LLVM OPTIMIZACIJE

Miona Sretenović
Ognjen Marković

CONSTANT PROPAGATION

- Cilj: zamena promenljivih njihovim konstantnim vrednostima
- Model: svaka promenljiva može biti Top (nepoznata), Const (poznata vrednost), ili Bottom (nedostižna)
- Algoritam:
  - Formiranje grafa instrukcija (CPI čvorovi + prethodnici)
  - Pravila (1–8) za širenje statusa kroz CFG
  - Iterativno dok se stanje ne stabilizuje.
- Modifikacija IR: zamena load/operacija konstantnim vrednostima

PRIMER

```
int main() {
    int a = 5;
    int b = a + 4;
    return b;
}
```

```
%1 = alloca i32, align 4
%2 = alloca i32, align 4
%3 = alloca i32, align 4
store i32 0, ptr %1, align 4
store i32 5, ptr %2, align 4
%4 = load i32, ptr %2, align 4
%5 = add nsw i32 %4, 4
store i32 %5, ptr %3, align 4
%6 = load i32, ptr %3, align 4
ret i32 %6
```

```
%1 = alloca i32, align 4
%2 = alloca i32, align 4
%3 = alloca i32, align 4
store i32 0, ptr %1, align 4
store i32 5, ptr %2, align 4
%4 = load i32, ptr %2, align 4
%5 = add nsw i32 5, 4
store i32 %5, ptr %3, align 4
%6 = load i32, ptr %3, align 4
ret i32 %6
```

CONSTANT FOLDING

- Cilj: evaluacija izraza sa konstantama već u kompajliranju

- Algoritam:
  - Prolazak kroz sve instrukcije
  - Ako je binarna operacija sa konstantnim operandima → izračunaj vrednost
  - Ako je poređenje sa konstantama → evaluiraj odmah
  - Ako je uslovni skok sa konstantnim uslovom → zameni direktno pravim successor blokom

- Rezultat: jednostavniji CFG i manje nepotrebnih izraza

PRIMER

```
int main() {
    int a = 5;
    int b = 5 + 4;
    return b;
}
```

```
%1 = alloca i32, align 4
%2 = alloca i32, align 4
%3 = alloca i32, align 4
store i32 0, ptr %1, align 4
store i32 5, ptr %2, align 4
%4 = load i32, ptr %2, align 4
%5 = add nsw i32 5, 4
store i32 %5, ptr %3, align 4
%6 = load i32, ptr %3, align 4
ret i32 %6
```

```
%1 = alloca i32, align 4
%2 = alloca i32, align 4
%3 = alloca i32, align 4
store i32 0, ptr %1, align 4
store i32 5, ptr %2, align 4
%4 = load i32, ptr %2, align 4
%5 = add nsw i32 5, 4
store i32 9, ptr %3, align 4
%6 = load i32, ptr %3, align 4
ret i32 %6
```

DEAD INSTRUCTION ELIMINATION

- Cilj: uklanjanje instrukcija koje ne utiču na ishod programa

- Algoritam:
  - Mapiranje svih instrukcija → označi koje se koriste, koje ne
  - Instrukcija bez efekta i bez upotrebe je kandidat za brisanje
  - Posebno: store se uklanja ako se promenljiva nikad ne čita
  - Iterativno ponavljanje dok nema više mrtvih instrukcija

