Mezikódy

Mezikódy

- Vysokoúrovňový mezikód
  - Reprezentace vstupního programu
    - Během fází, řešících konstrukce a pravidla vstupního jazyka
    - Užívá logické typy a operace vstupního jazyka
  - Nejčastěji ve formě anotovaného AST (abstract syntax tree)
    - Derivační strom podle abstraktní gramatiky
- Mezikód střední úrovně
  - Nejčastější hranice mezi front- a back-endem
  - Na vstupním jazyce nezávislá reprezentace
    - Užívá fyzické typy a operace na nich
  - Nejčastěji ve formě čtveřic
    - Tříadresové pseudoinstrukce, pomocné proměnné
    - Někdy ve speciálních formách (SSA – static single assignment)
    - Control-flow může být ve formě grafu BB (základních bloků)
- Nízkoúrovňový mezikód
  - Ekvivalent strojových instrukcí
    - Nekompaktní forma, symbolické a relokované operandy
    - Před alokací registrů forma s neomezeným počtem virtuálních registrů
  - Někdy v univerzální strojově nezávislé formě (GCC RTL)

Mezikódy

- SSA – Static Single Assignment
  - Do každé proměnné se přiřazuje pouze v jediném místě kódu
    - Jediný přiřazovací příkaz resp. instrukce mezikódu
    - Může být prováděn mnohokrát, je-li v cyklu
  - Aplikovatelné pouze na jednoduché lokální proměnné bez aliasu
  - Překladač upravuje mezikód do SSA formy až po provedení analýz/úprav:
    - Integrace procedur může odstranit aliasing (parametry předané odkazem)
    - Analýza aliasů vyloučí problematické proměnné
    - Dekompozice nealiasovaných lokálních proměnných typu struktura
- SSA zjednodušuje řadu algoritmů používaných v překladačích
- Konverze do SSA formy vyžaduje speciální operátor Φ
  - Alternativní přiřazení do téže proměnné

```
if C then X:=A else X:=B
```

  - se reprezentuje pomocí pomocných proměnných

```
if C then X1:=A else X2:=B;
X:=Φ(X1,X2)
```

  - Φ(X1,X2) znamená hodnota X1 nebo X2, podle toho, která byla (naposledy) definována
  - Na operátor Φ se pohlíží podobně jako na jiné (asociativní a komutativní) operátory
  - V závěrečné fázi překladu je Φ eliminován přejmenováním proměnných (tj. umístěním X1 a X2 do téhož místa, pokud to lze), může vést k duplikaci

Mezikódy

- **Informace uložené v mezikódu střední úrovně**
  - Seznam globálních proměnných
  - Další globální informace pro generovaný kód
  - Seznam procedur
  - Další informace pro debugger

Mezikódy

Informace uložené v mezikódu střední úrovně

- Seznam globálních proměnných
  - Velikost
  - Inicializace
  - Příznak konstantnosti
  - Jméno (pro linker)
  - Logický typ (pro debugger)
- Další globální informace pro generovaný kód
  - Konstanty (reálné, řetězcové, strukturované)
  - Tabulky virtuálních funkcí, RTTI
  - Často splývají s globálními proměnnými
- Seznam procedur
- Další informace pro debugger
  - Jména a konstrukce typů

Mezikódy

- **Popis procedury**
  - Jméno (pro linker a chybová hlášení)
- Seznam parametrů
  - Fyzický typ (+ velikost)
  - Umístění podle volací konvence
  - Jméno (pro debugger a chybová hlášení)
  - Logický typ (pro debugger a určení aliasů)
- Seznam lokálních proměnných
  - Fyzický typ (+ velikost)
  - Jméno (pro debugger a chybová hlášení)
  - Logický typ (pro debugger a určení aliasů)
  - Proměnné ve vnořených blocích se obvykle povyšují na úroveň procedury
- Kód procedury
- Další informace (popisy výjimek apod.)

Kód procedury

- Plně sekvenční forma
  - Posloupnost (pseudo-)instrukcí virtuálního stroje
  - Tok řízení popsán skokovými instrukcemi
- Částečně sekvenční forma
  - Tok řízení popsán grafem, jehož uzly jsou základní bloky
  - Každý základní blok obsahuje posloupnost (pseudo-)instrukcí
    - Skokové instrukce pouze na konci BB nebo zaznamenány jinak
- Nesekvenční forma
  - Tok řízení popsán grafem, jehož uzly jsou základní bloky
  - Tok dat uvnitř základního bloku popsán dagem
    - Různé formy podle stupně analýzy aliasů a rozsahů platnosti
    - Pokročilejší formy nahrazují lokální proměnné rozhraními bloků

Plně sekvenční čtveřicový mezikód

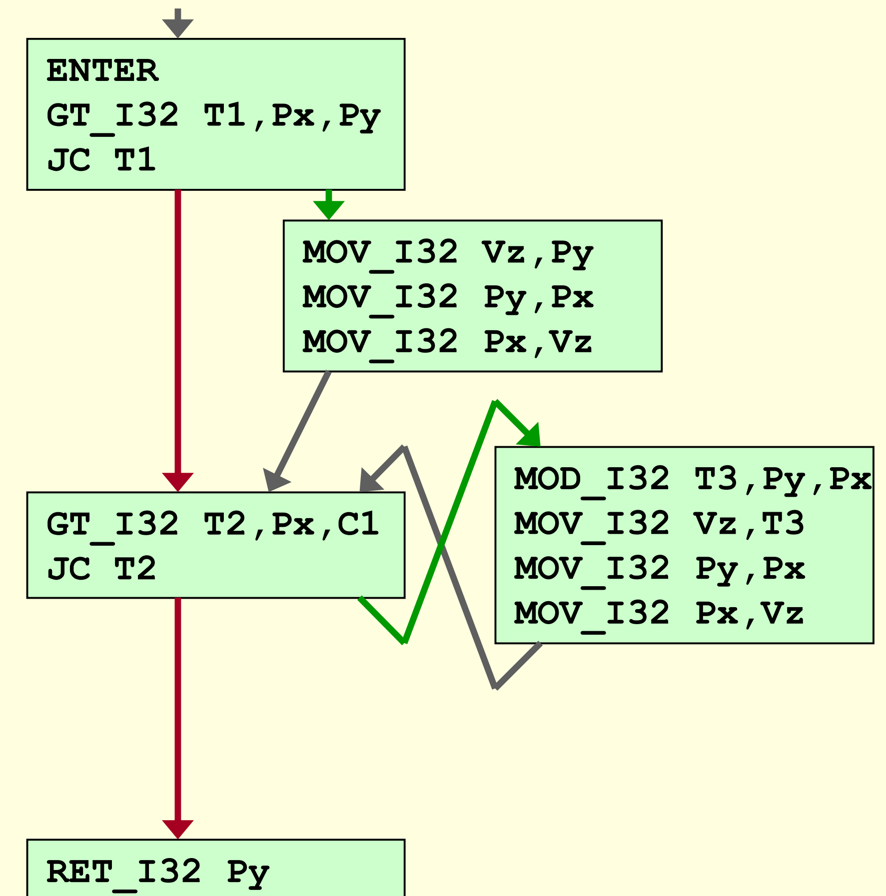

```
int gcd( int x, int y)
{ int z;
  if ( x > y )
  {
    z = y;
    y = x;
    x = z;
  }
  while ( x > 0 )
  {
    z = y % x;
    y = x;
    x = z;
  }
  return y;
}
```

```
CONST:(C1,I32,0)

PROC ”gcd”
PARAM:(Px,I32,”x”),(Py,I32,”y”)
VAR:(Vz,I32,”z”)
TMP:(T1,B),(T2,B),(T3,I32)

ENTER
GT_I32 T1,Px,Py
JF T1,L1
MOV_I32 Vz,Py
MOV_I32 Py,Px
MOV_I32 Px,Vz
L1:
GT_I32 T2,Px,C1
JF T2,L2
MOD_I32 T3,Py,Px
MOV_I32 Vz,T3
MOV_I32 Py,Px
MOV_I32 Px,Vz
JMP L1
L2:
RET_I32 Py
```

