GENEROVÁNÍ KÓDU
STRUKTURA PŘEKLADAČE, JEHO ZADNÍ ČÁSTI (BACKENDU), VNITŘNÍ FORMY

2016 Jan Janoušek

MI-GEN

Základní informace

Přednášky:

doc. Ing. Jan Janoušek, Ph.D.

- Office: A 1224
- Email:
  Jan.Janousek@fit.cvut.cz

Cvičení:

Ing. Petr Maj

Doporučená literatura

- Aho, Lam, Sethi, Ullman: Compiler: Principles, Techniques and Tools (2nd ed.), 2010.

tzv. *Dragon book*

- Melichar, Češka, Ježek, Richta: Konstrukce překladačů, ČVUT, 2006.

- Muchnick: Advanced compiler design and implementation, Morgan Kaufman Publishers, 2009.

- Fischer, LeBlanc: Crafting a Compiler, 1995.

- Grune, Bal, Jacobs, Langendoen: Modern Compiler Design, 2000.

Hodnocení

- Podmínky zápočtu a zkoušky
  - Semestrální práce max. 30 b.
  - Zápočtový test max. 30 b.
  - Zkouška 40 b. (povinná ústní zkouška s právem veta +/- 5 b.)
  - Výsledná známka se řídí Klasifikačním řádem ČVUT.

Souvislosti s ostatními předměty

- Návaznost předmětů na FIT ČVUT:
  - BI-AAG: obecná teorie formálních jazyků, gramatik a automatů. Zavedení základních formalismů.
  - BI-PJP: úvodní kurs do teorie deterministické syntaktické analýzy, překladu a tvorby překladačů (jednoduchý front-end řízený LL analyzátorem, přímé generování cílového kódu průchodem AST).

  Metody nejen pro překladače, ale obecně pro zpracování strukturovaného textu.

  Použité formalismy a postupy v praxi v BI-PJP: regulární gramatika, konečný automat, bezkontextová gramatika, deterministická analýza metodou shora-dolů, překladová gramatika, atributová gramatika, atributované překlady.

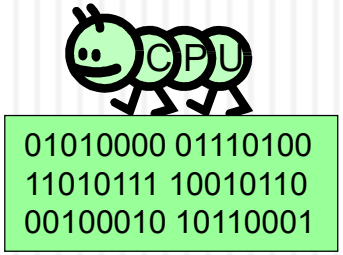
Souvislosti s ostatními předměty

Magisterské studium:

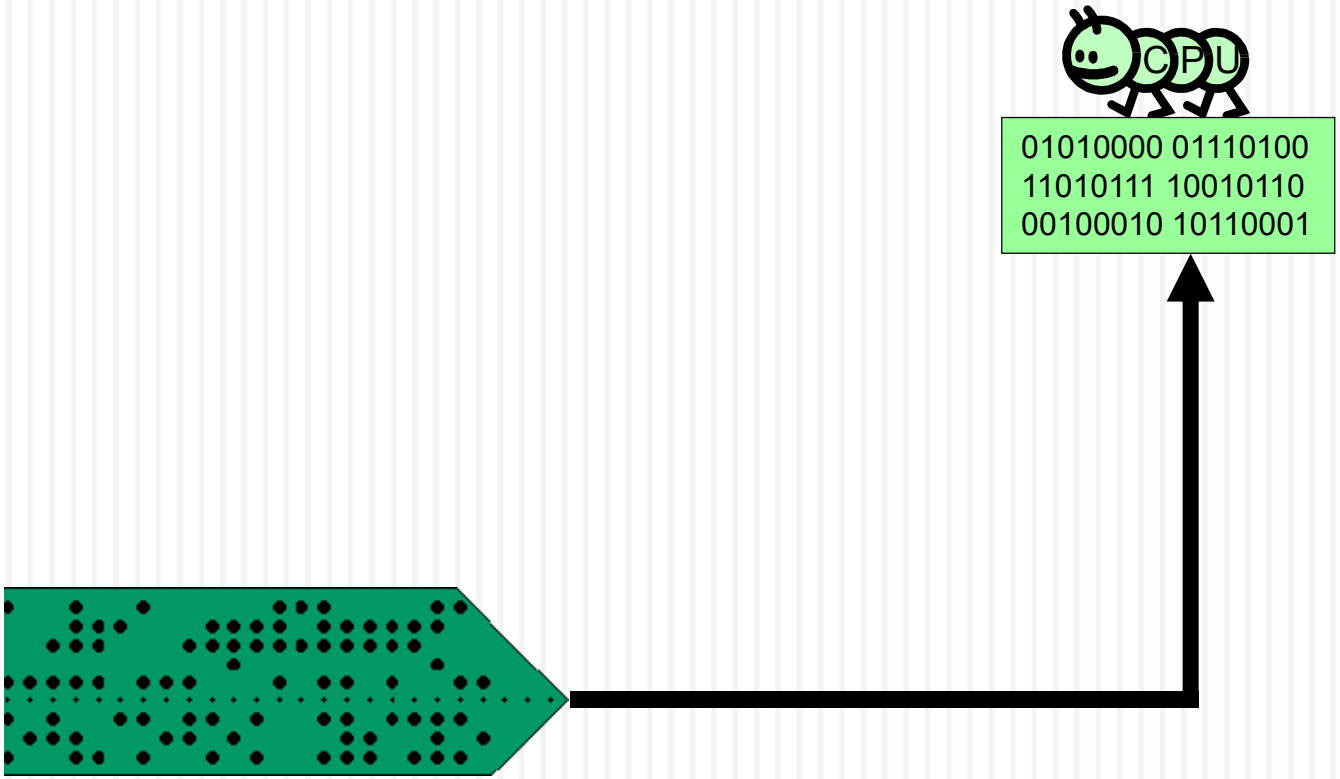
- MI-SYP: pokročilé techniky syntaktické analýzy a překladu, deterministická syntaktická analýza zdola nahoru (LR), paralelní syntaktická analýza.
- **MI-GEN:** vnitřní formy programu, zopakování funkcí frontendu: generování vnitřních forem, konstrukce zadní části překladače, tzv backend: generování kódu a optimalizace kódu.
- MI-RUN run-time prostředí v detailech