- Rezultat: smanjen broj nepotrebnih operacija

PRIMER

```
int main() {
    int a = 5;
    int b = 10;
    int c = a + b;

    int d = 20;

    if (c > 10)
        printf( "c>10");

    return 0;
}
```

```
%1 = alloca i32, align 4
%2 = alloca i32, align 4
%3 = alloca i32, align 4
%4 = alloca i32, align 4
%5 = alloca i32, align 4
store i32 0, ptr %1, align 4
store i32 5, ptr %2, align 4
store i32 10, ptr %3, align 4
%6 = load i32, ptr %2, align 4
%7 = load i32, ptr %3, align 4
%8 = add nsw i32 %6, %7
store i32 %8, ptr %4, align 4
store i32 20, ptr %5, align 4
%9 = load i32, ptr %4, align 4
%10 = icmp sgt i32 %9, 10
br i1 %10, label %11, label %13
```

```
%1 = alloca i32, align 4
%2 = alloca i32, align 4
%3 = alloca i32, align 4
store i32 5, ptr %1, align 4
store i32 10, ptr %2, align 4
%4 = load i32, ptr %1, align 4
%5 = load i32, ptr %2, align 4
%6 = add nsw i32 %4, %5
store i32 %6, ptr %3, align 4
%7 = load i32, ptr %3, align 4
%8 = icmp sgt i32 %7, 10
br i1 %8, label %9, label %11
```

DOMINATORI

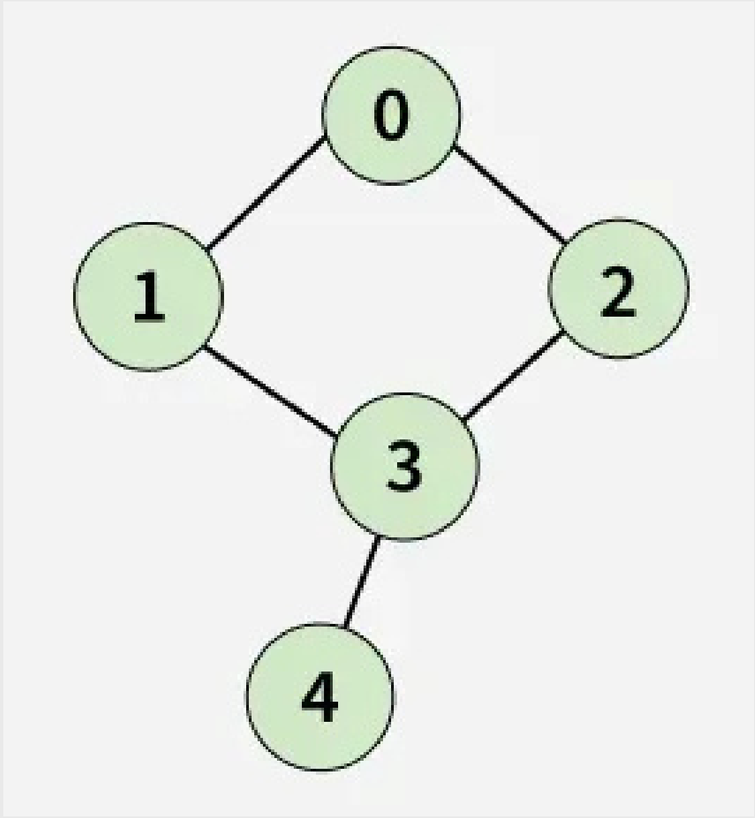
- Blok A dominira blok B ako svaki put od ulaznog (entry) bloka do B prolazi kroz A

- Common Subexpression Elimination (CSE):
  - Ako instrukcija I1 dominira I2 i daju isti rezultat, onda ne moramo ponovo računati I2, već propagiramo vrednost (uvek se izvrši pre nje)

- Dead Basic Block Elimination (DBBE):
  - Ako entry blok ne dominira blok B, taj blok se nikada ne može dostići → bezbedno ga brišemo

- Dominatori nam garantuju redosled izvršavanja i omogućavaju da bezbedno brišemo ili spajamo instrukcije i blokove

