18-747 Lecture 18: Advanced ILP Scheduling

James C. Hoe
Dept of ECE, CMU
November 5, 2001

Reading Assignments:

Announcements: Project 2 (and HW3) due on coming Friday (coming Monday)
Quiz 2 on December 3rd

Handouts:

Trace Scheduling *[Josh Fisher]*

- Generate multi-basic block traces based on profiling information
  - find the most often executed control path
- List schedule a trace at a time
  - optimize the execution of the trace *(common case)*
  - fix any problem with off-trace paths as necessary *(infrequently executed)*
- Good for very biased and predictable branching behavior
- Trace scheduling engendered the VLIW architecture innovation and was implemented in the Multiflow TRACE compiler, which provided the basis for superscalar compilation techniques now being used by Intel, HP, and DEC

Trace Scheduling Overview

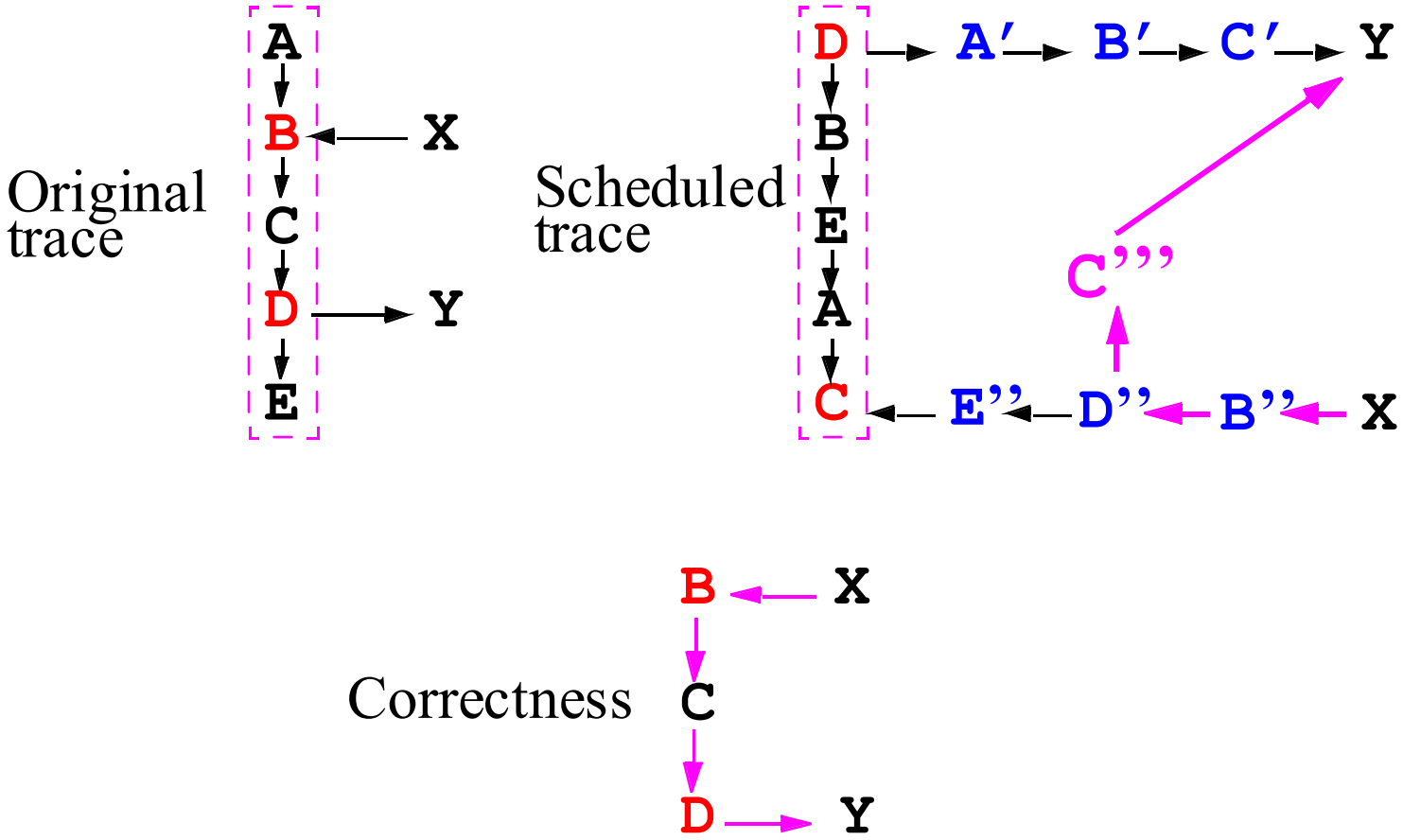
- Trace Selection
  - select seed (the highest frequency basic block)
  - extend trace (along the highest frequency edges)
    forward (successor of the last block of the trace)
    backward (predecessor of the first block of the trace)
  - don't cross loop back edge
  - bound max_trace_length heuristically
- Trace Scheduling
  - build data precedence graph for a whole trace
  - perform list scheduling and allocate registers
  - add compensation code to maintain semantic correctness
- Speculative Code Motion (upward)
  - Move an instruction above branches if safe

Compensation Code for Downward Motion

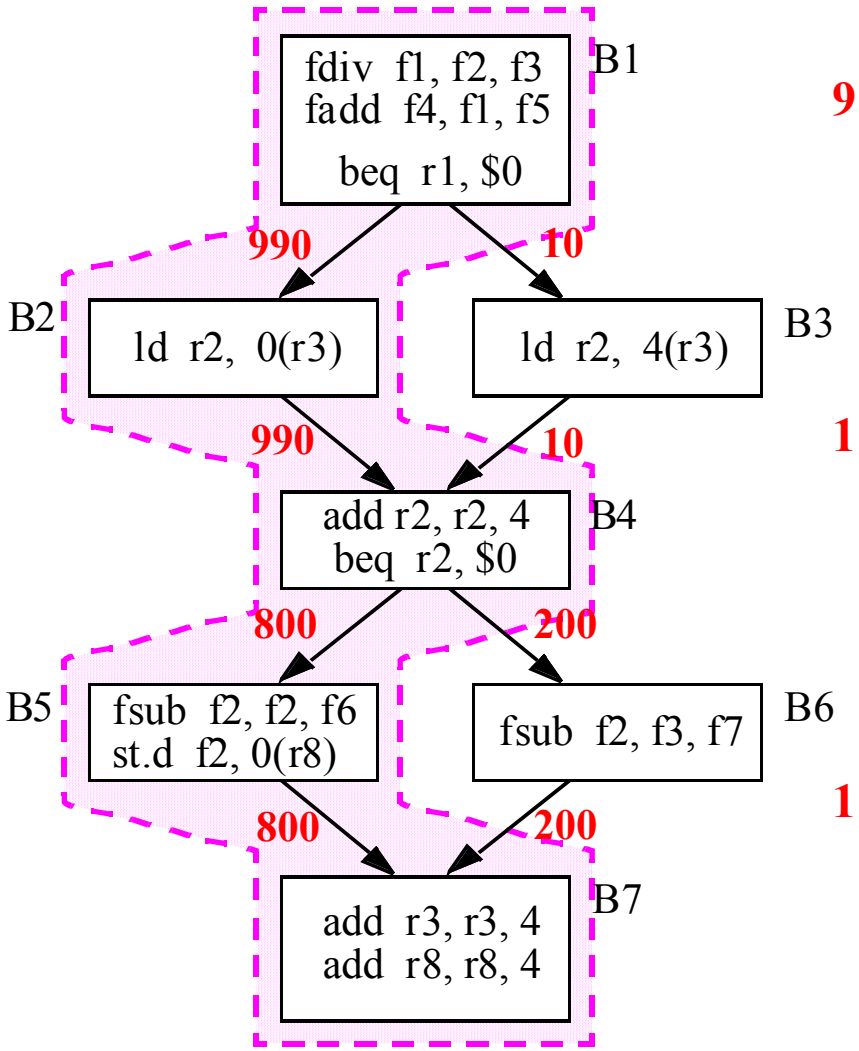
- Split Compensation Code:
  - Instruction with more than one successor (conditional branch)

