

Scheda di Lavoro:

Istruzioni MUL e DIV con 8086

Obiettivi dell'attività:

- Comprendere e utilizzare le istruzioni MUL e DIV (8 e 16 bit).
- Sviluppare competenze pratiche attraverso esercizi guidati e di difficoltà crescente.
- Incoraggiare l'uso di variabili, stringhe e procedure.

1. Introduzione Teorica

MUL (moltiplicazione) e DIV (divisione) sono istruzioni per operazioni aritmetiche su numeri senza segno.

N.B. Entrambe le istruzioni sono MONO-operando: ad esempio per MUL, il primo fattore della moltiplicazione (il moltiplicando) non viene indicato. Entrambe le istruzioni fanno uso IMPLICITO di AL (o AX per le operazioni a 16bit) per memorizzarlo.

- **MUL 8-bit:**

- **Operandi:** il moltiplicando è nel registro AL, il moltiplicatore è fornito come operando.
- **Risultato:** il prodotto è memorizzato in AX.
- **Criticità:** Se il risultato supera la capacità di un registro a 16 bit, il flag CF (Carry Flag) e OF (Overflow Flag) vengono settati a 1, indicando un overflow.

Esempio:

```
MOV AL, 5      ; Moltiplicando
MOV BL, 3      ; Moltiplicatore
MUL BL         ; AX = AL * BL
; CF e OF saranno impostati a 0 perché il prodotto è
; entro i limiti di AX
```

- **MUL 16-bit:**

- **Operandi:** il moltiplicando è nel registro AX, il moltiplicatore è fornito come operando.
- **Risultato:** il prodotto è memorizzato in DX:AX (DX contiene la parte alta, AX quella bassa).
- **Criticità:** Se DX contiene un valore diverso da zero dopo l'operazione, significa che il risultato non può essere rappresentato in un singolo registro.

Esempio:

```
MOV AX, 1234h ; Moltiplicando
MOV BX, 2      ; Moltiplicatore
MUL BX         ; DX:AX = AX * BX
; Se DX = 0, il risultato rientra nei 16 bit di AX
```

- **DIV 8-bit:**

- **Dividendo:** memorizzato in AX.
- **Divisore:** fornito come operando.
- **Quoziente:** memorizzato in AL.
- **Resto:** memorizzato in AH.
- **Criticità:** Un divisore pari a zero provoca un'eccezione (divisione per zero). Inoltre, se il quoziente è troppo grande per essere contenuto in AL, si verifica un errore di overflow.

Esempio:

```
MOV AX, 25     ; Dividendo
MOV BL, 4      ; Divisore
DIV BL         ; AL = AX / BL, AH = resto
; Verificare che BL ≠ 0 prima dell'operazione
```

- **DIV 16-bit:**

- **Dividendo:** memorizzato in DX:AX.
- **Divisore:** fornito come operando.
- **Quoziente:** memorizzato in AX.
- **Resto:** memorizzato in DX.
- **Criticità:** Come per la versione 8-bit, divisioni per zero o quozienti troppo grandi causano errori.

Esempio:

```
MOV DX, 0      ; Parte alta del dividendo
MOV AX, 100    ; Parte bassa del dividendo
MOV CX, 3      ; Divisore
DIV CX         ; AX = DX:AX / CX, DX = resto
; Verificare che CX ≠ 0 e che il quoziente sia contenibile
in AX
```

2. Esempi Guidati

Studiare ed implementare i seguenti sorgenti: fare attenzione ad adattare il codice al template “.exe” mettendo negli opportuni segmenti le istruzioni riportate di seguito. Aggiungere gli I/O mancanti dei dati e delle stringhe per migliorare l’esperienza utente.

Esempio 1: Conversione di un numero da STRINGA ad INTERO

```
MOV SI, OFFSET str ; Carica l'indirizzo della stringa in SI
MOV AL, [SI] ; Carica il primo carattere (decine) in AL
SUB AL, '0' ; Converti il carattere da ASCII a intero
MOV AH, 0 ; Azzera AH per lavorare con un numero 16-bit
MOV BL, 10 ; Moltiplicatore per il posto delle decine
MUL BL ; AX = decina * 10

INC SI ; Passa al carattere successivo (unità)
MOV DL, [SI] ; Carica il secondo carattere in DL
SUB DL, '0' ; Converti da ASCII a intero
ADD AL, DL ; Somma le unità al totale
; AX ora contiene il numero convertito
RET

str DB '42', 0 ; Stringa di input
```

