DECISIVE ASPECTS IN THE EVOLUTION OF MICROPROCESSORS

DEZSŐ SIMA, MEMBER, IEEE

The incessant demand for higher performance has provoked a dramatic evolution of the microarchitecture of high performance microprocessors. In this paper we focus on major architectural developments which were introduced for a more effective utilization of instruction level parallelism (ILP) in commercial, performance oriented microprocessors. We show that designers increased the throughput of the microarchitecture at the instruction level basically by the subsequent introduction of temporal, issue and intra-instruction parallelism in such a way that exploiting parallelism along one dimension gave rise to the introduction of parallelism along another dimension. Moreover, the debut of each basic technique used to introduce parallel operation along a certain dimension inevitably called for the introduction of further innovative techniques to avoid processing bottlenecks that arise. Pertinent relationships constitute an underlying logical framework for the fascinating evolution of microarchitectures, which is 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 succeeded in maintaining an incredibly rapid increase in performance. Figure 1 reviews how integer performance of the Intel family of microprocessors, for example, has been raised over the last 20 years [1], [2]. Given in terms of SPECint92, the performance has increased by the astonishingly large rate of approximately two orders of magnitude per decade.

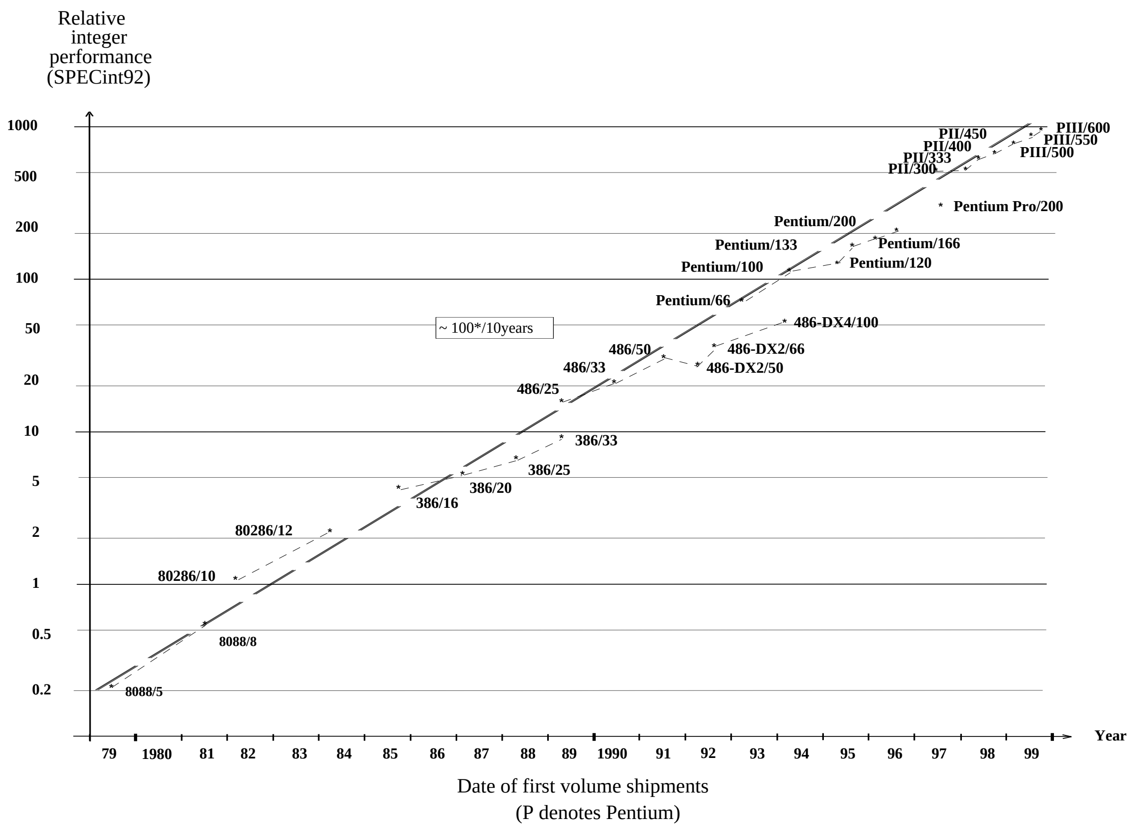


Figure 1: Increase over time of the relative integer performance of the Intel x86 processors

This impressive development and all the innovative techniques that were necessary to achieve it 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 the logical aspects, i.e. the incentives and implications of the major steps in the evolution of microprocessors.

Recently, as the techniques used to exploit available ILP mature the gap between available and exploited ILP is narrowing. This gives rise to developments basically in two major directions. (a) The first approach is to utilize ILP more aggressively. This is achieved by means of more powerful optimizing compilers and innovative techniques as discussed in section V.E. (b) The other current trend is to utilize parallelism at a level higher than the instruction level, i.e. at the thread or process level. This approach is marked by multiscalar processors [8], [9], trace processors [10] - [12], symmetrical multithreading (SMT) [13], [14] and chip multiprocessing (CMP) [15], [16]. In our paper we concentrate on the progress achieved in the first of these two areas. We explore in depth the utilization of instruction level parallelism (ILP) in commercial high performance microprocessors that are available on the market.

Our discussion begins in Section II with the reinterpretation of the notion of absolute processor performance. Our definition is aimed at considering the number of operations rather than the number of instructions executed by the processor per second. Based on this and on an assumed model of processor operation, we then identify the main dimensions of processor performance. In subsequent Sections III – VI we discuss feasible approaches to increase processor performance along each of the main dimensions. From these, we point out those basic techniques, which have become part of the mainstream evolution of microprocessors. We also identify the implications of their introduction by highlighting resulting potential bottlenecks and the techniques brought into use to cope with them. Section VII summarizes the main steps of the evolution of the microarchitecture of high performance microprocessors, followed by Section VIII which sums up the logical aspects of this evolution.

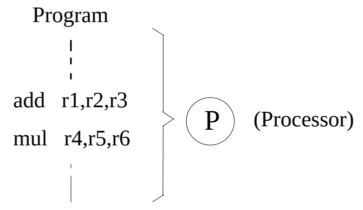
II. THE DESIGN SPACE OF INCREASING PROCESSOR PERFORMANCE

Today's industry standard benchmarks, including the SPEC benchmark suite [17] - [19], Ziff-Davis's Winstone [20] and CPUmark ratings [21] and BABCo's SYSmark scores [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 performance comparisons of processors, in processor presentations and in articles discussing the quantitative aspects of their evolution.

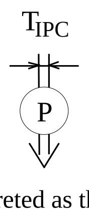
We note that computer manufacturers typically offer three product classes, (i) expensive high performance models, (ii) basic models emphasizing both cost and performance, and finally (iii) low cost models preferring cost over performance. For instance, Intel's Xeon line exemplifies high performance models, the company's Klamath line represents basic models, whereas their Celeron processors are low cost models. High performance models are obviously expensive, since all processor and system components should provide a high enough throughput, whereas low cost systems save on cost by using less ambitious and less expensive parts or subsystems.

In addition to the relative performance measures *absolute performance measures* are also used. Absolute processor performance (P_P) is usually interpreted as the average number of instructions executed by the processor per second. Nowadays, this is typically given in units such as MIPS (Million Instructions Per Second) or GIPS (Giga Instructions Per Second). Earlier synthetic benchmarks, like Whetstone [23] or Drystone [24], were also given as absolute measures.

P_P is clearly a product of the clock frequency (f_C), and the average number of instructions executed per clock cycle, called the *throughput* (T_{IPC}). Figure 2 illustrates T_{IPC} as the execution width of the processor (P).



$$P_P = f_C * T_{IPC} \quad \text{[MIPS,etc]} \tag{1}$$



T_{IPC}: Throughput, interpreted as the average number of *instructions* executed per cycle by the processor (P)

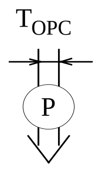
Figure 2: Usual, instruction-based interpretation of the notion of absolute processor performance

The processor's clock frequency indicates only a performance potential. Actual processor (or system) performance is further determined by the efficiency (i.e. throughput) of the microarchitecture and by the characteristics of the application processed. "Weak" components in the processor or in the whole installation, such as an inadequate branch handling subsystem of the microarchitecture or a long latency cache, may strongly impede performance.

Absolute measures are appropriate to use when the maximum performance of processors or the performance increase within particular processor lines is discussed. As our paper focuses on the evolution of microarchitectures from a performance perspective, we will apply the notion of *absolute* processor performance. However, we emphasize that absolute performance metrics are not suitable for comparing different processor lines whose Instruction Set Architectures (ISA) differ. This is because instructions from different ISAs do not necessarily accomplish the same amount of computation. For making performance comparisons in these cases, relative performance measures are needed.

As the use of multi-operation instructions has become a major trend in the recent evolution of microarchitectures, it is appropriate *to reinterpret the notion of absolute processor performance* by focusing on the number *of operations* rather than on the number of *instructions* executed per second. In this way, the notion of absolute processor performance more aptly reflects the work actually done. Here, again the *absolute processor performance* (denoted in this case by P_{PO}) can be given as the product of the clock frequency (f_C) and the *throughput* (T_{OPC}), which is now interpreted as the average number of operations executed per cycle (see Figure 3).

$$P_{PO} = f_C * T_{OPC} \quad \text{[MOPS,etc]} \tag{2}$$



T_{OPC} : Throughput, interpreted as the average number of *operations* executed per cycle

Figure 3: Operations-based interpretation of the notion of absolute processor performance

As shown in the Annex, T_{OPC} *can be expressed* by the operational parameters of the microarchitecture as follows:

$$T_{OPC} = \bar{n}_{OPC} = \underset{\substack{\downarrow \\ \text{Temporal} \\ \text{parallelism}}}{1/\bar{n}_{CPI}} * \underset{\substack{\downarrow \\ \text{Issue} \\ \text{parallelism}}}{\bar{n}_{ILP}} * \underset{\substack{\downarrow \\ \text{Intra-instruction} \\ \text{parallelism}}}{\bar{n}_{OPI}} \tag{3}$$

where

$\bar{n}_{CPI}$ is the average number of cycles between subsequent time when instructions are issued. Here we understand *instruction issue* as emanating instructions from the instruction cache/decoder subsystem for further processing, as detailed in Section V C. We note that in the literature this activity is often designated as dispatching instructions. In other words, $\bar{n}_{CPI}$ is the average length of the issue intervals in cycles.

For a traditional microprogrammed processor $\bar{n}_{CPI} >> 1$, whereas for a pipelined processor $\bar{n}_{CPI} \sim 1$. $\bar{n}_{CPI}$ reflects the *temporal parallelism* of instruction processing.

$\bar{n}_{ILP}$ is the average number of instructions issued per issue interval. For a scalar processor $\bar{n}_{ILP} = 1$, whereas for a superscalar one $\bar{n}_{ILP} > 1$. This term indicates the *issue parallelism* of the processor. Finally,

$\bar{n}_{OPI}$ shows the average number of operations per instruction, which reveals the *intra-instruction parallelism.* In the case of a traditional ISA $\bar{n}_{OPI} = 1$. Here we note that unlike RISC instructions operational CISC instructions allow to refer to memory operands as well. Consequently CISC instructions carry out on the average more complex operations than RISC instructions. For VLIW (Very Large Instruction Word) architectures $\bar{n}_{OPI} >> 1$.

Based on this model, processor performance P_{PO} can be reinterpreted as:

$$P_{PO} = \underset{\substack{\downarrow \\ \text{Clock} \\ \text{frequency}}}{f_C} * \underset{\substack{\downarrow \\ \text{Temporal} \\ \text{parallelism}}}{1/\bar{n}_{CPI}} * \underset{\substack{\downarrow \\ \text{Issue} \\ \text{parallelism}}}{\bar{n}_{ILP}} * \underset{\substack{\downarrow \\ \text{Intra-instruction} \\ \text{parallelism}}}{\bar{n}_{OPI}} \tag{4}$$

Here the clock frequency of the processor (f_c) depends first of all on the sophistication of the IC technology but also on the implementation of the microarchitecture. In pipelined designs the clock period and thus, the clock frequency, is determined by the propagation delay of the longest path in the pipeline stages. This equals the product of the gate delay and the number of gates in the longest path of any pipeline 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 i.e. 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 4).

$$P_{PO} = \underset{\substack{\uparrow \\ \text{Sophistication of the} \\ \text{technology}}}{f_C} * \underbrace{\frac{1}{\bar{n}_{CPI}} * \bar{n}_{ILP} * \bar{n}_{OPI}}_{\substack{\uparrow \\ \text{Efficiency of the processor-level} \\ \text{architecture} \\ \text{(ISA/microarchitecture)}}}$$

Figure 4: Constituents of processor performance

Equation (4) provides an appealing framework for a retrospective discussion of the major steps in increasing processor performance. According to equation (4) the main possibilities for boosting processor performance are to increase clock frequency, or to introduce and increase temporal, issue and intra-instruction parallelism, as summarized in Figure 5.

