3.1 Introduction

In this chapter we deal with superscalar processors. Since superscalar processors are currently the predominant class of processors, we discuss them in considerable detail.

In Section 3.1 we first give an overview of how superscalar processors emerged and gained a foothold in the market. Then we introduce the design space of superscalar instruction issue. Subsequent sections are devoted to the specific tasks of superscalar processing, such as parallel decoding (Section 3.2), superscalar instruction issue (Section 3.3), shelving (Section 3.4), register renaming (Section 3.5), parallel execution (Section 3.6), preserving the sequential consistency of execution (Section 3.7) and that of exception processing (Section 3.8). We then discuss the implementation of superscalar CISC-processors using a superscalar RISC core (Section 3.9). We conclude the chapter with case examples in Section 3.10, highlighting the R10000, the PowerPC 620 and the PentiumPro.

3.1.1 The emergence and spread of superscalar processors

The path to widespread use of superscalar instruction issue in main line processors took quite some time. As is usual in technology related developments, superscalar processors *emerged in three consecutive phases.* First, the *idea* was conceived, then a few *architecture proposals* and *prototype machines* appeared, and finally, in the last phase the *commercial products* reached the market.

The **idea** of superscalar issue was first formulated as early as in 1970 (Tjaden, Flynn 1970). It was later reformulated more precisely in the ‘80s (Torng 1982, Acosta & al. 1986).

Superscalar processor proposals and **prototype machines** followed as overviewed in Figure 3.1.1.

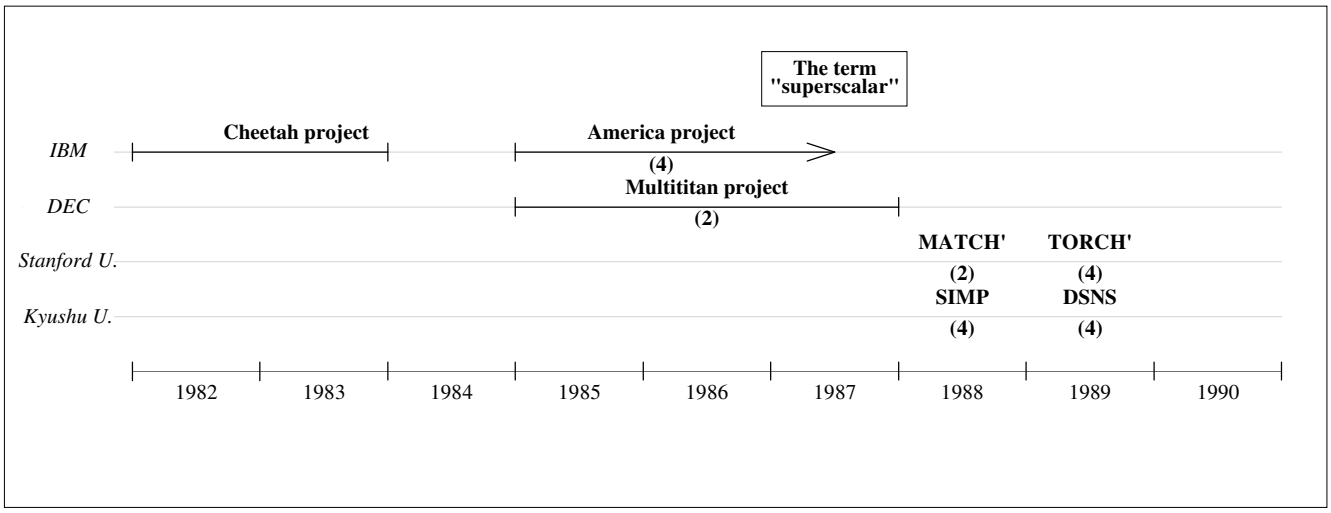


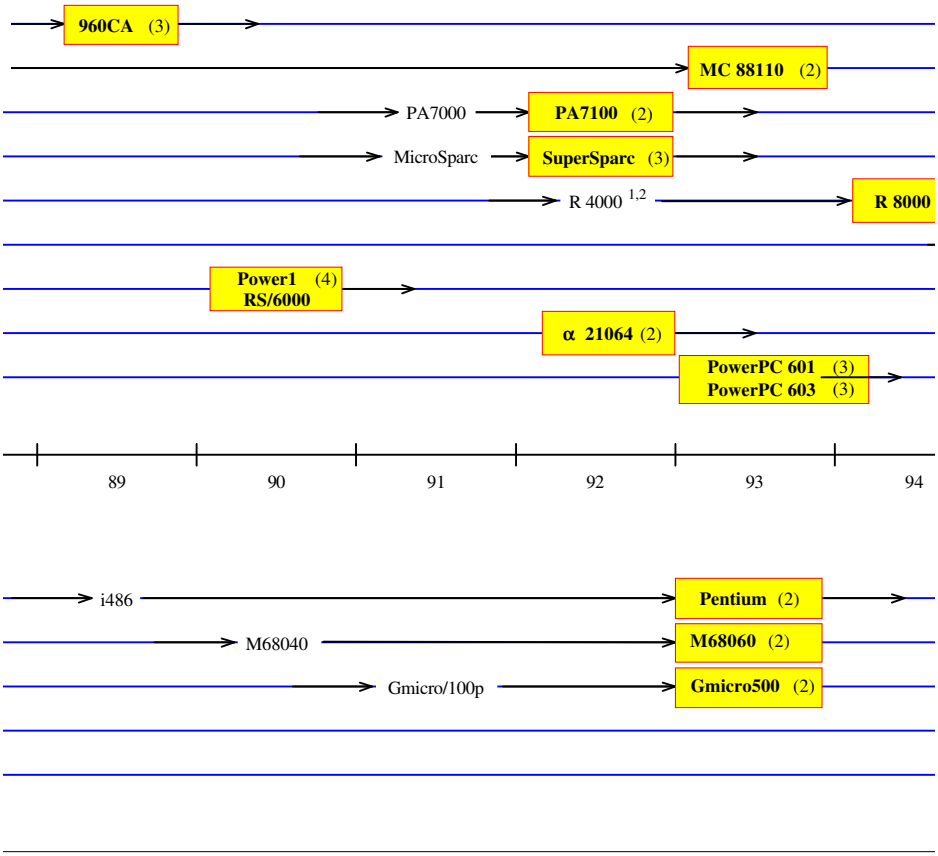
Figure 3.1.1: Overview of proposed or prototype superscalar processors

As far as prototype machines are concerned IBM was the first with two significant superscalar developments called the *Cheetah* and *America projects.* The *Cheetah project* (1982-83) and the subsequent *America project* (from 1985 on) were the test beds for IBM to study superscalar execution. The 4-way Cheetah machine served as a base for the America processor, which spawned the Power1, i.e. the RS/6000 (1990). Actually, the Power1 is almost identical with the America machine (Grohoski (1990)). It is interesting to note that the term "superscalar" is assumed to have first appeared in connection with these developments in an internal IBM Technical Report (Agarwala, T., Cocke, J.: High Performance Reduced Instruction Set Processors, 1987).

A second early player in the field of superscalar developments was DEC with its *Multititan project,* carried out from 1985 to 1987, (Jouppi & al. 1989). While the Multititan project was the continuation of project Titan (1984), whose goal was to construct a very high speed RISC processor, this project did not actually contribute much to the development of the α line of processors. As reported in Commerford (1992), the α development started in 1988 as a totally new project, then code-named RISCy VAX.

Two further projects also deserve mention. At Stanford University an R2000-based superscalar processor proposal called *Torch* was studied by extensive simulations. Torch was conceived as a 2-way superscalar processor which supported global scheduling in hardware. The other Stanford project was code-named Match. This was a simulated architecture proposal of Johnson (Johnson 1989). Finally, two superscalar prototype machines labeled *SIMP* and *DSNS* were developed at Kyushu University, Japan, around 1988-1989, (Kuga & al. 1991).

These research projects paved the way to the **commercial developments**, depicted in Figure 3.1.2. In this Figure we show how superscalar processors (highlighted), appeared in significant processor lines.



994) since superscalar architectures are intended to extend the performance of the high-end mo
than MIPS Inc., such as the R 4400 (1994) from IDT, Toshiba and NEC.
is is roughly equivalent to two CISC instructions per cycle.

