Kompilátory

Cvičenie 6: LLVM

Peter Kostolányi

21. novembra 2017

LLVM

- V podstate sada nástrojov pre tvorbu kompilátorov

LLVM

- V podstate sada nástrojov pre tvorbu kompilátorov
- Pôvodne Low Level Virtual Machine

LLVM

- V podstate sada nástrojov pre tvorbu kompilátorov
- Pôvodne Low Level Virtual Machine
- My budeme nástroje LLVM používať ako „back end“ kompilátora

LLVM

- V podstate sada nástrojov pre tvorbu kompilátorov
- Pôvodne Low Level Virtual Machine
- My budeme nástroje LLVM používať ako „back end“ kompilátora
- Medzijazyk LLVM IR a nástroje na jeho ďalšie spracovanie

LLVM

- V podstate sada nástrojov pre tvorbu kompilátorov
- Pôvodne Low Level Virtual Machine
- My budeme nástroje LLVM používať ako „back end“ kompilátora
- Medzijazyk LLVM IR a nástroje na jeho ďalšie spracovanie

Inštalácia:

LLVM

- V podstate sada nástrojov pre tvorbu kompilátorov
- Pôvodne Low Level Virtual Machine
- My budeme nástroje LLVM používať ako „back end“ kompilátora
- Medzijazyk LLVM IR a nástroje na jeho ďalšie spracovanie

Inštalácia:

- Na Linuxe treba nainštalovať balík LLVM

LLVM

- V podstate sada nástrojov pre tvorbu kompilátorov
- Pôvodne Low Level Virtual Machine
- My budeme nástroje LLVM používať ako „back end“ kompilátora
- Medzijazyk LLVM IR a nástroje na jeho ďalšie spracovanie

Inštalácia:

- Na Linuxe treba nainštalovať balík LLVM
- Pod Windowsom:

LLVM

- V podstate sada nástrojov pre tvorbu kompilátorov
- Pôvodne Low Level Virtual Machine
- My budeme nástroje LLVM používať ako „back end“ kompilátora
- Medzijazyk LLVM IR a nástroje na jeho ďalšie spracovanie

Inštalácia:

- Na Linuxe treba nainštalovať balík LLVM
- Pod Windowsom:
  - Je potrebné nainštalovať Cygwin s LLVM a GCC

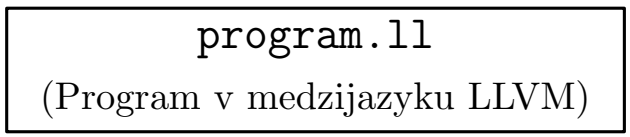
LLVM

- V podstate sada nástrojov pre tvorbu kompilátorov
- Pôvodne Low Level Virtual Machine
- My budeme nástroje LLVM používať ako „back end“ kompilátora
- Medzijazyk LLVM IR a nástroje na jeho ďalšie spracovanie

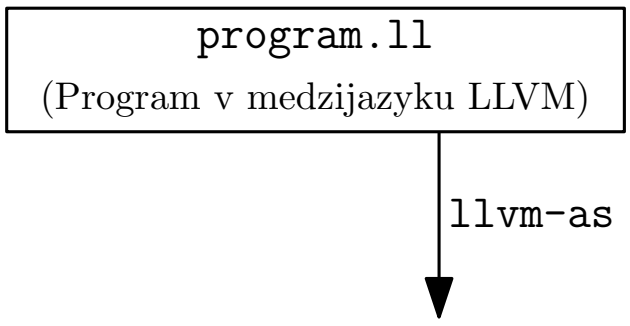
Inštalácia:

- Na Linuxe treba nainštalovať balík LLVM
- Pod Windowsom:
  - Je potrebné nainštalovať Cygwin s LLVM a GCC
  - Adresár s binárnymi súbormi je vhodné pridať do premennej `PATH`

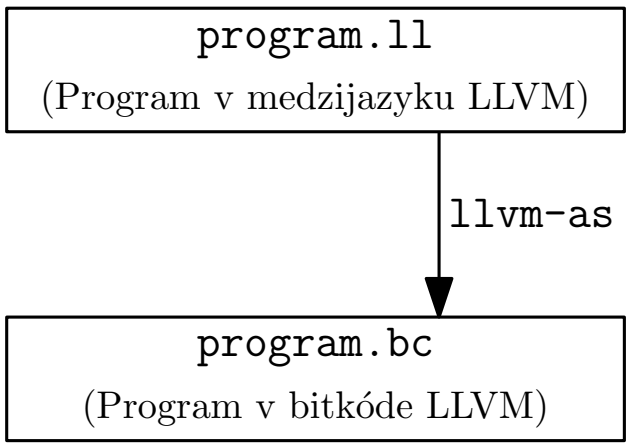
Medzijazyk, bitkód a preklad do strojového kódu



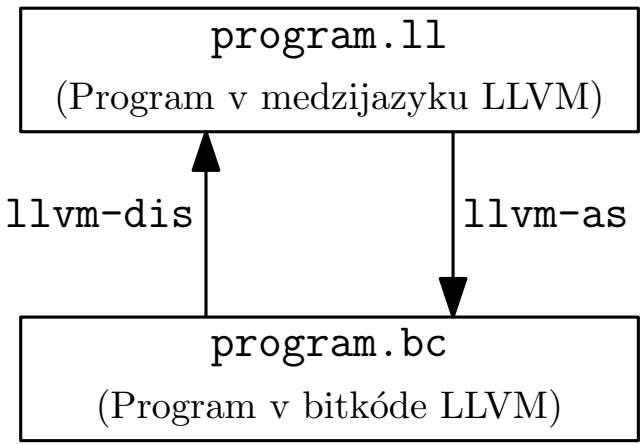
Medzijazyk, bitkód a preklad do strojového kódu



