Compiladores

JEAN LUCA BEZ

LUCAS MELLO SCHNORR

Otimização de Código II

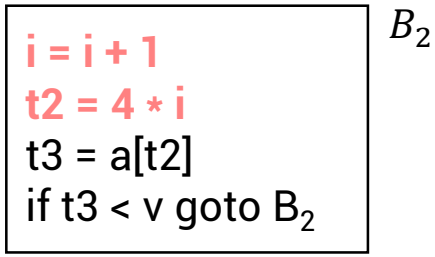
INF01147

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Variáveis de Indução e Redução de Força

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

- Encontrar variáveis de indução em loops e otimizar seu cálculo
- x é considerada **variável de indução** se houver uma constate positiva ou negativa c tal que, toda vez que x for atribuído, seu valor aumenta de acordo com c (`i` e `t2`)

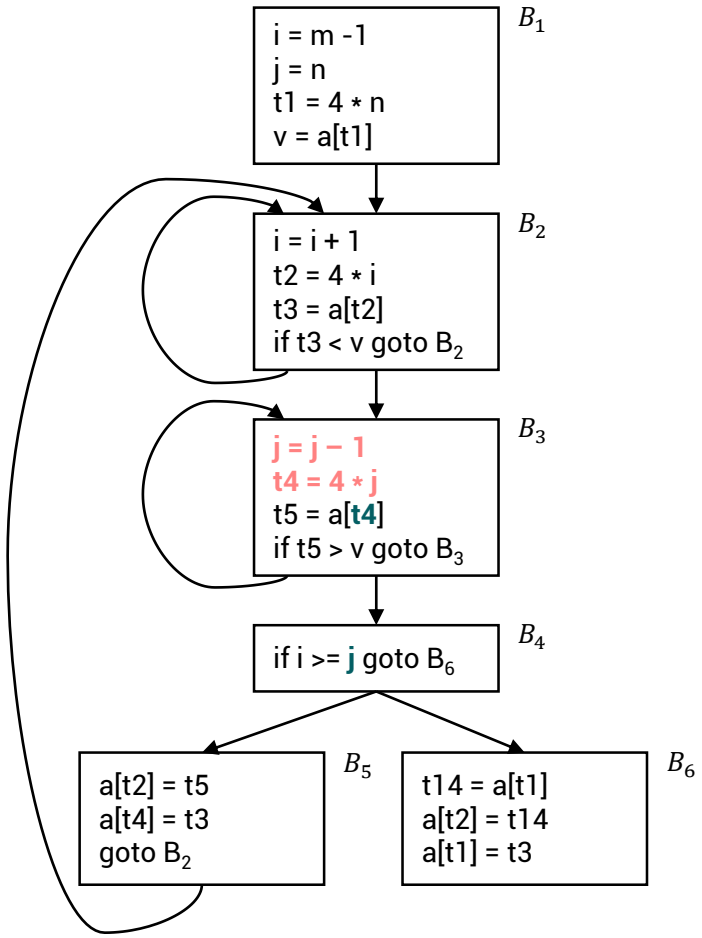


B_2

- Podemos **substituir** uma multiplicação por uma operação menos dispendiosa, como adição
- Transformação é conhecida como **redução de força**
- Variáveis de indução permitem aplicarmos esta transformação

