컴파일러 입문

제 11 장

코드 최적화

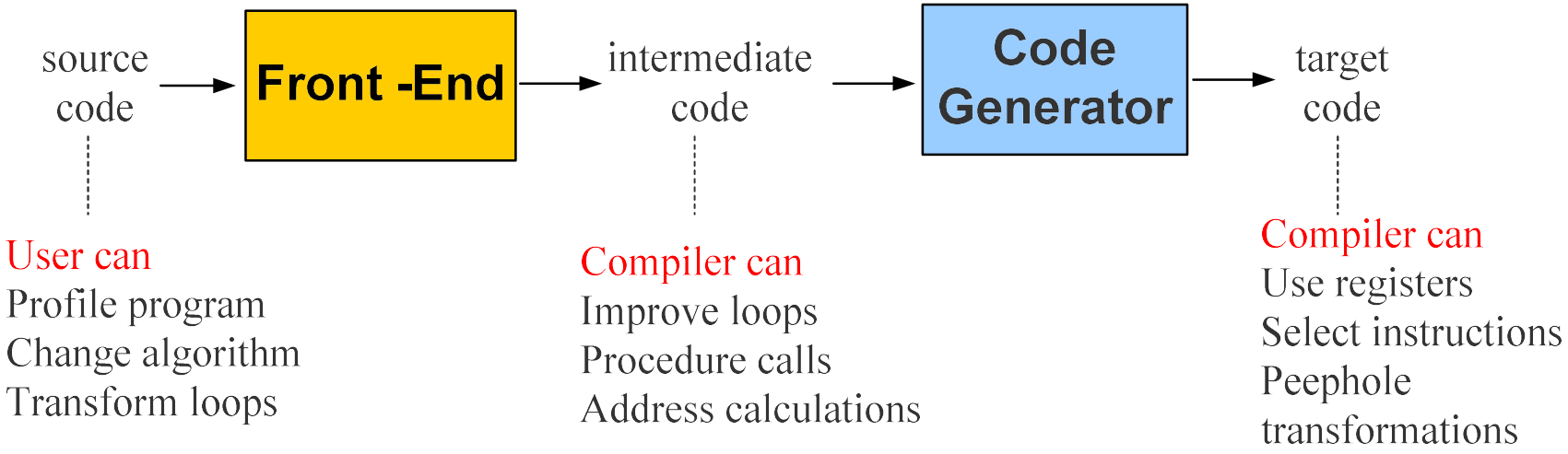
목 차

- 기본 블록
- 지역 최적화
- 루프 최적화
- 전역 최적화
- 기계 종속적 최적화

Code Optimization

- **Definition**
  - Preserve programming meaning
  - Speed up on average
  - Be worth the effort
- **분류**
  - **최적화 범위**
    - **지역 최적화**(local optimization)
    - **전역 최적화**(global optimization)
  - **최적화 코드**
    - **기계 독립적 최적화**(machine-independent optimzation)
    - **기계 종속적 최적화**(machine-dependent optimization)

Getting Better Performance

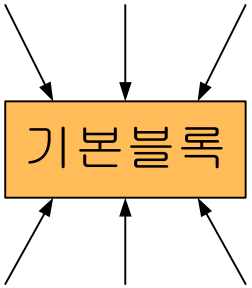


Basic Block (1/4)

- 지역 최적화의 기본 단위
- 시작부터 끝까지 순서적으로만 수행되는 문장 범위

[정의] 기본 블록

블록의 시작과 끝을 제외하고 블록의 내부 또는 외부로의 분기가 발생하지 않는 분해된 기본 코드들의 시퀀스

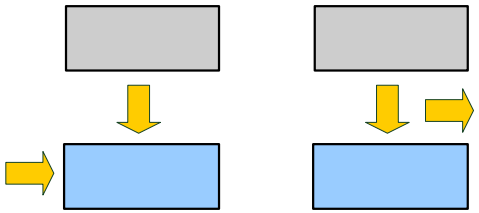


Basic Block (2/4)

