Microprocessor Lab

MCA- 209

SELF LEARNING MATERIAL

DIRECTORATE

OF DISTANCE EDUCATION

SWAMI VIVEKANAND SUBHARTI UNIVERSITY

MEERUT – 250 005,

UTTAR PRADESH (INDIA)

SLM Module Developed By :

Author:

Reviewed by :

Assessed by:

Study Material Assessment Committee, as per the SVSU ordinance No. VI (2)

Published by: Gayatri Sales

Typeset at: Micron Computers **Printed at:** Gayatri Sales, Meerut.

MICROPROCESSOR LAB

• Study of 8085 and 8086/8088 Kit.

• Assembly Language Programs for 8088 kit

(i) address and data transfer. (ii) addition, subtraction. (iii) block transfer. (iv) find greatest numbers. (v) find r's and (r-1)'s complements of signed and unsigned number

• Assembly Language Programs for 8086/8088

(i) Multiplication of two decimal/binary/hexadecimal/octal numbers. (ii) Division of two decimal/binary/hexadecimal/octal numbers. (iii) Conversion of lower case to upper case character.

• Test the performance of Booth's Algorithm for (i) Signed numbers. (ii) Unsigned numbers.

Unit-I

Introduction to Parallel computing

Before taking a toll on Parallel Computing, first let's take a look at the background of computations of a computer software and why it failed for the modern era.

Computer software were written conventionally for serial computing. This meant that to solve a problem, an algorithm divides the problem into smaller instructions. These discrete instructions are then executed on Central Processing Unit of a computer one by one. Only after one instruction is finished, next one starts.

Real life example of this would be people standing in a queue waiting for movie ticket and there is only cashier.Cashier is giving ticket one by one to the persons. Complexity of this situation increases when there are 2 queues and only one cashier.

So, in short Serial Computing is following:

In this, a problem statement is broken into discrete instructions.

Then the instructions are executed one by one.

Only one instruction is executed at any moment of time.

Look at point 3. This was causing a huge problem in computing industry as only one instruction was getting executed at any moment of time. This was a huge waste of hardware resources as only one part of the hardware will be running for a particular instruction and of time. As problem statements were getting heavier and bulkier, so does the amount of time in execution of those statements. Example of processors are Pentium 3 and Pentium 4.

Now let's come back to our real life problem. We could definitely say that complexity will decrease when there are 2 queues and 2 cashier giving tickets to 2 persons simultaneously. This is an example of Parallel Computing.

Parallel Computing

It is the use of multiple processing elements simultaneously for solving any problem. Problems are broken down into instructions and are solved concurrently as each resource which has been applied to work is working at the same time.

Advantages of Parallel Computing over Serial Computing are as follows:

It saves time and money as many resources working together will reduce the time and cut potential costs.

It can be impractical to solve larger problems on Serial Computing.

It can take advantage of non-local resources when the local resources are finite.

Serial Computing ‘wastes’ the potential computing power, thus Parallel Computing makes better work of hardware.

Types of Parallelism:

Bit-level parallelism: It is the form of parallel computing which is based on the increasing processor’s size. It reduces the number of instructions that the system must execute in order to perform a task on large-sized data.
Example: Consider a scenario where an 8-bit processor must compute the sum of two 16-bit integers. It must first sum up the 8 lower-order bits, then add the 8 higher-order bits, thus requiring two instructions to perform the operation. A 16-bit processor can perform the operation with just one instruction.

Instruction-level parallelism: A processor can only address less than one instruction for each clock cycle phase. These instructions can be re-ordered and grouped which are later on executed concurrently without affecting the result of the program. This is called instruction-level parallelism.

Task Parallelism: Task parallelism employs the decomposition of a task into subtasks and then allocating each of the subtasks for execution. The processors perform execution of sub tasks concurrently.

Why parallel computing?

The whole real world runs in dynamic nature i.e. many things happen at a certain time but at different places concurrently. This data is extensively huge to manage.

Real world data needs more dynamic simulation and modeling, and for achieving the same, parallel computing is the key.

Parallel computing provides concurrency and saves time and money.

Complex, large datasets, and their management can be organized only and only using parallel computing’s approach.

Ensures the effective utilization of the resources. The hardware is guaranteed to be used effectively whereas in serial computation only some part of hardware was used and the rest rendered idle.

Also, it is impractical to implement real-time systems using serial computing.

Applications of Parallel Computing:

Data bases and Data mining.

Real time simulation of systems.

Science and Engineering.

Advanced graphics, augmented reality and virtual reality.

Limitations of Parallel Computing:

It addresses such as communication and synchronization between multiple sub-tasks and processes which is difficult to achieve.

The algorithms must be managed in such a way that they can be handled in the parallel mechanism.

The algorithms or program must have low coupling and high cohesion. But it's difficult to create such programs.

More technically skilled and expert programmers can code a parallelism based program well.

Future of Parallel Computing: The computational graph has undergone a great transition from serial computing to parallel computing. Tech giant such as Intel has already taken a step towards parallel computing by employing multicore processors. Parallel computation will revolutionize the way computers work in the future, for the better good. With all the world connecting to each other even more than before, Parallel Computing does a better role in helping us stay that way. With faster networks, distributed systems, and multi-processor computers, it becomes even more necessary.

Parallelism in Uniprocessor Systems

Uniprocessor system

A uniprocessor system is defined as a computer system that has a single central processing unit that is used to execute computer tasks. As more and more modern software is able to make use of multiprocessing architectures, such as SMP and MPP, the term uniprocessor is therefore used to distinguish the class of computers where all processing tasks share a single CPU. Most desktop computers are now shipped with multiprocessing architectures.

Parallel computer structures

Parallel processing has been developed as an effective technology in modern computers to meet the demand for higher performance, lower cost and accurate results in real-life applications. Concurrent events are common in today's computers due to the practice of multiprogramming, multiprocessing, or multicomputing.

Modern computers have powerful and extensive software packages. To analyze the development of the performance of computers, first we have to understand the basic development of hardware and software.

Computer Development Milestones − There is two major stages of development of computer - mechanical or electromechanical parts. Modern computers evolved after the introduction of electronic components. High mobility electrons in electronic computers replaced the operational parts in mechanical computers. For information

transmission, electric signal which travels almost at the speed of a light replaced mechanical gears or levers.

Elements of Modern computers − A modern computer system consists of computer hardware, instruction sets, application programs, system software and user interface.

The computing problems are categorized as numerical computing, logical reasoning, and transaction processing. Some complex problems may need the combination of all the three processing modes.

Evolution of Computer Architecture − In last four decades, computer architecture has gone through revolutionary changes. We started with Von Neumann architecture and now we have multicomputers and multiprocessors.

Performance of a computer system − Performance of a computer system depends both on machine capability and program behavior. Machine capability can be improved with better hardware technology, advanced architectural features and efficient resource management. Program behavior is unpredictable as it is dependent on application and run-time conditions

Multiprocessors and Multicomputers

In this section, we will discuss two types of parallel computers −

Multiprocessors

Multicomputers

Shared-Memory Multicomputers

Three most common shared memory multiprocessors models are −

Uniform Memory Access (UMA)

In this model, all the processors share the physical memory uniformly. All the processors have equal access time to all the memory words. Each processor may have a private cache memory. Same rule is followed for peripheral devices.

When all the processors have equal access to all the peripheral devices, the system is called a symmetric multiprocessor. When only one or a few processors can access the peripheral devices, the system is called an asymmetric multiprocessor.

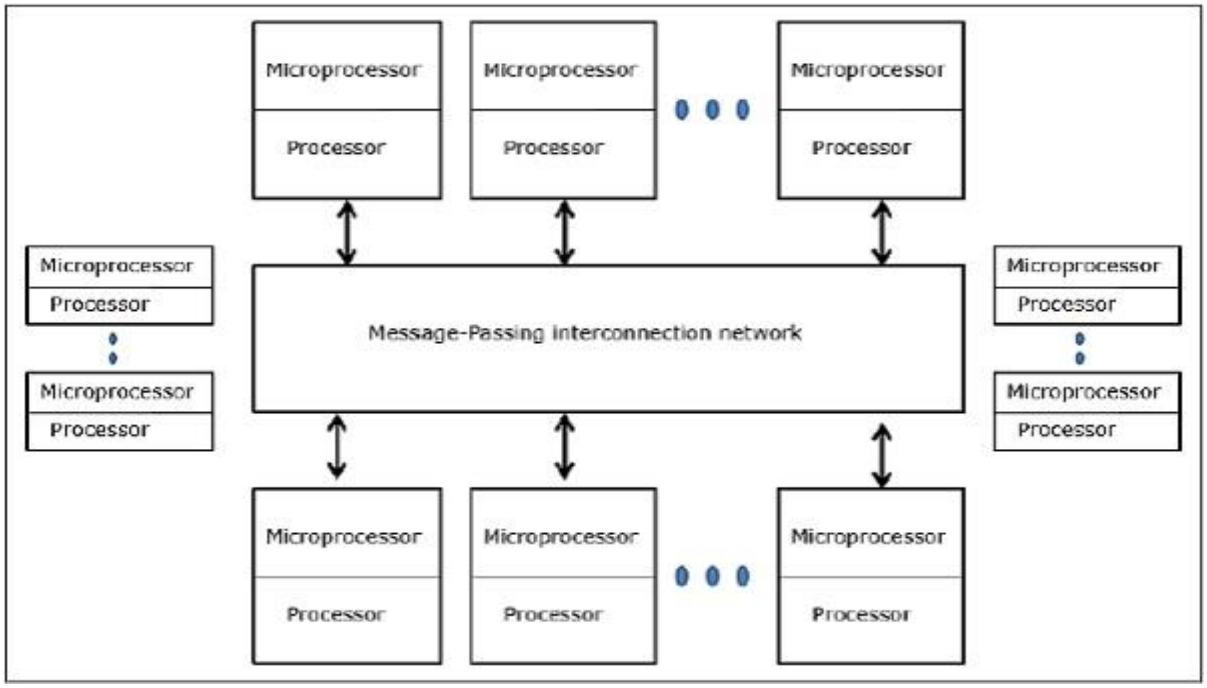
Non-uniform Memory Access (NUMA)

In NUMA multiprocessor model, the access time varies with the location of the memory word. Here, the shared memory is physically distributed among all the processors, called local memories. The collection of all local memories forms a global address space which can be accessed by all the processors.

Cache Only Memory Architecture (COMA)

The COMA model is a special case of the NUMA model. Here, all the distributed main memories are converted to cache memories.

Distributed - Memory Multicomputers − A distributed memory multicomputer system consists of multiple computers, known as nodes, inter-connected by message passing network. Each node acts as an autonomous computer having a processor, a local memory and sometimes I/O devices. In this case, all local memories are private and are accessible only to the local processors. This is why, the traditional machines are called no-remote-memory-access (NORMA) machines.



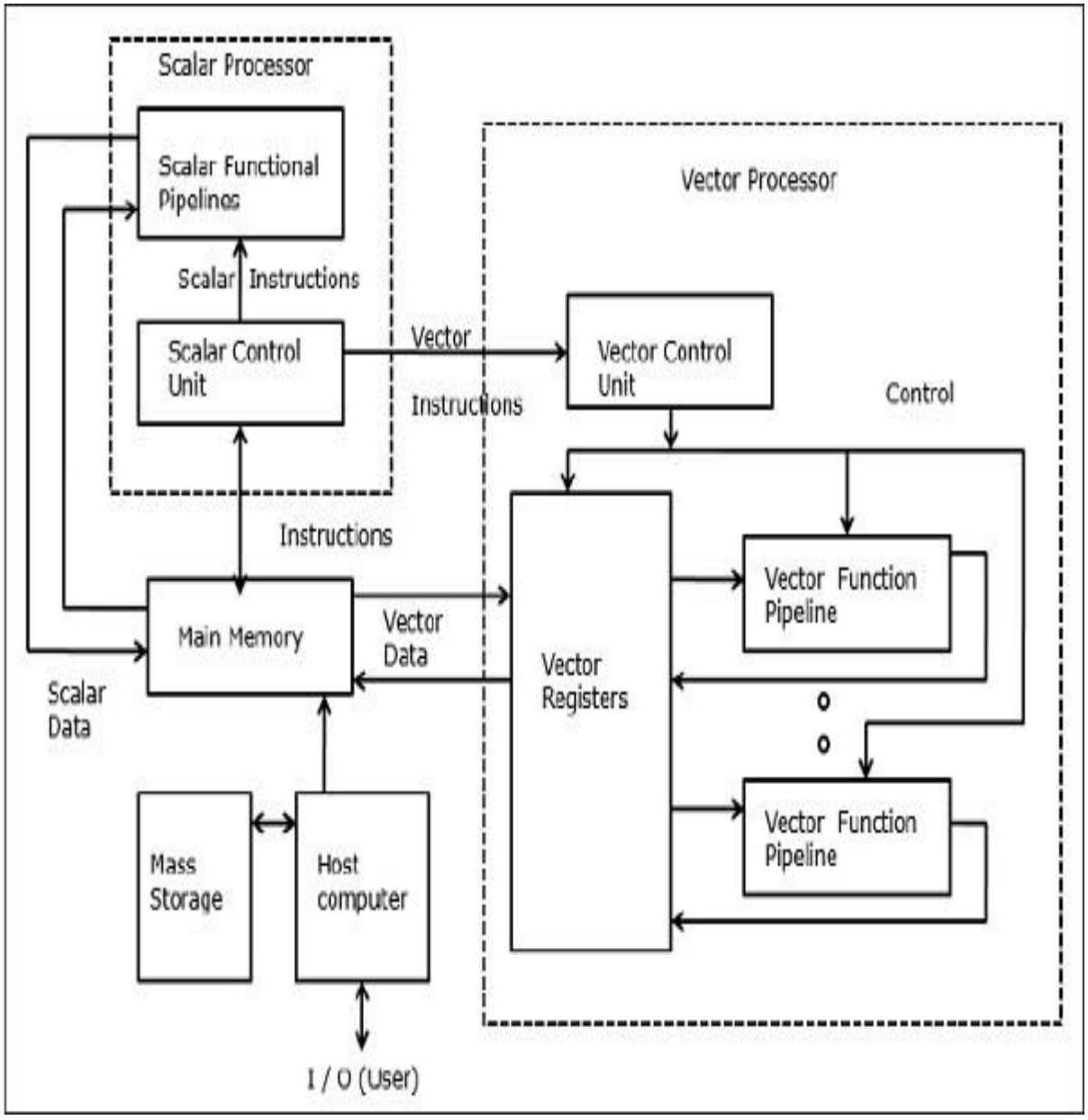
Multivector and SIMD Computers

In this section, we will discuss supercomputers and parallel processors for vector processing and data parallelism.

Vector Supercomputers

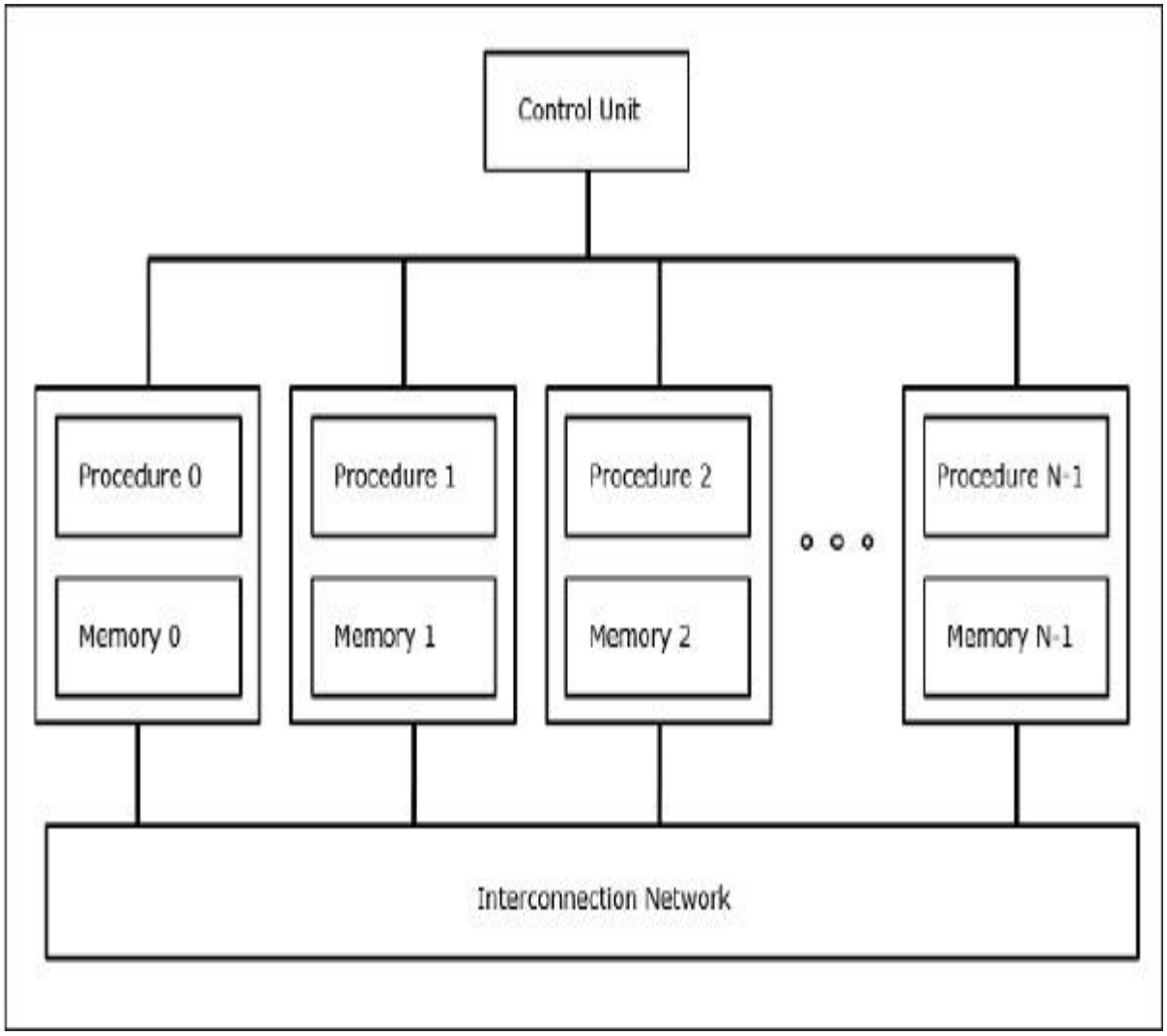
In a vector computer, a vector processor is attached to the scalar processor as an optional feature. The host computer first loads program and data to the main memory. Then the scalar control unit decodes all the instructions. If the decoded instructions are scalar operations or program operations, the scalar processor executes those operations using scalar functional pipelines.

On the other hand, if the decoded instructions are vector operations then the instructions will be sent to vector control unit.



SIMD Supercomputers

In SIMD computers, ‘N’ number of processors are connected to a control unit and all the processors have their individual memory units. All the processors are connected by an interconnection network.



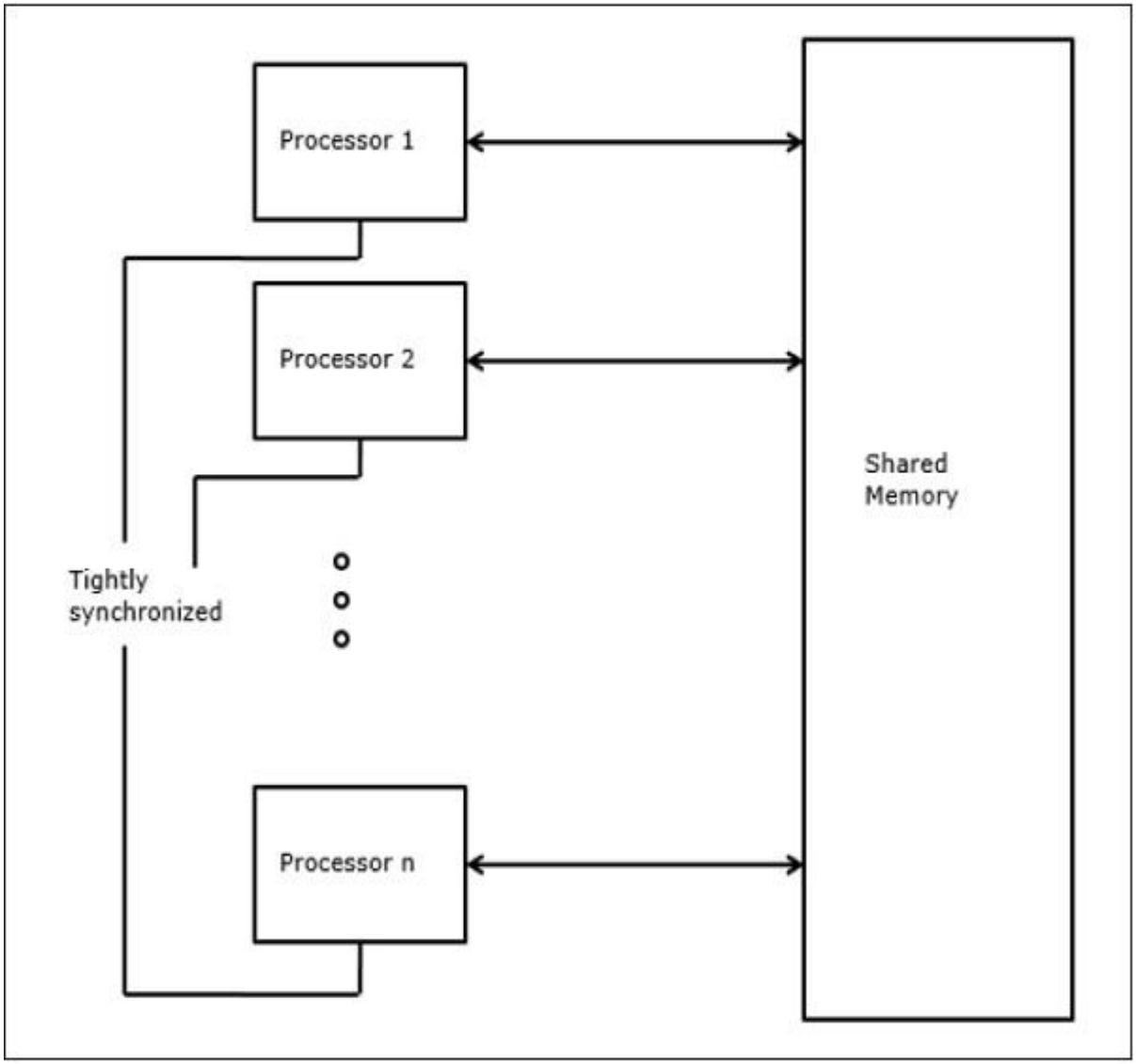
PRAM and VLSI Models

The ideal model gives a suitable framework for developing parallel algorithms without considering the physical constraints or implementation details.

