# DECISIVE ASPECTS IN THE EVOLUTION OF MICROPROCESSORS

DEZSŐ SIMA, MEMBER, IEEE

*The incessant market demand for higher and higher performance is forcing a continous growth in processor performance. This demand provokes ever increasing clock frequencies as well as an impressive evolution of the microarchitecture. In this paper we focus on major microarchitectural improvements that were introduced to achieve a more effective utilization of instruction level parallelism (ILP) in commercial, performance-oriented microprocessors. We will show that designers increased the throughput of the microarchitecture at the ILP level basically by subsequently introducing temporal, issue and intra-instruction parallelism in such a way that after exploiting parallelism along one dimension it became inevitable to utilize parallelism along a new dimension to further increase performance. Moreover, each basic technique used to implement parallel operation along a certain dimension inevitably resulted in processing bottlenecks in particular subsystems of the microarchitecture, whose elimination called for the introduction of additional innovative techniques. The sequence of basic and additional techniques introduced to increase the efficiency of the microarchitectures constitutes a fascinating framework for the evolution of microarchitectures, as presented in our paper.*

**Keywords -** *Processor performance, microarchitecture, ILP, temporal parallelism, issue parallelism, intra-instruction parallelism*

## I. INTRODUCTION

Since the birth of microprocessors in 1971, the IC industry has successfully maintained an incredibly rapid increase in performance. For example, as Figure 1 indicates, the integer performance of the Intel family of microprocessors has been raised over the last 20 years by an astonishingly high rate of approximately two orders of magnitude per decade [1], [2].

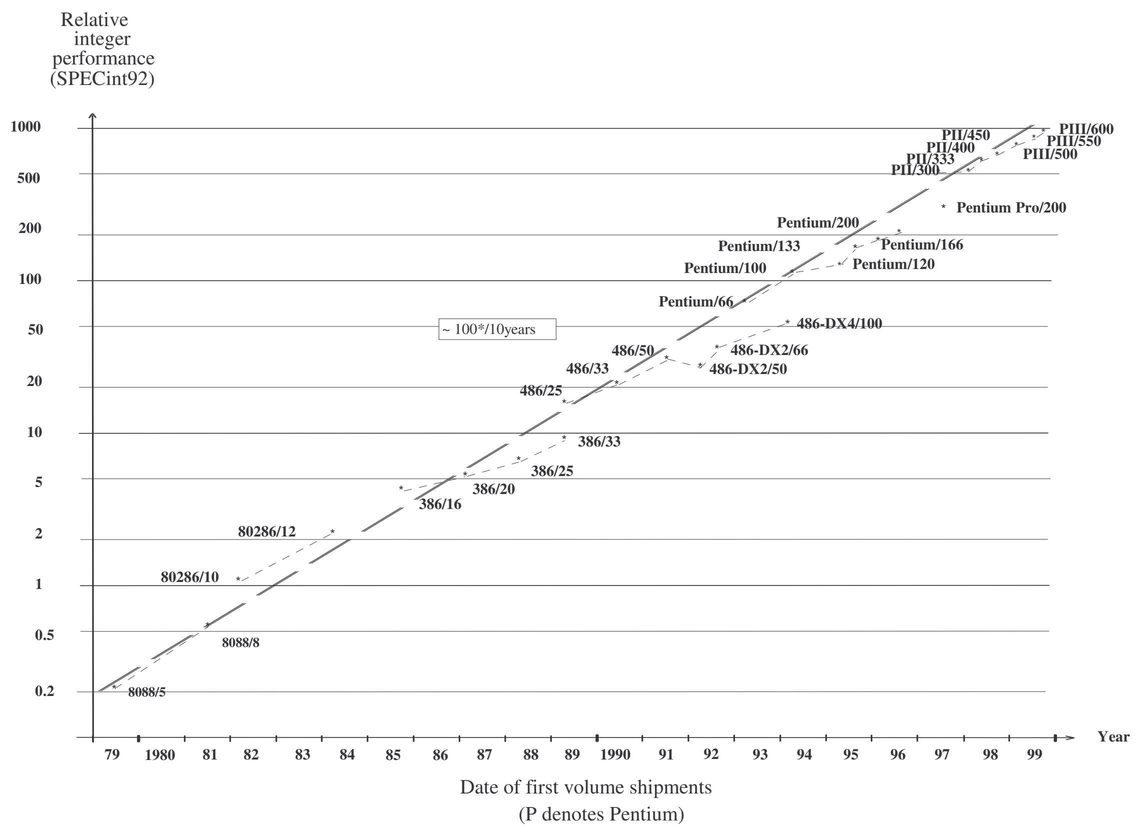



Figure 1: Increase over time of the relative integer performance of Intel x86 processors

This impressive development and the underlying innovative techniques have inspired a number of overview papers [3]–[7]. These reviews emphasized either the techniques introduced or the quantitative aspects of the evolution. In contrast, our paper addresses qualitative aspects, i.e. the incentives and implications of the major steps in microprocessor evolution.

With maturing techniques the "effective execution width" of the microarchitectures (in terms of executed instructions per cycle) approaches the available ILP (in terms of executable instructions per cycle). Recently this has given rise to development in two main directions: (a) the first approach is to utilize ILP more aggressively by means of more powerful optimizing compilers, trace processors [10]–[12] and innovative techniques as discussed in section V.E.; and (b) the other current trend is to also utilize parallelism at a higher-than-instruction (i.e. at the thread or process) level. This approach is marked by multiscalar processors [8], [9], symmetrical multithreading (SMT) [13], [14] and chip multiprocessing (CMP) [15], [16]. In our paper we concentrate on the progress achieved at the instruction level in commercial high performance microprocessors.[1]

The remainder of our paper is structured as follows. In Section II we discuss and reinterpret the notion of absolute processor performance in order to more accurately reflect the performance impact of different kinds of parallel operations in the microarchitecture. Based on this discussion we then identify the main dimensions of processor performance. In subsequent Sections III through VI we review major techniques aimed at increasing

[1] We note that computer manufacturers typically offer three product classes, (i) expensive high performance models designed as servers and workstations, (ii) basic models emphasizing both cost and performance, and finally (iii) low cost (value) models emphasizing cost over performance. For instance, Intel's Xeon line exemplifies high performance models, the company's Klamath, Deshutes, Katmai, Coppermine and Pentium 4 cores represent basic models, whereas their Celeron processors are low cost (value) models. High performance models are obviously expensive, since all processor and system components must provide a high enough throughput, whereas low cost systems save cost by using less ambitious and less expensive parts or subsystems. In order to avoid a large number of multiple references to superscalar processors in the text and in the figures, we give all references to superscalars only in Figure 22.

processor performance along each of the main dimensions. From these, we point out the basic techniques that have become part of the mainstream evolution of microprocessors. We also identify the potential bottlenecks they induce, and highlight the techniques brought into use to cope with these bottlenecks. Section VII summarizes the main evolutionary steps of the microarchitecture of high performance microprocessors, followed by Section VIII, which sums up the decisive aspects of this evolution.

## II. THE DESIGN SPACE OF INCREASING PROCESSOR PERFORMANCE

The results supplied by today's industry standard benchmarks, including the SPEC benchmark suite [17]–[19], Ziff-Davis's Winstone [20] and CPUmark [21] as well as BABCo's SYSmark [22], are all *relative performance measures*. This means that they give an indication of how fast a processor will run a set of applications under given conditions in comparison to a reference installation. These benchmarks are commonly used for processor performance comparisons, in microprocessor presentations and in articles discussing the quantitative aspects of the evolution.

Unlike relative performance measures, *absolute processor performance* ($P_P$) is usually interpreted as the average number of instructions executed by the processor per second. This score is typically given in units like MIPS (Million Instructions Per Second) or GIPS (Giga Instructions Per Second). Earlier synthetic benchmarks, like Whetstone [23] or Dhrystone [24], were also given as absolute measures.

$P_P$ can be expressed as the product of clock frequency ($f_C$) and the average number of instructions executed per clock cycle (IPC):

$$P_P = f_C * IPC \qquad (1)$$

IPC is also designated as the *throughput* and may be interpreted as the *execution width* of the processor (P).

Absolute measures are appropriate for use when the performance potential of processors is discussed. However, absolute performance metrics are not suitable for the comparison of processor lines whose Instruction Set Architectures (ISA) differ. The reason is that instructions from different ISAs do not necessarily perform the same amount of computation. For making performance comparisons in these cases, relative performance measures are needed.

As our paper focuses on the evolution of microarchitectures from a performance perspective, we will apply the notion of *absolute* processor performance. However, in order to identify the contribution of different sources of parallelism within the microarchitecture, in the following we will express IPC with internal operational parameters of the microarchitecture. Further on to take the impact of multi-operation instructions, such as SIMD instructions, into consideration, we will reinterpret the notion of absolute processor performance.

In expression (1) IPC—i.e. the average number of instructions executed per cycle—reflects the result of parallel instruction processing within the microarchitecture. Internal instruction parallelism may have however, two basic sources, pipelined instruction processing and superscalar instruction issue. As shown in the Annex (expression 13), parallelism arising from these two separate sources can be expressed as follows:

$$IPC = 1/CPI * ILP \qquad (2)$$

In Expression (2) CPI is the average time interval between two clock cycles in which instructions are issued, given in clock cycles. (For a more detailed explanation see the Annex.) Here instruction issue denotes the act of disseminating instructions from the instruction fetch/decode subsystem for further processing, as detailed in Section V. C. We note that in the literature this activity is often designated as dispatching instructions. For traditional microprogrammed processors CPI marks the average execution time (where CPI >> 1), whereas for ideal pipelined processors CPI equals 1. We emphasize that CPI reflects the *temporal parallelism* of instruction processing.

ILP is the average number of instructions issued per issue interval. (For a more detailed explanation see again the Annex.) For a scalar processor ILP = 1, whereas for a superscalar one ILP > 1. This term indicates the *issue parallelism* of the processor.

Furthermore, as the use of multi-operation instructions, such as SIMD instructions, has become a major trend, it is appropriate to reinterpret the notion of absolute processor performance, *while taking into account the number of data operations* processed by these instructions as well. This can be achieved by considering the average number of *operations* the processor executes per cycle (designated by OPC) rather than the average number of *instructions* processed per cycle (IPC). If we denote the average number of data operations executed by the instructions by OPI, then

$$OPC = IPC * OPI \qquad (3)$$

For a traditional ISA, we assume OPI = 1. For ISAs including multi-operation instructions such as SIMD instructions, OPI > 1, whereas for VLIW (Very Large Instruction Word) architectures, OPI >> 1, as detailed in Section VI. We point out that OPI reveals the *intra-instruction parallelism.*

With expressions (2) and (3), the average number of operations executed per cycle (OPC) is:

$$OPC = \underbrace{\frac{1}{CPI}}_{\text{Temporal parallelism}} * \underbrace{ILP}_{\text{Issue parallelism}} * \underbrace{OPI}_{\text{Intra-instruction parallelism}} \qquad (4)$$

Finally, absolute processor performance, interpreted as the average number of operations executed per second ($P_{PO}$) yields:

$$P_{PO} = f_c * \frac{1}{CPI} * ILP * OPI \qquad (5)$$



Figure 2: Constituents of processor performance

Here the clock frequency of the processor ($f_c$) depends on the sophistication of IC fabrication technology as well as the way the microarchitecture is implemented. In pipelined designs, the minimum clock period and thus the maximum clock frequency is determined by the worst case propagation delay of the longest path in the pipelined stages. This equals the product of the gate delay and the number of gates in the longest path of any pipelined stage. The gate delay depends mainly on the line width of the IC technology used, whereas the length of the longest path depends on the layout of the microarchitecture.

Very high clock rates presume very deeply pipelined designs, that is, pipelines with typically ten to twenty stages.

The remaining three components of processor performance, i.e. the temporal, issue and the intra-instruction parallelism, are determined mainly by the efficiency of the processor level architecture, that is, by both the ISA and the microarchitecture of the processor (see Figure 2).

Equation (5) provides an appealing framework for a discussion of the major possibilities in increasing processor performance. According to equation (5), the *key possibilities for boosting processor performance* are: (a) increasing the clock frequency and (b) introducing/increasing temporal, issue and intra-instruction parallelism, as summarized in Figure 3.

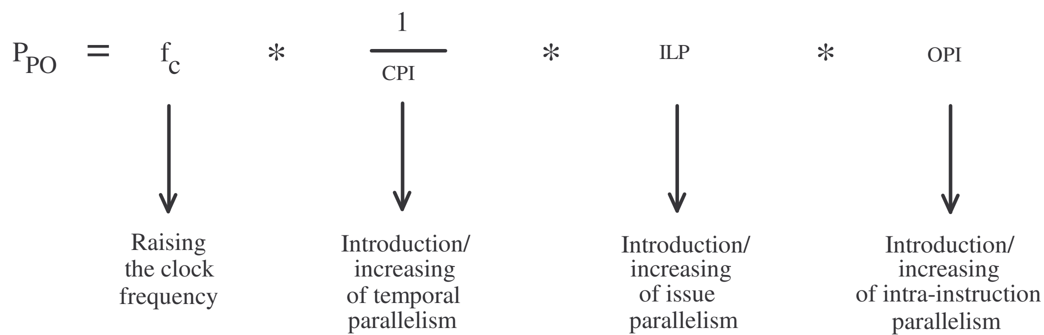



Figure 3: Main possibilities to increase processor performance

In subsequent sections we address each of these possibilities individually.

## III. INCREASING THE CLOCK FREQUENCY AND ITS IMPLICATIONS

*A. The growth rate of the clock frequency of microprocessors*

As an example, Figure 4 illustrates the phenomenal increase in the clock frequency of the Intel x86 line of processors [1] over the past two decades.

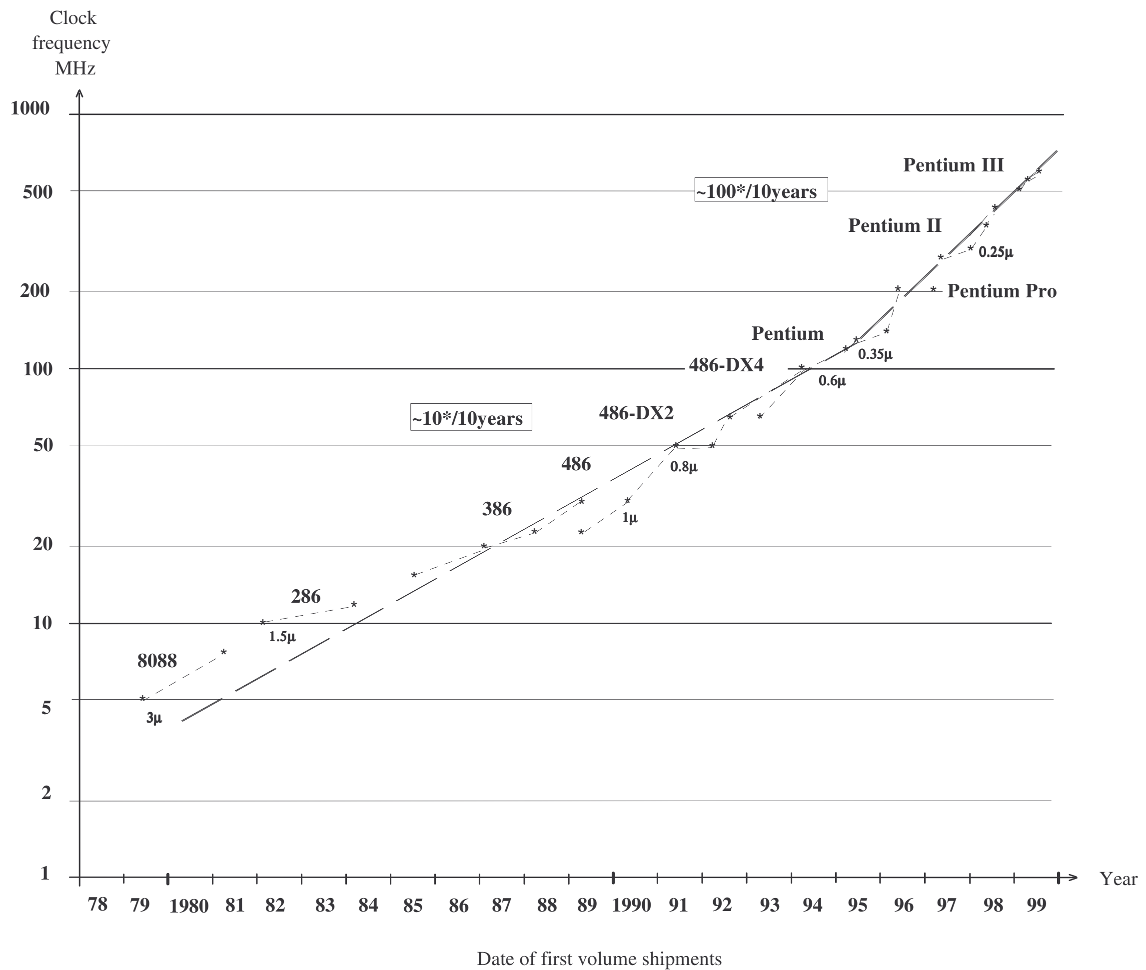



Figure 4: Historical increase in the clock frequency of Intel x86 processors

As Figure 4 indicates, the clock frequency was raised until the middle of the 1990s by approximately one order of magnitude per decade, and subsequently by about two orders of magnitude per decade. This massive frequency boost was achieved mainly by a

continuous downscaling of the chips through improved IC process technology, by using longer pipelines in the processors and by improving circuit layouts.