Částečně sekvenční čtveřicový mezikód


```
int gcd( int x, int y)
{ int z;
  if ( x > y )
  {
    z = y;
    y = x;
    x = z;
  }
  while ( x > 0 )
  {
    z = y % x;
    y = x;
    x = z;
  }
  return y;
}
```

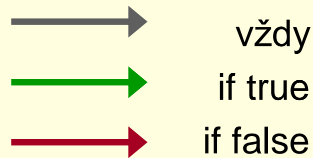
```
CONST:(C1,I32,0)

PROC "gcd"
PARAM:(Px,I32,"x"),(Py,I32,"y")
VAR:(Vz,I32,"z")
TMP:(T1,B),(T2,B),(T3,I32)
```



Nesekvenční mezikód (před analýzou aliasů)

```
int gcd( int x, int y)
{ int z;
  if ( x > y )
  {
    z = y;
    y = x;
    x = z;
  }
  while ( x > 0 )
  {
    z = y % x;
    y = x;
    x = z;
  }
  return y;
}
```

Control-Flow

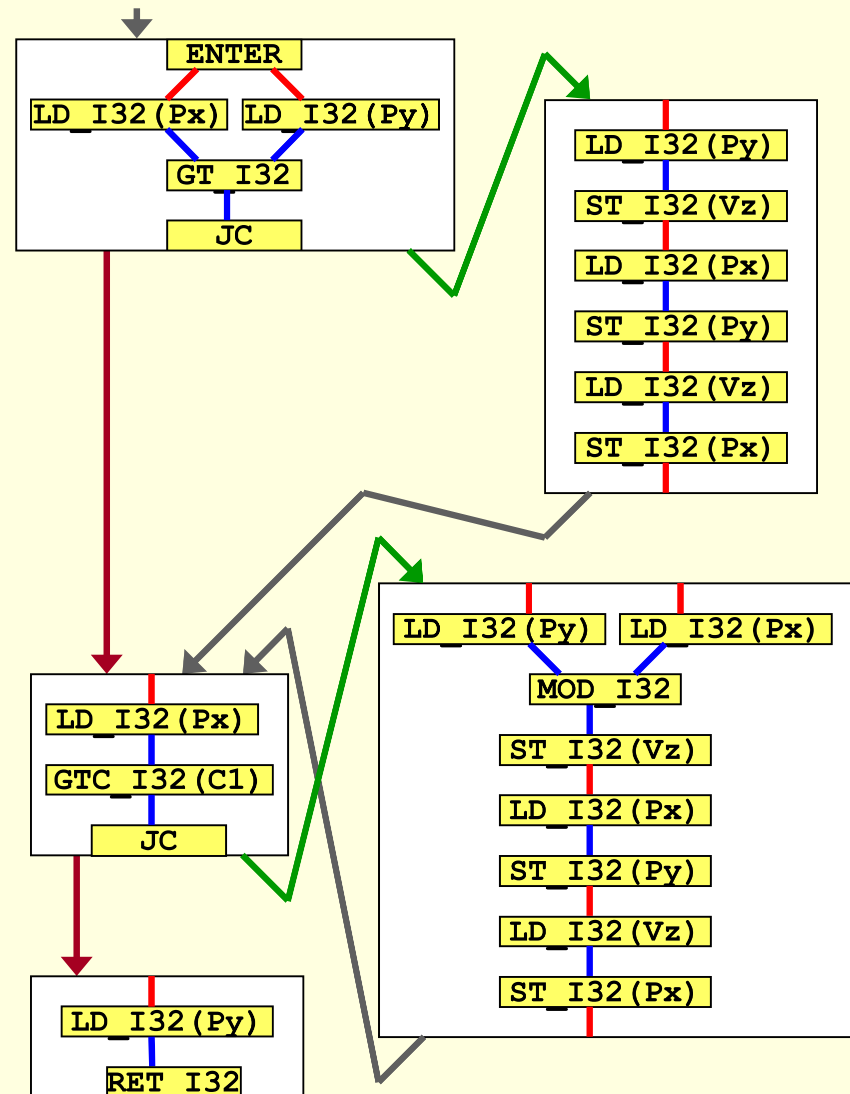


Dag

Data-flow

Závislosti

```
CONST:(C1,I32,0)

PROC "gcd"
PARAM:(Px,I32,"x"),(Py,I32,"y")
VAR:(Vz,I32,"z")
```



Mezikódy střední úrovně

- Plně sekvenční forma
- Částečně sekvenční forma
- Nesekvenční forma

- Všechny tyto formy lze generovat přímo z abstraktního syntaktického stromu
  - Jedním průchodem zdola nahoru
    - goto je nutné ošetřit dodatečnými zásahy (backpatching)
  - Strom nemusí fyzicky existovat, postačí průchod myšleným stromem
    - LR analýza: pravá derivace pozpátku
    - LL analýza rekurzivním sestupem
  - Většina front-endů přesto strom konstruuje
    - Složité konstrukce jazyka (šablony, předkompilované části)
    - Rozhraní mezi syntaktickým a sémantickým analyzátorem
    - Optimalizace

Mezikódy střední úrovně

- Plně sekvenční forma
- Částečně sekvenční forma
- Nesekvenční forma

- Táž forma se obvykle v průběhu překladu upravuje
  - Připojují se odvozené informace a optimalizační rozhodnutí
  - Provádějí se ekvivalentní úpravy (optimalizace)

- Jedna forma může mít různé variace
  - A to i uvnitř jednoho překladače
  - Odráží různé způsoby a/nebo různé stupně analýzy

- Řada překladačů užívá dvě z těchto forem
  - Z historických důvodů (stabilita rozhraní front-end/back-end)
  - Pro vytvoření druhé formy je nutná analýza první formy

