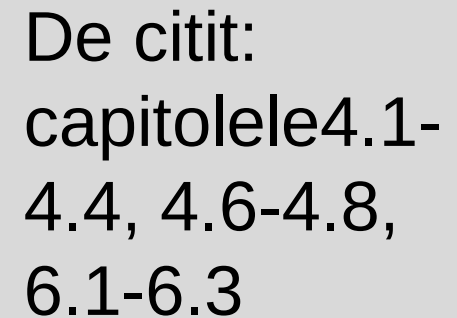

Moduri de adresare, declararea datelor, definire constante, operatori, macro-uri

Modificat: 15-Oct-18



De citit:
capitolele 4.1-
4.4, 4.6-4.8,
6.1-6.3

Cuprins capitol 5

- Sintaxa instrucțiunilor
 - Reguli sintactice
 - Semnificatia entitatilor unei linii de program
- Moduri de adresare
 - Declararea variabilelor
 - Pseudo-operatori
 - Macro-uri

Declarații

1. Directive

- » Îi spun asamblorului ce să facă
- » Ne-executabile
- » Nu generează cod mașină

2. Instrucțiuni

- » Îi spun CPU-ului ce să facă
- » Operațiuni + operanzi

3. Macrouri

- » Notăție scurtă pentru un grup de declarații
- » Substituții cu parametri

Directive

- Directive care descriu structura programului
 - * text, data, rodata, bss
 - * Import/export nume
- Reminder(curs 2): structura procesului
- Va urma(curs 6): structura binarului

Variabile globale

- Trei zone de memorie
 - * `.data`: initialize
 - * `.bss`: neinitialize (zero @load-time)
 - * `.rodata`: initialize, read-only
- `.data`: `int a = 10;`
- `.bss`: `int a;`
- `.rodata`: `const int a = 10;`
- Variabile locale declarate **'static'** in C

Directiva GLOBAL

- Directiva GLOBAL marcheaza etichetele vizibile global
 - » etichetele pot fi accesate si din alte module ale programului
 - Formatul este
`global label1, label2, . . .`
 - Aproape orice label poate fi declarat global
 - » Nume de proceduri
 - » Nume de variabile
 - » equated labels
- * Intr-o constructie GLOBAL, nu este necesar sa mentionam tipul labelului

Directiva EXTERN

- Directiva EXTERN ii spune asamblorului ca anumite labeluri nu sunt definite in modulul curent
 - * Asamblorul rezerva spatiu in fisierul obiect pentru a fi utilizat ulterior de linker
- Formatul este
extern label1, label2, . . .
unde **label1** si **label2** sunt declarate global folosind directiva **GLOBAL** in alte module

Istrucțiuni

[eticheta:] mnemonic [operanzi] [;comentariu]

- eticheta – șir de litere și cifre, care începe cu o literă
- mnemonic – nume care simbolizează o instrucțiune

- * **nop**

- * **jz begin**

- * **add eax, 1**

- begin:**

- * **mov [buffer + ebx], ax**

- operanzi – registru | locație memorie | imediat

- * **0 – 2 operanzi**

Sintaxa instrucțiunilor

- registru := EAX|EBX|AX|BX|AL|BL|..|ESI|..|CS|DS..|GS
- imediat:= număr sau expresie evaluată de asamblor
 - * Adresa unei variabile este un imediat
- locație:= [expresie care produce o adresă logică]
 - * Între paranteze drepte [expresie]
 - * [variabilă + reg_index + reg_bază + deplasament]
 - * deplasament = imediat
- **mov [ceva + esi + ebx*4 + 19], ax**
 - Locație = adresa variabilei ceva
 - Locație += 19; ceva + 19 este cunoscut la momentul asamblării
 - Locație += (ESI + EBX << 2)
 - La adresa indicată de Locație se stochează 16 biți din AX

