Exposing More Instruction-Level Parallelism

- Degree of instruction-level parallelism inside basic blocks is limited – typically to 2.
- The available parallelism in contemporary microprocessors grows. Better exploitation by
  - scheduling sequences of consecutive basic blocks
  - scheduling entire loops
  - speculation
- Two kinds of speculation:
  - dynamic: based on hardware branch prediction. In case of mispredicted branch forgetting or undoing effects of speculatively executed instructions.
  - static: generating compensation code for "speculatively placed" instructions.
  - Recently: Static data and control speculation with hardware support to deal with mispredictions (Intel IA-64).

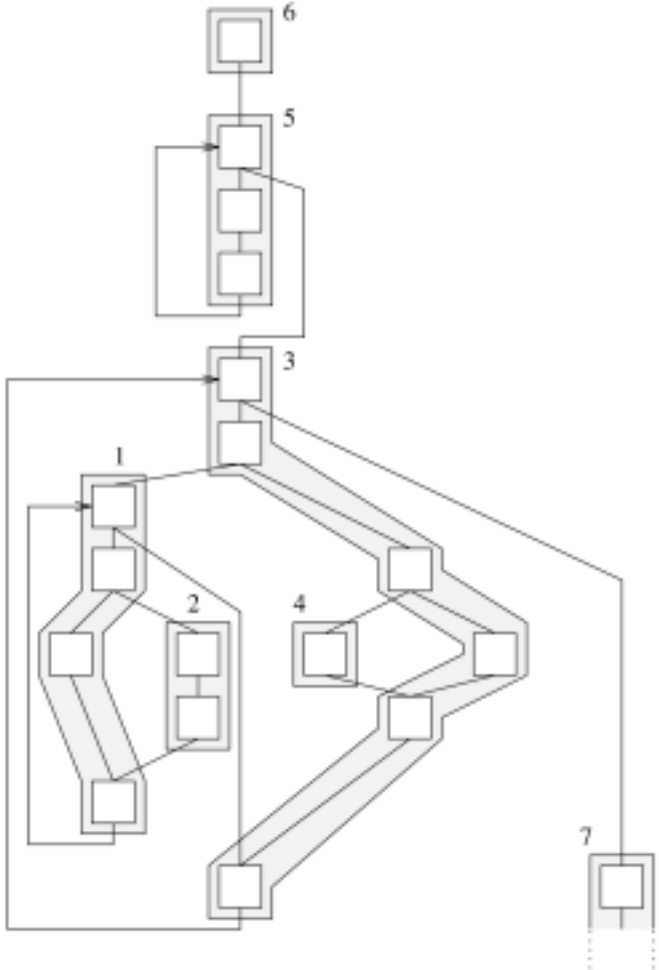
Trace and Superblock Scheduling

- Profile-directed scheduling of sequences of consecutive basic blocks (traces, superblocks)
- Program annotated with profiling information: each branch of a conditional associated with a relative frequency, or each basic block annotated with its execution frequency.
- Profile used to identify frequently taken paths.
- Idea: if a sequence of basic blocks is often executed one after another, they should be optimized jointly.
- Frequently taken paths (traces) are optimized at the cost of less frequently taken traces.

Trace Scheduling

- A trace is a sequence of consecutive basic blocks that are frequently executed one after another. A trace is never extended across loop boundaries.
- The control flow graph of a procedure is partitioned into a disjoint set of traces.
- The traces are formed in the order of decreasing execution frequency by repeatedly
  - selecting the basic block with highest execution frequency that has not been assigned to a trace yet as a seed of the new trace
  - joining predecessors and successors to that trace in decreasing frequency order until the frequency falls below a given threshold, or a loop boundary is reached.

