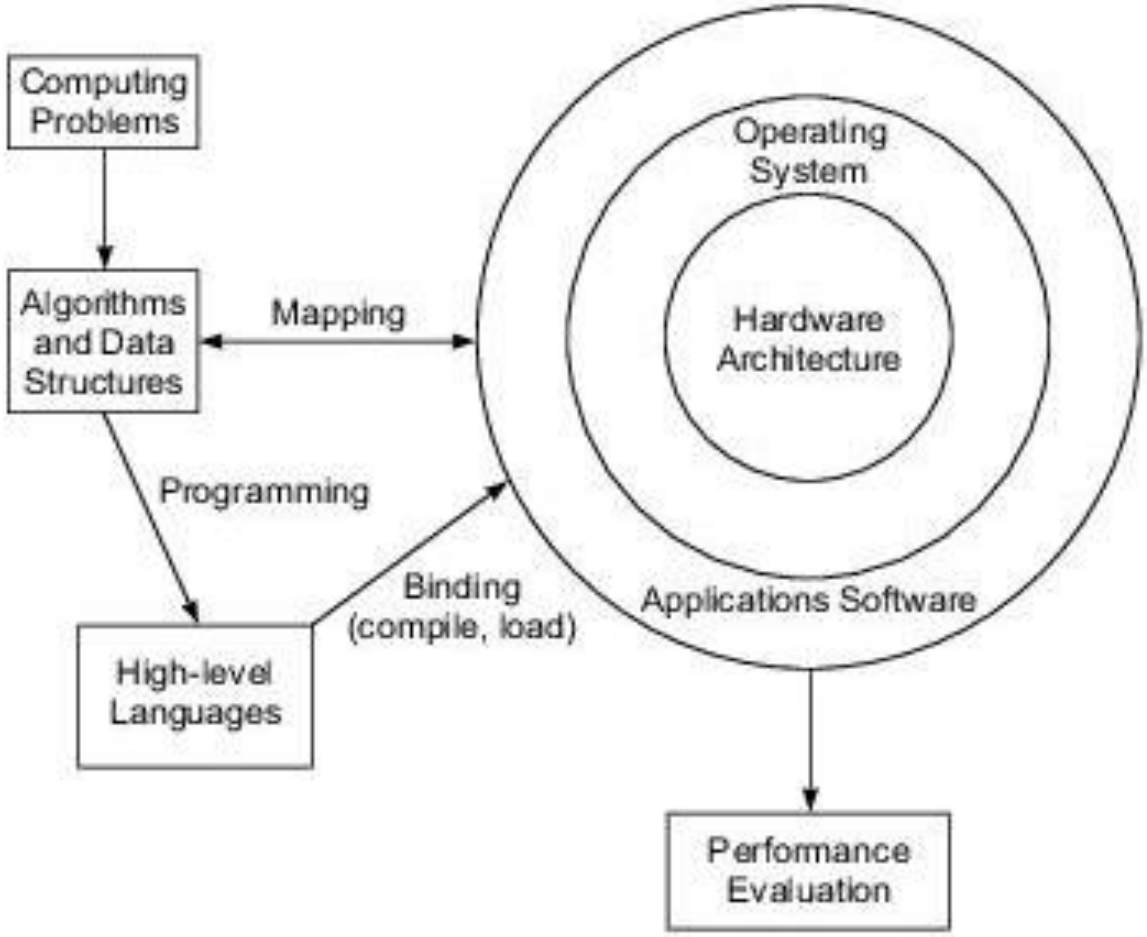
BCS-29
Advanced Computer Architecture

Parallel Computer Models

Elements of Modem Computers

- The hardware, software, and programming elements of modern computer systems can be characterized by looking at a variety of factors, including:
  - Computing problems
  - Algorithms and data structures
  - Hardware resources
  - Operating systems
  - System software support
  - Compiler support

Elements of Modem Computers



Computing Problems

- Numerical computing
  - complex mathematical formulations
  - tedious integer or floating-point computation
- Transaction processing
  - accurate transactions
  - large database management
  - information retrieval
- Logical Reasoning
  - logic inferences
  - symbolic manipulations

Algorithms and Data Structures

- Traditional algorithms and data structures are designed for sequential machines.
- New, specialized algorithms and data structures are needed to exploit the capabilities of parallel architectures.
- These often require interdisciplinary interactions among theoreticians, experimentalists, and programmers.

Hardware Resources

- The architecture of a system is shaped only partly by the hardware resources.
- The operating system and applications also significantly influence the overall architecture.
- The modem computer system demonstrates its power through coordinated efforts by hardware resources, an operating system, and application software.
- Not only the processor and memory architectures be considered, but also the architecture of the device interfaces (which often include their advanced processors).

Operating System

- Operating systems manage the allocation and deallocation of resources during user program execution.
- UNIX, Mach, and OSF/1 provide support for
  - multiprocessors and multicomputers
  - multithreaded kernel functions
  - virtual memory management
  - file subsystems
  - network communication services
- An OS plays a significant role in mapping hardware resources to algorithmic and data structures.

Operating System

- An effective operating systems manage the allocation and deallocation of resources during user program execution.
- An OS plays a significant role in mapping hardware resources to algorithmic and data structures.
- Efficient mapping will benefit the programmer and produce better source codes.
- The mapping of algorithmic and data structures onto the machine architecture includes processor scheduling, memory maps, Inter processor communications, etc. These activities are usually architecture-dependent.

System Software Support

- Compilers, assemblers, and loaders are traditional tools for developing programs in high-level languages. With the operating system, these tools determine the bind of resources to applications, and the effectiveness of this determines the efficiency of hardware utilization and the system's programmability.
- Most programmers still employ a sequential mind set, abetted by a lack of popular parallel software support.
- Parallel software can be developed using entirely new languages designed specifically with parallel support as its goal, or by using extensions to existing sequential languages.
- New languages have obvious advantages (like new constructs specifically for parallelism), but require additional programmer education and system software.
- The most common approach is to extend an existing language.

Compiler Support

- Preprocessors
  - use existing sequential compilers and specialized libraries to implement parallel constructs
- Precompilers
  - perform some program flow analysis, dependence checking, and limited parallel optimzations
- Parallelizing Compilers
  - requires full detection of parallelism in source code, and transformation of sequential code into parallel constructs
- Compiler directives are often inserted into source code to aid compiler parallelizing efforts

The history of Intel's processors

Intel 4004 (1971)

- 16-pin DIP package
- 4-bits processed at a time
- 12-bit addresses
- Clock: 740KHz
- Address Space: 4 KB
- Instruction Set: 46
- Registers: 16

Intel 4040 (1974)

- Instruction Set expanded to 60 instructions
- Program memory expanded to 8 KB (13-bit address space)
- Registers expanded to 24
- Subroutine stack expanded to 7 levels deep

8-bit Microprocessors

8008

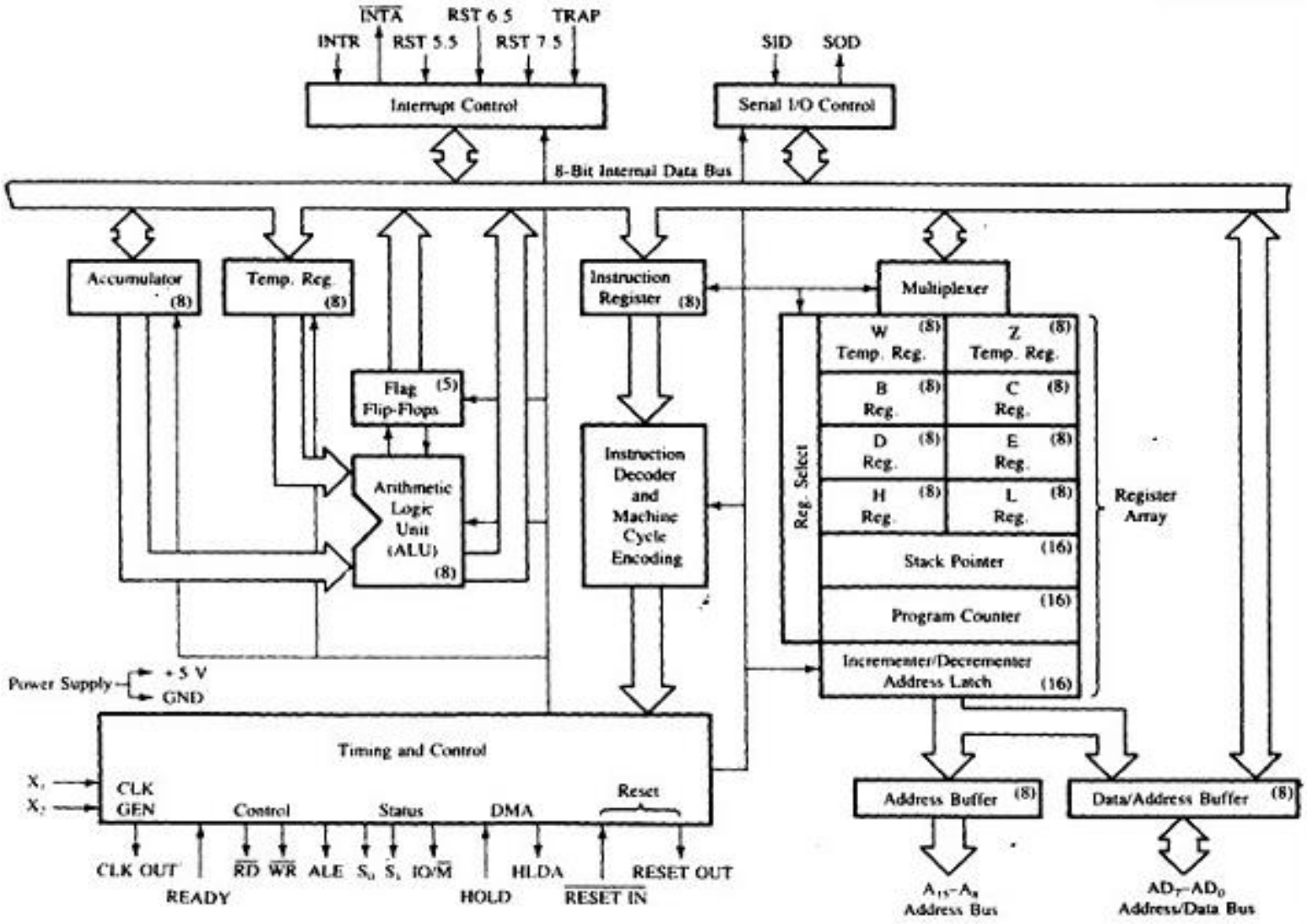
- Processed 8-bits at a time
- 14-bit addresses
- First to include an interrupt line

