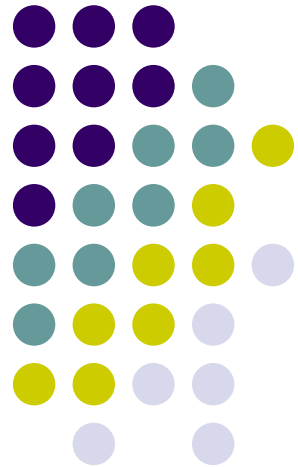
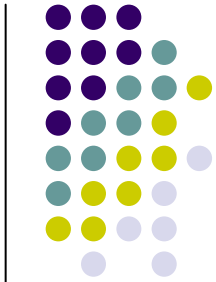
Teknik Kompiler 11

oleh: **antonius rachmat c, s.kom**



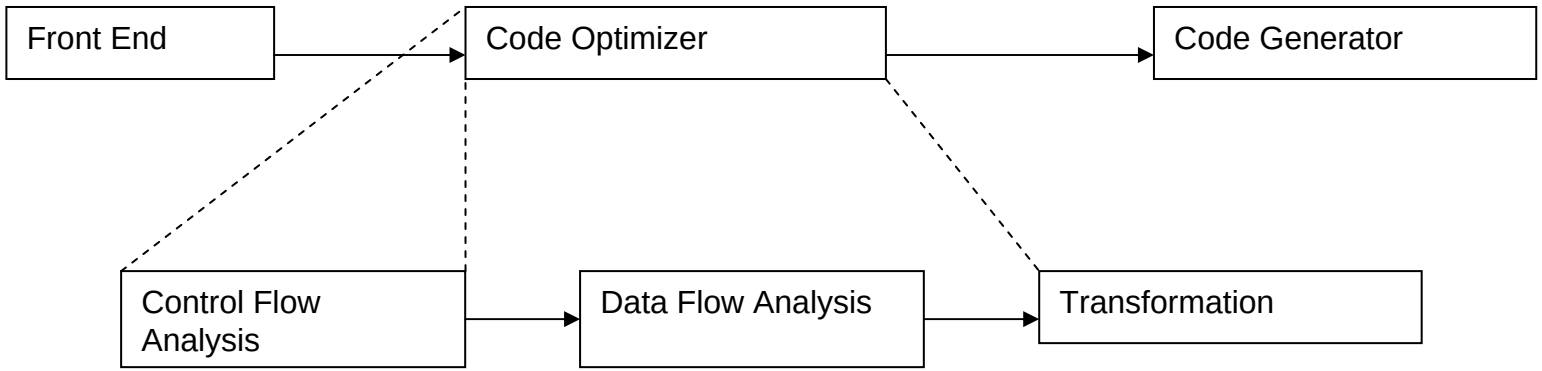
Code Optimization



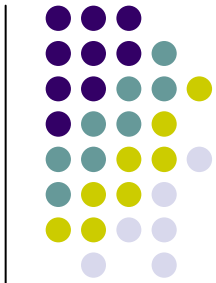
- Adalah bagian yang melakukan tranformasi code namun tetap menjaga kebenaran source code dan bahkan meningkatkan performa dari hasil input.
- Optimisasi kode digunakan untuk menghasilkan kode program yang lebih kecil dan lebih cepat eksekusinya.
- Kriteria Transformasi
  - Tetap menjaga makna source dan output program
  - Dapat mempercepat waktu eksekusi
  - Proses transformasi tidak terlalu lama

Isu-isu pada Code Optimization

- Teknik optimisasi bisa diterapkan pada:
  - source code
  - intermediate code
  - target machine code

