MODULE II

Processors and memory hierarchy – Advanced processor technology- Design Space of processors, Instruction Set Architectures, CISC Scalar Processors, RISC Scalar Processors, Superscalar and vector processors, Memory hierarchy technology.

2.1 : ADVANCED PROCESSOR TECHNOLOGY

Architectural families of modern processors are introduced here. Major processor families to be studied include the *CISC*, *RISC*, *superscalar*, *VLIW*, *superpipelined*, *vector*, and *symbolic* processors. Scalar and vector processors are for numerical computations. Symbolic processors have been developed for AI applications.

Qn:Explain design space of processor?

2.1.1 Design Space of Processors

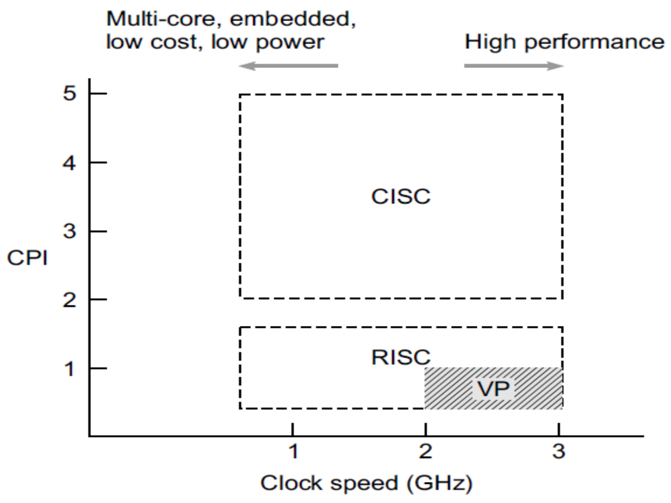


Fig. 4.1 CPI versus processor clock speed of major categories of processors

- Processor families can be mapped onto a coordinated space of clock *rate* versus *cycles per instruction* (CPI), as illustrated in Fig. 4.1.

  - As implementation technology evolves rapidly, the **clock rates** of various processors have moved from low to higher speeds toward the right of the design space **(ie increase in clock rate).** and processor manufacturers have been trying to **lower the CPI rate**(cycles taken to execute an instruction) using innovative hardware approaches.

  - Two main categories of processors are:-

    - **CISC** (eg:X86 architecture)
    - **RISC(**e.g. Power series, SPARC, MIPS, etc.) **.**

  - Under both CISC and RISC categories, products designed for multi-core chips, embedded applications, or for low cost and/or 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 processors has been marked VP; vector processing features may be associated with CISC or RISC main processors.

Qn:Compare CISC ,RISC, Superscalar and VLIW processors on the basis of design space?

Design space of CISC ,RISC, Superscalar and VLIW processors

- The CPI of different **CISC instructions** varies from **1 to 20**. Therefore, CISC processors are at the upper part of the design space. With advanced implementation techniques, the clock rate of today's CISC processors ranges up to a few GHz.

- With efficient use of pipelines, the average CPI of **RISC instructions** has 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. Thus the **effective CPI of a superscalar processor should be lower than that of a scalar RISC processor**. The **clock rate** of superscalar processors matches that of scalar RISC processors.

- **The *very long instruction word* (VLIW) architecture** can in theory use even more functional units than a superscalar processor. Thus the **CPI of a VLIW processor can be further lowered**. Intel's i860 RISC processor had VLIW architecture.

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.**

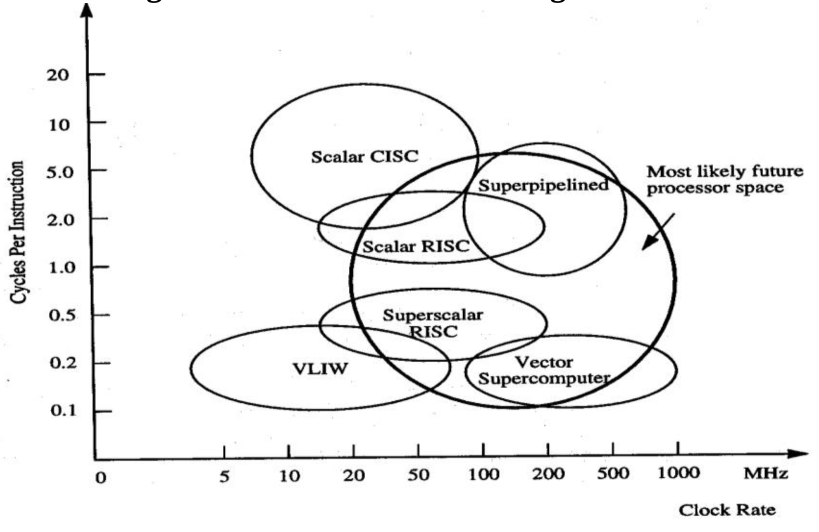


Figure 4.1 Design space of modern processor families.

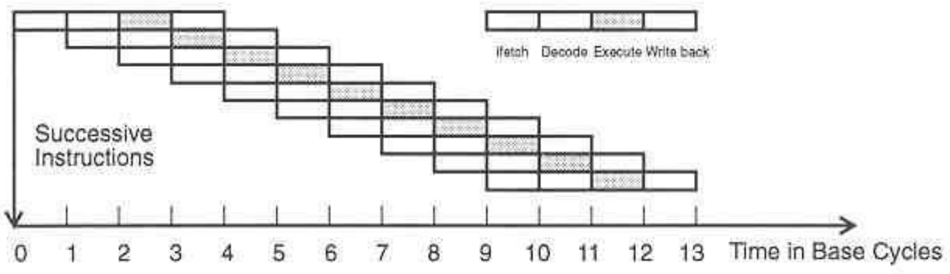
Instruction Pipelines

Qn:Explain the execution of instructions in base scalar and underpiprelined processors?

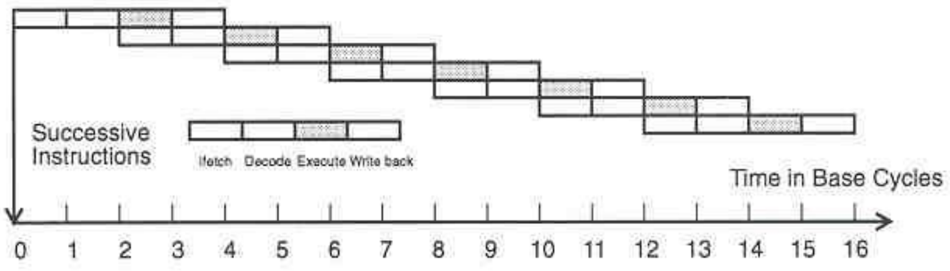
- Typical instruction includes four phases:
  - fetch

- o decode
- o execute
- o write-back

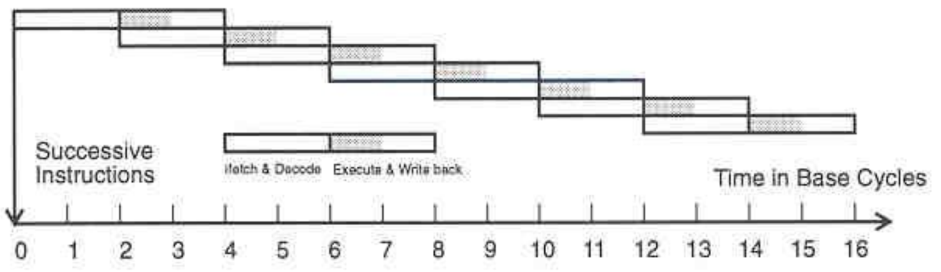
- These four phases are frequently performed in a pipeline, or "assembly line" manner, as illustrated on the figure below.



(a) Execution in a base scalar processor



(b) Underpipelined with two cycles per instruction issue



(c) Underpipelined with twice the base cycle

Figure 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)

Qn:Define the following g terms related to modern processor technology: a: Instruction issue latency b) Simple operation latency c) Instruction issue rate?

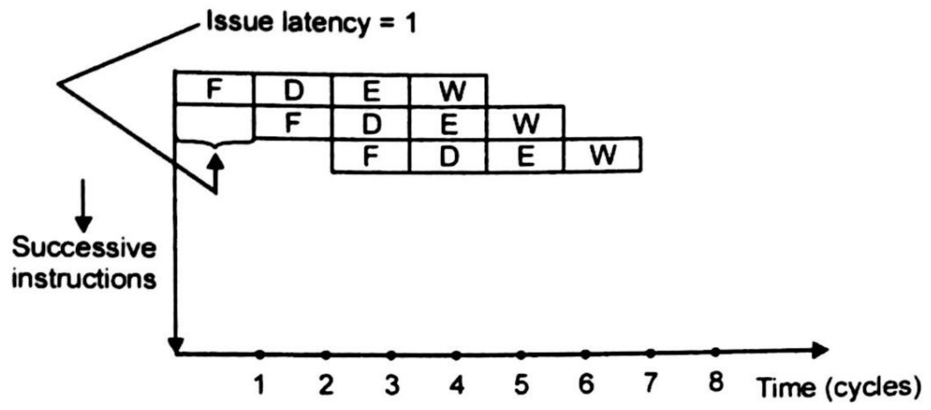
Pipeline Definitions

- **Instruction pipeline cycle** – the time required for each phase to complete its operation (assuming equal delay in all phases)
- **Instruction issue latency** – the time (in cycles) required between the issuing of two adjacent instructions
- **Instruction issue rate** – the number of instructions issued per cycle (the *degree* of a superscalar)

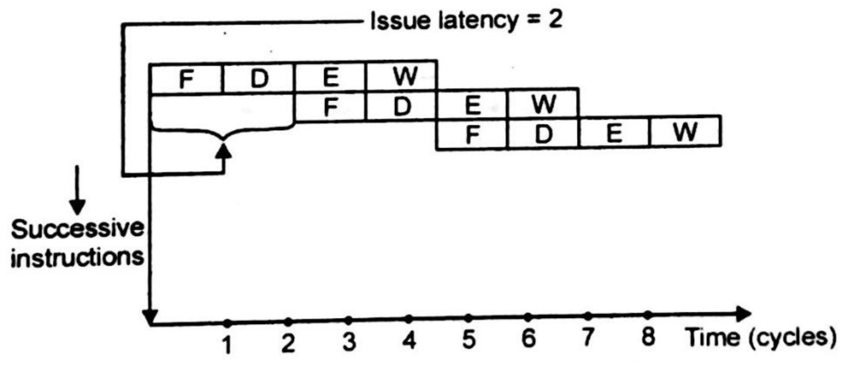
- **Simple operation latency** – the delay (after the previous instruction) associated with the completion of a simple operation (e.g. integer add) as compared with that of a complex operation (e.g. divide).
- **Resource conflicts** – when two or more instructions demand use of the same functional unit(s) at the same time.