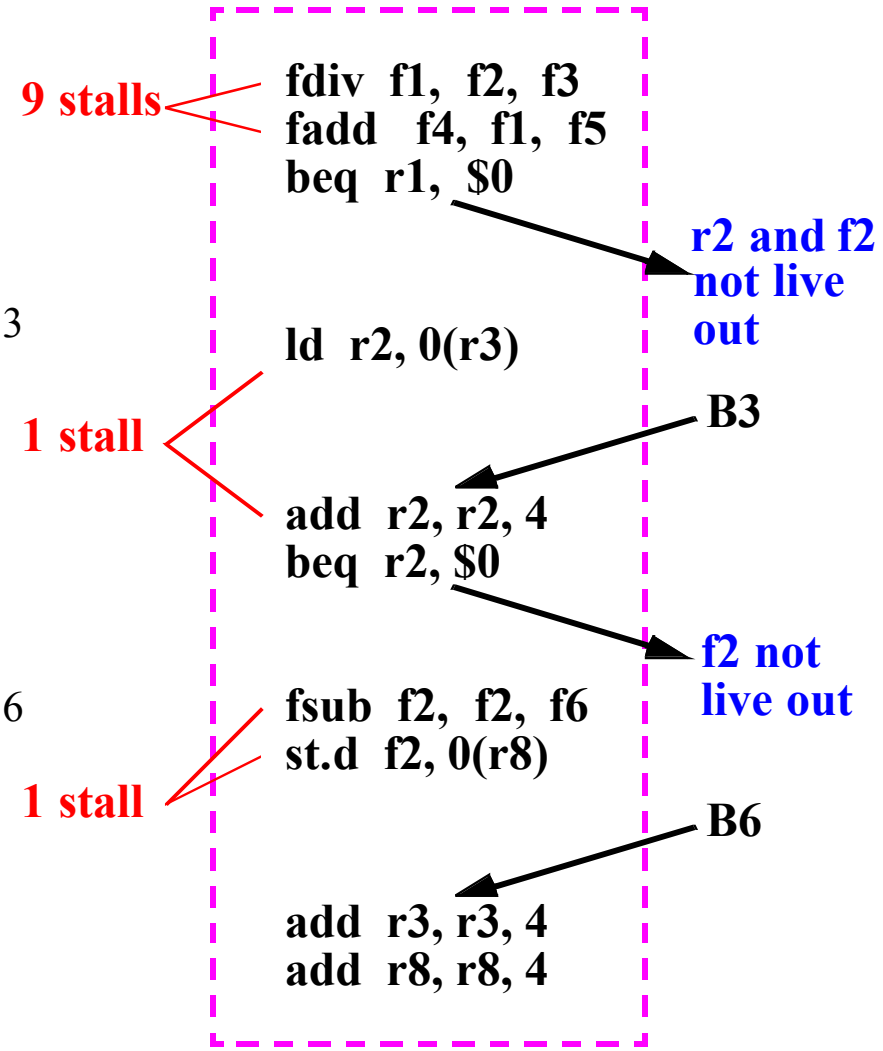
- Join Compensation Code:
  - Instruction with more than one predecessor

