Елиминација заједничких подизраза (LLVM)

Конструкција компилатора

Математички факултет

Аутори:
Александар Зечевић
Иван Гогић

Увод

Шта је заједнички подизраз?

```
int a, b, c, d, e;
d = a + b * c;
e = b * c + d;
```

```
int tmp;
tmp = b * c;
d = a + tmp;
e = tmp + d;
```

Увод

Шта је заједнички подизраз?

```
if (a < b) {}
if (b > a) {}
```

```
bool tmp;
tmp = a < b;
if (tmp) {}
if (tmp) {}
```

Увод

Шта није заједнички подизраз?

```
d = a * (b + c);
e = a * b + a * c;
```

Увод

Шта није заједнички подизраз?

- Семантички еквиваленти? GVN
- …, али лексички нису.

```
d = a * b;
c = a;
e = c * b;
```

Увод

…, и овде постоји заједнички подизраз.

```
int arr[256];
*(arr + i) = *(arr + i) + 5;
// arr[i] = arr[i] + 5;
```

```
int* tmp;
tmp = &arr[i];
*tmp = *tmp + 5;
```

Увод

- Закључак: Зашто настају заједнички подизрази?
  - Понављања у изворном коду
  - Изворни код је написан на језику високог нивоа, заједнички подизрази „скривени“ иза апстракција које језик пружа

CSE – LLVM оптимизација

```
do {
  gcse.runAnalysis();
} while (gcse.runPass());
```

CSE – LLVM оптимизација

- Expression
- LCSE
- GCSE

Expression = омотач инструкција

```
enum ExpressionType {
  BINARY_OP,
  CMP_INST,
  UNARY_OP,
  CAST_OP,
  LOAD_INST,
  GETELEMENTPTR_INST,
  SELECT_INST
};
```

```
struct Expression {
  ExpressionType ExprTyp;
  unsigned Opcode;
  Type* Typ;
  bool IsCommutative;
  std::vector<Value*> Args;
  Value* Result;

  Expression(Instruction& Inst);
  Expression() = default;

  bool operator ==(const Expression& Other) const;
  std::size_t hash() const;
};
```

Expression

- Када су изрази једнаки?

  - Лексички идентични

  - a + b == b + a

  - a < b == b > a

LCSE

- CSE на нивоу блока → није неопходан (GCSE може сам све да покрије), али смањује број инструкција које се разматрају даље у GCSE и олакшава логику у имплементацији GCSE

LCSE - имплементација

- Линеаран пролазак кроз блок

  -уколико је инструкција store, из листе тренутно доступних израза бришемо оне који као операнд имају ту вредност

  -иначе, проверавамо да ли је израз већ доступан, и уколико јесте замењујемо њиме, у супротном убацујемо тренутни израз у скуп доступних