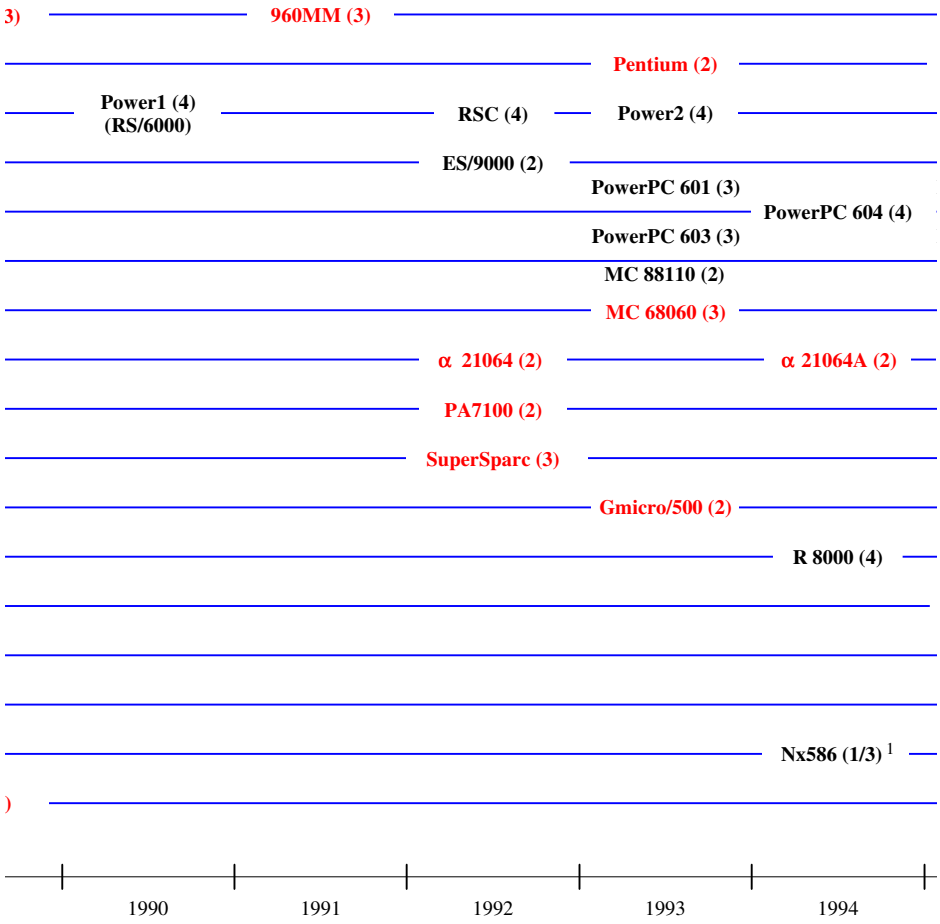
Figure 3.1.2: The appearance of superscalar processors

As Figure 3.1.2 shows, the Intel 960CA embedded RISC processor was the first commercial superscalar machine, introduced in 1989. In order to boost performance subsequently all major manufacturers were forced to introduce superscalar issue in their commercial processor lines. Regarding this we point out that superscalar issue was introduced earlier in RISC than in CISC processors.

Superscalar RISC *processors* emerged according to two different approaches. Some appeared as the result of *converting an existing (scalar) RISC line* into a superscalar one. Examples of this are the Intel 960, MC 88000, HP PA (Precision Architecture), Sun Sparc, MIPS R and AM 29000 RISC lines. The other major approach was to conceive *a new architecture*, and to implement it from the very beginning as a superscalar line. This happened when IBM announced its RS/6000 processor in 1990, later renamed the Power1. Other examples are the DEC α line and the PowerPC family from Apple, IBM and Motorola.

Due to their higher complexity superscalar *CISC processors* appeared on the market after a considerable delay. This is caused by two reasons. First, in contrast with RISCs, superscalar CISC processors have to decode multiple variable length instructions. Second, it is more demanding to implement a CISC-type memory architecture than a simple RISC-type load-store architecture. The Pentium and the MC 68060 are examples of the first superscalar CISC machines, which are available since 1993. Both were the result of converting existing CISC lines into superscalar ones. In contrast, the M1 and the K5 processors were designed *from scratch* as x86 compatible superscalar processors. All of these superscalar CISC processors have a low issue rate of around 2 due to the additional complexities of superscalar CISC processors mentioned above. We note that some CISC processors, for instance the Pentium, the Nx586 or the K5, are implemented using a superscalar RISC core, as explained in Section 3.9. In these processors the RISC core typically has an issue rate of 4, which equals a CISC-rate of "about 2".

Following their introduction superscalar processors rapidly conquered the market. In just a few years every major manufacturer were offering superscalar versions of their earlier scalar lines or had introduced a new superscalar family. Figure 3.1.3 provides an overview of the significant *superscalar lines and models* as well as *their manufacturers*. We also indicate the issue rate of the various processors.



[1] The Nx586 has scalar issue for CISC in
superscalar core for converted RISC in

Figure 3.1.3: Overview and evolution of commercial superscalar processors

Clearly, the market includes different classes of superscalar processors with varying application fields, performance levels and architectures. Two of the lines indicated are embedded: the Intel 960, and the Am 29000 superscalar lines. All other lines or models are of general use. These processors are typically intended for the high performance desktop and workstation market. The PowerPC 602 and PowerPC 603 are exceptions being low cost, low power models. In Figure 3 most architectures are RISCs. Only the Intel 80x86, ES/9000, MC 68000, Gmicro, K5, M1 and the Nx are of the CISC-type. Of these, the K5, M1 and the Nx586 are x86 compatible and were developed to compete with the Pentium, which holds a major share of the microprocessor market.

3.1.2 Specific tasks of superscalar processing

Superscalar processing can be broken down into a number of specific tasks, which we will review based on Figure 3.1.4. Since superscalar processors have to issue multiple instructions per cycle, the first task neccesarily is *parallel decoding.* Clearly, 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 is a partial decoding performed in advance of common decoding, while instructions are loaded into the instruction cache. The majority of the latest processors use predecoding, including the PowerPC 620, PA 7200, PA 8000, UltraSparc or R10000. Parallel decoding, including predeconding is discussed in Section 3.2.

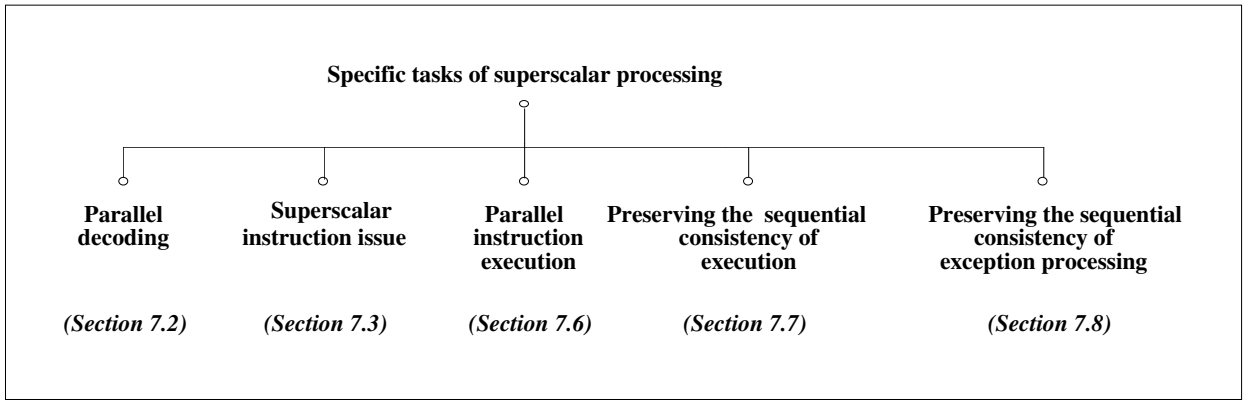


Figure 4 : Specific tasks of superscalar processing

The most crucial task of superscalar processing is *superscalar instruction issue.* Clearly, a higher issue rate gives rise to higher processor performance, but at the same time it *amplifies the restrictive effects of control- and data dependencies* on the processor performance as well[2]. For control- and data dependencies there are different reasons. Control dependencies clearly occur more frequently per issue in a superscalar processor, which issues multiple instructions in each cycle, than in a scalar processor. The frequency increase is roughly proportional to the issue rate of the processor.

As far as data dependencies are concerned a higher issue rate may also strongly impede processor performance. Consider here that in a scalar processor, for instance in a pipelined processor, issue blockages due to dependencies may be avoided in most cases through the use of a parallel optimising compiler. The compiler fills 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 otherwise wasted slots. However, when the issue rate is increased, say from 1 to 4, more and more independent instructions are needed to fill each issue slot. Very soon the point will be reached when far more independent instructions would be needed than are available. This qualitative picture may be quantified as follows. Assuming that a straightforward issue scheme is used, in a general purpose program, a common parallel optimising compiler will probably be able to fill most issue slots with independent instructions for a low issue rate of 2. However, when the issue rate is increased 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, in order to achieve higher performance, superscalar processors have introduced intricate *instruction issue policies*, involving advanced techniques such as *shelving, register renaming and speculative branch processing*. As a consequence, the instruction issue policy used becomes crucial for achieving higher processor performance. Section 3.3 is devoted entirely to this topic. Shelving, register renaming and speculative branch processing are dealt with in Sections 3.4, 3.5 and 8.4.4, respectively.

The next task is *parallel instruction execution*. It is a precondition of superscalar processing and is discussed in Section 3.6. However, while instructions are executed in parallel, instructions are usually completed 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 back into the specified register or memory location. While the results are generated in parallel by the EUs, the program state is updated in a decoupled manner sequentially in program order. As in the quest for higher performance superscalar processors tend to offer increasingly higher parallelism, the task of

preserving sequential consistency also becomes increasingly more demanding. For details see Section 3.7.

