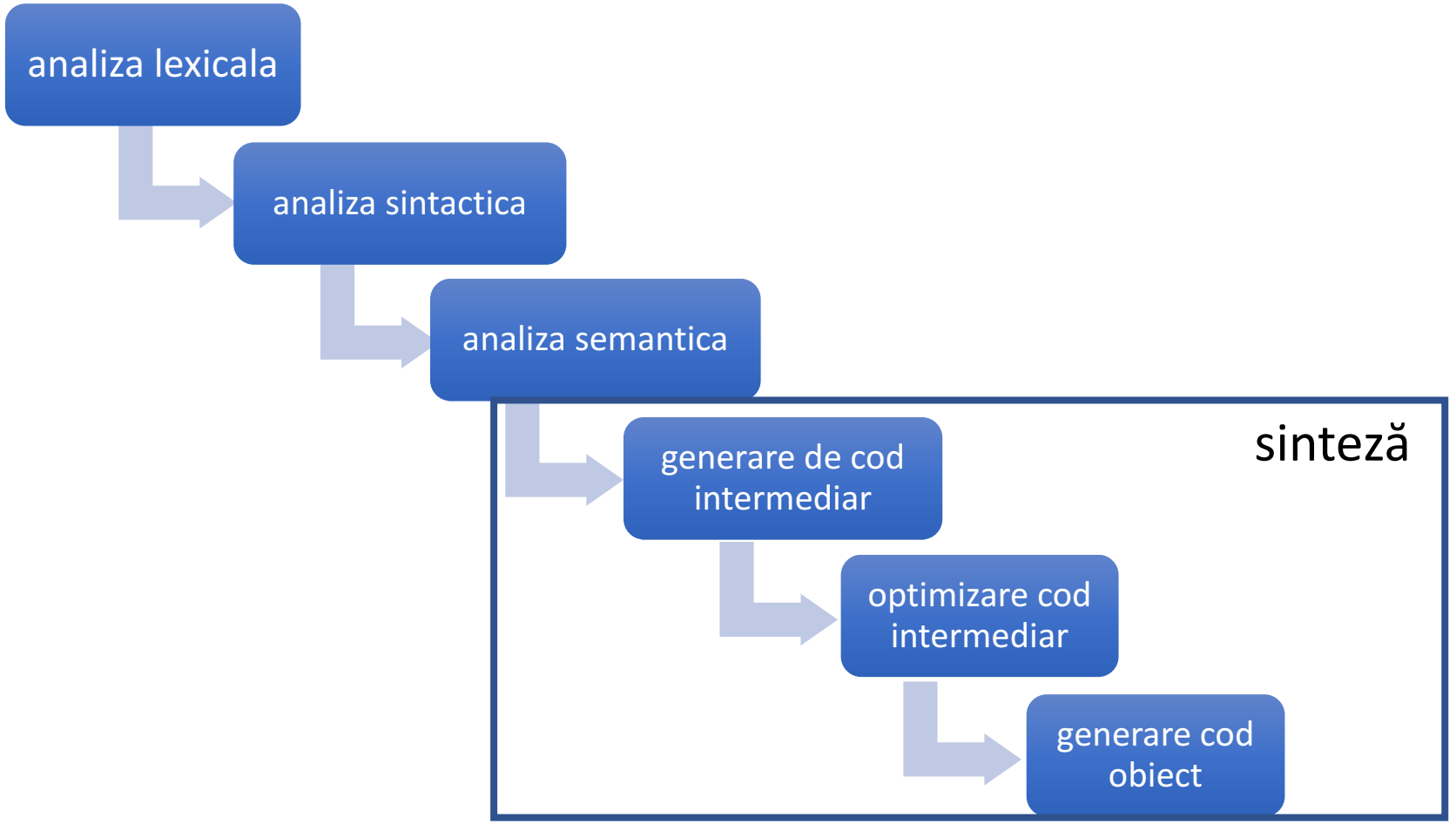
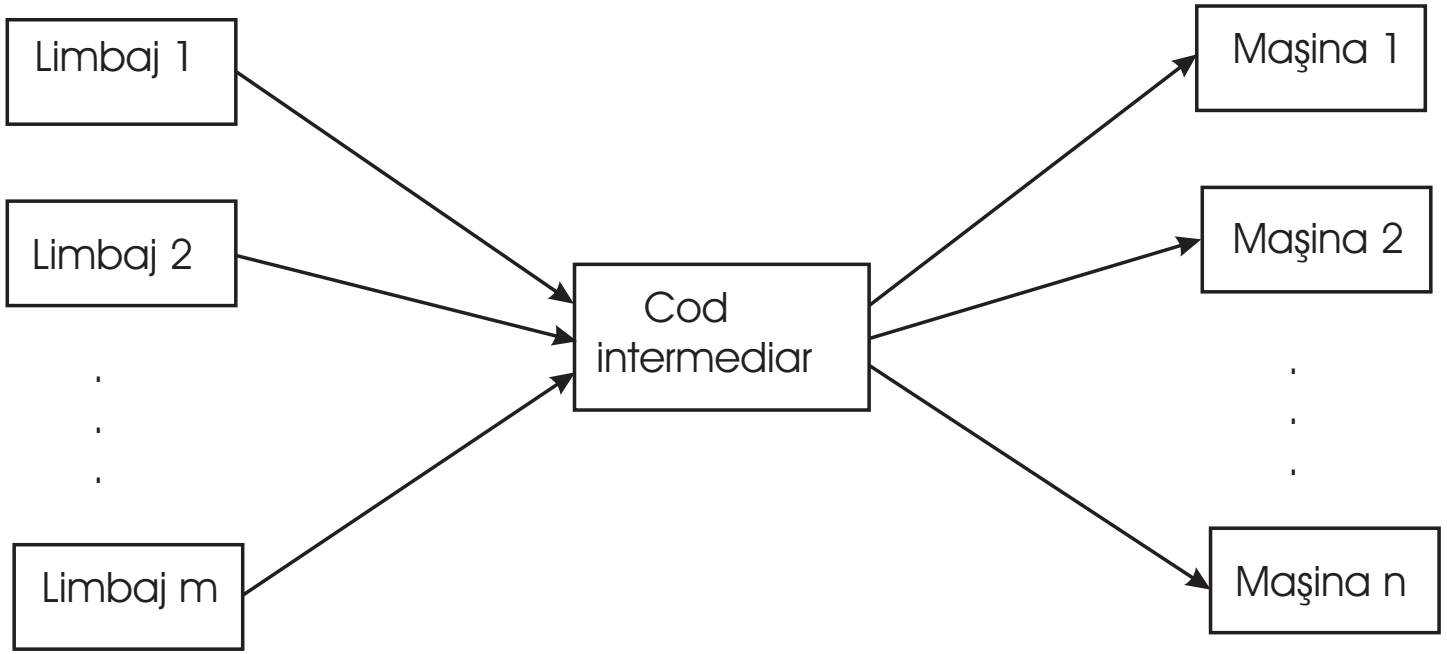
Curs 12

