
Tree-based VLIW Architecture

Jaime H. Moreno

IBM Research

“Conventional” VLIW approach

Object-code incompatibility

- Among VLIW and sequential (scalar, superscalar) implementations
- Among VLIW implementations with different width

Implementation-dependence

- Operations/branches per VLIW exposed in architecture

Architecture and implementation are highly related

Tree-based VLIW architecture

Recover traditional separation between architecture and implementation

- allow for different implementations of same architecture
- includes scalar, superscalar, and VLIW

Basic aspects

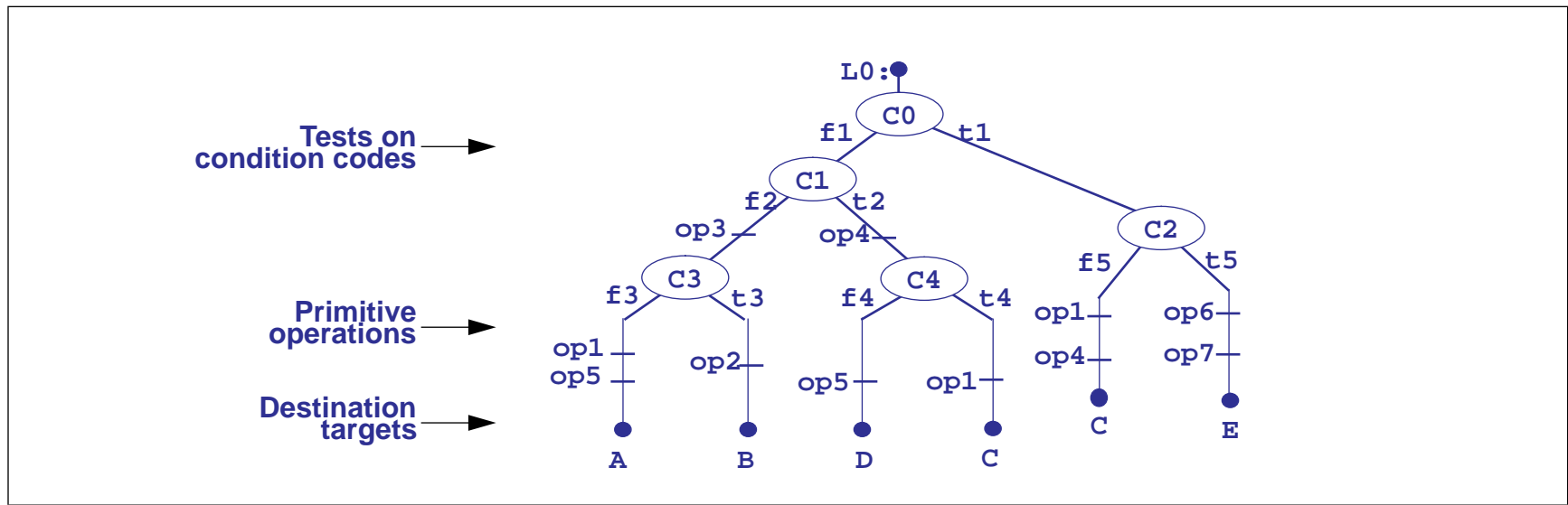
- ***implementation-independent*** representation of program in main memory
 - suitable for execution in any implementation
- translation into ***implementation-dependent*** format in I-cache

Assumptions

- there is sufficient instruction-level parallelism (ILP) in programs
- ILP is adequately expressed by ***multiway tree-instructions***

Program representation: multiway tree-instructions

- **Unlimited** number of internal nodes, arcs and targets
- **Sequential** semantics for operations in each path of the tree
 - guarantees compatibility across implementations
- **Entire tree executable in parallel**
 - VLIW compiler generates parallel trees
- **Subtree is also a valid tree**
 - possible to branch into a tree-instruction



