ACA_NOTES

(15CS72)

MODULE-2

Hardware Technologies: Processors and Memory Hierarchy, Advanced ProcessorTechnology, Superscalar and Vector Processors, Memory Hierarchy Technology, Virtual Memory Technology.

PROCESSORS AND MEMORY HIERARCHY

4.1 ADVANCED PROCESSORTECHNOLOGY

Major processor families include the CISC, RISC, superscalar, VLIW, superpipeline4 vector, and symbolic processors. Scalar and vector processors are for numerical computations. Symbolic processors Live been developed for Al applications.

4.1.1 Design Space of Processors

Various processor families can be mapped onto a coordinated space of clock rate verius cycles per insenicifon CPI) as illustrated in Figure 4.1 below, the broad CPI versus clock speed characteristics of major categories of current processors. The two broad categories are CISC and RISC. In CISC category, at present there is the only one dominant proience, the x86 processor architecture and in the RISC category, there are several examples, like Power series, SPARC, MIPS, etc.

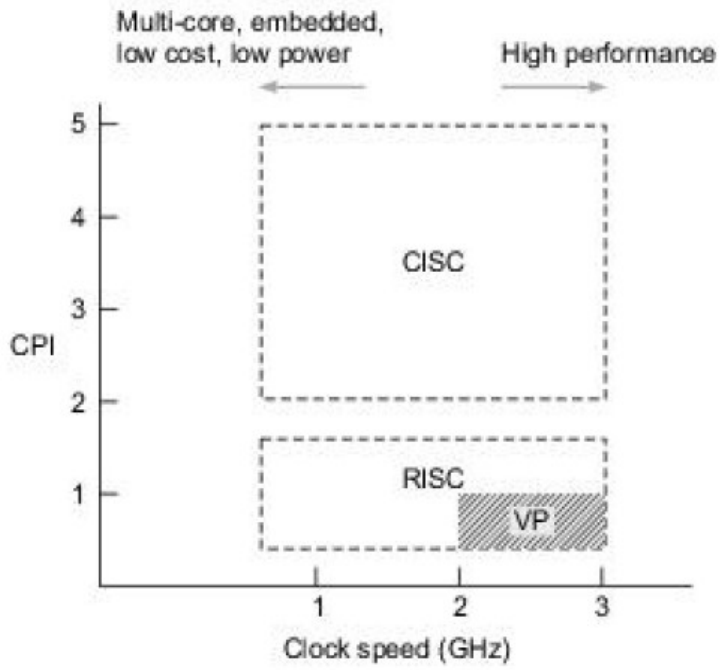


Fig. 4.1 CPI versus processor clock speed of major categories of processors

Under both CISC and RISC categories, products designed for multi-core chips, embedded applications, or for low cost andfor low power consumption, tend to have lower clock speeds. High performance processors must necessarily be designed to operate at high clock speeds. The category of vector processor has been marked VP, vector processing features may be associated with CISC or RISC main processors.

The Design Space

- Conventional processors like the Intel Pentium, M68040, IBM 390, etc are complex-instruction-set computer (CISC) architectures. The clock rate of today's CISC processors ranges up to a few Ghz. The CPI of different CISC instructions varies from 1 to 20. Therefore, CISC processors are at the upper part of the design space.
- Reduced-instruction-set computing (RISC) processors include SPARC, Alpha, ARM, etc. With the use of efficient pipelines, the average CPI of RISC instructions Lies been reduced to between one and two cycles.
- An important subclass of RISC processors are the superscalar processors, which allow multiple instructions to be issued simultaneously during each cycle.
- The very long instruction word (VLIW) architecture can in theory use even more functional units than a superscalar processor. Intel's i860 RISC processor had VLIW architecture.
- The processors in vector supercomputers use multiple functional units for concurrent scalar and vector operations. The effective CPI of a processor used in a supercomputer should be very low, positioned at the lower right corner of the design space. However, the cost and power consumption increase appreciably if processor design is restricted to the lower right corner.

Instruction Pipelines

The execution cycle of a typical instruction includes four phases: fetch, decodes execute, and wrife-back. These instruction phases are often executed by an insfruciion pipeline as demonstrated in below Fig. 4.2a. The pipeline, like an industrial assembly line, receives successive instructions from its input erd and executes them in a streamlined, overlapped fashion as they flow through.

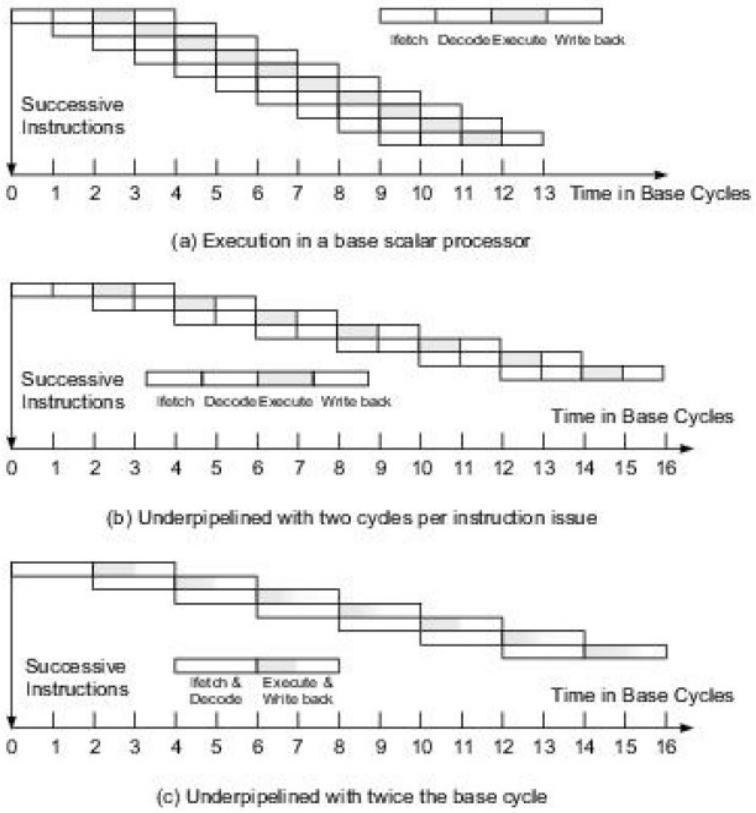


Fig. 4.2 Pipelined execution of successive instructions in a base scalar processor and in two underpipelined cases (Courtesy of Jouppi and Wall; reprinted from *Proc. ASPLOS*, ACM Press, 1989)

A pipeline cycle is intuitively defined as the time required for each phase to complete its operation, assuming equal delay in all phases (pipeline stages).

Instruction pipeline cycle is the clock period of the instruction pipeline.

Instruction issue latency is the time (in cycles) required between the issuing of two adjacent instructions.

Instruction issue rate is the number of instructions issued per cycle, is also called the degree of a superscalar processor.

Simple operation latency Simple operations make up the vast majority of' instructions executed by the machine, such as integer adds, loads, stores, branches, moves, etc. Complex operations are those requiring an order-of-magnitude longer latency, such as divides. cache misses, etc. These latencies are measured in number of cycles.

Resource conflicts This refers to the situation where two or more instructions demand use of the same functional unit at the same time.

A base scalar processor is defined as a machine with one instruction issued per cycle, a one-cycle latency for a simple operation, and a one-cycle latency between instruction issues. The instruction to pipeline can be fully utilized if successive innstructions can enter it continuously at the rate of one per cycle, as shown in Fig. 4.2a.

If the instruction issue latency is two cycles per instruction, the pipeline can be underutilized, as demonstrated in Fig. 4.2b. Another underpipelined situation is shown in Fig. 4.2c, in which the pipeline cycle time is doubled by combining pipeline stages. In this case, the fetch and decode phases are combined into one pipeline stage, and execute and write back are combined into another stage. This will also result in poor pipeline utilization. The effective CPI rating is 1 for the ideal pipeline in Fig. 4.2a, and 2 for the case in Fig.4.2b.

Figure 4.3 below shows the data path architecture and control unit of a typical, simple scalar processor which does not employ an instruction pipeline. Main memory, I/O controllers, etc. are connected to the external bus. The control unit generates control signals required for the fetch, decode, ALU operation, memory access, and write result phases of

instruction execution. The control unit itself may employ hardwired logic or microcoded logic. Modem RISC processors employ hardwired logic.

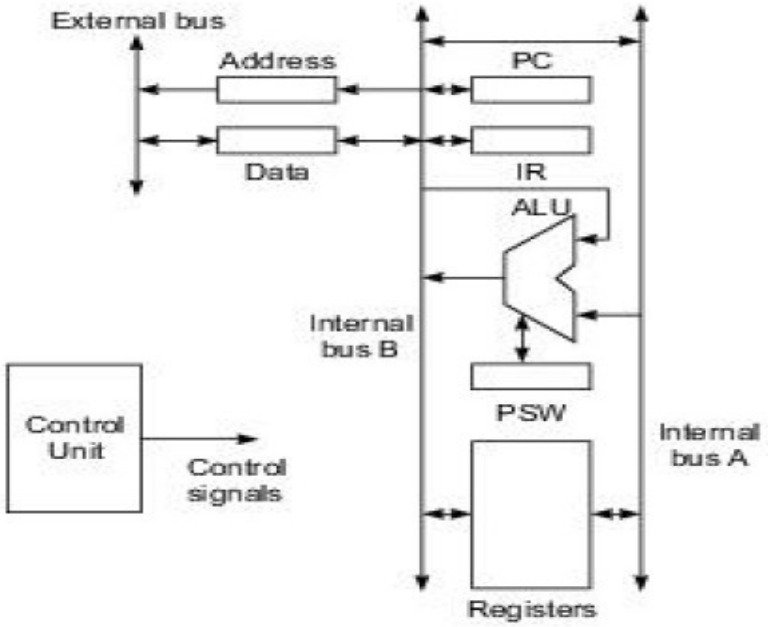


Fig. 4.3 Data path architecture and control unit of a scalar processor

4.1.2 Instruction-Set Architectures

The instruction set of a computer specifics the primitive commands or machine instructions that a programmer can use in programming the machine. The complexity of an instruction set is attributed to the instruction formats, data formats. addressing modes, general-purpose registers, opoode specifications, and flow control mechanisms used.

The major instruction set architectures are CISC and RISC.

A typical CISC instruction set contains approximately 120 to 350 instructions using variable instruction/ data formats, uses a small set of to 24 general-purpose registers (GPRO), and executes a large number of memory reference operations based on more than a dozen addressing modes. Many HLL statements are directly implemented in hardware/firmware in a CISC architecture.

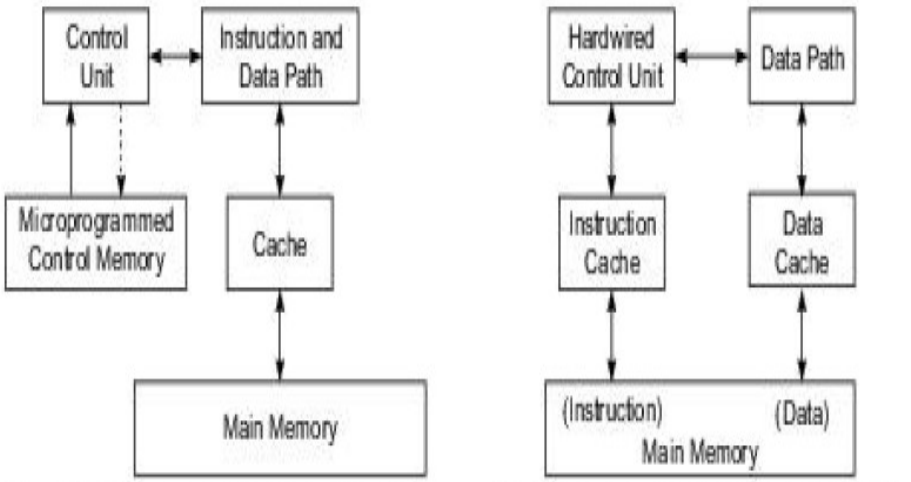
Why should we waste valuable chip area for rarely used instructions?

With low-frequency elaborate instructions demanding long microcodes to execute them, it might be more advantageous to remove them completely from the hardware and rely on software to implement them. Even if the software implementation was slow, the net result would be still a plus due to their low frequency of appearance. Pushing rarely used instructions into software would vacate chip areas for building more powerful RISC or superscalar processors, even with on-chip caches or floating-point units, and hardwired control would allow faster clock rates.

A RISC instruction set typically contains less than 100 instructions with a fixed instruction format (32 bits), Only three to five simple addressing modes are used. Most instructions are register-hased. Memory access is done by load/store instructions only A large register file (at least 32) is used to improve fast context .switching among multiple users, and most insructions execute in one cycle with hardwired control. The resulting benifits include a higher clock rate and a lower CPI. which lead to higher processor performance.

Architectural Distinctions

Hardware features built into CISC and RISC processors arc compared below. Figure 4.4 shows the architectural distinctions between traditional CISC and RISC. Some of the distinctions have since disappeared, however, because processors are now designed with features from both types. Conventional CISC architecture uses a unified cache for holding both instructions and data Therefore, they must share the same data/instruction path. In a RISC processor, separate instruction and data caches are used with different access paths. The use of micro programmed control was found in traditional CISC, and hardwired control in most RISC. Modern CISC also uses hardwired control and split caches such as the MC68040.