Features

- Seven 8-bit "scratchpad" registers: The main accumulator (A) and six other registers (B, C, D, E, H, and L).
- 14-bit program counter (PC).
- Seven-level push-down address call stack. Eight registers are actually used, with the top-most register being the PC.
- Four condition code status flags: carry (C), even parity (P), zero (Z), and sign (S).
- Indirect memory access using the H and L registers (HL) as a 14-bit data pointer (the upper two bits are ignored).

Elementary Discussion on Modem Computers

8085 architectural block diagram

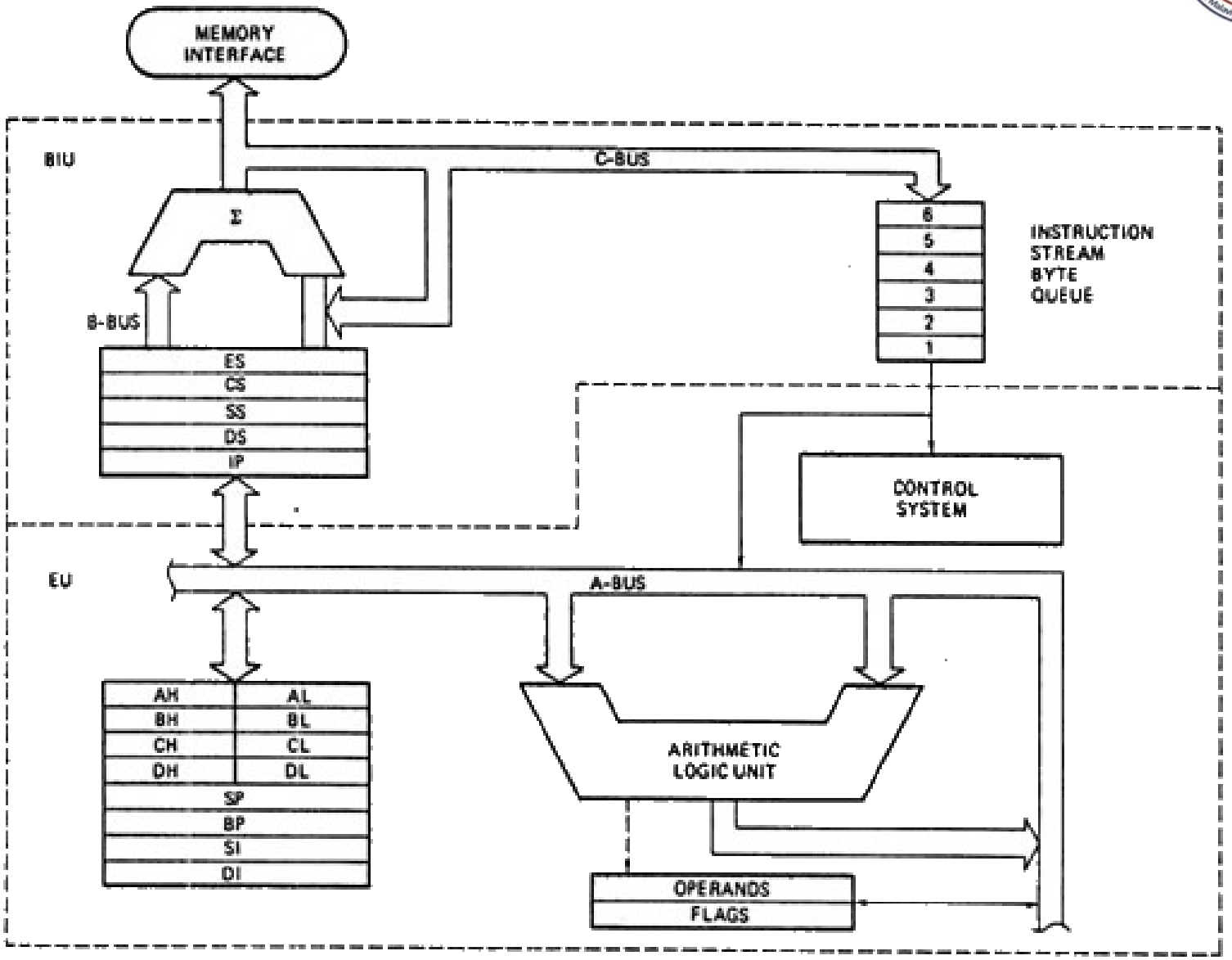


16-bit and 8-bit Microprocessors

- 8086 (16-bit) & 8088 (8-bit)
- 20-bit addresses
- Two processors which consisted of: Bus Interface Unit and Execution Unit
- Segmented Memory

Elementary Discussion on Modem Computers

8088 architectural block diagram



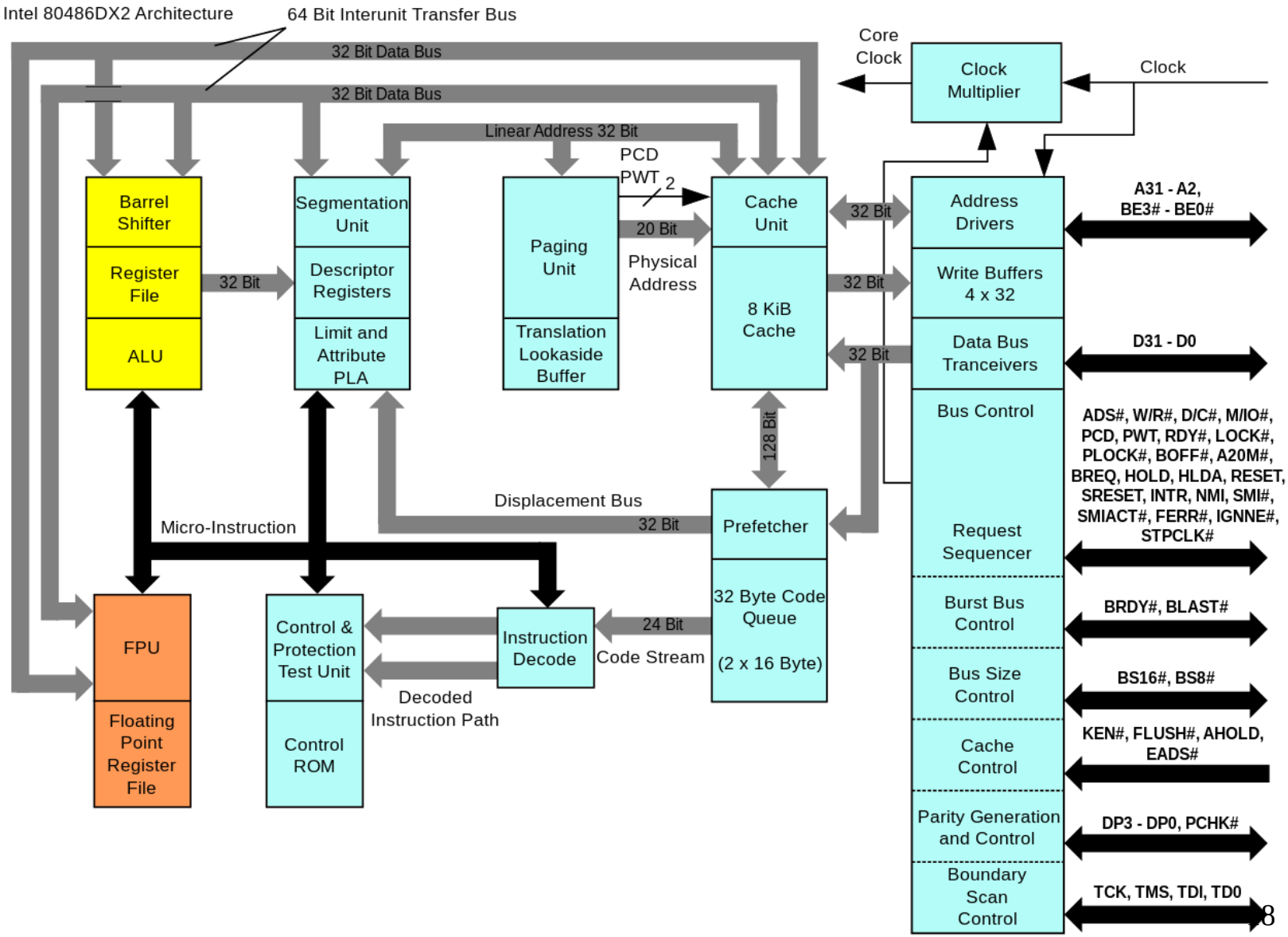
80286 & 80386

- Two modes
  - 8086 Real Address Mode
  - Protected Virtual Address Mode
- Protected Virtual Address Mode
  - More address space
  - Multi-user protection
    - Dedicated 286; task
  - Virtual memory
    - Interrupt handler