Použití překladače

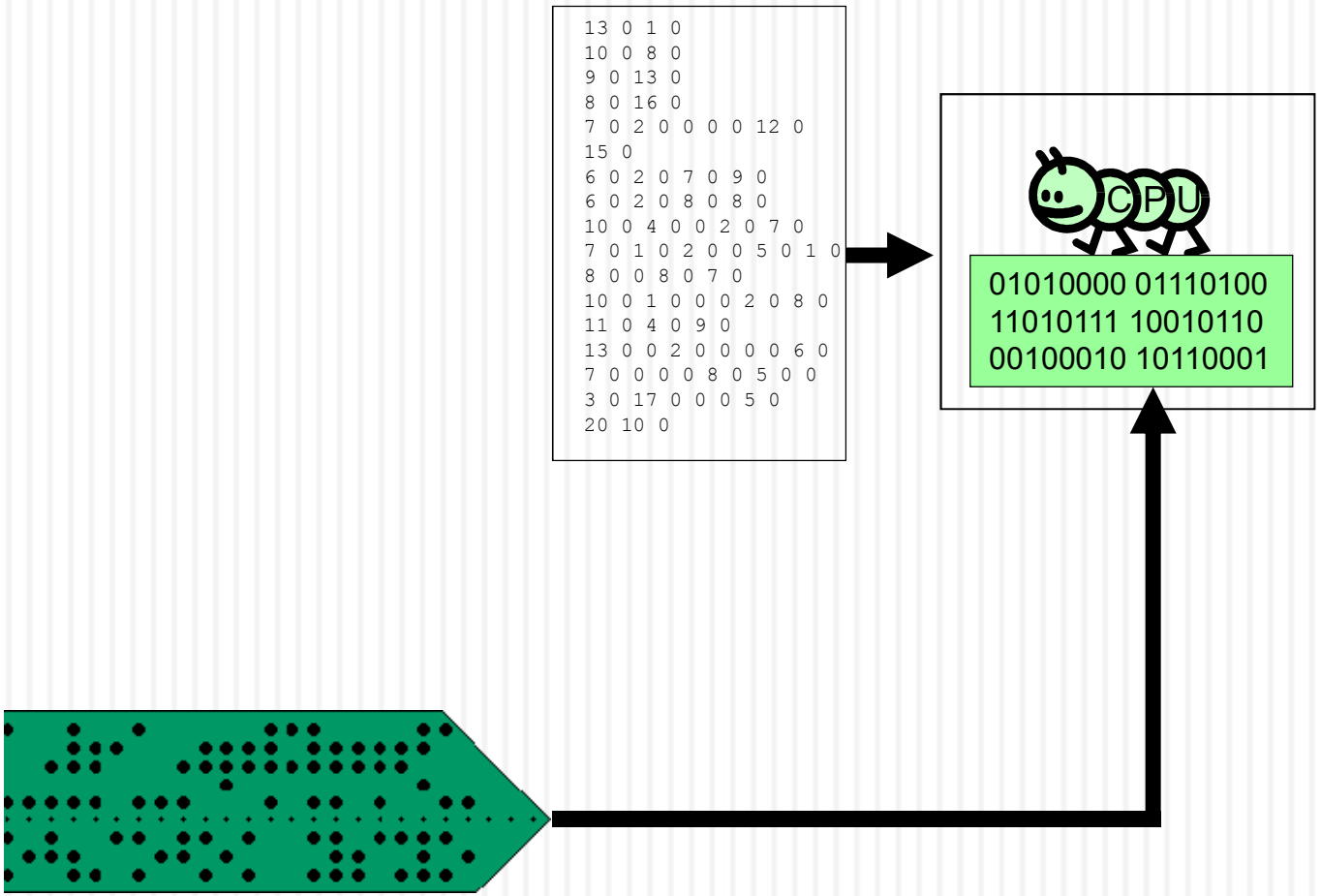
CPU rozumí pouze binárním kódu



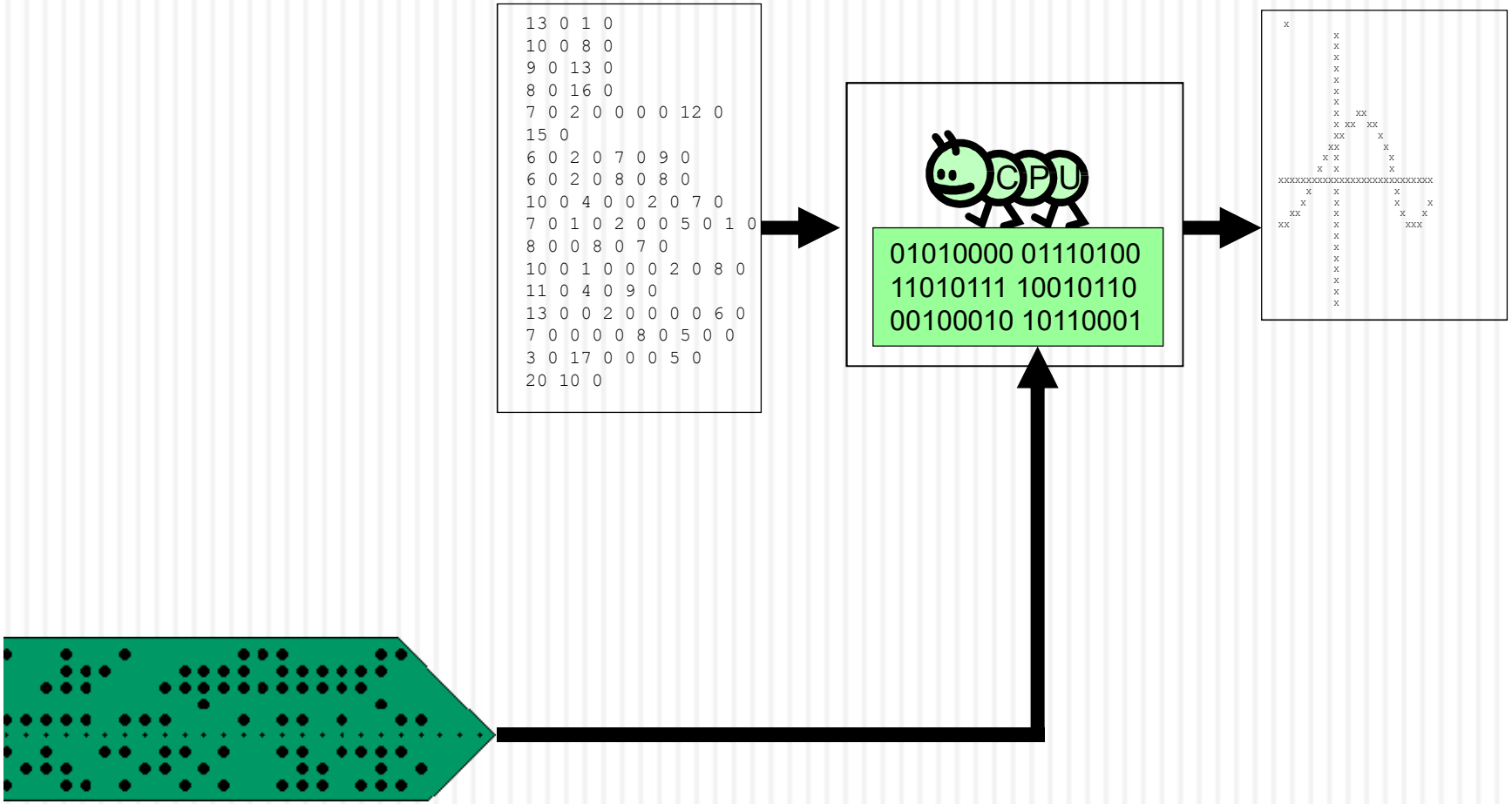
1945... – programování ve strojovém kódu



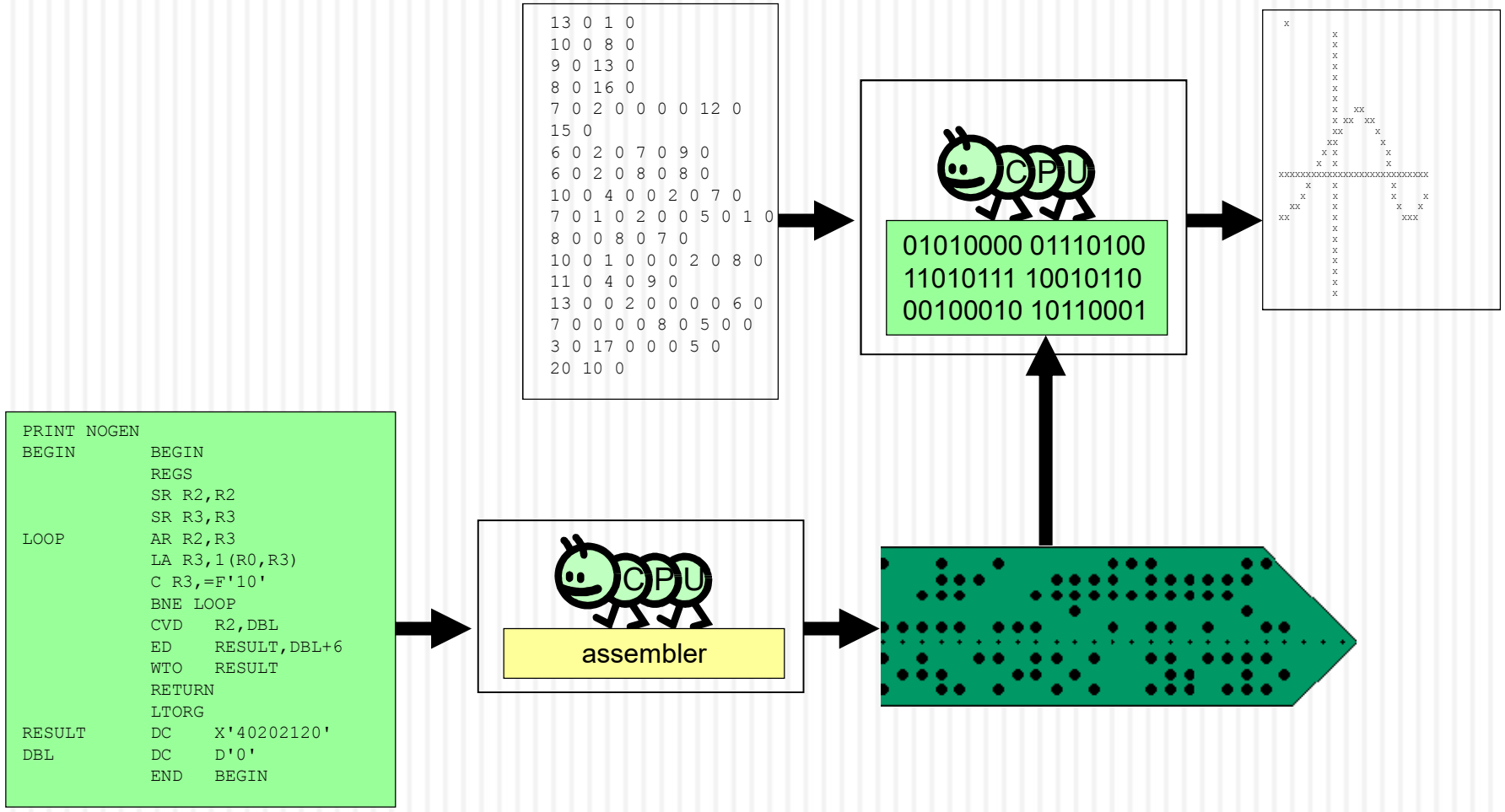
1945... – programování ve strojovém kódu