The models can be enforced to obtain theoretical performance bounds on parallel computers or to evaluate VLSI complexity on chip area and operational time before the chip is fabricated.

Parallel Random-Access Machines

Sheperdson and Sturgis (1963) modeled the conventional Uniprocessor computers as random-access-machines (RAM). Fortune and Wyllie (1978) developed a parallel random-access-machine (PRAM) model for modeling an idealized parallel computer with zero memory access overhead and synchronization.



An N-processor PRAM has a shared memory unit. This shared memory can be centralized or distributed among the processors. These processors operate on a synchronized read-memory, write-memory and compute cycle. So, these models specify how concurrent read and write operations are handled.

Following are the possible memory update operations −

Exclusive read (ER) − In this method, in each cycle only one processor is allowed to read from any memory location.

Exclusive write (EW) − In this method, at least one processor is allowed to write into a memory location at a time.

Concurrent read (CR) − It allows multiple processors to read the same information from the same memory location in the same cycle.

Concurrent write (CW) − It allows simultaneous write operations to the same memory location. To avoid write conflict some policies are set up.

VLSI Complexity Model

Parallel computers use VLSI chips to fabricate processor arrays, memory arrays and large-scale switching networks.

Nowadays, VLSI technologies are 2-dimensional. The size of a VLSI chip is proportional to the amount of storage (memory) space available in that chip.

We can calculate the space complexity of an algorithm by the chip area (A) of the VLSI chip implementation of that algorithm. If T is the time (latency) needed to execute the algorithm, then A.T gives an upper bound on the total number of bits processed through the chip (or I/O). For certain computing, there exists a lower bound, f(s), such that

A.T2 >= O (f(s))

Where A=chip area and T=time

Architectural Development Tracks

The evolution of parallel computers I spread along the following tracks −

Multiple Processor Tracks

Multiprocessor track

Multicomputer track

Multiple data track

Vector track

SIMD track

Multiple threads track

Multithreaded track

Dataflow track

In multiple processor track, it is assumed that different threads execute concurrently on different processors and communicate through shared memory (multiprocessor track) or message passing (multicomputer track) system.

In multiple data track, it is assumed that the same code is executed on the massive amount of data. It is done by executing same instructions on a sequence of data elements (vector track) or through the execution of same sequence of instructions on a similar set of data (SIMD track).

In multiple threads track, it is assumed that the interleaved execution of various threads on the same processor to hide synchronization delays among threads executing on different processors. Thread interleaving can be coarse (multithreaded track) or fine (dataflow track).

INTRODUCTION

Parallel processing has emerged as a key enabling technology in modern computers, driven by the everincreasing demand for higher performance, lower costs, and sustained productivity in real-life applications. Concurrent events are taking place in today's high performance computers due to the common practice of multiprogramming, multiprocessing, or multicomputing. Parallelism can be in the form of look ahead, pipelining vectorization concurrency, simultaneity, data parallelism, partitioning, interleaving, overlapping, multiplicity, replication, time sharing, space sharing, multitasking, multiprogramming, multithreading, and distributed computing at different processing levels. Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently

II. CLASSIFICATION

Flynn Classification: The four classifications defined by Flynn are based upon the number of concurrent instruction (or control) and data streams available in the architecture:

1. Single Instruction, Single Data stream (SISD) A sequential computer which exploits no parallelism in either the instruction or data streams. Single control unit (CU) fetches single Instruction Stream (IS) from memory. The CU then generates appropriate control signals to direct single processing element (PE) tooperate on single Data Stream (DS) i.e. one operation at a time Examples of SISD architecture are the traditional uniprocessor machines like a PC (currently manufactured PCs have multiple processors) or old mainframes.

2. Single Instruction, Multiple Data streams (SIMD) A computer which exploits multiple data streams against a single instruction stream to perform operations which may be naturally parallelized. For example, an array processor or GPU.

3. Multiple Instruction, Single Data stream (MISD) Multiple instructions operate on a single data stream. Uncommon architecture which is generally used for fault tolerance. Heterogeneous systems operate on the same data stream and must agree on the result. Examples include the Space Shuttle flight control computer.

4. Multiple Instruction, Multiple Data streams (MIMD) Multiple autonomous processors simultaneously executing different instructions on different data. Distributed systems are generally recognized to be MIMD architectures; either exploiting a single shared memory space or a distributed memory space. A multi-core superscalar processor is an MIMD processor.

PARALLEL PROCESSING APPLICATIONS

Parallel processing refers to the speeding up a computational task by dividing it into smaller jobs across multiple processors. Notable applications for parallel processing (also known as parallel computing) include computational astrophysics, geoprocessing (or seismic surveying), climate modeling, agriculture estimates, financial risk management, video color correction, computational fluid dynamics, medical imaging and drug discovery.

“They asked if I’d be willing to extend this into general parallel computing applications,” Hwu, who’s now considered a godfather of parallel computing, told Built In.

That effectively sparked the use of GPUs for general-purpose computing — and, eventually, for massively parallel systems as well. Believe it or not, the circuit your computer uses to render fancy graphics for video games and 3D animations is built from the same root architecture as the circuits that make possible accurate climate pattern prediction. Wild, huh? And GPUs’ parallel infrastructure continues to power the most powerful computers.

“If you look at the workhorses for the scientific community today, the new computers, like [IBM supercomputer] Summit, and also the next generation, like Aurora, they're largely based on this model now,” Hwu said.

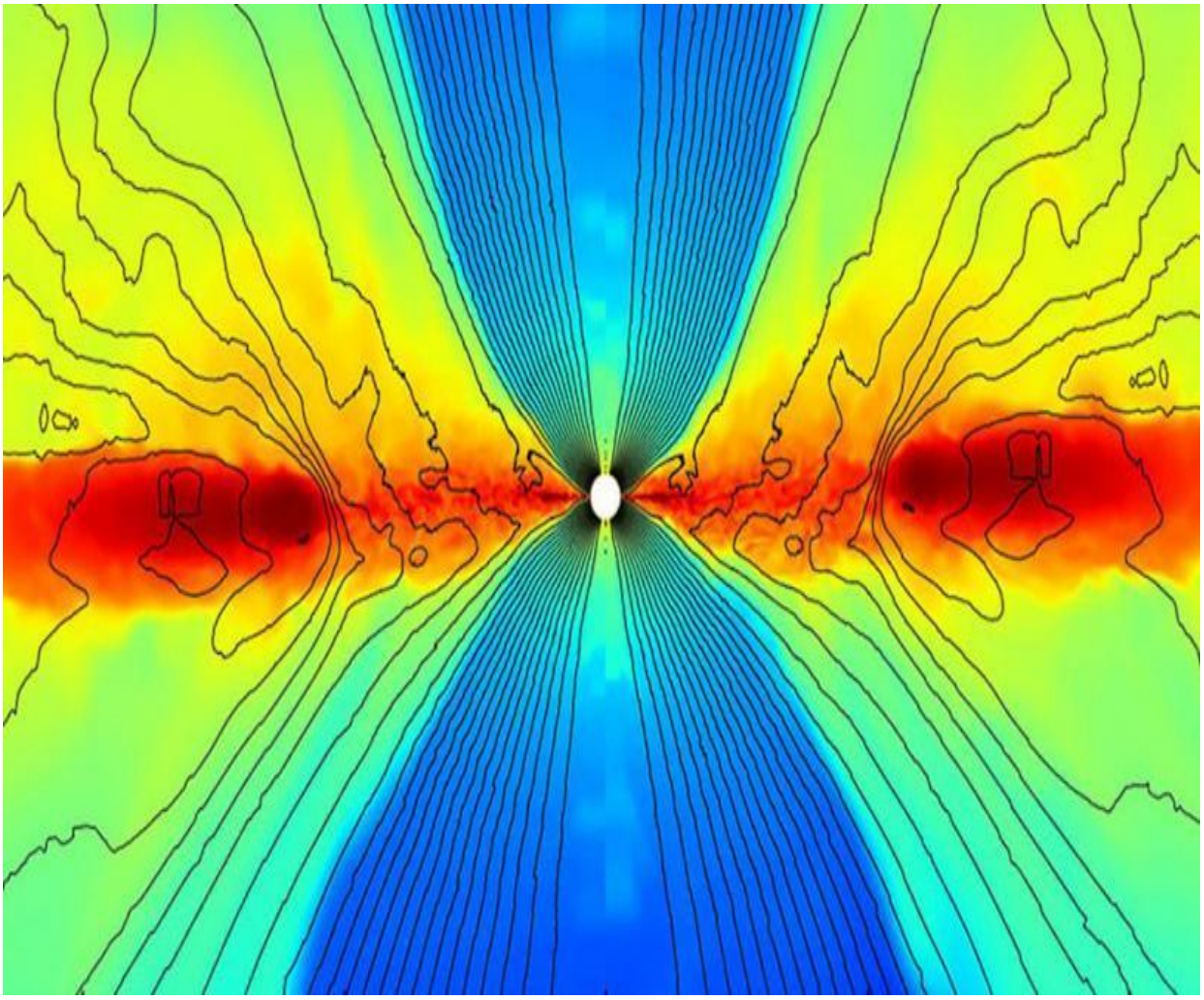
The model is a workhorse for medical and commercial applications, too, facilitating everything from drug discovery to interstellar simulations to post-production film techniques.

Here are just a few ways parallel computing is helping improve results and solve the previously unsolvable.

SCIENCE, RESEARCH & ENERGY

When you tap the Weather Channel app on your phone to check the day’s forecast, thank parallel processing. Not because your phone is running multiple applications — parallel computing shouldn’t be confused with concurrent computing — but because maps of climate and weather patterns require the serious computational heft of parallel.

Parallel computing is the backbone of other scientific studies, too, including astrophysic simulations, seismic surveying, quantum chromodynamics and more. Here’s a closer look at a few.



Northwestern University

NORTHWESTERN UNIVERSITY

Location: Evanston, Ill.

How it’s using parallel computing: Astronomy moves slowly. It can take millions of years for stars to collide, galaxies to merge or black holes to swallow astronomical objects — which is why astrophysicists must turn to computer simulations to study these kinds of processes. And such complex models demand massive compute power.

A recent breakthrough in the study of black holes, for example, happened courtesy of a parallel supercomputer. Researchers solved a four-decade-old mystery, proving that the innermost part of matter that orbits, then collapses into, black holes aligns with those black holes. That’s key to helping scientists better understand how this still-mysterious phenomenon behaves.

“These details around the black hole may seem small, but they enormously impact what happens in the galaxy as a whole,” said researcher Alexander Tchekhovskoy of Northwestern University, which partnered with the University of Amsterdam and the

University of Oxford on the study. “They control how fast the black holes spin and, as a result, what effect black holes have on their entire galaxies.”

DownUnderGeosolutions

DOWNUNDER GEOSOLUTIONS

Location: Houston, Texas

How it’s using parallel computing: One of the oil industry’s biggest players lives in suburban Houston and goes by the name Bubba. But Bubba’s no black-gold bigwig, it’s a supercomputer (among the fastest on the planet) owned by Australian geoprocessing company DownUnderGeoSolutions.

Seismic data processing has long helped provide a clearer picture of underground strata, an obvious must for industries like oil and gas. Supercomputing, though, is

practically de rigueur in energy excavation nowadays — especially as algorithms process massive amounts of data to help drillers mine difficult terrain, like salt domes. (French energy titan Total uses the world's most powerful commercial supercomputer.)

Bubba's backbone is formed by thousands of Intel Xeon Phi multiprocessors that are cooled in chilled oil baths, a technique that allows for extremely high-performance parallel processing. The hope is that by selling parallel power access to third-party companies, fewer energy outfits will feel compelled to build their own, less efficient systems.

University of Illinois

UNIVERSITY OF ILLINOIS

Location: Urbana-Champaign, Ill.

How it's using parallel computing: Every month, the U.S. Department of Agriculture estimates supply and demand figures for a number of major crops. The crucially important forecasts can impact everyone from legislators striving to stabilize markets to farmers who want to manage their finances.

Last year, researchers at U of I's Department of Natural Resources and Environmental Sciences topped the feds' industry-standard forecast by incorporating

more data — crop growth calculations and seasonal climate information as well as satellite figures — which it then crunched using machine learning algorithms processed by the university's parallel-data supercomputer, the petascale Blue Waters. Their prediction quickly proved more accurate by nearly five bushels per acre.

In 2019, the team turned its predictive eye to Australian wheat yields with similarly impressive results.

Commerce

THE COMMERCIAL WORLD

Even though parallel computing is often the domain of academic and government research institutions, the commercial world has definitely taken notice.

"The banking industry, investment industry traders, cryptocurrency — those are the big communities that are using a lot of GPUs for making money," Hwu said.

Parallel computing also has roots in the entertainment world — no surprise given that GPUs were first designed for heavy graphics loads. It's also a boon to industries that rely on computational fluid dynamics, a mechanical analysis that has several big commercial applications. Here's a closer look.

Wells Fargo

WELLS FARGO

Location: San Francisco, Calif.

How it's using parallel computing: Nearly every major aspect of today's banking, from credit scoring to risk modeling to fraud detection, is GPU-accelerated. In a way, the departure from traditional CPU-powered analysis was inevitable. GPU offloading came of age around 2008, just as lawmakers ushered in several rounds of post-crash financial legislation. "It's not uncommon now to find a bank with tens of thousands of Tesla GPUs," Xcelerit co-founder HichamLahlou told The Next Platform in 2017. "And this wouldn't have been the case without that mandatory push from regulation."

One early adopter was JPMorgan Chase, which announced in 2011 that its switch from CPU-only to GPU-CPU hybrid processing had improved risk calculations at its data centers by 40 percent and netted 80 percent savings. More recently, Wells Fargo used Nvidia's GPUs for processes as varied as accelerating AI models for liquidity risk and virtualizing its desktop infrastructure.

GPUs also found themselves at the center of a very 2019 financial trend: the crypto-mining craze. But chip sales have since stabilized after that particular boom and bust.

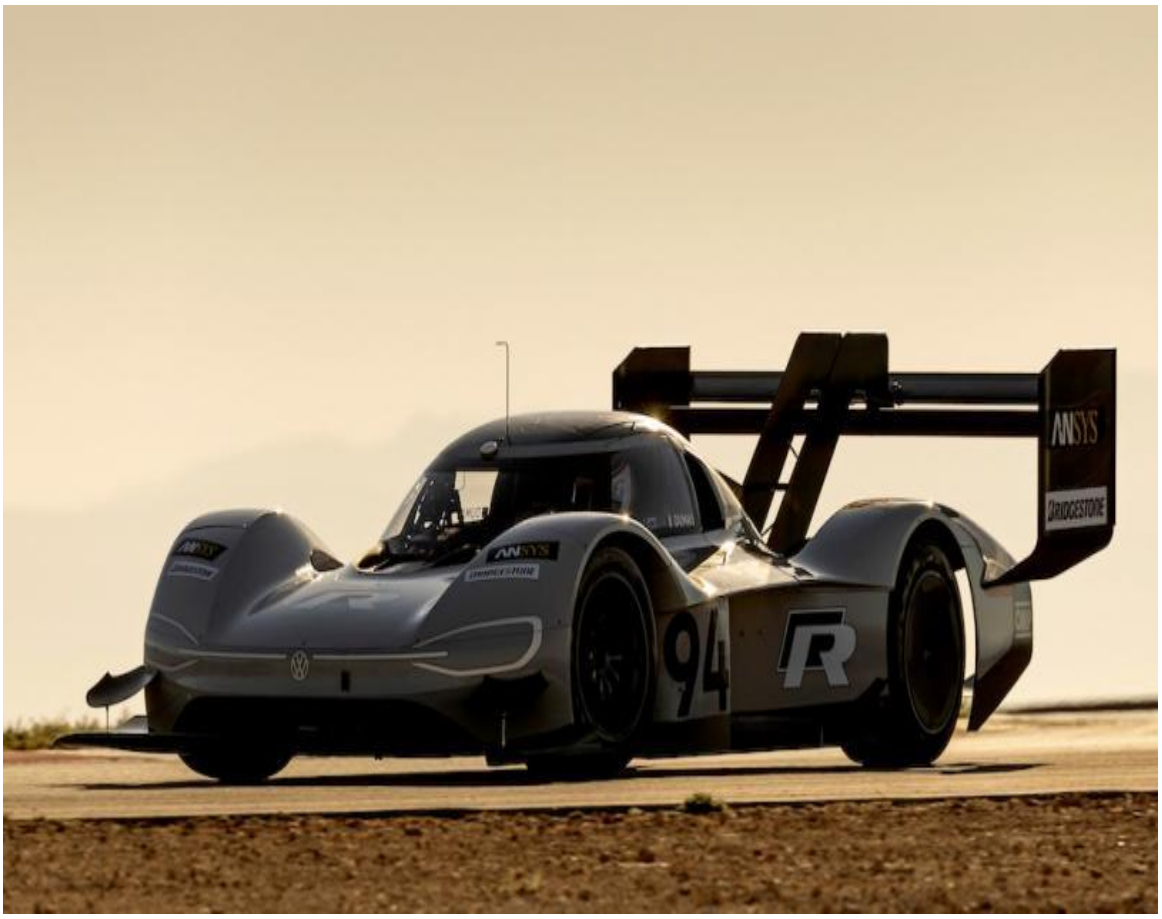
Blackmagic Design

BLACKMAGIC DESIGN

Location: Port Melbourne, Australia

How it's using parallel computing: If you saw either Brad Pitt's character working out his intergalactic daddy issues in Ad Astra or John Wick's latest round of elaborately

choreographed assassin dispatch, you also saw the handiwork of parallel processing. Both were colored using the Blackmagic Design's DaVinci Resolve Studio, one of a handful of Hollywood-standard post-production suites (including Adobe Effects and Avid Media Composer) that incorporates GPU-accelerated tools. "The high-quality rendering based on what they call the ray-tracing technique are all using some of these processors now," Hwu said. Color correction and 3D animation, he added, both commonly use GPU parallel processing.



Volkswagen

VOLKSWAGEN

Location: Wolfsburg, Germany

How it's using parallel computing: When French driver Romaine Dumas drove an electric Volkswagen prototype to auto-racing glory last year — smashing the Pikes Peak climb record by completing the first ever sub-eight-minute finish at the legendary track, kicking off a barnstorming tour of more all-time best finishes — the win was arguably as notable for computing as it was for electric automobiles.

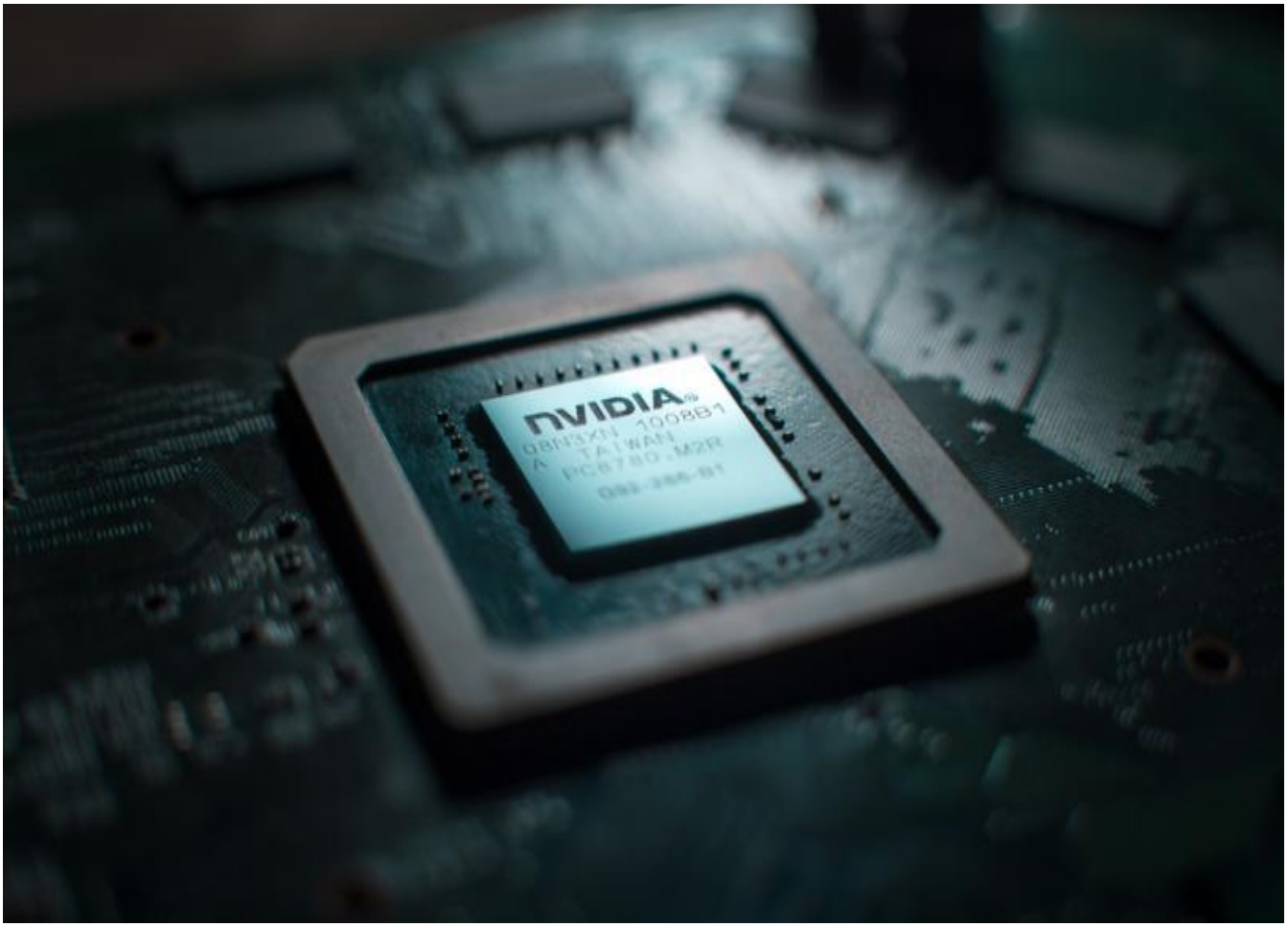
Engineers relied on Anasys Fluent software in at least two key facets: running a virtual simulation of the course, and finding the ideal balance of low weight and aerodynamic drag loss for the battery cooling system.

