Declararea datelor, definire constante, operatori, macro-uri

Modificat: 22-Oct-23

De citit: capitolele4.1-4.4, 4.6-4.8, 6.1-6.3

Cuprins

- Sintaxa instrucțiunilor
- Semnificatia entitatilor unei linii de program
- Moduri de adresare
- Pseudo-operatori
- Macro-uri

Semnificația entităților unei linii de program

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

Mnemonica(numele) instrucțiunii:

- Nume simbolic dat unui cod de instrucțiune
- Semnifică operatia 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 aleCPU:
    - date imediate (constante numerice)
    - locații de memorie (variabile)
    - porturi de intrare sau de ieșire (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):
  - expresie aritmetico-logica evaluabila la un numar Valoarea este conținută 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
  - Adresele variabilelor din .data, .bss sunt cunoscute = imediate

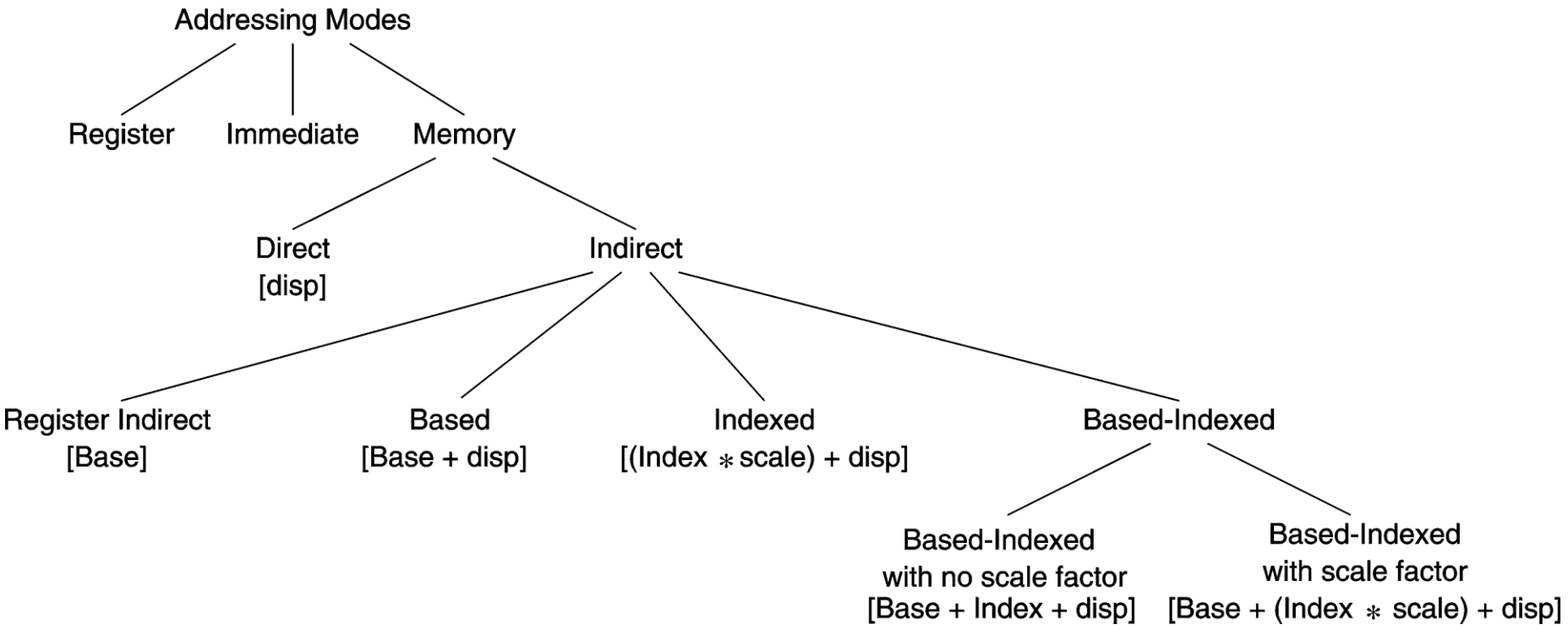
Semnificația entităților unei linii de program