Finally, during instruction execution exceptions may arise. Here too sequential consistency is important - and gives rise to the task called the *preservation of sequential consistency of exception processing*. We are concerned with this topic in Section 3.8.

3.2 Parallel decoding

As mentioned before, decoding in a superscalar processor is a much more complex task than in a scalar one, the more the higher the issue rate is. In order to illustrate this point, we compare the decode and issue tasks of a scalar and a superscalar processor (see Figure 3.2.1).

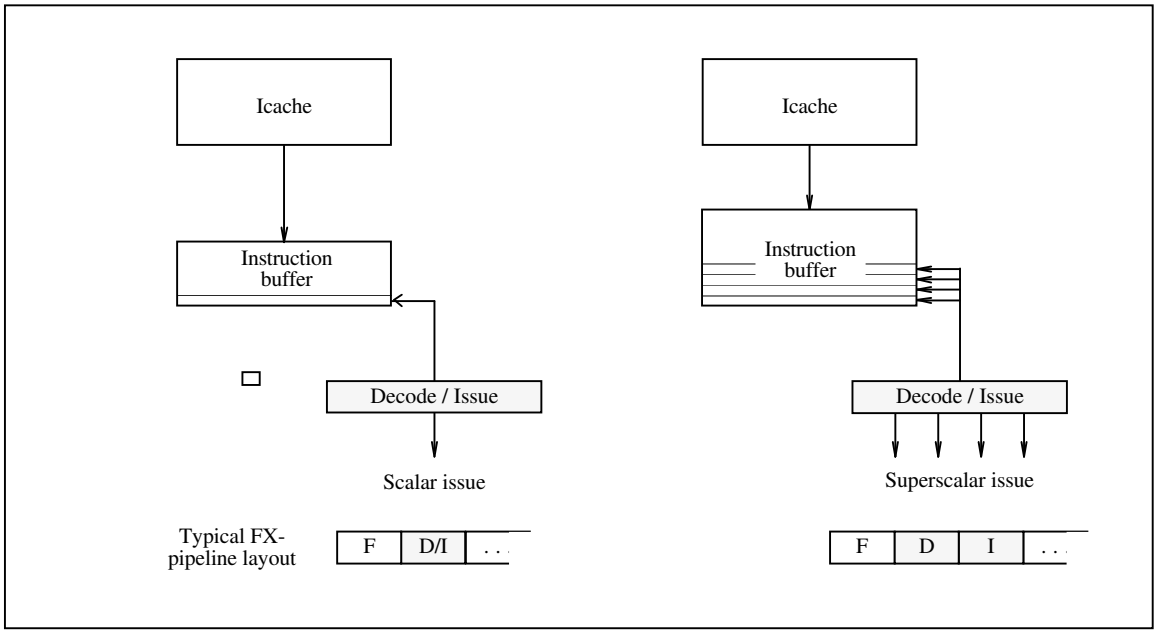


Figure 3.2.1: Contrasting decoding and instruction issue in a scalar and a 4-way superscalar processor

A scalar processor has to decode only a single instruction in each cycle as indicated on the left of Figure 3.2.1. It occasionally has to check for dependencies in order to decide whether this instruction can be issued or not. In comparison, a superscalar processor has to perform a much more complex task. As demonstrated on the right of Figure 3.2.1 it has to decode multiple instructions, say four, in a single clock cycle. It also needs to check for dependencies from two perspectives. First, related to all instructions being in execution and, second, among the instructions which are candidates for the next issue. Since, a superscalar processor has more EUs than a scalar one the number of instructions in execution is far higher than in the scalar case. This means that more comparisons have to be performed in the course of dependency checks. Clearly, the *decode-issue path* of superscalar processors is a much more

critical issue in achieving a high clock frequency than that of scalar processors. It is little wonder that superscalar processors tend to use two and sometimes even three or more pipeline cycles for decoding and issuing instructions. For instance, the PowerPC 601, PowerPC 604, and UltraSparc need two cycles, while the α 21064 requires 3 cycles, and the PentiumPro needs even 4.5 cycles therefore. One way to cope with this problem is through *predecoding*.

Predecoding shifts a part of the decode task up into the loading phase of the on-chip instruction cache, as depicted in Figure 3.2.2.

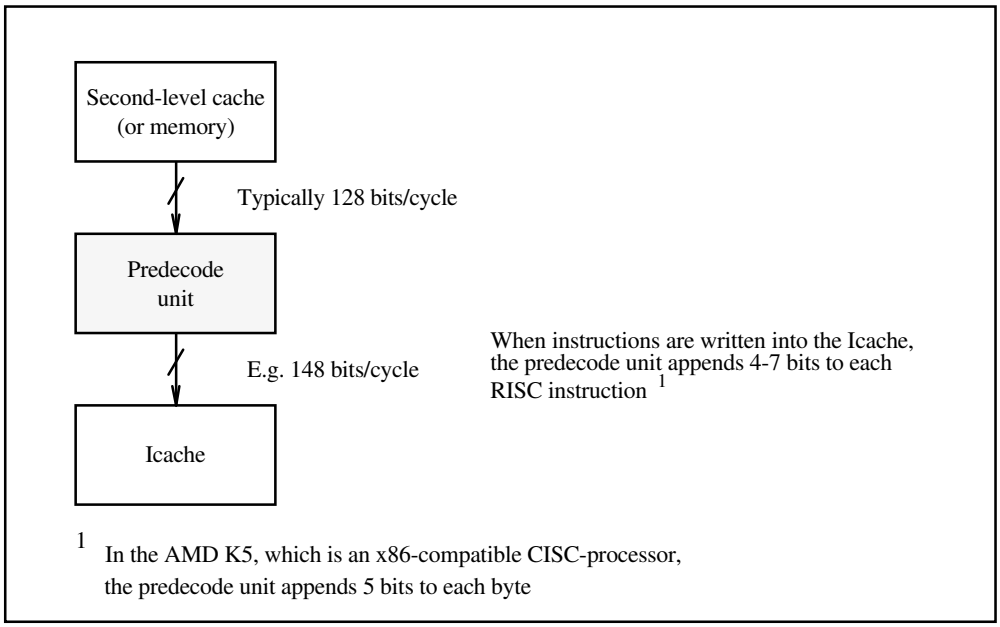


Figure 3.2.2: The principle of predecoding

Here, during loading of the Icache, a dedicated unit, called the *Predecode unit*, performs a partial decoding and appends a number of decode bits to each instruction. In the case of RISC-processors, for example 4 - 7 bits are usually attached which indicate :

- the instruction class,
- the type of resources which are required for the execution,
- in some processors even the fact that branch target addresses have already been calculated during predecoding (like in the Hal PM1 or in the UltraSparc).

The *number of the predecode bits* used is summarised in Table 3.2.1.

Type/year of first volume shipment	Number of predecode bits appended to each instruction
PA 7200 (1995)	5
PA 8000 (1996)	5
PowerPC 620 (1995)	7
UltraSparc (1995)	4
HAL PM1 (1995)	4
AMD K5 (1995)	5[1]
R10000 (1996)	4

Comment: [1] In the K5, 5 predecode bits are added to each byte

Table 3.2.1: Number of predecode bits used

For a CISC-processor such as the AMD K5 predecoding can determine where an instruction starts or ends, where the opcodes and prefixes are, etc. This requires quite a large number of extra bits. The K5, for instance, adds 5 extra bits to each byte. Thus, in this case more than 70 % additional storage space is needed in the instruction cache.

The trend toward predecoding is immediately evident when we look at the overview of their introduction in Figure 3.2.3. Predecoding was introduced in almost all of the most recent members of the predominant processor lines. Although some fairly recent processors, such as the PA 7100, PowerPC 601, PowerPC 604, R8000 or the SuperSparc did not use yet predecoding, subsequent processors of the same lines have since introduced it. Examples are the PA 7200, PA 8000, R10000, UltraSparc, Hal's PM1. The K5, a CISC processor with a RISC-core, has also introduced predecoding.

Superscalar processors without predecoding		Superscalar processors using predecoding			
Type/Year of first volume shipment	Issue rate instr./cycle		Issue rate instr./cycle	Clock rate[1] MHz	Pipeline cycles [2]
PA 7100 (1992)	2	→ PA7200 (1995)	2	100	n.a.
		→ PA8000 (1996)	4	150	n.a.
PowerPC 601 (1993)	4	→ PowerPC620 (1995)	4	133 MHz	F, D/I, E
PowerPC 604 (1994)	3				
R8000 (1994)	4	→ R10000 (1995)	4	200 MHz	F, D, I, E
SuperSparc (1992)	3	→ UltraSparc (1995)	4	167 MHz	F, D, I, E
		→ Hal PM1 (1995)	4	154 MHz	F, D/I, E
Am 29000 sup. (1995)	3	Am K5 (1995)	~2 [3]	n.a.	n.a.
α21164 (1995)	4				
PentiumPro (1995)	3				

1 In the table we list the initial clock rates.

2 The abbreviation of pipeline cycles is as follows: F: Fetch, D: Decode, I: Issue, E: Execute. Here, the subsequent cycles are omitted.

