Generarea codului obiect

LFTC curs14

Generarea codului obiect

- Translatarea instructiunilor codului intermediar in instructiuni ale codului obiect
- Cod intermediar: variabile, instructiuni abstracte, nu exista o stiva de executie
- Cod masina:
  - Numar finit de registri
  - Variabilele stocate in memorie
  - Se foloseste o stiva de executie si un set de instructiuni de accesare a stivei
  - Setul de instructiuni corespunde limbajului obiect

Generarea codului obiect

- Aspecte importante:
  - Alocarea registrilor – modul in care sunt stocate si manipulate variabilele
  - Selectarea instructiunilor – modul si ordinea in care se mapeaza instructiunile codului intermediar in instructiuni masina
- Generarea codului intermediar depinde de tipul calculatorului si de SO
- Tehnicile de structurare a codului obiect se clasifica in functie de tipul calculatorului in:
  - Calculator cu un acumulator
  - Calculator cu registri generali.

Generarea codului obiect pentru calculatoare cu acumulator

- Se bazeaza pe modelul **masinii de stiva** – model simplu de evaluare ce foloseste o stiva de valori pentru rezultate intermediare
- O masina de stiva consta dintr-**o stiva** pentru stocarea si manipularea valorilor si **doua tipuri de instructiuni**:
  - Instructiuni pentru **mutarea sau copierea** valorilor in si din capul memoriei stiva
  - Instructiuni pentru **operatii** asupra elementelor din capul memoriei stiva: operanzii se scot din stiva, se executa operatia si se depune rezultatul in stiva
- **Acumulatorul** este folosit pentru a **efectua** operatia, stiva fiind dedicata stocarii subexpresiilor si a rezultatului.
  - Nu foloseste registri, stiva fiind manipulata prin doua referinte, una spre capul ei si una care refera inceputul zonei alocate stivei

Exemplu

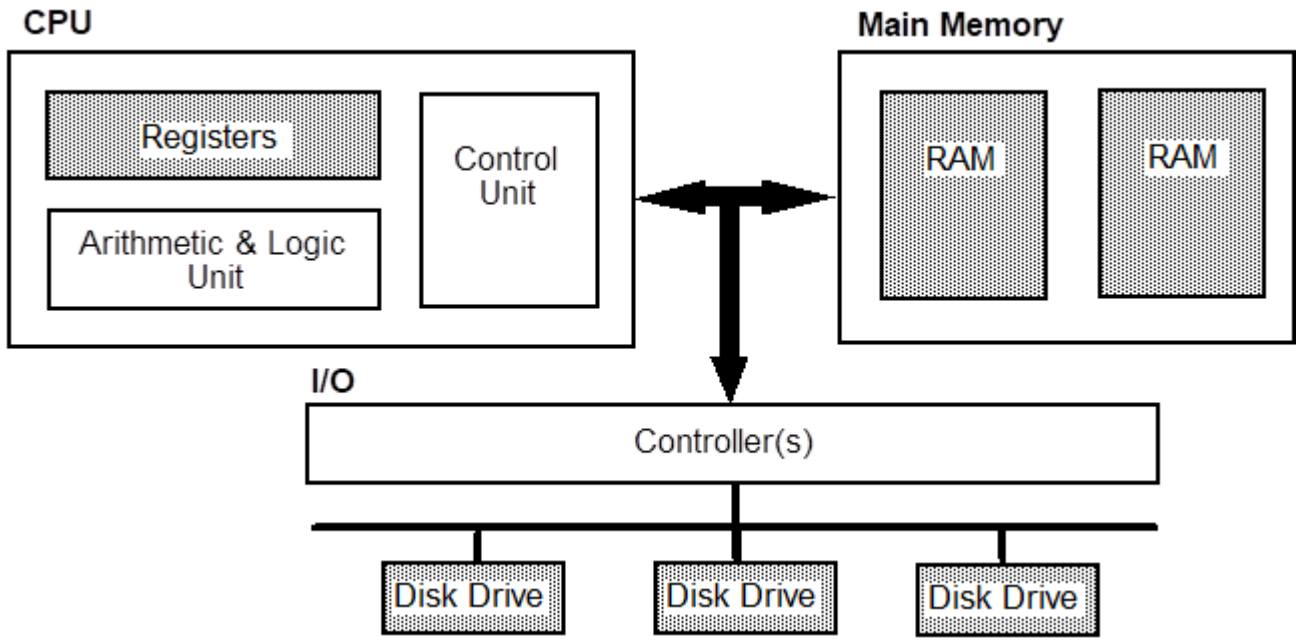
4*(3+2) fpp: 4 3 2 + *

Cod	Acc	Stiva
$acc \leftarrow 4$	*4*	$\emptyset$
$push\ acc$	*4*	*4*
$acc \leftarrow 3$	*3*	*4*
$push\ acc$	*3*	*3, 4*
$acc \leftarrow 2$	*2*	*3, 4*
$acc \leftarrow acc + cap$	*5*	*3, 4*
pop	*5*	*4*
$acc \leftarrow acc * cap$	*20*	*4*
pop	*20*	$\emptyset$

