

---

# OVERVIEW OF ILP ARCHITECTURES

In this chapter we trace the history of computer architecture, focusing on the evolution of techniques for instruction-level parallelism. After briefly summarizing the early years of machine design, we focus on the development of out-of-order, pipelined, and multiple-issue processors. These are further divided into processors that do instruction scheduling entirely in hardware (e.g., superscalar machines) and those that expose the instruction scheduling to the compiler, particularly VLIW machines such as the Multi-flow Trace, Cydra 5, and Itanium.

## 2.1 Historical Perspective

The primitive form of ILP, called *horizontal microcode* in current terminology, first appeared in Turing's 1946 design of Pilot ACE [CTW86], further discussed by Wilkes in [Wil51, WS53]. The initial footsteps towards parallel processing can be traced back to the Livermore Atomic Research Computer (LARC) [ECTS59] and the IBM 7030 (a.k.a. STRETCH) [Blo59] that overlapped memory operations with processor operations. The 7030 embodied several new architectural concepts, including a fully pipelined system with highly interleaved memory and speculation on branch as a bit set in the branch instruction. Other important commercial machines of the 1950s include the

IBM 704 and its successors, the 709 and 7094. The latter introduced I/O processors for better throughput between I/O devices and main memory.

In the early 1950s Engineering Research Associates (ERA) developed the transistorized ERA 1103 computer. In 1952 ERA was sold to Remington Rand, an early American computer manufacturer, best known as the original maker of the UNIVAC; Rand soon merged with Sperry Corporation. In 1957 some of the ERA founders, including William Norris and Seymour Cray, left Sperry to form CDC. In 1959 CDC released a 48-bit 1103 design as the CDC 1604. A 12-bit cut down version was also released as the CDC 160 in 1960, arguably the first minicomputer. In the early 1960s new versions of the basic 1604 architecture were re-built into the CDC 3000 series. In 1964 the CDC 6600 was released, which outperformed everything on the market at that time by roughly a factor of ten. The 6600 had a simple CPU, but used a series of external I/O processors to offload many common tasks, enabling the CPU to devote all of its time and circuitry to processing data while the other controllers dealt with tasks like punching cards and running disks. In 1969 CDC released the 7600 machine, which ran about four to five times the speed of the 6600; much of the performance improvement came from extensive use of pipelining.

In 1965 DEC developed the IMS 6100, a single chip design of the PDP-8 (Programmed Data Processors) minicomputer, a successor of the PDP-5 (1963). The 6100 was a 12-bit processor, which had exactly three registers: the PC, AC (an accumulator), and MQ. All 2 operand instructions read AC and MQ, and wrote back to AC. It had a 12-bit address bus, limiting RAM to only 4K and had no stack. Memory references were 7-bit (128 word) offset either from address 0 or the PC. The PDP-8 was succeeded by the PDP-11, designed in part by Gordon Bell. The PDP-11 was succeeded by the 32-bit VAX (Virtual Address Extension) architecture.

The 1970s saw the emergence of pipelined computers [Dav71, Sha72, DSTP75, Pat78]. Most notably, Davidson's group developed the theory of and algorithms for the design of pipelined machines [Dav74, TD74]. Patel and Davidson proposed the insertion of delays [PD76] for improving pipeline throughput; a discussion on the microprogramming of and algorithm development for pipelined processors by Kogge is available in [Kog77b] and [Kog77a] respectively. In [Lar73] Larson discussed the optimal number of pipeline stages.

Superscalar and superpipelining techniques are discussed in [JW89]. For further insight into the architecture of the early pipelined computers, the reader is advised to refer to [CK75, RL77, Kog81]. Scheduling algorithms for pipelined algorithms are reported in [BJS80, Rym82, GM86, SP89]. A survey of the instruction scheduling techniques for pipelined processors is available in [Smi89]. Techniques for code generation and code optimization for pipelined architectures are discussed in [HG82] and [Gro83] respectively.

The 1970s also saw the development of multiple processors and vector hardware for parallelism. The SOLOMON computer [GM63], developed by Westinghouse Corporation, and the ILLIAC IV [BBK<sup>+</sup>68, Dav69], jointly developed by Burroughs, the Department of Defense and the University of Illinois, were representative of the first parallel computers. The Texas Instrument Advanced Scientific Computer (TI-ASC) [Wat72] and the STAR-100 of CDC [HT72] were pipelined vector processors that demonstrated the viability of that design and set the standards for subsequent vector processors. At this time, Cray, Inc. introduced high performance vector supercomputers that had strong processors, high speed memory systems and powerful I/O systems. In 1976, the CRAY-1 [Rus78] became the supercomputer industry standard. Since vector parallelism is mostly distinct from ILP, we omit further discussion of vector processors in the rest of the book. The reader is advised to refer to the survey by [Don86] and the book by Schneck [Sch87] for a detailed description of vector architectures.

The idea of data-flow computing, and in particular data-flow graphs, originated from the formal studies in the mid-1960s [KM66, Ada68, RB69]. Data-flow computer architectures are based on the principle of activating an instruction contingent on the availability of the inputs the instruction needs. A distinctive feature of data-flow computers is that they do not have the program counter characteristic of von Neumann computers; also, they do not have addressable (i.e., random-access) memory. Dennis and Misunas first introduced an architecture for a data-flow processor in the early 1970s [DFL74, Den74, DK82] as well as an early data-flow machine [DM75]; other notable efforts were the LAU project at Toulouse, France [Pla76] and the DDM1 machine [Dav78]. The two forms of data-flow architecture are the *tagged-token architecture* and *static architecture*. The former provides hardware support for function application (including recursion) and

overlapped execution of successive loop iterations, whereas the latter requires compiler support for these programming features. The static model was best suited for regular numerical applications, whereas the tagged-token model was better suited to applications with less predictable runtime behavior, such as symbolic manipulation. Examples of the static data-flow architecture include [Cor79, THH80, HNI82, DG83]; examples of tagged-token model include [AK81] [WG82, GKW85, GDHH89, PC90]. Since data-flow architectures are not representative (in the true sense) of ILP computing, we omit further discussion of data-flow machines from this book. An overview of the evolution of the data-flow architectures is available in [Den91] and a survey of data-flow architectures is available in [Vee86].

In 1971 Intel developed the first single chip CPU, the Intel 4004 – a 4-bit processor meant for a calculator. It processed data in 4 bit units, but its instructions were 8 bits long. Program and data memory were separate, with 1K of data memory and a 12-bit PC for 4K program memory (in the form of a 4 level stack, used for CALL and RET instructions). There were also sixteen 4-bit (or eight 8-bit) general purpose registers. The 4004 had 46 instructions, using only 2,300 transistors in a 16-pin DIP and ran at a clock rate of 740kHz. The 4040 (1972) was an enhanced version of the 4004, adding 14 instructions, larger (8 level) stack, 8K program space, and interrupt abilities (including shadows of the first 8 registers). In 1973 Intel introduced the 8-bit 8008 processor.

In 1976 Texas Instruments developed one of the first true 16-bit microprocessors, the TMS 9900 (the distinction of being first probably goes to the National Semiconductor PACE or IMP-16P, or AMD 2901 bit slice processors in 16-bit configuration). It was designed as a single chip version of the TI 990 minicomputer series, much like the Intersil 6100 was a single chip PDP-8, and the Fairchild 9440 and Data General mN601 of Data General's Nova. The TMS 9900 had a 15-bit address space and two internal 16-bit registers. One unique feature was that all user registers were actually kept in memory, including stack pointers and the program counter. A single workspace register pointed to the 16 register set in RAM, so when a subroutine was entered or an interrupt was processed, only the single workspace register had to be changed, unlike some CPUs which required a dozen or more register saves before acknowledging a context switch.

Similar to the PDP-8 (and IMS 6100), the Signetics 2650 (1978) was based around a set of 8 bit registers with R0 used as an accumulator, and six other registers arranged in two sets (R1A-R3A and R1B-R3B); a status bit determined which register bank was active. The other registers were generally used for address calculations (ex. offsets) within the 15 bit address range. This design kept the instruction set simple—all loads/stores to registers went through R0. It also had a subroutine stack of eight 15 bit elements, with no provision for spilling over into memory. A number of other 16-bit processors were introduced at time, including the ZILOG Z8000 and Motorola 68000.

Advances in semiconductor technology in the 1980s had a profound impact on the computer industry, triggering the emergence of VLIW machines, which were a natural outgrowth of horizontal microcode and marking the onset of the era of modern superscalar machines. The development of VLIW and superscalar machines during the 1980s and the 1990s is discussed in Sections 2.4 and 2.5.

## 2.2 Superscalar and VLIW Machines

The goal of ILP is to speed up a single program written for a sequential processor by discovering and exploiting parallelism between individual machine instructions. Different types of ILP architectures accomplish this task in different ways. In this chapter, we define the two most important types of processors that utilize ILP and briefly trace their historical development from the sixties to the present.