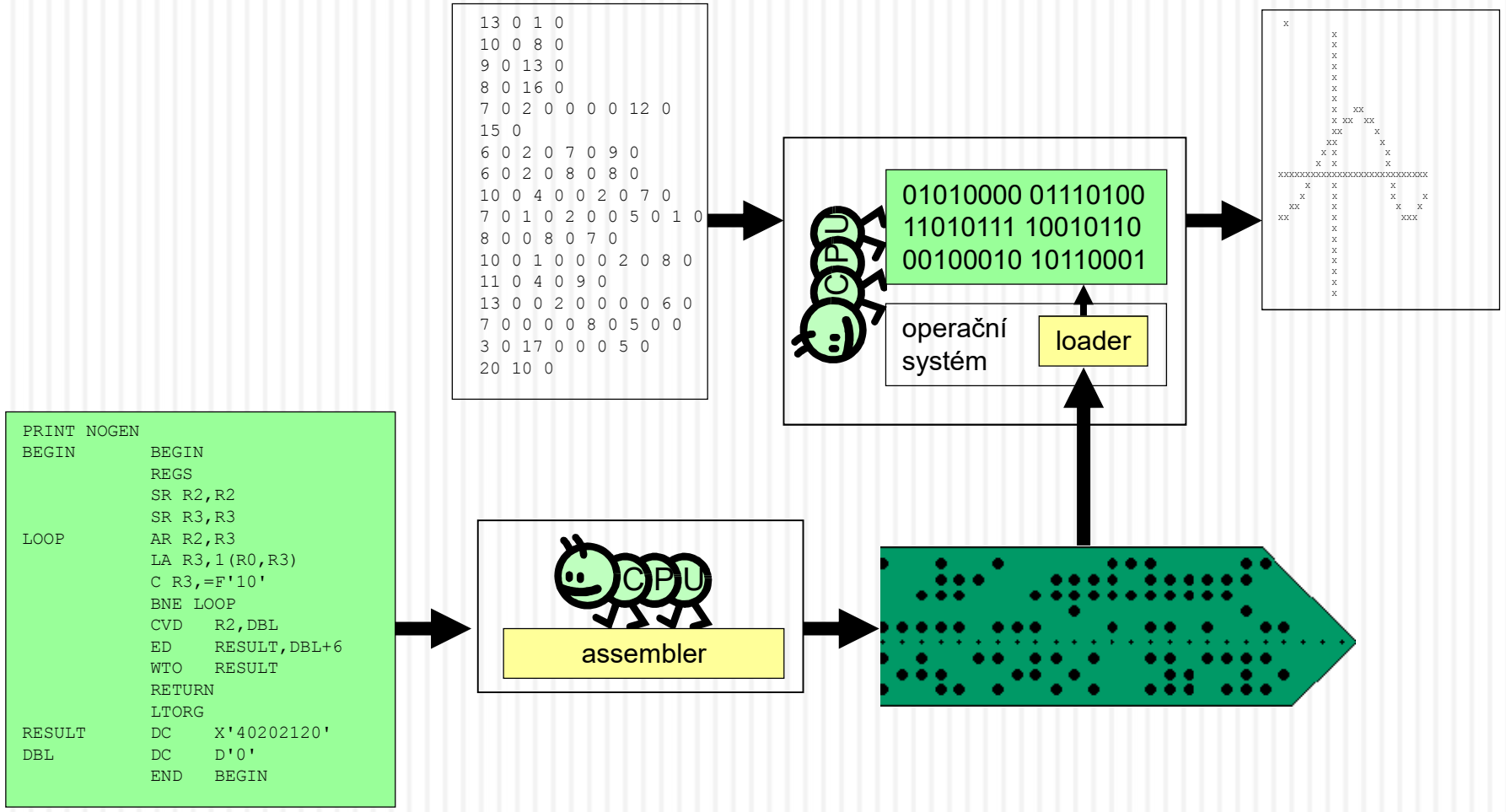
1945... – programování ve strojovém kódu