- 기본 블록 구성
  - leader와 다음 leader 이전에 나타나는 모든 코드

[정의] leader

- 프로그램의 시작 문장
- 조건부 분기 또는 무조건 분기의 목적지에 있는 문장



- 조건부 분기 바로 다음에 위치하는 문장

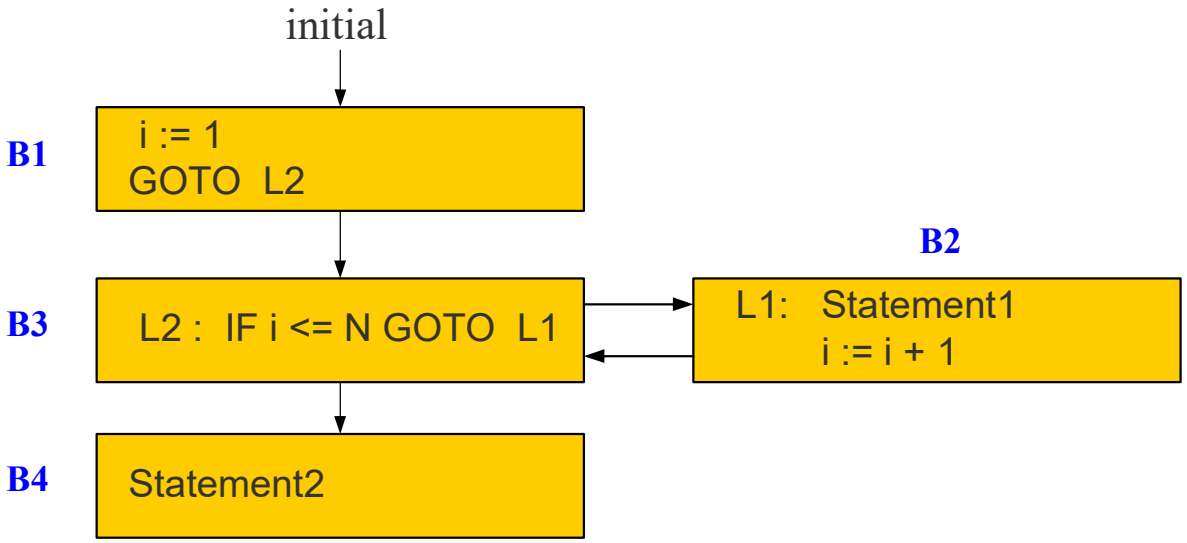
Basic Block (3/4)

```
FOR  i := 1 TO  N DO
      Statement1;
Statement2;
```

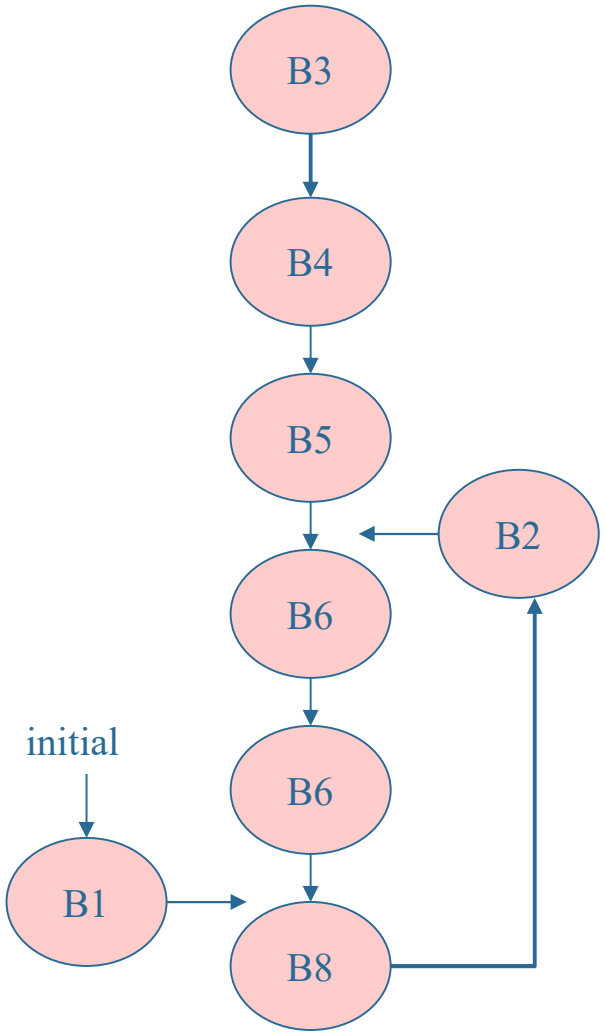
```
      i := 1
      GOTO  L2
L1 :  Statement1
      i := i + 1;
L2 :  IF i <= N GOTO  L1
      Statement2
```