Detekce základních bloků

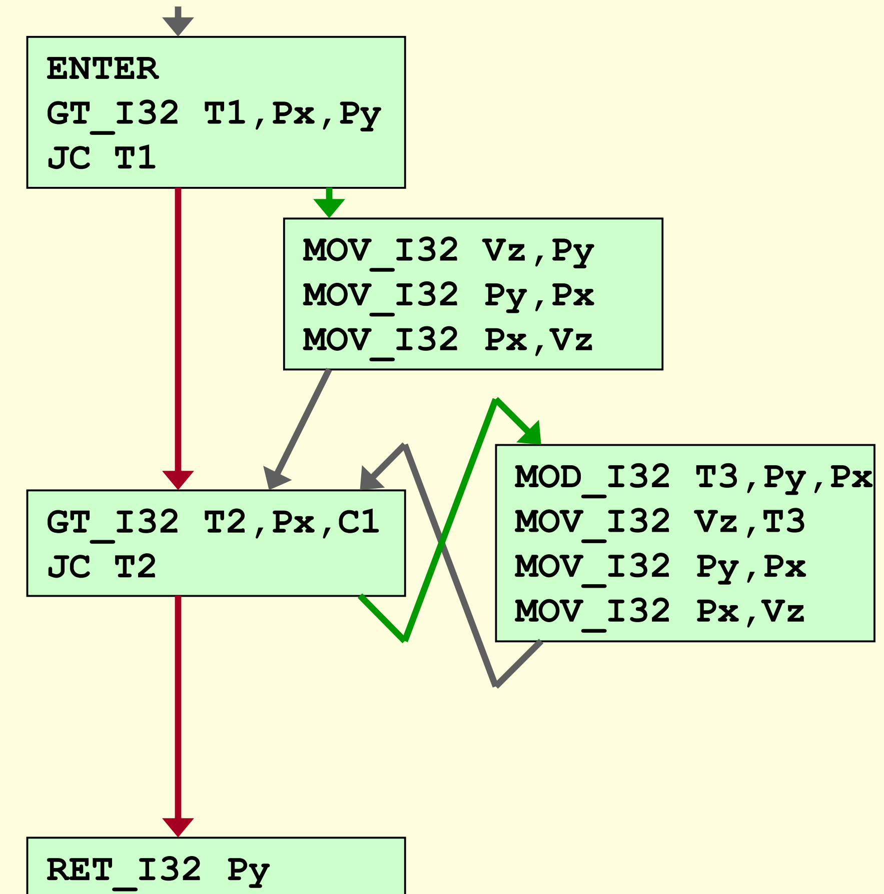

```
CONST:(C1,I32,0)

PROC ″gcd″
PARAM:(Px,I32,″x″),(Py,I32,″y″)
VAR:(Vz,I32,″z″)
TMP:(T1,B),(T2,B),(T3,I32)

ENTER
GT_I32 T1,Px,Py
JF T1,L1
MOV_I32 Vz,Py
MOV_I32 Py,Px
MOV_I32 Px,Vz
L1:
GT_I32 T2,Px,C1
JF T2,L2
MOD_I32 T3,Py,Px
MOV_I32 Vz,T3
MOV_I32 Py,Px
MOV_I32 Px,Vz
JMP L1
L2:
RET_I32 Py
```

- **V sekvenčním mezikódu**
  - Základní blok
    - Začíná
      - Na začátku procedury
      - V cíli skoku
      - Za podmíněným skokem
    - Končí
      - Podmíněným skokem
      - Nepodmíněným skokem
      - Návratem z procedury
      - Před cílem skoku

Detekce základních bloků


```
CONST:(C1,I32,0)

PROC "gcd"
PARAM:(Px,I32,"x"),(Py,I32,"y")
VAR:(Vz,I32,"z")
TMP:(T1,B),(T2,B),(T3,I32)

ENTER
GT_I32 T1,Px,Py
JF T1,L1
MOV_I32 Vz,Py
MOV_I32 Py,Px
MOV_I32 Px,Vz
L1:
GT_I32 T2,Px,C1
JF T2,L2
MOD_I32 T3,Py,Px
MOV_I32 Vz,T3
MOV_I32 Py,Px
MOV_I32 Px,Vz
JMP L1
L2:
RET_I32 Py
```

```
CONST:(C1,I32,0)

PROC "gcd"
PARAM:(Px,I32,"x"),(Py,I32,"y")
VAR:(Vz,I32,"z")
TMP:(T1,B),(T2,B),(T3,I32)
```



Detekce základních bloků

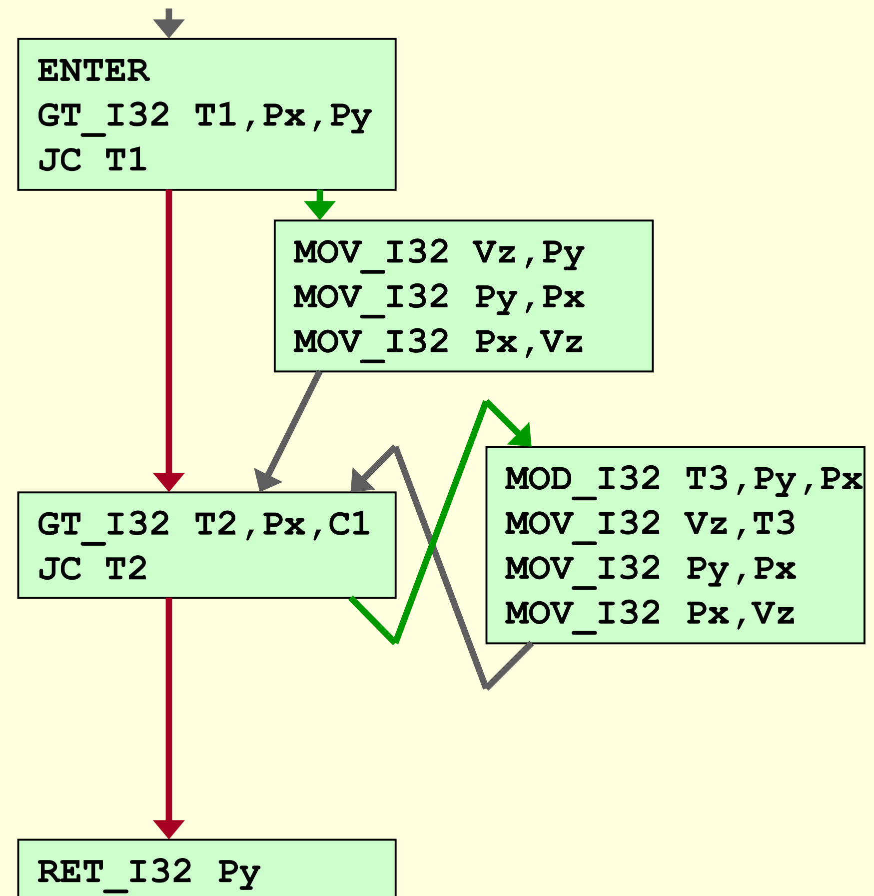

```
int gcd( int x, int y)
{ int z;
  if ( x > y )
  {
    z = y;
    y = x;
    x = z;
  }
  while ( x > 0 )
  {
    z = y % x;
    y = x;
    x = z;
  }
  return y;
}
```