Our starting point is a sequence of machine operations (e.g., memory loads and stores, integer additions, floating-point multiplications, branches, etc.), which is typically produced by a compiler for a high-level programming language such as C++ or Java. Usually there are certain *dependences* between these operations, based on which memory locations and registers are read and written by instructions in the sequence. In ILP processing, multiple machine operations may be executed in parallel; in order to preserve the original behavior of the program, the operations must be executed in such a way that their dependences are not violated. Thus, in an ILP system (compiler plus hardware) the following sequence of basic steps is involved in the execution of any program:

1. Find the dependences of the given sequence of operations;
2. Find sets of mutually independent operations (i.e., those without dependences among them);
3. Decide how to execute each set of independent operations: when should the operations execute, on which functional units, and using which registers.

Different classes of ILP architectures are characterized by the point at which the compiler hands over this sequence to the hardware.

In a *superscalar machine* [AC87], all three basic steps are handled in hardware. The processor considers instructions within a *window* of the program—a window is just a contiguous subsequence of instructions of a fixed length. Instructions that come before the window have already executed; instructions after the window have yet to execute or even be considered by the processor. The instructions inside the current window can be in a variety of states, including currently executing and waiting to execute. In each machine cycle, the processor considers those instructions inside the window that are not yet executing and identifies any instructions that can be issued in the current cycle. The processor must make sure that any operations that are issued are independent of all operations that precede it in the window (whether these operations are executing or not). The processor also tracks what machine resources (e.g., functional units) are being used by currently executing instructions; even if no dependence prevents an instruction from being issued, if a resource the instruction needs is not available the instruction must be delayed and reconsidered in future cycles. As instructions complete execution they leave the start of the window and the next instructions in the program are added to the end of the window. Some of the early superscalar processors include Apollo's DN10000 [Apo88], the ZS-1 [Smi89], the DN10000TX [BCF<sup>+</sup>91], the Metaflow architecture [PSS<sup>+</sup>91], the SuperSPARC<sup>TM</sup> [BK92], the PA-RISC processor [DWYF92] and the Motorola 88110 [DA92].

A *VLIW (Very Large Instruction Word)* machine is at the other extreme: all of the basic steps are handled in software. The compiler creates a sequence of *long instructions*, where an instruction consists of several machine operations that are to be executed simultaneously. Static parallelization enables exploitation of parallelism beyond the fixed window of operations used in superscalar machines, subject to

resource constraints. The compiler then directs the hardware to execute the operations on specific functional units and control steps.

VLIWs have long and continuing history of development and compete successfully in the processor marketplace. Notably, as early as 1989 Intel introduced a processor incorporating VLIW issue of instructions [KM89b]. In the embedded systems context, TI has had a long-running series of DSP processors (C6xxx, TM3282 and precursors) that use VLIW technology [ST09], while Philips Trimedia processors have found use in multimedia [vdWVD<sup>+</sup>05]. The SHARC DSP, by Analog Devices, is another example of a VLIW architecture that is commercially successful [Ana] as is the ST Microelectronics ST200 family based on the Lx architecture [FBF<sup>+</sup>00]. Tensilica's Xtensa LX2 processor's FLIX (Flexible Length Instruction eXtensions) is also VLIW-like [Cad]. The Infineon Carmel DSP is another VLIW processor core intended for SoC [Inf]. The Intel and HP IA-64 EPIC [Gep02, RAB<sup>+</sup>12]) Itanium, combines concepts derived from VLIW architectures with superscalar features. Finally, ELBRUS computers, started as a Soviet-era series of VLIW supercomputers e.g., [Elbb]. The company (Elbrus) still exists, and has recently announced a new multicore processor for release in 2015, the Elbrus 8C, on which little information is available in English [Elba] but which continue to utilize VLIW concepts within each core.

In this chapter, we describe some of the superscalar and VLIW machines that have been built over the years.

## 2.3 Early ILP Architectures

VLIW ideas originated in parallel microcode in computing's earliest days and in the first supercomputers. In 1964, Control Data Corporation (CDC) unveiled the 6600, considered by many to be the first supercomputer. In 1969, CDC offered the 7600, which had pipelined functional units. The IBM 360/91 was released during the same period; it employed instruction lookahead, separate floating point and integer functional units and a pipelined instruction stream. The IBM 360-195 was comparable to the CDC 7600, deriving much of its performance from a cache memory. In the 1970s, many attached array processors and dedicated signal processors used VLIW-like wide

instructions in ROM to compute fast Fourier transforms and other algorithms. We now present a brief overview of some of the early ILP architectures.

## CDC 6600

The first 6600 computer was delivered by Control Data Corporation in October, 1964. The CDC 6600 was a large-scale, solid-state, general-purpose computing system that broke away from the architecture of the IBM 7090, which was in widespread use at that time, and established a standard of performance against which machines were compared for years. The basic logic circuit used was a Direct-Coupled Transistor Logic Circuit (DCTL). The machine had a distributed architecture (central scientific processor supported by ten peripheral machines) and was a reduced instruction set (RISC) machine many years before the term was invented [PS81]. The CDC 6600 is credited with introducing many features which were novel at the time. It departed from tradition by using multiple function units, an interleaved memory, I/O peripheral processors with their own instruction sets and internal memory, and facilities for multiprogramming such as relocation registers. In this machine the hardware decided which operation to issue in a given cycle; its model of execution was that of superscalar processors.

The 6600 was a register-oriented machine. It had three sets of eight registers: eight 18-bit *A* registers for addressing, eight 18-bit *B* registers for indexing, and eight 60-bit *X* registers for arithmetic/logical manipulation. This separation of arithmetic processing from memory access time was important since clock time was 100 ns and memory cycle time was 1,000 ns (1 ms). The routing of data among registers and functional units was managed by a data traffic control called the *scoreboard*. Instructions were issued in order, with delays caused only by the unavailability of a functional unit or the result register (not by unavailability of read operands).

A key description of the CPU is ‘parallel by function.’ The CPU had 10 functional units: boolean, branch, divide, fixed add, floating add, increment (2 units), multiply (2 units), and shift, which enabled the machine to perform a sequence of independent operations in consecutive cycles. The 6600 was a three address machine with two instruction formats. The 15-bit short form instruction had 6 bits for the



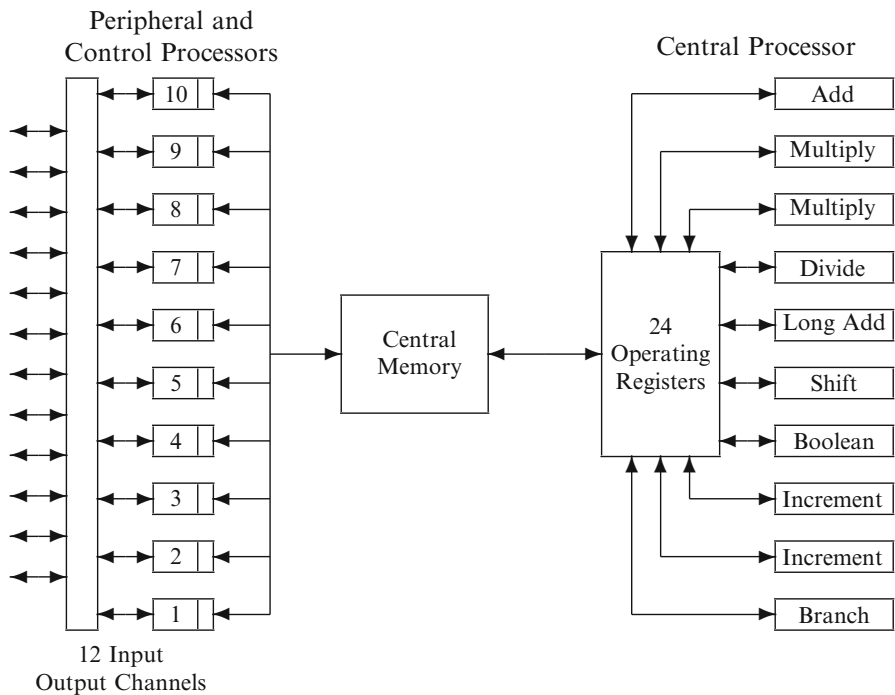


Figure 2.1 : CDC 6600 Block Diagram

opcode and 3 bits each for the result and the two operand fields. The result and operand fields specified the *A*, *B*, or *X* registers. A 30-bit long form instruction had 6 bits for opcode, 3 bits to specify a register to hold the result, 3 bits to specify a register for an operand, and 18 bits for an immediate operand. All computations were performed on quantities in registers; no arithmetic or logical instructions dealt with memory.

An instruction stack of eight 60-bit registers was used to hold a set of most recently executed instructions. An instruction fetched from the central memory first came to the stack's input register. Before it was accepted, the contents of the stack were shifted to push the oldest instruction out. This mechanism improved performance, since when control was transferred by a branch to an instruction already in the stack, that instruction did not need to be fetched from memory.

The central memory was divided into 32 banks of 4K 60-bit words. In the 17-bit address field, the low order 5 bits selected the bank and the high order 12 bits selected a word within a bank. Consecutive

words resided in different banks. Since the banks operated independently, the memory responded to multiple requests at once. This interleaving provided greater bandwidth for randomly addressed operands and maximum bandwidth for sequences of consecutive operands. There were 10 peripheral processor units (PPU's) to carry out I/O and other external tasks. Each PPU was a small high-speed processor equipped with a local memory of 4K 12-bit words and with access to the central processor's main memory. Further discussion of the 6600 architecture is available in [Tho70].

