Superscalar Instruction Issue

Dezsõ Sima

Kandó Polytechnic, Budapest

Before choosing parallel instruction issue to increase performance, we must identify the important design aspects and choices. DS trees can help to concisely represent the design space of instruction issue.

Clearly, instruction issue and execution are closely related: The more parallel the instruction execution, the higher the requirements for the parallelism of instruction issue. Thus, we see the continuous and harmonized increase of parallelism in instruction issue and execution.

This article focuses on superscalar instruction issue, tracing the way parallel instruction execution and issue have increased performance. It also spans the design space of instruction issue, identifying important design aspects and available design choices. The article also demonstrates a concise way to represent the design space using DS trees (see the related box), reviews the most frequently used issue schemes, and highlights trends for each design aspect of instruction issue.[1]

Processor evolution

Von Neumann processors evolved by and large in two respects. One reflects the technological improvements, which are capped by increasing clock rates. The second is the functional evolution of processors that came about primarily by raising the degree of parallelism in internal operations—first of the issue and instruction execution. Processor function evolved in three consecutive phases. First were the traditional von Neumann processors, which are characterized by both sequential issue and sequential instruction execution, as depicted in Figure 1.

The chase for more performance then gave rise to the introduction of parallel instruction execution. Designers introduced parallel instruction execution using one of two orthogonal concepts: multiple (nonpipelined) execution units (execution units) or pipelining. As a result, instruction-level parallel (ILP) processors emerged.

Because early ILP processors used sequential instruction issue, processors arriving in the second phase of the evolution were scalar ILP processors. Subsequently, the degree of parallel execution rose even further through use of multiple pipelined execution units. While increasing the execution parallelism, designers soon reached the point where sequential instruction issue could no

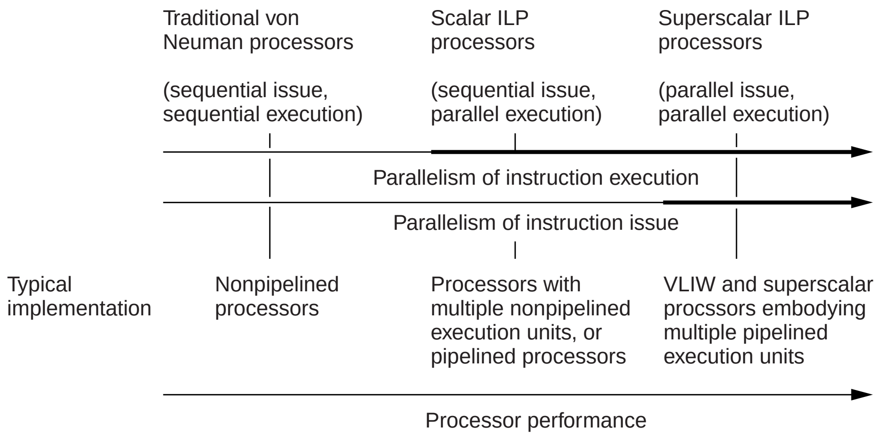

Traditional von Neuman processors
(sequential issue, sequential execution)
Scalar ILP processors
(sequential issue, parallel execution)
Superscalar ILP processors
(parallel issue, parallel execution)
Parallelism of instruction execution
Parallelism of instruction issue
Typical implementation
Nonpipelined processors
Processors with multiple nonpipelined execution units, or pipelined processors
VLIW and superscalar procssors embodying multiple pipelined execution units
Processor performance

Figure 1. Basic evolution phases of von Neumann processors. Thin portions of arrows indicate sequential operation; bold portions indicate parallel operation.

0272-1732/97/$10.00 © 1997 IEEE

DS trees

The concept of design space was introduced more than two decades ago.[1] Early interpretations used the design space mostly to represent a range of one or a few notable design parameters.[2] Later publications[3,4] began to use it more generally to represent all of the design aspects considered.

A design space should allow first of all the representation of the following basic design elements:

- orthogonal design aspects,
- design choices concerning a given design aspect, and
- concatenation of logically consecutive design choices or aspects.

DS trees are a convenient means to graphically represent design spaces in a consistent fashion.[5] If design space *A* involves the design aspects, say, of *B* and *C*, we can graph it as shown in Figure A.

If design aspect *B* gives two possible design choices, say b_1 and b_2, we can represent it as shown in Figure A2.

We can easily extend the introduced graphical elements for more than two design aspects or choices, as shown in Figure B. The basic components of a design tree may be concatenated to yield the DS tree representation of a design problem, item, and so on, as shown in Figure C.

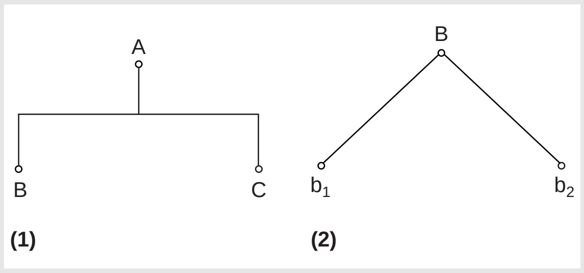


Figure A. Orthogonal design aspects (1) and alternative design choices for two possibilities (2).

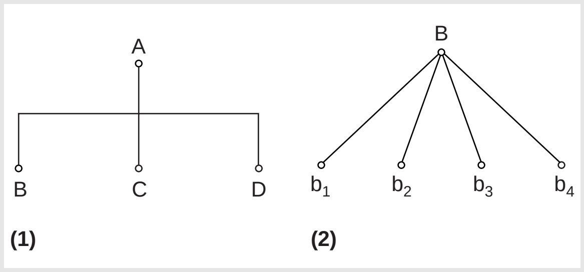


Figure B. Orthogonal design aspects (1) and alternative design choices (2) for more than two possibilities.

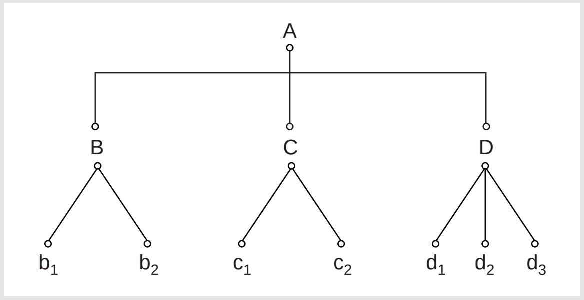


Figure C. Construction of a design tree by concatenating basic components.

References

1. M.R. Barbacci, “The Symbolic Manipulation of Computer Descriptions: ISPL Compiler and Simulator,” tech. report, Dept. of CS, Carnegie Mellon Univ., Pittsburgh, Penn., 1976.
2. L.S. Haynes et al., “A Survey of Highly Parallel Computing,” *Computer,* Vol. 15, No. 1, 1982, pp. 9-24.
3. M. Gfeller, “Walks Into the APL Design Space,” *APL Quote Quad,* Vol. 23, No. 1, July 1992, pp. 70-77.
4. D. Sima, “A Novel Approach for the Description of Computer Architectures,” DSc thesis, Hungarian Academy of Sciences, Budapest, 1993.
5. D. Sima, T.J. Fountain, and Kacsuk, *Advanced Computer Architectures,* Addison Wesley Longman, Harlow, England, 1997.

longer feed enough instructions to multiple pipelined execution units operating in parallel. Instruction issue became a bottleneck, as Flynn had foreseen long before.[2]

Thus the third phase in the von Neumann processor evolution became inevitable when designers replaced sequential instruction issue with parallel issue. This led to superscalar ILP processors. They were first implemented as statically scheduled VLIW architectures, which issue multioperation instructions.[3] Later more complex, dynamically scheduled superscalar processors appeared that were capable of issuing multiple instructions in each cycle.