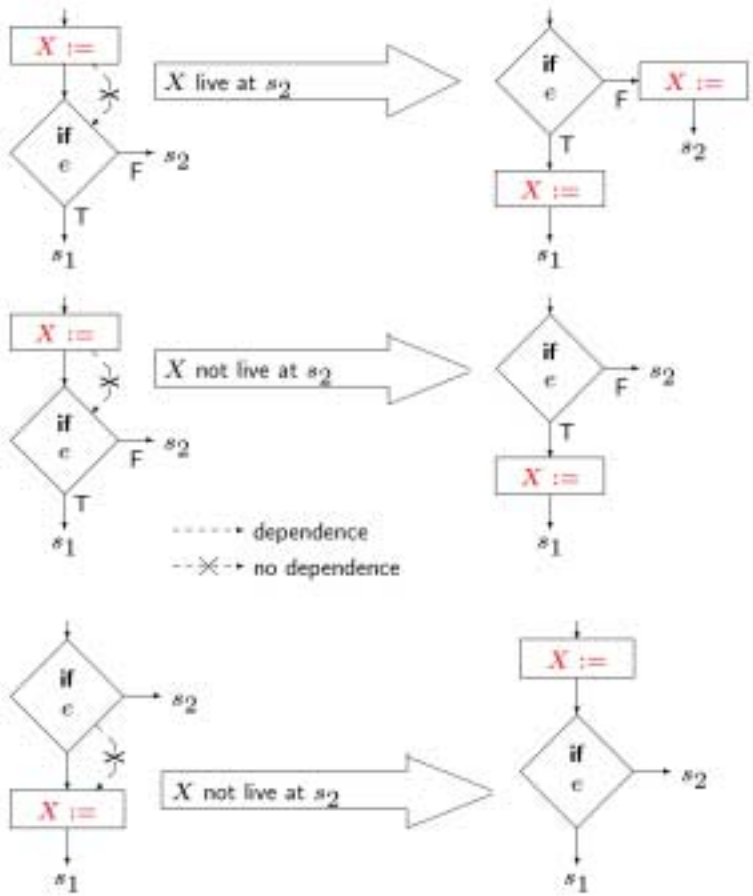
A Partitioning into Traces



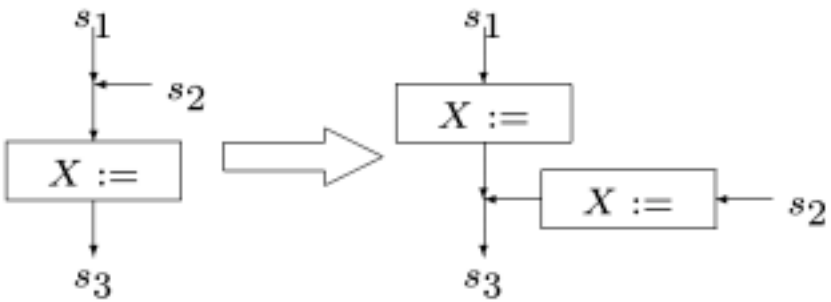
Trace Scheduling

- List scheduling of a trace leads to problems: the semantics may be destroyed by
  - code motion past side exists from the trace
  - code motion past side entrances to the trace
- Consequence: compensation code has to be inserted on off-trace paths.
- Problems of compensation code:
  - code growth
  - exceptions raised by compensation code

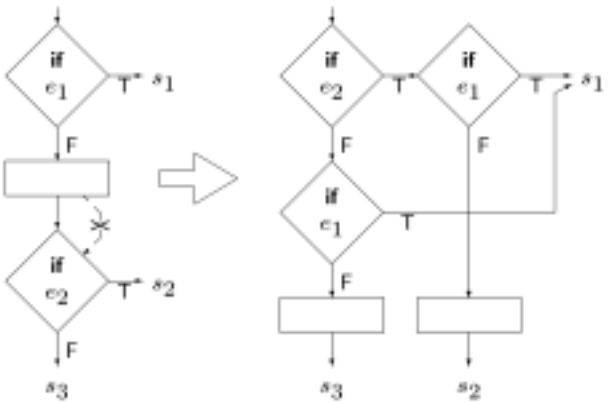
Moving Statements across Conditional Branches