(a) The CISC architecture with microprogrammed control and unified cache

(b) The RISC architecture with hardwired control and split instruction cache and data cache

Fig. 4.4 Distinctions between typical RISC and typical CISC processor architectures (Courtesy of Gordon Bell, 1989)

Table 4.1 *Characteristics of Typical CISC and RISC Architectures*

Architectural Characteristic	*Complex Instruction Set Computer (CISC)*	*Reduced Instruction Set Computer (RISC)*
Instruction-set size and instruction formats	Large set of instructions with variable formats (16–64 bits per instruction).	Small set of instructions with fixed (32-bit) format and most register-based instructions.
Addressing modes	12–24.	Limited to 3–5.
General-purpose registers and cache design	8–24 GPRs, originally with a unified cache for instructions and data, recent designs also use split caches.	Large numbers (32–192) of GPRs with mostly split data cache and instruction cache.
CPI	CPI between 2 and 15.	One cycle for almost all instructions and an average CPI < 1.5.
CPU Control	Earlier microcoded using control memory (ROM), but modern CISC also uses hardwired control.	Hardwired without control memory.

4.1.3 CISC Scalar Processors

A scalar processor executes with scalar data. The simplest scalar processor executes integer instructions susing fixed-point operands. More capable scalar processors execute both integer and floating-point operations. A modern scalar processor may possess both an integer unit and a floating-point unit. based on a complex instruction set a scalar processor can also use pipelnied design.

The three early representative CISC scalar processors are listed below. The VAX 8600 processor was built on a PC board. The i486 and M68040 were single-chip microprocessors. In any processor design, the designer attempts to achieve higher throughput in the processor pipelines.

Table 4.2 *Representative CISC Scalar Processors of year 1990*

Feature	*Intel i486*	*Motorola MC68040*	*NS 32532*
Instruction-set size and word length	157 instructions, 32 bits.	113 instructions, 32 bits.	63 instructions, 32 bits.
Addressing modes	12	18	9
Integer unit and GPRs	32-bit ALU with 8 registers.	32-bit ALU with 16 registers.	32-bit ALU with 8 registers.
On-chip cache(s) and MMUs	8-KB unified cache for both code and data. with separate MMUs.	4-KB code cache 4-KB data cache	512-B code cache 1-KB data cache.
Floating-point unit, registers, and function units	On-chip with 8 FP registers adder, multiplier, shifter.	On-chip with 3 pipeline stages, 8 80-bit FP registers.	Off-chip FPU NS 32381, or WTL 3164.
Pipeline stages	5	6	4
Protection levels	4	2	2
Memory organization and TLB/ATC entries	Segmented paging with 4 KB/page and 32 entries in TLB.	Paging with 4 or 8 KB/page, 64 entries in each ATC.	Paging with 4 KB/page, 64 entries.
Technology, clock rate, packaging, and year introduced	CHMOS IV, 25 MHz, 33 MHz, 1.2M transistors, 168 pins, 1989.	0.8–μm HCMOS, 1.2 M transistors, 20 MHz, 40 MHz, 179 pins, 1990.	1.25–μm CMOS 370K transistors, 30 MHz, 175 pins, 1987.
Claimed performance	24 MIPS at 25 MHz,	20 MIPS at 25 MHz, 30 MIPS at 60 MHz.	15 MIPS at 30 MHz.

Example 4.1 The Digital Equipment VAX 8600 processor architecture

The VAX 8600 was introduced by Digital Equipment Corporation in 1985. This machine implemented a typical CISC arch itecture with microprogrammed control. The instruction set contained about 300 instructions with 20 different addressing modes.GPRs in the instruction unit. Instruction pipelining was built with six stages in the VAX 8600, as in most else machines. The instruction unit prefetched and decoded instructions, handled branching operations, and supplied operands to the two functional units in a pipelined fashion.

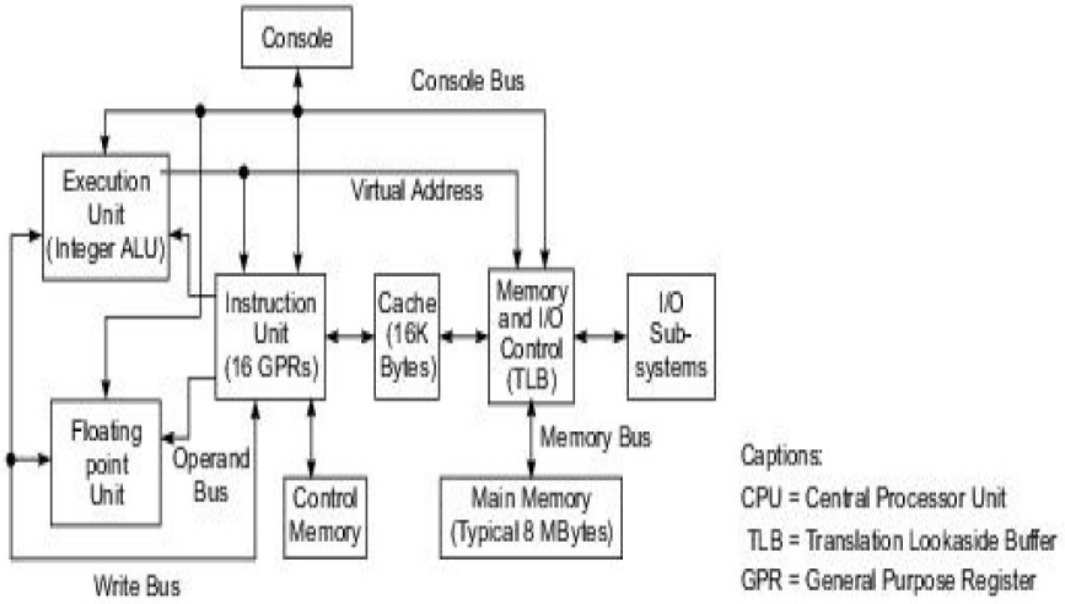


Fig. 4.5 The VAX 8600 CPU, a typical CISC processor architecture (Courtesy of Digital Equipment Corporation, 1985)

A translation look lookaside buffer was used in the memory control unit for fast generation of physical address from a virtual address. Both integer and floating-point units were pipelined. The performance of the processor pipelines relied heavily on the cache hit ratio and on minimal branching damage to the pipeline flow. The general philosophy of designing a CISC processor is to implement useful instructions in hardware/ firmware which may result in a shorter program length with a lower software overhead.

CISC Microprocessor Families

In 1971 the Intel 4004 appeared as the first microprocessor based on a 4-bit ALU. Since then, Intel has produced the 8-bit 8008, 8080, and 8085. Intel's 16-bit processors appeared in I973 as the 8086, 8088, 80186, and 80286. In 1985, the 80386 appeared as a 32-bit machine. The 80486 and Pentium are the latest 32-bit processors in the lntel 80x86 family.

Motorola produced its first 8-bit microprocessor, the MC6800 in 1974, then moved to the 16-bit 68000 in 1979, and then to the 32-bit 68020 in 1984. Then came the MC68030 and MC68040.

National Semiconductor's 32-bit microprocessor NS32532 was introduced in I988. These CISC microprocessor families have been widely used in the personal computer (PC) industry.

Table 4.2 *Representative CISC Scalar Processors of year 1990*

Feature	*Intel i486*	*Motorola MC68040*	*NS 32532*
Instruction-set size and word length	157 instructions, 32 bits.	113 instructions, 32 bits.	63 instructions, 32 bits.
Addressing modes	12	18	9
Integer unit and GPRs	32-bit ALU with 8 registers.	32-bit ALU with 16 registers.	32-bit ALU with 8 registers.
On-chip cache(s) and MMUs	8-KB unified cache for both code and data. with separate MMUs.	4-KB code cache 4-KB data cache	512-B code cache 1-KB data cache.
Floating-point unit, registers, and function units	On-chip with 8 FP registers adder, multiplier, shifter.	On-chip with 3 pipeline stages, 8 80-bit FP registers.	Off-chip FPU NS 32381, or WTL 3164.
Pipeline stages	5	6	4
Protection levels	4	2	2
Memory organization and TLB/ATC entries	Segmented paging with 4 KB/page and 32 entries in TLB.	Paging with 4 or 8 KB/page, 64 entries in each ATC.	Paging with 4 KB/page, 64 entries.
Technology, clock rate, packaging, and year introduced	CHMOS IV, 25 MHz, 33 MHz, 1.2M transistors, 168 pins, 1989.	0.8–μm HCMOS, 1.2 M transistors, 20 MHz, 40 MHz, 179 pins, 1990.	1.25–μm CMOS 370K transistors, 30 MHz, 175 pins, 1987.
Claimed performance	24 MIPS at 25 MHz,	20 MIPS at 25 MHz, 30 MIPS at 60 MHz.	15 MIPS at 30 MHz.

Example 4.2 The Motorola MC68040 microprocessor architecture.

The below Figure 4.6 shows the MC68040 architecture. The processor implements over 100 instructions using 16 general-purpose registers, a 4-Kbyte data cache, and a 4-Kbyte instruction cache, with separate Memory management units(MMU) supported by an address translation cache (ATC). The data formats range from 8 to 80 bits and has eighteen addressing modes. The instruction set includes data movement, integer, and floating point arithmetic and so on. The integer unit is organized in a six-stage instruction pipeline. The floating-point unit consists of three pipeline stages. All instructions are decoded by the integer unit. Floating-point instructions are forwarded to the floating-point unit for execution. Separate instruction and data buses are used to and from the instruction and data memory units, respectively. Both the address bus and the data bus are 32 bits wide. Three simultaneous memory requests can he generated by the dual MMUs, including data operand read and write and instruction pipeline refill. Snooping logic is built into the memory units for monitoring bus events for cache invalidation. The complete memory management is provided with support for virtual demand paged operating system.

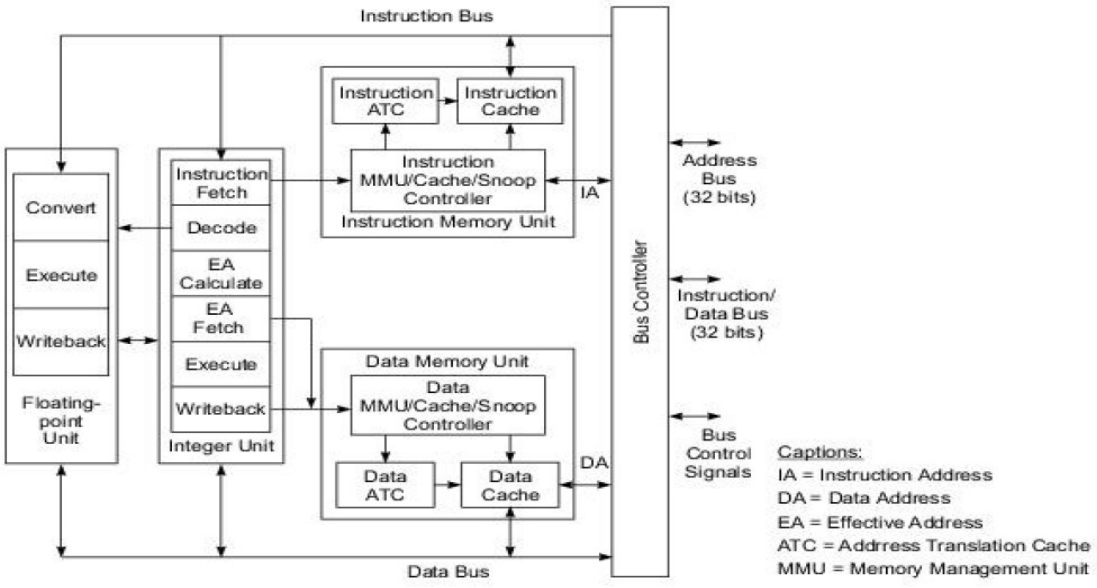


Fig. 4.6 Architecture of the MC68040 processor (Courtesy of Motorola Inc., 1991)

4.1.4 RISC Scalar Processors

- Generic RISC processors are called scalar RISC because they are designed to issue one instruction per cycle. In theory, both RISC and CISC scalar processors should perform about the same if they run with the same clock rate and with equal program length.
- The RISC design gains its power by pushing some of the less frequently used operations into software.The reliance on a good compiler is much more demanding in a RISC processor than in a CISC processor.
- Instruction-level parallelism is exploited by pipelining in both processor architectures. Without a high clock rate, a low CPI, and good compilation support, neither RISC nor CISC can perform well as designed. The simplicity introduced with a RISC processor may lead to the ideal performance of the base scalar machines.

Representative RISC Processors

Four representative RISC-based processors from the year 1990, the Sun SPARC, lntel i860, Motorola M88100, and AMD 29000, are summarized in below Table 4.3. On-chip floating-point units are built into the i860 and M88100, while the SPARC and AMD use off-chip floatingpoint units. We consider these four proccssors as generic scalar RISC, issuing essentially only one instruction per pipeline cycle. SPARC stands for scalable processor architecture. The scalability of the SPARC architecture refers to the use of a different number of register windows in different SPARC implementations.