Superscalar processors

Although the idea of superscalar issue was first formulated as early as 1970,[4] the concept gained attention only in the beginning of the 1980s.[5,6] IBM first implemented superscalar issue in a few experimental machines (Cheetah project around 1982-83, America project starting in 1985)[7] followed by DEC (Multititan project starting in 1985).[8] These and other research projects ultimately led to the commercial developments shown in Figure 2, next page.

The market includes different classes of superscalar processors with varying application fields, performance levels, and architectures. Two product lines are embedded: the Intel 960 and the Am 29000 superscalar processors. All other lines or models are of general use. These processors are typically intended for the high-performance desktop and workstation market. The low-cost, low-power PowerPC 602 and PowerPC 603 are exceptions.

Figure 2 (next page) shows most processors as RISCs. Only

Processor	1989	1990	1991	1992	1993	1994	1995	1996
Intel 960	960CA (3)[9]		960MM (3)[10]					960HA/HD/HT (3)[11]
Intel 80x86					Pentium (2)[12]		PentiumPro (~2)[13,14]	
IBM Power		Power1 (4)[13] (RS/6000)			Power2 (4)[14]			
IBM ES				ES/9000 (2)[15]				
PowerPC alliance					Power 601 (3)[16]		PowerPC 604 (4)[17]	PowerPC 620 (4)[19]
PowerPC					Power 603 (3)[20]		PowerPC 602 (2)[18]	
Motorola 8800					MC88110 (2)[21]			
Motorola 68000					MC68060 (3)[22]			
DEC Alpha				Alpha 21064 (2)[23,44]		Alpha 21064A (2)[23]	Alpha 21164 (4)[24]	
HP PA				PA7100 (2)[25]			PA 7200 (2)[26]	PA8000 (4)[27]
Sun/Hat/Sparc				SuperSparc (3)[28]			UltraSparc (4)[29]	
TRON Gmicro					Gmicro/500 (2)[30]		PM1(Sparc64) (4)[31]	
Mips R						R8000 (4)[32]		R10000 (4)[33,34]
AMD 29000							AM 29000 Sup (4)[35]	
AMD K5							K5 (~2)[36]	
Cyrix M1							M1) (2)[37]	
NexGen Nx						Nx586* (1/3)[38]		
Astronautics	ZS-1 (4)[39]							

Year

* Nx586 has scalar issue for CISC instructions but a three-way superscalar core for converted RISC instructions.

Figure 2. Overview and evolution of commercial superscalar processors, indicating the issue rate in parentheses. Sup indicates superscalar.

the Intel 80x86, ES/9000, MC68000, Gmicro, K5, M1, and the Nx lines are CISCs. Of these, the K5, M1, and Nx586 processors are compatible with x86s. They were developed to compete with the Pentium, which holds a major share of the microprocessor market.

The figure also indicates references to the superscalar processors discussed in this article. For better readability in the subsequent text, we omit any further reference to these processors. In this figure and in some subsequent figures, we refer to the first year of volume shipment of the processors.

Specific superscalar processing tasks

Superscalar processing breaks down into a number of specific tasks. Since superscalar processors issue multiple instructions every cycle, the first task necessarily is parallel decoding. Decoding in superscalar processors is a considerably more complex task than in the case of scalar processors and becomes even more sophisticated as the issue rate increases. Higher issue rates, however, can unduly lengthen the decoding cycle or can give rise to multiple decoding cycles unless decoding is enhanced. An increasingly common method of enhancement is predecoding. This partial decoding takes place in advance of common decoding, while instructions are loaded into the instruction cache. The majority of the latest processors use predecoding: the PowerPC 620, PA 7200, PA 8000, UltraSparc, and R10000.

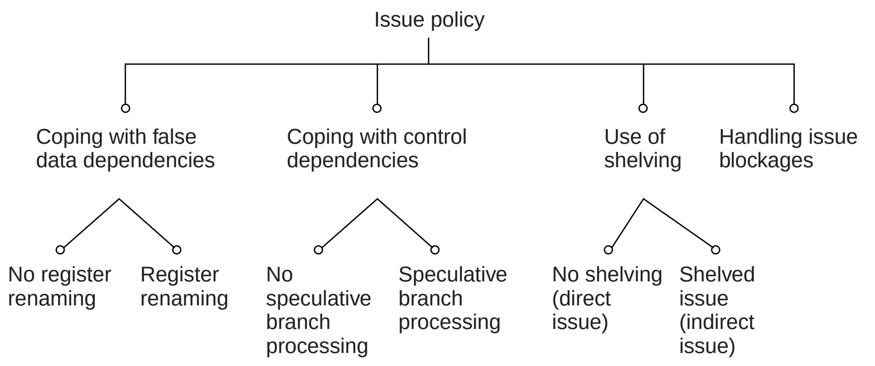
The most crucial task of superscalar processing is superscalar instruction issue. While a higher issue rate gives rise to higher processor performance, at the same time it amplifies the restrictive effects of control and data dependencies on the processor performance.[3] There are different reasons for this. Control dependencies occur more frequently per issue in a superscalar processor that issues multiple instructions in each cycle than they do in a scalar processor. The frequency increase is roughly proportional to the processor's issue rate.

Data dependencies also increase their frequency when the issue rate rises. However, this may strongly impede processor performance. For a pipelined scalar processor, designers may use a parallel optimizing compiler to avoid blockages due to dependencies in most cases. The compiler will fill otherwise-unused instruction slots, called bubbles, with independent instructions. Since for a scalar processor there are usually enough independent instructions available, a smart compiler can fill most of these slots. However, while increasing the issue rate, say from 1 to 4, the compiler needs more and more independent instructions to fill each issue slot. Very soon it reaches the point where far more independent instructions are needed than are available.

We may quantify this qualitative picture as follows. Assuming that a straightforward issue scheme is used, in a general-purpose program, a common parallel optimizing compiler will probably fill most issue slots with independent instructions for a low issue rate of 2. However, when the issue rate increases to more than 2, issue slots will be filled to an increasingly smaller degree due to the lack of sufficient independent instructions. In other words, while the issue rate is increased to over 2, the effective issue rate, reflecting the mean value of the instructions actually issued, will increase only slightly.

Therefore, to achieve higher performance, superscalar processors have introduced intricate instruction issue policies. These policies involve advanced techniques such as register renaming and shelving. As a consequence, the choice of instruction issue policy becomes crucial for achieving higher processor performance. We will revisit the topic of instruction issue in more detail later.

The next task, parallel instruction execution, is a precondition of superscalar processing. However, while instructions execute in parallel, instructions usually finish their execution out of order in respect to a sequentially operating processor. Because of this, specific means are needed to retain the logical consistency of program execution. This task is commonly called the preservation of the sequential consistency of instruction execution. Recent superscalar processors typically accomplish this by decoupling the generation of the results from writing them into the specified register or memory location. While the results are generated in parallel by the execution units, the program state is updated in a decoupled manner sequentially in program order. As superscalar processors tend to offer increasingly higher parallelism in the quest for higher performance, the task of preserving sequential consistency also becomes increasingly more demanding.



Issue policy
Coping with false data dependencies
Coping with control dependencies
Use of shelving
Handling issue blockages
No register renaming
Register renaming
No speculative branch processing
Speculative branch processing
No shelving (direct issue)
Shelved issue (indirect issue)

Figure 3. Design space of issue policies.

Finally, during instruction execution exceptions may arise. Here too sequential consistency is important—and gives rise to the task called preservation of sequential consistency of exception processing. The following focuses on the instruction issue.

