Preserving Sequential Consistency of Instruction Execution

In processors with multiple EUs operating in parallel, instructions can finish in an out-of-order fashion. Nevertheless, overall instruction execution should mimic sequential execution, i.e. it should preserve *sequential consistency*. Sequential consistency of instruction execution relates to two aspects:

1. to the order in which instructions are completed (*Processor Consistency*); and
2. to the order in which memory is accessed (*Memory Consistency*) due to load and store instructions or memory references of other instructions.

Processor Consistency

The term *Processor Consistency* is used to indicate the consistency of instruction completion with that of sequential instruction execution. Superscalar processors preserve either a *weak* or a *strong consistency*.

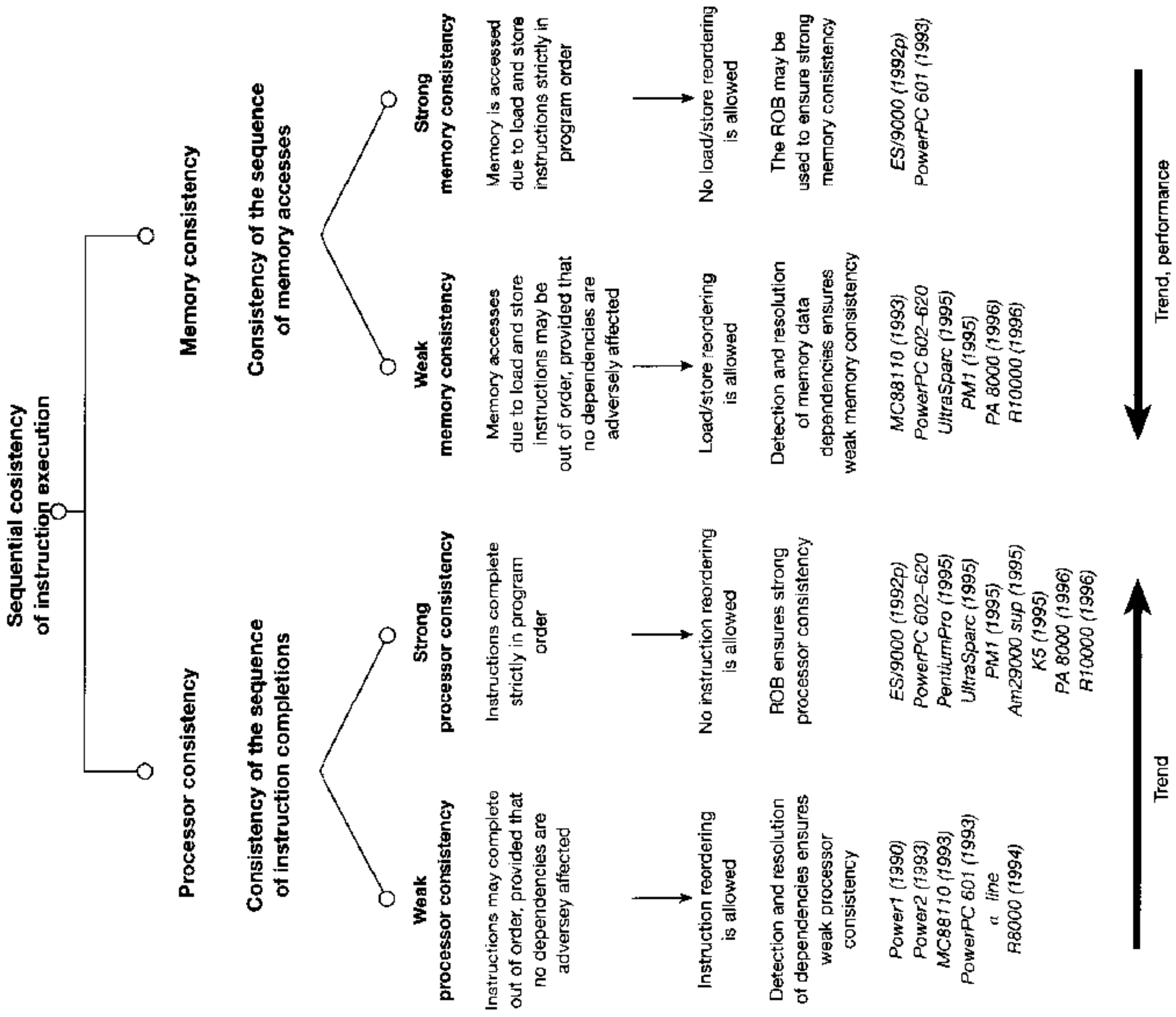
- A weak processor consistency means that instructions may *complete* out-of-order, provided that no data dependencies are violated. In order to achieve this, data dependencies need to be detected and resolved during execution.