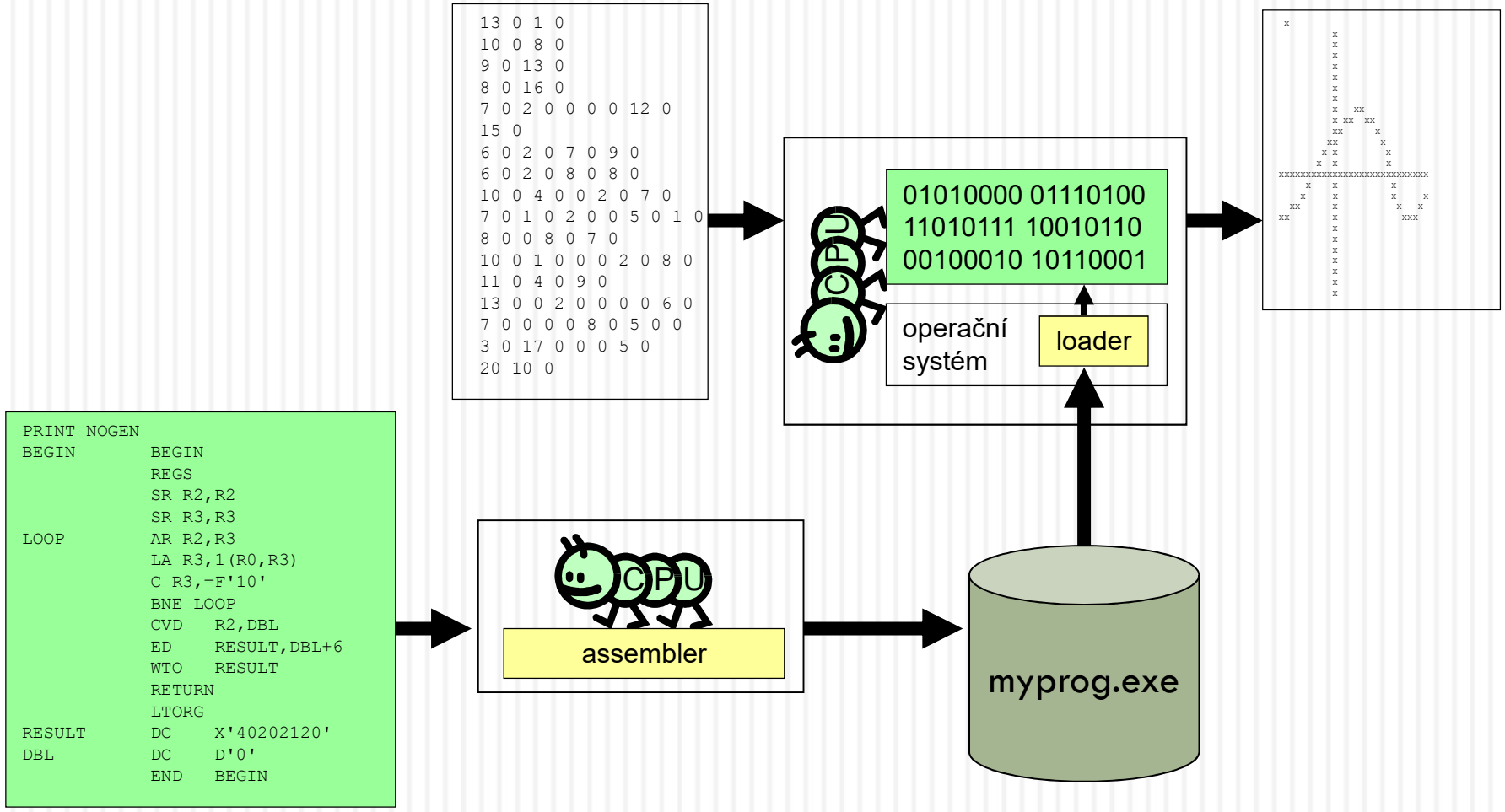
1950... – assembler



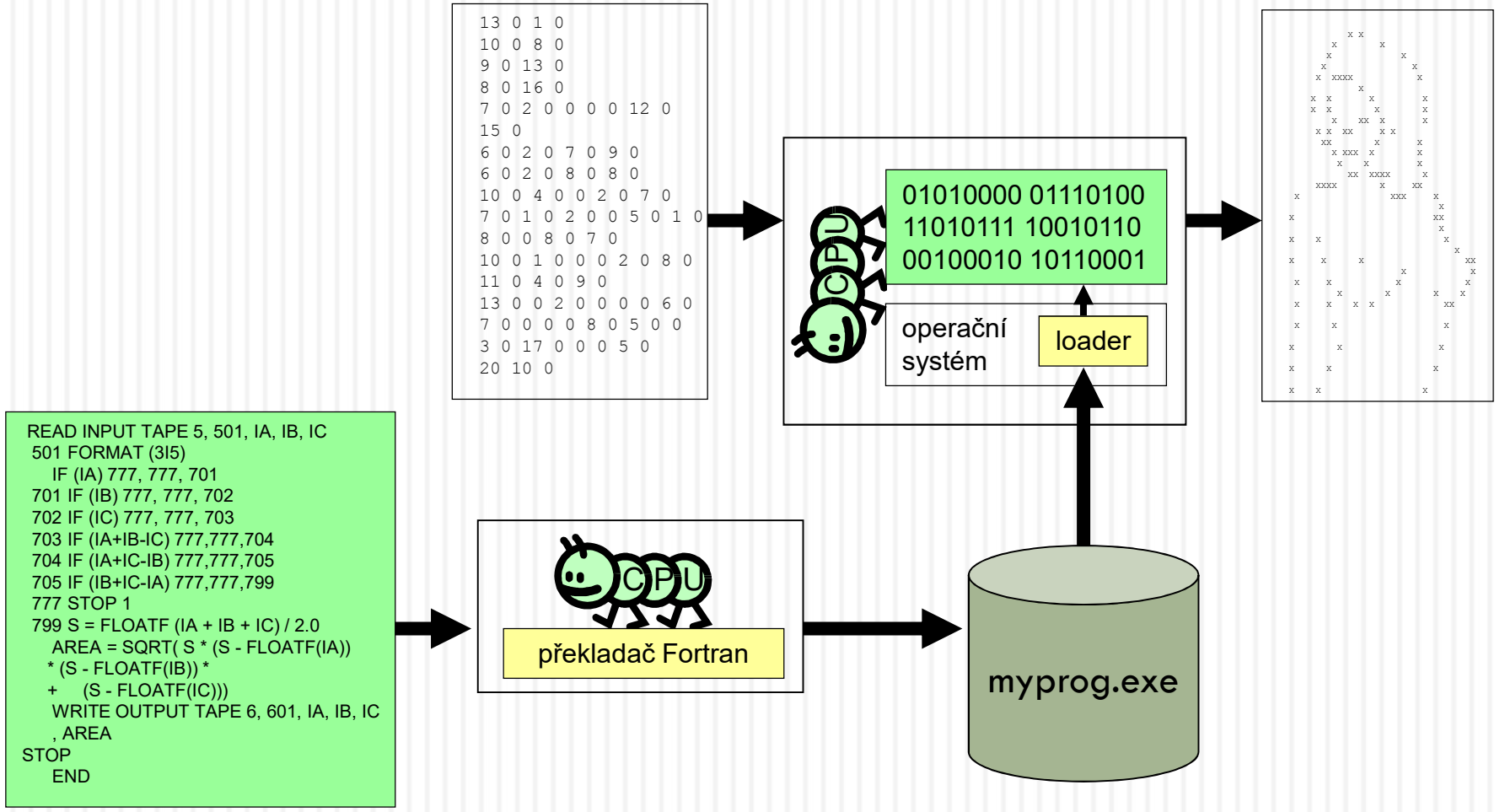
1950... – operační systém



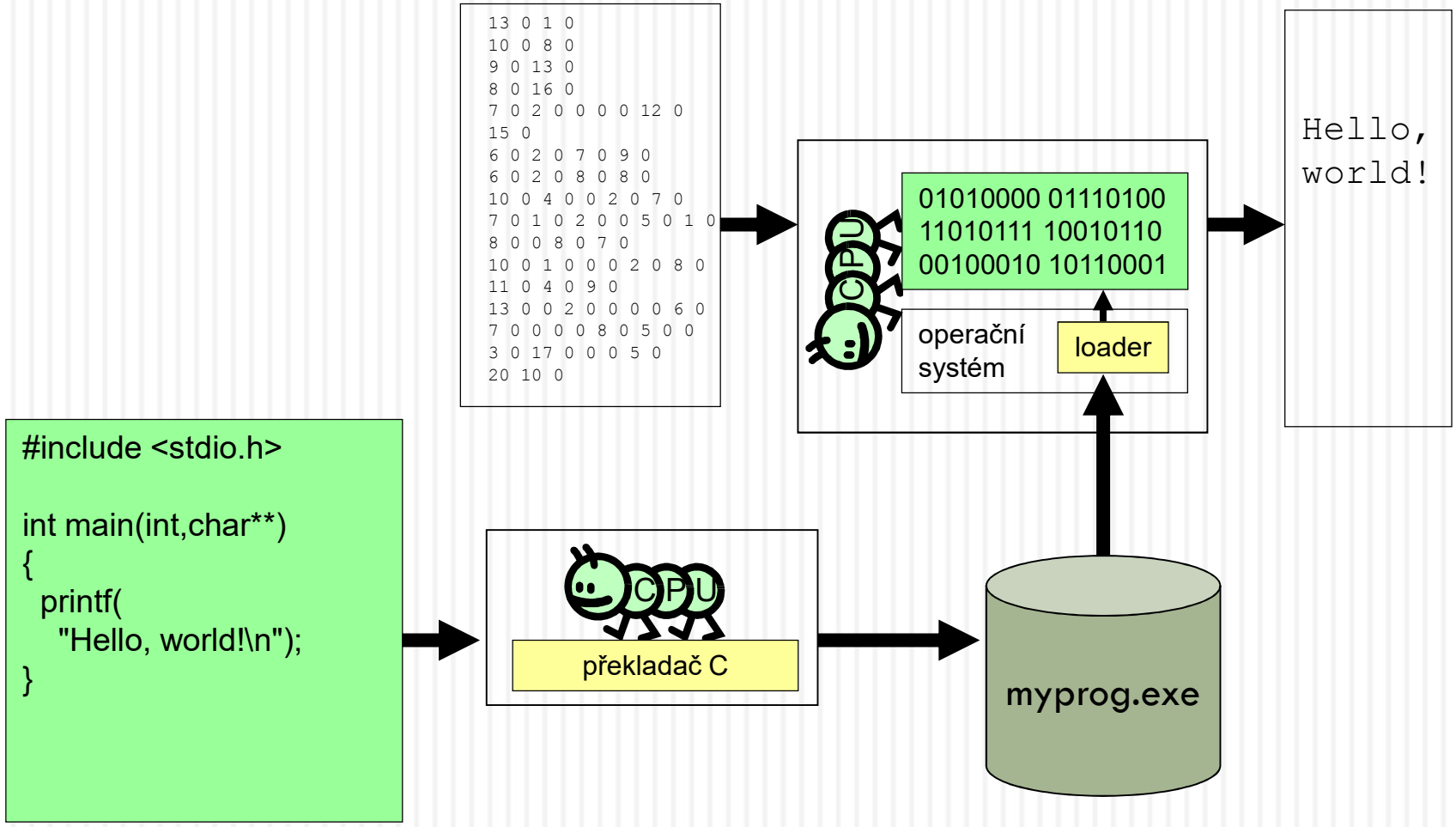
1950... – operační systém



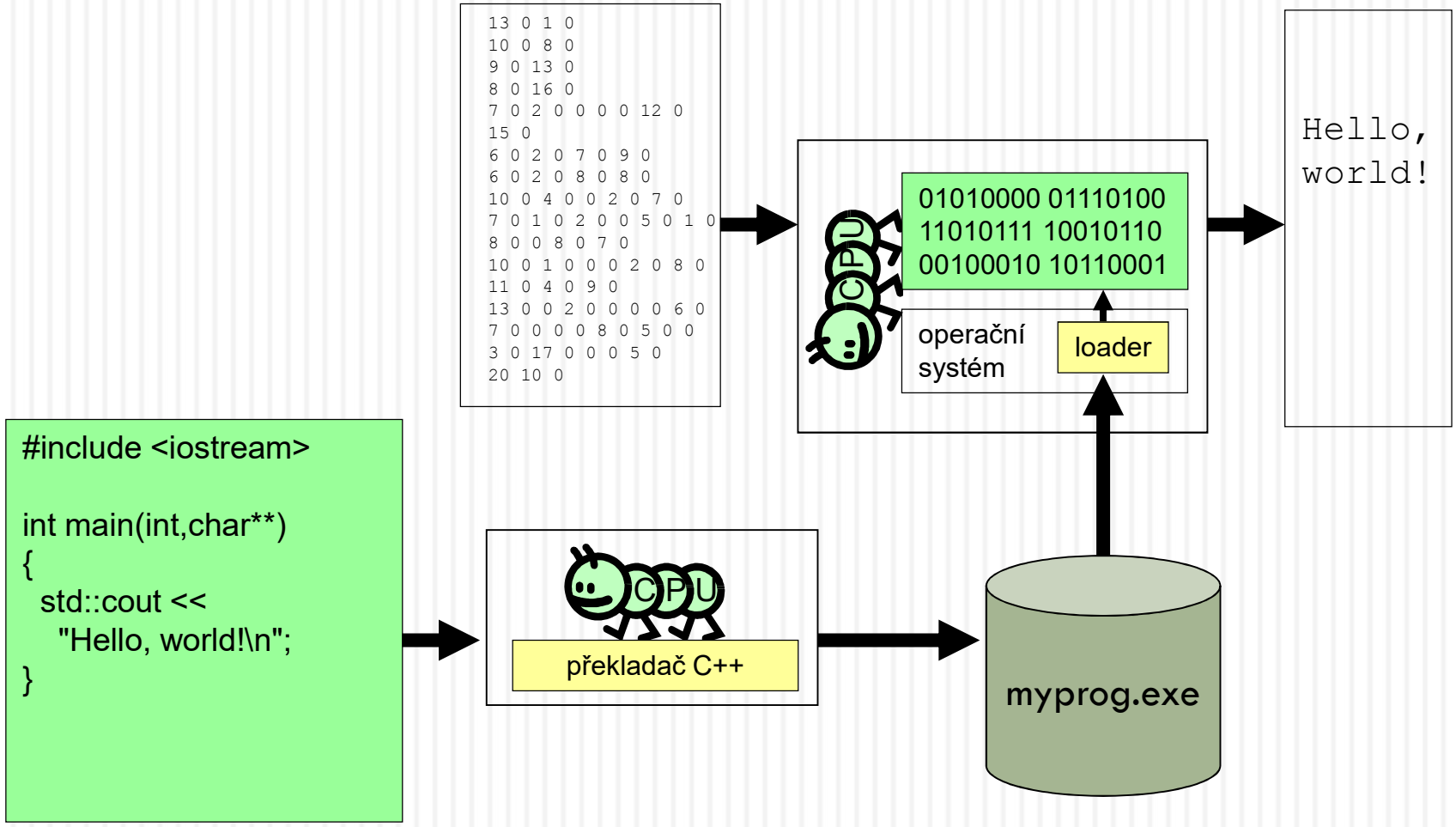
1950... – překladač



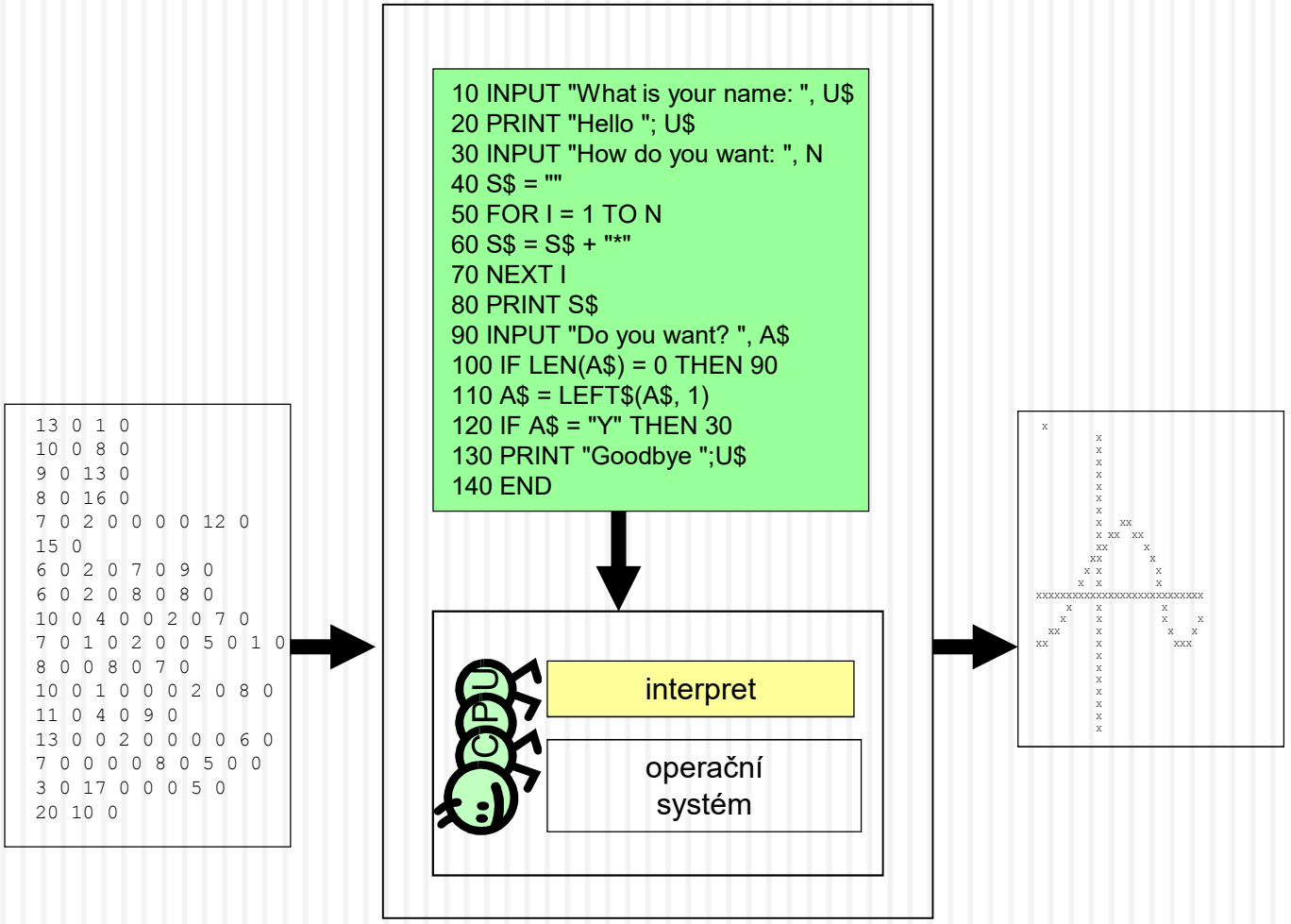