LCSE - пример

```c
#include <stdio.h>

int main() {
    int x, y, z, w, dj;

    x = z + w * dj;
    y = z + w * dj;

    int a1 = z + w;
    int a2 = w + z;

    int a[10];
    int i = 5;
    int e1 = a[i] + 4;
    int e2 = a[i] / 2;
    int e3 = a[i] + a[i] * a[i];
    int e4 = a[i] * a[i] - 11;

    return 0;
}
```

```
; Function Attrs: noinline nounwind optnone ssp uwtable
define i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %x = alloca i32, align 4
  %y = alloca i32, align 4
  %z = alloca i32, align 4
  %w = alloca i32, align 4
  %dj = alloca i32, align 4
  %a1 = alloca i32, align 4
  %a2 = alloca i32, align 4
  %a = alloca [10 x i32], align 16
  %i = alloca i32, align 4
  %e1 = alloca i32, align 4
  %e2 = alloca i32, align 4
  %e3 = alloca i32, align 4
  %e4 = alloca i32, align 4
  store i32 0, ptr %retval, align 4
  %0 = load i32, ptr %z, align 4
  %1 = load i32, ptr %w, align 4
  %2 = load i32, ptr %dj, align 4
  %mul = mul nsw i32 %1, %2
  %add = add nsw i32 %0, %mul
  store i32 %add, ptr %x, align 4
  %3 = load i32, ptr %z, align 4
  %4 = load i32, ptr %w, align 4
  %5 = load i32, ptr %dj, align 4
  %mul1 = mul nsw i32 %4, %5
  %add2 = add nsw i32 %3, %mul1
  store i32 %add2, ptr %y, align 4
  %6 = load i32, ptr %z, align 4
  %7 = load i32, ptr %w, align 4
  %add3 = add nsw i32 %6, %7
  store i32 %add3, ptr %a1, align 4
  %8 = load i32, ptr %w, align 4
  %9 = load i32, ptr %z, align 4
  %add4 = add nsw i32 %8, %9
  store i32 %add4, ptr %a2, align 4
  store i32 5, ptr %i, align 4
  %10 = load i32, ptr %i, align 4
  %idxprom = sext i32 %10 to i64
  %arrayidx = getelementptr inbounds [10 x i32], ptr %a, i64 0, i64 %idxprom
  %11 = load i32, ptr %arrayidx, align 4
  %add5 = add nsw i32 %11, 4
  store i32 %add5, ptr %e1, align 4
  %12 = load i32, ptr %i, align 4
  %idxprom6 = sext i32 %12 to i64
  %arrayidx7 = getelementptr inbounds [10 x i32], ptr %a, i64 0, i64 %idxprom6
  %13 = load i32, ptr %arrayidx7, align 4
  %div = sdiv i32 %13, 2
  store i32 %div, ptr %e2, align 4
  %14 = load i32, ptr %i, align 4
  %idxprom8 = sext i32 %14 to i64
  %arrayidx9 = getelementptr inbounds [10 x i32], ptr %a, i64 0, i64 %idxprom8
  %15 = load i32, ptr %arrayidx9, align 4
  %16 = load i32, ptr %i, align 4
  %idxprom10 = sext i32 %16 to i64
  %arrayidx11 = getelementptr inbounds [10 x i32], ptr %a, i64 0, i64 %idxprom10
  %17 = load i32, ptr %arrayidx11, align 4
  %18 = load i32, ptr %i, align 4
  %idxprom12 = sext i32 %18 to i64
  %arrayidx13 = getelementptr inbounds [10 x i32], ptr %a, i64 0, i64 %idxprom12
  %19 = load i32, ptr %arrayidx13, align 4
  %mul14 = mul nsw i32 %17, %19
  %add15 = add nsw i32 %15, %mul14
  store i32 %add15, ptr %e3, align 4
  %20 = load i32, ptr %i, align 4
  %idxprom16 = sext i32 %20 to i64
  %arrayidx17 = getelementptr inbounds [10 x i32], ptr %a, i64 0, i64 %idxprom16
  %21 = load i32, ptr %arrayidx17, align 4
  %22 = load i32, ptr %i, align 4
  %idxprom18 = sext i32 %22 to i64
  %arrayidx19 = getelementptr inbounds [10 x i32], ptr %a, i64 0, i64 %idxprom18
  %23 = load i32, ptr %arrayidx19, align 4
  %mul20 = mul nsw i32 %21, %23
  %sub = sub nsw i32 %mul20, 11
  store i32 %sub, ptr %e4, align 4
  ret i32 0
}
```