Table 4.3 *Representative RISC Scalar Processors of year 1990*

Feature	*Sun SPARC CY7C601*	*Intel i860*	*Motorola M 88100*	*AMD 29000*
Instruction set, formats, addressing modes.	69 instructions, 32-bit format, 7 data types, 4-stage instr. pipeline.	82 instructions, 32-bit format, 4 addressing modes.	51 instructions, 7 data types, 3 instr. formats, 4 addressing modes.	112 instructions, 32-bit format, all registers indirect addressing.
Integer unit, GPRs.	32-bit RISC/IU, 136 registers divided into 8 windows.	32-bit RISC core, 32 registers.	32-bit IU with 32 GPRs and scoreboarding.	32-bit IU with 192 registers without windows.
Caches(s), MMU, and memory organization.	Off-chip cacbe/MMU on CY7C604 with 64-entry TLB.	4-KB code, 8-KB data, on-chip' MMU, paging with 4 KB/page.	Off-chip M88200 caches/MMUs, segmented paging, 16-KB cache.	On-chip MMU with 32-entry TLB, with 4-word prefetch buffer and 512-B branch target cache.
Floating-point unit registers and functions	Off-chip FPU on CY7C602, 32 registers, 64-bit pipeline (equiv. to TI8848).	On-chip 64-bit FP multiplier and FP adder with 32 FP registers, 3-D graphics unit.	On-chip FPU adder, multiplier with 32 FP registers and 64-bit arithmetic.	Off-chip FPU on AMD 29027, on-chip FPU withAMD 29050.
Operation modes	Concurrent IU and FPU operations.	Allow dual instructions and dual FP operations.	Concurrent IU, FPU and memory access with delayed branch.	4-stage pipeline processor.
Technology, clock rate, packaging, and year	0.8-μm CMOS IV,33 MHz, 207 pins, 1989.	1-μm CHMOS IV, over 1M transistors, 40 MHz, 168 pins, 1989	1-μm HCMOS, 1.2M transistors, 20 MHz, 180 pins, 1988.	1.2-μm CMOS, 30 MHz, 40 MHz, 169 pins, 1988.
Claimed performance	24 MIPS for 33 MHz version, 50 MIPS for 80 MHz ECL version. Up to 32 register windows can be built.	40 MIPS and 60 Mflops for 40 MHz, i860/XP announced in 1992 with 2.5M transistors.	17 MIPS and 6 Mflops at 20 MHz, up to 7 special function units could be configured.	27 MIPS at 40 MHz, new version AMD 29050 at 55 MHz in 1990.

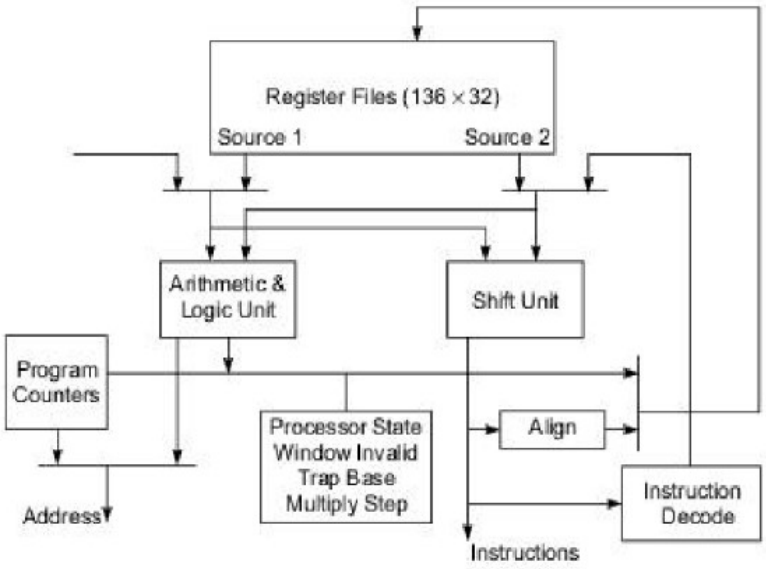
Example 4.3 The Sun Microsystems SPARC architecture

The SPARC has been implemented by a number of licensed manufacturers as summarized in Table 4.4. Difierent technologies and window numbers are used by difierent SPARC manufacutures.

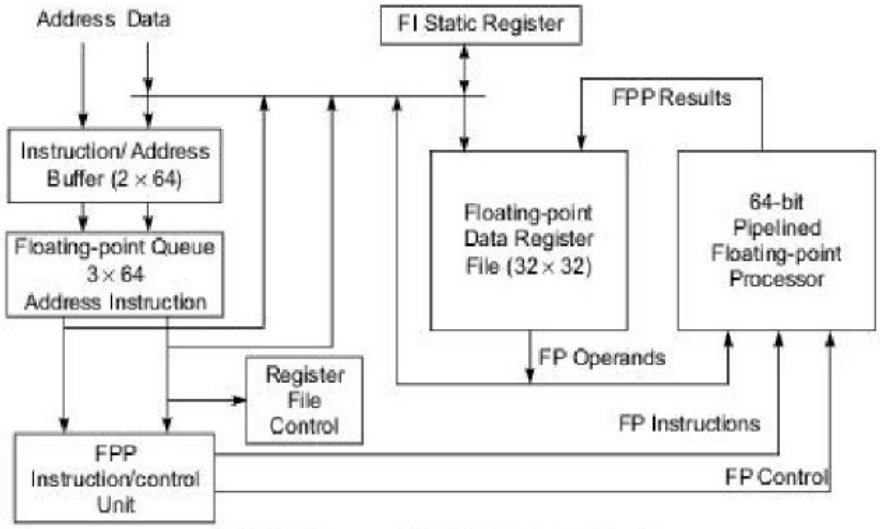
Table 4.4 *SPARC Implementations by Licensed Manufacturers (1990)*

SPARC Chip	*Technology*	*Clock Rate (MHz)*	*Claimed VAX MIPS*	*Remarks*
Cypress CY7C601 IU	0.8μm CMOS IV, 207 pins.	33	24	CY7C602 FPU with 4.5 Mflops DP Linpack, CY7C604 Cache/MMC, CY7C157 Cache.
Fujitsu MB 86901 IU	1.2-μm CMOS, 179 pins.	25	15	MB 86911 FPC FPC and TI 8847 FPP, MB86920 MMU, 2.7 Mflops DP Linpack by FPU.
LSI Logic L64811	1.0-μm HCMOS, 179 pins.	33	20	L64814 FPU, L64815 MMU.
TI 8846	0.8-μm CMOS	33	24	42 Mflops DP Linpack on TI-8847 FPP.
BIT IU B-3100	ECL family.	80	50	15 Mflops DP Linpack on FPUs: B-3120 ALU, B-3611 FP Multiply/Divide.

SPARC family chips produced by Cypress Semiconductors, the below Figure 4.7 shows the architecture of the Cypress CY7C601 SPARC processor and of the CY7C702 FPU. The Sun SPARC Instrucuction set contains 69 basic instructions. The FPU featunes 32 single-precision (32-bit) or 16 double—preeision (64-bit)floating-point registers (Fig. 4.7(b)). Fourteen of the 69 SPARC instructions are for floating-point operations. The SPARC architechture implements three basic instruction formats, all using a single word length of 32 bits.



(a) The Cypress CY7C601 SPARC processor

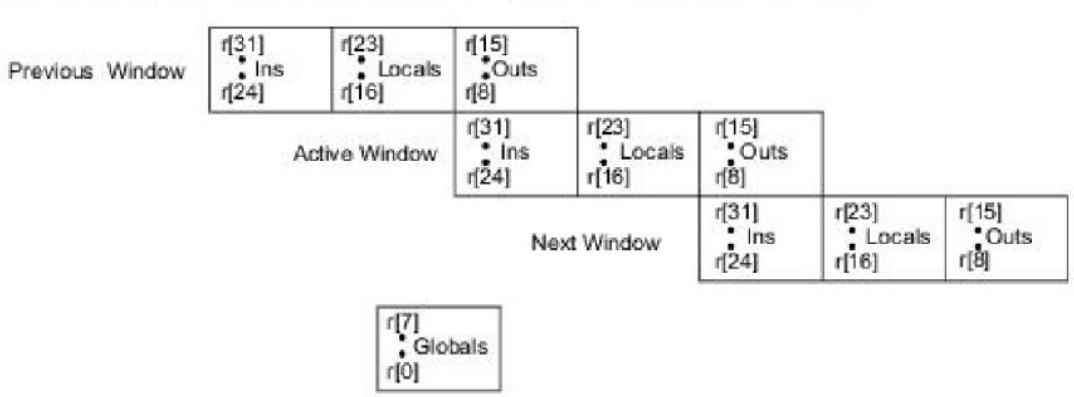


(b) The Cypress CY7C602 floating-point unit

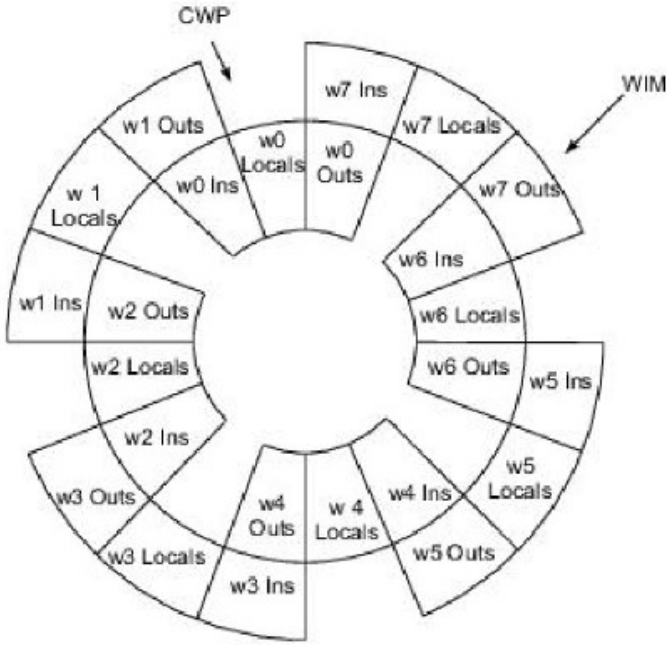
Fig. 4.7 The SPARC architecture with the processor and the floating-point unit on two separate chips (Courtesy of Cypress Semiconductor Co., 1991)

The SPARC runs each procedure with a set of thirty-two 32-bit IU registers. Eight of these registers are global registers shared by all procedures, and the remaining 24 are window registers associated with only each procedure. The concept of using overlapped register windows is illustrated in below **Fig. 4.8 for eight overlapping windows and eight globals with a total of 136 registers**, as implemented in the Cypress 1501. Each register window is divided into three eight-register sections, labeled Ins, Locals, and Outs. The

local registers are only locally addressable by each procedure. The Ins and Outs are shared among procedures.The calling procedure passes parameters to the called procedure via its Outs registers, which are the Ins registers of the called procedure. The window of the currently running procedure is called the active window pointed to by a current window pointer. A window invalid mask is used to indicate which window is invalid. The trap base register serves as a pointer to a trap handler.



(a) Three overlapping register windows and the globals registers



(b) Eight register windows forming a circular stack

Fig. 4.8 The concept of overlapping register windows in the SPARC architecture (Courtesy of Sun Microsystems Inc., 1987)

Example 4.4 The lntel i860 processor architecture

ln I989, lntel Corporation introduced the i860 microprocessor. It was a 64-bit RISC processor fabricated on a single chip. A schematic block diagram of major components in the i860 is shown in belowFig. 4.9. There were nine functional units interconnected by multiple data paths with widths ranging from 32 to I28 bits. All external or internal address buses were 32-bit wide, and the exxtemal data path or internal data bus was 64 bits wide. The instruction cache had 4 Kbytes organized as a two-way set-associative memory with 32 bytes per cache block. The data cachc was a two-way set associative memory of 8 Kbytes. Awrite-back policy was used. The bus control unit coordinated the 64-bit data transier between the chip and the outside world.The MMU implemented protected 4 Kbyte paged virtual memory via a TLB. The RISC integer unit exccuted load, store, integer,bit, and control instructions and fetched instructions for the floating-point control unit as well.

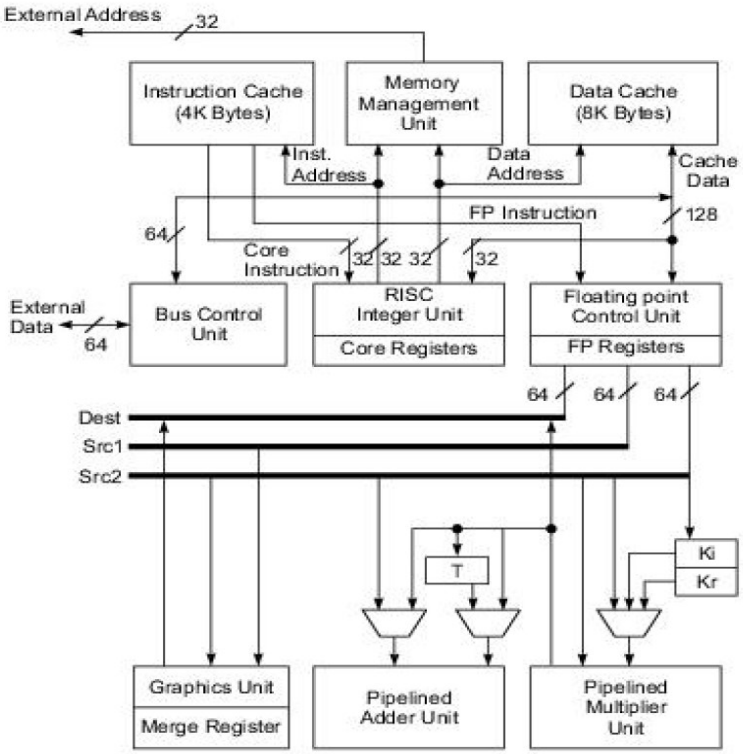


Fig. 4.9 Functional units and data paths of the Intel i860 RISC microprocessor (Courtesy of Intel Corporation, 1990)

There were two floating-point units, namely, the multiplier unit and the adder unit, which could be used separately or simultaneously under the coordination of the floating-point control unit. Special dual-operation floating-point instructions such as add and multiply and subtract and multiply used both the multiplier and adder units in parallel as shown in below Fig. 4. 10.

