# A4. Dezvoltarea programelor în limbaj de asamblare (I)

Etapele prin care trece un program scris în limbaj de asamblare până a deveni un program executabil sunt prezentate în schema din figura 8.1. Procesul de obţinere a executabilului este un process repetitiv; trecerea de la o fază la alta se face numai dacă au fost eliminate toate erorile sintactice din faza curentă. În ultima fază de testare cu ajutorul depanatorului se elimină erorile de logică mergând în paralel cu microprocesorul şi comparând în diferite puncte rezultatele obţinute cu cele dorite.

**Editorul de texte** este folosit la introducerea programului scris în limbaj de asamblare şi respectă convenţiile şi sintaxa impusă de asamblor rezultând un program sursă (**.asm**). Se recomandă utilizarea unui editor simplu ASCII.
**Asamblorul** este un program de conversie în cod maşină a programelor scrise în limbaj de asamblare (.asm) obţinând un fişier obiect (.**obj**). Asamblorul bsolute asigură şi prelucrarea etichetelor simbolice astfel încât fiecare adresă simbolică este înlocuită cu o adresă relativă sau absolută.
**Editorul de legături (Link-editorul)** realizează legarea mai multor module obiect rezultate din asamblări sau componente de bibliotecă obţinând un program în format absolut sau relocabil (toate adresele sunt relative la o bază şi se transformă în adrese absolute la încărcare) (.**com, .exe**).
**Depanatorul** este utilizat la testarea programului executabil şi încărcat în memorie, pentru a înlătura erorile semantice ale programului folosind facilităţile debbuger-ului.
Programul sursă trebuie să respecte convenţiile şi sintaxa pe care o cere programul asamblor. Liniile programului sursă pot conţine instrucţiuni sau directive de asamblare (pseudoinstrucţiuni).

### A4.1. Definirea şi iniţializarea datelor

Tipurile de date întâlnite în programe pot fi:

- constante
- variabile
- etichete

Instrucţiunile de transfer, aritmetice, logice (MOV, ADD, XOR) folosesc ca operanzi variabile şi constante în timp ce instrucţiunile de salt, apel (JMP, CALL) folosesc ca operanzi etichetele.

**Etichetele** sunt utilizate pentru identificarea codului, desemnând adrese în zona program.
Exemplu:

```
Aduna: add ax,bx
       inc bx
       jmp Aduna        ;Aduna - etichetă de tip NEAR
```