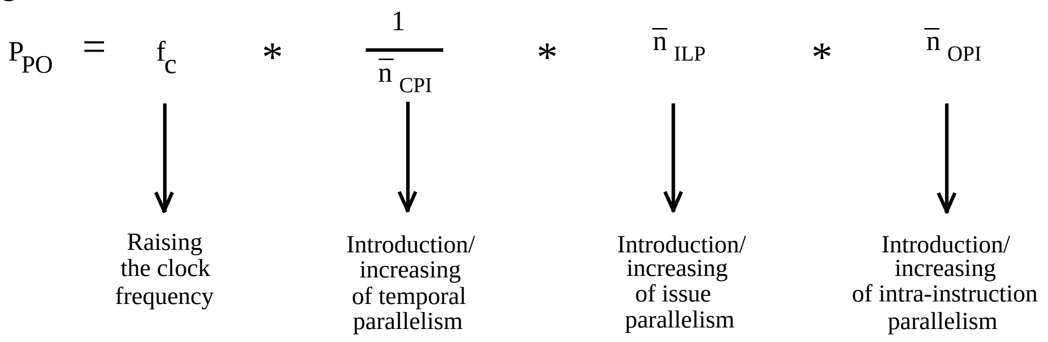

$P_{PO} = f_c * \frac{1}{\bar{n}_{CPI}} * \bar{n}_{ILP} * \bar{n}_{OPI}$
Raising the clock frequency
Introduction/ increasing of temporal parallelism
Introduction/ increasing of issue parallelism
Introduction/ increasing of intra-instruction parallelism

Figure 5: Main possibilities to increase processor performance

In the subsequent sections we address each of these possibilities individually.

III. INCREASING THE CLOCK FREQUENCY AND ITS RAMIFICATIONS

A. The Rate of Increasing the Clock Frequency of Microprocessors

Figure 6 illustrates the phenomenal increase in the clock frequency of the Intel x86 line of processors [1] over the past two decades.

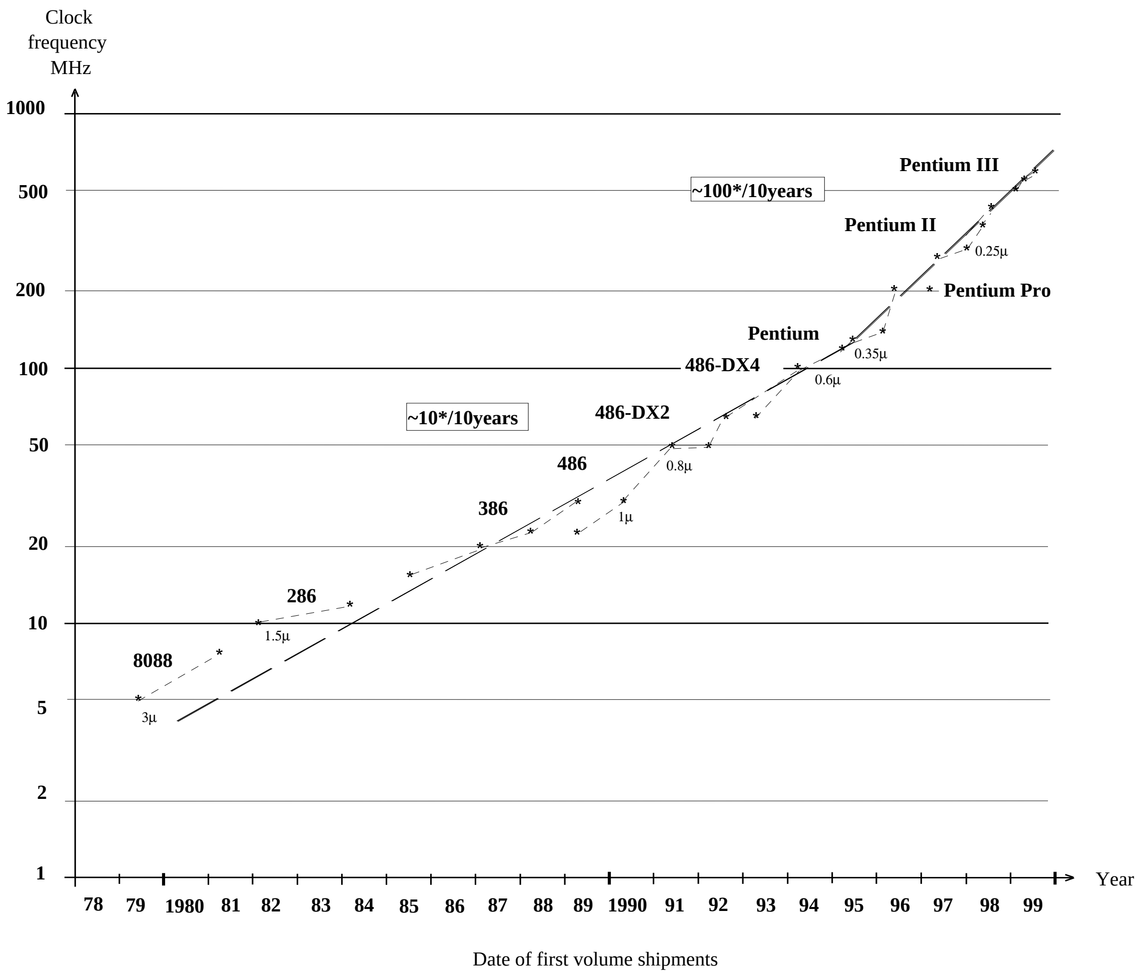


Figure 6: Historical increase in the clock frequency of the Intel x86 line of processors

As Figure 6 indicates, the clock frequency was raised until the middle of the 1990s by approximately an 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 scaling down of the chips through improved IC process technology, by using longer pipelines in the processors and by improving the 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 4), Intel's example of how it increased the efficiency of the microarchitecture in its processors is very telling.

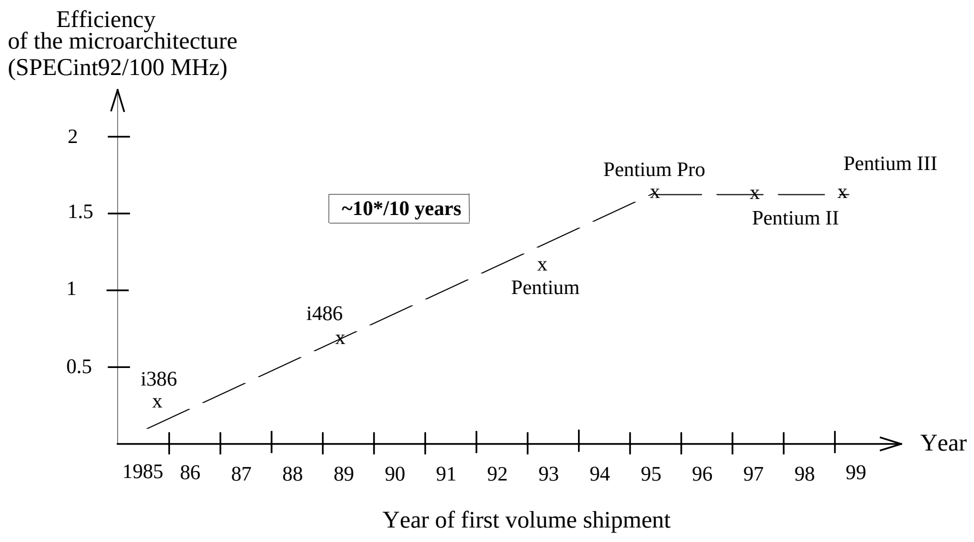


Figure 7: Increase in the efficiency of the microarchitecture of Intel's x86 line of processors

As Figure 7 shows, the overall efficiency (cycle by cycle performance) of the Intel processors [1] was raised between 1985 and 1995 by about an order of magnitude. In this decade both the clock frequency and the efficiency of the microarchitecture were increased approximately 10 times per decade, which resulted in an approximately two order of magnitude performance boost. But after the introduction of the Pentium Pro, Intel continued to use basically the same processor core in both its Pentium II and Pentium III processors[1]. The enhancements introduced, including multimedia (MM) and 3D support, higher cache capacity, increased bus frequency etc, made only a marginal contribution to the efficiency of the microarchitecture for general purpose applications, as reflected in the SPEC benchmark figures. Intel's design philosophy prefers now the increase of clock frequency over microarchitecture efficiency. This decision may stem from the view often emphasized by computer resellers that PC buyers usually go for clock rates and benchmark metrics not for efficiency metrics.

B. Implications of Increasing the Clock Frequency

In order to avoid bottlenecks in the system level architecture both raising the clock frequency of the processor and increasing the efficiency of the microarchitecture in terms of executing more instructions per cycle enforce designers to enhance both the processor bus (PC bus, front-end bus) and the memory subsystem.

1) Enhancing the processor bus: For higher clock frequencies and for more effective microarchitectures also the bandwidth of the processor bus needs to be increased for obvious reasons. This requirement has driven the evolution of processor bus standards. The progress achieved may be tracked by considering how the data width and the maximum clock frequency of major processor bus standards have evolved (see Figure 8).

[1] 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 24.

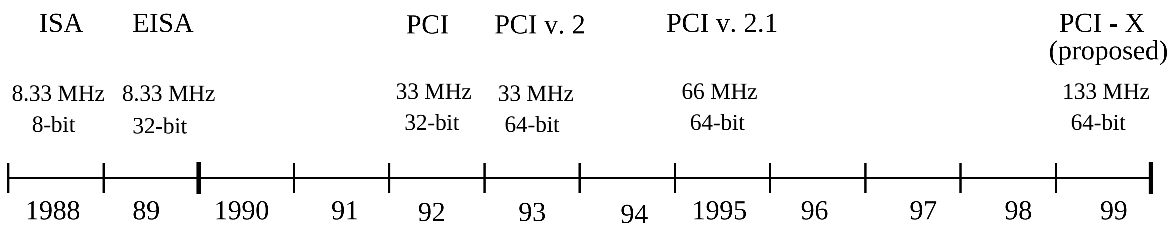


Figure 8: Evolution of processor bus standards

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

2) Enhancing the memory subsystem: Both higher clock frequencies and more efficient microarchitectures demand higher bandwidth and reduced load-use latencies (the time needed to use requested data) from the memory subsystem. There is a wide variety of approaches to achieve these goals including (a) enhanced main memory components, such as FPM DRAMs, EDO DRAMs, SDRAMs, SLDRAMs, RDRAMs, DRDRAMs [29], (b) introducing and enhancing caches, first of all through improved cache organizations, increasing number of cache levels, higher cache capacities, larger on-die cache portions [30], [31] and (c) introducing latency reducing or hiding techniques, such as software or hardware controlled data prefetching, [32], [33], lock-up free (non-blocking) caches, out-of order loads, speculative loads etc, as outlined later in Section V.E.5.b. Since this evolution is a topic of its own whose complexity is comparable to the evolution of the microarchitectures, we do not go into details here, but refer to the literature given.

Here we note that the bandwidth of the level 2 cache (L2 cache) may strongly impede system performance, first of all for small L1 caches. This is the reason for changing the way that L2 caches are connected to the processor. While L2 caches of previous models were coupled to the processor via the processor bus (for instance in the Pentium), recent high performance processors such as the Pentium Pro, Pentium II and Pentium III or AMD’s K6-3 usually provide a dedicated fast bus, called the backside bus.

IV. INTRODUCTION OF TEMPORAL PARALLELISM AND ITS RAMIFICATIONS

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 9. For sequential processing $\overline{n}_{\mathrm{CPI}}$, i.e. the average length of the issue intervals (in cycles), equals the average execution time of the instructions in cycles. In the figure $\overline{n}_{\mathrm{CPI}} = 4$. Usually, $\overline{n}_{\mathrm{CPI}} \gg 1$.

Assuming a given ISA, $\overline{n}_{\mathrm{CPI}}$ can be reduced by introducing some form of pipelined instruction processing, in other words by making use of temporal parallelism. In this sense $\overline{n}_{\mathrm{CPI}}$ reflects *the extent of temporal parallelism* in the instruction processing.

Basically, there are three main possibilities to introduce temporal parallelism by overlapping the processing of subsequent instructions; (a) overlap only the fetch phases with the last processing phase(s) of the preceding instruction, (b) overlap the execute phases of subsequent instructions processed in the same execution unit (EU) by means of pipelined execution units, or (c) overlap all phases of instruction processing by pipelined instruction processing, as shown in Figure 9. In the figure the arrows 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.