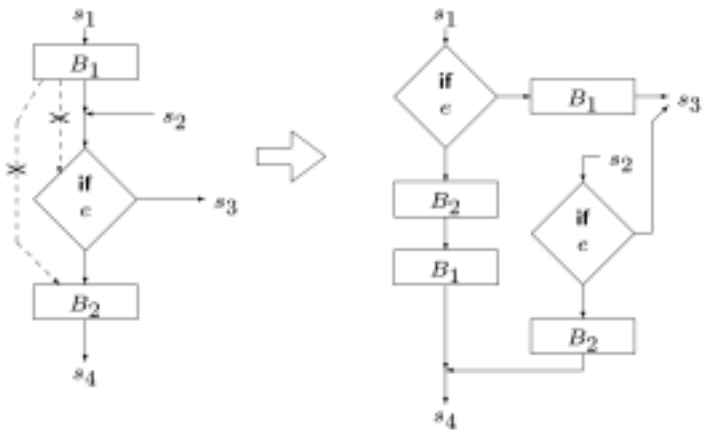
Moving a Statement past a Side



Branches and Joins



Exchanging branches



Moving a branch
up a control flow join

Superblock Scheduling

- Goal: Avoiding code motion past side entrances by tail duplication: copying code starting with side entrance and redirecting the branches.

- Superblock formation:
  - starts with a trace
  - produces a trace without side entrances

- Compensation code only for code motion past side exits.

Standard Algorithms for Code Generation

- Code selection by tree parsing.
- Register allocation by graph coloring.
- Instruction scheduling by
  - list scheduling
  - trace scheduling.
- Optimal algorithms for integrated code selection and register allocation for expression trees and special machine models.

Code Selection

Impact of Code Selection

```
struct {
 unsigned b1:1;
 unsigned b2:1;
 unsigned b3:1;
} s;
s.b3 = s.b1 &&
s.b2;
```

```
ld.bu  d5, [sp]48
extr.u d5, d5, #0, #1
ne d5, d5, #0
mov16  d4, d5
jeq    d4, #0, _9
ld.bu  d4, [sp]48
extr.u d4, d4, #1, #1
ne     d4, d4, #0
_9:
ld.b   d3, [sp]48
insert d3, d3, d4, #2, #1
st.b   [sp]48, d3
```

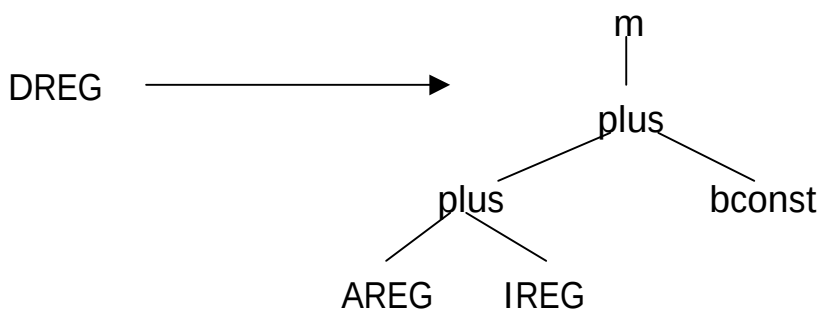
9 cycles, 42 bytes

```
ld.bu  d5, [sp]48
and.t  d3, d5, #0, d5, #1
insert d5, d5, d3, #2, #1
st.b   [sp]48, d5
```

3 cycles, 16 bytes

Generating Code Selectors

- Machine grammar: regular tree grammar;
  - terminals: operators from the program representation
  - non-terminals: represent storage resources
  - often ambiguous
  - each rule has associated costs
  - factorization, e.g. of addressing modes reduces size.



Generating Code Selectors

- A machine grammar enables IR trees for expressions to be derived. The derivation tree for an IR tree represents one possibility of generating code for the IR tree.

- The generated code selector
  - parses intermediate representations of programs
  - computes derivations according to the machine grammar, each corresponding to a sequence of machine instructions
  - has to select a cheapest derivation, corresponding to the (locally) cheapest code sequence
  - may compute costs in states or use dynamic programming