Fazele compilării



Generare de cod intermediar



Forme pentru codul intermediar

- Java bytecode – limbaj sursă: Java
  – limbaj mașină (dif. platforme) JVM
- MSIL (Microsoft Intermediate Language)
  – limbaje sursă: C#, VB, etc.
  – limbaj mașină (dif. platforme)
- GNU RTL (Register Transfer Language)
  – limbaj sursă: C, C++, Pascal, Fortran etc.
  – limbaj mașină (dif. platforme)

Reprezentări pentru codul intermediar

- Arbore atribuit: codul intermediar se generează în momentul analizei semantice
- Forma poloneză postfixată:
  - Nu conține paranteze
  - Operatorii apar în ordinea în care se execută
  - Ex.: MSIL

Exp = a + b * c	fpp = abc*+
Exp = a * b + c	fpp = ab*c+
Exp = a * (b + c)	fpp = abc+*

- Cod cu 3 adrese

Cod cu 3 adrese

- = secvență de instrucțiuni cu un format simplu, foarte apropiat de codul obiect, cu următoarea formă generală:

< rezultat >=< arg1 >< op >< arg2 >

Reprezentare:

- Cvadruple
- Triplete
- Triplete indirecte

- Cvadruple:

< op > < arg1 > < arg2 > < rezultat >

- Triplete:

< op > < arg1 > < arg2 >

(se consideră că tripletul memorează rezultatul)

Cazuri speciale:

1. Expresii cu operatori unari: **< rezultat >=< op >< arg2 >**
2. Atribuire de forma a := b atunci instrucțiunea cu trei adrese va fi **a = b** (lipsesc operatorul și al doilea operand)
3. Salt necondiționat: instrucțiunea va avea forma **goto L**, unde L este eticheta unei instrucțiuni din codul cu trei adrese.
4. Salt condiționat: if c goto L: dacă c este evaluat la adevărat atunci salt necondiționat la instrucțiunea cu eticheta L, altfel (dacă c este evaluat la fals) se execută instrucțiunea imediat următoare din codul cu trei adrese.
5. Apel de funcție p(x1, x2, ..., xn) – secvență de instrucțiuni: **param x1, param x2 , param xn, call p, n**
6. Variabile indexate: < arg1 >,< arg2 >,< rezultat > pot fi elemente de tablou, de forma **a[i]**
7. Modul de adresare al unei variabile (prin adresă sau referință): **&x, *x**

Exemplu: b*b–4*a*c

op	arg1	arg2	rez
*	b	b	t1
*	4	a	t2
*	t2	c	t3
-	t1	t3	t4

nr	op	arg1	arg2
(1)	*	b	b
(2)	*	4	a
(3)	*	(2)	c
(4)	-	(1)	(3)

Optimizarea codului intermediar

- Optimizări locale:
  - Realizarea unor calcule la compilare – valori constante
  - Eliminarea calculelor redundante
  - Eliminarea codului inaccesibil – if…then...else...

- Optimizarea ciclurilor:
  - factorizarea invarianților de cicluri
  - reducerea puterii operațiilor

Eliminarea calculelor redundante

Exemplu:

D:=D+C*B
A:=D+C*B
C:=D+C*B

(1)	*	C	B
(2)	+	D	(1)
(3)	:=	(2)	D
(4)	*	C	B
(5)	+	D	(4)
(6)	:=	(5)	A
(7)	*	C	B
(8)	+	D	(7)
(9)	:=	(8)	C

Factorizarea invarianților de cicluri

```
for(i=0, i<=n,i++)
    { x=y+z;
    a[i]=i*x}
```

```
x=y+z;
for(i=0, i<=n,i++)
    { a[i]=i*x}
```

Reducerea puterii operațiilor

```
for(i=k, i<=n,i++)
      { t=i*v;
      . . .}
```

```
t1=k*v;
for(i=k, i<=n,i++)
      { t=t1;
      t1=t1+v;...}
```

Generare cod obiect

= translatarea instrucțiunilor codului intermediar în instrucțiuni ale codului obiect (limbaj mașină)

- Depinde de “mașină”: arhitectură și SO

2 aspecte:

- alocarea regiștrilor - modul în care sunt stocate și manipulate variabilele;

- selectarea instrucțiunilor – modul și ordinea în care se mapează instrucțiunile codului intermediar în instrucțiuni mașină

Calculator cu acumulator

- O mașină de stivă constă dintr-o stivă pentru stocarea și manipularea valorilor și 2 tipuri de instrucțiuni:
  - pentru mutarea sau copierea valorilor în și din capul stivei în memorie
  - pentru operații asupra elementelor din capului stivei, care funcționează astfel: operanzii se scot din stivă, se execută operația respectivă și se depune în stivă rezultatul
- Acumulator pentru a efectua operația
- stiva pentru stocare subexpresi și rezultat

Calculator cu regiștri

- Regiștri +
- Memorie

- Instrucțiuni:
  - LOAD v,R – încarcă valoarea **v** în registrul **R**
  - STORE R,v – pune în memorie valoarea **v** din registrul **R**
  - ADD R1,R2 – adună la valoarea din registrul **R1**, valoarea din registrul **R2** și memorează în **R1** rezultatul (valoarea inițială se pierde!)

Observații:

1. Un registru poate fi disponibil sau ocupat =>

   VAR(R) = mulțimea variabilelor a căror valoare se află în registrul R

2. fiecare variabilă locul (registru, stivă sau memorie) în care se află valoarea curentă a variabilei =>

   MEM(x)= mulțimea locațiilor în care se află valoarea variabilei x (va fi câmp în tabela de simboluri)

Exemplu: F := A * B – (C + B) * (A * B)

Cod intermediar	Cod obiect	VAR	MEM
		VAR(R0)={} VAR(R1)={}	
(1)T1:=A*B	LOAD A, RO MUL B, R0	VAR(R0)={T1}	MEM(T1)={R0}
(2)T2:=C+B	LOAD C, R1 ADD B,R1	VAR(R1)=T2	MEM(T2)={R1}
(3)T3:=T2*T1	MUL R0 ,R1	VAR(R1)={T3}	MEM(T2)={} MEM(T3)={R1}
(4)F:=T1-T3	SUB R1, R0 STORE R0,F	VAR(RO)={F} VAR(R1)={}	MEM(T1)={} MEM(F)={R0} MEM(F)={RO,F}