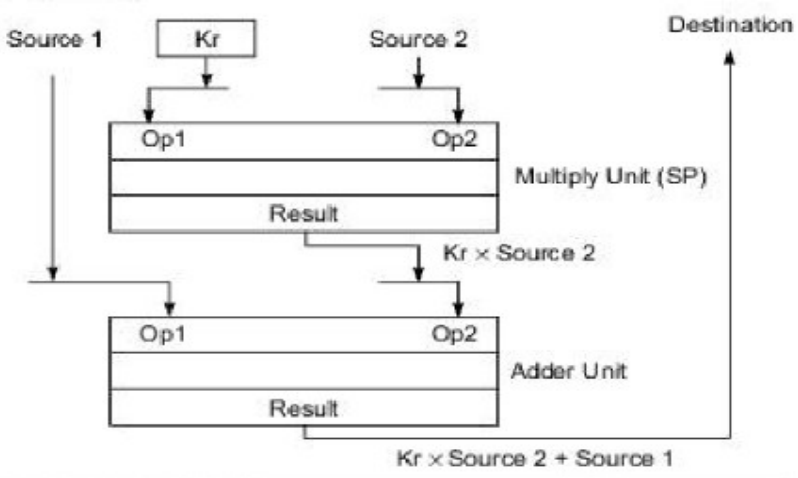


Fig. 4.10 Dual floating-point operations in the i860 processor

The RISC Impact:

The debate between RISC and CISC designers lasted for more than a decade. Based on Eq. l .3, it seems that RISC will outperform CISC if the program length docs not increase dramatically. Based on one reported experiment, converting from a CISC program to an equivalent RISC program increases the code length by only 40%.

4.2 SUPERSCALAR AND VECTOR PROCESSORS

A CISC or a RISC scalar processor can be improvcd with a superscal or vector architecture.Scalar proccssors are those executing one instruction per cycle. In a superscalarproccsser, multiple instructions are issued per cycle and multiple results are generated percycle.

4.2.1 Superscalar Processors

Superscalar processors are designed to exploit more instruction-level parallelism in user programs. Only independent instructions can be executed in parallel without causing a wait slate. The amount of instruction level parallelism varies widely depending on the type of code being executed.

Pipelining in Superscolor Processor

The fundamental structure of a three-issue superscalar pipeline is illustrated in below Fig. 4.11. Superscalar processors were originally developed as an altemative to vector processors, with a view to exploit higher degree of instruction level parallelism. A superscalar processor of degree m can issue m instructions per cycle. The base scalar processor, implemented either in RISC or ClSC, has m = l. ln order to fully utilize a superscalar processor of degree m, m instructions must be executable in parallel. In a supcrscalarprocessor due to the desire for a higher degree of instruction-level parallelism in programs, the superscalar processor depends more on an optimizing compiler to exploit parallelism.

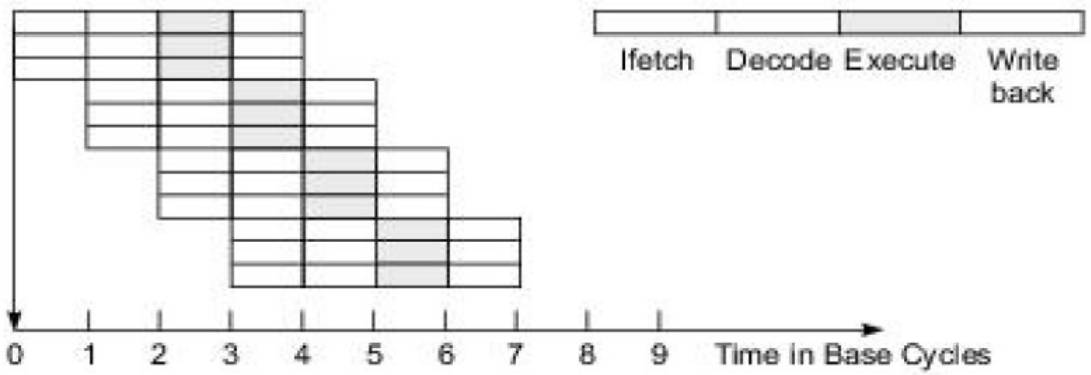


Fig. 4.11 A superscalar processor of degree m = 3

A typical superscalar architecture for a RISC processor is shown in below Fig. 4.12. The instruction cache supplies multiple instructions per fetch. The actual number of instructions issued to various functional units may vary in each cycle. The number is constrained by data dependences and resource conflicts among instructions that are simultaneously decoded. Multiple functional units are built into the integer unit and into the floating-point unit. Multiple data buses exist among the functional units. All functional units can be simultaneously used if conflicts and dependences do not exist among them during a given cycle.

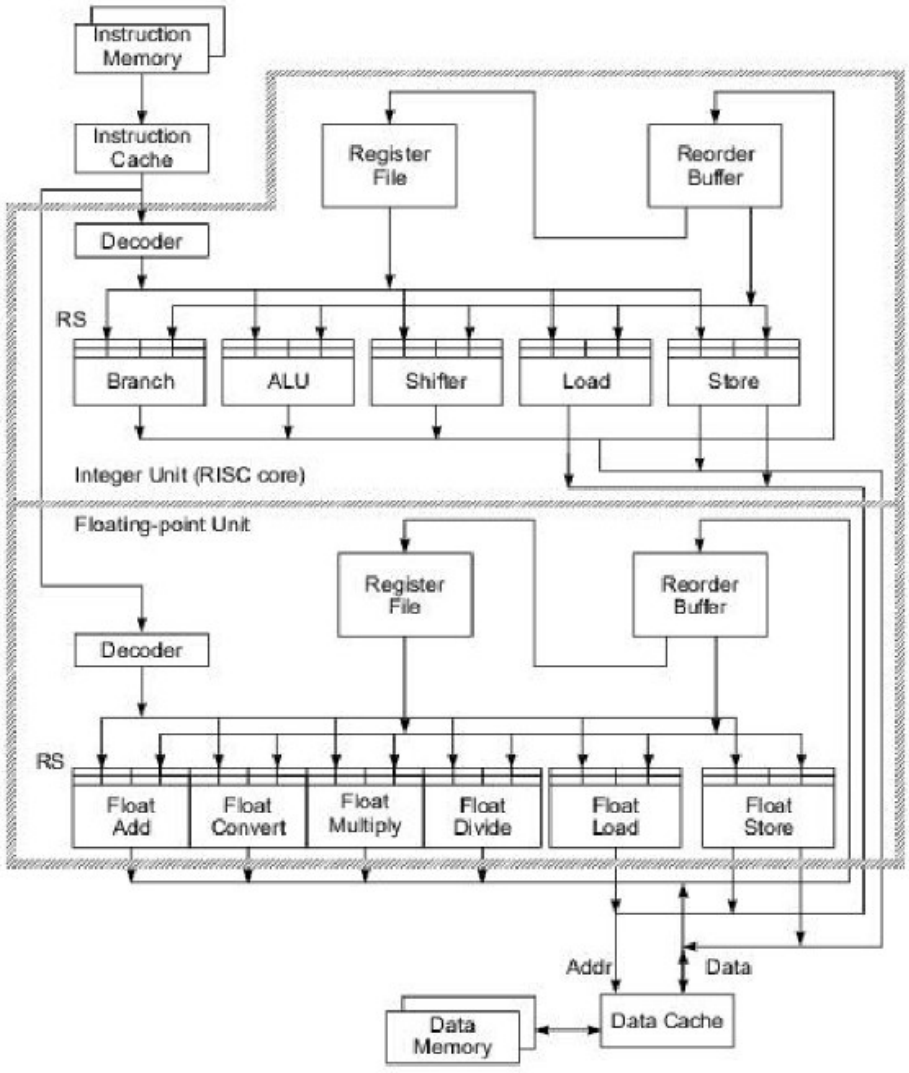


Fig. 4.12 A typical superscalar RISC processor architecture consisting of an integer unit and a floating-point unit (Courtesy of M. Johnson, 1991; reprinted with permission from Prentice-Hall, Inc.)

Representative Superscalar Processor: A number of comrnercially available processors have been implemented with the superscalar architecture. Notable early ones include the IBM RS/6000, DEC Alpha 21064, and lntel i960CA processors as summarized in Table 4.5 below.

Table 4.5 *Representative Superscalar Processors* (circa 1990)

Feature	*Intel i960CA*	*IBM RS/6000*	*DEC Alpha 21064*
Technology, clock rate, year	25 MHz 1986.	1-µm CMOS technology, 30 MHz, 1990.	0.75-µm CMOS, 150 MHz, 431 pins, 1992.
Functional units and multiple instruction issues	Issue up to 3 instructions (register, memory, and control) per cycle, seven functional units available for concurrent use.	POWER architecture, issue 4 instructions (1 FXU, 1 FPU, and 2 ICU operations) per cycle.	Alpha architecture, issue 2 instructions per cycle, 64-bit IU and FPU, 128-bit data bus, and 34-bit address bus implemented in initial version.
Registers, caches, MMU, address space	l-KB I-cache, 1.5-KB RAM, 4-channel I/O with DMA, parallel decode, multiported registers.	32 32-bit GPRs, 8-KB I-cache, 64-KB D-cache with separate TLBs.	32 64-bit GPRs, 8-KB I-cache, 8-KB D-cache, 64-bit virtual space designed, 43-bit address space implemented in initial version.
Floating-point unit and functions	On-chip FPU, fast multimode interrupt, multitask control.	On-chip FPU 64-bit multiply, add, divide, subtract, IEEE 754 standard.	On-chip FPU, 32 64-bit FP registers, 10-stage pipeline, IEEE and VAX FP standards.
Claimed per-formance and remarks	30 VAX/MIPS peak at 25 MHz, real-time embedded system control, and multiprocessor applications.	34 MIPS and 11 Mflops at 25 MHz on POWER station 530.	300 MIPS peak and 150 Mflops peak at 150 MHz, multiprocessor and cache coherence support.

Note: KB = Kbytes, FP = floating point.

Example 4.5 The IBM RS/6000 architecture

In early 1990, IBM announced the RISC System 6000. It was a superscalar processor as illustrated in below Fig. 4.13, with three functional units called the branch proeessor,fixed-point unit, and floafing-point unit, which could operate in parallel. The branch processor could arrange the execution of up to five instructions per cycle. These included one branch instruction in the branch processor, one fixed-point instruction in the FXU, one condition-register instruction in the branch processor, and one floating—point muitrpiy-add instruction in the FPU, which could be counted as two floating-point operations.

