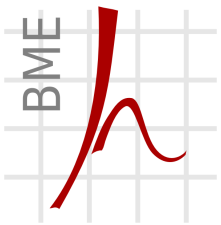
Fordítás – Kódoptimalizálás

Kód visszafejtés.



Híradástechnikai Tanszék

Izsó Tamás

2016. szeptember 29.

Section 1

Közbenső kód ábrázolás

Miért használunk közbenső kódot (Intermediate reprezentáció)

Assembly

- előny – alkalmas az optimalizálásra;
- hátrány – gépfüggő;
- hátrány – minden egyes processzorra újra kellene írni az optimalizálót.

IR

- előny – alkalmas az optimalizálásra;
- előny – gépfüggetlen.

IR fajtái

Forrásprogram és a gépi kód közötti átmenet.

- Magas szintű — eredeti program visszaállítható (kivétel megjegyzések, tagolások). Felhasználják szintaxis vezérelt editorokban, pretty printing funkciókban. (pl. AST). A parser a fordításnál ezt állítja elő;
- Alacsony szintű — minden sora majdnem egy gépi utasításra fordítható;
- Közepes szintű — bizonyos struktúrákat megőriznek a jobb optimalizálás érdekében. Például a magasszintű vezérlési szerkezeteket vagy index operátorokat tartalmaznak.

Reprezentáció:

- gráf alapú (AST, vezérlés-áram gráf, adatáram gráf) ;
- lineáris;
- vegyes.

Intermediate Language (IL)

IL a közbenső ábrázolás (IR) nyelvtani szabályait tartalmazza

- minden fordító saját IL-t használ, akár többet is;
- IL = *magasszintű assembly nyelv*
  - felhasználható regiszterek száma nincs korlátozva;
  - assembly nyelv szintű vezérlési szerkezetek
  - gépi utasításokat használnak, de egyes utasítások lehetnek magasszintűek is
    - pl. call több assembly utasításra fordul
    - legtöbb IL utasítás egy gépi utasításra fordítható

gcc fordító RTL belső ábrázolása

RTL = Register Transfer Language

Fordítási opció:gcc −c −fdump−rtl−all test1.c ⇒
test1.c.165r.expand

```
void fo() {
  int a=0;
  a=a+1;
}
```

```
(insn 5 2 6 2 (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
        (const_int 0 [0])) test1.c:2 -1
     (nil))
(insn 6 5 0 2 (parallel [
            (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                        (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
                (plus:SI (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                            (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
                    (const_int 1 [0x1])))
            (clobber (reg:CC 17 flags))
        ]) test1.c:3 -1
     (nil))
```

gcc fordító RTL belső ábrázolása

RTL = Register Transfer Language

Fordítási opció:gcc −c −fdump−rtl−all test1.c ⇒
test1.c.165r.expand

```
void fo() {
  int a=0;
  a=a+1;
}
```

Aktuális utasítás

```
(insn 5 2 6 2 (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
        (const_int 0 [0])) test1.c:2 -1
     (nil))
(insn 6 5 0 2 (parallel [
            (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                        (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
                (plus:SI (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                            (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
                    (const_int 1 [0x1])))
            (clobber (reg:CC 17 flags))
        ]) test1.c:3 -1
     (nil))
```

gcc fordító RTL belső ábrázolása

RTL = Register Transfer Language

Fordítási opció:gcc −c −fdump−rtl−all test1.c ⇒
test1.c.165r.expand

```
void fo() {
  int a=0;
  a=a+1;
}
```

Előző utasítás

```
(insn 5 2 6 2 (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
        (const_int 0 [0])) test1.c:2 -1
     (nil))
(insn 6 5 0 2 (parallel [
            (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                        (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
                (plus:SI (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                            (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
                    (const_int 1 [0x1])))
            (clobber (reg:CC 17 flags))
        ]) test1.c:3 -1
     (nil))
```

gcc fordító RTL belső ábrázolása

RTL = Register Transfer Language

Fordítási opció:gcc −c −fdump−rtl−all test1.c ⇒
test1.c.165r.expand

```
void fo() {
  int a=0;
  a=a+1;
}
```