## IBM 360/91

The IBM 360/91 was a landmark in the development of instruction level parallelism. This fact was not recognized at the time, and in fact the algorithms developed by R. Tomasulo [Tom67] for hardware-based ILP exploitation were abandoned in later machines in the 70's and 80's. However, these algorithms, with those of the CDC 6600, form the core of all superscalar processors in use today. The machine adopted an assembly-line processing approach, using the techniques of storage interleaving, arithmetic concurrency, and buffering to provide high performance. Although the machine offered less ILP than the CDC 6600 (due to fewer functional units) it was far more ambitious than the CDC 6600 in its attempt to rearrange the instruction stream to keep the functional units busy—a key characteristic of superscalar machines.

The architecture of the IBM 360/91 is shown in Figure 2.2. The machine had no cache (the concept of caching was not yet recognized at that time) and relied on a 16-way interleaving of memory to increase memory bandwidth, allowing multiple memory accesses to proceed in parallel. Memory was thus divided into sixteen separate memory banks, each with its own decoding logic, and the overall addressing was designed to allow consecutive addresses to fall in separate banks. When an access request (load) was gated to a bank that was already busy servicing a previous request, the new request was buffered (set-aside) and waited until the previous request had completed; in the meantime, further requests to other banks could proceed unimpeded. As with caching, this approach to memory organization leads to variable access times; for this machine, the effective access time was 360ns, with a range of 180ns to 600ns. Three memory

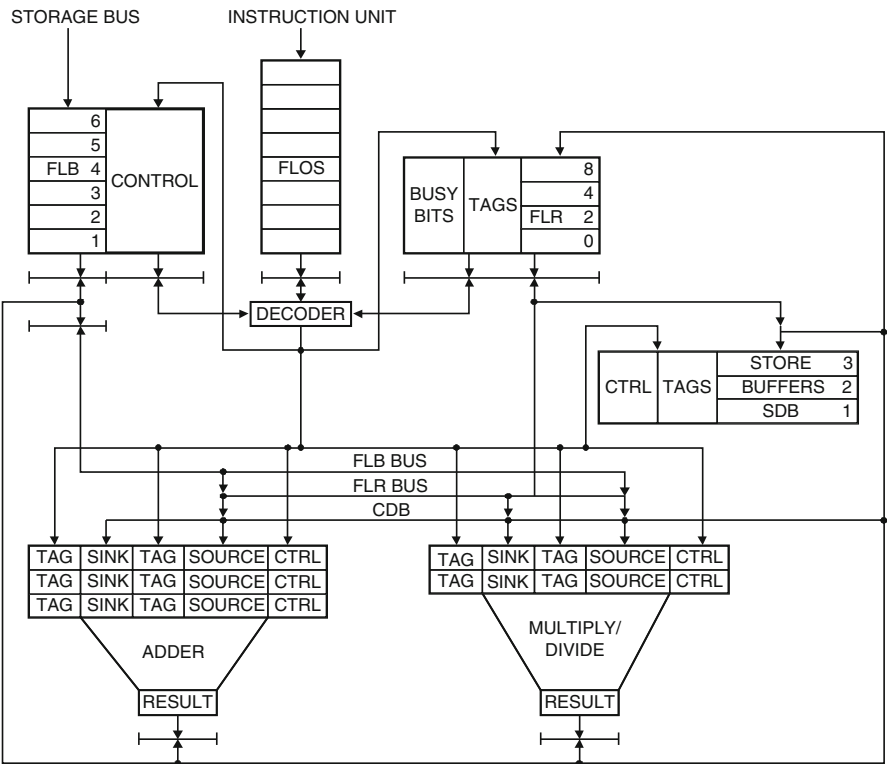


Figure 2.2 : IBM 360/91: Data Registers and Transfer Paths

address buffers and three memory data buffers provided “waiting room” for stores to memory and pending loads.

The CPU consisted of one fixed-point unit (non-pipelined, but most of the fixed-point operations executed in one 60ns cycle); a floating-point multiply/divide unit, and a floating point adder, both pipelined. Furthermore, the fetch/decode/execution of an operation was itself pipelined, with the machine fetching one new operation every cycle. The machine instruction (operation) set was, as was customary for virtually all machines of that vintage, a complex one (CISC) with most operations being able to accept one of their operands as a memory address (rather than a register). The format of the operations was of the two operand type, where the first (leftmost) operand served as both a source and destination. Fast algorithms for multiply and divide operations and carry lookahead adders were employed to provide additional concurrency. Further, the floating-point

units were linked to an internal bus to support parallel dispatch of operations to the execution units.

Once fetched and decoded, operations were issued to the various units for execution. Since the actual execution of an operation, and indeed the overall fetch/decode/fetch-operands/execute/store-result cycle for each instruction, took substantially more than the 60ns cycle (pipelining) it was quite likely that preceding operations, whose results were needed by an operation being readied for issue, were not yet available. In an attempt to minimize the impact of both operation and memory access latencies and their disruption of the pipeline, and enhance chances for parallel execution, extensive buffering and lookahead techniques were used. Eight instruction-fetch buffers held prefetched instructions and effectively provided primitive caching of short loops. Processing in loop mode was also possible and the implementation was similar to that of the CDC 6600. A detailed description of the architecture is available in [AST67].

## 2.4 ILP Architectures in the 80's

The first true VLIW machines were mini-supercomputers in the early 1980s from three companies: Multiflow, Culler, and Cydrome. The Multiflow TRACE provided multiple functional units to facilitate parallel execution of operations; seven 32-bit opcodes were packed into a 256-bit VLIW instruction. The Culler-7, based on the Harvard architecture, was a decoupled access-execute (DAE) system from Culler Scientific Systems. The design consisted of an "A" machine to control program sequencing and data memory addressing and access, and a microcoded "X" machine for floating-point computations that could run in parallel with the A machine. The Cydra 5, developed at Cydrome, also had multiple functional units to support VLIW execution; in addition, it also had a special mode wherein operations in an instruction could be executed sequentially. Although these machines were not commercially successful, the compiler techniques that were developed were very influential. Trace scheduling and software pipelining, pioneered by Fisher and Rau, respectively, are now central pillars of VLIW compiler technology. Other early VLIW efforts include Warp [AAG<sup>+</sup>85, GKLW85, AAG<sup>+</sup>87] and ChoPP [MPS87].

## Cydra 5

The Cydra 5, a heterogeneous multiprocessor, was an outcome of eight years of research at TRW Array Processors and at ESL (a subsidiary of TRW). The machine was comprised of a numeric processor and an interactive processor for all non-numeric work. The numeric processor was based on the directed-dataflow model of computation [Rau88], a successor of the polycyclic architecture developed at ESL in 1981 [RG81, RGP82]. The Motorola 68020 microprocessor was used as the interactive processor. A scheme proposed in [YYF85] was used to maintain cache coherence in the multiprocessor environment. The 64-way interleaved main memory provided high memory bandwidth. A maximum of two I/O processors could be attached to facilitate high bandwidth I/O transactions.

The architecture provided special features to enhance parallel execution of loops. The Complex Register Matrix supported overlapped execution of loop iterations; further, it could capture the complex resource usage patterns to facilitate *modulo scheduling* of loops (discussed in Chapter 6). To deal with *recurrences* (a recurrence is the use of a data value calculated in a previous loop iteration by the current loop iteration) the architecture allocated a context frame for each loop iteration. Values within a loop iteration were stored in a register context distinct from the register context of the previous iterations. Values stored within previous register contexts were accessible from the current iteration, facilitating the computation of recurrences. Further, the Conditional Scheduling Control hardware facilitated overlapped execution of loops with conditionals; the special hardware allowed conditional execution of operations based on certain predicate values.

The numeric processor of the Cydra 5 had seven functional units: a floating-point adder/integer ALU, a floating-point multiplier/divider, two memory reference ports, two address arithmetic units and a branch unit. Each functional unit was pipelined. The numeric processor supported two different instruction formats. The MultiOp instruction format had seven operations, similar to a VLIW instruction except for the existence of a predicate specifier. The typical format for each operation consisted of an opcode, two source register specifiers, one destination register specifier and a predicate register specifier. On the other hand, the UniOp instruction format allowed an issue of a single operation per instruction. The rationale behind

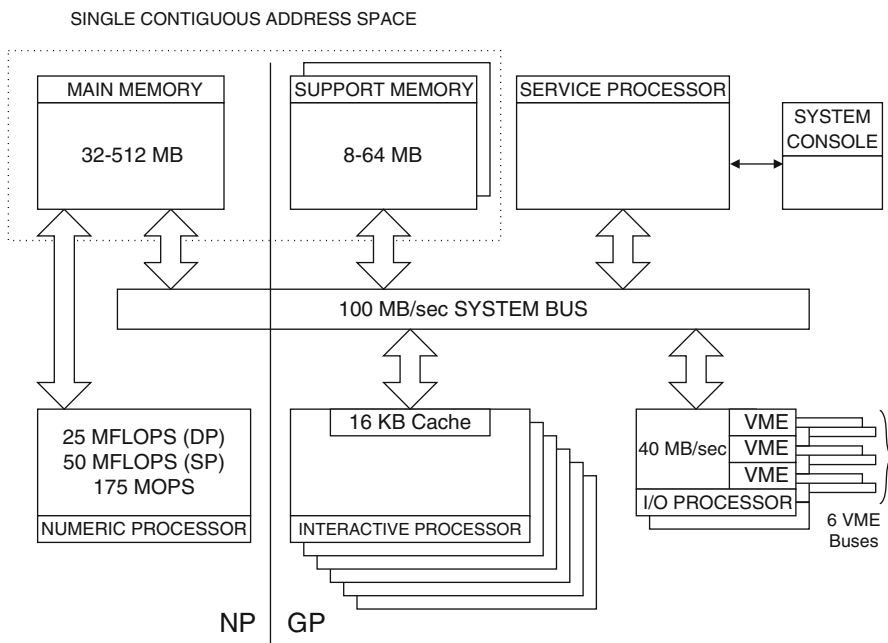


Figure 2.3 : Cydra 5 Block Diagram

the introduction of the UniOp instruction format was to mitigate the effect of MultiOp format on instruction cache performance for portions of code with little parallelism available. A detailed description of the architecture is available in [Rau88, SM88, RYYT89].