1970... – překladač C



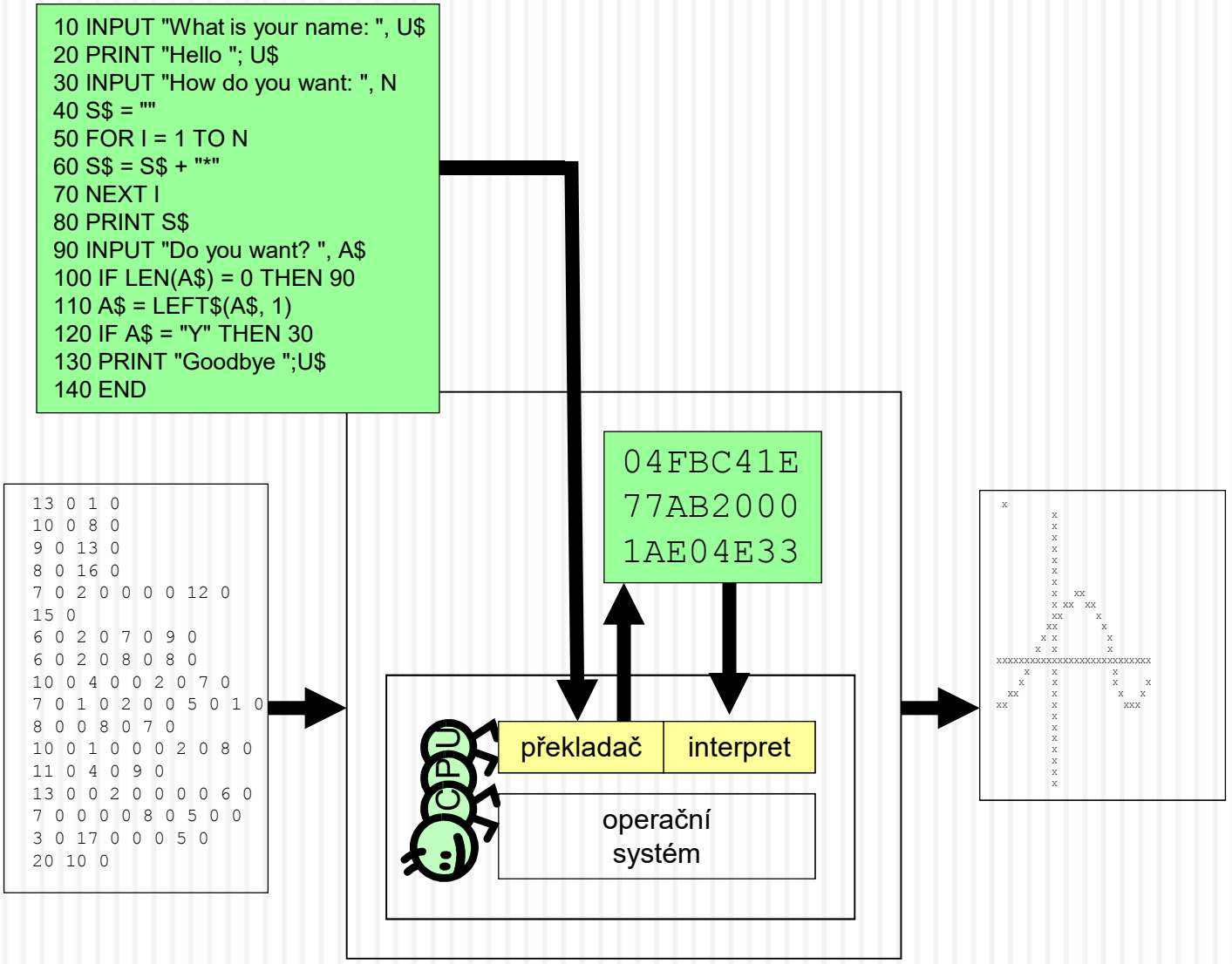
1980... – překladač C++



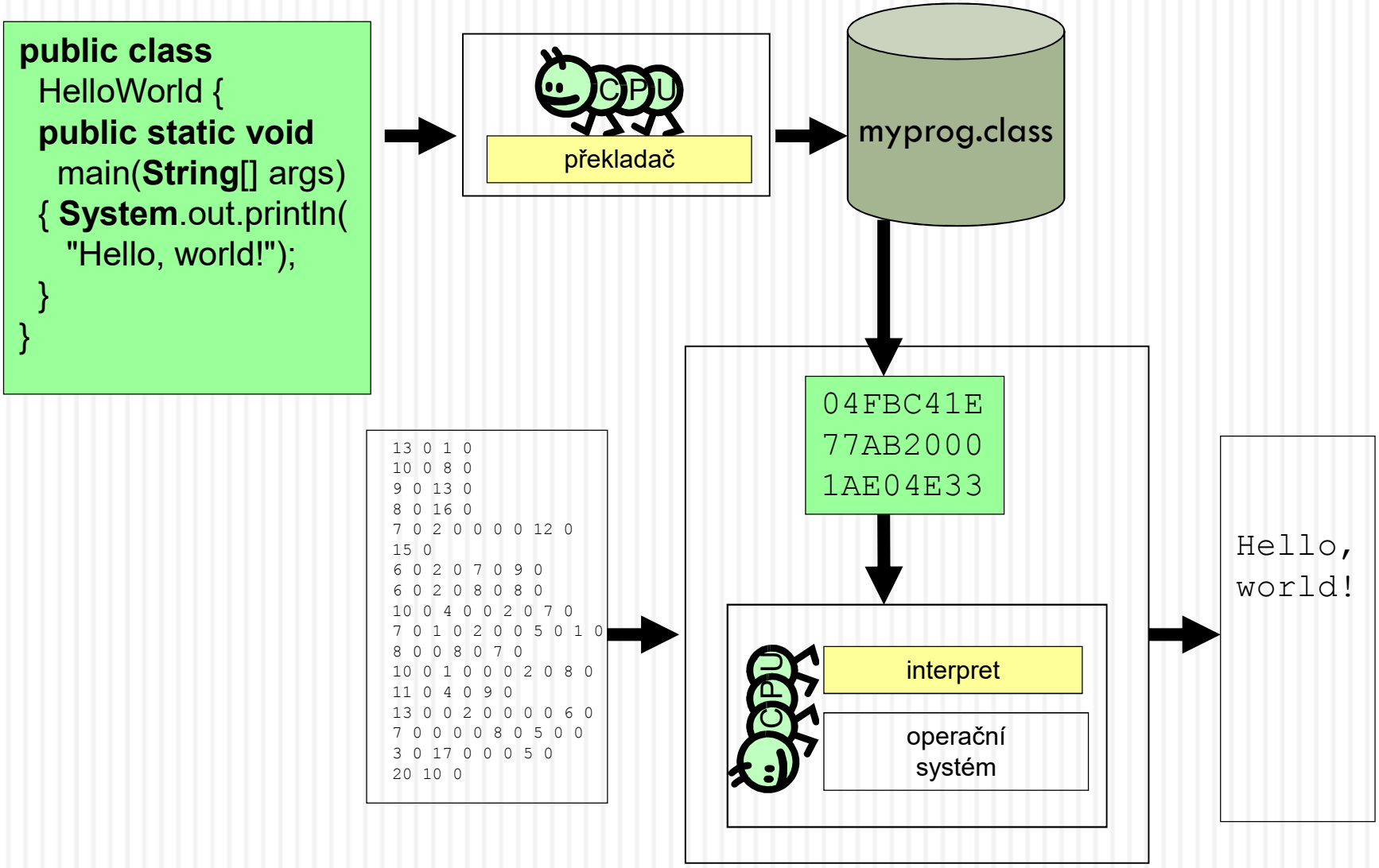
1960... – interpret(er)



Interpretace s mezikódem

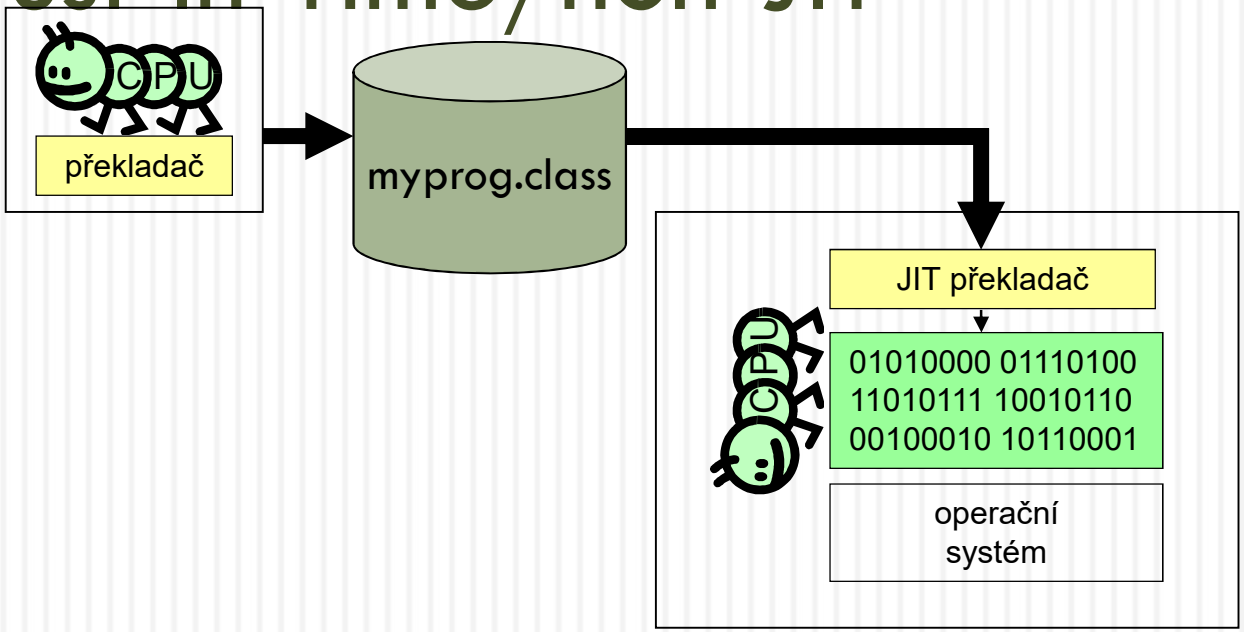


interpretovaný mezikód (bytecode)



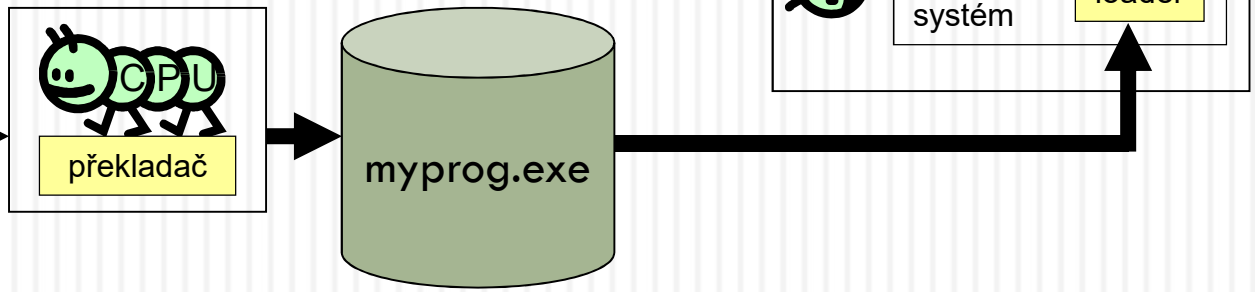