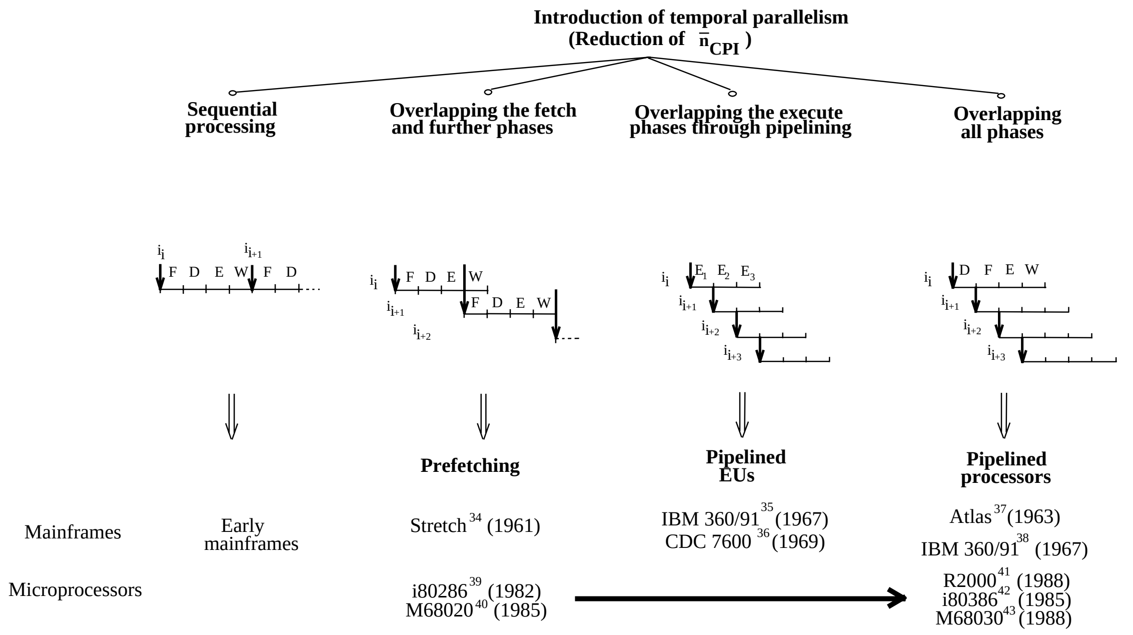

Introduction of temporal parallelism
(Reduction of $\bar{n}_{CPI}$)
Sequential processing
Overlapping the fetch and further phases
Overlapping the execute phases through pipelining
Overlapping all phases
i_i i_{i+1} F D E W F D
i_i F D E W i_{i+1} F D E W i_{i+2}
i_i E_1 E_2 E_3 i_{i+1} i_{i+2} i_{i+3}
i_i D F E W i_{i+1} i_{i+2} i_{i+3}
Prefetching
Pipelined EUs
Pipelined processors
Mainframes
Early mainframes
Stretch[34] (1961)
IBM 360/91[35] (1967)
CDC 7600[36] (1969)
Atlas[37] (1963)
IBM 360/91[38] (1967)
Microprocessors
i80286[39] (1982)
M68020[40] (1985)
R2000[41] (1988)
i80386[42] (1985)
M68030[43] (1988)

Figure 9: Main approaches to achieve temporal parallelism
(F: fetch phase, D: decode phase, E: execute phase, W: write phase)

The superscripts after the machine or processor designations are references to the related machines or processors.

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

(a) *Overlapping only the fetch phases with the last phase(s) of the proceeding instruction* is called *prefetching,* a term coined in the early days of computers [34]. Assuming that the processor overlaps the fetch phases with the write phases, as indicated in Figure 9, this technique reduces the average execution time by one cycle compared to fully sequential processing. However, control transfer instructions (CTIs), which divert instruction execution from the sequential path, make prefetched instructions obsolete. This lessens the performance gain of instruction prefetching to less than one cycle per instruction.

(b) The next possibility is *to overlap the execution phases of subsequent instructions* processed in the same EU by using pipelined execution units (EUs), [35], [36]. *Pipelined EUs* are able to accept a new instruction for execution in every new clock cycle even if their operation latency is greater than one cycle, provided that no dependencies exist between subsequent instructions. In this way, elements of vectors can be processed in a more effective way than in sequential processing, typically resulting in a considerable performance gain.

(c) Finally, the ultimate solution to exploit temporal parallelism is *to extend pipelining to all phases of instruction processing*, as indicated in Figure 9 [37], [38]. Fully *pipelined instruction processing* results in a one cycle mean time between subsequent instructions ($\bar{n}_{\text{CPI}} = 1$) provided that the instructions processed are free of dependencies. The related processors are known as *pipelined processors*, and include one or more pipelined EUs. We note that the execution phase of some instructions, such as division or square root calculation, is not pipelined in spite of pipelined instruction processing for implementation efficiency. This fact and occurring dependencies between subsequent instructions cause a slight increase of $\bar{n}_{\text{CPI}}$ during real pipelined instruction processing.

Pipelined processors ushered in the era of *instruction level parallel processors*, or *ILP processors* for short. In fact, both prefetching and overlapping of the execution phases of subsequent instructions provide already a kind of partial parallel execution. Nevertheless, processors providing these techniques alone are usually not considered to be ILP processors.

Different forms of temporal parallelism were introduced into mainframes in the early 1960s (see Figure 9). In microprocessors, prefetching arrived two decades later with the advent of 16-bit micros [39], [40]. Subsequently, because of their highest performance potential among the alternatives discussed, pipelined microprocessors emerged [41] - [43] and came into widespread use in the second half of the 1980s, as shown in Figure 10. Thus, pipelined microprocessors constitute the first major step in the evolution of prevailing microprocessors. Here we note that the very first step of the evolution of microprocessors was marked by increasing the word length 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 we discuss the evolution of the microarchitecture of microprocessors beginning with 16-bit processors.

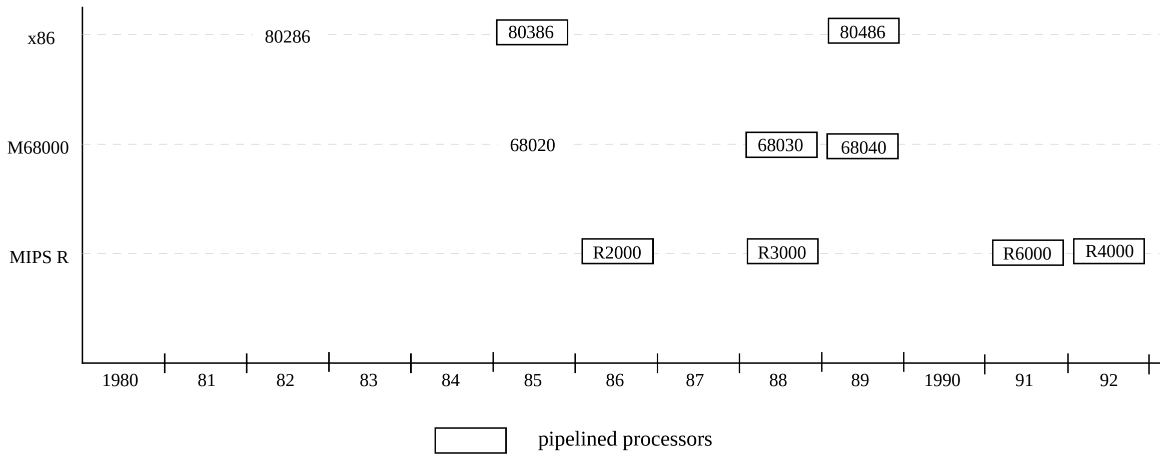


Figure 10: The introduction of pipelined microprocessors

C. Implications of the Introduction of Pipelined Instruction Processing

1) Overview: Pipelined instruction processing calls for a higher memory bandwidth and for an engineous processing of CTIs (control transfer instructions), as detailed subsequently. Thus, in order to avoid processing bottlenecks, two new techniques also needed to be introduced; caches and speculative branch processing.

2) The demand on higher memory bandwidth and the introduction of caches: If subsequent instructions are not dependent on each other a pipelined processor will fetch a new instruction in every new clock cycle. This requires a higher memory bandwidth for fetching instructions compared to sequential processing. Furthermore, due to the overlapped processing of instructions load and store instructions occur more frequently as well. Also in the case of memory architectures the processor needs to read and write more frequently memory operands. Consequently, pipelined instruction processing requires a higher memory bandwidth for both instructions and data. As the memory is typically slower than the processor, the increased memory bandwidth requirement of pipelined instruction processing accelerated and made inevitable the introduction of *caches,* an innovation pioneered in the IBM 360/85 [46] in 1968. With caches, frequently used program segments (cycles) could be held in a 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, in essence, along with the introduction of pipelined instruction processing (see Figure 11). 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. For this reason the development of caches and of their connection to the processor has remained one of the focal points of the evolution of microprocessors for more than one decade.

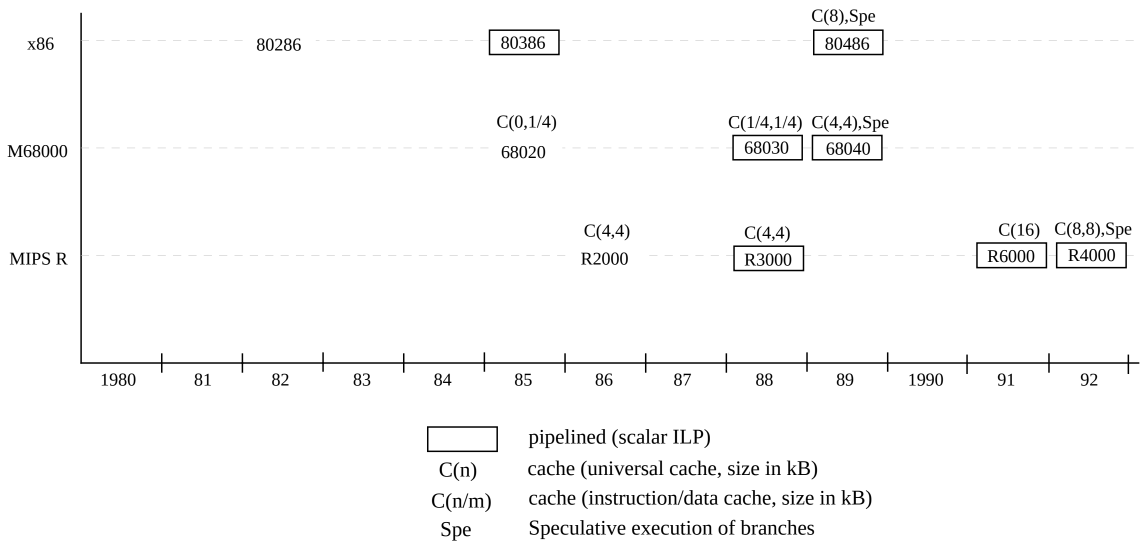


Figure 11: The introduction of caches and speculative branch processing

3) Performance degradation caused by CTIs and the introduction of speculative branch processing: The basic problem with pipelined processing of CTIs is that if the processor executes CTI's in a straightforward way, by the time it recognizes a CTI in the decode stage, it has already fetched the next sequential instruction. If, however, the next instruction to be executed is the branch target instruction rather than the next sequential one, the already fetched sequential one needs to be canceled. Thus, without any countermeasures, pipelined instruction processing gives rise to at least one wasted cycle, known as bubble, for each *unconditional CTI.*

Conditional CTIs can cause even more wasted cycles. Consider 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 will occur.

Speculative execution of branches or briefly speculative branching [47] – [50] can remedy this problem. Speculative branching means that the microarchitecture has a branch predictor that makes a guess for the outcome of each conditional branch and resumes fetching and issuing instructions along the guessed path. In this way conditional branches do not more block instruction issue, as demonstrated in Figure 12. Notice that in the figure the speculation goes only until the next conditional branch.

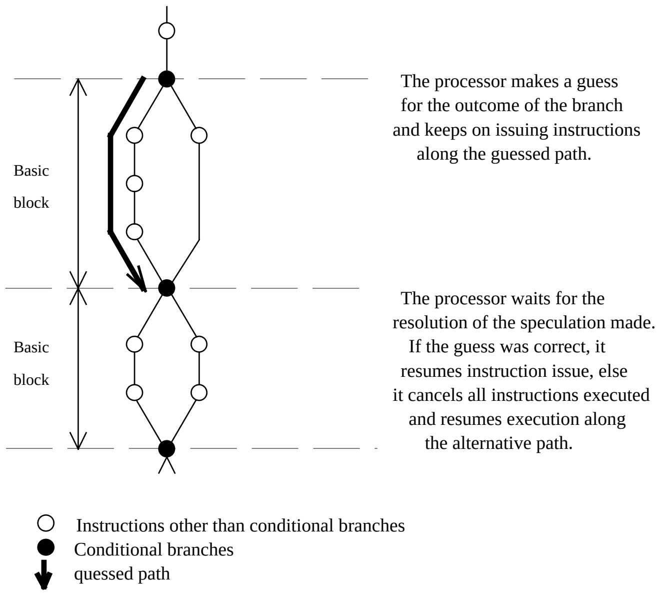

The processor makes a guess for the outcome of the branch and keeps on issuing instructions along the guessed path.
Basic block
Basic block
The processor waits for the resolution of the speculation made. If the guess was correct, it resumes instruction issue, else it cancels all instructions executed and resumes execution along the alternative path.
Instructions other than conditional branches
Conditional branches
quessed path

Figure 12: The principle of speculative execution assuming speculation along a single conditional branch

Later, when the specified condition becomes known, the processor checks whether it guessed right. In response to a correct guess it acknowledges the instructions processed. Otherwise it cancels the incorrectly executed instructions and resumes the execution along the correct path.

In order to exploit the intrinsic potential of pipelined instruction processing designers introduced both caches and speculative branch processing about the same time, as Figure 12 demonstrates.

4) Limits of utilizing temporal parallelism: With the massive introduction of temporal parallelism into instruction processing, the average length of the issue intervals can be decreased to almost one clock cycle. But $\overline{n}_{\mathrm{CPI}} = 1$ marks the limit achievable through temporal parallelism. A further substantial increase in performance needs the introduction of additional parallelism in the instruction processing along a second dimension as well. There are two possibilities for this: either to introduce issue parallelism or intra-instruction parallelism. Following the evolutionary path, we first discuss the former.