Variáveis de Indução e Redução de Força

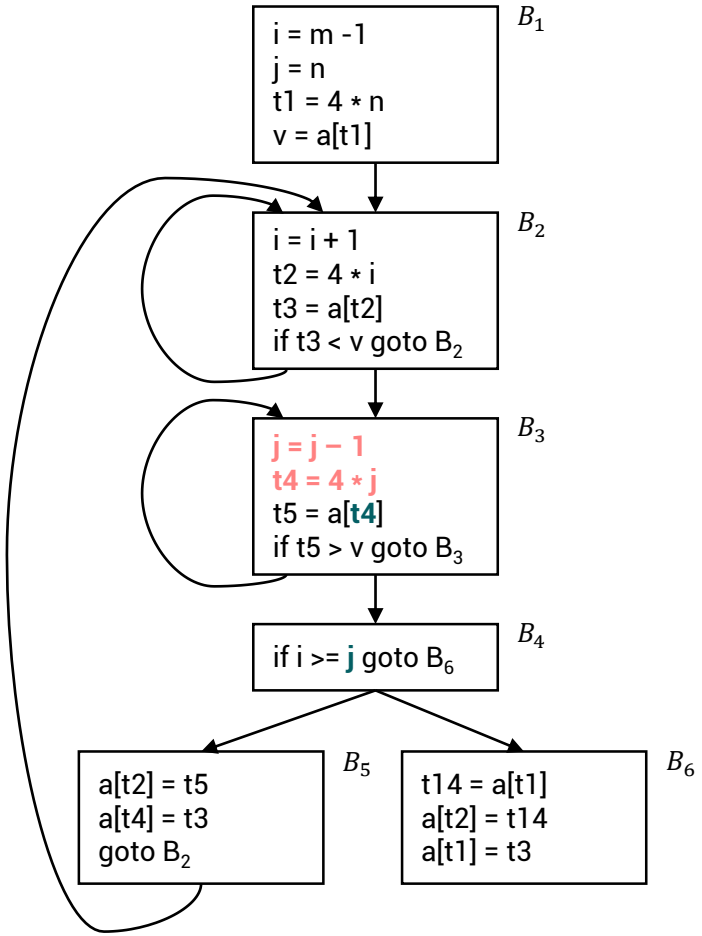
JEAN LUCA BEZ
LUCAS MELLO SCHNORR

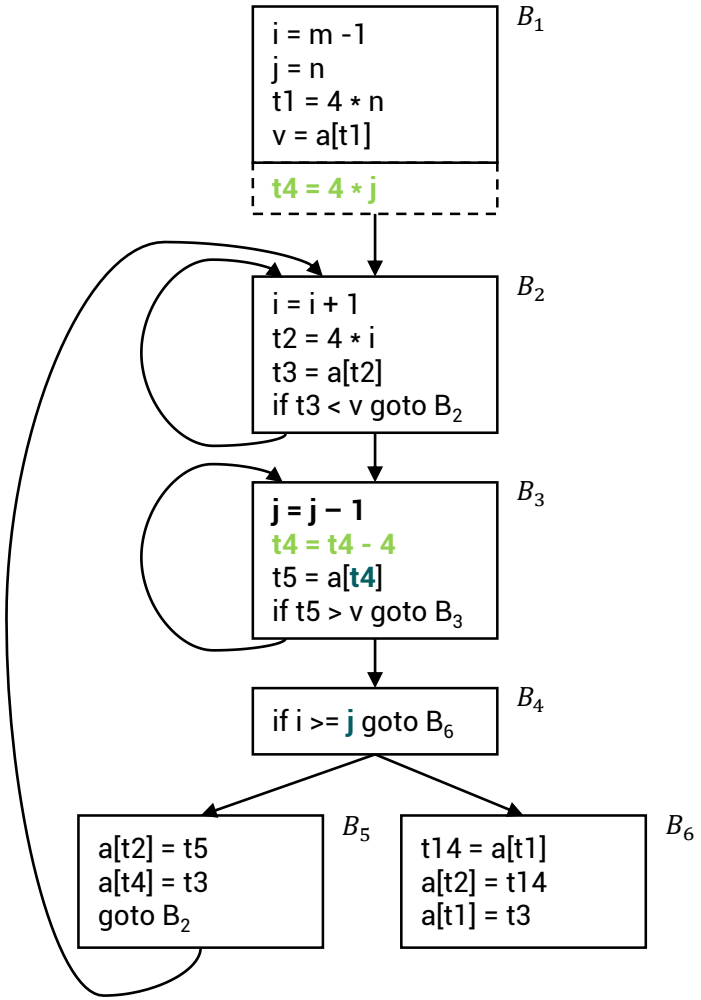


- Iniciamos sempre pelos loops mais internos
- Valores j e $t4$ permanecem em **sincronismo**
- Cada vez que j diminui em 1, o valor de $t4$ diminui em 4 $(4 * j)$
- Par de **variáveis de indução**
- Quando há 2 ou mais geralmente podemos nos livras de todas menos uma
- No caso de B_3 não podemos pois o valor é utilizado posteriormente
  - $t4$ em B_3
  - j em B_4
- Depois de atribuído $t4$ não é mais alterado em B_3
  - Relação se mantém
  - Podemos substituir e ganhar se multiplicação for mais cara que adição