Copied Split Instruction

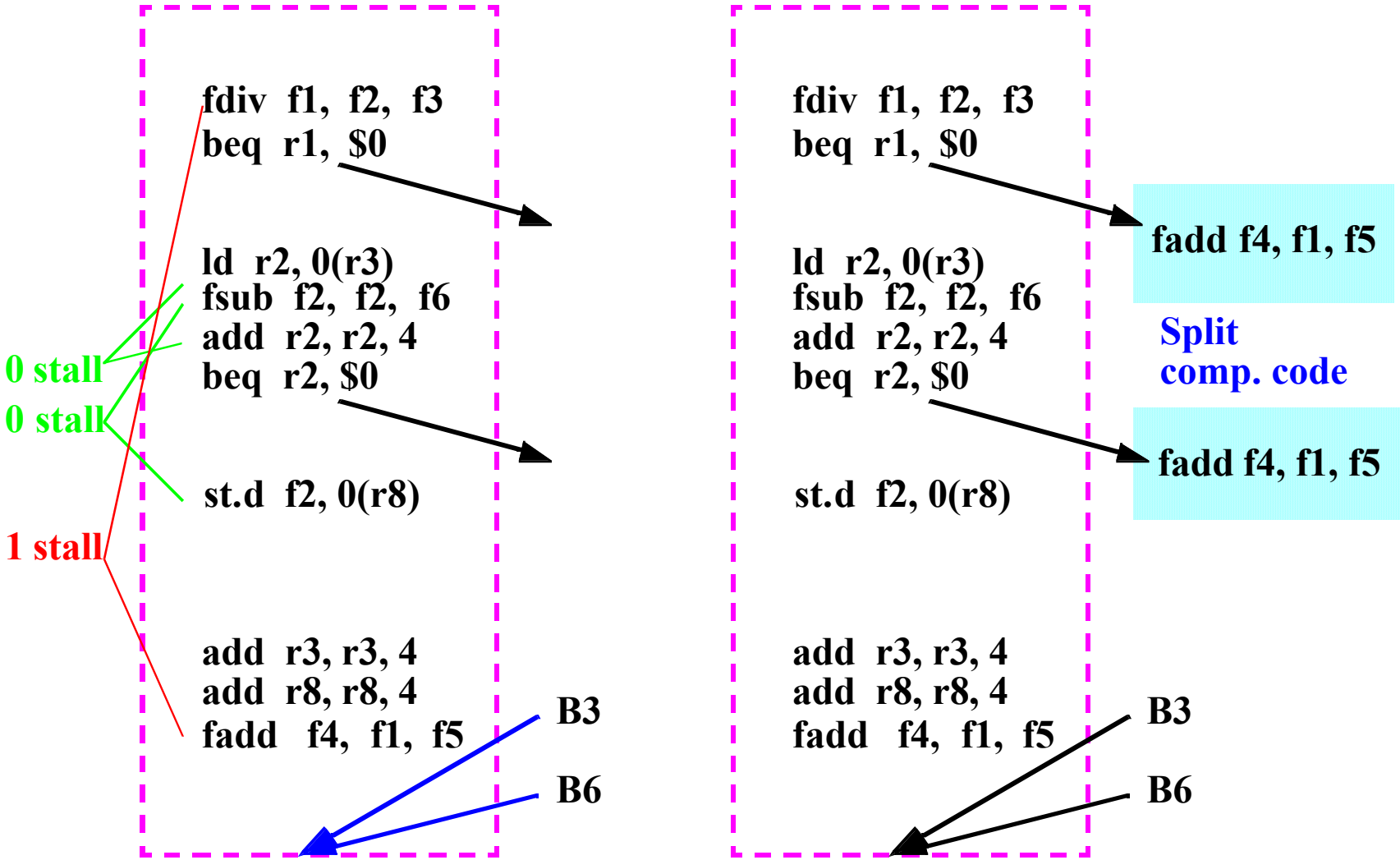


Trace Scheduling Example

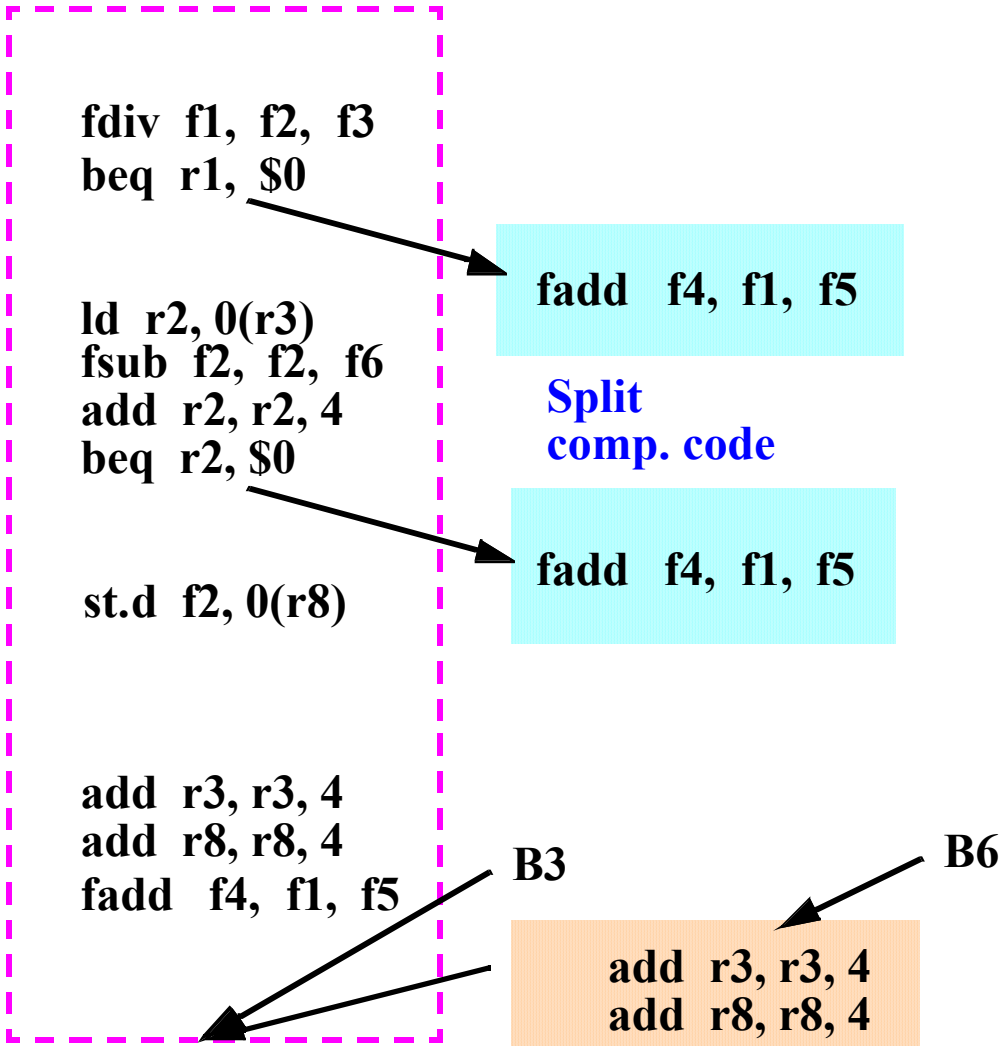




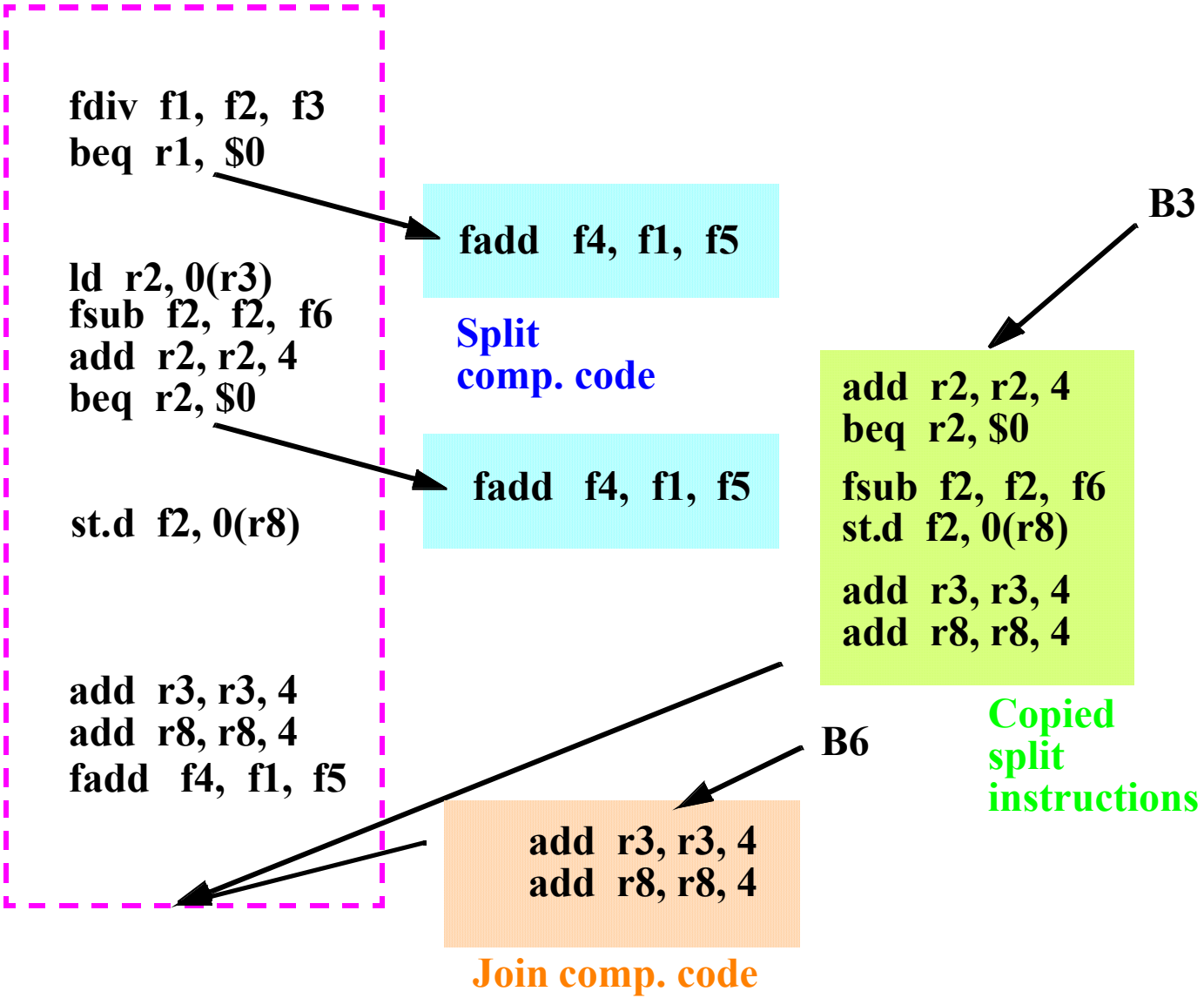
Compensation Code Example



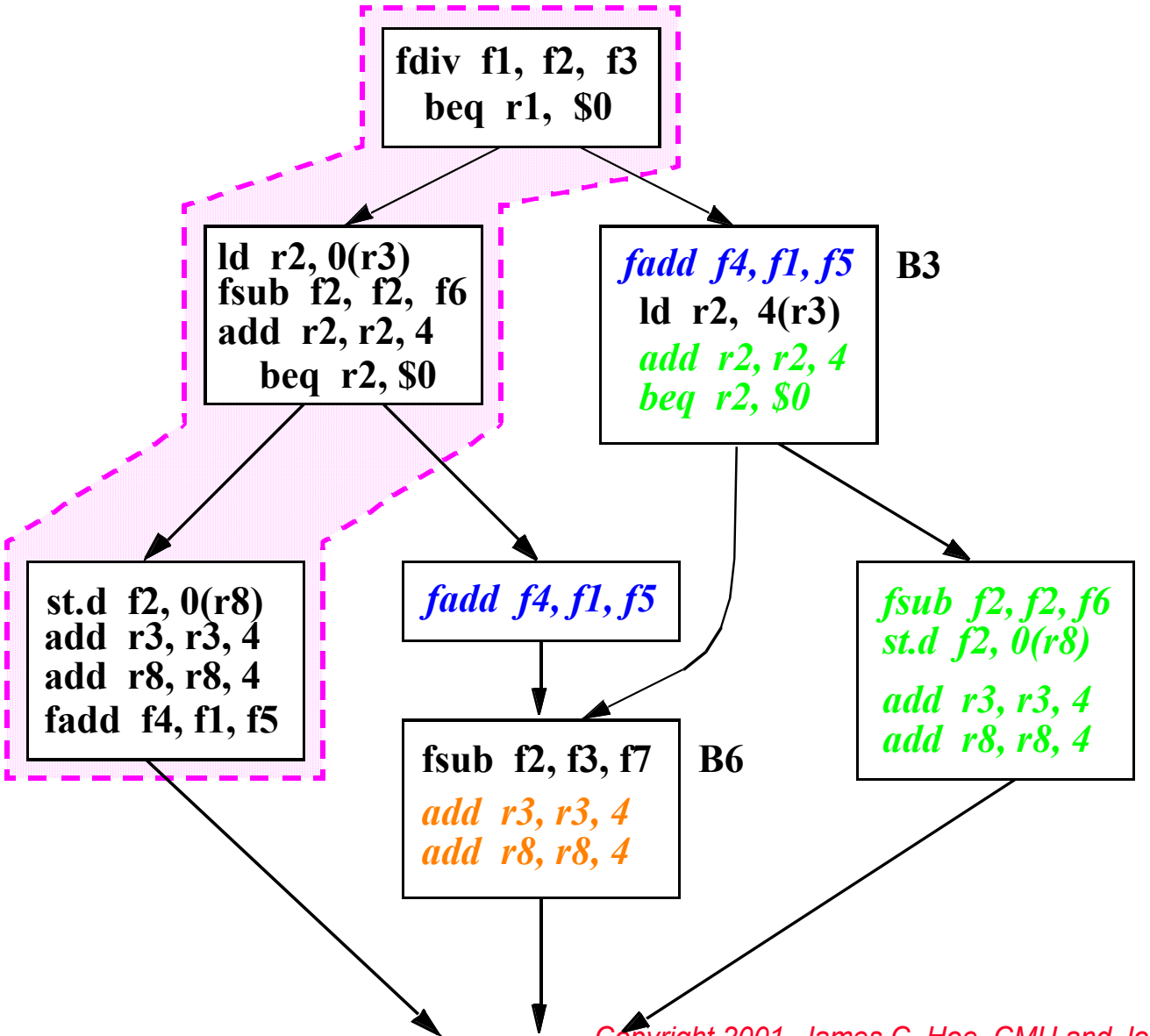
Compensation Code Example



Compensation Code Example

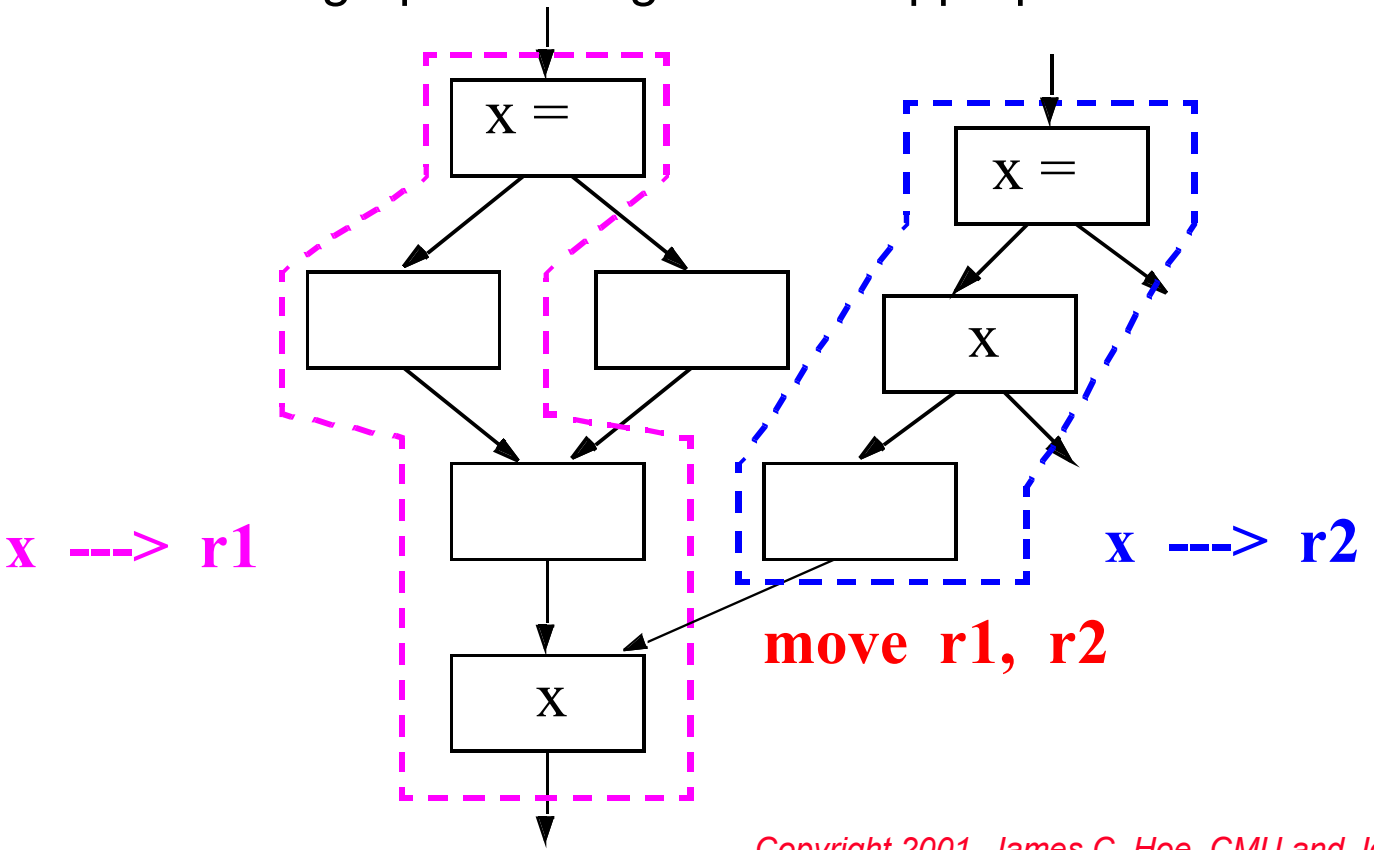


Compensation Code Illustration



Register Binding

- Perform register allocation for a trace
  - After scheduling a trace, do register allocation