3 The issue rate is 4 microinstructions/cycle, which equals roughly 2 x86 instructions/cycle

Figure 3.2.3: The introduction of predecoding

Predecoding results either in shortening of the overall cycle time or in the reduction of the number of cycles needed for decoding and instruction issue. For instance, the PowerPC 620, R10000 or and Hal's PM1 require only a single cycle for decoding and issue.

Finally, it is worth mentioning that predecoding has not yet been introduced in some very recent processors, such as the α 21164, PentiumPro or Nx586.

3.3 Superscalar instruction issue

3.3.1 The design space

Superscalar instruction issue can favourable be discussed within the framework of its *design space*. As Figure 3.3.1 illustrates, superscalar instruction issue comprises two major aspects, issue policy and issue rate.

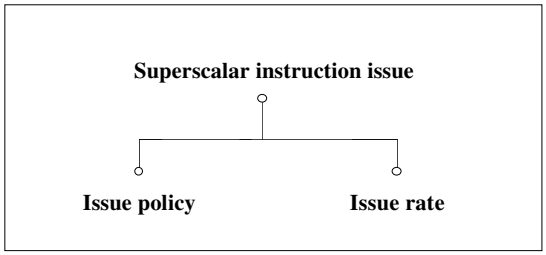


Figure 3.3.1: Design space of superscalar instruction issue

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 is able to issue in each cycle.

3.3.2 Issue policies

The design space of issue policy is considerably complex. As shown in Figure 3.3.2 it consists of four major aspects. The first two specify how false data and control dependencies are coped with during instruction issue. In both cases, the design options are either to eliminate them during instruction issue by using register renaming and speculative branch processing respectively or not. The third aspect determines whether issue blockages will be drastically reduced by the advanced technique of shelving. The final aspect specifies how to deal with issue blockages. In the following we provide a brief overview of the design aspects mentioned.

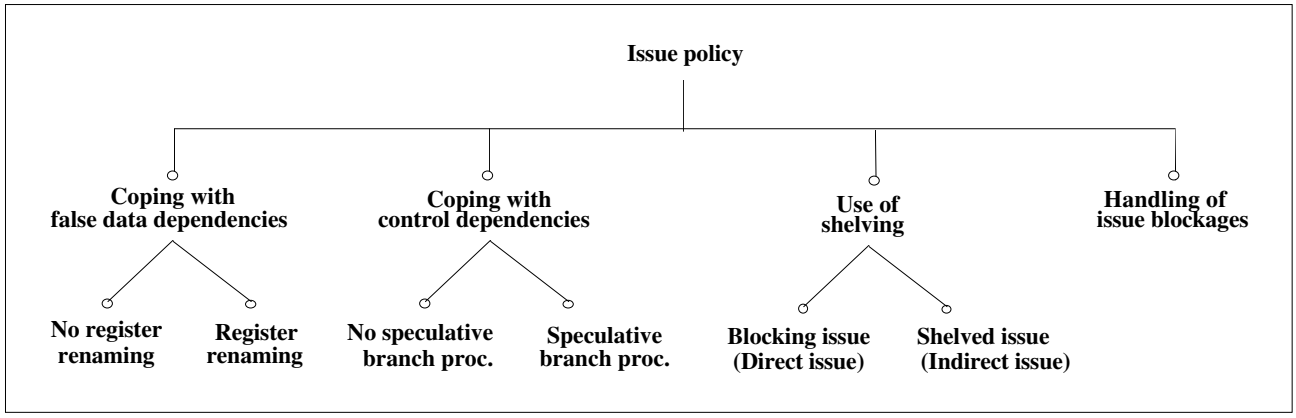


Figure 3.3.2: Design space of issue policies

Within the design space of issue policy the first aspect determines how the processor copes with *false data dependencies,* which occur either between the instructions to be issued and those in execution, or among the instructions to be issued. Specifically, we are concerned with the decision whether WAR (write after read) and WAW (write after write) dependencies occuring between register references will be eliminated or not. We emphasize that this aspect is confined to register data dependencies and does not cover possible false data dependencies

between memory data. The rationale therefore is that false data dependencies between memory data are much less frequent than those between register data, especially in RISC-architectures. Thus, recently used issue policies are not concerned with them.

As we know from Section 4, *false data dependencies* between register data may be removed by *register renaming*. The *principle of register renaming* is quite simple. If a WAW or WAR dependency is encountered, the destination register causing the dependency is renamed. 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 "spare register" instead. See Section 3.7 for details on this technique.

The second aspect of issue policies determines how *unresolved control dependencies* are dealt with. Obviously, a conditional control transfer instruction causes problems if the condition has not yet been produced by the generating instruction by the time the condition should be evaluated. Issue policies handle this in two different ways. Either they wait until the referenced condition becomes available, or they employ speculative execution. With *speculative execution of control transfer instructions*, or *speculative branch processing* for short, a guess is made as to the outcome of each unresolved conditional transfer instruction. Instruction issue is then resumed accordingly. Speculative branch processing is discussed in depth in Section 8.4.4.

The third aspect of issue policy concerns the *use of shelving*, an efficient technique to avoid issue blockages.. Concerning this aspect, simply speaking, the alternatives are either to allow dependent instructions to cause issue blockages (*blocking issue*) or to avoid them through the use of *shelving*.

The *blocking issue* (also called direct issue) is a strict approach related to dependencies. In this scheme decoded instructions to be issued are checked for dependencies, as depicted in Figure 3.3.3. When dependencies occur, instruction issue is blocked.

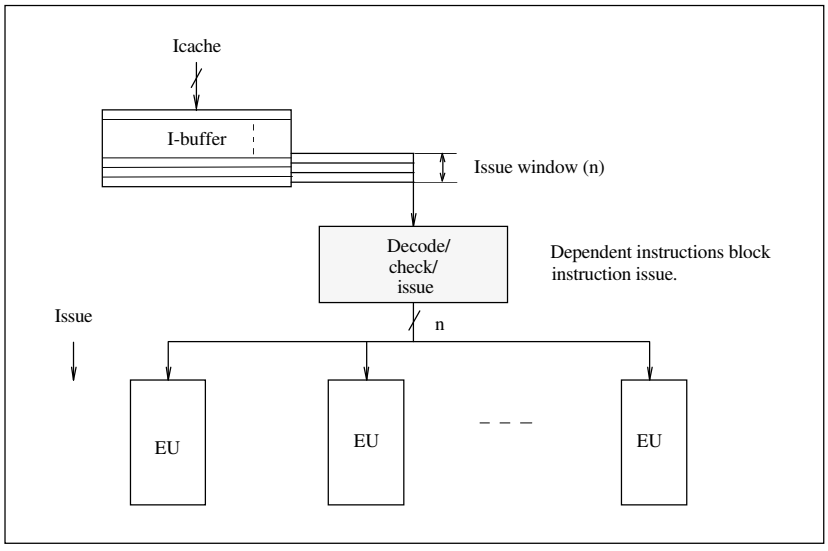


Figure 3.3.3: Principle of the blocking issue mode.
(For illustration purposes we have assumed a 4-instruction wide issue window).

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 (EUs). However, when dependencies occur, the issue blockage will be handled as specified by the corresponding aspect of the design space, which is discussed later in this section. 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.

Shelving (also called shelved issue, 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 EUs. With shelving, instructions are issued first to the shelving buffers with essentially no checks for dependencies, as illustrated in Figure 3.3.4. In this figure we assume that there are individual reservation stations in front of each EU. Other possibilities for the implementation of shelving buffers exist and will be discussed in Section 3.4.3. At this point we note that even if shelving is used, certain hardware constraints, such as smaller than needed bus widths, or lack of free entries in the shelving buffers, may continue to cause issue blockages. For the sake of simplicity we have ignored these constraints in this introduction.