Sintaxa instrucțiunilor x86

- * Instrucțiuni fara operand
 - » NOP
 - » MOVSb
- * instrucțiuni cu un operand
 - » PUSH EAX
 - » ROR DX
- * instrucțiuni cu doi operanzi
 - » MOV AX, BX
- * linie cu eticheta instrucțiune și comentariu
 - » START: MOV AX, BX ;mută conținut AX în BX
- * linie de comentariu
 - » ; aceasta este o linie de comentariu
- * linie cu eticheta
 - » ETICHETA:

Reguli sintactice

- o singură instrucțiune/directivă per linie
- o linie poate conține:
 - * nici o entitate de instrucțiune (camp) – linie goală
 - * Numai etichetă
 - * Numai comentariu
 - * eticheta, instrucțiune, directivă și comentariu
- Comentariu începe cu ';' și se încheie la sfârșitul liniei
- o instrucțiune x86 poate conține maxim 2 operanzi:
 - * op1 – destinația și primul termen al operației
 - * op2 – sursa sau al doilea termen al operației

Reguli sintactice - Example

- * **NOP**

- » Instrucțiune fara operanzi

- * **MOVSB**

- » instrucțiune cu operanzi impliciti

- * **MUL CL**

- » instrucțiune cu primul operand implicit ($AX := AL * CL$)

- * **MOV AX, BX**

- » $AX := BX$ adică AX – destinația , BX sursa transferului

- * **INC SI**

- » $SI := SI + 1$

- * **ADD CX, DX**

- » $CX := CX + DX$

- * **ADD AX, BX, CX**

- » Instrucțiune incorecta, prea multi operanzi

Reguli sintactice

- “case insensitive”
- Separarea câmpurilor se face cu spații, TAB-uri
- Pentru lizibilitate: separare coloane cu TAB:
eticheta: mnemonic operanzi ;comentariu
- Nume simbolice în loc de valori numerice
 - * adrese de variabila =>nume_variabila;
 - * adrese de instrucțiune =>eticheta;
 - * valori de constante numerice=>nume_constanta

Reguli sintactice- simboluri

- Simboluri, identificatori, etichete:
 - * secventa de litere, cifre si unele caractere speciale (ex: _, \$, @), care nu incepe cu o cifra
 - * Lungimea simbolului este arbitrara, dar se considera primele 31 caractere
 - * Exista simboluri rezervate, predefinite in limbaj (cuvinte cheie, mnemonice, directive, nume registre)
 - * exemple:
 - » L1 BletchRightHereRight_Here Item1 __Special
 - » \$1234 @Home \$_1 Dollar\$ WhereAml? @1234
 - * erori:
 - » 1TooMany – incepe cu o cifra
 - » Hello.There – contine punct
 - » \$ - \$ sau ? nu poate sa apara singur
 - » LABEL – cuvant rezervat.

Reguli sintactice - constante

- Constante:
 - * intregi: 12, 21d, 123h, 0fffh, 1011b
 - * reale (virgulă mobilă): 1.5, 1.0e10, 23.1e-12
 - * Șir de caractere: "text", 'Text', 'TEXT' 'TEXT'
 - * Constante simbolice:
 - » În asm: `unu equ 1`
 - » În C: `#define unu 1`

Reguli sintactice - operanzi

- Operanzii trebuie să aibă aceeași lungime
 - * excepții: operații de înmulțire și împărțire)
- cel mult un operand de tip locație de memorie
 - * Cum facem operații între două locații de memorie?
 - * Operațiile între registre – viteză mare
- Instrucțiunile sunt structural atomice
 - * Sunt independente între ele
 - * **nu există forme de programare structurată**
 - * Structurarea programului: la nivel logic, prin directive

Semnificația entităților unei linii de program

- Eticheta:
 - * Adresa instrucțiunii care urmează după etichetă
 - * Util pentru instrucțiuni de salt și apel de rutine

Mnemonică(numele) instrucțiunii:

- * Nume simbolic dat unui cod de instrucțiune
- * Semnifică operația elementară direct executabilă de CPU
- * Nu indica dimensiunile operanzilor (sunt inferate)
- * Aceeași instrucțiune poate avea mnemonici diferite(JZ JE)
- * Fiecărei instrucțiuni ASM îi corespunde strict o instrucțiune în cod mașină (relație biunivocă)