Generarea codului pentru calculatoare cu registri

- Modelul – **masina de registri** care consta intr-o zona pentru stocarea variabilelor, un set de registri pentru executia operatiilor si doua tipuri de instructiuni:
  - pentru incarcarea si salvarea valorilor intre memorie si registri
  - pentru efectuarea operatiilor din codul intermediar
- Formatul instructiunilor specifica locatia operanzilor, locatia rezultatului si locatia in care se incarca o anumita valoare
  - Incarcarea valorii v in registrul R: LOAD v, R
  - Salvarea in memorie a valorii v din registrul R: STORE R, v
- Pentru efectuarea operatiilor unul din registri care contine un operand va fi folosit pentru a retine rezultatul
- ADD R1, R2 are semnificatia de a aduna la R1 valoarea din R2

Generarea codului pentru calculatoare cu registri



Generarea codului pentru calculatoare cu registri

- Folosirea acestui model pentru calculatoare cu registri generali necesita stocarea unor informatii despre registri si variabile:
  - Un registru poate fi disponibil sau ocupat – **VAR(R)** – multimea variabilelor a caror valoare e stocata in R
  - Pentru fiecare variabila e necesar sa se cunoasca locul (registru, stiva, memorie) in care se afla valoarea curenta a sa – **MEM(X)** – multimea locatiilor in care se afla la un moment dat valoarea variabilei X. MEM va fi un camp in TS

Exemplu

F:=(C+B)*(A+B)-A*B

2 registri: R0, R1

Cod intermediar		Cod obiect	VAR	MEM
			VAR(R0)={} VAR(R1)={}	
(1)	T1:=C+B	LOAD C, R0 ADD B,R0	VAR(R0)={T1}	MEM(T1)={R0}
(2)	T2:=A+B	LOAD A, R1 ADD B, R1	VAR(R1)={T2}	MEM(T2)={R1}
(3)	T3:=(C+B)*(A+B)	MUL R0,R1	VAR(R1)={T3}	MEM(T2)={} MEM(T3)={R1}
(4)	T4:=A*B	LOAD A, R0 MUL B, R0	VAR(R0)={T4}	MEM(T1)={} MEM{T4}={R0}
(5)	T5:=T3-T4	SUB R1,R0 STORE R0,F	VAR(R0)={T5} VAR (R0)={}	MEM(T4)={} MEM(T5)={R0} MEM(F)={R0} MEM(F)={R0,F}

Alocarea registrilor

- Are ca scop selectarea variabilelor care trebuie incarcate in registri
- Numarul variabilelor dintr-un program e mare, numarul registrilor e limitat
- Un registru va fi obligat sa stocheze mai multe valori pe parcursul executiei unui program
- Pentru gestiunea variabilelor si alocarea optimala a registrilor se considera aparitia variabilelor in program si necesitatea utilizarii lor in intervalul imediat

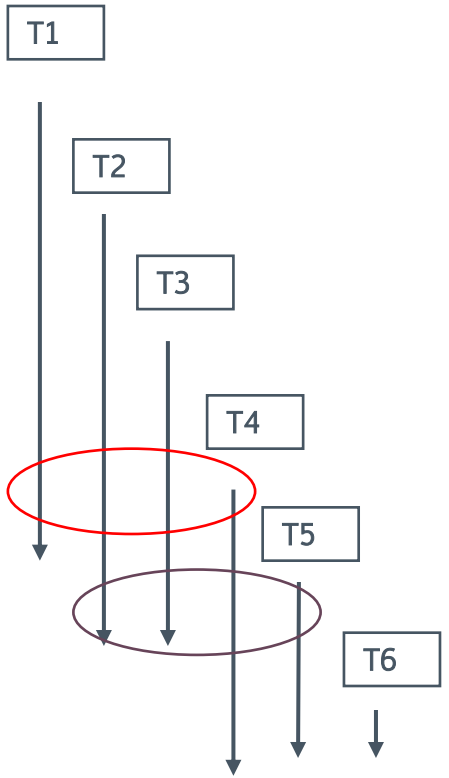
Alocarea registrilor

- O variabila *v* este **activa** intr-un punct de program *p* daca:
  - *v* a fost definita intr-o instructiune anterioara lui *p* pe orice drum;
  - *v* poate fi folosita de o instructiune *s* si exista un drum de la *p* la *s*
  - *v* nu e distrusa de la *p* la *s*.