Basic Block (4/4)

- 흐름 그래프(Flow Graph)
  - 기본 블록에 제어 흐름 정보를 추가한 방향 그래프
  - 전역 최적화에 필수



```
            I := 1
            GOTO    Itest                    B1
-------------------------------------------------
Iloop :     J := 1
            GOTO   Jtest                     B2
-------------------------------------------------
Jloop:      T1 := 4 * J
            T2 := A[T1]
            T3 := J + 1
            T4 := 4 * T3                     B3
            T5 := A[T4]
            IF T2 <= T5 GOTO Jplus
-------------------------------------------------
            T6 := 4 * J
            Temp := A[T6]
            A[T12] := Temp                   B4
-------------------------------------------------
Jplus :     J := J + 1                       B5
-------------------------------------------------
Jtest :     IF J <= I GOTO Jloop             B6
-------------------------------------------------
Iplus :     I := I + 1                       B7
-------------------------------------------------
Itest :     IF I <= n - 1 GOTO ILoop         B8
-------------------------------------------------
```



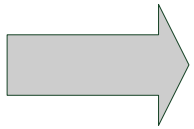
지역 최적화

- 기본 블록 내에서 최적화를 수행
- 기법 :
  - Common subexpression elimination
  - Strength reduction
  - Constant folding
  - Constant propagation
  - Algebraic simplification

Common Subexpression Elimination

- 공통 부분식의 제거
- 공통된 부분이 반복되어 나타나는 경우를 제거하는 방법

```
A := B + C + D;
E := B + C + F;
K := (B + C) * G;
```

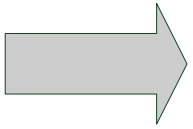


```
T := B + C;
A := T + D;
E := T + F;
K := T * G;
```

Strength Reduction

- 연산 강도 경감
- 연산자의 비용이 적은 연산자로 바꾸는 방법
  - 거듭제곱 -> 승산 -> 가산
  - 제산, 승산 -> 비트이동(shift)

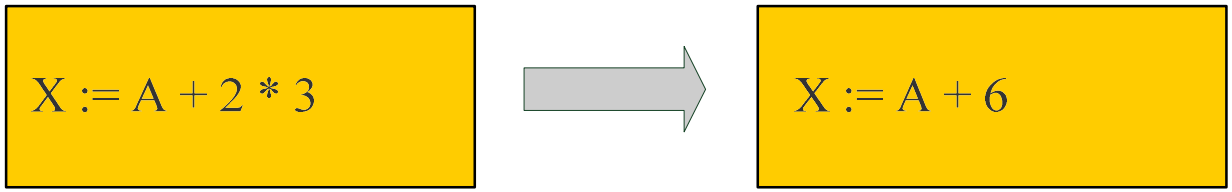
```
A = X ** 2;
Y = 3 * A;
X = A / 5;
X = A / 4;
```



```
A = X * X;
Y = A + A + A;
X = A * 0.2;
X = A rshr 2
```