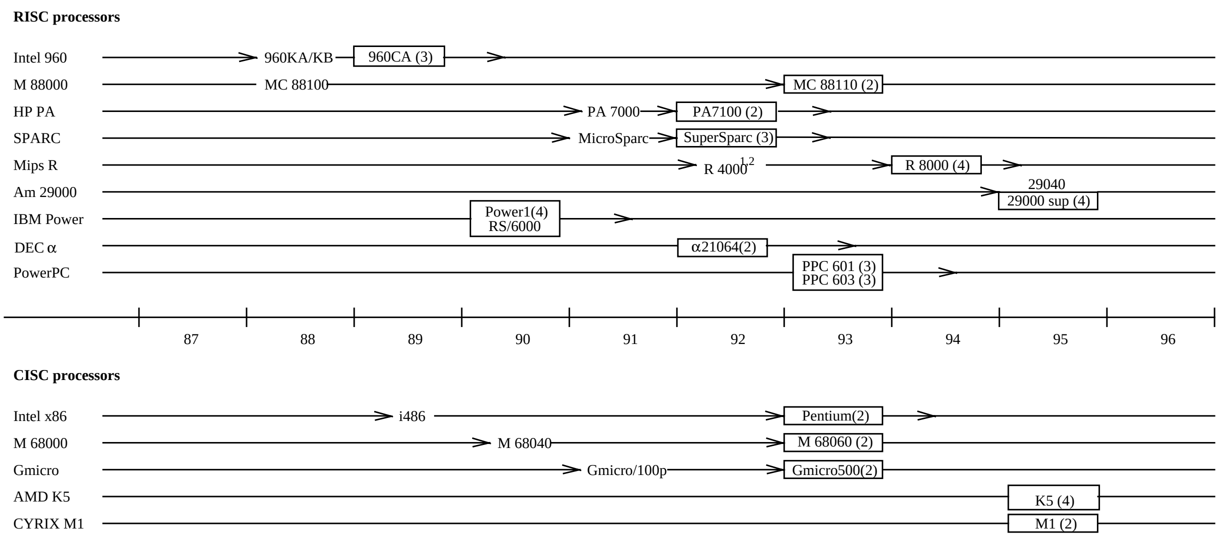
V. INTRODUCTION OF ISSUE PARALLELISM AND ITS RAMIFICATIONS

A. Introduction of Issue parallelism

Issue parallelism, also known as *superscalar instruction issue* [51] [5], [52], refers to the issuing of multiple decoded instructions per clock cycle by the instruction fetch/decode part of the microarchitecture for further processing. The maximum number of instructions issued per clock cycle is called the *issue rate* (n_i).

We note that in expression (3), which identifies the components of processor performance, issue parallelism is expressed by the average number of instructions issued per issue interval ($\overline{n}_{ILP}$) rather than by the average number of instructions issued per cycle ($\overline{n}_{IPC}$). Assuming pipelined instruction processing and superscalar issue, however, the average length of the issue intervals ($\overline{n}_{CPI}$) approaches one cycle. Thus, in expression (3) $\overline{n}_{ILP}$ equals roughly the average number of instructions issued per cycle ($\overline{n}_{IPC}$).

Issue parallelism is utilized by *superscalar processors*. They appeared after designers exhausted the full potential of pipelined instruction processing to boost performance, around 1990. Due to their higher performance, superscalars rapidly became predominant in all major processor lines, as Figure 13 shows.



1 We do not take into account the low cost R 4200 (1992) since superscalar architectures are intended to extend the performance of the high-end models of a particular line.
2 We omit processors offered by other manufactures than MIPS Inc., such as the R 4400 (1994) from IDT, Toshiba and NEC.
▭ denotes superscalar processors.
The figures in brackets denote the issue rate of the processors.

Figure 13: The appearance of superscalar processors

B. Overall implications of superscalar issue

Compared to pipelined instruction processing, where the processor issues at most one instruction per cycle for execution, superscalars issue up to n_i instructions per cycle, where n_i is the *issue rate*, as illustrated in Figure 14. As a consequence, on the average superscalars need to fetch $\overline{n}_{IPC}$ times more instructions and memory data and need to store $\overline{n}_{IPC}$-times more memory data per cycle (t_c) than pipelined processors. To put it another way, superscalars need a *higher memory bandwidth than* pipelined processors even assuming the same clock frequency. As the clock frequencies of the processors are rapidly increasing as well (see Figure 6), superscalars need an enhanced memory subsystem compared to those used with earlier pipelined processors, as already emphasized in connection with the main road of the evolution in Section III.B.2.

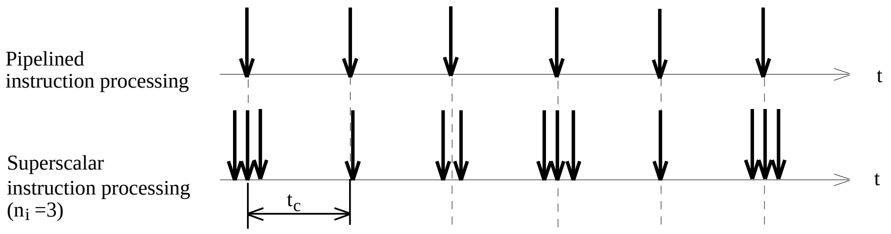
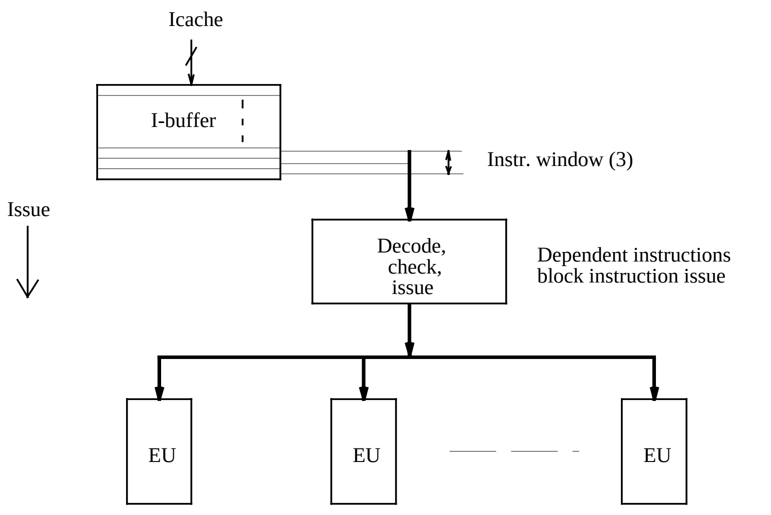

Pipelined instruction processing
t
Superscalar instruction processing (n_i =3)
t_c
t

Figure 14: Contrasting pipelined instruction processing with superscalar processing (The arrows indicate instructions)

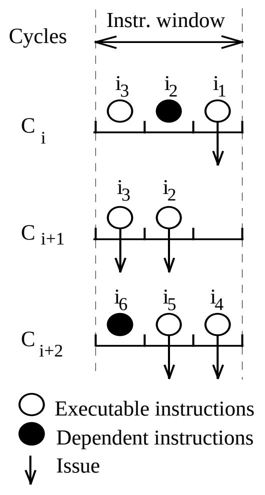
Superscalar issue also impacts branch processing. There are two reasons for this. First, with superscalar instruction issue branches occur on the average $\overline{n}_{IPC}$-times more frequently than with pipelined processing. Second, each wasted cycle that arises during branch processing can restrict multiple instructions from being issued. Consequently, superscalar processing needs a *more accurate branch speculation* or in general a *more advanced branch handling* than is used with pipelined processing. Moreover, as we will point out later in this section, one of the preconditions for increasing the throughput of superscalar processors is *to raise the sophistication of their branch handling* subsystem. For an overview of the evolution achieved in this respect we refer to [49], [53] - [55].

C. The Direct Issue Scheme and the Resulting Issue Bottleneck

1) *The Principle of the Direct Issue Scheme:* For 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 after decoding, executable instructions are issued immediately to the execution units (EUs), as shown in Figure 15. This scheme is based on an *instruction window* (issue window) whose width equals the issue rate. The window is filled with instructions from the last entries of the instruction buffer. The instructions held in the window are then decoded and checked as to they are dependent on instructions still being executed. Executable instructions are issued from the instruction window directly to free EUs. Dependent instructions remain in the window. Variants of this scheme differ on two aspects: how the window is filled and how dependencies are handled [49], [52].



(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 15: Principle of the direct issue scheme

In Figure 15b we demonstrate how the direct issue scheme works assuming an issue rate of three and the following variant of the basic scheme. (a) The processor issues instructions in order, meaning that a dependent instruction blocks the issue of subsequent not dependent instructions from the window, and (b) the processor needs to issue all instructions from the window before refilling it from the instruction buffer with the subsequent instructions. Examples of processors that issue instructions in this way are the Power1, the PA7100, and the SuperSparc. In one demonstration of this operations principle we take it for granted that in cycle c_i the instruction window is filled with the last three entries of the instruction buffer (instructions i_1 – i_3). We also suppose that in cycle c_i instructions i_1 and i_3 are free of dependencies but i_2 depends on instructions which are still in execution. Given this, in cycle c_i only instruction i_1 will be issued. Both i_2 and i_3 will be withheld in the window since i_2 is dependent and blocks the issue of any following 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 from the window. In the next cycle (c_{i+2}) the window is refilled with the subsequent three instructions (i_4-i_6) and the issue process resumes in a similar way.

2) *The Resulting Issue Bottleneck:* In the direct issue scheme all data or resource dependent instructions occurring in the instruction window block instruction issue. This fact restricts the average number of issued instructions per cycle ($\overline{n}_{IPC}$) to about two in general purpose applications [56], [57]. Obviously, when the microarchitecture is confined to issue on the average not more than about two instructions per cycle, its throughput is also limited to about two instructions per cycle, no matter how wide the microarchitecture is. Consequently, the direct issue scheme leads to an *issue bottleneck,* which limits the maximum throughput of the microarchitecture.

3) *The Throughput of Superscalar Microarchitectures That Use the Direct Issue Scheme:* From the point of view of the throughput ($\overline{n}_{IPC}$) the microarchitecture may be viewed roughly as a chain of subsystems that are linked together via buffers. Instructions are processed in a pipelined fashion as they flow through the chain of these subsystems, the kind and number of which depend on the microarchitecture in question. Typical subsystems fetch, decode and/or issue, execute as well as retire (i.e. complete in program order) instructions.

A simplified execution model of a superscalar RISC processor that employs the direct issue scheme is shown in Figure 16 below. The front end of the microarchitecture consists of the fetch and decode subsystem. Its task is to fill the instruction window.

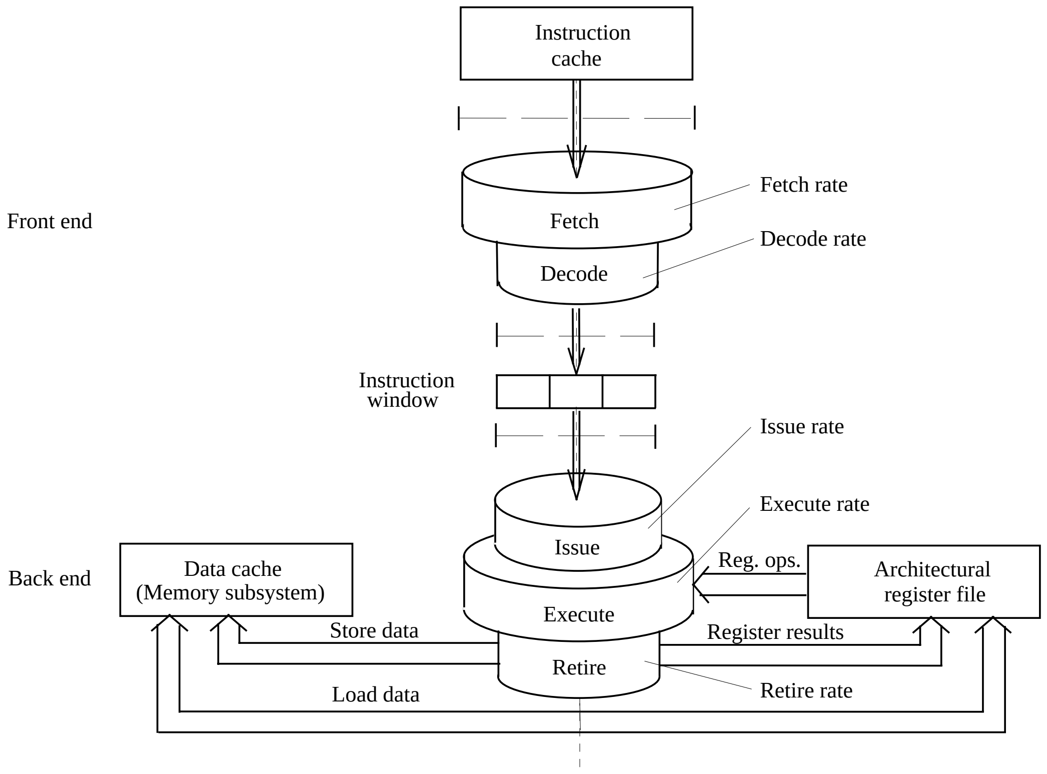


Figure 16: Simplified execution model of a superscalar RISC processor that employs direct issue

The window is depleted by the back end of the microarchitecture that includes the issue, execute and retire subsystems. In each cycle some instructions in the window are available for parallel execution, others are locked by dependencies. As EUs 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 number of instructions that are available for parallel execution in the instruction window per cycle. The issue subsystem forwards not dependent instructions from the instruction window for execution. Needed register operands are supplied from the architectural register file to the EUs, which constitute the execution subsystem. Executed instructions are completed in program order and the results generated are sent either to the architectural register file or to the memory.