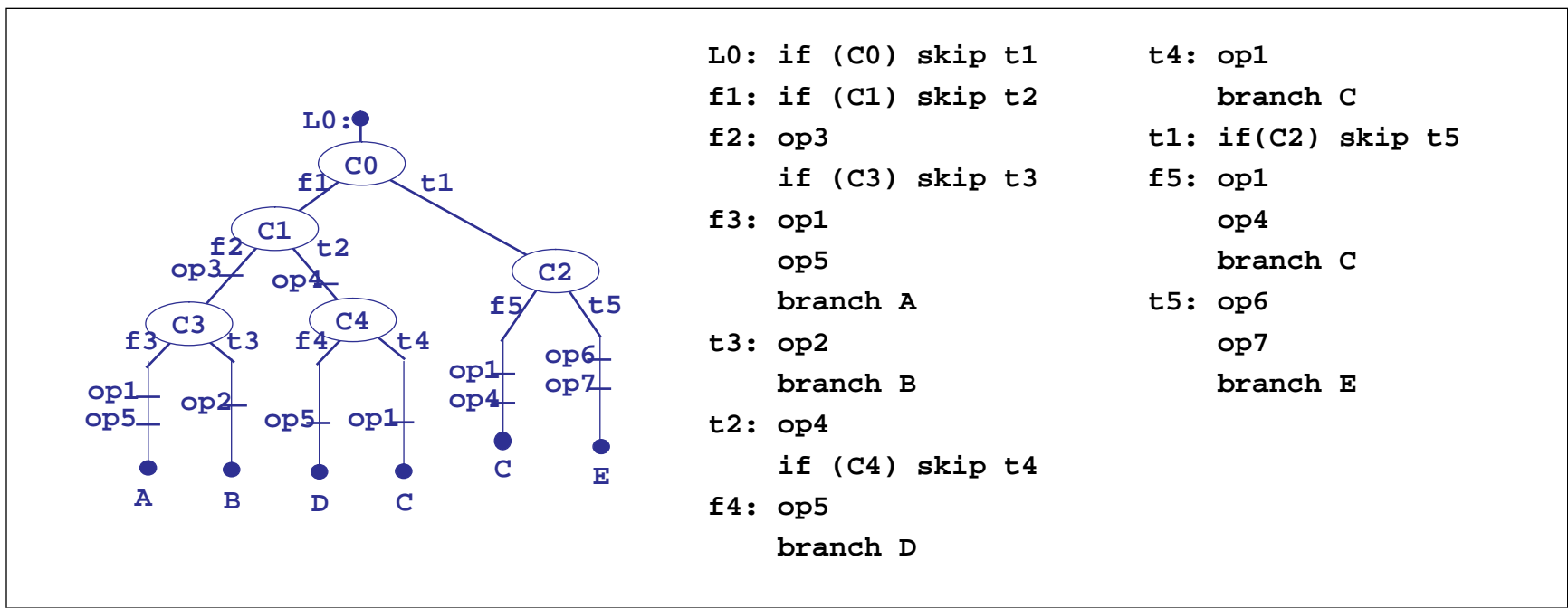
Representation in main memory

Sequential program from depth-first traversal of tree

- potential replication of some primitive operations

New instruction: conditional skip

- control flow within a tree-instruction



Representation in main memory (cont.)

