14 Superscalar Processors

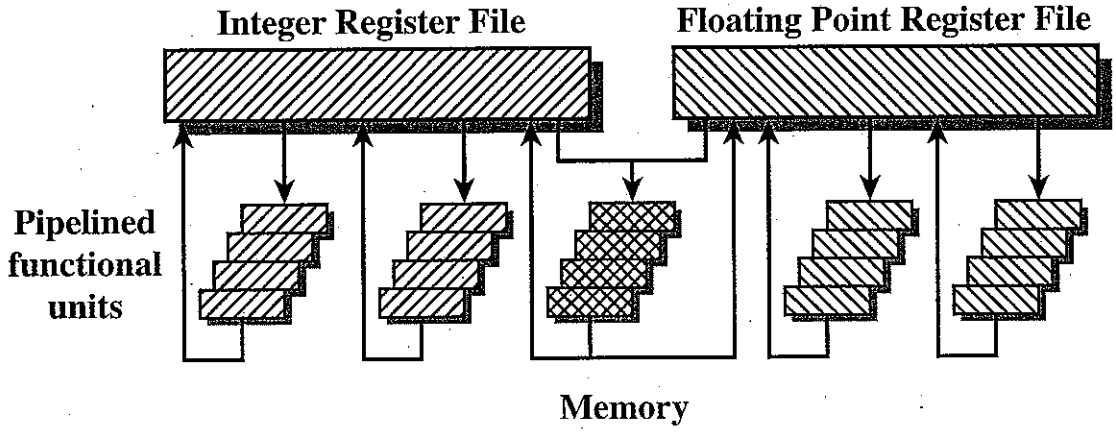
Motivation

- o Most operations are on scalar quantities (about 80%)
- o Speedup these operations will lead to large overall performance improvement.,

Instruction-level parallelism

- o Several scalar instructions can be initiated simultaneously and executed independently.
- o Superscalar processor includes pipelining where several instructions can be executed simultaneously in the same pipeline stage.

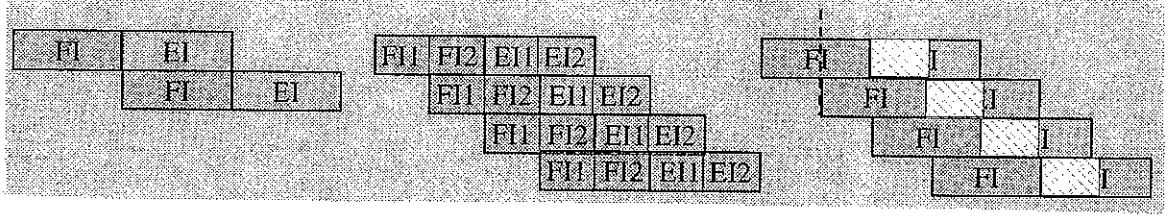
General Superscalar Organization



Note. Two integer, two floating-point, and one memory (load/store) operations can be executed at the same time.

Superpipelining

- Superpipelining is based on dividing the stages of a pipeline into several sub-stages, and thus increasing the number of instructions which are handled by the pipeline at the same time.
- For example, by dividing each stage into two sub-stages, a pipeline can perform at twice the speed in the ideal situation.
  - Many pipeline stages may perform tasks that require less than half a clock cycle.
  - No duplication of hardware is needed for these stages.