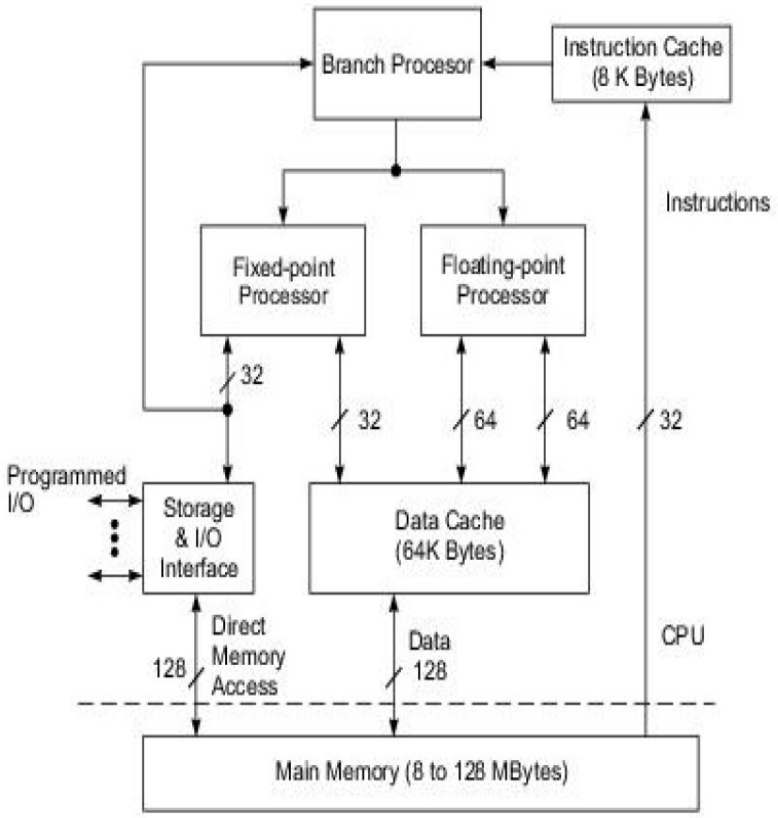
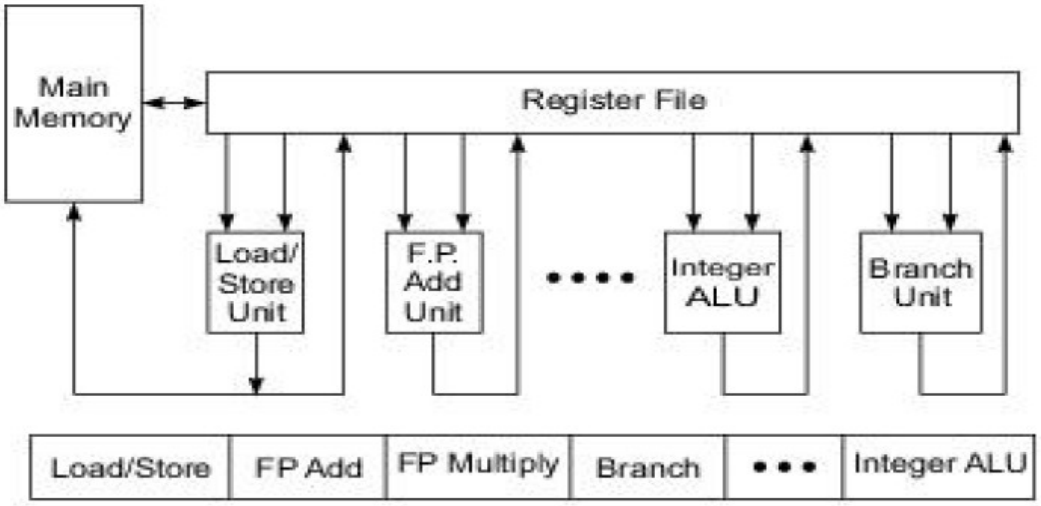


Fig. 4.13 The POWER architecture of the IBM RISC System/6000 superscalar processor (Courtesy of International Business Machines Corporation, 1990)

4.2.2 The VLIWArchitecture

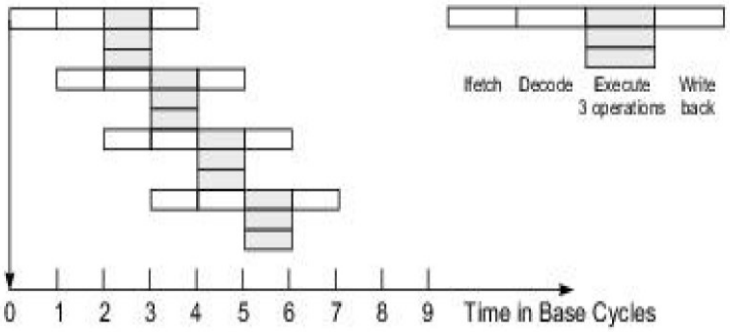
The VLIW architecture is generalized from two well established concepts: horizontal mierocoding and superscalar processing. A typical VLIW (very long instruction word) machine has instruction words hundreds of bits in length. As illustrated in bel Fig. 4.l4a, multiple functional units are used concurrently in a VLIW processor. All functional units share the use of a common large register file. Different fields of the long instruction word carry the opcodes to be dispatched to different functional units.



(a) A typical VLIW processor with degree $m = 3$

Pipelining in VLIW Processor:

The execution of instructions by an ideal VLIW processor is shown in below Fig. 4.14b, each instruction specifies multiple operations. lnstruction parallelism and data movement in a VLIW architecture are completely specified at compile time.



(b) VLIW execution with degree $m = 3$

Fig. 4.14 The architecture of a very long instruction word (VLIW) processor and its pipeline operations (Courtesy of Multiflow Computer, Inc., 1987)

VLIW machines behave much like superscalar machines with three diiferences:

1. The decoding of VLIW instructions is easier than that of superscalar instructions.
2. The code density of the superscalar machine is better whcn the available instruction level parallelism is less than that exploitable by the VLIW machine.
3. A superscalar machine can be object-code-compatible with a large family of non-parallel machines.

VLIW Opportunities

ln a VLIW architecture, random parallelism among scalar operations is exploited instead of regular or synchronous parallelism as in a vectorized supercomputer or in an SIMD computer.The sueeess of a VLIW processor depends heavily on the effieieney in code compaction.

The instruction parallelism embedded in the compacted code may require a difierent latency to be executed by different firnctional units even though the instructions are issued at the same time. Therefore, different implementations of the same VLIW architecture may not be binary-compatible with each other.

ln general-purpose applications, the architecture may not be able to perform well. Due to its lack of compatibility with conventional hardware and software, the VLIW architecture has not entered the mainstream of computers.

4.2.3 Vector and Symbolic Processors

A vector processor is specially designed to perform vector computations. A vector instruction involves a large array of operands. The same operation will be performed over an array or a string of data. Specialized vector processors are generally used in supercomputers.

A vector processor can assume either a register to register architecture which uses a shorter instructions and vector register files, or a memory to memory architecture which uses instructions longer in length, including memory addresses.

Vector Instructions

Register-based vector instructions appear in most register-to-register vector processors like Cray supercomputers. Denote a vector register of length n as V_I, a scalar register as s_i, and a memory array of length n as M(1:n). Typical register-based vector operations are listed below, where a vector operator is denoted by a small circle "o":

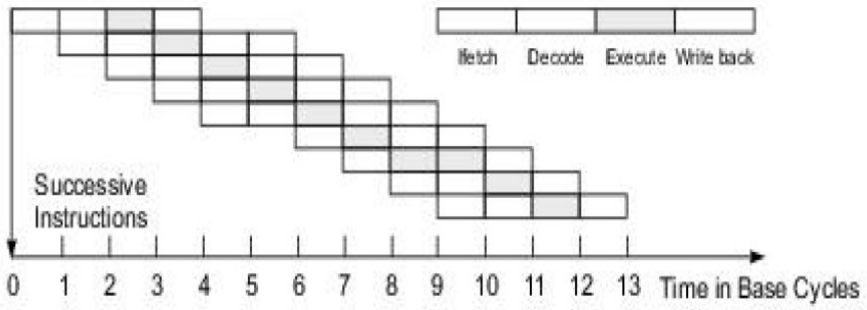
$$
\begin{array}{llllll}
\mathbf{V_1} & \text{o} & \mathbf{V_2} & \rightarrow & \mathbf{V_3} & \text{(binary vector)} \\
s_1 & \text{o} & \mathbf{V_1} & \rightarrow & \mathbf{V_2} & \text{(scaling)} \\
\mathbf{V_1} & \text{o} & \mathbf{V_2} & \rightarrow & s_1 & \text{(binary reduction)} \\
 & M(1:n) & \rightarrow & \mathbf{V_1} & \text{(vector load)} & \\
 & \mathbf{V_1} & \rightarrow & M(1:n) & \text{(vector store)} & \\
\text{o} & \mathbf{V_1} & \rightarrow & \mathbf{V_2} & \text{(unary vector)} & \\
\text{o} & \mathbf{V_1} & \rightarrow & s_1 & \text{(unary reduction)} &
\end{array}
\qquad (4.1)
$$

In all cases, these vector operations are performed by dedicated pipeline units, including functional pipelines and mensory-access pipelines.

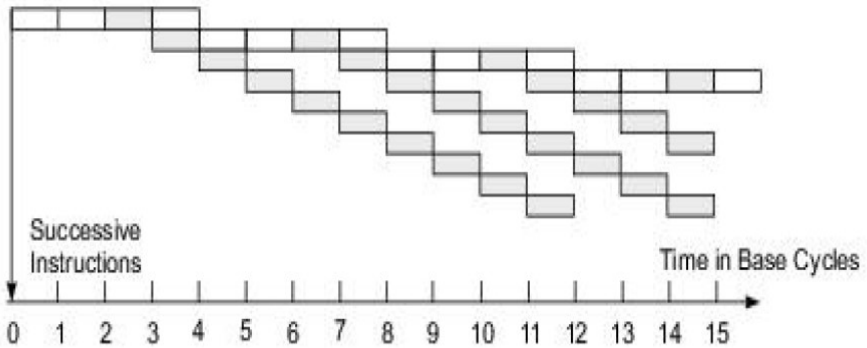
Memory-based vector operations are found in memory-to-memory vector processors such as those in the early supercomputer CDC Cyber 205. Listed below are a few examples:

$$
\begin{array}{rcll}
M_1(1:n) & o & M_2(1:n) & \rightarrow & M(1:n) \\
s_1 & o & M_1(1:n) & \rightarrow & M_2(1:n) \\
 & o & M_1(1:n) & \rightarrow & M_2(1:n) \\
M_1(1:n) & o & M_2(1:n) & \rightarrow & M(k)
\end{array} \qquad (4.2)
$$

Vector Pipeline: Vector processors take advantage of unrolled-loop-level parallelism. The vector pipelines can be attached to any scalar or superscalar processor.The pipclined execution in a vector proeessor is compared with that in a scalar processor in below Fig. 4.15. In Figure 4.15:a each scalar instruction executes only one operation over one data element.



(a) Scalar pipeline execution (Fig. 4.2a redrawn)



(b) Vector pipeline execution

Fig. 4.15 Pipelined execution in a base scalar processor and in a vector processor, respectively (Courtesy of Jouppi and Wall; reprinted from Proc. ASPLOS, ACM Press, 1989)

Symbolic Processors:

Symbolic processing has been applied in many areas, like theorem proving, pattern recognition, etc. ln these applications, data and knowledge representations, primitive operations, algorithmic behavior and special architectural features are different than in numerical computing. Symbolic processors have also been called prolog processors, Lisp processors, or symbolic processors. Below Table 4.6 summarizes these characteristics.

Table 4.6 *Characteristics of Symbolic Processing*

Attributes	*Characteristics*
Knowledge Representations	Lists, relational databases, scripts, semantic nets, frames, blackboards, objects, production systems.
Common Operations	Search, sort, pattern matching, filtering, contexts, partitions, transitive closures, unification, text retrieval, set operations, reasoning.
Memory Requirements	Large memory with intensive access pattern. Addressing is often content -based. Locality of reference may not hold.
Communication Patterns	Message traffic varies in size and destination; granularity and format of message units change with applications.
Properties of Algorithms	Nondeterministic, possibly parallel and distributed computations. Data dependences may be global and irregular in pattern and granularity.
Input/Output requirements	User-guided programs; intelligent person-machine interfaces; inputs can be graphical and audio as well as from keyboard; access to very large on-line databases.
Architecture Features	Parallel update of large knowledge bases, dynamic load balancing; dynamic memory allocation; hardware-supported garbage collection; stack processor architecture; symbolic processors.

Example 4.6 The Symbolics 3600 Lisp processor

The processor architecture ofthe Symbolics 3600 is shown in below Fig. 4.16. This was a stack-oriented machine. The Symbolics 3-500 executed most Lisp instructions in one machine cyele. integer instructions fetched operands form the stack buffer and the duplicate top of the stack in the scratch-pad memory. Floating-point addition, garbage collection, data type checking by the tag processor, and fixed-point addition could be carried out in parallel.

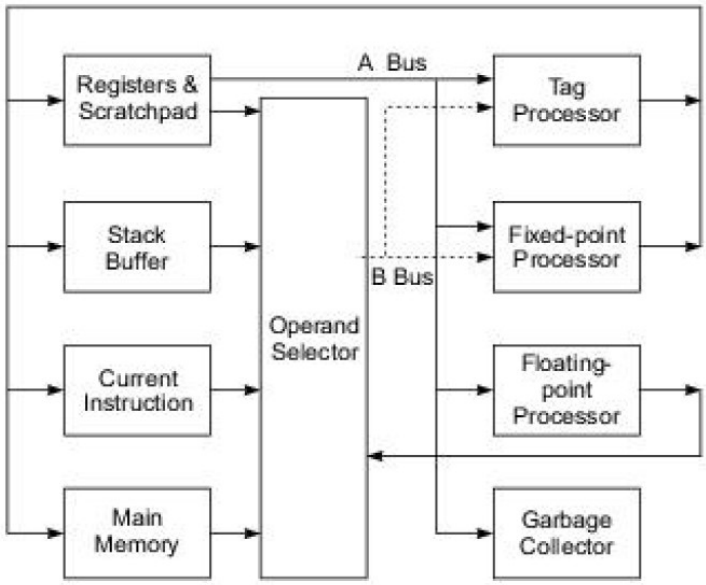


Fig. 4.16 The architecture of the Symbolics 3600 Lisp processor (Courtesy of Symbolics, Inc., 1985)

4.3 MEMORY HIERARCHY TECHNOLOGY

4.3.1 Hierarchical Memory Technology

Storage devices such as registers, caches, main memory, disk devices, and backup storages are often organized as a hierarchy as depicted in below in Fig. 4.17.

The memory technology and storage organization at each level are characterized by five parameters:

The access time(ti)- It refers to the round-trip time from the CPU to the ith-level memory

The memory size(si)- It is the number of bytes or words in level i.