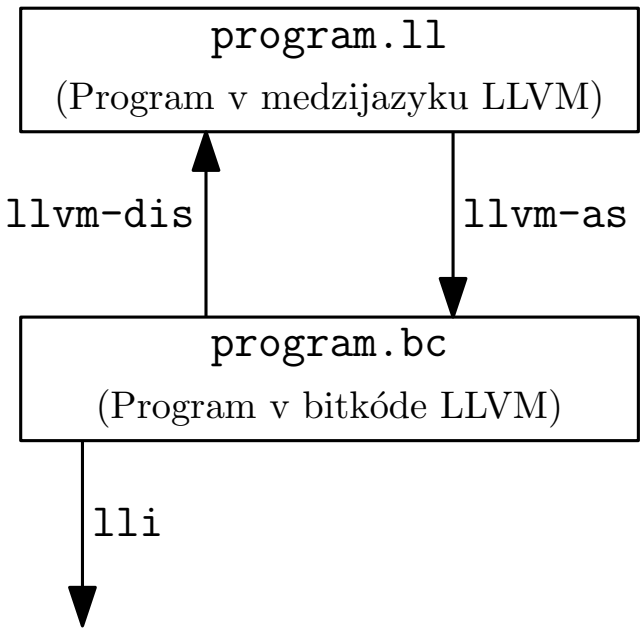
Medzijazyk, bitkód a preklad do strojového kódu



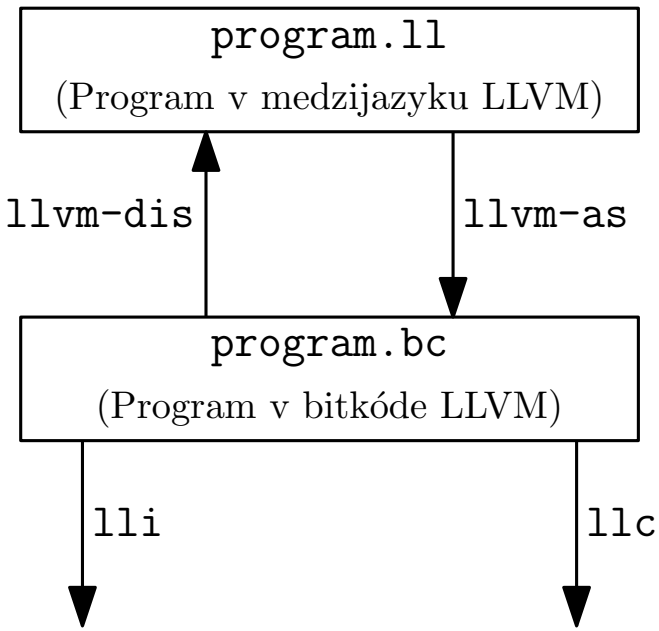
Medzijazyk, bitkód a preklad do strojového kódu



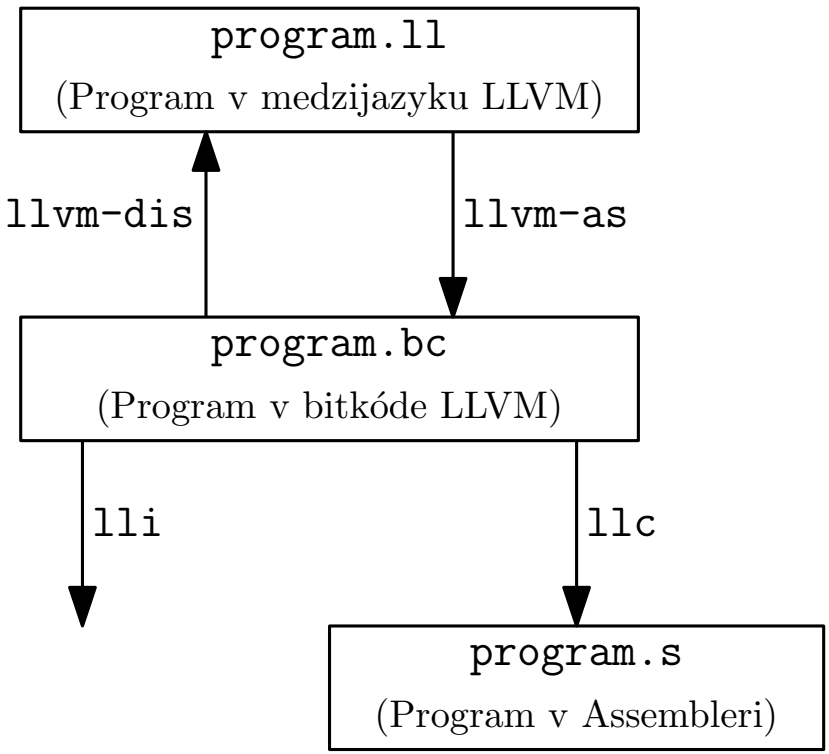
Medzijazyk, bitkód a preklad do strojového kódu