LCSE - пример

```c
#include <stdio.h>

int main() {
    int x, y, z, w, dj;

    x = z + w * dj;
    y = z + w * dj;

    int a1 = z + w;
    int a2 = w + z;

    int a[10];
    int i = 5;
    int e1 = a[i] + 4;
    int e2 = a[i] / 2;
    int e3 = a[i] + a[i] * a[i];
    int e4 = a[i] * a[i] - 11;

    return 0;
}
```

```
; Function Attrs: noinline nounwind optnone ssp uwtable
define i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %x = alloca i32, align 4
  %y = alloca i32, align 4
  %z = alloca i32, align 4
  %w = alloca i32, align 4
  %dj = alloca i32, align 4
  %a1 = alloca i32, align 4
  %a2 = alloca i32, align 4
  %a = alloca [10 x i32], align 16
  %i = alloca i32, align 4
  %e1 = alloca i32, align 4
  %e2 = alloca i32, align 4
  %e3 = alloca i32, align 4
  %e4 = alloca i32, align 4
  store i32 0, ptr %retval, align 4
  %0 = load i32, ptr %z, align 4
  %1 = load i32, ptr %w, align 4
  %2 = load i32, ptr %dj, align 4
  %mul = mul nsw i32 %1, %2
  %add = add nsw i32 %0, %mul
  store i32 %add, ptr %x, align 4
  store i32 %add, ptr %y, align 4
  %add3 = add nsw i32 %0, %1
  store i32 %add3, ptr %a1, align 4
  store i32 %add3, ptr %a2, align 4
  store i32 5, ptr %i, align 4
  %3 = load i32, ptr %i, align 4
  %idxprom = sext i32 %3 to i64
  %arrayidx = getelementptr inbounds [10 x i32], ptr %a, i64 0, i64 %idxprom
  %4 = load i32, ptr %arrayidx, align 4
  %add5 = add nsw i32 %4, 4
  store i32 %add5, ptr %e1, align 4
  %div = sdiv i32 %4, 2
  store i32 %div, ptr %e2, align 4
  %mul14 = mul nsw i32 %4, %4
  %add15 = add nsw i32 %4, %mul14
  store i32 %add15, ptr %e3, align 4
  %sub = sub nsw i32 %mul14, 11
  store i32 %sub, ptr %e4, align 4
  ret i32 0
}
```

LCSE - пример

- У наведеном примеру смањили смо број инструкција у IR са 75 на свега 40, што је заиста велика разлика

GCSE - анализа

- Прелазак на GCSE – требају нам изрази доступни на улазу у блокове

  -Анализа тока података → генералан поступак, у овом случају имамо анализу унапред

  -За сваки блок прво рачунамо скупове евалуираних и убијених израза (у односу на универзални скуп израза)

  -На основу њих правимо једначине анализе тока

GCSE – доступни изрази

- За сваки блок рачунамо скупове израза доступних на почетку (IN) и крају блока (OUT)

  -OUT скупови иницијализују се на универзални скуп

  -Једначине анализе тока:

$$\text{OUT}[B] = e\_gen_B \cup (\text{IN}[B] - e\_kill_B)$$

$$\text{IN}[B] = \bigcap_{P \text{ a predecessor of } B} \text{OUT}[P]$$

GCSE – доступни изрази

- Псеудокод целокупног алгоритма:

$\text{OUT}[\text{ENTRY}] = \emptyset;$
for (each basic block B other than ENTRY) $\text{OUT}[B] = U;$
while (changes to any OUT occur)
    for (each basic block B other than ENTRY) {
        $\text{IN}[B] = \bigcap_{P \text{ a predecessor of } B} \text{OUT}[P];$
        $\text{OUT}[B] = e\_gen_B \cup (\text{IN}[B] - e\_kill_B);$
    }

GCSE – замена израза

- Традиционални приступ: уводимо привремену променљиву у којој чувамо последње израчунате вредности израза на свим путањама
- LLVM: SSA форма

GCSE – замена израза

1. Израз А доминира изразом Б и израз А није убијен ни на једној путањи од А до Б.

2. Замена је доступна на излазу сваког блока претходника. Пронаћи их, и „споји“ phi инструкцијом.

GCSE – замена израза (пример 1)

```c
int main()
{
  int a, b, c, d;
  c = a + b;
  if (a > 0) {}
  else {}
  d = a + b;
}
```

```
define dso_local i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  %d = alloca i32, align 4
  store i32 0, ptr %retval, align 4
  %0 = load i32, ptr %a, align 4
  %1 = load i32, ptr %b, align 4
  %add = add nsw i32 %0, %1
  store i32 %add, ptr %c, align 4
  %2 = load i32, ptr %a, align 4
  %cmp = icmp sgt i32 %2, 0
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  br label %if.end

if.else:                                          ; preds = %entry
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %3 = load i32, ptr %a, align 4
  %4 = load i32, ptr %b, align 4
  %add1 = add nsw i32 %3, %4
  store i32 %add1, ptr %d, align 4
  %5 = load i32, ptr %retval, align 4
  ret i32 %5
}
```