Következő utasítás

```
(insn 5 2 6 2 (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
        (const_int 0 [0])) test1.c:2 -1
     (nil))
(insn 6 5 0 2 (parallel [
            (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                        (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
                (plus:SI (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                            (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
                    (const_int 1 [0x1])))
            (clobber (reg:CC 17 flags))
        ]) test1.c:3 -1
     (nil))
```

gcc fordító RTL belső ábrázolása

RTL = Register Transfer Language

Fordítási opció:gcc −c −fdump−rtl−all test1.c ⇒
test1.c.165r.expand

```
void fo() {
  int a=0;
  a=a+1;
}
```

Alapblokk száma

```
(insn 5 2 6 2 (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
        (const_int 0 [0])) test1.c:2 -1
     (nil))
(insn 6 5 0 2 (parallel [
            (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                        (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
                (plus:SI (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                            (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
                    (const_int 1 [0x1])))
            (clobber (reg:CC 17 flags))
        ]) test1.c:3 -1
     (nil))
```

gcc fordító RTL belső ábrázolása

RTL = Register Transfer Language

Fordítási opció:gcc −c −fdump−rtl−all test1.c ⇒
test1.c.165r.expand

```
void fo() {
  int a=0;
  a=a+1;
}
```

értékadás

```
(insn 5 2 6 2 (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
        (const_int 0 [0])) test1.c:2 -1
     (nil))
(insn 6 5 0 2 (parallel [
            (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                        (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
                (plus:SI (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                            (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
                    (const_int 1 [0x1])))
            (clobber (reg:CC 17 flags))
        ]) test1.c:3 -1
     (nil))
```

gcc fordító RTL belső ábrázolása

RTL = Register Transfer Language

Fordítási opció:gcc −c −fdump−rtl−all test1.c ⇒
test1.c.165r.expand

```
void fo() {
  int a=0;
  a=a+1;
}
```

```
(insn 5 2 6 2 (set (mem/c:SI (pl                    :SI 54 virtual-stack-vars)
                (const_int -4 [0                    fffc ])) [0 a+0 S4 A32])
        (const_int 0 [0])) test1.c:2 -1
     (nil))
(insn 6 5 0 2 (parallel [
            (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                        (const_int -4 [0xfffffffffffffffc ])) [0 a+0 S4 A32])
                (plus:SI (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                            (const_int -4 [0xfffffffffffffffc ])) [0 a+0 S4 A32])
                    (const_int 1 [0x1])))
            (clobber (reg:CC 17 flags))
        ]) test1.c:3 -1
     (nil))
```

memória hivatkozás (Signed Int)

gcc fordító RTL belső ábrázolása

RTL = Register Transfer Language

Fordítási opció:gcc −c −fdump−rtl−all test1.c ⇒
test1.c.165r.expand

```
void fo() {
  int a=0;
  a=a+1;
}
```

```
(insn 5 2 6 2 (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                (const_int -4 [0xfffffffffffffffc]))  [0 a+0 S4 A32])
        (const_int 0 [0])) test1.c:2 -1
     (nil))
(insn 6 5 0 2 (parallel [
            (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                        (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
                (plus:SI (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                            (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
                    (const_int 1 [0x1])))
            (clobber (reg:CC 17 flags))
        ]) test1.c:3 -1
     (nil))
```

előjeles int összeadás

gcc fordító RTL belső ábrázolása

RTL = Register Transfer Language

Fordítási opció:gcc −c −fdump−rtl−all test1.c ⇒
test1.c.165r.expand

```
void fo() {
  int a=0;
  a=a+1;
}
```

pointert tartalmazó regiszter hivatkozás

```
(insn 5 2 6 2 (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
        (const_int 0 [0])) test1.c:2 -1
     (nil))
(insn 6 5 0 2 (parallel [
            (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                        (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
                (plus:SI (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                            (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
                    (const_int 1 [0x1])))
            (clobber (reg:CC 17 flags))
        ]) test1.c:3 -1
     (nil))
```

