

Corso di Laurea in

"Ingegneria delle Tecnologie Informatiche" (9 CFU) "Ingegneria Informatica, Elettronica e delle Telecomunicazioni" (6 CFU)

# Architettura dei Calcolatori Elettronici

**Linguaggio Assembly 8086**

**Andrea Prati**

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

**I LINGUAGGI ASSEMBLY**

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Caratteristiche dei linguaggi Assembly

* Sono linguaggi di basso livello
* Vi è corrispondenza uno a uno con le istruzioni del linguaggio macchina
* I simboli mnemonici utilizzati sono associati a
  + Istruzioni
  + sequenze di istruzioni
  + indirizzi di memoria
  + aree di memoria
  + dispositivi di I/O
* Possibilità di utilizzare al meglio la macchina hardware
* La stesura di un programma Assembly è molto complessa
* Possibilità, nei macro-assemblatori, di definire macro-istruzioni
* Possibilità di introdurre nel programma chiamate di libreria

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Statements

* Un programma Assembly è composto di **Statements**. Ogni statement comprende una direttiva per l’Assemblatore e corrisponde ad una riga del programma.
* Se la direttiva corrisponde ad una istruzione macchina

eseguibile

CPU, essa è detta **Istruzione**, altrimenti è una Pseudo-Istruzione.

* Nel seguito verranno quindi analizzate:
  + Istruzioni
    - Etichette
    - Codici Operativi
    - Operandi
  + Pseudo-Istruzioni
  + Macro
  + Commenti

dalla

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Istruzioni

* Vengono tradotte dall’Assemblatore in istruzioni macchina. Ogni istruzione è composta in generale da:
  + una **Etichetta** (o Label)
  + un **Codice Operativo** (o Operation Code)
  + uno o più **Operandi** (o Operands)
* Esempio:

|  |  |  |
| --- | --- | --- |
| Label | OpCode | Operand(s) |
| START: | MOV | AX, BX |
|  | CMP | AX, 12h |
|  | JZ | EQUAL |
|  | INT | 21h |
|  | RET |  |
| EQUAL: | ... |  |
|  | ... |  |

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Etichette

* Sono identificatori associati ad una istruzione; l’assemblatore le sostituisce con l’indirizzo dell’istruzione che rappresentano.
* Offrono i seguenti vantaggi:
  + permettono di trovare più facilmente un punto del programma
  + permettono di non avere a che fare con indirizzi fisici
  + facilitano la modifica del programma
* Esempio:

|  |  |  |
| --- | --- | --- |
| 010C  010E | JLE SUB | (0111) DIGIT1  DL |
| DIGIT1: 0111 | MOV | CL, 2 |
| 0113 | SHL | DL, 1 |

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Codici operativi

* È lo di un’istruzione assembly: in altri termini specifica

mnemonico

l’operazione che deve essere eseguita dalla CPU

* È l’unico campo che non può mai mancare in un’istruzione
* Esempio:

|  |  |  |
| --- | --- | --- |
| START: | MOV | AX, BX |
|  | CMP | AX, 12h |
|  | JZ | EQUAL |
|  | INT | 21h |
|  | RET |  |
| EQUAL: | ... |  |
|  | ... |  |

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Operandi

* Contiene l’indicazione necessaria a reperire gli operandi (uno o più, a seconda dei casi) richiesti dall’istruzione.
* Sulla base di quanto indicato in questo campo, la CPU provvederà, durante l’esecuzione del programma, a reperire gli operandi:
  + nell’istruzione stessa
  + in un registro
  + in memoria
  + su una porta di I/O
* Esempio:

MOV AX, 2 MOV AX, BX

MOV AX, VALORE IN AX, DX

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Pseudo-istruzioni

* Sono comandi utilizzati durante il processo di assemblaggio (dall’Assemblatore o Assembler), che non vengono tradotti in istruzioni macchina eseguibili dalla CPU.
* Esempio:

SECTION DATA

LETTURA\_SN ;Inizio procedura LETTURA\_SN END ;Fine del codice da assemblare

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Macro

* Sono comandi utilizzati per semplificare la stesura di un

programma

complesso in cui c’è la necessità di ripetere più volte determinati segmenti di codice.

* Vengono tradotti in sequenze di istruzioni macchina eseguibili dalla CPU.
* Esempio:

%macro SHIFT\_LEFT\_AX\_4 0

|  |  |  |
| --- | --- | --- |
| SHL | AX, | 1 |
| SHL | AX, | 1 |
| SHL | AX, | 1 |
| SHL | AX, | 1 |

%endmacro

%macro LOAD\_AX\_AND\_MUL\_16 1

MOV AX, %1 SHIFT\_LEFT\_AX\_4

%endmacro

MOV AX, [MEM] SHIFT\_LEFT\_AX\_4 MOV BX, AX

LOAD\_AX\_AND\_MUL\_16 20 MOV BX, AX

...

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Commenti

* Sono parole o frasi inserite dal programmatore per rendere il programma più comprensibile; servono al programmatore stesso e a chi analizzerà in futuro il codice. Vengono ignorati dall’assemblatore, che si limita a visualizzarli quando si richiede il listato del programma. Tutti i caratteri compresi tra un ’;’ e un < CR >, vengono considerati commenti.
* Devono essere utili ed esplicativi; ad esempio:

; programma mal commentato

|  |  |  |
| --- | --- | --- |
| START: | MOV AX, BX | ;Carico AX con il contenuto di BX |
|  | CMP AX, 24 | ;Confronto AX con il valore 24 dec. |
|  | JZ EQUAL | ;Se AX=24 allora salta a EQUAL |
|  | INT 21h | ;Chiama l'INTERRUPT numero 21 hex. |
|  | RET | ;Ritorna alla procedura chiamante |
| EQUAL: | ... |  |

; programma ben commentato

START: MOV AX, BX ;Carico in AX il numero della riga

|  |  |  |
| --- | --- | --- |
| CMP AX, 24 | ;Se sono al termine | dello schermo |
| JZ EQUAL | ; allora non scrivo | nulla |
| INT 21h | ;Scrivi la prossima | riga del testo |
| RET | ;Ritorna |  |
| EQUAL: ... |  |  |

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Vantaggi dei programmi Assembly

* L’utilizzo del linguaggio Assembly anzichè di un linguaggio ad alto livello (tipo C o Pascal) è talvolta giustificato dalla maggiore efficienza del codice; infatti i programmi in Assembly sono tipicamente:
  + più veloci,
  + più corti,
  + ma più complessi

dei programmi scritti in linguaggi ad alto livello.

* La maggior complessità è data dal fatto che anche le più comuni routines devono essere sintetizzate dal programmatore (talvolta per semplificare la programmazione e per aumentare la compatibilità del codice, si utilizzano librerie general purpose, ma sono ovviamente meno efficienti).
* Come esempio si consideri un programma per stampare i numeri pari da 0 a 100:
* Il programma BASIC è:

|  |  |
| --- | --- |
| 100 | I=0 |
| 110 | PRINT I |
| 120 | I=I+2 |
| 130 | IF I<100 GOTO 110 |

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Vantaggi dei programmi Assembly

* Il codice Assembly generato da un compilatore BASIC è il seguente:

I DW ?

L00100: MOV I, 0 L00110: MOV AX, I

CALL STAMPA L00120: MOV AX, I

ADD AX, 2 MOV I, AX

L00130: MOV AX, I

CMP AX, 100 JB L00110

* Si notano almeno due semplici modifiche, che ne migliorano notevolmente le prestazioni:
  + L’uso di registri al posto di locazioni di memoria
  + L’uso di particolari caratteristiche dell’Assembly

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Vantaggi dei programmi Assembly

* Il programma scritto direttamente in Assembly è quindi il seguente:

MOV AX,0 ;Inizializza il valore del contatore CICLO: CALL STAMPA ;Stampa il valore corrente di AX

INC AX ;Calcola il nuovo numero pari a INC AX ; partire dal vecchio AX

CMP AX, 100 ;Se AX non ha raggiunto il valore JB CICLO ; massimo, ritorna a CICLO

* Il programma così ottenuto presenta rispetto a quello prodotto dal compilatore BASIC due fondamentali vantaggi:
  + è più veloce (perchè usa i registri e non locazioni di memoria)
  + è composto da un numero minore di istruzioni e quindi occupa una minore estensione di memoria
* NB: si noti che l’operazione generale di somma (in questo caso +2) è stata tradotta in una sequenza di operazioni elementari ad hoc.
* Per programmi più articolati risulta più evidente la maggiore complessità di sintesi direttamente in Assembly.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Messa a punto del codice

* Facciamo un esempio (dal Tannenbaum):

|  |  |  |
| --- | --- | --- |
|  | Anni/uomo di programmazione richiesti | Tempo d’esecuzione del programma in secondi |
| Linguaggio assemblativo | 50 | 33 |
| Linguaggio di alto livello | 10 | 100 |
| Approccio misto senza messa a punto |  |  |
| 10% più critico | **1** | **90** |
| Restante 90% | 9 | 10 |
| Totale | 10 | 100 |
| Approccio misto con messa a punto |  |  |
| 10% più critico | **6** | **30** |
| Restante 90% | 9 | 10 |
| Totale | 15 | 40 |

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio di procedura Assembly: assemblato e disassemblato

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | | | | | |
| ;\*  ;\*  ;\* | Procedura di attesa di risposta (S/N) via tastiera | | | | \*  \*  \* |
| ;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* | | | | | |
| LETTURA\_SN NUOVA\_LETTURA: | | MOV AH,07h | | ;Inizio procedura LETTURA\_SN  ;Servizio DOS 'Read Keyboard | |
| FINE\_LETTURA: | | INT 21h OR AL,20h  CMP AL,'n'  JZ FINE\_LETTURA CMP AL,'s'  JNZ NUOVA\_LETTURA  RET | | ; Without Echo'  ;Converte in minuscolo  ;Se il tasto premuto e' 'N'  ; esce dalla procedura  ;Se non e' 'S',  ; ne legge un altro  ;Ritorno alla proc. chiamante | |
| 57DA:00A2 | | B407 | MOV AH,07 | | |
| 57DA:00A4 | | CD21 | INT 21 | | |
| 57DA:00A6 | | 0C20 | OR AL,20 | | |
| 57DA:00A8 | | 3C6E | CMP AL,6E | | |
| 57DA:00AA | | 7404 | JZ 00B0 | | |
| 57DA:00AC | | 3C73 | CMP AL,73 | | |
| 57DA:00AE | | 75F2 | JNZ 00A2 | | |
| 57DA:00B0 | | C3 | RET | | |

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

**ARCHITETTURA LOGICA DELLA CPU INTEL 8086**

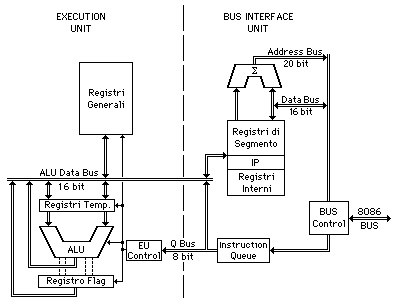
******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## L’ISA di riferimento

* Il processore Intel 8086 è un processore *general purpose* a 16 bit:
  + Capacità di indirizzamento di 1 Mbyte
  + 14 registri interni a 16 bit
  + 7 modi di indirizzamento
  + Alimentazione a 5 volt
  + 48 pin di interconnessione
  + Set di istruzioni esteso (CISC)
* Perché studiare l’8086/8088
  + È stato il primo processore Intel per il PC-IBM.
  + I programmi scritti per questo processore funzionano ancora sui processori moderni.
  + È più semplice, dato che ha meno istruzioni e funzionalità rispetto ai processori moderni.
  + Consente di apprendere tutte le nozioni fondamentali sulla programmazione di basso livello.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Intel 8086



**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## ISA

* In alcuni casi l’ISA è definita da standard (es.: SPARC V9 nel 1994), in altri è proprietaria e non nota (es: Intel)

# Scelte progettuali per l’ISA:

1. dove sono memorizzati gli operandi nella CPU
2. con che istruzioni si accede agli operandi
3. modello di memoria
4. formato delle istruzioni  linguaggio macchina
5. modalità di indirizzamento
6. tipo e struttura degli operandi
7. che tipo di operazioni sono previste

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Dove sono memorizzati gli operandi nella CPU?