Introduction to 80486

- Increased speed (2 x 80386)
- 80386 with an internal math coprocessor
- Upgradeable (Compatibility)
- Utilize new technology (RISC, cache)

80486 Processor



Intel Pentium

- Introduced in '93 but was expected in 1992
- 60/66 MHz Bus Speed
- Improved cache structure
  - Re-organized to form two caches that are each 8K bytes in size
    - One for caching data
    - Another for caching instructions
- Wider data bus width
  - Increased from 32 bit to 64 bits

Intel Pentium

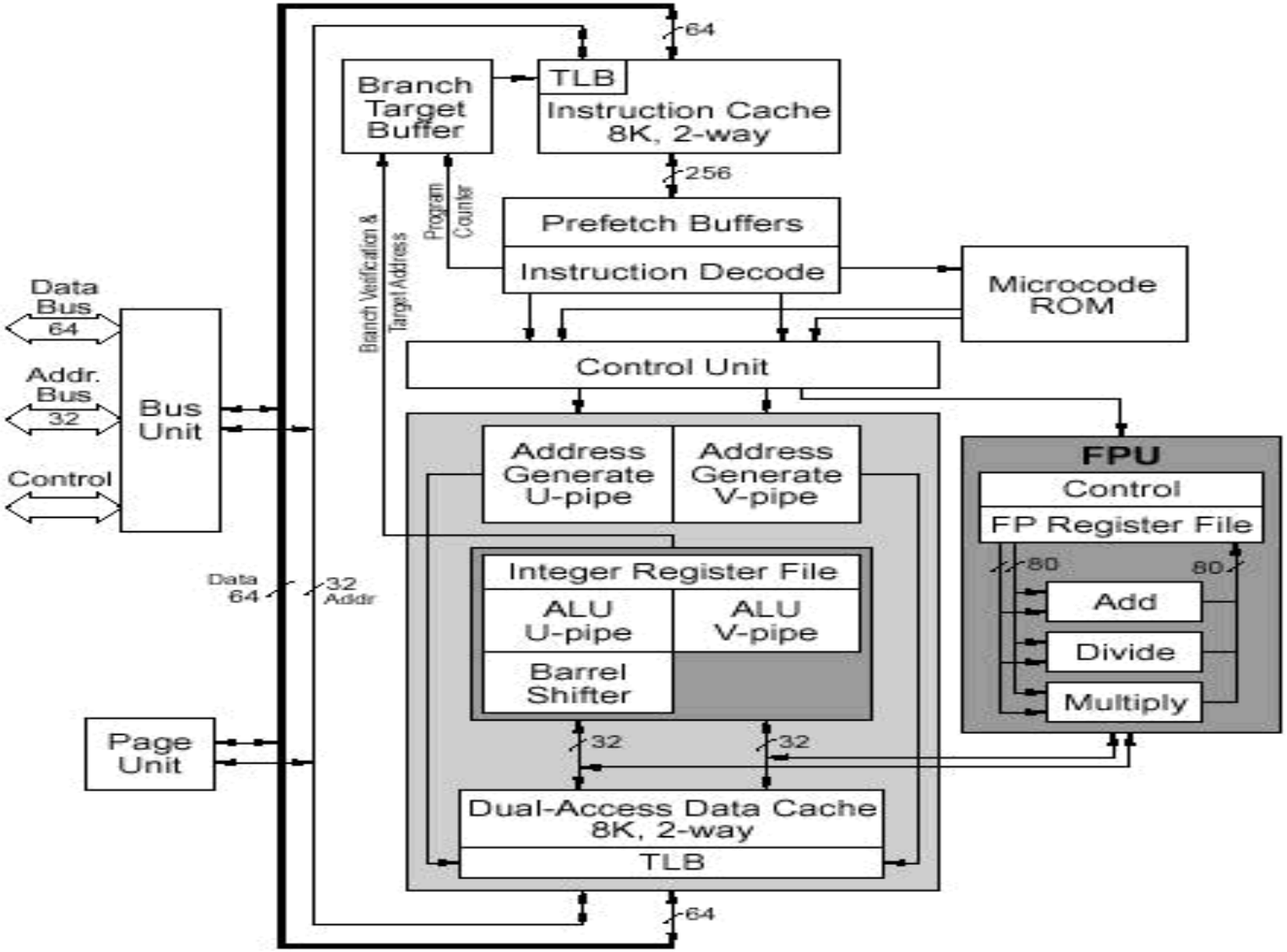
- Faster numeric processor
  - Operates about 5 times faster than the 486 numeric processor
- A dual integer processor
  - Often allows two instructions per clock
- Branch prediction logic
- MMX instructions --- a later introduction to Pentium

Intel Pentium

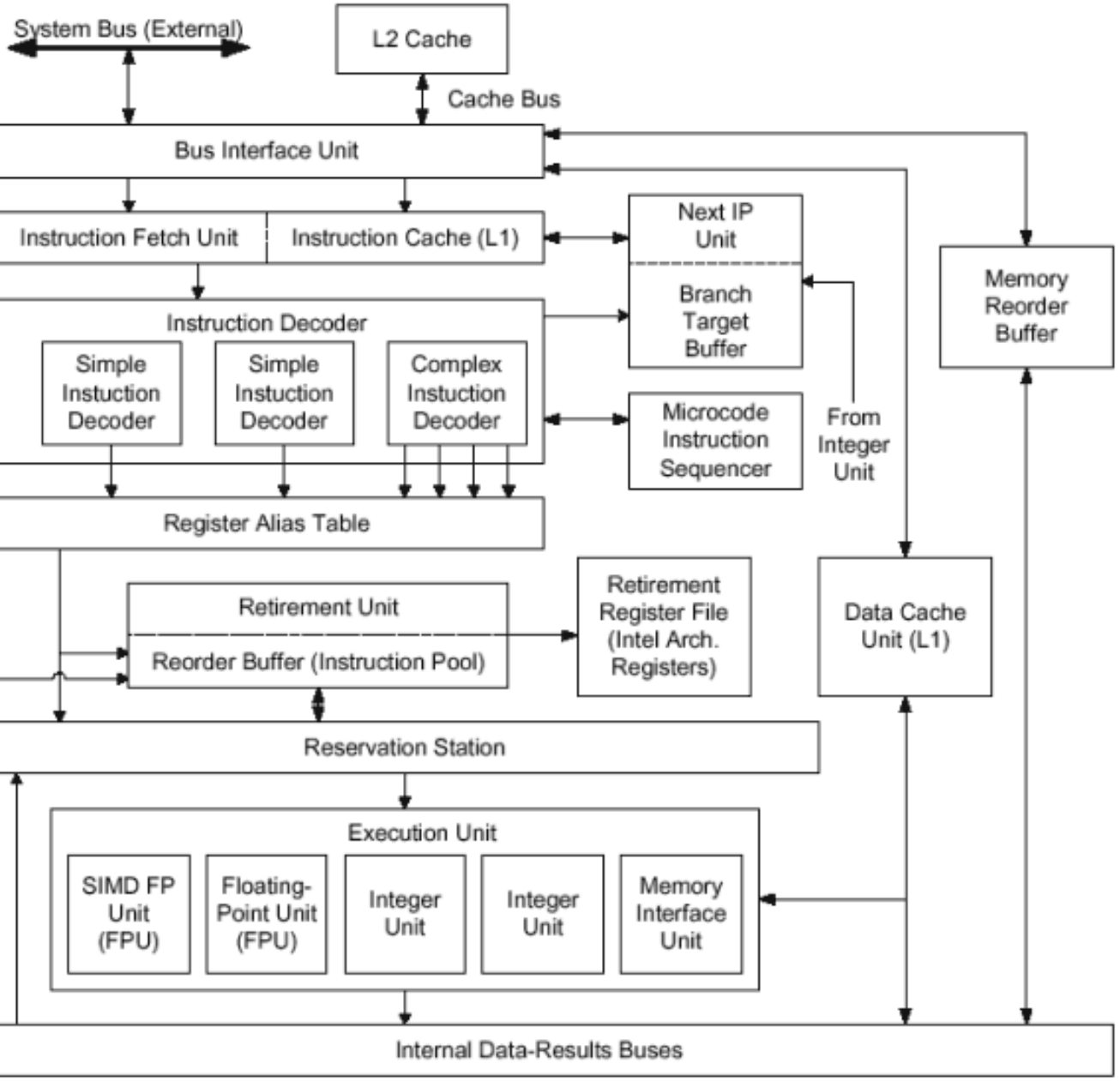
- Pentium has two execution lines
  - U-line --- can execute any Pentium instruction
  - V-line --- only executes *simple* instructions.
- Each line has 5 stages
  i. Pre-fetch
  ii. Instruction Decode
  iii. Address Generation
  iv. Execute, Cache, and ALU Access
  v. Write back