Variáveis de Indução e Redução de Força

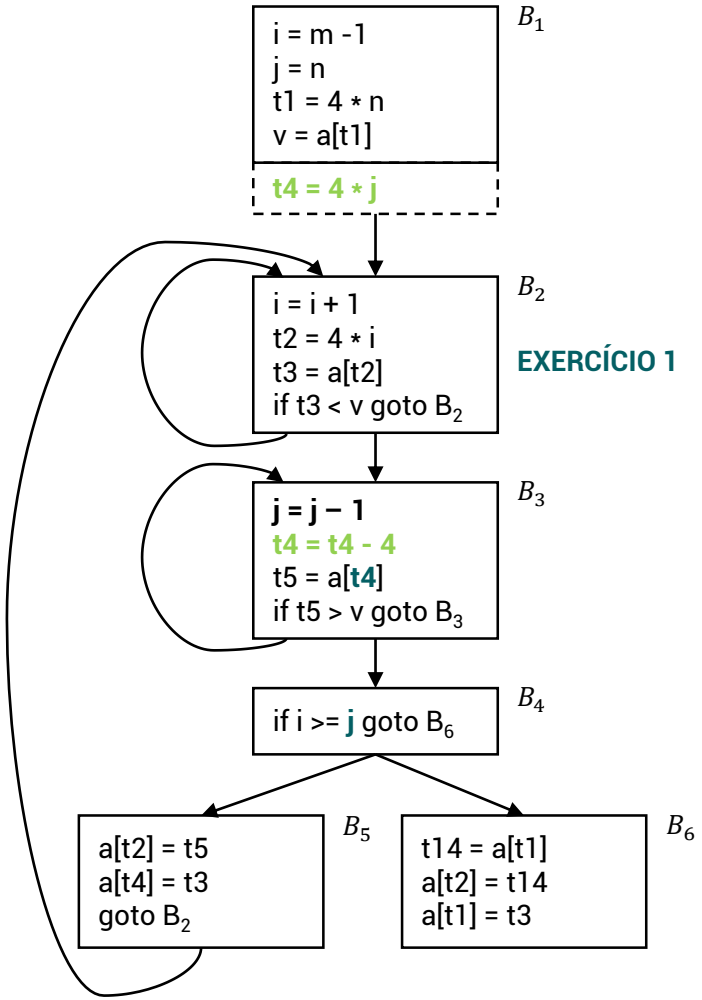
JEAN LUCA BEZ
LUCAS MELLO SCHNORR





Variáveis de Indução e Redução de Força

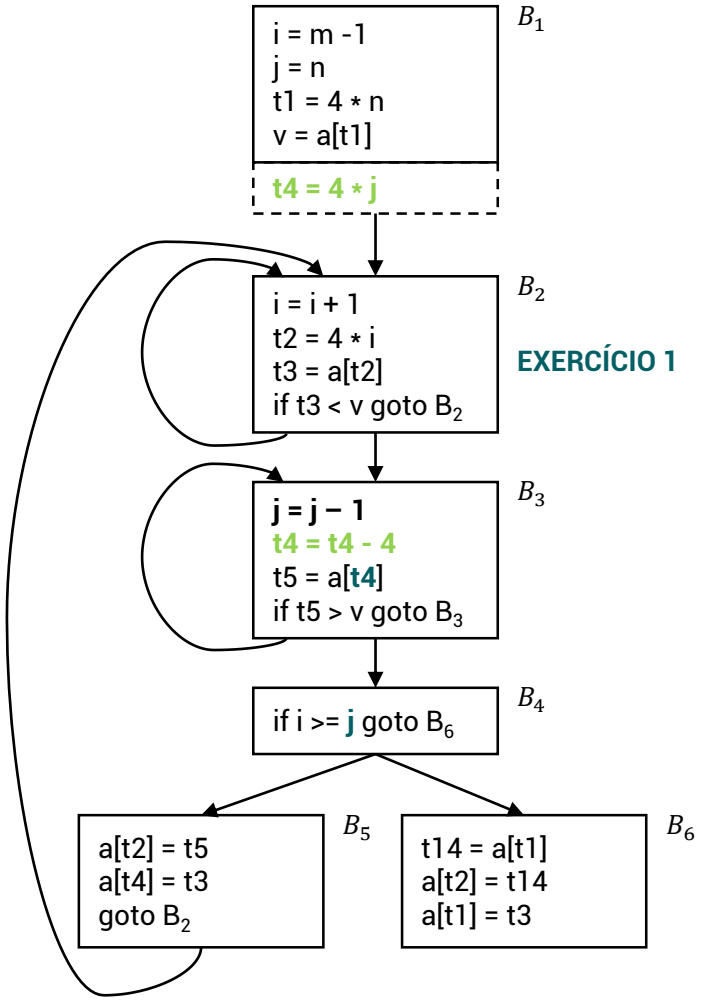