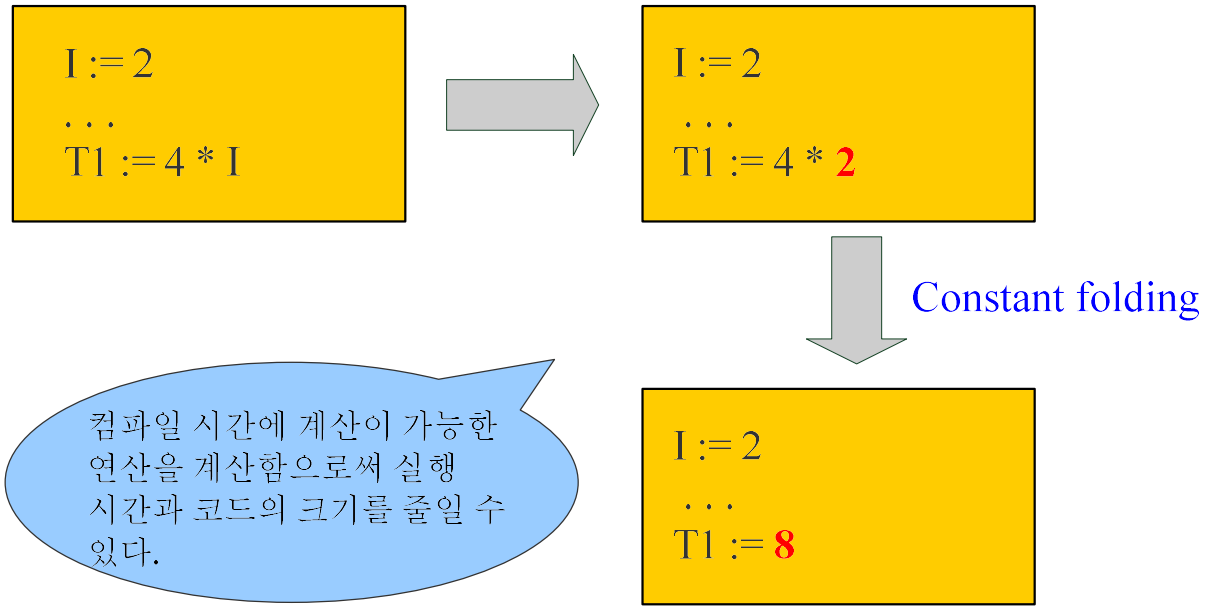
Constant Folding

- 상수 폴딩
- 컴파일 시간에 상수식을 직접 계산하여 그 결과를 사용하는 방법



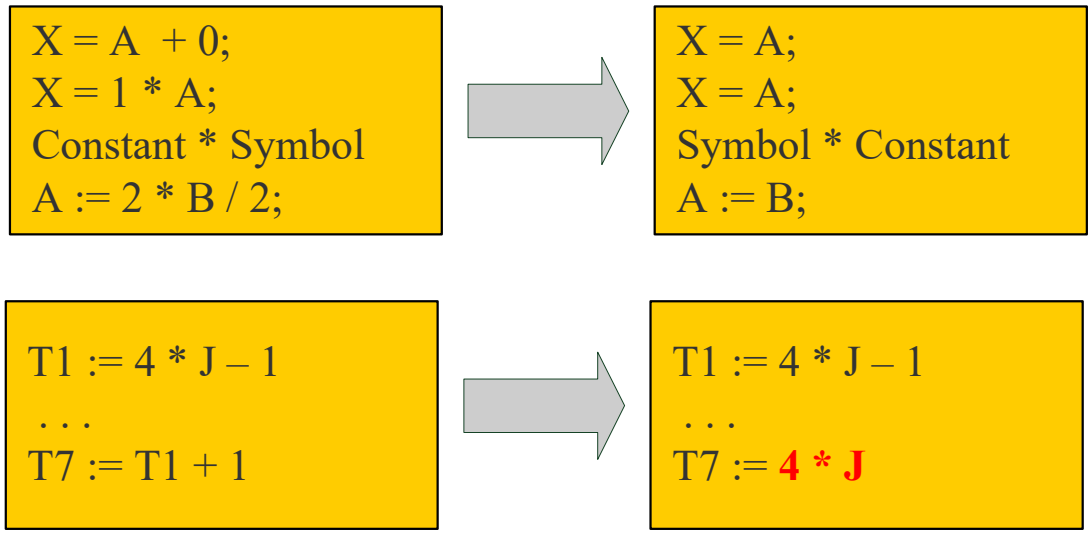
Constant Propagation

- 상수 전파
- 고정된 값을 갖는 변수를 상수로 대치하는 방법



Algebraic Simplification

- 대수학적 간소화
- 수학적인 대수 법칙을 이용하여 식을 간소화하는 방법

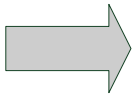


Loop Optimization

- 전체 코드의 10%가 실행시간의 90%를 차지
- 대부분의 실행 시간을 루프 내에서 소모
- 기법 :
  - 루프 불변 코드 이동
  - 연산 강도 경감
  - 루프 언롤링
  - 루프 융합
  - 영으로의 카운트

Loop Invariant

```
FOR  k := 1 TO 1000 DO
      c[k] := 2 * (p – q) * (n – k + 1) / (sqr(n) + n) ;
```