Pipelined Processors

- **Case 1 : Execution in base scalar processor -**
  - A base scalar processor, as shown in Fig. 4.2a and below. :
    - issues one instruction per cycle
    - has a one-cycle latency for a simple operation
    - has a one-cycle latency between instruction issues
    - can be fully utilized if instructions can enter the pipeline at a rate on one per cycle

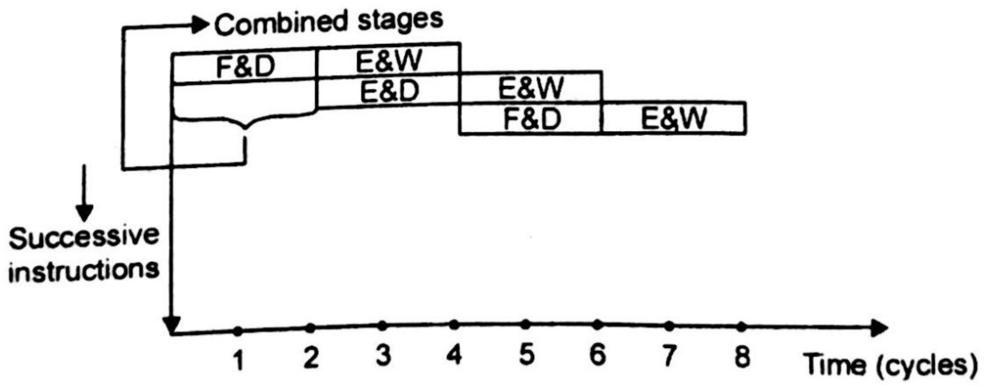


- **CASE 2 :** If the instruction **issue latency is two cycles per instruction**, the pipeline can be underutilized, as demonstrated in Fig. 4.2b and below:
  - **Pipeline Underutilization** – ex : issue latency of 2 between two instructions. – effective CPI is 2.



- **CASE 3 : Poor Pipeline utilization** – Fig. 4.2c and below:-, 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.**

- combines two pipeline stages into one stage – here the effective CPI is ½ only



- **The effective CPI rating is 1 for the ideal pipeline in Fig. 4.2a,** and **2** for the case in Fig. 4.2b. In Fig. 4.2c, the clock rate of the pipeline has been lowered by **one-half.**

- Underpipelined systems will have higher CPI ratings, lower clock rates, or both.

Qn:Draw and explain datapath architecture and control unit of a scalar processor?

Data path architecture and control unit of a scalar processor

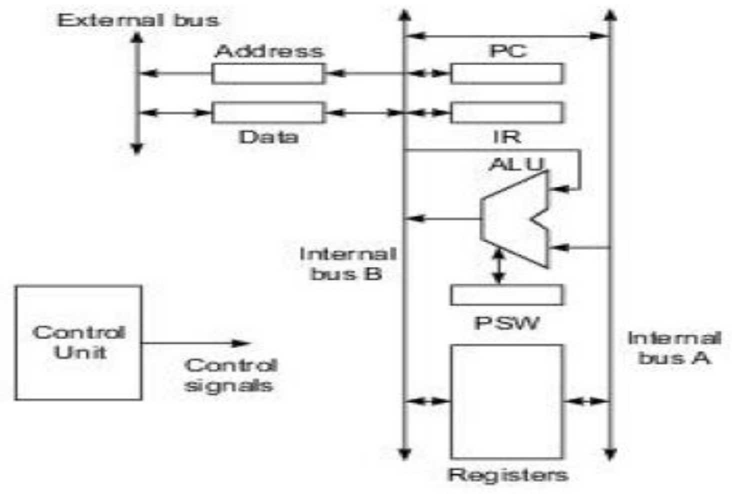


Fig. 4.3 Data path architecture and control unit of a scalar processor

- The data path architecture and control unit of a typical, simple scalar processor **which does not employ an instruction pipeline** is shown above.
- 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*,a nd *write result* phases of instruction execution.
- The control unit itself may employ hardwired logic, or—as was more common in older CISC style processors—microcoded logic.
- Modern RISC processors employ hardwired logic, and even modern CISC processors make use of many of the techniques originally developed for high-performance RISC processors.

2.1.2 Instruction-Set Architectures

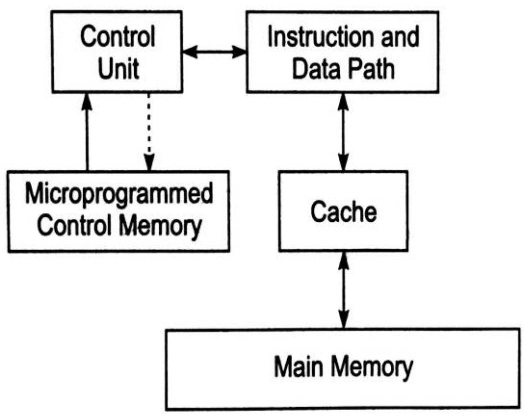
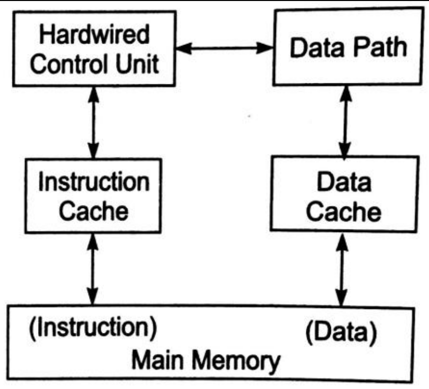
Qn:Distinguish between typical RISC and CISC processor architectures?

Qn:Compare ISA in RISC and CISC processors in terms of instruction formats, addressing modes and cycles per instruction?

Qn:List out the advantages and disadvantages of RISC and CISC 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, opcode specifications, and flow control mechanisms used.
- **ISA Broadly classified into 2:**
    - **CISC**
    - **RISC**
  - A computer with large number of instructions is called complex instruction set computer**(CISC)**
  - A computer that uses a few instructions with simple constructs is called Reduced Instruction set computers (**RISC**). These instructions can be executed at a faster rate.

S.No	**CISC**	**RISC**
1	Large set of instructions with variable format (16-64 bits per instr)	Small set of instructions with fixed (32 bit) format, mostly register based
2	12-24 addressing modes	3-5 addressing modes
3	8-24 general purpose registers	Large no of general purpose registers (32-195)
4	Have a unified cache for holding both instr and data	Use separate instruction and data cache
5	CPI btw 2 and 15	Avg CPI <1.5
6	CPU control is Earlier **Microcoded** using control memory(ROM) ,but modern CISC also uses hardwired control.	Hardwired control logic
7	Clock rates btw 33-50 MHz	Clock rate btw 50-150 MHz
8	Ex: 8086, 80386	Ex: SUN SPARC, ARM
9	**Architectural Distinctions:**	**Architectural Distinctions:**

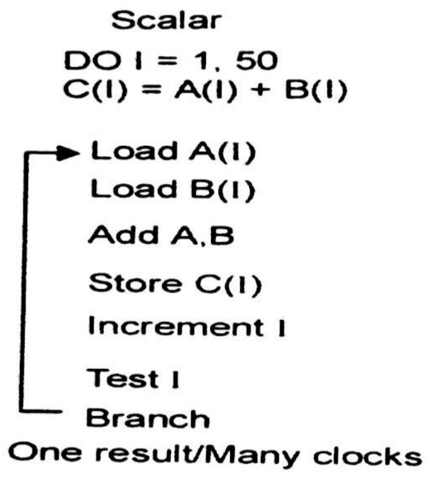
	 Control Unit Instruction and Data Path Microprogrammed Control Memory Cache Main Memory The CISC architecture with microprogrammed control and unified cache	 Hardwired Control Unit Data Path Instruction Cache Data Cache (Instruction) (Data) Main Memory The RISC architecture with hardwired control and split instruction cache and data cache

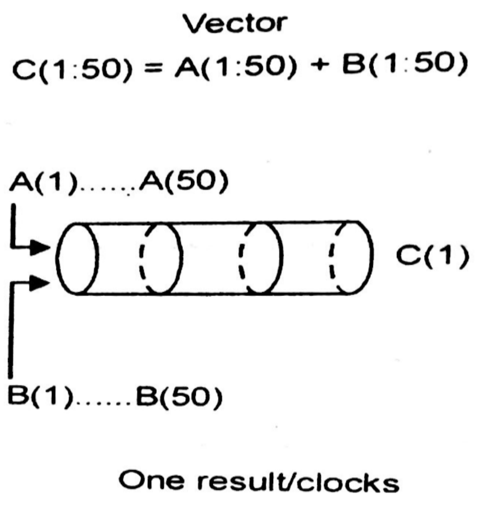
NOTE:**MC68040 and i586** are examples of **CISC processors** which uses split caches and hardwired control for reducing the CPI.(some CISC processor can also use split caches and hardwired control.

- **CISC Advantages**
  - **Smaller program size (fewer instructions)**
  - **Simpler control unit design**
  - **Simpler compiler design**
- **RISC Advantages**
  - **Has potential to be faster**
  - **Many more registers**
- **RISC Problems**
  - **More complicated register decoding system**
  - **Hardwired control is less flexible than microcode**

Qn:Differentiate between scalar processor and Vector processors?

Difference between Scalar and Vector processor

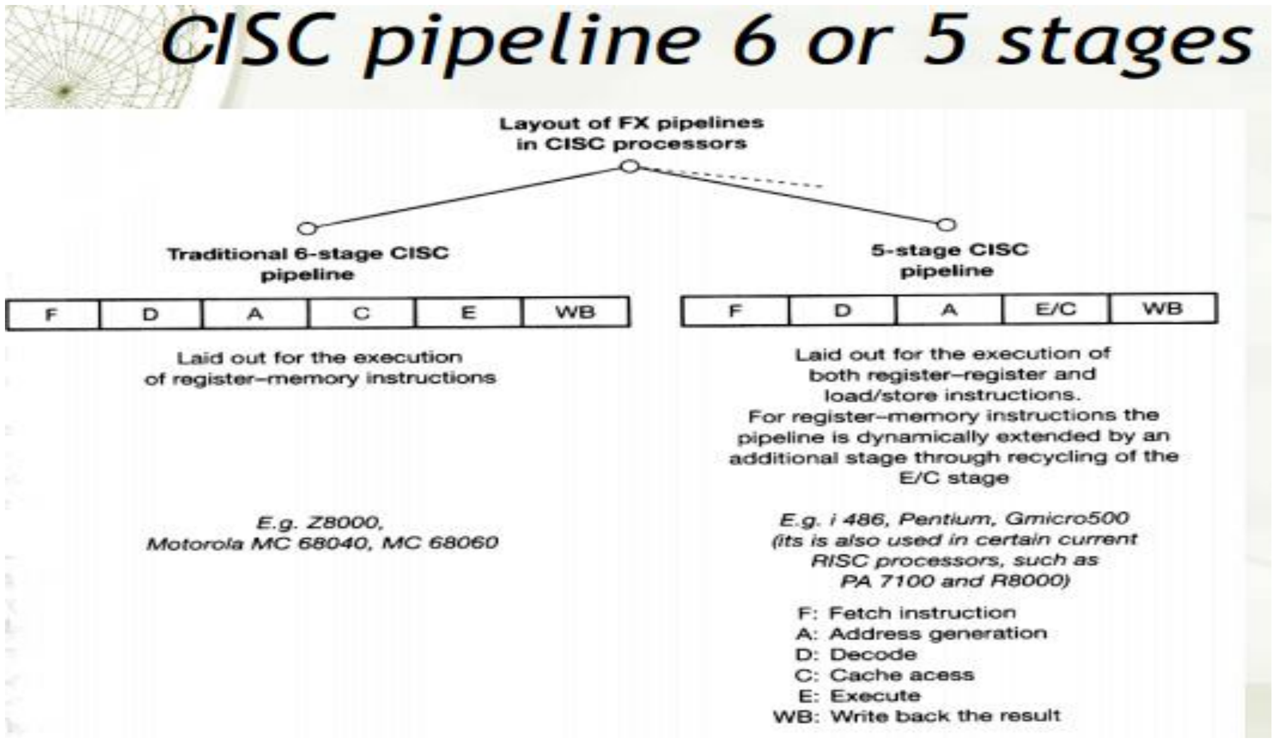
Scalar Processor	Vector Processor
1. One result/many clock cycles is produced	1. One result/clock cycle is produced
2.  Scalar DO I = 1, 50 C(I) = A(I) + B(I) Load A(I) Load B(I) Add A,B Store C(I) Increment I Test I Branch One result/Many clocks	2.

