Introduzione

Questo simulatore fornisce una versione semplificata dell'assemblatore sfruttato dal processore 8088 trasformando il codice inserito in input in codice eseguibile e ne simula l'esecuzione.

In questa guida è possibile trovare una breve descrizione dell'architettura simulata e della sintassi da rispettare nella scrittura del codice.

Architettura

L'architettura del 8088 è un architettura a 16 bit.
Il Bus dati diversamente dal 8086 è a 8 bit.
Il Bus indirizzi è a 20 bit e questo permette di indirizzare ben 2^20 celle di memoria (1024 Kbyte = 1 Mbyte).
In questo simulatore la memoria disponibile è invece di 2^9 ma nonostante questo il calcolo degli indirizzi viene effettuato come nel sistema originale.

RegistroDiSegmento * 16 + Offset

I registri presenti sono:

1. Registri a uso generale
2. Registri puntatori e indice
3. Registri IP e Flag
4. Registri di segmento

Registri a uso generale

AX – registro accumulatore usato come operando implicito per alcune istruzioni.

BX – registro base utilizzato per accedere alla memoria (segmento dati).

CX – registro contatore utilizzato per la scrittura dei costrutti iterativi.

DX – registro dati usato in copia ad AX per operazioni a 32 bit.

Si tratta di registri a 16 bit ai quali è anche possibile accedere con le operazioni a 8 bit. Ognuno di questi registri può infatti essere suddiviso in 2 parti (parte alta e parte bassa) nel seguente modo:

Registro	Parte Alta	Parte Bassa
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

Registri puntatore e indice

SP – Stack Pointer: contiene l'indirizzo della cima dello stack.

BP – Base Pointer: può puntare qualsiasi cella del segmento di stack.

SI – Surce Index: utilizzato come offset in combinazione con BP o BX.

DI – Destination Index: utilizzato come offset in combinazione con BP o BX.

Registri IP e Flag

IP – Instruction Pointer: punta alla prossima istruzione da eseguire

Flag – Il registro flag o PSW (Program Status Word) è un insieme di bit che assumono valori in base ai risultati delle operazioni per tenere traccia dello stato del sistema.

Nel simulatore sono presenti 3 flag:

1. C – Flag Carry: vale uno quando in un operazione si verifica un riporto
2. O – Flag Overflow: vale uno quando si verifica un overflow
3. I – Interrupt Flag: vale uno quando viene bloccata l'esecuzione.

Registri di Segmento

Dati, codice e stack si trovano in sezioni diverse della memoria ognuna delle quali è identificata da un indirizzo di segmento.

CS – Code Segment: punta al segmento di codice.

DS – Data Segment: punta al segmento dati.

SS – Stack Segment: punta al segmento di stack.

ES – Extra Segment: punta al segmento extra (non presente nel simulatore).

Questi registri sono sfruttati per l’indirizzamento ma nel simulatore l’utente non può interagirvi.

Sintassi del codice

Il codice va suddiviso in 4 settori

1. Intestazione dedicato alle costanti (vedi Costanti)
2. Testo dedicato alle istruzioni (vedi Istruzioni)
3. Dati dedicato alle variabili (vedi Variabili)
4. BSS dedicato allo spazio allocato (vedi Spazio Alloccato)

Di questi solo il settore Testo non è opzionale.
I settori vanno definiti nel seguente ordine e con la seguente sintassi:

```
Intestazione
.
.
.SECT .TEXT
.
.
Settore Testo
.
.
.SECT .DATA
.
.
Settore Dati
.
.
.SECT .BSS
.
.
Settore BSS
```

Costanti

_<*costante*> = <*valore*> !<*commento*>

Le costanti devono avere un nome univoco e un valore.

Il nome della costante deve essere preceduto dal carattere '_' (trattino basso).

Il valore può essere espresso in binario, ottale, decimale o esadecimale (vedi sintassi per l'utilizzo di valori costanti)

I commenti devono iniziare con il carattere '!' (punto esclamativo); in una riga ogni carattere successivo al '!' è considerato commento e viene escluso dall'assemblatore.

Istruzioni

<*etichetta*>: <*istruzione*> <*operando 1*>, <*operando 2*> !<*commento*>

Le etichette sono opzionali, devono iniziare con una lettera e devono terminare con il carattere ':' (due punti).

Due istruzioni non possono stare nella stessa riga (vedi lista completa delle istruzioni).

Gli operandi dipendono dall'istruzione che si vuole eseguire e devono essere separati dal carattere ',' (virgola).
Possono essere registri (vedi informazioni sull'architettura), etichette (che devono essere scritte senza il carattere ':'), valori numerici (vedi sintassi per l'utilizzo di valori costanti) oppure riferimenti alla memoria (vedi sintassi operandi referenziali).

I commenti devono iniziare con il carattere '!' (punto esclamativo); in una riga ogni carattere successivo al '!' è considerato commento e viene escluso dall'assemblatore.

Variabili

<*variabile*>: .<*tipo*> <*valore*> **!<*commento*>**

Il nome della variabile (come le etichette) deve iniziare con una lettera e deve terminare con il carattere ':' (due punti).

Il tipo può essere:

1. .BYTE **<*valore*>** valore di 8 bit (1 byte)
2. .BYTE **<*vettore*>** vettore di byte
3. .WORD **<*valore*>** valore di 16 bit (2 byte = 1 word)
4. .WORD **<*vettore*>** vettore di word
5. .ASCII **'<*carattere*>'** carattere, valore di 8 bit (1 byte)
6. .ASCII **“<*stringa*>”** vettore di caratteri

Un vettore ha la seguente sintassi: **<*valore1*>, <valore2>, …. <*valoreN*>**
ogni valore va separato dal successivo tramite il carattere ',' (virgola).