The design space

We can discuss superscalar instruction issue more favorably within the framework of its design space. Superscalar instruction issue comprises two major aspects, issue policy and issue rate.

The issue policy specifies how dependencies are handled during the issue process. The issue rate, on the other hand, specifies the maximum number of instructions a superscalar processor can issue in each cycle.

Issue policies. The design space of issue policy is considerably complex. As shown in Figure 3, it consists of four major aspects. The first two specify how we can cope with false data and control dependencies during instruction issue. In these cases, designers may opt to eliminate dependencies during instruction issue by using register renaming and speculative branch processing, respectively.

The third aspect determines whether the advanced technique of shelving will drastically reduce issue blockages due to occurring dependencies. The final aspect specifies how to deal with issue blockages. This article briefly reviews the four design aspects; for a detailed description refer to Sima et al.[1] and Hwang.[40] It also presents the DS tree concept.

False data dependencies. Within this design space of issue policy, the first aspect determines how the processor copes with false data dependencies. False dependencies occur either between the instructions to be issued and those in execution, or among the instructions to be issued. Specifically, we are concerned with whether to eliminate WAR (write after read) and WAW (write after write) dependencies occurring between register references. Note that this aspect is confined to register data dependencies and does not cover possible false data dependencies involving memory data. The rationale therefore is that false data dependencies involving memory data are much less frequent than those between register data, especially in RISC architectures. Thus, recently used issue policies do not address them.

The principle of removing false data dependencies between register data by renaming is quite simple.[4,41] When encountering a WAW or WAR dependency, the processor renames the destination register causing the dependency. This means that the result of the instruction causing a WAW or WAR dependency is written not into the specified register, but into a dynamically allocated buffer.

Control dependencies. The second aspect of issue policies determines how to deal with control dependencies. Obviously, a conditional control transfer instruction causes serious problems during pipelined instruction processing. This problem is more severe for unresolved control dependencies—that is, with conditional transfer instructions if the condition has not yet been produced by the time the condition should be evaluated. Issue policies handle control dependencies in two different ways. Either they block instruction issue until determination of the execution path to be followed, or they employ speculative execution. With speculative execution of control transfer instructions, or speculative branch processing for short,[42] the processor estimates the outcome of each conditional transfer instruction. Instruction issue is then resumed accordingly.

Shelving or indirect issue. The third aspect of issue policy concerns the use of shelving, an efficient technique to avoid issue blockages. Invented more than three decades ago,[43] shelving was forgotten and “reinvented” at the beginning of the 1990s. Designers have a choice between direct issues causing frequent issue blockages due to dependent instructions or to avoid blockages through shelving.

Direct issue is a strict approach related to dependencies. This scheme checks the decoded instructions to be issued for dependencies, as depicted in Figure 4, next page. Independent instructions move directly to the execution units, while occurring dependencies block instruction issue.

Checking for dependencies takes place in a so-called issue window. The issue window covers the next *n* instructions to be issued, where *n* is the issue rate. Actually, the issue window comprises the last *n* entries of the instruction buffer. In the absence of dependencies, all instructions in the window are issued directly to the execution units. However, when dependencies occur, the issue blockage will be handled as specified by the corresponding aspect of the design space, as discussed later.[8] Suffice it to say at this point that use of the blocking issue mode heavily impedes issue performance. A more advanced technique is to employ shelving.

Shelved issue (also called indirect issue) decouples instruction issue and dependency checking. This technique presumes that special instruction buffers, often referred to as reservation stations, are provided in front of the execution units. With shelving, instructions are issued first to the shelving buffers with essentially no need for dependency checks, as illustrated in Figure 5.

Other possibilities for the implementation of shelving buffers exist.[1] Note that even if shelving is used, certain hardware constraints—such as smaller than needed bus widths or a lack of free entries in the shelving buffers—may continue to cause issue blockages.

Shelving delays dependency checking to a later step of processing called dispatching. During dispatching, the processor checks the instructions held in the shelving buffers for dependencies, and forwards dependency-free instructions to available execution units. From another point of view, shelved instructions make up a dispatch window. Thus shelving substantially widens the window that is scanned for executable instructions. Clearly, this helps to raise processor performance.

Note that the two different terms, instruction issue and dispatch, express different actions. Furthermore, *issue* is used in two different interpretations. Without shelving—that is, when the direct issue is used—*issue* refers to the action of disseminating decoded independent instructions to the execution units. When shelving is used, *issue* designates the dissemination of decoded instructions to the shelving buffers. *Dispatch*, on the other hand, is only used in the case of shelving. Here, *dispatch* designates the dissemination of dependency-free instructions from the shelving buffers to the execution units. While useful, this clear distinction is not common in the literature, which uses both *issue* and *dispatch* in either interpretation.

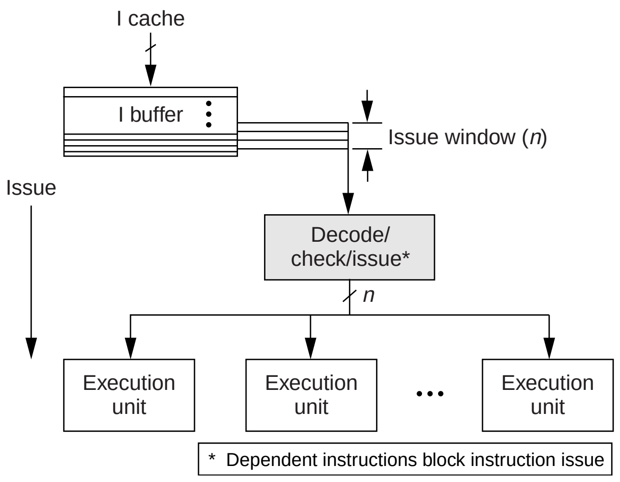
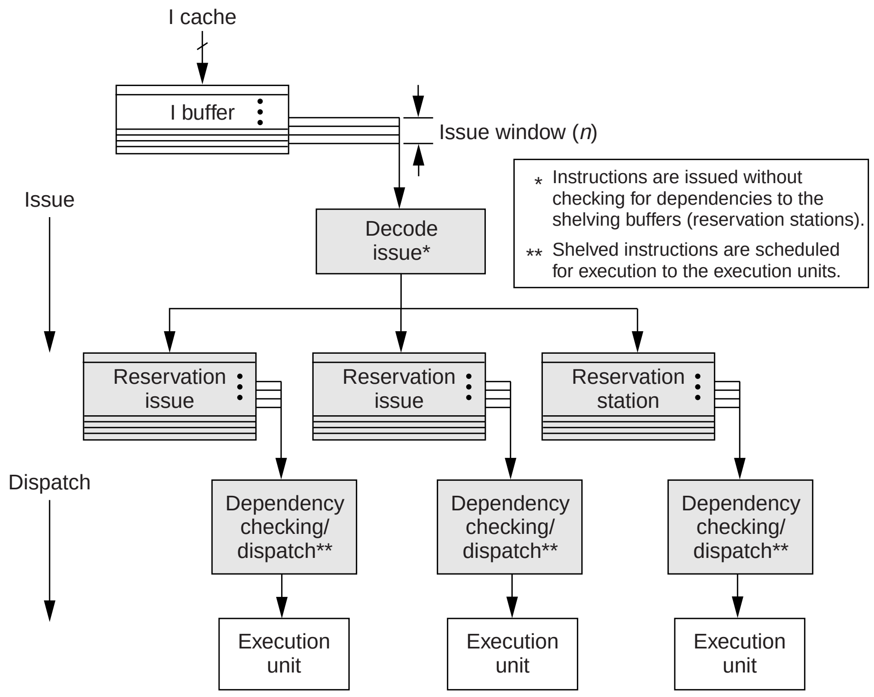


Figure 4. Principle of the blocking issue mode, assuming a four-instruction-wide issue window.