- O variabila v este **activa** intre punctul p_i imediat urmator definirii sale si punctul p_j imediat dupa ultima ei folosire. Intervalul $[p_i,p_j]$ este **intervalul activ** al variabilei v.

Exemplu

1) T1=x+4
2) T2=T1*3
3) T3=T1-y
4) T4=T1*T2
5) T5=T2-T3
6) T6=T4+T5



Intervalele active ale variabilelor

Selectarea instructiunilor

- Optimizarea codului – imbunatatirea performantei compilatorului respectiv la:
  - Lungimea programului
  - Numarul de registri alocati
- Ordinea executiei instructiunilor poate avea impact asupra numarului de incarcari/descarcari ale registrilor

1. T1=b*b
2. T2=a*c
3. T3=4-T2
4. T4=T1-T3

b*b-(4-a*c)

Selectarea instructiunilor

b*b-(4-a*c)

VARIANTA 1 - SECVENTIALA

```
1)  LOAD b, R0
2)  MUL b, R0
3)  LOAD a, R1
4)  MUL c, R1
5)  STORE R0,T1
6)  LOAD 4, R0
7)  SUB R0, R1
8)  LOAD T1,R0
9)  SUB R0, R1
10) STORE R1,T4
```

VARIANTA 2 - OPTIMIZATA

```
1)  LOAD a, R0
2)  MUL c, R0
3)  LOAD 4, R1
4)  SUB R1, R0          (R0=4-a*c)
5)  LOAD b, R1
6)  MUL b, R1           (R1=b*b)
7)  SUB R1, R0          (R0=b*b-(4-a*c)
8)  STORE R0, T4
```

Exemplu Java bytecode

```
1 public static void main(String[] args) {
2     int a = 1;
3     int b = 2;
4     int c = a + b;
5 }
```

args	a	b	c

```
1  public static void main(java.lang.String[]);
2  descriptor: ([Ljava/lang/String;)V
3  flags: (0x0009) ACC_PUBLIC, ACC_STATIC
4  Code:
5  stack=2, locals=4, args_size=1
6  0: iconst_1
7  1: istore_1
8  2: iconst_2
9  3: istore_2
10 4: iload_1
11 5: iload_2
12 6: iadd
13 7: istore_3
14 8: return
15 ...
```

Pune pe stiva constanta 1

Scoate operandul din varful stivei si il pune pe variabila locala de la pozitia 1, adica a