JEAN LUCA BEZ
LUCAS MELLO SCHNORR



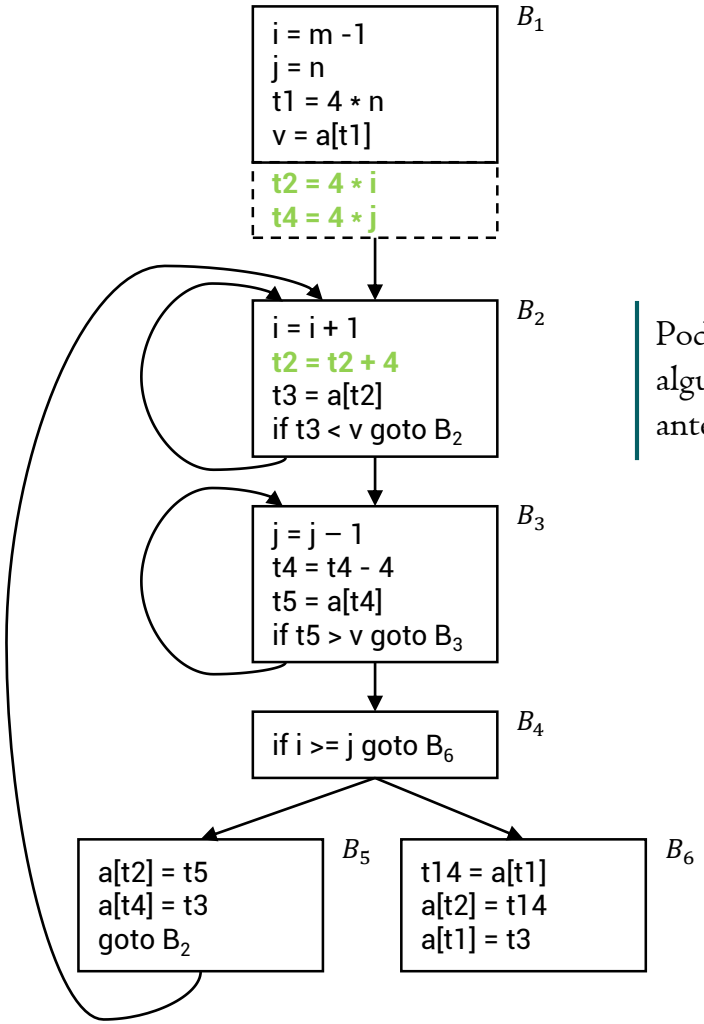
EXERCÍCIO 1

Variáveis de Indução e Redução de Força

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



