
124                 # do {
125
126                 MUL %BX     # dx_ax = ax * bx
127
128                 CMP %BX, %CX
129                 JE fine_factorial_inc
130
131                 INC %BX
132                 JMP ciclo_factorial_inc  # } while( cl > 1 )
133
134 fine_factorial_inc:
135                 # edx = ?_rh
136                 # eax = ?_rl
137                 SHL $16, %EDX    # edx = rh_0
138                 MOV %AX, %DX     # edx = rh_rl
139
140                 MOV %EDX, %EAX  # eax = rh_rl
141                 RET
142
```

INIZIALIZZO LA VARIABILE DA INCREMENTARE

APPLICO LA MUL A 16 BIT

CONFRONTO DOPO COME NEL DO-WHILE

- FINO ALL'ULTIMO STEP MI BASTA UN SOLO REGISTRO (AX) PER GESTIRE I RISULTATI PRECEDENTI.
- DX LO INTERPELLO SOLO ALLA FINE, COL RISULTATO FINALE

- EDX É OCCUPATO AL PIÚ NELLE PRIME 16 CIFRE MENO SIGNIFICATIVE
- EFFETTUO SHIFT E LIBERO LE CIFRE MENO SIGNIFICATIVE PER AX
- SPOSTO AX IN DX (DX EQUIVALENTE DELLE PRIME 16 CIFRE MENO SIGNIFICATIVE DI EDX)
- ADESSO HO TUTTO IL RISULTATO IN EDX
- SPOSTO IN EAX (ABBIAMO STABILITO DI PORRE IL RISULTATO LÌ)