* Memorizzazione degli operandi:

MDR

IR

AC

ALU

1. STACK
2. ACCUMULATORE
3. SET DI REGISTRI

* Esempio: C = A+B

**AC <- f(AC, MDR)**

|  |  |  |
| --- | --- | --- |
| **STACK** | **ACCUMULATORE** | **SET DI REGISTRI** |
| PUSH A | LOAD A | LOAD R1,A |
| PUSH B | ADD B | ADD R1,B |
| ADD | STORE C | STORE C,R1 |
| POP C |  |  |

* STACK: difficoltà di accesso, collo di bottiglia. *Vantaggi*: indipendenza dal register set. *Esempi*: Java Virtual Machine; unità floating point dei processori Intel x86
* ACCUMULATORE: gestione più semplice, ma l’accumulatore è il collo di bottiglia
* SET DI REGISTRI: molto generale, tutti gli operandi espliciti, codice più lungo

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Dove sono memorizzati gli operandi nella CPU?

* GPR (**General Purpose Registe**m**r**)acchina a set di registri:

General Purpose

**Register set**

ALU

MDR

Status Reg

Data Addr Contr.

MAR

IR

PC

Control FSM

* Altre scelte progettuali:
  + set di registri si dicono **ortogonali** se possono essere tutti usati indifferentemente l’uno dall’altro nelle istruzioni previste dall’ISA

Address Generator

* + registri della stessa lunghezza o di lunghezze diverse
* famiglia Intel: registri di lunghezza diversa non ortogonali
* famiglia Motorola**:** 8 registri di dato a 32 bit, ortogonali: 8 registri di address a 32 bit
* SPARC, PowerPC: 32 registri a 32 bit

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## La CPU 8086 del punto di vista del programmatore

* La parte fondamentale della CPU, per il programmatore, sono certamente i registri.
* Tre categorie: General Purpose Registers, Segment Registers, Miscellaneous Registers.
* I General Purpose Registers (registri a scopo generico) hanno in realtà uno scopo ben preciso insito nel loro nome, anche se molte istruzioni consentono di utilizzare i registri generici indipendentemente dal loro scopo primario.
* I registri sono AX, BX, CX, DX, SI, DI, BP, SP.
* Questi sono registri a 16 bit e per i primi 4 è possibile accedere direttamente agli 8 bit più significativi (parte alta/high) o agli 8 bit meno significativi (parte bassa/low) specificando H o L invece che X. Cioè, per esempio, CL è il registro a 8 bit corrispondente alla parte bassa di CX, mentre AH è la parte alta di AX.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## I registri generici

* Vediamo nel dettaglio lo scopo primario di ognuno di questi registri:
  + AX è l’accumulatore: serve per numerose operazioni matematiche o per speciali trasferimenti di dati.
  + BX è il registro base: serve per contenere l’indirizzo di partenza durante gli indirizzamenti in memoria.
  + CX è il registro di conteggio: serve per effettuare conteggi durante cicli o ripetizioni.
  + DX è il registro dati: serve per contenere parte di dati eccedenti durante le operazioni aritmetiche e per gli indirizzi delle istruzioni di I/O.
  + SI e DI sono registri indice utilizzati principalmente durante le operazioni con stringhe di byte. Tipicamente SI punta alla sorgente, mentre DI punta alla destinazione.
  + BP è il puntatore base e, in modo molto simile a BX, serve come indirizzo di partenza, tipicamente durante l’accesso a parametri e variabili di funzioni.
  + SP è il puntatore allo stack: l’8086 ha istruzioni per la gestione di questa struttura dati direttamente nella sua architettura e questo registro viene implicitamente referenziato da tutte queste istruzioni.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Registri speciali

* Esistono due ulteriori registri speciali che non vengono modificati direttamente dal programmatore, ma sono fondamentali per l’esecuzione del software:
  + IP è l’instruction pointer, cioè l’indirizzo, all’interno del code segment dal quale prelevare la prossima istruzione.
  + FLAG è un registro a 16 bit nel quale ogni bit ha un significato speciale che indica la modalità di funzionamento del software, lo stato del sistema o il risultato dell’istruzione precedente.
* Le istruzioni possono fare riferimento a tutti i registri generali e di segmento per leggerne o scriverne il valore, con l’eccezione di CS che può solo essere letto.
* Non è possibile fare riferimento a IP o a FLAG, perché questi sono implicitamente o esplicitamente modificati da istruzioni apposite.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Il registro FLAG

15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | OF | DF | IF | TF | SF | ZF |  | AF |  | PF |  | CF |

* Ecco una breve descrizione del significato dei singoli bit:
  + Overflow: Indica che operazione ha riportato un risultato troppo grande
  + Direction: Indica se decrementare o incrementare per le istruzioni con le stringhe
  + Interrupt Enable: Indica se le interruzioni mascherabili sono abilitate
  + Trap: Questo flag è usato dai debugger per eseguire i programmi un passo alla volta. Genera un INT 3 dopo ogni istruzione
  + Sign: Viene posto a 1 se il risultato di una operazione è negativo
  + Zero: Abilitato se il risultato di una operazione è 0
  + Auxiliary Carry: Indica un riporto o un prestito tra la parte bassa e quella alta di un numero. Viene usato dalle istruzioni aritmetico decimale.
  + Parity Flag: Posto a 1 quando c’è un numero pari di bit a 1 nel risultato dell’operazione. Utilizzato dai programmi di comunicazione.
  + Carry Flag: Indica un riporto o un prestito nella parte alta dell’ultimo risultato. Serve per realizzare istruzioni multi word.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Con che istruzioni si accede agli operandi

Con che istruzioni si accede agli operandi nelle macchine a set di registri?

* si possono dividere a seconda del numero di riferimenti diretti in memoria (0:3) indicati nelle istruzioni di ALU e del numero di operandi indicati in modo esplicito (0:3) nelle istruzioni
* Architetture **register-register** o **load-store** se le istruzioni di ALU non hanno riferimenti dirette in memoria (e si accede alla memoria solo con load e store)
* Architetture **register-memory** invece se esistono istruzioni di ALU con cui accedere direttamente alla memoria

|  |  |  |
| --- | --- | --- |
| register/register | (0,3) | Macchine RISC: MIPS, SPARC, Intel i860 |
| register/memory | (1,2) | macchine CISC:  Motorola, Intel, IBM 360 … |
| memory/memory | (3,3) | VAX |

**VAR3= VAR1+VAR2**

macchina register-memory (Intel) (sintassi destinazione,sorgente) mov AX,var1

add AX,var2 mov var3,AX

macchina register-register (Sparc) (sintassi sorgenti,destinazione) ld r1,var1

ld r2,var2 add r1,r2,r3 st var3,r3

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Modelli di memoria: ordinamento

Le istruzioni e i dati sono composti da più byte. L’indirizzamento però si riferisce sempre al byte

L’unità logica del dato è la parola che è composta da n byte. Viene acceduta indirizzando il primo degli n byte. Ad esempio, una parola a 32 bit (4 byte) occupa 4 indirizzi

Ordinamento

1. **in che ordine sono memorizzati**:

– **little endian**

il LSB (Least Significant Byte) all’indirizzo

più basso 

macchine Intel

– **big endian** il MSB

31 Little 0

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

MSB LSB



MEMORIA

31 Big 0

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

MSB LSB

0

1

2

3

4

5

(Most Significant Byte) 6

all’indirizzo

più basso  macchine Motorola

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Modelli di memoria: allineamento

### Indirizzi allineati

* + l’ISA definisce la larghezza di base della parola (es: 32 bit, 4 byte)

### se la parola è memorizzata a partire da un indirizzo non multiplo dei byte della parola, allora si dice che l’indirizzo non è allineato

* + sono necessarie più letture o più scritture per accedere ad un solo dato
  + architetture con allineamento permettono solo indirizzi di partenza allineati alla lunghezza della parola (es: multipli di 4 per ISA a 32 bit)  macchine RISC

31 0 31 0

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
| 5 | 4 | 3 | 1 |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  | 5 | 4 |
| 3 | 1 |  |  |
|  |  |  |  |

8

4

0

**allineato non allineato**

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Rilocazione: memoria lineare

* + EFFECTIVE ADDRESS (EA) è l’indirizzo effettivo che viene utilizzato per accedere alla memoria (ciò che viene scritto sul MAR) e deve tener conto della **rilocazione.**

### 1) modello di memoria lineare

La memoria è organizzata come un lungo vettore di byte da 0000….000 fino al massimo indirizzo possibile. La posizione del codice e dei dati nella memoria viene però deciso dal sistema operativo nel momento in cui codice e dati viene caricato dalla memoria di massa alla memoria centrale.

Il SO carica i dati a partire dall’indirizzo AD e il codice dall’indirizzo AC.

Per eseguire il codice il SO carica il valore AC nel PC, invece l’indirizzo dei dati che si suppone parta dall’indirizzo viene aggiustato in seguito alla

rilocazione sommando il valore AD dal linker E’ compito dell’assemblatore o del

dati

compilatore indicare se l’indirizzo è assoluto

AD (come avviene quando non è previsto il SO nei processori embedded) o se segue la rilocazione

codice

AC

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Rilocazione: modello segmentato

* Alcune ISA (come quella Intel) prevedono un modello segmentato in cui è possibile dividere codice e dati in più segmenti logici di memoria (anche eventualmente sovrapposti) la cui base è indicata in opportuni registri interni
* In questo caso l’EA viene calcolato **a tempo di esecuzione** sommando l’indirizzo indicato in memoria (secondo la modalità di indirizzamento) e gli opportuni registri di segmento.

**Architettura Intel**

* Registri di segmento
  + CS di default per il codice (a cui si somma il PC)
  + SS di default per lo stack (a cui si somma BP o SP)
  + DS di default per i dati (a cui si somma l’indirizzo dati)
  + ES aggiuntivo per dati se indicato esplicitamente
  + (FS aggiuntivo per dati se indicato esplicitamente)
  + (GS aggiuntivo per dati se indicato esplicitamente)

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Segmentazione 8086

* Lo spazio di memoria viene visto come un gruppo di **paragrafi** e

**segmenti:**

* + un paragrafo è una zona di memoria di 16 byte e non possono sovrapporsi
  + Un segmento:
    - è un'unità logica di memoria **indipendente**, indirizzabile separatamente dalle altre unità
    - inizia a un indirizzo di memoria multiplo di 16 (**allineato** ad un paragrafo)
    - è costituito da locazioni contigue di memoria
    - è al massimo di 64k byte
    - I segmenti possono essere sovrapposti
* Ogni segmento è identificabile univocamente dai 16 bit più significativi del suo indirizzo di partenza in memoria:

indirizzo fisico 220h indirizzo segmento 22h

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Calcolo dell'Effective Address

* L’indirizzo fisico di una cella di memoria è espresso da 20 bit (bus degli indirizzi a 20 bit – 1 MB indirizzabile); non è quindi possibile un indirizzamento mediante un solo registro a 16 bit. Esso è infatti ottenuto mediante la somma di due contributi:
  + il *Segment Address*: è l’indirizzo di testa del segmento e viene ottenuto moltiplicando per 16 il numero del segmento.
  + l’*Effective Address (EA)*: è l’indirizzo effettivo all’interno del segmento, calcolato come offset (spostamento) rispetto all’inizio del segmento stesso.
* NB: la moltiplicazione per 16 può essere notevolmente velocizzata da un semplice shift a sinistra di 4 posizioni della rappresentazione binaria del numero:

EADSx16 +adr mov AX,ALFA

* ALFA è indicato direttamente nella istruzione AX<--M[DS:ALFA] Ea=DSx16+ALFA

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## I registri di segmento

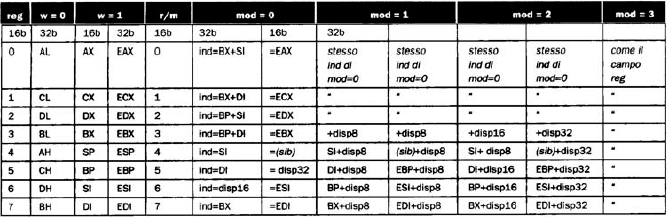
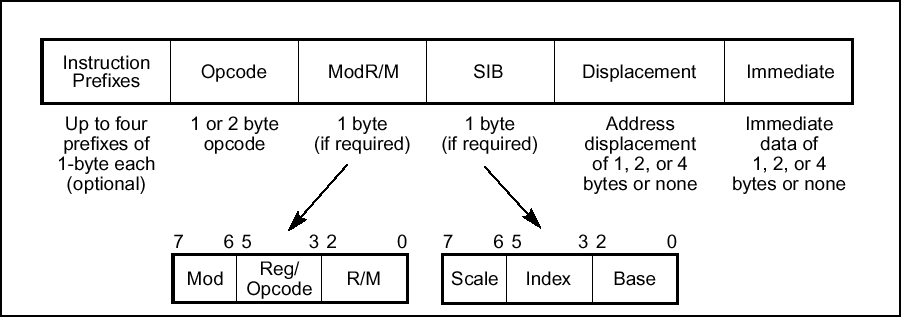
* Esistono quindi 4 registri di segmento che hanno la funzione di suddividere la memoria in parti relative ai dati in esse contenuti:
  + CS è l’indirizzo del segmento codice, cioè l’area di memoria da cui vengono lette le istruzioni da eseguire.
  + DS è l’indirizzo del segmento dati, cioè l’area di memoria che contiene le variabili o le costanti necessarie al programma.
  + ES è l’indirizzo del segmento ausiliario che consente di disporre rapidamente di ulteriore spazio per i dati senza dovere continuamente modificare DS.
  + SS è l’indirizzo del segmento di stack.
* L’utilizzo dei segmenti consente una notevole flessibilità, ma complica notevolmente la progettazione del software, in particolare per le strutture dati di dimensioni superiori ai 64KB.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Formato delle istruzioni

Che formato hanno le istruzioni?

* E’ l’aspetto più critico di un’ISA in quanto può caratterizzare generazioni di processori.
* Le informazioni che devono essere desunte da una istruzione sono: il **codice operativo**, gli **operandi**, il **risultato** (o l’indirizzo degli operandi e del risultato) e l’**indirizzo della prossima istruzione**. Sarebbero necessari troppi bit. Diverse ISA si differenziano da quali campi indicano esplicitamente
* Esempio (Intel 80x86 – dopo 386)



* La decodifica delle macchine Intel è pesante

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Modalità di Indirizzamento

* Indipendentemente dal tipo di rilocazione esistono molti modi per indicare nell’istruzione dove trovare l’operando:
  + REGISTRO: l’operando si trova in un registro
  + IMMEDIATO: l’operando è nella istruzione
  + MEMORIA: se è in memoria, ci sono diversi modi di calcolare l’indirizzo; alcune CPU prevedono molte modalità, maggiore flessibilità e diminuzione del numero delle istruzioni; ma questo implica reti logiche più complesse nella gestione ed un codice operativo con più bit

|  |  |  |
| --- | --- | --- |
| **Modo** | **Esempio** | **Funzionamento** |
| Registro | Add R4,R3 | R4  R4 + R3 |
| Immediato | Add R4, #3 | R4  R4 + 3 |
| Base | Add R4,100(R1) | R4  R4 + M[100+R1] |
| Indiretto(R) | Add R4,(R1) | R4  R4 + M[R1] |
| Indiciato | Add R3,(R1+R2) | R3  R3 + M[R1+R2] |
| Diretto | Add R1,(1001) | R1  R1 + M[1001] |
| Indiretto(M) | Add R1,@(R3) | R1  R1 + M[M[R3]] |
| Autoincr. | Add R1,(R2)+ | R1  R1 + M[R2] |
|  |  | R2  R2 + d |
| Autodecr | Add R1,- (R2) | R2  R2 - d |
|  |  | R1  R1 + M[R2] |
| Scalato | Add R1,100(R2)[R3] | R1  R1 + M[100+R2+R3\*d] |
|  |  | (d = dimensione dell’elemento |

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Modi di indirizzamento nella ISA Intel

* **Operando nei registri:** mov BL,AL
* **Operando immediato:** mov BL,12
* **Operando in memoria:** esistono 17 possibilità per specificare un indirizzo di memoria. Queste possono essere raggruppate in 3 categorie:
  + **Indirizzamento diretto**: si specifica l’indirizzo di memoria tramite un valore numerico detto displacement, cioè spostamento: mov AX,[ALFA]

EA  [DS:ALFA] AXM[EA]

* + **Indirizzamento indiretto tramite registro base**: si specifica l’indirizzo di memoria tramite il valore contenuto in uno tra i registri base BX o BP: mov AX,[BX]

EA  [DS:BX] AXM[EA]

* + **Indirizzamento indiretto tramite registro indice**: si specifica l’indirizzo di memoria tramite il valore contenuto in uno tra i registri indice SI o DI.
* È possibile combinare queste tre modalità ottenendo diverse combinazioni: mov AX,[ALFA+BX+SI]

EA  [DS:(ALFA+BX+SI)] AXM[EA]

* Per ricordare le combinazioni valide è sufficiente memorizzare la tabella seguente (ognuno dei tre elementi può non essere presente):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Disp | + | BX | + | SI |
| BP | DI |

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Modi di indirizzamento nella ISA Intel

* Abbiamo parlato di indirizzi in memoria, ma per l’8086 l’indirizzo fisico è sempre costituito da una coppia segmento:offset.
* Le modalità di indirizzamento viste si riferiscono sempre e solo al calcolo dell’offset, ovvero di un valore a 16 bit dato dalla somma dei tre possibili campi.

### Se nella modalità di indirizzamento compare il registro BP, il segmento di riferimento sarà SS, cioè l’indirizzo è relativo allo stack, altrimenti il riferimento è sempre DS, cioè il segmento dati.

* E’ poi possibile forzare esplicitamente l’utilizzo di un altro segmento tramite il cosiddetto *segment override* ovvero specificando un segmento davanti all’indirizzo seguito da due punti:

mov AX,[CS:BX+5]

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Modi di indirizzamento nel trasferimento di controllo

* **L’indirizzamento del codice è indicato dal program counter PC** (che in famiglia Intel si chiama **IP** – Instruction Pointer)
* In caso di istruzione di salto l’indirizzamento diretto indicato nell’istruzione può essere assoluto o relativo al PC (es: nell’ISA PowerPC viene indicato tramite un bit)
* Si possono utilizzare anche altri tipi di indirizzamento come indiretto tramite registro o indicizzato.
* Inoltre, nell’ISA Intel, posso usare l’indirizzamento relativo al PC:

1. **salti intrasegment diretto**

senza cambiare il CS, ma si ha un solo scostamento a 8 o 16 bit

1. **salti intrasegment indiretto**

senza cambiare il CS, ma con l’indirizzo in un registro con tutti i modi previsti per gli operandi

1. **salti intersegment diretto**

nell’istruzione viene indicato il nuovo CS e il nuovo IP

1. **salti intersegment indiretto**

viene sostituito il CS ed IP con due parole contenute in memoria a cui ci si riferisce con uno dei modi previsti per gli operandi

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Modi di indirizzamento di I/O

* Molte ISA non prevedono istruzioni diverse e spazio di indrizzamento separato per le periferiche di I/O (si parla di **MEMORY MAPPED I/O**)
* In questo caso per leggere o scrivere su periferiche si usano le stesse istruzioni per leggere o scrivere in memoria a particolari indirizzi RISERVATI dal S.O. all’I/O
* altre architetture come l’Intel ha **uno spazio di indirizzamento specifico per l’I/O ed istruzioni specifiche (SEPARATED I/O)**
* 1) **indirizzamento diretto,**

|  |  |  |  |
| --- | --- | --- | --- |
| indirizzo può essere | solo | a | 8 bit nella istruzione (fino a 256 indirizzi |
| diversi)  in AL,PORTA1 |  |  | AL I/O[PORTA1] |
| Out PORTA2, AL |  |  | I/O[PORTA2]AL |

* 2) **indirizzamento CON REGISTRO dx,**

indirizzo è a 16 bit nel registro DX

in AL,DX ALI/O[DX]

Out DX, AL I/O[DX]AL

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Confronto ISA 8086 – Pentium

**ISA 8086**

* Spazio di indrizzamento a 20 bit (1 MBYTE), indirizzabile in modo segmentato con registri di segmento x 16, più i vari modi di indirizzamento
* Spazio di I/O a 16 bit 64 Kbyte indirizzabile con DX
* permette il non allineamento con parole a 8 o 16 bit – ordinamento little endian
* registri non ortogonali a 8 e 16 bit permette l’indirizzamento di una memoria a stack di tipo LIFO (Last In First Out) tramite le istruzioni push e pop ed i registri (impliciti) SS:SP

**ISA Pentium**

* Spazio di indrizzamento a 32 bit (4 GBYTE), indirizzabile in modo segmentato con registri di segmento che sono puntatori a locazioni di memoria in cui è contenuto il vero indirizzo di segmento ed il limite del segmento stesso (se si supera il limite fissato si genera una eccezione). Spesso però si usa il modello lineare mettendo tutti i registri a 0
* Spazio di I/O a 16 bit 64 Kbyte, indirizzabile con DX
* permette il non allineamento con parole a 8 o 16 o 32 bit – ordinamento little endian
* registri non ortogonali a 8, 16 e 32 bit
* permette l’indirizzamento di una memoria a stack di tipo LIFO (Last In First Out) tramite le istruzioni push e pop ed i registri (impliciti) SS:SP
* ha registri per la gestione di SO multitask (ad es: GSR – Global Status Register)

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Tipi e struttura degli operandi

* L’ISA definisce un certo numero di tipi di dato direttamente supportati dalle istruzioni

### tipi di dati numerici:

* *unsigned integer* a 32 bit fino a 232-1
* *signed integer* a 32 bit fino a 231-1 in complemento a 2
* formato floating point a 32, 64 o 128 bit con formato (segno, esponente e mantissa). Alcune ISA seguono lo standard IEEE-754: 3 formati normalizzati (con mantissa da 1 a 2)
* *single precision* 32 bit (1 segno, 8 esponente, 23 mantissa)

da 2-126 a 2128 (circa 10-38)

* *double precision* 64 bit (1 segno, 11 esponente, 52 mantissa)

da 2-1022 a 21024 (circa 10-308)

* *extended precision* 80 bit (1 segno, 15 esponente, 64 mantissa)

### tipi di dati non numerici:

* *caratteri* ASCII a 8 bit e UNICODE a 16 bit
* alcune ISA hanno istruzioni speciali per le *stringhe*
* mappe *Booleane*
* dati *multimediali* a 8 bit

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Tipi di dati nel Pentium Pro

* byte signed integer 8 bit
* word signed integer 16 bit
* doubleword signed integer 32 bit
* byte unsigned integer 8 bit
* word unsigned integer 16 bit
* doubleword unsigned integer 32 bit
* bcd integer 8 bit
* packed bcd integer 4 bit
* near pointer 32 bit
* far pointer (logical address) 48 bit (16 selettore+32 bit)
* floating point 80 bit
* MMX data type 64 bit
  + 8 da 8 bit (pixel)
  + 4 da 16 bit (audio)

– 2 interi da 32 bit o 1 da 64 bit

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

1. **Tipi di operazioni previste**

Esistono diversi tipi di istruzioni:

1. aritmetiche e logiche add, sub,and...
2. di trasferimento dati ld, st, mov
3. di controllo br, jmp, call, ret
4. di sistema chiamate del SO, mem virtuale..
5. floating point fadd, fmul...
6. decimali bcd add, conversioni a caratteri
7. stringhe movs...
8. grafiche-multimediali compress, add…

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

**IL LINGUAGGIO ASSEMBLY 8086**

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Assemblatore

* + Il codice macchina è una sequenza numerica che codifica le istruzioni e gli operandi. Ad esempio, il codice operativo 8A 49 E4 indica di copiare il byte contenuto all’indirizzo di memoria dato da BX+DI-28 nel registro CL.
  + Non è molto comodo programmare calcolando il codice macchina, per questo motivo si utilizza un programma che traduce le istruzioni da una forma testuale più comprensibile al programmatore nel codice macchina.
  + L’istruzione precedente può allora essere scritta come:

mov cl,[bx+di-28]

* + Non esiste purtroppo un accordo ufficiale sulla sintassi che deve essere rispettata da un assemblatore ed esistono diverse varianti per lo stesso codice operativo.
    - Ad esempio l’istruzione precedente, con il Microsoft Macro Assembler (MASM), può essere scritta come mov cl,[bx+di]-28 o mov cl,[bx][di][-28] o mov cl,[di][-28][bx], ecc...