- **Ve zdrojovém kódu**
  - Základní blok
    - Začíná
      - Na začátku procedury
      - Na začátku then a else bloku
      - Na začátku těla cyklu
      - Za if příkazem
      - Za while cyklem
    - Končí
      - Na konci procedury
      - Na konci then a else bloku
      - Na konci těla cyklu
      - Na konci podmínky v if
      - Na konci podmínky ve while
      - Příkazem return/break apod.
  - Komplikace
    - Zkrácené vyhodnocování booleovských výrazů
    - Podmíněný výraz
    - Příkaz goto

Detekce základních bloků

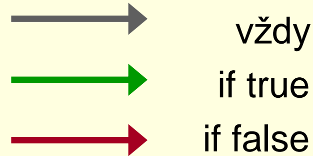

```
int gcd( int x, int y)
{ int z;
  if ( x > y )
  {
    z = y;
    y = x;
    x = z;
  }
  while ( x > 0 )
  {
    z = y % x;
    y = x;
    x = z;
  }
  return y;
}
```

```
CONST:(C1,I32,0)

PROC "gcd"
PARAM:(Px,I32,"x"),(Py,I32,"y")
VAR:(Vz,I32,"z")
TMP:(T1,B),(T2,B),(T3,I32)
```

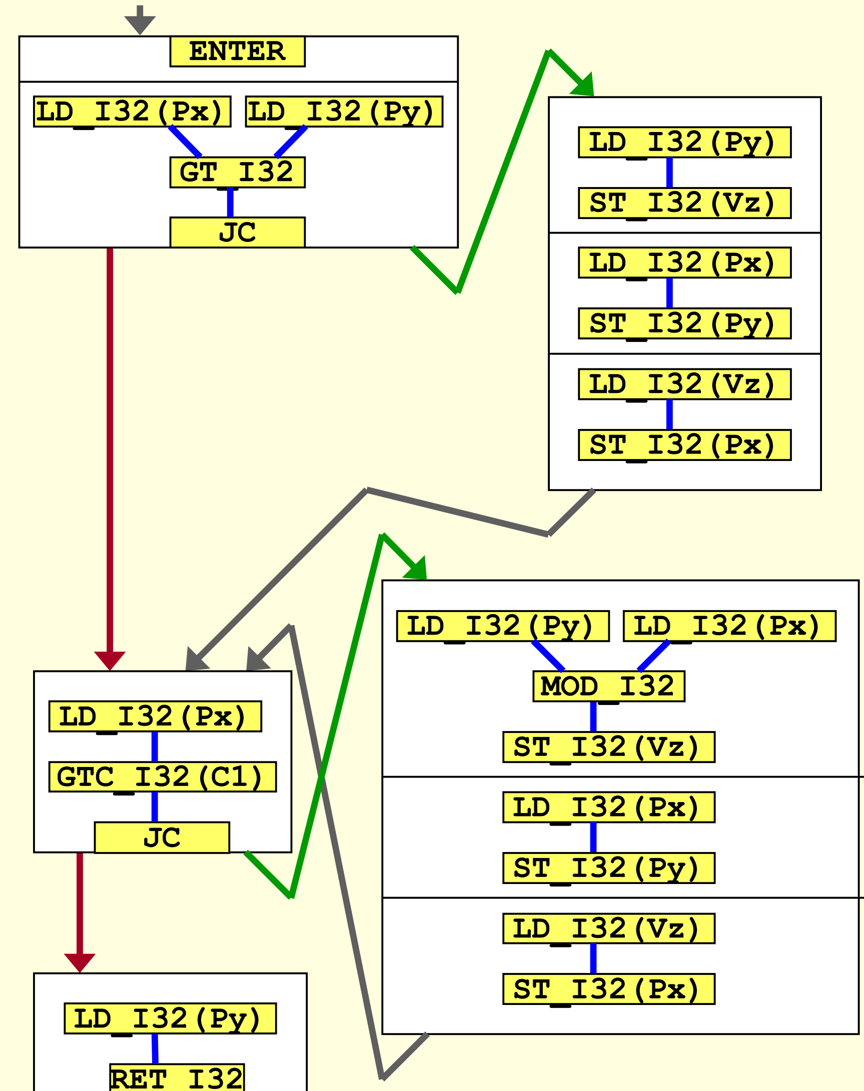


Nesekvenční mezikód s hranicemi příkazů


```
int gcd( int x, int y)
{ int z;
  if ( x > y )
  {
    z = y;
    y = x;
    x = z;
  }
  while ( x > 0 )
  {
    z = y % x;
    y = x;
    x = z;
  }
  return y;
}
```



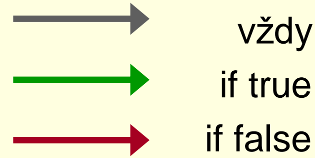
Dag

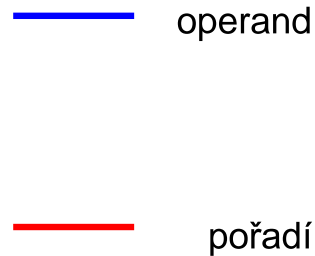


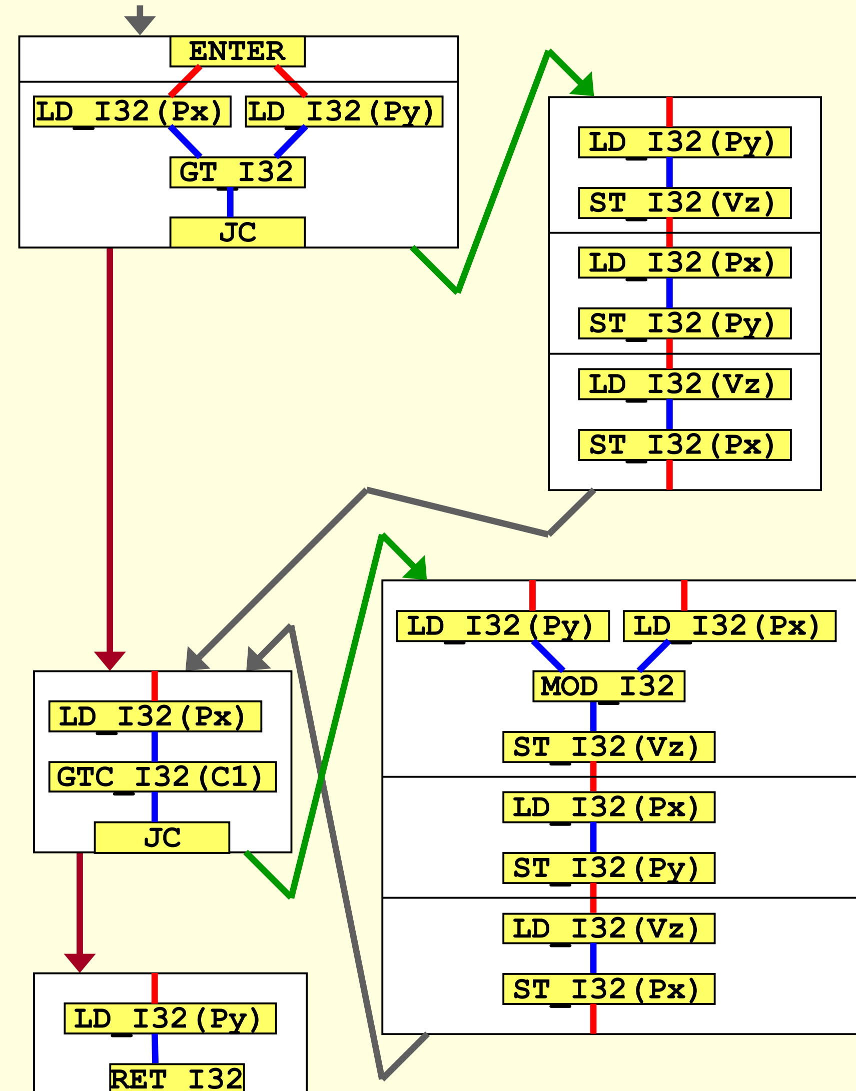

```
CONST:(C1,I32,0)

PROC "gcd"
PARAM:(Px,I32,"x"),(Py,I32,"y")
VAR:(Vz,I32,"z")
```

