Assembly

Struttura di un programma

```
.MODEL SMALL
.STACK
.DATA
.CODE
<etichetta>:
        MOV AX,@data ; @data è l'indirizzo del segmento dati
        MOV DS,AX

        ;inizio del codice
        <codice>
        ;fine del codice

        MOV AX,4C00H ; funzione DOS Exit
        INT 21H
END <etichetta>
```

Variabili

`<nome variabile> <tipo> <valore>`	Alloca nella RAM la variabile di nome <nome variabile>
`<nome variabile> <tipo> ?`	Alloca nella RAM una variabile non allocata
`<nome variabile> <tipo> <valore 0>,<valore 1>, <valore 2>,[…],<valore n>`	Alloca nella RAM un vettore con i valori
`<nome variabile> <tipo> <numero> DUP(<valore>)`	Alloca nella RAM <numero> di <simboli>

Tipo

`DB` (define byte) – 1 byte
`DW` (define world) – 2 byte
`DD` (define double) – 4 byte
`DQ` (define quad) – 8 byte
`DT` (define ten) – 10 byte

Valore

Numero	`5`	`4`	`10`	`-12`
Carattere ASCII	`'c'`	`'d'`	`'h'`	`'3'`
Stringa	`'stringa$'`	`'s', 't', 'r', 'i', 'n', 'g', 'a', '$'`		
Vettori	`5,4,10,-12`	`'s',1, 'g',10`		

L'Assembly non fa distinzione tra i numeri e i caratteri ASCII 850.

I numeri negativi rappresentati in C2 equivalgono ai caratteri ASCII speciali/estesi.

Le stringe non sono obbligate ad avere il $, ma è consigliato, è utile anche per evitare di avere una variabile "DIM".

Istruzioni e set di istruzioni

`MOV <destinazione>, <partenza>`	Copio il contenuto della locazione di memoria <partenza> in <destinazione>, la destinazione e la partenza devono avere la stessa dimensione

`MOV DL, <carattere da stampare>`
`MOV AH, 02h`
`INT 21h`
	Stampa il carattere <carattere da stampare> carattere nel stdout

`MOV DX, OFFSET <stringa da stampare>`
`MOV AH, 09h`
`INT 21h`
	Stampa la stringa <stringa da stampare> nel stdout, la stringa deve terminare con $

`MOV AH, 08h`
`INT 21h`
`MOV <carattere da leggere>, AL`
	Legge un carattere dallo stdin, viene salvato in <carattere da leggere>

`MOV AH, 01h`
`INT 21h`
`MOV <carattere da leggere>, AL`
	Legge un carattere dallo stdin, lo stampa nello stdout e viene salvato in <carattere da leggere>

Assembly

`INC <numero>`	Incrementa di 1 il numero presente nella variabile <numero> (C/C++ <numero>++)
`DEC <numero>`	Decrementa di 1 il numero presente nella variabile <numero> (C/C++ <numero>--)
`ADD <numero>, <incremento>`	Incrementa <numero> di <incremento> (C/C++ <numero>+=<incremento>) ⚠ Gli operandi devono avere la stessa dimensione e non devono essere entrambe locazioni di memoria
`SUB <numero>, <decremento>`	Decrementa <numero> di <decremento> (C/C++ <numero>-=<decremento>) ⚠ Gli operandi devono avere la stessa dimensione e non devono essere entrambe locazioni di memoria
`NEG <numero>`	Trasforma il <numero> nel suo negato
`MOV <risultato>, <operando 1>` `ADD <risultato>, <operando 2>`	Salva il <risultato> la somma di <operando 1> e <operando 2> (C/C++ <risultato>=<operando 1>+<operando 2>) ⚠ Il risultato e l'<operando 2> non devono essere entrambe locazioni di memoria
`MOV <risultato>, <operando 1>` `SUB <risultato>, <operando 2>`	Salva il <risultato> la differenza tra <operando 1> e <operando 2> (C/C++ <risultato>=<operando 1>-<operando 2>) ⚠ Il risultato e l'<operando 2> non devono essere entrambe locazioni di memoria

`MOV AL, <operando 1>`
`MUL <operando 2>`
`MOV <risultato>, AX`
	Salva in <risultato> il risultato del prodotto tra <operando 1> e <operando 2> (C/C++ <risultato>=<operando 1>*<operando 2>)
	⚠ Il risultato è a 2 byte, mentre gli addendi a 1 byte, il risultato è senza segno

`MOV AX, <operando 1>` `MUL <operando 2>`	Salva nei registro DX:AX il risultato del prodotto tra <operando 1> e <operando 2> (C/C++ DX:AX=<operando 1>*<operando 2>) ⚠ Il risultato è a 4 byte, mentre gli addendi a 2 byte, il risultato è senza segno

`MOV AL, <operando 1>`
`IMUL <operando 2>`
`MOV <risultato>, AX`
	Salva in <risultato> il risultato del prodotto tra <operando 1> e <operando 2> (C/C++ <risultato>=<operando 1>*<operando 2>)
	⚠ Il risultato è a 2 byte, mentre gli addendi a 1 byte, il risultato è con segno, come set di istruzione precedenti precedente è possibile avere la moltiplicazione con segno a 4-2-2 byte.