gcc fordító RTL belső ábrázolása

RTL = Register Transfer Language

Fordítási opció:gcc −c −fdump−rtl−all test1.c ⇒
test1.c.165r.expand

```
void fo() {
  int a=0;
  a=a+1;
}
```

```
(insn 5 2 6 2 (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                (const_int -4 [0xfffffffffffffffc ])) [0 a+0 S4 A32])
        (const_int 0 [0]))
    (nil))
(insn 6 5 0 2 (parallel [
            (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                        (const_int -4 [0xfffffffffffffffc ])) [0 a+0 S4 A32])
                (plus:SI (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                            (const_int -4 [0xfffffffffffffffc ])) [0 a+0 S4 A32])
                    (const_int 1 [0x1])))
            (clobber (reg:CC 17 flags))
        ]) test1.c:3 -1
    (nil))
```

az egyes részek párhuzamosan is kiértékelhetők

gcc fordító RTL belső ábrázolása

RTL = Register Transfer Language

Fordítási opció:gcc −c −fdump−rtl−all test1.c ⇒
test1.c.165r.expand

```
void fo() {
  int a=0;
  a=a+1;
}
```

```
(insn 5 2 6 2 (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
        (const_int 0 [0])) test1.c:2 -1
     (nil))
(insn 6 5 0 2 (parallel [
            (set (mem/c:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                        (const_int -4 [0xfffffffffffffffc])) [0 a+0 S4 A32])
                (plus:SI (mem/c:SI (plus:S
                            (const_int -4
                    (const_int 1 [0x1])))
            (clobber (reg:CC 17 flags))
        ]) test1.c:3 -1
     (nil))
```

A set utasítás állítja be a CC-t így legyőzi a plus utasítások lehetséges mellékhatásait.

Lineáris Intermediate Language típusai

- egy operandusú (stack szervezésű gépek pl. JVM);
- két operandusú $x \leftarrow x < op > y$;
- három operandusú $z \leftarrow x < op > y$;

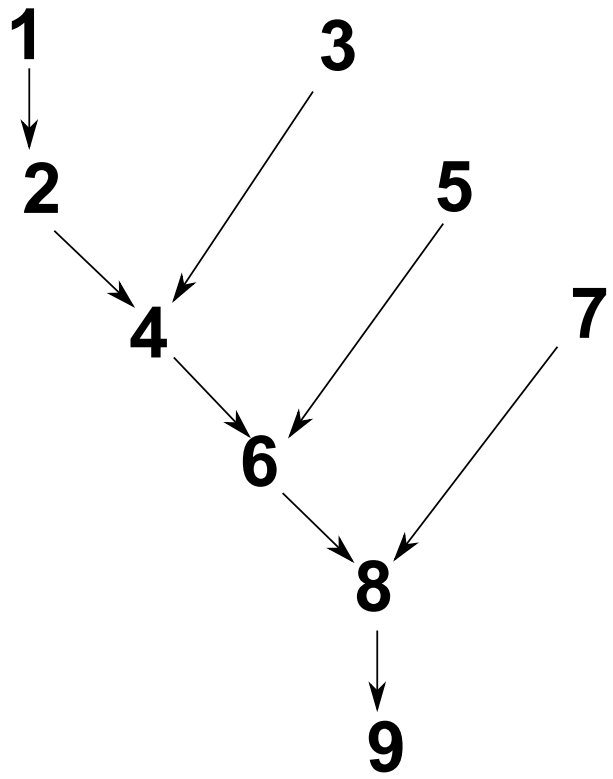
push 2	t1 = 2	t1 = 2
push y	t2 = y	t2 = y
mul	t1 = t1 $*$ t2	t3 = t1 $*$ t2
push x	t3 = x	t4 = x
sub	t3 = t3 $-$ t1	t5 = t4 $-$ t1
egy operandusú	két operandusú	három operandusú