  - \+ Most frequently executed traces have maximum freedom of register usage
  - \- Do not use graph coloring due to inappropriate framework

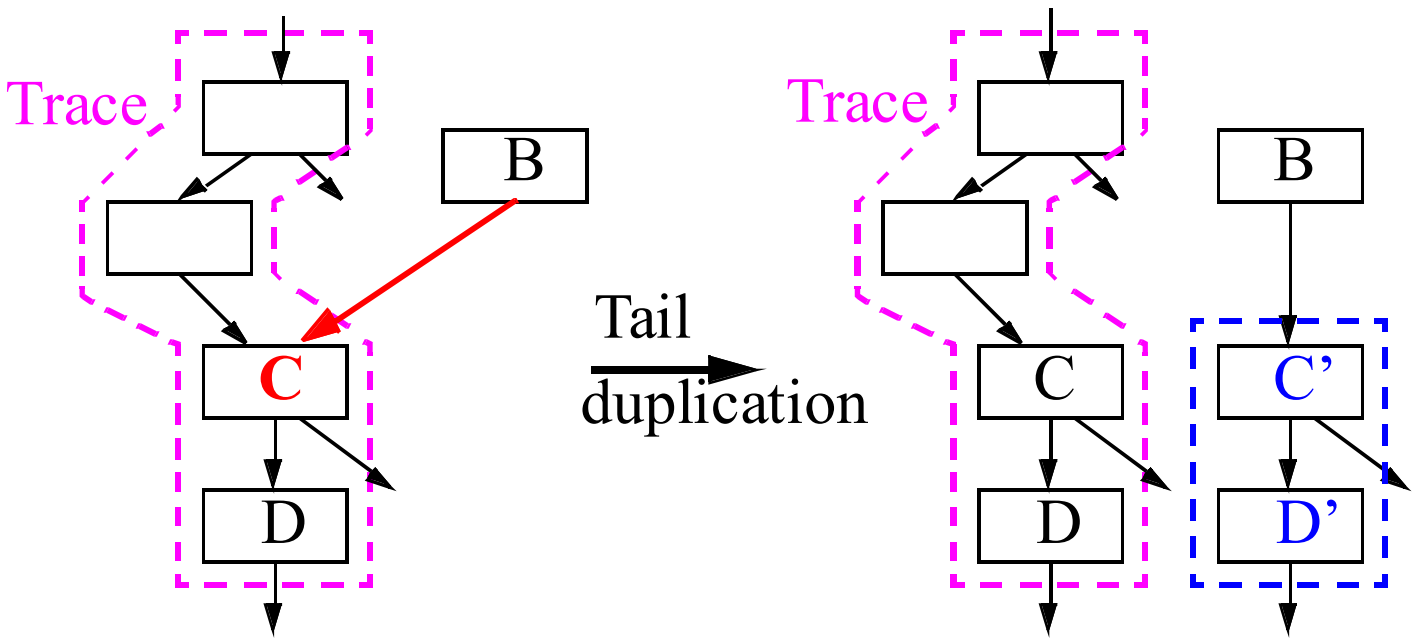


Superblock Scheduling

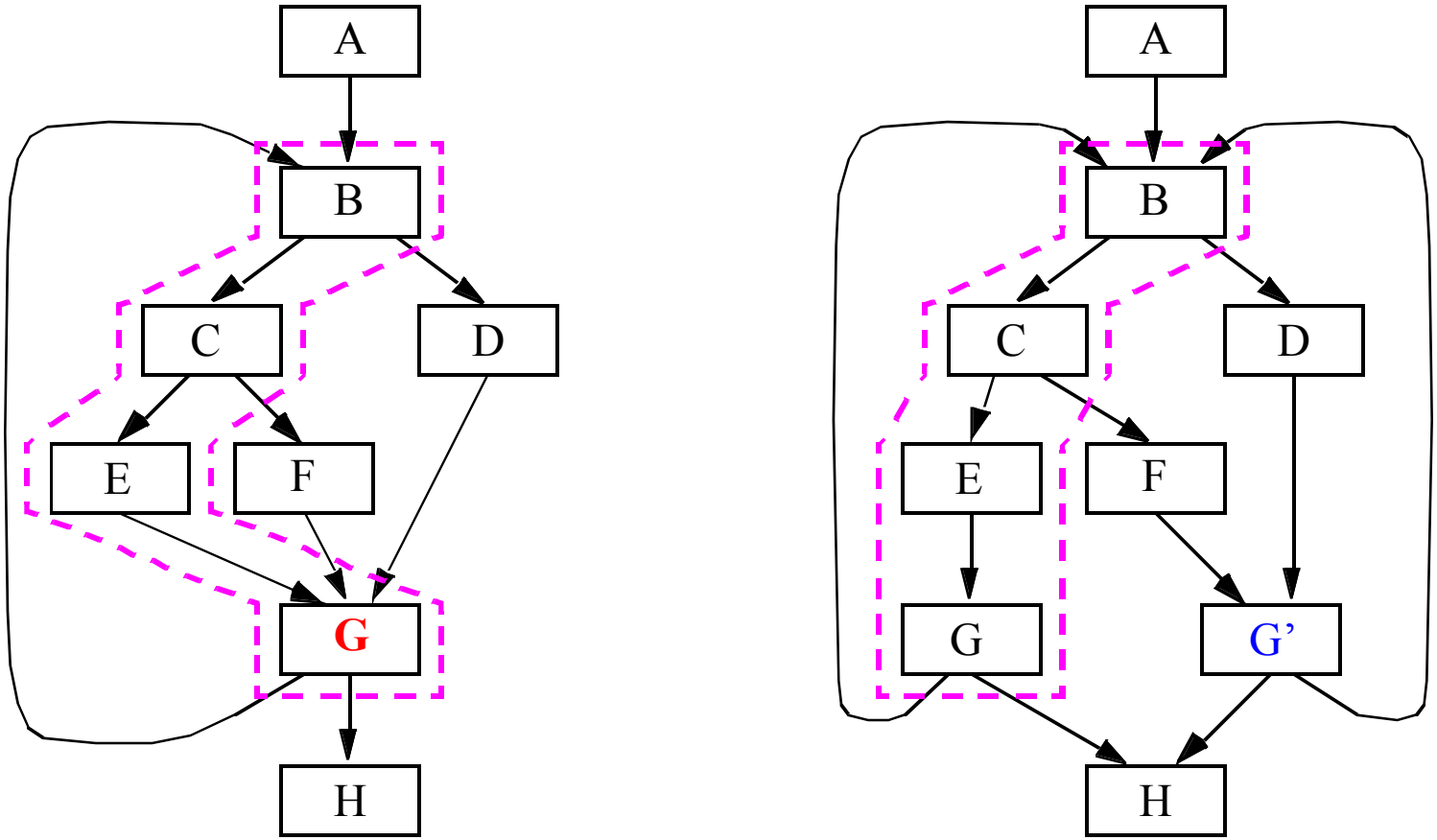
- Motivation
  - Trace scheduling is a good idea
  - Maintaining semantic correctness (compensation code) is a pain
- Superblock
  - Trace with one entry point (multiple entries create control flow joins)
  - May have multiple exits