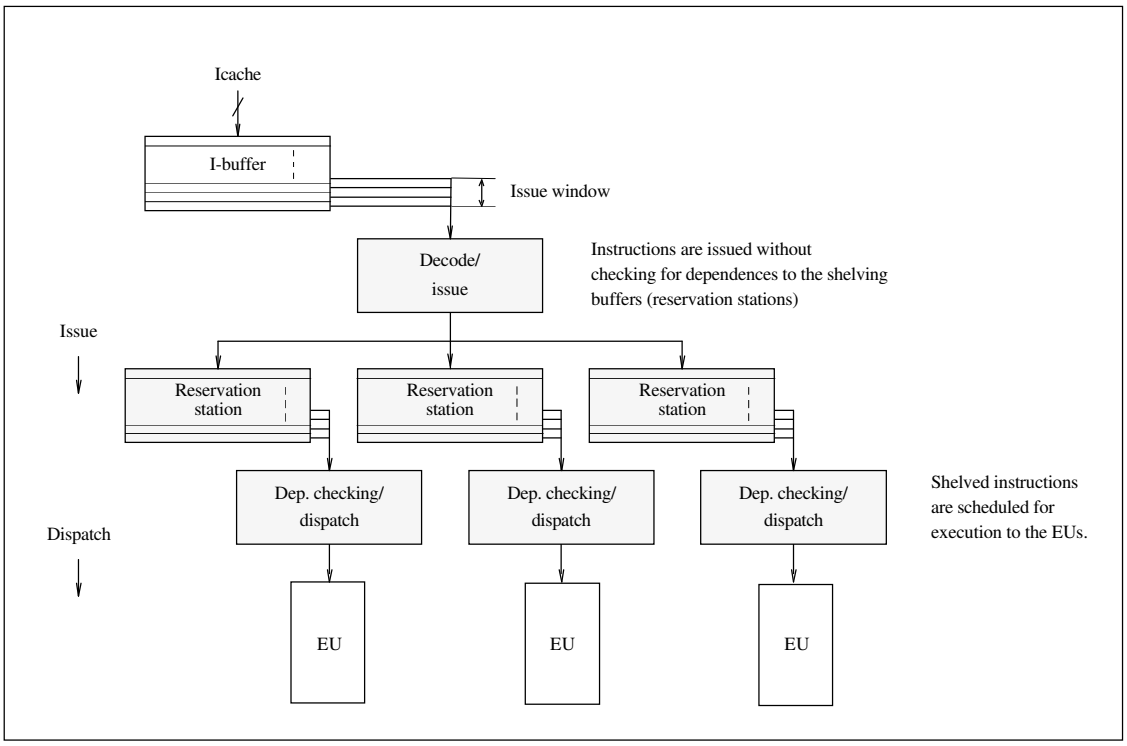


Figure 3.3.4: The principle of shelving, assuming that dedicated buffers (reservation stations) are available in front of each execution unit (EU)

When shelving is used, dependency checking is delayed to a later step of processing called dispatching. During *dispatching* the instructions held in the shelving buffers are checked for dependencies and dependency free-instructions are forwarded to available EUs. Because of their importance, shelving and dispatching are discussed in depth in Section 3.4.

Here, it is appropriate to make a comment regarding terminology. Note that we have used two different terms, instruction *issue* and *dispatch,* to express different actions. Furthermore, we have also used the term *"issue"* in two different interpretations. Without shelving, that is when the blocking issue is used, the term *"issue"* refers to the action of disseminating decoded independent instructions to the EUs. When shelving is used, "*issue"* designates the dissemination of decoded instructions to the shelving buffers. The term *"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 EUs. While useful, this clear distinction is not common in the literature and both "issue" and "dispatch" are used in either interpretation.

The last aspect of the design space of instruction issue declares how *issue blockages* are handled. *Instruction issue* may be blocked in both cases, with or without shelving. As long as shelving is not used, any dependencies encountered in the issue window immediately block the issue of instructions. In contrast, when shelving is used, issue blockages due to dependencies are generally avoided. Nevertheless, despite shelving certain resource dependencies can restrict the issue of instructions into the shelving buffers. Examples include the lack of free entries in shelving buffers or occasionally a too narrow bandwidth to transfer multiple instructions into the same shelving buffer.

The handling of issue blockages can be broken down into two aspects, as depicted in Figure 3.3.5. The first aspect, called *issue order*, specifies whether a dependent instruction blocks the issue of subsequent not dependent instructions in the issue window. The second aspect is the *alignment of instruction issue*. It determines whether a fixed or gliding issue window is used.

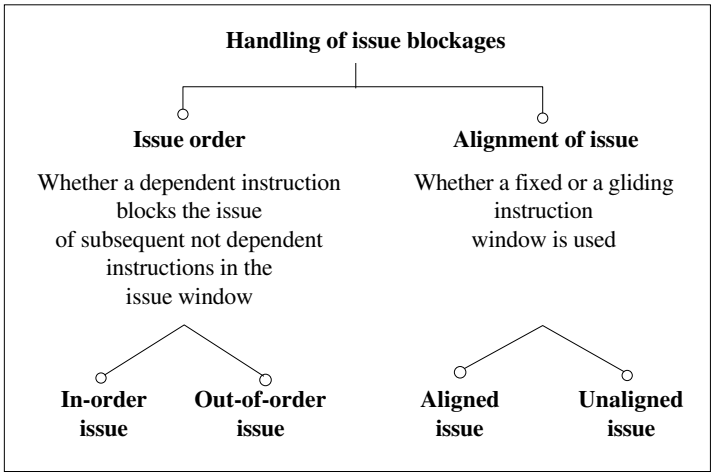


Figure 3.3.5: Design aspects and design choices concerning the handling of issue blockages

Figure 3.3.6 offers more details about issue order. As shown, if a dependent instruction blocks the issue of all subsequent instructions until the dependency is resolved, the issue order is called *in-order.*

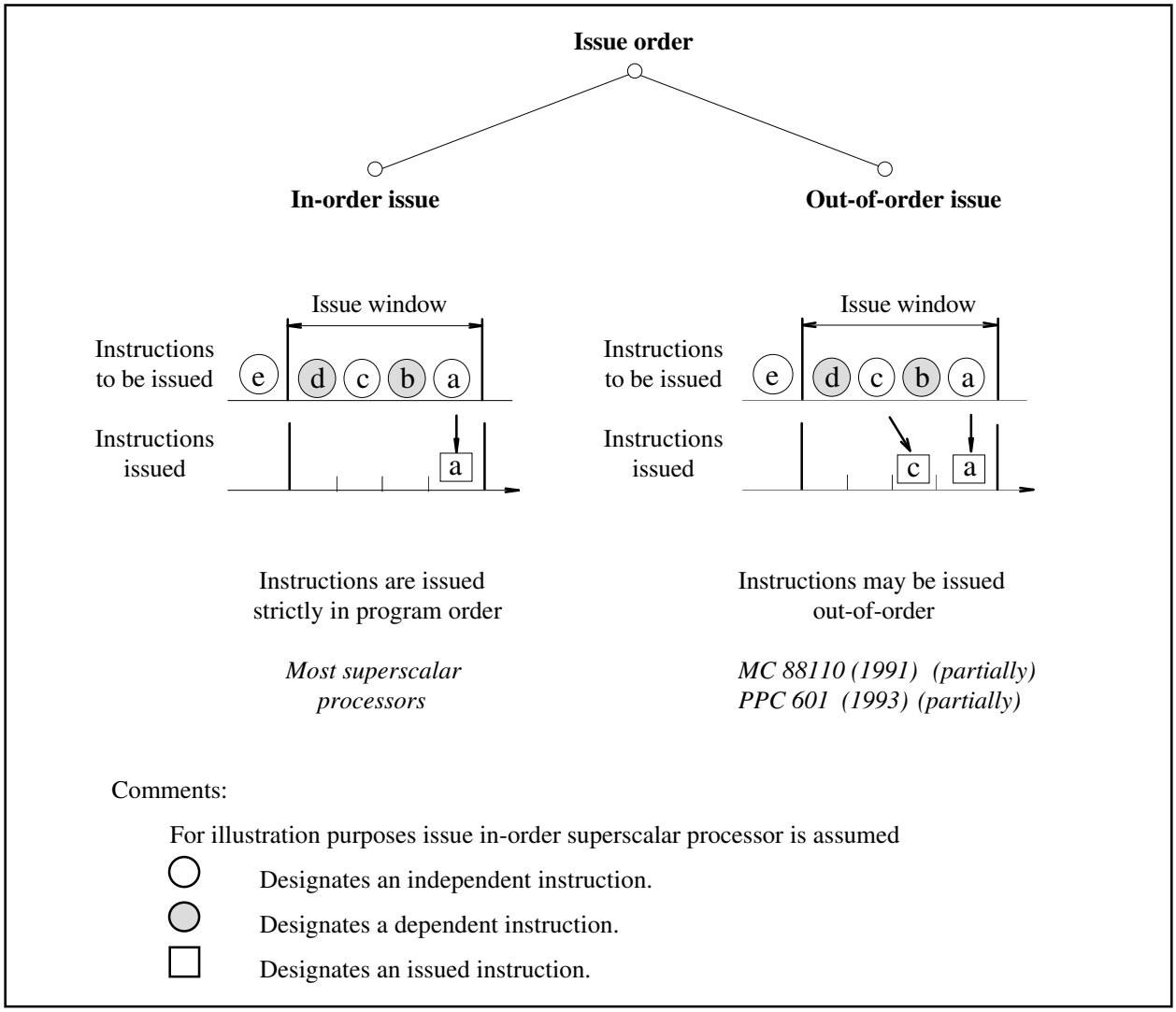


Figure 3.3.6: Issue order of instructions

However, restricting subsequent independent instructions from issue can severely impede performance. Therefore, a few superscalar processors which employ the blocking issue mode, such as the MC 88110 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 can not yet been issued as depicted in Figure 3.3.6. In both processors mentioned out-of-order issue is only partially allowed, that is only for particular instruction types. For instance, the PowerPC 601 issues branches and FP-instructions out-of-order, whereas the MC 88110 does so only for FP-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 for in-order issue. Second, *processors with shelving*, have almost *no motivation* for using *out-of-order issue*. Consider here that with shelving, the issue of instructions is rarely blocked due to resource constraints. Therefore, 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 in the near future.