Since processor performance may be increased either by raising the clock frequency or by increasing the efficiency of the microarchitecture or both (see Figure 2), Intel's example of how it increased the efficiency of the microarchitecture in its processors is very telling.

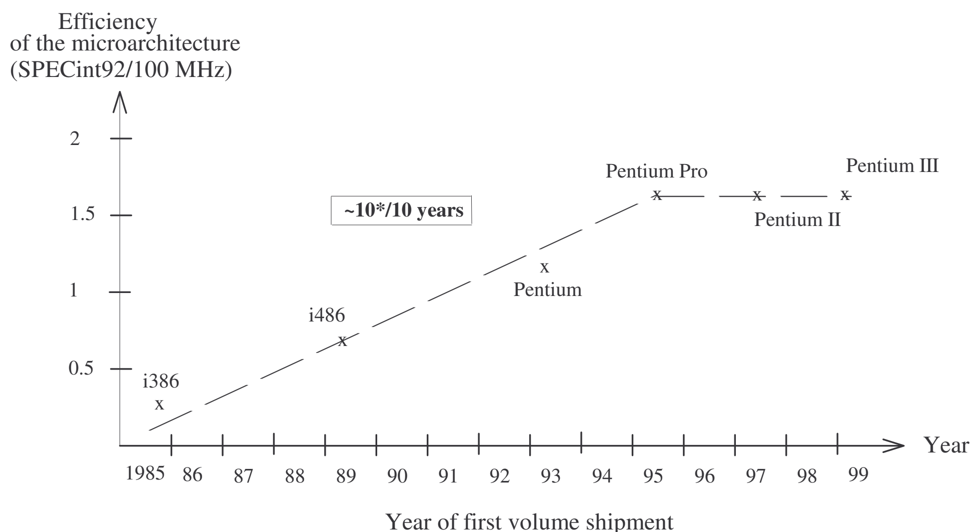



Figure 5: Increase in the efficiency of the microarchitecture of Intel's x86 line of processors

As Figure 5 shows, the overall efficiency (performance at the same clock frequency) of Intel processors [1] was raised between 1985 and 1995 by about an order of magnitude. During this period, both the clock frequency and the efficiency of the microarchitecture were increased approximately 10 times per decade, resulting in a performance boost of approximately two orders of magnitude per decade. However, after the introduction of the Pentium Pro (and until the arrival of the Pentium 4), Intel continued to use basically the same processor core in all of its Pentium II and Pentium III processors. The enhancements introduced—including multimedia (MM) and 3D support (SSE), doubling the size of both

level 1 instruction and data caches, etc. made only a marginal contribution to the efficiency of the microarchitecture in general purpose applications, as reflected in SPEC benchmark figures (see Figure 5). During this period of time Intel's design philosophy obviously prefered boosting clock frequency over enhancing microarchitecture efficiency. This decision may have stemmed from a view often emphasized by computer resellers: PC buyers usually go for clock rates and benchmark metrics rather than efficiency metrics.

We emphasize that the processor's clock frequency only indicates performance potential. Actual processor (or system) performance depends on the efficiency of the microarchitecture as well as on the characteristics of the application processed (as discussed in V.C.2). "Weak" components in the microarchitecture or in the entire system, such as an inadequate branch handling subsystem of the microarchitecture or a long latency cache in the system architecture may strongly impede performance.

### *B. Implications of increasing the clock frequency*

When increasing processor performance, either by raising the clock frequency or by increasing the throughput of the microarchitecture or by both, designers are forced *to enhance the system level architecture* as well in order to avoid arising bottlenecks. System level enhancements address principally the bus, memory and I/O subsystems. Since the evolution of the system level architecture is a topic of its own, whose complexity is comparable to the evolution of the microarchitectures, we do not go into details here, but indicate only a few dimensions of this evolution and refer to the literature given.

*1) Enhancing the bus subsystem:* For higher clock frequencies and for more effective microarchitectures, the bandwidth of the buses that connect the processor to the memory and the I/O subsystems needs to be increased for obvious reasons. This requirement has driven the evolution of front side processor buses (system buses), general purpose

peripheral buses (such as the ISA and the PCI buses), dedicated peripheral buses and ports intended to connect storage devices (IDE/ATA, SCSI standards), video (AGP), audio (AC'97) or low speed peripherals (USB bus, LPC port etc.). In order to exemplify the progress achieved, below is a diagram showing how the data width and the maximum clock frequency of major general purpose peripheral bus standards have evolved (see Figure 6).

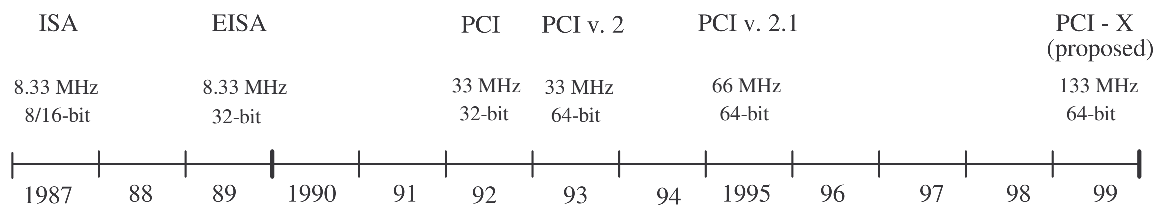



Figure 6: Evolution of major general purpose periheral buses

As depicted in the figure, the standardized 8/16-bit wide AT-bus, known as the ISA bus (International Standard Architecture) [25], was first extended to provide 32-bit data width (this extension is called the EISA bus [26]). The ISA bus was subsequently replaced by the PCI bus and its wider and faster versions, such as PCI versions 2, 2.1 [27] and the PCI-X proposal [28]. Figure 6 demonstrates that the maximum bus frequency was raised at roughly the same rate as the clock frequency of the processors.

*2) Enhancing the memory subsystem*: Higher clock frequencies and more efficient microarchitectures both demand higher bandwidth and reduced load-use latencies (the time needed to use requested data) from the memory subsystem. There is an impressive evolution along many dimensions towards achieving these goals, including (a) use of enhanced main memory components, such as FPM DRAMs, EDO DRAMs, SDRAMs,

RDRAMs, DRDRAMs [29], (b) introducing and enhancing caches, through improved cache organization, increasing the number of cache levels, implementing higher cache capacities, using directly connected or on-die level 2 caches etc., [30], [31] and (c) introducing latency reduction or hiding techniques, such as software or hardware controlled data prefetch, [32], [33], lock-up free (non-blocking) caches, out-of-order loads, speculative loads etc., as outlined later in Section V.E.5.b.

*3) Enhancing the I/O subsystem*: Concerning this point, we again do not delve into details, but rather just point out the spectacular evolution of storage devices (hard disks, CD-ROM players etc.) in terms of storage capacity and speed as well as the evolution of display devices in terms of their resolution etc. in order to better support more demanding recent applications such as multimedia, 3D graphics, etc.

# IV. INTRODUCTION OF TEMPORAL PARALLELISM AND ITS IMPLICATIONS

## *A. Overview of possible approaches to introduce temporal parallelism*

A traditional von Neumann processor executes instructions in a strictly sequential manner as indicated in Figure 7. For *sequential processing*, CPI, i.e. the average length of the issue intervals, equals the average execution time of the instructions. In the figure CPI = 4. Usually CPI >> 1.

*Assuming a given ISA*, CPI can be reduced by introducing some form of pipelining—in other words, by utilizing temporal parallelism. In this sense CPI reflects *the extent of*

*temporal parallelism* achieved in instruction processing, as already emphasized in Section II.

Basically, there are three main possibilities to overlap the processing of subsequent instructions. These are as follows: (a) overlapping the fetch phases and the last processing phase(s) of the preceding instruction, (b) overlapping the execute phases of subsequent instructions processed in the same execution unit (EU) by means of pipelined execution units, or (c) overlapping all phases of instruction processing using pipelined processors, as shown in Figure 7.

The arrows in the figure represent instructions to be executed. For illustration purposes we assume that instructions are processed in four subsequent phases, called the Fetch (F), Decode (D), Execute (E) and Write (W) phases.

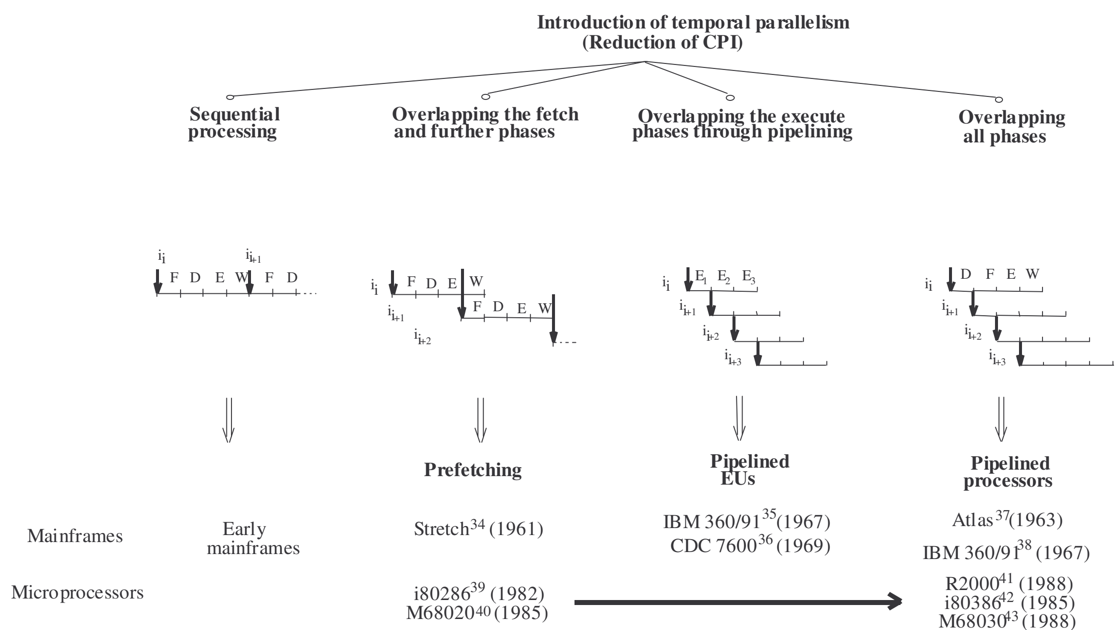



Figure 7: Main approaches to achieve temporal parallelism

(F: fetch phase, D: decode phase, E: execute phase, W: write phase)

The superscripts following machine or processor designations are references to the applicable machines or processors.

Dates in this and all subsequent figures indicate the year of first shipment (in the case of mainframes) or that of first volume shipment (in the case of microprocessors).

(a) *Overlapping the fetch phases and the last phase(s) of the preceding instruction* is called *prefetching,* a term coined in the early days of computing [34]. If the processor overlaps the fetch phases with the write phases, as indicated in Figure 7, the average execution time is reduced by one cycle compared to fully sequential processing. However, the execution of control transfer instructions (CTIs) lessens the achievable performance gain of instruction prefetching to less than one cycle per instruction, since CTIs divert instruction execution from the sequential path and thus render the prefetched instructions obsolete.

(b) The next possibility is *to overlap the execution phases of subsequent instructions* processed in the same pipelined execution unit (EUs) [35], [36]. *Pipelined EUs* execute a new instruction ideally in every new clock cycle, provided that subsequent instructions are independent. Clearly, pipelined EUs are very effective in processing vectors.

(c) Finally, the ultimate solution to exploit temporal parallelism is *to extend pipelining to all phases of instruction processing*, as indicated in Figure 7 [37], [38]. Fully *pipelined instruction processing* ideally results in a one cycle mean time between subsequent instructions (CPI = 1), provided that the instructions processed are free of dependencies. The related processors are known as *pipelined processors*, and contain one or more pipelined EUs.

We note that even in pipelined instruction processing the execution phase of some complex instructions, such as division or square root calculation, is not pipelined for the

sake of implementation efficiency. This fact and the occurrence of dependencies between subsequent instructions result in CPI values higher than 1 in real pipelined processors.

Although both prefetching and overlapping of the execution phases of subsequent instructions already represent a partial solution to parallel execution, processors providing these techniques alone are usually not considered to be instruction level parallel processors (ILP processors). On the other hand, pipelined processors are considered to belong to the *ILP processor* category.

Temporal parallelism was introduced first in mainframes (in the form of prefetching) in the early 1960's (see Figure 7). In microprocessors, prefetching arrived two decades later with the advent of 16-bit micros [39], [40]. Subsequently, pipelined microprocessors emerged and became the main road of the evolution because of their highest performance potential among the alternatives discussed [41] - [43]. They came into widespread use in the second half of the 1980s, as shown in Figure 8. We point out that pipelined microprocessors represent the second major step on the main road of microprocessor evolution. In fact, the very first step of this evolution was increasing the word length gradually from 4 bits to 16 bits, as exemplified by the Intel processors 4004, [44], 8008, 8080 and 8086 [45]. This evolution gave rise to the introduction of a new ISA for each wider word length until 16-bit ISAs arrived. For this reason, while focusing on performance issues, we discuss the evolution of the microarchitecture of microprocessors beginning with 16-bit processors.

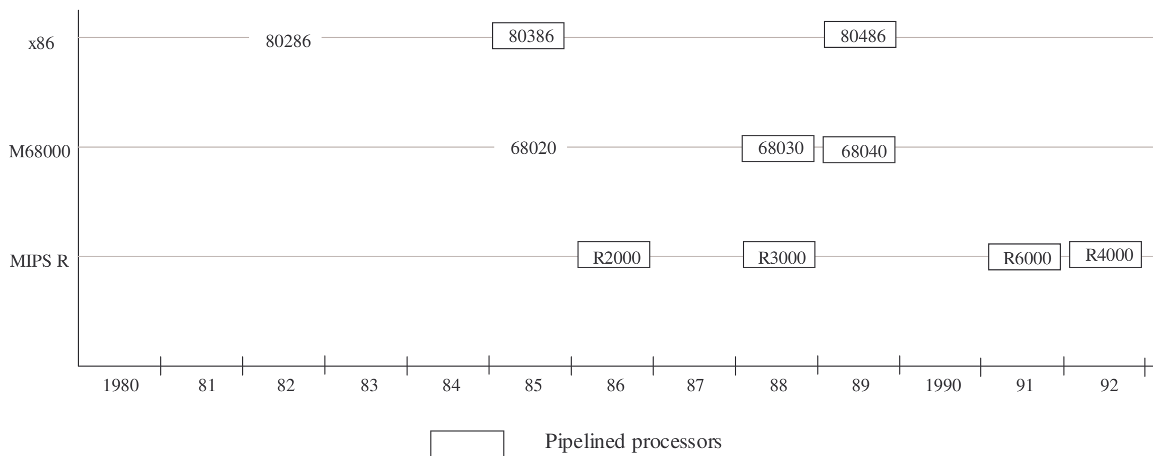



Figure 8: The introduction of pipelined microprocessors

*B. Implications of the introduction of pipelined instruction processing*

*1) Overview:* Pipelined instruction processing calls for higher memory bandwidth and smart processing of CTI's (control transfer instructions), as detailed below. The basic techniques needed to avoid processing bottlenecks due to the requirements mentioned above are caches and speculative branch processing.

*2) The demand for higher memory bandwidth and the introduction of caches*: A pipelined processor fetches a new instruction in every new clock cycle, provided that subsequent instructions are independent. This fact means that higher memory bandwidth is required for fetching instructions in comparison to sequential processing. Furthermore, pipelined instruction processing also increases the frequency of load and store instructions and, in the case of CISC architectures, the frequency of referenced memory operands. Consequently, pipelined instruction processing requires higher memory bandwidth for both instructions and data. As the memory is typically slower than the processor, the increase of the memory bandwidth requirement of pipelined instruction processing accelerated and inevitably brought about the introduction of *caches,* an innovation pioneered in the IBM

360/85 [46] in 1968. With caches, frequently used program segments (cycles) can be held in fast memory, which allows instruction and data requests to be served at a higher rate. Caches came into widespread use in microprocessors in the second half of the 1980s, essentially along with the introduction of pipelined instruction processing (see Figure 9). As the performance of microprocessors is increasing by a rate of about two orders of magnitude per decade (see Section A), there is a continuous demand to raise the performance of the memory subsystem as well. As a consequence, the enhancement of caches and their connection to the processor has remained one of the focal points of microprocessor evolution for more than one decade now.