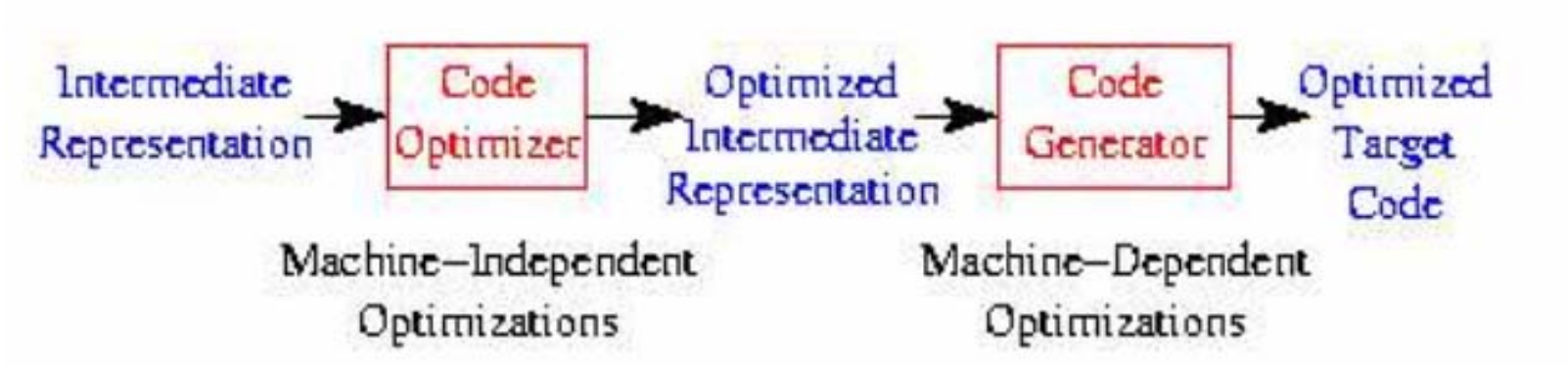


Pembagian Teknik Optimasi



- Berdasarkan tingkat ketergantungan pada mesin, teknik optimasi dibagi menjadi:
- ***Machine Dependent Optimizer***
  - Kode dioptimasi sehingga optimal dan efisien pada mesin tertentu saja. Optimasi ini memerlukan informasi mengenai feature yang ada pada mesin tujuan, mengambil keuntungan dari mesin tujuan agar kode lebih pendek dan eksekusinya cepat.
  - Yang dioptimasi adalah object code
- ***Machine Independent Optimizer***
  - Optimasi ini bisa diaplikasikan tanpa tergantung pada mesin tujuan.
  - Yang dioptimasi adalah intermediate code

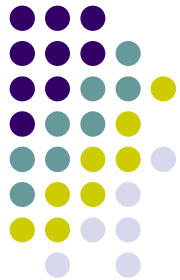
Example




```
x := x + 0
x := x * 1
x := y**2
z := 2*x
```

```
x := y*y
z := x + x
```

Example (2)

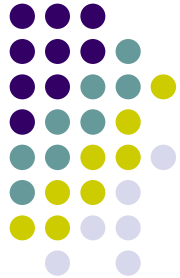


- T6 = 4 * I
- X = A[T6]
- T7 = 4 * I
- T8 = 4 * j
- T9 = A[T8]
- A[T7] = T9
- T10 = 4 * j
- A[T10] = X
- …

T6 = 4 * I
X = A[T6]
T8 = 4 * j
T9 = A[T8]
A[T6] = T9
A[T8] = X

….

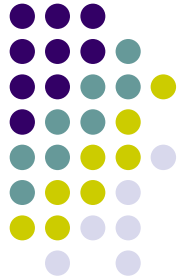
Teknik-teknik Optimasi Lokal



Peephole (Lubang Intip)

- Untuk lokal intermediate code atau bahasa assembly.
- Melihat dalam satu sekuens dari statement
- Mengkoreksi prosedur yang inefektif karena prosedur tersebut terlalu “berlebihan”.
- Digunakan untuk menghilangkan redundant instructions
- **Unreachable code**
  - L1: IF X > 2 GOTO L2
  - GOTO L3
  - L2: GOTO L4
  - GOTO L1 ; never executed
  - L3: X := X + 1
  - GOTO L1
  - L4: ...

Teknik-teknik Optimasi

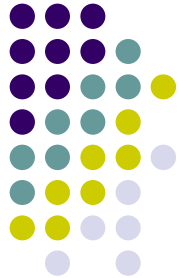


- Flow-of-control fixes: remove redundant
  - L1: IF X > 2 GOTO L4
  - X := X + 1
  - GOTO L1
  - L4: ...

More Peephole Optimizations

- Algebraic Simplification
  - x + 0 = 0 + x => x
  - x - 0 => x
  - x * 1 = 1 * x => x
  - x/1 => x

Teknik-teknik Optimasi



Strength Reduction

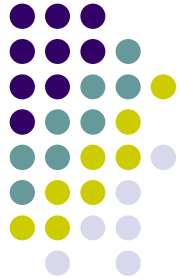
- Target hardware may have cheaper ways todo certain operations. E.g., multiplication or division by a power of 2 is better done by shifting.
  - MUL R1, 8 => SHL R1, 3, R2
  - X ^ 2 => X * X
  - 2 * X => X + X

Code Motion

```
T := b * d;
Case p of
   1: c:=a+T;
   2: m:=T-r;
   3: f:=a-T;
   4: c:=q/(T-r);
End;
```

```
Case p of
  1: c := a + b * d;
  2: m := b * d – r;
  3: f := a – b * d;
  4: c := q / (b * d + r);
End;
```

Optimasi Lokal

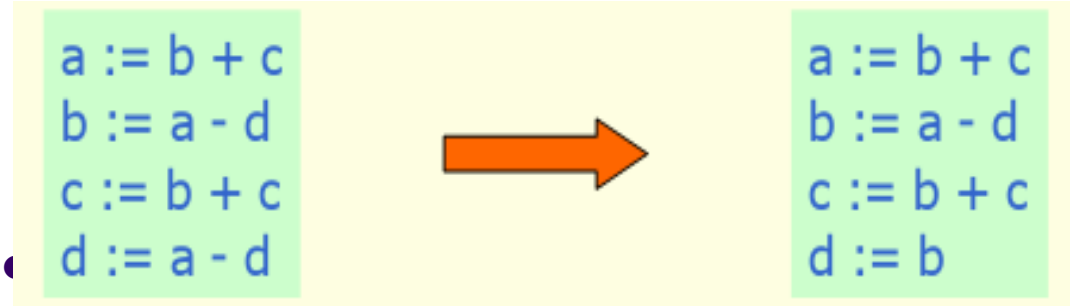


- Constant Folding
  - Mengganti konstanta atau ekspresi yang bisa dievaluasi pada saat compile time dengan nilai komputasinya.
  - Contoh:
    - X := 4 + 2 – B;
  - Diganti menjadi:
    - X := 6 – B;