Compared to RISC processors, advanced CISCs usually differ in that they convert CISC instructions into internal simple, RISC-like operations. Called differently in different processor lines (e.g. μops in Intel's Pentium Pro and subsequent models, RISC86 operations in AMD's K5 - K7, ROPs in Cyrix's M3) these internal operations are executed by a RISC kernel. The retire subsystem is then responsible for a reconversion by completing those internal operations, which are part of the same CISC instruction, conjointly.

Each of the above subsystems mentioned has a *maximum throughput* (bandwidth) in terms of the maximum number of instructions that can be processed per second. Instead of maximum throughput however, it is often more expressive to speak of the *width* of a subsystem, which reflects the maximum number of instructions that can be processed per cycle. The width of the fetch, decode, issue execute and retire subsystems is given by the fetch rate, the decode rate, the issue rate, the execution rate and the retire rate, respectively, as indicated in Figure 16. In this sense, the term *width of the microarchitecture* roughly characterizes the width of the whole microarchitecture despite the fact that the widths of its subsystems may differ. This is analogous to the notion of “word length of a processor”, which indicates the characteristic or the maximum length of the data processed.

In fact, the maximum throughput (or width) of a subsystem indicates only 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 throughput of the fetch subsystem, or the throughput of the execute subsystem depends strongly on what extent parallel executable instructions in the window can find needed hardware resources (EUs) from one cycle to the next. In any application, the smallest throughput of any subsystem will be the bottleneck that determines the resulting throughput of the whole microarchitecture.

As pointed out above, the direct issue scheme causes an issue bottleneck that restricts the average number of instructions that are available for parallel execution in the instruction window per cycle to about two instructions per cycle in general purpose applications. In accordance with this restriction, early superscalars usually have an issue rate of two to three (as indicated in Figure 13). Consequently, their execution subsystem typically consists of either two pipelines (Intel’s Pentium, Cyrix’s M1) or of two to four dedicated pipelined EUs (such as e.g. in DEC’s Alpha 21064 (now Compaq)).

In order to raise the throughput of the microarchitecture, designers of subsequent microprocessors needed to remove the issue bottleneck and at the same time to increase the throughput of all relevant subsystems of the microarchitecture. In the subsequent section we focus on the first topic, and deal with the second issue 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 basically by the use of *shelving*. However, in order to effectively capitalize on this technique, shelving is usually augmented by two additional techniques: *speculative execution of branches*, and *register renaming*.

2) *Shelving*: The basic technique used to remove the issue bottleneck is *instruction shelving*, also known as dynamic instruction issue [4], [5], [58]. Shelving presumes the availability of dedicated buffers, called *shelving buffers*, in front of the EUs as shown e.g. in Figure 17[1]. With shelving the processor first issues the instructions into available shelving buffers without checking for data- or control dependencies or for busy EUs. As data dependencies or busy execution units no longer restrict the flow of instructions, the issue bottleneck of the direct issue scheme is removed.

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

[2] Here we note that in addition to the individual shelving buffers indicated in Figure 17, there are a number of other solutions to implement shelving, as discussed e.g. in [49], [58]. For instance, Intel’s Pentium Pro, Pentium II and Pentium III use a centralized (shared) reservation station.

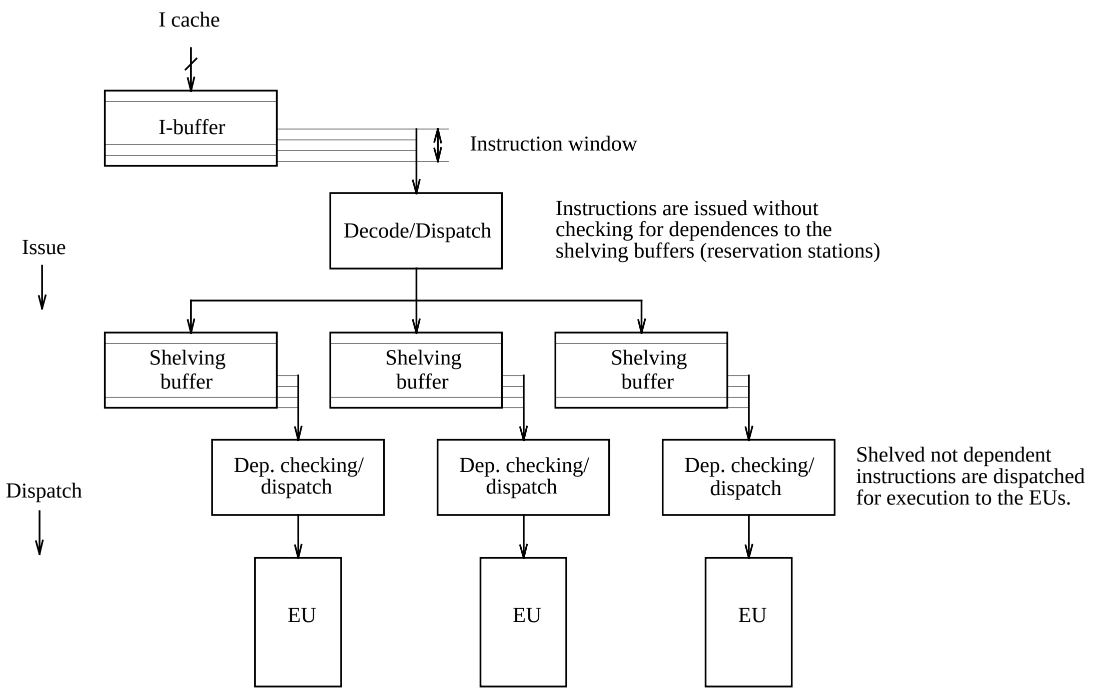


Figure 17: The principle of shelving assuming that the processor has individual shelving buffers (called reservation stations) in front of the execution units.

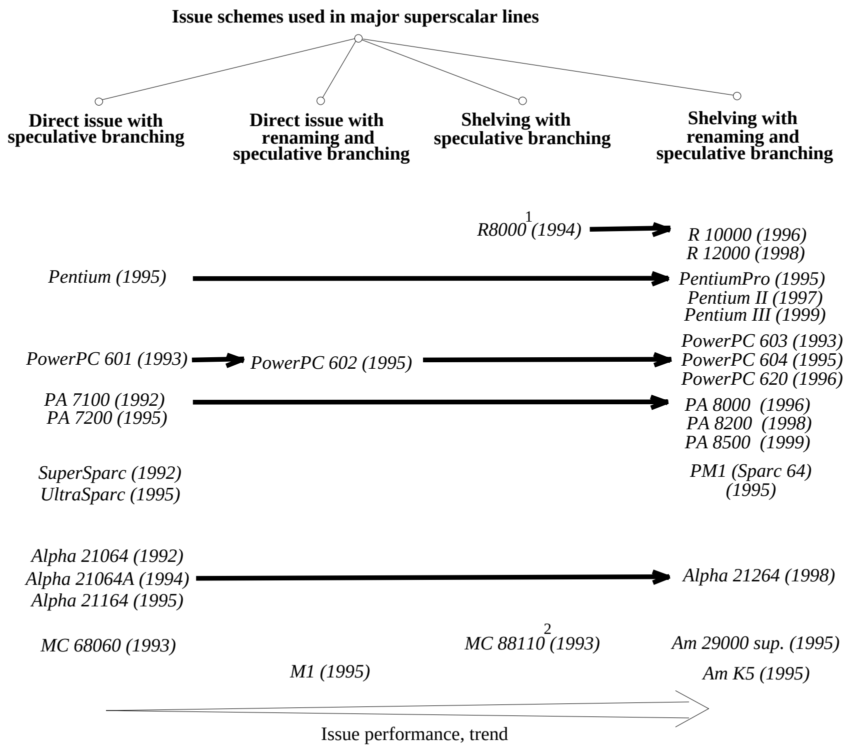
Shelving improves the throughput of the front end of the microarchitecture not only by removing the issue bottleneck of the direct issue scheme but also by significantly *widening the instruction window*. Under the direct issue scheme the processor tries to find executable instructions in a small instruction window, whose width equals its issue rate (usually 2 - 3). In contrast, with shelving the processor scans all shelving buffers for executable instructions. In this way the width of the instruction window is determined by the total capacity of all shelving 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 provide dozens of shelving buffers, shelving typically greatly widens the instruction window compared to the direct issue scheme. Since in a wider window the processor will find on the average more parallel executable instructions per clock cycle than in a smaller one, shelving also additionally increases the throughput of the front end of the microarchitecture.

3) *More Advanced Speculative Branching:* Wide instruction windows, however, call for speculation along multiple conditional branches, called *deep speculation*, in order to avoid the stalling of instruction issue due to multiple consecutive conditional branches. But the deeper branch speculation is, 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, *shelving typically requires deep speculation and a highly accurate prediction.* For this reason, the design of effective branch prediction techniques has been one of the corner stones in the development of high performance superscalars. For more details of advanced branch speculation techniques we refer to the literature [53] - [55].

4) Register Renaming: This is another technique used to increase the efficiency of shelving. 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 employs renaming, it allocates to each destination register a rename buffer that temporarily holds the result of the instruction. It also tracks actual register allocations, fetches source operands from renamed and/or architectural registers, writes the results from the rename buffers into the addressed architectural registers and reclaims rename buffers that are no longer needed [4], [5], [49].

The processor renames the destination and source registers of the 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 in the instruction window that are available for parallel execution per cycle.

Figure 18 tracks the introduction of shelving and renaming in major superscalar lines. As indicated, early superscalars typically made use of the direct issue scheme. A few subsequent processors introduced either renaming alone (like the PowerPC 602 or the M1) or shelving alone (such as the MC88110, R8000). But, in general shelving and renaming emerged conjointly in a “second wave” of superscalars, around the middle of the 1990s.



[1] The R8000 shelves only FP instructions.
[2] The MC88110, shelves only load/store instructions.

Figure 18: Introduction of shelving and renaming into superscalars

5) *The Throughput of Superscalar Microarchitectures That Use Shelving and Renaming:* RISC processors providing shelving and renaming are usually four instructions wide in design. This means that their fetch rate, decode rate, rename rate, dispatch rate and retire rate all equal four instructions per cycle.

In Figure 19 we show a simplified execution model of superscalar RISC processors that use shelving and renaming. In this model the front end of the microarchitecture includes the fetch, decode, rename and dispatch subsystems. It feeds instructions into the shelving buffers, which constitute the instruction window.

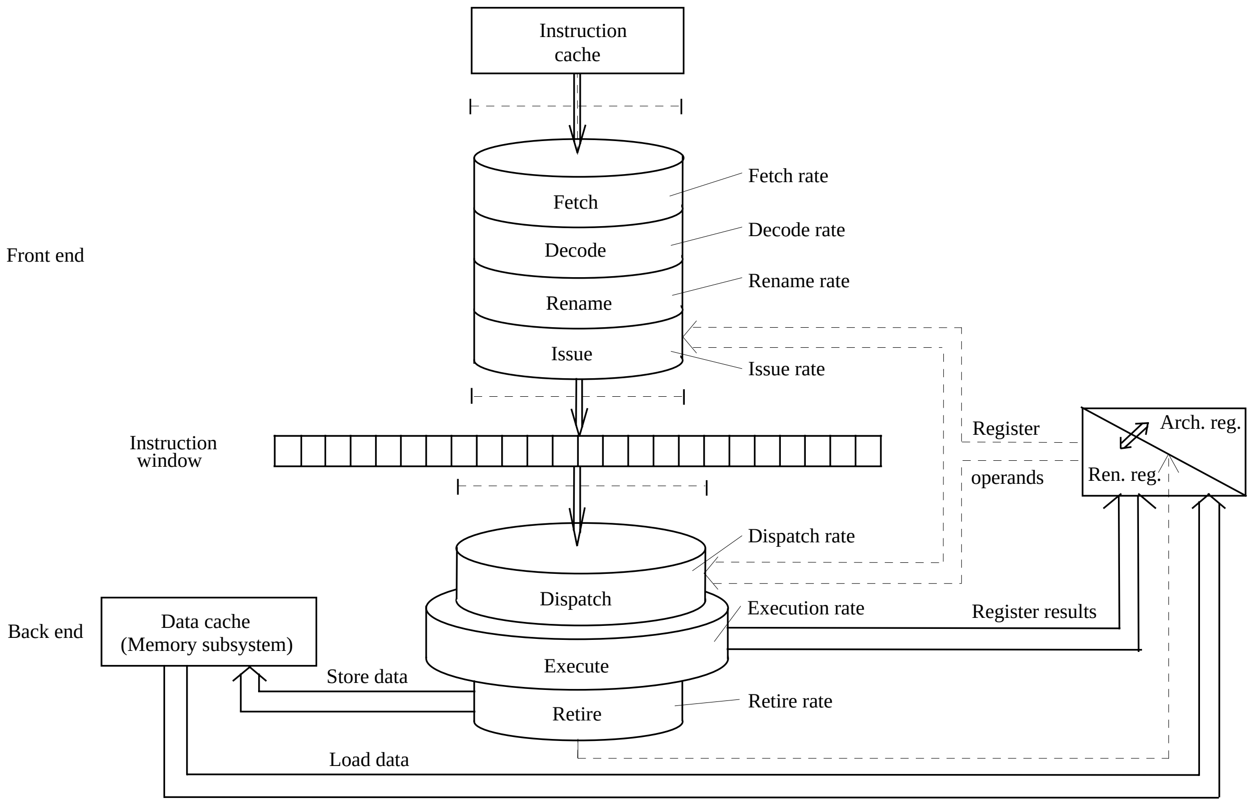


Figure 19: Simplified execution model of a superscalar RISC processor that employs both shelving and renaming

Executable instructions are dispatched from the window to available EUs. Required register operands are supplied either during instruction issue or during instruction dispatch. Register results and fetched memory data are forwarded to the rename registers, which temporarily hold all register results. Finally, executed instructions are retired in program order. At this stage register results are copied from the rename registers to the corresponding architectural registers and memory data are forwarded to the data cache in program order.