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

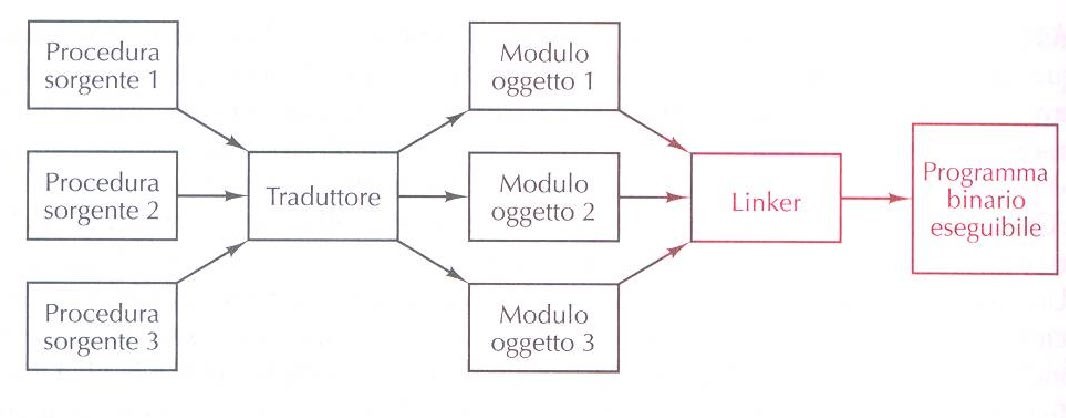
## Assemblatore

* + Durante questo corso utilizzeremo l'assemblatore NASM che può essere utilizzato gratuitamente e del quale sono disponibili i sorgenti in rete.
  + Rispetto a prodotti commerciali quali i prodotti Microsoft e Borland/Inprise questo assemblatore richiede meno dettagli e ci consentirà di concentrarci maggiormente sulla sintassi assembly.
  + Per i dettagli sulla sintassi di questo assemblatore si rimanda alla documentazione e agli esempi che vedremo durante il laboratorio.
  + Per l'installazione fare riferimento alle istruzioni sul sito Elly

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Processo di creazione eseguibile

Dal Tanenbaum



******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Elementi di base del linguaggio

* + Un programma scritto in Assembly 8086 è composto di Statements; normalmente ognuno di essi occupa una riga fino ad un < LF > o una coppia <CR >< LF >.
  + Uno statement può proseguire sulla riga successiva, se questa comincia con il carattere ’&’.
  + L’insieme dei caratteri utilizzabili è composto da
    - caratteri alfanumerici (maiuscole, minuscole, cifre),
    - caratteri non stampabili (spazio, TAB, <CR >, < LF >),
    - caratteri speciali (+ - \* / = ()[] <>;’.,\_:?@$&)
  + All’interno del programma possono comparire:
    - Identificatori
    - Costanti
    - Espressioni

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Identificatori

* + Sono usati come nomi assegnati ad entità definite dal programmatore (segmenti, variabili, label, etc.)
  + Sono composti da lettere, numeri o uno dei tre caratteri @ ? \_, ma non possono iniziare con un numero
  + Hanno lunghezza massima di 31 caratteri

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Costanti

* + Si possono utilizzare costanti:
    - binarie: 001101B
    - ottali: 15O, 15Q
    - esadecimali: 0Dh, 0BEACh (devono iniziare con un numero)
    - decimali: 13, 13D
    - ASCII: ’S’, ’Salve’
    - reali in base 10: 2.345678, 112E-3

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Espressioni

* + Si possono utilizzare i seguenti operatori:
    - Artimetici (+;-;\*;/; MOD, SHL, SHR)
    - Logici (AND, OR, XOR, NOT)
    - Relazionali (EQ, NE, LT, GT, LE, GE)
    - che ritornano un valore ($, SEG, OFFSET, LENGTH, TYPE)
    - Attributi (PTR, DS:, ES:, SS:, CS:, HIGH, LOW)

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Precedenze tra gli operatori

* + Gli operatori visti possono essere elencati in ordine di priorità decrescente nel modo che segue:
    - LENGTH, SIZE, WIDTH, MASK, (), [], <>
    - PTR, OFFSET, SEG, TYPE, THIS, segment override
    - HIGH, LOW

– + (unario), - (unario)

* + - \* , /, MODE, SHL, SHR

– +, -

* + - EQ, NE, LT, LE, GT, GE
    - NOT
    - AND
    - OR, XOR
    - SHORT
  + La priorità può essere modificata tramite l’uso delle parentesi tonde.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Insieme delle istruzioni 8086

* + Possiamo suddividere le istruzioni 8086 in gruppi funzionali:
    - Trasferimento di dati
    - Aritmetica binaria
    - Trasferimento di controllo
    - Logica binaria
    - Shift e Rotate
    - Operazioni su stringhe di dati
    - Istruzioni per il controllo dei flag
    - Aritmetica decimale (Binary Coded Decimal)
    - Varie
  + Di seguito esamineremo ogni gruppo di istruzioni, rimandando per i dettagli sulla sintassi alle guide elettroniche fornite sul sito.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Trasferimento di dati

* + Fanno parte di questo gruppo:
    - MOV *dest,sorg* Sposta il contenuto del secondo operando nel

primo

* + - XCHG *dest,sorg* Scambia il contenuto dei due operandi
    - PUSH *word* Inserisce una word nello stack
    - POP *word* Estrae una word dallo stack
    - IN *accum,porta* Legge un dato dalla porta specificata
    - OUT *porta,accum* Scrive un dato sulla porta specificata
  + L’istruzione MOV è certamente la più usata e semplice da comprendere. XCHG consente di scambiare due registri senza passare per una variabile temporanea.
  + È importante notare che non è possibile muovere o scambiare dati tra memoria e memoria, ma solo tra memoria e registri/valori (si veda lucido successivo per maggiori dettagli)
  + Le istruzioni di IN e OUT consentono di accedere ai registri delle periferiche collegate al bus di sistema.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Trasferimento di dati

* + **Non sono ammessi** i seguenti trasferimenti:
    - *memoria*  *memoria*

Si deve passare attraverso un registro general-purpose: esempio:

MOV AX, SRC MOV DEST, AX

* + - *segment register*  *immediato*

Si deve passare attraverso un registro general-purpose: esempio:

MOV AX, DATA\_SEG MOV DS, AX

* + - *segment register*  *segment register*

Si deve passare attraverso un registro general-purpose (4 cicli): esempio:

MOV AX, ES MOV DS, AX

oppure attraverso lo stack (26 cicli): esempio:

PUSH ES POP DS

* + - Qualsiasi trasferimento che utilizzi CS come destinazione

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Trasferimento di dati

* + Le istruzioni PUSH e POP lavorano con lo stack. Per la precisione, l’istruzione PUSH AX è equivalente a:

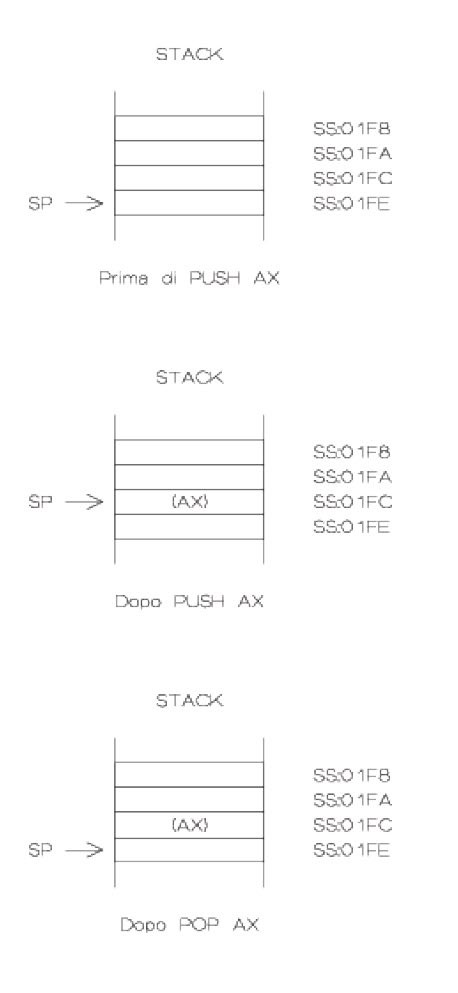
SP = SP - 2 [SS:SP] = AX

* + Quello che può sembrare strano è il fatto che il puntatore allo stack venga decrementato ad ogni inserimento. In realtà questo consente una modalità di programmazione molto compatta (nota anche come Modello di memoria *Tiny*) in cui tutti i segmenti hanno lo stesso valore, i dati seguono immediatamente il codice e lo stack parte dalla fine del segmento:

0 FFFF

|  |  |  |  |
| --- | --- | --- | --- |
| Codice | Dati | … | Stack |

* + Questo modello di memoria ha un chiaro svantaggio, cioè limita il programma e tutti i suoi dati ad un massimo di 64KB, ma elimina la necessità di occuparsi della segmentazione.



******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Trasferimento di dati

**Linguaggio Assembly 8086 58**

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Aritmetica binaria

* + Fanno parte di questo gruppo:
    - ADC *dest,sorg* Add with Carry
    - ADD *dest,sorg* Addition
    - CMP *dest,sorg* Compare
    - DEC *dest* Decrement
    - DIV *sorg* Divide, Unsigned
    - IDIV *sorg* Integer Divide, Signed
    - IMUL *sorg* Integer Multiply, Signed
    - INC *dest* Increment
    - MUL *sorg* Multiply, Unsigned
    - NEG *dest* Negate
    - SBB *dest,sorg* Subtract with Borrow
    - SUB *dest,sorg* Subtract
  + Le operazioni più chiare sono certamente INC e DEC che aumentano o diminuiscono di uno il valore di un registro o di una varibile in memoria.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Aritmetica binaria

* + Esistono poi ADD e SUB che effettuano somma e sottrazione del secondo operando rispetto al primo e salvano il risultato nel primo.
  + CMP è identico a SUB, però non salva il risultato, dato che serve unicamente per confrontare il contenuto di due registri e quindi effettua una differenza per impostare i flag, ma non usa il risultato per nessuno scopo.
  + ADC e SBB sono le versioni di somma e sottrazione che tengono conto di riporto o prestito dell’operazione precedente, utili per effettuare elaborazioni con numeri più grandi di 16 bit.
  + MUL effettua la moltiplicazione di AL o AX con il byte o word passate come parametro e mette il prodotto rispettivamente in AX o in DX:AX (cioè la parte alta in DX e la parte bassa in AX). IMUL è l’equivalente per numeri con segno.
  + DIV divide AX o DX:AX per il byte o word passate come parametro e mette il quoziente e il resto in AL e AH o in AX e DX. IDIV è l’equivalente per numeri con segno.
  + Nel caso che il risultato della DIV sia troppo grande per il registro destinazione o che il divisore sia 0, viene generato un INT 0h (Divisione per zero).
  + La NEG produce il negato del registro passato come parametro.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Formato dei dati nelle istruzioni aritmetiche