Such cooling is one of a number of so-called computational fluid dynamics (CFD) simulations users can run on Ansys, a program that easily supports GPU acceleration. It’s one of the more headline-worthy examples of how high-powered parallel computing has become a go-to for all manner of CFD research in everything from numerical weather prediction combustion engine optimization.

Medicine & Drug Discovery

MEDICINE & DRUG DISCOVERY

Emergent technology is reshaping the medical landscape in countless ways, from virtual reality that ameliorates macular degeneration to developments in bioprinting tissue and organs to the countless ways Amazon is poised to further impact healthcare. Parallel computing has for years made its presence felt in this arena, but it’s poised to fuel even more breakthroughs. Here’s how.



Nvidia

NVIDIA

Location: Santa Clara, Calif.

How it’s using parallel computing: One of the first industries that saw a sea change thanks to parallel processing, particularly the GPU-for-general-computing revolution, was medical imaging. Today, a whole body of medical literature exists cataloging how the high computation and bandwidth led to vast improvements in speed and definition for, well, just about everything: MRI, CT, X-rays, optical tomography and more.

The next great leap in medical imaging will likely be similarly parallel-focused, and parallel pioneer Nvidia is at the forefront. Using the company’s recently released toolkit, radiologists can more easily access the powers of AI, which helps imaging systems handle growing amounts of data and computational weight. Dubbed Clara, the GPU-leveraging system reportedly lets doctors create imaging models with ten times less data than otherwise required. Among the institutions that have already signed on are Ohio State University and the National Institutes of Health.

Acellera

ACELLERA

Location: London

How it’s using parallel computing: If you think of parallel processing as a nesting doll, one of the innermost figures could be a life-saving drug. Parallel programming is an ideal architecture for running simulations of molecular dynamics, which has proven to be highly useful in drug discovery.

Medical research company Acellera has developed multiple programs that harness the powerful offloading infrastructure of GPUs: simulation code ACEMD and Python package HTMD. They’ve been used to perform simulations on some of the world’s most powerful computers, including a Titan run that helped scientists better understand how our neurotransmitters communicate. And Acellera has partnered with the likes of Janssen and Pfizer for pharma research.

Since advanced parallel computing allows for fine-grain study of molecular machinery, it could also have major applications in the study of genetic disease — something researchers are currently looking into.

Oak Ridge National Laboratory

OAK RIDGE NATIONAL LABORATORY

Location: Oak Ridge, Tenn.

How it's using parallel computing: Beyond image rendering and pharmaceutical research, parallel processing's herculean data-analytics power hold great promise for public health. Take one particularly harrowing epidemic: veteran suicide. Around 20 veterans have died by suicide every day since 2014, according to data from the Department of Veterans Affairs. The issue is one of precious few to garner genuine bipartisan attention.

After the VA developed a model that dove into veterans' prescription and refill patterns, researchers at the Oak Ridge National Laboratory were then able to run the algorithm on a high-performance computer 300 times faster than the VA's

capabilities. The hope is to eventually use IBM's fabled (and GPU-packed) Summit supercomputer to allow real-time risk alerts to be sent to doctors.

"We don't want veterans to walk into a clinic and get missed because someone hasn't been specifically trained to recognize these symptoms," said ORNL researcher EdmonBegoli. "We never want it to be too late to reach someone."

Pipelining Processing

Pipelining is the process of accumulating instruction from the processor through a pipeline. It allows storing and executing instructions in an orderly process. It is also known as pipeline processing.

Pipelining is a technique where multiple instructions are overlapped during execution. Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure. Instructions enter from one end and exit from another end.

Pipelining increases the overall instruction throughput.

In pipeline system, each segment consists of an input register followed by a combinational circuit. The register is used to hold data and combinational circuit performs operations on it. The output of combinational circuit is applied to the input register of the next segment.

Pipeline system is like the modern day assembly line setup in factories. For example in a car manufacturing industry, huge assembly lines are setup and at each point, there are robotic arms to perform a certain task, and then the car moves on ahead to the next arm.

Types of Pipeline

It is divided into 2 categories:

Arithmetic Pipeline

Instruction Pipeline

Arithmetic Pipeline

Arithmetic pipelines are usually found in most of the computers. They are used for floating point operations, multiplication of fixed point numbers etc. For example: The input to the Floating Point Adder pipeline is:

X = A*2^a

Y = B*2^b

Here A and B are mantissas (significant digit of floating point numbers),
while a and b are exponents.

The floating point addition and subtraction is done in 4 parts:

Compare the exponents.

Align the mantissas.

Add or subtract mantissas

Produce the result.

Registers are used for storing the intermediate results between the above operations.

Instruction Pipeline

In this a stream of instructions can be executed by
overlapping fetch, decode and execute phases of an instruction cycle. This type of technique is used to increase the throughput of the computer system.

An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline. Thus we can execute multiple instructions simultaneously. The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.

Pipeline Conflicts

There are some factors that cause the pipeline to deviate its normal performance. Some of these factors are given belo

1. Timing Variations

All stages cannot take same amount of time. This problem generally occurs in instruction processing where different instructions have different operand requirements and thus different processing time.

2. Data Hazards

When several instructions are in partial execution, and if they reference same data then the problem arises. We must ensure that next instruction does not attempt to access data before the current instruction, because this will lead to incorrect results.

3. Branching

In order to fetch and execute the next instruction, we must know what that instruction is. If the present instruction is a conditional branch, and its result will lead us to the next instruction, then the next instruction may not be known until the current one is processed.

4. Interrupts

Interrupts set unwanted instruction into the instruction stream. Interrupts effect the execution of instruction.

5. Data Dependency

It arises when an instruction depends upon the result of a previous instruction but this result is not yet available.

Advantages of Pipelining

The cycle time of the processor is reduced.

It increases the throughput of the system

It makes the system reliable.

Disadvantages of Pipelining

The design of pipelined processor is complex and costly to manufacture.

The instruction latency is more.

Arithmetic Pipeline :

An arithmetic pipeline divides an arithmetic problem into various sub problems for execution in various pipeline segments. It is used for floating point operations,

multiplication and various other computations. The process or flowchart arithmetic pipeline for floating point addition is shown in the diagram.

Floating point addition using arithmetic pipeline :
The following sub operations are performed in this case:

Compare the exponents.

Align the mantissas.

Add or subtract the mantissas.

Normalise the result

First of all the two exponents are compared and the larger of two exponents is chosen as the result exponent. The difference in the exponents then decides how many times we must shift the smaller exponent to the right. Then after shifting of exponent, both the mantissas get aligned. Finally the addition of both numbers take place followed by normalisation of the result in the last segmentX=0.3214*10^3 and Y=0.4500*10^2

Explanation:

First of all the two exponents are subtracted to give 3-2=1. Thus 3 becomes the exponent of result and the smaller exponent is shifted 1 times to the right to give

Y=0.0450*10^3

Finally the two numbers are added to produce

Z=0.3664*10^3

As the result is already normalized the result remains the same.

2. Instruction Pipeline :
In this a stream of instructions can be executed by overlapping fetch, decode and execute phases of an instruction cycle. This type of technique is used to increase the throughput of the

computer system. An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline. Thus we can execute multiple instructions simultaneously. The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration.

In the most general case computer needs to process each instruction in following sequence of steps:

Fetch the instruction from memory (FI)

Decode the instruction (DA)

Calculate the effective address

Fetch the operands from memory (FO)

Execute the instruction (EX)

Store the result in the proper place

The flowchart for instruction pipeline is shown below

Here the instruction is fetched on first clock cycle in segment 1.
Now it is decoded in next clock cycle, then operands are fetched and finally the instruction is executed. We can see that here the fetch and decode phase overlap due to pipelining. By the time the first instruction is being decoded, next instruction is fetched by the pipeline.

In case of third instruction we see that it is a branched instruction. Here when it is being decoded 4th instruction is fetched simultaneously. But as it is a branched instruction it may point to some other instruction when it is decoded. Thus fourth instruction is kept on hold until the branched instruction is executed. When it gets executed then the fourth instruction is copied back and the other phases continue as usual.

Unit-II

Principles of designing pipelined processors

Internal forwarding and register tagging

A forwarding rule and its corresponding IP address represent the frontend configuration of a Google Cloud load balancer.

Note: Forwarding rules are also used for protocol forwarding, Classic VPN gateways, and Traffic Director to provide forwarding information in the control plane. This page only discusses forwarding rules in the context of Google Cloud load balancers.

Each forwarding rule references an IP address and one or more ports on which the load balancer accepts traffic. Some Google Cloud load balancers limit you to a predefined set of ports, and others let you specify arbitrary ports.

The forwarding rule also specifies an IP protocol. For Google Cloud load balancers, the IP protocol is always either TCP or UDP.

Depending on the load balancer type, the following is true:

A forwarding rule specifies a backend service, target proxy, or target pool.

A forwarding rule and its IP address are internal or external.

Also, depending on the load balancer and its tier, a forwarding rule is either global or regional.

Internal forwarding rules

Internal forwarding rules forward traffic that originates inside a Google Cloud network. The clients can be in the same Virtual Private Cloud (VPC) network as the backends, or the clients can be in a connected network.

Internal forwarding rules are used by two types of Google Cloud load balancers:

internal TCP/UDP load balancers

internal HTTP(S) load balancers

Internal TCP/UDP load balancers

With an internal TCP/UDP load balancer, the supported traffic type is IPv4, and the supported protocol is either TCP or UDP (not both).

Each internal TCP/UDP load balancer has at least one regional internal forwarding rule. The regional internal forwarding rules point to the load balancer's regional internal backend service. The following diagram shows how a forwarding rule fits into the Internal TCP/UDP Load Balancing architecture.

Internal TCP/UDP Load Balancing forwarding rule (click to enlarge)

The following diagram shows how the load balancer components fit within a subnet and region.

The internal forwarding rule must be in a region and a subnet. The backend service only needs to be in the region.

High-level internal TCP/UDP load balancer example (click to enlarge)

For more information about internal TCP/UDP load balancers

With an internal HTTP(S) load balancer, the supported traffic type is IPv4, and the supported protocol can be HTTP, HTTPS, or HTTP/2.

Each internal HTTP(S) load balancer has exactly one regional internal forwarding rule. The regional internal forwarding rule points to the load balancer's regional target HTTP or HTTPS proxy. The following diagram shows how a forwarding rule fits into the Internal HTTP(S) Load Balancing architecture.

Internal HTTP(S) Load Balancing forwarding rule (click to enlarge)

For more information about internal HTTP(S) load balancers, see the Internal HTTP(S) Load Balancing overview. For information about configuring internal HTTP(S) load balancers, see Preparing for Internal HTTP(S) Load Balancing setup.

External forwarding rules

External forwarding rules forward traffic that originates from the internet, outside of your VPC network.

External forwarding rules are used by the following Google Cloud load balancers:

external HTTP(S) load balancers

SSL proxy load balancers

TCP proxy load balancers

network load balancers

HTTP(S) load balancers

The external HTTP(S) load balancers support both Premium Tier and Standard Tier. The forwarding rule and IP address both depend on the tier that you select for the load balancer.

In an external HTTP(S) load balancer, a forwarding rule points to a target proxy.

In Premium Tier, an external HTTP(S) load balancer uses a global external IP address, which can be either IPv4 or IPv6, and a global external forwarding rule. You can provide a globally accessible application that directs end users to backends in the closest region and distributes traffic among multiple regions. Because a global external forwarding rule uses a single external IP address, you don't need to maintain separate DNS records in different regions or wait for DNS changes to propagate.

You can have two different global external IP addresses pointing to the same external HTTP(S) load balancer. For example, in Premium Tier, the global external IP address for one forwarding rule can be IPv4, and the global external IP address for a second forwarding rule can be IPv6. Both forwarding rules can point to the same target proxy. As a result, you can provide both an IPv4 and an IPv6 address for the same external HTTP(S) load balancer. For more information, see the IPv6 termination documentation.

In Standard Tier, an external HTTP(S) load balancer uses a regional external IP address, which must be IPv4, and a regional external forwarding rule. An external HTTP(S) load balancer in Standard Tier can only distribute traffic to backends within a single region.

The following diagram shows how a forwarding rule fits into the HTTP(S) Load Balancing architecture.

HTTP(S) Load Balancing forwarding rule (click to enlarge)

For more information about external HTTP(S) load balancers, see the HTTP(S) Load Balancing overview.

SSL proxy load balancers

An SSL proxy load balancer is similar to an external HTTP(S) load balancer because it can terminate SSL (TLS) sessions. SSL proxy load balancers do not support path-based redirection like external HTTP(S) load balancers, so they're best suited for handling SSL for protocols other than HTTPS, such as IMAP or WebSockets over SSL. For more information, see the SSL FAQ.

In an SSL proxy load balancer, a forwarding rule points to a target proxy.

SSL proxy load balancers support both Premium Tier and Standard Tier. The forwarding rule and IP address both depend on the tier that you select for the load balancer.

In Premium Tier, an SSL proxy load balancer uses a global external IP address, which can be either IPv4 or IPv6, and a global external forwarding rule. You can provide a globally accessible application that directs end users to backends in the closest region and distributes traffic among multiple regions. Because a global external forwarding rule uses a single external IP address, you don't have to maintain separate DNS records in different regions or wait for DNS changes to propagate.

It is possible to have two different global external IP addresses pointing to the same SSL proxy load balancer. For example, in Premium Tier, the global external IP address for one forwarding rule can be IPv4, and the global external IP address for a second forwarding rule can be IPv6. Both forwarding rules can point to the same target proxy. As a result, you can provide both an IPv4 and an IPv6 address for the same SSL proxy load balancer. For more information, see the IPv6 termination documentation.

In Standard Tier, an SSL proxy load balancer uses a regional external IP address, which must be IPv4, and a regional external forwarding rule. An SSL proxy load balancer in Standard Tier can only distribute traffic to backends within a single region.

A TCP proxy load balancer offers global TCP proxying capability, without SSL offload. TCP proxy load balancers support both Premium Tier and Standard Tier. The forwarding rule and IP address both depend on the tier that you select for the load balancer.

In a TCP proxy load balancer, a forwarding rule points to a target proxy.

In Premium Tier, a TCP proxy load balancer uses a global external IP address, which can be either IPv4 or IPv6, and a global external forwarding rule. You can provide a globally accessible application that directs end users to backends in the closest region and distributes traffic among multiple regions. Because a global external forwarding rule uses a single external IP address, you don't have to maintain separate DNS records in different regions or wait for DNS changes to propagate.

It is possible to have two different global external IP addresses pointing to the same TCP proxy load balancer. For example, in Premium Tier, the global external IP address for one forwarding rule can be IPv4, and the global external IP address for a second forwarding rule can be IPv6. Both forwarding rules can point to the same target proxy. As a result, you can provide both an IPv4 and an IPv6 address for the same TCP proxy load balancer. For more information, see the IPv6 termination documentation.

In Standard Tier, a TCP proxy load balancer uses a regional external IP address, which must be IPv4, and a regional external forwarding rule. A TCP proxy load

balancer in Standard Tier can only distribute traffic to backends within a single region.

The following diagram shows how a forwarding rule fits into the TCP Proxy Load Balancing architecture.

TCP Proxy Load Balancing forwarding rule (click to enlarge)

.

The network load balancers distribute either TCP or UDP traffic among backends in a single region, and they support both Premium Tier and Standard Tier. A network load balancer uses a regional external forwarding rule and a regional external IPv4 address (regardless of tier). The regional external IP address can be accessed anywhere on the internet.

For backend service-based network load balancers, the regional external forwarding rule points to a backend service. For target pool-based network load balancers, the forwarding rule points to a target pool.

To support backend instances in more than one region, you must create a network load balancer in each region. This is the case regardless of tier. The following figure shows Network Load Balancing with three load balancers for three different regions. Each load balancer has its own regional external forwarding rule with its own regional external IPv4 address.

In Network Service Tiers, the distinction between Standard Tier and Premium Tier depends on how far traffic is routed over the public internet:

Standard Tier: Offloads traffic as close as possible to the Google data center. This means that traffic is typically routed over the public internet for a longer distance, compared with Premium Tier.

Premium Tier: Routes traffic over Google's private network as far as possible before leaving Google Cloud to get to the end user.

Scheme	Target	Address type	Address scope	Address tier	Reservable address	Notes
EXTERNAL HTTP(S) Load Balancing SSL Proxy Load Balancing	Target HTTP proxy Target HTTPS proxy	External	Regional or global, matching the forward	Premium Tier: Global external IP addres	Yes, optional	IPv6 available with a global external

TCP Proxy Load Balancing	Target SSL proxy Target TCP proxy		ing rule	s and forwarding rule Standard Tier: Regional external IP address and forwarding rule		address (Premium Tier)
EXTERNAL Network Load Balancing	Backend service Target pool	External	Regional	Standard or Premium	Yes	IPv6 not supported
EXTERNAL Classic VPN	See the Classic VPN documentation	External	Regional	Cloud VPN doesn't have Network Service Tiers	Yes, required	IPv6 not supported
INTERNAL Internal TCP/UDP Load Balancing	Backend service	Internal	Regional	Premium	Yes, optional	Must be from the primary IP range of the associated subnet
INTERNAL_MANAGED	Target HTTP	Intern	Region	Premiu	Yes,	Must be from

Internal HTTP(S) Load Balancing	proxy Target HTTPS proxy	al	al	m	optional	the primary IP range of the associated subnet
INTERNAL_SELF_MANAGED Traffic Director	Target HTTP proxy Target gRPC proxy	Internal	Global	Not applicable	No	0.0.0.0, 127.0.0.1, or any RFC 1918 address is allowed

The internal load balancers (HTTP(S) and TCP/UDP) must use Google's private network, and they are therefore always in the Premium Tier. Internal load balancing is always regional.

Only the external load balancers (HTTP(S), TCP proxy, SSL proxy, and TCP/UDP network) can be routed over the public internet. You can choose whether your external load balancer is in the Premium Tier, using Google's private network, or in the Standard Tier, using the public internet.

Network Load Balancing is always regional, regardless of tier.

With Premium Tier, external HTTP(S) load balancers, TCP proxy load balancers, and SSL proxy load balancers are global. Their forwarding rules, IP addresses, and backend services are global. In Standard Tier, these load balancers are effectively regional. Their backend services remain global, but their forwarding rules and IP addresses are regional.

IP address specifications

The forwarding rule must have an IP address that your customers use to reach your load balancer. The IP address can be static or ephemeral.

A static IP address provides a single reserved IP address that you can point your domain to. If you ever need to delete your forwarding rule and re-add it, you can continue using the same reserved IP address.

An ephemeral IP address remains constant while the forwarding rule exists. When you choose an ephemeral IP address, Google Cloud associates an IP address with

your load balancer's forwarding rule. If you need to delete the forwarding rule and re-add it, the forwarding rule might receive a new IP address.

Depending on the load balancer type, the IP address can have various attributes. The following table summarizes the valid IP address configurations, based on the load balancing scheme and the target of the forwarding rule.