Srovnání Just-In-Time/non-JIT

```
public class
  HelloWorld {
  public static void
    main(String[] args)
  { System.out.println(
     "Hello, world!");
  }
}
```



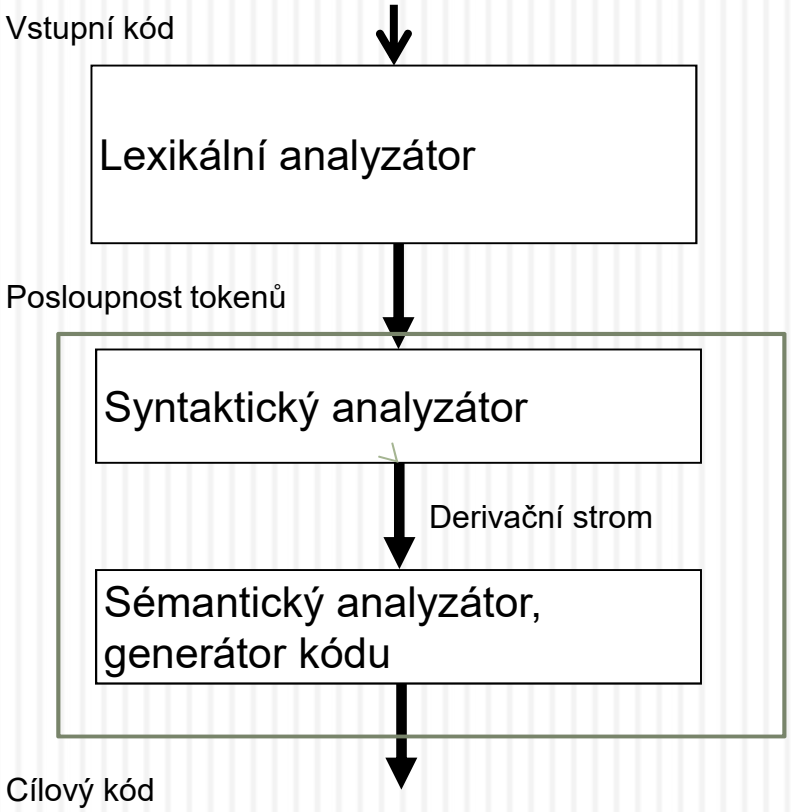

```
#include <iostream>

int main(int,char**)
{
  std::cout <<
    "Hello, world!\n";
}
```