* + Il processore 8086 può eseguire operazioni aritmetiche su numeri nei seguenti formati:
    - numeri binari senza segno, su 8 o 16 bit
    - numeri binari con segno, su 8 o 16 bit
    - numeri decimali *packed*, in cui ogni byte contiene due numeri decimali codificati in BCD; la cifra più significativa è allocata nei 4 bit superiori
    - numeri decimali *unpacked*, in cui ogni byte contiene un solo numero decimale BCD nei 4 bit inferiori; i 4 bit superiori devono essere a 0 se il numero è usato in una operazione di moltiplicazione o divisione

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Operazioni su 32 bit

* + Le operazioni di somma e sottrazione possono essere facilmente eseguite anche su operandi di dimensioni superiori a 16 bit, usando le istruzioni
    - **ADC** (ADd with Carry)
    - **SBB** (SuBtract with Borrow)
  + che eseguono rispettivamente le seguenti operazioni:
    - *destination*  *destination+source+Carry*
    - *destination*  *destination – source - Carry*
  + Esempio 1: Per sommare i 32 bit memorizzati in BX:AX con DX:CX, lasciando il risultato in BX:AX,

ADD AX, CX ;Somma i 16 bit meno significativi ADC BX, DX ;Somma i 16 bit piu` significativi

* + Esempio 2: Per sottrarre i 32 bit memorizzati in BX:AX a DX:CX, lasciando il risultato in BX:AX,

SUB AX, CX ;Sottrae le word meno significative SBB BX, DX ;Sottrae le word piu` significative

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Moltiplicazione e Divisione

* + Il processore 8086, a differenza di molti processori ad 8 bit, dispone delle istruzioni di moltiplicazione e divisione.
  + Per entrambe le operazioni esistono forme distinte a seconda che gli operandi siano interi senza segno (MUL o DIV) o interi con segno (IMUL e IDIV).
  + Le due operazioni hanno *un solo* operando, che deve essere un registro generale o una variabile (cioè il contenuto di una locazione di memoria).
  + A seconda delle dimensioni di tale operando (byte o word), si hanno operazioni di tipo byte oppure word. **Se si vuole caricare un dato da memoria o usare un dato in memoria come operando bisogna specificarne la dimensione**. In NASM questo si fa in questo modo:

mov ax,0100h mul **word [0010]**

* + e stessa cosa con **byte** (8 bit) o **dword** (32 bit). Mettendo come sopra word la mul viene considerata a 16 bit quindi l'altro operando è AX, mentre se avessi messo byte considerava come altro operando AL.
  + Nel caso di MOV su registro non è necessario mettere lo specificatore di dimensione perché è automaticamente quella del registro.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Moltiplicazione e Divisione

### Moltiplicazione

* + Si distinguono i due casi:
    - operazioni di tipo byte:

AX  AL \* source-8-bit

* + - operazioni di tipo word: DX:AX  AX \* source-16-bit

### Divisione

* + Si distinguono i due casi:
    - operazioni di tipo byte:

AL  INT( AX / source-8-bit )

AH  resto

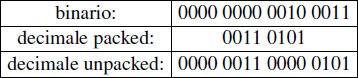
* + - operazioni di tipo word:

AX  INT( DX:AX / source-16-bit ) DX  resto

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Operazioni su Numeri Decimali

* + Il processore 8086 dispone di alcune istruzioni che permettono di eseguire le 4 operazioni fondamentali anche sui numeri decimali.
  + Esse non hanno operandi, in quanto lavorano sempre sul registro AL (AX per la moltiplicazione). Nel caso della divisione l’istruzione di conversione deve essere applicata al dividendo (in AX) prima della divisione.
  + Il risultato della conversione è memorizzato ancora nel registro AL (AX per la moltiplicazione e la divisione).
  + Esempio di diversa rappresentazione:



******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Conversione decimale-binario

* + Le Istruzioni di conversione decimale-binario sono:
    - **AAA**: converte il risultato di un’addizione in decimale unpacked
    - **AAS**: converte il risultato di una sottrazione in decimale unpacked
    - **AAM**: converte il risultato di una moltiplicazione in decimale unpacked
    - **AAD**: converte il dividendo di una divisione da decimale unpacked a binario
    - **DAA**: converte il risultato di un’addizione in decimale packed
    - **DAS**: converte il risultato di una sottrazione in decimale packed

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Trasferimento di controllo

* + Fanno parte di questo gruppo:
    - CALL *ind* Esegue la procedura all’indirizzo *ind*
    - RET Ritorna da una procedura
    - INT *num* Esegue l’interruzione software *num*
    - IRET Ritorna da una interruzione software
    - JMP *ind* Salta all’indirizzo *ind*
    - Jxx *ind* Salta all’indirizzo *ind* se è verificata la condizione xx
    - LOOP *ind* Decrementa CX e se non è zero salta a *ind*
    - LOOPxx *ind* Come LOOP, ma salta solo se è verificata xx
  + L’istruzione più chiara è certamente JMP, che può essere considerato un sostituto per MOV IP,*ind* e che chiarisce il concetto indicando che si “salta” in un’altra zona del programma.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Trasferimento di controllo

* + Le istruzioni di salto condizionato sono: JA (JNBE), JAE (JNB), JB (JNAE), JBE (JNA), JE, JG (JNLE), JGE (JNL), JL (JNGE), JLE (JNG), JNE

A=above (superiore, senza segno), B=below (inferiore , senza segno), E=equal (uguale), G=greater (maggiore, con segno), L=less (minore , con segno), N=not

* + Questi salti fanno riferimento al risultato dell’operazione aritmetica CMP (compare) che esegue una sottrazione ed aggiorna i FLAG in modo opportuno. Quindi queste interpretazioni logiche non sono altro che una combinazione dello stato dei FLAG.
  + Esistono salti che fanno esplicito riferimento ai FLAG: JC, JNC, JNO, JNP (JPO), JNS, JNZ, JO, JP (JPE), JS, JZ

C=carry (riporto), O=overflow, S=sign (segno), Z=zero, P=parity

* + Esiste inoltre JCXZ che salta se CX è uguale a 0.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Trasferimento di controllo

|  |  |  |
| --- | --- | --- |
| **Istruzione** | **Descrizione** | **Salta se ...** |
| JA | Jump if Above | CF = 0 e ZF = 0 |
| JAE | Jump if Above or Equal | CF = 0 |
| JB | Jump if Below | CF = 1 |
| JBE | Jump if Below or Equal | CF = 1 o ZF = 1 |
| JC | Jump if Carry | CF = 1 |
| JE | Jump if Equal | ZF = 1 |
| JG | Jump if Greater | ZF = 0 e SF = OF |
| JGE | Jump if Greater or Equal | SF = OF |
| JL | Jump if Less | SF ≠ OF |
| JLE | Jump if Less or Equal | ZF = 1 o SF ≠ OF |
| JNA | Jump if Not Above | CF = 1 o ZF = 1 |
| JNAE | Jump if Not Above nor Equal | CF = 1 |
| JNB | Jump if Not Below | CF = 0 |
| JNBE | Jump if Not Below nor Equal | CF = 0 e ZF = 0 |
| JNC | Jump if No Carry | CF = 0 |

|  |  |  |
| --- | --- | --- |
| JNE | Jump if Not Equal | ZF = 0 |
| JNG | Jump if Not Greater | ZF = 1 o SF ≠ OF |
| JNGE | Jump if Not Greater nor Equal | SF ≠ OF |
| JNL | Jump if Not Less | SF = OF |
| JNLE | Jump if Not Less nor Equal | ZF = 0 e SF = OF |
| JNO | Jump if No Overflow | OF= 0 |
| JNP | Jump if No Parity (odd) | PF = 0 |
| JNS | Jump if No Sign | SF = 0 |
| JNZ | Jump if Not Zero | ZF = 0 |
| JO | Jump on Overflow | OF =1 |
| JP | Jump on Parity (even) | PF = 1 |
| JPE | Jump if Parity Even | PF = 1 |
| JPO | Jump if Parity Odd | PF = 0 |
| JS | Jump on Sign | SF = 1 |
| JZ | Jump if Zero | ZF = 1 |

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Le istruzioni CALL e RET

* + L’Assembly 8086 permette l’uso delle **Procedure**, in modo simile a quanto avviene nei linguaggi ad alto livello: l’unica differenza sta nell’impossibilità di passare dei parametri nel modo consueto.
  + Mediante una Procedura è possibile scrivere una volta per tutte quelle parti di codice che vengono ripetutamente eseguite in un programma, ottenendo molti vantaggi:
    - Maggiore leggibilità del codice
    - Risparmio di tempo per il programmatore
    - Risparmio di memoria occupata dal codice
    - Possibilità di ricorsione e annidamento (limitate solo dalle dimensioni dello stack)
    - Una Procedura è diversa da una Macro!
  + Le istruzioni CALL e RET permettono di effettuare una chiamata ad una procedura e di ritornare da essa.
  + CALL è la versione di JMP che consente di tornare da dove si era venuti. Per fare questo prima del salto viene eseguito un PUSH IP. Se si salta in un altro segmento viene prima salvato CS, poi IP ed infine si salta. RET ritorna alla posizione precedente.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Le istruzioni CALL e RET

* + Una procedura può essere:
    - **NEAR**: può essere chiamata solo dall’interno dello stesso segmento di codice cui appartiene;
    - **FAR**: può essere chiamata dall’interno di un segmento di codice qualsiasi.
  + Il tipo di procedura (NEAR o FAR) deve essere dichiarato all’atto della creazione della procedura stessa.
  + La **CALL** provvede a:
    - salvare il valore corrente di IP (e di CS nel caso di procedura FAR) nello stack, tramite un’operazione di PUSH;
    - caricare in IP (e in CS) il valore corrispondente all’indirizzo di partenza della procedura chiamata.
  + La **RET** provvede a:
    - ripristinare tramite un’istruzione POP il valore di IP (e di CS nel caso di procedura FAR) salvato nello stack, riprendendo quindi l’esecuzione del programma a partire dell’istruzione successiva all’ultima CALL.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Istruzioni LOOP e INT

* + L’istruzione LOOP consente di eseguire i tipici cicli di una programmazione strutturata. La variabile di controllo è CX e LOOP è equivalente a un decremento di CX, seguito da un salto se CX è diverso da 0.
  + Esistono inoltre LOOPE (LOOPZ) e LOOPNE (LOOPNZ) che combinano il test CX diverso da 0 con il risultato di una operazione di compare precedente.
  + Infine l’istruzione INT consente di invocare le interruzioni software, che per ora possiamo considerare come chiamate a funzione tramite un indice numerato. Successivamente nel corso ne capiremo il reale significato. Per ora basti sapere che INT è una CALL far (cioè inter-segmento) che salva sullo stack anche il registro FLAG.
  + Per questo motivo esiste un ritorno speciale IRET che prima estrae dallo stack IP e CS e poi anche FLAG.

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Interrupt software

* + Come detto, gli interrupt software si invocano tramite l’istruzione INT *n*, dove *n* rappresenta l’interrupt type. Alla maggior parte degli interrupt software sono associate più funzioni, richiamabili tramite un indice contenuto, solitamente, in AH.
  + Gli interrupt software nell’ISA Intel sono raggruppabili in:
    - *Interrupt BIOS*: agiscono a livello di BIOS
    - *Interrupt DOS*: agiscono a livello di sistema operativo
  + Una guida completa agli interrupt software ed in generale all’Assembly 8086 è scaricabile da Internet cercando “Norton Guide”
  + Le funzioni definite a livello di BIOS sono solitamente più complesse, ma permettono anche un livello di parametrizzazione maggiore.