Elementary Discussion on Modem Computers

Pentium architectural block diagram



Elementary Discussion on Modem Computers



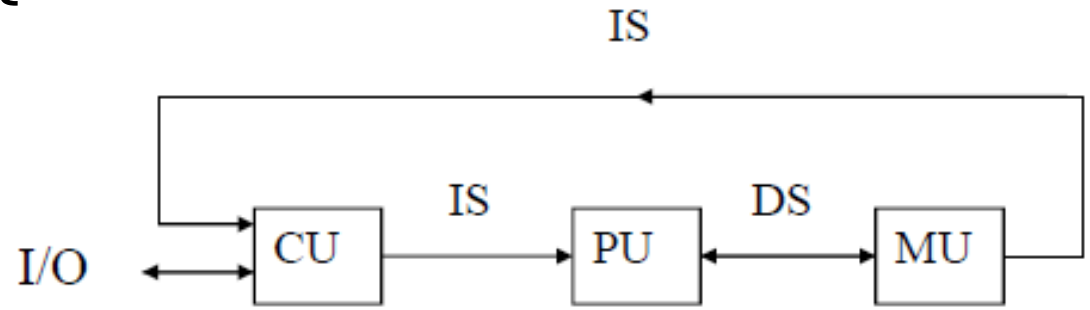
P6 Microarchitecture

Classification of Parallel Architectures

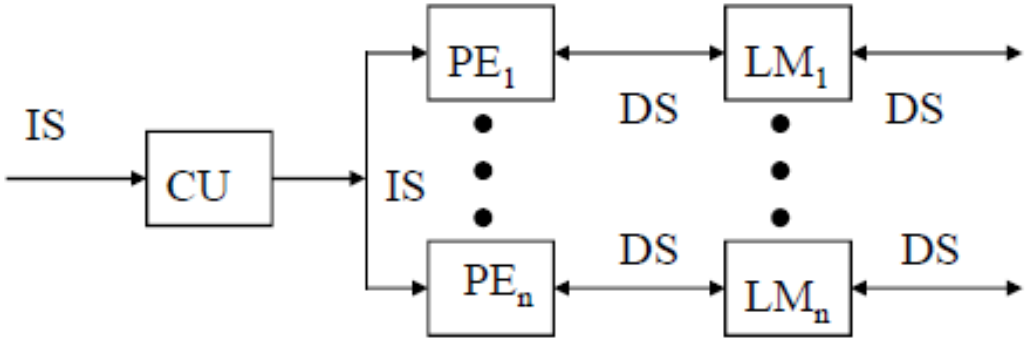
- **Classification based on Architectural schemes**
  - Flynn's classification
    - It is based upon the number of concurrent instruction (or control) and data streams available in the architecture:
  - Feng's classification
    - This classification is mainly based on degree of parallelism to classify parallel computer architecture.
  - Handle's classification
    - Handler's proposed an elaborate notation for expressing the pipelining and parallelism of computers. He divided the computer at three levels such as Processor Control Unit(PCU), Arithmetic Logic Unit(ALU), Bit Level Circuit(BLC)

Flynn's classification

- Single instruction, single data stream (SISD)

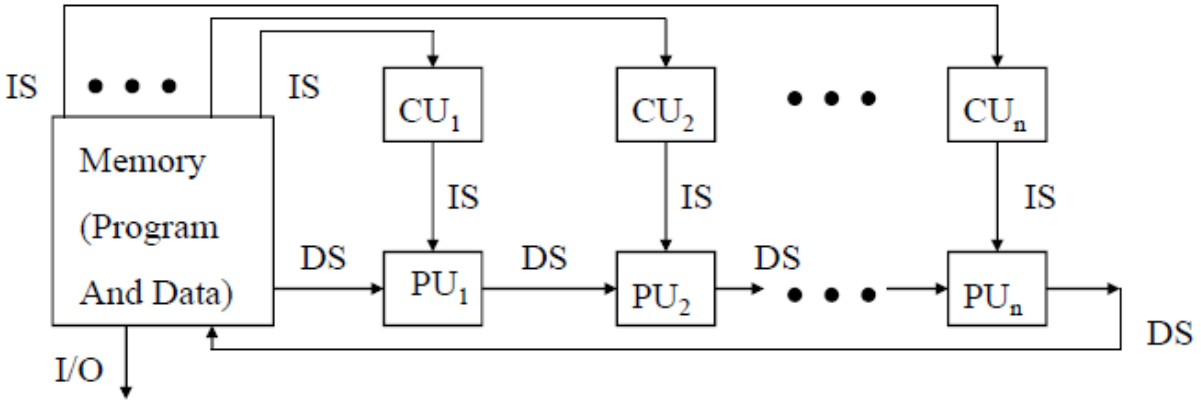


- Single instruction, multiple data streams (SIMD)
  - vector computers with scalar and vector hardware

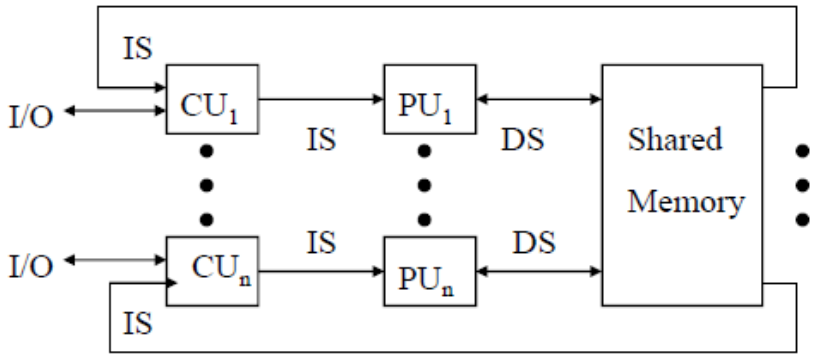


Flynn's classification

- Multiple instructions, single data stream (MISD): systolic arrays



- Multiple instructions, multiple data streams (MIMD): parallel computers



- Among parallel machines, MIMD is most popular, followed by SIMD, and finally MISD.

Feng's classification

- Feng suggested the use of degree of parallelism to classify various computer architectures.
- The maximum number of binary digits that can be processed within a unit time by a computer system is called the maximum parallelism degree P.
- A bit slice is a string of bits one from each of the words at the same vertical position.
- Under above classification
  - Word Serial and Bit Serial (WSBS)
  - Word Parallel and Bit Serial (WPBS)
  - Word Serial and Bit Parallel(WSBP)
  - Word Parallel and Bit Parallel (WPBP)

Feng's classification

- WSBS has been called bit parallel processing because one bit is processed at a time.
- WPBS has been called bit slice processing because m-bit slice is processes at a time.
- WSBP is found in most existing computers and has been called as Word Slice processing because one word of n-bit processed at a time.
- WPBP is known as fully parallel processing in which an array on nxm-bits is processes at one time.

Handle's classification

- Handler's proposed an elaborate notation for expressing the pipelining and parallelism of computers. He divided the computer at three levels.
  - Processor Control Unit(PCU)
  - Arithmetic Logic Unit(ALU)
  - Bit Level Circuit(BLC)

  PCU corresponds to CPU, ALU corresponds to a functional unit or PE's in an array processor. BLC corresponds to the logic needed for performing operation in ALU.

  He uses three pairs of integers to describe computer:

  Computer = (k*k',d*d , w*w')

  Where,

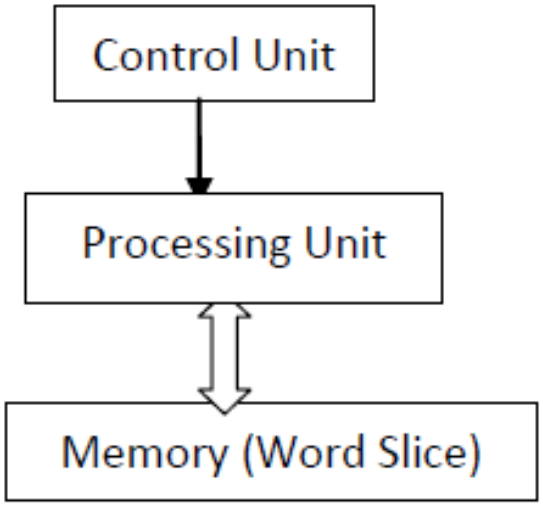
  k= no. of PCUs k'=no. of PCUs which are pipelined;

  d=no. of ALUs control by each PCU d'=no. of ALUs that can be pipelined

  w=no. of bits or processing elements in ALU w'=no. of pipeline segments

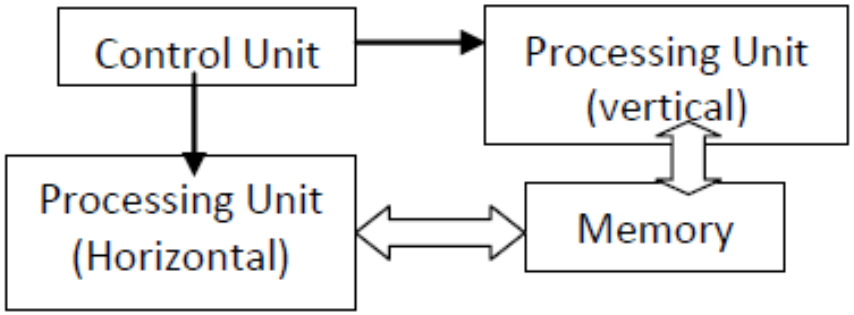
Shor's Classification:

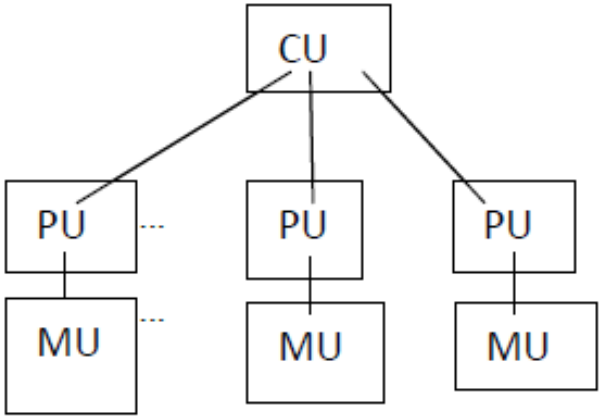
- In this classification computers are classified on the basis of organization of the constituent elements in the computer. He proposed 6 machines which are recognized and distinguished by numerical designators.
- Machine1:



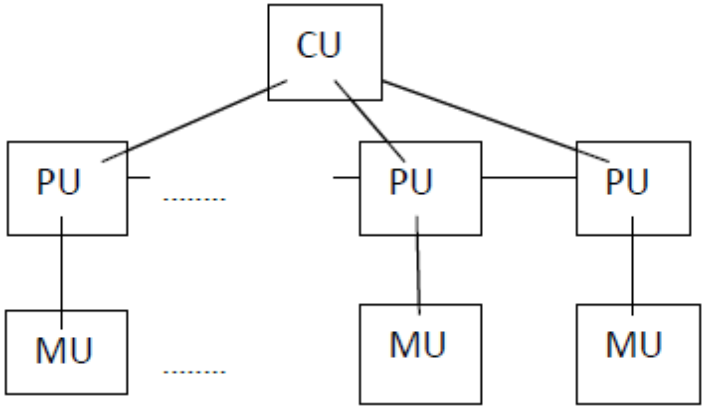
Shor's Classification:

Machine2:





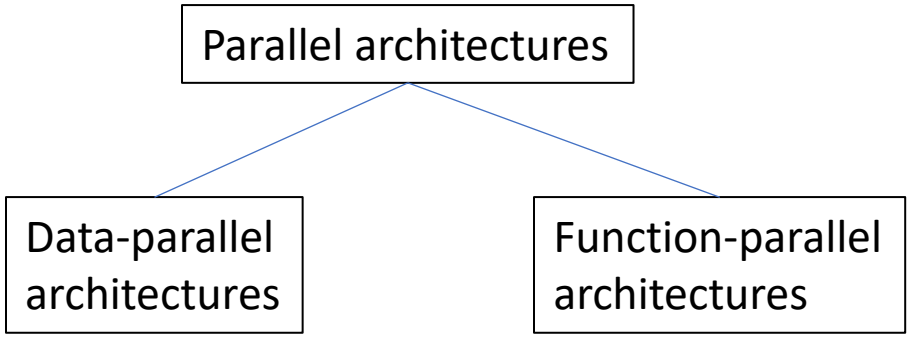
Machine3:



Machine4:

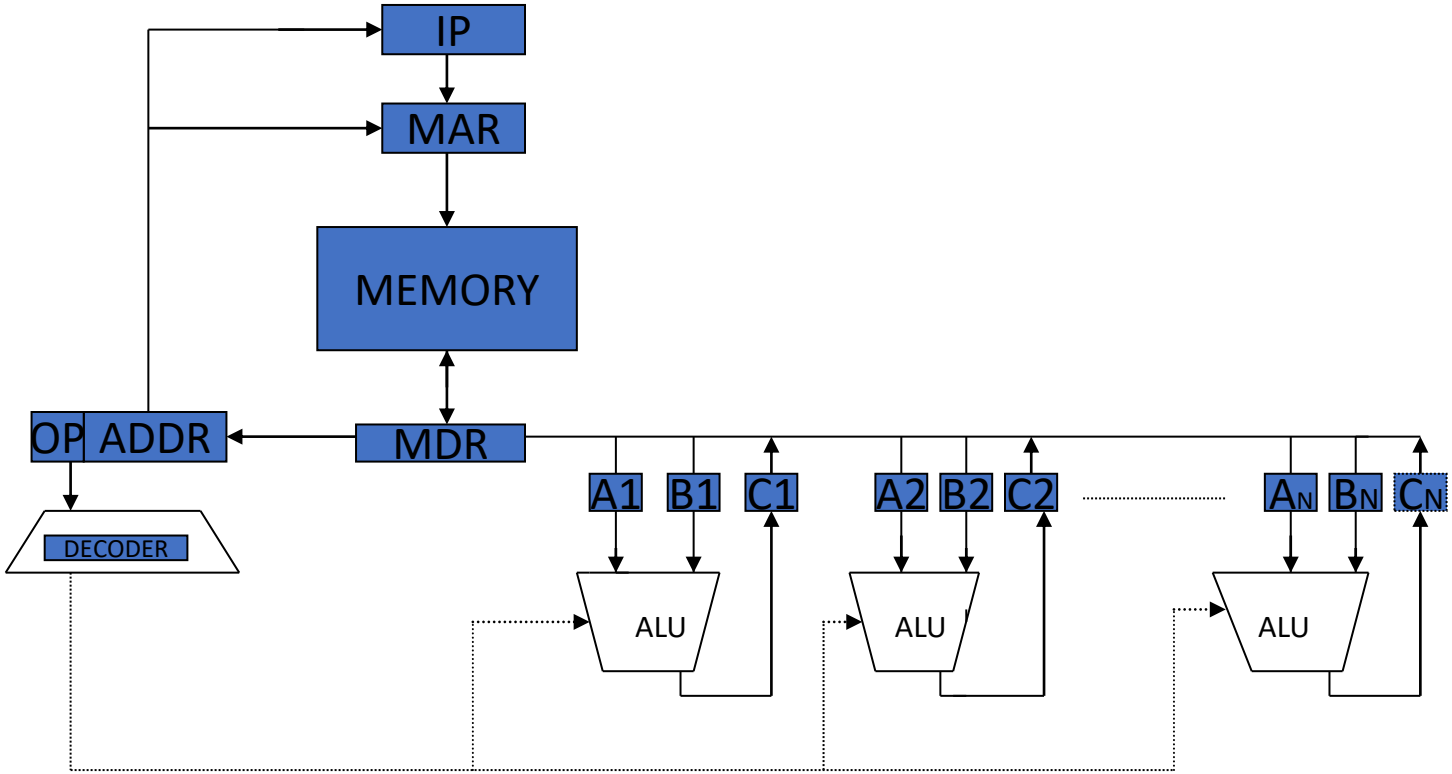
Modern classification(Sima, Fountain, Kacsuk)

- Classify based on how parallelism is achieved
  - by operating on multiple data: data parallelism
  - by performing many functions in parallel: function parallelism
    - Control parallelism, task parallelism depending on the level of the functional parallelism.