Optimasi Lokal

- Redundant Common Sub Expression Elimination
  - Membuang/mengganti ekspresi yang sudah pernah dikomputasi dan ternyata digunakan lagi pada urutan instruksi berikutnya.
  - Contoh:
    - A:=**B+C**+Z;
    - D:=Y+**B+C**;
  - Kompuatasi B+C harusnya bisa diperoleh dari baris pertama saja tanpa harus dikomputasi ulang. Catatan: asal belum ada perubahan nilai variabel B dan C.
  - Contoh:
    - i := **j + 1**
    - a[i] := a[i] + **j + 1**

Optimasi Lokal



- a := b + 1 => a := b + 1
- c := a c := a ; maybe can now omit
- d := c d := a

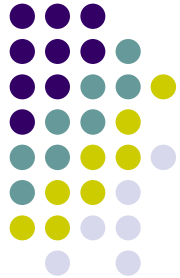
- a := d + e; b:= d + e; => a := d + e; b:= a;

- **Algebraic Identities**
  - E.g., use associativity and commutativity of +
  - a := b + c => a := b + c
  - b := c + d + b b := b + c + d ; now do CSE

Optimasi Lokal

- Optimasi pada Perulangan
  - **Loop Unrolling**
  - Mengganti suatu loop dengan menulis statement dalam loop beberapa kali. Hal ini didasari oleh pemikiran bahwa pada level rendah loop akan melakukan operasi:
    - Inisialisasi nilai awal variabel loop. Dilakukan sekali saja pada awal loop
    - Pengetesan kondisi loop
    - Penambahan/pengurangan nilai (increment/decrement) variabel loop dengan jumlah tertentu.
    - Operasi yang terjadi pada loop body.

Optimasi Lokal (loop)



```
For i:=1 to 2 do
    A[i] := 0;
```

Diperlukan :

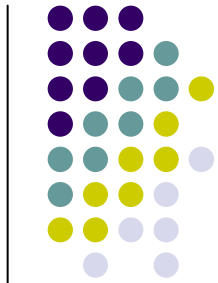
- Satu instruksi pengesetan variabel awal (i:=1)
- Diperlukan pengecekan kondisi
- Diperlukan increment variabel loop
- Diperlukan 2x operasi assigment A[i] :=0
- Jadi total ada 5 instruksi

Dioptimasi jadi:

```
A[1] := 0;
A[2] := 0;
```

Hanya membutuhkan 2 instruksi.

Optimasi Lokal



Frequency Reduction

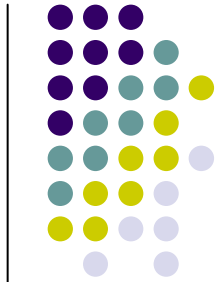
- Pemindahan statement ke tempat yang lebih jarang dieksekusi

```
For i:=1 to 10 do
Begin
        X := 5;
        A := A + 2;
End;
```

- Terlihat bahwa X tidak mengalami perubahan nilai, tetap 5 terus selama 10 kali loop. Maka dioptimasi menjadi:

```
X := 5;
For i:=1 to 10 do
Begin
        A := A + 2;
End;
```

Optimasi Lokal



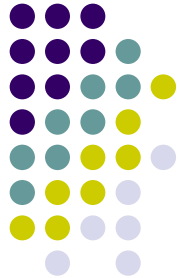
Strength Reduction

- Penggantian jenis operasi tertentu dengan jenis operasi lain yang lebih cepat ekesekusinya. Misal operasi perkalian lebih lambat maka bisa diganti dengan operasi penjumlahan

- Contoh lain:

  A=A+1 bisa diganti dengan **INC(A)**

Optimasi Global



- Dilakukan pada seluruh program
- **Dead Code**
  - Kode yang tidak pernah dieksekusi
    - X:=5;
    - If X = 0 then B:=B+1;
- **Unused Parameter**
  - Parameter yang tidak pernah dipakai

```
Procedure Kali(a,b,c:integer);
Begin
   Result := a*b;
End;
```

Optimasi Global

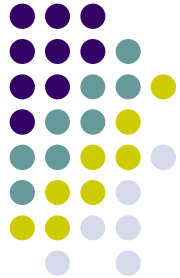
- **Unused Variabel**
  - Variabel yang tidak terpakai
    - Procedure Coba;
    - Var a,b,c;
    - Begin
    - Result:=a*b;
    - End;
  - **Variabel tanpa nilai awal**
    - Var a,b:integer;
    - Begin
    - A:=1;
    - A:=A*B;
    - End;

Latihan

- A=B+10*4
- C=B+D
- F=B+D-G
- FOR I=1 TO 100 DO
- BEGIN
  - X = X+1
  - A=A+X
  - B=7

  END

NEXT



- **Code Generation**