**Linguaggio Assembly 8086**

**73**

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Interrupt software

* + Le operazioni causate da una **INT** sono:
    - salvataggio nello stack del Flag Register
    - azzeramento dei flag TF (Trap Flag) e IF (Int. Enable/Disable)
    - salvataggio nello stack del registro CS
    - caricamento in CS della seconda word dell’Interrupt Vector
    - salvataggio nello stack del registro IP
    - caricamento in IP della prima word dell’Interrupt Vector
  + Le operazioni causate da una **IRET** (senza operandi) sono:
    - preleva dallo stack il valore di IP;
    - preleva dallo stack il valore di CS;
    - preleva dallo stack il valore del Flag Register

**Linguaggio Assembly 8086 74**

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Interrupt software

* + Gli interrupt software più comuni e che in parte vedremo sono i seguenti:
* **INT 10h** Video Services *(BIOS)*
* **INT 16h** Keyboard Services *(BIOS)*
* **INT 17h** Printer Services *(BIOS)*
* **INT 1Ah** Time of Day *(BIOS)*
* **INT 21h** MS-DOS Services *(DOS)*
  + La maggior parte di questi interrupt ha poi una serie di **funzioni** associate a quell’interrupt che si specificano mettendo il numero della funzione nel registro AH prima del commando INT x
  + Alcune di queste funzioni hanno dei parametri di input (da inserire in appositi registri – che vedremo) e/o forniscono un risultato (sempre in uno specifico registro)

**Linguaggio Assembly 8086**

**75**

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Interrupt 10h – Video Services

### INT 10h – funzione 0Eh – Write Character in Teletype (TTY) Mode

* + Altri parametri in input:

AL = codice ASCII del carattere da scrivere BH = numero pagina (solo in modo testo)

BL = colore di foreground (solo in modo grafico)

In modo testo (quello che useremo noi) il carattere viene visualizzato con colore di background e foreground preimpostati (si veda funzione 09h).

### INT 10h – funzione 09h – Write Character and Attribute at Cursor

* + Altri parametri in input:

AL = codice ASCII del carattere da scrivere BH = numero pagina (solo in modo testo) BL = attributo colore

CX = numero di volte che il carattere va scritto

Nota: questa funzione non sposta il cursore. La vediamo solo, eventualmente, per poter cambiare il colore di background e foreground dei caratteri

**Linguaggio Assembly 8086**

**76**

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Interrupt 10h – Video Services

Codifica di colore e attributi dei caratteri in modo testo su console

7 6 5 4 3 2 1 0

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |  |  |  |
|  |  | | | | | | | |

blink background foreground

**Foreground** (colore del carattere)

00H nero 08H grigio

01H blu 09H blu acceso

02H verde 0AH verde acceso 03H azzurro 0BH azzurro acceso 04H rosso 0CH rosso acceso 05H magenta 0DH magenta acceso 06H marrone 0eH giallo

07H bianco 0FH bianco acceso

**Background** (sfondo): solo valori da 00H a 07H



**Blink**: il carattere lampeggia

Es: con valore 29H il risultato è il seguente

**Linguaggio Assembly 8086**

**77**

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Interrupt 16h – Keyboard Services

**INT 16h – funzione 00h – Keyboard Read**

* Valori ritornati:

AL = codice ASCII del carattere letto

AH = scan code (utile differenziare tra caratteri ottenuto con tasti diversi – tipo tastierino numerico)

### INT 16h – funzione 02h – Get Shift Status

* + Valore ritornato:

AL = stato dei tasti speciali codificati come segue:

7 6 5 4 3 2 1 0

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | |  | |  | |  | |  | |  | |  | |
|  |  | |  | |  | |  | |  | |  | |  | |  |
| Insert | | | NumLk | | | | Alt | | | | Shift SX | | | |

CapsLk ScrollLk Ctrl Shift DX

Il valore del bit è a 1 se il tasto speciale è premuto o attivato.

**Linguaggio Assembly 8086**

**78**

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Interrupt 1Ah – Time of Day

### INT 1Ah – funzione 02h – Read Real-Time Clock Time

* + Valori ritornati:

CH = ora (in BCD) CL = minuti (in BCD)

DH = secondi (in BCD)

### INT 1Ah – funzione 04h – Read Real-Time Clock Date

* + Valori ritornati:

CH = secolo (in BCD) – es: 20

CL = anno (in BCD) – es: 24 (quindi 2024 come anno) DH = mese (in BCD)

DL = giorno (in BCD)

**Linguaggio Assembly 8086**

**79**

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Interrupt 21h – funzioni per output

* + Vediamo ora l’interrupt DOS 21h che è di gran lunga il più utilizzato e che ha moltissime funzioni. Le raggruppiamo in funzioni per inviare in **output** e per ricevere in **input**, considerando sia **standard output** (*stdout*, tipicamente il monitor) e **standard input** (*stdin*, tipicamente la tastiera), ma anche i **file**

**INT 21h – funzione 02h – Character Output**

* Altri parametri in input:

DL = codice ASCII del carattere da scrivere

### INT 21h – funzione 06h – Direct Console Character I/O

* + Per output funziona esattamente come la funzione 02h precedente. Inoltre permette anche di fare input da standard input (tastiera) mettendo DL=FFh e il dato letto viene memorizzato in AL. Però non funziona come la funzione 00h dell’INT 16h vista prima (o come quella che vediamo subito dopo) perché **non aspetta la pressione di un tasto**, ma verifica che ci sia un tasto (o dato) presente. Se non c’è lo zero flag ZF viene messo a 1 **(che potrei usare per fare un ciclo finchè non premo tasto).** Utile per la

lettura da standard input diversi (es: ridirezioni con pipe di file)

**Linguaggio Assembly 8086**

**80**

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Interrupt 21h – funzioni per output

### INT 21h – funzione 09h – Print String

* + Altri parametri in input:

DX = indirizzo (offset) della stringa da scrivere a schermo.

La stringa deve essere terminata da $ (poi vedremo che noi tratteremo le stringhe in modo diverso, ma per alcuni casi potremmo utilizzare anche questa funzione dell’INT 21h).

### INT 21h – funzione 40h – Write to File or Device, Using a Handle

* + Altri parametri in input:

BX = file handle (vedremo meglio dopo). BX=1 è stdout CX = numero di byte da scrivere

DX = indirizzo (offset) della stringa da scrivere a schermo.

* + Valori ritornati:

AX = numero di byte scritti

**Linguaggio Assembly 8086**

**81**

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Interrupt 21h – funzioni per input

**INT 21h – funzione 01h – Read Keyboard Character and Echo**

Valori ritornati:

AL = codice ASCII del carattere letto

Scrive anche il carattere letto direttamente sullo standard output (echo).

•

### INT 21h – funzione 3Fh – Read from File or Device, Using a Handle

* + Altri parametri in input:

BX = file handle (vedremo meglio dopo). BX=0 è stdin CX = numero di byte da leggere

DX = indirizzo (offset) della stringa in cui memorizzare i dati

* + Valori ritornati:

AX = numero di byte letti

**INT 21h – funzione 4Ch – Terminate a Process (EXIT)**

* Altri parametri di input:

AL = codice di uscita

**Linguaggio Assembly 8086**

**82**

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 1

Vediamo il primo esempio di programma assembly:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CPU 8086  Leggi: | mov int | ah,00h 16h | ;  ; | Questa è la funzione di lettura di un carattere |
|  | cmp | al,1bh | ; | Il codice ASCII è 1B (ESC)? |
|  | je | Fine | ; | Se sì, vai alla fine |
|  | mov | ah,0eh | ; | Funzione di scrittura a video |
|  | mov | bx,00h | ; | Pagina 0 (BH) |
|  | int | 10h |  |  |
|  | jmp | Leggi | ; | Leggi un altro carattere |
| Fine: | mov int 21h | ax, 4C00h | ; | servizio esci (return code=0) |

La prima riga contiene una direttiva per l’assemblatore che specifica il codice è Assembly per 8086.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 1

* + Nella seconda riga vediamo una delle principali caratteristiche dell’assemblatore, cioè quella di consentire la dichiarazione di *etichette* per indicare la posizione di una istruzione senza conoscere l’indirizzo.
  + La sintassi del NASM prevede che ogni etichetta rappresenti un indirizzo. Quindi può essere usata come valore immediato nelle istruzioni. Nel programma assemblato le etichette verranno sostituite con il loro valore numerico.
  + Che valore verrà sostituito all’etichetta *Leggi*?
  + L’etichetta *Leggi* indica l’indirizzo di una MOV che azzera il registro AH. Poi viene chiamato l’INT 16h.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 1

* + Il programma prosegue con una CMP, per verificare se l’utente ha premuto il carattere ESC. Come già spiegato, CMP esegue una sottrazione tra AL e 1Bh e imposta il flag zero (ZF). Possiamo allora verificare se sono uguali, con un JE. JE è infatti un sinonimo per JZ, che però mette in evidenza (solo a livello concettuale) che la differenza è stata eseguita con lo scopo di verificare una uguaglianza. Nel caso si sia premuto ESC, si salta all’etichetta *Fine*.
  + Il carattere letto viene poi visualizzato tramite la funzione 0Eh dell’interrupt 10h (Video and Screen Services). Infine si salta nuovamente all’etichetta *Leggi*.
  + Nel caso si arrivi a *Fine*, viene eseguita una chiamata all'INT 21h con codice 4C00h, quindi la funzione (AH=4Ch) per uscire a DOS con codice di uscita 0 (AL=00).

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Definizione di dati in Assembler

* + Un dato si definisce con la sintassi

[nome:] tipo espressione [, espressione] ...

* + dove:
    - **nome** - nome simbolico del dato
    - **tipo** - lunghezza del dato (se scalare) o di ogni elemento del dato (se array) - i tipi più utilizzati sono:
      * **DB** riserva uno o più byte (8 bit)
      * **DW** riserva una o più word (16 bit)
      * **DD** riserva una o più doubleword (32 bit)

### espressione - contenuto iniziale del dato:

* + - * un'espressione costante
      * una stringa di caratteri (solo DB)
      * un'espressione che fornisce un indirizzo (solo DW e DD)
      * un'espressione duplicata
  + Se si vuole creare una variabile non inizializzata si usano rispettivamente

**RESB**, **RESW** e **RESD**

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempi di definizione di dati

ByteVar: DB 0 ;1 byte inizializzato a 0 ByteArray: DB 1,2,3,4 ; array di 4 byte String: DB '8','0','8','6' ;array di 4 caratteri String: DB '8086' ;equivale al precedente Titolo: DB 'Titolo',0dh,0ah ;stringa che contiene

;anche una coppia di caratteri CR/LF

Zeros: times 256 DB 0 ;array di 256 byte

inizializzati a 0;

Tabella: RESB 50 ;array di 50 byte non

;inizializzati WordVar: DW 100\*50 ;scalare di una word

|  |  |  |
| --- | --- | --- |
| Matrix: | DW | 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0 ;array |
| NearPointer: | DW | di 16 word;  Matrix ;contiene l'offset di Matrix |
| DoubleVar: | RESD | 1 ;scalare di una doubleword |
| FarPointer: | DD | Matrix ;contiene l'offset e |

;l'indirizzo del segmento di

;Matrix

**NOTA:** per l’ultima assegnazione va detto che a *compile-time* l’indirizzo di

segmento non è nota – solo a *execution-time*

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | **Esempio 2** |
| CPU 8086 |  |  |  |
| SECTION data |  |  |  |
| CaratteriEsa: |  | DB | '0123456789ABCDEF' |
| Numero: | RESB | 1 |  |
| SECTION text |  |  |  |
| ..start: |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Leggi: | mov mov mov int | ax,  ds, | data ax ah,00h 16h | ;  ;  ; | faccio puntare DS alle variabili Questa è la funzione di lettura di un carattere |
|  | cmp je |  | al,1bh Fine | ;  ; | Il codice ASCII è 1B (ESC)? Se sì, vai alla fine |
|  | mov call |  | [Numero],al ScriviNumero |  |  |
|  | mov mov |  | al,' ' ah,0eh | ; | Funzione di scrittura a video |
|  | mov int |  | bx,00h 10h | ; | Pagina 0 (BH) |
| Fine: | jmp ret |  | Leggi | ; | Leggi un altro carattere |

**Ling**(**u**s**a**e**gg**g**io**u**A**e**ss**)**embly 8086 88**

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 2

ScriviNumero:

mov ah,0

mov al,[Numero]

mov bl,16

div bl

mov ah,0

mov si,ax

mov al,[CaratteriEsa+si]

mov ah,0eh

mov bx,00h

int 10h

mov ah,0

mov al,[Numero]

mov bl,16

div bl

xchg ah,al

mov ah,0

mov si,ax

mov al,[CaratteriEsa+si]

mov ah,0eh

mov bx,00h

int 10h

ret

* In questo esempio vediamo come possono essere dichiarate delle variabili in memoria (db), come può essere utilizzata la chiamata a funzione e un primo esempio di indirizzamento indiretto tramite registro indice e scostamento.
* Nonostante la sintassi rigida del NASM non lo consenta, è possibile vedere [CaratteriEsa+si] come CaratteriEsa[si], cioè sommare l’indice è equivalente a usare un indice in un vettore di byte.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Logica binaria

* + Fanno parte di questo gruppo:
    - AND *dest,sorg* AND bit a bit
    - NOT *dest* NOT bit a bit
    - OR *dest,sorg* OR bit a bit
    - TEST *dest,sorg* Test
    - XOR *dest,sorg* OR esclusivo bit a bit
  + AND, NOT, OR e XOR eseguono l’operazione logica indicata sui bit dei registri forniti come parametri. TEST è invece un AND che però non salva il risultato, un po’ come CMP, ma modifica i FLAG. Questa istruzione serve per verificare se un certo bit di un byte o word è a 1.

