Kompajlerska infrastruktura LLVM i optimizacije

Đorđe Todorović
Matematički fakultet, Univerzitet u Beogradu
Beograd, Mart 2023.

SYRMIA

- https://www.syrmia.com/
- Kancelarije u Beogradu, Novom Sadu, Nišu i Banja Luci
- Preko 200 inženjera
- Projekti
  - Sistemski softver (kompajleri, alati, OS, itd.)
    - LLVM, GNU GCC, GNU GDB, QEMU, Linux …
  - Programski jezici C, C++, Python, Asembler, Rust i drugi
- Program stipendiranih praksi
  - Prijave na jobs@syrmia.com

SYRMIA

Kompajleri

```c
#include <stdio.h>

int main()
{
    printf("Hello world!\n");
    return 0;
}

```

```
0000000 facf feed 000c 0100 0000 0000 0002 0000
0000010 0011 0000 0420 0000 0085 0020 0000 0000
0000020 0019 0000 0048 0000 5f5f 4150 4547 455a
0000030 4f52 0000 0000 0000 0000 0000 0000 0000
0000040 0000 0000 0001 0000 0000 0000 0000 0000
0000050 0000 0000 0000 0000 0000 0000 0000 0000
0000060 0000 0000 0000 0000 0019 0000 0188 0000
0000070 5f5f 4554 5458 0000 0000 0000 0000 0000
0000080 0000 0000 0001 0000 4000 0000 0000 0000
0000090 0000 0000 0000 0000 4000 0000 0000 0000
00000a0 0005 0000 0005 0000 0004 0000 0000 0000
00000b0 5f5f 6574 7478 0000 0000 0000 0000 0000
00000c0 5f5f 4554 5458 0000 0000 0000 0000 0000
```

```
$ clang test.c
$ ./a.out
Hello World!
```

Kompajleri

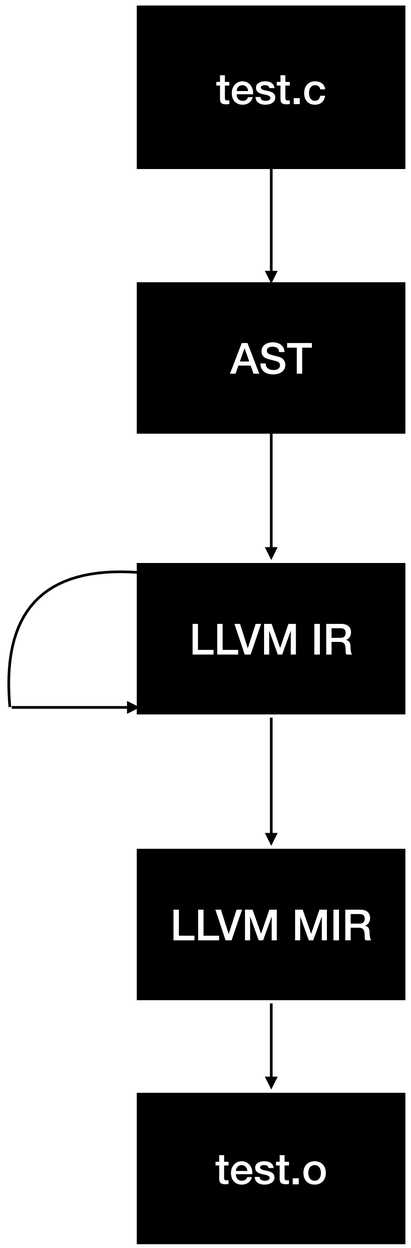
- Faze prevođenja
  - Prednji deo (eng. Frontened)
  - Srednji deo (eng. Middleend)
  - Zadnji deo (eng. Backend)

Projekat LLVM

- Autor: Kris Latner
  - 2000. master teza
  - 2005. Apple
- Ekosistem
  - llvm/ clang/ mlir/ lld/ lldb/ flang/ …
- LLVM IR, MLIR
- clang, rustc, swiftc
- Alati
  - debageri, profajleri, linkeri, sanitajzeri…
  - …