Semnificația entităților unei linii de program

- Operand:
 - * camp care exprima un termen al operatiei elementare exprimate prin mnemonica
 - * Indica locul si modul de regasire al operandului (modul de adresare folosit)
 - * tipuri de operanzi:
 - » registre interne ale CPU:
 - » date imediate (constante numerice)
 - » locatii de memorie (variabile)
 - » porturi de intrare sau de iesire (registre de I/E)

Semnificația entităților unei linii de program

- Registre interne:
 - * Registre generale:
 - » (8 biti) AH,AL,BH,BL,CH,CL,DH,DL
 - » (16 biti) AX, BX,CX,DX, SI,,DI,SP, BP
 - » (32 biti) EAX, EBX,ECX,EDX, ESI,EDI,ESP, EBP
 - * registrespeciale: CS,DS, SS, ES, FS,GS, GDTR, LDTR , CR0,..CR4, PSW
- Date imediate (constante):
 - * Numar sau expresie aritmetico-logica evaluabila la un numar =>expresia trebuie sa contina numai constante
 - * Valoarea este continuta in codul instrucțiunii
 - * Lungimea constantei – in acord cu lungimea celui de al doilea operand (octet, cuvant sau dublu-cuvant)
 - * ex: 0, -5, 1234h, 0ABCDh, 11001010b, 1b, $8 * 4 - 3$

Semnificația entităților unei linii de program

- Locatii de memorie (variabile):
 - * expresie care exprima adresa unei locatii de memorie de o anumita lungime
 - * Lungimea variabilei:
 - » in acord cu al doilea operand (daca exista)
 - » se deduce din declaratia numelui de variabila
 - » se indica in mod explicit ('byte', 'word', 'dword')
 - * Adresa variabilei:
 - » adresa de segment:
 - specificata in mod implicit – continutul registrului DS
 - exprimata in mod explicit: <reg_segment>:<variabila>
 - ex: CS: Var1, ES: [100h]
 - » adresa de offset – adresa relativa in cadrul segmentului

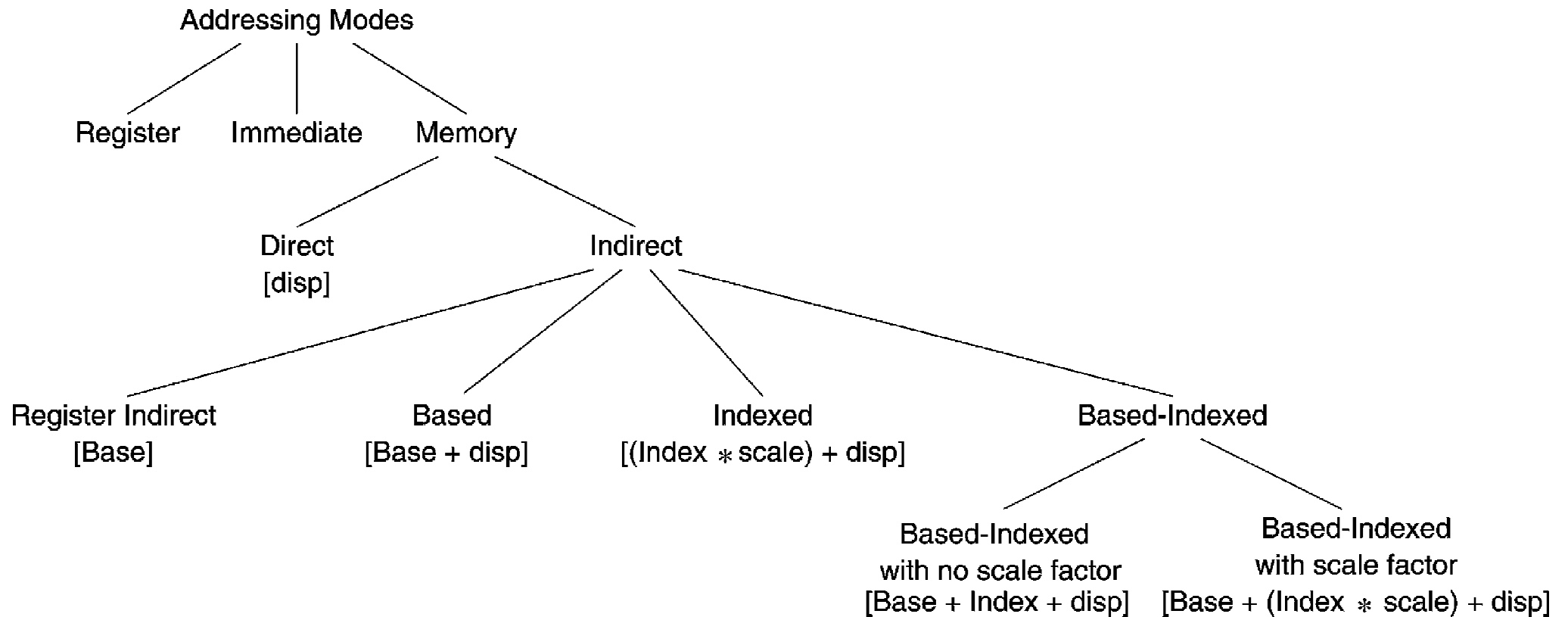
Semnificația entităților unei linii de program