Esempio 2: Stampa di un numero unsigned a due cifre

```
data segment
; add your data here!
num DB 35 ; Numero a due cifre da stampare
ends

stack segment
dw 128 dup(0)
ends

code segment
start: ; Imposta il data segment:
MOV AX, data
MOV DS, AX
; Carica il numero da stampare
XOR AH, AH ; Resetta la parte alta di AX
MOV AL, num ; Carica il valore della variabile num in AL
MOV BL, 10 ; Divisore a 8 bit per separare decine e unità'
DIV BL ; AL -> quoziente (decine), AH -> resto (unità)
```

```

    MOV BL, AH      ; Salva in BL il resto
; Stampa la decina
    ADD AL, '0'     ; Converte la decina in carattere ASCII
    MOV DL, AL      ; Carica il carattere nel registro DL per la stampa
    MOV AH, 02H     ; Funzione DOS per stampare un carattere
    INT 21H        ; Interruzione per eseguire la stampa
; Stampa l'unita'
    MOV AL, BL      ; Carica il resto (unita') in AL
    ADD AL, '0'     ; Converte l'unità in carattere ASCII
    MOV DL, AL      ; Carica il carattere nel registro DL per la stampa
    MOV AH, 02H     ; Funzione DOS per stampare un carattere
    INT 21H        ; Interruzione per eseguire la stampa

    MOV AX, 4c00h ; exit to operating system.
    INT 21H
ends
end start ; set entry point and stop the assembler.

```

Esempio 3: Moltiplicazione e divisione combinata

Problema:

Un autotrasportatore deve calcolare:

- Il costo totale di un viaggio in base al numero di chilometri percorsi e al costo del carburante per chilometro (moltiplicazione).
- Quanti viaggi può effettuare con il budget disponibile (divisione).
- L'importo residuo del budget dopo aver effettuato il massimo numero di viaggi (divisione con resto).

Nota: nel seguente sorgente,

- ★ si fa uso delle direttive standard per assembler MASM/TASM. Vengono qui riportate per offrire un confronto rispetto alle meno diffuse direttive utilizzate in EMU80886.
- ★ viene definita ed utilizzata una procedura che fa uso dello stack per convertire in stringa un numero intero, e quindi stamparlo: approfondite autonomamente oppure usatela com'è. [Qui un breve tutorial sull'uso dello stack](#)
- ★ si lascia per esercizio di completare il programma con gli output di stringa mancanti

```

.MODEL SMALL
.STACK 100H
.DATA
distance      DW 250      ; Distanza per viaggio (km)
cost_per_km   DW 3        ; Costo per chilometro
budget        DW 1000     ; Budget totale disponibile
cost_per_trip DW 0        ; Costo totale per viaggio

```

```

max_trips      DW 0          ; Numero massimo di viaggi
remaining_budget DW 0        ; Residuo del budget

.CODE
MAIN PROC
    MOV AX, @DATA            ; Inizializza il segmento dati
    MOV DS, AX

; Costo per viaggio (cost_per_trip = distance * cost_per_km)
    MOV AX, distance         ; Carica la distanza in AX
    MOV BX, cost_per_km      ; Carica il costo per km in BX
    MUL BX                   ; AX = distance * cost_per_km
    MOV cost_per_trip, AX    ; Salva il risultato in cost_per_trip

; Numero massimo di viaggi (max_trips = budget / cost_per_trip)
    MOV AX, budget           ; Carica il budget in AX
    MOV BX, cost_per_trip    ; Carica il costo per viaggio in BX
    DIV BX                   ; AX = budget / cost_per_trip, DX = resto
    MOV max_trips, AX        ; Salva il numero massimo di viaggi
    MOV remaining_budget, DX ; Salva il budget residuo

; Stampa dei risultati
; Stampa "Budget disponibile:
    MOV AX, budget           ; Carica il costo per viaggio
    CALL PRINT_NUMBER        ; Stampa il numero

; Stampa "Costo per viaggio:"
; ....
; Stampa "Viaggi massimi:"
; ....
; Stampa "Budget residuo:"
; ....
; Fine del programma
    MOV AH, 4CH              ; Termina il programma
    INT 21H

; Procedura per stampare un numero (AX contiene il numero da stampare)
PRINT_NUMBER PROC
    XOR DX, DX               ; Resetta DX
    XOR CX, CX               ; Resetta CX - contatore cifre
    MOV BX, 10               ; Divisore per ottenere le cifre
DIVIDE_LOOP:
    DIV BX                   ; Divide AX per 10
    PUSH DX                  ; Salva il resto sullo stack
    XOR DX, DX               ; Resetta DX per la prossima divisione
    INC CX
    OR AX, AX                ; Controlla se il quoziente e' 0
    JZ PRINT_DIGITS         ; Se si', passa alla stampa
    JMP DIVIDE_LOOP          ; Altrimenti continua a dividere
PRINT_DIGITS:
    POP DX                   ; Recupera la cifra dallo stack
    ADD DL, '0'              ; Converte la cifra in carattere ASCII
    MOV AH, 02H              ; Funzione DOS: stampa carattere
    INT 21H
    LOOP PRINT_DIGITS        ; Se si', continua a stampare

    MOV DX, 0DH
    INT 21H