Nesekvenční mezikód před analýzou aliasů


```
int gcd( int x, int y)
{ int z;
  if ( x > y )
  {
    z = y;
    y = x;
    x = z;
  }
  while ( x > 0 )
  {
    z = y % x;
    y = x;
    x = z;
  }
  return y;
}
```

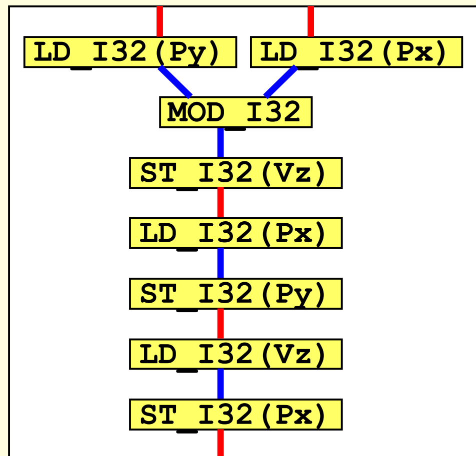


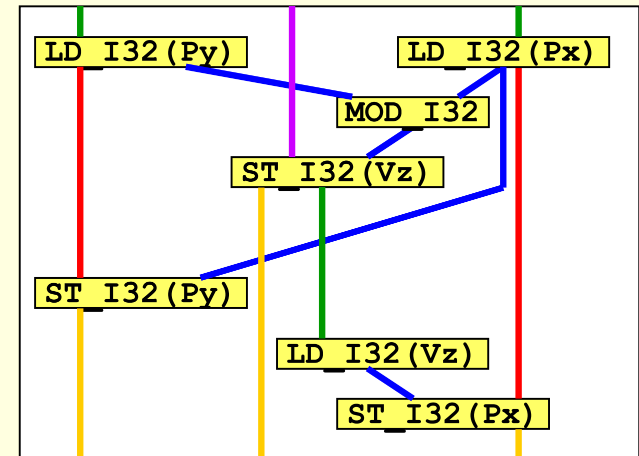




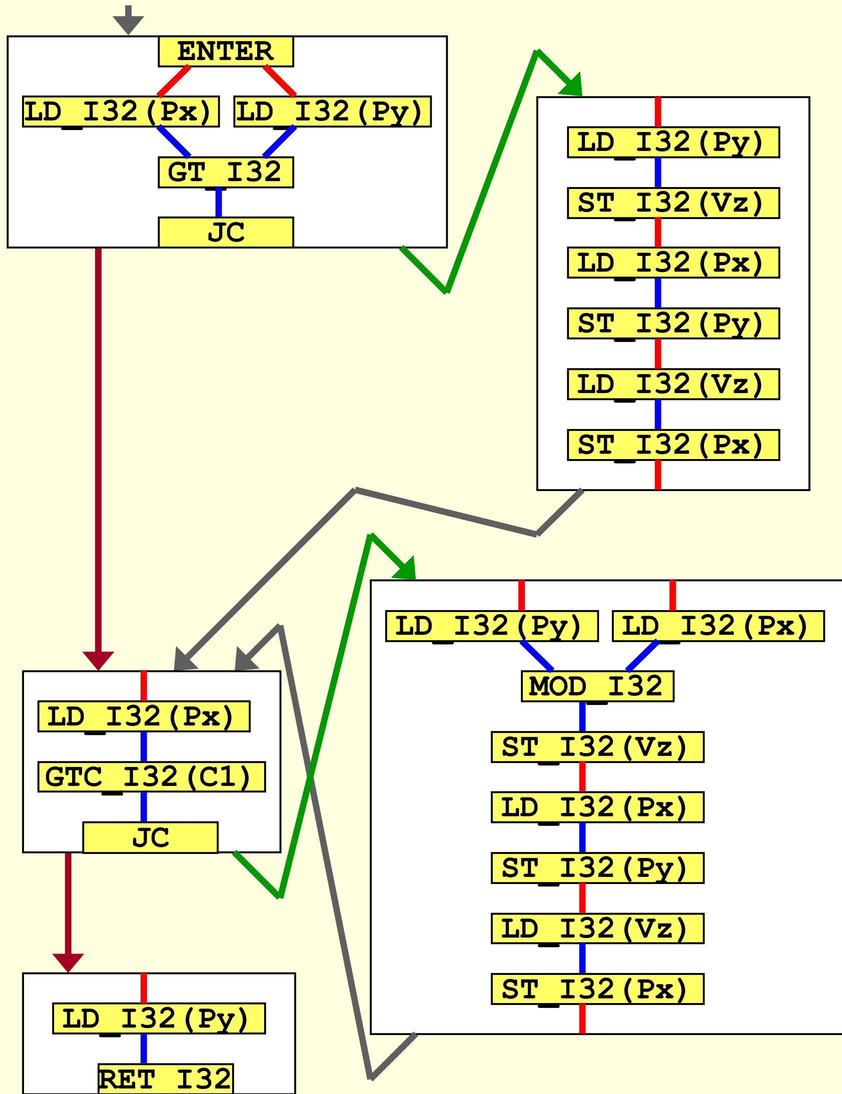

```
CONST:(C1,I32,0)

PROC "gcd"
PARAM:(Px,I32,"x"),(Py,I32,"y")
VAR:(Vz,I32,"z")
```