I cache
I buffer
Issue window (*n*)
Issue
Decode issue*
* Instructions are issued without checking for dependencies to the shelving buffers (reservation stations).
** Shelved instructions are scheduled for execution to the execution units.
Reservation issue
Reservation issue
Reservation station
Dispatch
Dependency checking/ dispatch**
Dependency checking/ dispatch**
Dependency checking/ dispatch**
Execution unit
Execution unit
Execution unit

Figure 5. The principle of shelving, assuming that dedicated buffers (reservation stations) are available in front of each execution unit.

Handling issue blockages. The last aspect of the design space of instruction issue declares how occurring issue blockages are handled. We can break this down into two aspects. The first, called issue order, specifies whether in case of issue blockage the program order is retained. The second aspect is the alignment of instruction issue. It determines whether a fixed or gliding issue window is used.

Figure 6 illustrates two possible schemes for issue order, which may be followed when issue is blocked. In the first scheme, called in-order issue, instruction issue preserves the program order. Then a dependent instruction blocks the issue of all subsequent instructions until the dependency is resolved.

However, restricting subsequent independent instructions from issue can severely impede performance. Therefore, a few superscalar processors that employ the direct issue mode, such as the MC88110 and the PowerPC 601, have introduced out-of-order issue. This scheme allows independent instructions to be issued from the issue window even if a dependent instruction has not yet been issued. In both processors, out-of-order issue is only partially allowed for particular instruction types. For instance, the PowerPC 601 issues branches and, in an out-of-order style, floating-point instructions, whereas the MC88110 does so only for floating-point instructions.

At first sight it is surprising how few superscalar processors use out-of-order issue. There are two reasons for this. First, preserving sequential consistency for out-of-order issue requires a much higher effort than does in-order issue. Second, processors with shelving have almost no motivation to use out-of-order issue. With shelving, the issue of instructions is rarely blocked, as discussed earlier, and out-of-order issue would only have a marginal benefit. As a consequence, we expect mainstream superscalar processors to continue to employ in-order issue.

The second aspect of handling issue blockages determines the alignment of instruction issue. As Figure 7 and Table 1 indicate, an aligned instruction issue requires a fixed window. This means that no instructions of the next window are considered as candidates for issue until all instructions in the current window have been issued. By its nature, the point of issue alignment is relevant only for superscalar processors.

Aligned instruction issue is typical for the first generation of superscalar processors, such as the i960CA and Power1. But in the direct-issue mode, aligned issue considerably reduces the effective issue rate. Therefore, a number of subsequent superscalar processors that still use the direct-issue mode introduced unaligned instruction issue. Then a gliding window is employed whose width equals the issue rate. In each cycle, all instructions in the window are checked for dependencies. Independent instructions are issued

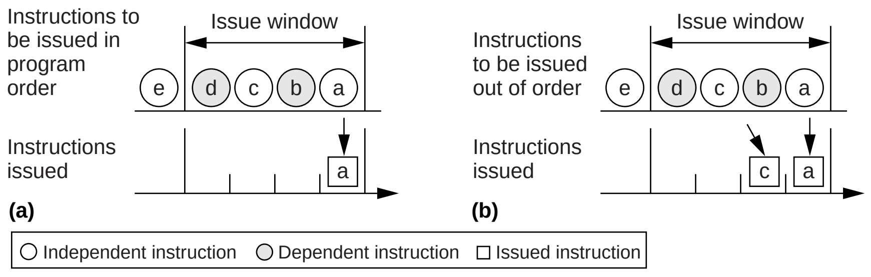


Figure 6. Issue order of instructions: in-order issue (a) used in most superscalar processors and out-of-order issue (b) employed for particular data types in the MC88110 and in the Power PC 601. Figure assumes a four-way superscalar processor with aligned issue.

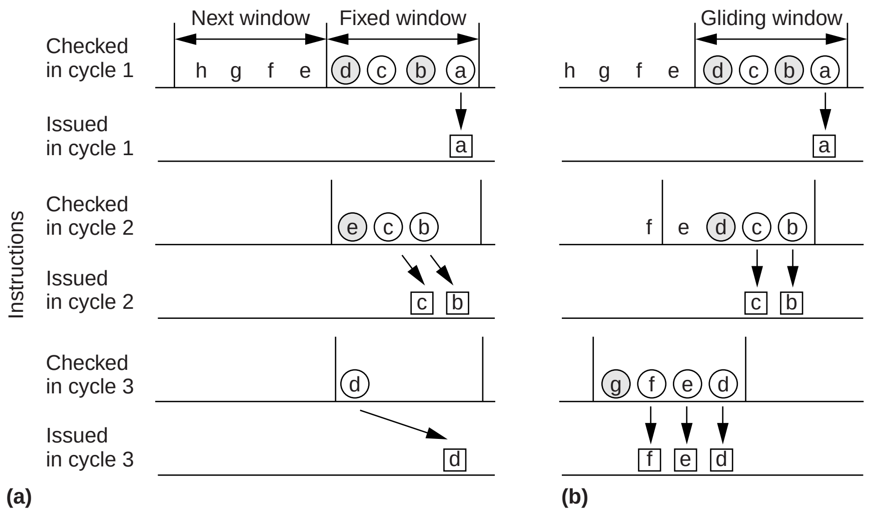


Figure 7. Contrasting aligned and unaligned issue of instructions: aligned issue from a fixed window (a), unaligned issue from a gliding window (b), and processor examples (c). Figure assumes a four-issue, in-order superscalar processor.

Table 1. Typical superscalar processors using aligned and unaligned instruction issue. Year of first appearance in parentheses.

First-generation/recent processors using shelving	Follow-on processors not using shelving
i9609CA (1989)	
Power1 (1990)	
PA 7100 (1992)	
SuperSparc (1992)	M88110 (1993)
	M68060 (1993)
Alpha 21064 (1992)	PA 7100LC (1993)
PowerPC 603 (1993)	R8000 (1994)
Alpha 21064A (1994)	PA 7200 (1995)
PowerPC 604 (1994)	UltraSparc (1995)
Alpha 21164 (1995)	
PM1/Sparc64 (1995)	
R10000 (1995)	
PowerPC 620 (1996)	