$x - 2 * y$

Adatfüggőségi gráf

1.	loadAI	r_0,	0	$\rightarrow$	r_1
2.	add	r_1,	r_1	$\rightarrow$	r_1
3.	loadAI	r_0,	8	$\rightarrow$	r_2
4.	mult	r_1,	r_2	$\rightarrow$	r_1
5.	loadAI	r_0,	16	$\rightarrow$	r_2
6.	mult	r_1,	r_2	$\rightarrow$	r_1
7.	loadAI	r_0,	24	$\rightarrow$	r_2
8.	mult	r_1,	r_2	$\rightarrow$	r_1
9.	storeAI	r_1		$\rightarrow$	$r_0, 0$

ILOC kód



storeAI $r_i \Rightarrow r_j, 4 \quad \equiv \quad \text{Memory}(r_j + 4) \leftarrow \text{Contents}(r_i)$

Quadruples (négyes)

Művelet	Op1	Op2	Eredmény
loadI	2		t_1
load	y		t_2
mult	t_1	t_2	t_3
load	x		t_4
sub	t_4	t_3	t_5

Triples (hármas)

A triples, indirekt triples ábrázolás csak ciklus és elágazásmentes, azaz alapblokkban alkalmazható. (Alapblokk definíciója később lesz.)

Zárójelben írt érték az adott utasítás eredményére hivatkozik.

(1)	loadI	*2*	
(2)	load	*y*	
(3)	mult	(1)	(2)
(4)	load	*x*	
(5)	sub	(4)	(3)

Hátránya, hogy a sorok átrendezése nehézkes.

Indirekt triples

Zárójelbe írt érték az adott utasításhoz rendelt [*j*] indexen keresztül hivatkozik arra az utasításra, melynek az eredményét használja. Ha az utasítás sorrendje változik, akkor az index értéke is módosul.

Index	sorszám			
[1]	(1)	loadI	2	
[2]	(2)	load	*y*	
[3]	(3)	mult	(1)	(2)
[4]	(4)	load	*x*	
[5]	(5)	sub	(4)	(3)

SSA static single assignment

```
x ← ...
y ← ...
while( x < k )
{
    x ← x + 1;
    y ← y + x;
}
```

```
x0 ← ...
y0 ← ...
if (x0 ≥ k) goto next
loop: x1 ← φ(x0, x2);
      y1 ← φ(y0, y2);
      x2 ← x1 + 1;
      y2 ← y1 + x2;
      if (x2 < k) goto loop
next: ....
```

SSA static single assignment

- Fordítókkal foglalkozó tudomány a változó definiciós pontjának (def) azt a helyet nevezi, ahol a változó értéket kap. Egy változóhoz több helyen is rendelhetünk új értéket.
- A kifejezések jobb oldalán meghivatkozott változó a változó használatát (use) jelenti.
- SSA esetén a többször definiált változókat a definiálás helyén egy egyedi indexszel különböztetjük meg.
- A változók használatakor megmondjuk, hogy a használt (use) változó melyik utasításban kapott értéket, azaz hol definiáltuk.
- Ha a használt változó definiciós pontja nem egyértelmű, mert a program különböző ágán eljutva, több ágon is definiáltuk a változót, akkor ennek leírására egy ϕ függvényt használunk.
  1. A ϕ függvény megadja, hogy a változó értéke mely pontokban kaphatott értéket.
  2. A ϕ függvény a fordítónak szól, belőle kód nem keletkezik.
  3. A ϕ függvény paramétere minimum kettő, maximum akármennyi, de csak indexben eltérő változó lehet.

