TEMA 1 − VISIÓ GENERAL DEL i386

1.1.− Estructura bàsica d'un programa

.model large

.386

"definició de constants"

"definició de la pila" = . stack "mida" = 100 h

"definició de les variables globals"

"definició del codi"

end

DEFINICIÓ DE CONSTANTS

DEFINICIÓ DE VARIABLES GLOBALS

.data

X DB "valor inicial"

DW ?

DD

Directiva	**Significat**	**Mida de la dada**
DB	Define Byte	1 byte
DW	Define Word	2 bytes
DD	Define Double Word	4 bytes

1.2.− Instruccions

Característica bàsica del i386: màquina de 2 operands. Només es poden especificar dos operands en una mateixa instrucció.

Exemple: si es vol fer una suma, el resultat s'ha de guardar en un dels dos operands

Classificació de les instruccions

TIPUS D'INSTRUCCIÓ	**SINTAXI**
Moviment	**MOV** d, f ; d f **XCHG** op1, op2 ; op1 op2
	ADD d, f ; d d + f

Aritmètiques	**SUB** d, f ; d d − f
Lògiques	AND op1, op2 ; op1 op1 · op2 OR op1, op2 ; op1 op1 + op2 XOR op1, op2 ; op1 op1 " op2 NOT op ; op op
Transferència de control	Salt condicional: CMP op1, op2 JXX ETIQUETA JE salta si op1=op2 JNE salta si op1 " op2 JG salta si op1 > op2 JGE salta si op1 " op2 JL salta si op1 < op2 JLE salta si op1 " op2 Salt incondicional: JMP ETIQUETA

1.2.− Modes d'adreçament

Es defineixen com el mode d'especificar un operand d'una instrucció

Registres i386

De propòsit general (8 de 32 bits), són:

EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP

Substituir ? per la lletra de registre que correspongui.

Els registres ESI, EDI es tracten de manera diferent:

Substituir ? per la lletra de registre que correspongui.

Els registres ESP i EBP controlen la pila.

A) MODE REGISTRE

Exemples:

MOV EBX, EAX; EBX EAX

MOV BX, AX

MOV BL, AL

MOV AH, AL

B) MODE INMEDIAT

Exemples:

MOV EAX, 0 ; inicialitza el registre a 0

MOV EBX, N; mou la contant N al registre EBX

MOV ECX, N+4; ECX N+4

C) MODE MEMÒRIA

Memòria

Espai d'@ físic

Espai d'@ lògic

Per adreçar l'espai d'@ lògic s'han d'especificar 2 coses:

− SEGMENT

− DESPLAÇAMENT = Posició concreta a la que es vol accedir (OFFSET)

Per especificar un segment s'utilitza un registre de segment:

L'i386 té 6 registres de segment de 16 bits:

DS Data Segment Indica que volem accedir al segment de dades

CS Code Segment Indica que volem accedir al segment de codi

SS Stack Segment Indica que volem accedir al segment de la pila

ES, FS, GS Els utilitzarem per accedir a dades

AMB AQUESTS REGISTRES S'ESPECIFICA L'ADREÇA A LA QUE COMENÇA EL SEGMENT

Per especificar un desplaçament utilitzarem sempre 32 bits que tindran una restricció:

Aquesta restricció la tenim perque treballem en **MODE REAL**, el primer del 8086.

Els que diu l'@ real són els altres 16 bits. Tenim, per tant, 216 possibles desplaçaments, és a dir, 64 KB.

Per calcular l'@ física a partir de la lògica:

Nota: multiplicar * 16 és desplaçar 4 bits a l'esquerra la dada, això dóna velocitat a la operació.

El valor concret de l'@ física dels segments el calcula el Sistema Operatiu, de manera que reparteix la memòria com vol.

Es multiplica * 16 per calcular l'@ física perque el 8086 estava dissenyat així:

A l'hora d'adreçar a memòria es desplaçaven 4 bits cap a l'esquerra, de manera que es guanyen 4 bits per segment.

Ara, en desplaçar 4 bits, tenim segments de 20 bits (220 b=1 MB), ja que tenim registres de 32 bits.

El **MODE ABSOLUT** permet especificar l'@ en la mateixa instrucció.

Exemples:

El que fa és:

Exemple d'instruccions:

```
.model large

.386

N EQU 3

.stack 100h

.data

VAR1 DD 10

VAR2 DW −1

VAR3 DB 16,1

VAR4 DD 0

.code

.startup

MOV EAX, N ; EAX N=3

MOV EAX, N+4 ; " MOV EAX, 7

MOV EAX, DS:VAR1 ; " MOV EAX, DS:0000 "

; EAX Md [DS:VAR1]
```

MOV EAX, VAR2

MOV AX, VAR2

.exit

.end

Tot el que estigui a la memòria es representa en Hexadecimal.