The second aspect of the handling of issue blockages determines the *alignment of instruction issue.* The alignment of instruction issue refers to the point whether instructions are issued from a fixed window or from a gliding window. As Figure 3.3.7 indicates, in the case of an **aligned instruction issue** a *fixed window* is used. 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, Power1, etc. But in the blocking issue mode, aligned issue considerably reduces the effective issue rate. Therefore, in a number of subsequent superscalar processors which still used the blocking issue mode, **unaligned instruction issue** was introduced. In this case 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 from the window either in an in-order or in an out-of-order fashion, as discussed before. 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, as Figure 3.3.7 shows.

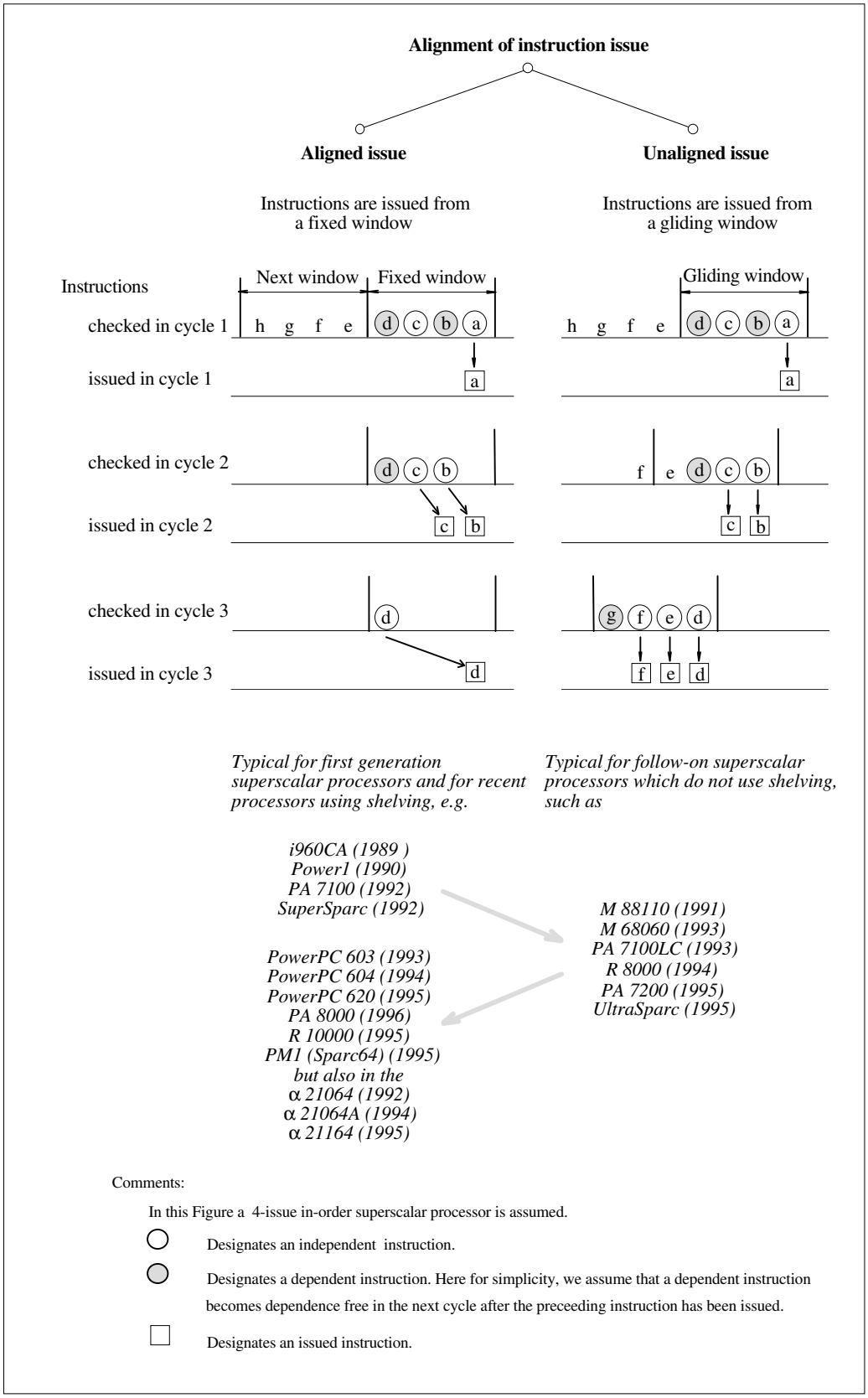


Figure 3.3.7: Contrasting aligned and unaligned issue of instructions

Unaligned instruction issue is typical for the second wave of superscalar processors that employ the blocking issue mode. Examples of this type are the MC 68060, PA 7200, R8000 and the UltraSparc. We note that an early forerunner from the first wave, the MC 88110, also offered unaligned issue. 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 which remain relate to trivial hardware requirements such as availability of shelving buffers, buses, etc. As a consequence, recent superscalar processors which employ shelving have returned to aligned instruction issue. Examples of these are the PowerPC 603, PowerPC 604, PowerPC 620 and the R10000.

The α line of processors are somewhat a special case in this regard. All the α processors employ aligned issue, even the most recent α 21164, despite the fact that this model does not incorporate shelving. Still, at the time of writing, the α 21164 model has the highest performance figure among the microprocessors in terms of the SPEC95 benchmark figures.

We can sum up the discussion of *instruction alignment* by saying that unaligned instruction issue has been introduced to increase the performance of superscalar processors employing the blocking issue mode. However, with the application of shelving unaligned instruction issue has lost its rationale, and most recently introduced superscalar processors use aligned issue.

3.3.3 Most frequently used issue policies

In the following, we survey the most frequently used issue policies, in three scenarios. We first focus on scalar processors, then on superscalar processors, and conclude with the broad picture covering both.

With reference to Figure 3.3.2 we remind the reader that the design space of instruction issue policies is spanned by four major aspects: how false data- and control dependencies are coped with, whether shelving is used or not as well as how issue blockages are handled. In addition, the handling of issue blockages covers two aspects: issue order and alignment. Thus, altogether, the design space of instruction issue covers 5 aspects, each one representing a binary decision. Any combination of the design choices mentioned 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 the aspect of issue order, since most processors employ an in-order

issue. Furthermore, we can omit the point of issue alignment in the case of scalar and superscalar processors which make use of shelving, as explained earlier. However, for superscalar processors using the blocking issue mode, we must still consider the alignment of instruction issue aspect. Thus, all in all for scalar processors we have to consider 3 and for superscalar processors 4 major issue aspects. In the following sections we point out the most frequently used issue policies in the simplified design spaces for scalar and superscalar processors.

Instruction issue policies of scalar processors

While discussing instruction issue policies in the case of scalar processors, we have to take into account the three basic issue aspects, that is whether renaming, speculative execution and shelving are employed or not.

Of the resulting eight possible issue policies, *scalar processors* use predominantly two, as indicated in Figure 3.3.8. We refer to them as the *traditional scalar issue* and *its enhancement with speculative execution*. In addition, there are two further policies of historical importance, designated as the *traditional scalar issue with shelving* and *the traditional scalar issue with shelving and renaming*.

