LABORATOR - nr. 1

PĂRȚI ALE UNUI COMPILATOR

1. Analiza lexicală
2. Gestiunea datelor
3. Analiza sintactică
4. Generarea codului intermediar
5. Optimizarea codului
6. Generarea codului obiect

Se consideră instrucțiunea

$$A := (B + C * D + E) * F$$

A, B, C, D, E, F sunt variabile de tip real.

Să se traducă această instrucțiune urmărind toate etapele proiectării unui compilator.

1. Analiza lexicală.

$$< id, A >:= (< id, B > + < id, C > * < id, D > + < id, E >) * < id, F >$$

2. Gestiunea datelor.

Nr. crt.	Identificatorul	Informația
1.	A	Variabilă în virgulă mobilă
2.	B	Variabilă în virgulă mobilă
3.	C	Variabilă în virgulă mobilă
4.	D	Variabilă în virgulă mobilă
5.	E	Variabilă în virgulă mobilă
6.	F	Variabilă în virgulă mobilă

3. Analiza sintactică.

Se construieşte arborele sintactic reprezentat în figura 1, unde :

$$id_1 \overset{not}{=} A \quad id_2 \overset{not}{=} B \quad id_3 \overset{not}{=} C \quad id_4 \overset{not}{=} D \quad id_5 \overset{not}{=} E \quad id_6 \overset{not}{=} F$$

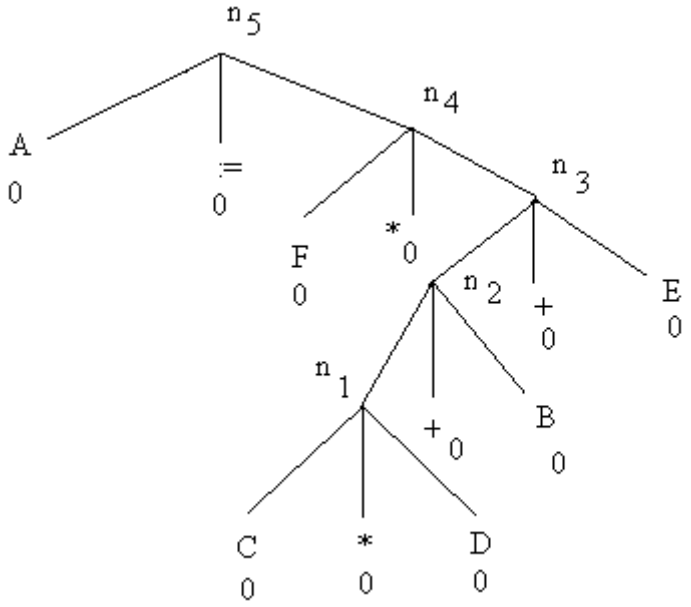

Fig. 1

4. Generarea codului

Pentru obținerea codului se foloseşte algoritmul de traducere orientată de sintaxă corespunzător instrucțiunii de asignare simplă.

Intrare: un arbore etichetat reprezentând o instrucțiune de asignare care implică operatorii + şi *. Presupunem că numerele de nivel ale nodurilor sunt calculate cu formulele:

- $l(n) = 0$, dacă n este nod maximal (nu are descenți);
- $l(n) = \max\{l(n_i) \mid n_i \textit{ descendent al lui } n\} + 1$, dacă n este nod interior;

Ieşire: Codul în limbajul de asamblare care realizează asignarea

Metoda: Execută paşii 1° şi 2° pentru toate nodurile de nivel 0, apoi pentru cele de nivel 1,2, etc., până când toate nodurile au fost atinse

1^0 Presupunem că n este un nod terminal cu eticheta $< id >_j$.

(ii) Dacă în tabelul de identificatori, la poziția j este o variabilă, atunci $C(n)$ este numele variabilei;

(ii) Dacă în tabelul de identificatori, la poziția j este constnta k, atunci $C(n)$ este $= k$;

2^0 Dacă n este un nod terminal (maximal) cu eticheta =, * sau +, atunci $C(n)$ este secvența vidă;

3^0 Dacă n este un nod de tipul (a) şi descendenții săi sunt n_1, n_2, n_3 atunci $C(n)$ este: LOAD $C(n_3)$

STORE $C(n_1)$

4^0 Dacă n este un nod de tipul (b) şi descendenții săi sunt n_1, n_2, n_3, atunci $C(n)$ este:

$C(n_3)$
STORE $\$l(n)$
OAD $C(n_1)$
ADD $\$l(n)$

5^0 Dacă n este un nod de tipul (c) şi descendenţii săi sunt n_1, n_2, n_3, atunci $C(n)$ este:

$C(n_3)$
STORE $\$l(n)$
LOAD $C(n_1)$
MPY $\$l(n)$

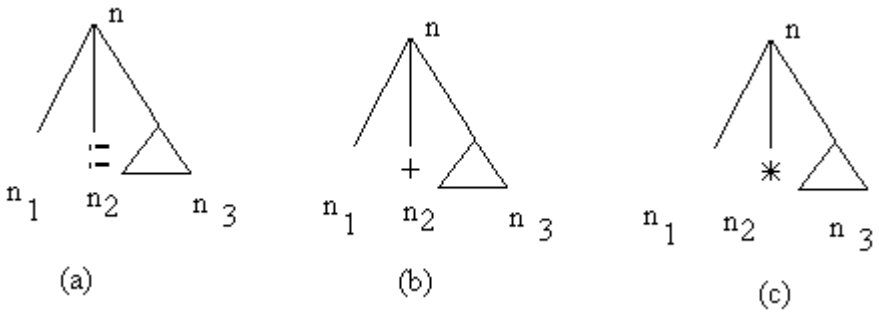

Fig. 2

Pentru arborele sintactic din exemplu se obține următorul cod:

n_1 :
D
STORE $1
LOAD C
MPY $1

n_2 :
B
STORE $2
LOAD D
STORE $1
LOAD C
MPY $1
ADD $2

n_3:
E
STORE $3
LOAD B
STORE $2
LOAD D
STORE $1
LOAD C
MPY $1
ADD $2
ADD $3

n_4:
E
STORE $3

n_5:
LOAD E
STORE $3

```
LOAD  B                    LOAD  B
STORE $2                   STORE $2
LOAD  D                    LOAD  D
STORE $1                   STORE $1
LOAD  C                    LOAD  C
MPY   $1                   MPY   $1
ADD   $2                   ADD   $2
ADD   $3                   ADD   $3
STORE $4                   STORE $4
LOAD  F                    LOAD  F
MPY   $4                   MPY   $4
                           STORE A
```

5. Optimizarea codului

Se utilizează regulile:

(a) Deoarece + este un operator comutativ, o secvență de instrucțiuni

```
LOAD α
ADD β
```

poate fi înlocuită cu secvența

```
LOAD β
ADD α
```

(b) Deoarece * este un operator comutativ, o secvență de instrucțiuni

```
LOAD α
MPY β
```

poate fi înlocuită cu secvența

```
LOAD β
MPY α
```

(c) Secvența de forma:

```
STORE α
LOAD α
```

poate fi ştearsă dacă α nu mai apare mai departe în prog ram sau dacă α este memorat înainte de a fi utilizat din nou.

(d) Secvența de forma:

```
LOAD α
STORE β
```

Poate fi ştearsă dacă est urmată de o nouă instrucțiune LOAD, verificând că nu există transfer la STORE β, şi că secvența menționată de β este înlocuită cu α, până când apare un nou STORE β.

Vom optimiza codul obținut în exemplu:

```
LOAD  E                    LOAD  E
```

```
STORE $3                          STORE $3
LOAD  B                           LOAD  B
STORE $2                          STORE $2
LOAD  D                           LOAD  D
STORE $1            ⇒             STORE $1          ⇒
LOAD  C                           LOAD  C
MPY   $1                          MPY   $1
ADD   $2                          ADD   $2
ADD   $3                          ADD   $3
STORE $4                          STORE $4
LOAD  F                           LOAD  $4     regula (c)
MPY   $4    regula (b)            MPY   F
STORE A                           STORE A
```

```
LOAD  E                           LOAD  E
STORE $3                          STORE $3
LOAD  B                           LOAD  B
STORE $2                          STORE $2
LOAD  D             ⇒             LOAD  D           ⇒
STORE $1                          STORE $1
LOAD  C                           LOAD  $1     regula (c)
MPY   $1    regula (b)            MPY   C
ADD   $2                          ADD   $2
ADD   $3                          ADD   $3
MPY   F                           MPY   F
STORE A                           STORE A
```

```
LOAD  E     regula (d)            LOAD  B      regula (d)
STORE $3                          STORE $2
LOAD  B                           LOAD  D
STORE $2                          MPY   C
LOAD  D             ⇒             LOAD  D           ⇒
M(PY  C                           ADD   $2
ADD   $2                          ADD   E
ADD   $3                          MPY   F
MPY   F                           STORE A
STORE A
```

Codul optimizat este:

```
LOAD  D
```

```
MPY  C
ADD  B
ADD  E
MPY  F
STORE  A
```

Efectul acestei secvențe de instrucțiuni este prezentat în continuare:

$Ac \leftarrow D$

$Ac \leftarrow D * C$

$Ac \leftarrow D * C + B$

$Ac \leftarrow D * C + B + E$

$Ac \leftarrow (D * C + B + E) * F$

$A \leftarrow (D * C + B + E) * F$

Notația *Ac* simbolizează registrul acumulator.

TEMĂ : 1.Să se traducă instrucțiunea

$$VAL := PROD * (SR + PQ) * 5$$

conform modelului prezentat anterior.

2.Scrieți un program pentru generarea codului corespunzător instrucțiunii de asignare simplă. Programul primeşte la intrare arborele sintactic şi datele de ieşire reprezintă codul neoptimizat.

3.Scrieți un program care primeşte la intrare codul neoptimizat al unei instrucțiuni de asignare simplă (având doar operatorii + şi *), iar la ieşire realizează codul optimizat, folosind cele patru reguli prezentate anterior.