Superblock Formation Example

- Identify traces using profiling information
- Use tail duplication to eliminate side entry points

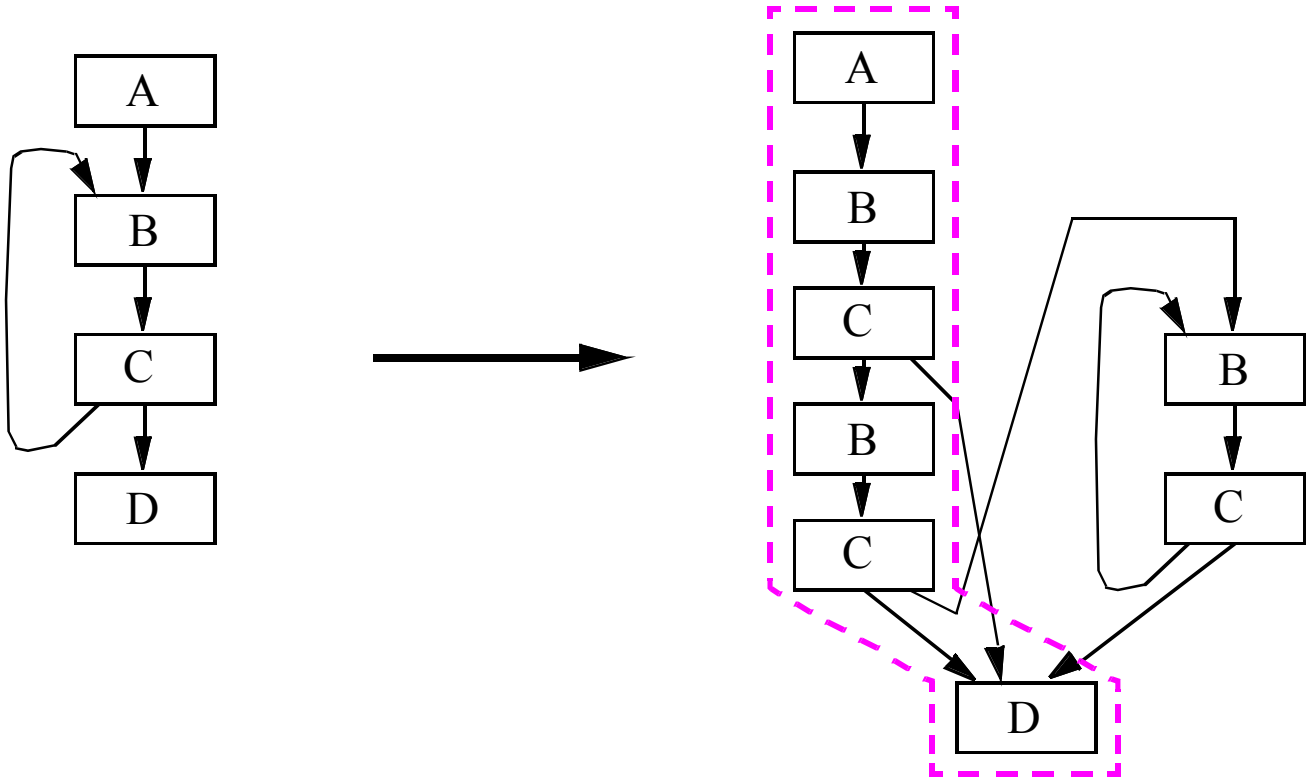


Superblock Formation Example



Superblock Enlarging

- Branch Target Expansion
  - Expand along likely-taken path
- Loop Unrolling & Loop Peeling



ILP Optimization

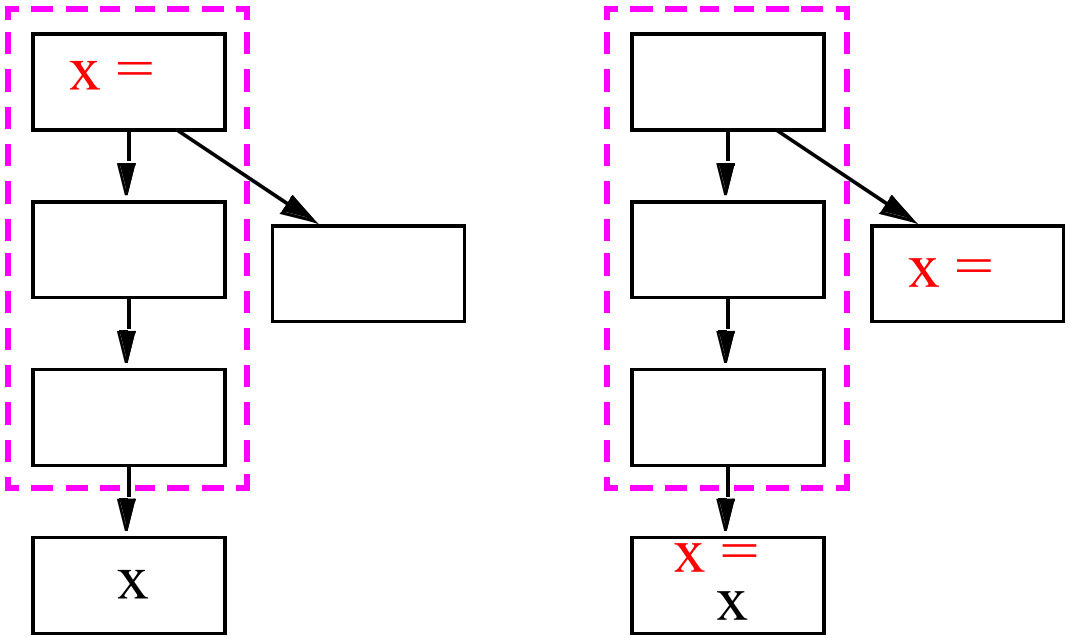
- Basic Block Size

	Average Block Size:
Basic block:	3 instructions
Superblock-original:	4 instructions
Superblock-formation:	10 instructions
Superblock-enlargement:	13 instructions

- Dependence Elimination
  - Code transformations to eliminate data dependencies
  - Give code scheduler more freedom to move instructions

