CS3005D Compiler Design

Winter 2024

Lecture #31

Code Optimization, Data-flow Analysis

Saleena N

CSED NIT Calicut

March 2024

Identifying redundant expressions

```
t1=i*4
x=a[t1]
t2=i*4
```

Identifying redundant expressions

```
t1=i*4
x=a[t1]
t2=i+1
i=t2
t3=i*4
```

Available Expressions

Expressions available at each *program point*[1]?

```
t1=i*4
x=t1
t2=i+1
i=t2
t3=i*4
```

[1] for each instruction, a point before the instruction and a point after the instruction

```
    if(i<10) goto L1
    goto L2
L1:t1=i*4
    t2=a[t1]
    x=t2
    goto L3
L2:t3=i*4
    t4=b[t3]
    x=t4
L3:t5=i*4
    c[t5]=x
```

Any redundant computations?

Available expressions at each program point?

```
    if(i<10) goto L1
    goto L2
L1:t1=i*4
    t2=a[t1]
    x=t2
    goto L3
L2:t3=i*4
    t4=b[t3]
    x=t4
L3:t5=i*4
    c[t5]=x
```

Any redundant computations?

Available Expressions

```
    if(i<10) goto L1
    goto L2
L1:t1=i*4
    t2=a[t1]
    x=t2
    goto L3
L2:t3=i*4
    t4=b[t3]
    x=t4
L3:t5=i*4
    c[t5]=x
```

- expression `i*4` is evaluated in both the branches of the conditional.
- `i*4` is *available* at the input point of the statement labelled L3, and hence occurrence of `i*4` in this statement is redundant.

```
    if(i<10) goto L1
    goto L2
L1:t1=i*4
    t2=a[t1]
    x=t2
    goto L3
L2:t3=i*4
    t4=b[t3]
    x=t4
L3:t5=i*4
    c[t5]=x
```

Identifying the redundancy

- requires information regarding the expressions that are computed along the control flow paths reaching `L3`.
- the information is obtained by doing a *data-flow analysis* of the program.

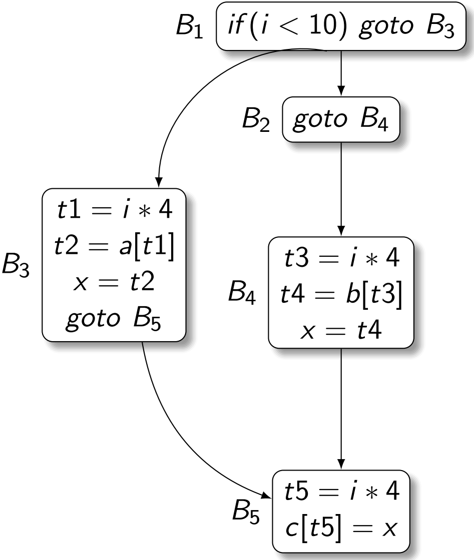
Data-flow Analysis

Derives information regarding the flow of data along program execution paths.

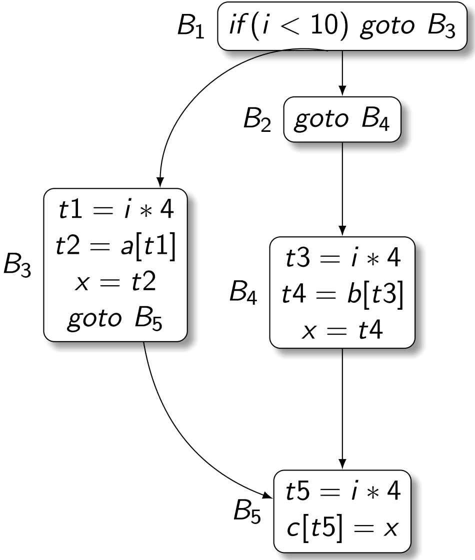
> *Available Expression Analysis* - computes the set of expressions available at every program point. Global CSE requires this information.

Analysis done in a Control Flow Graph representation of the program.