	 Vector
	C(1:50) = A(1:50) + B(1:50)
	A(1)......A(50)
	C(1)
	B(1)......B(50)
	One result/clocks

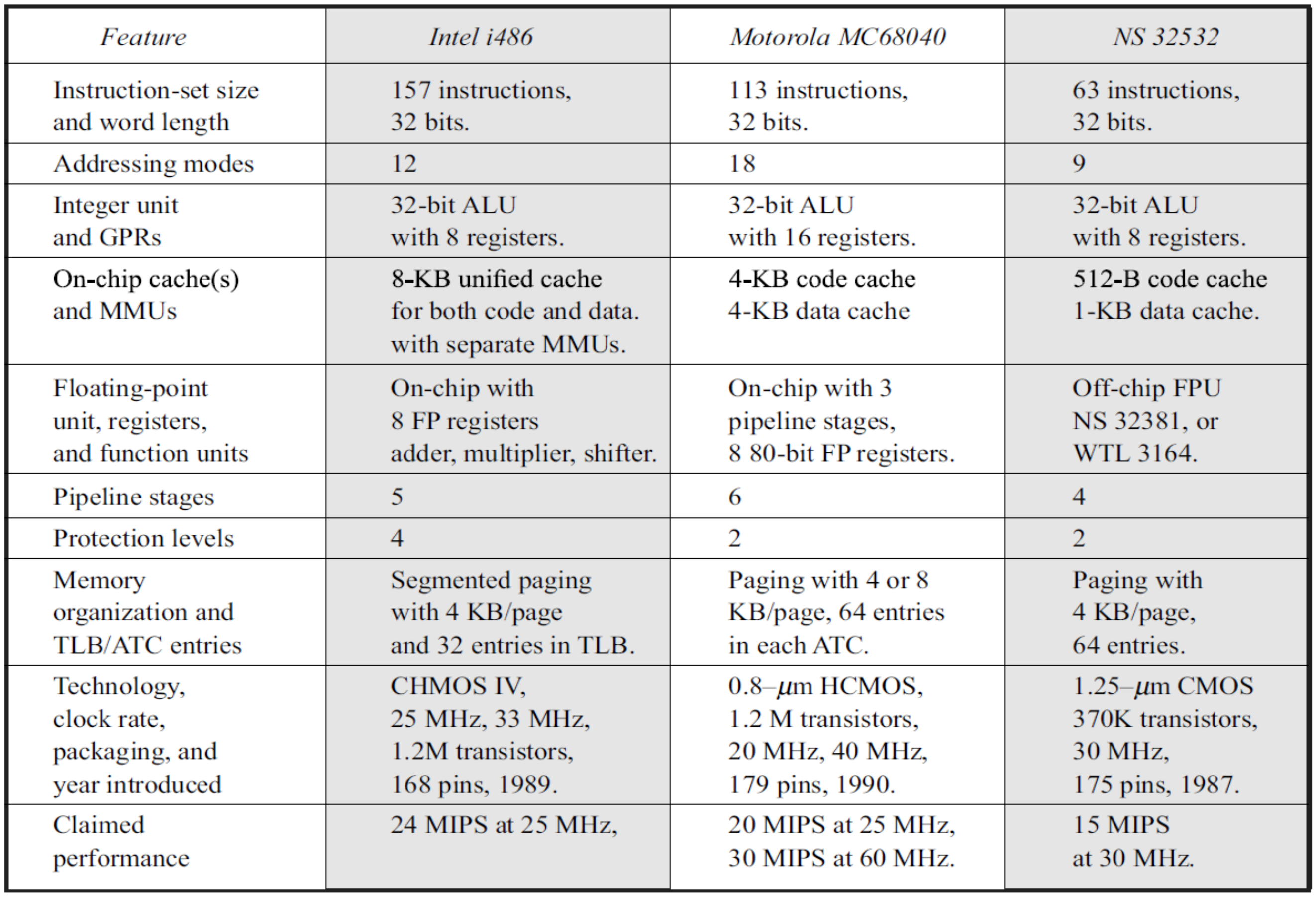
2.1.3 CISC SCALAR PROCESSORS

- Executes scalar data.
- Executes integer and fixed point operations.
- Modern scalar processors executes both integer and floating-point unit and even multiple such units.
- Based on a complex instruction set, a *CISC scalar processor* can also use pipelined design.
- Processor may be **underpipelined** due to data dependence among instructions, resource conflicts, branch penalties and logic hazards.
-

Qn:Expalin 5 or 6 stage pipeline of CISC processors?


CISC pipeline 6 or 5 stages
Layout of FX pipelines in CISC processors
Traditional 6-stage CISC pipeline
F
D
A
C
E
WB
Laid out for the execution of register–memory instructions
E.g. Z8000, Motorola MC 68040, MC 68060
5-stage CISC pipeline
F
D
A
E/C
WB
Laid out for the execution of both register–register and load/store instructions. For register–memory instructions the pipeline is dynamically extended by an additional stage through recycling of the E/C stage
E.g. i 486, Pentium, Gmicro500 (its is also used in certain current RISC processors, such as PA 7100 and R8000)
F: Fetch instruction
A: Address generation
D: Decode
C: Cache acess
E: Execute
WB: Write back the result

CISC Processor examples:



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.

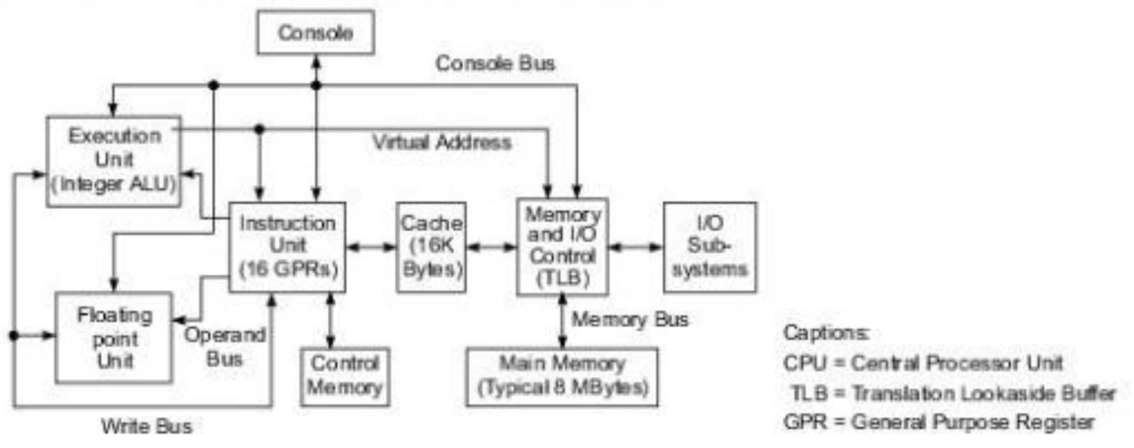
CISC Microprocessor Families:* *widely used in Personal computers industry

INTEL: *4-bit* Intel 4004
8-bit –Intel 8008, 8080, and 8085.
16-bit - 8086, 8088, 80186, and 80286.
32 bit-, the 80386
The 80486 and Pentium are the latest 32-bit processors in the Intel 80x86 family.

MOTOROLA: 8 –bit MC6800
16 bit MC68000
32 bit MC68020, MC68030, MC68040.

National Semiconductor’s:32 bit –NS32532

TYPICAL CISC PROCESSOR ARCHITECTURE- VAX 8600 Processor architecture


Console
Console Bus
Virtual Address
Execution Unit (Integer ALU)
Instruction Unit (16 GPRs)
Cache (16K Bytes)
Memory and I/O Control (TLB)
I/O Sub-systems
Floating point Unit
Operand Bus
Control Memory
Memory Bus
Main Memory (Typical 8 MBytes)
Write Bus
Captions:
CPU = Central Processor Unit
TLB = Translation Lookaside Buffer
GPR = General Purpose Register

- VAX 8600 processor uses typical CISC architecture with **microprogrammed control**.
- The **instruction set** contained about **300 instructions** with **20** different **addressing modes**.
- The CPU in the VAX 8600 consisted of **two functional units** for concurrent execution of **integer and floating-point instructions.**
- The **unified cache** was used for holding both instructions and data.
- There were **16 GPRs** in the instruction unit.
- **Instruction pipelining** was built with six stages in the VAX 8600.
- The **instruction unit prefetched and decoded instructions**, handled branching operations, and **supplied operands to the two functional units** in a pipelined fashion.
- A **translation lookaside buffer [TLB)** was used in the memory control unit **for fast generation of a physical address from a virtual address.**
- Both integer and floating-point units were pipelined.
- The **CPI of VAX 8600** instruction varied from **2 to 20** cycles. Because both multiply and divide instructions needs execution units for a large number of cycles.

The Motorola MC68040 microprocessor architecture

Figure below shows the MC68040 architecture. Features are listed in the above table.