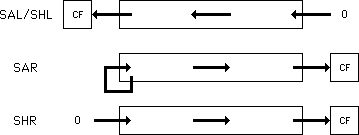
******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

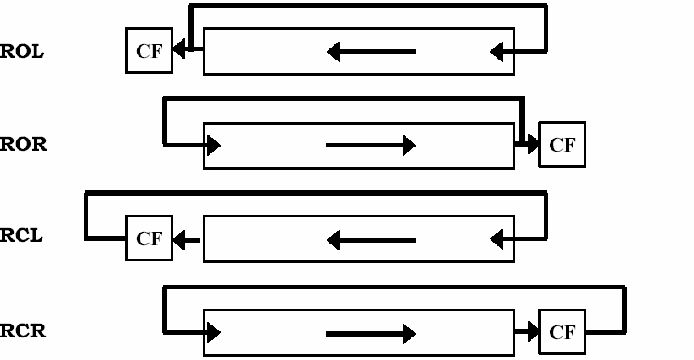
## Shift e Rotate

* + Fanno parte di questo gruppo:
    - SHL *dest,count* Spostamento logico a sinistra
    - SHR *dest,count* Spostamento logico a destra
    - SAL *dest,count* Spostamento aritmetico a sinistra
    - SAR *dest,count* Spostamento aritmetico a destra
    - ROL *dest,count* Rotazione verso sinistra
    - ROR *dest,count* Rotazione verso destra
    - RCL *dest,count* Rotazione attraverso il carry verso sinistra
    - RCR *dest,count* Rotazione attraverso il carry verso destra
  + Gli shift logici spostano tutti i bit in una certa direzione, espellendo gli estremi nel carry e inserendo degli zeri come nuovi bit.
  + SAL è identico a SHL, mentre SAR inserisce bit uguali al bit di segno originale, preservando quindi il segno del numero.
  + I rotate sono equivalenti agli shift, ma reinseriscono il bit “espulso” dalla parte opposta come nuovo bit. Le rotazioni attraverso il carry invece inseriscono il carry.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Shift e Rotate

****

****

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 3

ScriviNumero:

xor bh,bh

mov dl,2

mov dh,[Numero]

Ripeti:

mov cl,4

ror dh,cl

mov bl,dh

and bl,0fh

mov al,[CaratteriEsa+bx] mov ah,0eh

mov bx,00h

int 10h

dec dl

jnz Ripeti ret

* In questo esempio è stata riscritta la procedura ScriviNumero, utilizzando anche le istruzioni logiche e la rotate.
* Nella prima riga, XOR viene utilizzato per azzerare BH.
* Il parametro *count* per gli shift e i rotate può essere soltanto 1 oppure CL. Attenzione perché NASM consente anche altri valori, ma questi funzionano soltanto dal 80386 in avanti.
* Notare anche la mancanza di una CMP prima di JNZ.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Operazioni su stringhe di dati

* + Fanno parte di questo gruppo:
    - CMPS (CMPSB/CMPSW) Confronta stringhe di byte o word
    - MOVS (MOVSB/MOVSW) Copia stringhe di byte o word
    - LODS (LODSB/LODSW) Carica una stringa in AL o AX
    - STOS (STOSB/STOSW) Scrive AL o AX in una stringa
    - SCAS (SCASB/SCASW) Confronta AL o AX con una stringa
  + Prefissi relativi a questo gruppo
    - REP/REPE/REPZ Ripetono l’operazione se CX0 e se ZF=1, poi CX=CX-1
    - REPNE/REPNZ Ripetono l’operazione se CX0 e se ZF=0, poi CX=CX-1
  + Queste operazioni sono istruzioni complesse che consentono di lavorare con stringhe di dati in modo semplice. L’indirizzo della stringa sorgente è sempre DS:SI e quello destinazione è ES:DI.
  + Ognuna di queste operazioni ha una forma con operandi a dimensione variabile e due senza operandi (B per byte e W per word), ma in realtà la prima è solo un modo per consentire più chiarezza nel codice e coincide esattamente con le altre due.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Operazioni su stringhe di dati

* + Tutte le istruzioni stringa incrementano di uno in caso di byte o di due in caso di word SI e DI. Se però il DF (flag direzione) è a 1 la direzione viene invertita e quindi SI e DI vengono decrementati.
    - MOVS(B/W) copia un dato da DS:SI a ES:DI
    - LODS(B/W) carica un dato da DS:SI in AL o AX
    - STOS(B/W) scrive il dato da AL o AX a ES:DI
  + Solitamente per MOVS e STOS si utilizza il prefisso REP che consente di ripetere la copia o la scrittura CX volte. LODS si può ripetere, ma è totalmente inutile, dato che AL/AX viene continuamente sovrascritto.
    - CMPS(B/W) confronta il dato in DS:SI con quello in ES:DI
    - SCAS(B/W) confronta il dato in ES:DI con AL o AX
  + CMPS e SCAS vengono tipicamente ripetuti con REPE/Z o REPNE/Z a seconda che si voglia continuare fino a che il dato è uguale o diverso.

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 4

SECTION data

mov bx,0000h

; Dati del programma

Stringa1: resb 255 Stringa2:db 16,'Prova di stringa'

times 255-17 db ' '

SECTION text

..start: mov ax, data

mov ds, ax

**mov es, ax**

*; Copio Stringa2 in Stringa1*

xor ch,ch

mov cl,[Stringa2]

mov [Stringa1],cl

mov si,Stringa2+1

mov di,Stringa1+1 rep movsb

*; Visualizzo Stringa1*

mov cl,[Stringa1]

mov si,Stringa1+1

mov ah,0eh

Stampa: lodsb

int 10h

loop Stampa

*; Riempio Stringa2 di ‘\*’*

mov al,'\*'

mov cl,100

mov [Stringa2],cl

mov di,Stringa2+1 rep stosb

; Visualizzo Stringa2 mov cl,[Stringa2]

mov si,Stringa2+1

mov ah,0eh

mov bx,0000h Stampa2: lodsb

int 10h

loop Stampa2

Fine: mov ax, 4c00h

int 21h

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 4

* + In questo esempio vengono utilizzate le istruzioni per le stringhe.
  + Per comodità si sono definite le stringhe al modo del PASCAL, ovvero con un byte iniziale che contiene la lunghezza della stringa e uno spazio di 255 byte.
  + Quindi *Stringa1* punta ad una struttura di 256 byte, in cui il primo indica il numero di caratteri effettivamente presenti nel buffer successivo.
  + *Stringa2* è analoga, ma viene inizializzata con un testo.
  + Il programma copia *Stringa2* in *Stringa1*, ne visualizza il contenuto, riempie

*Stringa2* di asterischi e mostra il risultato.

* + Provate a modificare il programma in modo che la parte che esegue la visualizzazione sia una procedura da invocare con una CALL.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Istruzioni per il controllo dei flag

* + Fanno parte di questo gruppo:
    - CLC/STC Clear/Set Carry Flag (pone CF a zero o a uno)
    - CLD/STD Clear/Set Direction Flag (pone DF a zero o a uno)
    - CLI/STI Clear/Set Interrupt-Enable Flag (pone IF a zero o a uno)
    - CMC Complement Carry Flag (inverte lo stato di CF)
    - LAHF Copia FLAG in AH (solo SF, ZF, AF, CF e PF)
    - SAHF Copia AH nei FLAG (solo SF, ZF, AF, CF e PF)
    - POPF Estrae i FLAG dallo stack
    - PUSHF Memorizza i FLAG sullo stack
  + Queste istruzioni sono molto chiare e tipicamente vengono usate CLD e STD prima delle istruzioni stringa, CLC, STC e CMC prima di alcune operazioni aritmetiche su numeri grandi, CLI e STI prima e dopo l’esecuzione di sezioni che lavorano con l’hardware o con le tabelle degli interrupt.
  + Le altre servono tipicamente nelle routine di risposta agli interrupt.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 4 bis

* + Vediamo ora un esempio di accesso alle **stringhe zero-terminate** "alla C"

Scrivere in Assembly per Intel 80x86 un programma che riceve in ingresso una stringa zero-terminata stringa. La stringa è composta da 2 parole di lunghezza qualsiasi (anche diverse tra loro) separate da un solo spazio. Il programma deve scambiare di posto le due parole e scrivere il risultato nella stringa (zero-terminata) mirror. Ad esempio, se la stringa in ingresso vale ‘Grande Pennello’, la stringa risultante dal mirror deve valere ‘Pennello Grande’.

Le variabili del programma sono le seguenti:

stringa: db ‘Buona Pasqua’,0 mirror: resb 100

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 4 bis

CPU 8086

SECTION data

stringa: db 'Buona Pasqua',0 mirror: resb 100

SECTION text

..start:

mov ax, data mov ds, ax mov es, ax

mov si, stringa mov di, mirror

|  |  |  |  |
| --- | --- | --- | --- |
| Ciclo: | lodsb | ; | scorro fino a trovare lo spazio |
|  | cmp al,' ' |  |  |
|  | je Ciclo2 |  |  |
|  | cmp al, 0 |  |  |
|  | jne Ciclo |  |  |
| Ciclo2: | lodsb | ; | copio nella stringa destinazione |
|  | cmp al, 0 |  |  |
|  | je ricomincia |  |  |
|  | stosb |  |  |
|  | jmp Ciclo2 |  |  |

ricomincia: mov al, ' '

stosb ; metto lo spazio in mirror

mov si, stringa ; mi riposiziono all'inizio di stringa

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 4 bis

Ciclo3: lodsb

cmp al, ' ' je Fine stosb

jmp Ciclo3

Fine: mov al, 0

stosb ; metto il terminatore in mirror

mov si, mirror ; mi riposiziono all'inizio di mirror per stamparla mov ah,0eh

mov bx,0000h Stampa: lodsb

cmp al, 0

je fineStampa int 10h

jmp Stampa fineStampa:

mov ax, 4c00h int 21h

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Passaggio di parametri a funzione

* Il passaggio dei parametri alle funzioni, in Assembler, può tipicamente avvenire in 3 modalità:
  + tramite registro
  + tramite variabili in memoria
  + tramite stack
* La prima modalità ha il difetto che, essendo il set di registri limitato, non è efficiente nel caso di più parametri o di chiamata ricorsiva della funzione
* La seconda modalità non è particolarmente indicata nel caso di chiamata ricorsiva della funzione
* La modalità più usata (anche da linguaggi di alto livello come il C) è quella tramite lo stack.
* Prima di chiamare con CALL la funzione metto nello stack i vari parametri (in ordine inverso in C, diretto in Pascal) della funzione
* La funzione chiamata utilizzerà BP per accedere ai parametri (salva quindi il vecchio valore a sua volta nello stack). Per chiamate a funzioni intrasegment, quindi, il primo parametro passato dalla funzione chiamante sarà ad indirizzo SP+4 (+2 per il PUSH IP della CALL e +2 per il PUSH del vecchio valore di BP), il secondo a SP+6, ecc.