Pune pe stiva constanta 2

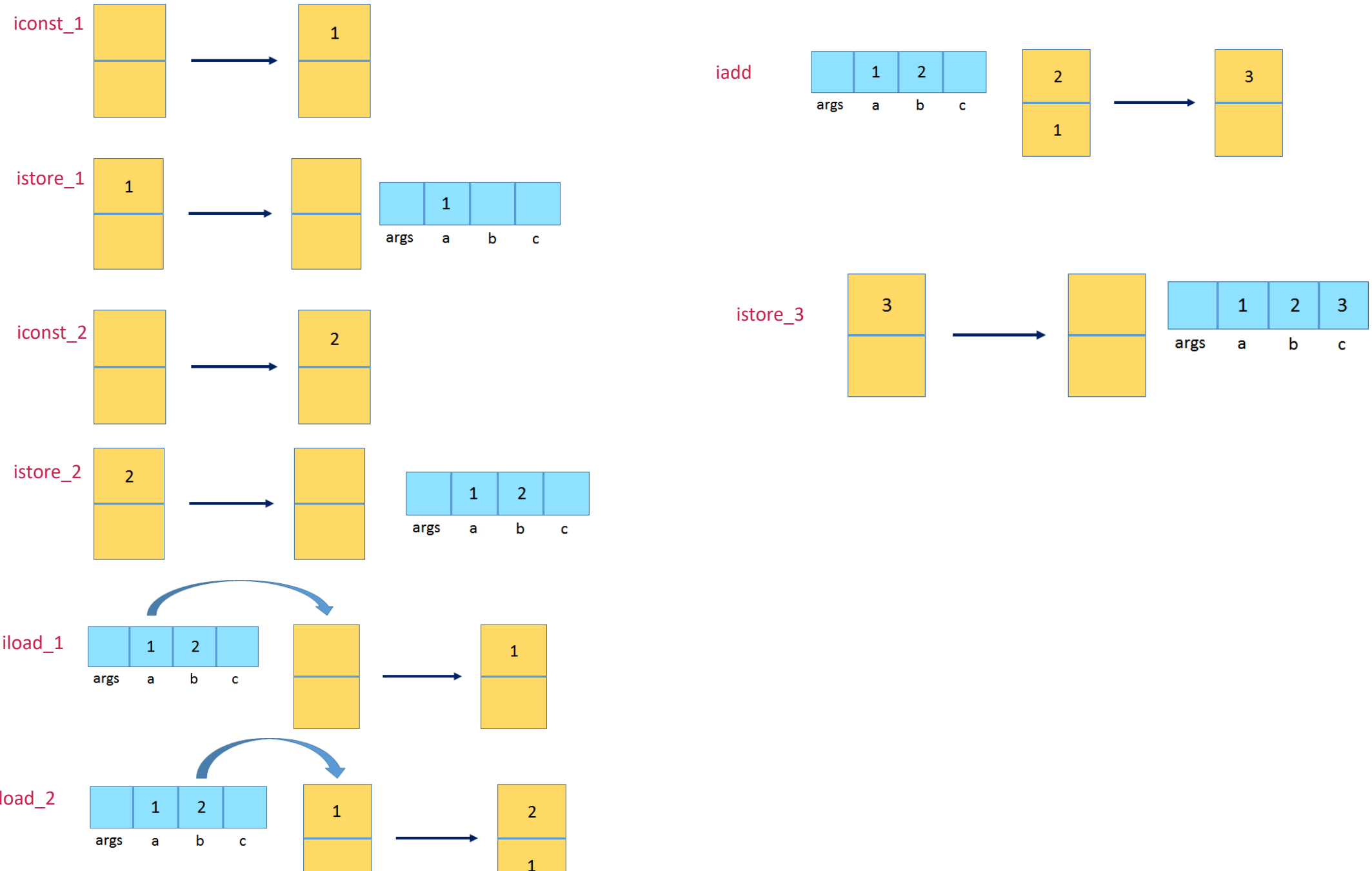
Scoate operandul din varful stivei si il pune pe variabila locala de la pozitia 2, adica b

Incarca variabila de la pozitia 1 (a) si o adauga in stiva

Incarca variabila de la pozitia 2 (b) si o adauga in stiva

Scoate cele doua valori din stiva, le aduna si pune rezultatul in stiva

Scoate operandul din varful stivei si il pune pe variabila locala de la pozitia 3, adica c


iconst_1
1
istore_1
1
1
args a b c
iconst_2
2
istore_2
2
1 2
args a b c
iload_1
1 2
args a b c
1
iload_2
1 2
args a b c
1
2
1
iadd
1 2
args a b c
2
1
3
istore_3
3
1 2 3
args a b c

Tratarea erorilor

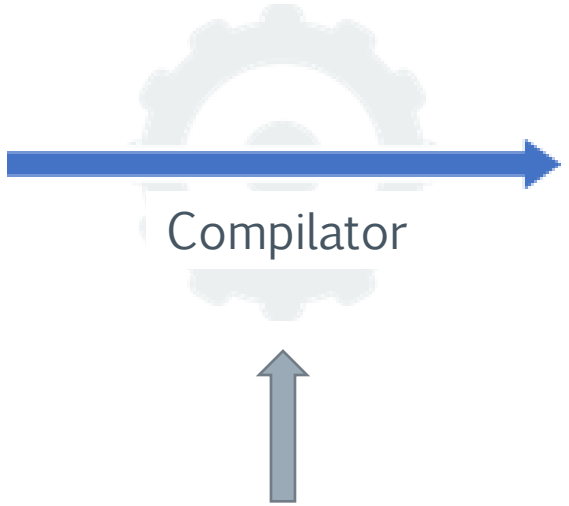
- Detectie
- Diagnostic
- Corectare

- Erori:
  - Lexicale
  - Sintactice
  - Semantice

Compilator

```
#include <stdio.h>

int main()
{
  printf(
       "Codeforwin");
  return 0;
}
```