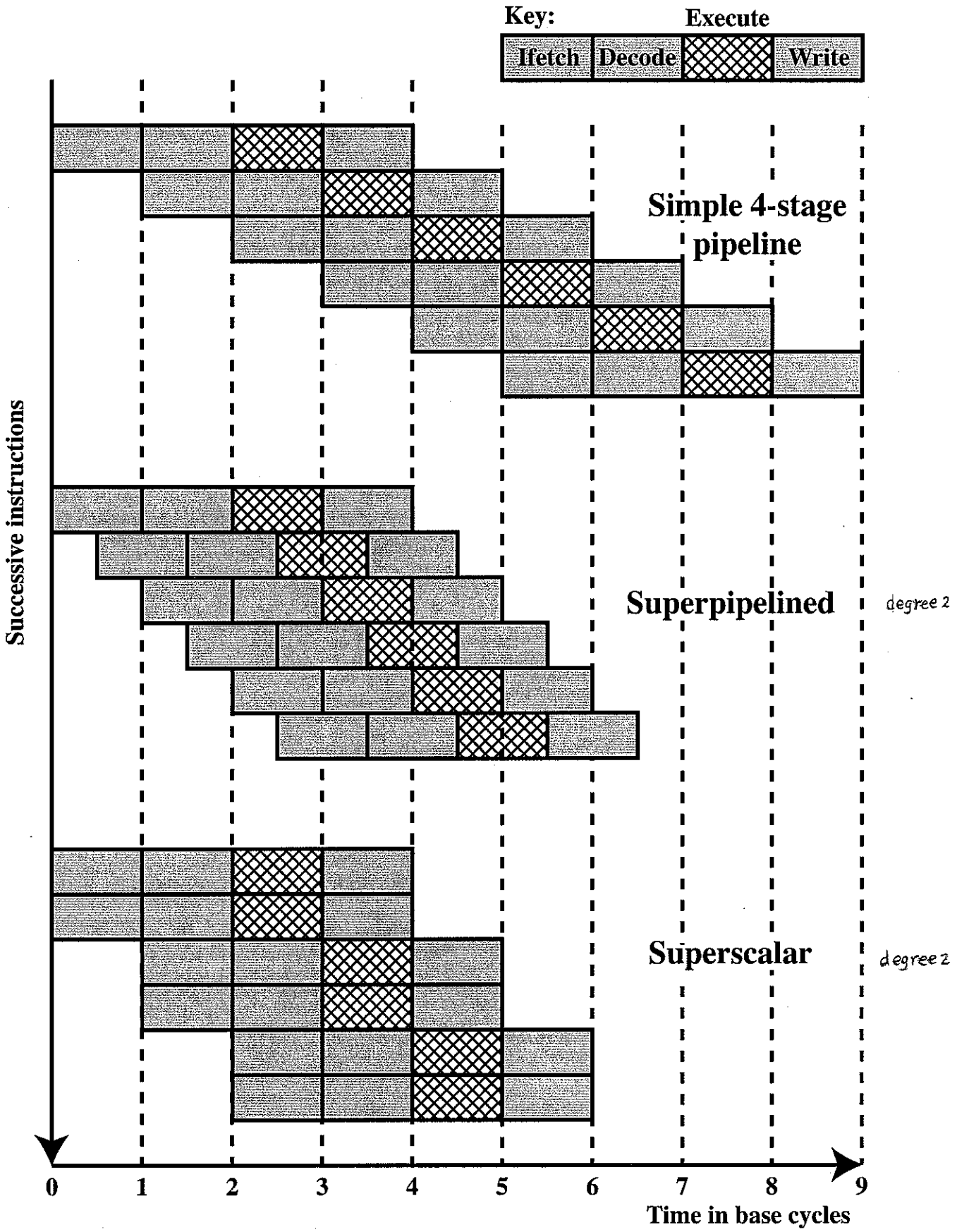
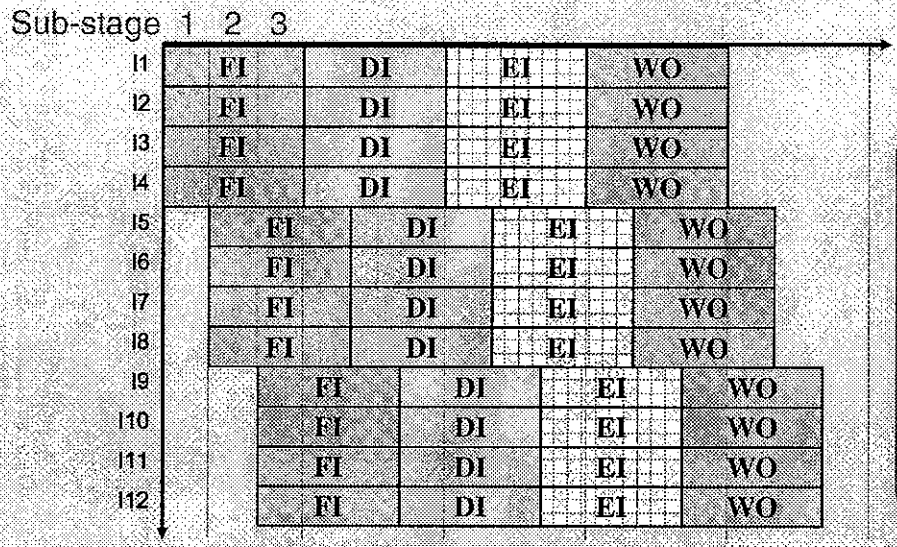


Figure 14.2 Comparison of Superscalar and Superpipeline Approaches

Superpipelined Superscalar Design



Superpipeline of degree 3 and superscalar of degree 4:

- 12 times speed-up over the base machine.
- 48 times speed-up over sequential execution.

- This is a new trend of architecture design:
  - Pentium Pro(P6): 3-degree superscalar, 12-stage superpipeline
  - PowerPC 620: 4-degree superscalar.

Parallel Execution Limitation

(1) Resource conflicts

- o Several instructions compete for the same hardware resource at the same time.
  Ex. Two arithmetic instructions need the same floating-point unit for execution

- o Solution – several hardware units for the same function.
  Ex. Having two floating-point units.
  Note. The hardware unit can be pipelined.

(2) Procedural dependency

- o Branch – cannot execute instructions after a branch in parallel with instructions before a branch.

- o Variable-length instruction - it cannot be fetched and issued in parallel, since an instruction has to be decoded in order to identify the following one.

Therefore, superscalar techniques are more efficiently applicable to RISC, with fixed instruction length and format.

Data Conflicts

- o Caused by data dependencies between instructions in the program.
- o To increase the degree of parallel execution, Superscalar provides a great liberty in the order in which instructions can be issued and executed.
- o Therefore, data dependencies have to be considered and dealt with much more carefully.