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 5

SECTION data

; Definisco i dati globali

Stringa: db 'Hello world, con funzione in stile C.',0dh,0ah,0 SECTION text

..start:

mov ax, data

mov ds, ax

mov ax,Stringa

push ax

call ScriviStringa

**add sp,2**

|  |  |  |
| --- | --- | --- |
| ; Fine | programma |  |
| mov | ah,4ch | ; Function 4Ch (76) - Terminate a Process (EXIT) |
| mov | al,0 | ; Codice di ritorno |
| int  **(SEGUE)** | 21h |  |

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 5

; Scrive una stringa ASCIIZ (classica NULL terminated stile C)

; In input riceve l'indirizzo della stringa come parametro sullo stack. ScriviStringa:

**push bp mov bp,sp**

mov ah,02h ; Function 02h (2) - Character Output mov bx,[bp+4] ; Leggo l'indirizzo della stringa

.AltroCarattere:

mov dl,[bx]

cmp dl,0

je .Fine

int 21h

inc bx

jmp .AltroCarattere

.Fine:

**pop bp**

ret

; ScriviStringa

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 6

mov ax,8 push ax

call CalcolaFattoriale add sp,2

mov ax,4c00h int 21h

; Funzione che calcola il fattoriale di un numero a 16 bit e

; ritorna il risultato in AX.

; int CalcolaFattoriale (int Num); CalcolaFattoriale:

push bp mov bp,sp

; Devo fare Num\*CalcolaFattoriale(Num-1), quindi devo prima ottenere

; il risultato di CalcolaFattoriale(Num-1) mov ax,1

mov bx,[bp+4] dec bx

; Se mi hanno passato un 1 allora adesso ho uno zero e quindi posso

; ritornare 1. jz Fine push bx

call CalcolaFattoriale add sp,2

; Il risultato è in AX, che è proprio il registro usato dalla MUL mul word [bp+4]

; Il risultato va in DX:AX, ma ignoriamo la parte finita in DX. Fine:

pop bp ret

; CalcolaFattoriale

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 7

SECTION data

; Definisco le stringhe

Stringa1: db "Questa è la prima stringa e",0 Stringa2: db " deve essere concatenata a questa",0 Stringa3: times 255 db 0

..start:

mov ax, data mov ds, ax

; Salto all'inizio del programma call \_main

; Fine programma mov ax,4c00h int 21h

; Funzione principale

\_main:

push bp mov bp,sp

mov ax,Stringa2 push ax

mov ax,Stringa1 push ax

mov ax,Stringa3 push ax

call ConcatenaStringhe

**add sp,6**

mov ax,Stringa3 push ax

call ScriviStringa ; da esempio 5 add sp,2

Fine:

pop bp ret

**Linguaggio Assembly 8086**

**(CONTINUA…10)6**

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 7

; Concatena szSorg1 e szSorg2 e le copia in szDest.

; void ConcatenaStringhe (char \*szDest, char \*szSorg1, char \*szSorg2); ConcatenaStringhe:

push bp mov bp,sp

; Copio il contenuto di szSorg1 in szDest

mov di,**[bp+4]** ; Leggo l'indirizzo della stringa di destinazione

mov si,**[bp+6]** ; Leggo l'indirizzo della prima stringa sorgente mov bx,**[bp+8]** ; Leggo l'indirizzo della seconda stringa sorgente

.AltroCarattere: mov al,[si] cmp al,0

je .AltraStringa mov [di],al

inc si

inc di

jmp .AltroCarattere

; Copio il contenuto di szSorg2 in szDest

.AltraStringa: mov al,[bx] mov [di],al inc bx

inc di

cmp al,0

jne .AltraStringa pop bp

ret

; ConcatenaStringhe

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 8

SECTION data Numero DW 134

Stringa resb 6

..start:

mov ax, data mov ds, ax

mov ax, Stringa push ax

mov ax, [Numero] push ax

call ConvertiDecimale add sp, 4

mov ax, Stringa push ax

call ScriviStringa ; come esempi precedenti add sp, 2

; Fine programma mov ax,4c00h

int 21h

**(CONTINUA…)**

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 8

; Converte un numero a 16 bit in decimale senza gli zeri iniziali.

; In input riceve un intero a 16 bit passato sullo stack

; e il puntatore alla stringa che deve ricevere il carattere.

; ConvertiDecimale (int16 iNumero, char \*szStringa); ConvertiDecimale:

push bp mov bp,sp

; Alloco una stringa da 5 caratteri sullo stack

**sub sp,5**

; Uso si come indice nella stringa. xor si,si

mov ax,[bp+4] ; Carico in un registro il numero

mov bx,10 ; Preparo il divisore

; Inizio dividendo per 10 per calcolare la cifra meno significativa

CD\_AltroNumero: dec si

xor dx,dx div bx

|  |  |  |
| --- | --- | --- |
| add | dl,30h | ; converto in codice ASCII |
| mov | [bp+si],dl | ; salvo il carattere |
| cmp | ax,0 |  |
| jne | CD\_AltroNumero |  |

**(CONTINUA…)**

******ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 8

; Copio i caratteri nella stringa destinazione

mov di,[bp+6] ; di è l'indice per la stringa destinazione CD\_AltroCarattere:

mov dl,[bp+si] mov [di],dl inc di

inc si

jnz CD\_AltroCarattere

mov byte [di],00h ; Terminatore per la stringa destinazione mov sp,bp ; Libero la memoria dallo stack

pop bp ret

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Interrupt 21h – lavorare con i file

### INT 21h – funzione 3Ch – Create a File

* Altri parametri in input:

DX = indirizzo (offset) della variabile **ASCIIZ** contenente il nome del file.

CX = attributi del file

* Valori ritornati:

AX = file handle se non ci sono errori (CF=0) o codice dell’errore (se CF=1)

Se il file esiste già viene «troncato» cioè cancellato e ricreato con dimensione 0 byte.

Alcuni codici di errore: 3 (percorso non trovato), 4 (nessun handle disponibile), 5 (accesso negato – es: file in sola lettura).

Per gli attributi del file (nel registro CX) posso scegliere (anche combinandoli): 00h - normale

01h - sola lettura (read-only) 02h - nascosto (hidden)

04h - file di sistema

**Linguaggio Assembly 8086**

**111**

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Interrupt 21h – lavorare con i file

### INT 21h – funzione 3Dh – Open a File

* Altri parametri in input:

DX = indirizzo (offset) della variabile **ASCIIZ** contenente il nome del file.

AL = modalità di apertura

* Valori ritornati:

AX = file handle se non ci sono errori (CF=0) o codice dell’errore (se CF=1)

Alcuni codici di errore (oltre a quelli della slide precedente): 2 (file non trovato), 12 (modalità di apertura non valida).

Per le modalità di apertura usiamo i 3 bit meno significativi di AL (gli altri bit riguardano le modalità di condivisione tra processi che non vedremo):

00h - sola lettura (read-only) 01h - sola scrittura (write-only) 02h - lettura/scrittura

**Linguaggio Assembly 8086**

**112**

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Interrupt 21h – lavorare con i file

### INT 21h – funzione 3Eh – Close a File Handle

* Altri parametri in input:

BX = file handle

* Valori ritornati:

AX = codice dell’errore (se CF=1)

Alcuni codici di errore (oltre a quelli delle slide precedenti): 6 (handle invalido). Già viste le funzioni per leggere (3Fh) e scrivere (40h) da file con l’handle.

### INT 21h – funzione 41h – Delete a File

* Altri parametri in input:

DX = indirizzo (offset) della variabile **ASCIIZ** contenente il nome del file.

* Valori ritornati:

AX = codice dell’errore (se CF=1)

**Linguaggio Assembly 8086**

**113**

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Interrupt 21h – lavorare con i file

### INT 21h – funzione 42h – Move File Pointer (LSEEK)

* Altri parametri in input:

BX = file handle

CX:DX = offset in bytes (intero con segno a 32 bit) AL = modalità

* Valori ritornati:

DX:AX = nuova posizione del puntatore nel file (sempre intero con segno a 32 bit) o in AX codice dell’errore (se CF=1)

Sposta in avanti (se positivo) o indietro (se negativo) il puntatore nel file, secondo questa modalità (in AL):

1. - Sposta il puntatore di CX:DX bytes dall’inizio del file
2. - Sposta il puntatore di CX:DX bytes dall’attuale posizione
3. - Sposta il puntatore di CX:DX bytes dalla fine del file

**Linguaggio Assembly 8086**

**114**

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 9

SECTION data

Filename: db 'fprova',0

Stringa: db 10,'Buongiorno'

N: resb 1

handle: resw 1

SECTION text

..start:

mov ax, data mov ds, ax mov ah, 3ch mov cx, 00h

mov dx, Filename int 21h

jc fine

mov [handle], ax mov ah, 40h

mov dx, Stringa+1 mov bx, [handle] xor ch, ch

mov cl, [Stringa] int 21h

mov [N], ax mov ah, 3eh

mov bx, [handle] int 21h

fine:

mov ax,4c00h int 21h

**Linguaggio Assembly 8086**

**115**

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 10

; Viene dato un file **binario** contenente un array di dati interi

; a 16 bit. Il file contiene come primo byte il numero di

; elementi dell’array e a seguire i vari elementi. Il programma

; deve gli elementi a due a due, **lavorando direttamente sul**

; **file**. Esempio: se l’array contiene i valori [23, 35, -7, 11,

; 24, -89], alla fine deve avere [35, 23, 11, -7, -89, 24].

; Gestire anche il caso di numero di elementi dispari

SECTION data

Filename: db 'filearr',0 N: resb 1

Temp: resw 2

handle: resw 1

SECTION text

..start:

mov ax, data mov ds, ax

; apertura del file in lettura/scrittura mov ah, 3dh

mov al, 02h

mov dx, Filename int 21h

jc fine

mov [handle], ax

**(CONTINUA)**

**Linguaggio Assembly 8086**

**116**

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 10

; lettura della dimensione dell’array mov ah, 3fh

|  |  |  |
| --- | --- | --- |
| mov xor | bx,  ch, | [handle] ch |
| mov | cl, | 1 |
| mov | dx, | N |
| int | 21h |  |

**shr byte [N], 1**

; dimezzo perchè devo fare N/2 scambi. E così risolvo anche il

; problema di N dispari

Ciclo:

mov ah, 3fh

mov bx, [handle] xor ch, ch

**mov cl, 4 ; così leggo due valori alla volta**

mov dx, Temp int 21h

;scambio le due word in Temp mov ax, [Temp]

mov bx, [Temp+2] mov [Temp+2], ax mov [Temp], bx

**(CONTINUA)**

**Linguaggio Assembly 8086**

**117**

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

## Esempio 10

;mi sposto indietro di 4 dalla posizione attuale ... mov ah, 42h

mov bx, [handle] **mov cx, 0ffffh mov dx, -4**

mov al, 1h int 21h

;... e le scrivo nel file mov ah, 40h

mov bx, [handle] xor ch, ch

mov cl, 4 mov dx, Temp int 21h

**dec byte [N]**

jne Ciclo

; chiudo il file mov ah, 3eh

mov bx, [handle] int 21h

fine:

mov ax,4c00h int 21h

**Linguaggio Assembly 8086**

**118**

**ARCHITETTURA DEI CALCOLATORI ELETTRONICI – Prof. Andrea Prati**

**Esempio 11**

* Si crei un file «studenti» contenente un elenco di studenti secondo la seguente formattazione:

Nome:Cognome:Sesso(M/F):Voto(intero 8 bit)

* Si chiedano i dati da tastiera e si memorizzino sul file.
* Si legga poi il file e si calcoli il voto medio delle sole studentesse
* Vedremo una possibile soluzione in laboratorio

**Linguaggio Assembly 8086**

**119**