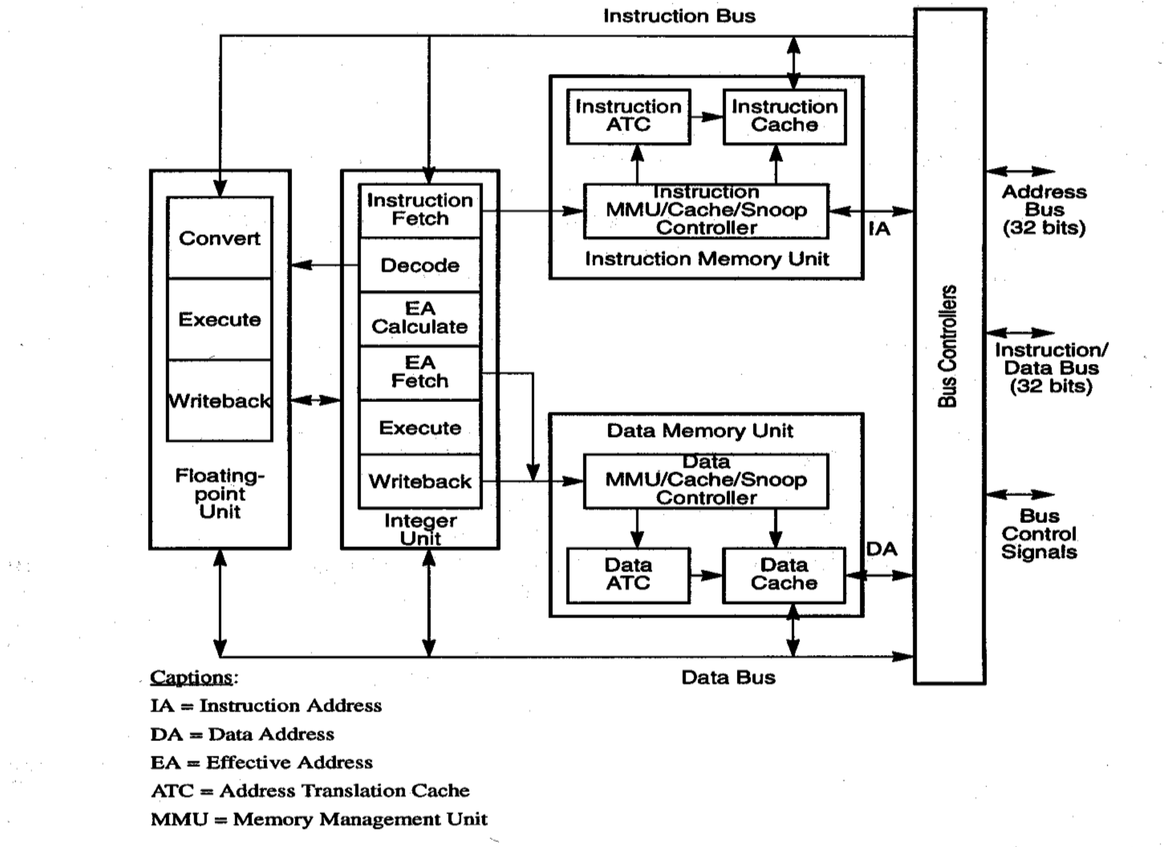


Figure 4.6 Architecture of the MC68040 processor. (Courtesy of Motorola Inc., 1991)

The architecture has involved

- Separate **instruction and data memory unit**, with a 4-Kbyte data cache, and a 4-Kbyte instruction cache, with separate ***memory management units*** **(MMUs)** supported by an ***address translation cache*** **(ATC),** equivalent to the TLB used in other systems.

- The processor implements 113 instructions using 16 general-purpose registers.

- **18-Addressing modes includes**:- register direct and indirect, indexing, memory indirect, program counter indirect, absolute, and immediate modes.

- The **instruction set** includes data movement, integer, BCD, and floating point arithmetic, logical, shifting, bit-field manipulation, cache maintenance, and multiprocessor communications, in addition to program and system control and memory management instructions

- 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 from memory units, respectively. **Dual MMUs allow interleaved fetch of instructions and data from the main memory.**

- 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.

- Each of the two **ATCs has 64 entries** providing fast translation from virtual address to physical address.

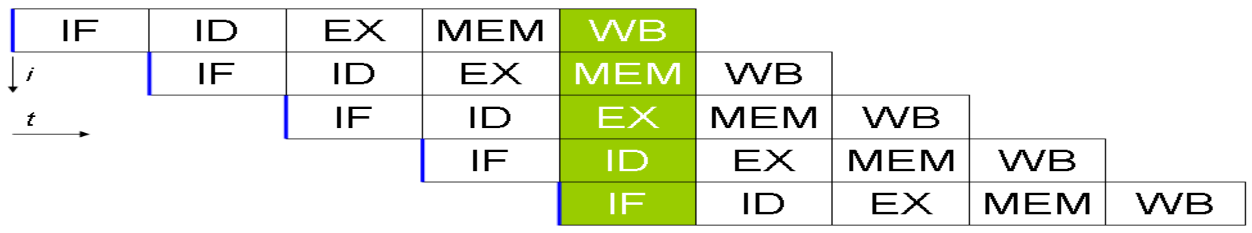
2.1.4 <u>RISC SCALAR PROCESSORS</u>

Qn:Explain the relationship between the integer unit and floating point unit in most RISC processor with scalar organization?

- Generic RISC processors are called *scalar* RISC because they are designed to issue one instruction per cycle, similar to the base scalar processor
- Simpler: - RISC design Gains power by pushing some less frequently used operations into software.
- Needs a good compiler when compared to CISC processor.
- Instruction-level parallelism is exploited by pipelining

Qn:Expalin classic 5 stage pipeline of RISC processors?

RISC Pipelines



Basic five-stage pipeline in a RISC machine (IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back). The vertical axis is successive instructions; the horizontal axis is time. So in the green column, the earliest instruction is in WB stage, and the latest instruction is undergoing instruction fetch.

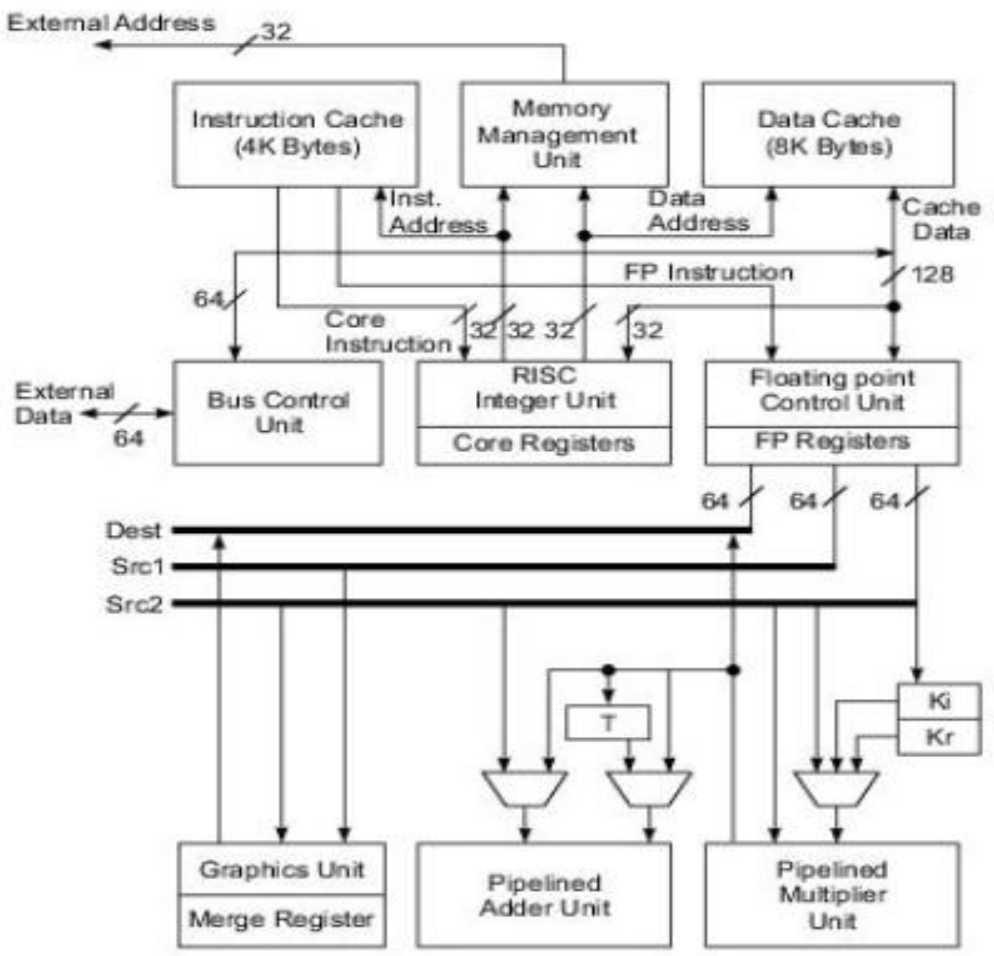
RISC Processor examples:

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.

<u>The Intel i860 Processor Architecture - RISC</u>

- It was a **64-bit RISC processor** fabricated on a single chip containing more than l million transistors.
- The peak performance of the i860 was designed to reach 80 Mflops single-precision or 60 Mflops double-precision, or 40 MIPS in 32-bit integer operations at a 40-MHz clock rate.
- In the block diagram there were **nine functional units** (shown in 9 boxes) interconnected by multiple data paths with widths ranging from 32 to 128 bits.



- All **external or internal address buses were 32-bit wide**, and the **external data path or internal data bus was 64 bits wid**e. However, the internal RISC integer ALU was only 32 bits wide.
- The **instruction cache had 4 Kbytes** organized as a two-way set-associative memory with 32 bytes per cache block. lt transferred 64 bits per clock cycle, equivalent to 320 Mbytes/s at 40 MHz.
- The **data cache** was a two-way set associative memory of **8 Kbytes**. lt transferred 128 bits per clock cycle (640 Mbytes/s) at 40 MHZ .
- The **bus control unit** coordinated the **64-bit data transfer between the chip and the outside world**.
- The **MMU** implemented protected **4 Kbyte** paged virtual memory of 2^32 bytes via a TLB .
- The **RISC integer unit executed load, store. Integer , bit, and control instructions** and **fetched instructions for the floating-point control** unit as well.
- 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 .
- The **graphics unit** supported **three-dimensional drawing in a graphics frame buffer**, with color intensity, shading, and hidden surface elimination.

- The **merge register** was used only by vector integer instructions. This register **accumulated the results of multiple addition operations .**

2.2: SUPERSCALAR AND VECTOR PROCESSORS

Qn: Describe the structure of a superscalar pipeline in detail.Discuss the various instruction issue and completion policies in superscalar processors?

A CISC or a RISC scalar processor can be improved with **a *superscalar* or *vector*** architecture. **Scalar processors** are those executing one instruction per cycle. Only one instruction is issued per cycle, and only one completion of instruction is expected from the pipeline per cycle.

In a superscalar processor, multiple instructions are issued per cycle and multiple results are generated per cycle.

A vector processor executes vector instructions on arrays of data; each vector instruction involves a string of repeated operations, which are ideal for pipelining with one result per cycle.

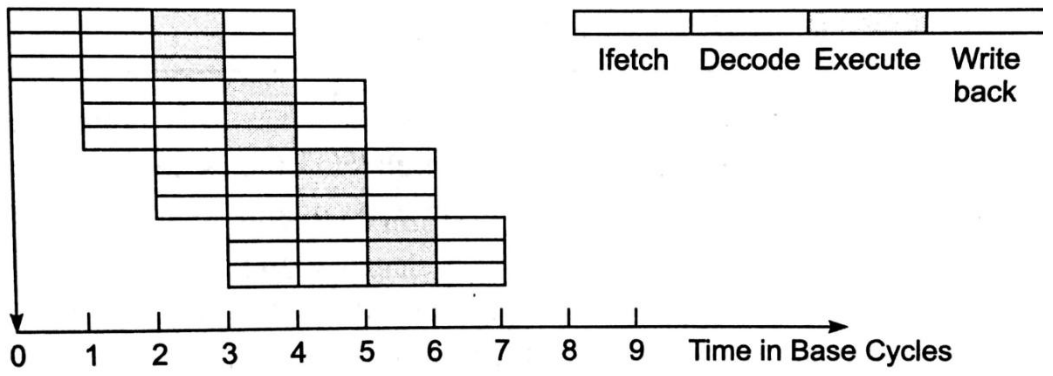
2.2.1 SUPERSCALAR PROCESSORS

Qn:explain pipelining in superscalar procesors?