## Multiflow TRACE

Multiflow Computer Inc. built the world's first VLIW computer along with—and perhaps more importantly—a compiler that could expose ILP. Multiflow showed that a compiler could find large amounts of parallelism (well beyond what was commonly believed to be the limit [TF70, FR72]) by scheduling beyond basic blocks boundaries, using a technique called *trace scheduling* (discussed in Chapter 4). There were three series of Multiflow machines, viz., the 200 series, the 300 series and the 500 series. The 200 and the 300 series came in three widths: a 7-wide, which had a 256-bit instruction; a 14-wide, which had a 512-bit instruction; a 28-wide with a 1024-bit instruction. The wider machines were realized as a combination of 7/300s. We briefly discuss the 300 series.

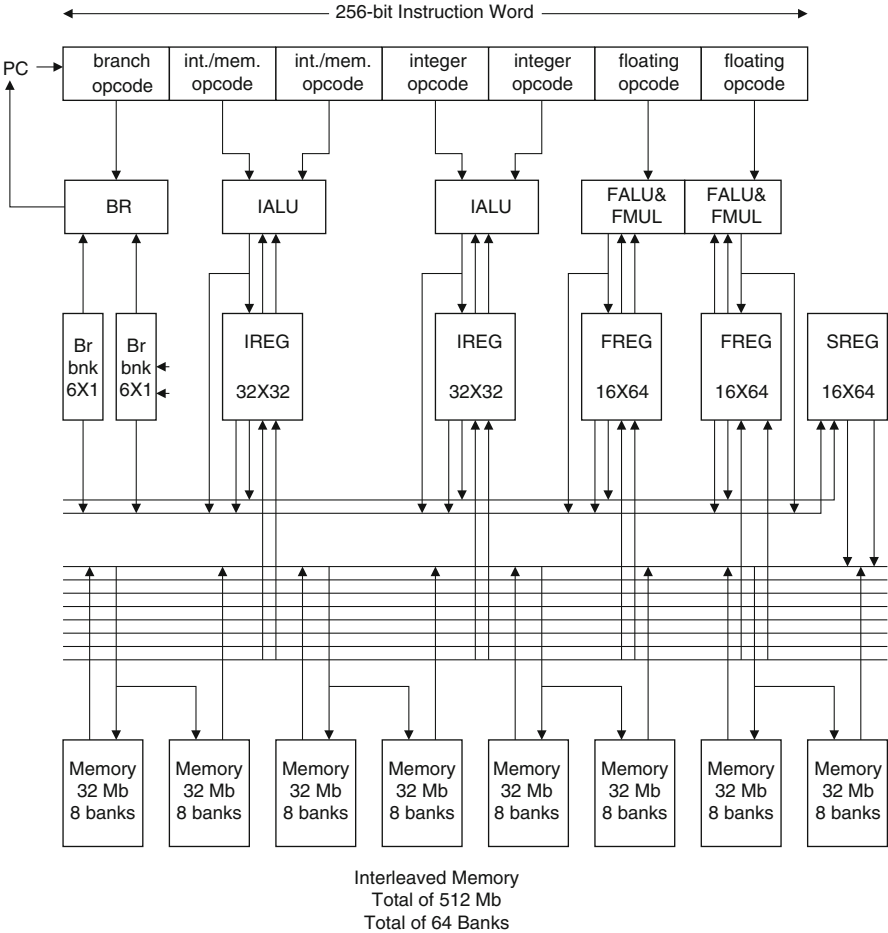


Figure 2.4 : The Multiflow TRACE 7/300 Block Diagram

The architecture of Multiflow TRACE 7/300 is shown in Figure 2.4. The core of the computation engine was built using 8000 gate CMOS gate arrays with 154 signal pins. Advanced Schottky TTL was used for “glue” logic and bus transceivers. The machine had two integer units, two floating-point units, and a branch unit. The integer units could each perform two operations per 130ns cycle (four in all), whereas a floating-point unit could perform one operation per cycle. Separate register files (sixty four 32-bit registers) were provided to the integer and floating point units; each register file could handle eight reads, writes and bus-to-bus forwards, plus bypassing

from each write port to read port. The integer unit also included dynamic address translation hardware (TLB) and supported demand-paged virtual addressing. The machine had an instruction cache of 8K but had no data cache. The memory systems supported 512 MB of physical memory with up to 64-way interleaving, thus providing enormous memory bandwidth; the memory system was pipelined to exploit parallelism among memory references. It had two I/O processors which supported a 246 MB/sec DMA channel to the main memory.

Operations were RISC-like: fixed 32-bit length, fixed-format, three register operations with memory accessed through explicit loads and stores. Operations could be completed either in a single cycle or were explicitly pipelined. In the same spirit as RISC machines such as MIPS [HJP<sup>+</sup>82] and the IBM 801 [Rad82], the microarchitecture was exposed to the compiler for resource management. However, TRACE provided many more functional units to facilitate exploitation of parallelism beyond basic block boundaries. An instruction could contain 4 integer operations, 2 floating-point operations and a branch operation. A detailed description of the architecture is available in [CNO<sup>+</sup>88, LFK<sup>+</sup>93].

## 2.5 ILP Architectures in the 90's

After a few years of operation, Cydrome and Multiflow both closed their doors after failing to establish a large enough market. HP hired Bob Rau and Mike Schlansker of Cydrome, and they began the FAST (Fine-grained Architecture and Software Technologies) research project at HP in 1989; this work later developed into HP's PlayDoh architecture [VK94]. In 1990, Bill Worley at HP started the PA-Wide Word project (PA-WWW, also known as SWS, SuperWorkStation).

In 1992, Worley recommended that HP seek a manufacturing partner for PA-WWW, and in December 1993 HP approached Intel; co-operation between the two companies was announced in June 1994. The term EPIC (Explicitly Parallel Instruction Computing) was coined to describe the design philosophy and architecture style envisioned by HP, and the instruction set architecture was named IA-64. Itanium is the name of the first implementation (it was previously known by the project name Merced) of the IA-64 [SA00]. Because Itanium



exposed instruction scheduling to the compiler (in contrast to doing all instruction scheduling dynamically in hardware), we discuss the Itanium design in more detail in Section 2.6.

The late 80's and the early 90's saw a great deal of activity in the development of VLIW [KM89a, LS90], superscalar [Gro90, McG90, NNN<sup>+</sup>91] and superpipelined machines [JW89]. In the late 1980s, Intel introduced the i860 RISC microprocessor and IBM introduced the first superscalar workstation RS/6000. The i860 had two modes of operation: a scalar mode and a VLIW mode. In the VLIW mode, the processor always fetched two instructions and assumed that one was an integer instruction and the other floating-point. A single program could switch between the scalar and VLIW modes, thus implementing a crude form of code compression.

Today the VLIW philosophy is extensively used for designing processors for non-numerical applications such as the ST VLIW core for IP video telephony, the MAP1000A VLIW Mediaprocessor for media applications [BLO00]. Philips Semiconductors' VLIW TriMedia TM1300 processor chip, designed for high end applications such as video conferencing, TV set-top boxes, and digital video cameras, can issue up to 5 operations in parallel; the maximum clock frequency is 166MHz. It has 128 32-bit general purpose registers and on-chip hardware support for audio-video I/O. Similarly, the TI TMS320C6701 processor, designed for DSP applications, can execute up to 8 operations per cycle (6 float, 2 integer). It has 128K on-chip RAM and thirty two 32-bit registers [TMS]. The internal memory has two blocks: 64 KB for program memory which is configurable as cache or memory-mapped program space and 64 KB for internal data memory. The Transmeta Crusoe TM5700 and TM5900 processors provide power-efficient VLIW solutions for embedded designs. Each supports multiple integer and floating-point execution units, separate 64 KB instruction and data caches, MMU and multimedia instructions. In the rest of this section, we briefly discuss some contemporary ILP architectures.