Out of Order Execution

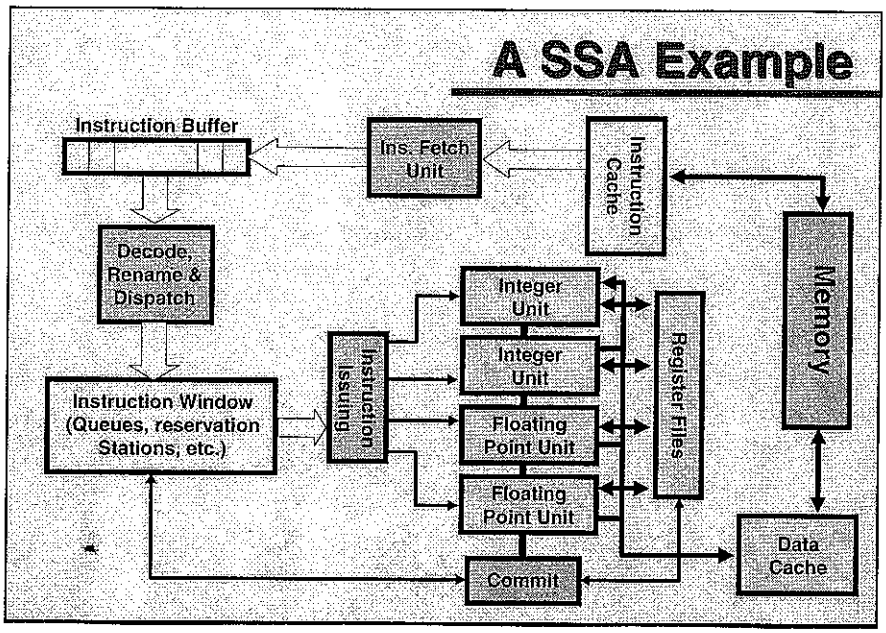
- Superscalar exploits the potential of instruction-level parallelism presented in the program.
- This is often achieved by **dynamic instruction scheduling**:
  - Instructions are issued for execution dynamically, in parallel and out of order.
  - Out of order issuing: Instructions are issued independent of their original sequential order, based on dependencies and availability of resources.
- The results must be identical with those produced by strictly sequential execution.

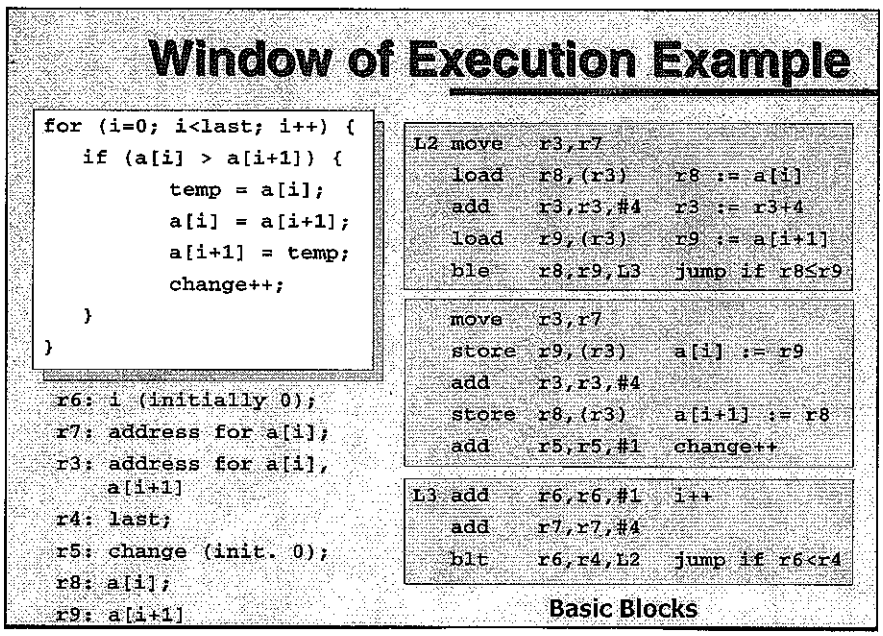
Window of Execution

- Due to data dependencies, only some part of instructions are potential subjects for parallel execution.
- In order to find instructions to be issued in parallel, the processor has to select from a sufficiently large instruction
- Window of Execution
  - The set of instructions that is considered for execution at a certain moment
  - Any instruction in the window can be issued for parallel execution, subject to data dependencies and resource constraints.
- The number of instructions in the window should be as large as possible. However, the capacity to fetch instructions at a high rate is limited. And there is the problem of branches. Also, the cost of hardware is an issue.

- The window of execution can be extended over basic block borders by branch prediction – Speculative execution

- With speculative execution, instructions of the predicted path are entered into the window of execution.
  - Instructions from the predicted path are executed tentatively.
  - If the prediction turns out to be correct the state change produced by these instructions will become permanent and visible ("commit"); if not, all effects are removed





Data Dependencies

- o True data dependency
- o Output dependency
- o Anti-dependency

} Artificial dependencies

(3) True data dependency (write-read dependency)

```
ex.  add   r1, r2   //write
     :
     mov  r3, r1    // read
```

Ex. I1: R3 ← R3 + R5
I2: R4 ← R3 + 1
I3: R3 ← R5 + 1
I4: R7 ← R3 + R3

(4) Output dependency (write-write dependency)
Q. Can I1 and I3 be executed in parallel?
A. I3 must complete after I1 produce the new value of R3

(5) Anti-dependency (read-write dependency)
Q. Can I2 and I3 be executed in parallel?
A. I3 cannot complete execution before I2 fetch R3 value.

Register Renaming

o Output dependencies and anti-dependencies can usually be eliminated by using additional registers.

Ex 1.

R3 ← R3 + R5	R3 ← R3 + R5
:	:
R3 ← R5 + 1	R7 ← R5 + 1

Ex 2.

R4 ← R3 + 1	R4 ← R3 + 1
R3 ← R5 + 1	R7 ← R5 + 1

Instruction vs. Machine Parallelism

o Instruction parallelism

average number of instructions in a program that a processor might be able to execute at the same time

o Machine parallelism

ability of the processor to take advantage of the instruction-level parallelism of the program

To achieve high performance, we need both instruction-level parallelism and machine parallelism.