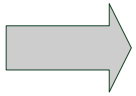


```
fact := 2 * (p – q) ;
denom := sqr(n) + n ;
FOR  k := 1 TO 1000 DO
    c[k] := fact * (n – k + 1) / denom ;
```

Loop Unrolling

- 루프의 반복 횟수를 감소시키는 방법

```
FOR  k := 1 TO 1000 DO
      c[k] := 0 ;
```

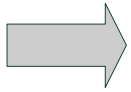


```
FOR  k := 1 TO 1000 STEP 2 DO
      BEGIN
          c[k] := 0 ;
          c[k + 1] := 0 ;
      END
```

Count up to Zero

- 루프의 종료 조건을 검사하는 경우 0인지 아닌지를 검사하는 경우가 효율적

```
FOR  k := 0 TO N - 1 DO
   BEGIN
     …  k
   END
```

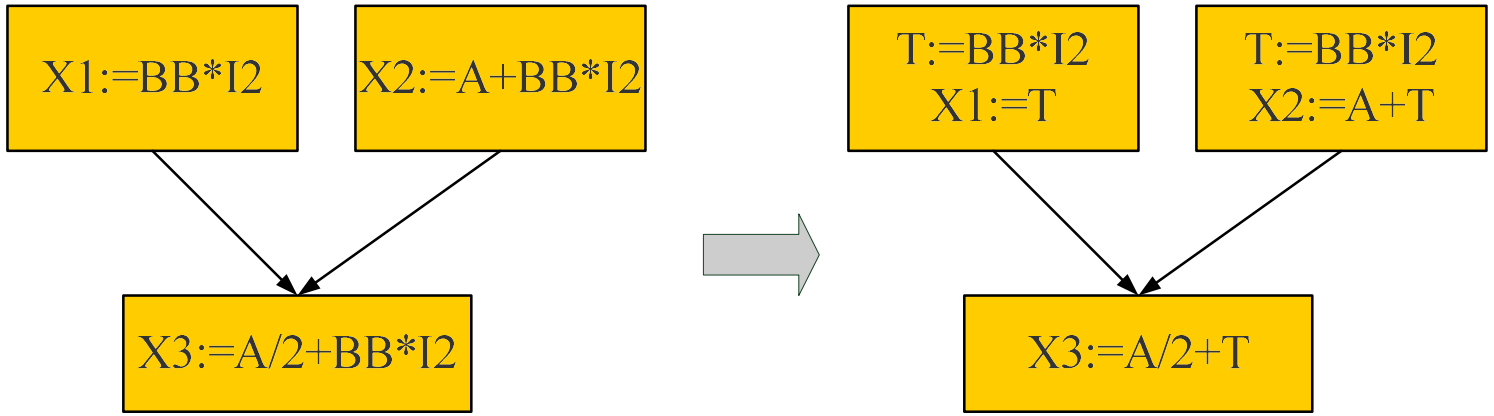


```
FOR  k := N – 1 DOWNTO 0  DO
 BEGIN
      … N - k
 END
```