Scheme	Target	Ports must be specified	Behavior when ports are unspecified	Port requirements
EXTERNAL	Target HTTP proxy	Yes	N/A	80, 8080
EXTERNAL	Target HTTPS proxy	Yes	N/A	443
EXTERNAL	Target SSL proxy	Yes	N/A	25, 43, 110, 143, 195, 443, 465, 587, 700, 993, 995, 1883, 3389, 5222, 5432, 5671, 5672, 5900, 5901, 6379, 8085, 8099, 9092, 9200, and 9300
EXTERNAL	Target TCP proxy	Yes	N/A	25, 43, 110, 143, 195, 443, 465, 587, 700, 993, 995, 1883, 3389, 5222, 5432, 5671, 5672, 5900, 5901, 6379, 8085, 8099, 9092, 9200, and 9300
EXTERNAL	Target VPN Gateway	Yes	N/A	500, 4500
EXTERNAL	Target pool	No	All ports (1-65535) are forwarded	Must be contiguous
	Backend	Yes	N/A	A list of up to five (contiguous or non-

	service			contiguous) ports or a single port range (contiguous). You cannot specify both in the same forwarding rule. To configure all ports, use one of these methods: set --ports=1-65535 using the gcloud command line tool, or set portRange to 1-65535 using the API, or leave both ports and portRange fields empty in the API.
INTERNAL	Backend service	Yes	N/A	Up to five (contiguous or non-contiguous) or you can specify ALL
INTERNAL_MANAGED	Target HTTP proxy Target HTTPS proxy	Yes	N/A	One of 80 or 8080 443
INTERNAL_SELF_MANAGED	Target HTTP proxy Target HTTPS proxy	Yes	N/A	Must be a single value. Within a VPC network, no two forwarding rules for Traffic Director can have the same IP address and port specification.

Multiple forwarding rules with a common IP address

Two or more forwarding rules with the EXTERNAL load balancing scheme can share the same IP address if the following are true:

The ports used by each forwarding rule do not overlap.

The Network Service Tiers of each forwarding rule matches the Network Service Tiers of the external IP address.

Examples:

A network load balancer that accepts traffic on TCP port 79 and another network load balancer that accepts traffic on TCP port 80 can share the same regional external IP address.

You can use the same global external IP address for an external HTTP(S) load balancer (HTTP and HTTPS).

If the forwarding rule's load balancing scheme is INTERNAL or INTERNAL_MANAGED, multiple forwarding rules can use the same IP address. For more information, see Internal TCP/UDP Load Balancing overview.

If the forwarding rule's load balancing scheme is INTERNAL_SELF_MANAGED for Traffic Director, it must have a unique IP address.

Port specifications

The following table summarizes the valid port configurations, based on the load balancing scheme and the target of the forwarding rule.

Hazard detection

There are many definitions for hazard but the most common definition when talking about workplace health and safety is “A hazard is any source of potential damage, harm or adverse health effects on something or someone.”

The CSA Z1002 Standard "Occupational health and safety - Hazard identification and elimination and risk assessment and control" uses the following terms:

Harm – physical injury or damage to health.

Hazard – a potential source of harm to a worker.

Basically, a hazard is the potential for harm or an adverse effect (for example, to people as health effects, to organizations as property or equipment losses, or to the environment).

Please see the OSH Answers on Hazard and Risk for more information.

What is hazard identification?

Hazard identification is part of the process used to evaluate if any particular situation, item, thing, etc. may have the potential to cause harm. The term often used to describe the full process is risk assessment:

Identify hazards and risk factors that have the potential to cause harm (hazard identification).

Analyze and evaluate the risk associated with that hazard (risk analysis, and risk evaluation).

Determine appropriate ways to eliminate the hazard, or control the risk when the hazard cannot be eliminated (risk control).

Overall, the goal of hazard identification is to find and record possible hazards that may be present in your workplace. It may help to work as a team and include both people familiar with the work area, as well as people who are not – this way you have both the experienced and fresh eye to conduct the inspection.

When should hazard identification be done?

Hazard identification can be done:

During design and implementation

Designing a new process or procedure

Purchasing and installing new machinery

Before tasks are done

Checking equipment or following processes

Reviewing surroundings before each shift

While tasks are being done

Be aware of changes, abnormal conditions, or sudden emissions

During inspections

Formal, informal, supervisor, health and safety committee

After incidents

Near misses or minor events

Injuries

To be sure that all hazards are found:

Look at all aspects of the work and include non-routine activities such as maintenance, repair, or cleaning.

Look at the physical work environment, equipment, materials, products, etc. that are used.

Include how the tasks are done.

Look at injury and incident records.

Talk to the workers: they know their job and its hazards best.

Include all shifts, and people who work off site either at home, on other job sites, drivers, teleworkers, with clients, etc.

Look at the way the work is organized or done (include experience of people doing the work, systems being used, etc).

Look at foreseeable unusual conditions (for example: possible impact on hazard control procedures that may be unavailable in an emergency situation, power outage, etc.).

Determine whether a product, machine or equipment can be intentionally or unintentionally changed (e.g., a safety guard that could be removed).

Review all of the phases of the lifecycle.

Examine risks to visitors or the public.

Consider the groups of people that may have a different level of risk such as young or inexperienced workers, persons with disabilities, or new or expectant mothers.

What types of hazards are there?

A common way to classify hazards is by category:

biological – bacteria, viruses, insects, plants, birds, animals, and humans, etc.,

chemical – depends on the physical, chemical and toxic properties of the chemical,

ergonomic – repetitive movements, improper set up of workstation, etc.,

physical – radiation, magnetic fields, temperature extremes, pressure extremes (high pressure or vacuum), noise, etc.,

psychosocial – stress, violence, etc.,

safety – slipping/tripping hazards, inappropriate machine guarding, equipment malfunctions or breakdowns.

How do I know what is a hazard?

Another way to look at health and safety in your workplace is to ask yourself the following questions. These are examples only. You may find other items or situations that can be a hazard. List any item that should be examined. During the risk assessment process, the level of harm will be assessed.

What materials or situations do I come into contact with? Possibilities could include:

electricity

chemicals (liquids, gases, solids, mists, vapours, etc.)

temperature extremes of heat or cold (e.g., bakeries, foundries, meat processing)

ionizing/non-ionizing radiation (e.g., x-rays, ultraviolet (sun) rays)

oxygen deficiency

water

What materials or equipment could I be struck by?

moving objects (e.g., forklifts, overhead cranes, vehicles)

flying objects (e.g., sparks or shards from grinding)

falling material (e.g., equipment from above)

What objects or equipment could I strike or hit my body upon, or that part of my body might be caught in, on, or between?

stationary or moving objects

protruding objects

sharp or jagged edges

pinch points on machines (places where parts are very close together)

objects that stick out (protrude)

moving objects (conveyors, chains, belts, ropes, etc.)

What could I fall from? (e.g., falls to lower levels)

objects, structures, tanks, silos, lofts

ladders, overhead walkways

roofs

trees, cliffs

What could I slip or trip on? (e.g., falls on same level)

obstructions on floor, stairs

surface issues (wet, oily, icy)

footwear that is in poor condition

How could I overexert myself?

lifting

pulling

pushing

carrying

repetitive motions

What other situations could I come across?

unknown/unauthorized people in area

a potentially violent situation

working alone

confined space

missing/damaged materials

new equipment/procedure at work site

fire/explosion

chemical spill or release

Where can I find more information about hazards?

It may be necessary to research about what might be a hazard as well as how much harm that hazard might cause. Sources of information include:

Safety Data Sheets (SDSs).

Manufacturer's operating instructions, manuals, etc.

Test or monitor for exposure (occupational hygiene testing such as chemical or noise exposure).

Results of any job safety analysis.

Experiences of other organizations similar to yours.

Trade or safety associations.

Information, publications, alerts, etc. as published by reputable organizations, labour unions, or government agencies.

What if I am new to the workplace?

If you are new to your workplace, to learn about the hazards of your job, you can:

ask your supervisor

ask a member of the health and safety committee or your health and safety representative

ask about standard operating procedures and precautions for your job

check product labels and safety data sheets

pay attention to signs and other warnings in your work

watch for posters or instructions at the entrance of a chemical storage room to warn of hazardous products

ask about operating instructions, safe work procedures, processes, etc.

Job sequencing and collision prevention

Problem Statement

In job sequencing problem, the objective is to find a sequence of jobs, which is completed within their deadlines and gives maximum profit.

Solution

Let us consider, a set of n given jobs which are associated with deadlines and profit is earned, if a job is completed by its deadline. These jobs need to be ordered in such a way that there is maximum profit.

It may happen that all of the given jobs may not be completed within their deadlines.

Assume, deadline of ith job Ji is di and the profit received from this job is pi. Hence, the optimal solution of this algorithm is a feasible solution with maximum profit.

Thus, D(i)>0D(i)>0 for 1⩽i⩽n1⩽i⩽n.

Initially, these jobs are ordered according to profit,
i.e. p1⩾p2⩾p3⩾...⩾pnp1⩾p2⩾p3⩾...⩾pn.

Algorithm: Job-Sequencing-With-Deadline (D, J, n, k)

D(0) := J(0) := 0

k := 1

J(1) := 1 // means first job is selected

fori = 2 … n do

r := k

while D(J(r)) > D(i) and D(J(r)) ≠ r do

r := r – 1

if D(J(r)) ≤ D(i) and D(i) > r then

for l = k … r + 1 by -1 do

J(l + 1) := J(l)

J(r + 1) := i

k := k + 1

Analysis

In this algorithm, we are using two loops, one is within another. Hence, the complexity of this algorithm is O(n2)O(n2).

Example

Let us consider a set of given jobs as shown in the following table. We have to find a sequence of jobs, which will be completed within their deadlines and will give maximum profit. Each job is associated with a deadline and profit.

Solution

To solve this problem, the given jobs are sorted according to their profit in a descending order. Hence, after sorting, the jobs are ordered as shown in the following table.

Job	J2	J1	J4	J3	J5
Deadline	1	2	2	3	1
Profit	100	60	40	20	20

From this set of jobs, first we select J2, as it can be completed within its deadline and contributes maximum profit.

Next, J1 is selected as it gives more profit compared to J4.

In the next clock, J4 cannot be selected as its deadline is over, hence J3 is selected as it executes within its deadline.

The job J5 is discarded as it cannot be executed within its deadline.

Thus, the solution is the sequence of jobs (J2, J1, J3), which are being executed within their deadline and gives maximum profit.

Total profit of this sequence is 100 + 60 + 20 = 180

A collision avoidance system (CAS), also known as a pre-crash system, forward collision warning system, or collision mitigation system, is a motorcar safety system designed to prevent or reduce the severity of a collision.[2] In its basic form, a forward collision warning system monitors a vehicle's speed, the speed of the vehicle in front of it, and the distance between the vehicles, so that it can provide a warning to the driver if the vehicles get too close, potentially helping to avoid a crash.[3] Various technologies and sensors that are used include radar (all-weather) and sometimes laser (LIDAR) and cameras (employing image recognition) to detect an imminent crash. GPS sensors can detect fixed dangers such as approaching stop signs through a location database.[2][4][5][6] Pedestrian detection can also be a feature of these types of systems.

Collision avoidance systems range from widespread systems mandatory in some countries, such as autonomous emergency braking (AEB) in the EU, agreements

between car makers and safety officials to make crash avoidance systems eventually standard, such as in the United States,[7] to research projects including some manufacturer specific devices.

Advanced emergency braking system (AEBS) as defined by UN ECE regulation 131 is considered as: a system which can automatically detect a potential forward collision and activate the vehicle braking system to decelerate the vehicle with the purpose of avoiding or mitigating a collision.[8] UN ECE regulation 152 says deceleration can be 5 metres per second squared.[9]

Once an impending collision is detected, these systems provide a warning to the driver. When the collision becomes imminent, they can take action autonomously without any driver input (by braking or steering or both). Collision avoidance by braking is appropriate at low vehicle speeds (e.g. below 50 km/h (31 mph)), while collision avoidance by steering may be more appropriate at higher vehicle speeds if lanes are clear.[10] Cars with collision avoidance may also be equipped with adaptive cruise control, using the same forward-looking sensors.

AEB differs from forward collision warning: FCW alerts the driver with a warning but does not by itself brake the vehicle

Vector Processing

Sandra was writing a GIS program for computing Geographical Information Systems (GIS) data for 20,000 and rivers and lakes in the Midwest. The calculations require a large number of variables and data types including image data, map data, and data from satellite images. Sandra created 300 different variables to hold data for the various computations. When the program was executed, it had to be left overnight to complete. Her teammate Jen was running a similar program and it took only a fraction of the time. Jen said she was using a vector for processing the data.

Characteristics of a Vector

The two main characteristics of a vector are that it helps with fast processing and that the size of the vector does not need to be decided in advance.

Vector Increases Efficiency

If you have the following 5 integers: 20, 25, 30, 35, 40 and you need to add 10 to each of the integers, one method is to declare 5 variables to store each of the 5 integers. So, you have a = 20, b = 25, c = 30 and so on. The computer first finds 20, add 10 to 20; then find 25 and add 10 to 25 and so on.

Using the vector method, instead of having 5 separate variables, you could have one vector variable z. The vector saves all 5 numbers in its array: z[0] = 20; z[1] = 25; z[2] = 30 and so on. The processor considers the entire vector as one variable and instead of having to add 10 five separate times adds 10 at one time to all the elements in the vector, thereby increasing computing efficiency.

Vector is Dynamic

A vector is dynamic because you do not have to decide in advance the size of the vector, and it can increase or decrease in size dynamically, as the program runs, depending on the number of variables the vector needs to hold. Therefore, in Sandra's program, she does not need to know or decide in advance that the vector needs to hold 20,000 variables for image data.

She can simply declare a dynamic vector and add more variables as the program runs. She could customize the program so that a user can input the number of rivers and lakes and the vector will expand to hold the number. In the same way, she can programmatically eliminate a certain number of rivers and lakes and the vector will automatically get smaller to hold the reduced number.

A vector processor is a computer with parallel processors that have the capability for vector processing

Characteristics of Vector Processors

A vector processor is a CPU (Central Processing Unit) in a computer with parallel processors and the capability for vector processing. The main characteristic of a vector processor is that it makes use of the parallel processing capability of the processor where two or more processors operate concurrently. This makes it possible for:

the processors to perform multiple tasks simultaneously

or

for the task to be split into different subtasks handled by different processors and combined to get the result.

The vector processor considers all of the elements of the vector as one single element as it traverses through the vector in a single loop. Computers with vector processors find many uses that involve computation of massive amounts of data such as image processing, artificial intelligence, mapping the human genome, space simulations, seismic data, and hurricane predictions

SIMD array processors

Array Processor performs computations on large array of data. These are two types of Array Processors: Attached Array Processor, and SIMD Array Processor. These are explained as following below.

1. Attached Array Processor :
To improve the performance of the host computer in numerical computational tasks auxiliary processor is attatched to it.

Attached array processor has two interfaces:

Input output interface to a common processor.

Interface with a local memory.

Here local memory interconnects main memory. Host computer is general purpose computer. Attached processor is back end machine driven by the host computer.

The array processor is connected through an I/O controller to the computer & the computer treats it as an external interface.

2. SIMD array processor :

This is computer with multiple process unit operating in parallel Both types of array processors, manipulate vectors but their internal organization is different.

SIMD is a computer with multiple processing units operating in parallel.

The processing units are synchronized to perform the same operation under the control of a common control unit. Thus providing a single instruction stream, multiple data stream (SIMD) organization. As shown in figure, SIMD contains a set of identical processing elements (PES) each having a local memory M.

Each PE includes –

- ALU
- Floating point arithmetic unit
- Working registers

Master control unit controls the operation in the PEs. The function of master control unit is to decode the instruction and determine how the instruction to be executed. If the instruction is scalar or program control instruction then it is directly executed within the master control unit.

Main memory is used for storage of the program while each PE uses operands stored in its local memory.

Masking and Data routing

The main reason for applying masking to a data field is to protect data that is classified as personally identifiable information, sensitive personal data, or commercially sensitive data. However, the data must remain usable for the purposes of undertaking valid test cycles. It must also look real and appear consistent. It is more common to have masking applied to data that is represented outside of a corporate production system. In other words, where data is needed for the purpose of application development, building program extensions and conducting various test cycles. It is common practice in enterprise computing to take data from the production systems to fill the data component, required for these non-production environments. However, this practice is not always restricted to non-production environments. In some organizations, data that appears on terminal screens to call centre operators may have masking dynamically applied based on user security permissions (e.g. preventing call centre operators from viewing Credit Card Numbers in billing systems).

The primary concern from a corporate governance perspective[4] is that personnel conducting work in these non-production environments are not always security cleared to operate with the information contained in the production data. This practice represents a security hole where data can be copied by unauthorized personnel, and security measures associated with standard production level controls can be easily bypassed. This represents an access point for a data security breach.

The overall practice of data masking at an organizational level should be tightly coupled with the Test Management Practice[5][6] and underlying Methodology and should incorporate processes for the distribution of masked test data subsets

Background

Data involved in any data masking or obfuscation must remain meaningful at several levels:

The data must remain meaningful for the application logic. For example, if elements of addresses are to be obfuscated and city and suburbs are replaced with substitute cities or suburbs, then, if within the application there is a feature that validates postcode or post code lookup, that function must still be allowed to operate without error and operate as expected. The same is also true for credit-card algorithm validation checks and Social Security Number validations.

The data must undergo enough changes so that it is not obvious that the masked data is from a source of production data. For example, it may be common knowledge in an organisation that there are 10 senior managers all earning in excess of $300K. If a test environment of the organisation's HR System also includes 10 identities in the same earning-bracket, then other information could be pieced together to reverse-engineer a real-life identity. Theoretically, if the data is obviously masked or obfuscated, then it would be reasonable for someone intending a data breach to assume that they could reverse engineer identity-data if they had some degree of knowledge of the identities in the production data-set. Accordingly, data obfuscation or masking of a data-set applies in such a manner as to ensure that identity and sensitive data records are protected - not just the individual data elements in discrete fields and tables.

The masked values may be required to be consistent across multiple databases within an organization when the databases each contain the specific data element being masked. Applications may initially access one database and later access another one to retrieve related information where the foreign key has been masked (e.g. a call center application first brings up data from a customer master database and, depending on the situation, subsequently accesses one of several other databases with very different financial products.) This requires that the masking applied is repeatable (the same input value to the masking algorithm always yields the same output value) but not able to be reverse engineered to get back to the original value. Additional constraints as mentioned in (1) above may also apply depending on the data element(s) involved. Where different character sets are used across the databases that need to connect in this scenario, a scheme of converting the original values to a common representation will need to be applied, either by the masking algorithm itself or prior to invoking said algorithm.

Techniques

Substitution

Substitution is one of the most effective methods of applying data masking and being able to preserve the authentic look and feel of the data records.

It allows the masking to be performed in such a manner that another authentic-looking value can be substituted for the existing value. There are several data field types where this approach provides optimal benefit in disguising the overall data subset as to whether or not it is a masked data set. For example, if dealing with source data which contains customer records, real life surname or first name can be randomly substituted from a supplied or customised look up file. If the first pass of the substitution allows for applying a male first name to all first names, then the second pass would need to allow for applying a female first name to all first names where gender equals "F." Using this approach we could easily maintain the gender mix within the data structure, apply anonymity to the data records but also maintain a realistic looking database, which could not easily be identified as a database consisting of masked data.

This substitution method needs to be applied for many of the fields that are in DB structures across the world, such as telephone numbers, zip codes and postcodes, as well as credit card numbers and other card type numbers like Social Security numbers and Medicare numbers where these numbers actually need to conform to a checksum test of the Luhn algorithm.

In most cases, the substitution files will need to be fairly extensive so having large substitution datasets as well the ability to apply customized data substitution sets should be a key element of the evaluation criteria for any data masking solution.

Shuffling

The shuffling method is a very common form of data obfuscation. It is similar to the substitution method but it derives the substitution set from the same column of data that is being masked. In very simple terms, the data is randomly shuffled within the column. However, if used in isolation, anyone with any knowledge of the original data can then apply a "What If" scenario to the data set and then piece back together a real identity. The shuffling method is also open to being reversed if the shuffling algorithm can be deciphered.

Shuffling, however, has some real strengths in certain areas. If for instance, the end of year figures for financial information in a test data base, one can mask the names of the suppliers and then shuffle the value of the accounts throughout the masked database. It is highly unlikely that anyone, even someone with intimate knowledge of the original data could derive a true data record back to its original values.

Number and date variance

The numeric variance method is very useful for applying to financial and date driven information fields. Effectively, a method utilising this manner of masking can still leave a meaningful range in a financial data set such as payroll. If the variance

applied is around +/- 10% then it is still a very meaningful data set in terms of the ranges of salaries that are paid to the recipients.

The same also applies to the date information. If the overall data set needs to retain demographic and actuarial data integrity, then applying a random numeric variance of +/- 120 days to date fields would preserve the date distribution, but it would still prevent traceability back to a known entity based on their known actual date or birth or a known date value for whatever record is being masked.

Encryption

Encryption is often the most complex approach to solving the data masking problem. The encryption algorithm often requires that a "key" be applied to view the data based on user rights. This often sounds like the best solution, but in practice the key may then be given out to personnel without the proper rights to view the data. This then defeats the purpose of the masking exercise. Old databases may then get copied with the original credentials of the supplied key and the same uncontrolled problem lives on.

Recently, the problem of encrypting data while preserving the properties of the entities got recognition and a newly acquired interest among the vendors and academia. New challenge gave birth to algorithms called FPE (format preserving encryption). They are based on the accepted AES algorithmic mode that makes them being recognized by NIST. [10]

Nulling out or deletion

Sometimes a very simplistic approach to masking is adopted through applying a null value to a particular field. The null value approach is really only useful to prevent visibility of the data element.

In almost all cases, it lessens the degree of data integrity that is maintained in the masked data set. It is not a realistic value and will then fail any application logic validation that may have been applied in the front end software that is in the system under test. It also highlights to anyone that wishes to reverse engineer any of the identity data that data masking has been applied to some degree on the data set.

Masking out

Character scrambling or masking out of certain fields is also another simplistic yet very effective method of preventing sensitive information to be viewed. It is really an extension of the previous method of nulling out, but there is a greater emphasis on keeping the data real and not fully masked all together.