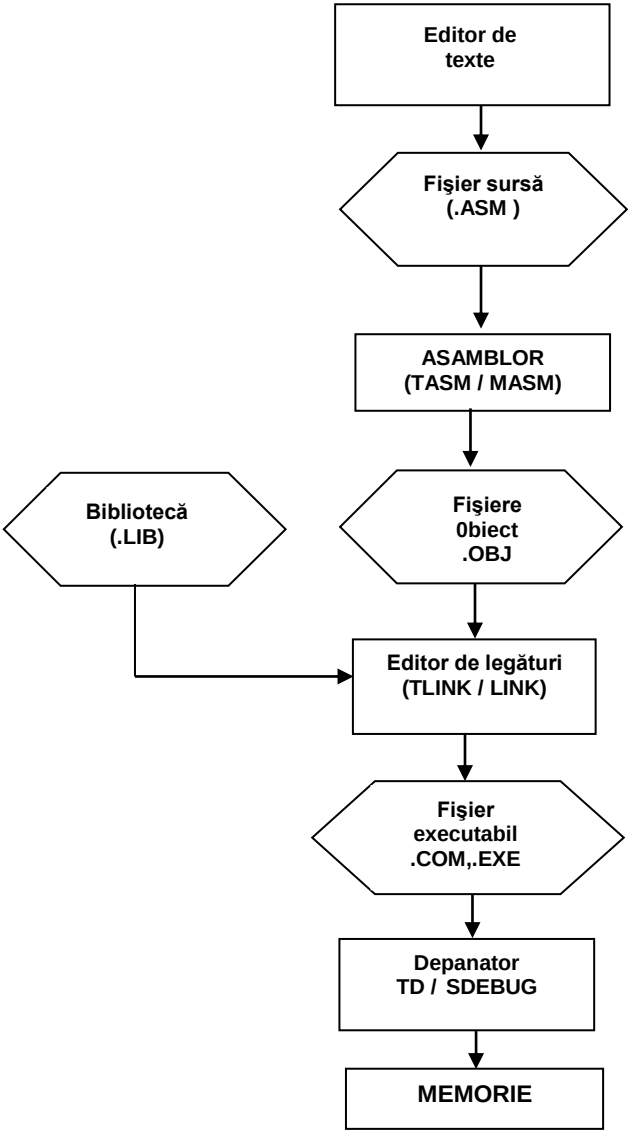


Fig.A4.1. Dezvoltarea aplicaţiilor în limbaj de asamblare

**Constantele** pot fi absolute (numerice) sau simbolice – nume generice asociate unor valori numerice. Constantele numerice pot fi de tipurile prezentate mai jos.

| | |
|---|---|
| Binare | ex. 11001110B |
| Zecimale | ex. 28[D] |
| Hexa | ex. 32h, 0A5h |
| Octale | ex. 56o |
| Caracter | ex. 'alba' |

Constantele simbolice se pot defini astfel:
Nume EQU expresie

Exemplu: `CR EQU 0Dh`

**Variabilele** identifică datele (un spaţiu de memorie rezervat) şi pot fi de următoarele tipuri:

BYTE - în memorie sau în registru de 8 biţi
- întreg cu/fără semn pe 8 biţi
- caracter ASCII

WORD - în memorie sau în registru pe 16 biţi
- întreg cu/fără semn pe 16 biţi
- secvenţă de 2 caractere ASCII
- adresă de memorie

DWORD - în memorie sau într-o pereche de regiştri pe 16 biţi
- întreg cu/fără semn pe 32 biţi
- număr real în simplă precizie
- adresă de memorie pe 32 biţi

QWORD - în memorie
- întreg cu/fără semn pe 64 biţi
- număr real în dublă precizie

TBYTE - în memoria internă sau în registrele coprocesorului aritmetic
- număr real în precizie extinsă (80 biţi)
- număr întreg ca secvenţă de cifre BCD

Variabilele se definesc folosind directivele: DB, DW, DD, DQ, DT cu sintaxa de mai jos:
Nume **directiva** lista_de_valori

unde în lista_de_valori putem avea :
- constante ex. A DB 0CFh, 21
- simbolul (?) pentru locaţie neiniţializată ex. STIVA DW 256 DUP(?)

- variabilă sau etichetă
- şir ASCII ex. SIR DB 'Exemplu'
- operatorul DUP pentru iniţializare tablou, ex: STRING DB 10 DUP(0).

**Operatori**. Limbajul de asamblare pune la dispoziţia utilizatorilor mai mulţi operatori pentru modificarea, combinarea, compararea sau analiza operanzilor. Operatorii apar în expresii aritmetice şi logice evaluate la momentul asamblării şi din care rezultă constante numerice.

- *operatori aritmetici şi logici*: +, -, *, /, MOD, SHL, SHR, NOT, AND, OR, XOR

Exemple:
```
MOV BX,6        ; BX=6
SHL BX,3        ; BX=30H
XOR BX,6        ; BX=BX⊕6, BX=36H
```

- *operatori relaţionali* : EQ, NE, LT, LE, GT, GE care au semnificaţii evidente şi returnează valori logice codificate ca 0 sau secvenţe de 1 pe un anumit numar de biti.

Exemple:
```
A  DB 5 EQ 7    ; A=0
C  DW 1 NE 5    ; C=0FFFFh
```

- *operatori de atribuire:* definesc constante simbolice

```
nume_var EQU expresie
```

Exemplu:
```
N EQU 17        ; sau N=17
```

- *operatori care returnează valori*: se aplică unor variabile sau etichete returnând valori ale acestora.
- *operatorul SEG* aplicat variabilelor sau etichetelor returnează adresa de segment asociată variabilei respective

```
MOV AX, SEG V1
```

- *operatorul OFFSET*, similar operatorului SEG returnează offset-ul variabilei/etichetei

```
MOV BX,OFFSET V1
```