- Locații de memorie: indicate cu [adresă]
  - Operatorul [] semnifică dereferențierea
  - Lungimea variabilei:
    - » in acord cu al doilea operand (dacă exista)
    - » se indica in mod explicit ('byte', 'word', 'dword')
    - » Nu există tipuri
    - » Adresa este pentru un octet (0..4GB)
  - Adresa variabilei seg:offset
    - » 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
  - pe 32 biți (acest semestru)
  - exprimabilă in mai multe moduri:
    - » Imediat sau adresă cunoscută la momentul linkeditării
      - MOV AX, [100h] ; se încarcă doi octeți de la adresa 0x100
      - MOV AX, [v1] ; AL:=octetul de la v1, AH:=octetul de la v1+1
      - MOV AX, [v1+1] ; AL:=octetul de la v1+1, AH:=octetul de la v1+2
      - După linkeditare v1 și v1+1 au valori cunoscute ~ imediate
    - » Expresie evaluată la momentul execuției
      - '[<reg_baza>] [+<reg_index>*scale] [<deplasament>']'
      - MOV AX, [EBX+ECX*4+100h]
  - <reg_baza>:= EAX|EAX|ECX|EDX|ESI|EDI|EBP|ESP
  - <reg_index>:= EAX|EAX|ECX|EDX|ESI|EDI|EBP
  - scale := 1|2|4|8
  - Expresia produce adresa primului octet al unei variabile

Moduri de adresare



Moduri de adresare

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

* MOV AL, 12h
* MOV AL, 120
* MOV AX, 12ABh
* MOV AL, 260 – warning, ajunge 4 în AH
* MOV EAX, ceva ; adresa lui ceva este cunoscută

Moduri de adresare

- Adresarea de tip registru:
  - Operandul este continut intr-un registru
  - timp de acces mic; nu necesita transfer pe magistrala
  - Instrucțiune scurta (nu specifică operand)
  - Numar limitat de registre interne => nu toate variabilele pot fi pastrate in registre
  - Exista limitari in privinta registrelor speciale (ex: segment)
  - MOV EAX, EBX
  - MOV DS,AX
  - MOV BX, AL – eronat
  - MOV DS, 1234H - eronat

Moduri de adresare

- Adresarea directa cu deplasament:
  - Operandul este specificat printr-o adresa de memorie Adresa operandului este continuta in codul instrucțiunii
  - Instrucțiunea poate lucra cu **o singura locatie de memorie** (octet, cuvânt, sau dublu-cuvânt)
  - Necesita pasul data-fetch cu memoria =>timp mai mare

  exemple:
  - MOV AL, [100h]
  - MOV EBX, var1
  - MOV CX, [1234h]
  - MOV var2, SI ; eroare

Moduri de adresare

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

Moduri de adresare

- 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:
    <nume_var>'['<reg_index>']' <reg_index>:=SI|DI
  - '['<reg_index>+<deplasament>']'
  - exemple:
  - MOV AX, [var + EBX]
  - MOV CX, [ESI+100H]
  - MOV [var + EDI], AL
  - MOV word [var + 10H], 1234H

Moduri de adresare

- Adresarea 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>
  - 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+EBX+ESI]
  - MOV CX, [EBX+ESI+100H]
  - MOV [var + EBP+EDI], AL
  - MOV [var + EBP+ESI], 1234H
  - MOV [var + BP + DI], AL - eroare, necesită adresa pe 32biți
  - MOV [100h + EBP + ESI], 1234H

Moduri de adresare (32 biți)

- 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. Dcuvant si 8 pt. qcuvant
  - Util pentru tablourilor cu elemente de 2/4/8 octeți
    - '['<reg_index>*n']'
    - '['<reg_index>*n + <deplasament> ']'
    - '[' <reg_baza> + <reg_index>*n']'
    - '[' <reg_baza> + <reg_index>*n + <deplasament> ']'
    - MOV AX, [ESI*2]
    - MOV DX, [EAX*4+12h]
    - MOV CX, [100h + EBX + AX*1]; eroare dimensiune operanzi