Faze prevođenja - C

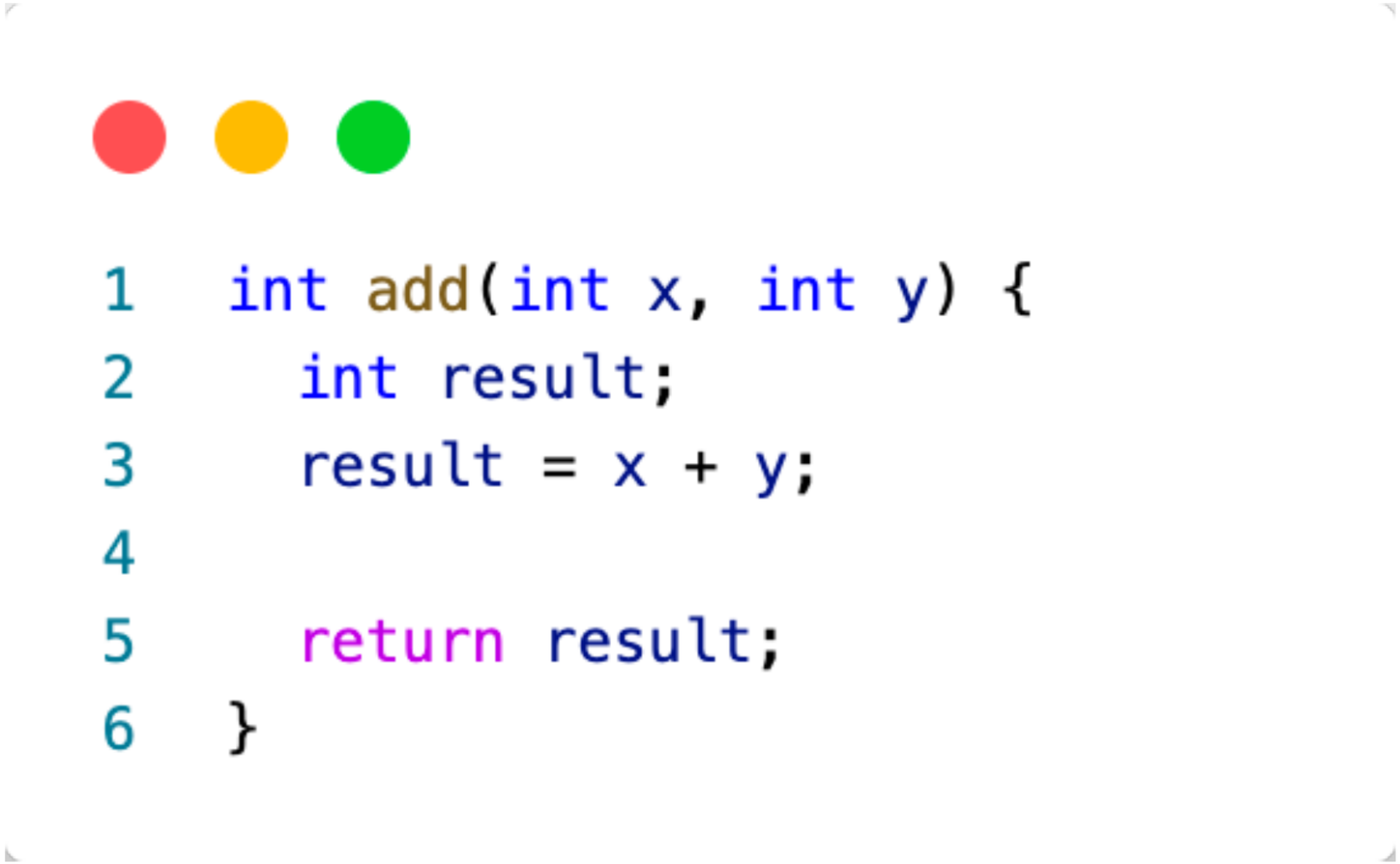
- Prednji deo
  - clang/
- Srednji deo
  - LLVM IR
  - llvm/lib/Transforms/
- Zadnji deo
  - SelectionDAG/GlobalIsel/FastIsel
  - LLVM MIR
  - AsmPrinter