Sintassi valori costanti

I valori possono essere espressi in binario, ottale, decimale o esadecimale seguendo la seguente sintassi:

1. Esadecimale: **0x<*valore*>**
2. Decimale: **<*valore*>** oppure **<*valore*>d**
3. Ottale: **0o<*valore*>**
4. Binario: **<*valore*>b**

I commenti devono iniziare con il carattere '!' (punto esclamativo); in una riga ogni carattere successivo al '!' è considerato commento e viene escluso dall'assemblatore.

Spazio Allocato

<*variabile*>: .SPACE <*spazio*> **!<*commento*>**

Il nome della variabile (come le etichette) deve iniziare con una lettera e deve terminare con il carattere ':' (due punti).

Lo spazio sta a indicare quante celle di memoria verranno allocate.

I commenti devono iniziare con il carattere '!' (punto esclamativo); in una riga ogni carattere successivo al '!' è considerato commento e viene escluso dall'assemblatore.

Sintassi operandi referenziali

<offset numerico> **(*<registro base>*)** **(*<registro offset>*)**

L'offset numerico e il registro di offset (comprese le sue relative parentesi) sono opzionali.

L'offset numerico indica a quante celle, oltre quella puntata dal registro base, si trova la cella che si vuole puntare.

Il registro di offset indica a quante celle, oltre quella puntata dal registro base + l'offset numerico, si trova la cella che si vuole puntare.

I registri che possono essere utilizzati come registro base sono:

1. BX (DS : BX)
2. BP (SS : BP)
3. SI (DS : SI)
4. DI (DS : DI)

Solo i primi due (BX e BP) consentono l'utilizzo del segmento di offset.

I registri che possono essere utilizzati come registro di offset sono:

1. SI
2. DI

OPPURE

(*<indirizzo numerico>*)

L'indirizzo numerico punta a una cella nel segmento dati (DS : *indirizzoNumerico*)

Set Istruzioni

MOV<B> <operando destinazione>, <operando sorgente>

Copia e sposta il valore dell'operando sorgente in quello di destinazione.
Nella versione MOVB gli operandi devono essere a 8 bit.

MOV registro, registro
MOV registro, indirizzo
MOV registro, valore
MOV indirizzo, registro
MOV indirizzo, valore

ADD<B> <operando destinazione>, <operando sorgente>

Somma il valore dei due operandi e lo salva nell'operando di destinazione.
Nella versione ADDB gli operandi devono essere a 8 bit.

ADD registro, registro
ADD registro, indirizzo
ADD registro, valore
ADD indirizzo, registro
ADD indirizzo, valore

SUB<B> <operando destinazione>, <operando sorgente>

Esegue una sottrazione con il valore dei due operandi e lo salva nell'operando di destinazione.
Nella versione *SUBB* gli operandi devono essere a 8 bit.

SUB registro, registro
SUB registro, indirizzo
SUB registro, valore
SUB indirizzo, registro
SUB indirizzo, valore

INC<B> <operando destinazione>

Incrementa l'operando di destinazione.
Nella versione *INCB* l' operando devono essere a 8 bit.

INC registro
INC indirizzo

DEC<B> <operando destinazione>

Decrementa l'operando di destinazione.
Nella versione *DECB* l' operando devono essere a 8 bit.

DEC registro
DEC indirizzo

MUL<B> <operando sorgente>

Moltiplica il registro AX per l'operando sorgente e salva il risultato in DX : AX.
Nella versione *MULB* moltiplica il registro AL per l'operando sorgente e salva il risultato in AX.

MUL registro
MUL indirizzo

DIV<B> <operando sorgente>

Divide il registro DX : AX per l'operando sorgente e salva il risultato in AX e il resto in DX.
Nella versione *DIVB* divide il registro AX per l'operando sorgente e salva il risultato in AL e il resto in AH.

DIV registro
DIV indirizzo

PUSH <operando sorgente>

Inserisce un nuovo elemento nello stack.

PUSH registro
PUSH indirizzo
PUSH valore

POP <operando destinazione>

Rimuove un elemento dallo stack e lo salva nell'operando di destinazione.

POP registro
POP indirizzo

JMP <etichetta>

Salta all'istruzione successiva all'etichetta.

CMP<B> <operando 1>, <operando 2>

Esegue una sottrazione tra i due operandi per confrontarli e altera i flag.
Nella versione *CMPB* gli operandi devono essere a 8 bit.

CMP registro, registro
CMP registro, indirizzo
CMP registro, valore
CMP indirizzo, registro
CMP indirizzo, valore

LOOP <etichetta>

Salta all'istruzione successiva all'etichetta finché CX non vale 0.

JB <etichetta>

Salta all'istruzione successiva all'etichetta se il flag carry vale 1.

JBE <etichetta>

Salta all'istruzione successiva all'etichetta se il flag carry e il flag zero valgono 1.

JA <etichetta>

Salta all'istruzione successiva all'etichetta se il flag carry e il flag zero valgono 0.

JAE <etichetta>

Salta all'istruzione successiva all'etichetta se il flag carry vale 0.

JE <etichetta>

Salta all'istruzione successiva all'etichetta se il flag zero vale 1.

JNE <etichetta>

Salta all'istruzione successiva all'etichetta se il flag zero vale 0.

JCXZ <etichetta>

Salta all'istruzione successiva all'etichetta se CX vale 0.

HLT

Imposta il flag interrpution a 1 arrestando l'esecuzione del programma.

CALL <etichetta>

Inserisce nello stack il valore del registro IP e salta all'istruzione successiva all'etichetta con il codice della funzione (i parametri vanno passati tramite lo stack).

RET

Rimuove dallo stack il valore del registro IP precedente alla chiamata della funzione e salta all'istruzione successiva.