- Strong processor consistency means instructions are forced to *complete* in program order. Usually this achieved by employing a reorder buffer (ROB).

Memory Consistency

The other aspect of superscalar instruction execution is whether memory access are performed in the same order as in a sequential processor.

- Memory consistency is *weak* if memory access may be out-of-order compared with a strict sequential program execution. However, data dependencies must not be violated. In other words, weak consistency allows load/store reordering providing that dependencies, paricularly memory data dependencies, are detected and resolved.
- In the case of strong memory consistency, where memory access occurs strictly in program order, load/store reordering is forbidden.


Sequential cosistency
of instruction execution
Processor consistency
Memory consistency
Consistency of the sequence
of instruction completions
Consistency of the sequence
of memory accesses
Weak
processor consistency
Strong
processor consistency
Weak
memory consistency
Strong
memory consistency
Instructions may complete
out of order, provided that
no dependencies are
adversey affected
Instructions complete
strictly in program
order
Memory accesses
due to load and store
instructions may be
out of order, provided that
no dependencies are
adversely affected
Memory is accessed
due to load and store
instructions strictly in
program order
Instruction reordering
is allowed
No instruction reordering
is allowed
Load/store reordering
is allowed
No load/store reordering
is allowed
Detection and resolution
of dependencies ensures
weak processor
consistency
ROB ensures strong
processor consistency
Detection and resolution
of memory data
dependencies ensures
weak memory consistency
The ROB may be
used to ensure strong
memory consistency
Power1 (1990)
Power2 (1993)
MC88110 (1993)
PowerPC 601 (1993)
α line
R8000 (1994)
ES/9000 (1992p)
PowerPC 602–620
PentiumPro (1995)
UltraSparc (1995)
PM1 (1995)
Am29000 sup (1995)
K5 (1995)
PA 8000 (1996)
R10000 (1996)
MC88110 (1993)
PowerPC 602–620
UltraSparc (1995)
PM1 (1995)
PA 8000 (1996)
R10000 (1996)
ES/9000 (1992p)
PowerPC 601 (1993)
Trend
Trend, performance

Load/Store Reordering

Load and store instructions involve actions affecting both the processor and the memory. While executing, both loads and stores must first wait for their addresses to be computed by an ALU or address unit.

- Then loads can access the data cache to fetch the requested memory data and the instruction is completed when the data is written into the architectural register.

- After receiving the generated address, a store instruction must wait for its operands before being considered completed.

Load/Store Reordering$_{(cont)}$

A processor that supports *weak memory consistency* allows the reordering of memory accesses. This is advantageous for at least three reasons:

1. it permits load/store bypassing,

2. it makes speculative loads or stores feasible, and

3. it allows cache misses to be hidden.

Load/Store Bypassing

Load/Store bypassing means that either load can bypass stores or vice versa, provided that no memory data dependencies are violated.

- Permitting loads to bypass stores has the advantage of allowing the runtime overlapping of loops. This is achieved by allowing loads at the beginning of an iteration to access memory without having to wait until stores at the end of the previous iteration are completed.
- In order to avoid fetching a false data value, a load can bypass pending stores if none of the preceding stores have the same target address as the load.