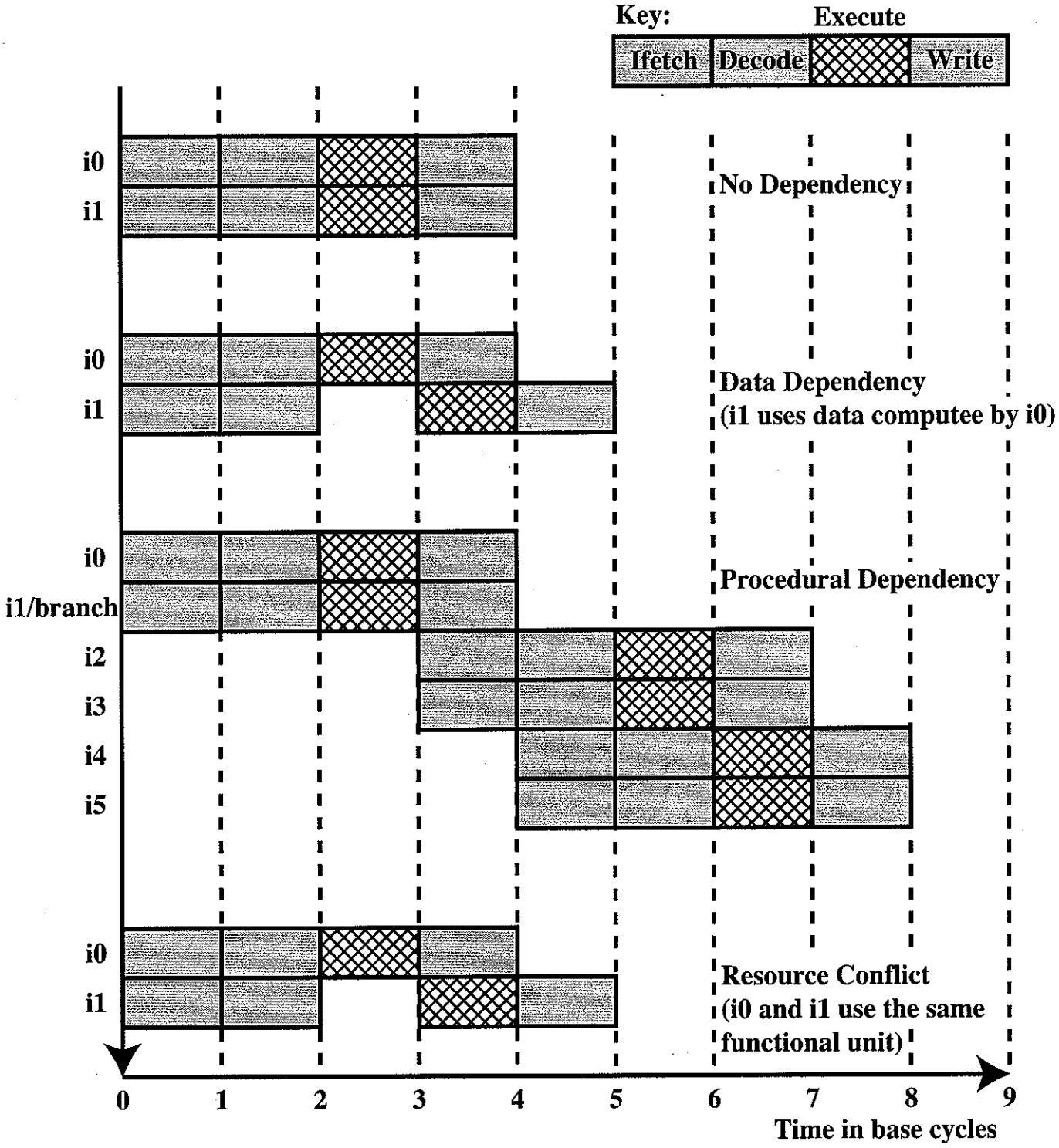


Figure 14.3 Effect of Dependencies

Instruction Issue and Completion

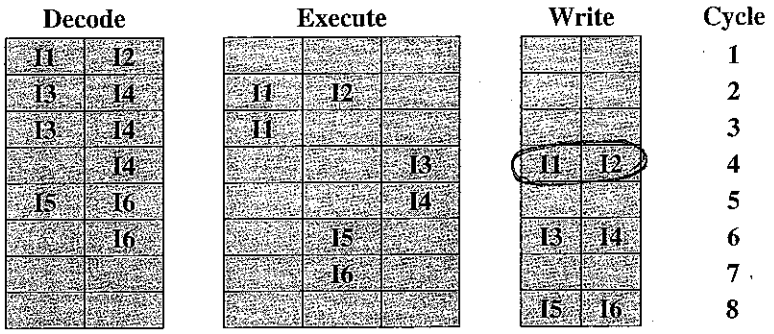
Parallel instruction execution can be characterized by the order of the following three activities:

- **Instruction issue** — an instruction is initiated and starts execution.
- **Instruction completion** — an instruction has competed its specified operations.
- **Instruction commit** — the results of the instruction operations are written back to the register files or cache.
  - The machine state is changed.