DOMINATORI



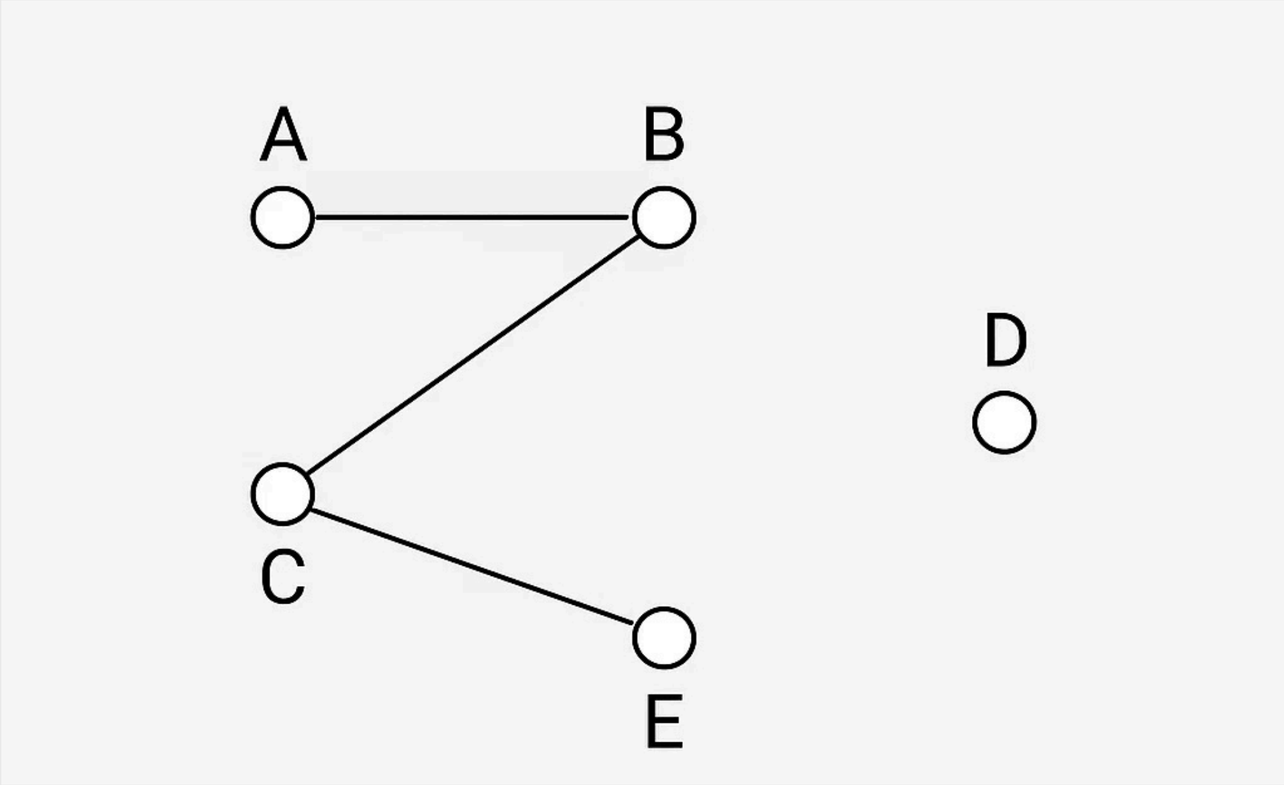
DEAD BASIC BLOCK ELIMINATION

- Cilj: brisanje blokova koda koji se nikad ne izvršavaju

- Algoritam:
  - DFS od entry bloka → označavanje dostižnih blokova
  - Ako blok nije posećen ili ga entry ne dominira → mrtav je
  - Brišu se instrukcije i sam blok iz CFG-a

- Rezultat: čist CFG, bez nedostižnih delova

DEAD BASIC BLOCK ELIMINATION



PRIMER

```
int main() {
  printf("Reachable block\n");
  goto skip;
  dead_block:
    printf("Unreachable block\n");
 skip:
 return 0;
}
```

```
 %1 = alloca i32, align 4
 store i32 0, ptr %1, align 4
%2 = call i32 (ptr, ...) @printf(ptr noundef @.str)
 br label %3
3:                                ; preds = %0
 ret i32 0
```

```
 %1 = alloca i32, align 4
 store i32 0, ptr %1, align 4
 %2 = call i32 (ptr, ...) @printf(ptr noundef @.str)
 br label %5
3:                                ; No predecessors!
 %4 = call i32 (ptr, ...) @printf(ptr noundef @.str.1)
 br label %5
5:                                ; preds = %3, %0
 ret i32 0
```