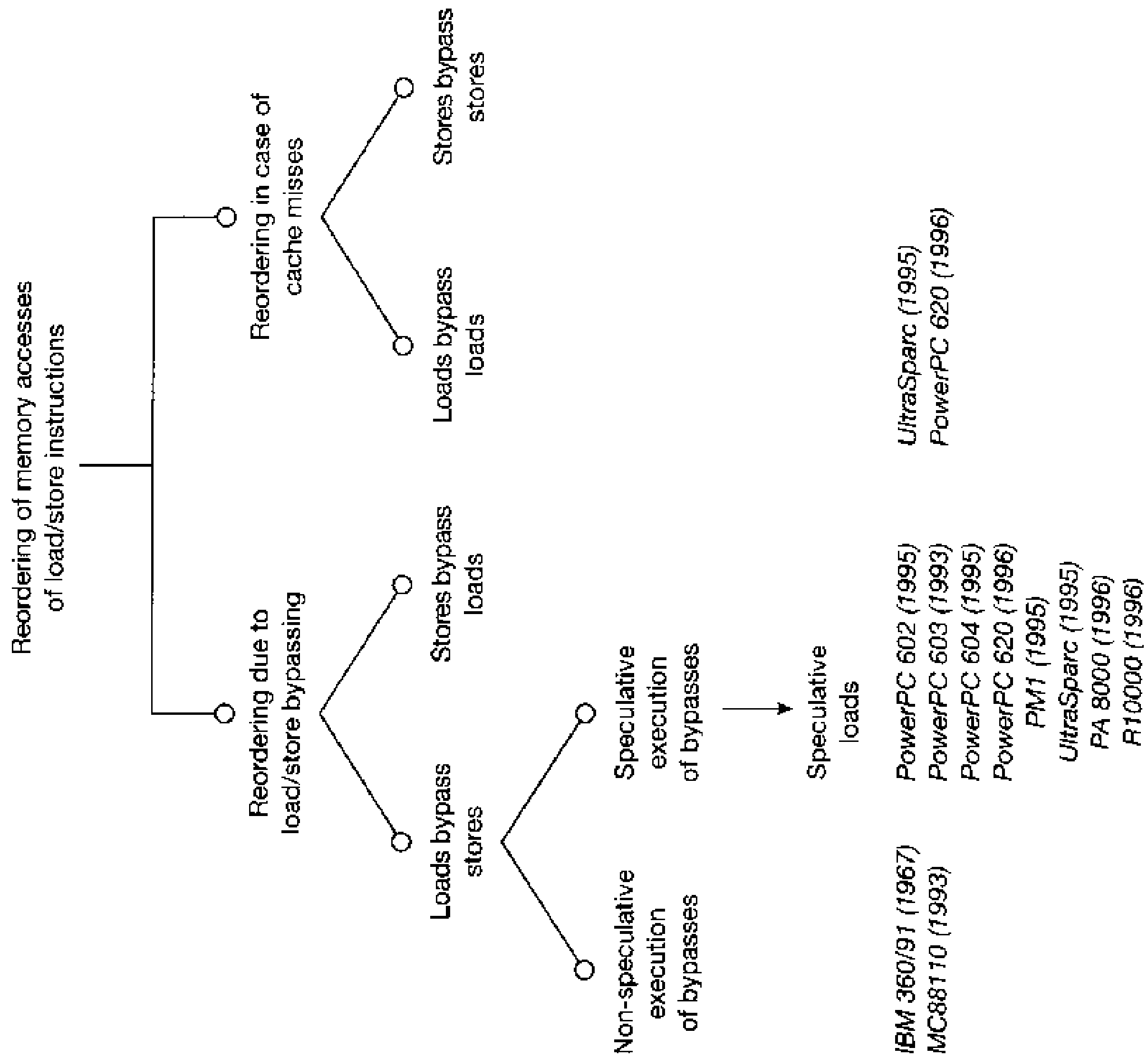
However, certain addresses of pending stores may not be available.