- adresa de offset
 - * Adresa poate sa fie pe 16 biti (modul real) sau pe 32 biti (modul protejat)
 - * exprimabila in mai multe moduri:
 - » adresafizica: valoare concreta de adresa
 - '['<adresa_offset>']', ex: MOV AX, [100h]
 - '['[<reg_baza>][+<reg_index>][<deplasament>']'
 - » ex: MOV AX, [BX+SI+100h]
 - » Adresa simbolica: - nume simbolic dat unei variabile
 - VAR1, TEXT, VAR+5
 - VAR[BX], VAR[BX+SI]
 - * <reg_baza>:= BX|BP|EBX|EBP
 - * <reg_index>:=SI|DI|ESI|EDI

Semnificația entităților unei linii de program

- Porturi de Intrare/Iesire
 - * Registre continute in interfetele de intrare/iesire
 - * spatiul de adresare maxim: 64Ko (adr. maxima 0FFFFH)
 - * Pe aceeași adresa pot fi 2 registre:
 - » un reg. de intrare si unul de iesire
 - * Porturile apar doar in instrucțiunile IN si OUT
 - * specificare:
 - » direct, prin adresa fizică (daca adresa este exprimabila pe un octet) sau nume simbolic
 - » ex: IN AL, 12h
 - » OUT 33h, AL
 - » indirect, prin adresa continuta in registrul DX
 - » ex: IN AL, DX
 - » OUT DX, AL

Moduri de adresare



Moduri de adresare pt. ISA x86 (16 biti/8086)

Adresarea imediata:

- * Operandul este o constanta
- * Operandul este continut in codul instrucțiunii
- * Operandul este citit o data cu instrucțiunea
- * Instrucțiunea poate lucra cu o singura valoare
- * Lungimea constantei este in acord cu celalalt operand
- * Flexibilitate limitata

exemple:

- * `MOV AL, 12h` `MOV AL, 120`
- * `MOV AX, 12ABh` `MOV AL, 260 – eroare`
- * `MOV EAX, ceva` ; adresa lui ceva este cunoscută

Moduri de adresare pt. ISA x86 (16 biti/8086)

- Adresarea de tip registru:
 - * Operandul este continut intr-un registru al UCP
 - * timp de acces foarte mic; nu necesita ciclu de transfer pe magistrala
 - * Instrucțiune scurta (nr. mic de biti pt. specificare operand)
 - * Numar limitat de registre interne => nu toate variabilele pot fi pastrate in registre
 - * Exista limitari in privinta registrelor speciale (ex: segment)
 - * exemple:
 - * `MOV AX, BX` `MOV DS, AX`
 - * `MOV BX, AL` – eronat `MOV DS, 1234H` - eronat