EXERCÍCIO 1

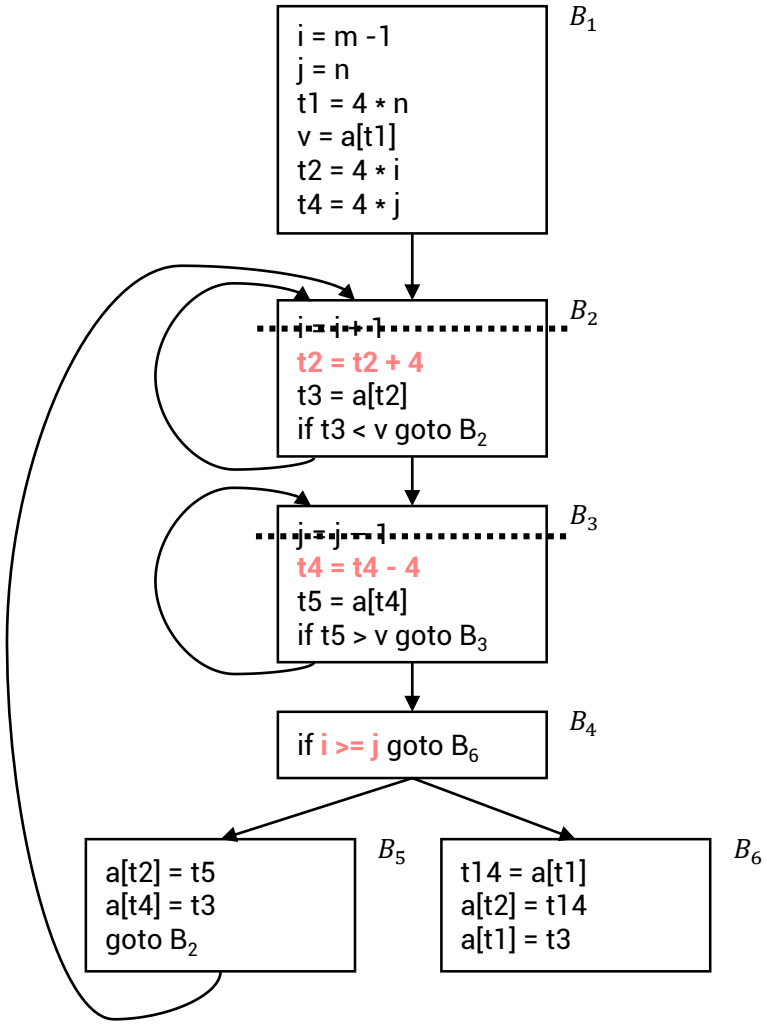


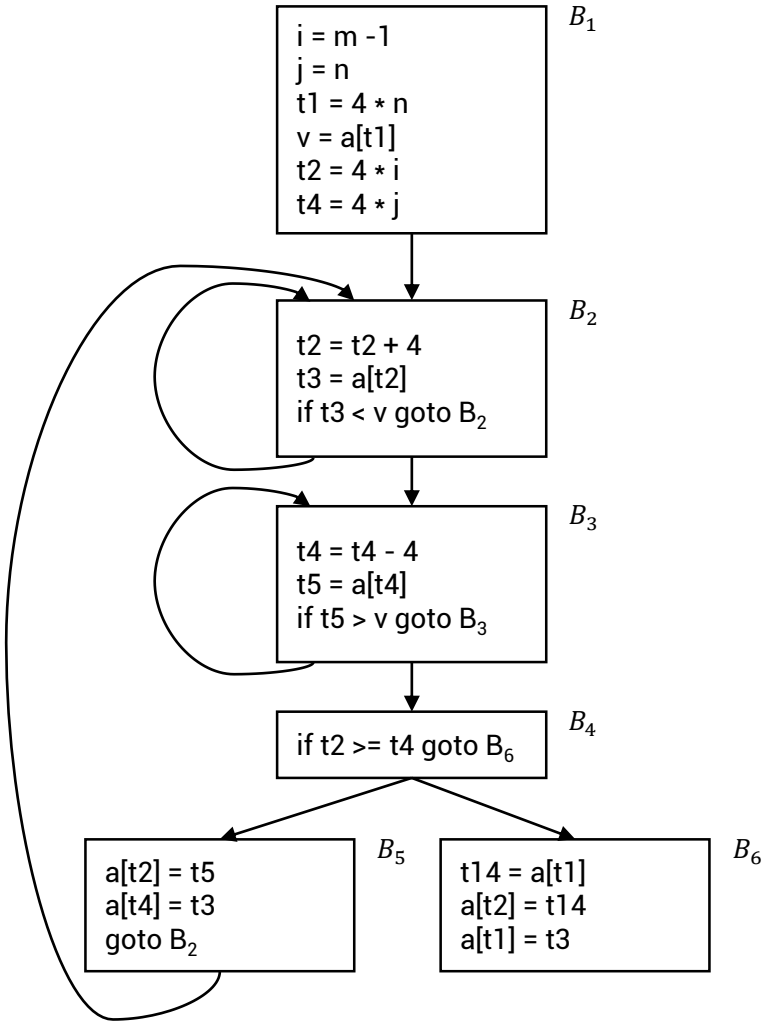
> Podemos aplicar mais alguma otimização que antes não era possível?