Speculative Loads

Speculative loads avoid delaying memory accesses until all required addresses have been computed and clashing addresses can be ruled out.

The correctness of speculative loads must be checked and, if necessary, be undone. Note that speculative loads are quite similar to speculative branches.

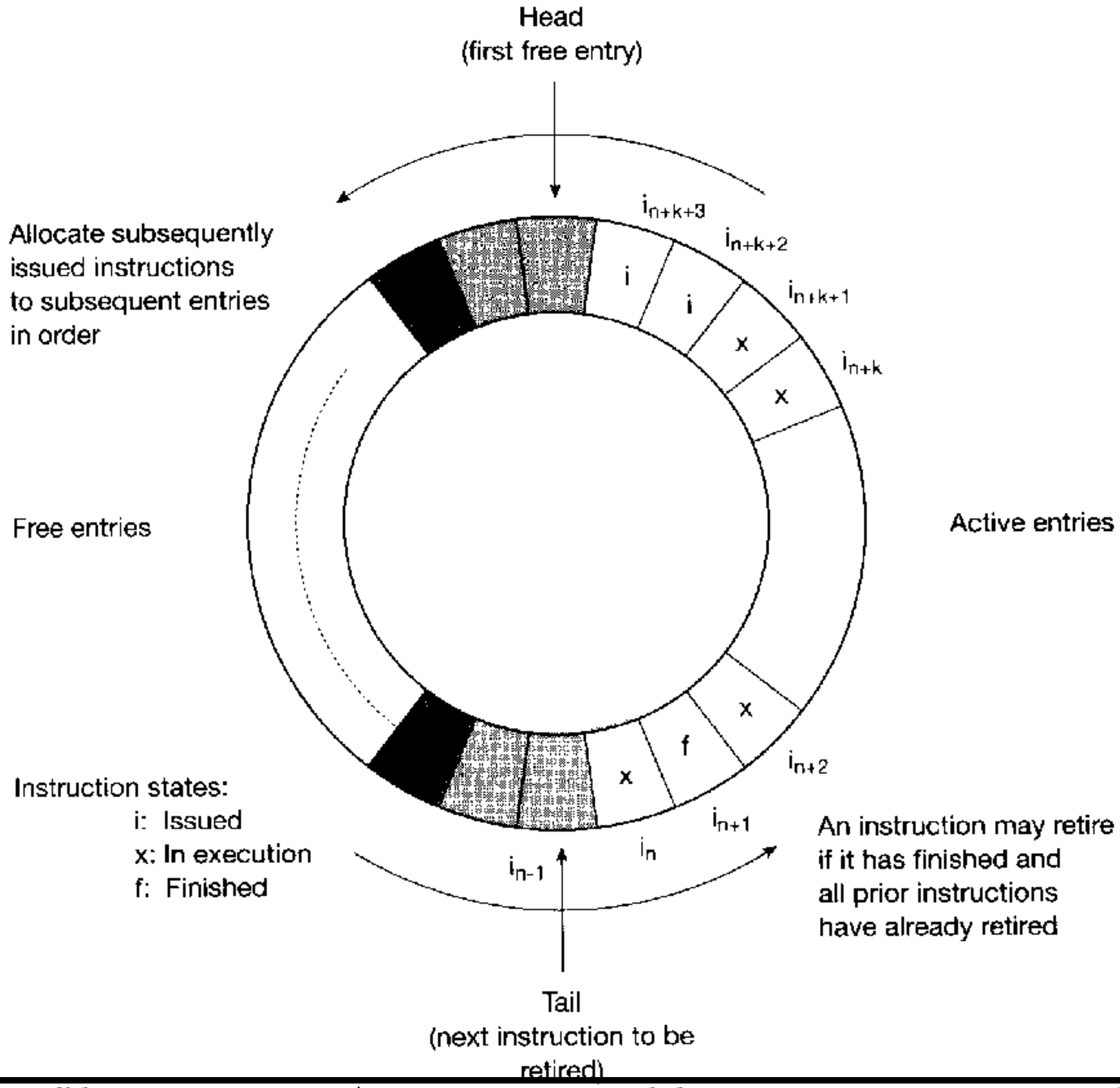
The address checks are usually carried out by writing the computed target address of loads and stores into the ROB (or DRIS) and performing the address comparison there.