GCSE – замена израза (пример 1)

```
define dso_local i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  %d = alloca i32, align 4
  store i32 0, ptr %retval, align 4
  %0 = load i32, ptr %a, align 4
  %1 = load i32, ptr %b, align 4
  %add = add nsw i32 %0, %1
  store i32 %add, ptr %c, align 4
  %2 = load i32, ptr %a, align 4
  %cmp = icmp sgt i32 %2, 0
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  br label %if.end

if.else:                                          ; preds = %entry
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %3 = load i32, ptr %a, align 4
  %4 = load i32, ptr %b, align 4
  %add1 = add nsw i32 %3, %4
  store i32 %add1, ptr %d, align 4
  %5 = load i32, ptr %retval, align 4
  ret i32 %5
}
```

```
define dso_local i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  %d = alloca i32, align 4
  store i32 0, ptr %retval, align 4
  %0 = load i32, ptr %a, align 4
  %1 = load i32, ptr %b, align 4
  %add = add nsw i32 %0, %1
  store i32 %add, ptr %c, align 4
  %cmp = icmp sgt i32 %0, 0
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  br label %if.end

if.else:                                          ; preds = %entry
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %add1 = add nsw i32 %0, %1
  store i32 %add1, ptr %d, align 4
  %2 = load i32, ptr %retval, align 4
  ret i32 %2
}
```

GCSE – замена израза (пример 1)

```
define dso_local i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  %d = alloca i32, align 4
  store i32 0, ptr %retval, align 4
  %0 = load i32, ptr %a, align 4
  %1 = load i32, ptr %b, align 4
  %add = add nsw i32 %0, %1
  store i32 %add, ptr %c, align 4
  %cmp = icmp sgt i32 %0, 0
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  br label %if.end

if.else:                                          ; preds = %entry
  br label %if.end

if.end:                                           ; preds = %if.else, %if.th
  %add1 = add nsw i32 %0, %1
  store i32 %add1, ptr %d, align 4
  %2 = load i32, ptr %retval, align 4
  ret i32 %2
}
```

```
define dso_local i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  %d = alloca i32, align 4
  store i32 0, ptr %retval, align 4
  %0 = load i32, ptr %a, align 4
  %1 = load i32, ptr %b, align 4
  %add = add nsw i32 %0, %1
  store i32 %add, ptr %c, align 4
  %cmp = icmp sgt i32 %0, 0
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  br label %if.end

if.else:                                          ; preds = %entry
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  store i32 %add, ptr %d, align 4
  %2 = load i32, ptr %retval, align 4
  ret i32 %2
}
```

GCSE – замена израза (пример 2a)

```c
int main()
{
    int a, b, c, d;
    if (a > 0)
        c = a + b;
    else
        d = b + a;
    a = a + b;
}
```

```
define dso_local i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  %d = alloca i32, align 4
  store i32 0, ptr %retval, align 4
  %0 = load i32, ptr %a, align 4
  %cmp = icmp sgt i32 %0, 0
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  %1 = load i32, ptr %a, align 4
  %2 = load i32, ptr %b, align 4
  %add = add nsw i32 %1, %2
  store i32 %add, ptr %c, align 4
  br label %if.end

if.else:                                          ; preds = %entry
  %3 = load i32, ptr %b, align 4
  %4 = load i32, ptr %a, align 4
  %add1 = add nsw i32 %3, %4
  store i32 %add1, ptr %d, align 4
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %5 = load i32, ptr %a, align 4
  %6 = load i32, ptr %b, align 4
  %add2 = add nsw i32 %5, %6
  store i32 %add2, ptr %a, align 4
  %7 = load i32, ptr %retval, align 4
  ret i32 %7
}
```