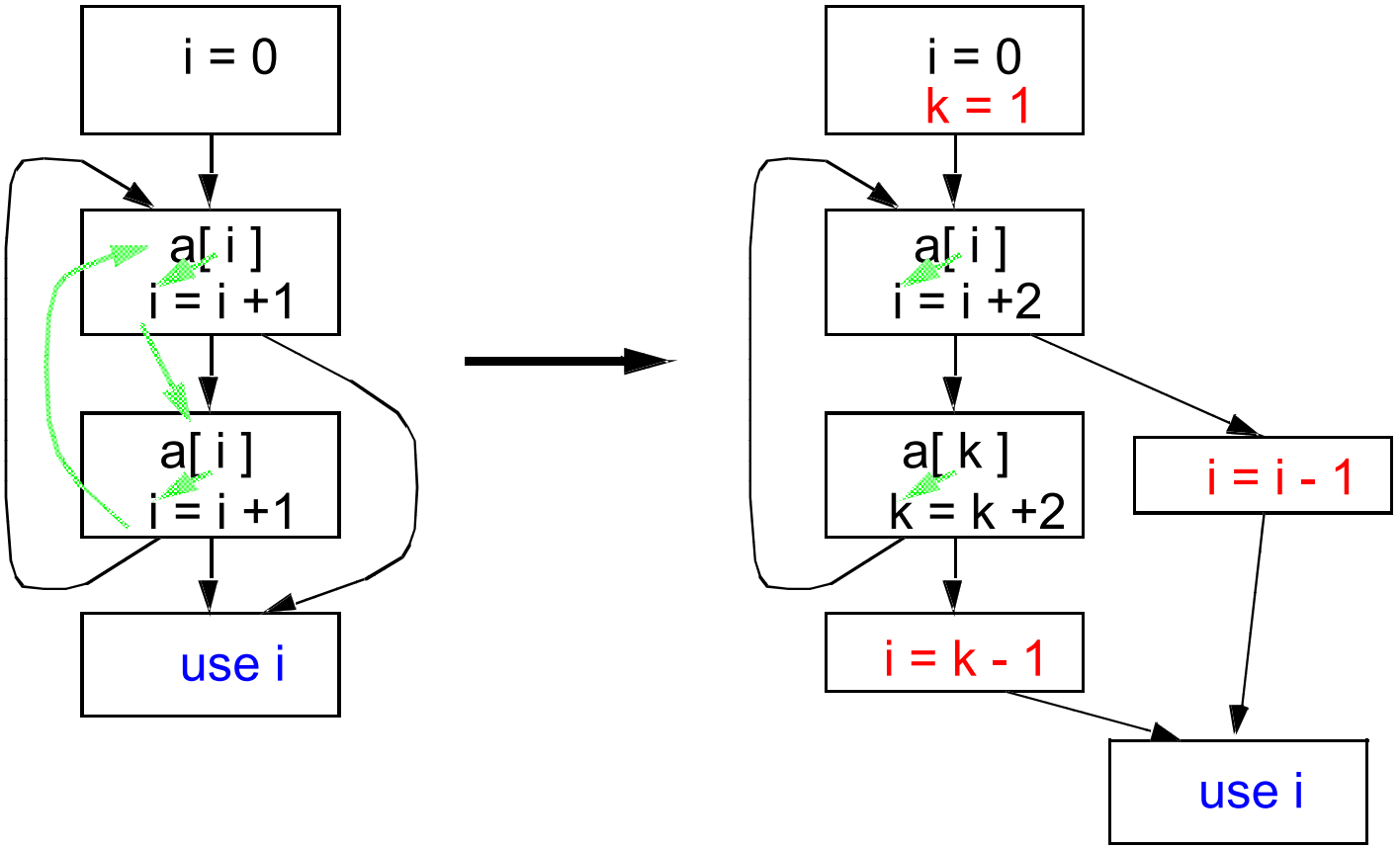
Operand Migration

- Move instructions whose results are not used within trace to less frequently executed paths



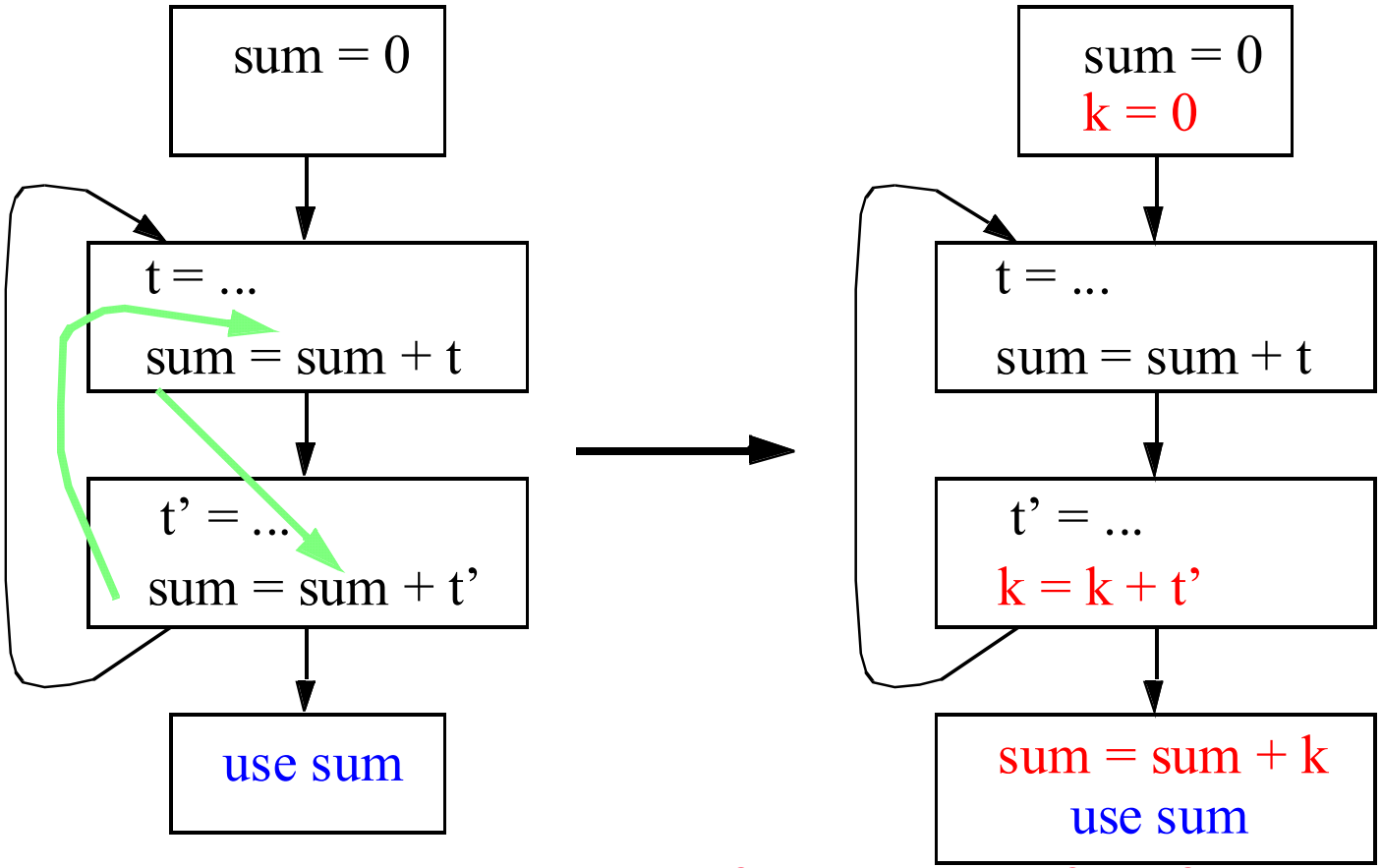
Induction Variable Expansion

- Eliminate redefinitions of induction variables within unrolled loops
- Insert code to maintain semantic correctness



Accumulator Variable Expansion

- Accumulate a sum or product in each iteration
- Insert code to maintain semantic correctness
- May not be safe for floating point



Symbolic Memory Disambiguation

Simple Example:

```
c = a + b;   ⟶   load r1, r29+_a
                 load r2, r29+_b
                 add  r3, r1, r2
                 store r0+_c, r3
```

- It is easy to determine at compile time that **a** and **b** have different addresses.
- It is also easy to determine if they are in separate memory banks.