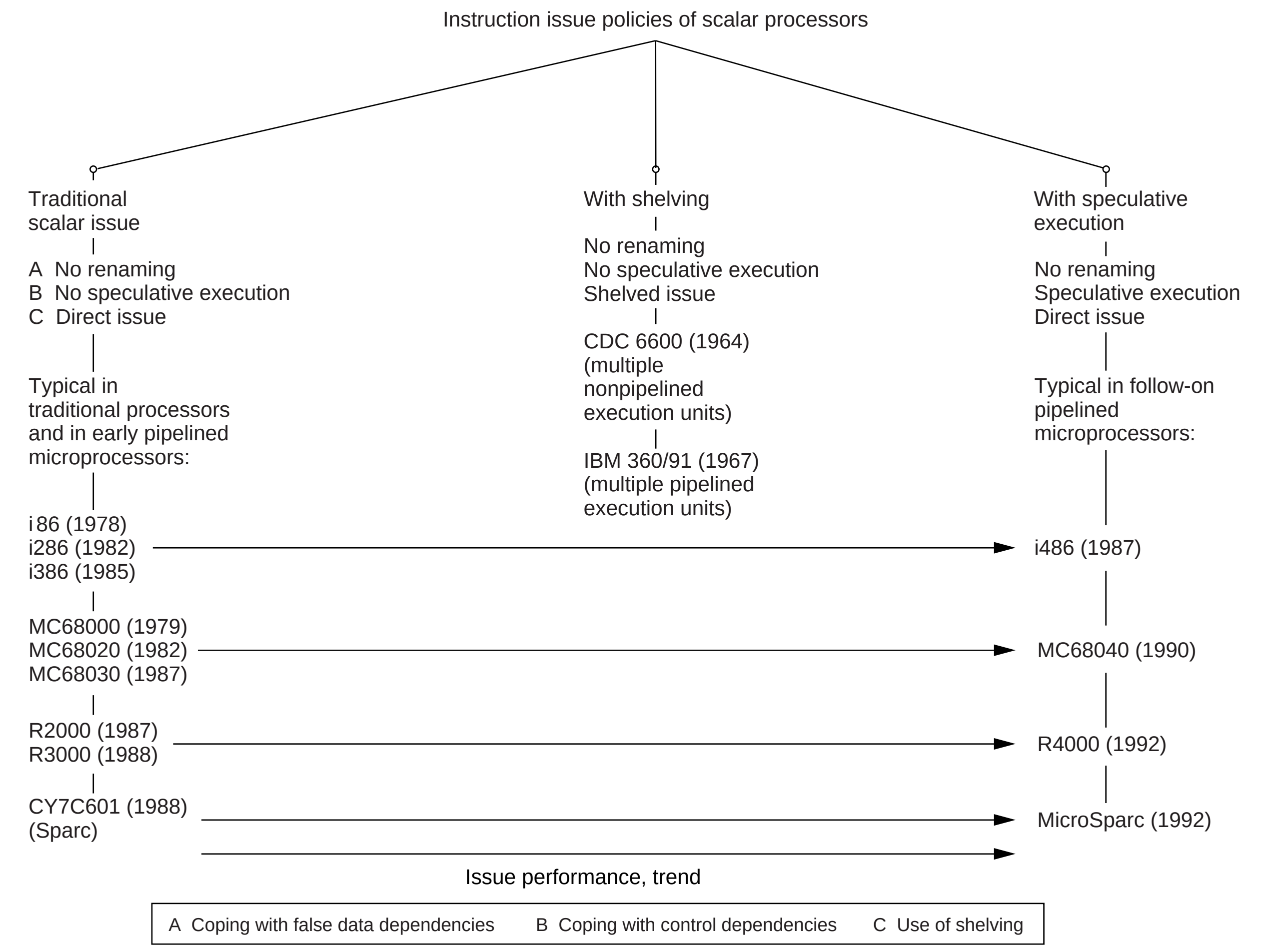

Instruction issue policies of scalar processors
Traditional scalar issue
A No renaming
B No speculative execution
C Direct issue
Typical in traditional processors and in early pipelined microprocessors:
i86 (1978)
i286 (1982)
i386 (1985)
MC68000 (1979)
MC68020 (1982)
MC68030 (1987)
R2000 (1987)
R3000 (1988)
CY7C601 (1988) (Sparc)
With shelving
No renaming
No speculative execution
Shelved issue
CDC 6600 (1964) (multiple nonpipelined execution units)
IBM 360/91 (1967) (multiple pipelined execution units)
With speculative execution
No renaming
Speculative execution
Direct issue
Typical in follow-on pipelined microprocessors:
i486 (1987)
MC68040 (1990)
R4000 (1992)
MicroSparc (1992)
Issue performance, trend
A Coping with false data dependencies
B Coping with control dependencies
C Use of shelving

Figure 8. Most frequently used instruction issue policies of scalar processors.

from the window either in an in-order or in an out-of-order fashion. After instruction issue, however, the window is refilled. We could say, it is shifted along the instruction stream by as many instructions as issued in the last cycle.

Unaligned instruction issue is typical for the second wave of superscalar processors that employ the direct-issue mode: the MC88110, MC68060, PA 7200, R8000, and UltraSparc. However with the introduction of shelving, the motivation for using unaligned instruction issue has strongly diminished. This is understandable, since shelving delays dependency checks until instruction dispatching. Thus, while using shelving, instructions can be issued with almost no restrictions to the shelving buffers. The only restrictions that remain relate to trivial hardware requirements such as availability of shelving buffers, buses, and so on. As a consequence, recent superscalar processors that employ shelving have returned to aligned instruction issue. Examples of these are the PowerPC 603, PowerPC 604, PowerPC 620, and R10000.

The Alpha line of processors is somewhat a special case in this regard. All the Alpha processors mentioned employ aligned issue, even the recent Alpha 21164.

In summary, unaligned instruction issue increases the performance of superscalar processors employing the direct-issue mode. However, with the introduction of shelving, unaligned instruction issue has lost its rationale, and the most recently introduced superscalar processors use aligned issue.

Most frequently used issue policies. The following surveys the most frequently used issue policies, in three scenarios: scalar processors, superscalar processors, and the broad picture covering both.

By taking into account that the handling of issue blockages covers two subaspects (issue order and alignment) altogether, the design space of instruction issue covers five aspects. Each of the five represent a binary decision. Any combination of the design choices results in a possible issue policy for a total of 2^5 = 32 possible issue policies. Clearly, the possible issue policies are not all of equal importance.

While considering most frequently used issue policies, we can reduce the design space of instruction issue by neglecting less important aspects. First, for both scalar and superscalar processors we can ignore issue order, since most processors employ an in-order issue. Furthermore, we can omit issue alignment in the case of scalar and superscalar processors that make use of shelving, as explained earlier.

However, for superscalar processors using the direct-issue mode, we must still consider the alignment of instruction issue. Thus, for scalar processors we have to consider three, and for superscalar processors, four major issue aspects.

Scalar processors. While discussing instruction issue policies for scalar processors, we must consider the three basic issue aspects, that is, whether to employ renaming, speculative execution, and shelving.

Of the resulting eight possible issue policies, scalar processors use two predominantly, as indicated in Figure 8. They are the traditional scalar issue and its enhancement with speculative execution. In addition, there is a further policy of historical importance: the traditional scalar issue with shelving.

Early nonpipelined processors issue instructions in sequence. This means that the next instruction was issued only after the previous one had been completed. Obviously, these processors did not use renaming, speculative branch processing, or shelving. Our classification designates this issue mode as the traditional scalar issue policy. Early pipelined microprocessors also employed this issue policy. Examples are the i86, i286, i386, the MC68000, MC68020, and MC68030.

Later, when ILP processors—processors with multiple execution units or pipelined processors—emerged, the traditional scalar issue policy proved too restrictive on performance. Two approaches relieved this. One was introduced in the CDC6600, the other in the IBM 360/91. The CDC6600, a supercomputer of its time, was one of the first ILP processors. It achieved high performance by introducing multiple nonpipelined, parallel working execution units. This processor relaxed the instruction issue bottleneck by enhancing the traditional issue scheme with shelving.

The IBM 360/91 pioneered the other approach. This processor was a more advanced ILP processor built of multiple pipelined floating-point execution units. It also employed a shelving for floating-point instructions. In addition, the 360/91 performed a kind of register renaming to assure the logical integrity of the program execution. However, this type of renaming did not remove false data dependencies. Although these innovations contributed significantly to the high performance of the processors mentioned, the associated exceptionally high hardware cost was at that time prohibitive for their widespread use. Thus, almost a quarter of a century passed before processors again employed shelving.

Instead, the spread of pipelining determined the main evolution line of the issue scheme. In pipelined processors, conditional branches, especially unresolved conditional branches, can considerably reduce performance in the direct-issue mode, as discussed earlier. Obviously, the performance cutback increases as the number of pipeline stages increases. Therefore, more advanced pipelined microprocessors, which often were implemented with an increased number of stages, inevitably enhanced the traditional scalar issue with speculative execution, as indicated in Figure 8. Examples of this type are the i486, MC68040, R4000, and MicroSparc.