Reordering of memory accesses of load/store instructions
Reordering due to load/store bypassing
Reordering in case of cache misses
Loads bypass stores
Stores bypass loads
Loads bypass loads
Stores bypass stores
Non-speculative execution of bypasses
Speculative execution of bypasses
Speculative loads
IBM 360/91 (1967)
MC88110 (1993)
PowerPC 602 (1995)
PowerPC 603 (1993)
PowerPC 604 (1995)
PowerPC 620 (1996)
PM1 (1995)
UltraSparc (1995)
PA 8000 (1996)
R10000 (1996)
UltraSparc (1995)
PowerPC 620 (1996)

The Reorder Buffer

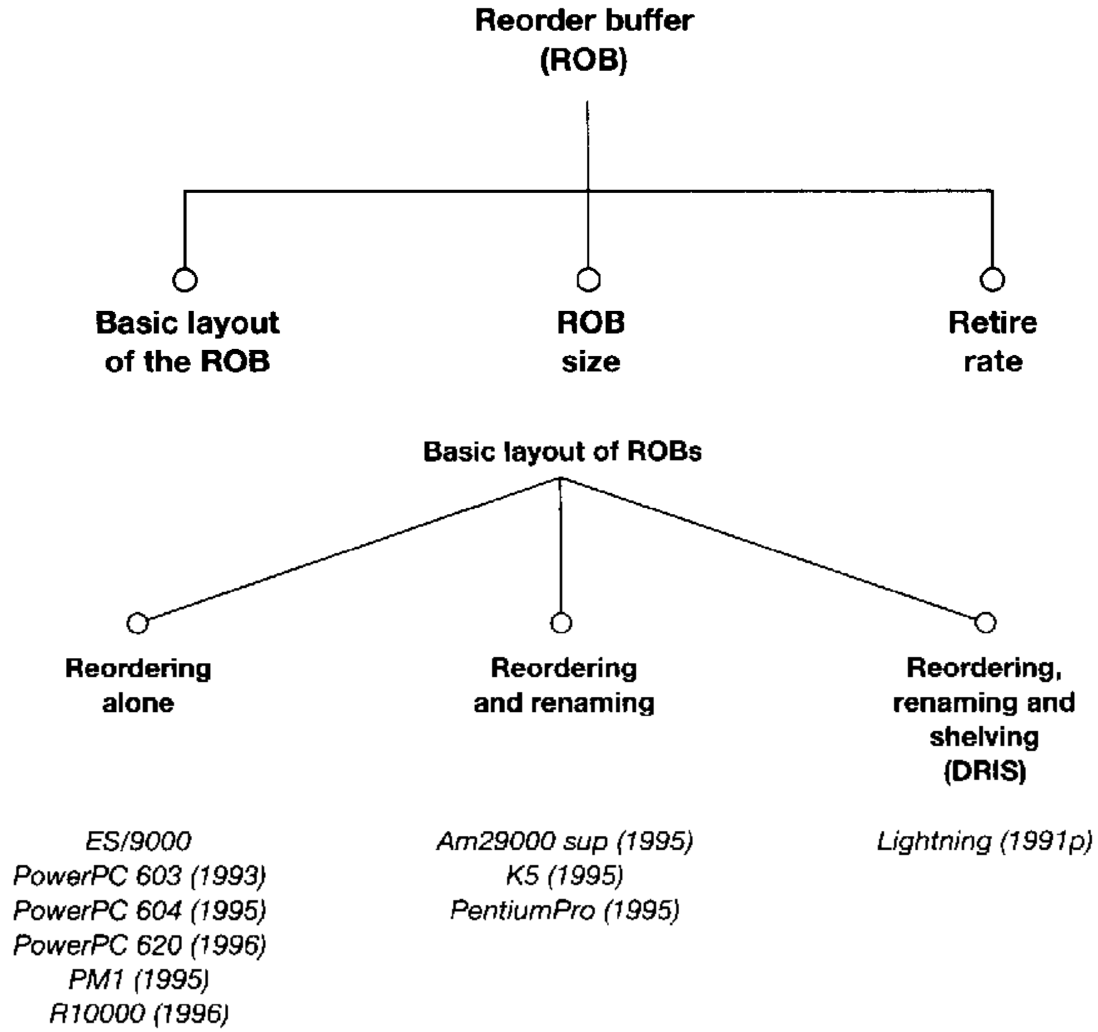
The ROB was first described in 1988 to solve the precise interrupt problem. Today, the ROB is used as a tool which assures sequential consistency of execution in the case of multiple EUs operating in parallel.
The ROB is a circular buffer with head and tail pointers. Instructions enter the buffer in program order and retire only if they have finished and all of their previous instructions have retired. Thus, sequential consistency is preserved by only allowing instructions to complete, i.e. to update the program state by writing their result into the referenced architectural register(s) or memory, in strict program order.