Representation directly executable by scalar/superscalar processor

End of tree-instruction

Unconditional branch

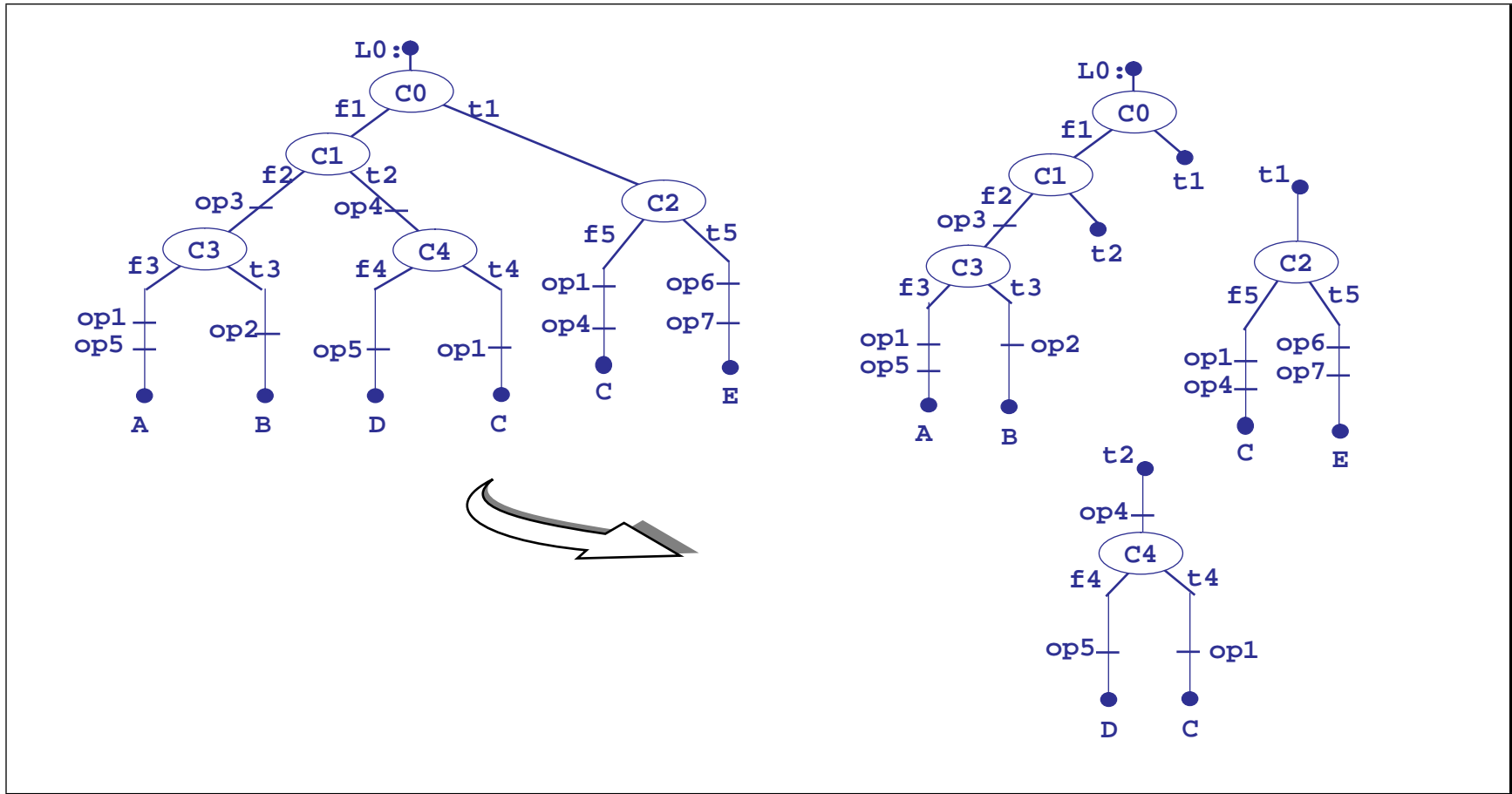
Next primitive instruction unreachable from any skip

```
L0: if (C0) skip t1      t4: op1
f1: if (C1) skip t2      branch C
f2: op3                 t1: if(C2) skip t5
    if (C3) skip t3      f5: op1
f3: op1                 op4
    op5                 branch C
    branch A           t5: op6
t3: op2                 op7
    branch B           branch E
t2: op4                 L1: .....
    if (C4) skip t4
f4: op5
    branch D
```

Execution of tree-instruction in VLIW processor with limited resources

Basic concept: "Pruning" the tree

- semantics remain unchanged



Pruning procedure

- Replacement of skip instructions by ***conditional branch instructions***; or
- Introduction of ***implicit branch instructions*** (to next sequential memory address)

Allows for

- insufficient resources: operations, branching abilities
- dependencies within tree-instructions

Number of primitive instructions does not change

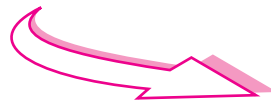
Branching into subtrees still possible

Pruning performed at I-cache reload time

Example: pruning

- Skip instructions replaced by *conditional branch instructions*
- Large tree-instruction decomposed into smaller ones

```
L0: if (C0) skip t1    t4: op1
f1: if (C1) skip t2    branch C
f2: op3                t1: if(C2) skip t5
    if (C3) skip t3    f5: op1
f3: op1                op4
    op5                branch C
    branch A          t5: op6
t3: op2                op7
    branch B          branch E
t2: op4
    if (C4) skip t4
f4: op5
    branch D
```