Példa 3 operandusú IL-re

```
int a;                    |  t0 = b + c;
int b;                    |  a = t0 + d;
int c;                    |  t1 = a * a;
int d;                    |  t2 = b * b;
a = b + c + d;            |  b = t1 + t2;
b = a * a + b * b         |
```

temporális változók: t0, t1, t2

gcc fordító Gimple belső ábrázolása

Fordítási opció:gcc −c −fdump−tree−all test2.c

```
int a;

int main()
{
  int x = 10;

  int y = 5;

  x = a + x * y;

  y = y - a * x;

}
```

test2.c

```
main ()
{
  int D.1373;
  int a.0;
  int a.1;
  int D.1376;
  int x;
  int y;

  x = 10;
  y = 5;
  D.1373 = x * y;
  a.0 = a;
  x = D.1373 + a.0;
  a.1 = a;
  D.1376 = a.1 * x;
  y = y - D.1376;
}
```

test2.c.0004t.gimple

gcc fordító Gimple belső ábrázolása

Fordítási opció:gcc −c −fdump−tree−all test3.c

```
int main()
{
  int a=2, b=3, c=4;
  while (a<=7)
  {
    a = a+1;
  }

  if (a<=12)
     a = a+b+c;
}
```

test3.c

```
main ()
{
  int D.1378;
  int a;
  int b;
  int c;

  a = 2;
  b = 3;
  c = 4;
  goto <D.1373>;
  <D.1372>:
  a = a + 1;
  <D.1373>:
  if (a <= 7) goto <D.1372>; else goto <D.1374>;
  <D.1374>:
  if (a <= 12) goto <D.1376>; else goto <D.1377>;
  <D.1376>:
  D.1378 = a + b;
  a = D.1378 + c;
  <D.1377>:
}
```

test3.c.0004t.gimple

3 operandusú Intermediate Language

- egy utasításnak maximum 3 operandusa lehet;
- összetett kifejezésnél temporális változókat kell bevezetni;
- van két operandusú utasítás, pl. t1 = 5;

Intermediate Language utasításkészlete

Értékadás:

- var = constant;
- var = string;
- var1 = var2;
- var = label;

Intermediate Language utasításkészlete

Aritmetikai operátorok:

- var1 = var2 op var3;
- var1 = constant op var2;
- var1 = var2 op constant;
- var = constant1 op constant2;
- operátorok +.−.∗./,%,|,& stb;

Intermediate Language utasításkészlete

Logikai értékek

- 0 – hamis;
- nem 0 – igaz.

Logikai műveletek:

- var3 = var2 == var1;
- var3 = var2 < var1;
- var3 = var2 || var1;
- var3 = var2 && var1;
- a többi logikai kifejezést ezek segítségével kell előállítani.

Intermediate Language

Vezérlésátadó utasítások:

- névvel rendelkező cimke L1:
- Goto label;
- IfZ value Goto label;
- IfZ csak a Goto utasítással együtt fordulhat elő;

Intermediate Language

Függvényhívó utasítás:

- LCall L1 – esetén a hívott függvény címe fordítási időben ismert.
- ACall t1 – t1 függvény címe futáskor áll elő (pl. virtuális függvénytábla).

- LCall L1;
- t1 = LCall L1;
- ACall t1;
- t0 = ACall t1;

Intermediate Language

Függvény definiálás:

- BeginFunc n; – Függvény törzsének a kezdete, ahol n a lokális adatok számára szükséges hely bájtokan.
- EndFunc – függvény vége;
- Return t1 – visszatérés visszaadott értékkel;
- Return – visszatérés a függvényből.

Intermediate Language

Memória címzés:

- t1 = ∗t2
- t1 = ∗(t2+offset)
- ∗t1 = ∗t2
- ∗(t1 + offset) = ∗t2
- offset – negatív vagy pozitív egész érték.

Tömb:

- t1[t2] = t3
- t3 = t1[t2]
- az index operátor a C-ben megismert módon működik.