Superscalar processors. When we expand our discussion of instruction policies to superscalar processors; we must consider more than the three basic issue aspects and add issue alignment. This results in 12 feasible issue policies. Of these, superscalar processors employ mainly five. As Figure 9 shows, they are designated as the straightforward superscalar issue with and without issue alignment, the straightforward superscalar issue with shelving, the straightforward superscalar issue with renaming, and the advanced superscalar issue.

One commonly used instruction policy is advanced superscalar issue. At present, this policy is the most relevant one.

The simplest, often-used policy is the straightforward aligned superscalar issue. It does not permit renaming, uses aligned direct issue, and employs speculative execution. According to this policy, instruction issue is blocked for data and resource dependencies, whereas control dependencies are met with speculative branch processing. Furthermore, it uses the aligned issue. This simple scheme is widely used in first-generation superscalar processors, such as the Pentium, PA7100, PowerPC 601, and Alpha 21064. Note that beyond first-generation superscalar processors, a few later members of well-known processor families such as the Alpha 21164 also retained this straightforward issue policy.

Another group of processors makes use of the straightforward superscalar issue, in a more advanced form, with alignment free issue. This additional feature contributes to increased performance. Examples of processors using this policy are the R8000, PA 7200, and UltraSparc.

The next issue policy—the straightforward superscalar issue policy with shelving—does not employ renaming, handles control dependencies with speculative execution, and introduces shelving. This scheme is used only rarely in a few superscalar processors, such as the MC88110 and R8000. Both these processors accommodate only partial shelving. The MC88110 shelves only stores and conditional branches, whereas the R8000 shelves only floating-point instructions.

As its name implies, the straightforward superscalar issue policy with renaming, the fourth scheme, offers renaming and speculative branch processing but sticks at direct issue. Only a few recently introduced processors, such as the M1 and PowerPC 602 employ this policy. The PowerPC 602 in addition is not a true superscalar processor. It actually has a scalar issue with some enhancements, which appears only in certain situations as a superscalar processor.

The final policy commonly used is advanced superscalar issue. At present, this policy is the most relevant one. It employs register renaming, speculative branch processing, and shelving. Here, register renaming eliminates false register data dependencies, speculative execution copes with control dependencies, and shelving removes issue blockages due to dependencies.

Obviously, this issue policy is the most advanced in the framework of the design space considered. The most recent superscalar processors employing this scheme include the R10000, Pentium Pro, PowerPC 603, PowerPC 620, and a number of other recent processors.

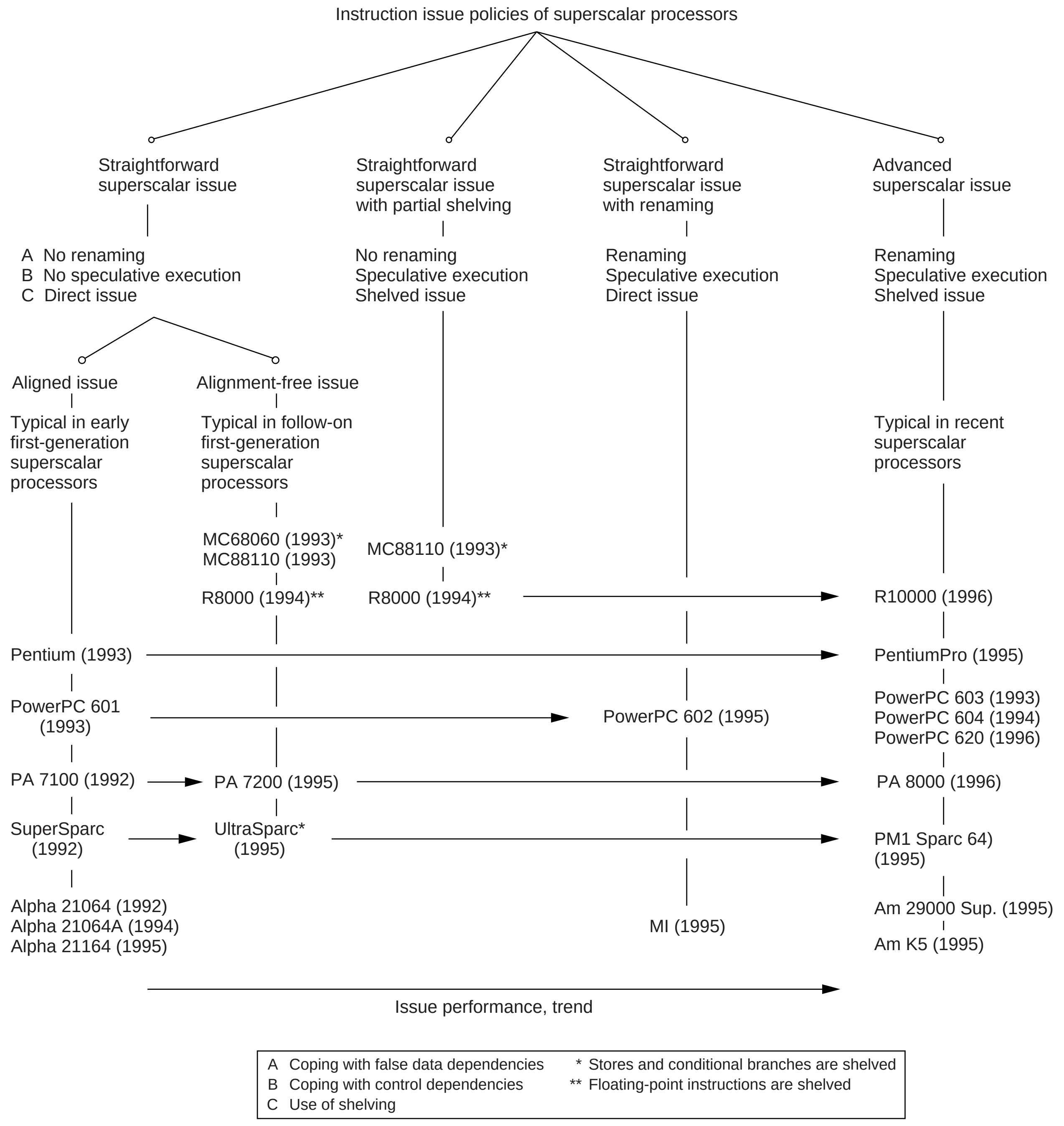


Figure 9. Most frequently used instruction issue policies of superscalar processors.

Trend in instruction issue policies. Figure 10 summarizes the most frequently used issue policies in scalar and superscalar processors. Clearly, all major processor families show the same evolution path. This path is characterized by the following sequence of issue policies: traditional scalar issue, traditional scalar issue with speculative execution, straightforward superscalar issue with and without issue alignment, and advanced superscalar issue.

The performance of issue policies increases more or less in the same sequence as they have been discussed. Figure 10 shows the issue policies used in subsequent members of important processor families.