- Designed to exploit instruction level parallelism in user pgms
- Independent instr's can be executed in parallel.
- A superscalar processor of degree m can issue m instructions per cycle.
- Superscalar processors were originally developed as an **alternative to vector processors**, with a view to **exploit higher degree of instruction level parallelism.**

<u>Pipelining in Superscalar Processors</u>

- The fig shows a three-issue (m=3) superscalar pipeline, m instructions execute in parallel.

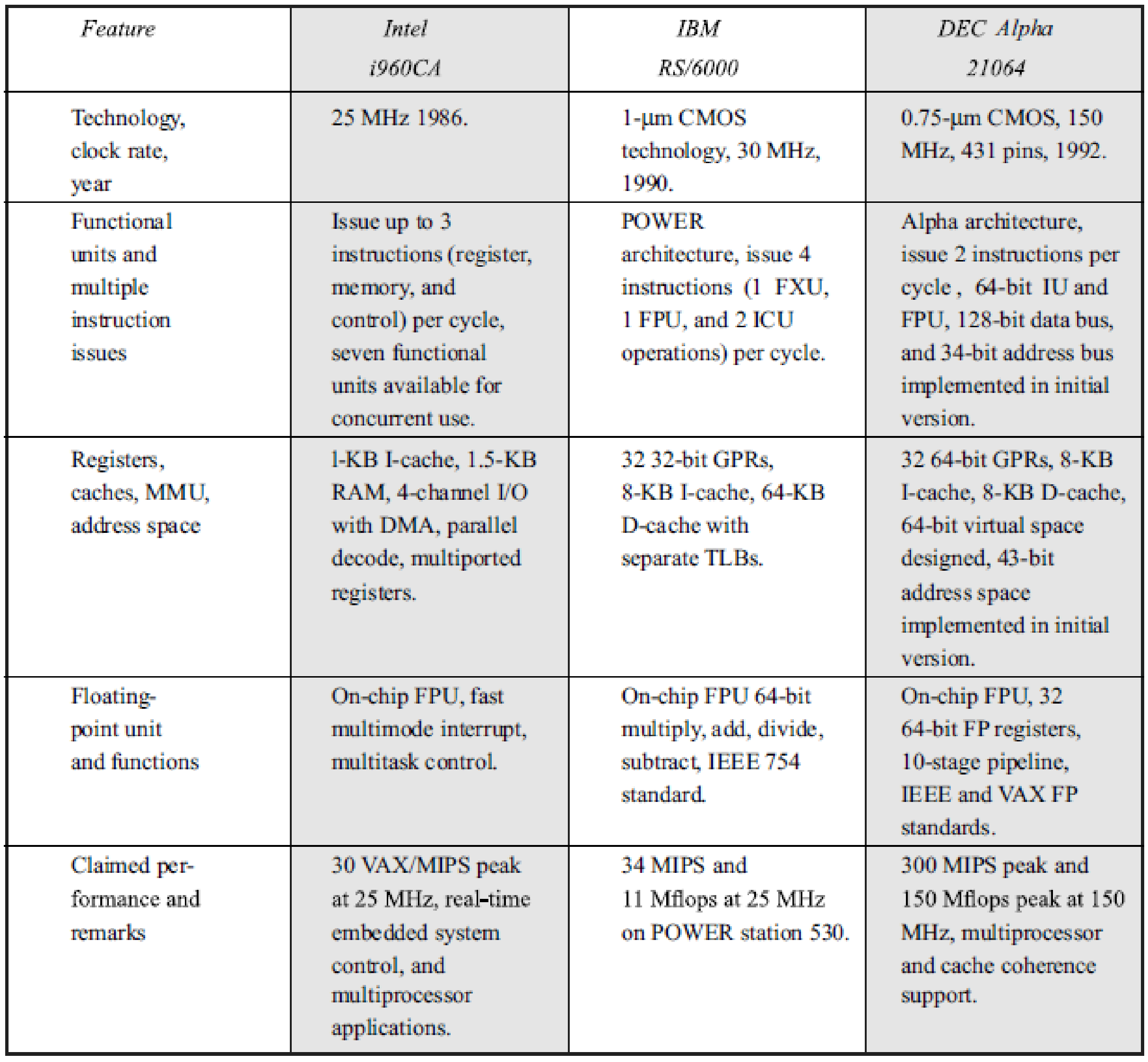


A superscalar processor of degree m = 3

- Instruction issue degree (m) in a superscalar **processors is limited btw 2 to 5.(2<m<5)**
- A superscalar processor of degree m can issue m instructions per cycle. In this sense, the base scalar processor, implemented either in RISC or CISC, has m = 1.

- In order to fully utilize a superscalar processor of degree m, m instructions must be executable in parallel.
- This situation may not be true in all clock cycles. In that case, some of the pipelines may be stalling in a wait state.
- In a superscalar processor, the simple operation latency should require only one cycle, as in the base scalar processor.
- **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**.

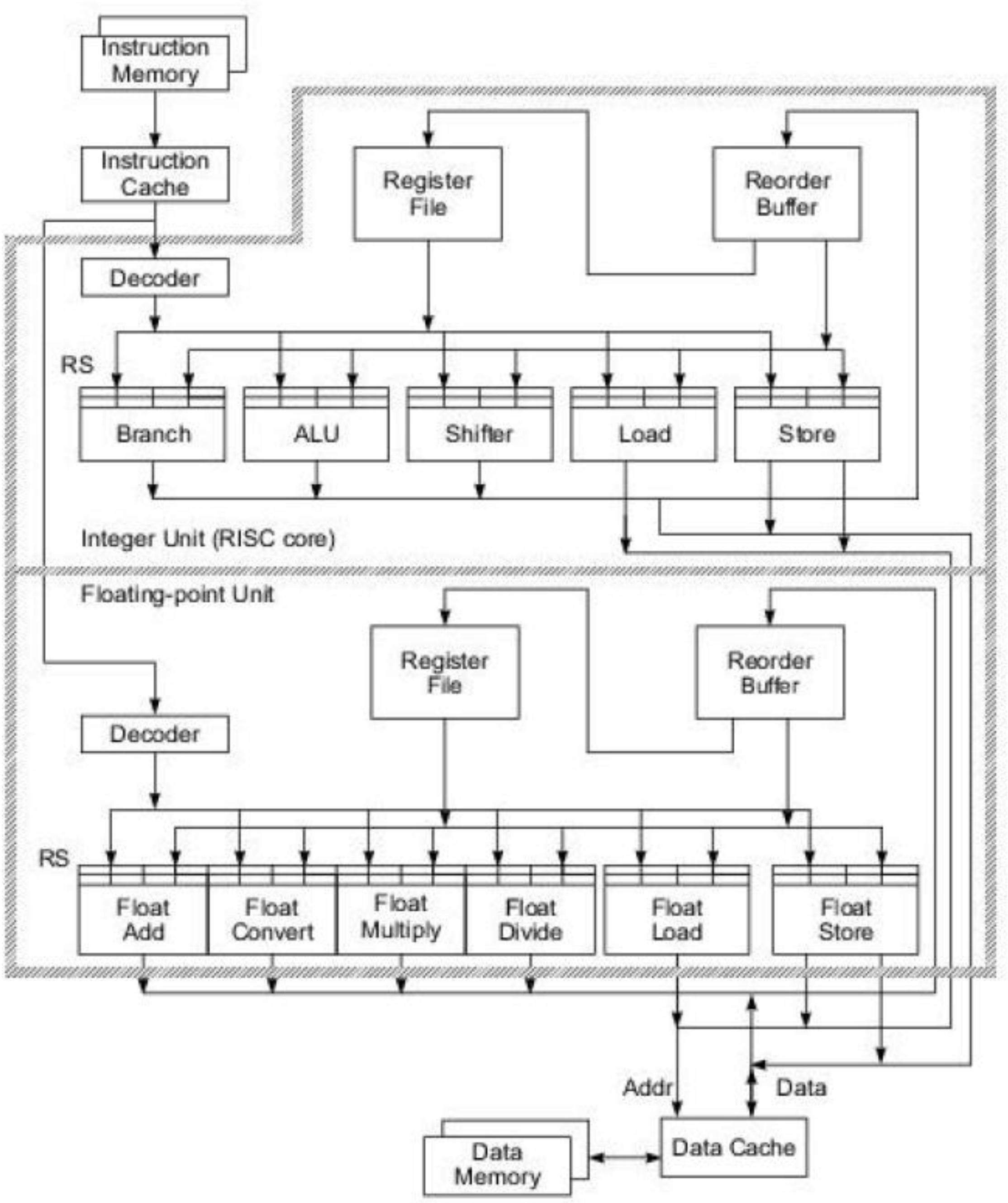
- Table below lists some landmark examples of superscalar processors from the early 1990s.



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	1-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 performance 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.

A typical superscalar architecture for a RISC processor:



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.)

- The **instruction cache** supplies multiple instructions per fetch. However, 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. In theory, all functional units can be simultaneously used if conflicts and dependences do not exist among them during a given cycle

- The maximum number of instructions issued per cycle **ranges from two to five** in these superscalar processors.

- Typically. the **register files** in the lU and FPU each have 32 registers. Most superscalar processors implement both the IU and the FPU on the same chip.

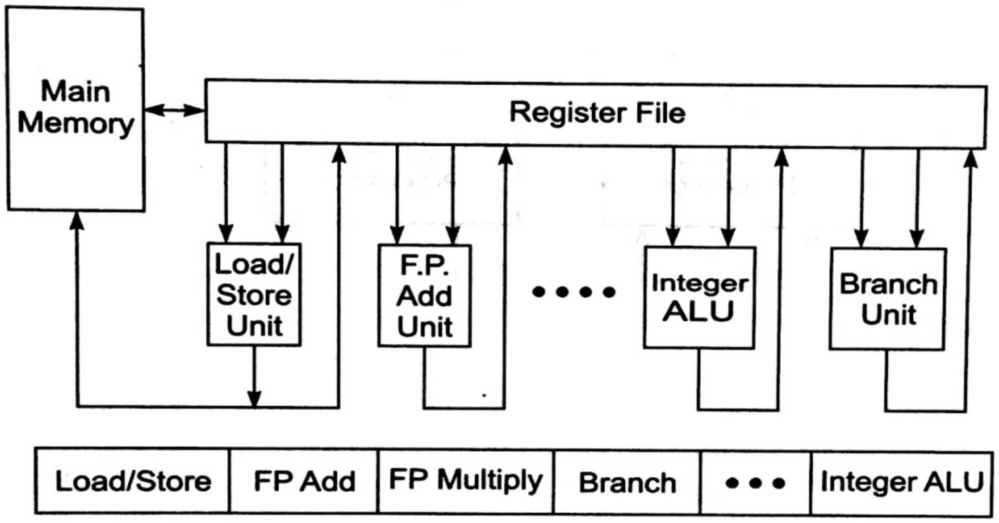
- The superscalar degree is low due to limited instruction parallelism that can be exploited in ordinary programs.

Qn:Why are Reservation stations and reorder buffers are needed in a super scalar processor?