This is commonly applied to credit card data in production systems. For instance, an operator at a call centre might bill an item to a customer's credit card. They then quote a billing reference to the card with the last 4 digits of XXXX XXXXxxxx 6789. As an operator they can only see the last 4 digits of the card number, but once the

billing system passes the customer's details for charging, the full number is revealed to the payment gateway systems.

This system is not very effective for test systems, but it is very useful for the billing scenario detailed above. It is also commonly known as a dynamic data masking method.[11][12]

Additional complex rules

Additional rules can also be factored into any masking solution regardless of how the masking methods are constructed. Product agnostic White Papers[13] are a good source of information for exploring some of the more common complex requirements for enterprise masking solutions, which include Row Internal Synchronisation Rules, Table Internal Synchronisation Rules and Table[14] to Table Synchronisation Rules.

Different types[edit]

Data masking is tightly coupled with building test data. Two major types of data masking are static and on-the-fly data masking. [15]

Static data masking

Static Data Masking is usually performed on the golden copy of the database, but can also be applied to values in other sources, including files. In DB environments, production DBAs will typically load table backups to a separate environment, reduce the dataset to a subset that holds the data necessary for a particular round of testing (a technique called "subsetting"), apply data masking rules while data is in stasis, apply necessary code changes from source control, and/or and push data to desired environment.[16]

Deterministic data masking

Deterministic Masking is the process of replacing a value in a column with the same value whether in the same row, the same table, the same database/schema and between instances/servers/database types. Example: A database has multiple tables, each with a column that has first names. With deterministic masking the first name will always be replaced with the same value – “Lynne” will always become “Denise” – wherever “Lynne” may be in the database.[17]

Statistical data obfuscation

There are also alternatives to the static data masking that rely on stochastic perturbations of the data that preserve some of the statistical properties of the original data. Examples of statistical data obfuscation methods include differential privacy [18] and the DataSifter method .[19]

On-the-fly data masking

On-the-Fly Data Masking[20] happens in the process of transferring data from environment to environment without data touching the disk on its way. The same technique is applied to "Dynamic Data Masking" but one record at a time. This type of data masking is most useful for environments that do continuous deployments as well as for heavily integrated applications. Organizations that employ continuous deployment or continuous delivery practices do not have the time necessary to create a backup and load it to the golden copy of the database. Thus, continuously sending smaller subsets (deltas) of masked testing data from production is important. In heavily integrated applications, developers get feeds from other production systems at the very onset of development and masking of these feeds is either overlooked and not budgeted until later, making organizations non-compliant. Having on-the-fly data masking in place becomes essential.

Dynamic data masking

Dynamic Data Masking is similar to On-the-Fly Data Masking but it differs in the sense that On-the-Fly Data Masking is about copying data from one source to another source so that the latter can be shared. Dynamic data masking happens at runtime, dynamically, and on-demand so that there doesn't need to be a second data source where to store the masked data dynamically.

Dynamic data masking enables several scenarios, many of which revolve around strict privacy regulations e.g. the Singapore Monetary Authority or the Privacy regulations in Europe.

Dynamic data masking is attribute-based and policy-driven. Policies include:

Doctors can view the medical records of patients they are assigned to (data filtering)

Doctors cannot view the SSN field inside a medical record (data masking).

Dynamic data masking can also be used to encrypt or decrypt values on the fly especially when using format-preserving encryption.

Several standards have emerged in recent years to implement dynamic data filtering and masking. For instance, XACML policies can be used to mask data inside databases.

There are five possible technologies to apply Dynamic data masking:

In the Database: Database receives the SQL and applies rewrite to returned masked result set. Applicable for developers & DBAs but not for applications (because connection pools, application caching and data-bus hide the application user identity from the database and can also cause application data corruption).

Network Proxy between the application and the database: Captures the SQL and applies rewrite on the select request. Applicable for developers & DBAs with simple 'select'requests but not for stored procedures (which the proxy only identifies the exec.) and applications (because connection pools, application caching and data-bus

hide the application user identity from the database and can also cause application data corruption).

Network Proxy between the end-user and the application: identifying text strings and replacing them. This method is not applicable for complex applications as it will easily cause corruption when the real-time string replacement is unintentionally applied.

Code changes in the applications & XACML: code changes are usually hard to perform, impossible to maintain and not applicable for packaged applications. Some applications like Oracle E- Business Suite, PeopleSoft and JD Edwards allow adding an API code to their application code to enable dynamic data masking.[21]

Within the application run-time: By instrumenting the application run-time, policies are defined to rewrite the result set returned from the data sources, while having full visibility to the application user. This method is the only applicable way to dynamically mask complex applications as it enables control to the data request, data result and user result.

Supported by a browser plugin: In the case of SaaS or local web applications, browser add-ons can be configured to mask data fields corresponding to precise CSS Selectors. This can either be accomplished by marking sensitive fields in the application, for example by a HTML class or by finding the right selectors that identify the fields to be obfuscated or masked.

Data masking and the cloud

In latest years, organizations develop their new applications in the cloud more and more often, regardless of whether final applications will be hosted in the cloud or on-premises. The cloud solutions as of now allow organizations to use Infrastructure as a Service or IaaS, Platform as a Service or PaaS, and Software as a Service or SaaS. There are various modes of creating test data and moving it from on-premises databases to the cloud, or between different environments within the cloud. Dynamic Data Masking becomes even more critical in cloud when customers need to protecting PII data while relying on cloud providers to administer their databases. [22] Data masking invariably becomes the part of these processes in SDLC as the development environments' SLAs are usually not as stringent as the production environments' SLAs regardless of whether application is hosted in the cloud or on-premises.

Unit-III

SIMD Interconnection network:

Dynamic network analysis

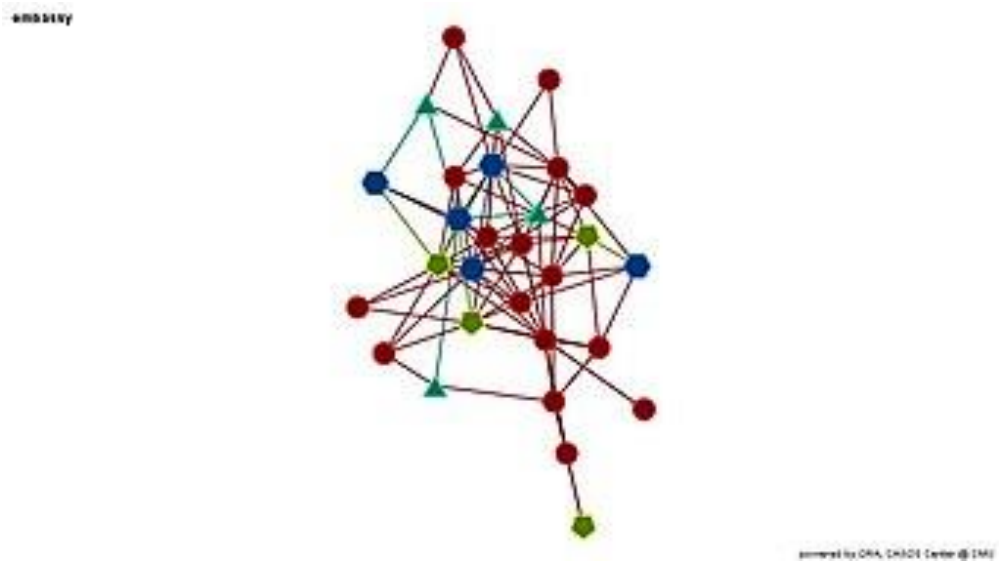
(DNA) is an emergent scientific field that brings together traditional social network analysis (SNA), link analysis (LA), social simulation and multi-agent systems (MAS) within network science and network theory.

There are two aspects of this field. The first is the statistical analysis of DNA data. The second is the utilization of simulation to address issues of network dynamics. DNA networks vary from traditional social networks in that they are larger, dynamic, multi-mode, multi-plex networks, and may contain varying levels of uncertainty. The main difference of DNA to SNA is that DNA takes interactions of social features conditioning structure and behavior of networks into account. DNA is tied to temporal analysis but temporal analysis is not necessarily tied to DNA, as changes in networks sometimes result from external factors which are independent of social features found in networks. One of the most notable and earliest of cases in the use of DNA is in Sampson's monastery study, where he took snapshots of the same network from different intervals and observed and analyzed the evolution of the network.[1] An early study of the dynamics of link utilization in very large-scale complex networks provides evidence of dynamic centrality, dynamic motifs, and cycles of social interactions.[2][3]

DNA statistical tools are generally optimized for large-scale networks and admit the analysis of multiple networks simultaneously in which, there are multiple types of nodes (multi-node) and multiple types of links (multi-plex). Multi-node multi-plex networks are generally referred to as meta-networks or high-dimensional networks. In contrast, SNA statistical tools focus on single or at most two mode data and facilitate the analysis of only one type of link at a time.

DNA statistical tools tend to provide more measures to the user, because they have measures that use data drawn from multiple networks simultaneously. Latent space models (Sarkar and Moore, 2005)[4] and agent-based simulation are often used to examine dynamic social networks (Carley et al., 2009).[5] From a computer simulation perspective, nodes in DNA are like atoms in quantum theory, nodes can be, though need not be, treated as probabilistic. Whereas nodes in a traditional SNA model are static, nodes in a DNA model have the ability to learn. Properties change over time; nodes can adapt: A company's employees can learn new skills and increase their value to the network; or, capture one terrorist and three more are forced to improvise. Change propagates from one node to the next and so on. DNA

adds the element of a network's evolution and considers the circumstances under which change is likely to occur.



An example of a multi-entity, multi-network, dynamic network diagram

There are three main features to dynamic network analysis that distinguish it from standard social network analysis. First, rather than just using social networks, DNA looks at meta-networks. Second, agent-based modeling and other forms of simulations are often used to explore how networks evolve and adapt as well as the impact of interventions on those networks. Third, the links in the network are not binary; in fact, in many cases they represent the probability that there is a link.

Meta-network

A meta-network is a multi-mode, multi-link, multi-level network. Multi-mode means that there are many types of nodes; e.g., nodes people and locations. Multi-link means that there are many types of links; e.g., friendship and advice. Multi-level means that some nodes may be members of other nodes, such as a network composed of people and organizations and one of the links is who is a member of which organization.

While different researchers use different modes, common modes reflect who, what, when, where, why and how. A simple example of a meta-network is the PCANS formulation with people, tasks, and resources.[6] A more detailed formulation considers people, tasks, resources, knowledge, and organizations.[7] The ORA tool was developed to support meta-network analysis.[8]

Illustrative problems that people in the DNA area work on

Developing metrics and statistics to assess and identify change within and across networks.

Developing and validating simulations to study network change, evolution, adaptation, decay. See Computer simulation and organizational studies

Developing and testing theory of network change, evolution, adaptation, decay[9]

Developing and validating formal models of network generation and evolution

Developing techniques to visualize network change overall or at the node or group level

Developing statistical techniques to see whether differences observed over time in networks are due to simply different samples from a distribution of links and nodes or changes over time in the underlying distribution of links and nodes

Developing control processes for networks over time

Developing algorithms to change distributions of links in networks over time

Developing algorithms to track groups in networks over time

Developing tools to extract or locate networks from various data sources such as texts

Developing statistically valid measurements on networks over time

Examining the robustness of network metrics under various types of missing data

Empirical studies of multi-mode multi-link multi-time period networks

Examining networks as probabilistic time-variant phenomena

Forecasting change in existing networks

Identifying trails through time given a sequence of networks

Identifying changes in node criticality given a sequence of networks anything else related to multi-mode multi-link multi-time period networks

Studying random walks on temporal networks[10]

Quantifying structural properties of contact sequences in dynamic networks, which influence dynamical processes[11]

Assessment of covert activity[12] and dark networks[13]

Citationalanalysis[14]

Social media analysis[15]

Assessment of public health systems[16]

Analysis of hospital safety outcomes[17]

Assessment of the structure of ethnic violence from news data[18]

Assessment of terror groups[19]

Online social decay of social interactions[20]

Visualization of large financial networks over time[21]

Modelling of classroom interactions in schools[

Cube interconnection network,

The minimum distance between a pair of nodes is the minimum number of communication links (hops) that data from one of the nodes must traverse in order to reach the other node. 12. Cube interconnection network: It is a 3 dimensional interconnection network. ... At node 1, t0 is set to 0; thus T now becomes equal to 100.7 April 2015

Shuffle exchange and Omega Network

In order to provide full connectivity the Benes network requires 2log2(N) + 1 stages, each with N/2 switch-nodes. However, it is possible to reduce the cost of a multi-stage network still further by using a class of networks, which are not full connection networks, known as shuffle-exchange networks. In general, shuffle-exchange networks consist of a sequence of log2(N) exchange permutations interspersed with shuffle or butterfly permutations.

On first inspection the following discussion on shuffle-exchange permutations may appear to be simply a notational convenience, but it is important to understand how a sequence of shuffle and exchange permutations can together form a useful network. The key to understanding multi-stage permutation networks is to consider the effect each successive permutation has on the label of an object in passage through the network. Assume that S is the label of an object entering the network, and D is the label of the destination of that object. We associate a temporary label L with the object, and this is initially set to S. If we can modify L by a sequence of permutations so that it becomes equal to D then the object will arrive at its destination.

Since the E1 permutation provides us with the choice of inverting the least significant bit of the input label or leaving it intact, it is possible to use the E1 permutation to make the least significant bit in L equal to the least significant bit in D. This is the basic step in converting from L to D, and the choice of ε1 or I permutation determines the switch-node setting in the general exchange box of Figure 1. The next step is to expose the next bit in L to the E1 permutation, and this is done most simply by shifting L by one bit. This is directly equivalent to a perfect-shuffle permutation on all labels L in the range 0 to N, as shown for N=8 in Figure 2. After n=log2N applications of the shuffle and exchange permutations all bits in L will have been changed, and L will be equal to D. As a direct consequence of this, the object located at label L will have been routed to the output port identified by D, and the network will have performed its function.

A number of important multi-stage shuffle-exchange networks have been devised. The banyan, omega and indirect binary n-cube networks are discussed in the next section, Shuffle Exchange Network Examples

Figure 1. Generalised exchange switch mappings

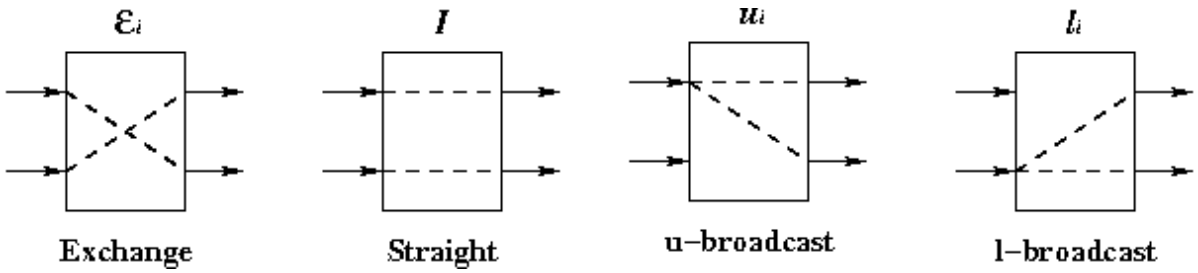
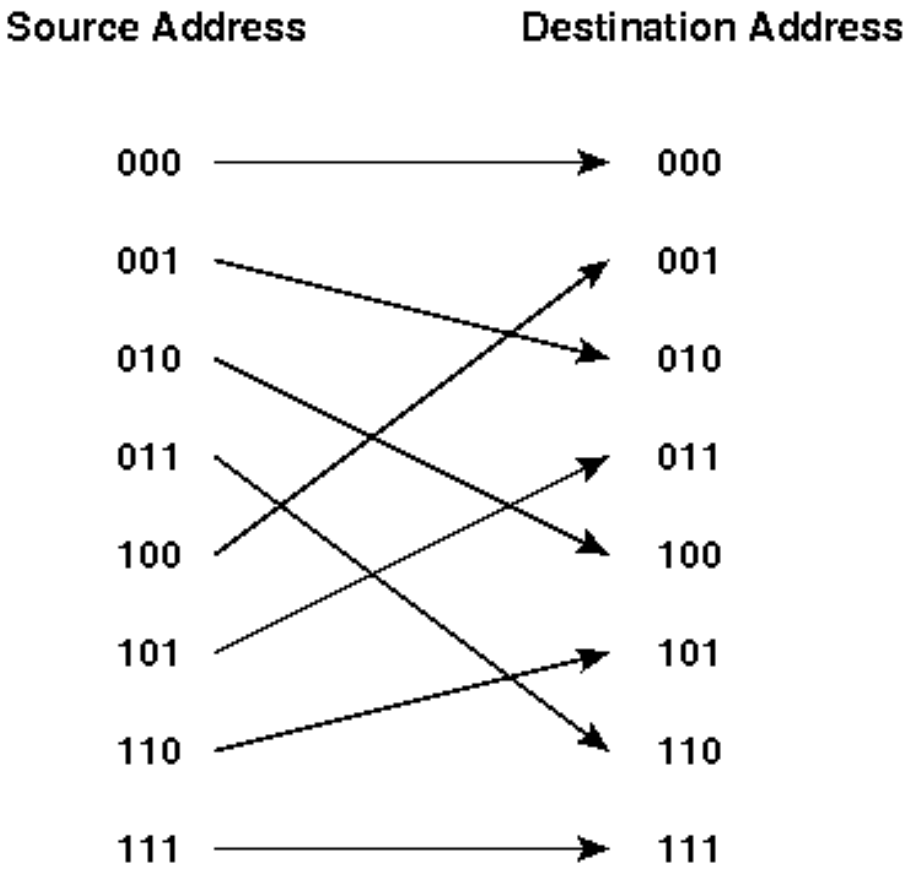


Figure 2. The shuffle permutation for N = 8



SIMD matrix multiplication

I am currently reading an article on github about performance optimisation using Clang's extended vector syntax. The author gives the following code snippet:

The templated code below implements the innermost loops that calculate a patch of size regA x regB in matrix C. The code loads regA scalars from matrixA and regB SIMD-width vectors from matrix B. The program uses Clang's extended vector syntax.

```
/// Compute a RAxRB block of C using a vectorized dot product, where RA is the
/// number of registers to load from matrix A, and RB is the number of registers
/// to load from matrix B.
template<unsigned regsA, unsigned regsB>
voidmatmul_dot_inner(int k, const float *a, intlda, const float *b, intldb,
float *c, intldc) {
float8csum[regsA][regsB] = {{0.0}};
for (int p = 0; p < k; p++) {

// Perform the DOT product.
for (int bi = 0; bi <regsB; bi++) {
float8 bb = LoadFloat8(&B(p, bi * 8));
for (intai = 0; ai<regsA; ai++) {
float8aa = BroadcastFloat8(A(ai, p));
csum[ai][bi] += aa * bb;
}
}
}

// Accumulate the results into C.
for (intai = 0; ai<regsA; ai++) {
for (int bi = 0; bi <regsB; bi++) {
AdduFloat8(&C(ai, bi * 8), csum[ai][bi]);
}
}
}
```

The code, outlines below, confuses me the most. I read the full article and understood the logic behind using blocking and calculating a small patch, but I can't entirely understand what does this bit means:

```
// Perform the DOT product.
```

```
for (int bi = 0; bi <regsB; bi++) {
float8 bb = LoadFloat8(&B(p, bi * 8)); //the pointer to the range of values?
for (intai = 0; ai<regsA; ai++) {
float8aa = BroadcastFloat8(A(ai, p));
csum[ai][bi] += aa * bb;
}
}
}
```

Multiprocessor Architecture:

A multiprocessor system is defined as "a system with more than one processor", and, more precisely, "a number of central processing units linked together to enable parallel processing to take place".[1][2][3]

The key objective of a multiprocessor is to boost a system's execution speed. The other objectives are fault tolerance and application matching.[4]

The term "multiprocessor" can be confused with the term "multiprocessing". While multiprocessing is a type of processing in which two or more processors work together to execute multiple programs simultaneously, multiprocessor refers to a hardware architecture that allows multiprocessing.[5]

Multiprocessor systems are classified according to how processor memory access is handled and whether system processors are of a single type or various ones.