`MOV AX, <divisore>`
`DIV <dividendo>`
`MOV <risultato>, AL`
`MOV <resto>, AH`
	Salva in <risultato> il risultato della divisione tra <dividendo> e <divisore> (C/C++ <risultato>=<divisore>/<dividendo>)
	Invece salva in <resto> il resto della divisione (C/C++ <risultato>=<divisore>%<dividendo>)
	⚠ Il dividendo è a 2 byte, gli altri a 1 byte, è possibile eseguire la divisione con segno tramite IDIV e a 4-2-2-2 byte, in questo caso il resto è in DX, il risultato in AX.

Assembly

`<dimensione> PTR <identificatore>`	Forza l'assemblatore ad utilizzare la <dimensione> per il determinato <identificatore>
`JMP <identificatore>`	Vai a <identificatore>: senza condizioni
`CMP <elemento 1>, <elemento 2>` `JL <identificatore>`	Vai a <identificatore>: se <elemento 1> è minore di <elemento 2> (C\C++ <elemento 1> < <elemento 2>)
`CMP <elemento 1>, <elemento 2>` `JE <identificatore>`	Vai a <identificatore>: se <elemento 1> è uguale ad <elemento 2> (C\C++ <elemento 1>==<elemento 2>)
`CMP <elemento 1>, <elemento 2>` `JNE <identificatore>`	Vai a <identificatore>: se <elemento 1> è diverso da <elemento 2> (C\C++ <elemento 1> != <elemento 2>)
`CMP <elemento 1>, <elemento 2>` `JG <identificatore>`	Vai a <identificatore>: se <elemento 1> è maggiore di <elemento 2> (C\C++ <elemento 1> > <elemento 2>)
`CMP <elemento 1>, <elemento 2>` `JLE <identificatore>`	Vai a <identificatore>: se <elemento 1> è minore o uguale ad <elemento 2> (C\C++ <elemento 1> <= <elemento 2>)
`CMP <elemento 1>, <elemento 2>` `JLG <identificatore>`	Vai a <identificatore>: se <elemento 1> è maggiore o uguale ad <elemento 2> (C\C++ <elemento 1> <= <elemento 2>)

Indirizzamento

`MOV al, 0f5`		Immediato: utilizza un valore immediato, se il numero è esadecimale ed inizia per una lettera metterci 0 davanti, non è sempre possibile
`MOV al, ah`		Registro: utilizza il valore di un registro
`MOV al, <variabile>`		Diretto/Variabile: utilizza il valore di una <variabile>
`LEA <puntatore>, <variabile>`		Salva su <puntatore> l'offset di una certa <variabile> <puntatore> è quasi sempre il registro bx
`MOV dl, ds:[bx]`	`MOV dl, [bx]`	Indiretto: il registro bx contiene l'offset di una certa variabile, ottenibile per esempio con LEA, ds: è opzionale
`MOV dl, ds:[bx+1]`	`MOV dl, [bx+1]`	Indiretto con spiazzamento: il registro bx contiene l'offset di una certa variabile, a cui si somma una certa costante, ds: è opzionale
`MOV dl, [bx+<variabile>]`		Indiretto indicizzato: il registro bx contiene l'offset di una certa variabile, a cui si somma una certa variabile, ds: è opzionale
`MOV dl, [bx+<variabile>+1]`		Indiretto indicizzato con spiazzamento: il registro bx contiene l'offset di una certa variabile, a cui si somma una certa variabile, a cui si somma una certa costante, ds: è opzionale
`MOV dl, <vettore>[<i>]`		Indiretto indicizzato C/C++: è possibile utilizzare una variante del metodo indiretto per essere utilizzato come nei vettori in C/C++, dove <i> può essere un numero immediato e/o una variabile

Assembly

Esempio di costrutti tipici del C/C++

CMP <condizione 1>, <condizione 2>
JE IF
JNE ELSE
IF:
<codice if>
JMP FINE
ELSE:
<codice else>
JMP FINE
FINE:
	if(<condizione 1>==<condizione 2>)
	{
	  <codice if>
	}
	else
	{
	  <codice else>
	}

CMP <condizione 1>, <condizione 2>
JE FINE
WHILE:
<codice while>
JMP WHILE:
FINE:
	while(<condizione 1>!=<condizione 2>)
	{
	  <codice while>
	}

DOWHILE: <codice do-while> CMP <condizione 1>, <condizione 2> JMP DOWHILE	do{ <codice do-while> }while(<condizione 1>!=<condizione 2>)

MOV AL,0
FOR:
<codice for>
INC AL
CMP AL,<numero>
JL FOR
	for(int i=0;i< <numero>;i++)
	{
	  <codice for>
	}

CMP <condizione 1>, <condizione 2>
JL IF
JE ELSEIF
JG ELSE
IF:
<codice if>
JMP FINE
ELSEIF:
<codice else if>
JMP FINE:
ELSE:
<codice else>
JMP FINE:
FINE:
	if(<condizione 1>< <condizione 2>)
	{
	  <codice if>
	}
	else if(<condizione 1>==<condizione 2>)
	{
	  <codice else if>
	}
	else
	{
	  <codice else>
	}