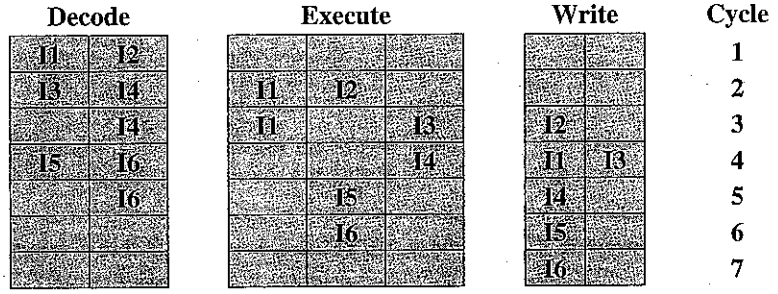
Instruction Execution Policies

- Instructions will be executed in an order different from the strictly sequential one, with the restriction that **the results must be correct.**

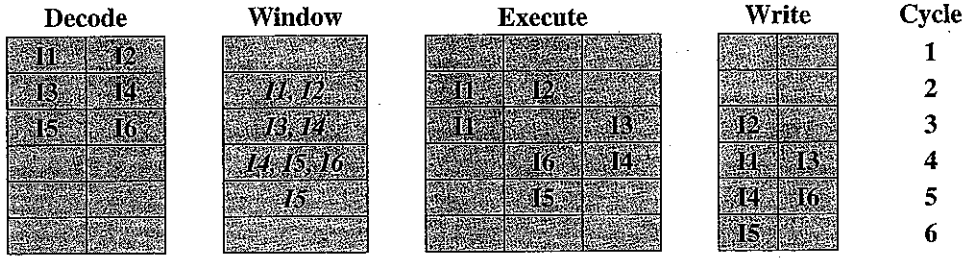
- **Execution policies** usually used:
  - In-order issue with in-order completion.
  - In-order issue with out-of-order completion.
  - Out-of-order issue with out-of-order completion.



(a) In-order issue and in-order completion



(b) In-order issue and out-of-order completion



(c) Out-of-order issue and out-of-order completion

Figure 14.4 Superscalar Instruction Issue and Completion Policies

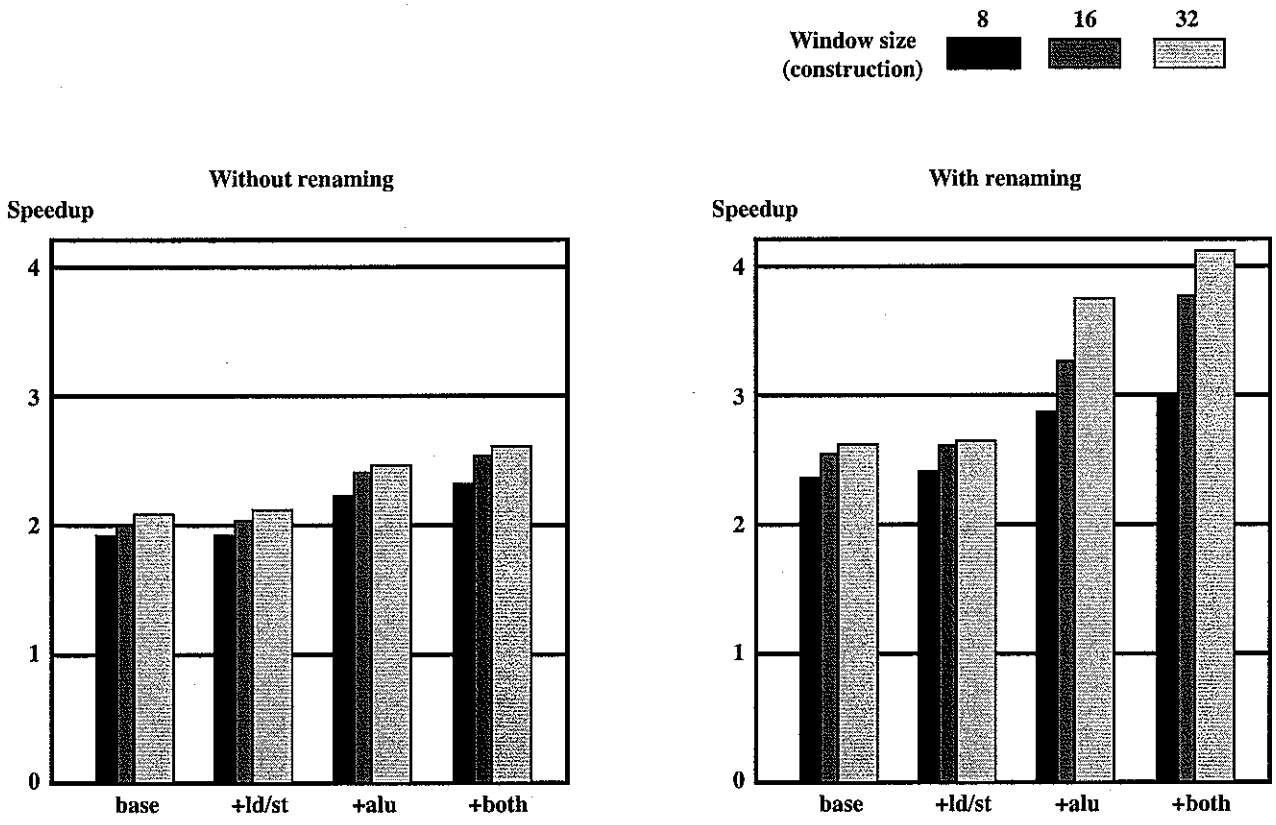


Figure 14.5 Speedups of Various Machine Organizations Without Procedural Dependencies