Međukod - LLVM IR

- LLVM Intermediate Representation
  - Arhitekturalno nezavistan
  - Nezavistan od višeg programskog jezika
  - Pogodan za manipulaciju koristeći alate (eng. Tooling)
  - SSA forma
  - Čitljiv

Međukod - LLVM IR



```
1   int add(int x, int y) {
2     int result;
3     result = x + y;
4
5     return result;
6   }
```

```
$ clang -O0 add.c -S -emit-llvm
```

Međukod - LLVM IR

```
1   ; Function Attrs: noinline nounwind optnone ssp uwtable
2   define i32 @add(i32 %0, i32 %1) #0 {
3     %3 = alloca i32, align 4
4     %4 = alloca i32, align 4
5     %5 = alloca i32, align 4
6     store i32 %0, i32* %3, align 4
7     store i32 %1, i32* %4, align 4
8     %6 = load i32, i32* %3, align 4
9     %7 = load i32, i32* %4, align 4
10    %8 = add nsw i32 %6, %7
11    store i32 %8, i32* %5, align 4
12    %9 = load i32, i32* %5, align 4
13    ret i32 %9
14  }
15
```

LLVM IR API

- llvm::Module
- llvm::Function
- llvm::Instruction
- llvm::Value
- Passes
  - ModulePass
  - FunctionPass
  - LoopPass
  - …

```
1  class ConstraintElimination : public FunctionPass {
2    public:
3    static char ID;
4    bool runOnFunction(Function &F) override {
5      return eliminateConstraints(F);
6    }
7  }
8
```

LLVM Machine IR

- Novi nivo apstrakcije
- Arhitektura hardvera
- Arm, Mips, X86…
- Registri
- Memorija

LLVM Machine IR

```
# Machine code for function add: NoPHIs
Frame Objects:
  fi#0: size=4, align=4, at location [SP]
  fi#1: size=4, align=4, at location [SP]
  fi#2: size=4, align=4, at location [SP]
Function Live Ins: $w0, $w1

bb.1 (%ir-block.2):
  liveins: $w0, $w1
  STRWui killed renamable $w0, %stack.0, 0
  STRWui killed renamable $w1, %stack.1, 0
  renamable $w8 = LDRWui %stack.0, 0
  renamable $w9 = LDRWui %stack.1, 0
  renamable $w8 = nsw ADDWrr killed renamable $w8, killed renamable $w9
  STRWui killed renamable $w8, %stack.2, 0
  renamable $w0 = LDRWui %stack.2, 0
  RET_ReallyLR implicit killed $w0

# End machine code for function add.