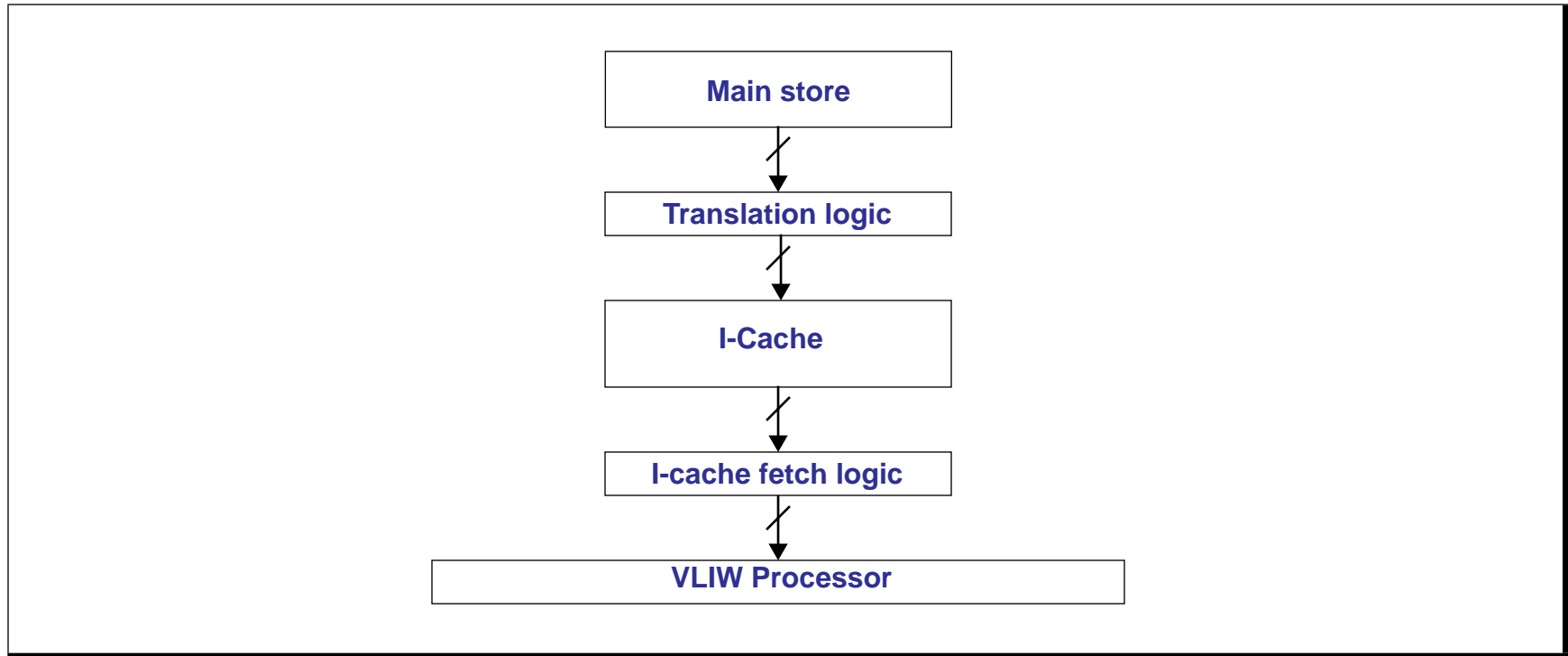


```
L0: if (C0) br t1
    if (C1) br t2
f2: op3
    if (C3) skip t3
f3: op1
    op5
    branch A
t3: op2
    branch B
```

```
t2: op4
    if (C4) skip t4
f4: op5
    branch D
t4: op1
    branch C
```

```
t1: if(C2) skip t5
f5: op1
    op4
    branch C
t5: op6
    op7
    branch E
```

Overall structure



Outcome from translation process

Variable-length tree-instructions which fit VLIW implementation constraints

Multiple variable-length trees per I-cache line

- I-cache line longer than maximum VLIW size; or
- trees may straddle consecutive I-cache lines

Additional predecoded information per I-cache line to simplify implementation

- boundaries of trees within line
- type of operation per parcel
- implicit branches

L0: if(C0) br t1	if(C1) br t2	op3	if(C3) br t3	op1	op5	br A	t3: op2
br B	t2: op4	if(C4) sk t4	op5	br D	t4: op1	br C	t1: if(C2) sk t5
op1	op4	br C	t5: op6	op7	br E		