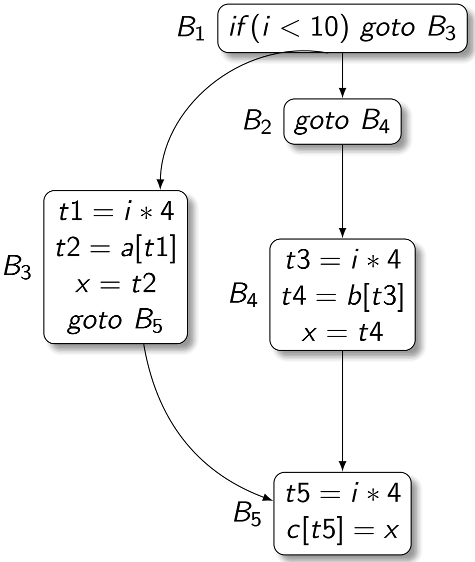
Control Flow Graph



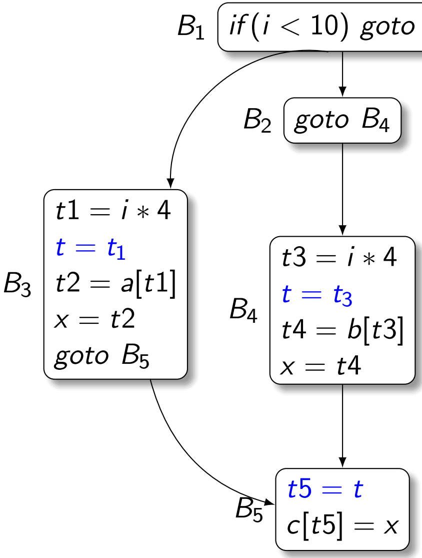
Available Expressions at the entry and exit of each basic block?



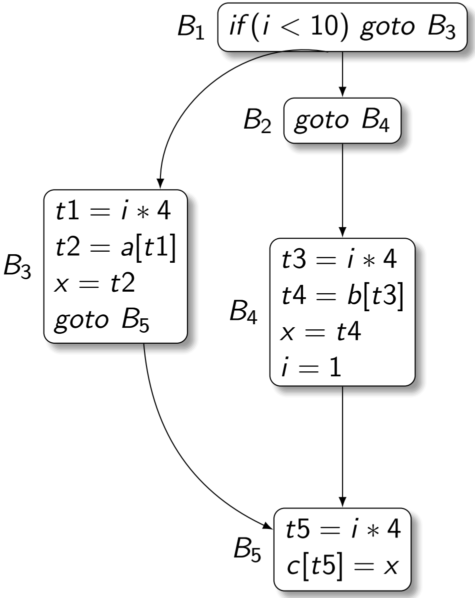
Expression $i * 4$ is available at the entry of B_5. The computation of $i * 4$ in B_5 is redundant.



Global Common Subexpression Elimination: Expression $i * 4$ is available at the entry of B_5. The redundancy in B_5 can be eliminated.



Available Expressions at the entry and exit of each basic block?



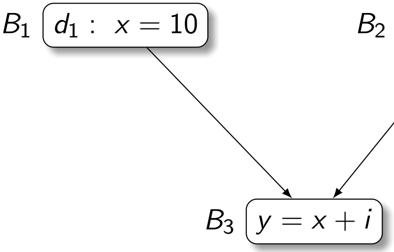
Available Expressions

An expression $x + y$ is *available* at a point p if every path from the entry node to p evaluates $x + y$, and after the last such evaluation, prior to reaching p, there are no assignments to x or y.

Reaching Definitions Analysis

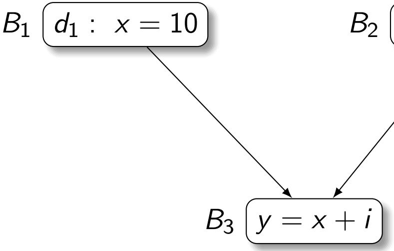
Computes for every program point, the set of definitions of variables that *may reach* that point.

Reaching Definitions



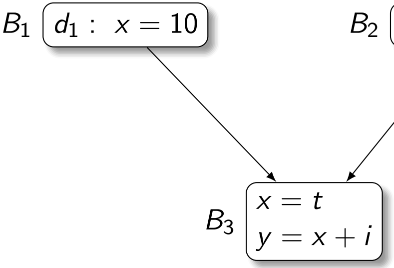
Reaching definitions at the entry point of B_3: $\{d_1, \ d_2\}$

Reaching Definitions



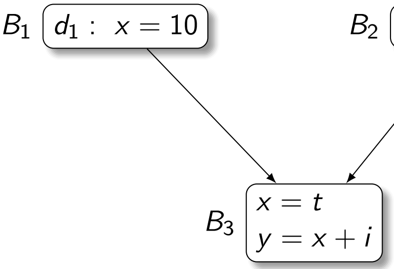
Reaching definitions at the entry point of B_3: $\{d_1, \ d_2\}$.
x in B_3 is constant.
Do Constant Propagation?