Példa 3 operandusú IL-re

```
int x;
int y;



while(x<=y) {
        x = x + 2;
}


y = x;
```

```
L0:
    t0 = x < y;
    t1 = x == y;
    t2 = t0 || t1;
    IfZ t2 Goto L1;
    x = x + 2;
    Goto L0
L1:
    y = x;
```

Section 2

Optimalizálás

Miért van szükség optimalizálásra

- AST-ből IR-re való egyszerű áttérés redundanciát okoz;
- egyes részszámításokat
  - fel lehet gyorsítani;
  - közös részeket össze lehet vonni;
  - felesleges részeket ki lehet hagyni
- mert a programozók lusták
  - for ciklusba, vagy a feltétel részbe írnak a ciklus végrehajtása során nem változó kifejezéseket;

A kód optimalizálása kihívást jelent

- Cél:
  - az eredményeket helyességét ne befolyásolja ;
  - a lehető leghatékonyabb IR-t állítsa elő;
  - ne tartson sok ideig.
- Valóság:
  - néha hibát okoz a kód futásánál;
  - sokáig tart a kódgenerálás.
- Legtöbb optimalizálási algoritmus NP-teljes, ha egyáltalán létezik megoldás.

A program szemamtikáját nem befolyásoló optimalizálás

- felesleges temporális változók megszüntetése;
- fordítási időben ismert konstans kifejezések kiszámítása;
- ciklusban lévő invariáns részek kiemelése;
- . . .
- ellenpéldaként meg lehet említeni (szemantikát befolyásoló optimalizálás) a buborékos rendezés kicserélése gyorsrendezésre.

Quicksort algoritmus

```
void quicksort(int m, int n )
{
    int i, j;
    int v, x;
    if( n <= m ) return;
    /* IR kód kezdete */
    i=m-1; j=n; v = a[n];
    for(;;) {
        do i = i+1; while( a[i] < v );
        do j = j-1; while( a[j] > v );
        if( i >= j ) break;
        x = a[i]; a[i]= a[j]; a[j]=x;
    }
    x=a[i]; a[i]=a[n]; a[n]=x;
    /* IR kód vége */
    quicksort(m,j); quicksort(i+1,n);
}
```

IR megjelenítése

```
    i = m-1
    j = n
    t1 = 4*n
    v = a[t1]
L0:
    i = i+1
    t2 = 4*i
    t3 = a[t2]
    if t3<v goto L0
L1:
    j = j-1
    t4 = 4*j
    t5 = a[t4]
    If t5>v goto L1
    if i>=j goto L2
    t6 = 4*i
    x = a[t6]
    t7 = 4*i
    t8 = 4*j
    t9 = a[t8]
    a[t7] = t9
    t10 = 4*j
    a[t10] = x
    goto L0
```