COMMON SUBEXPRESSION ELIMINATION

- Cilj: izbegavanje višestrukog računanja istog izraza

- Algoritam:
  - Mapiranje izraza (opcode + operandi) na prvu instrukciju koja ga je izračunala
  - Ako isti izraz naiđe ponovo i prethodna instrukcija dominira → zameni sve upotrebe sa ranijom
  - Za load: poslednji validni load se pamti, kasniji se mogu zameniti ako nema intervenišućeg store

- Rezultat: manji broj duplikata i optimizovan kod

PRIMER

```
int main() {
  int a = 5, b = 7;

  int x = a;
  int y = a;
  int z = b;
  b = 10;
  int w = b;
  int u = x + y;
  int v = y + x;

  printf("%d\n", u + v + z + w);

  return 0;
}
```

PRIMER

```
%1 = alloca i32, align 4
%2 = alloca i32, align 4
%3 = alloca i32, align 4
%4 = alloca i32, align 4
%5 = alloca i32, align 4
%6 = alloca i32, align 4
%7 = alloca i32, align 4
%8 = alloca i32, align 4
%9 = alloca i32, align 4
store i32 0, ptr %1, align 4
store i32 5, ptr %2, align 4
store i32 7, ptr %3, align 4
%10 = load i32, ptr %2, align 4
store i32 %10, ptr %4, align 4
%11 = load i32, ptr %2, align 4
store i32 %11, ptr %5, align 4
%12 = load i32, ptr %3, align 4
```

```
%1 = alloca i32, align 4
%2 = alloca i32, align 4
%3 = alloca i32, align 4
%4 = alloca i32, align 4
%5 = alloca i32, align 4
%6 = alloca i32, align 4
%7 = alloca i32, align 4
%8 = alloca i32, align 4
%9 = alloca i32, align 4
store i32 0, ptr %1, align 4
store i32 5, ptr %2, align 4
store i32 7, ptr %3, align 4
%10 = load i32, ptr %2, align 4
store i32 %10, ptr %4, align 4
store i32 %10, ptr %5, align 4
%11 = load i32, ptr %3, align 4
store i32 %11, ptr %6, align 4
store i32 10, ptr %3, align 4
```

PRIMER

```
store i32 %12, ptr %6, align 4
store i32 10, ptr %3, align 4
%13 = load i32, ptr %3, align 4
store i32 %13, ptr %7, align 4
%14 = load i32, ptr %4, align 4
%15 = load i32, ptr %5, align 4
%16 = add nsw i32 %14, %15
store i32 %16, ptr %8, align 4
%17 = load i32, ptr %5, align 4
%18 = load i32, ptr %4, align 4
%19 = add nsw i32 %17, %18
store i32 %19, ptr %9, align 4
%20 = load i32, ptr %8, align 4
%21 = load i32, ptr %9, align 4
%22 = add nsw i32 %20, %21
%23 = load i32, ptr %6, align 4
%24 = add nsw i32 %22, %23
%25 = load i32, ptr %7, align 4
%26 = add nsw i32 %24, %25
%27 = call i32 (ptr, ...) @printf(...)
ret i32 0
```

```
%12 = load i32, ptr %3, align 4
store i32 %12, ptr %7, align 4
%13 = load i32, ptr %4, align 4
%14 = load i32, ptr %5, align 4
%15 = add nsw i32 %13, %14
store i32 %15, ptr %8, align 4
store i32 %15, ptr %9, align 4
%16 = load i32, ptr %8, align 4
%17 = load i32, ptr %9, align 4
%18 = add nsw i32 %16, %17
%19 = load i32, ptr %6, align 4
%20 = add nsw i32 %18, %19
%21 = load i32, ptr %7, align 4
%22 = add nsw i32 %20, %21
%23 = call i32 (ptr, ...) @printf(...)
ret i32 0
```