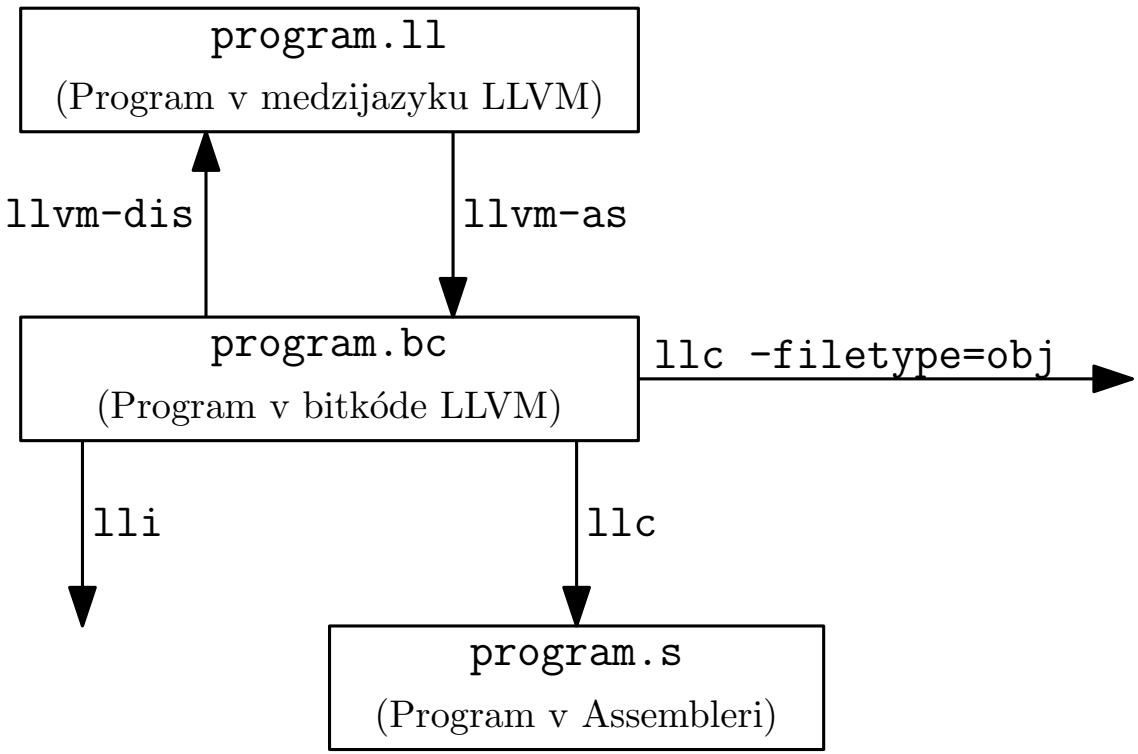
Medzijazyk, bitkód a preklad do strojového kódu



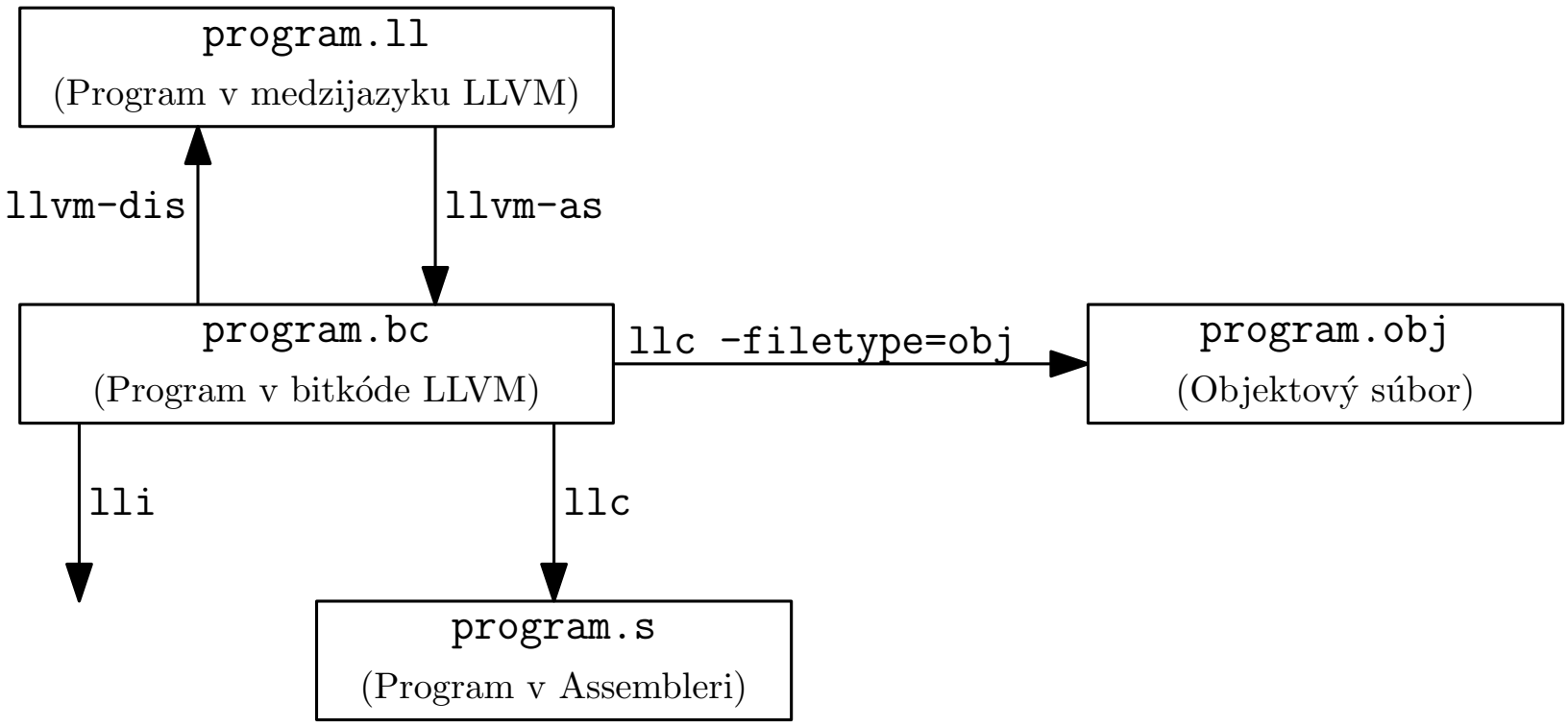
Medzijazyk, bitkód a preklad do strojového kódu