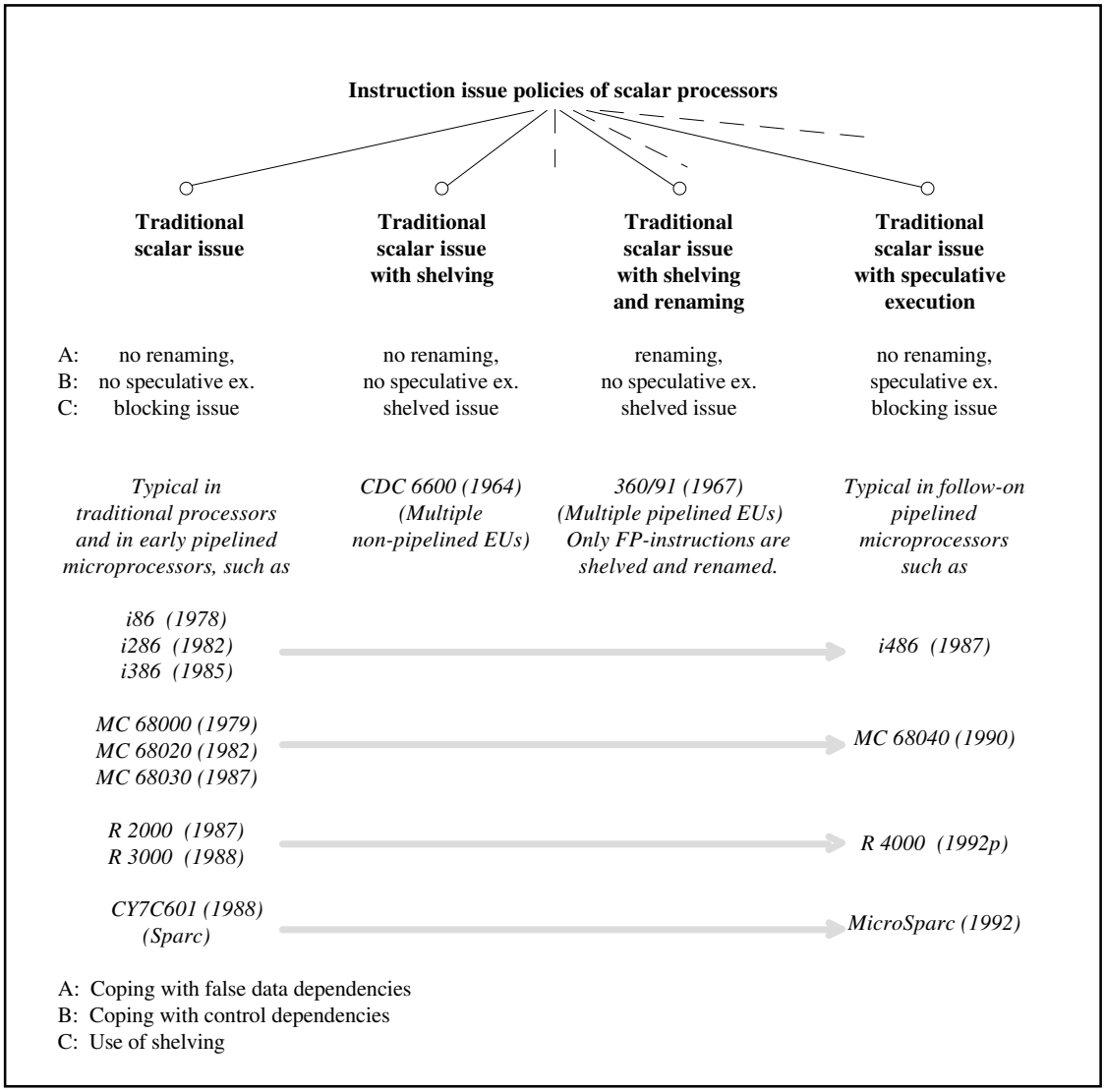


Figure 3.3.8: Most frequently used instruction issue policies of scalar processors

Early non-pipelined processors issued 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. In our classification this issue mode is designated as the *traditional scalar issue policy*. This issue policy was also employed in early pipelined microprocessors. Examples are the i86, i286, i386 or the MC 68000, MC 68020, MC 68030 etc.

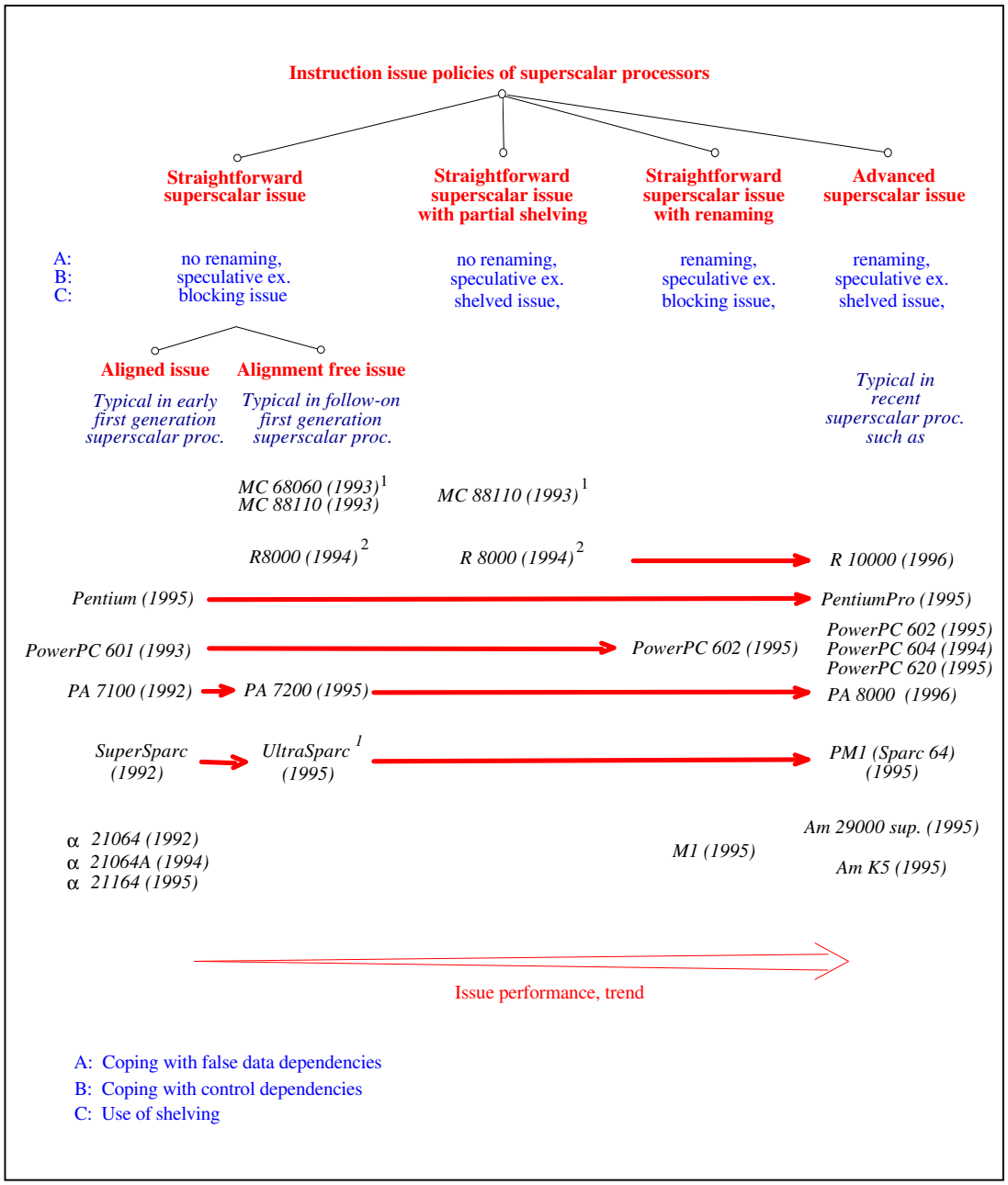
Later, when ILP-processors, that is processors with multiple EUs or pipelined processors, emerged the traditional scalar issue policy proved too restrictive on performance. Two approaches were followed to relieve this. One was introduced in the CDC6600, the other in the IBM 360/91. The CDC 6600, a supercomputer of its time, was one of the first ILP-processors. It achieved high performance by introducing *multiple non-pipelined parallel working EUs*. In this processor, the instruction issue bottleneck was relaxed by *enhancing the traditional issue scheme* with *shelving*. The other approach was pioneered in the IBM 360/91.

This processor was a more advanced ILP-processor from that time, built of multiple pipelined FP-execution units. It also employed a more advanced issue scheme, involving both shelving and renaming. 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 *shelving* was again employed in processors.

Actually, the main line of the evolution of the issue scheme was determined by the spreading of *pipelining*. In pipelined processors unresolved conditional branches can considerably reduce performance in the blocking issue mode, since then instructions are prevented from issue at each unresolved conditional branch. Obviously, the performance cut back increases as the number of pipeline stages increases. Therefore, more advanced pipelined microprocessors, which often were implemented with an increased number of stages had inevitably to enhance the *traditional scalar issue with speculative execution of unresolved conditional branches*, as indicated in Figure 3.3.8 above. Examples of this type are the i486, MC 68040, R4000 or the MicroSparc.

Instruction issue policies of superscalar processors

When we expand our discussion of instruction policies to superscalar processors, we have to take into account beyond the three basic issue aspects considered above, issue alignment as well. This results in 12 feasible issue policies. Of these superscalar processors employ mainly five. As Figure 3.3.9 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*.



Comment: [1] Stores and conditional branches are shelved

[2] FP-instructions are shelved

Figure 3.3.9.: Most frequently used instruction issue policies of superscalar processors

The simplest, most often used policy is the *straightforward aligned superscalar issue*. This policy does not provide for renaming, uses aligned blocking 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, the aligned issue is used. This simple scheme is widely used in first generation superscalar processors, such as the Pentium, PA7100, PowerPC 601 or the α 21064. It is worth noting that beyond first generation superscalar processors, a few later

members of well-known processor families also retained this straightforward issue policy, like the α 21164. Despite this fact, at the time of writing, the α 21164 is the performance leader among microprocessors.

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 in the MC 88110 and the R8000. Both these processors accommodate only partial shelving, The MC 88110 shelves only stores and conditional branches, whereas the R8000 shelves only FP-instructions.

As its name implies, the *straightforward superscalar issue policy with renaming,* the fourth scheme, offers renaming and speculative branch processing but sticks at the blocking issue. This policy has only been employed in a few recently introduced processors, such as the M1 or the PowerPC 602. 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.