Odstranění závislostí po analýze aliasů





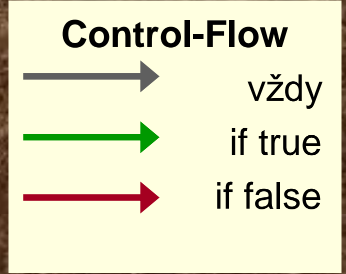
Odstranění závislostí po analýze aliasů



ENTER
LD_I32(Px)
LD_I32(Py)
GT_I32
JC
LD_I32(Py)
LD_I32(Px)
ST_I32(Py)
ST_I32(Vz)
LD_I32(Vz)
ST_I32(Px)
LD_I32(Py)
LD_I32(Px)
ST_I32(Py)
MOD_I32
ST_I32(Vz)
LD_I32(Vz)
ST_I32(Px)
LD_I32(Px)
GTC_I32(C1)
JC
LD_I32(Py)
RET_I32

Nesekvenční mezikód po analýze aliasů

```
int gcd( int x, int y)
{ int z;
  if ( x > y )
  {
    z = y;
    y = x;
    x = z;
  }
  while ( x > 0 )
  {
    z = y % x;
    y = x;
    x = z;
  }
  return y;
}
```



ENTER
LD_I32(Px) LD_I32(Py)
GT_I32
JC

LD_I32(Py) LD_I32(Px)
ST_I32(Py) ST_I32(Vz)
LD_I32(Vz)
ST_I32(Px)

LD_I32(Px)
GTC_I32(C1)
JC

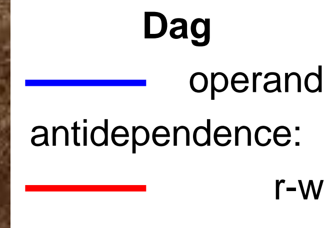
LD_I32(Py) LD_I32(Px)
ST_I32(Py) MOD_I32
ST_I32(Vz)
LD_I32(Vz)
ST_I32(Px)

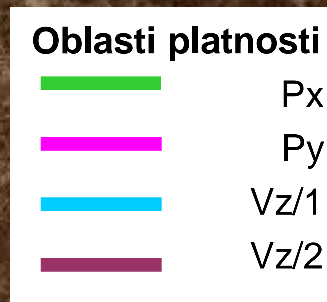
LD_I32(Py)
RET_I32

```
CONST:(C1,I32,0)

PROC "gcd"
PARAM:(Px,I32,"x"),(Py,I32,"y")
VAR:(Vz,I32,"z")
```

Nesekvenční mezikód s rozsahy platnosti


```
int gcd( int x, int y)
{ int z;
  if ( x > y )
  {
    z = y;
    y = x;
    x = z;
  }
  while ( x > 0 )
  {
    z = y % x;
    y = x;
    x = z;
  }
  return y;
}
```






```
CONST:(C1,I32,0)

PROC ″gcd″
PARAM:(Px,I32,″x″),(Py,I32,″y″)
VAR:(Vz,I32,″z″)
```

```
ENTER
LD_I32(Px)   LD_I32(Py)
GT_I32
JC

LD_I32(Py)   LD_I32(Px)
ST_I32(Py)   ST_I32(Vz)
LD_I32(Vz)
ST_I32(Px)

LD_I32(Px)
GTC_I32(C1)
JC

LD_I32(Py)   LD_I32(Px)
ST_I32(Py)   MOD_I32
ST_I32(Vz)
LD_I32(Vz)
ST_I32(Px)

LD_I32(Py)
RET_I32
```

Určení rozsahů platnosti (live-range analysis)

- Rozsah platnosti (životnost) jednoduché lokální proměnné bez aliasu
  - Proměnná je v bodě a živá, jestliže existuje cesta control-flow grafem do bodu b taková, že:
    - Na cestě a-b není žádný zápis do této proměnné
    - V bodě b je tato proměnná čtena
- Jiné druhy proměnných se obvykle nezkoumají
  - Pro složené proměnné (pole/struktury) neplatí předpoklad, že zápisem do proměnné se předchozí obsah stává irelevantní
    - Složená proměnná je tedy živá všude, kde existuje cesta ke čtení
  - U proměnných s aliasem nelze obvykle jednoznačně určit čtení a zápisy
    - Definici lze upravit i pro nejistá čtení a zápisy
  - Účelem zkoumání životnosti je především alokace registrů
    - Složené a aliasované proměnné do registrů umístit nelze

Algoritmus: Určení rozsahů platnosti

- VAR – množina proměnných
- Orientovaný graf control-flow v proceduře: (BB,CF)
  - BB – množina základních bloků
  - CF $\subseteq$ BB $\times$ BB – přechody mezi základními bloky
- Lokální analýza
  - Vnitřní chování každého základního bloku
  - W : BB $\times$ VAR $\rightarrow$ Bool – blok zapisuje do proměnné
  - R : BB $\times$ VAR $\rightarrow$ Bool – blok čte proměnnou před prvním zápisem
  - Triviální algoritmus
- Globální analýza
  - Platnost proměnných na začátku každého základního bloku
  - L : BB $\times$ VAR $\rightarrow$ Bool – proměnná je živá na začátku bloku
  - Polynomiální algoritmus
- Dopočet
  - Platnost proměnných na koncích bloků a uvnitř bloků
  - Detekce čtení nezapsaných proměnných
  - Triviální algoritmus
- Vylepšení
  - Určení komponent souvislosti a přeznačení proměnných

Algoritmus: Určení rozsahů platnosti

- Vstup
  - (BB,CF) – graf control flow
  - W(b,v) – blok b zapisuje do proměnné v
  - R(b,v) – blok b čte proměnnou v před prvním zápisem do ní
- Výstup
  - L(b,v) – proměnná v je živá na začátku bloku b

```
for each b in BB
  L(b,v) = R(b,v);
do {
  for each <b1,b2> in CF
    L(b1,v) |= ~ W(b1,v) & L(b2,v);
} while changed;
```

- Algoritmus se provádí vektorově
  - Pro všechny proměnné (v) najednou
    - Překladač využije SIMD instrukce
  - O(|BB|*|CF|*|VAR|)

Určení rozhraní BB

ENTER
LD_I32(Px) LD_I32(Py)
GT_I32
JC

LD_I32(Py) LD_I32(Px)
ST_I32(Py) ST_I32(Vz)
LD_I32(Vz)
ST_I32(Px)

LD_I32(Py) LD_I32(Px)
ST_I32(Py) MOD_I32
ST_I32(Vz)
LD_I32(Vz)
ST_I32(Px)

LD_I32(Px)
GTC_I32(C1)
JC

LD_I32(Py)
RET_I32

ENTER
LD_I32(Px) LD_I32(Py)
GT_I32
JC

LD_I32(Py) LD_I32(Px)
ST_I32(Py) ST_I32(Vz)
LD_I32(Vz)
ST_I32(Px)

LD_I32(Py) LD_I32(Px)
ST_I32(Py) MOD_I32
ST_I32(Vz)
LD_I32(Vz)
ST_I32(Px)