Reaching Definitions



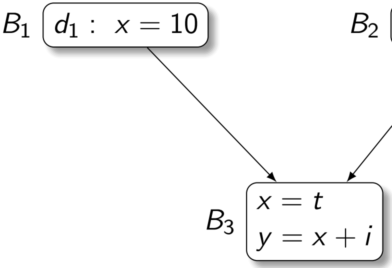
Reaching definitions at the entry point of B_3: $\{d_1, \ d_2\}$.
Do Constant Propagation?

Reaching Definitions: *gen* and *kill*



B_1 *generates* d_1, B_2 *generates* d_2.
$gen(B_1) = \{d_1\}$
$gen(B_2) = \{d_2\}$
$x = t$ *kills* definitions d_1 and d_2.

Reaching Definitions: *IN* and *OUT*



$OUT(B_1) = \{d_1\}$
$OUT(B_2) = \{d_2\}$
$IN(B_3) = OUT(B_1) \bigcup OUT(B_2) = \{d_1, \ d_2\}$.

$x = t$ *kills* definitions d_1 and d_2.
use of x in $y = x + i$ is not a constant.

Reaching Definitions

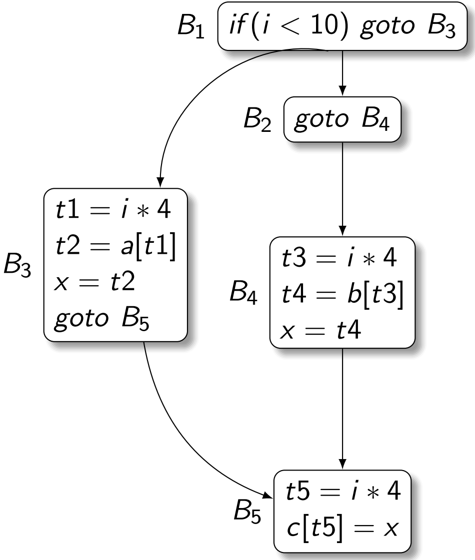
$$OUT(B) = gen_B \bigcup (IN(B) - kill_B)$$

$$IN(B) = \bigcup_{P \text{ a predecessor of } B} OUT[P]$$

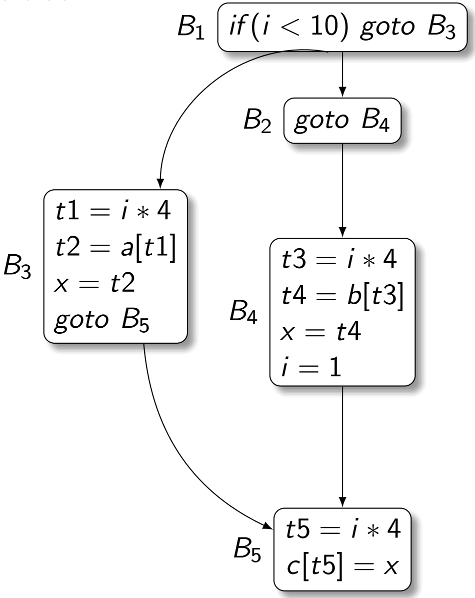
Compute *gen* and *kill* for each basic block.

An iterative algorithm to compute $IN(B)$ and $OUT(B)$ for each basic block B.

Available Expression Analysis: Compute *gen*, *kill*, *IN*, *OUT* for each basic block.



Available Expression Analysis: Compute *gen*, *kill*, *IN*, *OUT* for each basic block.



Live Variable Analysis

A variable x is *live* at a program point p, if the value of x at p could be used along some path in the flow graph starting at p. Otherwise, x is *dead* at p.

Liveness information required for dead-code elimination and Register Allocation.

Topics for self-study

DAG representation of basic blocks and local optimizations - ALSU sections 8.5.1, 8.5.2 and 8.5.3.

References

References:

- Aho A.V., Lam M.S., Sethi R., and Ullman J.D. Compilers: Principles, Techniques, and Tools (ALSU). Pearson Education, 2007.

Further reading:

- ALSU Section 8.4, Chapter 9