The ROB can effectively support both *speculative execution* and *interrupt handling*.


Head
(first free entry)
i_{n+k+3}
i_{n+k+2}
i_{n+k+1}
i_{n+k}
i
i
x
x
Allocate subsequently
issued instructions
to subsequent entries
in order
Free entries
Active entries
x
f
x
i_{n+2}
i_{n+1}
i_n
i_{n-1}
Instruction states:
i: Issued
x: In execution
f: Finished
An instruction may retire
if it has finished and
all prior instructions
have already retired
Tail
(next instruction to be
retired)

The Reorder Buffer for Speculation

In speculative execution, the processor carries on executing instructions in spite of an unresolved condition such as an unresolved conditional branch or a memory check. Later, when the condition is resolved, it becomes clear whether the speculatively executed instructions can be affirmed; if not, they have to be cancelled and the correct instructions executed. Each ROB entry can be extended to include a speculative status field which indicates if the corresponding instruction has been executed speculatively. Finished instruction are not allowed to retire while in a speculative state.


Reorder buffer
(ROB)
Basic layout
of the ROB
ROB
size
Retire
rate
Basic layout of ROBs
Reordering
alone
Reordering
and renaming
Reordering,
renaming and
shelving
(DRIS)
ES/9000
PowerPC 603 (1993)
PowerPC 604 (1995)
PowerPC 620 (1996)
PM1 (1995)
R10000 (1996)
Am29000 sup (1995)
K5 (1995)
PentiumPro (1995)
Lightning (1991p)

Preserving Sequential Consistency of Exception Processing

When instruction execute in parallel, *interrupt requests*, which are caused by exceptions arising during instruction execution, are also generated *out of order.* If these requests are acted upon imediately, interrupts occur out of order, that is, in a order that is different on a sequential processors. In this case, the sequential consistency of the interrupts is *weak*.

When imprecise interrupts occur, the processor is unable to reconstruct the correct state unless addition mechanisms are employed.