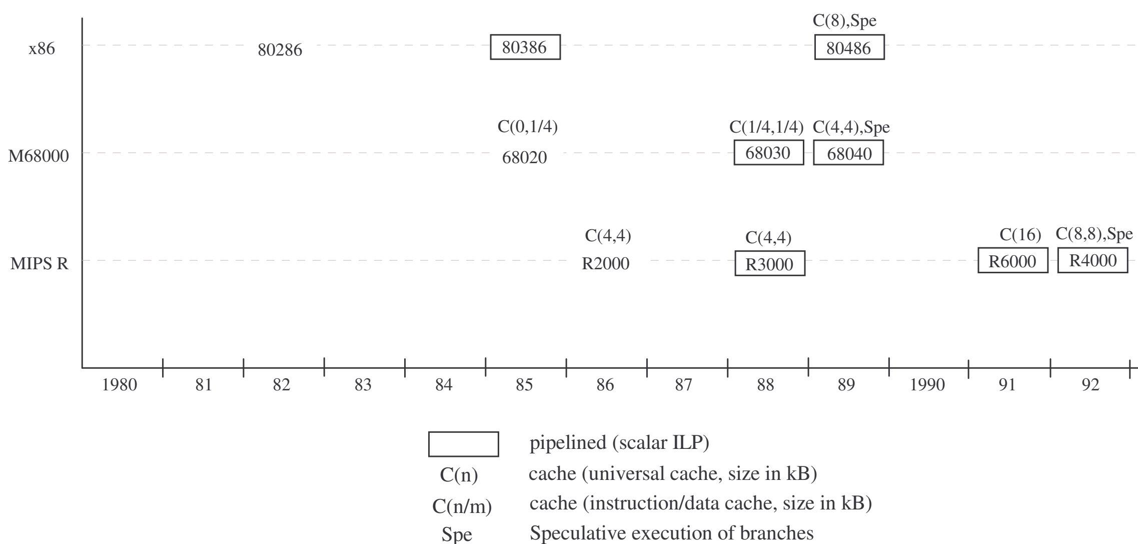



Figure 9: The introduction of caches and speculative branch processing

*3) Performance degradation caused by unconditional CTI's and the introduction of speculative branch processing:* The main problem with pipelined processing of *unconditional CTI's* is as follows. If the processor executes CTI's in a straightforward way, then by the time it recognizes a CTI in the decode stage, it will already have fetched

the next sequential instruction. As the unconditional CTI directs the processor to branch, the next instruction to be executed is the branch target instruction rather than the next sequential one, which is already fetched. Then this sequential instruction needs to be canceled and at least one wasted cycle, also known as a bubble, appears.

*Conditional CTIs* can cause even more wasted cycles. Consider here that for each conditional CTI the processor needs to know the specified condition prior to deciding whether to issue the next sequential instruction or to fetch and issue the branch target instruction. Thus each unresolved conditional branch would basically lock up the issue of instructions until the processor can decide whether the sequential path or the branch target path needs to be followed. Consequently, if a conditional CTI refers to the result of a long latency instruction, such as a division, dozens of wasted cycles would occur.

*Speculative execution of branches* or briefly speculative branching [47]–[50] can remedy this problem. Speculative branching requires the microarchitecture to make a guess for the outcome of each conditional branch and resume instruction processing along the estimated path. Assuming the use of this technique, conditional branches no longer hinder instruction issue, as demonstrated in Figure 10. Notice that in the figure the speculation goes only until the next conditional branch.

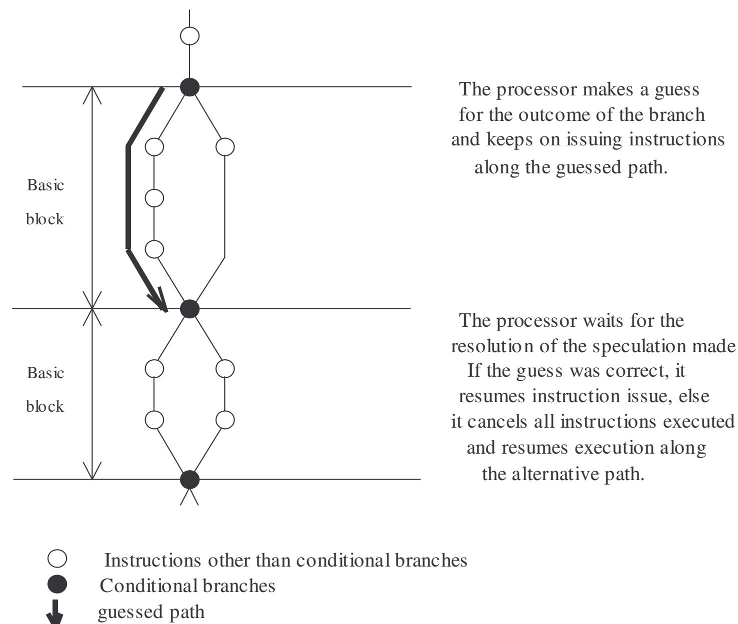



Figure 10: The principle of speculative execution assuming speculation along a single conditional branch

Later, when the specified condition becomes known, the processor checks the guess made. For a correct guess it acknowledges the instructions processed. Otherwise it cancels incorrectly executed instructions and resumes execution along the correct path.

In order to exploit the intrinsic potential of pipelined instruction processing, designers introduced both caches and speculative branch processing at about the same time, as Figure 9 demonstrates.

*4) Limits of utilizing temporal parallelism:* With the massive incorporation of temporal parallelism into instruction processing, the average length of issue intervals can be reduced to almost one clock cycle. However, CPI = 1 marks the absolute limit achievable through temporal parallelism. Any further substantial performance increase calls for the introduction of parallel operation along another dimension. There are two possibilities for

this: either to introduce issue parallelism or intra-instruction parallelism. Following the evolutionary path of microprocessors, we first discuss the former alternative.

# V. THE INTRODUCTION OF ISSUE PARALLELISM AND ITS IMPLICATIONS

## *A. The introduction of issue parallelism*

Issue parallelism, also known as *superscalar instruction issue* [5], [51], [52], refers to the capability of the processor to issue multiple decoded instructions per clock cycle from the decode unit for further processing. The peak rate of instructions issued per clock cycle is called the *issue rate* ($n_{ir}$).

After designers exhausted the full potential of pipelined instruction processing around 1990 the introduction of issue parallelism became the main option of increasing processor performance. Due to their higher performance over pipelined processors, superscalars rapidly began to dominate all major processor lines, as Figure 11 shows.

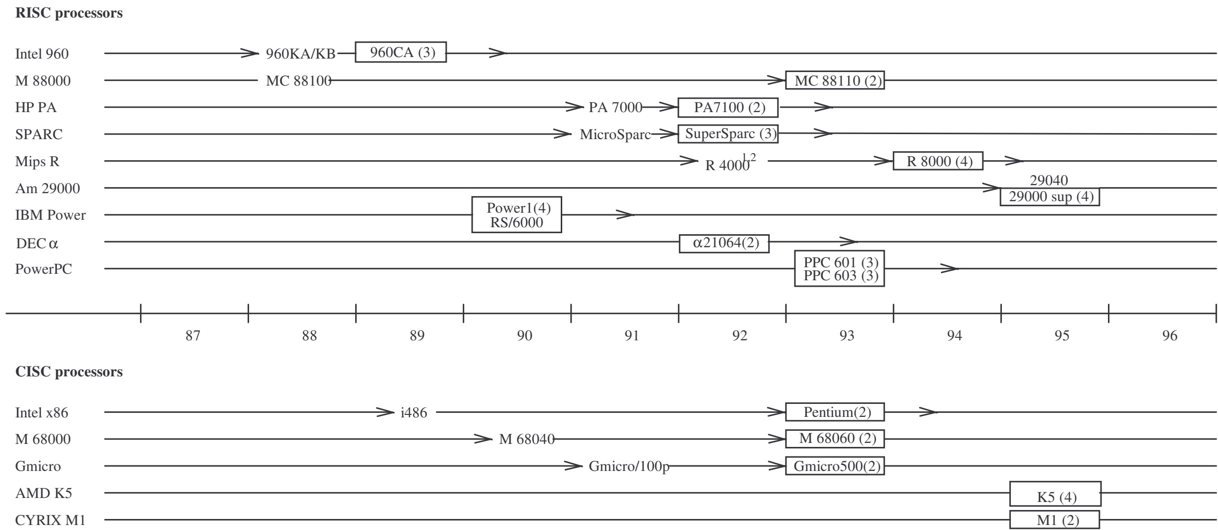



Figure 11: The appearance of superscalar processors

### *B. Overall implications of superscalar issue*

The main components of processor performance were identified in expression (5). Here *issue parallelism* is expressed by the average number of instructions issued per issue interval (ILP) rather than by the average number of instructions issued per clock cycle (IPC). However, assuming both pipelined instruction processing and superscalar instruction issue, the average length of issue intervals (CPI) approaches one cycle. Thus for superscalar processors ILP in expression (5) roughly equals the average number of instructions issued per clock cycle (IPC):

$$IPC \sim ILP$$

Unlike pipelined processors that issue at most one instruction per cycle for execution, superscalars issue up to $n_{ir}$ instructions per cycle, as illustrated in Figure 12. As a consequence, superscalars must be able to fetch $n_{ir}$ times as much instructions and memory

data and must store $n_{ir}$ times as much memory data per cycle ($t_c$) than pipelined processors. In other words, superscalars require $n_{ir}$ times *higher memory bandwidth* than pipelined processors at the same clock frequency. As clock frequencies of processors are rapidly increasing over time as well (see Figure 4), superscalars that arrived after pipelined processors, definitely need a highly enhanced memory subsystem compared to those used with pipelined processors, as already emphasized while discussing the main road of microarchitecture evolution in Section III.B.2.

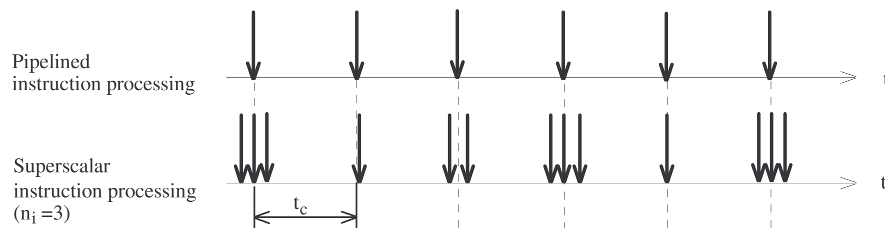



Figure 12: Contrasting pipelined instruction processing with superscalar processing

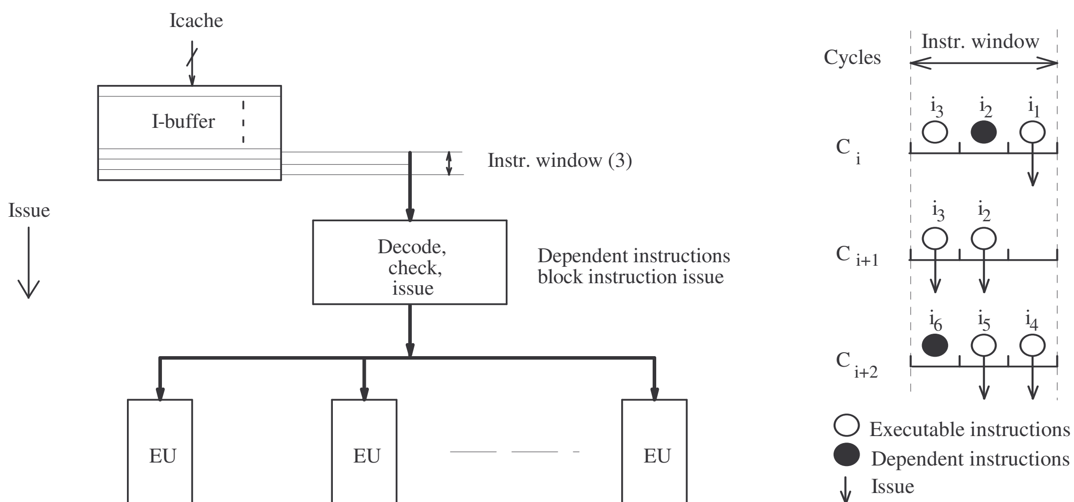
(arrows indicate instructions)

Superscalar issue also impacts branch processing. There are two reasons for this. First, branches occur up to $n_{ir}$ times more frequently with superscalar instruction issue than with scalar pipelined processing. Second, each wasted cycle that arises during branch processing can restrict multiple instructions from being issued. Consequently, superscalar processing requires *more accurate branch speculation* or, in general, *more advanced branch handling* than is used with pipelined processing. Thus superscalar instruction issue also gave rise to an impressive evolution of the *branch handling* subsystem. For an overview of the progress achieved so far we refer to [49], [53] - [55].

*C. The direct issue scheme and the resulting issue bottleneck*

1) *The principle of the direct issue scheme:* While issuing multiple instructions per cycle early superscalars typically used some variants of the *direct issue scheme* in conjunction with a simple *branch speculation* [52]. Direct issue means that decoded instructions are issued immediately, i.e. without buffering, to the execution units (EU's), as shown in Figure 13.

The issue process itself can best be described by introducing the concept of the *instruction window* (issue window). The instruction window, whose width equals the issue rate ($n_{ir}$), contains the last $n_{ir}$ entries of the instruction buffer. The instructions held in the window are decoded and checked for dependencies. Executable instructions are issued from the instruction window directly to free EU's, whereas dependent instructions remain in the window until existing dependencies become resolved. Variants of this scheme differ on two aspects: how dependent instructions affect the issue of subsequent executable instructions held in the window [49], [52] and how the window is shifted after issuing instructions.



(a): Simplified structure of a superscalar microarchitecture that employs the direct issue scheme and has an issue rate of three

(b): The issue process

Figure 13: Principle of the direct issue scheme

In Figure 13b we demonstrate the direct issue scheme for an issue rate of three ($n_{ir} = 3$) with the following two assumptions: (a) the processor issues instructions in order, meaning that a dependent instruction blocks the issue of all subsequent independent instructions from the window, and (b) the processor needs to issue all instructions from the window before shifting it along the instruction stream. Examples of processors that issue instructions this way are the Power1, the PA7100, and the SuperSparc. In the figure we assume that in cycle $c_i$ the instruction window holds instructions $i_1$–$i_3$. If in cycle $c_i$ instructions $i_1$ and $i_3$ are free of dependencies, but $i_2$ depends on instructions that are still in execution, only instruction $i_1$ can be issued in cycle $c_i$, but both $i_2$ and $i_3$ will be withheld in the window, since $i_2$ is dependent and blocks the issue of any subsequent instruction. Let us assume that in the next cycle ($c_{i+1}$) $i_2$ becomes executable. Then in cycle $c_{i+1}$ instructions $i_2$ and $i_3$ will be issued for execution as well. In the next cycle ($c_{i+2}$) the window is shifted by three along the instruction stream, so it then holds the subsequent three instructions ($i_4$–$i_6$) and the issue process resumes in a similar way.

2) *The throughput of superscalar microarchitectures that use the direct issue scheme:* As far as the throughput (IPC) of the microarchitecture is concerned, the microarchitecture may best be viewed as a chain of subsystems linked together via buffers. Instructions are processed in the microarchitecture by flowing through the subsystems in a pipelined fashion. These subsystems are typically responsible for fetching, decoding and/or issuing,

executing and finally retiring (i.e. completing in program order) instructions. The kind and number of subsystems depend on the microarchitecture in question.

A simplified *execution model of a superscalar RISC processor* that employs the direct issue scheme is shown in Figure 14 below. Basically, the microarchitecture consists of a front and a back end that are connected by the instruction window. The *front end* consists of the fetch and decode subsystems, and its task is to „fill" the instruction window.

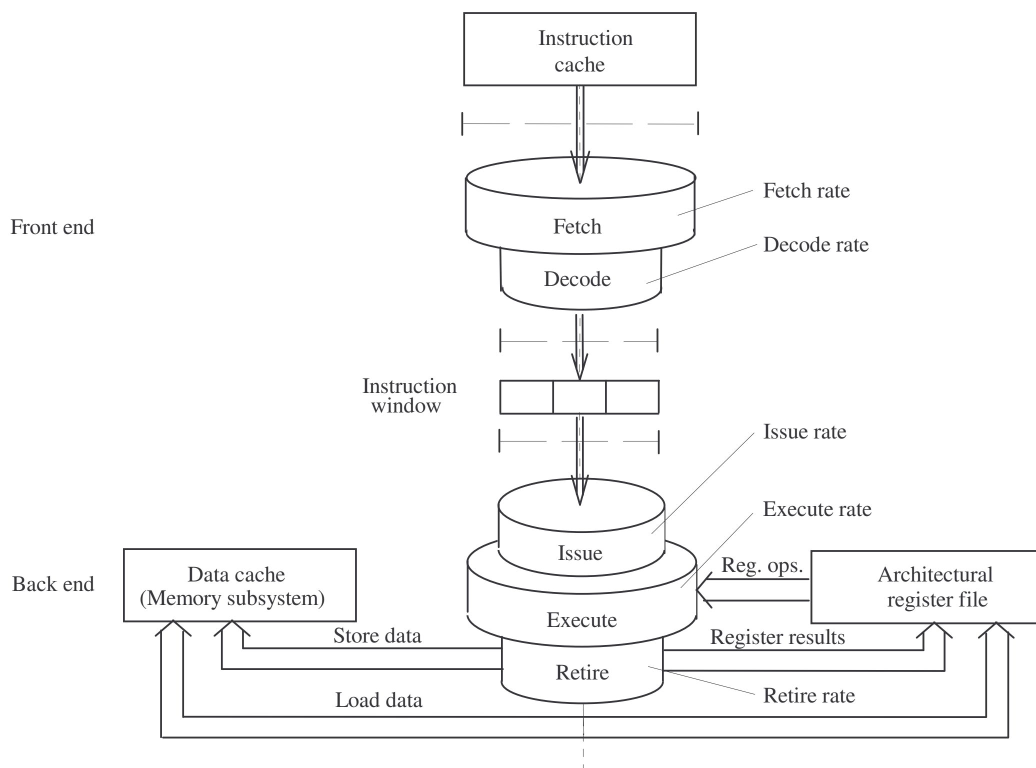



Figure 14: Simplified execution model of a superscalar RISC processor that employs direct issue

The instruction window is „depleted" by the *back end* of the microarchitecture that also takes care of executing the issued instructions. The back end contains the issue, execute and retire subsystems. The issue subsystem forwards executable instructions from the instruction window to the execute subsystem. The execute subsystem performs the

operations required, where referenced register operands are supplied from the architectural register file to the EU's. Finally, executed instructions are completed by the retire subsystem in program order and the results generated are sent either to the architectural register file or to the memory.

