Limbaje Formale și Compilatoare (LFC)
- Curs -

Ș.l.dr.ing. Octavian MACHIDON

Universitatea
Transilvania
din Brașov

Astăzi

- Aspecte administrative
- Introducere în compilatoare

Desfășurarea orelor

- Curs săptămânal:
  - Marți, 14:00 – 15:50, KB8
- Laborator la două saptămâni:
  - Marți, 12:00 – 13:50, KB2
  - Marți, 16:00 – 17:50, KB2
- Cadru didactic curs & laborator:
  - Octavian Machidon
  - Contact: octavian.machidon@unitbv.ro

Propunere

		MARTI	
Grupa	Sgr.	14,00-15,50	16,00-17,50
4LF771	A	LFC,C, KB8, Machidon_O_M	LFC, L, KB2, Machidon_O_M
	B		
			LFC, L, KB2, Machidon_O_M
4LF772	A	LFC,C, KB8, Machidon_O_M	
			LFC, L, KB2, Machidon_O_M

Despre disciplina LFC

- Abordează atât elemente teoretice cât și practice
- Necesită cunoștințe de programare
- În prima parte a semestrului, orele de laborator vor avea aspectul unor ore de seminar (activitate teoretică, la tablă)
- Spre finalul semestrului, laboratorul va presupune realizarea unor aplicații software (programare)

Resurse

- Notițele de curs
- Problemele prezentate și discutate la laborator (atât teoretice cât și practice)
- Îndrumar de laborator:
  - Limbaje Formale și Compilatoare – Îndrumar de laborator. Octavian Machidon. Editura Universității Transilvania, 2017
- Două cărți:
  - Introduction to Automata Theory, Languages, and Computation. Jeffrey Ullman and John Hopcroft
  - Compilers: Principles, Techniques, and Tools. Alfred Aho, Jeffrey Ullman, Monica S. Lam, and Ravi Sethi
    - Se găsesc online în format electronic
    - Nu sunt obligatorii

Evaluare

- Evaluare pe parcursul semestrului:
  - Examen parțial, din prima parte a cursului
  - Pe baza exercițiilor făcute la laborator
- Evaluare la finalul semestrului:
  - Evaluarea unor teme practice rezolvate la laborator
  - Examen scris (Parte teoretică + Parte practică – exerciții)
- Nota finală:
  - 30% $P_{teoretică}$ + 40% $P_{practică}$ + 20% $T_{laborator}$ + 10% Oficiu

Ce sunt compilatoarele?

- Compilatoarele „traduc” informația dintr-o reprezentare într-o alta
- De regulă prin informație înțelegem un program (cod sursă)
- Practic, un compilator poate fi asimilat cu un „traducător” dar:
  - Compilatoarele realizează transformarea codului de nivel înalt în cod de nivel jos (cod obiect)
  - În general prin translatare („traducere”) se înțelege transformarea informației menținând nivelul de abstractizare

Exemple

- Compilatoare (tipice):
  - gcc, javac
- Compilatoare (atipice):
  - latex (transformă un document în comenzi de imprimare DVI)
- Translatoare:
  - f2c: Fortran-to-C (ambele de nivel înalt)
  - latex2html: Latex-to-HTML (ambele documente)
  - Dvi2ps: DVI-to-PostScript (ambele de nivel jos)

De ce sunt necesare compilatoarele?

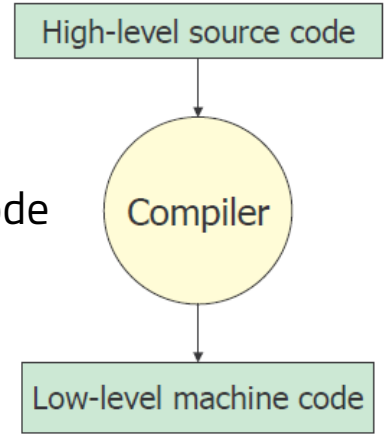
- Este dificilă scrierea, depanarea, executarea și înțelegerea programelor scrise în limbaj de asamblare:

```
section     .text
    global _start   ;must be declared for linker (ld)
_start:                 ;tells linker entry point
    mov     edx,len     ;message length
    mov     ecx,msg     ;message to write
    mov     ebx,1       ;file descriptor (stdout)
    mov     eax,4       ;system call number (sys_write)
    int     0x80        ;call kernel

    mov     eax,1       ;system call number (sys_exit)
    int     0x80        ;call kernel

section     .data
msg db 'Hello, world!', 0xa  ;string to be printed
len equ $ - msg     ;length of the string
```

- Există și excepții când este de preferat scrierea manuală a codului în limbaj de asamblare:
    - Scrierea de drivere

Ce face deci un compilator?