```

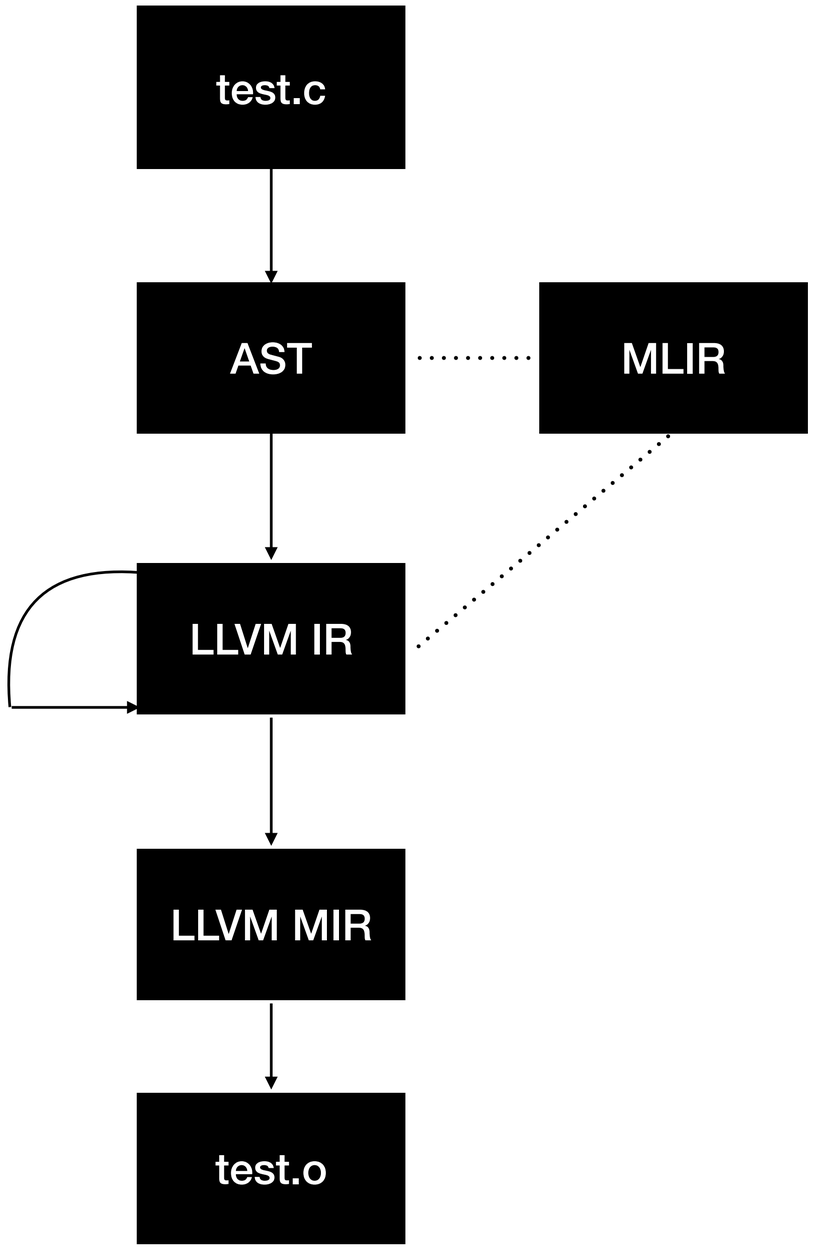
LLVM MIR API

- llvm::MachineFunction
- llvm::MachineInstruction
- Passes
  - llvm::MachineFunctionPass
  - llvm::MachineModulePass

```
1  class LiveDebugValues : public MachineFunctionPass {
2  public:
3    static char ID;
4
5    bool runOnMachineFunction(MachineFunction &MF) override {
6      ...
7    }
8  }
```

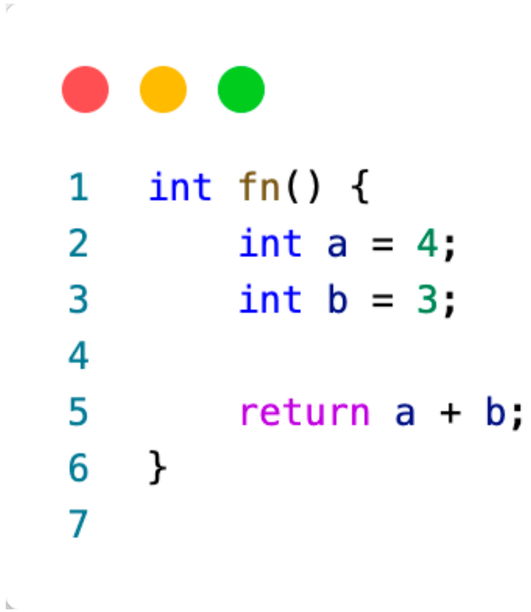
MLIR

- Multi-level Intermediate representation
- Novi nivo apstrakcije
- Podiže robusnost
- Domenski specifični problemi
- Dijalekti
  - CIR (https://github.com/llvm/clangir)
  - Optimizacije vezane za sam programski jezik (push_back vs emplace_back)