The cost per byte(ci)- The cost of the ith-level memory is estimated by the product ci*si.

The transfer bandwidth(bi)- It refers to the rate at which information is transferred between adjacent levels.

The unit of transfer(xi)- It refers to the grain size for data transfer between levels i and i+1.

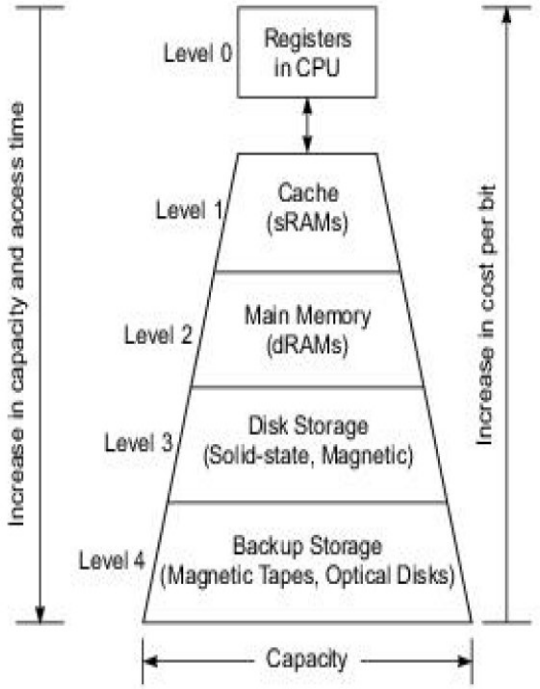


Fig. 4.17 A four-level memory hierarchy with increasing capacity and decreasing speed and cost from low to high levels

Register and Cache: The registers are parts of the processor, multi-level caches are built either on the processor chip or on the processor board. Register assignment is made by the compiler. Register transfer operations are directly controlled by the processor after instructions are decoded. Register transfer is conducted at processor speed, in one clock cycle.

Main Memory: The main memory is sometimes called the primary memory of a computer system. It is usually much larger than the cache and oflcn implemented by the most cost-effective RAM chips, such as DDR SDRAMs. The main memory is managed by a MMU in cooperation with the operating system.

Disk Drives and Backup Storage: The disk storage is considered the highest level of on-line memory. lt holds the system programs such as the OS and compilers, and user programs and their daia sets. Optical disks and magnetic tape units are off-line memory for use as archival and backup storage. Disk drives are also available in the form of RAID arrays.

Peripheral Technology: Peripheral devices include printers, plotters, terminals, monitors, graphics displays ,etc.Some I/O devices are tied to special-purpose applications.

Table 4.7 *Memory Characteristics of a Typical Mainframe Computer in 1993*

Memory level Characteristics	*Level 0 CPU Registers*	*Level 1 Cache*	*Leve 2 Main Memory*	*Level 3 Disk Storage*	*Level 4 Tape Storage*
Device technology	ECL	256K-bit SRAM	4M-bit DRAM	1-Gbyte magnetic disk unit	5-Gbyte magnetic tape unit
Access time, t_i	10 ns	25–40 ns	60–100 ns	12–20 ms	2–20 min (search time)
Capacity, s_i (in bytes)	512 bytes	128 Kbytes	512 Mbytes	60–228 Gbytes	512 Gbytes–2 Tbytes
Cost, c_i (in cents/KB)	18,000	72	5.6	0.23	0.01
Bandwidth, b_i (in MB/s)	400–800	250–400	80–133	3–5	0.18–0.23
Unit of transfer, x_i	4–8 bytes per word	32 bytes per block	0.5–1 Kbytes per page	5–512 Kbytes per file	Backup storage
Allocation management	Compiler assignment	Hardware control	Operating system	Operating system/user	Operating system/user

4.3.1 Inclusion, Coherence, and Locality

Information stored in a memory hierarchy (M1,M2,...,Mn) satisfies the following three important properties Inclusion, Coherence, and Locality as shown in below Fig 4.18.

Inclusion Property

The indusion property is stat$M_1 \subset M_2 \subset M_3 \subset \ldots \subset M_n$. If an information word is found in level Mi, then copies of the same word can also be found in all upper levels M_{i+1},M_{i+2},...,M_n. A word stored in M_{i+1} may not be found in M_i. A word miss in Mi implies that it is also missing from all lower levels M_{i-1},M_{i-2},...,M_1. The highest level is the backup storage, where everything can be found.

The cache (M_1) is divided into eache blocks.Each block may be typically 32 bytes(8 words). Blocks (such as a and b in Fig. 4.18) are the units of data transfer between the Cache and main memory, or between L_1 and L_3 Cache, etc. The main memory (M3) is divided into pages, say, 1 Kbytes each. Each page contains 128 blocks as shown in Fig. 4.18. Scattered pages are organized as a segment in the disk memory, for example, segment F contains page A, page B, and other pages.

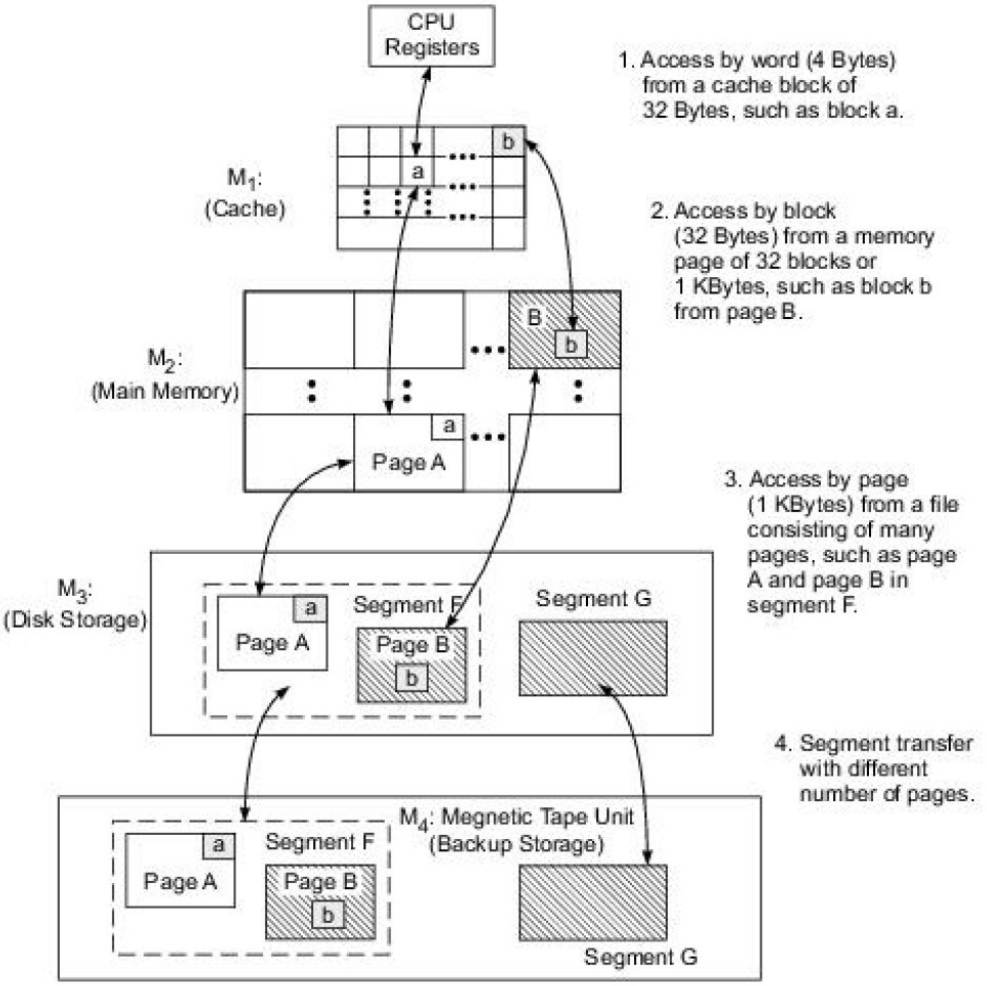


Fig. 4.18 The inclusion property and data transfers between adjacent levels of a memory hierarchy

Coherence Property

The Coherence property requires that copies of the information item at sueeessive memory levels be consistent. if a word is modified in the cache, copies of that word must be updated immediately or eventually at all higher levels.

There are two strategies tor maintaining the coherence in a memory hierarchy:

The first method is called write-through (WT), which demands immediate update in M_{i+1}, if a word is modified in M_i, for i= 1,2,.., n-1.

The second method is write-Back (WB), which delays the update in M_{i+1}, until the word being modified in Mi is replaced or removed from M_i.

Locality of Reference:

The memory hierarchy was developed based on a program behavior known as Locality of Reference. Memory references are generated by the CPU for either instruction or data access. These accesses tend to be clustered in certain regions in time, space, and ordering.

There are three dimensions of the locality property: Temporal, Spatial, anrl Sequential. During the lifetime of a software process, a number of pages are used dynamically. The references to these pages follow certain access pattems as illustrated in Fig. 4.19.

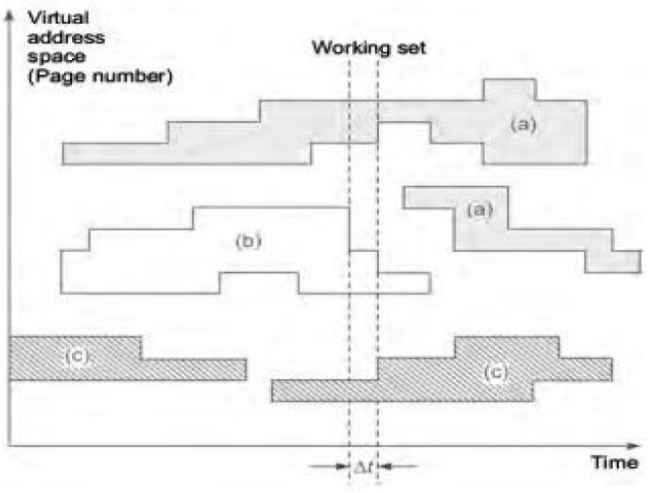


Fig. 4.19 Memory reference patterns in typical program trace experiments, where regions (a), (b), and (c) are generated with the execution of three software processes

These memory reference patterns are caused by the following locality properties:

Temporal Locality- Recently referenced items(instructions or data)are likely to be referenced again in the near future. This is often caused by special program constructs such as iterative loops or subroutines. Thus temporal locality tends to cluster the access in the reoently used areas.

Spatial Locality-This refers to the tendency for a process to access items whose addresses are near one another. For example, operations on tables or arrays involve accesses of a certain clustered area in the address space. Program segments, such as routines and macros, tend to be stored in the same neighborhood of the memory space.

Sequential locality—ln typical programs, the execution of instructions follows a sequential order (or the program order) unless branch instructions create out-of-order executions. The ratio of in-order rxecution to out-of-order execution is roughly 5 to 1 in ordinary programs. Besides, the access of a large data array also follows a sequential order.

Memory Design Implications

Each type of locality of reference affects the design of the memory hierarchy. the temporal locality leads to the popularity of the Least Recently Used(LRU) replacement algorithm. The spatial locality assists in determining the size of unit data transfers between adjacent memory levels. The temporal locality also helps determine the size of memory at successive levels.The sequential locality affects the determination of grain size for optimal scheduling.Prefetch techniques are heavily afiected by the locality properties. The principle of localities guides the design of cache, main memory, and even virtual memory organization.

The Working Sets

Figure 4.19 above shows the memory reference patterns of three running programs or three software processes. As a function of time, the virtual address space (identified by page numbers) is clustered into regions due to the locality of references.During the execution of a program, the working set changes slowly and maintains a certain degree of continuity as demonstrated in Fig. 4.19. This implies that the working set is often accumulated at the innermost (lowest) level such as the cache in the memory hierarchy. This will reduce the effective memory access time with a higher hit ratio at the lowest memory level. The time window Δt is a critical parameter set by the OS kemel which affects the size of the working set and thus the desired cache size.

4.3.3 Memory Capacity Planning

The performance of a memory hierarchy is determined by the effective access time of T_{eff} to any level in the hierarchy. lt depends on the hit ratios and Access frequencies at successive levels.

Hit Ratios

Hit ratio is a concept defined for any two adjacent levels of a memory hierarchy. When an information item is found in Mi, it is called a hit, otherwise a miss. Consider memory levels M_i and M_{i-1}.in a hierarchy i= 1, 2,..,n. The hit ratio hi at Mi is the probability that an information item will be found at M_i. lt is a firnction of the characteristics of the two adjacent levels M_{i-1} and M_i. The miss ratio at M_i is defined as $1-h_i$. Successive hit ratios are independent random variables with values between 0 and 1.

The access frequency to M_i, is defined as $f_i=(1-h_1)(1-h_2)..(1-h_i-1)h_i$. This is the probability of successfully aecessing Mi, when there are i-1 misses at the lower levels and a hit at M_i.

Note that $\sum fi=1$, for i=1 to n. And $f_1=h1$.

Due to the locality property, the access frequencies decreases very rapidly from low to high levels that is, f1>>f2>>..>>fn,. This implies that the inner levels of memory are accessed more often than the outer levels.