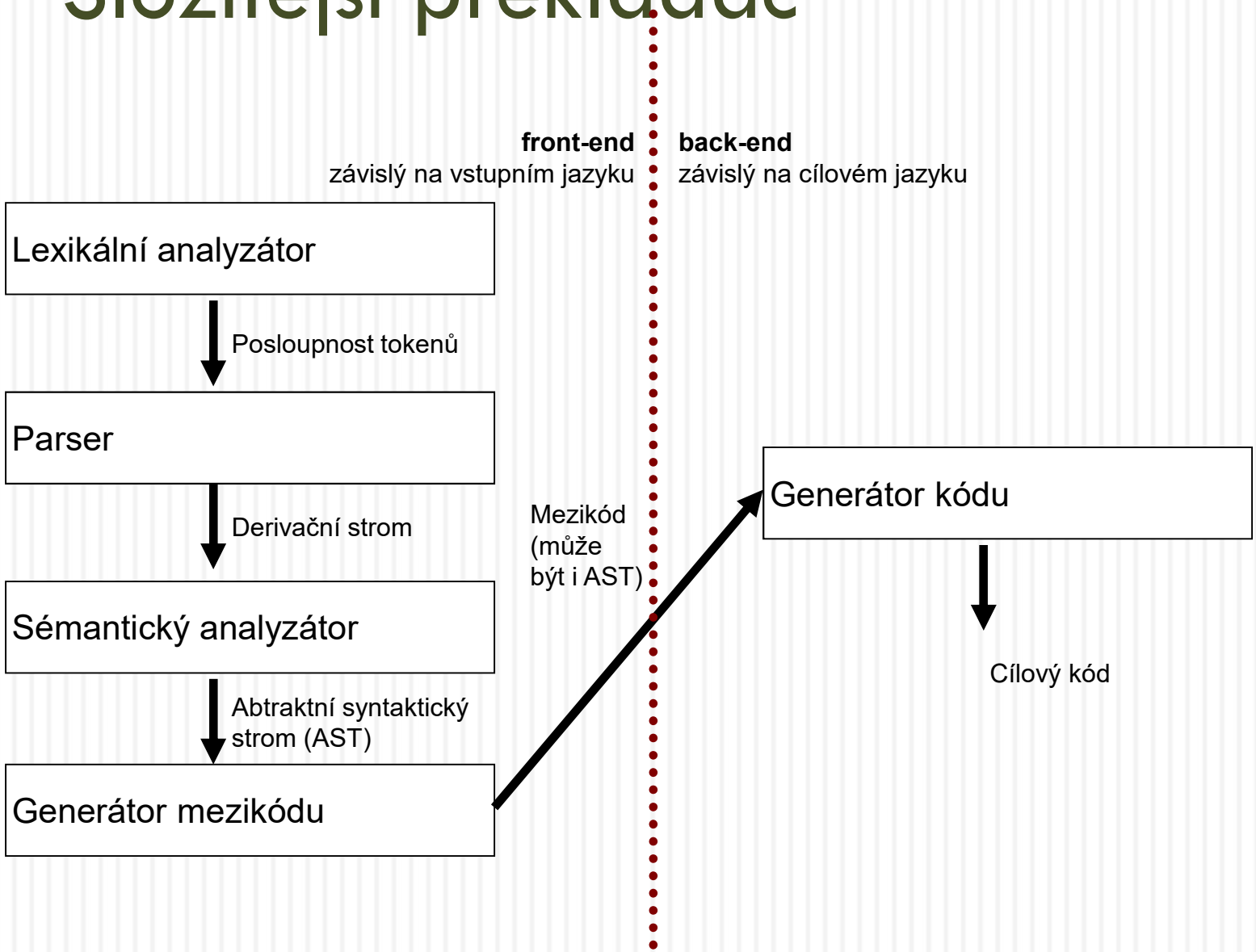


Struktura překladače

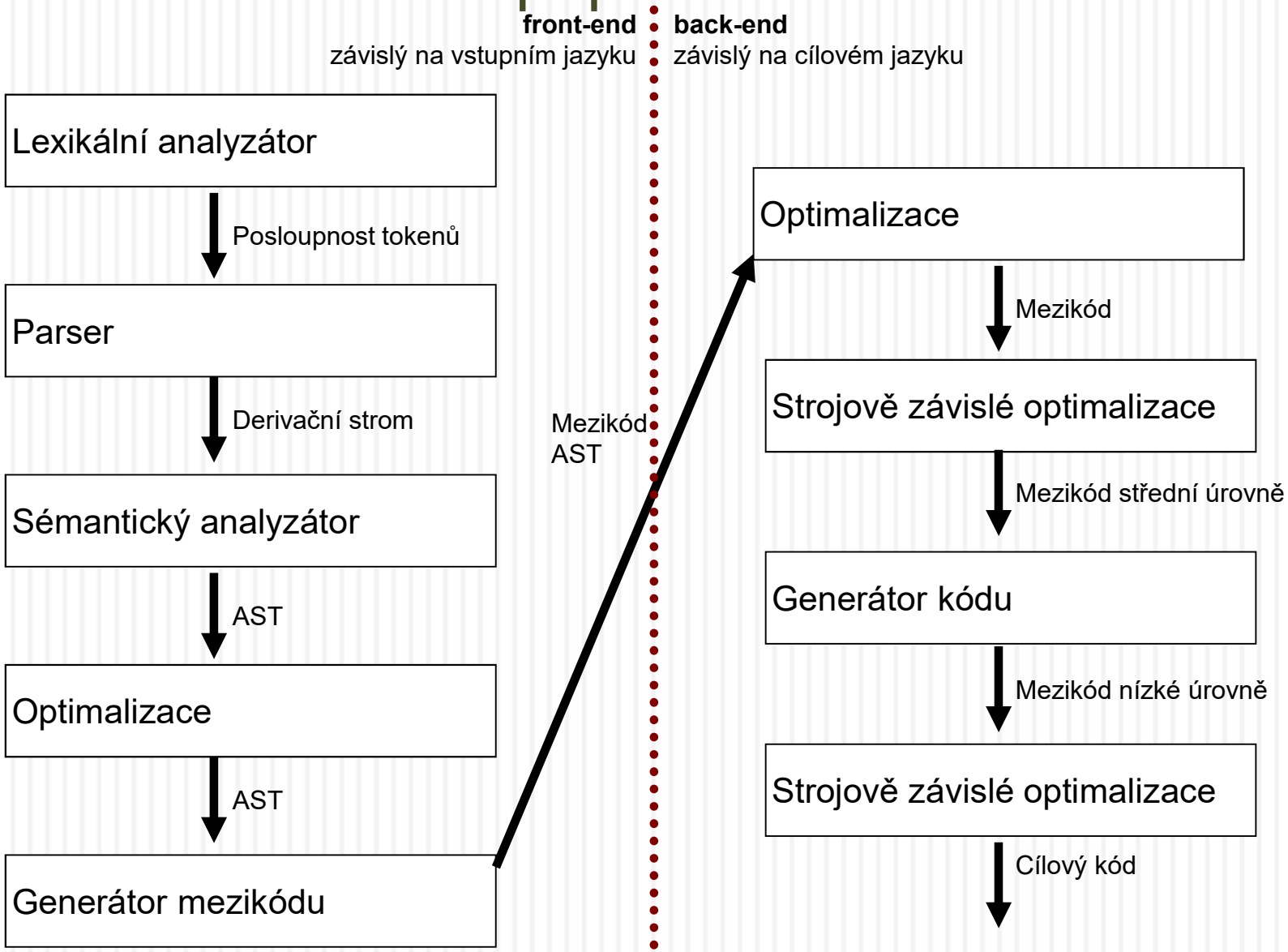
Velice jednoduchý překladač:



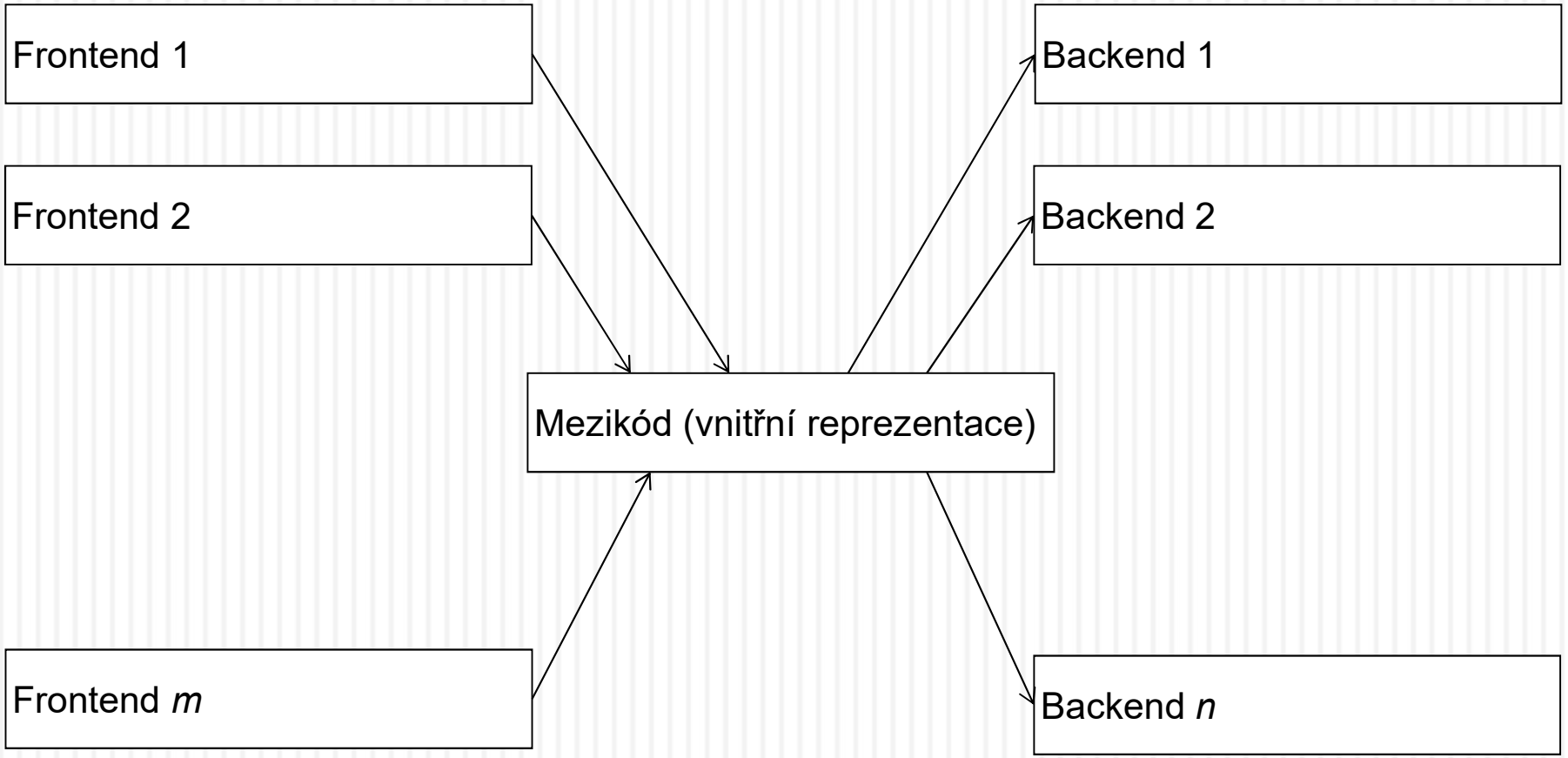
Složitější překladač



Ještě více složitější překladač



Překladač pro *m* vstupních a *n* výstupních jazyků

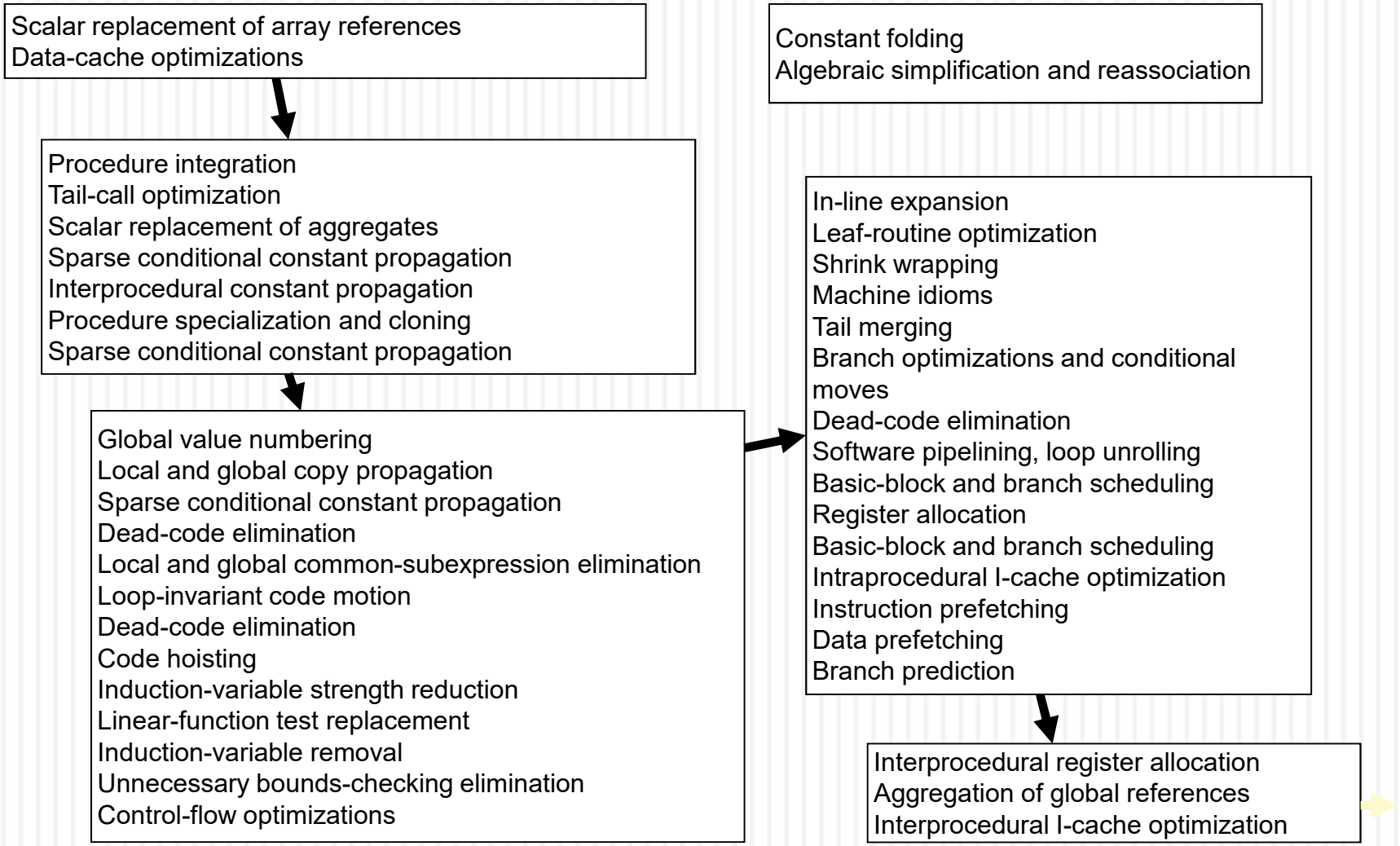


Vnitřní reprezentace – mezikódy

- Různých úrovní
- Jednotlivé typy se můžou mezi sebou převádět, pro různé fáze překladu se používají určité vhodné typy
- Nejčastější formy:
  1. AST (abstraktní syntaktický strom) – forma vyšší úrovně
  2. 3 adresový kód - idealizovaný asembler, v nejvyšší úrovni s neomezeným počtem registrů

Fuknce BACKENDu: Postupné úpravy vnitřního kódu od kódu vyšší úrovně směrem ke kódu nižší úrovně.

Optimalizace



Příklady IR

GNU Compiler Collection (gcc) používá tři vnitřní kódy:

1. GENERIC - AST

2. GIMPLE – 3adresový kód, stále IR vyšší úrovně,

3. Register Transfer Language (RTL) - IR nižší úrovně, předstupeň před assemblerem

LLVM používá LLVM Bitcode – lineární kód

Another example

- .NET compilers:
  - Various frontends
  - Intermediate representation (bytecode) MSIL (Microsoft Intermediate Language), interpreted by .NET environment

Three address code (3AC)

3AC

- linear code

- ways of generating:
  - Syntax/directed translation produced by the frontend (the translation is defined by an attribute translation grammar)
  - From an AST by traversing the AST

Main properties

- A sequence of simple instructions with **at most one** operation on the right hand side, the result of each instruction is assigned to a temporary

  ie. x = y **op** z

- an unfinite number of temporaries (something like registers and memory cells)

- Example: x+y*z

t1 = y * z
t2 = x + t1

Another example

a = b*-c+b*-c

```
t1 := -c
t2 := b * t1
t3 := -c
t4 := b * t3
t5 := t2 + t4
a := t5
```

```
t1 := -c
t2 := b * t1
a := t2 + t2
```

Parameters of operations in 3AC

- Names: source-programs names – usually implemented as pointers to the symbol table
- Compiler-generated temporaries
- Constants

Further properties

- Arrays:

x [y] = z
x = y [z]

- Labels:

L:

3AC instructions

- assignments with binary operator: x = y **op** z
- assignments with unary operator: x = **op** y
- copy instructions: x = y
- Control flow instructions:
  - ifFalse x goto L
  - ifTrue x goto L
  - goto L

3AC instructions

- Procedure calls

param *x1*
param *x2*
…
param *xn*
call *p*, *n*

- Addresses and pointer assignments:

x=&y
x=*y
*x=y

Example: translation of a statement

Statement

If *expr* then *stmt1*

is translated to this 3AC code:

```
        code to compute expr into x
        ifFalse x goto after
        code for stmt1
after:
```

Most common implementations of 3AC

- Quadruples
- Triples
- Indirect triples

Quadruples

best for easy optimization

Names of temporaries

	op	arg1	arg2	res
(0)	uminus	c		t1
(1)	*	b	t1	t2
(2)	uminus	c		t3
(3)	*	b	t3	t4
(4)	+	t2	t4	t5
(5)	:=	t5		a

Triples

Does not contain names of temporaries, which are replaced by position of instructions

	op	arg1	arg2
(0)	uminus	c	
(1)	*	b	(0)
(2)	uminus	c	
(3)	*	b	(2)
(4)	+	(1)	(3)
(5)	:=	a	(4)

Indirect triples

Positions of instructions are connected with pointers to triples (the right table)

	op	arg1	arg2
(0)	uminus	c	
(1)	*	b	(0)
(2)	uminus	c	
(3)	*	b	(2)
(4)	+	(1)	(3)
(5)	:=	a	(4)

	stmt
35	(0)
36	(1)
37	(2)
38	(3)
39	(4)
40	(5)