## Sun UltraSPARC

SPARC Version 9 (SPARC-V9) was the most significant change to the SPARC architecture [WG00] since its release in 1987. SPARC-V9 incorporated several important enhancements: 64-bit addresses and 64-bit

integer data, features to support optimizing compilers and high performance operating systems, superscalar implementation, fault tolerance, etc. As one of the first implementations of this new architecture, the UltraSPARC-I added a number of other features like a nine-stage pipeline allowing up to 4 instructions to be issued per cycle, dynamic branch prediction, on-chip 16K data and 16K instruction caches with up to 4 MB external cache, and on-chip graphics and imaging support. It was implemented using 0.5  $\mu\text{m}$ , 4-layer metal CMOS technology operating at 3.3 volts.

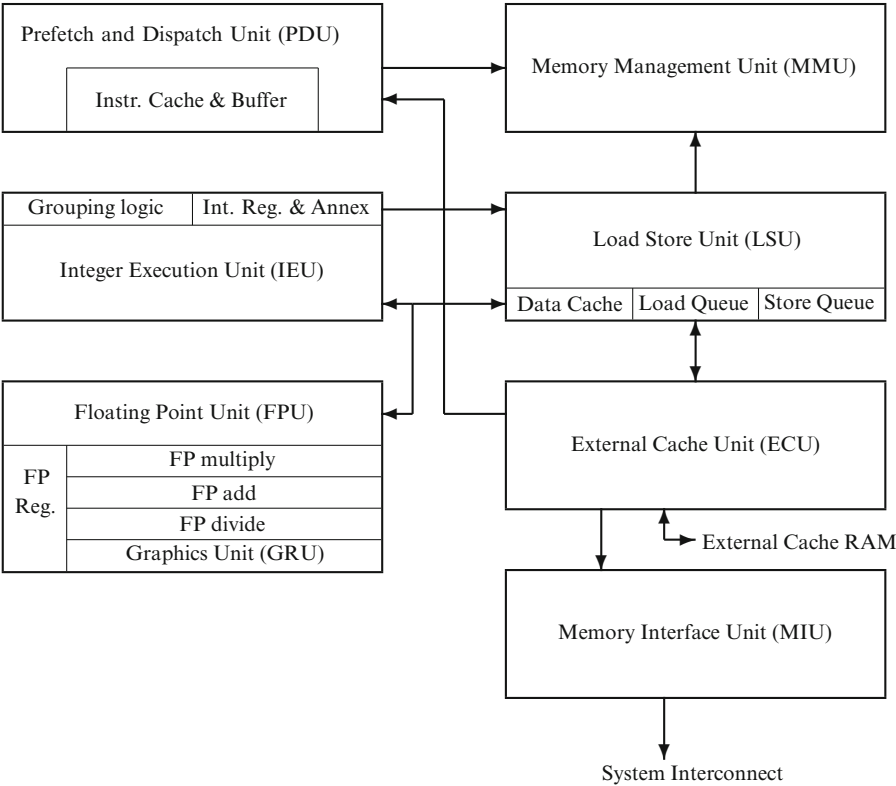


Figure 2.5 : UltraSPARC-II Block Diagram

UltraSPARC-II was the second generation member of the UltraSPARC family of processors [GBC<sup>+</sup>95, TO96]. In addition to using a new process technology, it provided a higher clock frequency, multiple SRAM modes and system to processor clock ratios. The architecture of UltraSPARC-II is shown in Figure 2.5. The Prefetch and Dispatch Unit (PDU) ensures that all execution units remain busy by

pre-fetching the instructions. Instructions can be prefetched from all levels of the memory hierarchy, including the I-cache, E-cache, and main memory. The machine used a double-instruction-issue pipeline with nine stages: fetch, decode, grouping, execution, cache access, load miss, integer pipe wait, trap resolution, and writeback. The latency (time from start to end of execution) of most instructions was nine clock cycles. However, at any given time, as many as nine instructions could be executed simultaneously, producing an overall rate of execution of one clock per instruction in many cases.

The branch prediction mechanism was based on a two-bit state machine that predicts branches based on the specific branch's most recent history. A two-bit prediction was maintained for every two instructions in the I-cache, allowing state information for up to 2,048 branches to be maintained. Branch following, the ability to rapidly fetch predicted branch targets, helped reduce fetch latency by providing a fuller instruction buffer, which in turn enhanced instruction execution rate.

The Integer Execution Unit (IEU) incorporated 2 ALU's for arithmetic, logical, and shift operations; an 8-window register file; result bypassing; and a Completion Unit. The Floating Point Unit (FPU) consisted of five separate functional units to support floating-point and multimedia operations. The Memory Management Unit (MMU) provided the functionality of a reference MMU and an IOMMU, handling all memory operations as well as arbitration between data stores and memory.

## **Intel Pentium Pro**

The Pentium Pro was a 32-bit Intel architecture microprocessor with dynamic execution features: out-of-order, speculative, superscalar, superpipelined, and micro-dataflow. It was implemented using a 0.6 $\mu$ m 4 layer metal BiCMOS process technology [Sch94, CS95]. High performance was achieved through the use of a large, full-speed cache accessed through a dedicated bus interface feeding a generalized dynamic execution microengine. A primary 64-bit processor bus included additional pipelining features to provide high throughput to the CPU and cache.

This processor implemented dynamic execution using an out-of-order, speculative execution engine, with register renaming of

integer [HP90], floating point and flag variables, multiprocessing bus support, and carefully controlled memory access reordering. The flow of Intel architecture instructions was predicted and these instructions were decoded into micro-operations ( $\mu$ ops), or series of  $\mu$ ops; the  $\mu$ ops were register-renamed, placed into an out-of-order speculative pool of pending operations, executed in dataflow order (when operands were ready), and retired to permanent machine state in source program order. This was accomplished with one general mechanism to handle unexpected asynchronous events such as mis-predicted branches, instruction faults and traps, and external interrupts. Dynamic execution—the combination of branch prediction, speculation and micro-dataflow—was the key to high performance.

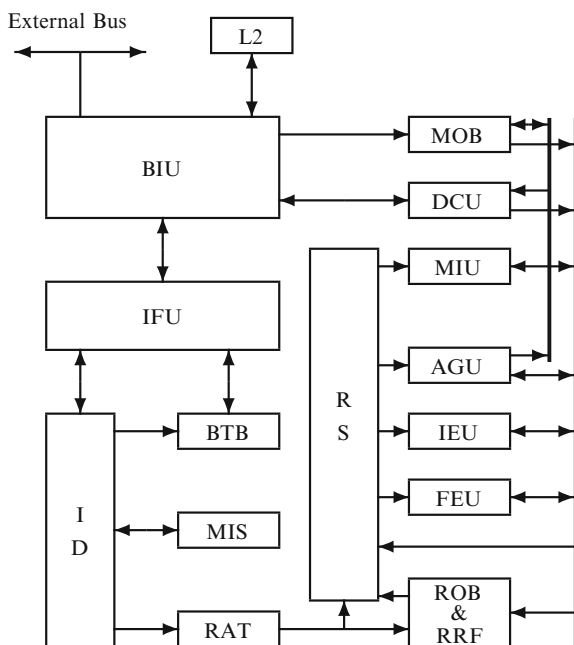


Figure 2.6 : Pentium Pro Basic CPU Block Diagram

The basic operation of the microarchitecture is described below (see Figure 2.6). The In-Order-Front-End consisted of the following parts: Instruction Fetch Unit (IFU), Branch Target Buffer (BTB), Instruction Decoder (ID), Micro-Instruction Sequencer (MIS), and Register Alias Table (RAT). The Reorder Buffer (ROB) & Real Register File (RRF) constituted the In-Order-Back-End, while the Out-Of-Order-Core consisted of the following parts: Reservation Stations (RS),

Memory Order Buffer (MOB), Data Cache Unit (DCU), Memory Interface Unit (MIU), Address Generation Unit (AGU), Integer Execution Unit (IEU) and Floating-point Execution Unit (FEU). The BIU is the Bus Interface Unit. The BTB had a non-trivial design using a 512 entry two-level adaptive algorithm. The IFU, which contained the I-cache, is where the Intel architecture instructions constituting the program lived. The BTB assisted the IFU in choosing an instruction cache line for the next instruction fetch. I-cache line fetches were pipelined with a new instruction line fetch commencing on every CPU clock cycle.

Three parallel instruction decoders (ID) converted multiple Intel Architecture instructions into multiple sets of micro-ops ( $\mu$ ops) each clock cycle. Three micro-ops were generated per clock, making the Pentium Pro processor a superscalar processor of degree 3. The sources and destinations of these  $\mu$ ops were renamed by the Register Alias Table (RAT), which eliminated register reuse artifacts, and forwarded to the Reservation Station (RS) and to the ReOrder Buffer (ROB). The renamed  $\mu$ ops were queued in the RS where they waited for their source data, which could come from several places, including immediates, data bypassed from just-executed  $\mu$ ops, data present in a ROB entry, and data residing in architectural registers (such as EAX). The queued  $\mu$ ops were dynamically executed according to their true data dependences and execution unit availability (IEU, FEU, AGU). The order in which  $\mu$ ops executed had no particular relationship to the order implied by the source program. Memory operations were dispatched from the RS to the Address Generation Unit (AGU) and to the Memory Ordering Buffer (MOB). The MOB ensured that memory access ordering rules were observed.