Effective Access Time

Every time a miss occurs, a penalty must be paid to access the next higher level of memory. The misses are called **block misses** in the cache and **page faults** in the main memory. The time penalty for a page fault is much longer than that for a block miss due to the fact that t1<t2<t3. A cache miss is 2 to 4 times as costly as a cache hit, but a page fault is 1000 to 10,000 times as costly as a page hit.

Using the access frequencies f_i for $i = 1, 2, \ldots, n$, we can formally define the *effective access time* of a memory hierarchy as follows:

$$T_{eff} = \sum_{i=1}^{n} f_i \cdot t_i$$

$$= h_1 t_1 + (1 - h_1) h_2 t_2 + (1 - h_1)(1 - h_2) h_3 t_3 + \ldots +$$

$$(1 - h_1)(1 - h_2) \ldots (1 - h_{n-1}) t_n \qquad (4.3)$$

Hierarchy Optimization The total cost of a memory hierarchy is estimated as follows:

$$C_{\text{total}} = \sum_{i=1}^{n} c_i \cdot s_i \qquad (4.4)$$

This implies that the cost is distributed over n levels. Since $c_1 > c_2 > c_3 > \ldots c_n$, we have to choose $s_1 < s_2 < s_3 < \ldots s_n$. The optimal design of a memory hierarchy should result in a T_{eff} close to the t_1 of M_1 and a total cost close to the cost of M_n. In reality, this is difficult to achieve due to the tradeoffs among n levels.

The optimization process can be formulated as a linear programming problem, given a ceiling C_0 on the total cost— that is, a problem to minimize

$$T_{eff} = \sum_{i=1}^{n} f_i \cdot t_i \qquad (4.5)$$

subject to the following constraints:

$$s_i > 0, t_i > 0 \quad \text{for } i = 1, 2, \ldots, n$$

$$C_{\text{total}} = \sum_{i=1}^{n} c_i \cdot s_i < C_0 \qquad (4.6)$$

Example 4.1 The design of a memory hierarchy

Consider the design of a three-level memory hierarchy with the following specifications for memory characteristics:

Memory level	*Access time*	*Capacity*	*Cost/Kbyte*
Cache	t_1 = 25 ns	s_1 = 512 Kbytes	c_1 = $0.12
Main memory	t_2 = unknown	s_2 = 32 Mbytes	c_2 = $0.02
Disk array	t_3 = 4 ms	s_3 = unknown	c_3 = $0.00002

The design goal is to achieve an effective memory access time t= 10.04μs with a cache hit ratio h1 = 0.98 and a hit ratio h2 = 0.9 in main memory. Also, the total cost of the memory hierarchy is upper-bounded by $15,00. The memory hierarchy cost is calculated as

C=c1s1+c2s2+c3s3<=15000

The maximum capacity of the disk is thus obtained as s_3 = 39.8 Gbytes without exceeding the budget.

Next, we want to choose the access tirne(t_2) of the RAM to build the main memory. The effective memory access time is calculated as

$T=h_1t_1+(1-h_1)h_2t_2+(1-h_1)(1-h_2)h_3t_3<=10.04$

Substituting all known parameters, we have $10.04 \times 10^{-6} = 0.98 \times 25 \times 10^{-9}$ $+0.02 \times 0.9 \times t2 +0.02 \times 0.1 \times 1 \times 4 \times 10^{-3}$. Thus t2=903ns.

Suppose one wants to double the main memory to 64 Mbytes at the expense of reducing the disk capacity under the same budget limit. This change will not affect the cache hit ratio. But it may increase the hit ratio in the main memory, and there by, the effective memory access time will be reduced.

4.4 VIRTUAL MEMORY TECHNOLOGY

4.4.1 Virtual Memory Models

The main memory is considered the physical memory in which multiple running programs may reside.The limited-size physical memory cannot load in all programs fully and simultaneously. The virtual memory was introduced to alleviate this problem. The idea is to expand the use of the physical memory among many programs with the help of an auxiliary memory such as disk arrays.Only active programs or portions of them become residents of the physical memory at one time. Active portions of programs can be loaded in and out from disk to physical memory dynamically under the coordination of the operating system. To the users, virtual memory provides almost unbounded memory space to work with.

Address Space

Each word in the physical memory is identified by a unique physical address. All memory words in the main memory form a physical address space. Virtual addresses are those used by machine instructions making up an executablc program.The virtual addresses must be translated into physical addresses at run time. A system of translation tables and mapping functions are used in this process.

Address Mapping Let V be the set of virtual addresses generated by a program running on a processor. Let M be the set of physical addresses allocated to run this program. A virtual memory system demands an automatic mechanism to implement the following mapping:

$$f_t : V \rightarrow M \cup \{\phi\} \tag{4.9}$$

This mapping is a time function which varies from time to time because the physical memory is dynamically allocated and deallocated. Consider any virtual address $v \in V$. The mapping f_t is formally defined as follows:

$$f_t(v) = \begin{cases} m, & \text{if } m \in M \text{ has been allocated to store the} \\ & \quad \text{data identified by virtual address } v \\ \phi, & \text{if data } v \text{ is missing in } M \end{cases} \tag{4.10}$$

In other words, the mapping $f_t(v)$ uniquely translates the virtual address v into a physical address m if there is a *memory hit* in M. When there is a *memory miss*, the value returned, $f_t(v) = \phi$, signals that the referenced item (instruction or data) has not been brought into the main memory at the time of reference.

VIRTUAL MEMORY MODELS

Private Virtual Memory

The below fig 4.20a shows a private virtual memory space associated with each processor, as was seen in the VAX/11 and in most UNIX systems. Each private virtual space is divided into pages. Virtual pages from different virtual spaces are mapped into the same physical memory shared by all processors. The **advantages** of using private virtual memory include the use of a small processor address space(32 bits), protection on each page or on a per-process basis, and the use of private memory maps, which require no locking.The **shortcoming** lies in the synonym problem, in which diiferent virtual addresses in different virtual spaces point to the same physical page.

Shared Virtual Memory

This model combines all the virtual address spaces into a single globally shared virtual space (Fig. 4.20b). Each processor is given a portion of its shared virtual memory to declare their addresses. Diffrent processors may use disjoint spaces. Some areas of virtual space can be also shared by multiple processors. Examples of maehines using shared virtual memory includethe IBM801 and RT. The **advantages** in using shared virtual memory include the fact that all addresses are unique. Synonyms are not allowed in a globally shared virtual memory.The page table must allow shared accesses. Therefore, Mutual exclusion(locking) is needed to enforce protected access. Segmentation is built on top of the paging system to confine each process to its own address space (segments).

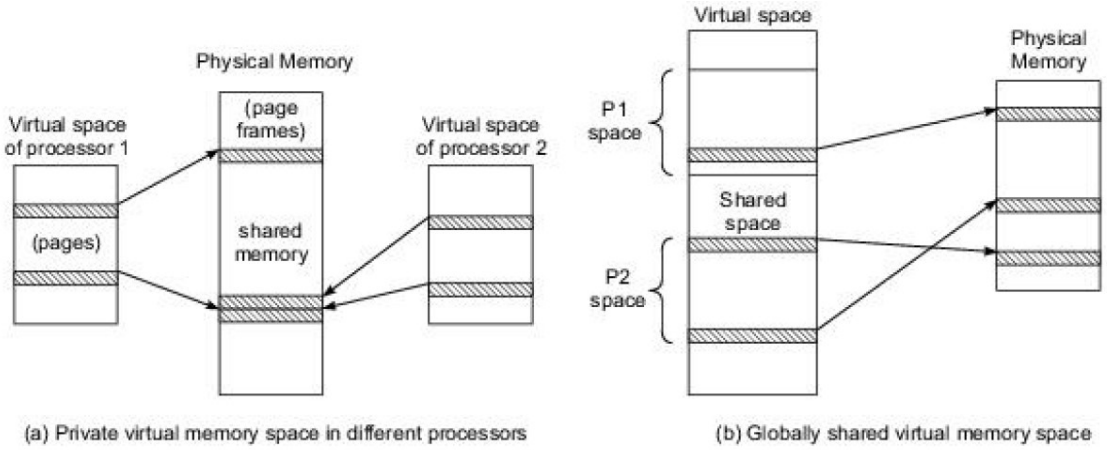


Fig. 4.20 Two virtual memory models for multiprocessor systems (Courtesy of Dubois and Briggs, tutorial, *Annual Symposium on Computer Architecture*, 1990)

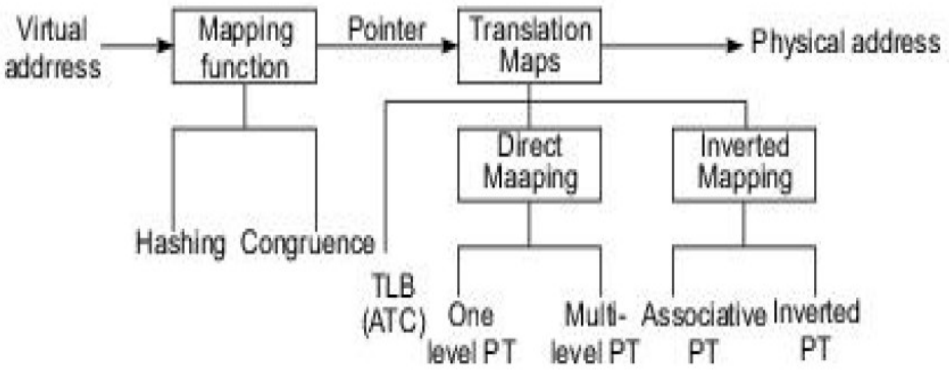
4.4.1 TLB, Paging, and Segmentation

Both the virtual memory and physical memory are partitioned into fixed-length pages.The purpose of memory allocation is to allocate pages of virtual memory to the page frames of the physical memory.

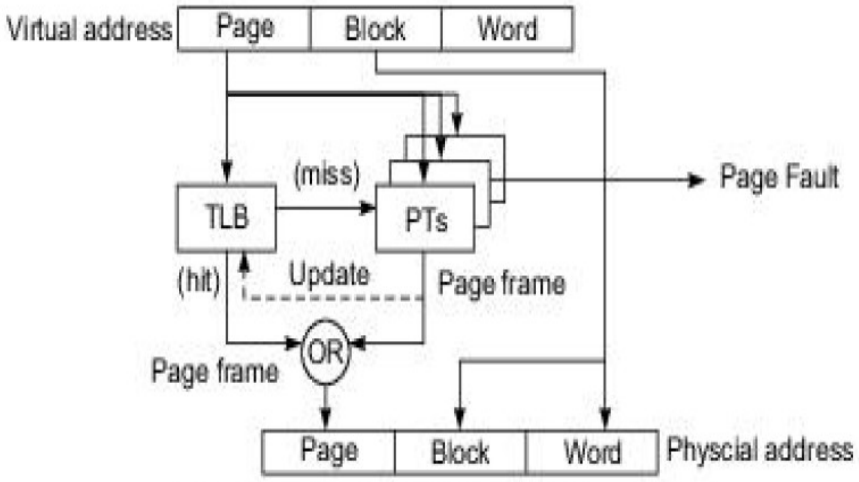
Address Translation Mechanism:

The process of address translation is the translation of virtual addresses into physical addresses. Various schemes for virtual address translation are surnmarized in Fig. 4.21a, which needs the use of translation maps which can be implemented in various ways. Translation maps are stored in the eache, in associative memory, or in the main memory. To access these maps, a mapping function is applied to the virtual address. This function generates a pointer to the desired translation map. This mapping can be implemented with a hashing or congruenee function.

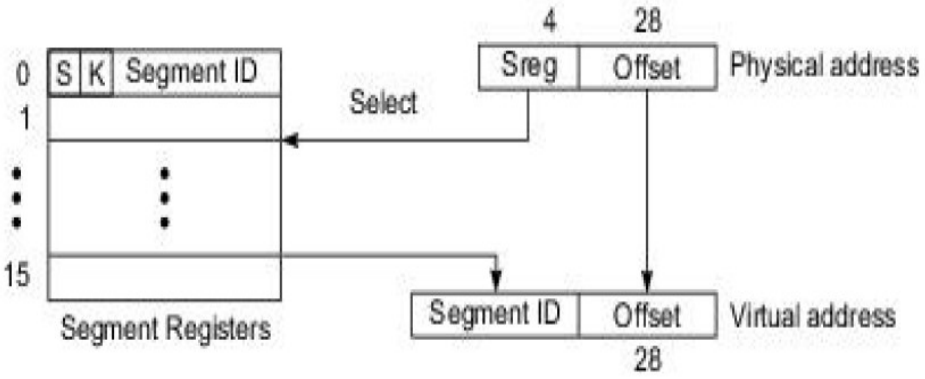
Hashing is a simple computer technique for converting a long page number into a short one with fewer bits. The hashing function should randomize the virtual page number and produce a unique bashed number to be used as the pointer.



(a) Virtual addrress translation schemes (PT = page table)



(b) Use of a TLB and PTs for address translation



(c) Inverted address mapping

Fig. 4.21 Address translation mechanisms using a TLB and various forms of page tables

Translation Lookaside Buffer

Translation maps appear in the form of a Translation Lookaside Buffer (TLB) and page tables(PTs). The TLB is a high-speed lookup table which stores the most recently or likely referenced page entries. A page consists of essentially a (virtual page number; page frame number) pair. The use of a TLB and P'T.s for address translation is shown in Fig 4.2l b. Each virtual address is divided into three fields: The leftmost field holds the virtual page number, the middle field identifies the cache block number, and the rightmost field is the word address within the block.