Variáveis de Indução e Redução de Força

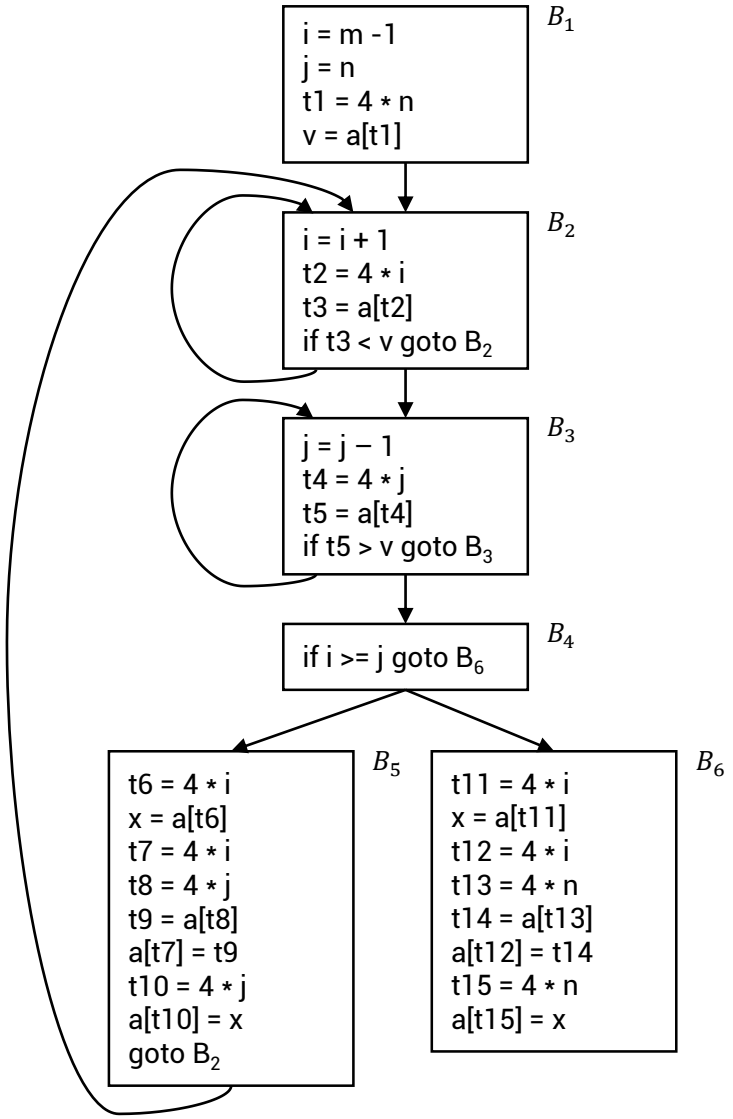
JEAN LUCA BEZ
LUCAS MELLO SCHNORR

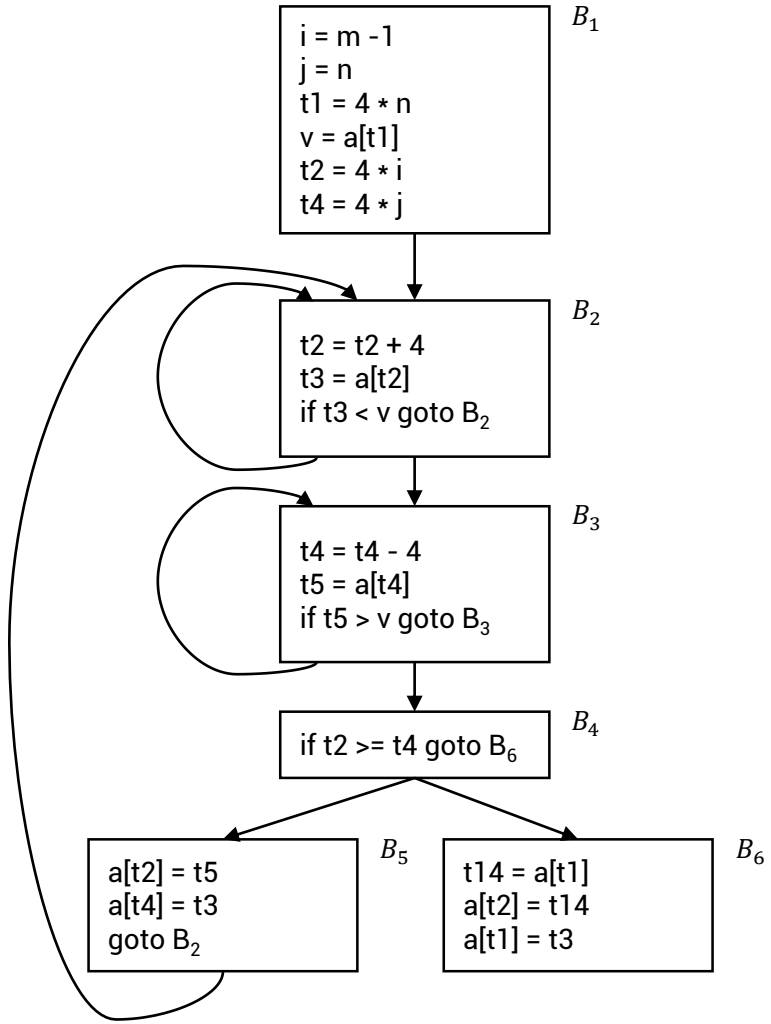
$$i \geq j$$
$$4 * i \geq 4 * j$$
$$t2 \geq t4$$