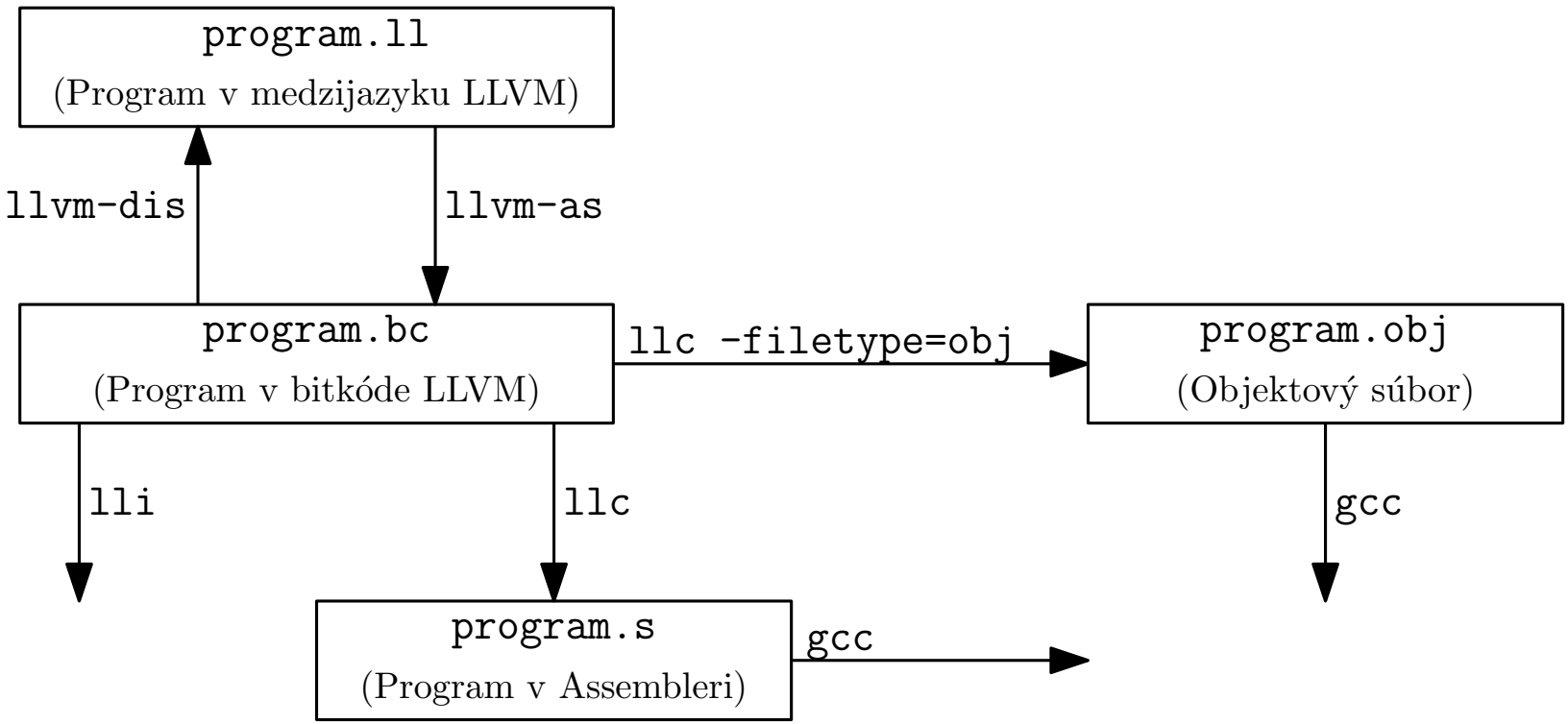
Medzijazyk, bitkód a preklad do strojového kódu