To produce the physical address consisting of the page frame number, the block number, and the word address. The first step of the translation is to use the virtual page number as a key to search through the TLB for a match. ln case of a match(hit) in the TLB, the page frame number is retrieved from the matched page entry. The cache block and word address are copied directly. ln case the match cannot be found (a miss) in the TLB, a hashed pointer is used to identify one of the page tables where the desired page frame number can be retrieved.

Paged Memory

Paging is a technique for partitioning both the physical memory and virtual memory into fixed-size pages. Page tables are used to map between pages and page frames. These tables are implemented in the main memory upon creation of user processes. The page table entries (PTE) contain (virtual page, page frame) address pairs. lf the demanded page cannot be found in the PT, a page fault is declared. A page fault implies that the referenced page is not resident in the main memory. When a page fault occurs, the running process is suspended. A context switch is made to another ready-to-run process while the missing page is transferred from the disk or tape unit to the physical memory.

Segmented Memory

A large number of pages can be shared by segmenting the virtual address space among multiple user programs simultaneously. A segment of scattercd pages is formed logically in the virtual memory space. Segrnents are defined by users in order to declare a portion of the virtual address space. segments can have variable lengths. The management of a segmented memory system is much more complex due to the nonuniform segment size.The segmented memory is arranged as a two-dimensional address space. Each virtual address in this space has a prefix. field called the segment number and a postfix field called the offset within the segment.

Paged Segment

The paging and segmentation can be combined to implement a type of virtual memory with paged segments. Within each segment, the addresses are divided into fixed-size pages. Each virtual address is thus divided into three fields. The upper field is the segment number,the middle one is the page number, and the lower one is the offset within each page.Paged segment offer the advantages of both paged memory and segmented memory. For users, program files can be better logically structured.

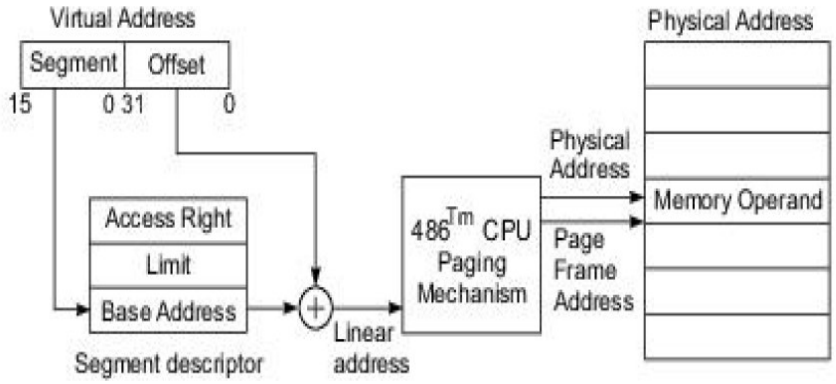
Inverted Paging

The direct paging works well with a small virtual address space such as 32 bits. ln modern computers, the virtual address is large. A large virtual address space demands either large PTs or multilevel direct paging which will slow down the address translation process and thus lower the performance. The address translation maps can also be implemented with inverted mapping (Fig. 4.21 c). An inverted page table is created for each page frame that has been allocated to users. Inverted page tables are accessed either by an associative search or by the use of a hashing fitnetion. ln using an inverted PT, only virtual pages that are currently resident in physical memory are included. This provides a significant reduction in the size of the page tables.

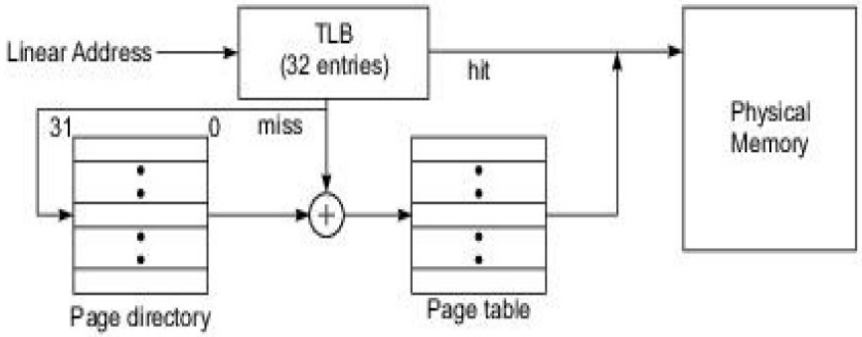
The generation of a long virtual address from a short physical address is done with the help of segment registers, as demonstrated in Fig. 4.21 c. Thc leading 4 bits (denoted sreg) of a32-bit address name a segment register. The register provides a segment id that replaces the 4-bit sreg to form a long virtual address. Given a virtual address to be translated, the hardware searches the inverted PT for that address and, if it is found, uses the table index of the matching entry as the address of the desired page frame. A hashing table is used to search through the inverted PT. Because of limited physical space, no multiple levels are needed for the inverted page table.

Example 4.8 Paging and segmentation in the Intel i486 processor

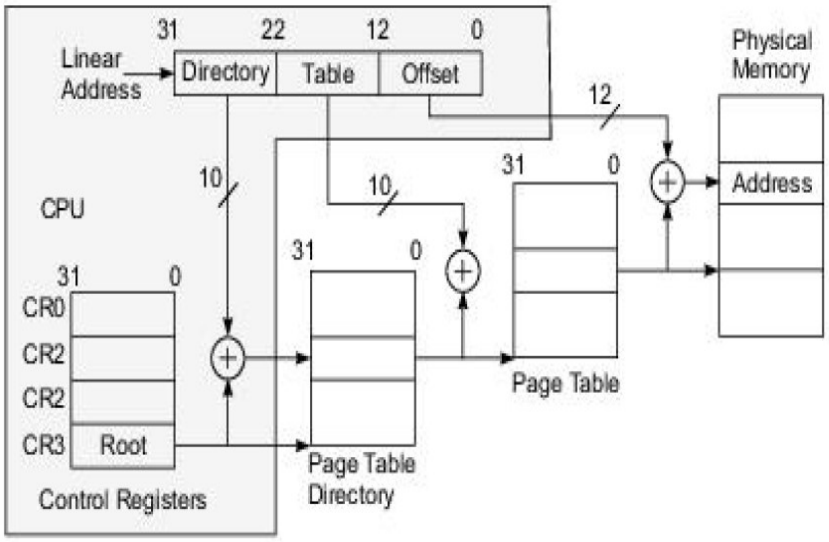
The i486 features both segmentation and paging capabilities. The maximal memory size in real mode is 1 Mbyte. Protected mode allows A segment can have any length from l byte to 4 Gbytes. A segment can start at any base address, and storage overlapping between segments is allowed. The virtual address (Fig. 4.22a) has a 16-bit segment selector to determine the base address of the linear address space to be used with thc i486 paging system. The 32-bit offset specifies the internal address within a segment. The i486 can be used with four different memory organizations, pure paging, pure segmentation, segmented paging, or pure Physical addressing without paging and segmentation. A 32-entry TLB (Fig 4.22b) is used to convert the linear address directly into the physical address without resorting to the two level paging scheme (Fig 4.22c). The standard page size on the i486 is 4 Kbytes $=2^{12}$ bytes. Four control registers are nsed to select between regular paging and page fault handling.The page table directory (4 Kbytes) allows 1024 page directory entries. Each page table at the second level is 4 Kbytes and holds up to 1034 PTEs. The upper 20 linear address bits are compared to determine if there is a hit. The hit ratios of the TLB and of the page tables depend on program behavior and the efiieicncy of the update (page replacement) policies. A 98% hit ratio has been observed in TLB operations.



(a) Segmentation to produce the linear address



(b) The TLB operations



(c) A two-level paging scheme

Fig. 4.22 Paging and segmentation mechanisms built into the Intel i486 CPU (Courtesy of Intel Corporation, 1990)

4.4.3 Memory Replacement Policies

Memory management policies include the allocation and deallocation of memory pages to active processes and the replacement of memory pages. **Page replacement** refers to the process in which a resident page in main memory is replaced by a new page transferred from the disk.

The goal of a page replacement policy is to minimize the number of possible page faults so that the effective memory-access time can be reduced. The effectiveness of a replacement algorithm depends on the program behavior and memory traffic patterns encountered. A good policy should match the program locality property. The policy is also affected by page size and by the number of availablc frames.

Page Trace

A page trace is a sequence of page frame nunrbers (PFNs) generated during the execution of a given program and is used to analyze the performance of a paging memory system. By tracing the successive PFNs in a page trace against the resident page numbers in the page frames, one can determine the occurrence of page hits or page faults.

Consider a page trace $P(n) = r(1)r(2) \ldots r(n)$ consisting of n PFNs requested in discrete time from 1 to n, where $r(t)$ is the PFN requested at time t. We define two reference distances between the repeated occurrences of the same page in $P(n)$.

The *forward distance* $f_t(x)$ for page x is the number of time slots from time t to the first repeated reference of page x in the future:

$$f_t(x) = \begin{cases} k, & \text{if } k \text{ is the smallest integer such that} \\ & r(t+k) = r(t) = x \text{ in } P(n) \\ \infty, & \text{if } x \text{ does not reappear in } P(n) \text{ beyond time } t \end{cases} \quad (4.11)$$

Similarly, we define a *backward distance* $b_t(x)$ as the number of time slots from time t to the most recent reference of page x in the past:

$$b_t(x) = \begin{cases} k, & \text{if } k \text{ is the smallest integer such that} \\ & r(t-k) = r(t) = x \text{ in } P(n) \\ \infty, & \text{if } x \text{ never appeared in } P(n) \text{ in the past} \end{cases} \quad (4.12)$$

Let $R(t)$ be the *resident set* of all pages residing in main memory at time t. Let $q(t)$ be the page to be replaced from $R(t)$ when a page fault occurs at time t.

Page Replacement Policies The following page replacement policies are specified in a demand paging memory system for a page fault at time t.

(1) *Least recently used* (LRU)—This policy replaces the page in $R(t)$ which has the longest backward distance:

$$q(t) = y, \quad \text{iff} \quad b_t(y) = \max_{x \in R(t)} \{b_t(x)\} \tag{4.13}$$

(2) *Optimal* (OPT) *algorithm*—This policy replaces the page in $R(t)$ with the longest forward distance:

$$q(t) = y, \quad \text{iff} \quad f_t(y) = \max_{x \in R(t)} \{f_t(x)\} \tag{4.14}$$

(3) *First-in-first-out* (FIFO)—This policy replaces the page in $R(t)$ which has been in memory for the longest time.

(4) *Least frequently used* (LFU)—This policy replaces the page in $R(t)$ which has been least referenced in the past.

(5) *Circular FIFO*—This policy joins all the page frame entries into a circular FIFO queue using a pointer to indicate the front of the queue. An *allocation bit* is associated with each page frame. This bit is set upon initial allocation of a page to the frame.

When a page fault occurs, the queue is circularly scanned from the pointer position. The pointer skips the allocated page frames and replaces the very first unallocated page frame. When all frames are allocated, the front of the queue is replaced, as in the FIFO policy.

(6) *Random replacement*—This is a trivial algorithm which chooses any page for replacement randomly.

Example 4.9 Page tracing experiments and interpretation of results

Consider a paged virtual memory system with a two-level hierarchy: main memory M_1 and disk memory M_2. For clarity of illustration, assume a page size of four words. The number of page frames in M_1 is 3, labeled a, b and c; and the number of pages in M_2 is 10, identified by 0, 1, 2, …, 9. The ith page in M_2 consists of word addresses $4i$ to $4i + 3$ for all $i = 0, 1, 2, \ldots, 9$.

A certain program generates the following sequence of word addresses which are grouped (underlined) together if they belong to the same page. The sequence of page numbers so formed is the *page trace*:

Word trace:	0,1,2,3,	4,5,6,7,	8,	16,17,	9,10,11,	12,	28,29,30,	8,9,10,	4,5,	12,	4,5
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Page trace:	0	1	2	4	2	3	7	2	1	3	1

Page tracing experiments are described below for three page replacement policies: LRU, OPT, and FIFO, respectively. The successive pages loaded in the page frames (PFs) form the trace entries. Initially, all PFs are empty.

	PF	0	1	2	4	2	3	7	2	1	3	1	Hit Ratio
LRU	*a*	0	0	0	4	4	4	7	7	7	3	3	
	b		1	1	1	1	3	3	3	1	1	1	$\frac{3}{11}$
	c			2	2	2	2	2	2	2	2	2	
	Faults	*	*	*	*		*	*		*	*		
OPT	*a*	0	0	0	4	4	3	7	7	7	3	3	
	b		1	1	1	1	1	1	1	1	1	1	$\frac{4}{11}$
	c			2	2	2	2	2	2	2	2	2	
	Fault	*	*	*	*		*	*			*		
FIFO	*a*	0	0	0	4	4	4	4	2	2	2	2	
	b		1	1	1	1	3	3	1	1	1	1	$\frac{2}{11}$
	c			2	2	2	2	7	7	7	3	3	
	Faults	*	*	*	*		*	*	*	*	*		