• Moduri de adresare, observații

- **nu există tipuri**
- **toate declarațiile (db, dw, dd, …) sunt pointeri!**
- **numele este adresa primului octet al variabilei**
- **folosim numele când e necesară adresa**
- **folosim dereferențierea cu [ ] când e necesar conținutul**

```
z dw 0x1234
; z = adresa octetului 0x34
; z nu are tip

mov ebx, z
; ebx conține adresa

mov ax, [z]
; ax conține 0x1234
; se cer explicit 2 octeți
; nu există tipuri
```

```
short z = 0x1234;
// z este variabilă cu tip 16bit

p = &z;
// p conține adresa lui z,
// necesită 4 octeți

short a = z;
// se copiaza 2 octeți (tipul)
// a conține 0x1234
```

demo

- Se folosește un schelet simplu hello.asm (curs-02) cu variabilele

```
v1 dd 0xabcd1234
v2 dd 0x7890aabb
```

Se adaugă instrucțiuni care exemplifică adresarea

```
1. mov ebx, 0x56559026
2. mov ebx, v1
3. mov ebx, [v1]
4. mov eax, v2
5. mov bx, [eax]
```

demo

```
$ objdump -s ./hello.o ; adresele alocate
$ objdump -s ./hello   ; adresele după link
$ objdump -M intel -d hello
```

Se observă codul generat pentru

- **Instrucțiuni 1 & 2**
- **Diferența între 2 și 3**

Rulare în gdb:

- **Observare efect instrucțiuni**
- **Examinarea memoriei cu x/8xb &v1**

Ce este v1?

- este un imediat (calculat la momentul linkeditării)
- este adresa primului octet al unei variabile (instr. 2)
- este un pointer (instr. 3)
  - Operatorul [] indică un acces la memorie

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

- define byte:

  <nume_var> DB ?|<valoare>

  - 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 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

- define word :

<nume_var> DW ?|<valoare>

  - se rezerva o locatie de memorie de 2 octeti;
  - Locatia este initializata cu <valoare>, sau neinitializata '?'
  - <nume_var> - eticheta ce simbolizeaza adresa variabilei
  - <valoare> - valoare 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

- define double word:

<nume_var> DD ?|<valoare>

  - se rezerva o locatie de memorie de 4 octeti (int în C);
  - Locatia este initializata cu <valoare>, sau neinitializata '?'
  - <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    ?
l      db    6
j      db    -7
l      db    255
k      db    -23
bits   db    10101111b
car    db    'A'
Cuv    dw    1234h
Var    dw    0FFFFh
Dcuv   dw    12345678h
```

erori

```
l      db    260
al     dw    23
tt     db    -130
```

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 expresiea 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 instrucțiunilor

- TIMES
  - Este un prefix ce produce repetarea de un numar specificat de ori a instrucțiunii sau a declaratiei de date
- Exemple:
  - 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 instrucțiuni (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
- $ - contorul de alocare la linia curentă

```
buffer: db ’hello, world’
len equ $ - buffer
```

Echivalent cu

```
buffer: db ’hello, world’
endbuf:
len equ endbuf - buffer
```

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 (obtine segmentul unui simbol)

Forțare de tip (coercion)

- <tip>poate fi:
  - BYTE (1 octet)
  - WORD (2 octeți)
  - DWORD (4 octeți)
  - QWORD (8 octeți)
  - TBYTE (10 octeți)

Exemplu

```
mov dword [z], -1
;0x5655902a <z>: 0xff 0xff 0xff 0xff
add byte [z], 1
;0x5655902a <z>: 0x00 0xff 0xff 0xff + ZF CF PF AF
mov dword [z], -1
;0x5655902a <z>: 0xff 0xff 0xff 0xff
add word [z], 1
;0x5655902a <z>: 0x00 0x00 0xff 0xff + ZF CF PF AF
mov dword [z], -1
;0x5655902a <z>: 0xff 0xff 0xff 0xff
add dword [z], 1
;0x5655902a <z>: 0x00 0x00 0x00 0x00 + ZF CF PF AF
```

- Care este diferența față de cast în C?

Macro-uri

- 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
sal EAX,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 vs 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?