Moduri de adresare pt. ISA x86 (16 biti/8086)

- Adresarea directa (cu deplasament):
 - * Operandul este specificat printr-o adresa de memorie (adresa relativa fata de inceputul unui segment)
 - * Adresa operandului este continuta in codul instructiunii
 - * Instructiunea poate lucra cu o singura locatie de memorie (octet, cuvânt sau dublu-cuvânt)
 - * Necesita ciclu suplimentar de transfer cu memoria => timp de executie mai mare
 - * Adresarea directa se foloseste pt. variabile simple (date nestructurate)

exemple:

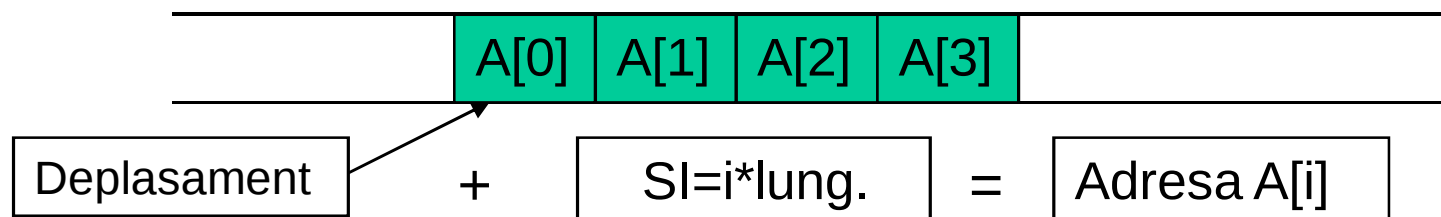
- * `MOV AL, [100h]` `MOV BX, var1`
- * `MOV CX, [1234h]` `MOV var2, SI`

Moduri de adresare pt. ISA x86 (16 biti/8086)

- Moduri indirecte de adresare:
- Adresarea indirecta prin registru:
 - » Adresa operandului se specifica intr-un registru
 - » Registrele folosite pt. adresare: SI, DI, BX, BP
 - » Instrucțiunea contine adresa registrului
 - » mod flexibil de adresare
 - » exemple:
 - » MOV AL, [SI]
 - » MOV [BX], CX

Moduri de adresare pt. ISA x86 (16 biti/8086)

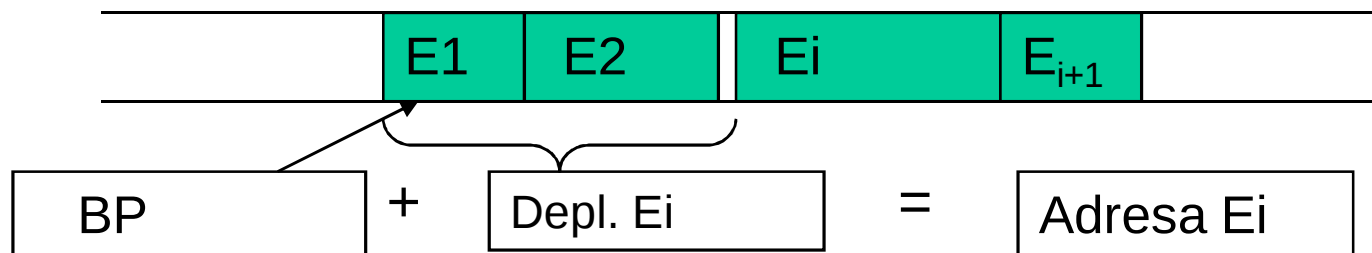
- Adresarea (indirecta) indexata:
 - * Adresa operandului se exprima printr-o adresa de baza, data de <deplasament> si un index dat de continutul unui registru
 - * mod de adresare folosit pentru structuri de date de tip sir, vector, tablou
 - * sintaxa: $\text{<nume_var>['<reg_index>']}$
 $\text{<reg_index>:=SI|DI}$
 - * $\text{['<reg_index>+<deplasament>']}$
 - * exemple:
 - * `MOV AX, VAR[BX]` `MOV CX, [SI+100H]`
 - * `MOV VAR[DI], AL` `MOV VAR[10H], 1234H`