We note that the dispatch rates are typically higher than the issue rates as indicated in Figure 19. In most cases they amount to five to eight instructions per cycle (see Table 1). There are two reasons for this; (a) to sustain a high enough execution bandwidth despite complex instructions with repetition rates of more than one cycle (like division, square root etc.), and (b) to provide enough execution resources (EUs) for a wide variety of possible mixes of dispatched instructions. The execution rates are usually even higher then the dispatch rates because multiple multi-cycle EUs often share the same issue bus (that excludes the issue of multiple instructions per cycle to them) but they can operate simultaneously.

Comparison of issue and issue 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)	4	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 shelving and renaming are concerned, they typically decode up to three CISC instructions per clock cycle, and usually include an internal conversion to RISC-like operations, as discussed earlier. As x86 CISC instructions generate on the average about 1.2-1.5 RISC like instructions [59], the front end of advanced CISC processors have roughly the same width than that of advanced RISC processors in terms of RISC like operations.

It is interesting to consider how the introduction of shelving and renaming contributes to increasing the *efficiency of microarchitectures*. In Figure 20 we show the cycle by cycle relative performance of processors in terms of their SPECint95 scores, standardized to 100 MHz. Designs using shelving and renaming are identified by framed processor designations. As this figure demonstrates, superscalars providing shelving and renaming have a true advantage over microarchitectures using direct issue. In this respect comparable models are e.g. Pentium vs. PentiumPro, PowerPC601 vs. PowerPC604, PA7100 vs. PA8000, R8000 (which shelves only FP instructions) and R10000 or Alpha 21064 vs. Alpha 21264. These comparisons are slightly distorted by the fact that shelved designs are typically wider than microarchitectures with direct issue. In order to include this aspect, in Figure 20 we also indicate the issue rates of the processors in brackets after the processor designations.

We note that the UltraSparc family of superscalars is the only line that has not yet introduced shelving and renaming. In order to reduce time-to-market, designers ruled out a shelved design at the beginning of the design process [60]. This restricts the cycle by cycle throughput of the UltraSparc line well below comparable advanced RISC designs which employ both shelving and renaming (such as the R12000, PA 8200, PA8500 or the Alpha 21264).

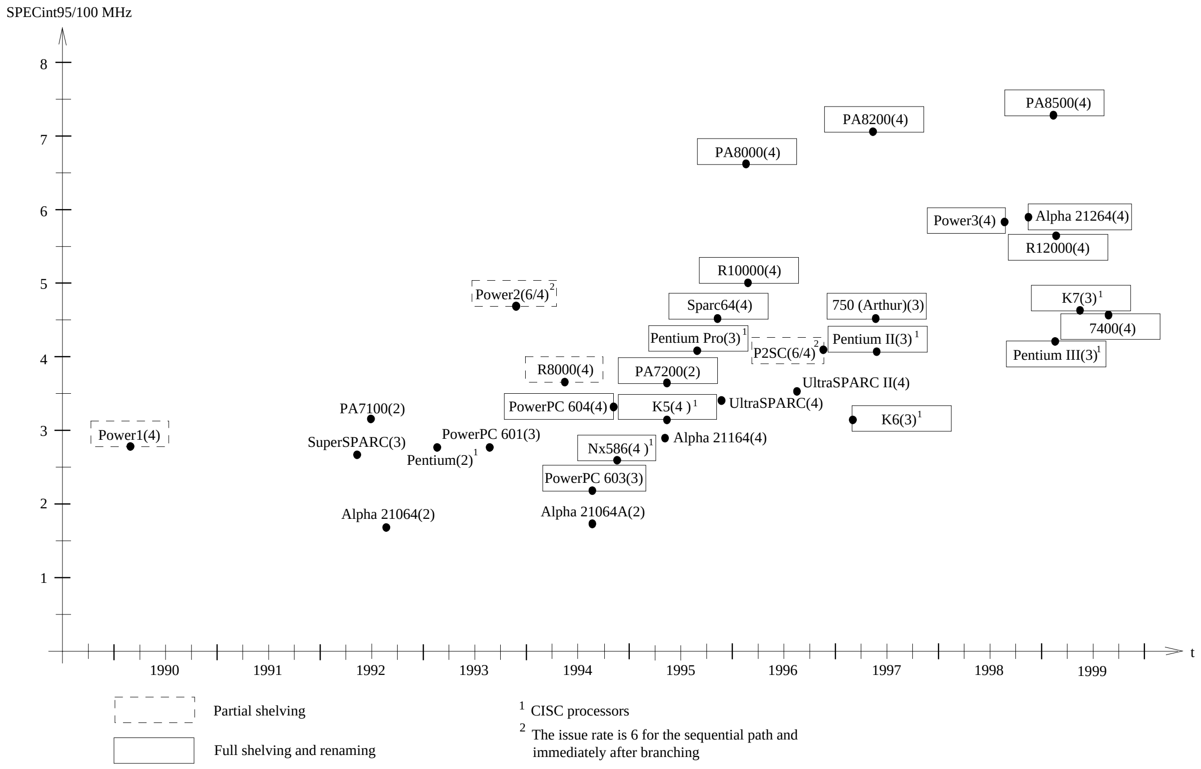


Figure 20: Efficiency of microarchitectures

Finally, we point out one important characteristic of the internal operation of superscalars that use shelving, renaming and speculative branch processing. If all these are used, only RAW dependencies between register data and memory data dependencies restrict the processor from executing instructions in parallel from the instruction window, not counting any obvious hardware limitations. Consequently, the microarchitecture executes instructions with register operands (and literals) internally according to the *dataflow principle of operation*. For these instructions basically producer-consumer type register data dependencies build the *dataflow limit of execution.*

E. Approaches to Further Increase the Throughput of Superscalar Microarchitectures

1) Overview: Further raising the throughput of the microarchitecture is a real challenge, as it requires a concerted enhancement of all subsystems involved. This usually requires numerous iterative cycle by cycle simulations of a number of benchmark applications to discover and remove bottlenecks from the microarchitecture.

Beyond shelving and renaming, there are a number of noticeable techniques that have been used or proposed to increase the throughput of particular subsystems.

2) Increasing the Throughput of the Instruction Fetch Subsystem: Ideally, the instruction fetch subsystem supplies instructions for processing at the fetch rate. However, some occurrences, for example unconditional or conditional branches or cache misses, may interrupt the continuous supply of instructions for a 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], and (c) various instruction fetching schemes to reduce the impediments of cache misses [33]. 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, decoding becomes considerably more complex than in the scalar case since multiple instructions now need to be decoded per cycle. Moreover, assuming shelving and renaming, decoding is just one part of a time critical path, which consists of decoding, renaming and dispatching of the instructions. Along this path a variety of checks need to be carried out to see whether there are enough empty rename or shelving buffers, or whether required buses are wide enough to forward multiple instructions into the same buffer, etc. As a consequence, higher dispatch rates (rates of 3 or higher) can unduly lengthen the time critical path. This would either give rise to a lower clock frequency or to additional clock cycles, which increases the penalty for mispredicted branches. An appropriate technique to remedy this problem is predecoding [49].

The fundamental idea behind *predecoding* is to perform partial decoding already when the processor fetches instructions into the instruction buffer, as indicated in Figure 21. Predecoding may include identifying the instruction type, recognizing branches, determining the instruction length (in the case of a CISC-processor), etc.

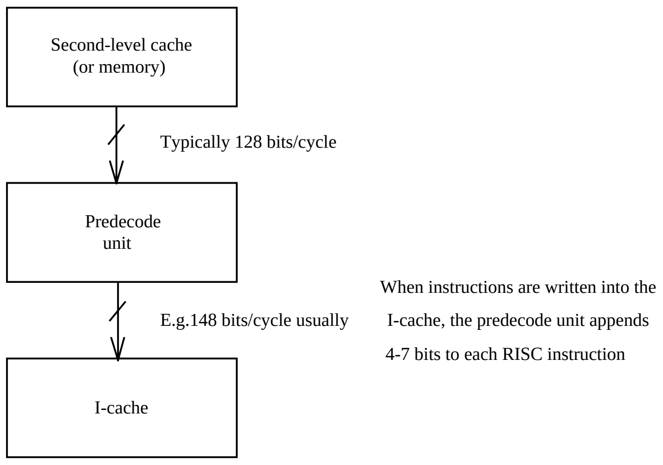

Second-level cache
(or memory)
Typically 128 bits/cycle
Predecode
unit
E.g.148 bits/cycle usually
When instructions are written into the
I-cache, the predecode unit appends
4-7 bits to each RISC instruction
I-cache

Figure 21: The basic idea of predecoding

Predecoding emerged with the second wave of superscalars about the middle of the 1990s and soon became a standard feature in RISC processors. We note that the introduction of predecoding into CISC processors is not so imperative as it is in the case of RISC processors for two reasons. First, CISC processors typically have a lower issue rate than RISC processors (mostly three in recent CISC processors). Second, they usually include an internal conversion into RISC-like operations as discussed earlier. This conversion decouples decoding and instruction issue, reducing the complexity of the decode task. We must add that trace processors, a kind of thread level parallel processors, also predecode instructions to remove complexity out of the time critical decode-rename-dispatch path [10] - [12].

4) Increasing the Throughput of the Dispach Subsystem: In order to increase the throughput of the dispatch subsystem either the issue rate needs to be raised or the instruction window needs to be widened.

(a) Raising the dispatch rate is the brute force solution to increase the throughput of the dispatch subsystem. It presumes more execution resources, such as EUs, datapaths etc. and logic for checking executable instructions in the window. For an overview of the dispatch rates of superscalars see Table 1.

(b) Widening the instruction window is a more subtle approach to raise the throughput of the dispatch subsystem. It is based on the fact that in a wider instruction window the processor will obviously find more instructions for parallel execution per cycle than in a smaller one. For this reason recent processors typically have wider instructions windows by providing more shelving buffers than preceding ones, as shown in Table 2. However, a wider window also 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: The width of the instruction window in superscalar processors that use shelving

Finally, we note that powerful parallel optimizing compilers also contribute to an increase in the average number of instructions that are available in the window for parallel execution per cycle. Nevertheless, in our paper we focus on the microarchitecture itself and do not discuss compiler issues. Interested readers are referred to the literature [61] - [62].

5) Increasing the Throughput of the Execution Subsystem

Three major possibilities exist to increase the throughput of the execution subsystem; (a) to raise the execution rate of the processor by providing more simultaneously operating EUs, (b) to shorten the repetition rates of the EUs (i.e. the number of cycles needed until an EU accepts a new instruction for execution), and (c) to shorten the execution latencies of the instructions i.e. the number of cycles needed until the result of an instruction becomes available for a subsequent instruction. Subsequently, we discuss only the last issue mentioned.

As far as execution latencies are concerned, we emphasize that if shelving and renaming are used, decoded, renamed and issued instructions wait for execution in the shelving buffers, i.e. in the instruction window. Clearly, the earlier existing RAW-dependencies are resolved in the instruction window, the more instructions will on average be available for parallel execution on the average per cycle. This calls for *shortening the execution latencies of the instructions*. Subsequently, we review techniques used or proposed either a) for register instructions or b) for load/store instructions.

a) Basically two techniques are used to shorten the execution latencies of register instructions, which are described below.

(i) *Result forwarding* is a widely used technique to shorten execution latencies of instructions operating on register data. As Figure 22 shows, result forwarding provides a bypass route from the outputs of the EUs to their inputs in order to make the results immediately available for subsequent instructions. In this way execution latencies are shortened by the time needed first to write the results into the specified destination register and then to read them from there for a subsequent instruction.

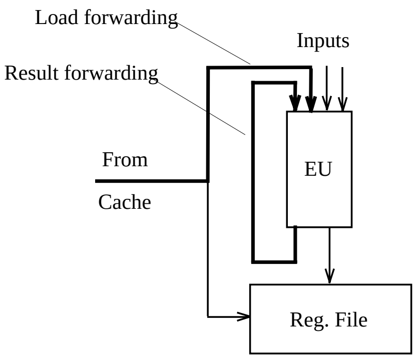


Figure 22: The principle of result and load forwarding

Implementing result forwarding requires a large number of buses, as a separate bus is needed from the output of each EU to the input to all EUs that may use it. Result forwarding is now widely used in superscalars.

(ii) Exceeding the dataflow limit of execution in the case of long register operations, such as division, by using intricate techniques like *value prediction* [63] - [66] or *value reuse* [67] - [71]. This is now a major research topic.

b) *Shortening the execution latencies of load/store instructions* is a crucial point for increasing the throughput of the microarchitecture for at least two reasons; first, load/store instructions amount to about 25 – 35 % of all instructions [72]. Second, the memory subsystem is typically slower than the processing pipeline. There are three major approaches to addressing this problem; (i) to use load forwarding, (ii) to introduce out of order loads, and (iii) to exceed the dataflow limit of execution caused by load operations.