Once a  $\mu$ op had executed and its destination data was produced, that result datum was forwarded to subsequent  $\mu$ ops needing it, and the  $\mu$ op became a candidate for *retirement*, the act of taking the machine's speculative state and irrevocably committing it to permanent machine state. Retirement hardware in the ROB used  $\mu$ op timestamps to reimpose the original program order on the  $\mu$ ops as their results were committed to permanent machine state in the Retirement Register File (RRF). The retirement process observed not only the original program order, it also correctly handled interrupts and faults, and flushed all or part of its state on detection of a mispredicted branch. When a  $\mu$ op was retired, the ROB wrote that  $\mu$ op's result into the appropriate RRF entry and notified the RAT of that retirement so

that subsequent register renaming could be activated. The component included separate data and instruction L1 caches (each 8KB), and a unified 256KB non-blocking L2 cache. The L1 Data Cache was dual-ported and non-blocking, supporting one load and one store per cycle. The L2 cache interface ran at the full CPU clock speed, and could transfer 64 bits per cycle. The external bus was also 64-bit and could sustain a data transfer every bus-cycle. This external bus operated at 1/2, 1/3, or 1/4 of the CPU clock speed. Further discussion of the Intel Pentium Pro is available in [Pap96].

## Mips R10000

The Mips R10000 was a four-way superscalar RISC processor implementing the 64-bit Mips 4 instruction set architecture. It was implemented using 0.35- $\mu\text{m}$  CMOS process technology on a  $16.64 \times 17.934$  mm chip. The processor had five independent pipelined execution units (see Figure 2.7), including a non-blocking load/store unit, dual 64-bit integer ALU's, and 64-bit IEEE standard 754-1985 floating point units. In addition, there was a floating point adder and a multiplier,

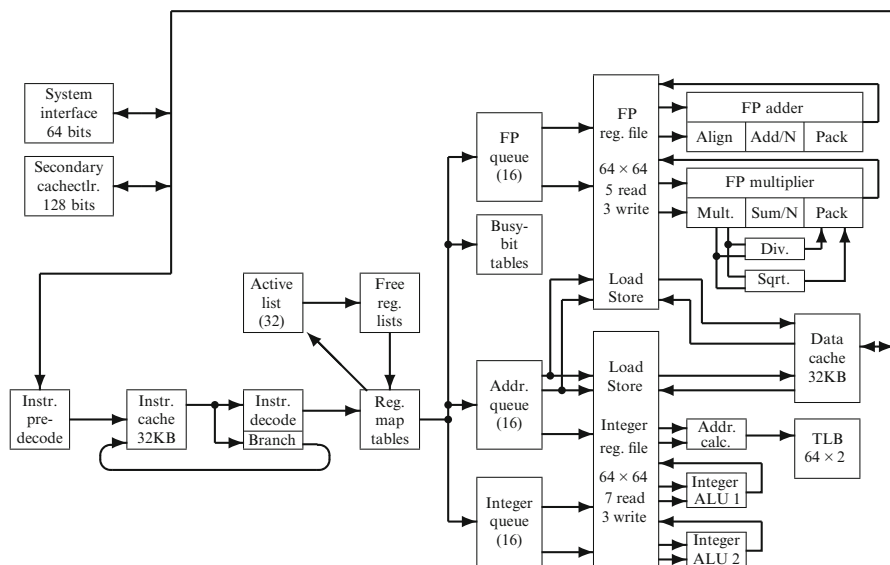


Figure 2.7 : Mips R10000 Block Diagram

each with two-cycle latency. The hierarchical, non-blocking memory subsystem included:

- two on-chip, two-way associative primary caches: a 32KB instruction cache and a 32KB two-way-interleaved data cache,
- an external two-way-associative secondary cache, a 128-bit wide synchronous static RAM, and
- a 64-bit multiprocessor system interface with split transaction protocol.

The R10000 fetched and decoded four instructions per cycle, and speculatively executed beyond branches with a four-entry branch stack. The R10000 implemented non-blocking caches to overlap cache refill operations (executed on a miss in the instruction cache) with other instructions that could continue out of order. The processor looked ahead up to 32 instructions to find parallelism. It predicted the direction of conditional branches and fetched instructions speculatively along the predicted path. The prediction was made using a 2-bit algorithm based on a 512-entry branch history table. In the Mips architecture, like the early RISC processors, a delay slot instruction, i.e., the instruction immediately following a branch or a jump, was executed while reading the target instruction from the cache. When a program took a jump or branch, the processor discarded the instructions fetched beyond the delay slot.

There were six independent pipelines: an instruction fetch pipeline and a five stage execution pipeline corresponding to each pipelined functional unit. The instruction fetch pipeline occupied stages 1–3 of the R10000 core pipeline. In stage 1, R10000 fetched and aligned the next four instructions. In stage 2, it decoded and renamed these instructions using map tables and computed the target addresses for the branch and jump instructions. In stage 3, it wrote the renamed instructions into instruction queues; the integer and floating-point instructions had separate queues. Instructions waited in these queues until their operands were ready. The processor read operands from the register files in the second half of stage 3 and issued the instruction for execution in stage 4. The result of an execution was either written into a register file or bypassed into operand registers; there

are no separate structures such as reservation stations or reorder buffers. A detailed description of the R10000 architecture is available in [Yea96].

## IBM/Motorola/Apple PowerPC

In February of 1990 IBM started the delivery of RS/6000, an implementation of the POWER Architecture [BGM90]. The POWER architecture embodied most of the RISC features: instructions were of fixed length (4 bytes), with consistent formats, and the architecture was load-store. The architecture provided a set of general purpose registers for fixed-point computation, including the computation of memory addresses; in addition, it provided a separate set of floating-point registers for floating-point computation. In October of 1993 IBM delivered the POWER2 processor [WD94] which had twice the number of execution units of the earlier POWER designs.

PowerPC Architecture was a 64-bit architecture, a superset of the POWER 32-bit architecture. This architecture extended addressing and fixed-point computation to 64 bits, and supported dynamic switching between the 64-bit mode and the 32-bit mode. The first PowerPC processor, the MPC601, was shipped in 1994; later, Motorola (now Freescale Semiconductor Inc.) and IBM unveiled the MPC604 processor with performance upto 100 MHz. In 2000, Freescale introduced the MPC74xx family of PowerPC processors (G4 family). The G4 core called the MPC7447A processor is a 32-bit implementation of the superscalar PowerPC RISC architecture, operates at 1.42 GHz and has on-chip power management features. The key architectural features include 512 KB of on-chip L2 cache, a 64-bit bus interface and a full 128-bit implementation of Freescale's AltiVec™ technology [Ful98]. The AltiVec technology features SIMD (Single Instruction, Multiple Data) functionality that supplements the host processor with an advanced 128-bit vector execution unit specifically designed to accelerate embedded application processing needs. This new engine facilitates simultaneous execution of up to 16 operations in a single clock cycle.

The block diagram of the MPC7447A PowerPC processor is shown in Figure 2.8. The processor has 11 independent execution units: four integer units (three simple plus one complex), a double-precision floating point unit, four AltiVec technology units (simple, complex,



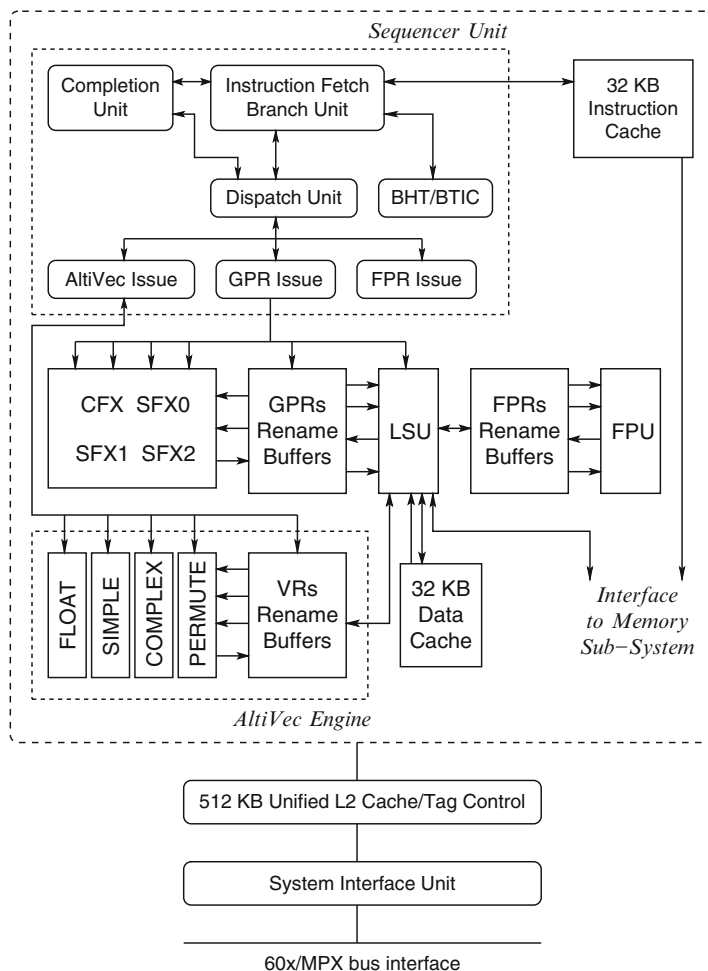


Figure 2.8 : The MPC7447A PowerPC processor

floating and permute), a load/store unit and a branch processing unit. The processor is capable of issuing four instructions per clock cycle (three instructions plus one branch). The microprocessor reference manual has further details [MPC05].