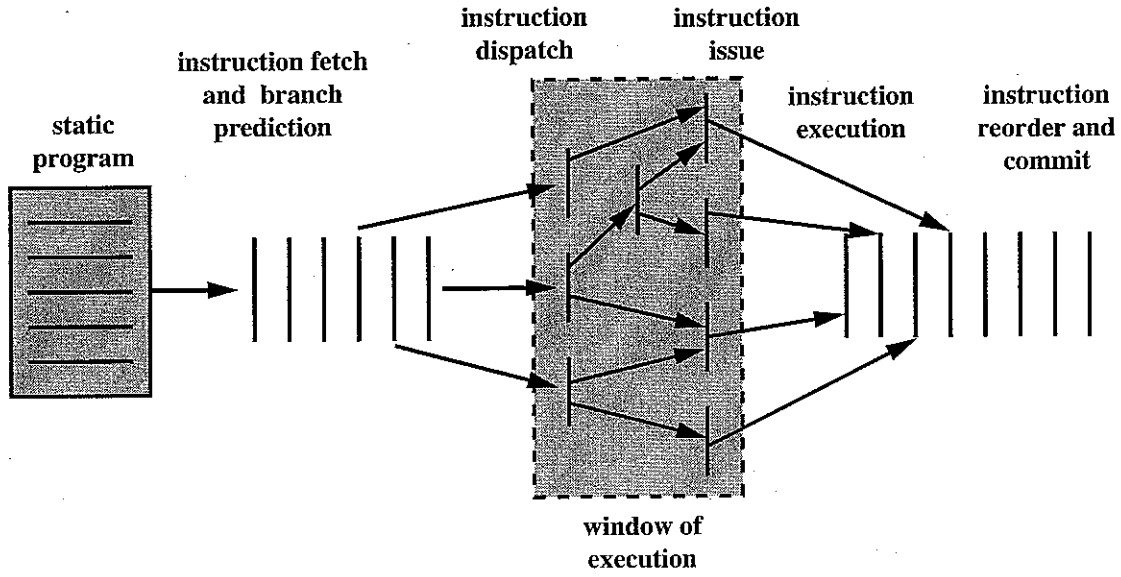
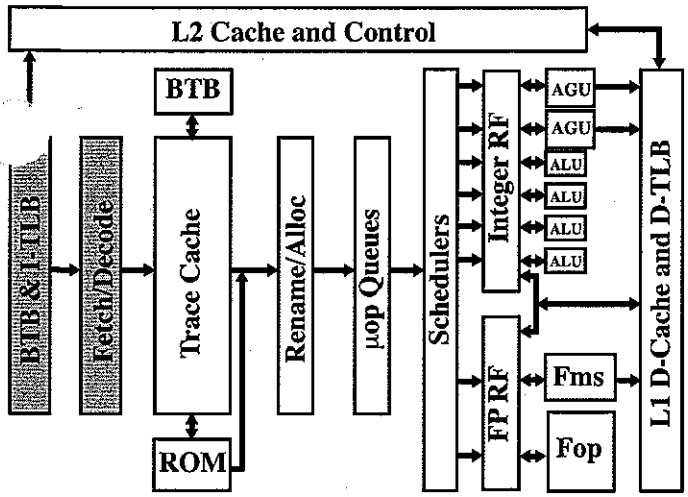


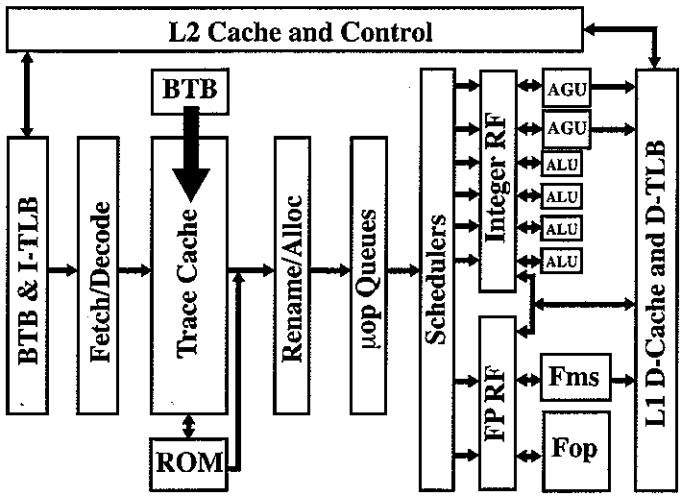
Figure 14.6 Conceptual Depiction of Superscalar Processing [SMIT95]

Final Remarks

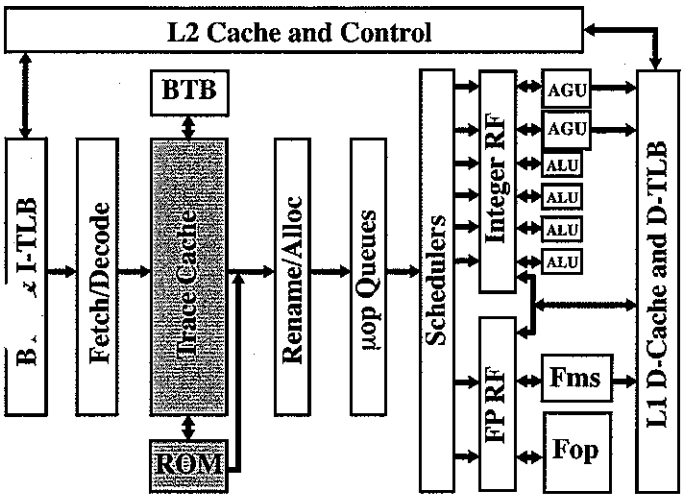
- The following techniques are main features for superscalar processors:
  - Several pipelined units which are working in parallel;
  - Out-of-order issue and out-of-order completion;
  - Register renaming.
- All of the above techniques are aimed to enhance performance.
- Experiments have shown:
  - Only adding additional functional units is not very efficient;
  - Out-of-order issue is extremely important, it allows to look ahead for independent instructions;
  - Register renaming can improve performance with more than 30%, in this case performance is limited only by true dependencies.
  - It is important to provide a fetching/decoding capacity so that the window of execution is sufficiently large.