- Transformă codul sursă (nivel înalt) în cod de asamblare și cod mașină
- Codul sursă de nivel înalt:
  - Este optimizat pentru a putea fi citit și înțeles de om
  - Folosește denumiri (uneori sugestive) pentru variabile și metode
- Codul de asamblare și codul mașină:
  - Optimizat pentru hardware
  - Conține intstrucțiuni mașină (adrese hexazecimalede memorie și regiștrii)
  - Foarte greu de înțeles de om



Exemplu:

- Cod sursă:

```
int expr(int n)
{
   int d;
   d = 4 * n * n * (n + 1) * (n + 1);
   return d;
}
```

- Cod asm:

```
lda $30,-32($30)
stq $26,0($30)
stq $15,8($30)
bis $30,$30,$15
bis $16,$16,$1
stl $1,16($15)
lds $f1,16($15)
sts $f1,24($15)
ldl $5,24($15)
bis $5,$5,$2
s4addq $2,0,$3
ldl $4,16($15)
mull $4,$3,$2
ldl $3,16($15)
addq $3,1,$4
mull $2,$4,$2
ldl $3,16($15)
addq $3,1,$4
mull $2,$4,$2
stl $2,20($15)
ldl $0,20($15)
br $31,$33
$33:
bis $15,$15,$30
ldq $26,0($30)
ldq $15,8($30)
addq $30,32,$30
ret $31,($26),1
```

Cerințele unui compilator

- Eficiență:
  - Generarea de cod mașină care descrie aceleași operații de calcul ca și codul sursă
  - Există o transformare unică între cele 2?
  - Există un algoritm pentru o transformare ideală?
  - ! Compilatorul trebuie să permită optimizări prin care se gasesc variante mai bune (dpdv al lungimii codului, timpului de execuție, etc...)
- Corectitudine:
  - Trebuie executate exact aceleași calcule de codul mașină ca și de cel sursă
  - Compilatorul trebuie să asigure depanarea programelor, deci corectitudinea compilării trebuie să fie garantată!

Cum are loc „traducerea”?

- Proces complex, codul sursă și cel generat fiind complet diferite
- Proces etapizat:
  - Pași intermediari
  - La fiecare este folosită o anumită reprezentare specifică a codului intermediar

Structura unui compilator

1. Analiză lexicală
2. Parsare (Analiză sintactică)
3. Analiză semantică
4. Optimizare
5. Generare de cod

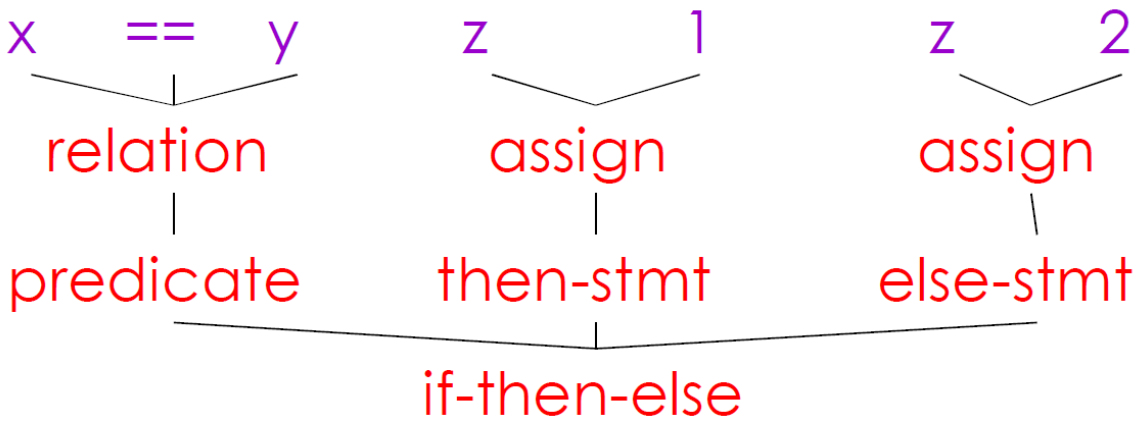
- Primele trei etape pot fi privite analog cu înțelegerea unui limbaj natural