(i) *Load forwarding* is similar to result forwarding, described above. It shortens the load latency (i.e. the time needed until the result of a load operation becomes available for a subsequent instruction) by forwarding fetched data immediately to the inputs of the EUs, as indicated in Figure 22. 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 elder, not yet executable ones. This technique effectively contributes to the reduction of the impediments of load misses. Out of order execution of loads can be implemented in a number of ways. *Speculative loads* (Power-PC 620, R10000, Sparc64, Nx586), and *store forwarding* (Nx586, Cyrix’s 686 MX, M3, K-3, UltraSparc-3) are implementation alternatives that are 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 feasible *to exceed the dataflow limit caused by load operations*. *Load value prediction* [50], [75] and *load value reuse* [85], [69], [75] are techniques proposed for this reason.

6) Limits of Utilizing Issue Parallelism: Obviously, there is a practical limit beyond which the width of the microarchitecture cannot be efficiently increased. This limit is set by the extent of instruction level parallelism available in programs. As general-purpose programs exhibit an average instruction level parallelism of about 4 - 8 [77] and recent microarchitectures already have a width of about four, there does not seem to be too much room for a performance increase through a further widening the microarchitecture, at least for general-purpose applications. Nevertheless, additional considerable performance increase may be achieved at the instruction level for dedicated use by utilizing parallelism along the third dimension, called the intra-instruction parallelism.

VI. Introduction of intra-instruction parallelism

A. Major Approaches to Introduce Intra-instruction Parallelism

Introducing intra-instruction parallelism as well, through including multiple data operations into the instructions can boost processor performance further. This can be achieved using one of three different approaches; (a) dual-operation instructions, (b) SIMD-instructions and (c) VLIW-instructions, as indicated is Figure 23.

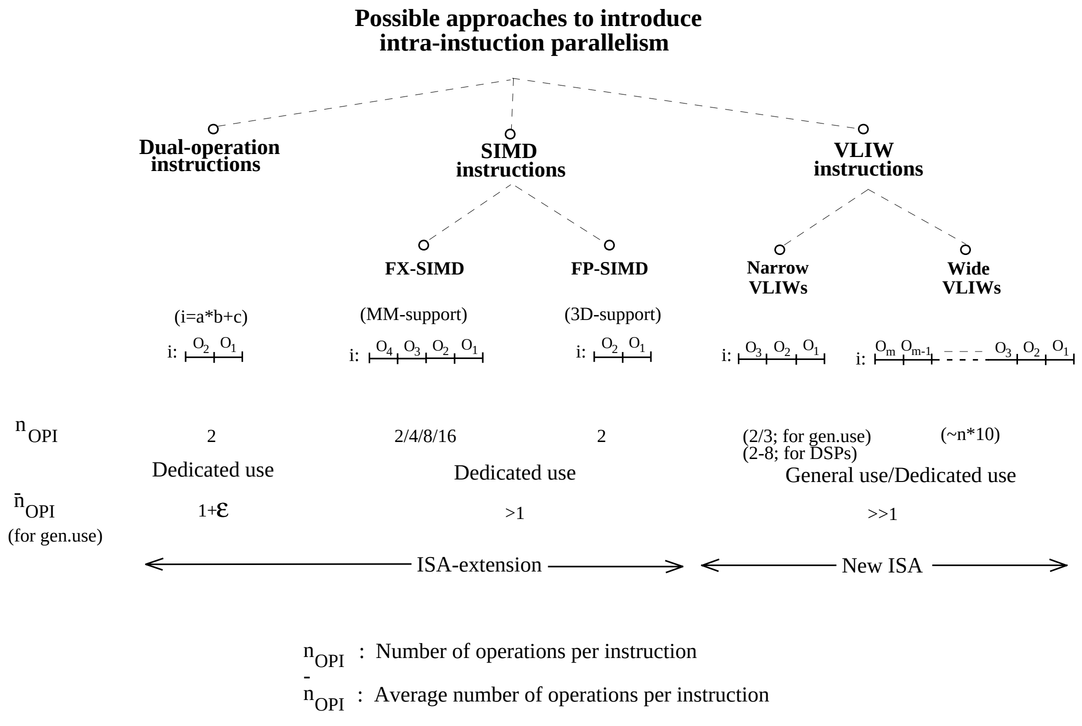


Figure 23: Possibilities to introduce intra-instruction parallelism

(a) Dual-operation instructions as the name suggests, include two different data operations in the same instruction. The most widely used one is the *multiply-add instruction* (multiply-and-accumulate or fused multiply-add instruction), which calculates the dot product (x = a * b + c) for floating-point data. Clearly, the introduction of dual-operation instructions calls for an appropriate ISA extension.

Multiply-add instructions were introduced in the early 1990s into the POWER [78], PowerPC [79], PA-RISC [80] and MIPS-IV [81] ISAs and into the respective models. The multiply-add instruction is effective only for numeric computations. Thus, in general purpose applications they only marginally increase the average number of operations per instruction ($\bar{n}_{\mathrm{OPI}}$).

(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 carries out four fixed point additions on the four 16-bit operand pairs held in the 64-bit registers MM1 and MM2.

As Figure 23 indicates, SIMD instructions may be defined either for fixed point data or for floating point data. *Fixed point SIMD instructions* support multimedia applications, i.e. multiple (2/4/8/16) operations on pixels, whereas *floating point SIMD instructions* accelerate 3D graphics by executing usually two floating point operations simultaneously. Clearly, the introduction of SIMD instructions into a traditional ISA requires an appropriate ISA extension.

Fixed point SIMD instructions were pioneered in 1993-1994 in the processors MC88110 and PA-7100LC, as shown in Figure 24. Driven by the spread of multimedia applications, SIMD extensions soon became a standard feature of most established processor families (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]). 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 processors K6-2, K6-3 and Pentium III, followed by the G4 and K7 as indicated in Figure 24.

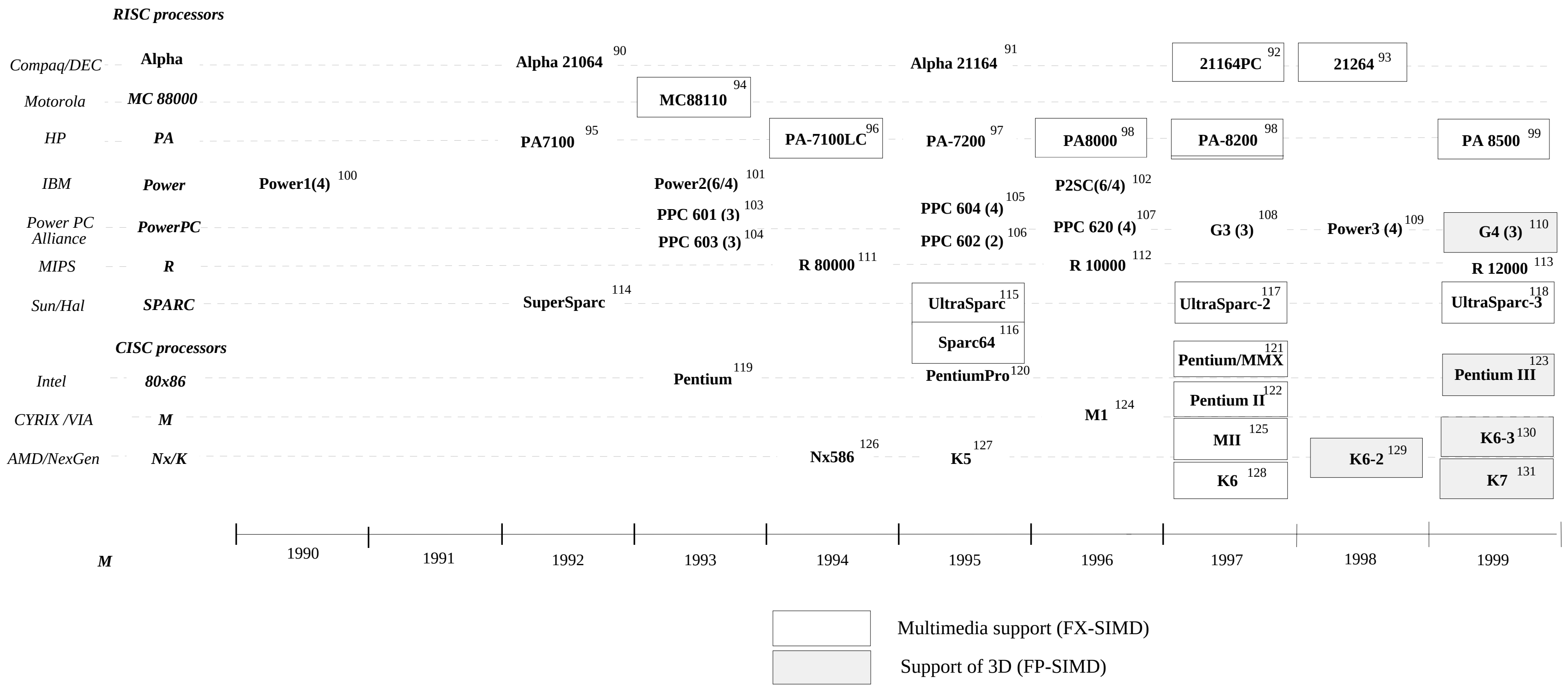


Figure 24: The emergence of FX-SIMD and FP-SIMD instructions in microprocessors
(The references to superscalar processors are given as superscripts behind the processor designations)

Clearly, multimedia and 3D support will boost processor performance only in dedicated applications. For instance, based on Media Benchmark ratings Intel stated a per cycle performance gain of about 37 % in supporting multimedia for its Pentium II over Pentium Pro [132]. Intel has also published figures showing that its Pentium III, which supports 3D, has about 61% cycle by cycle performance gain over Pentium II while 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 modest cycle by cycle performance gain for general-purpose applications. For instance, Pentium II offers only a 3-5 % cycle by cycle performance increase over Pentium Pro, whereas Pentium III shows a similarly slight cycle by cycle benefit over Pentium II in terms of SPECint95 ratings [1].

(c) The third major possibility to introduce intra-instruction parallelism is the *VLIW* (*Very Long Instruction Word*) *approach*. In VLIWs different fields of the same instruction word control simultaneously operating EUs available in the microarchitecture. As a consequence, VLIW processors with a large number of EUs 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 in the same instruction word [134].

Unlike superscalars, VLIWs 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, specifically the number, types, repetition rates, latencies of the EUs, load use latencies of the caches etc. This results on the one hand in a complex and technology dependent compiler. On the other hand, it also leads to reduced hardware complexity in contrast with 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 1980s (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 designate these traditional designs as *wide VLIWs* as they incorporate a large number of EUs, typically, on the order of 10.

Wide VLIWs disappeared quickly from the market. This was in part due to their deficiencies - technology sensitivity of their 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 in VLIWs at the end of the 1990's for both DSP and general purpose applications. *VLIW based DSPs* are intended for multimedia applications, such as Philip's TM1000 TriMedia processors [138], TI's TMS320C6000 cores [139], the SC140 core from Motorola and Lucent [140] and ADI's TigerSharc [141]. With some justification these designs can be designated as *narrow VLIWs* in contrast to earlier VLIW designs mentioned above.

General purpose narrow VLIWs with 3-4 operations per instruction are also emerging, including Intel's Itanium (alias Merced) [142], Sun's MAJC processor units used in their MCP chips [143] and Transmeta's Crusoe processors [144], which have become rivals of superscalars.

ISA extensions providing dual-operations or SIMD-instructions as well as DSP oriented VLIWs are intended for dedicated applications, by contrast traditional wide VLIWs and the latter mentioned narrow VLIWs are of general-purpose use. For general purpose applications only VLIWs are expected to carry out on the average considerably more than one operation per instruction ($n_{opi} >> 1$).

VII. THE MAIN ROAD OF THE EVOLUTIONARY PATH

As pointed out before, increasing utilization of available instruction level parallelism marks the main path of processor evolution. This has been achieved through the introduction of one after another temporal, issue and intra-instruction parallelism (see Figure 25). This sequence has been determined basically by the objective to boost performance while maintaining upward compatibility with preceding models. Nevertheless, the price to pay for increased performance is the decreasing efficiency of hardware utilization.

In this respect scalar pipelined processors, which use only temporal parallelism, led to the best hardware utilization since in essence, all stages of their pipeline are involved in the processing of instructions. Superscalar processors, which use issue parallelism as well, follow with somewhat lower hardware utilization due to the availability of multiple (parallel) execution paths. SIMD hardware extensions, which also enable the exploitation of intra-instruction parallelism, are least utilized as they are used only for MM and 3D applications. To sum this up in another way, higher per cycle throughput necessarily leads to higher hardware redundancy, as indicated in Figure 25.