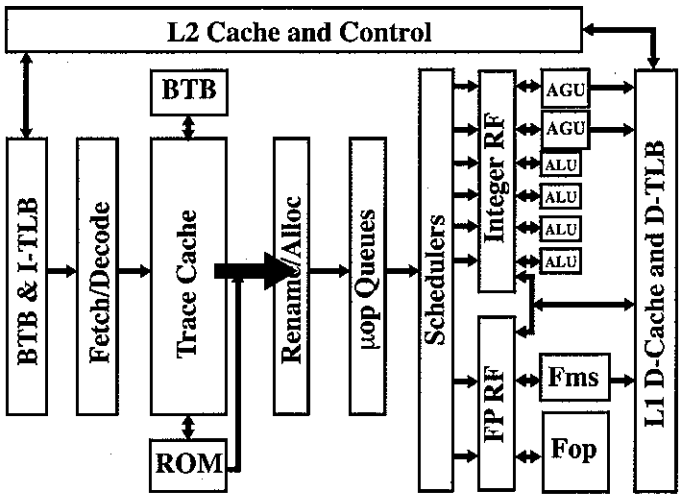
(a) Generation of micro-ops



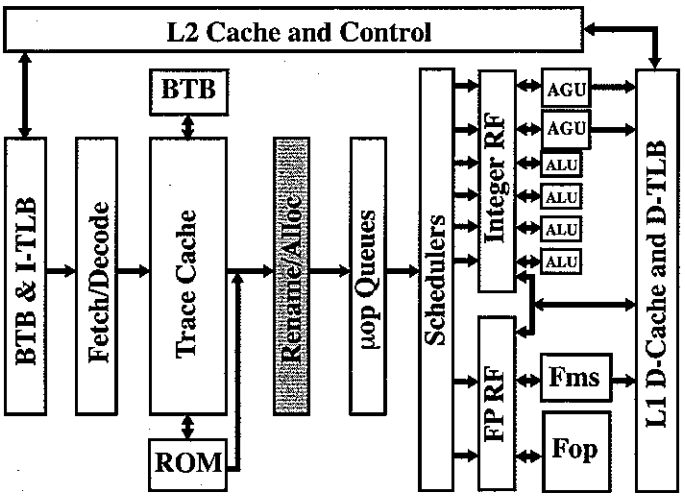
(b) Trace cache next instruction pointer