- Besides the register files, **reservation stations and reorder buffers** can be used to establish instruction *windows*. The purpose is to **support instruction lookahead and internal data forwarding**, which are needed to schedule multiple instructions simultaneously.

2.2.2 **VLIW Architecture** (Very Long Instruction Word)(**VLIW processor as an extreme example of a superscalar processor)**

- Horizontal micro-coding + superscalar processing = VLIW architecture
- VLIW machine has instruction words hundreds of bits in length



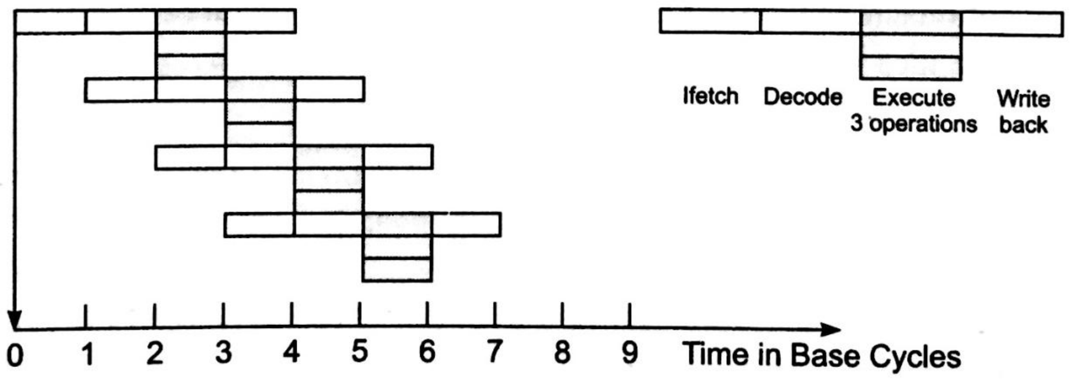
A typical VLIW processor with degree *m* = 3

- As shown above, Multiple functional units are use concurrently in a VLIW processor.
- All functional units share a common large register file.
- The operations to be simultaneously executed by functional units are synchronized in a VLIW instruction. ,say 256 or 1024 bits per instruction word.
- Different fields of the long instruction word carry the opcodes to be dispatched to different functional units.
- Programs written in short instruction words (32 bits) must be compacted together to form the VLIW instructions – **the code compaction must be done by compiler.**

Qn:Explain pipelining in VLIW processors?

<u>Pipelining in VLIW Processor</u> –

- The execution of instructions by an ideal VLIW processor is shownbelow: each instruction specifies multiple operations. The effective CPI becomes 0.33 in this example.



VLIW execution with degree *m* = 3

- VLIW machines behave much like **superscalar machines** with **three differences**:

  1. **The decoding of VLIW instructions is easier** than that of superscalar instructions.

  2. **The code density of the superscalar machine is better** when the available instruction-level parallelism is less than that exploitable by the VLIW machine. This is because the fixed VLIW format includes bits for non-executable operations, while the superscalar processor issues only executable instructions.

  3. A superscalar machine can be object-code-compatible with a large family of non-parallel machines. On the contrary, VLlW machine exploiting different amounts of parallelism would require different instruction sets.

- lnstruction parallelism and data movement in a VLIW architecture are completely **specified at compile time.** Run-time resource scheduling and synchronization are in theory completely eliminated.

- One can view a **VLIW processor as an extreme example of a superscalar processor** in which all independent or unrelated operations are already synchronously compacted together in advance.

- The CPI of a VLIW processor can be even lower than that of a superscalar processor. For example, the Multiflow trace computer allows up to seven operations to be executed concurrently with 256 bits per VLIW instruction.

Qn:Explain the difference between superscalar and VLIW architectures in terms of hardware and software requirements?

Comparison between Superscalar and VLIW

Superscalar	VLIW
1. Code size is smaller	**1.** code size is larger
2. Complex hardware for decoding and issuing instruction	**2.** simple hardware for decoding and issuing
3. Compatible across generations	**3.** not compactable across generations.
4. No change in hardware is required	**4.** Requires more registers but simplified hardware
5. They are scheduled dynamically by processor	**5.** Scheduled dynamically by compiler.

Application – VLIW processors useful for special purpose DSP(digital signal processing) ,and scientific application that requires high performance and low cost. But they are less successful as General purpose computers. 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

- Vector processor is specially designed to perform vector computations, vector instruction involves large array of operands.(same operation will be performed over an array of data)
- Vector processors can be **register-to-register** architecture (use shorter instructions and vector register files) or **memory-to-memory** architecture (use longer instructions including memory address).

Vector Instructions

Qn:List out register based and memory based vector operations?

- Register-based vector instructions appear in most register-to-register vector processors like Cray supercomputers.

- We denote vector register of length n as V_1, a scalar register as s_i ,a memory array of length n as M(1 : n). operator denoted by a small circle ‘o’.

- **Typical register based vector operations are**:

$$V_1 \circ V_2 \rightarrow V_3 \quad \text{(binary vector)}$$
$$s_1 \circ V_1 \rightarrow V_2 \quad \text{(scaling)}$$
$$V_1 \circ V_2 \rightarrow s_1 \quad \text{(binary reduction)}$$

$$
\begin{array}{llll}
 & M(1:n) \rightarrow & \mathbf{V_1} & \text{(vector load)} \\
 & \mathbf{V_1} \rightarrow & M(1:n) & \text{(vector store)} \\
\text{o} & \mathbf{V_1} \rightarrow & \mathbf{V_2} & \text{(unary vector)} \\
\text{o} & \mathbf{V_1} \rightarrow & s_1 & \text{(unary reduction)}
\end{array}
$$

- **Vector length should be equal** in the two operands used in **binary vector instruction**.

- **The reduction** is an operation on **one or two vector operands**, and the result is a **scalar**—such as the dot product between two vectors and the maximum of all components in a vector.

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

- Long vectors exceeding the register length n must be segmented to fit the vector registers n elements at a time.

- **Memory based Vector operations:**

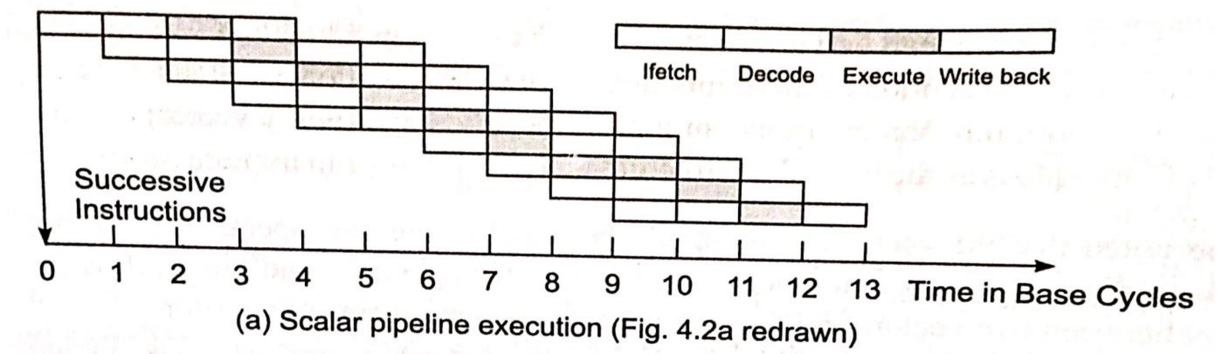
$$
\begin{array}{rcccl}
M_1(1:n) & \text{o} & M_2(1:n) & \rightarrow & M(1:n) \\
s_1 & \text{o} & M_1(1:n) & \rightarrow & M_2(1:n) \\
 & \text{o} & M_1(1:n) & \rightarrow & M_2(1:n) \\
M_1(1:n) & \text{o} & M_2(1:n) & \rightarrow & M(k)
\end{array}
$$

where M1(1 : n) and M2(1 : n) are two vectors of length n and M(k) denotes a scalar quantity stored in memory location k. Note that the vector length is not restricted by register length. Long vectors are handled in a streaming fashion using super words cascaded from many shorter memory words.

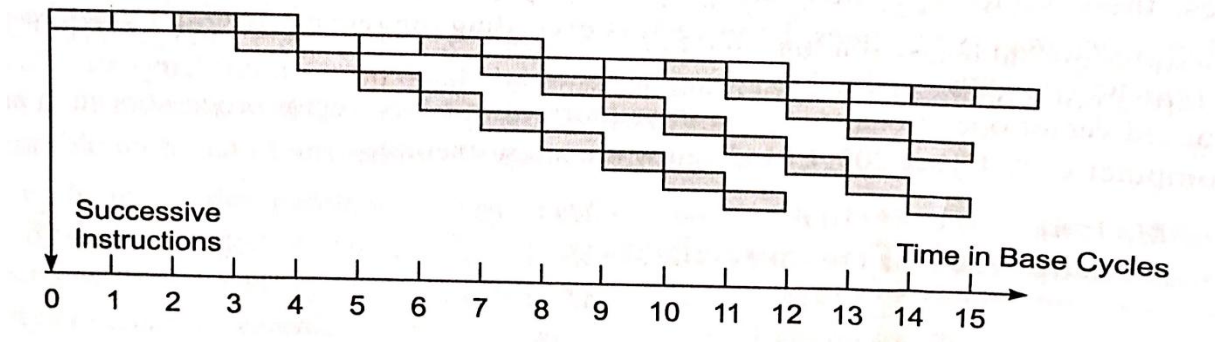
Vector Pipelines

Qn:Compare scalar and vector pipeline execution?

The pipelined execution of a Vector processor compared to a scalar processor (fig below). **Scalar instruction executes only one operation** over one data element whereas each **vector instruction executes a string of operations**, one for each element in the vector.



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



(b) Vector pipeline execution

SYMBOLIC PROCESSORS

Qn:Explain the characteristics of symbolic processors?

- Applied in areas like – theorem proving, pattern recognition, expert systems, machine intelligence etc because in these applications data and knowledge representations, operations, memory, I/o and communication features are different than in numerical computing.
- Also called **Prolog processor, Lisp processor or symbolic manipulators**.