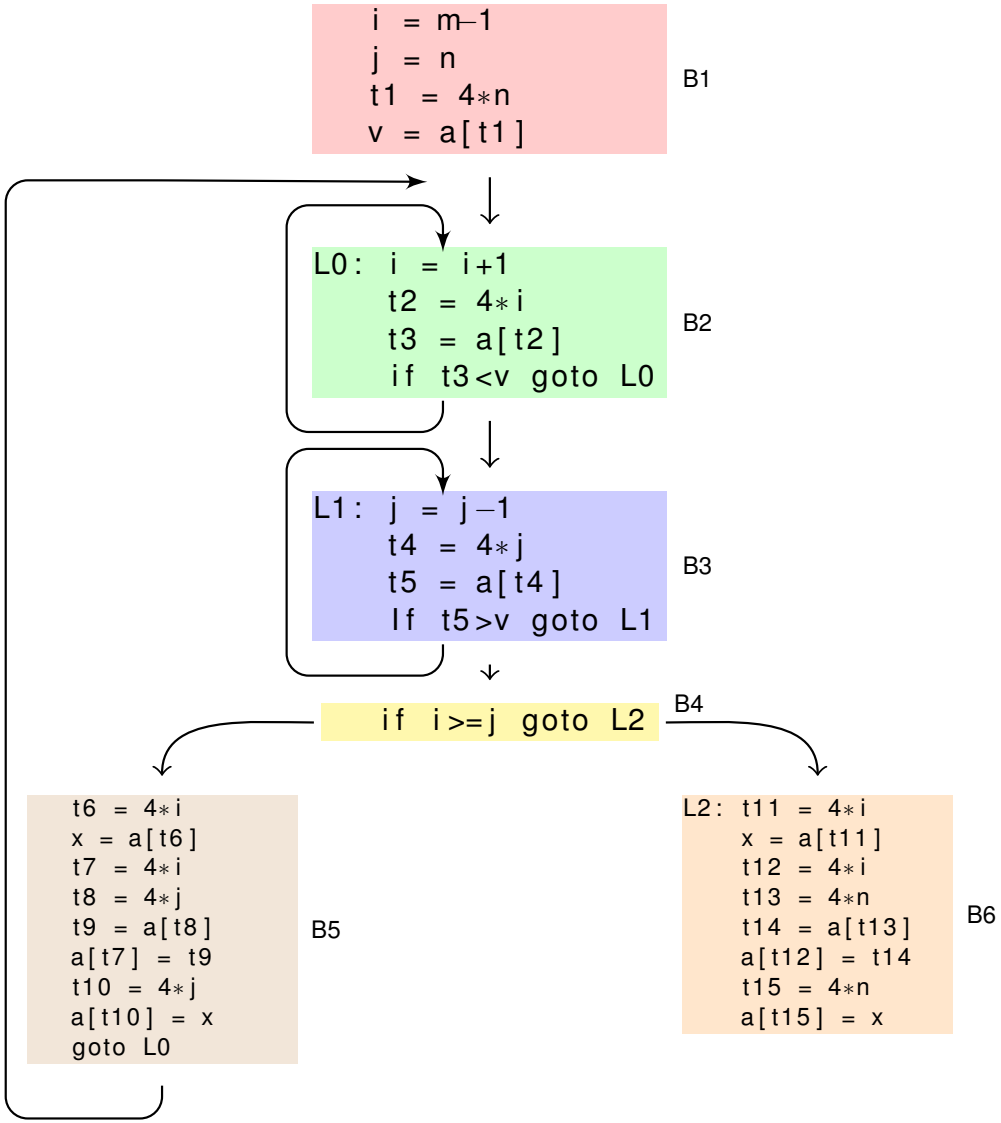
```
L2:
    t11 = 4*i
    x = a[t11]
    t12 = 4*i
    t13 = 4*n
    t14 = a[t13]
    a[t12] = t14
    t15 = 4*n
    a[t15] = x
```

IR megjelenítése – alapblokkokra bontás

```
    i = m-1
    j = n
    t1 = 4*n
    v = a[t1]
L0:
    i = i+1
    t2 = 4*i
    t3 = a[t2]
    if t3<v goto L0
L1:
    j = j-1
    t4 = 4*j
    t5 = a[t4]
    If t5>v goto L1
    if i>=j goto L2
    t6 = 4*i
    x = a[t6]
    t7 = 4*i
    t8 = 4*j
    t9 = a[t8]
    a[t7] = t9
    t10 = 4*j
    a[t10] = x
    goto L0
```

```
L2:
    t11 = 4*i
    x = a[t11]
    t12 = 4*i
    t13 = 4*n
    t14 = a[t13]
    a[t12] = t14
    t15 = 4*n
    a[t15] = x
```

IR megjelenítése – alapblokkokra bontás



Alapblokk (Basic Block)

- az IR utasításainak a lineáris sorozata;
- egy belépési ponttal rendelkezik;
- egy kilépési pontja van, és ez a sorozat végén található;
- olyan utasítások sorozata, amelyek a blokk végrehajtása során biztosan végrehajtódnak.
- Alapblokkok kezdete és vége :
  - a program első utasítása az első blokk kezdete;
  - minden olyan hely, amelyre távolról át lehet adni a vezérlést (Goto, LCall, ACall) egy új blokk kezdetét jelenti;
  - ugró utasítás csak a blokk végén lehet;
  - minden blokk vége után egy új blokk kezdődik (kivéve az utolsót).