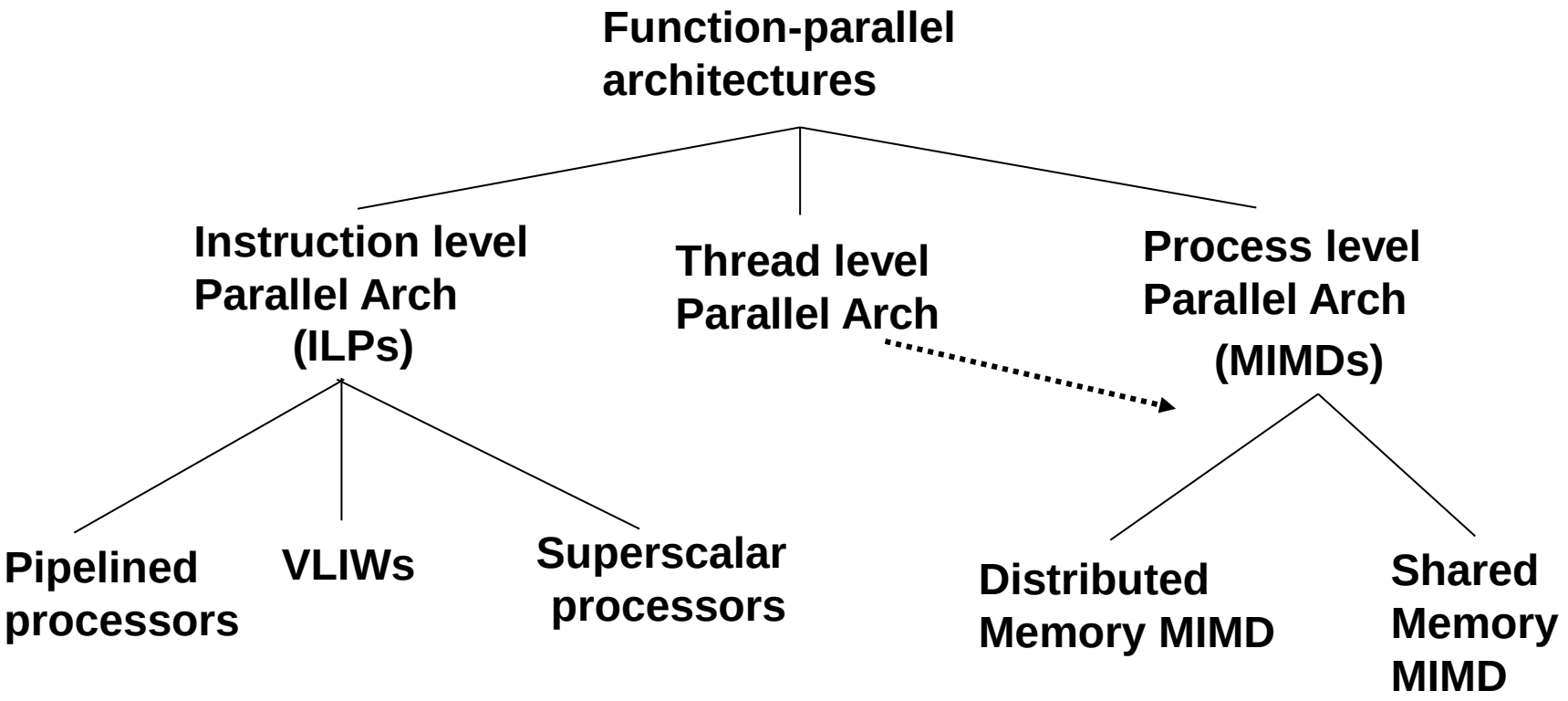


Data parallel architectures

- Vector processors, SIMD (array processors), systolic arrays.



Function Parallel Architectures



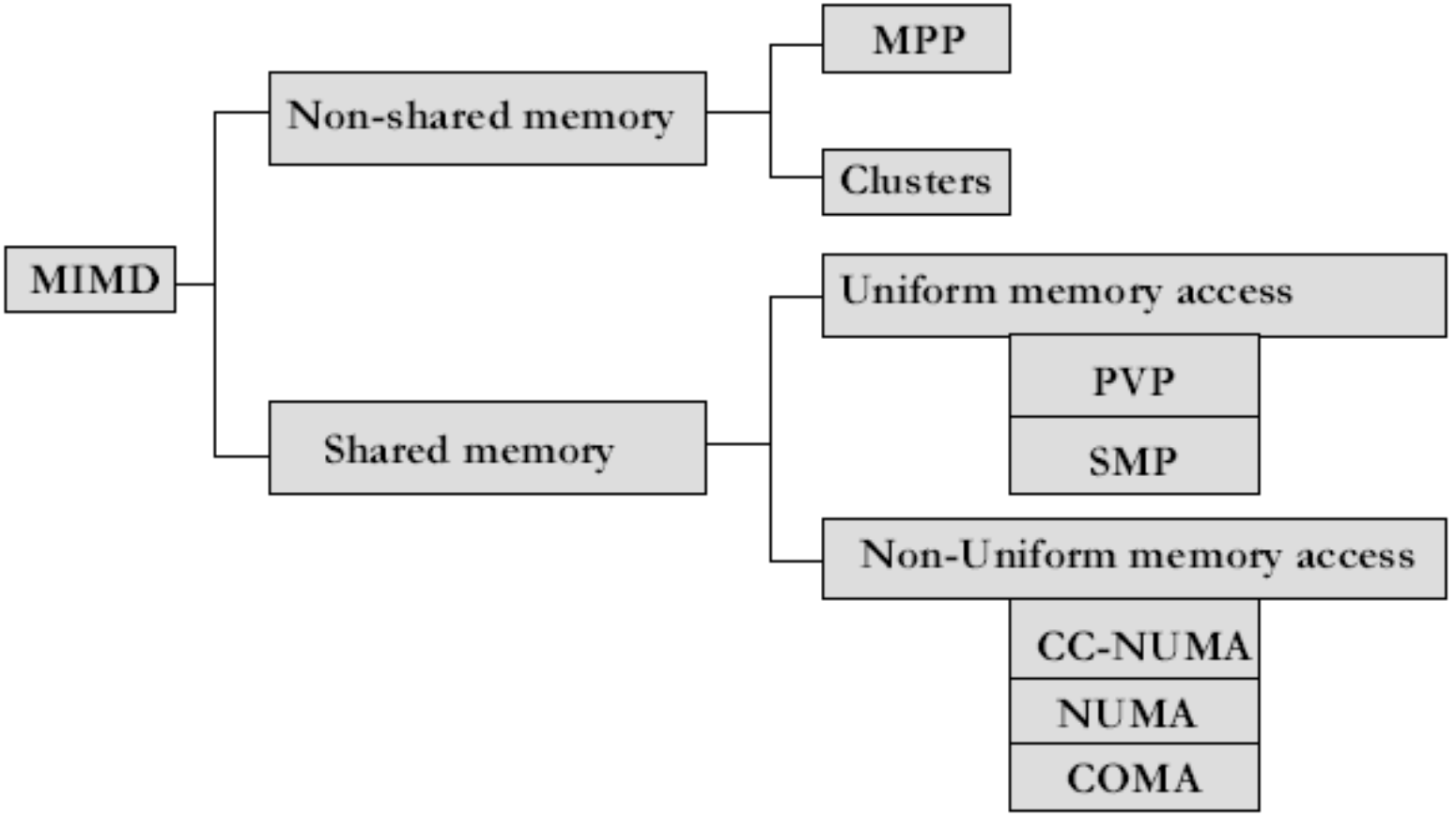
Categories of Parallel Computers(MIMD)

- Considering their architecture only, there are two main categories of parallel computers:
  - systems with shared common memories, and
  - systems with unshared distributed memories.

Shared-Memory Multiprocessors

- Shared-memory multiprocessor models:
  - Uniform-memory-access (UMA)
  - Nonuniform-memory-access (NUMA)
  - Cache-only memory architecture (COMA)
- These systems differ in how the memory and peripheral resources are shared or distributed.

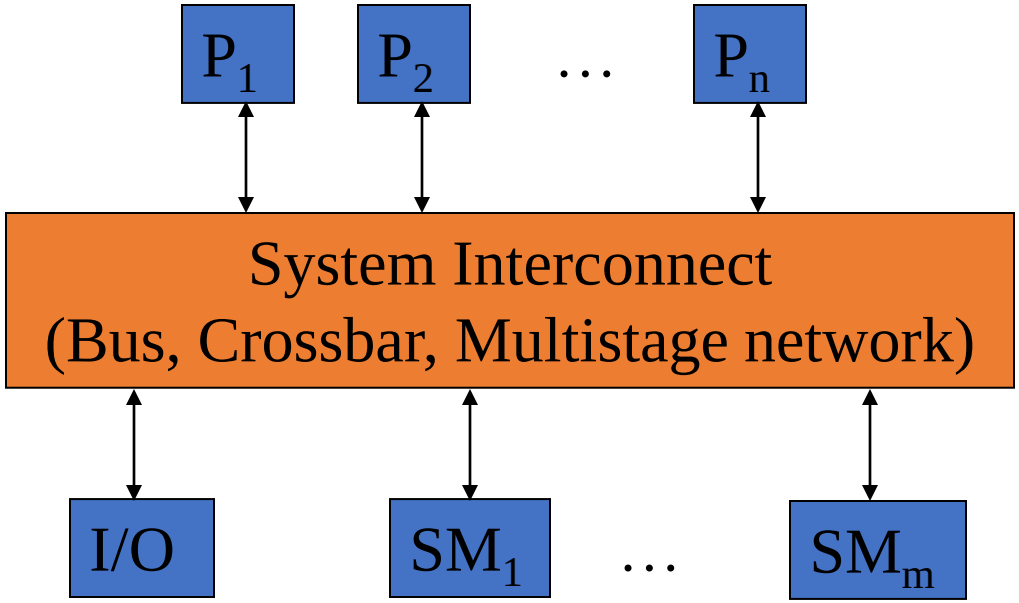
Categories of Parallel Computers(MIMD)



The UMA Model

- Physical memory uniformly shared by all processors, with equal access time to all words.
- Processors may have local cache memories.
- Peripherals also shared in some fashion.
- *Tightly coupled systems* use a common bus, crossbar, or multistage network to connect processors, peripherals, and memories.
- Many manufacturers have multiprocessor (MP) extensions of uniprocessor (UP) product lines.
- Synchronization and communication among processors achieved through shared variables in common memory.
- Symmetric MP systems – all processors have access to all peripherals, and any processor can run the OS and I/O device drivers.
- Asymmetric MP systems – not all peripherals accessible by all processors; kernel runs only on selected processors (master); others are called *attached processors* (AP).