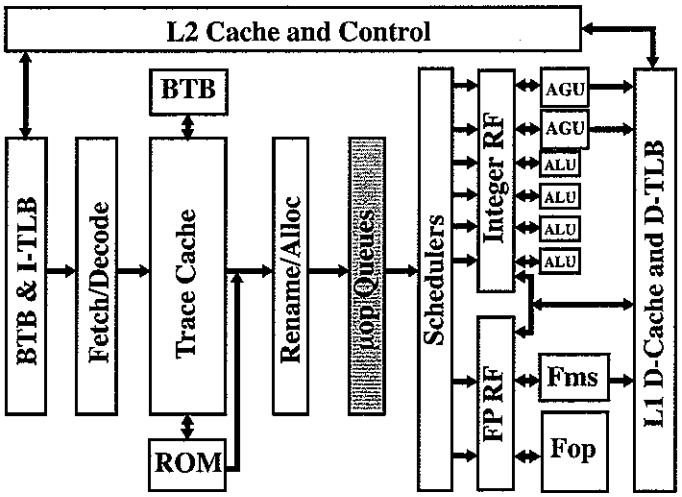
(c) Trace cache fetch



(d) Drive

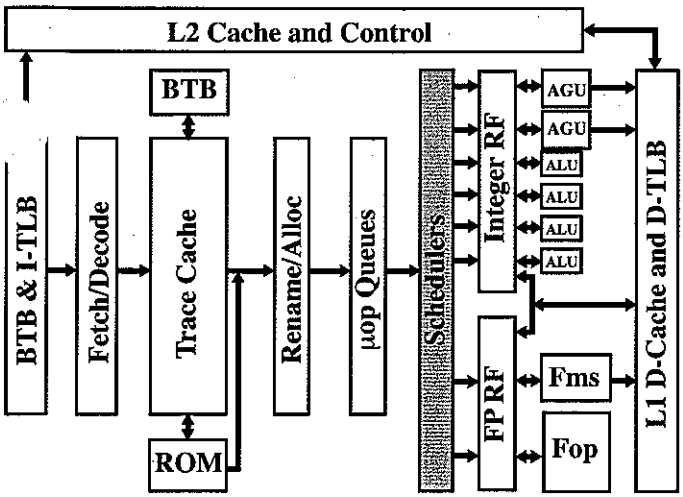


(e) Allocate; Register renaming

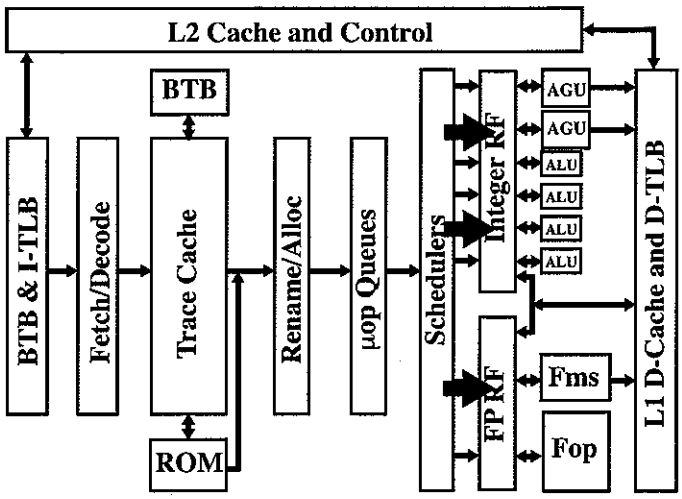


(f) Micro-op queuing

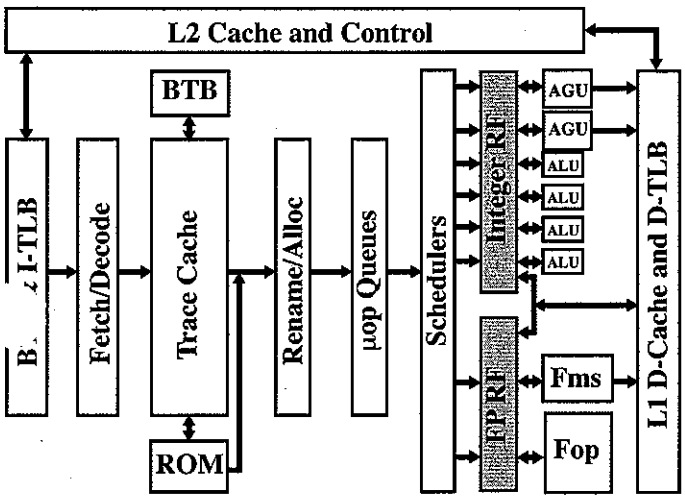
Figure 14.9 Pentium Pipeline Operation (page 1 of 2)



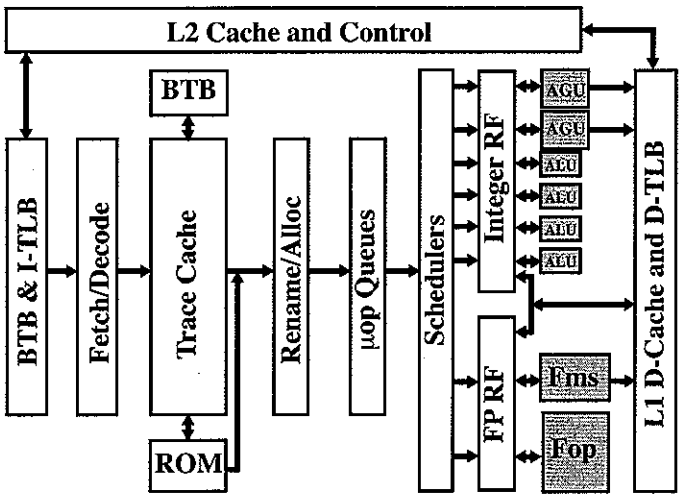
(g) Micro-op scheduling



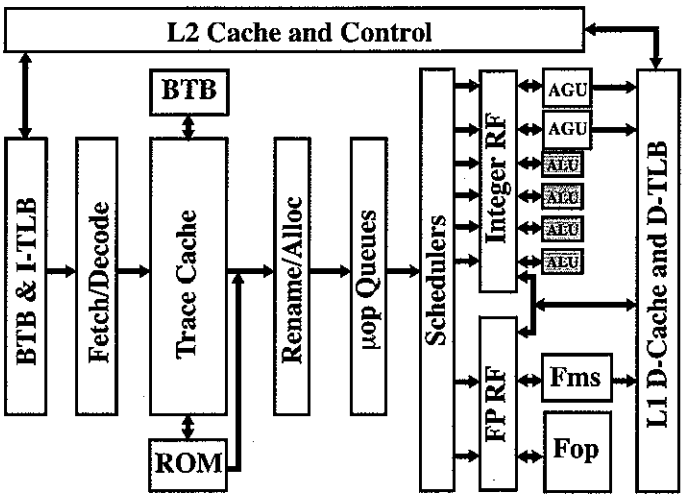
(h) Dispatch



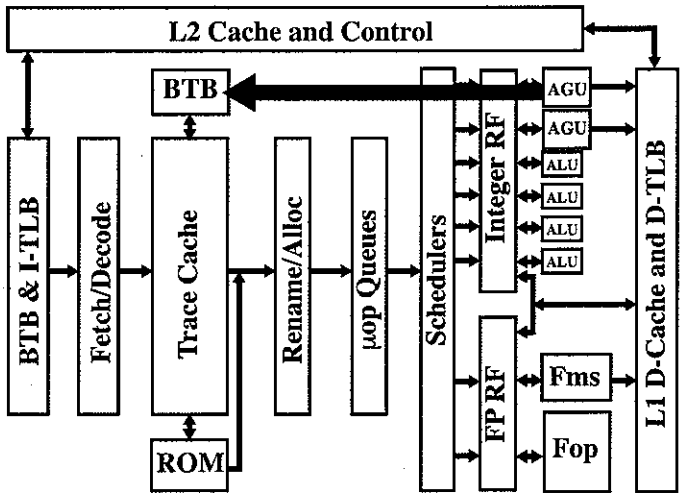
(i) Register file



(j) Execute; flags



(k) Branch check



(l) Branch check result

Figure 14.9 Pentium Pipeline Operation (page 2 of 2)