Exemplu

- Analiza lexicală: recunoașterea cuvintelor

If|x|==|y| then|z|=| 1;| else| z|=| 2;|

- Parsarea (Analiza sintactică): recunoașterea structurii frazei/propoziției



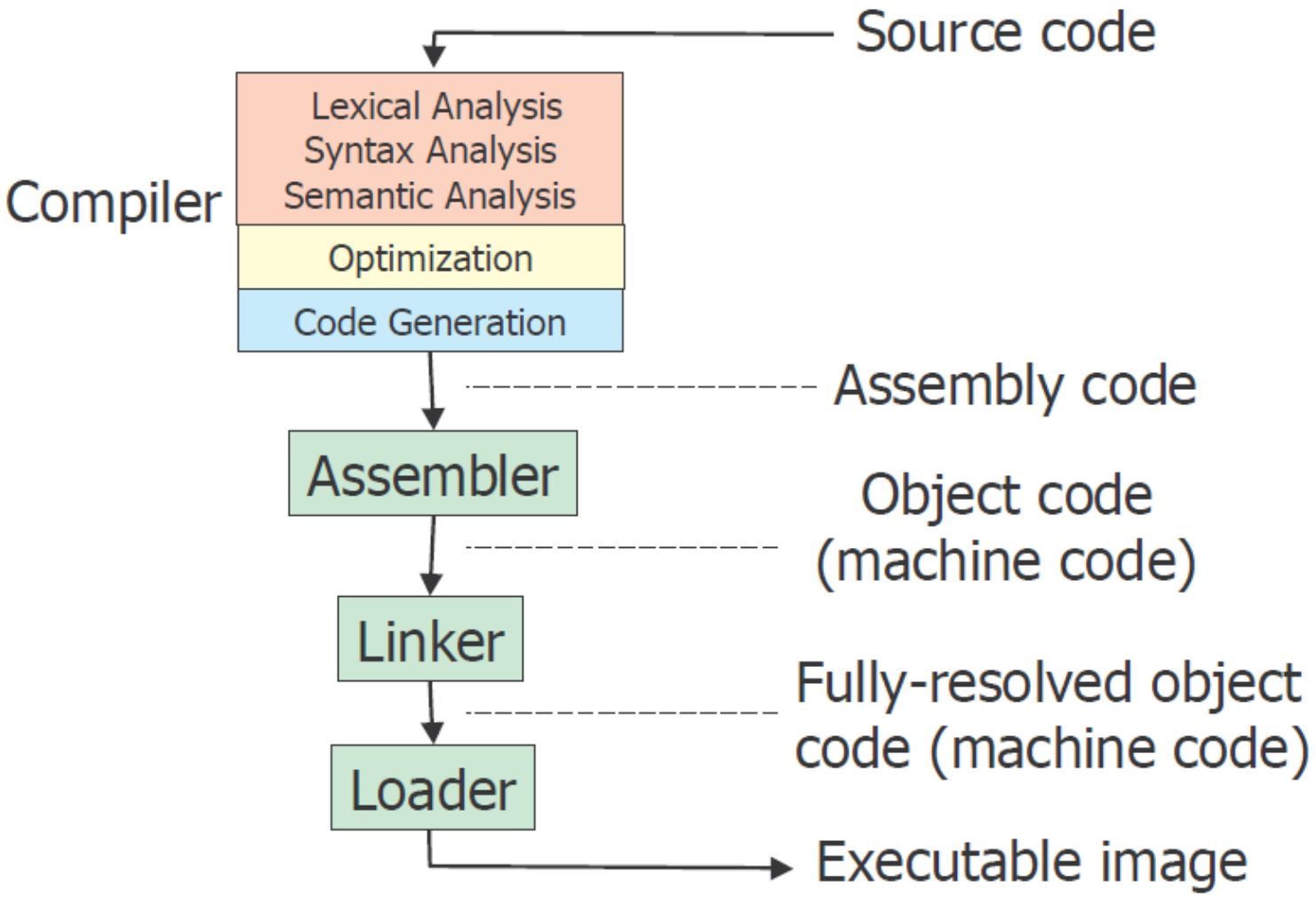
- Analiza semantică: recunoașterea sensului
  - În cazul compilatorului, există reguli care se aplică pentru a detecta și semnala ambiguități

Analiza semantică

- Ce afișează codul C++?

```
{
    int Jack = 3;
    {
        int Jack = 4;
        cout << Jack;
    }
}
```

O vedere de ansamblu...



Întrebări?