Multiprocessor system types

There are many types of multiprocessor systems:

Loosely coupled multiprocessor system

Tightly coupled multiprocessor system

Homogeneous multiprocessor system

Heterogeneous multiprocessor system

Shared memory multiprocessor system

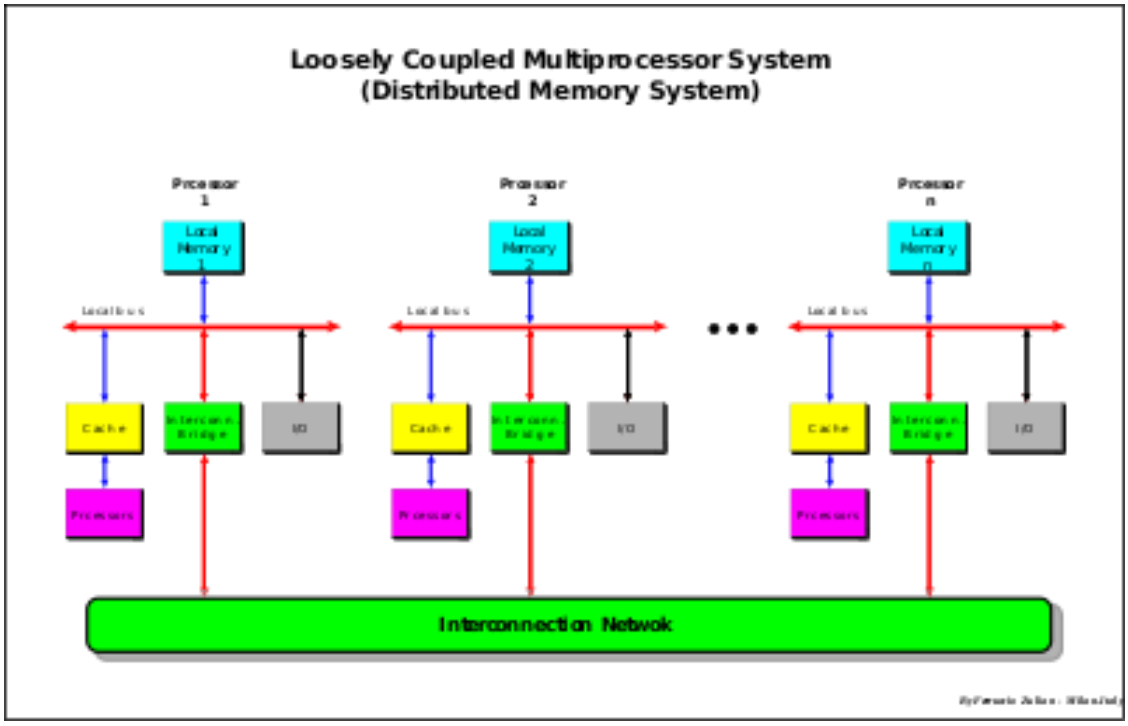
Distributed memory multiprocessor system

Uniform memory access (UMA) system

cc–NUMA system

Hybrid system – shared system memory for global data and local memory for local data

Loosely-coupled (distributed memory) multiprocessor system



Loosely Coupled Multiprocessor System

In loosely-coupled multiprocessor systems, each processor has its own local memory, input/output (I/O) channels, and operating system. Processors exchange data over a high-speed communication network by sending messages via a technique known as "message passing". Loosely-coupled multiprocessor systems are also known as distributed-memory systems, as the processors do not share physical memory and have individual I/O channels.

System characteristics

These systems are able to perform multiple-instructions-on-multiple-data (MIMD) programming.

This type of architecture allows parallel processing.

The distributed memory is highly scalable.

Tightly-coupled (shared memory) multiprocessor system

Multiprocessor system with a shared memory closely connected to the processors.

A symmetric multiprocessing system is a system with centralized shared memory called main memory (MM) operating under a single operating system with two or more homogeneous processors.

There are two types of systems:

Uniform memory-access (UMA) system

NUMA system

Uniform memory access (UMA) system

Heterogeneous multiprocessing system

Symmetric multiprocessing system (SMP)

Heterogeneous multiprocessor system

A heterogeneous multiprocessing system contains multiple, but not homogeneous, processing units – central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs), or any type of application-specific integrated circuits (ASICs). The system architecture allows any accelerator – for instance, a graphics processor – to operate at the same processing level as the system's CPU.

Symmetric multiprocessor system

Symmetric multiprocessing system

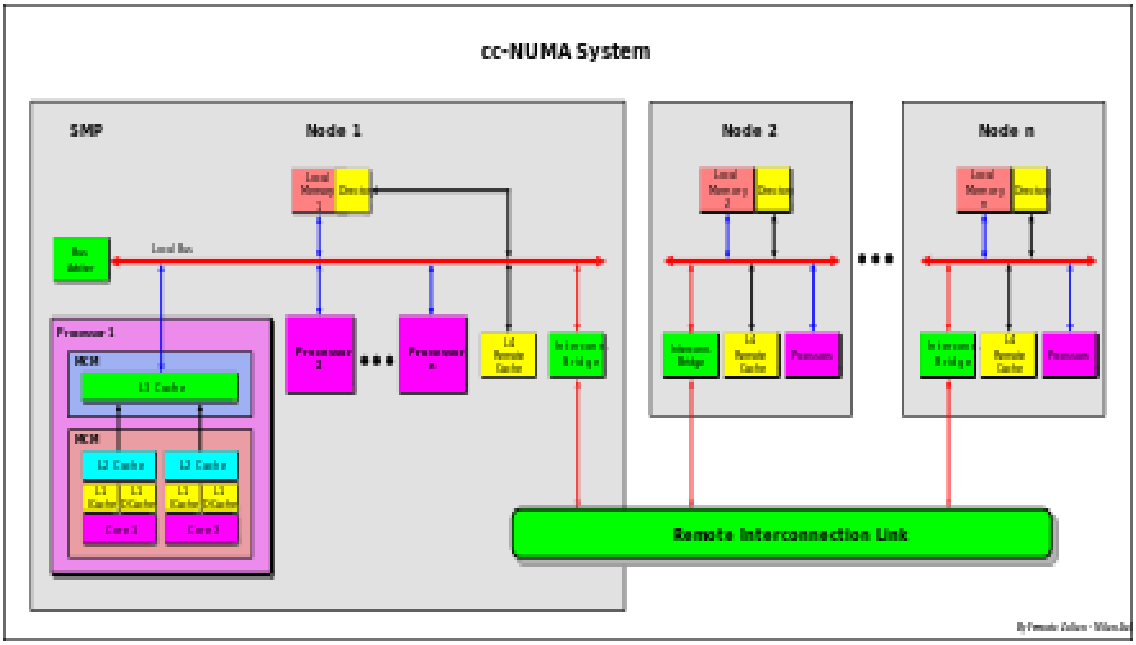
Systems operating under a single OS (operating system) with two or more homogeneous processors and with a centralized shared main memory.

A symmetric multiprocessor system (SMP) is a system with a pool of homogeneous processors running under a single OS with a centralized, shared main memory. Each processor, executing different programs and working on different sets of data, has the ability to share common resources (memory, I/O device, interrupt system, and so on) that are connected using a system bus, a crossbar, or a mix of the two, or an address bus and data crossbar.

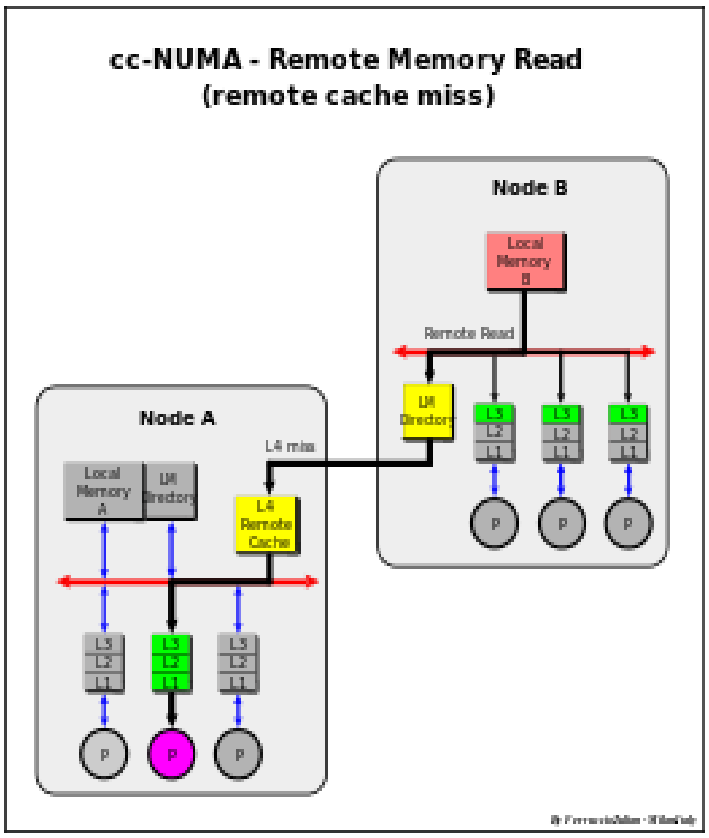
Each processor has its own cache memory that acts as a bridge between the processor and main memory. The function of the cache is to alleviate the need for main-memory data access, thus reducing system-bus traffic.

Use of shared memory allows for a uniform memory-access time (UMA).

cc-NUMA system



cc-NUMA System



cc-NUMA Remote Memory Read

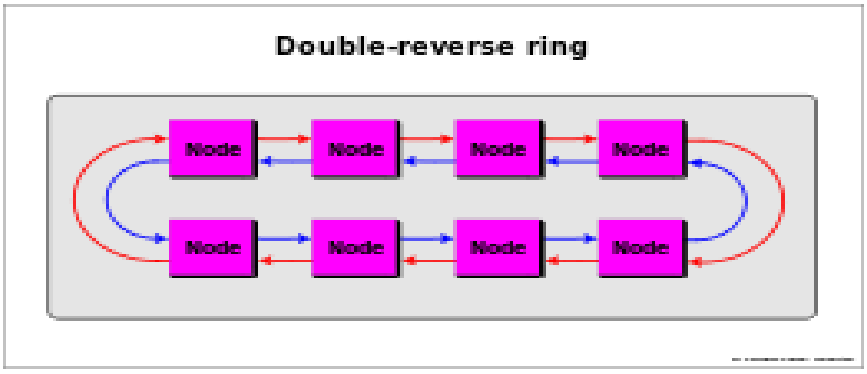
It is known that the SMP system has limited scalability. To overcome this limitation, the architecture called "cc-NUMA" (cache coherency–non-uniform memory access) is normally used. The main characteristic of a cc-NUMA system is having shared global memory that is distributed to each node, although the effective "access" a processor has to the memory of a remote component subsystem, or "node", is slower compared to local memory access, which is why the memory access is "non-uniform".

A cc–NUMA system is a cluster of SMP systems – each called a "node", which can have a single processor, a multi-core processor, or a mix of the two, of one or other kinds of architecture – connected via a high-speed "connection network" that can be a "link" that can be a single or double-reverse ring, or multi-ring, point-to-point connections,[6][7] or a mix of these (e.g. IBM Power Systems[6][8]), bus interconnection (e.g. NUMAq[9]), "crossbar", "segmented bus" (NUMA Bull HN ISI ex Honeywell,[10]) "mesh router", etc.

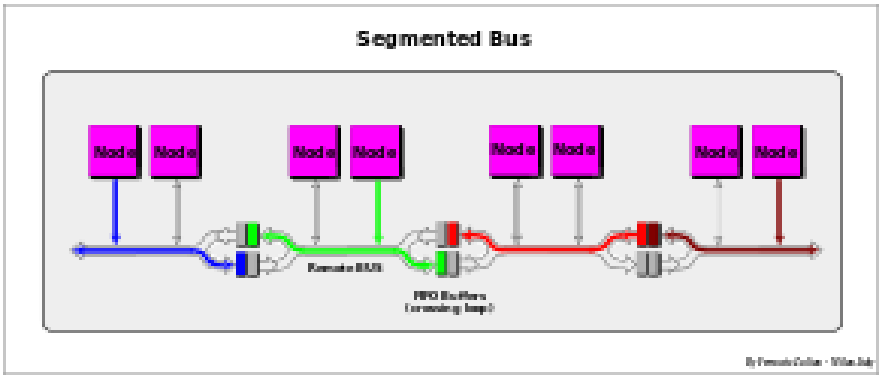
cc-NUMA is also called "distributed shared memory" (DSM) architecture.[11]

The difference in access times between local and remote memory can be also an order of magnitude, depending on the kind of connection network used (faster in segmented bus, crossbar, and point-to-point interconnection; slower in serial rings connection).

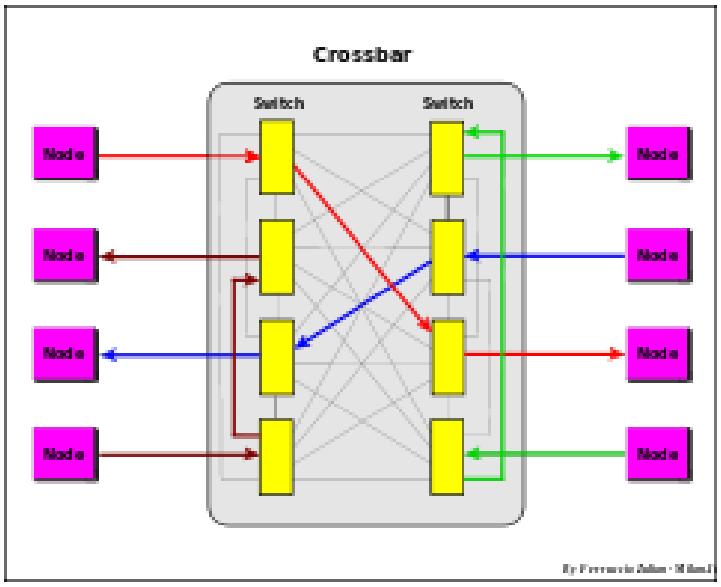
Examples of interconnection



Double-reverse ring



Segmented Bus



Crossbar

To overcome this limit, a large remote cache (see Remote cache) is normally used. With this solution, the cc-NUMA system becomes very close to a large SMP system.

Tightly-coupled versus loosely-coupled architecture

Both architectures have trade-offs which may be summarized as follows:

Loosely-coupled architectures feature high performances of each individual processor but do not enable for easy real-time balancing of the load among processors.

Tightly-coupled architectures feature easy load-balancing and distribution among processors but suffer from the bottleneck consisting in the sharing of common resources through one or more buses.

Multiprocessor system featuring global data multiplication

An intermediate approach, between those of the two previous architectures, is having common resources and local resources, such as local memories (LM), in each processor.

The common resources are accessible from all processors via the system bus, while local resources are only accessible to the local processor. Cache memories can be viewed in this perspective as local memories.

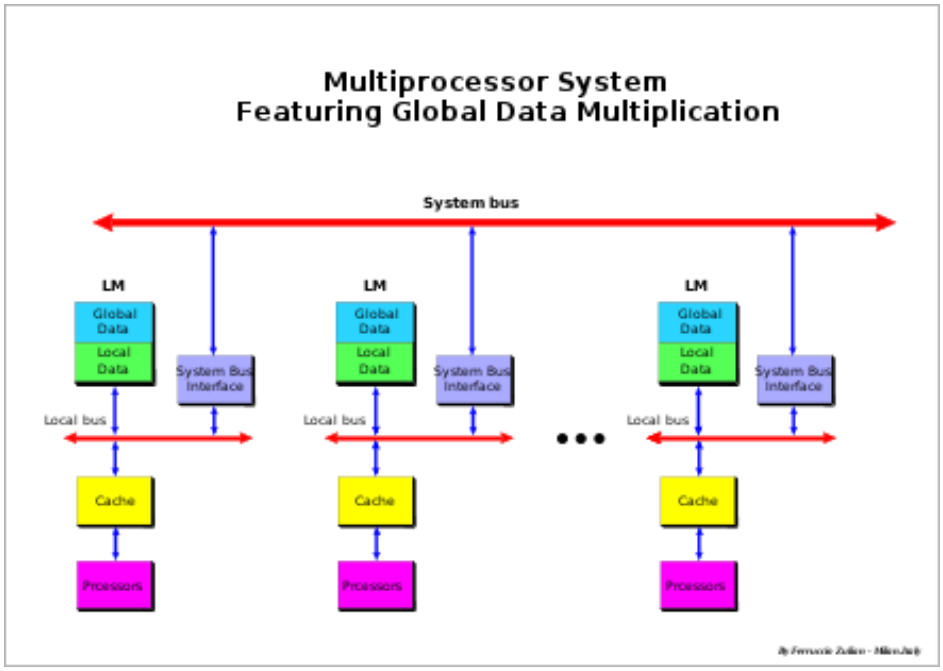
This system (patented by F. Zulian [12]), used on the DPX/2 300 Unix based system (Bull Hn Information Systems Italia (ex Honeywell)),[13][14] is a mix of tightly and loosely coupled systems and makes use of all the advancements of these two architectures.

The local memory is divided into two sectors, global data (GD) and local data (LD).

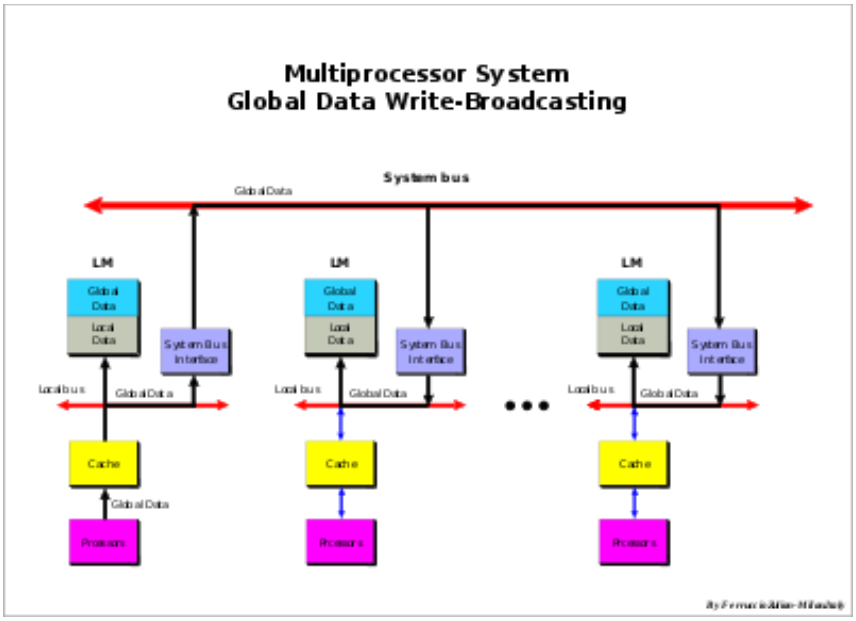
The basic concept of this architecture is to have global data, which is modifiable information, accessible by all processors. This information is duplicated and stored in each local memory of each processor.

Each time the global data is modified in a local memory, a hardware write-broadcasting is sent to the system bus to all other local memories to maintain the global data coherency. Thus, global data may be read by each processor accessing its own local memory without involving the system bus. System bus access is only required when global data is modified in a local memory to update the copy of this data stored in the other local memories.

Local data can be exchanged in a loosely coupled system via message-passing



Multiprocessor System Featuring Global Data Multiplication

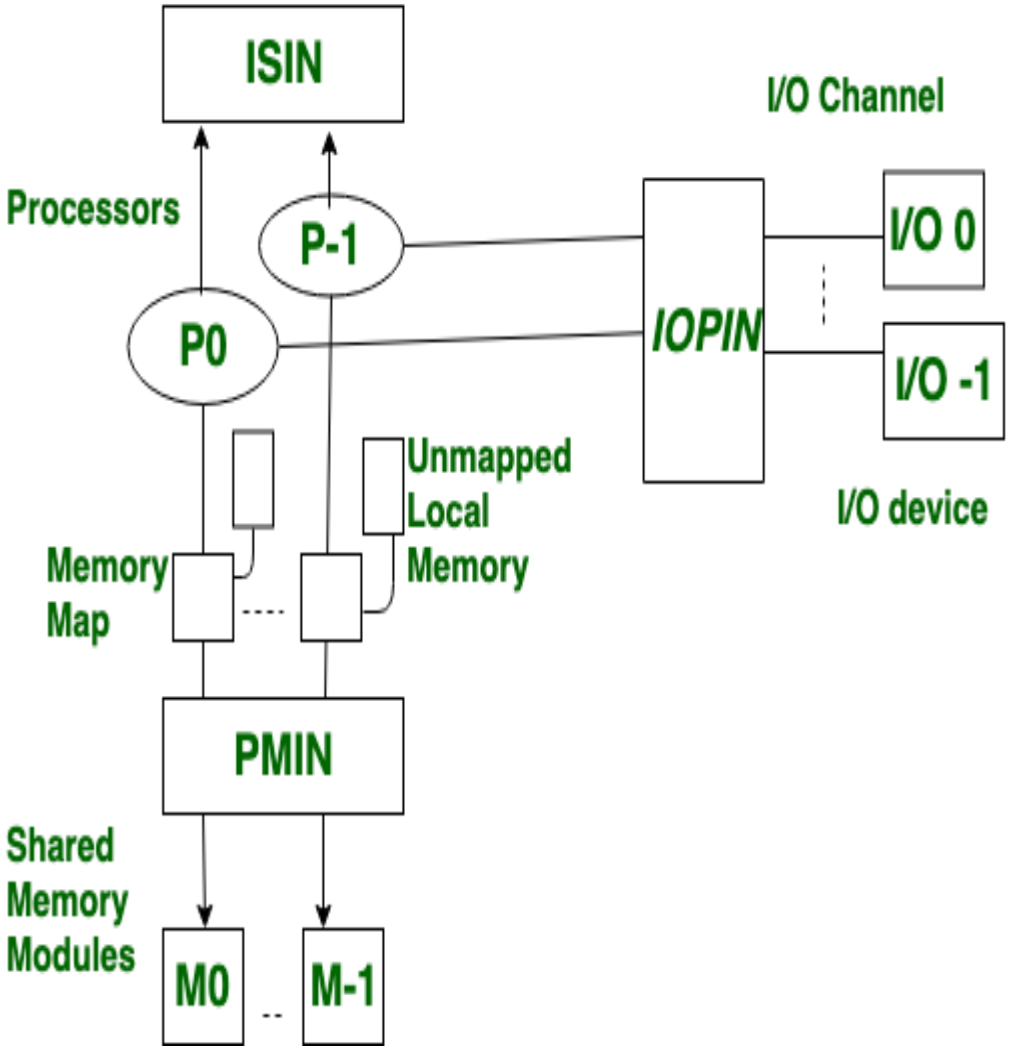


Multiprocessor System Featuring Global Data Multiplation - Global Data Write-Broadcasting

Tightly and loosely coupled multiprocessors.

Loosely Coupled Multiprocessor System:
It is a type of multiprocessing system in which, There is distributed memory instead of shared memory. In loosely coupled multiprocessor system, data rate is low rather than tightly coupled multiprocessor system. In loosely coupled multiprocessor system, modules are connected through MTS (Message transfer system) network.