Issue rate. The second major aspect of the design space of instruction issue is the issue rate (the degree of superscalarity). This rate refers to the maximum number of instructions a superscalar processor can issue in the same cycle. Superscalar operation may be implemented by issuing two, three, or more instructions in each cycle. Evidently, a high-

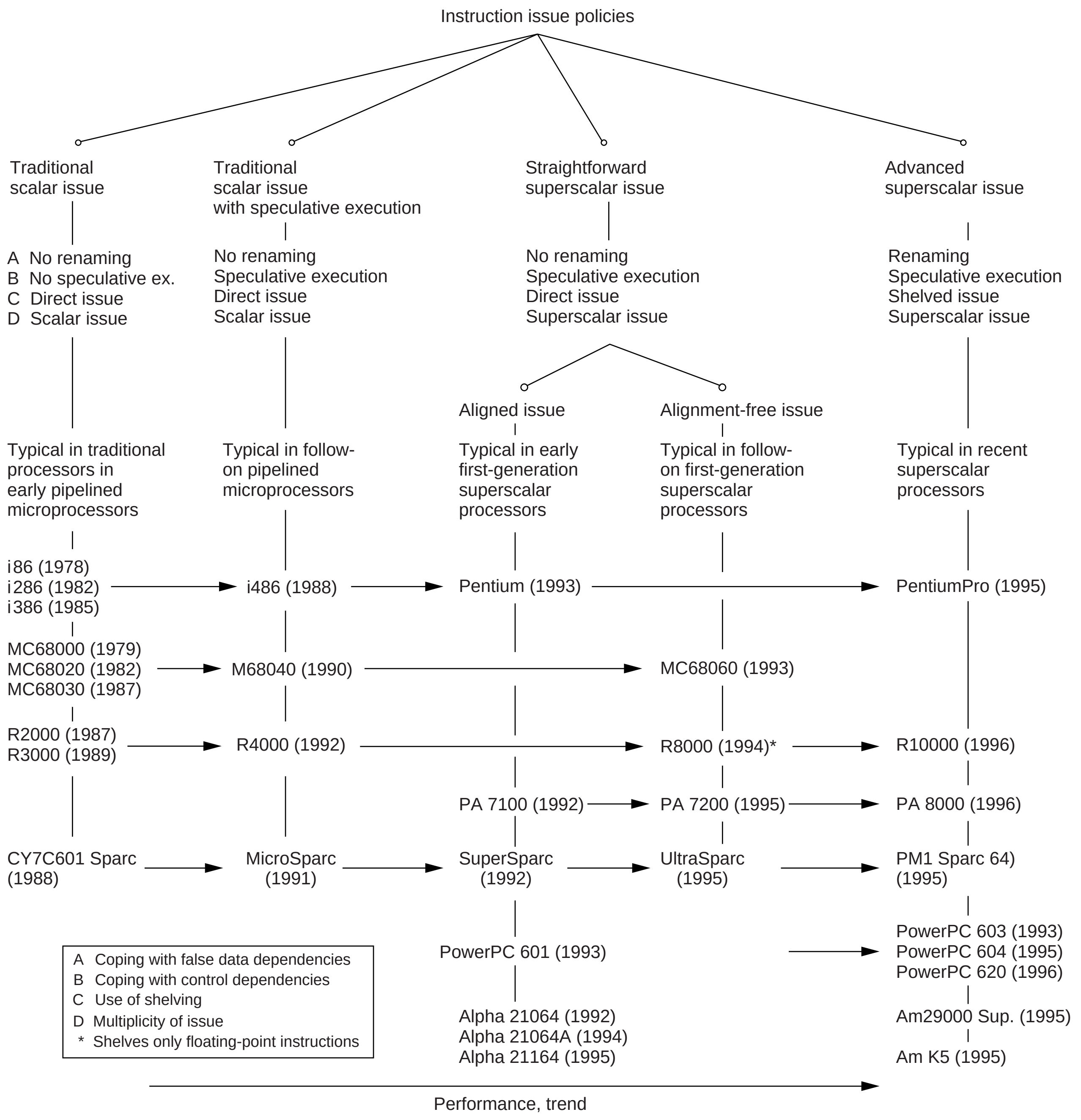


Figure 10. Most frequently used instruction issue policies and their trend.

er issue rate offers a higher performance potential, however its implementation is more complex. Superscalar CISC processors are usually restricted to issuing only two, or about two, instructions per cycle. In contrast, early superscalar RISC processors typically issued two or three instructions in each cycle. Recent RISC processors typically issue four instructions per cycle, as shown in Figure 11.

This figure exposes the evolution of the issue rate experienced in five particular RISC lines. First Alpha processors (21064 and 21064A) as well as the first superscalar members of the HP Precision-Architecture line (PA 7100, PA 7200) were restricted to issuing two instructions per cycle. In contrast, the first members of the PowerPC line (PowerPC 601, PowerPC 603) and the SuperSparc issued three instructions per cycle. For the time being, all upcoming models of the lines mentioned except the Power 2 are four-issue superscalar processors. The Power 2 issues six instructions per cycle and is now the issue-rate leader.

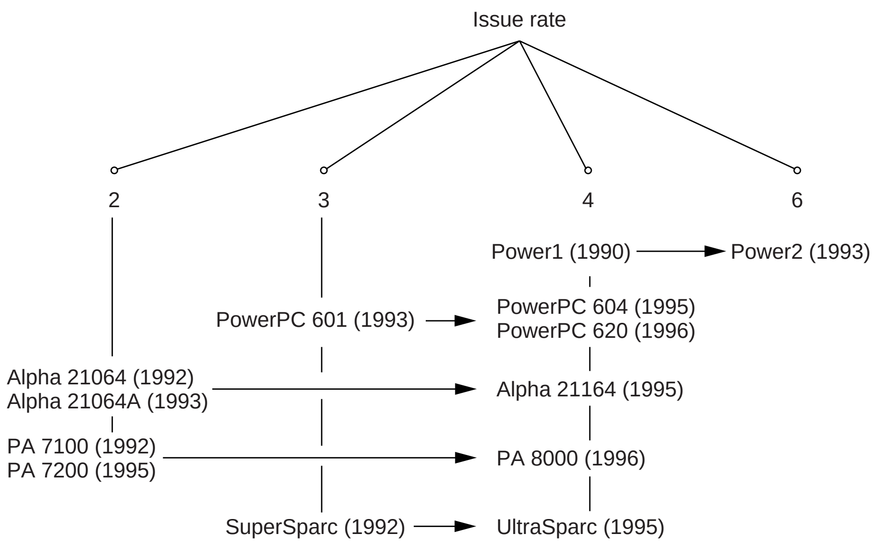


Figure 11. Evolution of the issue rate in RISC processors. This figure does not include low-cost, low-performance models such as the PowerPC 603 (1993), PA7100LC (1993), and PowerPC 602 (1995), since they follow a different design philosophy than the performance oriented models considered here.

IN THE PAST FEW YEARS instruction issue evolved dramatically in main line processors. Remarkably, the instruction issue policy in all significant processor families followed the same evolution path. The traditional scalar issue was first enhanced with speculative branch processing, then the straightforward superscalar issue policy emerged. That finally gave way to the advanced superscalar issue policy.

The advanced superscalar issue policy came into widespread use virtually in all recent models providing speculative branch processing, renaming, and shelving. Speculative branch processing meets a performance degradation due to control transfer instructions, and renaming removes issue blockages caused by false data dependencies. Shelving widens the instruction window that is scanned in each cycle for executable—that is, not data-dependent—instructions.

Two implications follow the use of the enhanced superscalar issue policy. First, the processor executes shelved instructions essentially on a dataflow basis. Second, assuming a given execution and memory bandwidth, the branch prediction's hit rate and the dispatch window's width become the main constraints on processor performance.

Acknowledgment

Addison Wesley Longman Ltd. granted permission to reprint the figures of this article, which were originally published in Sima, Fountain, and Kacusk,[1] Chapters 4 and 7.