Enmagatzematge de la informació a memòria

Little endian

B0
B1
B2
B3

Big endian

B3
B2
B1
B0

Continguts equivalents

Gestió de la pila

Pila: estructura de dades amb accés LIFO (Last In First Out)

Instruccions

PUSH operand (registre, inmediat, operand en mode adreçament)

Exemples: PUSH EAX (1)

PUSH EBX (2)

POP operand

Exemples: PUSH EBX (1)

PUSH EAX (2)

Altres exemples:

PUSH 10 Ambigüitat: no es diu quina longitud agafem per guardar

PUSH DWORD PTR 10 Ara ja no hi ha ambigüitat, perquè guardem la dada en

mida DWORD.

.data

VAR DD 10

PUSH VAR Això és vàlid pq ja se sap el tamany de la dada (l'hem declarat abans).

POP DWORD PTR 10 No és vàlid, perquè no es pot fer un POP d'un inmediat.

PUSH i POP no admeten operands de mida byte

Cas:

Necessito un registre pel programa i tots estan ocupants

PUSH EAX

Treballo amb el registre. Quan acabi:

POP EAX

Recupero la dada que habia al registre.

RESUM TEMA 1

De propòsit general ! EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP

De segment ! DS, SS, CS, ES, FS, GS

Registres: Instruction Pointer ! EIP = IP

Paraula d'estat (EFLAGS) ! Guarda els bits de condició:

Z ! ZF

S ! SF

Carry ! CF

Overflow ! OF

Memòria:

Adreça lògica ! segment : desplaçament

Adreca física ! segment * 16 + desplaçament

Pila

Segments Dades

Codi

Moviment de dades

Aritmètiques

Lògiques

De transferència de control

Registre

Modes d'adreçament Inmediat

Memòria (absolut)

. code indica l'inici del programa

. startup

...

... Encapsulen el programa

...

. exit Instruccions del programa

Nom del símbol

Directiva de l'assemblador

Número

N EQU "valor"

Reserva l'espai però sense assignar−li valor

Amb això es declara la mida de la variable

E?X

?H

?L

0

8

15

31

E?I

?I

0

15

31

byte

@

0

2n

Espai lineal.

En cada casella hi cap 1 byte.

Equivalències:

1 KB = 210 Bytes

1 MB = 220 Bytes

1 GB = 230 Bytes

64 = 26

64 MB = 220 + 26 = 226

Per fer els MB

Per fer els 64

SEGMENT DE DADES

(DS)

SEGMENT DE CODI

(CS)

SEGMENT DE PILA

(SS)

Espai segmentat.

Aquests tres segments els té qualsevol programa

0

16

32

Aquesta meitat sempre estarà a 0

@F = f(@L)

@F = SEGMENT * @L + DESPLAÇAMENT

@

15

0

AX

BX

CX

DX

SI

DI

SP

BP

DS

CS

SS

ES

15

0

SEGMENT

Els : (dos punts) s'utilitzen per a separar el segment del desplaçament

Etiqueta que defineix el desplaçament.

Estaria declarada abans així:

.data

VAR DD 10

MOV EAX, DS:VAR

Indica el tipus de dada que anem a buscar:

b 8 bits [Byte]

w 16 bits [Word]

d 32 bits [DWord]

EAX Md [DS:VAR] = 10

Defineix una variable nova cada vegada que es posa una coma. La primera variable és VAR3. La segona no té un nom concret. Les dues són de tipus BYTE

Suma l'inmediat al valor de la constant i després el mou

MOV EAX, VAR1

Per defecte agafa el **DS**

ERROR!!!!

Perque EAX és de 32 bits i VAR2 és de 16 bits

Ara ja no hi ha problema perque són de la mateixa longitud, a l'agafar la meitat del registre. La part alta del registre queda sense modificar

− VAR DB 16,1

− MOV EAX, VAR1+4

EAX 0110FFFF

Little endian al revès

Enters de dec a hex:

−1 1 1 FFFE FFFF

dec dec (+) hex Ca15 + 1

− VAR1 DD 10 " 00 00 00 0A

− VAR2 DW −1 " FF FF

0A

00

00

00

FF

FF

10

01

0000

0001

0002

0003

0004

0005

0006

0007

1 BYTE

B3

B2

B1

B0

0

7

15

23

31

De − a + pes

(És el que utilitza el iX86)

De + a − pes

Podem usar qualsevol de les 3 representacions, sempre que a l'@ li augmentem el número de bytes que hi ha en cada posició de memòria

0000

0001

0002

0003

0004

0005

0006

0A

00

00

FF

FF

10

01

00

0007

0000

0002

0004

0006

00 0A

00 00

FF FF

01 10

0000

0004

00 00 00 0A

01 10 FF FF

8 BITS

[1 BYTE]

16 BITS

[1 WORD]

32 BITS

[1 DWORD]

APLIQUEM LITTLE ENDIAN AL REVÈS

PILA

SS

ESP

Creix

Es buida

SS Stack Segment

.stack "mida"

ESP : Extended Stack Pointer desplaçament per accedir a les diferents dades de la pila

L'ESP apunta sempre a l'última posició de la pila (la pila està buida)

PUSH op ; ESP ESP − nº bytes operand

M [SS:ESP] operand

Accès a memòria

Stack Segment amb desplaçament

0

EAX

EBX

1

2

ESP

ESP ESP + nº bytes operand

POP op ; operand M [SS:ESP]

0

EAX

EBX

1

2

ESP

Instruccions