- *operatorul THIS* creează un operand care are o adresă de segment şi un offset identic cu contorul de locaţii curent. Sintaxa de utilizare este: THIS tip

```
VAR EQU THIS WORD
```

unde tip poate fi: BYTE, WORD, DWORD, QWORD sau TBYTE pentru variabile sau NEAR/FAR pentru etichete.
- *operatorul TYPE* se aplică variabilelor sau etichetelor şi returnează tipul acestora. Pentru variabile se returnează valorile 1, 2, 4, 8 sau 10 (dacă anterior au fost definite cu directivele: DB, DW, DD, DQ, DT), iar pentru structuri numărul de octeţi pe care este memorată structura respectivă. Pentru etichete returnează tipul NEAR sau FAR.
Exemplu:

```
VAR DW 1,2,10h          ; variabila VAR este de tip cuvânt, cu valorile 1,2 şi 10h
    sub BP, TYPE VAR    ; asamblorul înlocuieşte TYPE VAR cu 2.
```

- *operatorul LENGTH* se aplică variabilelor şi returnează numărul de elemente definite în variabila respectivă

Exemplu:

```
TAB     DW 50 DUP(?)
MOV CX, LENGTH TAB     ; cx=50
```

- *operatorul SIZE* aplicat variabilelor returnează dimensiunea în octeţi a variabilei respective, astfel pentru exemplul anterior expresia SIZE TAB =100.
Pentru o variabilă oarecare "X" avem: SIZE X = (LENGTH X) * (TYPE X).

- *operatorul PTR* are ca efect schimbarea tipului variabilei/etichetei şi utilizarea lui este obligatorie în cazul unor referinţe anonime la memorie. Sintaxa de utilizare este:

tip PTR expresie

Exemplu:

```
INC BYTE PTR[BX]; octetul adresat de BX este incrementat

JMP FAR PTR Var ; salt de tip FAR
```

Exemplu:

```
DATA DB 03,04,05,06     ;variabila DATA declarată ca sir
                        ;de octeti
mov AX, WORD PTR DATA   ;asamblorul foloseste elementele
                        ;variabilei DATA ca şi cuvinte
mov WORD PTR DATA+2,DX ;se obţine compatibilitate cu
                        ;tipul cuvânt al registrelor
```

- *operatorii NEAR/FAR/SHORT* poziţionează tipul unei etichete în modul specificat de operator.

Exemple:

```
JMP     NEAR  ET1       ;salt intrasegment
JMP     SHORT ET        ;salt scurt sub 128 octeţi
```

- *operatorii HIGH/LOW* returnează octetul cel mai semnificativ respectiv cel mai puţin semnificativ al unei expresii.

Exemple:
```
FULL    EQU     1234H
MOV AH, HIGH FULL       ;AH=12H
MOV AL, LOW FULL        ;AL=34H.
```

Directiva *LABEL*, având forma generală:
***nume* LABEL *tip*** ,unde ***tip*** poate fi
- NEAR sau FAR dacă ceea ce urmează sunt instrucţiuni si eticheta va fi folosită pentru referirea la instrucţiuni de tip JMP/CALL şi
-BYTE, WORD, DWORD dacă ceea ce urmează reprezintă definiţii de date.

Exemplu:
```
sirb LABEL BYTE   ; s-a definit eticheta cu numele sirb ce
                  ; permite accesul la variabila sirw octet
                  ; cu octet
sirw DW 1234h
 mov AL, sirb     ; AL=34h
```

### A4.2 Definirea segmentelor

Un modul dintr-un program poate fi: o porţiune dintr-un segment, un segment, porţiuni din segmente diferite sau mai multe segmente. Modulele obiect se leagă la link-editare. Instrucţiunile şi datele dintr-un program .asm, indiferent de modul de dezvoltare, trebuie să se găsească în interiorul unui segment. Definirea unui segment se face după modelul de mai jos, folosind directivele SEGMENT respectiv ENDS:

*nume* **SEGMENT** *[tip_aliniere] [tip_combinare] ['clasa']*
.
.
.
*nume* **ENDS**

unde :
- ***nume*** este numele segmentului
- ***tip_aliniere*** specifică tipul adresei de inceput a segmentului unde va fi relocat segmentul în memorie şi poate fi: PARA (implicit) paragraf – multiplu de 16 bytes, BYTE, WORD, DWORD, PAGE - multiplu de 256 bytes.
- ***tip_combinare*** specifică dacă şi cum se va combina segmentul respectiv cu alte segmente la editarea de legături. Poate fi:
  - PUBLIC : segmentul curent se va concatena cu alte segmente având acelaşi nume şi atributul PUBLIC;
  - COMMON: segmentele cu acelaşi nume şi atribut se vor suprapune, lungimea finală va fi maximul lungimii segmentelor componente;

STACK : specifică segmentul curent ca fiind stiva programului (LIFO) dacă sunt mai multe segmente cu tipul STACK se vor concatena;
AT exp : indică faptul că segmentul curent va fi plasat la o adresă fizică absolută dată de exp în memorie;
Necombinabil - implicit (nu se scrie nimic).

- ***clasa*** indică un nume de clasă pentru segment care poate fi : 'code' , 'data', 'stack'. Dacă segmentul are şi nume de clasă atunci atributele COMMON şi PUBLIC vor acţiona numai asupra segmentelor cu acelaşi nume şi din aceeaşi clasă.

Directiva **ASSUME** reg_seg: nume_seg [, ….] realizează o conexiune logică între definirea datelor şi instrucţiunilor în segmente la asamblare şi accesul, la execuţie, la date şi instrucţiuni prin intermediul regiştrilor segment. Directiva ASSUME nu realizează încărcarea regiştrilor segment cu adresele de segment corespunzătoare.

Exemplu: `ASSUME CS:COD_SEG, DS:D_SEG, SS:STIVA, ES:NOTHING.`
Deci la instrucţiunea : `MOV BX,CEVA ; BX=(DS:CEVA).`

Directiva **GROUP** se utilizează la gruparea mai multor segmente sub acelaşi nume (chiar având nume şi atribute diferite) şi a căror lungime nu depăşeşte 64Ko. Sintaxa directivei este :

```
NUME_GRUP       GROUP   NUME_SEG1, NUME_SEG2, …
```

Exemplu:

```
DATA1   SEGMENT
Var1   db   10h
DATA1   ENDS
DATA2   SEGMENT
Var2   dw   20h
DATA2   ENDS
DATA   GROUP   DATA1, DATA2
       ; după această directivă GROUP, putem avea
       ; în segmentul de cod:        mov   ax, data
                                     mov   ds, ax
                                     mov   ax, OFFSET Var1
       ; echivalentă cu instrucţiunea
       ; mov ax, OFFSET data:Var1
       ; sau directive de forma:  ASSUME   ds : data
```

Utilizarea acestei directive reprezintă o altă modalitate de combinare a mai multor segmente pe lângă cea oferită de atributul PUBLIC.

Directiva **END** marchează sfârşitul logic al unui program. Ea apare obligatoriu în toate modulele. Eticheta este punctul de start după încărcarea programului : `END eticheta`

**Contorul de locaţii** indică offsetul curent, fiind accesibil prin $.

```
mesaj db "un mesaj"
lung_mesaj EQU $-mesaj
```

Exemplu:

```
sirAscii db "12345"           ;sir de octeţi: coduri Ascii ale
                              ;carac. "1","2","3","4","5".
lung_sir EQU $-sirAscii       ; valoarea $-nume variabila
               ;indică lungimea (asem. length) variabilei
```

Directiva **ORG** modifică explicit contorul de locaţii.