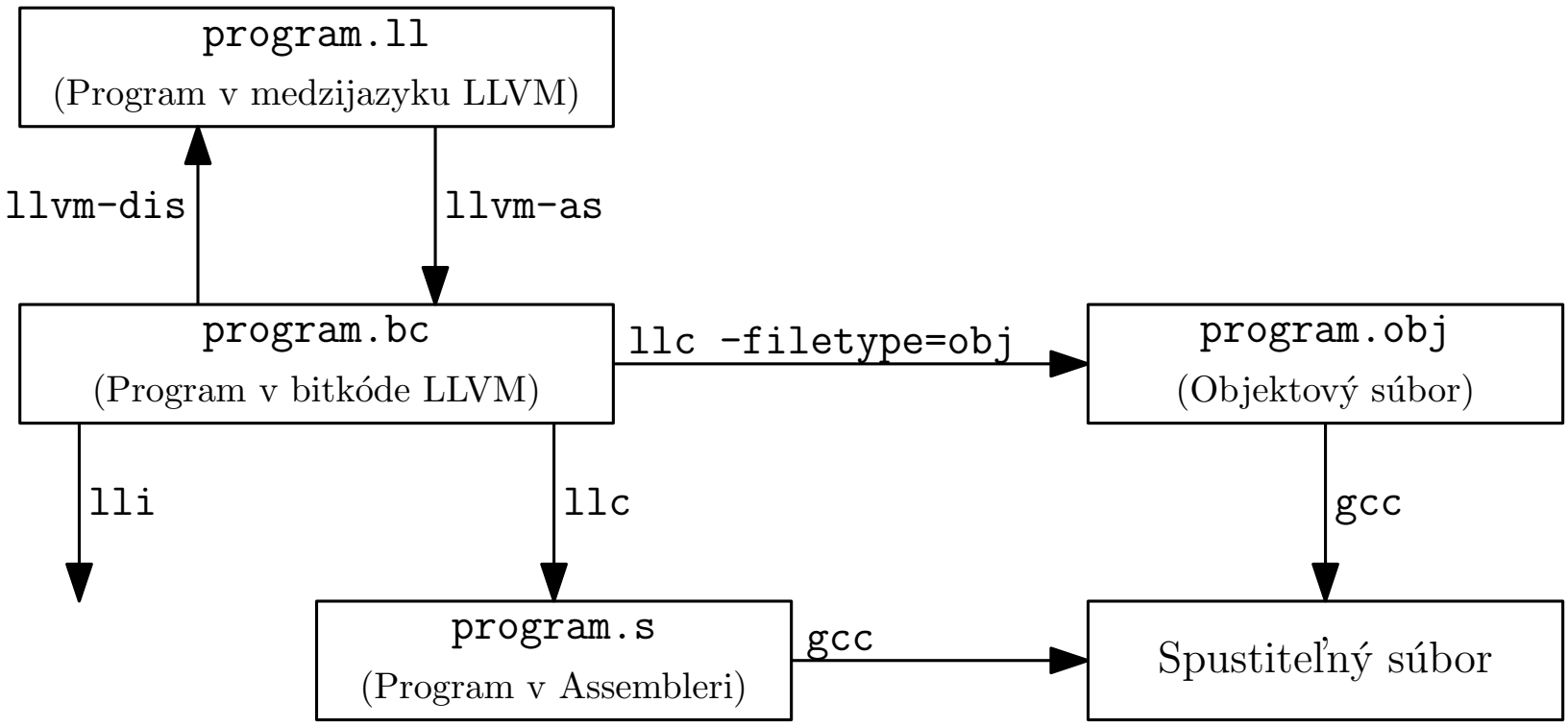
Medzijazyk, bitkód a preklad do strojového kódu



Medzijazyk, bitkód a preklad do strojového kódu



Medzijazyk, bitkód a preklad do strojového kódu



Najdôležitejšie nástroje

`llvm-as`: LLVM „assembler“ prekladajúci z „človekom čitateľného“ medzijazyka do bitkódu

Najdôležitejšie nástroje

`llvm-as`: LLVM „assembler“ prekladajúci z „človekom čitateľného“ medzijazyka do bitkódu

`llvm-dis`: LLVM „disassembler“ realizujúci opačnú transformáciu

Najdôležitejšie nástroje

`llvm-as`: LLVM „assembler“ prekladajúci z „človekom čitateľného“ medzijazyka do bitkódu

`llvm-dis`: LLVM „disassembler“ realizujúci opačnú transformáciu

`lli`: LLVM interpreter pre programy v LLVM bitkóde

Najdôležitejšie nástroje

`llvm-as`: LLVM „assembler“ prekladajúci z „človekom čitateľného“ medzijazyka do bitkódu

`llvm-dis`: LLVM „disassembler“ realizujúci opačnú transformáciu

`lli`: LLVM interpreter pre programy v LLVM bitkóde

`llc`: LLVM kompilátor prekladajúci z LLVM bitkódu do „ozajstného“ assembleru, resp. priamo vytvárajúci objektové súbory

Najdôležitejšie nástroje

`llvm-as`: LLVM „assembler“ prekladajúci z „človekom čitateľného“ medzijazyka do bitkódu

`llvm-dis`: LLVM „disassembler“ realizujúci opačnú transformáciu

`lli`: LLVM interpreter pre programy v LLVM bitkóde

`llc`: LLVM kompilátor prekladajúci z LLVM bitkódu do „ozajstného“ assembleru, resp. priamo vytvárajúci objektové súbory

`opt`: Optimalizátor LLVM kódu

Najdôležitejšie nástroje

`llvm-as`: LLVM „assembler“ prekladajúci z „človekom čitateľného“ medzijazyka do bitkódu

`llvm-dis`: LLVM „disassembler“ realizujúci opačnú transformáciu

`lli`: LLVM interpreter pre programy v LLVM bitkóde

`llc`: LLVM kompilátor prekladajúci z LLVM bitkódu do „ozajstného“ assembleru, resp. priamo vytvárajúci objektové súbory

`opt`: Optimalizátor LLVM kódu

`clang`: „Front end“ kompilátora pre jazyk C využívajúci LLVM ako svoj „back end“

Zdroje informácií o LLVM

Dokumentácia k medzijazyku LLVM:

Zdroje informácií o LLVM

Dokumentácia k medzijazyku LLVM:

- `http://llvm.org/docs/LangRef.html`

Zdroje informácií o LLVM

Dokumentácia k medzijazyku LLVM:

- `http://llvm.org/docs/LangRef.html`
- Popisy všetkých dostupných inštrukcií

Zdroje informácií o LLVM

Dokumentácia k medzijazyku LLVM:

- `http://llvm.org/docs/LangRef.html`
- Popisy všetkých dostupných inštrukcií

Dokumentácia k nástrojom LLVM:

Zdroje informácií o LLVM

Dokumentácia k medzijazyku LLVM:

- `http://llvm.org/docs/LangRef.html`
- Popisy všetkých dostupných inštrukcií

Dokumentácia k nástrojom LLVM:

- `https://llvm.org/docs/CommandGuide/`

Zdroje informácií o LLVM

Dokumentácia k medzijazyku LLVM:

- `http://llvm.org/docs/LangRef.html`
- Popisy všetkých dostupných inštrukcií

Dokumentácia k nástrojom LLVM:

- `https://llvm.org/docs/CommandGuide/`

Preklad z C do LLVM:

Zdroje informácií o LLVM

Dokumentácia k medzijazyku LLVM:

- `http://llvm.org/docs/LangRef.html`
- Popisy všetkých dostupných inštrukcií

Dokumentácia k nástrojom LLVM:

- `https://llvm.org/docs/CommandGuide/`

Preklad z C do LLVM:

- `clang -S -emit-llvm cokolvek.c`

„Hello, World!“ v medzijazyku LLVM

Súbor `Hello.ll`:

„Hello, World!“ v medzijazyku LLVM

Súbor `Hello.ll`:

```
1 @s = global [16 x i8] c"Hello, world!\0D\0A\00"

3 declare i32 @printf(i8*, ...)

5 define i32 @main() {                    ; toto je komentar
  start:
7     %pointer = getelementptr [16 x i8], [16 x i8]* @s, i32 0, i32 0
      call i32 (i8*, ...) @printf(i8* %pointer)
9     ret i32 0
  }
```

„Hello, World!“ v medzijazyku LLVM

Súbor `Hello.ll`:

```
@s = global [16 x i8] c"Hello, world!\0D\0A\00"

declare i32 @printf(i8*, ...)

define i32 @main() {                    ; toto je komentar
start:
    %pointer = getelementptr [16 x i8], [16 x i8]* @s, i32 0, i32 0
    call i32 (i8*, ...) @printf(i8* %pointer)
    ret i32 0
}
```

- `llvm-as Hello.ll`

„Hello, World!“ v medzijazyku LLVM

Súbor `Hello.ll`:

```
1 @s = global [16 x i8] c"Hello, world!\0D\0A\00"

3 declare i32 @printf(i8*, ...)

5 define i32 @main() {                    ; toto je komentar
  start:
7     %pointer = getelementptr [16 x i8], [16 x i8]* @s, i32 0, i32 0
      call i32 (i8*, ...) @printf(i8* %pointer)
9     ret i32 0
  }
```

- `llvm-as Hello.ll`
- `lli Hello.bc`

„Hello, World!“ v medzijazyku LLVM

Súbor `Hello.ll`:

```
1 @s = global [16 x i8] c"Hello, world!\0D\0A\00"

3 declare i32 @printf(i8*, ...)

5 define i32 @main() {                    ; toto je komentar
  start:
7     %pointer = getelementptr [16 x i8], [16 x i8]* @s, i32 0, i32 0
      call i32 (i8*, ...) @printf(i8* %pointer)
9     ret i32 0
  }
```

- `llvm-as Hello.ll`
- `lli Hello.bc`
- `llc -filetype=obj Hello.bc`

„Hello, World!“ v medzijazyku LLVM

Súbor `Hello.ll`:

```
1 @s = global [16 x i8] c"Hello, world!\0D\0A\00"

3 declare i32 @printf(i8*, ...)

5 define i32 @main() {                    ; toto je komentar
  start:
7     %pointer = getelementptr [16 x i8], [16 x i8]* @s, i32 0, i32 0
      call i32 (i8*, ...) @printf(i8* %pointer)
9     ret i32 0
  }
```

- `llvm-as Hello.ll`
- `lli Hello.bc`
- `llc -filetype=obj Hello.bc`
- `gcc -o Hello Hello.obj`

Typy v LLVM

Typy „prvej triedy“:

Typy v LLVM

Typy „prvej triedy“:

`iN:` N-bitové celé čísla pre nejaké $1 \leq N \leq 2^{23} - 1$

Typy v LLVM

Typy „prvej triedy“:

`iN:` N-bitové celé čísla pre nejaké $1 \leq N \leq 2^{23} - 1$

`float:` 32-bitové „floating-point“ reálne čísla

Typy v LLVM

Typy „prvej triedy“:

`iN`: N-bitové celé čísla pre nejaké $1 \leq N \leq 2^{23} - 1$

`float`: 32-bitové „floating-point“ reálne čísla

`double`: 64-bitové „floating-point“ reálne čísla

Typy v LLVM

Typy „prvej triedy“:

- `iN`: N-bitové celé čísla pre nejaké $1 \leq N \leq 2^{23} - 1$
- `float`: 32-bitové „floating-point“ reálne čísla
- `double`: 64-bitové „floating-point“ reálne čísla
- Smerníky: Napríklad `i32*`, `float*`

Typy v LLVM

Typy „prvej triedy“:

`iN`:	N-bitové celé čísla pre nejaké $1 \leq N \leq 2^{23} - 1$
`float`:	32-bitové „floating-point“ reálne čísla
`double`:	64-bitové „floating-point“ reálne čísla
Smerníky:	Napríklad `i32*`, `float*`
	...