We note that the microarchitecture of advanced CISC processors shows some differences in comparison to RISC processors. Advanced CISC's usually convert CISC instructions into simple RISC-like internal operations. Denoted differently in different processor lines (e.g. "µops" in Intel's Pentium Pro and subsequent models, "RISC86 operations" in AMD's K5–K7, and "ROP's" in Cyrix's M3), these internal operations are executed by a RISC kernel. For CISC processors the retire subsystem also performs a "reconversion" by completing those internal operations that belong to the same CISC instruction together. Thus the execution model of RISC processors is basically valid for CISC processors as well.

Let us now discuss the notions of throughput and the "width" of the microarchitecture in relation to the execution model presented.

Each subsystem has a *maximum throughput* in terms of the maximum number of instructions that may be performed per cycle. Maximum throughput values of each subsystem are disignated as the fetch, decode, issue, execution and the retire rate, respectively, as indicated in Figure 14. Now, the maximum throughput of a subsystem or of the entire microarchitecture can be interpreted as its *width*. Therefore, the width of the fetch, decode, issue, execute and retire subsystems is represented by their respective rates. Clearly, the *width of the entire microarchitecture* is determined by the smallest value of its subsystems. This notion is analogous to the notion of "word length of a processor" that indicates the characteristic length of instructions and the data processed.

In fact, the width of a subsystem only indicates its *performance potential*. When running an application, subsystems have actually less throughput, since they usually operate under worse than ideal conditions. For instance, branches decrease the actual throughput of the fetch subsystem; or the actual throughput of the issue subsystem depends on the number of parallel executable instructions available in the window from one cycle to the next. In any application, the smallest throughput of any subsystem will be the *bottleneck* that determines the resulting throughput (IPC) of the entire microarchitecture.

3) *The issue bottleneck of the direct issue scheme:* In each cycle some instructions in the instruction window are available for parallel execution, while others are locked by dependencies. As EU's finish the execution of instructions, existing dependencies become resolved and formerly dependent instructions become available for parallel execution. Clearly, a crucial point for the throughput of the microarchitecture is the average number of instructions that are available for parallel execution in the instruction window per cycle. In the direct issue scheme all data or resource dependencies occurring in the instruction window block instruction issue. This actually limits the average number of issued instructions per cycle (ILP) to about two in general purpose applications [56], [57]. Obviously, when the microarchitecture is confined to issue only up to approx. two instructions per cycle on average, its throughput is also limited to about two instructions per cycle, no matter how wide other subsystems of the microarchitecture are. Consequently, the direct issue scheme leads to an *issue bottleneck* that severely limits the maximum throughput of the microarchitecture.

In accordance with this restriction, early superscalars usually have an issue rate of two to three (as indicated in Figure 11). Consequently, their execution subsystems typically

consist of either two pipelines (Intel's Pentium, Cyrix's M1) or two to four dedicated pipelined EU's (such as e.g. in DEC's (now Compaq's) Alpha 21064).

In order to increase the throughput of the microarchitecture, designers had to remove the issue bottleneck and at the same time increase the throughput of all relevant subsystems of the microarchitecture. In the subsequent section we focus on the first topic, while the second issue is discussed in Section E.

*D. Basic techniques introduced to remove the issue bottleneck and to increase the number of parallel executable instructions in the instruction window.*

1) *Overview:* The issue bottleneck can be addressed primarily by using *dynamic instruction scheduling*. However, in order to effectively capitalize on this technique, dynamic instruction scheduling is usually augmented by *register renaming*. Furthermore, the processor is assumed to make use of *speculative execution of branches*, a technique already introduced in pipelined processors.

2) *Dynamic instruction scheduling*: The key technique used to remove the issue bottleneck is *dynamic instruction scheduling,* also known as shelving [4], [5], [58]. Dynamic instruction scheduling means buffered instruction issue. It presumes the availability of dedicated buffers, called *issue buffers* (or "reservation stations" in specific implementations) in front of the EU's, as shown e.g. in Figure 15[2]. With dynamic instruction scheduling the processor first issues the instructions into available issue buffers without checking either for data or control dependencies or for busy EU's. As data

[2] Here we note that beyond individual reservation stations that serve individual EU's as shown in Figure 15, there are a number of other solutions to implement dynamic instruction scheduling [49], [58]. For instance, the prevailing solution is either to use group reservation stations serving EU's of the same type (e.g. fixed point units) or to have centralized reservation stations (unified reservation stations), as implemented in Intel's Pentium Pro, Pentium II and Pentium III processors.

dependencies or busy execution units no longer restrict the flow of instructions, the issue bottleneck of the direct issue scheme is removed.

With dynamic instruction scheduling the processor is able to issue as many instructions into the issue buffers as its issue rate (usually 4) in each cycle, provided that no hardware restrictions occur. Possible hardware restrictions include missing free issue buffers or datapath width limitations. Nevertheless, in a well-designed microarchitecture the hardware restrictions mentioned will not severely impede the throughput of the issue subsystem. Issued instructions remain in the issue buffers until they become free of dependencies and can be *dispatched* for execution.

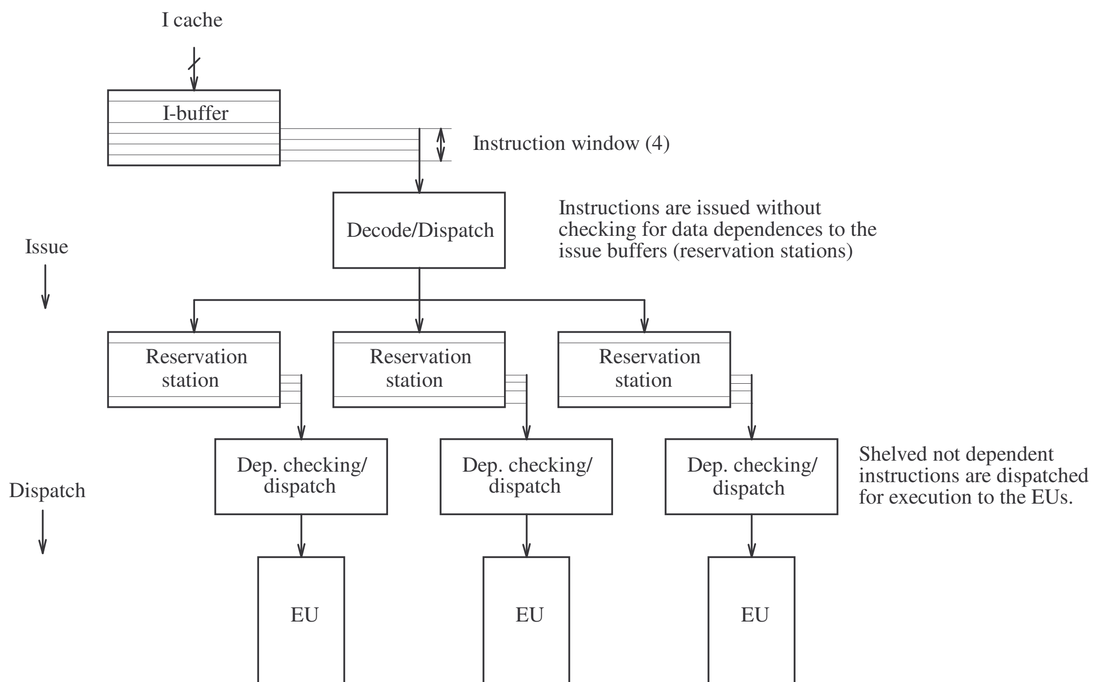



Figure 15: The principle of dynamic instruction scheduling, assuming that the processor has individual issue buffers (called reservation stations) in front of the execution units.

Dynamic instruction scheduling improves the throughput of the front end of the microarchitecture not only by removing the issue bottleneck of the direct issue scheme but

also by significantly *widening the instruction window*. Under the direct issue scheme the processor attempts to find executable instructions in a small instruction window whose width equals the processor's issue rate (usually 2–3). In contrast, when dynamic instruction scheduling is used, the processor scans the issue buffers for executable instructions. This way the width of the instruction window is determined by the total capacity of all issue buffers available, while its actual width equals the total number of instructions held in the window (which may change dynamically from one cycle to the next). As processors usually contain dozens of issue buffers, dynamic instruction scheduling greatly widens the instruction window in most cases compared to the direct issue scheme. Since the processor will find in a wider window on average more parallel executable instructions per clock cycle than in a smaller one, dynamic scheduling increases the throughput of the front end of the microarchitecture even more.

*3) Register renaming:* This is another technique used to increase the efficiency of dynamic instruction scheduling. Register renaming removes false data dependencies, i.e. write after read (WAR) and write after write (WAW) dependencies between register operands of subsequent instructions. If the processor uses renaming, it allocates to each destination register a rename buffer that temporarily holds the result of the instruction. It also tracks current register allocations, fetches source operands from renamed and/or architectural registers, writes the results from the rename buffers into the addressed architectural registers and finally reclaims rename buffers that are no longer needed. Renaming must also support a recovery mechanism for erroneously speculated branches or interrupts accepted [4], [5], [49].

The processor renames destination and source registers of instructions during instruction issue. As renaming removes all false register data dependencies between the instructions

held in the instruction window, it considerably increases the average number of instructions available in the instruction window for parallel execution per cycle.

Figure 16 tracks the introduction of dynamic instruction scheduling and renaming in major superscalar lines. As indicated, early superscalars (the "first wave") typically made use of the direct issue scheme. A few subsequent processors introduced either renaming alone (like the PowerPC 602 or the M1) or dynamic instruction scheduling alone (such as the MC88110, R8000). In general, however, dynamic instruction scheduling and renaming emerged together in a "second wave" of superscalars in the mid-1990's.

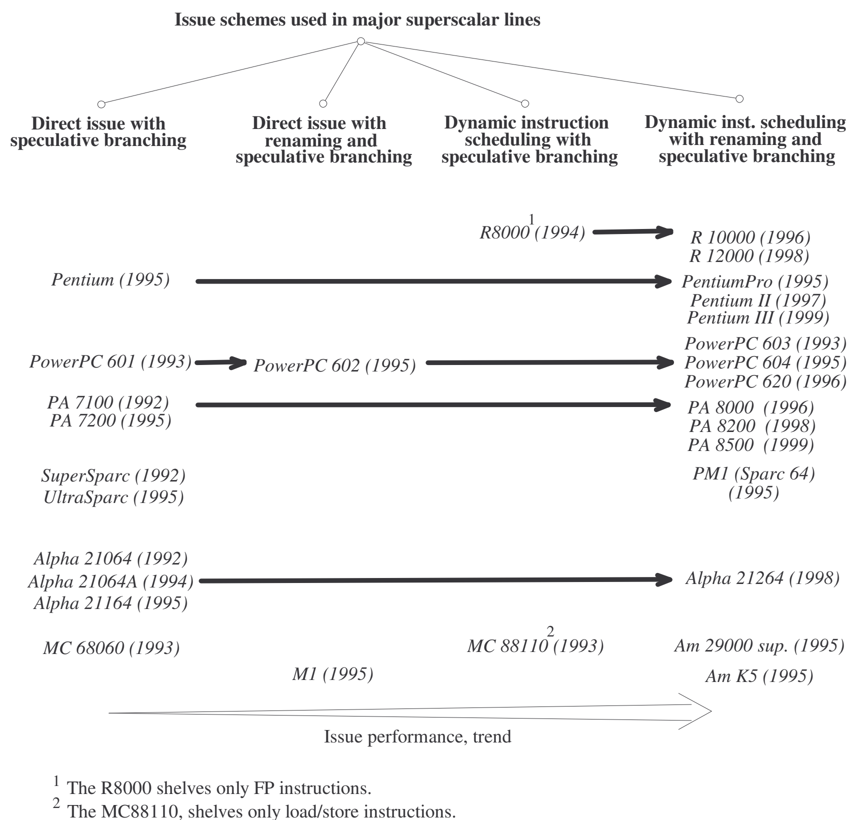



Figure 16: Introduction of dynamic instruction issue and renaming in superscalar processors

4) *Advanced speculative branching:* Wide instruction windows, however, require speculation along multiple conditional branches—called *deep speculation*—in order to avoid stalling instruction issue due to multiple consecutive conditional branches. However, the deeper branch speculation (i.e. the more consecutive branches a guessed path may involve), the higher the penalty for wrong guesses in terms of wasted cycles. As a consequence, *dynamic instruction scheduling calls for deep speculation and highly accurate branch prediction*. For this reason, the design of effective branch prediction techniques has been a major cornerstone in the development of high performance superscalars. For more details of advanced branch speculation techniques we refer to the literature [53] - [55].

5) *The throughput of superscalar microarchitectures that use dynamic instruction scheduling and renaming:* RISC processors providing dynamic instruction scheduling and renaming are usually four instructions wide by design, which means that their fetch rate, decode rate, rename rate, dispatch rate and retire rate all equal four instructions per cycle.

In Figure 17 we show a simplified *execution model of superscalar RISC processors that use dynamic instruction scheduling and renaming*. In this model the front end of the microarchitecture contains the fetch, decode, rename and the issue subsystems. The front end feeds instructions into the issue buffers constituting the instruction window.

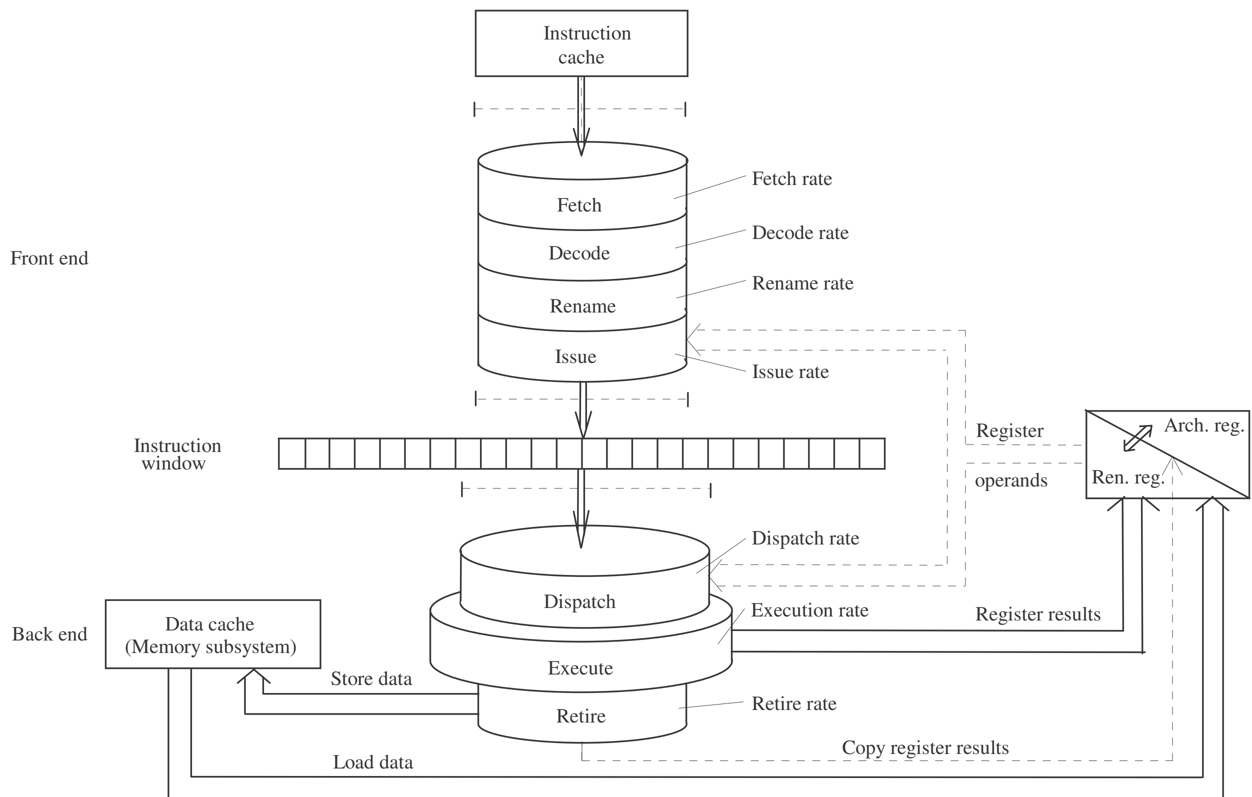



Figure 17: Simplified execution model of a superscalar RISC processor that employs both dynamic instruction scheduling and renaming

Executable instructions are dispatched from the window to available EU's by the dispatch subsystem. Referenced register operands are supplied either during instruction issue or during instruction dispatch. The execute subsystem performs the operations as required. Register results and fetched memory data are forwarded to the rename registers that temporarily hold all register results. Finally, executed instructions are retired in program order by the retire subsystem. Register results are copied at this stage from rename registers to the corresponding architectural registers, and memory data are forwarded to the data cache in program order.