Exemplu: `ORG 100H.  ;asamblează la offsetul absolut 100h=256`

**Structura unui program .COM :**

```
PR_COM SEGMENT PARA PUBLIC 'CODE'
ORG 100h
ASSUME CS:PR_COM,DS: PR_COM, SS: PR_COM, ES: PR_COM,
Start: JMP inceput
       ; datele
inceput:
       ; proceduri
       ..
       MOV AX,4C00H
       INT 21H

PR_COM ENDS
       END Start
```

Obţinerea programului executabil (.com) pentru aplicaţii (medii) Microsoft şi Borland se face prin secvenţa de prelucrări de mai jos.

| | |
|---|---|
| **MASM PR_COM.ASM** | **TASM PR_COM.ASM** |
| **LINK PR_COM.OBJ** | **TLINK /t PR_COM.OBJ** |
| **EXE2BIN PR_COM.EXE** | **PR_COM.COM** |

**Structura unui program .EXE :**

```
STIVA  SEGMENT PARA STACK 'STACK'
       DW 512 DUP(?)
STIVA  ENDS
```

```
DATA   SEGMENT       PARA   PUBLIC 'DATA'
       ;
       ;DATE
       ;
DATA   ENDS
COD    SEGMENT PARA PUBLIC 'CODE'

       MAIN PROC FAR

       ASSUME CS:COD, DS:DATA, SS:STIVA,ES: NOTHING
       PUSH   DS
       XOR    AX,AX
       PUSH   AX
       MOV    AX,DATA
       MOV    DS,AX
       ...
       RET
       ;proceduri
MAIN   ENDP
       COD    ENDS
       END    MAIN
```

Obţinerea programului executabil (.exe) se face astfel:

**MASM PROG_COM.ASM TASM PROG_COM.ASM**
**LINK PROG_COM.OBJ TLINK PROG_COM.OBJ**

### A4.3 Exerciţii şi teme

1. Cum se defineşte o variabilă “x” pe cuvânt, iniţializată cu 5?

2. Cum se defineşte un şir de 100 octeţi neiniţializaţi?

3. Cum se încarcă într-un registru adresa variabilei x? Dar a şirului definit anterior? Ce registru se foloseşte pentru adresare?

4. Se consideră o variabilă definită astfel: SIR db 1,2,3,4,5,6,7,8,9,0. Comentaţi instrucţiunea mov ax, word ptr SIR[2]. Ce se va încărca în registrul AX ?

5. Definiţi o constantă cu numele CR, având valoarea 0Dh.

6. Specificaţi ce va conţine registrul AL, în urma execuţiei instrucţiunii: mov al,Var1.

7. Care va fi valoarea lui B din expresia: B dw 6 LT 7 ?

8. Fie variabila: SIR dd 2,3,4,5,6,7. Specificaţi care vor fi valorile pentru length, size şi type.

9. Scrieţi instrucţiunile corespunzătoare următoarelor operaţii:

- copiere dată imediată 1200h în registrul segment DS
- continutul locatiei de memorie 10h să fie mutat în locaţia de memorie 20h
- conţinutul registrului segment DS să fie mutat în registrul segment ES.

10. Scrieţi un program complet care adună două numere, urmărind paşii:
    a. Definiţi o variabilă “a” pe un octet iniţializată cu 2;
    b. Definiţi o variabilă “b” pe un octet iniţializată cu 3;
    c. Definiţi o variabilă “sum” pe un octet neiniţializată;
    d. Scrieţi secvenţa care adună cele două numere şi depune rezultatul în “sum”;
    e. Completaţi toate aceste fragmente în modelul de structură a programului .exe şi salvaţi cu extensia .asm;
    f. Asamblare: ***tasm*** program.asm. Studiaţi opţiunile de asamblare: ***tasm ?***; creaţi fişier listing pentru programul scris;
    g. Linkeditare: ***tlink*** program.obj;
    h. Execuţie asistată: ***td*** program.exe.

11. Scrieţi un program complet care determină maximul dintr-un şir.
    a. Definiţi un şir de 10 octeţi iniţializaţi cu numere aleatoare;
    b. Definiţi o variabilă “maxim” pe octet;
    c. Scrieţi o secvenţă de program care determină maximul dintr-un şir;
    d. Scrieţi sursa (nume.asm);
    e. Asamblaţi aplicaţia; generaţi şi listingul;
    f. Linkeditaţi aplicaţia;
    g. Executaţi programul cu td.exe.

12. După modelul prezentat, scrieţi un program care determină minimul dintr-un şir de 10 numere.

13. Scrieţi un program care calculează suma elementelor unui şir de numere. Parcurgeţi şirul în două moduri: prin adresare bazată indexată şi folosind instrucţiuni specifice şirurilor.