Sequential consistency of exception processing

Weak consistency

↓

Imprecise interrupts

Power1 (1990)
Power2 (1993)
α processors

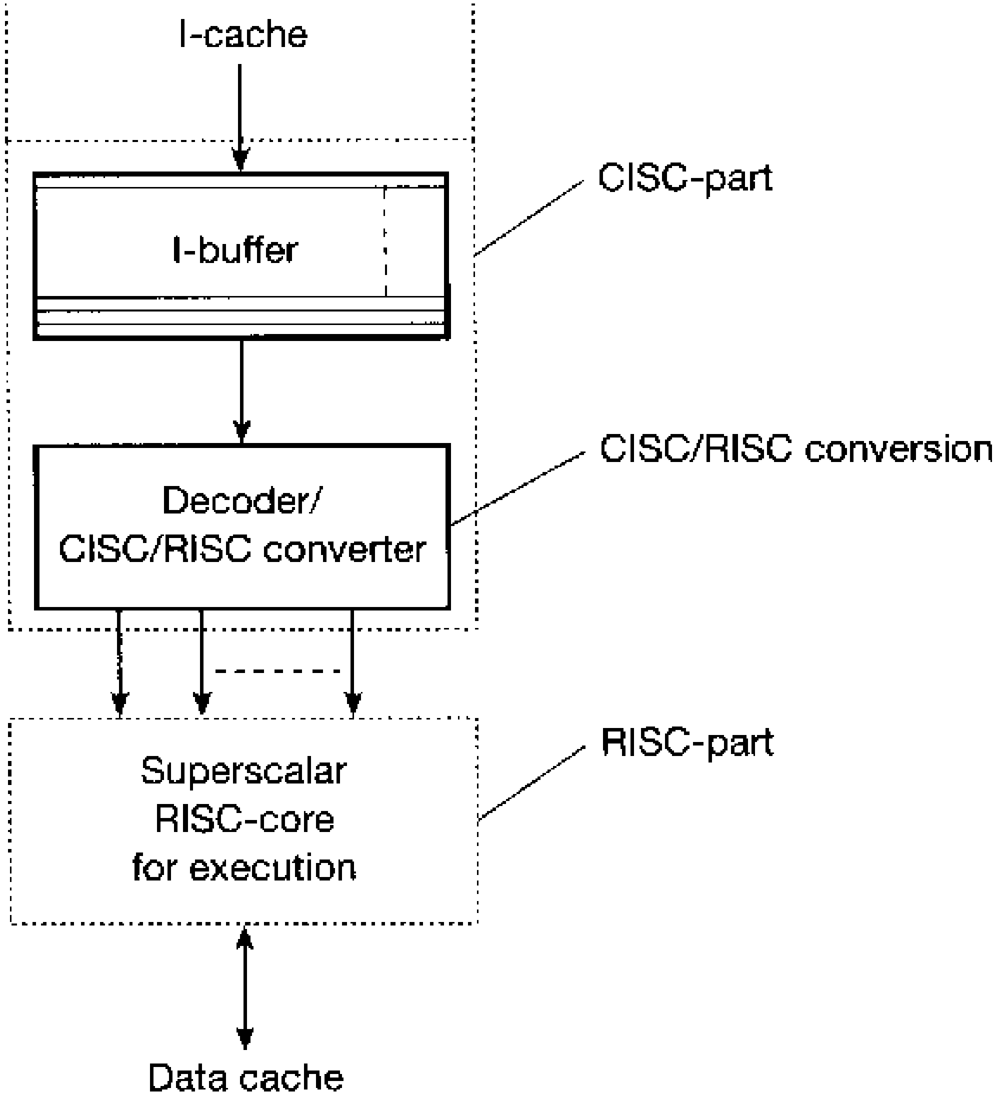
Strong consistency

↓

Precise interrupts

MC88110 (1993)
Pentium (1993) and usually processors making use of an ROB, such as:

ES/9000 (1992p)
PowerPC line
PA 8000 (1996)
R10000 (1996)


I-cache
I-buffer
CISC-part
Decoder/
CISC/RISC converter
CISC/RISC conversion
Superscalar
RISC-core
for execution
RISC-part
Data cache

Summary

- Very Long Instruction Word (VLIW) Processors
- Superscalar Processors
  - Parallel Decoding
  - Instruction Issue
    - Shelving
    - Register Renaming
  - Parallel Instruction Execution
  - Preserving the sequential consistency of execution and of exceptions.
    - The Reorder Buffer