Outcome from I-cache access process

Fixed-length Very-Long Instruction Word (VLIW)

Expansion/adjustment to fit VLIW format

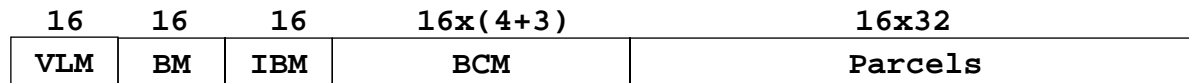
- uses predecoded information stored in I-cache line
- additional predecoded information to simplify implementation
 - target execution mask
 - branch conditions mask
 - branch targets

L0: if(C0) br t1	if(C1) br t2	if(C3) br t3	br A	op1	op5	op3	nop
t3: br B	nop	nop	nop	op2	nop	nop	nop
t2: if(C4) sk t4	br D	br C	nop	op5	t4: op1	nop	nop

Example: I-cache line format

Predecoded masks

VLIW parcels (operations, branches)



VLM: VLIW length mask

BM: branches mask

IBM: implicit branches mask

BCM: branch conditions mask

Parcels: operations

Translation logic (high-level description)

Translation performed on main memory *blocks*

For each word in a main memory block

```
if (Opcode = skip) then
    if (target is beyond end_of_block) then
        Replace Opcode by cond_branch;
    if (Num_paths equal to Num_max) then
        insert_implicit_branch (in previous operation);
        start_new_tree;
    else
        increase_count_number_of_paths;
    endif;
    save_skip_target;
elseif (Opcode = branch) then
    if (next_instr not reachable from any skip) then
        start_new_tree_in_next_instr;
    endif;
endif;
```

Example: VLIW format

Predecoded masks: BCM, TEM

Branch targets

VLIW parcels (operations, branches)