Moduri de adresare pt. ISA x86 (16 biti/8086)

- Adresarea (indirecta) bazata:

- * Adresa operandului se exprima printr-o adresa de baza, data de un registru si o adresa relativa data de <deplasament>
- * mod de adresare folosit pentru structuri de date de tip inregistrare
- * formal este identica cu adresarea indexata, dar alta interpretare
- * sintaxa: $\langle \text{nume_var} \rangle [\langle \text{reg_index} \rangle]$
 $\langle \text{reg_baza} \rangle := \text{BX} | \text{BP}$
- * $[\langle \text{reg_baza} \rangle + \langle \text{deplasament} \rangle]$
- * exemple:
- * MOV AX, VAR[BX] MOV CX, [SI+100H]
- * MOV VAR[DI], AL $\text{MOV [SI][100h], 1234H}$



Moduri de adresare pt. ISA x86 (16 biti/8086)

- Adresarea mixta (bazat indexata):
 - * Adresa operandului se exprima printr-o adresa de baza, data de un registru, un index dat de un registru si o adresa relativa data de <deplasament>
 - * mod de adresare folosit pentru structuri complexe de date de tip inregistrare de vectori sau vector de inregistrari
 - * Modul cel mai flexibil de adresare, dar **necesita 2 adunari**
 - * sintaxa: <nume_var>['<reg_baza>+<reg_index>']
 - * ['<reg_baza>+<reg_index>+<deplasament>']
 - * ['<reg_baza>']['<reg_index>']['<deplasament>']
 - * exemple:
 - * MOV AX, VAR[BX+SI] MOV CX, [BX+SI+100H]
 - * MOV VAR[BP+DI], AL MOV VAR[BP+SI], 1234H
 - * MOV VAR[BP][DI], AL MOV [100h][BP][SI], 1234H

Moduri de adresare pt. ISA x86 (32 biti/386+)

- Extensia registrelor generale la 32 biti: EAX, EBX, ...
- Toate registrele generale pot fi folosite pentru adresarea indirecta prin registru, indexata, bazata si mixta;
- ex: [EAX], [ECX], VAR[EAX+ECX], [DX+AX+100h]
- la adresarea mixta primul registru se considera registru de baza iar al doilea registru index
- Observatii:
 - * Registrul SP nu poate fi folosit ca registru index
 - * Daca se foloseste SP sau BP atunci implicit se lucreaza cu reg. segment SS

Moduri de adresare pt. ISA x86 (32 biti/386+)

- Adresarea indexata, scalata:
 - * Permite multiplicarea registrului index cu un factor egal cu lungimea unui element din sir:
 - » 1 pt. octet, 2 pt. cuvant, 4 pt. dcuvantsi 8 pt. qcuvant
 - * Simplifica parcurgerea tablourilor a caror elemente sunt mai mari de 1 octet
 - * sintaxa: (in loc de '+' se poatefolosi '[')'
 - » '['<reg_index>*n']'
 - » '['<reg_index>*n + <deplasament>']'
 - » '['<reg_baza> + <reg_index>*n']'
 - » '['<reg_baza> + <reg_index>*n + <deplasament>']'
 - » ex: MOV AX, [SI*2] MOV DX, [AX*4+12h]
 - » MOV CX, 100h[BX][AX*1]

Declararea variabilelor

- Scopul:
 - * Utilizarea unor nume simbolice in locul unor adrese fizice
 - * rezervarea de spatiu in memorie si initializarea variabilelor
 - * pt. verificarea utilizarii corecte a variabilelor (verificare de tip)
- Modul de declarare: - prin directive
- Directiva (pseudo-instrucțiune):
 - * entitate de program utilizata pentru controlul procesului de compilare, editare de legaturi si lansarea programului
 - * directivele NU SE EXECUTA; in programul executabil nu exista cod aferent pentru directive
 - * se folosesc pentru:
 - » Declararea variabilelor si a constantelor
 - » Declararea segmentelor si a procedurilor
 - » Controlul modului de compilare, editare de legaturi, etc.