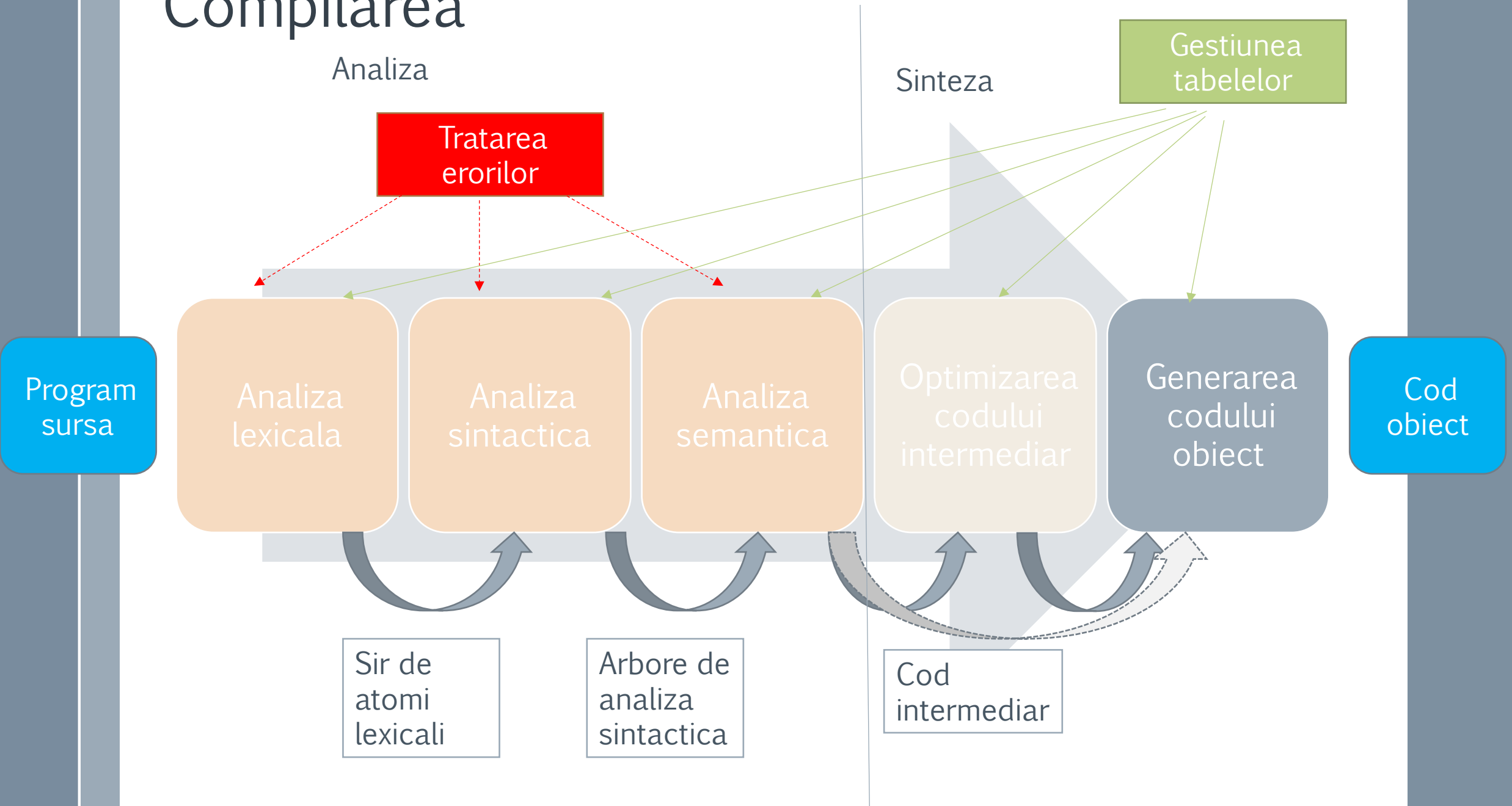

```
10101011100011001000
11110001111000110000
10101000111000101000
10110111010011111111
10001010101110010100
11101011010101010100
10101100010001110110
10101000111010100110
01010101000011011100
10111101011010111010
00011111111111000100
```

Cod sursa

Cod obiect

program

Compilarea


Analiza
Sinteza
Gestiunea tabelelor
Tratarea erorilor
Program sursa
Analiza lexicala
Analiza sintactica
Analiza semantica
Optimizarea codului intermediar
Generarea codului obiect
Cod obiect
Sir de atomi lexicali
Arbore de analiza sintactica
Cod intermediar

Reguli examen

› Cartea de identitate la voi

› Participarea la examen cu grupa in care sunteti inscrisi

Model examen

- Pe platforma Moodle (va rog sa va asigurati ca aveti laptop/telefon incarcate)
- Intrebarile legate de subiecte pe chat moodle prin mesaje private
- Model examen LFTC- MI: Model examen ianuarie 2025