BCM: Branch conditions mask

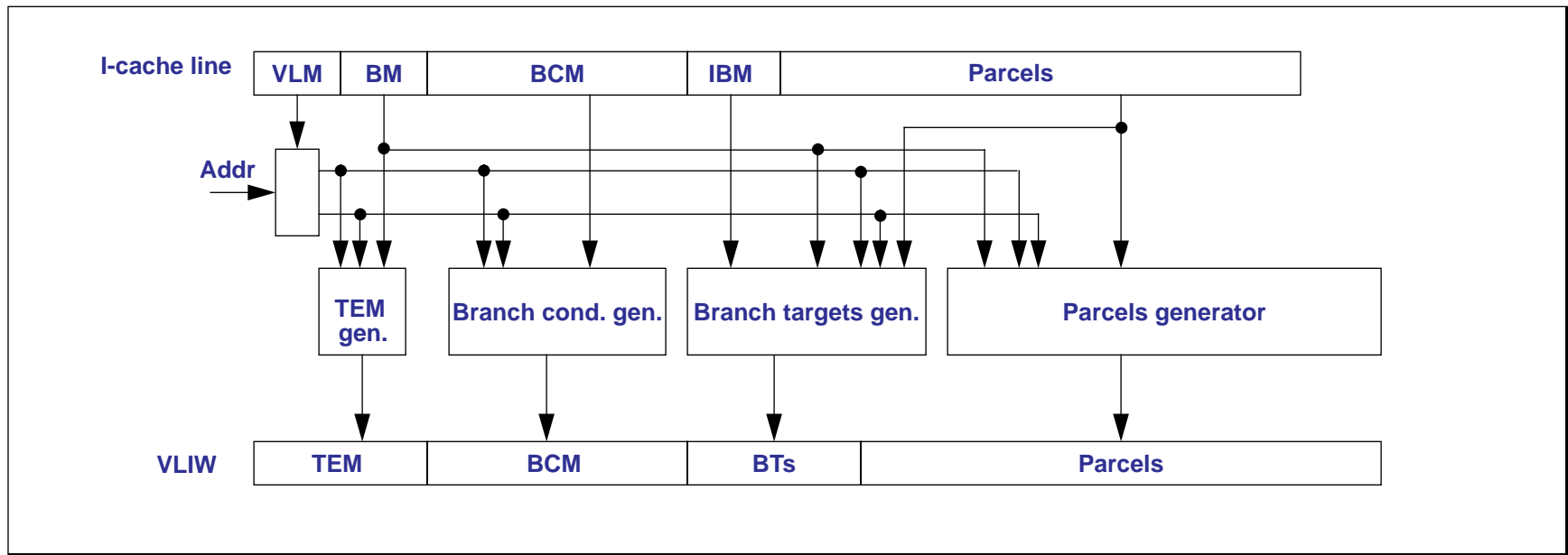
BTs: branch targets

TEM: target execution mask

Parcels: operations

I-cache access logic

Expansion/alignment network



Summary

Manageable complexity in translation and cache access logic

No code expansion due to no-ops

Branching into tree-instruction possible

- less code expansion due to replication

Hides implementation details from architecture

Same program executable in wide variety of implementations

- including superscalar processors

Bounds (intuitively)

Lower bound: 4-word VLIW, 8-word I-cache line

- straddle I-cache lines (perhaps)

Upper bound

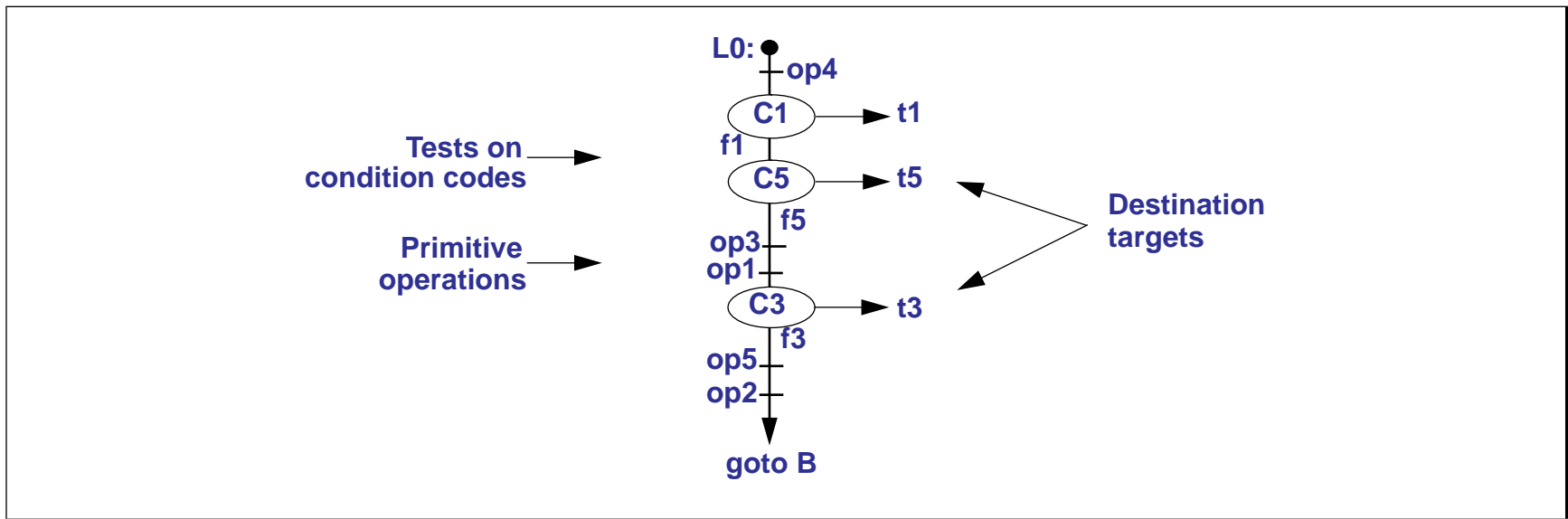
- limited by number of registers and instruction-level parallelism

Impact of large tree-instructions executed on smaller implementation?

Execution of sequential code in VLIW processor

Program interpretation: *vertical tree-instructions*

- **Fall-through path in sequential program, up to unconditional branch**
 - Conditional branches regarded as “skip” followed by “branch”
- **Unlimited number of internal nodes, arcs and targets**
- **Entire tree executable in parallel, if no internal dependencies**
 - Implicit branches if dependent operations exist
- **Subtree is also a valid tree**



Implementation aspects

Vertical tree-instructions are prunable

- Insertion of implicit unconditional branches

Same translation and I-cache fetch processes

Same overall processor structure

Uses the same features for VLIW code as well as sequential code