The UMA Multiprocessor Model



Performance Calculation

- Consider two loops. The first loop adds corresponding elements of two *N*-element vectors to yield a third vector. The second loop sums elements of the third vector. Assume each add/assign operation takes 1 cycle, and ignore time spent on other actions (e.g. loop counter incrementing/testing, instruction fetch, etc.). Assume interprocessor communication requires *k* cycles.
- On a sequential system, each loop will require N cycles, for a total of 2*N* cycles of processor time.
- On an *M*-processor system, we can partition each loop into M parts, each having $L = N / M$ add/assigns requiring *L* cycles. The total time required is thus 2*L*. This leaves us with *M* partial sums that must be totaled.
- Computing the final sum from the *M* partial sums requires $l = \log_2(M)$ additions, each requiring *k* cycles (to access a non-local term) and 1 cycle (for the add/assign), for a total of $l \times (k+1)$ cycles.
- The parallel computation thus requires

$$2N / M + (k + 1) \log_2(M) \text{ cycles.}$$

Performance Calculation

- Assume N = 2^{20}.
- Sequential execution requires 2N = 2^{21} cycles.
- If processor synchronization requires k = 200 cycles, and we have M = 256 processors, parallel execution requires

$$2N / M + (k + 1) \log_2(M)$$
$$= \quad 2^{21} / 2^8 + 201 \times 8$$
$$= \quad 2^{13} + 1608 \;=\; 9800 \text{ cycles}$$

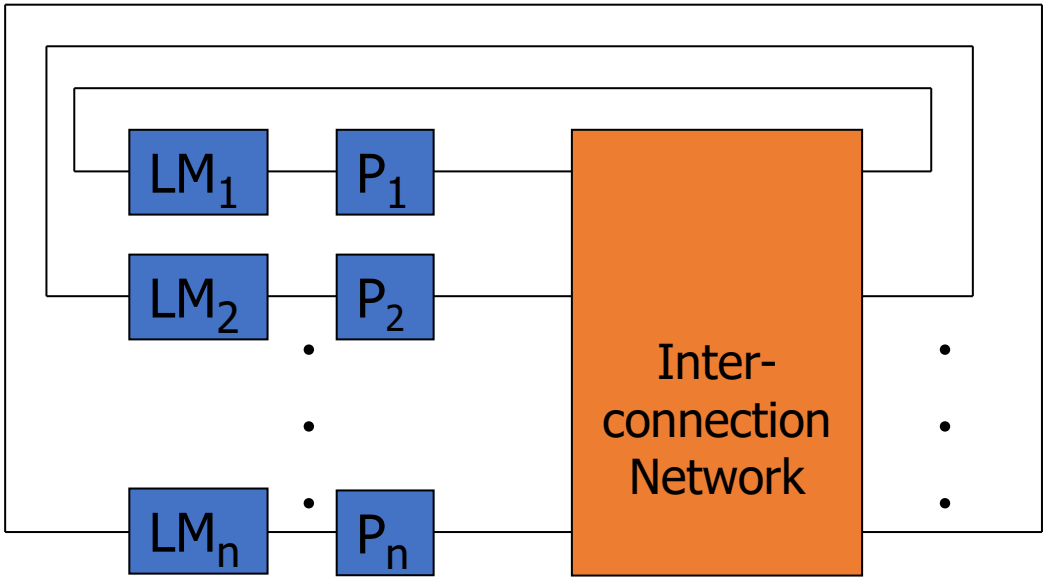
- Comparing results, the parallel solution is 214 times faster than the sequential, with the best theoretical speedup being 256 (since there are 256 processors). Thus the efficiency of the parallel solution is 214 / 256 = 83.6 %.

The NUMA Model

- Shared memories, but access time depends on the location of the data item.
- The shared memory is distributed among the processors as local memories, but each of these is still accessible by all processors (with varying access times).
- Memory access is fastest from the locally-connected processor, with the interconnection network adding delays for other processor accesses.
- Additionally, there may be global memory in a multiprocessor system, with two separate interconnection networks, one for clusters of processors and their cluster memories, and another for the global shared memories.

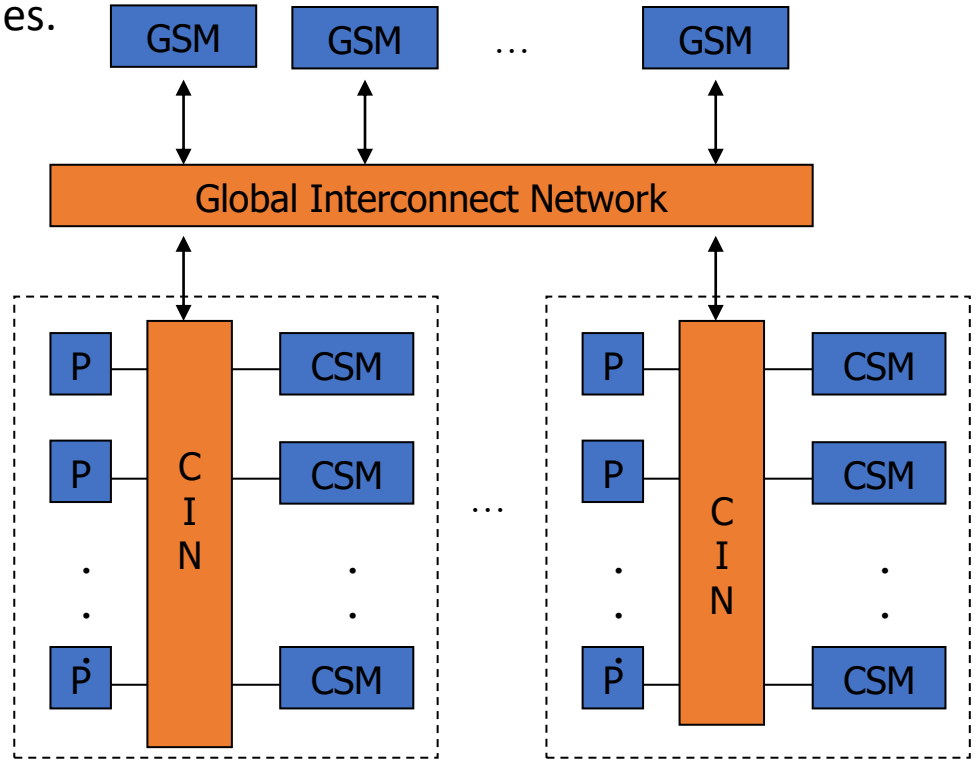
Shared Local Memories

- The shared memory is physically distributed to all processors, called local memories. The collection of all local Memories forms a global address space accessible by all processors.
- It is faster to access a local memory with a local processor. The access of remote memory attached to other processors takes longer time due to the added delay through the interconnection network.



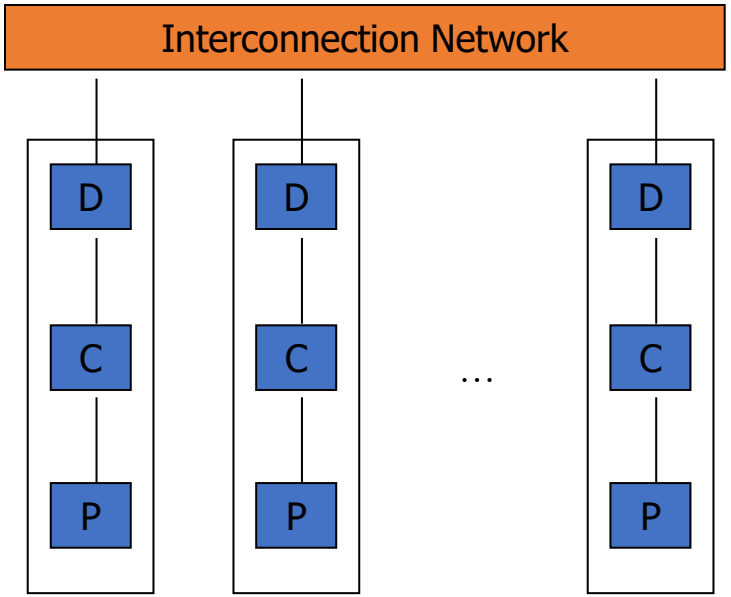
Hierarchical Cluster Model

- In the hierarchically structured multiprocessor the processors are divided into several cluster. Each cluster is itself an UMA or a NUMA multiprocessor.
- The clusters are connected to global shared-memory modules. The entire system is considered a NUMA multiprocessor.
- All processors belonging to the same cluster are allowed to uniformly access the cluster shared-memory modules.