In June 2003, Apple introduced the superscalar PowerPC G5 which supported up to 215 in-flight instructions. It was implemented using 90 nm, silicon-on-insulator (SOI) technology. The processor had 12 functional units, two double-precision floating-point units, and a 128-bit Velocity Engine for image and video processing applications.

In addition, it had fast L1 and L2 caches and a dual-channel frontside bus at up to 1.25 GHz which provided high bandwidth to and from the rest of the system, allowing large numbers of tasks to run concurrently. Further, it supported group instruction dispatching, deep queues, and three-stage branch prediction logic. The success or failure of each prediction was captured in three large 16K branch history tables local, global, and selector that were used to improve the accuracy of future branch predictions. Also, the PowerPC G5 supported symmetric multiprocessing; the dual independent frontside buses allow each processor to handle its own tasks at maximum speed with minimal interruption.

## HP PA-8000

The PA-8000 RISC CPU is the first of a new generation of Hewlett-Packard microprocessors. It was implemented using HP's 0.5- $\mu\text{m}$ , 3.3-V CMOS process; the die measures 17.68 mm  $\times$  19.1 mm and contains 3.8 million transistors. It is a four-way superscalar machine that supports 64-bit computing and combines speculative execution with dynamic instruction reordering. The PA-8000 provides glue logic support for up to four-way multiprocessing via a Runway system bus, a 768-MB/s split-transaction bus that allows each processor to generate multiple outstanding memory requests. The PA-8000 implements PA (Precision Architecture) 2.0, a binary compatible extension of the previous PA-RISC architecture.

The functional block diagram of PA-8000 is shown in Figure 2.9. The functional units in the processor include two 64-bit integer ALU's, two 64-bit shift/merge units, dual floating-point multiply-and-accumulate (FMAC) units, and dual divide/square root units. Each FMAC unit is optimized to perform the operation  $A * B + C$ . The FMAC units are fully pipelined, while the divide/square root units are not. The peak throughput of the PA-8000 is four floating-point operations per cycle [DH96]. The processor incorporates two complete load/store pipes, including two address adders, a 96-entry dual-ported TLB, and a dual-ported cache. The dual load/store units and the memory system interface are shown on the right side of Figure 2.9.

The most important feature of the architecture is the 56-entry instruction reorder buffer, consisting of the ALU buffer that can store

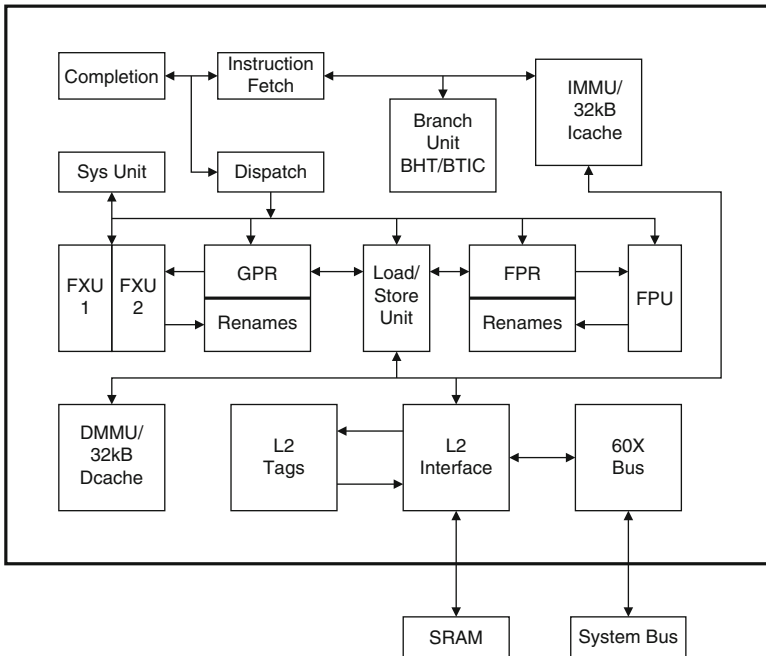


Figure 2.9 : Block Diagram of the HP PA-8000

28 computation instructions and the memory buffer that can store 28 load and store instructions. These buffers track the dependences between the instructions they hold and let any instructions in the window execute as soon as they are ready to fire. The reorder buffer also tracks branch prediction outcomes. The PA-8000 has a 32-entry branch target address cache (BTAC). The BTAC is fully associative; it associates the address of a branch instruction with the address of its target. The processor implements both static and dynamic modes of branch prediction. Each TLB entry contains a bit that indicates which mode should be used by the branch prediction hardware. The software can select the mode on a page-by-page basis. A detailed description of the HP PA-8000 architecture is available in [Kum97].

## 2.6 Itanium

Intel and Hewlett-Packard teamed up in the late 1990s to develop a new 64-bit processor design. The goal was to design an architecture with explicit support for parallel execution and its instruction

set. In addition, the designers wanted to handle branches efficiently. These goals eventually led the design of the *Explicitly Parallel Instruction Computing* (EPIC) architecture [SR00]. In the rest of the section, we first discuss the EPIC philosophy and then present a brief overview of the Itanium architecture.

### 2.6.1 The EPIC Philosophy

The EPIC architecture represented an evolution of VLIW that incorporated many superscalar concepts; it retains the VLIW philosophy of parallelizing the original sequential code at compile time, but augments it with hardware features—akin to those in superscalar processors—to better cope up with what actually happens at runtime. In comparison with superscalar architectures, EPIC provides an alternate style of architecture with support for high levels of ILP with reduced hardware complexity. Further, EPIC provides architectural features for compiler-driven control of mechanisms that the microarchitecture usually handles, such as cache hierarchy management and the associated decisions as to what data to promote up the hierarchy and what to replace.

EPIC inherits the architectural ideas pioneered at Multiflow (hardware support for control speculation of loads, parallel issue of multiple branch operations) and Cydrome (predicated execution, support for software pipelining in the form of rotating register files and special loop-closing branches). The key innovations in the EPIC architecture were:

- Inclusion of wired-OR and wired-AND predicate setting compares and the availability of two target compares.
- Branch support for software pipelining of while loops.
- Architectural support for data speculation.
- Support for data cache bypassing, specifically inclusion of the source specifier and the target cache specifier to deal with the data cache hierarchy.

Because most codes (especially non-floating point codes) have frequent conditional branches, any limitations in scheduling branch operations implies that branch operations will be a performance bottleneck in ILP. EPIC provides architectural features to better overlap

branch operations with other operations. Branch operations in EPIC are divided into three components: i) prepare-to-branch, which computes the branch's address, ii) a compare, which computes the branch condition, and iii) an actual branch, which specifies when control is transferred. Prescheduling of prepare-to-branch and compare operations enables speculative prefetching of instructions at the branch target. After executing the compare operation, the hardware dismisses the unnecessary speculative prefetches. These mechanisms allow parts of a branch computation to be scheduled in parallel with other operations without relying on the movement of the actual branching operation itself.

The ever widening processor-memory performance gap has been the driving force to use caches to hide memory latency. EPIC provides architectural features to facilitate compiler-driven channeling of data through the cache hierarchy, including load operations with a source cache specifier that the compiler uses to dictate to the hardware where within the cache hierarchy it can expect to find the data. The prediction can be done based on analytical or cache miss profiling techniques. In a similar fashion, the compiler uses the target cache specifier to indicate the cache level to which the load or store should promote or demote the data. The target cache specifier helps reduce cache misses in the first- and second-level caches by controlling their contents. The compiler can exclude data with low temporal locality from the first- or second-level cache, and can remove data from the cache level on the last use. Further, the data prefetch cache allows the prefetching of data with poor temporal locality into a low-latency cache without displacing the first-level cache's contents.

Static aliasing among memory references poses another impediment to exploiting parallelism. In general, a compiler must be conservative, i.e., it assumes aliasing between two memory references unless it can prove that they cannot alias. In order to alleviate this limitation, EPIC provides support for data speculation, whereby a conventional load operation is broken down into two components: data-speculative load and data-verifying load. The compiler moves a data-speculative load above potentially aliasing stores to hide the memory access latency. It schedules the data-verifying load after the potentially aliasing stores and uses hardware to detect whether the memory locations are in fact the same (aliased). In the absence of aliasing, execution proceeds as usual; otherwise, the processor is stalled and

the load operation is re-executed. Further, EPIC provides support for aggressive data-speculative code motion, wherein operations that use the result of data-speculative loads are also moved above potentially aliasing stores. Next, we discuss the Itanium architecture, an example of the EPIC philosophy.

### 2.6.2 Itanium Architecture

The Intel Itanium processor, a 64-bit architecture, was the first implementation of the family of processors based on the IA-64 architecture. The processor employs EPIC design concepts (discussed in the previous subsection) for exploiting higher levels of ILP; it goes beyond RISC and CISC approaches by coupling hardware with compiler technology to make parallelism explicit to the processor. The hardware provides abundant execution resources and supports various dynamic run-time optimizations for high performance.