We note that dispatch rates are typically higher than issue rates, as indicated in Figure 17. In most cases, the dispatch rate is five to eight instructions per cycle (see Table 1). There are two reasons for this: (a) to sustain a high enough execution width even though complex instructions that often have much higher repetition rates than one cycle (like division, square root etc.); and (b) to provide ample execution resources (EU's) for a wide variety of possible mixes of dispatched instructions. Execution rates are usually even higher than dispatch rates, because multiple multi-cycle EU's are typically able to operate in parallel, but for cost reasons they often share the same bus, which allows only one instruction to be issued to them per cycle.

| Comparison of issue and dispatch rates of recent superscalar processors | | |
|---|---|---|
| Processors/year of volume shipment | Issue rate (instr./cycle) | Dispath rate [a] (instr./cycle) |
| PowerPC 603 (1993) | 3 | 3 |
| PowerPC 604 (1995) | 4 | 6 |
| Power2 (1993) | 4/6 [b] | 10 |
| Nx586 (1994) | 3/4 [c,d] | 3/4 [c,d] |
| K5 (1995) | 4 [d] | 5 [d] |
| PentiumPro (1995) | 3 | 5 [d] |
| PM1 (Sparc 64) (1995) | 4 | 8 |
| PA8000 (1996) | 4 | 4 |
| R10000 (1996) | 4 | 5 |
| Alpha 21264 (1998) | 4 | 6 |

[a] Because of address calculations performed separately, the given numbers are usually to be interpreted as operations/cycle. For instance, the Power2 performs maximum 10 operations/cycle, which corresponds to 8 instr./cycle.
[b] The issue rate is 4 for sequential mode and 6 for target mode.
[c] Both rates are 3 without an optional FP-unit (labelled Nx587) and 4 with it.
[d] Both rates refer to RISC operations (rather than to the native CISC operations) performed by the superscalar RISC core.

Table 1: Issue and dispatch rates of superscalar processors

As far as advanced CISC processors with dynamic instruction scheduling and renaming are concerned, they typically decode up to three CISC instructions per clock cycle and usually perform an internal conversion to RISC-like operations, as discussed earlier. As x86 CISC instructions generate on average approx. 1.2–1.5 RISC-like instructions [59], the front end of advanced CISC processors has roughly the same width than that of advanced RISC processors in terms of RISC-like operations.

Another interesting consideration is how the introduction of dynamic instruction scheduling and renaming contributes to increasing the *efficiency of microarchitectures*. In Figure 18 we show the relative cycle-by-cycle performance of processors in terms of their SPECint95 scores standardized to 100 MHz. Designs using dynamic scheduling and renaming are identified by outlined processor designations. As this figure demonstrates, superscalars featuring dynamic scheduling and renaming have a true advantage over microarchitectures using direct issue. Models comparable in this respect are e.g. Pentium vs. Pentium Pro, PowerPC 601 vs. PowerPC 604, PA7100 vs. PA8000, R8000 (which “shelves” only FP instructions) and R10000 or Alpha 21064 vs. Alpha 21264. These comparisons are slightly distorted due to the fact that designs with dynamic instruction scheduling are typically wider than microarchitectures with direct issue. In order to include this aspect, we also indicate the issue rates of the processors after the processor designations in brackets (see Figure 18).

We note that the UltraSparc superscalar family is the only line that has not yet introduced dynamic scheduling and renaming. In order to reduce time-to-market, designers ruled out a “shelved” design at the beginning of the design process [60]. This caps the cycle-by-cycle throughput of the UltraSparc line well below comparable advanced RISC

designs that make use of both dynamic scheduling and renaming (such as the R12000, the PA 8200 and PA8500 or the Alpha 21264).

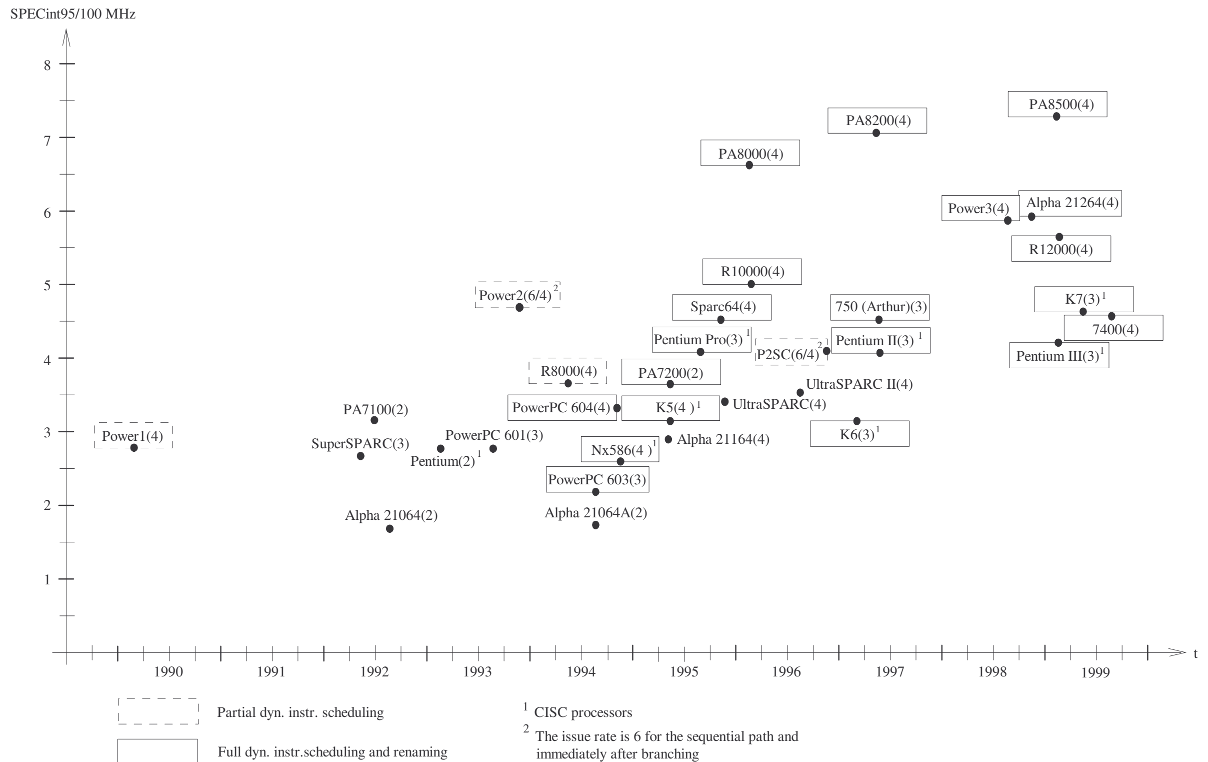



Figure 18: Efficiency of microarchitectures

Finally, we point out one important feature characterizing the internal operation of superscalars that use dynamic instruction scheduling, renaming and speculative branch processing. If all these techniques are used, only RAW dependencies between register operands as well as memory data dependencies restrict the processor from executing instructions in parallel from the instruction window (not considering any obvious hardware limitations). Consequently, the microarchitecture executes instructions with register operands (and literals) internally according to the *dataflow principle of operation*. Then

basically only producer-consumer type register data dependencies set the *dataflow limit of execution.*

### *E. Approaches to increase the throughput of particular subsystems of superscalar microarchitectures*

*1) Overview*: Raising the throughput of the microarchitecture is a real challenge, as it requires a properly orchestrated enhancement of all subsystems involved. In addition to dynamic instruction scheduling and renaming there are a number of techniques that have been used or proposed to increase the throughput of particular subsystems. Below we give an overview of these possibilities.

*2) Increasing the throughput of the instruction fetch subsystem:* Ideally, the instruction fetch subsystem supplies instructions for processing at the fetch rate. However, conditional branches or cache misses may interrupt the continuous stream of instructions for a large number of cycles. Designers introduced a handful of advanced techniques to cope with these challenges, including: (a) more intricate branch handling schemes, as already discussed, (b) diverse techniques to access branch target paths as quickly as possible using Branch History Tables, Branch Target Buffers, Subroutine Return Stacks etc. [49], (c) various instruction prefetch schemes to reduce latencies incurred by cache misses [33], and (d) trace caches [10], [11], [12]. Current processors improve the throughput of the fetch subsystem by continuously refining these techniques.

*3) Increasing the throughput of the decode subsystem:* With superscalar instruction issue, multiple instructions need to be decoded per cycle, so decoding becomes much more complex than in scalar processors. Moreover, assuming dynamic instruction scheduling and renaming, a time critical path arises that consists of decoding, renaming and issuing the instructions to the issue buffers. A variety of checks need to be carried out along this path to see whether there are enough empty rename or issue buffers, or whether required buses are wide enough to forward multiple instructions into the same buffer, etc. As a

consequence, higher issue rates (3 or higher) can unduly lengthen this time critical path. Pipelined instruction processing segments this path into decode, rename and issue subtasks, where each subtask takes one or more clock cycles (pipeline stages) to perform the particular subtasks mentioned. If, assuming higher issue rates, the time to perform one of the subtasks becomes longer, either the clock frequency must be lowered or additional clock cycle slots (pipeline stages) need to be included, which unfortunately also increases the penalty for mispredicted branches. An appropriate technique to avoid lengthened decode times with higher issue rates is known as predecoding [49].

The fundamental idea behind *predecoding* is to reduce the complexity of the decode stage by partially decoding instructions while fetching them into the instruction buffer, as indicated in Figure 19. The results of predecoding may include identified instruction types, recognized branches, determined instruction length (in the case of a CISC processor), etc.

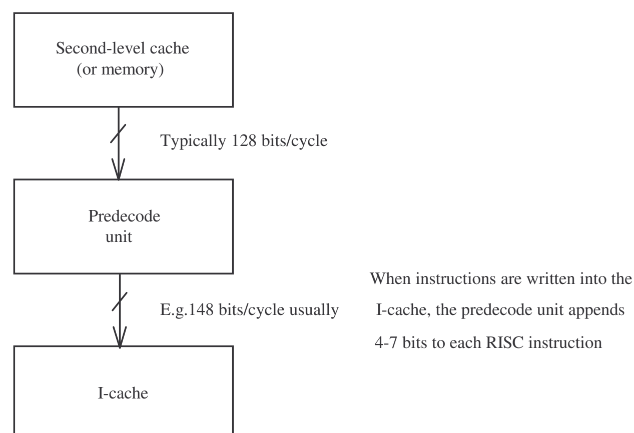



Figure 19: The basic idea behind predecoding

Predecoding appeared with the second wave of superscalars approximately in the mid-1990s, and soon became a standard feature in both RISC and CISC processors. We note that trace processors also predecode instructions to alleviate the complexity of the time critical decode–rename–issue path [10] - [12].

*4) Increasing the throughput of the dispatch subsystem:* In order to increase the throughput of the dispatch subsystem, either the dispatch rate needs to be raised or the instruction window widened.

*(a) Raising the dispatch rate,* i.e. the maximum number of instructions that can be dispatched per cycle, is the "brute force" solution to increase the throughput of the dispatch subsystem. It requires more execution resources, such as EU's, datapaths and more complex logic to select executable instructions from the window. Table 1 indicates the dispatch rates of various superscalar processors.

*(b) Widening the instruction window* is a more subtle approach to raise the throughput of the dispatch subsystem. This approach is motivated by the expectation that more parallel executable instructions per cycle can be found in a wider instruction window than in a smaller one. This is the reason why recent processors typically have wider instruction windows (by providing more issue buffers) than earlier ones, as shown in Table 2. However, a wider window requires deeper and more accurate branch speculation, as we emphasized earlier.

| | Processor | Width of the instr. window |
|---|---|---|
| RISC processor | PowerPC 603 (1993) | 3 |
| | PM1 (Sparc64) (1995) | 36 |
| | PowerPC 604 (1995) | 12 |
| | PA8000(1996) | 56 |
| | PowerPC 620 (1996) | 15 |
| | R10000 (1996) | 48 |
| | Alpha 21264 (1998) | 35 |
| | Power3 (1998) | 20 |
| | R12000 (1998) | 48 |
| | PA8500 (1999) | 56 |
| CISC processor | Nx586 (1994) | 42 |
| | K5 (1995) | 12 |
| | PentiumPro (1995) | 20 |
| | K6 (1996) | 24 |
| | Pentium II (1997) | 20 |
| | K7 (1998) | 54 |
| | M3 (2000) | 56 |

Table 2: Width of the instruction window in superscalar processors that use dynamic instruction issue

Finally, we note that parallel optimizing compilers also contribute to increase the average number of parallel executable instructions available in the window per cycle. As our paper focuses on the microarchitecture itself and does not discuss compiler issues, readers interested in this topic are referred to the literature [61] - [62].

*5) Increasing the throughput of the execution subsystem:* There are three chief ways to increase the throughput of the execution subsystem: (a) increasing the execution rate of the processor by providing more EU's that are able to operate simultaneously; (b) reducing the repetition rates of EU's (i.e. the number of cycles needed until an EU can accept a new instruction for execution); and (c) shortening the execution latency of EU's (i.e. the number of cycles needed until the result of an instruction becomes available to a subsequent instruction). Below we will discuss only the last issue mentioned.

If the processor performs both dynamic instruction scheduling and renaming, then decoded, issued and renamed instructions wait for execution in the issue buffers, i.e. in the instruction window. Clearly, the earlier existing RAW dependencies are resolved in the instruction window, the more instructions will be available for parallel execution on the average per cycle. This calls for *shortening the execution latencies of instructions*. Subsequently, we review techniques used or proposed to achieve this objective either a) for register instructions or b) for load/store instructions.

*a) Shortening the execution latencies of register instructions.* Basically, the following two techniques are used to achieve this goal.

(i) *Result forwarding* provides a bypass from the outputs of EU's to their inputs in order to make the results immediately available for subsequent instructions, as indicated in Figure 20. This way execution latencies can be shortened by the time needed to first write the results into the specified destination register and then to read them from there for a subsequent instruction.

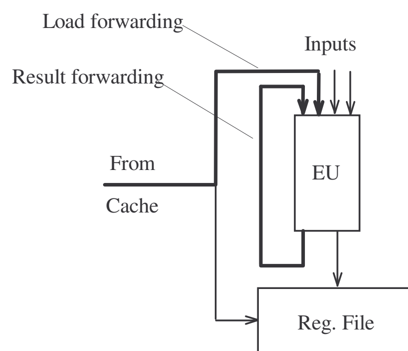



Figure 20: The principle of result and load forwarding

However, implementing result forwarding requires a relatively large number of buses, since a separate bus is needed from the output ports of each EU to the input ports

of all EU's that may need the results. This technique has already been introduced in pipelined processors, such as the i486, and now it is a mature, established technique widely used in superscalars.

*(ii) Exceeding the dataflow limit of execution for multi-cycle register operations*, such as division. This can be achieved by using intricate techniques like *value prediction* [63] - [66] or *value reuse* [67] - [71]. These are current research topics.

b) *Shortening the execution latencies of load/store instructions.* This requirement is a crucial point for increasing the throughput of the microarchitecture for two reasons: first, load/store instructions represent approximately 25–35 % of all instructions [72], and second, the memory subsystem is typically slower than the processor's pipeline. There are three major approaches to address this problem: (i) using load forwarding, (ii) introducing out of order loads, and (iii) exceeding the dataflow limit of execution imposed by load operations.

(i) *Load forwarding* is a technique similar to result forwarding described above. It cuts load latencies (i.e. the time needed until the result of a load operation becomes available to a subsequent instruction) by immediately forwarding fetched data to the input ports of the EU's, as indicated in Figure 20. This technique is also widely used in current superscalars.

(ii) *Out of order execution of loads* is a technique to bypass younger, already executable loads over older loads and stores not yet ready for execution. This technique effectively contributes to reducing delays caused by load misses. Out of order execution of loads can be implemented in a number of ways. *Speculative loads* (PowerPC 620, R10000, Sparc64, Nx586) and *store forwarding* (Nx586, Cyrix's 686 MX, M3, K-3, UltraSparc3) are implementation alternatives already employed in current processors, whereas *dynamically speculated loads* [73] - [75] and *speculative store forwarding* [50] are new alternatives that have been proposed.