```

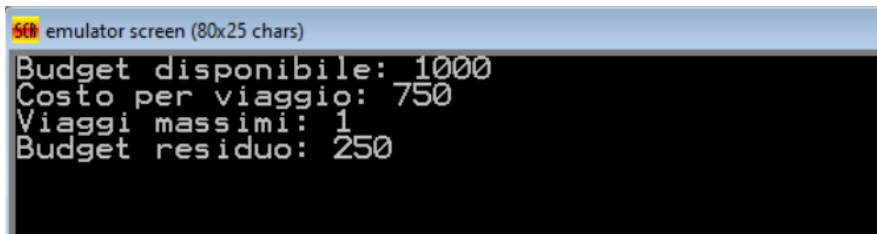
```

    MOV DX, 0AH
    INT 21H
    RET
PRINT_NUMBER ENDP

END MAIN

```

Output atteso:



```

5th emulator screen (80x25 chars)
Budget disponibile: 1000
Costo per viaggio: 750
Viaggi massimi: 1
Budget residuo: 250

```

3. Quesiti di Comprensione e Verifica

1. Quali registri vengono utilizzati implicitamente da MUL e DIV?
2. Quali registri vengono utilizzati per memorizzare il risultato di una moltiplicazione a 8 bit?
3. Come verificare la correttezza di una divisione a 16 bit?
4. Cosa accade se il risultato di una divisione produce un quoziente troppo grande per il registro di destinazione?
5. Come si controllano i flag CF e OF dopo una moltiplicazione? Qual è il loro significato?
6. Perché è importante verificare il valore del divisore prima di eseguire una divisione?
7. Spiegare il comportamento della divisione a 16 bit quando $DX \neq 0$ prima dell'operazione.

4. Errori Comuni e Debugging

Overflow in MUL:

- Sintomo: Flags CF e OF attivi dopo l'operazione.
- Soluzione: Verificare i valori iniziali degli operandi per assicurarsi che il prodotto rientri nella capacità del registro destinazione.

Divisione per zero in DIV:

- Sintomo: Blocco del programma o comportamento indefinito.
- Soluzione: Aggiungere un controllo prima di DIV per verificare che il divisore sia diverso da zero.

Resto inatteso con DIV:

- Sintomo: Il valore di AH o DX non è quello previsto.
- Soluzione: Verificare che il dividendo e il divisore siano corretti e che il risultato sia interpretabile.

5. Esercizi

ES01_CALC_2_0: Implementa una calcolatrice che richieda l'operazione da eseguire e a seguire i due operandi (mono cifra). Calcola il risultato e lo stampa. Il programma deve supportare le 4 operazioni: +, -, x, :

```
Inserire operazione, operando1 e operando2: x69
Il prodotto vale 54

Inserire operazione, operando1 e operando2: :73
Il quoziente e' 2 con resto 1
```

ES02_STR2INT: partendo dall'esempio1, estendere la logica del programma in modo che supporti numeri fino a 255 (ancora meglio sarebbe fino a 65535).

ES02bis_GETINT: partendo dall'esercizio precedente, trasforma il codice nella procedura GETINT che permette di inserire un numero da tastiera e lo ritorna mediante il registro AX. Chiaramente implementa un 'main' che permetta di collaudare la procedura.

ES03_INT2STR: partendo dall'esempio2, estendere la logica del programma in modo che supporti numeri a 8bit e stampi numeri fino a 255 (per i più ardimentosi numeri fino a 65535).

ES03bis_PRINT_INT: Implementare una procedura che stampa a schermo il numero a più cifre passato attraverso AX

ES04_CALC_3_0: Implementare un calcolatrice che supporti quante più operazioni aritmetiche possibili, supportando numeri a più cifre.

ES05_TRANSPORTER: dopo aver completato il codice dall'esempio3, implementare l'input dei dati (sfruttare le procedure definite negli esercizi precedenti)

ES06_SOMMAPRODOTTO: in molti sistemi di calcolo un'operazione molto utilizzata è la sommatoria dei prodotti su due array. Implementa un programma che effettua il prodotto di due array, elemento per elemento (tipo $arr1[i] * arr2[i]$) e ne effettua la somma. I due array sono dichiarati in memoria ed il programma stampa la somma finale.

```
DATA SEGMENT
    array1 DB 1, 3, 5, 7, 9, 11, 13, 15, 2, 4    ; Primo array
    array2 DB 4, 8, 12, 0, 15, 3, 9, 5, 7, 1    ; Secondo array
DATA ENDS
...
```

```
SOMMAPRODOTTO: 466
```