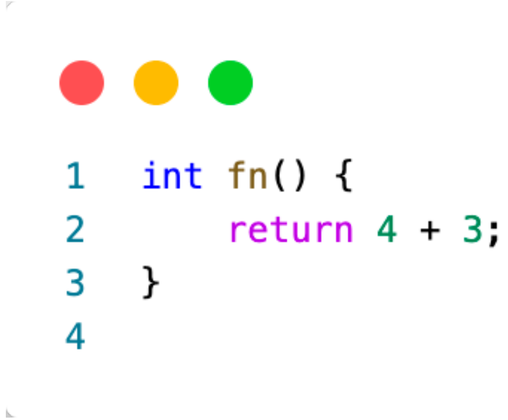


Optimizacije

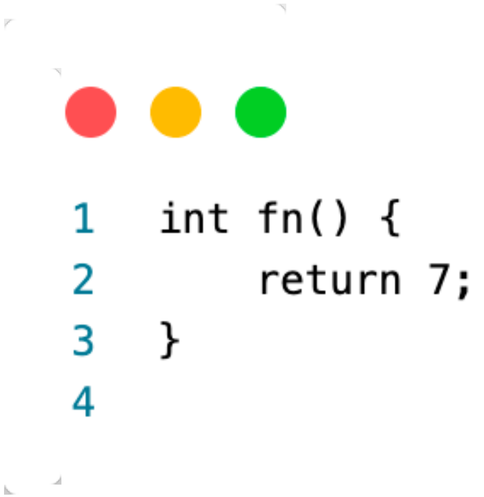
Constant propagation



```
1   int fn() {
2       int a = 4;
3       int b = 3;
4
5       return a + b;
6   }
7
```



```
1   int fn() {
2       return 4 + 3;
3   }
4
```



```
1   int fn() {
2       return 7;
3   }
4
```

- clang AST

Dead Code Elimination

- LLVM IR
- llvm-project/llvm/lib/Transforms/Scalar/DCE.cpp

```
define i32 @add(i32 %0, i32 %1) {
  %res = add i32 4, i32 6
  ret i32 10
}

```

```
static bool DCEInstruction() {
  if (isInstructionTriviallyDead(I, TLI))
    I->eraseFromParent();
  ...
}

```

```
define i32 @add() {
  ret i32 10
}

```

Optimizacije vezane za Arhitekturu

- LLVM Machine IR
- llvm-project/llvm/lib/CodeGen/*
- llvm-project/llvm/lib/Target/*
- X86:
  - Mikroarhitektura
  - xor brža i manje memorije zauzima

mov $eax, 0 —> xor $eax, $eax

Naši projekti bazirani na LLVM-u?

crash-analyzer

- Automatska detekcija uzroka problema (debagovanje uz pomoć debagera vs kompajler)
- Bazirano na llvm-u i lldb-u: https://github.com/cisco-open/llvm-crash-analyzer
- Komponente
  - Decompiler
  - CorefileReader
  - TaintAnalysis
    - Concrete Reverse Execution
    - Register Equivalence

crash-analyzer - primer

```c
void f() {
  T p;
  p.fn = NULL; // blame point
  g(&p);
}

int main() {
 f();
 return 0;
}

```

```c
void h(int *r) {
  *r = 3; // crash
}

void g (T*q) {
  int *t = q->fn;
  h(t);
}