GCSE – замена израза (пример 2a)

```
define dso_local i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  %d = alloca i32, align 4
  store i32 0, ptr %retval, align 4
  %0 = load i32, ptr %a, align 4
  %cmp = icmp sgt i32 %0, 0
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  %1 = load i32, ptr %a, align 4
  %2 = load i32, ptr %b, align 4
  %add = add nsw i32 %1, %2
  store i32 %add, ptr %c, align 4
  br label %if.end

if.else:                                          ; preds = %entry
  %3 = load i32, ptr %b, align 4
  %4 = load i32, ptr %a, align 4
  %add1 = add nsw i32 %3, %4
  store i32 %add1, ptr %d, align 4
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %5 = load i32, ptr %a, align 4
  %6 = load i32, ptr %b, align 4
  %add2 = add nsw i32 %5, %6
  store i32 %add2, ptr %a, align 4
  %7 = load i32, ptr %retval, align 4
  ret i32 %7
}
```

```
define dso_local i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  %d = alloca i32, align 4
  store i32 0, ptr %retval, align 4
  %0 = load i32, ptr %a, align 4
  %cmp = icmp sgt i32 %0, 0
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  %1 = load i32, ptr %b, align 4
  %add = add nsw i32 %0, %1
  store i32 %add, ptr %c, align 4
  br label %if.end

if.else:                                          ; preds = %entry
  %2 = load i32, ptr %b, align 4
  %add1 = add nsw i32 %2, %0
  store i32 %add1, ptr %d, align 4
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %3 = phi i32 [ %2, %if.else ], [ %1, %if.then ]
  %add2 = add nsw i32 %0, %3
  store i32 %add2, ptr %a, align 4
  %4 = load i32, ptr %retval, align 4
  ret i32 %4
}
```

GCSE – замена израза (пример 2a)

```
define dso_local i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  %d = alloca i32, align 4
  store i32 0, ptr %retval, align 4
  %0 = load i32, ptr %a, align 4
  %cmp = icmp sgt i32 %0, 0
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  %1 = load i32, ptr %b, align 4
  %add = add nsw i32 %0, %1
  store i32 %add, ptr %c, align 4
  br label %if.end

if.else:                                          ; preds = %entry
  %2 = load i32, ptr %b, align 4
  %add1 = add nsw i32 %2, %0
  store i32 %add1, ptr %d, align 4
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %3 = phi i32 [ %2, %if.else ], [ %1, %if.then ]
  %add2 = add nsw i32 %0, %3
  store i32 %add2, ptr %a, align 4
  %4 = load i32, ptr %retval, align 4
  ret i32 %4
}
```

```
define dso_local i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  %d = alloca i32, align 4
  store i32 0, ptr %retval, align 4
  %0 = load i32, ptr %a, align 4
  %cmp = icmp sgt i32 %0, 0
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  %1 = load i32, ptr %b, align 4
  %add = add nsw i32 %0, %1
  store i32 %add, ptr %c, align 4
  br label %if.end

if.else:                                          ; preds = %entry
  %2 = load i32, ptr %b, align 4
  %add1 = add nsw i32 %2, %0
  store i32 %add1, ptr %d, align 4
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %3 = phi i32 [ %add1, %if.else ], [ %add, %if.then ]
  store i32 %3, ptr %a, align 4
  %4 = load i32, ptr %retval, align 4
  ret i32 %4
}
```

GCSE – замена израза (пример 2б)

```c
int main()
{
  int a, b, c, d;
  c = a + b;
  if (a > 0) {
    a = 2;
    c = a + b;
  } else {
    d = 3;
  }
  d = a + b;
}
```

```
define dso_local i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  %d = alloca i32, align 4
  store i32 0, ptr %retval, align 4
  %0 = load i32, ptr %a, align 4
  %1 = load i32, ptr %b, align 4
  %add = add nsw i32 %0, %1
  store i32 %add, ptr %c, align 4
  %2 = load i32, ptr %a, align 4
  %cmp = icmp sgt i32 %2, 0
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  store i32 2, ptr %a, align 4
  %3 = load i32, ptr %a, align 4
  %4 = load i32, ptr %b, align 4
  %add1 = add nsw i32 %3, %4
  store i32 %add1, ptr %c, align 4
  br label %if.end

if.else:                                          ; preds = %entry
  store i32 3, ptr %d, align 4
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %5 = load i32, ptr %a, align 4
  %6 = load i32, ptr %b, align 4
  %add2 = add nsw i32 %5, %6
  store i32 %add2, ptr %d, align 4
  %7 = load i32, ptr %retval, align 4
  ret i32 %7
}
```