(iii) It is also possible *to exceed the dataflow limit caused by load operations*, either by *load value prediction* [50], [75] or by *load value reuse* [85], [69], [75]. These issues are recent research topics.

Finally, we emphasize that the overall design of a microarchitecture calls for discovering and removing possible bottlenecks in individual subsystems. This task usually requires a tedious, iterative cycle-by-cycle simulation on a number of benchmark applications.

*6) Limits of utilizing issue parallelism:* Obviously, it is rather impractical to widen the microarchitecture beyond the extent of available instruction level parallelism. As general-purpose programs have on average no more than about 4–8 parallel executable instructions per cycle [77] and recent microarchitectures are already at least four-wide designs, not much room seems to remain for performance increase through widening the microarchitecture even further, at least for general purpose applications.

## *VI. INTRODUCTION OF INTRA-INSTRUCTION PARALLELISM*

### *A. Key approaches to introduce intra-instruction parallelism*

The last major possibility to increase processor performance at the instruction level is to introduce *multiple data operations* within instructions. This type of parallelism is called *intra-instruction* parallelism. Three different approaches exist for the implementation of multiple-data-operation instructions: (a) dual-operation instructions, (b) SIMD instructions and (c) VLIW instructions, as indicated in Figure 21. Its introduction requires, however, either an *extension of the ISA* by adding instructions that perform multiple data operations and an appropriate enhancement of the microarchitecture to enable their execution (for the approaches (a) and (b)) or a *completely new ISA* (for (c)).

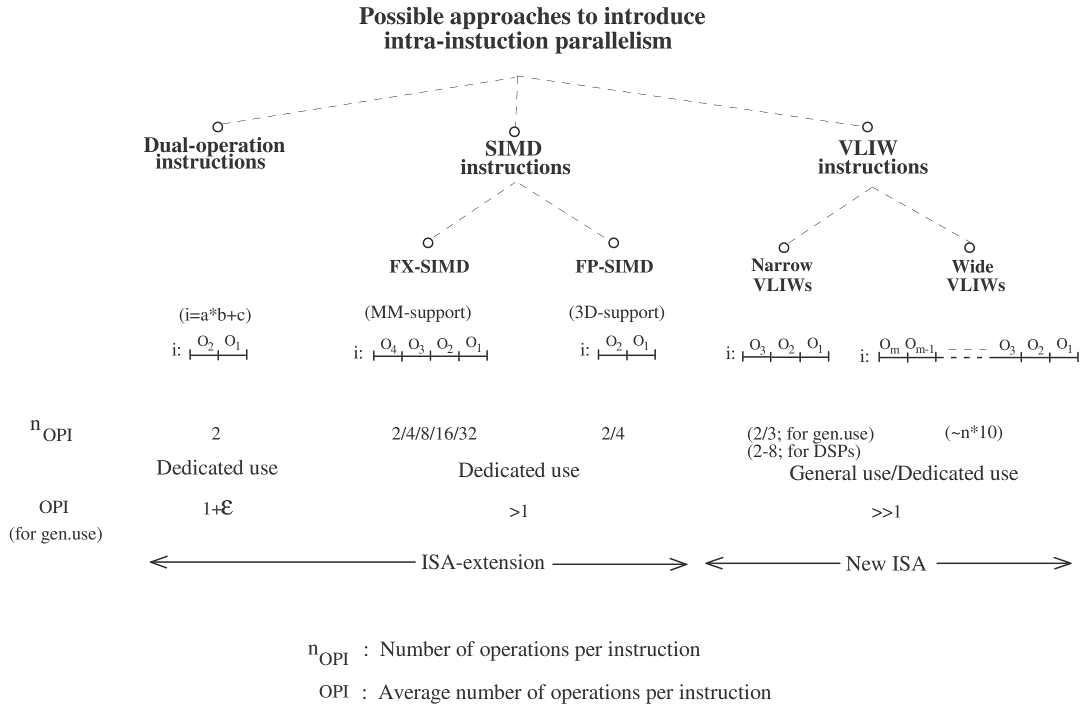



Figure 21: Possibilities to introduce intra-instruction parallelism

*(a) Dual-operation instructions* comprise, as their name suggests, two different data operations within the same instruction. The most widely used one is the *multiply-add instruction* (“multiply-and-accumulate” or “fused multiply-add” instruction) that calculates the dot product (x = a * b + c) for floating-point data.

Multiply-add instructions were introduced in the early 1990’s in the POWER [78], PowerPC [79], PA-RISC [80] and MIPS-IV [81] ISA’s and in the respective microprocessor models. However, this instruction is only useful for numeric computations, and thus it only marginally increases the average number of operations executed per instruction (OPI) in general purpose applications. Other examples of dual-operation instructions include fused load/op, shift & add, etc. instructions.

*(b) SIMD instructions* allow the same operation to be performed on more than one set of operands. E.g. in Intel's MMX multimedia extension [82], the

PADDW   MM1, MM2

SIMD instruction performs four fixed point additions on the four 16-bit operand pairs held in the 64-bit registers MM1 and MM2.

As Figure 21 indicates, SIMD instructions may refer either to fixed point or to floating point data. *Fixed point SIMD instructions* enhance multimedia applications, i.e. multiple (2/4/8/16/32) operations on display pixels, whereas *floating point SIMD instructions* accelerate 3D graphics by executing (usually) two floating point operations simultaneously.

Fixed point SIMD instructions were pioneered in 1993–1994 in the MC88110 and PA-7100LC processors, as shown in Figure 22. Driven by the proliferation of multimedia applications, SIMD extensions (such as AltiVec from Motorola [83], MVI from Compaq [84], MDMX from MIPS [85], MAX-2 from Hewlett-Packard [86], VIS from Sun [87] and MMX from Intel [82]) soon became a standard feature of most established processor families. Floating point SIMD extensions, such as 3DNow! from AMD, CYRIX and IDT [88] and SSE from Intel [89] emerged in 1998 in order to support 3D applications. They were implemented in the K6-2, K6-3 and Pentium III processors, followed later by the G4 and K7, as indicated in Figure 22.

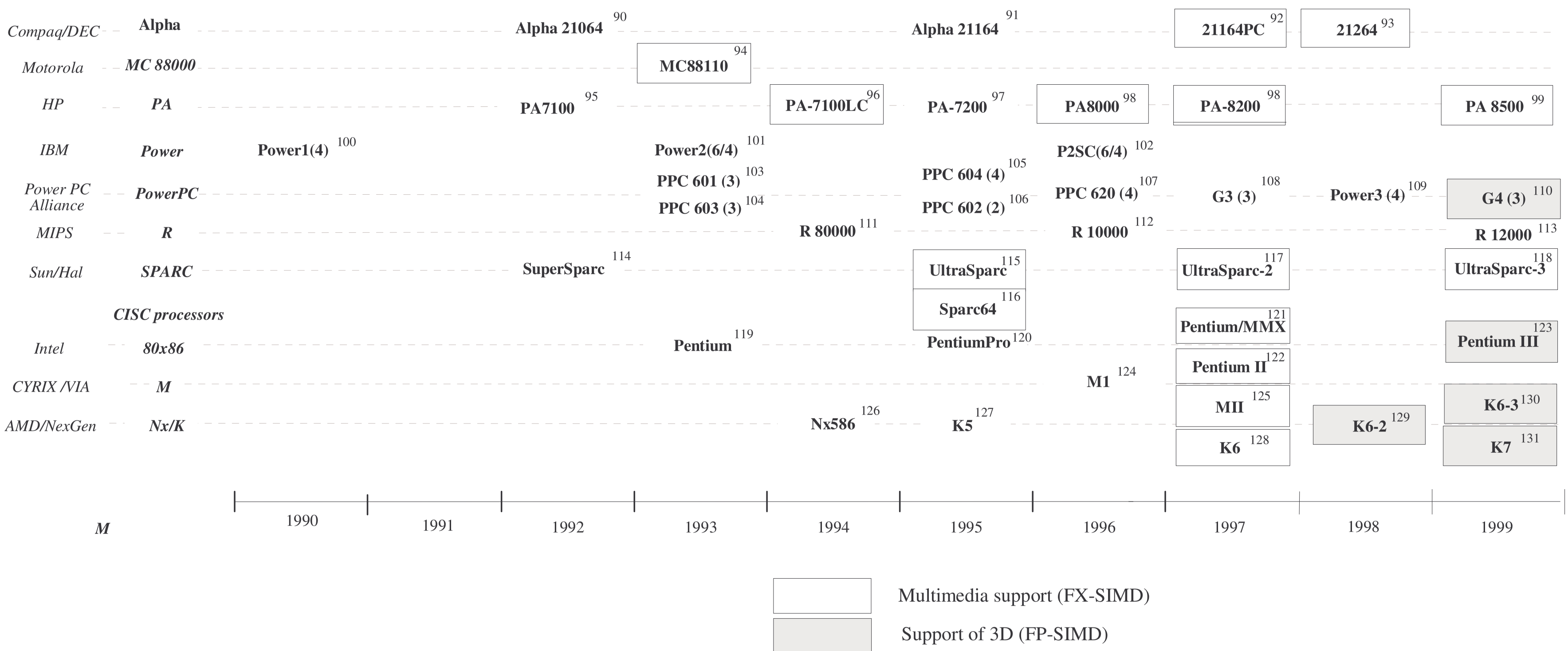



Figure 22: The emergence of FX-SIMD and FP-SIMD instructions in microprocessors

(The references to superscalar processors are given as superscripts behind the processor designations)

Clearly, multimedia and 3D support will boost processor performance mostly in dedicated applications. For instance, based on Media Benchmark ratings Intel stated a per cycle performance gain of about 37 % from multimedia support in its Pentium II over Pentium Pro [132]. Intel has also published figures demonstrating that its 3D-enabled Pentium III has a cycle by cycle performance gain of approx. 61% over Pentium II running the 3D Lighting and Transformation Test of the 3D WinBench99 benchmark suite [133]. On the other hand, multimedia and 3D support results in only a rather modest performance gain for general purpose applications measured in terms of SPECint92 benchmark ratings normed to the same clock frequency. For instance, the Pentium II offers only a 3–5 % performance increase over the Pentium Pro at the same clock frequency, whereas Pentium III shows a similarly slight benefit over Pentium II at the same clock frequency [1].

(c) The third major possibility to introduce intra-instruction parallelism is the *VLIW* (*Very Long Instruction Word*) *approach*. In VLIW's, different fields of the same instruction word control simultaneously operating EU's of the microarchitecture. As a consequence, VLIW processors with a large number of EU's need very long instruction words, hence the name. For instance, Multiflow's TRACE VLIW machine used 256-bit to 1024-bit long instruction words to specify 7 to 28 simultaneous operations within the same instruction word [134].

Unlike superscalars, VLIW's are scheduled statically. This means that the compiler takes all responsibilities for resolving all types of dependencies. To be able to do so, the compiler needs intimate knowledge of the microarchitecture concerning the number, types, repetition rates and latencies of the EU's, load-use latencies of the caches etc. On the one hand, this results in a complex and technology-dependent compiler, while on the other hand it leads to reduced hardware complexity as opposed to comparable superscalar

designs. In addition, the compiler is expected to perform aggressive parallel optimization in order to find enough executable operations for high throughput.

VLIW proposals emerged as paper designs in the first half of the 1980's (Polycyclic architecture [135], ELI-512 [136]), followed by two commercial machines in the second half of the 1980s (Multiflow's TRACE [134] and Cydrome's Cydra-5 [137]). We will term these traditional designs as *wide VLIW's,* since they incorporate a large number of EU's, typically in the range of 10 or more.

Wide VLIW's disappeared from the market fairly quickly, which was partly due to their deficiencies—technological sensitivity of compilers, wasted memory fetch bandwidth owing to sparsely populated instruction words, etc. [4]—as well as to the onus of their manufacturers being start-up companies.

The reduced hardware complexity of VLIW designs versus superscalar designs and the progress achieved in compiler technology have led to a revival of VLIW's in the late 1990's, both for DSP and general purpose applications. *VLIW-based DSP's*, such as Philips' TM1000 TriMedia processors [138], TI's TMS320C6000 cores [139], the SC140 core from Motorola and Lucent [140] and ADI's TigerSharc [141] are intended for multimedia applications. We have good reason to term these designs as *narrow VLIW's* in contrast to the earlier VLIW designs mentioned above.

*General purpose narrow VLIW's* with 3–4 operations per instruction have recently emerged on the horizon, including Intel's Itanium (a. k. a. Merced) [142] that implements the EPIC (Explicitly Parallel Instruction Computing) VLIW philosophy, Sun's MAJC processor units used in their MCP chips [143] and Transmeta's Crusoe processors [144], which have become rivals of superscalars.

In summary, out of the above approaches designed to introduce intra-instruction parallelism only traditional wide VLIW's and general purpose narrow VLIW's are able to

perform considerably more than one operation per instruction (OPI>>1) on average for general purpose applications. On the other hand, dual-operation and SIMD instructions as well as DSP-oriented VLIW's are intended for dedicated applications.

## *VII. THE MAIN ROAD OF THE MICROARCHITECTURE EVOLUTION*

As pointed out before, the main road of the microarchitecture evolution is marked by an increasing utilization of available instruction level parallelism. This took place while designers introduced one after another temporal, issue and intra-instruction parallelism in new microarchitectures (see Figure 23). This sequence has been determined basically by the objective to boost performance while *maintaining upward compatibility* with preceding models. Nevertheless, the price to be paid for increased performance is *decreasing efficiency of hardware utilization.*

In this respect we point out that scalar pipelined processors that only make use of temporal parallelism exhibit the best hardware utilization, since in essence all stages of their pipelines are always used to process instructions. Superscalar processors that also utilize issue parallelism make less efficient use of their hardware resources due to the availability of multiple (parallel) execution paths. SIMD hardware extensions—which also enable architectures to exploit intra-instruction parallelism—are the least utilized, as they are used only for MM and 3D applications. In summary, higher per cycle throughput necessarily leads to higher hardware redundancy, as indicated in Figure 23.

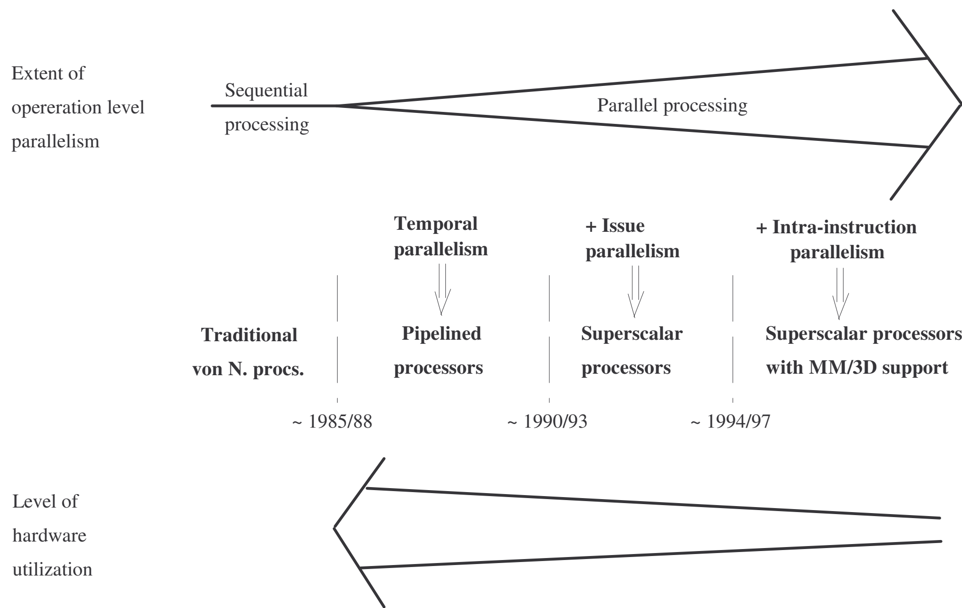



Figure 23: Main road of the evolution of microarchitectures

We note that beyond the above discussed evolutionary scenario, a second scenario was also open for the development of microarchitectures.

**a. Evolutionary scenario (Superscalar approach)**

**b. Radical scenario (VLIW approach)**

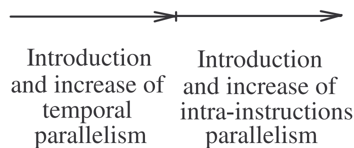



Figure 24: Possible scenarios for the development of processors

In this second scenario, the introduction of temporal parallelism is followed immediately by the debut of intra-instruction parallelism in the form of VLIW instructions, as indicated in Figure 24. Clearly, introducing multiple data operations per instruction

instead of issuing and executing multiple instructions per clock cycle is a competitive alternative of boosting throughput. However, in contrast to the evolutionary scenario that preserves upward compability, this scenario represents in a sense a quite *"radical" path*, since the introduction of multi-operation VLIW instructions demands a completely new ISA. This is the key reason why this alternative, pioneered by wide VLIW's at the end of the 1980's, turned out to be a dead end.