Typy v LLVM

Typy „prvej triedy“:

- `iN`: N-bitové celé čísla pre nejaké $1 \leq N \leq 2^{23} - 1$
- `float`: 32-bitové „floating-point“ reálne čísla
- `double`: 64-bitové „floating-point“ reálne čísla
- Smerníky: Napríklad `i32*`, `float*`

  ...

Zložené typy:

Typy v LLVM

Typy „prvej triedy“:

- `iN`: N-bitové celé čísla pre nejaké $1 \leq N \leq 2^{23} - 1$
- `float`: 32-bitové „floating-point“ reálne čísla
- `double`: 64-bitové „floating-point“ reálne čísla
- Smerníky: Napríklad `i32*`, `float*`
- ...

Zložené typy:

- Polia: Napríklad `[4 x i32]` alebo `[3 x [4 x i32]]`

Typy v LLVM

Typy „prvej triedy“:

- `iN`: N-bitové celé čísla pre nejaké $1 \leq N \leq 2^{23} - 1$
- `float`: 32-bitové „floating-point“ reálne čísla
- `double`: 64-bitové „floating-point“ reálne čísla
- Smerníky: Napríklad `i32*`, `float*`

  ...

Zložené typy:

- Polia: Napríklad `[4 x i32]` alebo `[3 x [4 x i32]]`

  ...

Typy v LLVM

Typy „prvej triedy“:

- `iN`: N-bitové celé čísla pre nejaké $1 \leq N \leq 2^{23} - 1$
- `float`: 32-bitové „floating-point“ reálne čísla
- `double`: 64-bitové „floating-point“ reálne čísla
- Smerníky: Napríklad `i32*`, `float*`
- ...

Zložené typy:

- Polia: Napríklad `[4 x i32]` alebo `[3 x [4 x i32]]`
- ...

Ďalšie typy:

Typy v LLVM

Typy „prvej triedy“:

- `iN`: N-bitové celé čísla pre nejaké $1 \leq N \leq 2^{23} - 1$
- `float`: 32-bitové „floating-point“ reálne čísla
- `double`: 64-bitové „floating-point“ reálne čísla
- Smerníky: Napríklad `i32*`, `float*`

...

Zložené typy:

- Polia: Napríklad `[4 x i32]` alebo `[3 x [4 x i32]]`

...

Ďalšie typy:

- Napríklad funkcie

Identifikátory v LLVM

- Globálne identifikátory musia začínať symbolom „@“

Identifikátory v LLVM

- Globálne identifikátory musia začínať symbolom „@“
- Lokálne identifikátory musia začínať symbolom „%“

Identifikátory v LLVM

- Globálne identifikátory musia začínať symbolom „@“
- Lokálne identifikátory musia začínať symbolom „%“
- Korektné pomenovania sú popísané regulárnym výrazom `[%@][-a-zA-Z$._][-a-zA-Z$._0-9]*`

Identifikátory v LLVM

- Globálne identifikátory musia začínať symbolom „@“
- Lokálne identifikátory musia začínať symbolom „%“
- Korektné pomenovania sú popísané regulárnym výrazom `[%@][-a-zA-Z$._][-a-zA-Z$._0-9]*`

Globálne premenné sú vždy smerníky

Identifikátory v LLVM

- Globálne identifikátory musia začínať symbolom „@“
- Lokálne identifikátory musia začínať symbolom „%“
- Korektné pomenovania sú popísané regulárnym výrazom `[%@][-a-zA-Z$._][-a-zA-Z$._0-9]*`

Globálne premenné sú vždy smerníky

- Napríklad `@n = global i32 41` definuje premennú `@n` typu `i32*`

Identifikátory v LLVM

- Globálne identifikátory musia začínať symbolom „@“
- Lokálne identifikátory musia začínať symbolom „%“
- Korektné pomenovania sú popísané regulárnym výrazom `[%@][-a-zA-Z$._][-a-zA-Z$._0-9]*`

Globálne premenné sú vždy smerníky

- Napríklad `@n = global i32 41` definuje premennú `@n` typu `i32*`

```
1  @n = global i32 41
   @format = global [5 x i8] c"%d\0D\0A\00"
3
   declare i32 @printf(i8*, ...)
5
   define i32 @main() {
7  start:
       %nn = load i32, i32* @n
9      %r = add i32 %nn, 1
       %pointer = getelementptr [5 x i8], [5 x i8]* @format, i32 0, i32 0
11     call i32 (i8*, ...) @printf(i8* %pointer, i32 %r)
       ret i32 0
13 }
```

Štruktúra programu v LLVM

Každá funkcia pozostáva z niekoľkých základných blokov:

Štruktúra programu v LLVM

Každá funkcia pozostáva z niekoľkých základných blokov:

- Skoky v programe: iba z koncov blokov a iba na začiatky blokov

Štruktúra programu v LLVM

Každá funkcia pozostáva z niekoľkých základných blokov:

- Skoky v programe: iba z koncov blokov a iba na začiatky blokov
- Každý základný blok musí skončiť príkazom na ukončenie bloku (podmienený alebo nepodmienený `br`, `ret`, ...)