The COMA Model

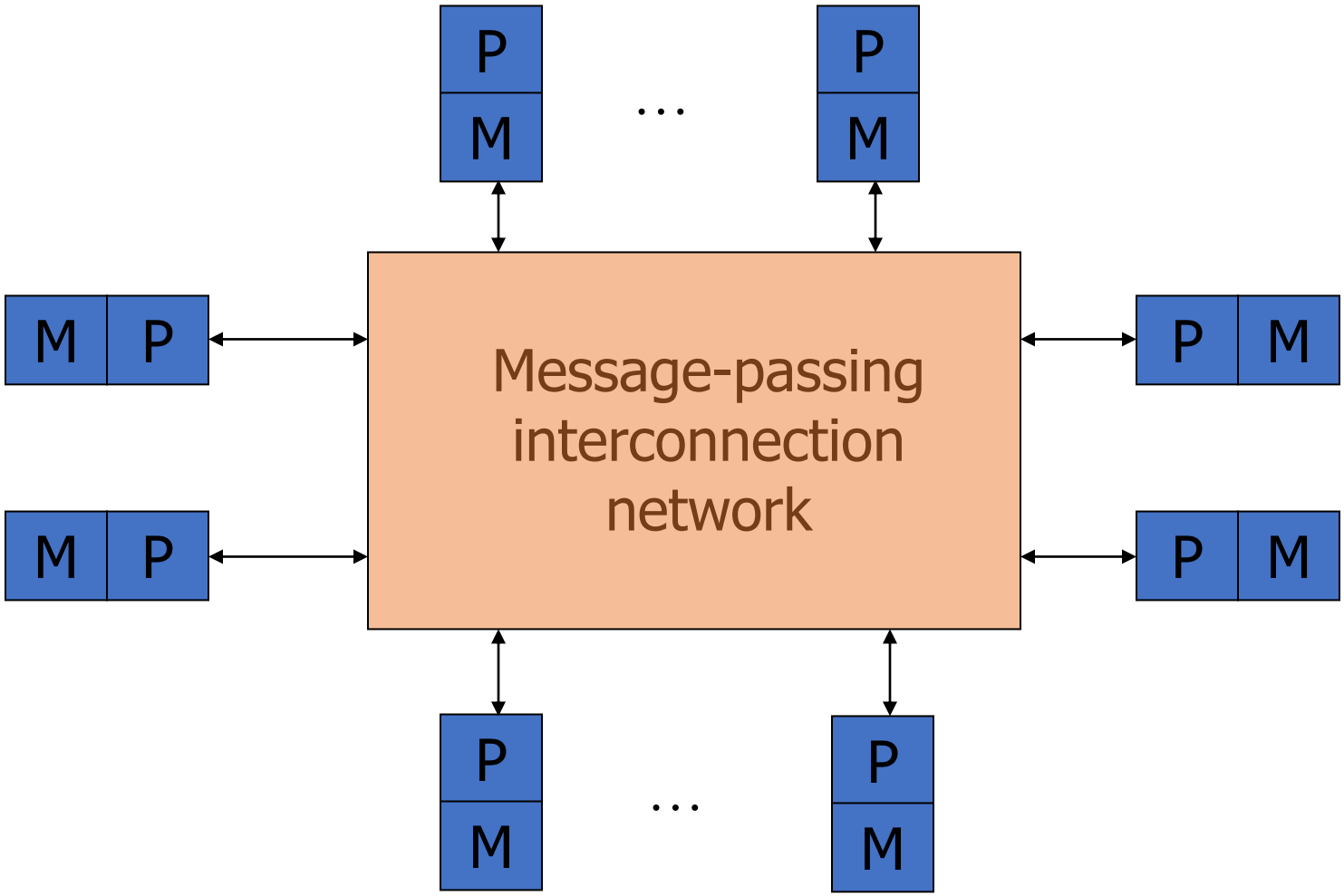
- In the COMA(Cache-Only Memory Architecture) model, processors only have cache memories; the caches, taken together, form a global address space.
- Each cache has an associated directory that aids remote machines in their lookups; hierarchical directories may exist in machines based on this model.
- Initial data placement is not critical, as cache blocks will eventually migrate to where they are needed.



Distributed-Memory Multicomputers

- This system consists of multiple computers, often called nodes, inter-connected by a message-passing network. Each node is an autonomous computer consisting of a processor, local memory and sometimes attached disks or I/O peripherals.
- The message-passing network provides point-to-point static connections among the nodes.
- All local memories are private and are accessible only by local processors. For this reason, traditional multicomputers have also been called no-remote-memory-access(NORMA) machines.
- Internode communication is carried out by passing messages through the static connection network. With advances in interconnection and network technologies, this model of computing has gained importance.

Distributed-Memory Multicomputers



Programming Environments

- Programmability depends on the programming environment provided to the users.
- Conventional computers are used in a sequential programming environment with tools developed for a uniprocessor computer.
- Parallel computers need parallel tools that allow specification or easy detection of parallelism and operating systems that can perform parallel scheduling of concurrent events, shared memory allocation, and shared peripheral and communication links.
- **Implicit Parallelism:**
- **Explicit Parallelism**

Programming Environments

- **Implicit Parallelism:**
  - Use a conventional language (like C, Fortran, Lisp, or Pascal) to write the program.
  - Use a parallelizing compiler to translate the source code into parallel code.
  - The compiler must detect parallelism and assign target machine resources.
  - Success relies heavily on the quality of the compiler.
- **Explicit Parallelism**
  - Programmer write explicit parallel code using parallel dialects of common languages.
  - Compiler has reduced need to detect parallelism, but must still preserve existing parallelism and assign target machine resources.

System Attributes to Performance

- Performance depends on
  - hardware technology
  - architectural features
  - efficient resource management
  - algorithm design
  - data structures
  - language efficiency
  - programmer skill
  - compiler technology

Performance Indicators

- Turnaround time depends on:
  - disk and memory accesses
  - input and output
  - compilation time
  - operating system overhead
  - CPU time
- Since I/O and system overhead frequently overlaps processing by other programs, it is fair to consider only the CPU time used by a program, and the user CPU time is the most important factor.

Clock Rate and CPI

- CPU is driven by a clock with a constant *cycle time* τ (usually measured in nanoseconds).
- The inverse of the cycle time is the *clock rate* ($f = 1/\tau$, measured in megahertz).
- The size of a program is determined by its *instruction count*, I_c, the number of machine instructions to be executed by the program.
- Different machine instructions require different numbers of clock cycles to execute. CPI (*cycles per instruction*) is thus an important parameter.

Average CPI

- It is easy to determine the average number of cycles per instruction for a particular processor if we know the frequency of occurrence of each instruction type.

- Of course, any estimate is valid only for a specific set of programs (which defines the instruction mix), and then only if there are sufficiently large number of instructions.

- In general, the term CPI is used with respect to a particular instruction set and a given program mix.

Performance Factors (1)

- The time required to execute a program containing I_c instructions is just $T = I_c \times CPI \times \tau$.

- Each instruction must be fetched from memory, decoded, then operands fetched from memory, the instruction executed, and the results stored.

- The time required to access memory is called the memory cycle time, which is usually *k* times the processor cycle time τ. The value of *k* depends on the memory technology and the processor-memory interconnection scheme.

Performance Factors (2)

- The processor cycles required for each instruction (CPI) can be attributed to
  - cycles needed for instruction decode and execution (p), and
  - cycles needed for memory references ($m \times k$).

- The total time needed to execute a program can then be rewritten as $T = I_c \times (p + m \times k) \times \tau$.

System Attributes

- The five performance factors (I_c, p, m, k, τ) are influenced by four system attributes:
  - instruction-set architecture (affects I_c and p)
  - compiler technology (affects I_c and p and m)
  - CPU implementation and control (affects $p \times \tau$)
  - cache and memory hierarchy (affects memory access latency, $k \times \tau$)

- Total CPU time can be used as a basis in estimating the execution rate of a processor.

MIPS Rate

- If C is the total number of clock cycles needed to execute a given program, then total CPU time can be estimated as $T = C \times \tau = C / f$.

- Other relationships are easily observed:
  - CPI = C / I_c
  - $T = I_c \times CPI \times \tau$
  - $T = I_c \times CPI / f$

- Processor speed is often measured in terms of *millions of instructions per second*, frequently called the MIPS rate of the processor.

MIPS Rate

$$MIPS\,rate = \frac{I_c}{T \times 10^6} = \frac{f}{CPI \times 10^6} = \frac{f \times I_c}{C \times 10}$$

- The MIPS rate is directly proportional to the clock rate and inversely proportion to the CPI.
- All four system attributes (instruction set, compiler, processor, and memory technologies) affect the MIPS rate, which varies also from program to program.

Throughput Rate

- The number of programs a system can execute per unit time, W_s , in programs per second.
- CPU throughput, W_p, is defined as

$$W_p = \frac{f}{I_c \times CPI}$$

- In a multiprogrammed system, the system throughput is often less than the CPU throughput.

Consider the execution of an object code with 200,000 instructions on a 40 MHz processor. The program instruction mix is as follows:

Instruction Type	CPI	Instruction mix
Arithmetic and logic	1	60%
Load/Store with cache hit	2	18%
Branch	4	12%
Memory reference with cache miss	8	10%

1. Calculate the average CPI when program is executed on uniprocessor with the above trace result.
2. Calculate the corresponding MIPS rate based in the CPI obtained.

$$MIPS_{rate} = \frac{f}{CPI * 10^6} = \frac{40 * 10^6}{2.24 * 10^6} = 17.86$$

$I_c = 200000$ and $f = 40$ MHz

$$CPI_{avg} = \frac{\sum I_c.CPI}{\sum I_c}$$

$\sum I_c.CPI$ = 120,000 + 72,000 + 96,000 + 160,000 =448,0000

$\sum I_c$ = 120,000 + 36,000 + 24,000 + 20,000 =220,0000

$CPI_{avg} =$ **2.24**

$$MIPS_{rate} = \frac{f}{CPI * 10^6} = \frac{40 * 10^6}{2.24 * 10^6} = 17.86$$