Declararea variabilelor

- Octeti:

- » sintaxa:

- <nume_var> DB ?|<valoare>

- » semnificatia:

- se rezerva o locatie de memorie de 1 octet;
 - Locatia este initializata cu <valoare>, sau este neinitializata daca apare '?'
 - <nume_var> - eticheta ce simbolizeaza adresa variabilei
 - <valoare> - valoare numerica in intervalul [0..255] sau [-128..127]
 - Poate pastra: un numar intreg fara semn, un numar intreg cu semn, un cod ASCII, 2 cifre BCD

Declararea variabilelor

- Cuvinte:

- » sintaxa:

- `<nume_var> DW ?|<valoare>`

- » semnificatia:

- se rezerva o locatie de memorie de 2 octeti;
 - Locatia este initializata cu `<valoare>`, sau ramane neinitializata daca apare '?'
 - `<nume_var>` - eticheta ce simbolizeaza adresa variabilei
 - `<valoare>` - valoare numerica in intervalul $[0..2^{16}-1]$ sau $[-2^{15}..2^{15}-1]$
 - Poate pastra: un numar intreg fara semn, un numar intreg cu semn, 2 coduri ASCII, 4 cifre BCD

Declararea variabilelor

- Dublu-cuvinte:

- » sintaxa:

- `<nume_var> DD ?|<valoare>`

- » semnificatia:

- se rezerva o locatie de memorie de 4 octeti;
 - Locatia este initializata cu `<valoare>`, sau este neinitializata daca apare '?'
 - `<nume_var>` - eticheta ce simbolizeaza adresa variabilei
 - `<valoare>` - valoare numerica in intervalul $[0..2^{32}-1]$ sau $[-2^{31}.. 2^{31}-1]$
 - poatepastra: un numar intreg fara semn, un numar intreg cu semn, 4 coduri ASCII, 8 cifre BCD,

Exemple de declaratii de variabile simple

• m	db	?	erori		
• l	db	6	l	db	260
• j	db	-7	al	dw	23
• l	db	255	tt	db	-130
• k	db	-23			
• bits	db	10101111b			
• car	db	'A'			
• Cuv	dw	1234h			
• Var	dw	0FFFFh			
• Dcuv	dw	12345678h			

Declararea variabilelor

- Variabile simple lungi:
 - * dq (define QWORD/quad-word)
 - » variabilape 8 octeti; folosit pentru pastrarea intregilor f. mari sau a valorilor in flotant (dubla precizie)
 - * dt (define TBYTE/ten-bytes)
 - » Variabila pe 10 octeti; format folosit pt. coprocesorul matematic; se reprezinta 10 cifre BCD (despachetat) sau nr. flotant pe 80 biti

Declararea variabilelor

- Exemple comentate:

```
db 0x55                ; just the byte 0x55
db 0x55,0x56,0x57      ; three bytes in succession
db 'a',0x55            ; character constants are OK
db 'hello',13,10,'$'   ; so are string constants
dw 0x1234              ; 0x34 0x12
dw 'a'                 ; 0x61 0x00 (it's just a number)
dw 'ab'                ; 0x61 0x62 (character constant)
dw 'abc'               ; 0x61 0x62 0x63 0x00 (string)
dd 0x12345678          ; 0x78 0x56 0x34 0x12
dd 1.234567e20         ; floating-point constant
dq 0x123456789abcdef0  ; eight byte constant
dq 1.234567e20         ; double-precision float
dt 1.234567e20         ; extended-precision float
```

Declararea constantelor

- Scop: - nume simbolic dat unei valori des utilizate
- Sintaxa:
 - * `<nume_constanta>equ|= <expresie>`
- Semnificatia:
 - * la compilare `<numeconstanta>` se inlocuieste cu `<expresie>` ; este o constructie de tip MACRO
 - * sintaxa se verificadoar la inlocuire
 - * `<expresie>` este o expresia ritmetico-logica evaluabila in momentul compilarii => termenii sunt constante sau operatorul '\$'
 - * '\$' – reprezinta valoarea contorului curent de adrese