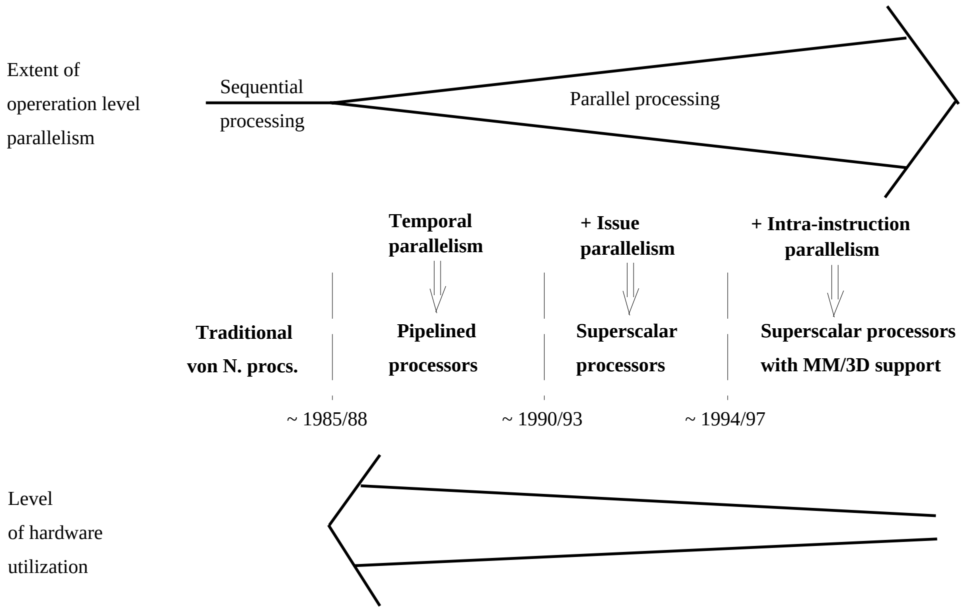


Figure 25: Main stages in the evolution of the microarchitecture of processors

We note that the history of microprocessors reveals a second possible evolutionary scenario as well. This “revolutionary” scenario is characterized by only two consecutive phases as opposed to the three that marks the evolutionary scenario, and has described before, as Figure 26 indicates.

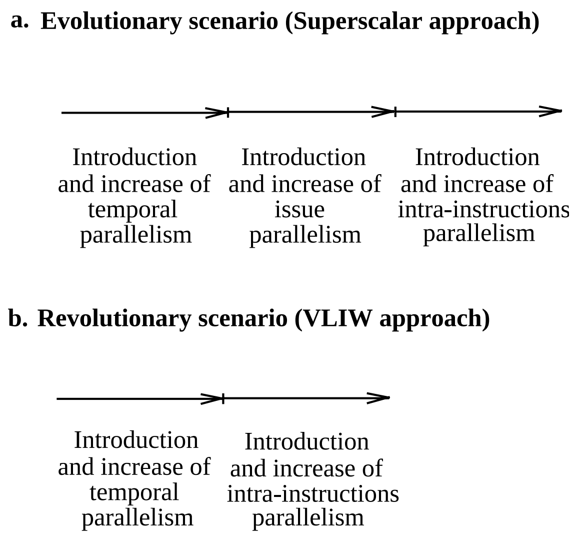

a. Evolutionary scenario (Superscalar approach)
Introduction and increase of temporal parallelism
Introduction and increase of issue parallelism
Introduction and increase of intra-instructions parallelism
b. Revolutionary scenario (VLIW approach)
Introduction and increase of temporal parallelism
Introduction and increase of intra-instructions parallelism

Figure 26: Possible scenarios for the evolution of processors

Following this path, we see that the introduction of temporal parallelism was followed by the debut of intra-instruction parallelism in the form of issuing VLIW-instructions. Clearly, introducing multiple data operations per instruction instead of multiple instructions per clock cycles is a competing alternative for boosting throughput. In broad terms, the main path was chosen not for technological reasons but because it allowed manufacturers to retain compatibility. The competing scenario represents in a sense a rather revolutionary path, as the introduction of multi-operation VLIW instructions demanded a completely new ISA. At the end of the 1980s this alternative turned out to be a dead end for wide VLIWs.

VIII. CONCLUSIONS

The steady demand for higher processor performance has provoked the successive introduction of temporal, issue and intra-instruction parallelism into processor operation. Consequently, traditional sequential processors, pipelined processors, superscalar processors and superscalar processors with multimedia and 3D support mark subsequent evolutionary phases of microprocessors, as indicated in Figure 27.

On the other hand the introduction of each basic technique mentioned gave rise to specific system bottlenecks whose resulution called for innovative new techniques.

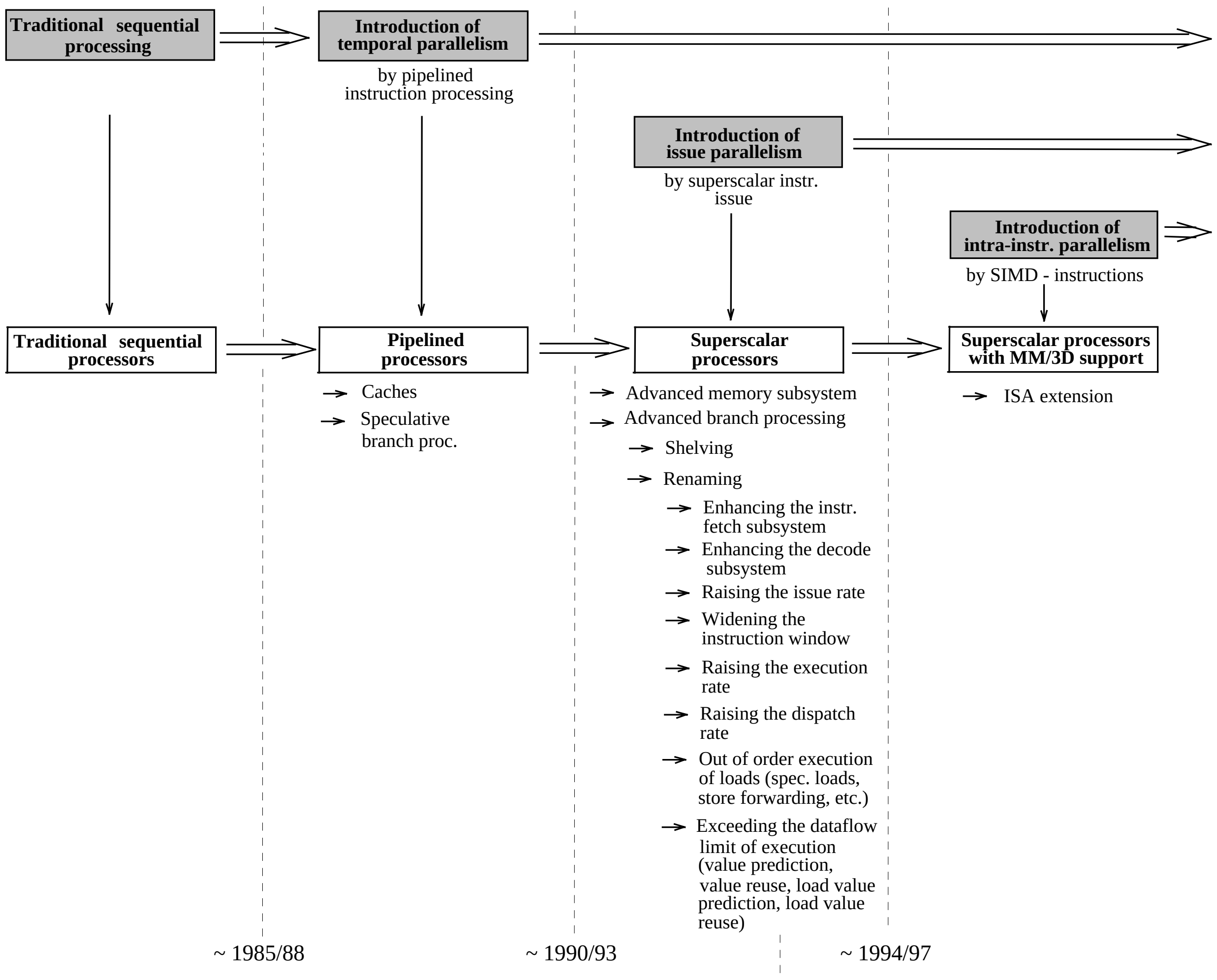

Traditional sequential processing
Introduction of temporal parallelism
by pipelined instruction processing
Introduction of issue parallelism
by superscalar instr. issue
Introduction of intra-instr. parallelism
by SIMD - instructions
Traditional sequential processors
Pipelined processors
Caches
Speculative branch proc.
Superscalar processors
Advanced memory subsystem
Advanced branch processing
Shelving
Renaming
Enhancing the instr. fetch subsystem
Enhancing the decode subsystem
Raising the issue rate
Widening the instruction window
Raising the execution rate
Raising the dispatch rate
Out of order execution of loads (spec. loads, store forwarding, etc.)
Exceeding the dataflow limit of execution (value prediction, value reuse, load value prediction, load value reuse)
Superscalar processors with MM/3D support
ISA extension
~ 1985/88
~ 1990/93
~ 1994/97

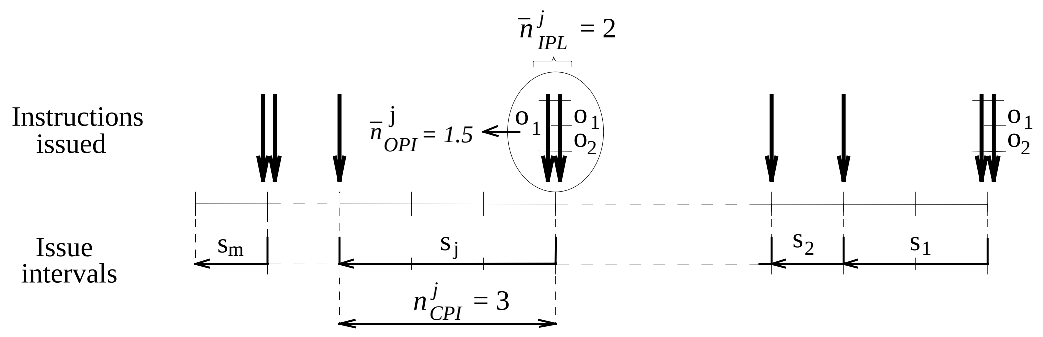
Figure 27: Major steps in the evolution of microprocessors

Thus, the emergence of pipelined instruction processing stimulated the introduction of caches and of speculative branch processing. The debut of superscalar instruction issue gave rise to more advanced memory subsystems and to more advanced branch processing. The desire to further increase per cycle performance called for avoiding the issue bottleneck of the straightforward direct issue scheme by the introduction of shelving and renaming. An additional performance increase press for a concerted enhancement of all relevant subsystems of the microarchitecture, as outlined in the paper. Finally, the utilization of intra-instruction parallelism through SIMD instructions required an adequate extension of the ISA. All in all, these decisive aspects constitute a framework, which explains the sequence of major innovations encountered in the course of processor evolution.

ANNEX

The throughput of the processor (T_{opc}). To express the throughput of the processor (T_{OPC}) by the operational parameters of the microarchitecture, we assume the following *model of processor operation* (see Figure 28).

In the figure the arrows indicate decoded instructions which have been issued for processing.



s_j	:	jth issue interval
n^j_{ILP}	:	number of instructions issued at the beginning of s_j
$\bar{n}^j_{OPI}$	:	average number of operations included in the instructions issued in s_j
n^j_{CPI}	:	length of s_j (in cycles)

Figure 28: Assumed model of processor operation

(a) We take for granted 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 by a set of three parameters which are given for each of the issue intervals $s_{j.}$, j = 1...m. The set of the chosen parameters is as follows (see Figure 28):

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 passed until the processor is ready to issue instructions again.

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

$$n^j_{OPC} = \frac{n^j_{ILP} * \bar{n}^j_{OPI}}{n^j_{CPI}} \qquad (5)$$

Now let us consider n^j_{OPC} to be a stochastic variable, which is derived from the stochastic variables n^j_{ILP}, $\bar{n}^j_{OPI}$ and n^j_{CPI}, as indicated in (5). Assuming that the stochastic variables involved are independent, the throughput of the processor (T_{OPC})

that 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:

$$T_{OPC} = \bar{n}_{OPC} = \underset{\substack{\downarrow \\ \text{Temporal} \\ \text{parallelism}}}{1/\bar{n}_{CPI}} * \underset{\substack{\downarrow \\ \text{Issue} \\ \text{parallelism}}}{\bar{n}_{ILP}} * \underset{\substack{\downarrow \\ \text{Intra-instruction} \\ \text{parallelism}}}{\bar{n}_{OPI}} \qquad (5)$$

ACKNOWLEDGMENT

The author would like to thank the anonymous reviwers for their valuable comments and suggestions on an earlies version of this paper.

REFERENCES

[1] ___, “Intel Microprocessor Quick Reference Guide,” [Online] 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] P. 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,” [Online] http://www.specbench.org/osg/cpu92/

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

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

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

[22] ___, “SYSmark Bench Suite,” [Online] 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. 8th ASCA*, 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 and 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 AISCA*, 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 AISCA*, 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 AISCA*, 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, [Online] http://www.digital.com/semiconductor/mvibackgrounder.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," [Online] 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," [Online] http://www.rs6000.ibm.com/resource/technology/index.html .

[110] A. Patrizio and M. Hachman, "Motorola announces G4 chip," [Online] 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] R. P. Colwell. and R. L. Steck, "A 0,6 µm BiCMOS processor with

dynamic execution," Intel Corp., 1995.

[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," [Online] 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," [Online] 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 AISCA*, 1982, pp. 131-139.

[136] J. A. Fisher, "Very long instruction word architectures and the ELI-512," in *Proc. 10th AISCA*, 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 Paper, 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.