GCSE – замена израза (пример 2б)

```
define dso_local i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  %d = alloca i32, align 4
  store i32 0, ptr %retval, align 4
  %0 = load i32, ptr %a, align 4
  %1 = load i32, ptr %b, align 4
  %add = add nsw i32 %0, %1
  store i32 %add, ptr %c, align 4
  %2 = load i32, ptr %a, align 4
  %cmp = icmp sgt i32 %2, 0
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  store i32 2, ptr %a, align 4
  %3 = load i32, ptr %a, align 4
  %4 = load i32, ptr %b, align 4
  %add1 = add nsw i32 %3, %4
  store i32 %add1, ptr %c, align 4
  br label %if.end

if.else:                                          ; preds = %entry
  store i32 3, ptr %d, align 4
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %5 = load i32, ptr %a, align 4
  %6 = load i32, ptr %b, align 4
  %add2 = add nsw i32 %5, %6
  store i32 %add2, ptr %d, align 4
  %7 = load i32, ptr %retval, align 4
  ret i32 %7
}
```

```
define dso_local i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  %d = alloca i32, align 4
  store i32 0, ptr %retval, align 4
  %0 = load i32, ptr %a, align 4
  %1 = load i32, ptr %b, align 4
  %add = add nsw i32 %0, %1
  store i32 %add, ptr %c, align 4
  %cmp = icmp sgt i32 %0, 0
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  store i32 2, ptr %a, align 4
  %2 = load i32, ptr %a, align 4
  %add1 = add nsw i32 %2, %1
  store i32 %add1, ptr %c, align 4
  br label %if.end

if.else:                                          ; preds = %entry
  store i32 3, ptr %d, align 4
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %3 = phi i32 [ %0, %if.else ], [ %2, %if.then ]
  %add2 = add nsw i32 %3, %1
  store i32 %add2, ptr %d, align 4
  %4 = load i32, ptr %retval, align 4
  ret i32 %4
}
```

GCSE – замена израза (пример 2б)

```
define dso_local i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  %d = alloca i32, align 4
  store i32 0, ptr %retval, align 4
  %0 = load i32, ptr %a, align 4
  %1 = load i32, ptr %b, align 4
  %add = add nsw i32 %0, %1
  store i32 %add, ptr %c, align 4
  %cmp = icmp sgt i32 %0, 0
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  store i32 2, ptr %a, align 4
  %2 = load i32, ptr %a, align 4
  %add1 = add nsw i32 %2, %1
  store i32 %add1, ptr %c, align 4
  br label %if.end

if.else:                                          ; preds = %entry
  store i32 3, ptr %d, align 4
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %3 = phi i32 [ %0, %if.else ], [ %2, %if.then ]
  %add2 = add nsw i32 %3, %1
  store i32 %add2, ptr %d, align 4
  %4 = load i32, ptr %retval, align 4
  ret i32 %4
}
```

```
define dso_local i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  %d = alloca i32, align 4
  store i32 0, ptr %retval, align 4
  %0 = load i32, ptr %a, align 4
  %1 = load i32, ptr %b, align 4
  %add = add nsw i32 %0, %1
  store i32 %add, ptr %c, align 4
  %cmp = icmp sgt i32 %0, 0
  br i1 %cmp, label %if.then, label %if.else

if.then:                                          ; preds = %entry
  store i32 2, ptr %a, align 4
  %2 = load i32, ptr %a, align 4
  %add1 = add nsw i32 %2, %1
  store i32 %add1, ptr %c, align 4
  br label %if.end

if.else:                                          ; preds = %entry
  store i32 3, ptr %d, align 4
  br label %if.end

if.end:                                           ; preds = %if.else, %if.then
  %3 = phi i32 [ %add, %if.else ], [ %add1, %if.then ]
  store i32 %3, ptr %d, align 4
  %4 = load i32, ptr %retval, align 4
  ret i32 %4
}
```

Изворни код

https://github.com/ivangogic/LLVM-CSE

Литература

1. Aho, Alfred; Lam, Monica; Sethi, Ravi; Ullman, Jeffrey (2007). Compilers: Principles, Techniques, and Tools

2. Steven S. Muchnick (1997). Advanced Compiler Design and Implementation