Tightly Coupled Multiprocessor System:
It is a type of multiprocessing system in which, There is shared memory. In tightly coupled multiprocessor system, data rate is high rather than loosely coupled multiprocessor system. In tightly coupled multiprocessor system, modules are connected through PMIN, IOPIN and ISIN networks.



Let’s study the difference between loosely coupled and tightly coupled multiprocessor system:

S.NO	LOOSELY COUPLED	TIGHTLY COUPLED
1.	There is distributed memory in loosely coupled multiprocessor system.	There is shared memory, in tightly coupled multiprocessor system.

2.	Loosely Coupled Multiprocessor System has low data rate.	Tightly coupled multiprocessor system has high data rate.
3.	The cost of loosely coupled multiprocessor system is less.	Tightly coupled multiprocessor system is more costly.
4.	In loosely coupled multiprocessor system, modules are connected through Message transfer system network.	While there is PMIN, IOPIN and ISIN networks.
5.	In loosely coupled multiprocessor, Memory conflicts don’t take place.	While tightly coupled multiprocessor system have memory conflicts.
6.	Loosely Coupled Multiprocessor system has low degree of interaction between tasks.	Tightly Coupled multiprocessor system has high degree of interaction between tasks.
7.	In loosely coupled multiprocessor, there is direct connection between processor and I/O devices.	While in tightly coupled multiprocessor, IOPIN helps connection between processor and I/O devices.
8.	Applications of loosely coupled multiprocessor are in distributed computing systems.	Applications of tightly coupled multiprocessor are in parallel processing systems.

Unit -IV

Multiprocessor scheduling strategies and deterministic scheduling models

Data are characteristics or information, usually numerical, that are collected through observation.[1] In a more technical sense, data are a set of values of qualitative or quantitative variables about one or more persons or objects, while a datum (singular of data) is a single value of a single variable.[2]

Although the terms "data" and "information" are often used interchangeably, these terms have distinct meanings. In some popular publications, data are sometimes said to be transformed into information when they are viewed in context or in post-analysis.[3] In academic treatments of the subject, however, data are simply units of information. Data is employed in scientific research, businesses management (e.g., sales data, revenue, profits, stock price), finance, governance (e.g., crime rates, unemployment rates, literacy rates), and in virtually every other form of human organizational activity (e.g., censuses of the number of homeless people by non-profit organizations).

Data are measured, collected and reported, and analyzed, whereupon it can be visualized using graphs, images or other analysis tools. Data as a general concept refers to the fact that some existing information or knowledge is represented or coded in some form suitable for better usage or processing. Raw data ("unprocessed data") is a collection of numbers or characters before it has been "cleaned" and corrected by researchers. Raw data needs to be corrected to remove outliers or obvious instrument or data entry errors (e.g., a thermometer reading from an outdoor Arctic location recording a tropical temperature). Data processing commonly occurs by stages, and the "processed data" from one stage may be considered the "raw data" of the next stage. Field data is raw data that is collected in an uncontrolled "in situ" environment. Experimental data is data that is generated within the context of a scientific investigation by observation and recording.

Data has been described as the new oil of the digital economy.[4][5]

Data documents

Whenever data needs to be registered, data exists in the form of a data documents. Kinds of data documents include:

data repository

data study

data set

software

data paper

database

data handbook

data journal

Some of these data documents (data repositories, data studies, data sets and software) are indexed in Data Citation Indexes, while data papers are indexed in traditional bibliographic databases, e.g., Science Citation Index. See further.[14]

Data collection

Gathering data can be accomplished through a primary source (the researcher is the first person to obtain the data) or a secondary source (the researcher obtains the data that has already been collected by other sources, such as data disseminated in a scientific journal). Data analysis methodologies vary and include data triangulation and data percolation.[15] The latter offers an articulate method of collecting, classifying and analyzing data using five possible angles of analysis (at least three) in order to maximize the research's objectivity and permit an understanding of the phenomena under investigation as complete as possible: qualitative and quantitative methods, literature reviews (including scholarly articles), interviews with experts, and computer simulation. The data are thereafter "percolated" using a series of pre-determined steps so as to extract the most relevant information.

In other fields

Although data are also increasingly used in other fields, it has been suggested that the highly interpretive nature of them might be at odds with the ethos of data as "given". Peter Checkland introduced the term capta (from the Latin capere, “to take”) to distinguish between an immense number of possible data and a sub-set of them, to which attention is oriented.[16] Johanna Drucker has argued that since the humanities affirm knowledge production as "situated, partial, and constitutive," using data may introduce assumptions that are counterproductive, for example that phenomena are discrete or are observer-independent.[17] The term capta, which emphasizes the act of observation as constitutive, is offered as an alternative to data for visual representations in the humanities.

Data flow graphs

A data flow graph is a model of a program with no conditionals. In a high-level programming language, a code segment with no conditionals—more precisely, with only one entry and exit point—is known as a basic block. Fig. 5.4 shows a simple

basic block. As the C code is executed, we would enter this basic block at the beginning and execute all the statements.

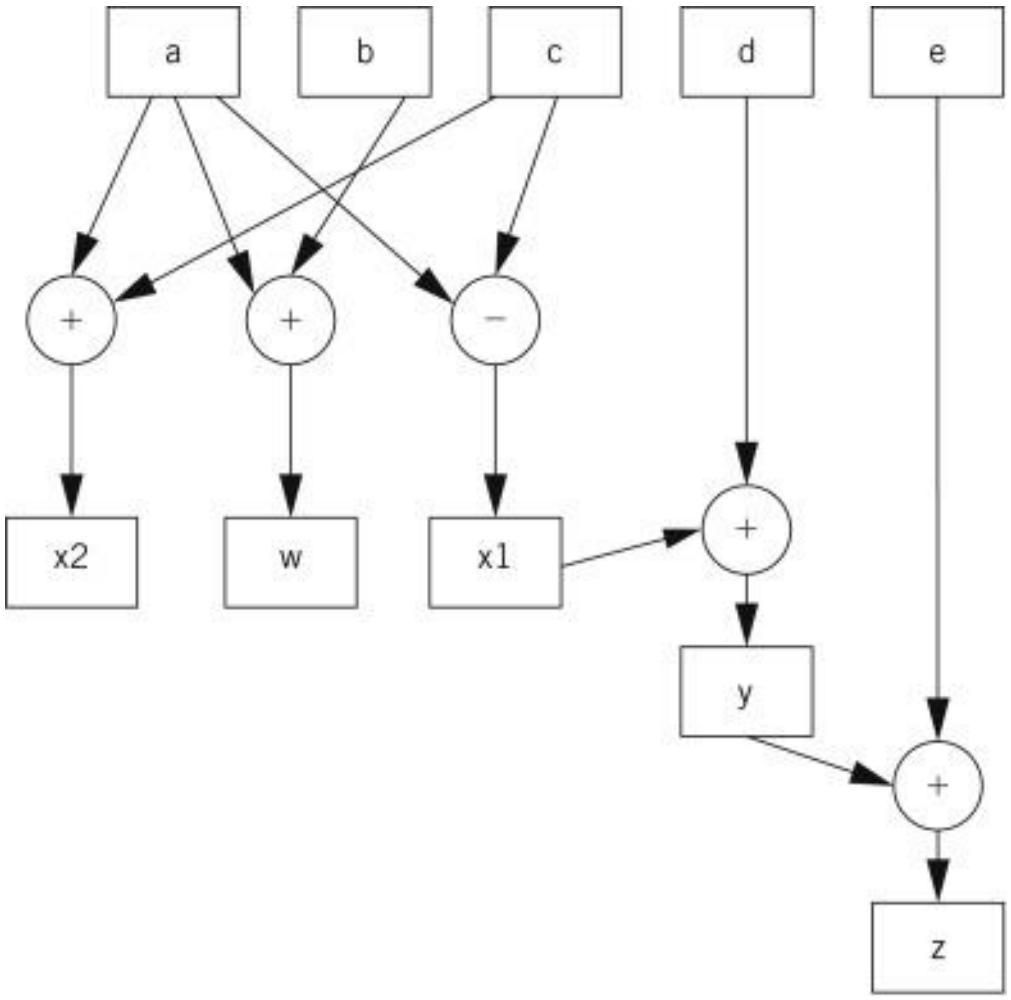
```
w = a + b;
x = a - c;
y = x + d;
x = a + c;
z = y + e;
```

Before we are able to draw the data flow graph for this code, we need to modify it slightly. There are two assignments to the variable x—it appears twice on the left side of an assignment. We need to rewrite the code in single-assignment form, in which a variable appears only once on the left side. Because our specification is C code, we assume that the statements are executed sequentially, so that any use of a variable refers to its latest assigned value. In this case, x is not reused in this block (presumably it is used elsewhere), so we just have to eliminate the multiple assignment to x. The result is shown in Fig. 5.5 where we have used the names x1 and x2 to distinguish the separate uses of x.

```
w = a+b;
x1 = a-c;
y = x1+d;
x2 = a+c;
z = y+e;
```

The single-assignment form is important because it allows us to identify a unique location in the code where each named location is computed. As an introduction to the data flow graph, we use two types of nodes in the graph—round nodes denote operators and square nodes represent values. The value nodes may be either inputs to the basic block, such as a and b, or variables assigned to within the block, such as w and x1. The data flow graph for our single-assignment code is shown in Fig. 5.6. The single-assignment form means that the data flow graph is acyclic—if we assigned to x multiple times, then the second assignment would form a cycle in the graph including x and the operators used to compute x. Keeping the data flow graph acyclic is important in many types of analyses we want to do on the graph. (Of

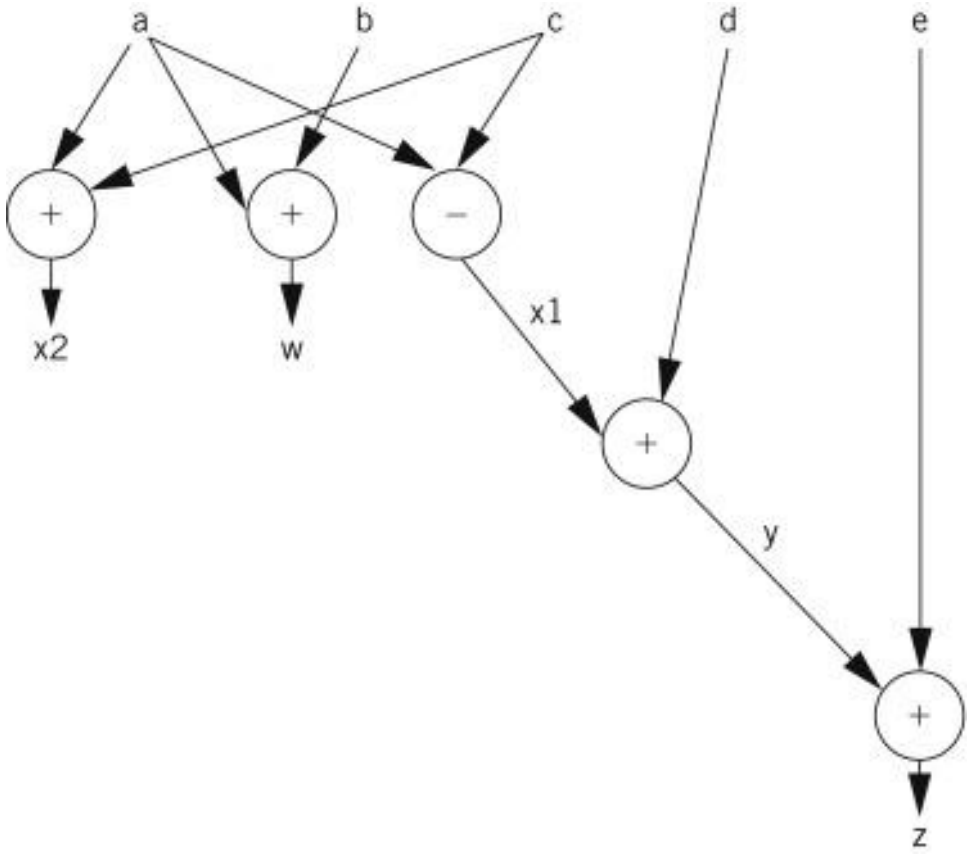
course, it is important to know whether the source code actually assigns to a variable multiple times, because some of those assignments may be mistakes. We consider the analysis of source code for proper use of assignments in Section 5.5.)



Sign in to download full-size image

Figure 5.6. An extended data flow graph for our sample basic block.

The data flow graph is generally drawn in the form shown in Fig. 5.7. Here, the variables are not explicitly represented by nodes. Instead, the edges are labeled with the variables they represent. As a result, a variable can be represented by more than one edge. However, the edges are directed and all the edges for a variable must come from a single source. We use this form for its simplicity and compactness.



Sign in to download full-size image

Figure 5.7. Standard data flow graph for our sample basic block.

The data flow graph for the code makes the order in which the operations are performed in the C code much less obvious. This is one of the advantages of the data flow graph. We can use it to determine feasible reorderings of the operations, which may help us to reduce pipeline or cache conflicts. We can also use it when the exact order of operations simply does not matter. The data flow graph defines a partial ordering of the operations in the basic block. We must ensure that a value is computed before it is used, but generally there are several possible orderings of evaluating expressions that satisfy this requirement.

5.3.2 Control/data flow graphs

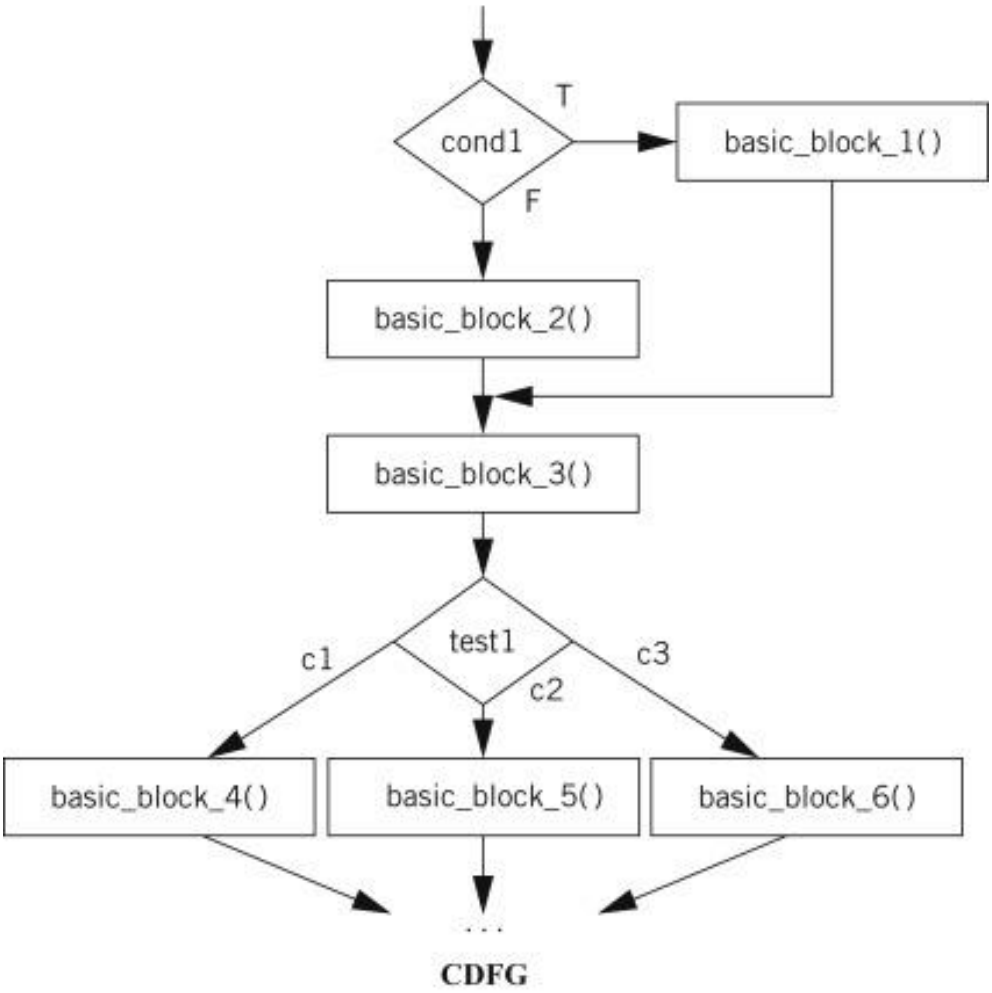
A CDFG uses a data flow graph as an element, adding constructs to describe control. In a basic CDFG, we have two types of nodes: decision nodes and data flow nodes. A data flow node encapsulates a complete data flow graph to represent a basic block. We can use one type of decision node to describe all the types of control in a sequential program. (The jump/branch is, after all, the way we implement all those high-level control constructs.)

Fig. 5.8 shows a bit of C code with control constructs and the CDFG constructed from it. The rectangular nodes in the graph represent the basic blocks. The basic blocks in the C code have been represented by function calls for simplicity. The

diamond-shaped nodes represent the conditionals. The node's condition is given by the label, and the edges are labeled with the possible outcomes of evaluating the condition.

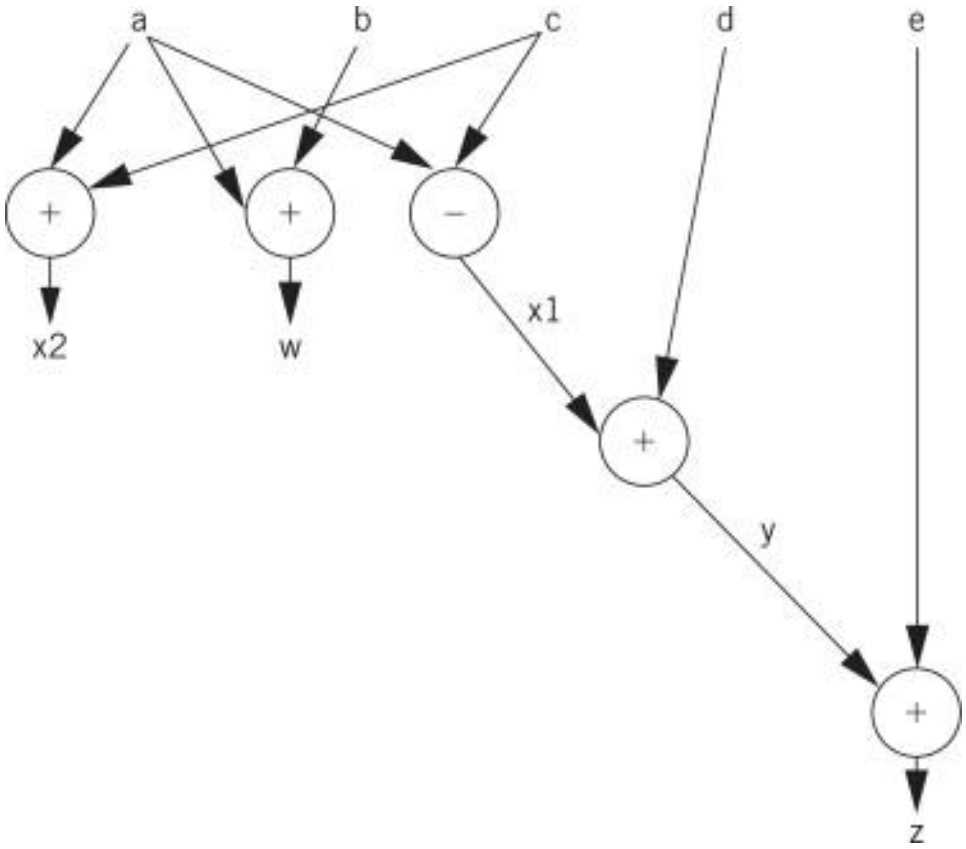
```
if (cond1)
      basic_block_1();
else
      basic_block_2();
basic_block_3();
switch (test1) {
      case c1: basic_block_4(); break;
      case c2: basic_block_5(); break;
      case c3: basic_block_6(): break;
}
```

C code



CDFG

The data flow graph is generally drawn in the form shown in Figure 5.7. Here, the variables are not explicitly represented by nodes. Instead, the edges are labeled with the variables they represent. As a result, a variable can be represented by more than one edge. However, the edges are directed and all the edges for a variable must come from a single source. We use this form for its simplicity and compactness.



Sign in to download full-size image

The data flow graph for the code makes the order in which the operations are performed in the C code much less obvious. This is one of the advantages of the data flow graph. We can use it to determine feasible reorderings of the operations, which may help us to reduce pipeline or cache conflicts. We can also use it when the exact order of operations simply doesn't matter. The data flow graph defines a partial ordering of the operations in the basic block. We must ensure that a value is computed before it is used, but generally there are several possible orderings of evaluating expressions that satisfy this requirement.

Introduction to 8 Bit and 16 Bit

Microcontrollers are like small computers that can carry out small programs and are often used for automation and robotics. The most popular to those who are just starting out are 8 bit and 16 bit microcontrollers. The main difference between 8 bit and 16 bit microcontrollers is the width of the data pipe. As you may have already deduced, an 8 bit microcontroller has an 8 bit data pipe while a 16 bit microcontroller has a 16 bit data pipe.

This fundamental difference between 8 bit and 16 bit microcontrollers is felt during mathematical operations. A 16 bit number gives you a lot more precision than 8 bit numbers. Although relatively rare, using an 8 bit microcontroller may not suffice the required accuracy of the application. 16 bit microcontrollers are also more efficient in

processing math operations on numbers that are longer than 8 bits. A 16 bit microcontroller can automatically operate on two 16 bit numbers, like the common definition of an integer. But when you are using an 8 bit microcontroller, the process is not as straightforward. The functions implemented to operate on such numbers will take additional cycles. Depending on how processing intensive your application is and on how many calculations you do, this may affect the performance of the circuit.