JEAN LUCA BEZ
LUCAS MELLO SCHNORR





BLOCO	ORIGINAL	OTIMIZADO	DIFERENÇA
B1	4	6	+2
B2	4	3	-1
B3	4	3	-1
B4	1	1	0
B5	9	3	-6
B6	8	3	-5

Desdobramento de *loop*

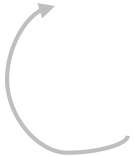
JEAN LUCA BEZ
LUCAS MELLO SCHNORR

HIGH LEVEL

```
for i from 1 to 10
   print "Hello!"
endFor
```

LOW LEVEL

```
set $i to 1

loop_begin:
   print "Hello!"
   increment $i
   jump if not equal $i, 10, loop_begin
```



- Instruções `jump` são **ineficientes**, porque?

Desdobramento de *loop*

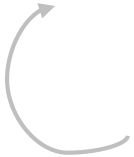
JEAN LUCA BEZ
LUCAS MELLO SCHNORR

HIGH LEVEL

```
for i from 1 to 10
   print "Hello!"
endFor
```

LOW LEVEL

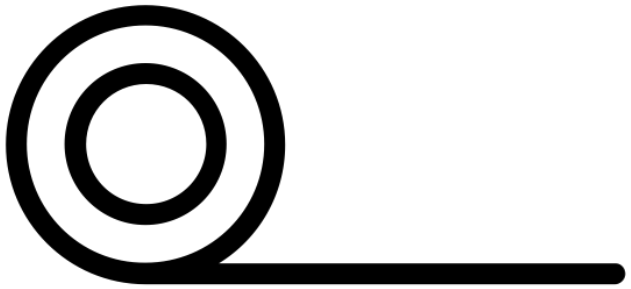
```
set $i to 1

loop_begin:
   print "Hello!"
   increment $i
   jump if not equal $i, 10, loop_begin
```



- Instruções `jump` são **ineficientes**, porque?
- Instruções passam por um **pipeline**
  - Para acelerar colocamos instruções no pipeline antes mesmo da primeira terminar
  - Só se saberá que é `jump` tarde de mais
  - Flush no pipeline a começar novamente
  - Atraso na execução

Desdobramento de *loop*

JEAN LUCA BEZ
LUCAS MELLO SCHNORR



- Desdobramento de loop ou **loop unrolling**
- Otimização aplicada a certos tipos de loops

- Replicar código dentro do corpo do loop um determinado número de vezes
- O número de cópias é chamado de **loop unrolling factor**
- O número de iterações é dividido por este fator

Desdobramento de *loop*

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

- Objetivo é **reduzir** número de jumps
- Vantagens:
  - Reduz overheads relacionadas a manutenção de loops
  - Aritmética para incrementar o parâmetro do loop
  - Verificar condição de término do loop

HIGH LEVEL

```
for i from 1 to 10
   print "Hello!"
endFor
```

Desdobramento de *loop*

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

ORIGINAL

```
for i from 0 to 100
   print "Hello!"
endFor
```

UNROLLED

```
for i from 0 to 20
   print "Hello!"
   print "Hello!"
   print "Hello!"
   print "Hello!"
   print "Hello!"
endFor
```

Eliminamos:

- ? `jumps`
- ? incrementos de i
- ? verificações de término

Desdobramento de *loop*

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

ORIGINAL

```
for i from 0 to 100
   print "Hello!"
endFor
```

UNROLLED

```
for i from 0 to 20
   print "Hello!"
   print "Hello!"
   print "Hello!"
   print "Hello!"
   print "Hello!"
endFor
```

Eliminamos:

80 `jumps`

80 incrementos de i

80 verificações de término

Desdobramento de *loop*

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

CÓDIGO ORIGINAL

```
for (i = 0; i < N; i++) {
   S(i);
}
```

EXERCÍCIO 2

Como ficaria este loop ao desdobrar por um fator de 4?