Characteristics of Symbolic processing

Attributes	Characteristics
Knowledge representation	Lists, relational databases, Semantic nets, Frames, Production systems
Common operations	Search, sort, pattern matching, unification
Memory requirement	Large memory with intensive access pattern
Communication pattern	Message traffic varies in size, destination and format
Properties of algorithm	Parallel and distributed, irregular in pattern
Input / Output requirements	Graphical/audio/keyboard. User guided programs, machine interface.
Architecture Features	Parallel update, dynamic load balancing and memory allocation

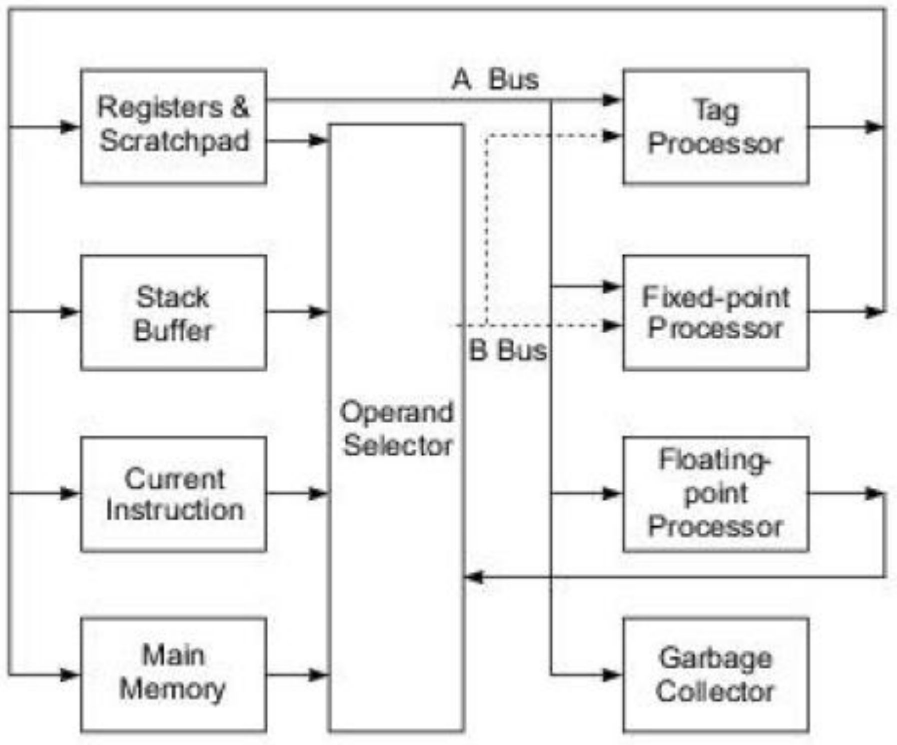
For example, a Lisp program can be viewed as a set of functions in which data are passed from function to function. The concurrent execution of these functions forms the basis for parallelism. The applicative and recursive nature of Lisp requires an environment that efficiently supports stack computations and function calling. The use of linked lists as the basic data structure makes it possible to implement an automatic garbage collection mechanism.

Instead of dealing with numerical data, symbolic processing deals with logic programs, symbolic lists, objects, scripts, blackboards, production systems, semantic networks, frames, and artificial neural networks. Primitive operations for artificial intelligence include search,compare, logic inference, pattern matching, unification. Filtering, context, retrieval, set operations, transitive closure, and reasoning operations. These operations demand a special instruction set containing compare, matching, logic, and symbolic manipulation operations. Floating point operations are not often used in these machines.

Example: **The Symbolics 3600 Lisp processor**

The processor architecture of the Symbolics 3600 is shown in Fig. 4.16. This was a stack-oriented machine. The division of the overall mar:h.ine architecture into layers allowed the use of a simplified instruction-set design, while implementation was carried out with a stack-oriented machine. Since most operands were fetched from the stack, the stack buffer and scratch-pad memories were implemented as fast caches to main memory.

The Symbolics 3600 executed most Lisp instructions in one machine cycle. 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.

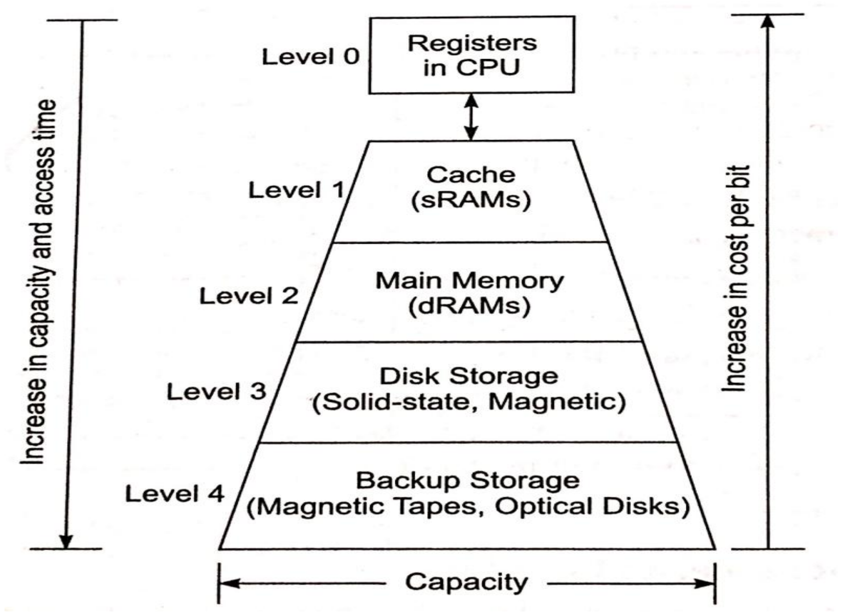


The architecture of the Symbolics 3600 Lisp processor (Courtesy of Symbolics, Inc., 1985)

2.3 :MEMORY HIERARCHY TECHNOLOGY

Qn:Describe the memory level hierarchy?

2.3.1 Hierarchical Memory Technology



Level 0
Registers in CPU
Level 1
Cache (sRAMs)
Level 2
Main Memory (dRAMs)
Level 3
Disk Storage (Solid-state, Magnetic)
Level 4
Backup Storage (Magnetic Tapes, Optical Disks)
Increase in capacity and access time
Increase in cost per bit
Capacity

The memory technology and storage organization at each level is **characterized by 5 parameters**

1. **Access Time (t_i) -** refers to round trip time for CPU to the ith-level memory.
2. **Memory size(s_i) -** is the number of bytes or words in level i
3. **Cost per byte(c_i)** – is the cost of ith level memory esitamed by $c_i s_i$
4. **Transfer bandwidth(b_i)** – rate at which information is transferred between adjacent levels.
5. **Unit of transfer(x_i) -** grain size of data transfer between levels i and i+1

Memory devices at lower levels (top of hierarchy) are:

- Faster to access
- Smaller in size
- More expensive/byte
- Higher bandwidth
- Uses smaller unit of transfer

Registers

- Registers are part of processor – Thus, at times not considered a part of memory
- Register assignment made by compiler
- Register operations directly controlled by compiler – thus register transfers take place at processor speed

Cache

- Controlled by MMU
- Can be implemented at one or multiple levels

- Information transfer between CPU and cache is in terms of words(4 or 8 bytes- depends on word length of machine)
- Cache is divided into cache blocks(typically 32 bytes)
- Blocks are unit of data transfer between cache and Main memory or btw L1 and L2 cache

Main Memory (primary memory)

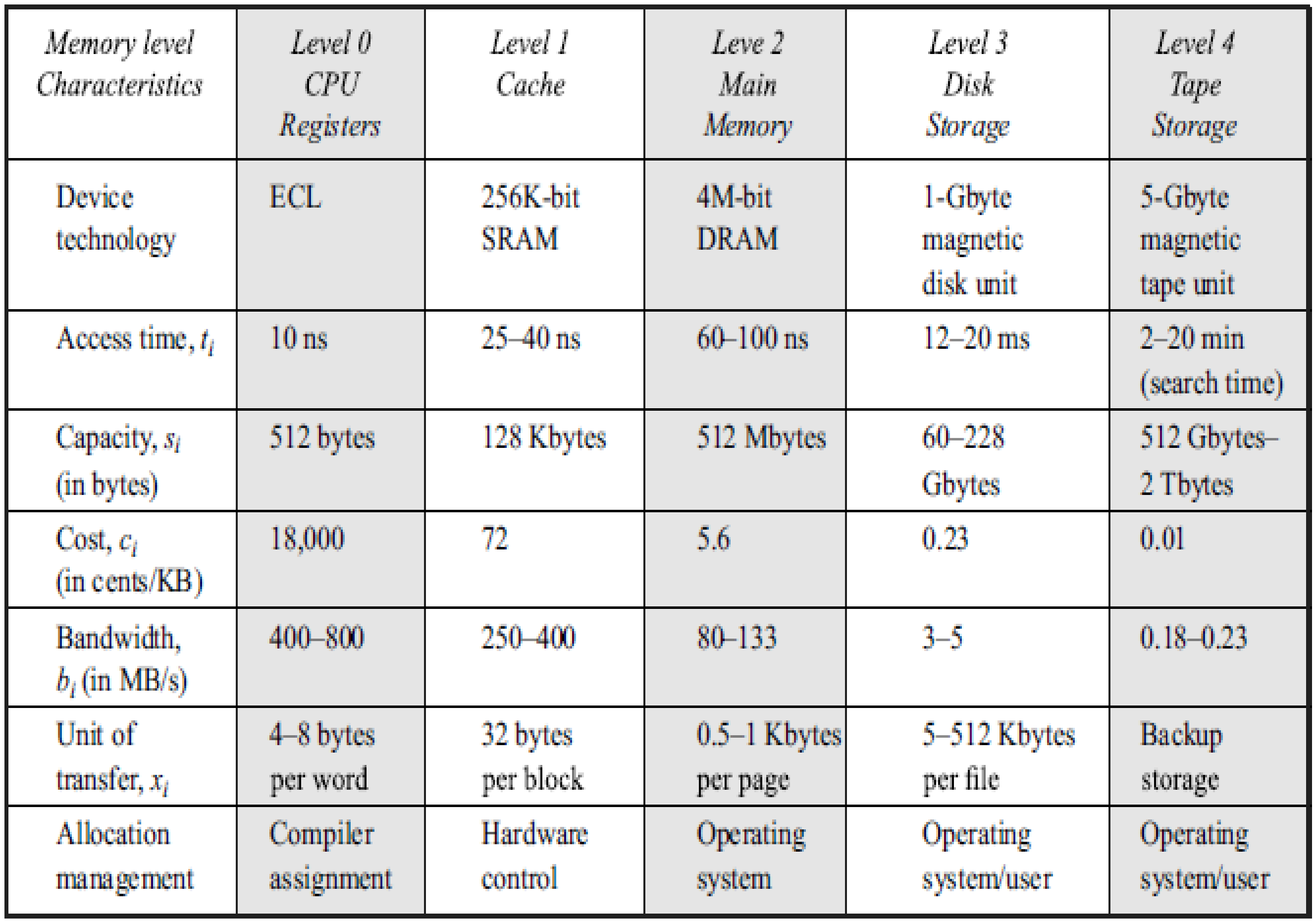
- Larger than cache
- Implemented by cost effective RAM chips (ex DDR SDRAMs)
- Managed by MMU in cooperation with OS
- Divided into pages(ex 4 Kbytes), each page contain 128 blocks
- Pages are the unit of information transferred between disk and main memory
- Scattered pages are organized as segments

Disk Drives and Backup Storage

- Highest level of memory
- Holds system programs like OS and compilers, user pgms, data etc

A typical workstation computer has the cache and main memory on a processor board and hard disks in an attached disk drive. Table below presents representative values of memory parameters for a typical 32-bit mainframe computer built in 1993.