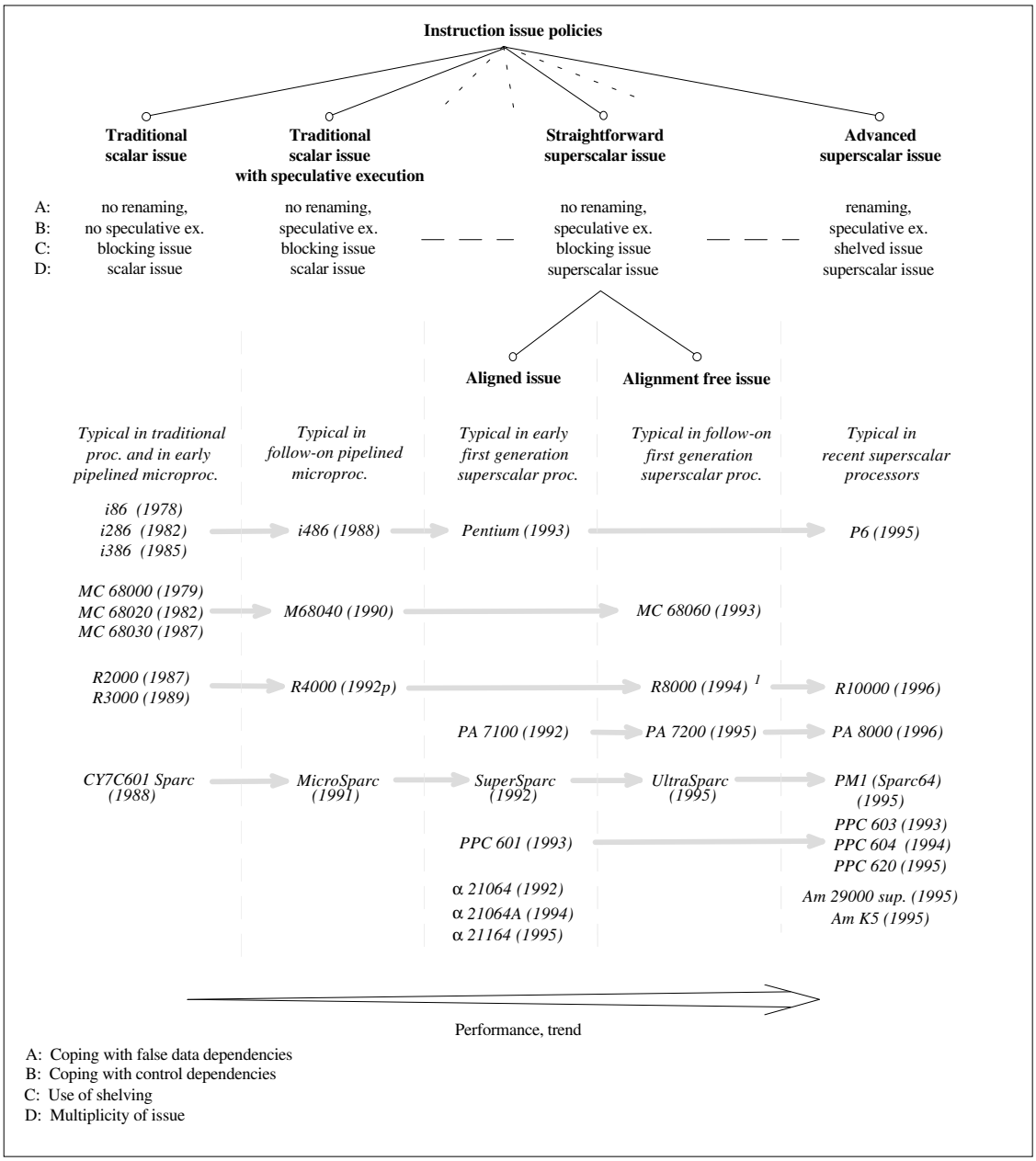
The final policy commonly used is the *advanced superscalar issue policy.* At present this policy is the most relevant one. It employs register renaming, speculative branch processing and shelving. According to this policy, false register data dependencies are eliminated by register renaming, speculative execution is used to cope with unresolved conditional control dependencies and shelving removes issue blockages due to dependencies, provided that enough free shelving buffer entries and wide enough data paths are available. Obviously, this issue policy is the most advanced in the framework of the design space considered. The most recent superscalar processors employ this scheme, including the R10000, PentiumPro, also the PowerPC 603 - PowerPC 620 and a number of other recent processors.

The performance of the discussed issue policies increases more or less in the same sequence as they have been discussed. At the same time, this sequence also marks the trend of the instruction issue policies. Figure 3.3.9 highlights this trend by showing the issue policies used in subsequent members of important superscalar processor families.

The trend in instruction issue policies

In Figure 3.3.10 we summarise the most frequently used issue policies both in scalar and superscalar processors. These policies are: the traditional scalar issue, the traditional scalar issue with speculative execution, the straightforward superscalar issue with and without issue alignment, and the advanced superscalar issue.

In this figure we also reveal the trend of issue policies by indicating the issue policies used in subsequent processors of the same line. Again, the performance of the issue policies mentioned increases from left to right. At the same time this coincides with the trend of issue policies.

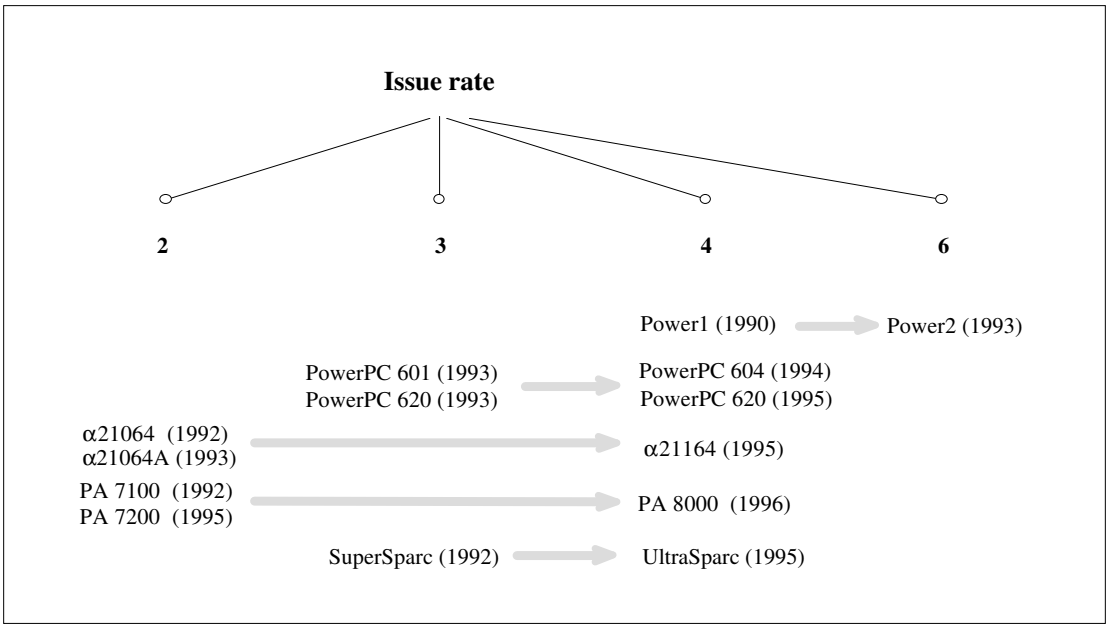


Comment: [1] The R8000 shelves only FP-instructions

Figure 3.3.10: Most frequently used instruction issue policies and their trend

Issue rate

The second major aspect of the design space of instruction issue is the *issue rate* (also known as the *degree of superscalarity*). The *issue 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 higher issue rate offers a higher performance potential, however its implementation requires more complex circuitry. As already mentioned, superscalar CISC processors are usually restricted to issuing only two, or about two instructions per cycle. In contrast, early superscalar RISC processors were able to issue typically 2 - 3 instructions in each cycle, whereas recent RISC processors typically issue 4 instruction in each cycle, as shown in Figure 3.3.11.



Comment:

We have not included low cost, low performance models such as the PowerPC 603 (1993), PA7100LC (1993) or the PowerPC 602 (1995), since they follow a different design philosophy than the performance oriented models considered here.

Figure 3.3.11: Evolution of the degree of superscalarity in RISC processors

This figure exposes the *evolution of the issue rate* experienced in five particular RISC lines. First α processors (the α21064 and α 21064A) as well as first superscalar members of the HP Precision Architecture line (the PA 7100, PA 7200) were restricted to issuing two instructions per cycle. By contrast, first members of the PowerPC line (PowerPC 601), PowerPC 603) and the SuperSparc were capable of issuing 3 instructions per cycle, whereas the Power1 is able to

issue even 4 instructions. For the time being all upcoming models of the lines mentioned except the Power 2 are four-issue superscalar processors. The Power 2 is able to issue even 6 instructions per cycle, and is now the issue-rate leader.