LD_I32(Px)
GTC_I32(C1)
JC

LD_I32(Py)
RET_I32

Nesekvenční mezikód s rozhraními BB

```
int gcd( int x, int y)
{ int z;
  if ( x > y )
  {
    z = y;
    y = x;
    x = z;
  }
  while ( x > 0 )
  {
    z = y % x;
    y = x;
    x = z;
  }
  return y;
}
```

ENTER
LD_I32(Px) LD_I32(Py)
GT_I32
JC

LD_I32(Py) LD_I32(Px)
ST_I32(Py) ST_I32(Vz)
LD_I32(Vz)
ST_I32(Px)

LD_I32(Px)
GTC_I32(C1)
JC

LD_I32(Py) LD_I32(Px)
ST_I32(Py) MOD_I32
ST_I32(Vz)
LD_I32(Vz)
ST_I32(Px)

LD_I32(Py)
RET_I32

```
CONST:(C1,I32,0)

PROC "gcd"
PARAM:(Px,I32,"x"),(Py,I32,"y")
VAR:(Vz,I32,"z")
```

Náhrada proměnných rozhraním BB

ENTER
LD_I32(Px)
LD_I32(Py)
GT_I32
JC

LD_I32(Py)
LD_I32(Px)
ST_I32(Py)
ST_I32(Vz)
LD_I32(Vz)
ST_I32(Px)

LD_I32(Px)
GTC_I32(C1)
JC

LD_I32(Py)
LD_I32(Px)
ST_I32(Py)
MOD_I32
ST_I32(Vz)
LD_I32(Vz)
ST_I32(Px)

LD_I32(Py)
RET_I32

ENTER
GT_I32
JC

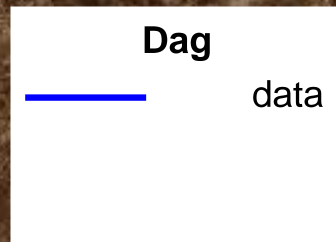
GTC_I32(C1)
JC

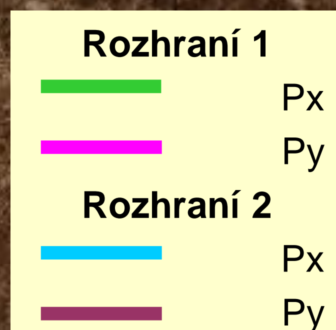
MOD_I32

RET_I32

Nesekvenční mezikód bez proměnných

```
int gcd( int x, int y)
{ int z;
  if ( x > y )
  {
    z = y;
    y = x;
    x = z;
  }
  while ( x > 0 )
  {
    z = y % x;
    y = x;
    x = z;
  }
  return y;
}
```



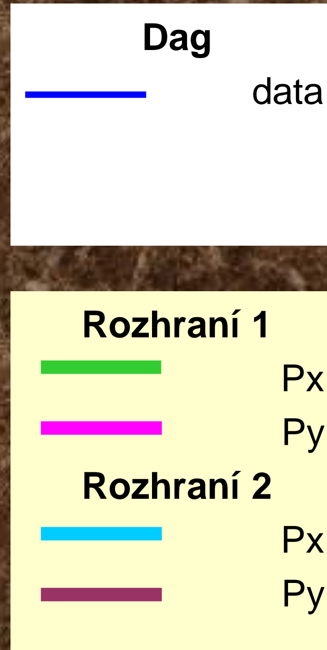


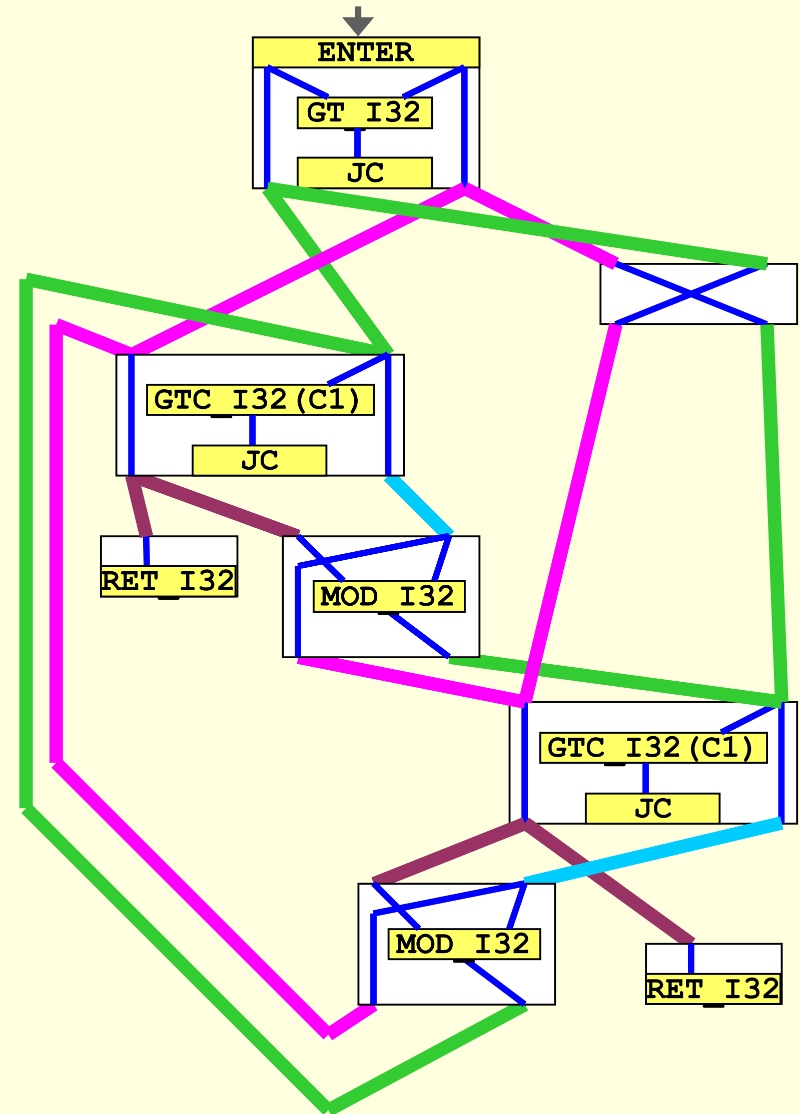

```
CONST:(C1,I32,0)

PROC "gcd"
PARAM:(Px,I32,"x"),(Py,I32,"y")
```

ENTER
GT_I32
JC
MOD_I32
GTC_I32(C1)
JC
RET_I32

Nesekvenční mezikód po duplikaci BB


```
int gcd( int x, int y)
{ int z;
  if ( x > y )
  {
    z = y;
    y = x;
    x = z;
  }
  while ( x > 0 )
  {
    z = y % x;
    y = x;
    x = z;
  }
  return y;
}
```