Declararea constantelor

- Exemple:

* trei equ 3

* true equ 0

* text db 'acesta este un text'

* lung_text equ \$-text

* Adr_port equ 378h

Repetarea declaratiilor sau a instructiunilor

- TIMES
 - * Este un prefix ce produce repetarea de un numar specificat de ori a instructiunii sau a declaratiei de date
- Example:
 - * Alocare 64 octeti:
zerobuf: times 64 db 0
 - * Initializare si alocare pana la 64 octeti:
buffer: db 'hello, world'
times 64 - \$ + buffer db ''
 - * Executie multipla a unei instructiuni (loop unrolling trivial)
times 100 movsb

Pseodo-operatori

- Expresii aritmetico-logice
 - * trebuie sa se evalueze in procesul de compilare
 - * contin constante si variabile de compilare
- \$ - pozitia la inceputul liniei ce contine expresia curenta
 - * Ex: JMP \$; reprezinta bucla infinita
- \$\$ - pozitia de la inceputul sectiunii curente
 - * Ex: \$-\$\$; reprezinta unde ne aflam in sectiune

Pseudo-operatori

- Operatori logici, la fel ca in C.
- In ordinea cresterii prioritatilor:
- Operatori pe biți
 - * `|`, `^`, `AND`
- Operatori shiftare de biți
 - * `<<`, `>>`
- Operatori binari:
 - * `+`, `-`
 - * `*`, `/`, `//` (signed), `%`, `%%` (signed)
- Operatori unari:
 - * `+`, `-`, `~`, `!`, `SEG` (obține segmentul unui simbol)

Forțare de tip (coercion)

- sintaxa:
 - * `<tip><expresie>`
- `<tip>` poate fi:
 - * `BYTE` (1 octet)
 - * `WORD` (2 octeti)
 - * `DWORD` (4 octeti)
 - * `QWORD` (8 octeti)
 - * `TBYTE` (10 octeti)
- exemple de utilizare:
 - * `inc byte [BP-10]`
 - * `fadd qword [EDX+ECX*8]`

Macro-uri

- Sunt forme prescurtate de scriere a unor secvențe de program care se repeta

- sintaxa:

```
%macro macro_name[para_count]  
<macro body>  
%endmacro
```

- Exemplu definire:

```
%macro multEAX_by_16  
salEAX,4  
%endmacro
```

- Exemplu utilizare:

```
...  
mov EAX,27  
multEAX_by_16  
...
```

Macro-uri cu parametri

- `<para_count>` specifica numarul de parametri
- `<%n>` identifica al n-lea parametru
- Exemplu definire:

%macro mult_by_16 1

sal %1,4

%endmacro

- Exemplu utilizare:

mult_by_16 DL

- *macro-ul se expandeaza la:*

sal DL,4

Macro-uri v.s. Proceduri

- Macro-uri:
 - * la fiecare apel se copiaza secventa de instrucțiuni
 - * nu sunt necesare instrucțiuni de apel (CALL) si de revenire din rutina (RET)
 - * nu se foloseste stiva
 - * transferul de parametri se realizeaza prin copierea numelui
- Proceduri:
 - * o singura copie pt. mai multe apeluri
 - * se folosesc instrucțiuni de apel si de revenire
 - * se utilizeaza stiva la apel si la revenire
 - * transferul de parametri se face prin registri sau stiva

Avantajele si dezavantajele macro-uri

- Avantaje:
 - * pot fi create “instrucțiuni” noi
 - * poate duce la o programare mai eficienta
 - * executie mai eficienta in comparatie cu apelurile de proceduri
- Dezavantaje:
 - * pot ascunde operatii care afecteaza continutul registrilor
 - * utilizarea extensiva a macrourilor ingreuneaza intelegerea si mentenanta programului

Intrebari?