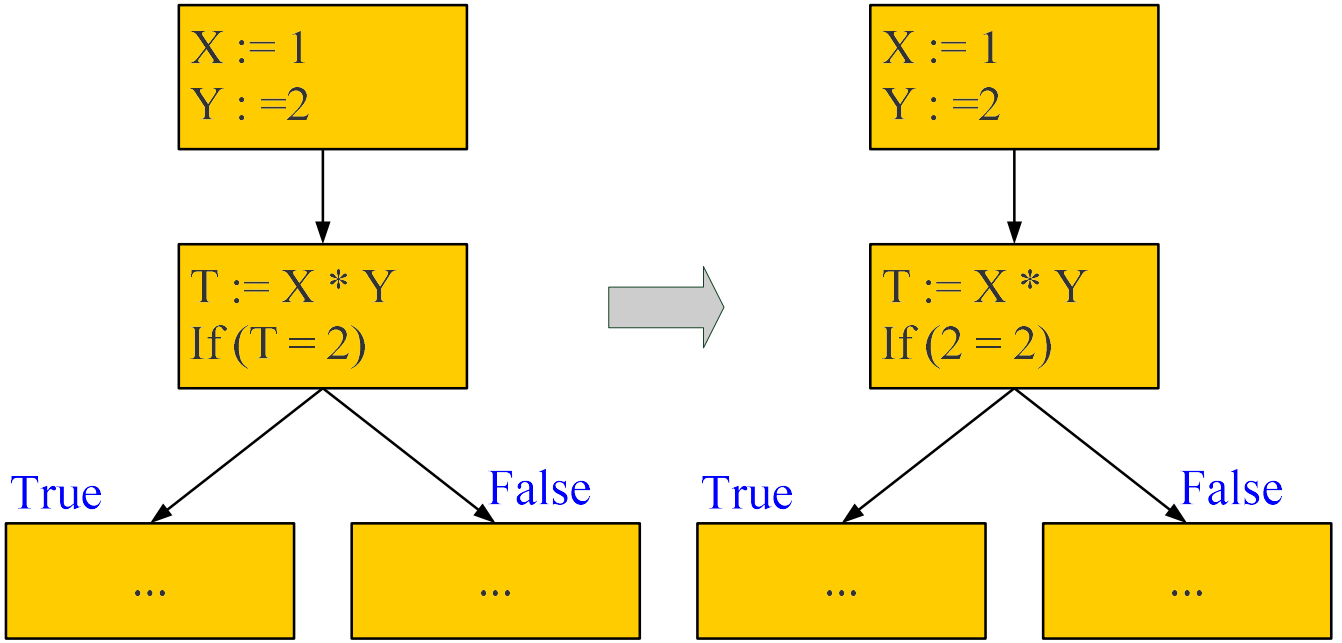
전역 최적화

- 기본 블록 간의 정보와 흐름 그래프를 이용하여 프로그램의 전체적인 흐름 분석을 통한 최적화
- 기법 :
  - 공통 부분식의 제거
  - 전역 상수 폴딩과 전파
  - 도달될 수 없는 코드의 제거
  - 조건문의 재구성
  - 연속 GOTO 축약