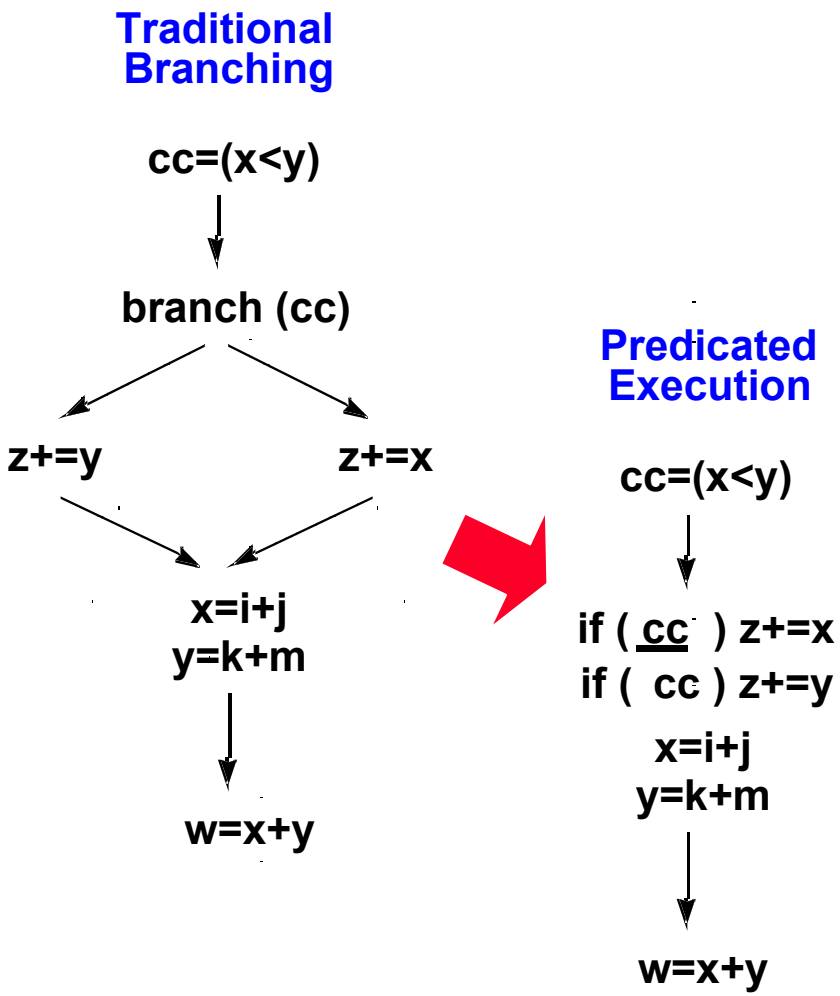
- More complex situations arise when arrays or pointers are used:
  - Is a[j] independent of a[j+k]?
  - Is a[j] independent of a[j+1]?
  - Is a[j] independent of b[j]?
  - Is a[j] independent of *msg?
- Symbolic analysis of the source code (or the dataflow graphs) answers these questions
- The scheduler then embeds this information within the program

Predicated Execution

- Predicated Execution removes branches by *conditionally* executing operations
- Removing branches combines multiple basic blocks into larger basic blocks
- Branch related stalls are eliminated
- Additional opportunities for scheduling optimizations appear

Example:

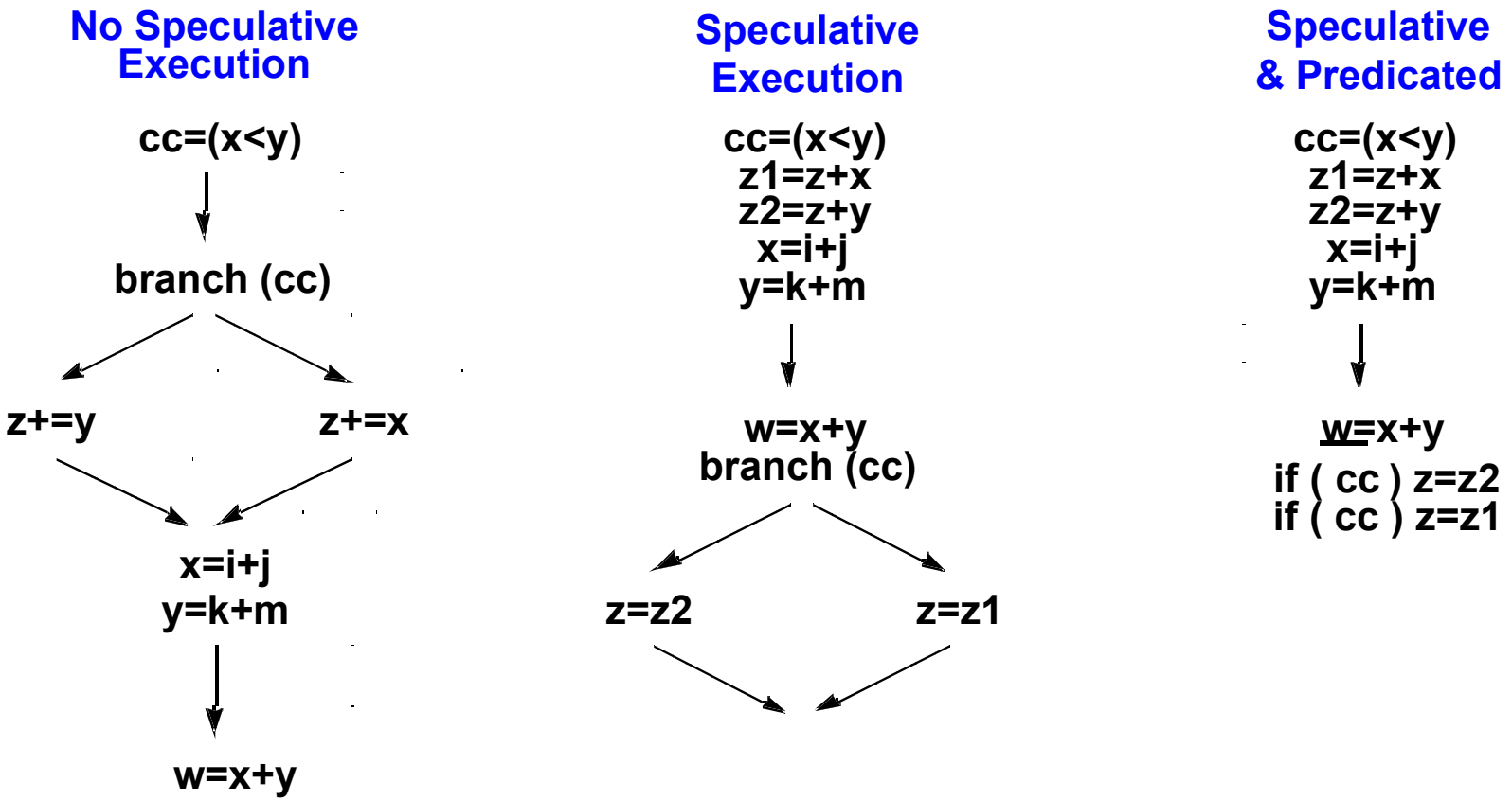
```
if (x<y) then
   z+=x;
else
   z+=y;
x=i+j;
y=k+m;
w=x+y;
```



Execution time is reduced from 5 cycles to 3 cycles.

Speculative Execution

- Speculative Execution involves computing results before it is known if they will be used in the program.