Štruktúra programu v LLVM

Každá funkcia pozostáva z niekoľkých základných blokov:

- Skoky v programe: iba z koncov blokov a iba na začiatky blokov
- Každý základný blok musí skončiť príkazom na ukončenie bloku (podmienený alebo nepodmienený `br`, `ret`, ...)
- Prvý základný blok funkcie:

Štruktúra programu v LLVM

Každá funkcia pozostáva z niekoľkých základných blokov:

- Skoky v programe: iba z koncov blokov a iba na začiatky blokov
- Každý základný blok musí skončiť príkazom na ukončenie bloku (podmienený alebo nepodmienený `br`, `ret`, ...)
- Prvý základný blok funkcie:
  - Vykonáva sa po zavolaní funkcie

Štruktúra programu v LLVM

Každá funkcia pozostáva z niekoľkých základných blokov:

- Skoky v programe: iba z koncov blokov a iba na začiatky blokov
- Každý základný blok musí skončiť príkazom na ukončenie bloku (podmienený alebo nepodmienený `br`, `ret`, ...)
- Prvý základný blok funkcie:
  - Vykonáva sa po zavolaní funkcie
  - Môže, ale nemusí byť pomenovaný

Štruktúra programu v LLVM

Každá funkcia pozostáva z niekoľkých základných blokov:

- Skoky v programe: iba z koncov blokov a iba na začiatky blokov
- Každý základný blok musí skončiť príkazom na ukončenie bloku (podmienený alebo nepodmienený `br`, `ret`, ...)
- Prvý základný blok funkcie:
  - Vykonáva sa po zavolaní funkcie
  - Môže, ale nemusí byť pomenovaný
  - V programe nemôže byť žiaden skok na takýto blok

Štruktúra programu v LLVM

Každá funkcia pozostáva z niekoľkých základných blokov:

- Skoky v programe: iba z koncov blokov a iba na začiatky blokov
- Každý základný blok musí skončiť príkazom na ukončenie bloku (podmienený alebo nepodmienený `br`, `ret`, ...)
- Prvý základný blok funkcie:
  - Vykonáva sa po zavolaní funkcie
  - Môže, ale nemusí byť pomenovaný
  - V programe nemôže byť žiaden skok na takýto blok
- Na množine vrcholov zodpovedajúcich blokom možno skonštruovať graf programu

SSA forma a funkcia Φ

- SSA = Static Single Assignment

SSA forma a funkcia Φ

- SSA = Static Single Assignment
- Do každej premennej možno priraďovať práve raz

SSA forma a funkcia Φ

- SSA = Static Single Assignment
- Do každej premennej možno priraďovať práve raz
- Problém napríklad pri vetvení programu, cykloch, atď.

SSA forma a funkcia Φ

- SSA = Static Single Assignment
- Do každej premennej možno priraďovať práve raz
- Problém napríklad pri vetvení programu, cykloch, atď.
- Funkcia Φ realizovaná inštrukciou `phi`:

SSA forma a funkcia Φ

- SSA = Static Single Assignment
- Do každej premennej možno priraďovať práve raz
- Problém napríklad pri vetvení programu, cykloch, atď.
- Funkcia Φ realizovaná inštrukciou `phi`:
  - Umožňuje priradiť do premennej jednu z viacerých možných hodnôt na základe naposledy vykonaného bloku

SSA forma a funkcia Φ

- SSA = Static Single Assignment
- Do každej premennej možno priraďovať práve raz
- Problém napríklad pri vetvení programu, cykloch, atď.
- Funkcia Φ realizovaná inštrukciou `phi`:
  - Umožňuje priradiť do premennej jednu z viacerých možných hodnôt na základe naposledy vykonaného bloku
  - Inštrukcie `phi` musia byť na začiatku bloku (nesmú byť pred nimi inštrukcie iné ako `phi`)

Použitie externých modulov

Funkcie definované v `library.c` možno použiť v LLVM kóde nasledovne:

Použitie externých modulov

Funkcie definované v `library.c` možno použiť v LLVM kóde nasledovne:

- Nech `main.ll` obsahuje deklarácie a volania funkcií z `library.c`

Použitie externých modulov

Funkcie definované v `library.c` možno použiť v LLVM kóde nasledovne:

- Nech `main.ll` obsahuje deklarácie a volania funkcií z `library.c`
- `gcc -fPIC -shared -o library.so library.c`

Použitie externých modulov

Funkcie definované v `library.c` možno použiť v LLVM kóde nasledovne:

- Nech `main.ll` obsahuje deklarácie a volania funkcií z `library.c`
- `gcc -fPIC -shared -o library.so library.c`
- `llvm-as main.ll`

Použitie externých modulov

Funkcie definované v `library.c` možno použiť v LLVM kóde nasledovne:

- Nech `main.ll` obsahuje deklarácie a volania funkcií z `library.c`
- `gcc -fPIC -shared -o library.so library.c`
- `llvm-as main.ll`
- `lli -load=./library.so main.bc`

Použitie externých modulov

Funkcie definované v `library.c` možno použiť v LLVM kóde nasledovne:

- Nech `main.ll` obsahuje deklarácie a volania funkcií z `library.c`
- `gcc -fPIC -shared -o library.so library.c`
- `llvm-as main.ll`
- `lli -load=./library.so main.bc`
- `llc -load=./library.so -filetype=obj main.bc`

Použitie externých modulov

Funkcie definované v `library.c` možno použiť v LLVM kóde nasledovne:

- Nech `main.ll` obsahuje deklarácie a volania funkcií z `library.c`
- `gcc -fPIC -shared -o library.so library.c`
- `llvm-as main.ll`
- `lli -load=./library.so main.bc`
- `llc -load=./library.so -filetype=obj main.bc`
- `gcc -L. -l:library.so -o main main.obj`