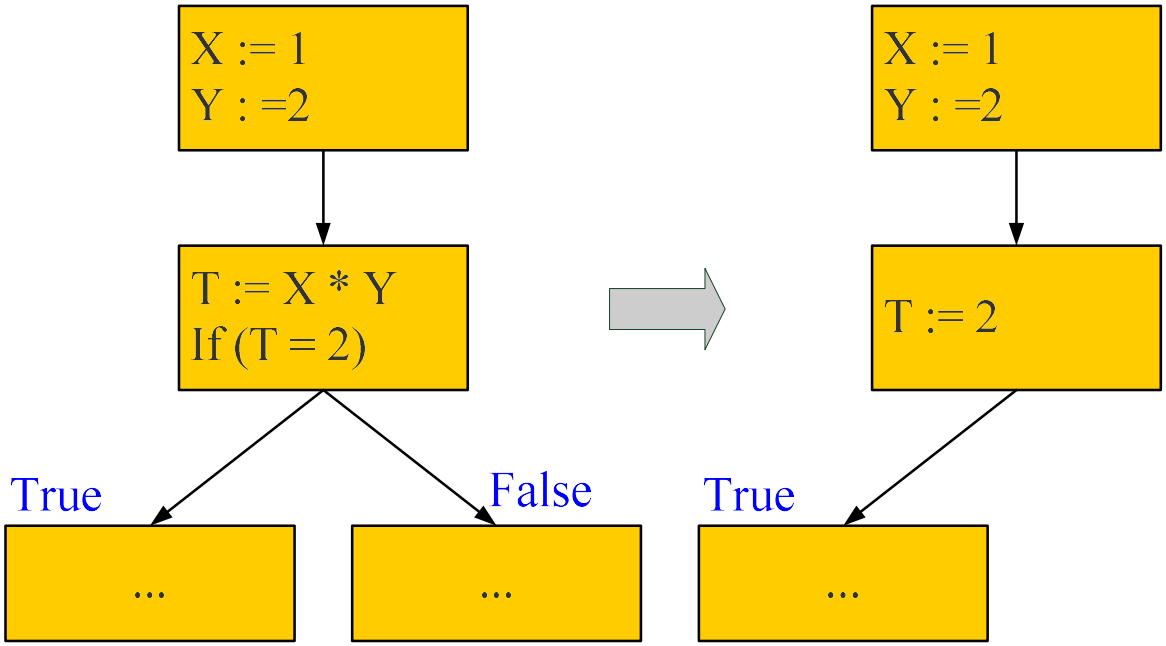
공통 부분식의 제거



전역 상수 폴딩과 전파



도달될 수 없는 코드의 제거

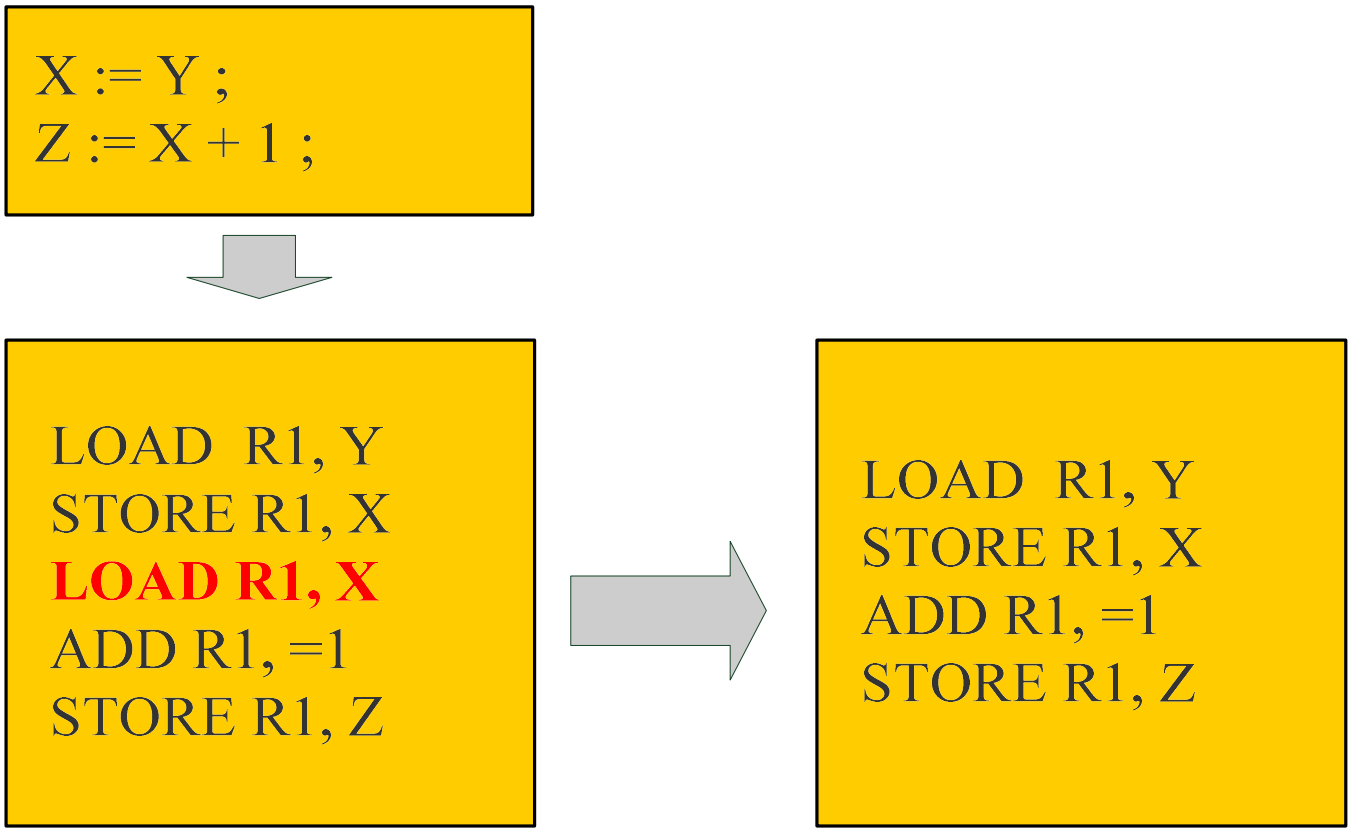


기계 종속적 최적화

- Postcode optimizer

- 기법 :
  - 중복된 load 제거
  - 효율적인 명령어 선택
  - 레지스터 할당과 배정
  - 연산 순서 재조정

중복된 load의 제거


X := Y ;
Z := X + 1 ;
LOAD R1, Y
STORE R1, X
LOAD R1, X
ADD R1, =1
STORE R1, Z
LOAD R1, Y
STORE R1, X
ADD R1, =1
STORE R1, Z