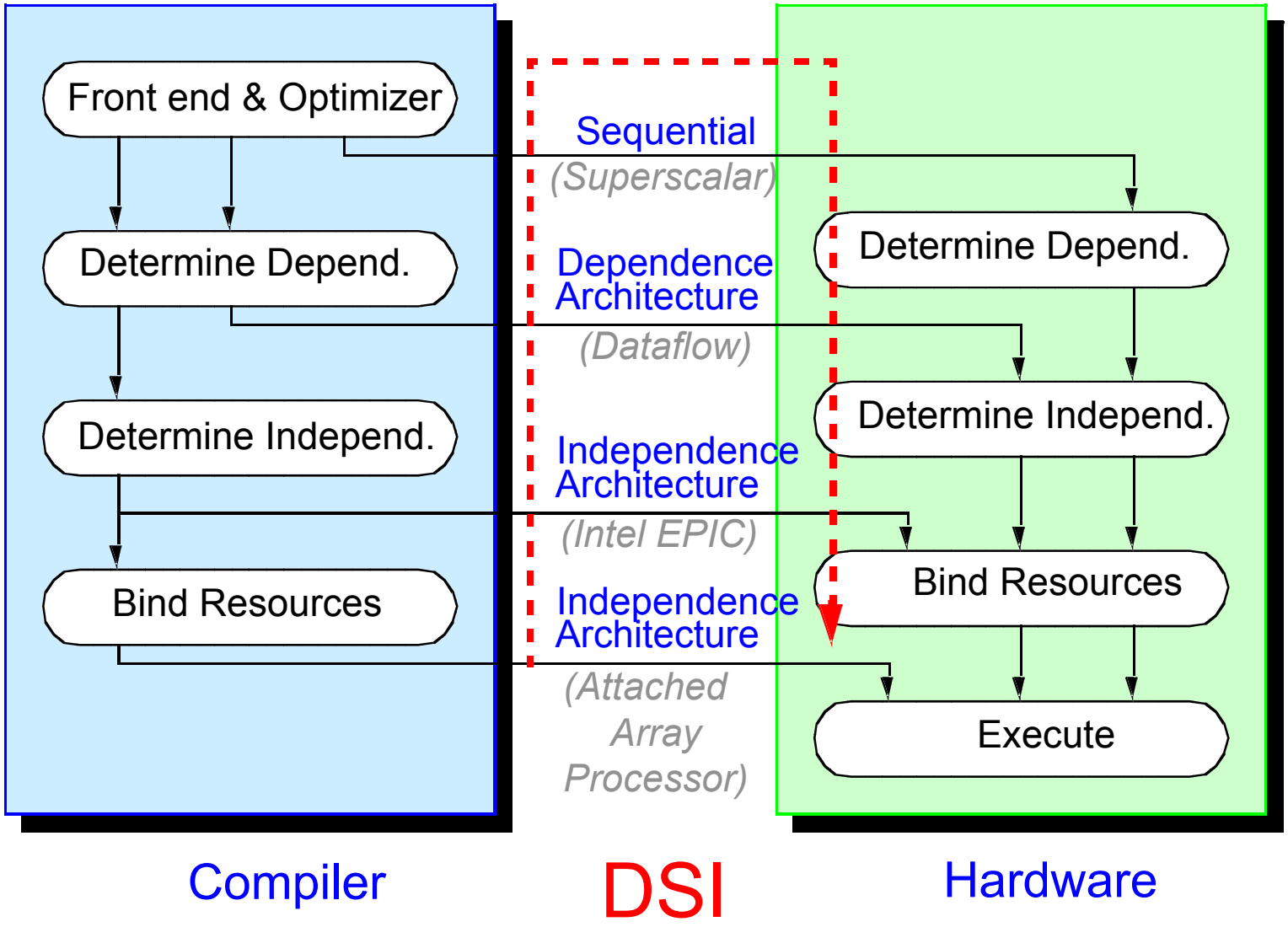
Superblock List Scheduling

- Restricted percolation
  - No architecture support
  - Instructions that could cause exceptions are not moved beyond branches
  - Memory load/store, integer divide and floating point instructions

- General percolation
  - Architecture support (non-trapping instructions)
  - Write garbage value when exceptions occur for non-trapping instructions

We will see this when we discuss Intel EPIC

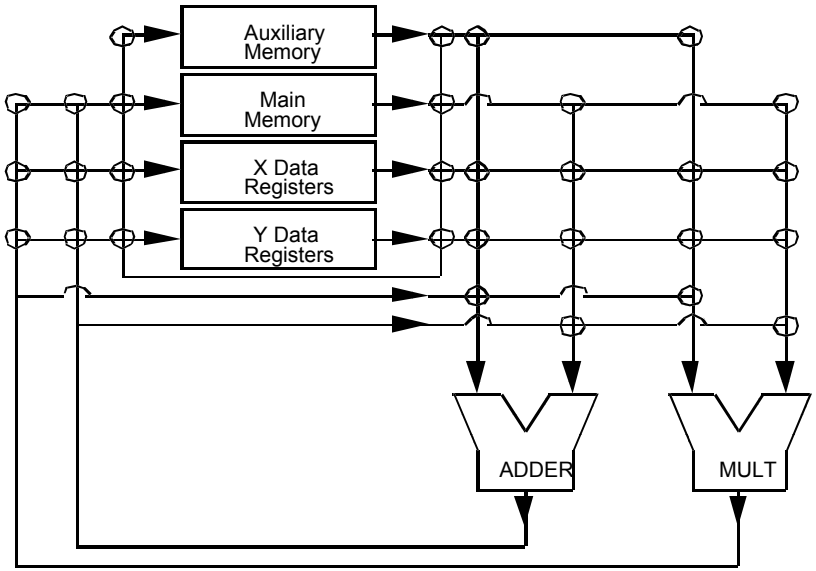
Compiler/Hardware Interactions



[B. Rau & J. Fisher, 1993]

Attached Array Processors

- The FPS-120B (1975) and FPS-164 (1980) were early user programmable microcoded engines 64-bit instruction word
  - contained ten fields called parcels - each specified one operation.
  - six functional units executed the instructions
  - Data path topology is optimized for vector (or array) dot products, FFT, and convolution

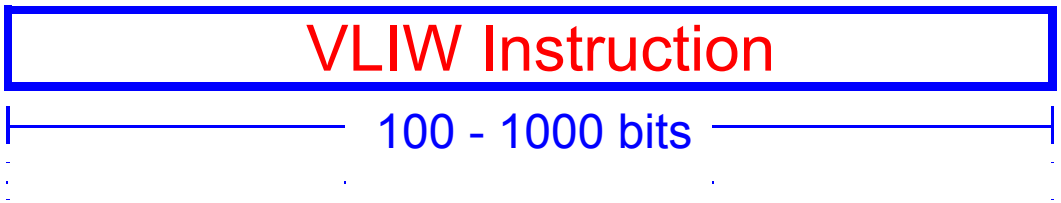


- A FORTRAN compiler could schedule vectorizable loops with software pipelining.
- Most users relied on hand-coded library routines (supplied by the manufacturer)

Peak performance was 12MFLOPS.

Principles of VLIW Operation

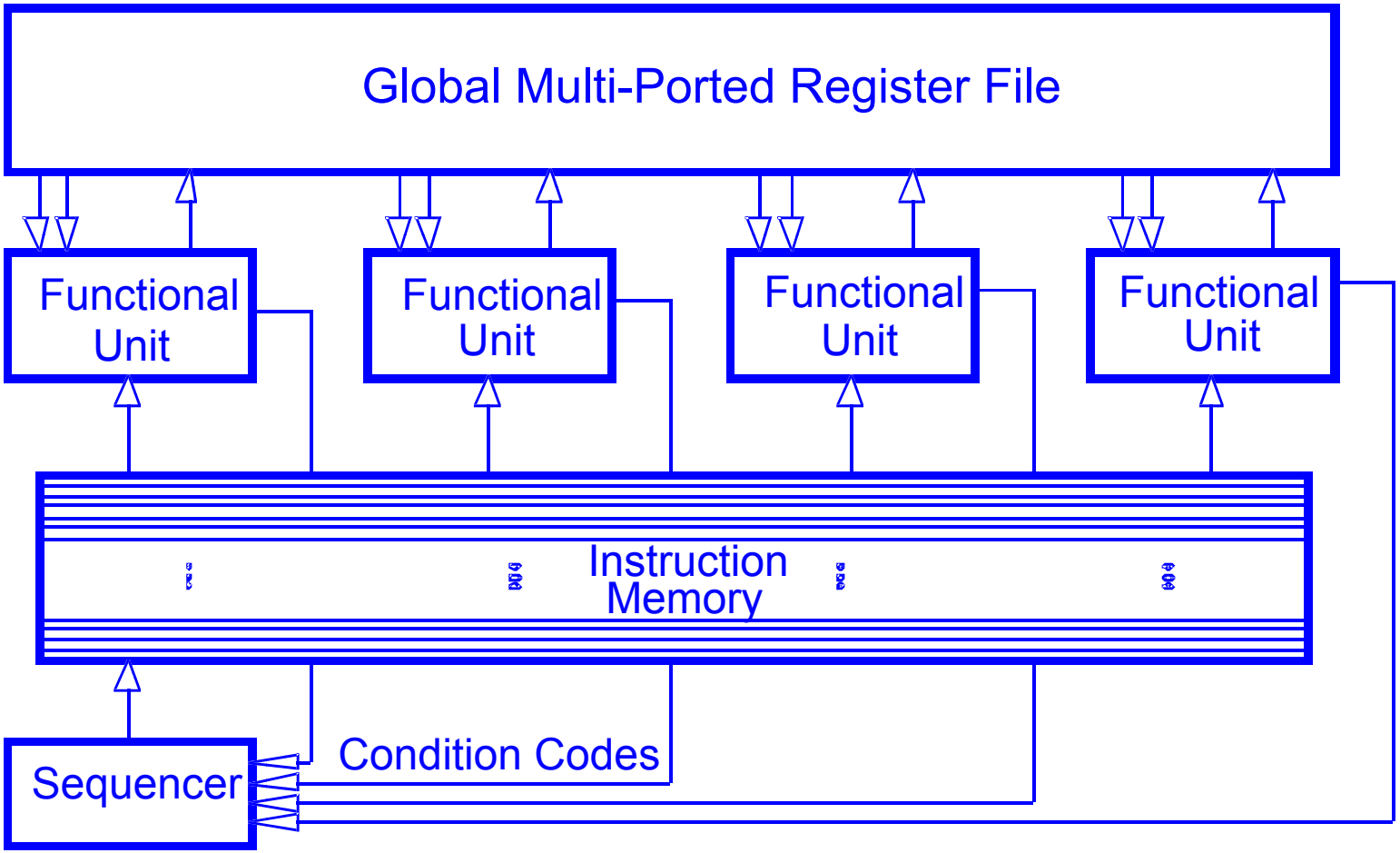
- Statically scheduled ILP architecture.
- Wide instructions specify many independent simple operations.



- Multiple functional units executes all of the operations in an instruction concurrently, providing fine-grain parallelism within each instruction
- Instructions directly control the hardware with no interpretation and minimal decoding.
- A powerful optimizing compiler is responsible for locating and extracting ILP from the program and for scheduling operations to exploit the available parallel resources

The processor does not make any run-time control decisions below the program level

VLIW Execution Characteristics



Basic VLIW architectures are a generalized form of horizontally microprogrammed machines