References

1. D. Sima, T.J. Fountain, and Kacusk, "Advanced Computer Architectures," Addison Wesley Longman, Harlow, England, 1997.
2. M.J. Flynn, "Very High Speed Computing Systems," *Proc. IEEE,* Vol. 54, Dec. 1966, pp. 1901-1909.
3. B.R. Rau and J.A. Fisher, "Instruction Level Parallel Processing: History, Overview and Perspective," *J. Supercomputing,* Vol. 7, 1993, pp. 9-50.
4. G.S. Tjaden and M.J. Flynn, "Detection and Parallel Execution of Independent Instructions," *IEEE Trans. Computers,* Vol. C-19, No. 10, 1970, pp. 889-895.
5. H.C. Torng, "Arithmetic Engines in the VLSI Environment," Tech. Reports EE-CEG-82-7 and EE-CEG-83-3, School of Elec. Eng., Cornell Univ., Ithaca, N. Y., Oct. 1982 and July 1983.
6. R.D. Acosta et al., "An Instruction Issuing Approach to Enhancing Performance in Multiple Functional Unit Processors," *IEEE Trans. Computers,* Vol. C-35, No. 9, Sept. 1986, pp. 815-828.
7. G.F. Grohoski, "Machine Organization of the IBM RISC System/6000 Processor," *IBM J. Res. Development,* Vol. 34, No. 1, 1990, pp. 37-58.
8. N.P. Jouppi et al., "A Unified Vector/Scalar Floating-Point Architecture," *Proc. Third Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* 1989, pp. 134-143.
9. S. McGeady, "The i960CA SuperScalar Implementation of the 80960 Architecture," *Proc. Compcon,* 1990, pp. 232-239.
10. S. McGeady et al., "Performance Enhancements in the Superscalar i960MM Embedded Microprocessor," *Proc. Compcon,* 1991, pp. 4-7.
11. B. Case, "Intel Reveals Next-Generation 960 H-Series," *Microprocessor Report,* Vol. 8, No. 13, 1994, pp. 11-15.
12. D. Alpert and D. Avnon, "Architecture of the Pentium Microprocessor," *IEEE Micro,* June 1993, pp. 11-21.
13. G.F. Grohoski, "Machine Organization of the IBM RISC System/6000 Processor," *IBM J. Res. Development,* Vol. 34, No. 1, 1990, pp. 37-58.
14. L. Gwennap, "Intel's P6 Uses Decoupled Superscalar Design," *Microprocessor Report,* Vol. 9, No. 2, pp. 9-15.
15. J.S. Liptay, "Design of the IBM Enterprise Sytem/9000 High-End Processor," *IBM J. Res. Dev.,* Vol. 36, No. 4, July 1992, pp. 713-731.
16. M. Becker et al., "The PowerPC 601 Microprocessor," *IEEE Micro,* Oct. 1994, pp. 54-68.
17. S.P. Song et al., "The PowerPC 604 RISC Microprocessor," *IEEE*

Micro, Oct. 1994, pp. 8-17.
18. D. Levitan et al., "The PowerPC 620 Microprocessor: A High Performance Superscalar RISC Microprocessor," *Proc. Compcon,* 1995, pp. 285-291.
19. D. Ogden et al., "A New PowerPC Microprocessor for Low Power Computing Systems," *Proc. Compcon,* 1995, pp. 281-284.
20. B. Burgess et al., "The PowerPC 603 Microprocessor," *Comm. ACM,* Vol. 37, pp. 34-42.
21. K. Diefendorff and M. Allen, "Organization of the Motorola 88110 Superscalar RISC Microprocessor," *IEEE Micro,* Apr. 1992, pp. 40-63.
22. J. Circello et al., "The Superscalar Architecture of the MC68060," *IEEE Micro,* Apr. 1995, pp. 10-21.
23. *DECchip 21064 and DECchip 21064A Alpha AXP Microprocessors Hardware Reference Manual,* DEC, Maynard, Mass. 1994.
24. J.H. Edmondson et al., "Superscalar Instruction Execution in the Alpha Microprocessor," *IEEE Micro,* Apr. 1995, pp. 33-43.
25. T. Asprey et al., "Performance Features of the PA7100 Microprocessor," *IEEE Micro,* June 1993, pp. 22-35.
26. G. Kurpanek et al., "PA-7200: A PA-RISC Processor with Integrated High Performance MP Bus Interface," *Proc. Compcon,* 1994, pp. 375-382.
27. L. Gwennap, "PA-8000 Combines Complexity and Speed," *Microprocessor Report,* Vol. 8, No. 15, 1994, pp. 1-8.
28. G. Blanck and S. Krueger, "The SuperSPARC Microprocessor," *Proc. Compcon,* 1992, pp. 136-141.
29. D. Greenley et al., "UltraSparc: The Next Generation Superscalar 64-bit SPARC," *Proc. Compcon,* 1995, pp. 442-451.
30. K. Kchiyard et al., "The Gmicro/500 Superscalar Microprocessors with Branch Buffers." *IEEE Micro,* Oct. 1993, pp. 12-22.
31. N. Patkar et al., "Microarchitecture of HAL's CPU," *Proc. Compcon,* 1995, pp. 259-266.
32. P. Y-T. Hsu, "Designing the FPT Microprocessor," *IEEE Micro,* Apr. 1994, pp. 23-33.
33. J. Heinrich, *MIPS R10000 Microprocessor User's Manual, V.2.0,* Mips Technologies, Mountain View, Calif., Sept. 1996.
34. L. Gwennap, "MIPS R10000 Uses Decoupled Architecture," *Microprocessor Report,* Vol. 8, No. 14, 1994, pp. 18-22.
35. B. Case, "AMD Unveils First Superscalar 29K Core," *Microprocessor Report,* Vol. 8, No. 14, 1994, pp. 23-26.
36. M. Slater, "AMD's K5 Designed to Outrun Pentium," *Microprocessor Report,* Vol. 8, No. 14, 1994, pp. 1-11.
37. B. Burkhardt, "Delivering Next-Generation Performance on Today's Installed Computer Base," *Proc. Compcon,* 1994, pp. 11-16.
38. L. Gwennap, "NexGen Enters Market with 66-MHz Nx586," *Microprocessor Report,* Vol. 8, No. 4, 1994, pp. 12-17.
39. J.E. Smith, "Dynamic Instruction Scheduling and the Astronautics ZS-1," *Computer,* July 1988, pp. 21-34.
40. K. Hwang, "Advanced Computer Architecture," McGraw-Hill, New York, 1993.
41. R.M. Keller, "Look-Ahead Processors," *Computing Surveys,* Vol. 7, No. 4, 1975, pp. 177-195.
42. J.K. Lee and A. Smith, "Branch Prediction Strategies and Branch Target Buffer Design," *Computer,* Vol. 17, No. 1, 1984, pp. 6-22.
43. J.E. Thornton, "Design of a Computer: The CDC 6600," Scott and Foresman, Glenview, Ill., 1970.

Dezsõ Sima is the director of the Institute of Informatics at Kandó Polytechnic, Budapest, Hungary. He has taught computer architecture at the Technical University of Dresden, Germany, and at Kandó Polytechnic. He was the first professor to hold the Barkhausen Chair at the Technical University of Dresden, Germany. He has also been a guest lecturer on computer architectures at European universities and a visiting professor at South Bank University in London.

Sima holds an MSc in electrical engineering and a PhD in telecommunications, both from the Technical University of Dresden. He serves as acting president of the John von Neumann Society of Computing in Hungary. He is an IEE Fellow and a member of the IEEE and the Computer Society.

Direct questions concerning this article to the author at Kandó Polytechnic, H-1431, Budapest 8 Pf. 112, Hungary; sima@novserv.obuda.kando.hu.

Reader Service Card

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 156 Medium 157 High 158