Desdobramento de *loop*

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

CÓDIGO ORIGINAL

```
for (i = 0; i < N; i++) {
   S(i);
}
```

DESDOBRANDO UM LOOP FOR

```
for (i = 0; i+4 < N; i+=4) {
   S(i);
   S(i+1);
   S(i+2);
   S(i+3);
}
for ( ; i < N; i++) {
   S(i);
}
```

EXERCÍCIO 2

Como ficaria este loop ao desdobrar por um fator de 4?

Desdobramento de *loop*

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

EXERCÍCIO 3

Como ficaria este loop ao desdobrar por um fator de 4?

CÓDIGO ORIGINAL

```
repeat
   S;
until C;
```

Desdobramento de *loop*

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

EXERCÍCIO 3

Como ficaria este loop ao desdobrar por um fator de 4?

CÓDIGO ORIGINAL

```
repeat
   S;
until C;
```

DESDOBRANDO UM LOOP REPEAT

```
repeat {
   S;
   if (C) break;
   S;
   if (C) break;
   S;
   if (C) break;
   S;
} until C;
```

Desdobramento de *loop*

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

CÓDIGO ORIGINAL

```
for (i = 0; i < N; i++) {
   S(i);
}
```

CÓDIGO ORIGINAL

```
repeat
   S;
until C;
```

DESDOBRANDO UM LOOP FOR

```
for (i = 0; i+4 < N; i+=4) {
   S(i);
   S(i+1);
   S(i+2);
   S(i+3);
}
for ( ; i < N; i++) {
   S(i);
}
```

DESDOBRANDO UM LOOP REPEAT

```
repeat {
   S;
   if (C) break;
   S;
   if (C) break;
   S;
   if (C) break;
   S;
} until C;
```

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

ORIGINAL

```
for (i=0; i<N; i++)
   a[i]=0;
for (i=0; i<N; i++)
   b[i]=0;
```

LOOP FUSION

```
for (i=0; i<N; i++) {
   a[i]=0;
   b[i]=0;
}
```

- As vezes, dois loops podem ser **substituídos** por um
- Reduzir overhead dos loops
- Acelerar a execução
- Reduzir o espaço de código
- Condições:
  - Índices dos loops devem ser iguais
  - Computação em um loop não pode depender da computação do outro loop

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

EXERCÍCIO 4

Como ficaria este código ao utilizar loop fusion?

CÓDIGO ORIGINAL

```
(1) for i=1 to N do
(2)    A[i] = B[i] + 1
(3) endfor
(4) for i=1 to N do
(5)    C[i] = A[i] / 2
(6) endfor
(7) for i=1 to N do
(8)    D[i] = 1 / C[i+1]
(9) endfor
```

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

EXERCÍCIO 4

Como ficaria este código ao utilizar loop fusion?

CÓDIGO ORIGINAL

```
(1) for i=1 to N do
(2)    A[i] = B[i] + 1
(3) endfor
(4) for i=1 to N do
(5)     C[i] = A[i] / 2
(6) endfor
(7) for i=1 to N do
(8)     D[i] = 1 / C[i+1]
(9) endfor
```

CÓDIGO OTIMIZADO

```
(1) for i=1 to N do
(2)     A[i] = B[i] + 1
(5)     C[i] = A[i] / 2
(6) endfor
(7) for i=1 to N do
(8)     D[i] = 1 / C[i+1]
(9) endfor
```

Unswitching

JEAN LUCA BEZ
LUCAS MELLO SCHNORR

CÓDIGO ORIGINAL

```
for each polygon in scene
   if time == "day" then
      draw polygon in "yellow"
   else if time == "night" then
      draw polygon in "blue"
   endIf
endFor
```

CÓDIGO OTIMIZADO

```
if time == "day" then
   for each polygon in scene
      draw polygon in "yellow“
   endFor
else if time == "night" then
   for each polygon in scene
      draw polygon in "blue"
   endFor
endIf
```

- Remover uma condição que não é alterada
- Qual a vantagem?
  - Não precisamos verificar a cada iteração