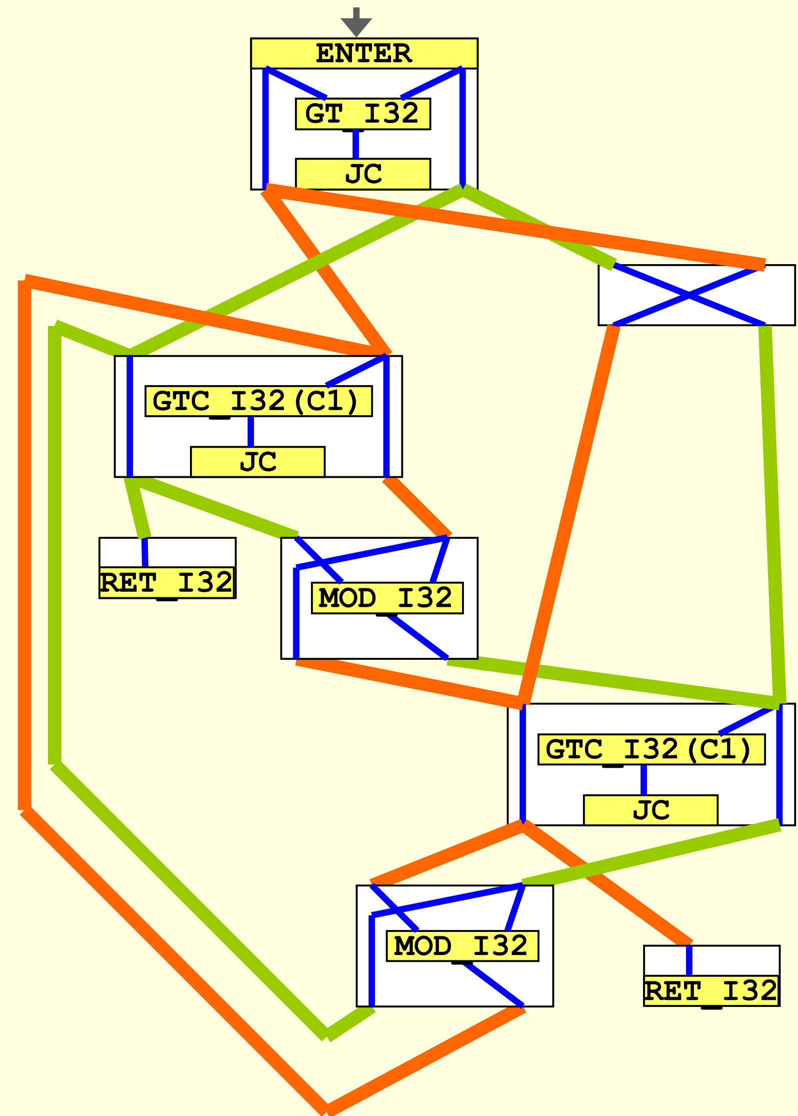





```
CONST:(C1,I32,0)

PROC "gcd"
PARAM:(Px,I32,"x"),(Py,I32,"y")
```

Nesekvenční mezikód po přeznačení

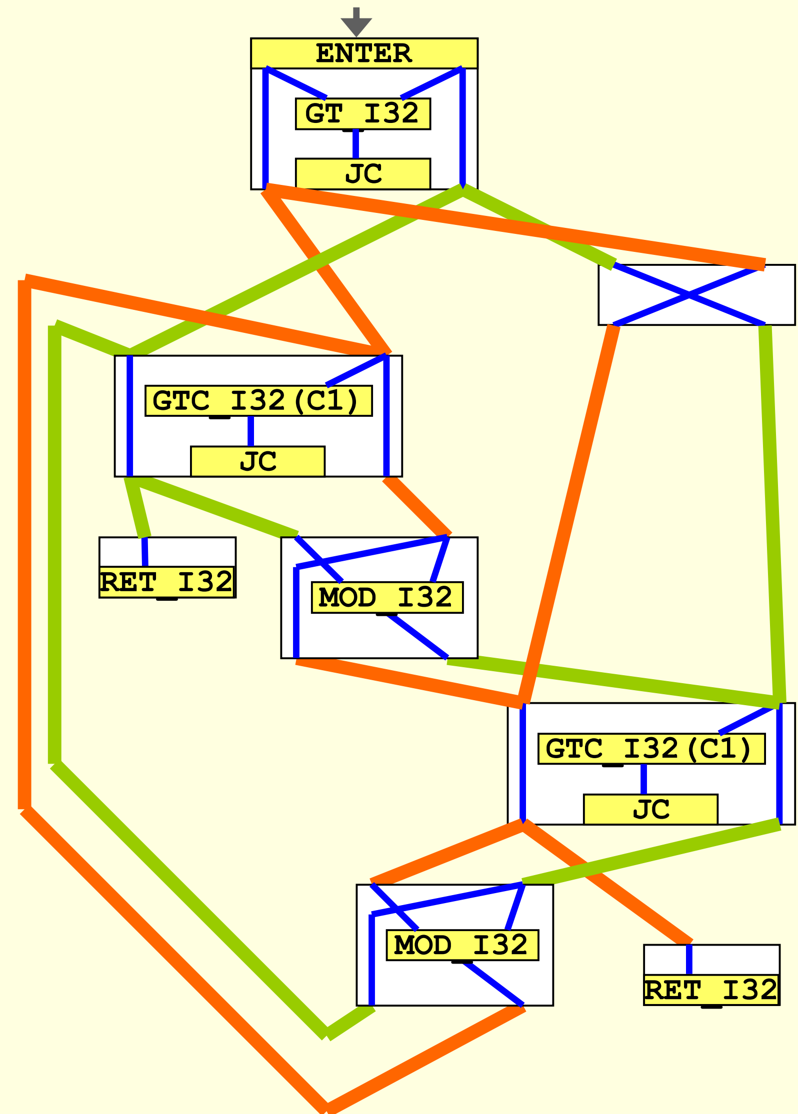
```
int gcd( int x, int y)
{ int z;
  if ( x > y )
  {
    z = y;
    y = x;
    x = z;
  }
  while ( x > 0 )
  {
    z = y % x;
    y = x;
    x = z;
  }
  return y;
}
```




```
CONST:(C1,I32,0)

PROC "gcd"
PARAM:(Px,I32,"x"),(Py,I32,"y")
```

Nesekvenční mezikód po přeznačení


```
int gcd( int x, int y)
{ if ( x > y )
    goto L2;
L1:
  if ( x <= 0 )
    return y;
  y = y % x;
L2:
  if ( y <= 0 )
    return x;
  x = x % y;
  goto L1;
}
```



Nesekvenční mezikód po optimalizaci skoků


```
int gcd( int x, int y)
{ if ( x > y )
    goto L4;
  if ( x <= 0 )
    goto L3;
L1:
  if ( (y = y % x) <= 0 )
    goto L5;
L2:
  if ( (x = x % y) > 0 )
    goto L1;
L3:
  return y;
L4:
  if ( y > 0 )
    goto L2;
L5:
  return x;
}
```