## *VIII. CONCLUSIONS*

As we pointed out in our paper, microarchitectures evolved at the instruction level basically in three consecutive cycles, following a twisting upward curve not unlike a spiral with three windings. Each cycle added a new dimension of parallelism to the microarchitecture by means of a new basic technique. However, once a basic technique enhances a particular subsystem of the microarchitecture, other subsystems become the bottleneck of processor performance. Consequently, the introduction of a new basic technique eventually calls for several additional techniques to resolve these new bottlenecks. Each cycle ends up when the additional techniques, introduced to augment the basic technique, enable it to achieve its full potential. This, however, gives rise to a new cycle in which a new dimension of parallel operation must be introduced to the microarchitecture in order to increase processor performance even further.

In particular, temporal parallelism was the first to make its debut with pipelined processors, as Figure 25 shows. The emergence of pipelined instruction processing stimulated the introduction of caches and of speculative branch processing. Thus the entire potential of temporal parallelism could be exhausted. For further performance increase, issue parallelism became utilized next via the introduction of superscalar processors. Superscalars evolved in two waves, differing basically in the effective width of the

microarchitecture. First wave superscalars made use of direct (unbuffered) instruction issue, accompanied by advanced branch processing and a more powerful memory subsystem. The issue bottleneck inherent to the direct issue scheme basically limited the microarchitecture to a two-wide design. However, the demand for still higher throughput called for widening the microarchitecture. This gave rise to a second wave of superscalars featuring dynamic instruction scheduling (buffered instruction issue), register renaming and several additional techniques to widen particular subsystems, as outlined in the paper. Finally, having exhausted the extent of instruction level parallelism available in general purpose programs, intra-instruction parallelism has been introduced with SIMD instructions. However, this enhancement, effective primarily in emerging multimedia and 3D applications, required a considerable extension of the ISA.

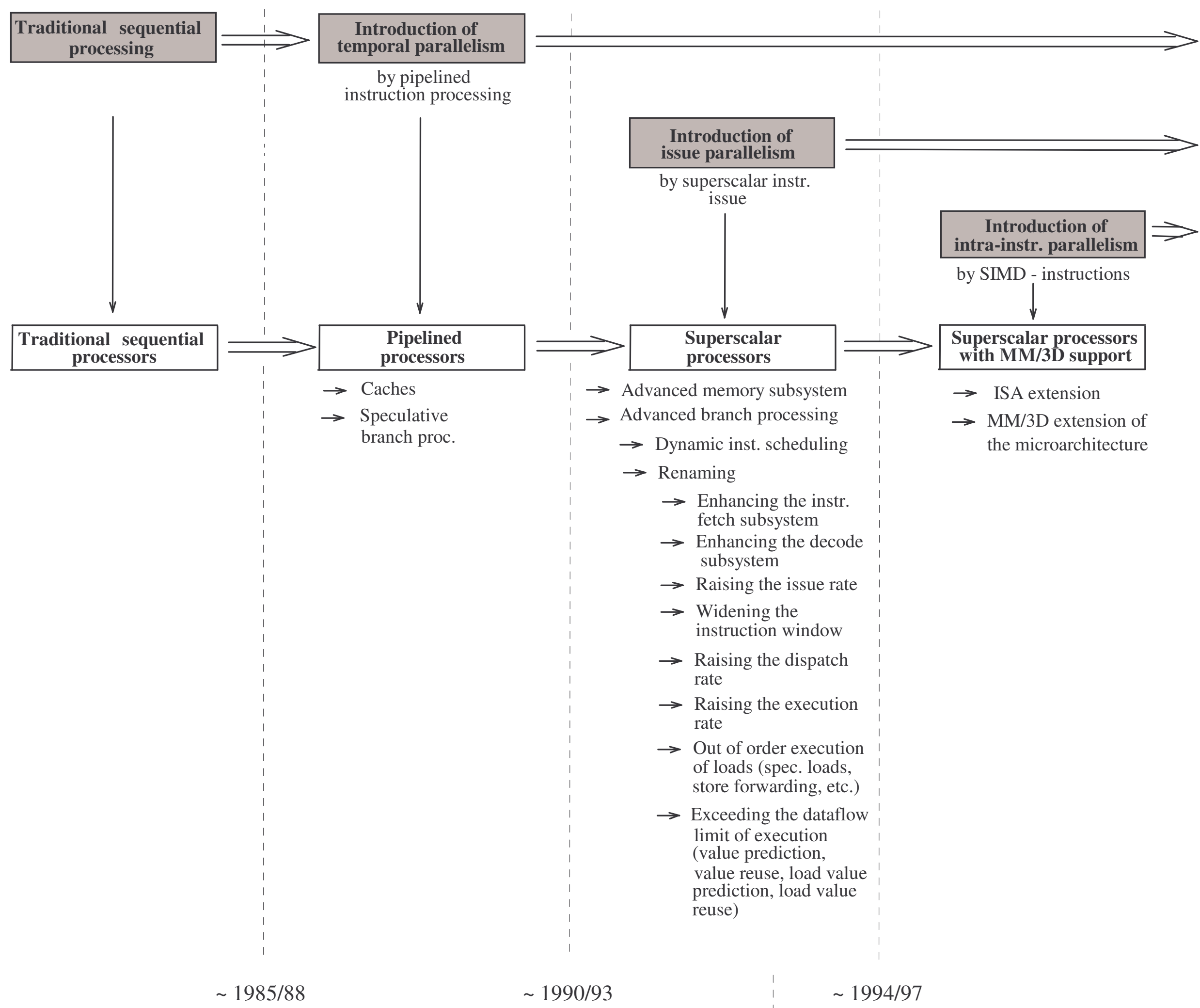



Figure 25: Major steps in the evolution of microprocessors

All the decisive aspects mentioned above constitute a framework that explains the main road of the microarchitecture evolution, including the sequence of major innovations encountered.

*ANNEX*

*The throughput of the processor* ($T_{OPC}$). To express the throughput of the processor ($T_{OPC}$) with the operational parameters of the microarchitecture, we assume the following *model of processor operation* (see Figure 26).

In the figure, the arrows indicate decoded instructions issued for processing.

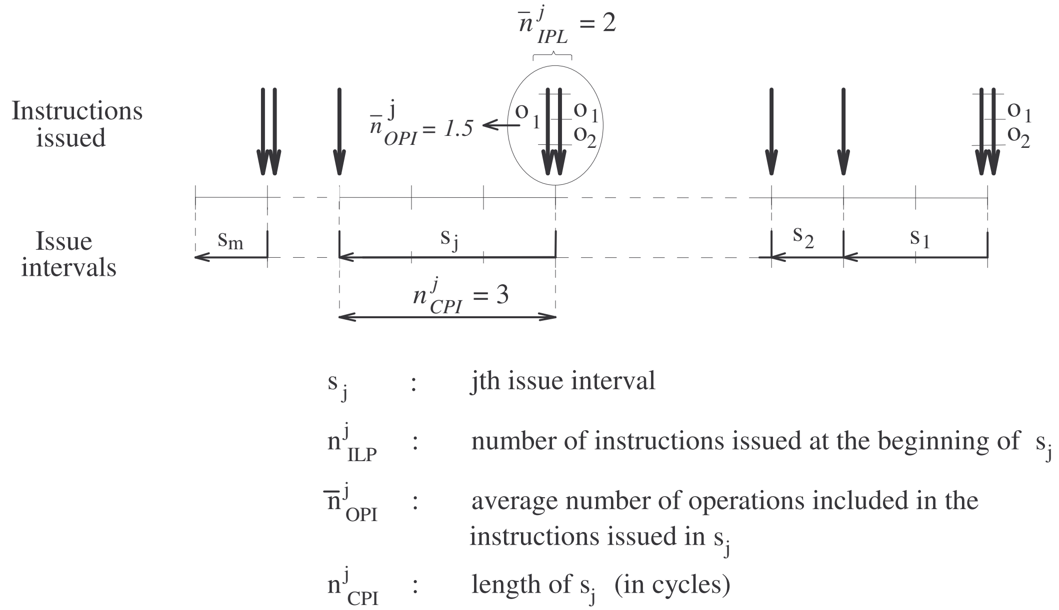



Figure 26: Assumed model of processor operation

(a) We assume that the processor operates in cycles, issuing in each cycle 0, 1...$n_i$ instructions, where $n_i$ is the issue rate of the processor.

(b) We allow instructions to include more than one operation.

(c) Out of the cycles needed to execute a given program, we focus on those in which the processor issues at least one instruction. We call these cycles *issue cycles,* and denote them by $c_j$, j = 1...m. The issue cycles $c_j$ subdivide the execution time of the program into *issue intervals* $s_j$, j = 1...m, such that each issue interval begins with an issue cycle and lasts until

the next issue cycle begins. $s_1$ is the first issue interval, whereas $s_m$ is the last one belonging to the given program.

(d) We describe the operation of the processor using a set of three parameters which are given for each of the issue intervals $s_{j.}$, j = 1...m. The set of the parameters chosen is as follows (see Figure 26):

$n^j_{IPL}$ = the number of instructions issued at the beginning of the issue interval $s_j$, j = 1...m,

$\bar{n}_{OPI}$ = the average number of operations included in the instructions, which are issued in the issue interval $s_j$, j = 1...m,

$n^j_{CPI}$ = the length of the issue interval $s_j$ in cycles, j = 1...m. Here $n^m_{CPI}$ is the length of the last issue interval, which is interpreted as the number of cycles to be elapsed until the processor is ready to issue instructions again.

Then, in issue interval $s_j$ the processor issues $n^j_{OPC}$ operations per cycle, where:

$$n^j_{OPC} = \frac{n^j_{ILP} * \bar{n}^j_{OPI}}{n^j_{CPI}} \qquad (6)$$

Now let us consider $n^j_{OPC}$ as a stochastic variable, which is derived from the stochastic variables $n^j_{ILP}$, $\bar{n}^j_{OPI}$ and $n^j_{CPI}$, as indicated in (6). Assuming that the stochastic variables involved are independent, the throughput of the processor ($T_{OPC}$), which is the average value of $n_{OPC}$ ($\bar{n}_{OPC}$), can be calculated from the averages of the three stochastic variables included, as indicated below:

$$T_{OPC} = \bar{n}_{OPC} = 1/\bar{n}_{CPI} * \bar{n}_{ILP} * \bar{n}_{OPI} \quad (7)$$

Temporal parallelism (↓ $1/\bar{n}_{CPI}$), Issue parallelism (↓ $\bar{n}_{ILP}$), Intra-instruction parallelism (↓ $\bar{n}_{OPI}$)

Finally, we rename the terms introduced above for better readability as follows:

$$\bar{n}_{CPI} = CPI \quad (8)$$

$$\bar{n}_{ILP} = ILP \quad (9)$$

$$\bar{n}_{OPI} = OPI \quad (10)$$

$$T_{OPC} = OPC \quad (11)$$

Thus we obtain for the average number of operations processed per cycle (OPC):

$$OPC = 1/CPI * ILP * OPI \quad (12)$$

As according to expression (3) OPC = IPC * OPI, it follows from (3) and (12) that

$$IPC = 1/CPI * ILP \quad (13)$$

## *ACKNOWLEDGMENTS*

The author would like to thank the anonymous reviewers for their valuable comments and suggestions on earlier drafts of this paper.

REFERENCES

[1] ___,“Intel Microprocessor Quick Reference Guide,” http://developer.intel.com/pressroom/kits/processors/quickref.html.

[2] L. Gwennap, “Processor performance climbs steadily”, *Microprocessor Report*, vol. 9, no. 1, pp. 17-23, 1995.

[3] J. L. Hennessy, “VLSI processor architecture,” *IEEE Transactions on Computers*, vol. C-33, no. 12, pp. 1221-1246, Dec. 1984.

[4] B. R. Rau and J. A. Fisher, “Instruction level parallel processing: history, overview and perspective,” *The Journal of Supercomputing*, vol. 7, no. 1, pp. 9-50, 1993.

[5] J. E. Smith and G. S. Sohi, “The microarchitecture of superscalar processors,” *Proc. IEEE*, vol. 83, no. 12, pp. 1609-1624, Dec. 1995.

[6] A. Yu, “The Future of microprocessors”, *IEEE Micro*, vol. 16, no. 6, pp. 46-53, Dec. 1996.

[7] K. Diefendorff, “PC processor microarchitecture, a concise review of the techniques used in modern PC processors,” *Microprocessor Report*, vol. 13, no. 9, pp. 16-22, 1999.

[8] M. Franklin, “The Multiscalar Architecture,” Ph.D. thesis, TR 1196, Comp. Science Dept., Univ. of Wisconsin-Madison, 1993.

[9] G. S. Sohi, S. E. Breach, and T. N. Vijaykumar, “Multiscalar Processors,” in *Proc. 22th ISCA*, 1995, pp. 415-425.

[10] E. Rothenberg, Q. Jacobson, Y. Sazeides and J. Smith, “Trace Processors,” in *Proc. Micro 30*, 1997, pp. 138-148.

[11] J. E. Smith and S. Vajapeyam, ” Trace processors: Moving to fourth generation microarchitectures,” *IEEE Computer*, vol. 30, no. 9, pp. 68-74, Sept. 1997

[12] Y. N. Patt, S. J. Patel, M. Evers, D. H. Friendly, and J. Stark, “One billion transistors, one uniprocessor, one chip,” *IEEE Computer*, vol. 30, no. 9, pp. 51-57, Sept. 1997.

[13] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: Maximizing on-chip parallelism," in *Proc. 22th ISCA*, 1995, pp. 392-403.

[14] S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, R. L. Stamm, and D. M. Tullsen, "Simultaneous multithreading: A platform for next generation processors," *IEEE Micro*, vol. 17, no. 5, pp. 12-19, Sept./Oct. 1997.

[15] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang "The case for a single chip multiprocessor," in *Proc. ASPLOS VII*, 1996, pp. 2-11.

[16] L. Hammond, B. A. Nayfeh and K. Olukotun, "A single-chip multiprocessor," *IEEE Computer,* vol. 30, no. 9, pp. 79-85, Sept. 1997.

[17] ___, "SPEC Benchmark Suite, Release 1.0," SPEC, Santa Clara, CA, Oct. 1989.

[18] ___, "SPEC CPU92 Benchmarks," http://www.specbench.org/osg/cpu92/

[19] ___, "SPEC CPU95 Benchmarks," http://www.specbench.org/osg/cpu95

[20] [11] ___, "Winstone 99," http://www1.zdnet.com/zdbob/winstone/winstone.html.

[21] ___, "WinBench 99," http://www.zdnet.com/zdbop/winbench/winbench.html

[22] ___, "SYSmark Bench Suite," http://www.babco.com/

[23] H. J. Curnow and B. A. Wichmann, "A synthetic benchmark," *The Computer J.*, vol. 19, no. 1, pp. 43-49, Jan. 1976.

[24] R. P. Weicker, "Drystone: A synthetic systems programming benchmark," *Comm. ACM*, vol. 27, no. 10, pp. 1013-1030, Oct. 1984.

[25] D. Anderson and T. Shanley, *ISA System Architecture*, 3rd ed. Reading, MA: Addison-Wesley Developers Press, 1995.

[26] D. Anderson and T. Shanley, *EISA System Architecture*, 2nd ed. Reading,

MA: Addison-Wesley Developers Press, 1995.

[27] D. Anderson and T. Shanley, *PCI System Architecture*, 4th ed. Reading, MA: Addison-Wesley Developers Press, 1999.

[28] ___, "PCI-X Addendum Released for Member Review," http://www.pcisig.com/

[29] V. Cuppu, B. Jacob, B. Davis, and T. Mudge, "A performance comparison of contemporary DRAM architectures," in *Proc. 26th ISCA*, 1999, pp. 222 – 233.

[30] G. S. Sohi and M. Franklin, "High bandwith data memory systems for superscalar processors," in *Proc. ASPLOS IV*, 1991, pp. 53-62.

[31] T. Juan, J. J. Navarro, and O. Teman, "Data caches for superscalar processors," in *Proc. ICS'97*, 1997, pp. 60–67.

[32] D. Burger, J. R. Goodman, and A. Kägi, "Memory bandwidth limitations of future microprocessors," in *Proc. ISCA*, 1996, pp. 78-89.

[33] W. C. Hsu and J. E. Smith, "A performance study of instruction cache prefetching methods", *IEEE Trans. Computers*, vol. 47, no. 5, pp. 497-508, May 1998.

[34] E. Bloch, "The engineering design of the STRETCH computer", in *Proc. East. Joint Comp. Conf.,* New York: Spartan Books, 1959, pp. 48-58.

[35] R. M. Tomasulo, "An efficient algorithm for exploiting multiple arithmetic units," *IBM J. Res. and Dev.* vol. 11, no.1, pp. 25-33, Jan. 1967.

[36] R. W. Hockney and C. R. Jesshope, *Parallel Computers*. Bristol: Adam Hilger, 1981.

[37] T. Kilburn, D. B. G. Edwards, M. J. Lanigan, and F. H. Sumner, "One-level storage system," *IRE Trans. EC-11*, vol. 2, pp. 223-235, Apr. 1962.

[38] D. W. Anderson, F. J. Sparacio and F. M. Tomasulo, "The IBM System/360

Model 91: Machine philosophy and instruction-handling," *IBM Journal*, vol. 11, no 1, pp. 8-24, Jan. 1967.