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



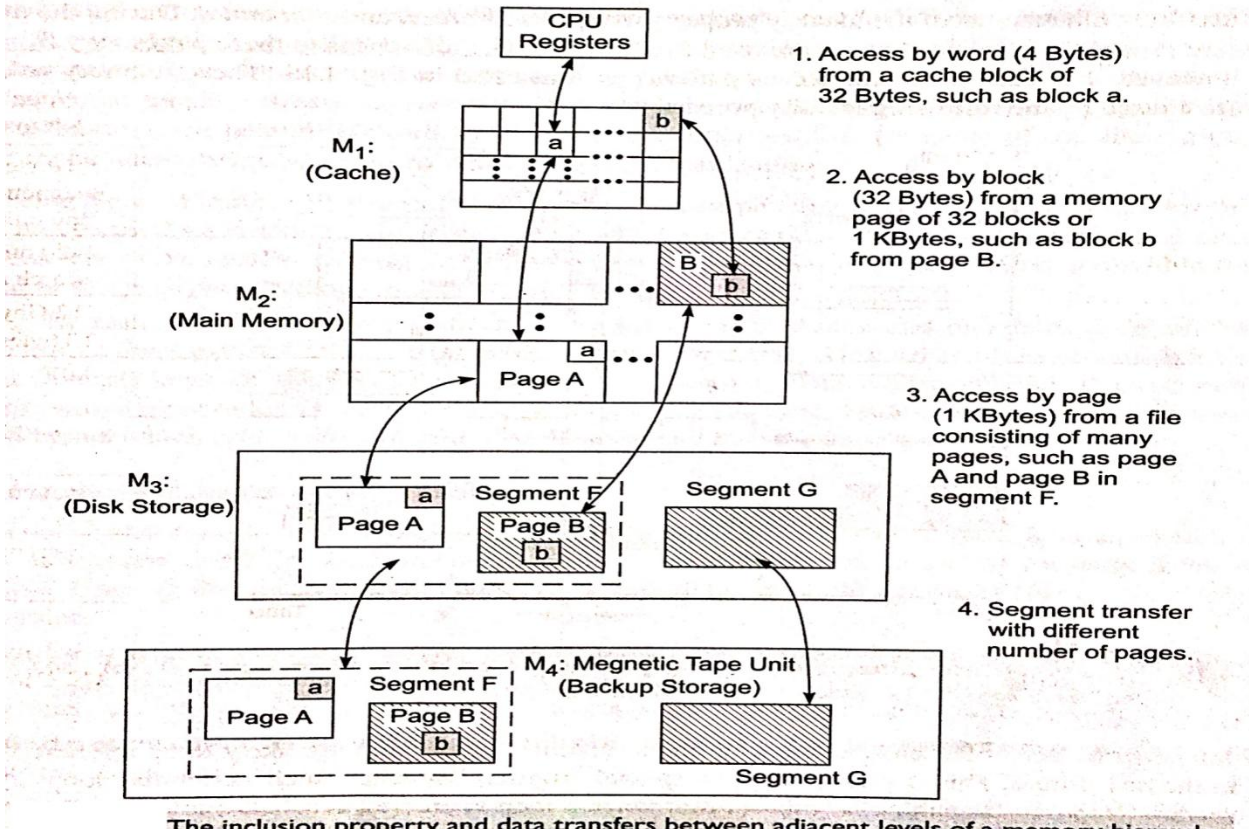
Memory level Characteristics	*Level 0 CPU Registers*	*Level 1 Cache*	*Leve 2 Main Memory*	*Level 3 Disk Storage*	*Level 4 Tape Storage*
Device technology	ECL	256K-bit SRAM	4M-bit DRAM	1-Gbyte magnetic disk unit	5-Gbyte magnetic tape unit
Access time, t_i	10 ns	25–40 ns	60–100 ns	12–20 ms	2–20 min (search 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

2.3.2 ***INCLUSION, COHERENCE and LOCALITY properties***

Qn: Explain the properties of inclusion, coherence and locality?

Information a memory hierarchy(M1, M2,…, Mn) should satisfy these 3 imp properties as shown in figure below:-

- **inclusion,**
- **coherence,**
- **locality**


CPU Registers
1. Access by word (4 Bytes) from a cache block of 32 Bytes, such as block a.
M1: (Cache)
2. Access by block (32 Bytes) from a memory page of 32 blocks or 1 KBytes, such as block b from page B.
M2: (Main Memory)
Page A
3. Access by page (1 KBytes) from a file consisting of many pages, such as page A and page B in segment F.
M3: (Disk Storage)
Segment F
Segment G
Page B
4. Segment transfer with different number of pages.
M4: Megnetic Tape Unit (Backup Storage)

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

Inclusion Property

- Stated as M_1 C M_2 C M_3 C……C M_n (C – symbol for subset)
- Implies all information are originally stored in outermost level M_n
- During processing subsets of M_n are copied into $M_{n-1.}$
- Thus if an information word is found in Mi then copies of same word can also be found in upper levels Mi+1, Mi+2…….Mn. However a word stored in Mi+1 may not be found in Mi
- A word miss in Mi implies its missing from all lower levels- thus highest level is the backup storage where everything is found

- Information transfer between **the CPU and cache**----- is in terms of ***words*** (4 or 8 bytes each depending on the word length of a machine). The cache (*M*1) is divided into *cache blocks*, also called *cache lines* by some authors.

- Information transfer between **cache and main memory,** or between *L*1 and *L*2 cache---- is in terms of **Blocks**(such as “a” and “b”). Each block may be typically 32 bytes (8 words). The main memory (*M*2) is divided into *pages*, say, 4 Kbytes each.
- Information transfer between **disk and main memory**.---- is in terms of **Pages**. Scattered pages are organized as a segment in the disk memory, for example, segment F contains page A, page B, and other pages.

- Data transfer between the **disk and backup storage** is ------ at the file level, such as **segments F and G**.

Coherence Property

QN:distinguish between write through and writeback caches?

- The property requires that copies of same information item at all memory levels should be consistent (same)
- If a word is modified in cache – copies of of the word should be updated at all higher levels.
- Frequently used information are usually found at lower levels – in order to minimize effective access time
- 2 general properties to maintain coherence in memory hierarchy
  1. **Write Through** – demands immediate update in Mi+1 if a word is updated in Mi for i=1,2,3…n-1.
  2. **Write Back** –delays the update in Mi+1 until word being modified in Mi is replaced or removed from Mi.

Locality of Reference and memory design implications

Qn:Compare spatial locality, temporal locality and sequential locality?

The memory hierarchy was developed based on a program behavior known as **locality of references.** 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.** Most programs use a certain portion of memory during a particular time period. **90-10 rule**: states that Typical pgm may spend 90% of the its execution time on only 10% of its code (ex loop in a code)

- 3 dimensions of locality property and its memory design implication(shown in bullets)

  1. **Temporal locality –** recently referenced items are likely to be referenced again in near future. (ex: loop portion in a code is executed continuously until loop terminates)
     - leads to usage of least recently used replacement algorithm
     - helps to determine size of memory at successive levels
  2. **Spatial locality –** tendency for a process to access items whose addresses are near one another. (ex: array elements , macros, routines etc)
     - Assists in determining size of of unit data transfer between adjacent memory levels
  3. **Sequential locality** – execution of instructions follows a sequential order unless branched instruction occurs
     - Affects determination of grain size for optimal scheduling

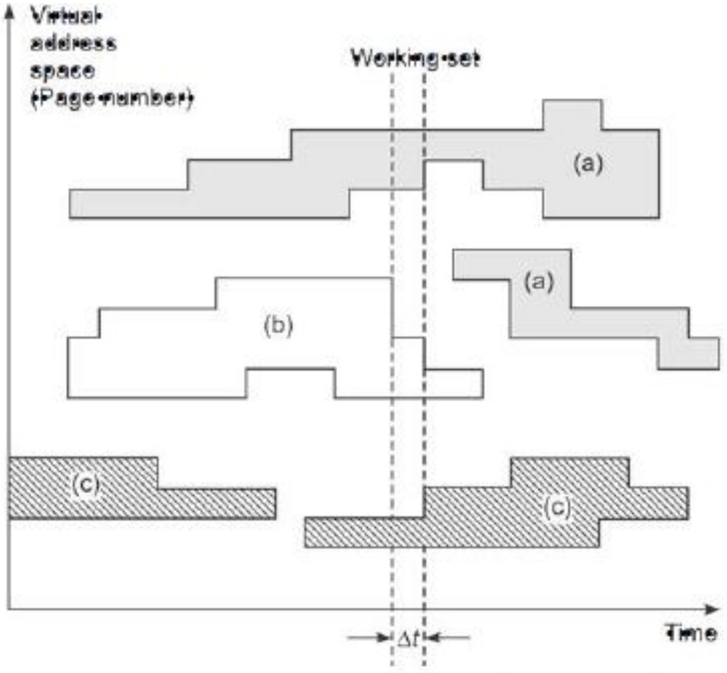
Principle of localities guide in design of cache memory , main memory and even virtual memory organization..

Working Sets

Figure Below, 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.

The subset of addresses (or pages) referenced within a given time window $(t, t + \Delta t)$ is called the *working set* by Denning (1968).

During the execution of a program, the working set changes slowly and maintains a certain degree of continuity as demonstrated in Fig. below. 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 memoryaccess time with a higher hit ratio at the lowest memory level. The time window Δt is a critical parameter set by the OS kernel which affects the size of the working set and thus the desired cache size.



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

2.3.3 <u>*MEMORY CAPACITY PLANNING*</u>

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

Terms-

Hit Ratio – when an information item is found in Mi, we call it a **hit**, otherwise a **miss**.

Considering memory level Mi to Mi-1 in a hierarchy , i=1,2,3…n-1. The **hit ratio ,hi** at **Mi** is the probability that an item required will be found in Mi. The miss ratio at Mi is **1-hi**.

The hit ratios at successive levels are a function of memory capacities, management policies and program behavior.

Every time a miss occurs a penalty has to be paid to access next higher level of memory. **Cache miss is 2 to 4 times costlier than a cache hit**. **Page faults are 1000 to 10000 times costly than page hit**

We assume $h_0 = 0$ and $h_n = 1$, which means CPU always accesses M_1 first, and the access to the outermost memory M_n is always a hit.

Access frequency to Mi is defined as fi =(1-h1)(1-h2)….(1-hi-1)hi. This is the probability of successfully accessing Mi when there are i-1 misses at the lower levels and a hit at Mi. Note that $\sum_{i=1}^{n} fi = 1$ and $f_1=h_1$

Due to the **locality property**, the access frequencies decrease very rapidly from low to high levels; that is, $f1 >> f2 >> f3 >> \ldots >> fn$. This implies that the inner levels of memory are accessed more often than the outer levels.

Effective access time

In practice, we wish to achieve as high a hit ratio as possible at M1. Every time a miss occurs, a penalty must be paid to access the next higher level of memory. The misses have been called **block misses** in the cache and **page faults** in the main memory because blocks and pages are the units of transfer between these levels.

Using the access frequencies f_i=1,2,….n, we can formally define the effective access time of a memory hierarchy as follows

$$\sum_{i=1}^{n} fi.ti = h_1t_1+(1-h_1)h_2t_2+(1-h_1)(1-h_2)h_3t_3+\ldots\ldots(1-h_1)(1-h_2)\ldots.(1-h_{n-1})t_n$$

Hierarchy Optimization

The total cost of a memory hierarchy is estimated as follows

$$C_{total} = \sum_{i=1}^{n} Ci.Si$$

This implies cost is distributed across n levels.

The optimal design of memory should result in total **effective access time close to effective access time of cache** and a **total cost of highest memory level**-----practically not possible.