The (first generation) Itanium processor provides a 6 instruction-wide and 10-stage deep pipeline, running at 800 MHz on a 0.18- $\mu$ m process. This provides both large hardware support to exploit ILP as well as high frequency for minimizing the latency of each instruction. The processor provides the following execution units: 4 integer units, 2 floating point units, 4 multimedia units, 2 load/store units and 3 branch units. Performance beyond compile-time parallelization is achieved via dynamic prefetch, branch prediction, a register scoreboard and non-blocking three-level caches. Further, Itanium supports an SIMD instruction set architecture (ISA), facilitating exploitation of data parallelism; the SIMD ISA is compatible with Intel's MMX technology. Similarly, it supports floating point SIMD operations and is compatible with the Intel's Streaming SIMD extensions (SSE). The basic block diagram of the microarchitecture is shown in Figure 2.10.

Itanium provides a large number of both integer and floating point registers: 128 64-bit integer registers, 128 82-bit floating point registers, 64 predicate registers and 8 branch registers for speculative execution. Registers are renamed to facilitate function calls via variable-size windows; the registers are rotated in the context of software pipelining of loops. Further, Itanium provides hardware support for floating point MAC operations, maximum and minimum operations, and fully pipelined divide and sqrt primitives. It also supports an IA32 hardware execution unit for backward compatibility with

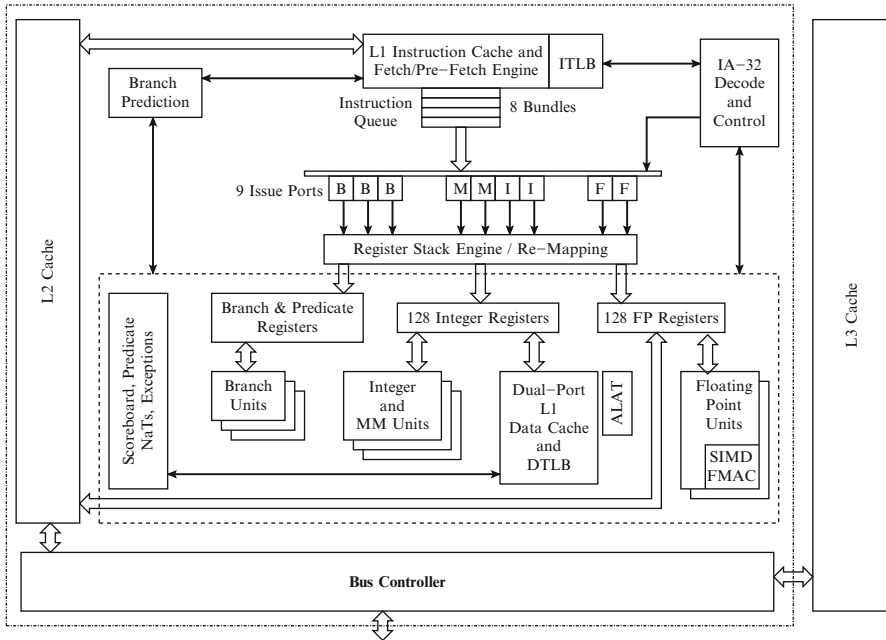


Figure 2.10 : Itanium Processor Block Diagram

traditional Intel x86 instructions. In addition, Itanium provides hardware support, called ALAT (Advanced Load Address Table), for data speculation; ALAT contains information about outstanding advanced loads. The register stack engine spills/fills registers as needed and works upon stack overflow/underflow; it ensures efficient stack register usage and provides a programming model that provides a register stack with a logically infinite number of registers.

The Itanium processor has three levels of processor cache. The instruction and data L1 caches are 16KB in size, 4-way set associative with 32B line size. The L1 data cache is dual-ported; thus, it can support two concurrent loads/stores. The L1 data cache only caches data for the integer unit, not for the floating-point units. The unified L2 cache is 96KB in size, 6-way set associative with a 64B line size and uses a write-back with a write-allocate policy. The L3 cache is either 2MB or 4MB in size, 4-way set associative with 64B line size. The L3 cache is accessed via a high bandwidth 128-bit back-side bus allowing memory accesses at processor core speed.

Itanium executes instructions in parallel in what are known as *instruction bundles*, comprised of 3 instructions each. The processor attempts to schedule two instruction bundles in parallel per clock, allowing the processor to schedule and execute a maximum of 6 instructions in a single clock cycle. Each 128-bit sized bundle contains 41-bit instruction slots and a 5-bit template field, as shown in Figure 2.11.

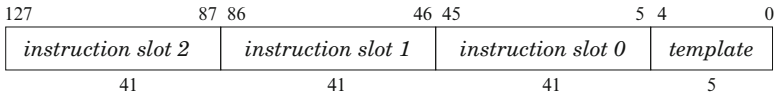


Figure 2.11 : Bundle Format in Itanium

The core pipeline of Itanium consists of 10 stages, as shown in Figure 2.12. First, the IPG stage generates the instruction pointers. Then the FET stage fetches instructions by accessing L1 I-Cache and L1-ITLB. The ROT stage formats the instruction stream and loads the instruction buffer to be used by the rest of the pipeline.



Figure 2.12 : 10-stage Itanium pipeline

The EXP stage expands the instruction templates and issues the instructions. The REN stage handles register renaming for register stack and register rotation. The WLD (word-line decode) stage decodes the instructions and the REG stage delivers data to the functional units either from the registers or by bypassing the data generated by functional units for consumption by chained instructions. This stage also generates spill or fill instructions required by the register stack engine. The EXE stage delivers instructions and the data to the functional units. The DET stage detects exceptions and branch mispredictions. The pipeline is flushed by exceptions and branch mispredictions, causing highest priority pipeline stalls. Stalls in data delivery and floating point micropipelines are also detected at this stage. Finally, the WRB stage writes the output to the appropriate output registers.

The Itanium architecture provides explicit support for software pipelining, discussed in Chapter 7. The pipeline stages are defined



using full predication and other special branch handling features specific to software pipelining (e.g., loop branches and loop registers LC, EC). Register rotation is used to alleviate loop copy overhead; similarly, predicate rotation is used to remove the overhead associated with the prologue and epilogue. Further, the hardware supports multi-way branching, a key requirement for software pipelining of loops with conditionals.

## FURTHER READING

The history of parallelism is very well discussed in the book by Hockney and Jesshope [HJ81]. A detailed discussion on the evolution of computer architecture is available in [Bae80, SBN82, Smi88]. As discussed earlier, the development of CDC 6600 and IBM 360 mark the onset of ILP processors; further discussion on the CDC 6600 and the IBM 360 family is available in [AWZ64, CDF64] and [Amd64, AB]64] respectively. The state of the art for minis in the mid-seventies is well covered in [Kou73]. The 4- and 8-bit microprocessors are well-covered in [HJ76].

An early empirical evaluation of some features of instruction set processor architectures is presented in [Lun75, Lun77]. The RISC concept was first introduced in [PS81, PS82], which was later enhanced in [KSPS83]. A comprehensive discussion of the architecture and the pipeline of RISC 1 and RISC II is available in [Kat85]. One of the early RISC architectures, SOAR, is discussed in [UBF<sup>+</sup>84]. A discussion of RISC principles, with good explanation of instruction pipelines and how RISC can take advantage of them is present in [Pat85].

Overlapped execution in microprogramming was discussed in [Hig78]; an excellent discussion of overlap and pipelining can be found in [CK75, RL77]. Support for overlapped loop execution is discussed in [DHB89]. A detailed description of the IBM RISC System/6000 is discussed in [BGM90]. The book by Stone [Sto93] is a good reference for computational scientists, containing detailed explanations of pipelining and memory organization, chapters on scientific applications, vector machines, and parallel processing. The book by Trew [TW91] presents a good survey of parallel computer systems.

Look-ahead processors, the forerunners of today's superscalar processors built in the 1960s, performed out-of-order execution while issuing multiple operations per cycle. A survey of the early look-ahead processors is available in [Kel75]. An excellent reference on superscalar processor design and its complexity is the book by Johnson [Joh91]. On the other side of the spectrum, the earliest VLIW processors built were the so-called attached array processors [RC69, IBM76, FSP79, INT89]. Some of the popular products were the Floating Point Systems AP-120B, the FPS-164 and the FPS-264. A very good reference for processor arrays is the book by Hockney and Jesshope [HJ88]; the book is also a good resource for computational scientists, with a nice history of high performance computing and a comprehensive survey of parallel algorithms for important matrix operations in addition to parallel and vector computer architecture. The book by Hwang [Hwa93] is a good reference for facts

about machines such as, but not limited to, CM-5, KSR-1, and Paragon X/PS. The reader is advised to refer to [FO84, Fis87, Fis90, BYA93] for further discussion of the early VLIW architectures. The performance of Intel i860 microprocessor is discussed in [Atk91]. A detailed description of the Warp architecture is available in [GO98]. A VLIW approach towards embedded computing is discussed in [FFY04].

Instruction Level Parallelism

Aiken, A.; Banerjee, U.; Kejariwal, A.; Nicolau, A.

2016, XXI, 255 p. 78 illus., 30 illus. in color., Hardcover

ISBN: 978-1-4899-7795-3