[39] ___, „80286 High performance microprocessor with memory management and protection," Microprocessors, vol. 1. Mt. Prospect, IL: Intel, pp. 3. 60-3. 115, 1991.

[40] T. L. Johnson, "A comparison of M68000 family processors," *BYTE*, vol. 11, no. 9, pp. 205-218, Sept. 1986.

[41] G. Kane and J. Heinrich, *MIPS RISC Architecture*. Englewood Cliffs, NJ: Prentice Hall, 1992.

[42] ___, „80386 DX High performance 32-bit CHMOS microprocessor with memory management and protection," Microprocessors, vol. 1. Mt. Prospect, IL: Intel, pp. 5. 287-5. 424, 1991.

[43] ___, "The 68030 microprocessor: a window on 1988 computing," *Computer Design*, vol. 27, no. 1, pp. 20-23, Jan. 1988.

[44] F. Faggin, M. Shima, M. E. Hoff, Jr., H. Feeney, and S. Mazor, "The MCS-4: An LSI Micro Computer System," in *Proc. IEEE Region 6 Conf.*, 1972, pp. 8-11.

[45] S. P. Morse, B. W. Ravenel, S. Mazor, and W. B. Pohlman, "Intel microprocessors: 8008 to 8086," Intel Corp. 1978, in D. P. Siewiorek, C. G. Bell and A. Newell, *Computer Structures: Principles and Examples*. McGraw-Hill Book Comp., New York: 1982.

[46] C. J. Conti, D. H. Gibson, and S. H. Pitkowsky, "Structural aspects of the System/360 Model85, Part 1: General Organization," *IBM Syst. J.*, vol. 7, no. 1, pp. 2-14, Jan. 1968.

[47] J. E. Smith, "A study of branch prediction strategies," in *Proc. 8$^{th}$ ISCA*, May 1981, pp. 135-148.

[48] K. F. Lee and A. J. Smith, "Branch prediction strategies and branch target buffer design," *Computer,* vol. 17, no. 1, pp. 6-22, Jan. 1984.

[49] D. Sima, T. Fountain, and P. Kacsuk, *Advanced Computer Architectures.* Harlow: Addison-Wesley, 1997.

[50] M. H. Lipasti and J. P. Shen, "Superspeculative microarchitecture for beyond AD 2000," *IEEE Computer*, vol. 30, no. 9, pp. 59-66, Sept. 1997.

[51] Y. Patt, W.-M. Hwu, and M. Shebanow, “HPS, A new microarchitecture: Rationale an Introduction,” in *Proc. MICRO28*, Asilomar, CA, Dec. 1985, pp. 103-108.

[52] D. Sima, “Superscalar instruction issue,” *IEEE Micro*, vol. 17, no. 5, pp. 28-39, Sept./Oct. 1997.

[53] T.-Y. Yeh and Y. N. Patt, “Alternative implementations of two-level adaptive branch prediction,” in *Proc. 19th ISCA*, 1992, pp. 124-134.

[54] S. McFarling, “Combining Branch Predictors”, TR TN-36, WRL, June 1993.

[55] S. Duta and M. Franklin, “Control flow prediction schemes for wide-issue superscalar processors,” *IEEE Trans. Parallel and Distributed Systems*, vol. 10, no. 4, pp. 346-359, April 1999.

[56] N. P. Jouppi and D. W. Wall, “Available instruction-level parallelism for superscalar and superpipelined machines,” in *Proc. ASPLOS-III*, 1989, pp. 272-282.

[57] M. S. Lam and R. P. Wilson, “Limits of control flow on parallelism,” in *Proc. 19th ISCA*, 1992, pp. 46-57.

[58] D. Sima, “The design space of shelving,” *J. Systems Architecture*, vol. 45, no. 11, pp. 863-885, 1999.

[59] L. Gwennap, „Nx686 Goes Toe-to-Toe with Pentium Pro,” *Microprocessor Reports*, vol. 9, no. 14, pp. 1, 6-10, Oct. 1998.

[60] R. Yung, „Evaulation of a Commercial Microprocessor,” Ph. D. dissertation, University of California, Berkeley, June 1998.

[61] S. V. Adve, “Changing interaction of compiler and architecture,” *IEEE Computer*, vol. 30, no. 12, pp. 51-58, Dec. 1997.

[62] J. Shipnes and M. Phillips, „A Modular approach to Motorola PowerPC Compilers,” *Comm. ACM*, vol. 37, no. 6, pp. 56-63, June 1994.

[63] C. Fu, M. D. Jennings, S. Y. Larin, and T. M. Conte, “Value speculation scheduling for high performance processors,” in *Proc. ASPLOS-VIII,* 1998, pp. 262-271.

[64] M. H. Lipasti and J. P. Shen, "Exceeding the dataflow limit via value prediction," in *Proc. MICRO29*, 1996, pp. 226-237.

[65] Y. Sazeides and J. E. Smith, "The predictability of data values," in *Proc. MICRO30*, 1997, pp. 248-258.

[66] B. Calder, P. Feller, and A. Eustace, "Value profiling," in *Proc. MICRO30*, 1997, pp. 259-269.

[67] D. Michie, "Memo functions and machine learning," *Nature*, no 218, pp. 19-22, 1968.

[68] S. Richardson, "Exploiting trivial and redundant computation," in *Proc. 11th Symp. Computer Arithmetic*, 1993, pp. 220-227.

[69] A. Sodani and G.S. Sohi, "Dynamic instruction reuse," in *Proc. 24th ISCA*, 1997, pp. 194-205.

[70] A. Sodani and G.S. Sohi, "An empirical analysis of instruction repetition," in *Proc. ASPLOS VIII*, 1998, pp. 35-45.

[71] D. Citron, D. Feitelson, and L. Rudolph, "Accelerating multi-media processing by implementing memoing in multiplication and division," in *Proc. ASPLOS VIII*, 1998, pp. 252-261.

[72] M. Butler, T.-Y. Yeh, Y. Patt, M. Alsup, H. Scales, and M. Shebnow, “Single instruction stream parallelism is greater than two, “ in *Proc. 18th ISCA*, 1991, pp. 276-286.

[73] A. Moshovos et al., "Dynamic speculation and synchronization of data dependencies," in *Proc. 24th ISCA*, 1997, pp. 181-193.

[74] G. Z. Chrysos and J. S. Emer, "Memory dependence prediction using store sets," in *Proc. 25th ISCA*, 1998, pp. 142-153.

[75] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen, "Value locality and load value prediction," in *Proc. ASPLOS VII*, 1996, pp. 138-147.

[76] M. Franklin and G. S. Sohi, "ARB: a hardware mechanism for dynamic reordering of memory references," *IEEE Trans. Computers*, vol. 45, no. 5, pp. 552-571, May 1996.

[77] D. W. Wall, "Limits of instruction level parallelism,"in *Proc. ASPLOS IV*, 1991, pp. 176-188.

[78] R. R. Oehler and M. W. Blasgen, "IBM RISC System/6000: Architecture and performance," *IEEE Micro*, vol. 11, no. 3, pp. 14-17, 56-62, May/June 1991.

[79] K. Diefendorff and E. Shilha, "The PowerPC user instruction set architecture," *IEEE Micro,* vol. 14, no. 5, pp. 30-41, Sept./Oct. 1994.

[80] D. Hunt, "Advanced performance features of the 64-bit PA-8000," in *Proc. COMPCON*, 1995, pp. 123-128.

[81] ___, "MIPS IV Instruction Set Architecture," White Paper, MIPS Technologies Inc., Mountain View, CA, 1994.

[82] A. Peleg and U. Weiser, "MMX technology extension to the Intel architecture," *IEEE Micro*, vol. 16, no. 4, pp. 42-50, July/Aug. 1996.

[83] S. Fuller, "Motorola's AltiVec technology," White Paper, Austin Tx: Motorola Inc., 1998.

[84] ___, "Advanced Technology for Visual Computing: Alpha Architecture with MVI," White Paper, http://www.digital.com/semiconductor/mvi-backgrounder.htm

[85] D. Sweetman, *See MIPS Run.* San Francisco, CA: Morgan Kaufmann, 1999.

[86] R. B. Lee, “Subword parallelism with MAX-2,” *IEEE Micro*, vol. 16, no. 4, pp. 51-59, July/Aug. 1996.

[87] L. Kohn, G. Maturana, M. Tremblay, A. Prabhu, and G. Zyner, “The Visual Instruction Set (VIS) in UltraSPARC”, in *Proc. COMPCON*, 1995, pp. 462-469.

[88] S. Oberman, G. Favor, and F. Weber, “AMD 3DNow! technology: architecture and implementations,” *IEEE Micro*, vol. 19, no. 2, pp. 37-48, March/Apr. 1999.

[89] ___, Intel Architecture Software Developers Manual, http://developer.intel.com/design/PentiumIII/manuals/

[90] ___, “DECchip 21064 and DECchip 21064A Alpha AXP Microprocessors Hardware Reference Manual,” Maynard, MA: DEC, 1994.

[91] ___, “Alpha 21164 Microprocessor Hardware Reference Manual,” Maynard, MA: DEC, 1994.

[92] ___, “Microprocessor Hardware Reference Manual,” Sept. 1997

[93] D. Leibholz and R. Razdan, "The Alpha 21264: a 500 MIPS out-of-order execution microprocessor," in *Proc. COMPCON*, 1997, pp. 28-36.

[94] K. Diefendorff and M. Allen, “Organization of the Motorola 88110 superscalar RISC microprocessor,” *IEEE Micro*, vol. 12, no. 2, pp. 40-62, March/Apr. 1992.

[95] T. Asprey, G. S. Averill, E. Delano, B. Weiner, and J. Yetter, “ Performance features of the PA7100 microprocessor, *IEEE Micro*, vol. 13, no. 3, pp. 22-35, May/June 1993.

[96] R. L. Lee, “Accelerating multimedia with enhanced microprocessors,” *IEEE Micro*, vol. 15, no. 2, pp. 22-32, March/Apr. 1995.

[97] G. Kurpanek, K. Chan, J. Zheng, E. CeLano, and W. Bryg, "PA-7200: A PA-RISC processor with integrated high performance MP bus interface," in *Proc. COMPCON,* 1994, pp. 375-82.

[98] A. P. Scott et. al., "Four-Way Superscalar PA-RISC Processors," *Hewlett-Packard Journal*, pp. 1-9, Aug. 1997.

[99] G. Lesartre and D. Hunt, "PA-8500: The Continuing Evolution of the PA-8000 Family," PA-8500 Document, Hewlett-Packard Company, pp. 1-11, 1998.

[100] G. F. Grohoski, "Machine organization of the IBM RISC System/6000 processor," *IBM J. Research and Development*, vol. 34, no. 1, pp. 37-58, Jan. 1990.

[101] S. White and J. Reysa, "PowerPC and POWER2: Technical Aspects of the New IBM RISC System/6000," Austin, *TX:,* IBM Corp. 1994.

[102] L. Gwennap, "IBM crams Power2 onto single chip," *Microprocessor Report*, vol. 10, no. 11, pp. 14-16, 1996.

[103] M. Becker, "The PowerPC 601 microprocessor," *IEEE Micro*, vol. 13, no. 5, pp. 54-68, Sept./Oct. 1993.

[104] B. Burgess et al., "The PowerPC 603 microprocessor," *Comm. ACM,* vol. 37, no. 6, pp. 34-42, Apr. 1994.

[105] S. P. Song et al., "The PowerPC 604 RISC microprocessor," *IEEE Micro*, vol. 14, no. 5, pp. 8-17, Sept./Oct. 1994.

[106] D. Ogden et al., "A new PowerPC microprocessor for low power computing systems," in *Proc. COMPCON,* 1995, pp. 281-284.

[107] D. Levitan et al., "The PowerPC 620 microprocessor: a high performance superscalar RISC microprocessor," in *Proc. COMPCON,* 1995, pp. 285-291.

[108] ___, “MPC750 RISC Microprocessor User’s Manual,” Motorola Inc., 1997.

[109] M. Papermaster, R. Dinkjian, M. Jayfiield, P. Lenk, B. Ciarfella, F. O'Conell, and R. Dupont, "POWER3: Next generation 64-bit PowerPC processor design," http://www.rs6000.ibm.com/resource/technology/index.html.

[110] A. Patrizio and M. Hachman, "Motorola announces G4 chip," http://www.techweb. com/wire/story/twb19981016S0013

[111] P. Y-T. Hsu, "Designing the FPT microprocessor", *IEEE Micro*, vol. 14, no. 2, pp. 23-33, March/Apr. 1994.

[112] ___, "R10000 Microprocessor Product Overview", MIPS Technologies Inc., Oct. 1994.

[113] I. Williams, "An Illustration of the Benefits of the MIPS R12000 Microprocessor and OCTANE System Architecture," White Paper, Mountain View, CA: Silicon Graphics, 1999.

[114] ___,"The SuperSPARC microprocessor Technical White Paper", Mountain View, CA: Sun Microsystems, 1992.

[115] UltraSparc D. Greenley et al., "UltraSPARC: The next generation superscalar 64-bit SPARC," in *Proc. COMPCON,* 1995, pp. 442-461.

[116] N. Patkar, A. Katsuno, S. Li, T. Maruyama, S. Savkar, M. Simone, G. Shen, R. Swami, and D. Tovey, "Microarchitecture of Hal's CPU," in *Proc. COMPCON,* 1995, pp. 259-266.

[117] G. Goldman and P. Tirumalai, "UltraSPARC-II: the advancement of UltraComputing," in *Proc. COMPCON,* 1996, pp. 417-423.

[118] T. Hore and G. Lauterbach, "UltraSparc-III," *IEEE Micro*, vol. 19, no. 3, pp. 73-85, May/June 1999.

[119] D. Alpert and D. Avnon, "Architecture of the Pentium microprocessor," *IEEE Micro*, vol. 13, no. 3, pp. 11-21, May/ Jun. 1993.

[120] ?

[121] M. Eden and M. Kagan, "The Pentium Processor with MMX technology," in *Proc. COMPCON*, 1997, pp. 260-262.

[122] ___, "P6 Family of Processors," Hardware Developers Manual, Sept. 1998

[123] J. Keshava and V. Pentkovski, "Pentium III Processor implementation tradeoffs," *Intel Technology Journal*, pp.1-11, 2nd Quarter 1999.

[124] ___, "The Cyrix M1 architecture," Richardson, TX: Cyrix Corp. 1995.

[125] ___, "Cyrix 686 MX processor," Richardson, TX: Cyrix Corp. 1997.

[126] ___, "Nx586 Processor Product Brief," http://www.amd.com/products/cpg/nx586/nx586brf.html.

[127] ___, "AMD-K5 Processor Technical Reference Manual," Advanced Micro Devices Inc., 1996.

[128] B. Shriver and B. Smith, *The Anatomy of a High-Performance Microprocessor.* Los Alamitos, CA: IEEE Computer Society Press, 1998.

[129] ___, "AMD-K6-2 Processor Technical Reference Manual," Advanced Micro Devices Inc., 1999.

[130] ___, "AMD-K6-3 Processor Technical Reference Manual," Advanced Micro Devices Inc., 1999.

[131] ___, "AMD Athlon Processor Technical Brief," Advanced Micro Devices Inc., 1999.

[132] M. Mittal, A. Peleg, and U. Weiser, "MMX technology overview," *Intel Technology Journal,* pp. 1-10, 3rd Quarter 1997.

[133] ---, "3D Winbench 99-3D Lightning and Transformation Test," http://developer.intel.com/procs/perf/PentiumIII/ed/3dwinbench.html

[134] R. P. Colwell, R. P. Nix, J. O. Donell, D. B. Papworth, and P. K. Rodman," A VLIW architecture for a trace scheduling compiler," *IEEE*

*Trans. Computers*, vol. 37, no. 8, pp. 967-979, Aug. 1988.

[135] B. R. Rau, C. D. Glaser, and R. L. Picard, "Efficient code generation for horizontal architectures: compiler techniques and architectural support," in *Proc. 9th ISCA*, 1982, pp. 131-139.

[136] J. A. Fisher, "Very long instruction word architectures and the ELI-512," in *Proc. 10th ISCA*, 1983, pp. 140-150.

[137] B. R. Rau, D. W. L. Yen, W. Yen, and R. A. Towle, "The Cydra 5 departmental supercomputer", *Computer*, vol. 22, no. 1, pp. 12-35, Jan. 1989.

[138] ___, "TM1000 Preliminary Data Book", Philips Electronics Corporation, 1997.

[139] ___, "TMS320C6000 Technical Brief", Texas Instruments, February 1999.

[140] ___, "SC140 DSP Core Reference Manual", Lucent Technologies, Inc., December 1999.

[141] S. Hacker, "Static Superscalar Design: A new architecture for the TigerSHARC DSP Processor," White Papers, Analog Devices Inc.

[142] ___, "Inside Intel's Merced: A Strategic Planning Discussion", An Executive White Paper, July 1999.

[143] ___, "MAJC Architecture Tutorial", Whitepaper, Sun Microsystems, Inc.

[144] ___, "Crusoe Processor", Transmeta Corporation, 2000.