```

Backtrace:

```
#0  h (r=0x0) at test.c:7
#1  g (q=0x7ffd71b147e8) at test.c:12
#2  f () at test.c:20
#3  main () at test.c:24
```

crash-analyzer - komponente

- CoreReader - komponenta koja čita informacije o registrima, memoriji itd. iz datoteke *corefile*
  - Koristi *libLLDB*
- Decompiler
  - llvm::MachineModule
  - Dodaci: crash-start, register-values MF atribute, informacije za debagovanje
- Analiza
  - analiza unazad (Backward, post order)
  - Pomoćne informacije dobijamo iz ConcreteReverseExec i Register Equivalance (xor eax, eax) - analize unapred (forward, reverse post order)

crash-analyzer

```
$ llvm-crash-analyzer --core-file=core.base-case.40698 base-case --print-potential-crash-cause-
Crash Analyzer -- crash analyzer utility

Loading core-file core.base-case.40698
core-file processed.

Decompiling...
Decompiling f
Decompiling g
Decompiling h
Decompiling main
Decompiled.

Analyzing...

Blame Function is f
From File test0.c:18:8
```

Debugify Original

- Testiranje debug informacija tokom transformacija
- Cuvanje debug informacije u optimizpvanom kodu je izazovno
- Automatsko trazenje “krivca”
- https://llvm.org/docs/HowToUpdateDebugInfo.html#test-original-debug-info-preservation-in-optimizations
- https://djolertrk.github.io/2021/07/13/test-debug-info-preserve.html

```
djtodorovic@Djordjes-MacBook-Pro predavanje_matf % clang -Xclang -fverify-debuginfo-preserve -O2 -g add.c
Annotation2Metadata: PASS
Force set function attributes: PASS
Infer set function attributes: PASS
Interprocedural Sparse Conditional Constant Propagation: PASS
Called Value Propagation: PASS
Global Variable Optimizer: PASS
Promote Memory to Register: PASS
Dead Argument Elimination: PASS
Combine redundant instructions: PASS
Simplify the CFG: PASS
Globals Alias Analysis: PASS
SROA: PASS
Early CSE w/ MemorySSA: PASS
Speculatively execute instructions if target has divergent branches: PASS
Jump Threading: PASS
Value Propagation: PASS
Simplify the CFG: PASS
```

Alati

- llvm-locstats
  - https://llvm.org/docs/CommandGuide/llvm-locstats.html

```
$ llvm-locstats a.out
```

```
  ================================================
           Debug Location Statistics
  ================================================
       cov%          samples       percentage(~)
  ------------------------------------------------
     0%                    1               16%
     (0%,10%)              0                0%
     [10%,20%)             0                0%
     [20%,30%)             0                0%
     [30%,40%)             0                0%
     [40%,50%)             0                0%
     [50%,60%)             1               16%
     [60%,70%)             0                0%
     [70%,80%)             0                0%
     [80%,90%)             1               16%
     [90%,100%)            0                0%
     100%                  3               50%
  ================================================
  -the number of debug variables processed: 6
  -PC ranges covered: 81%
  ------------------------------------------------
  -total availability: 83%
  ================================================
```

- llvm-dwarfdump —show-section-sizes
  - https://llvm.org/docs/CommandGuide/llvm-dwarfdump.html#cmdoption-llvm-dwarfdump-show-section-sizes

```
$ llvm-dwarfdump —show-section-sizes a.out
SECTION       SIZE (b)
-------------       --------
.debug_info         525 (2.84%)
.debug_loc          164 (0.89%)
.debug_abbrev       246 (1.33%)
...
 Total Size: 1298  (7.03%)
 Total File Size: 18472
```

Ostalo

- Optimizacije
  - RISCV (cttz)
  - Mips (load/store multiple)
  - Arm (load hoist in licm optimisation) - https://www.phoronix.com/news/LLVM-Clang-14-Hoist-Load
    - Uvrštena u 20 najbitnih izmena za LLVM 14
  - …

Kompajler GNU GCC

- GCC
  - GIMPLE
  - RTL (Register Transfer Language)
    - Lisp izmešan sa C
- Optimizacije
  - MIPS
    - NanoMIPS
      - Raspoređivač instrukcija
  - RISCV
    - -ivopts
    - -cttz-lower
    - …

HVALA!

SYRMIA