Another key difference between 8 bit and 16 bit microcontrollers is in their timers. 8 bit microcontrollers can only use 8 bits, resulting in a final range of 0x00 – 0xFF (0-255) every cycle. In contrast, 16 bit microcontrollers, with its 16 bit data width, has a range of 0x0000 – 0xFFFF (0-65535) for every cycle. A longer timer maximum value can surely come in handy in certain applications and circuits.

Initially, the price of 16 bit microcontrollers was way above that of 8 bit microcontrollers. But as time progressed and designs improved, the price of 8 bit and 16 bit microcontrollers has reduced quite a lot. 8 bit microcontrollers can be purchased dirt cheap. While 16 bit microcontroller cost more, prices tend to vary a lot depending on the features that are included in the microcontroller

Intel Microprocessor Architecture and Register set

The Intel Core microarchitecture (previously known as the Next-Generation Micro-Architecture) is a multi-core processor microarchitecture unveiled by Intel in Q1 2006. It is based on the Yonah processor design and can be considered an iteration of the P6 microarchitecture introduced in 1995 with Pentium Pro. High power consumption and heat intensity, the resulting inability to effectively increase clock speed, and other shortcomings such as an inefficient pipeline were the primary reasons why Intel abandoned the NetBurst microarchitecture and switched to a completely different architectural design, delivering high efficiency through a small pipeline rather than high clock speeds. The Core microarchitecture initially did not reach the clock speeds of the NetBurst microarchitecture, even after moving to 45 nm lithography. However after many generations of successor microarchitectures which used Core as their basis (such as Nehalem, Sandy Bridge and more), Intel managed to eventually surpass the clock speeds of Netburst with the Devil's Canyon (Improved version of Haswell) microarchitecture reaching a base frequency of 4 GHz and a maximum tested frequency of 4.4 GHz using 22 nm lithography.

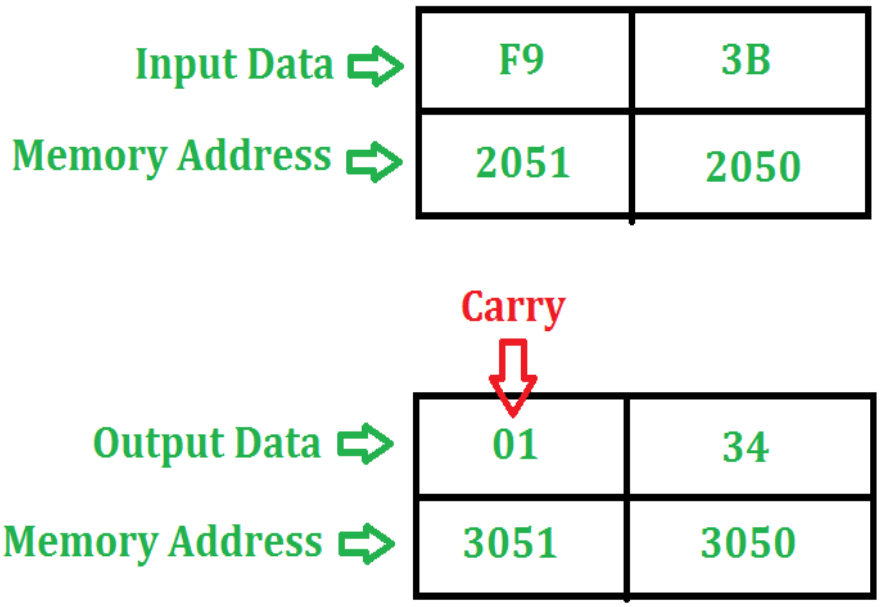
The first processors that used this architecture were code-named 'Merom', 'Conroe', and 'Woodcrest'; Merom is for mobile computing, Conroe is for desktop systems, and Woodcrest is for servers and workstations. While architecturally identical, the three processor lines differ in the socket used, bus speed, and power consumption. The initial mainstream Core-based processors were branded Pentium Dual-Core or Pentium and low end branded Celeron; server and workstation Core-based processors were branded Xeon, while Intel's first 64-bit desktop and mobile Core-based processors were branded Core 2.

Unit-V

Assembly language programming based on Intel 8085

Problem – Write an assembly language program to add two 8 bit numbers stored at address 2050 and address 2051 in 8085 microprocessor. The starting address of the program is taken as 2000.

Example –



Algorithm –

Load the first number from memory location 2050 to accumualtor.

Move the content of accumulator to register H.

Load the second number from memory location 2051 to accumaltor.

Then add the content of register H and accumulator using “ADD” instruction and storing result at 3050

The carry generated is recovered using “ADC” command and is stored at memory location 3051

Program –

MEMORY ADDRESS	MNEMONICS	COMMENT
2000	LDA 2050	A<-[2050]
2003	MOV H, A	H<-A
2004	LDA 2051	A<-[2051]
2007	ADD H	A<-A+H
2006	MOV L, A	L←A
2007	MVI A 00	A←00
2009	ADC A	A←A+A+carry

Explanation –

LDA 2050 moves the contents of 2050 memory location to the accumulator.

MOV H, A copies contents of Accumulator to register H to A

LDA 2051 moves the contents of 2051 memory location to the accumulator.

ADD H adds contents of A (Accumulator) and H register (F9). The result is stored in A itself. For all arithmetic instructions A is by default an operand and A stores the result as well

MOV L, A copies contents of A (34) to L

MVI A 00 moves immediate data (i.e., 00) to A

ADC A adds contents of A(00), contents of register specified (i.e A) and carry (1). As ADC is also an arithmetic operation, A is by default an operand and A stores the result as well

MOV H, A copies contents of A (01) to H

SHLD 3050 moves the contents of L register (34) in 3050 memory location and contents of H register (01) in 3051 memory location

HLT stops executing the program and halts any further execution

Data transfer or transfer is any information that is transferred from one location to another through some communication method. For example, with digital data transmission data signals are sent and received using the binary code.

Another example is for this page to be visible, all text, images, and other data was transferred over the Internet to your computer. For a file to appear on a USB drive, it must first be copied or moved from the hard drive to the USB drive.

How is information transferred on the Internet

Data can be transferred to and from computers over the Internet using one of following methods.

Send

If a user wants to transfer or send data to the Internet, they upload that data. Online file storage, like a NAS or SAN, is often used to store uploaded data.

Receive

If a user wants to receive or pull data from the Internet, they download that data. The data is often downloaded from a NAS or SAN file storage system.

Receive and send

It's also possible for users to upload and download data to and from other computers directly over the Internet. Users can utilize peer-to-peer communication to transfer files directly between computers, bypassing a file storage system.

What do I need to know about data transfer?

Data transfer refers to the secure exchange of large files between systems or organizations. In an internal context, data transfer is often used as an alternative to a holistic enterprise application integration system. However, data transfer is most often used to share data securely among business partners, suppliers, or government agencies for cooperative purposes.

What are the challenges of data transfer?

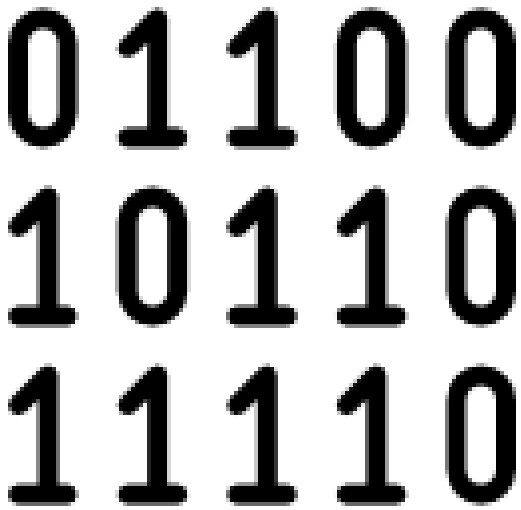
Data transfers are often used to share secure enterprise data with a business partner. Because the data is moving beyond the enterprise perimeter, care must be taken to secure the data. Besides security, another challenge of data transfer is complexity. Data transfers involving hundreds or thousands of files require a reporting and tracking component to make sure that data is copied completely and accurately and also to create a historical record of what was transferred, including metadata such as time, file size, and destination. Performance is also a concern as data transfer often involves very large files that can take a long time to process.

What are the benefits of data transfer?

A secure data transfer process will ensure that organizations meet compliance requirements. It will provide a record of the transactions so that the organization will know exactly what information was shared and so that any anomalies can be investigated. It will also provide a high degree of automation and performance enhancement, thus minimizing costs to complete the transaction.

Functions of the arithmetic logic unit (ALU)

Jump to navigationJump to search



How does a CPU work?

An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer. Modern CPUs contain very powerful and complex ALUs. In addition to ALUs, modern CPUs contain a control unit (CU).

Most of the operations of a CPU are performed by one or more ALUs, which load data from input registers. A register is a small amount of storage available as part of a CPU. The control unit tells the ALU what operation to perform on that data and the ALU stores the result in an output register. The control unit moves the data between these registers, the ALU, and memory.[1

What Is an ALU?

An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer. Modern CPUs contain very powerful and complex ALUs. In addition to ALUs, modern CPUs contain a control unit (CU).

Most of the operations of a CPU are performed by one or more ALUs, which load data from input registers. A register is a small amount of storage available as part of a CPU. The control unit tells the ALU what operation to perform on that data, and the

ALU stores the result in an output register. The control unit moves the data between these registers, the ALU, and memory.

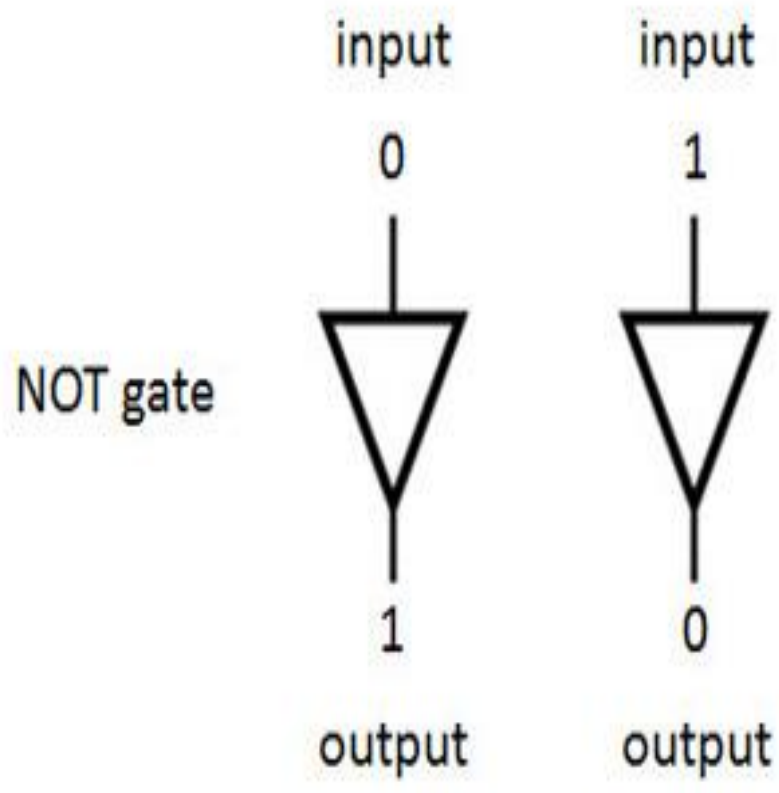
How an ALU Works

An ALU performs basic arithmetic and logic operations. Examples of arithmetic operations are addition, subtraction, multiplication, and division. Examples of logic operations are comparisons of values such as NOT, AND, and OR.

All information in a computer is stored and manipulated in the form of binary numbers, i.e. 0 and 1. Transistor switches are used to manipulate binary numbers since there are only two possible states of a switch: open or closed. An open transistor, through which there is no current, represents a 0. A closed transistor, through which there is a current, represents a 1.

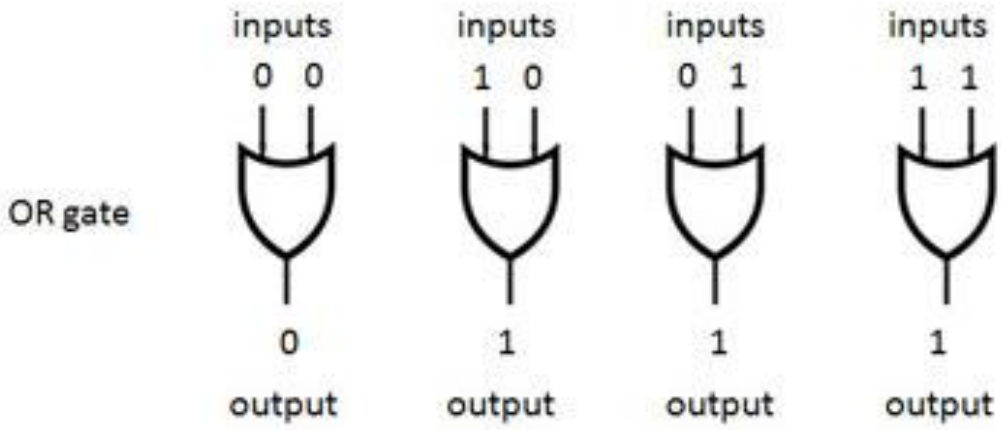
Operations can be accomplished by connecting multiple transistors. One transistor can be used to control a second one - in effect, turning the transistor switch on or off depending on the state of the second transistor. This is referred to as a gate because the arrangement can be used to allow or stop a current.

The simplest type of operation is a NOT gate. This uses only a single transistor. It uses a single input and produces a single output, which is always the opposite of the input. This figure shows the logic of the NOT gate:



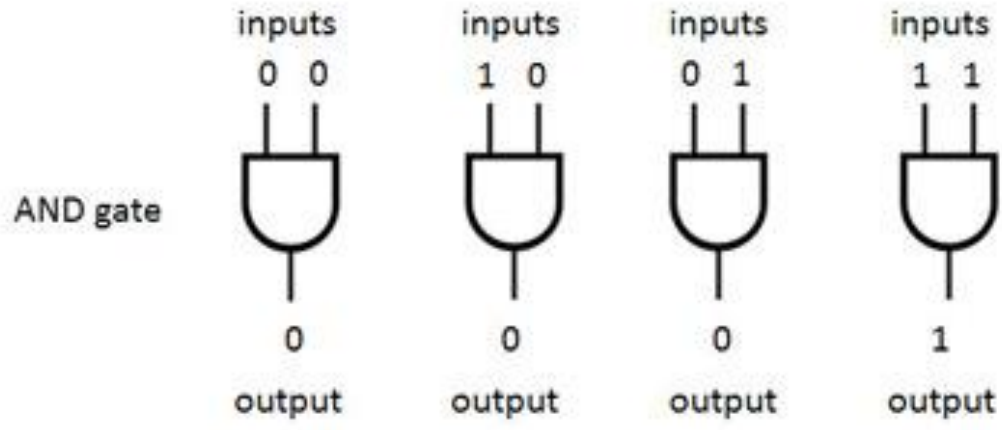
How a NOT gate processes binary data

Other gates consist of multiple transistors and use two inputs. The OR gate results in a 1 if either the first or the second input is a 1. The OR gate only results in a 0 if both inputs are 0. This figure shows the logic of the OR gate:



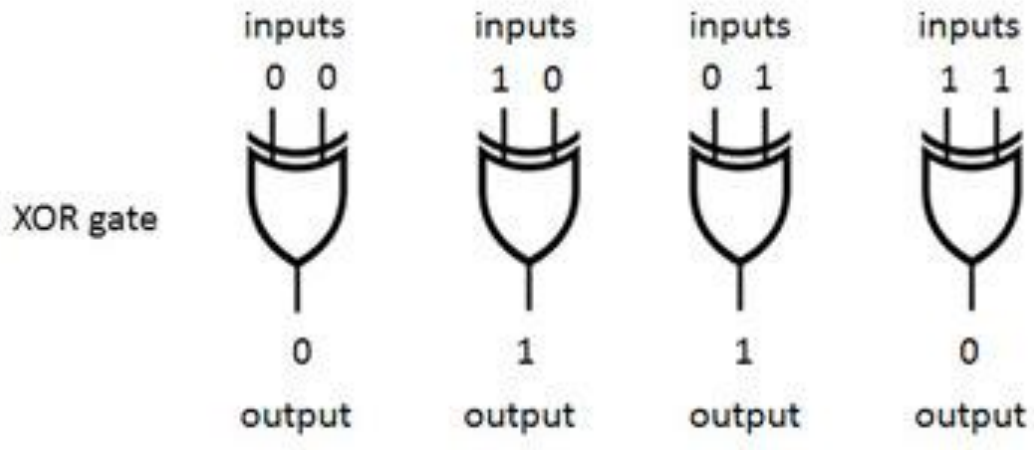
How an OR gate processes binary data

The AND gate results in a 1 only if both the first and second input are 1s. This figure shows the logic of the AND gate:



How an AND gate processes binary data

The XOR gate, also pronounced X-OR gate, results in a 0 if both the inputs are 0 or if both are 1. Otherwise, the result is a 1. This figure shows the logic of the XOR gate:



How an XOR gate processes binary data.

Branch operations

A branch operations manager is the subordinate of a branch manager who is delegated responsibility for all aspects of the office's technical and physical infrastructure, often including the supervision of all support staff.

Branch operations managers have varied backgrounds. They may have expertise in back-office operations, information technology, or other fields.

Responsibilities

A branch operations manager has responsibility for ensuring that financial advisors, sales assistants, and other branch office staff have the tools and infrastructure to perform their jobs and that this infrastructure is kept in working order. This infrastructure includes, but is not limited to:

Communications

Workstations

PCs

Copiers and fax machines

Power

Heating, ventilation, and air conditioning (HVAC)

Building maintenance

Furniture

Real estate operations

Mailroom operations

Cashier operations

Branch operations managers also play an important liaison role with the central IT and operations departments. They assist financial advisors and sales assistants with problems that they cannot resolve on their own. Depending on the structure of the firm, branch operations managers may or may not have either solid-line (primary) or dotted-line (secondary) reporting relationships to the central operations area of the firm.

Compensation

Branch operations managers are paid a salary and bonuses. Their bonus may be tied partly to overall branch results, or it may not.

Career Progression

Depending on the structure of the firm, the next logical step up for a branch operations manager might be as the supervisor of operations for a larger or more prestigious office, or for a larger aggregation of offices within the firm's branch office hierarchy, such as an office complex or a sales region. As a result, a willingness to relocate often is required to facilitate career advancement.

Relocation also may be necessary as a result of the closure of underperforming offices and the opening of new offices in geographic regions with sales growth or anticipated sales growth. Lastly, since experienced branch operations personnel are in demand throughout the securities brokerage industry, many opportunities to advance by changing firms also exist.

There also are chances for advancement by exiting branch operations management entirely. For example, one might have built the skill set necessary to win promotion into a spot within the central operations or information technology organizations of the firm.

Alternatively, the firsthand knowledge of workflows and processes that have been gained by an experienced branch operations manager can prove to be invaluable for financial management organizations that are involved in management reporting and transfer pricing.

For example, at Merrill Lynch, when the retail product profitability group made a significant push to upgrade its methodologies in the early 1990s and expand the knowledge base of its staff, its manager actively recruited experienced operations professionals to vastly increase the sophistication with which expenses were analyzed and allocated.

Looping Counting

Although every while loop has the same general form, while loops can be used to serve a variety of different purposes. A common form of loop is the counting loop, in which the loop repeats a fixed number of times.

Download count1.c and read it into an Emacs buffer. Compile and run the program. The program will prompt you for an integer; try entering 10 the first time, 5 the second time, and -5 the third time. Notice that the program prints out all of the integers from 1 up to the one that you entered. If the integer that you enter is smaller than 1, no integers are printed out.

The counting loop contained in count1.c is

printf("The integers from 1 up to %d are:\n", limit);

i = 1;

while (i<= limit) {

printf("%d ", i);

i = i + 1;

}

printf("\n");

A counting loop typically uses a variable to count from some initial value to some final value. This variable is often called the index variable. In this example, the index variable is i.

There are usually six components to a counting loop:

Setup statements. Statements before the loop that do something that needs to be done exactly once before the loop starts. In this case, the setup statement is

printf("The integers from 1 up to %d are:\n", limit);

Index initialization statement. A statement before the loop that gives the index variable its initial value. In this case, the initializing statement is

i = 1;

Index control expression. An expression that controls the loop by stopping it when the index variable exceeds its limit. In this case, the index control expression is

(i<= limit)

Body statements. Statements inside the body of the loop that do something with the index variable each trip through the loop. In this case, the body statement is

printf("%d ", i);

Index update statement. A statement inside the body of the loop that changes the value of the index variable each trip through the loop. In this case, the index update statement is

i = i + 1;

Final statements. Statements after the loop that do something that needs to be done exactly once after the loop stops. In this case, the final statement is

printf("\n");

A counting loop with always have an index initialization statement, an index control expression, one or more body statements, and an index update statement. It will not always have setup statements or final statements.

Exercises

Modify count1.c so that it allows the user to specify both the starting point and the stopping point.

Modify count1.c so that it prints only the even numbers less than or equal to limit.

A Counting Loop: Factorial

Download count2.c and read it into an Emacs buffer. This program is divided into a factorial function, which takes an integer n as its parameter and returns n!, and a main function, which exercises factorial. We will focus on the factorial function.

The counting loop that appears in factorial is

product = 1;

i = 1;

while (i<= n) {

product = product * i;

i = i + 1;

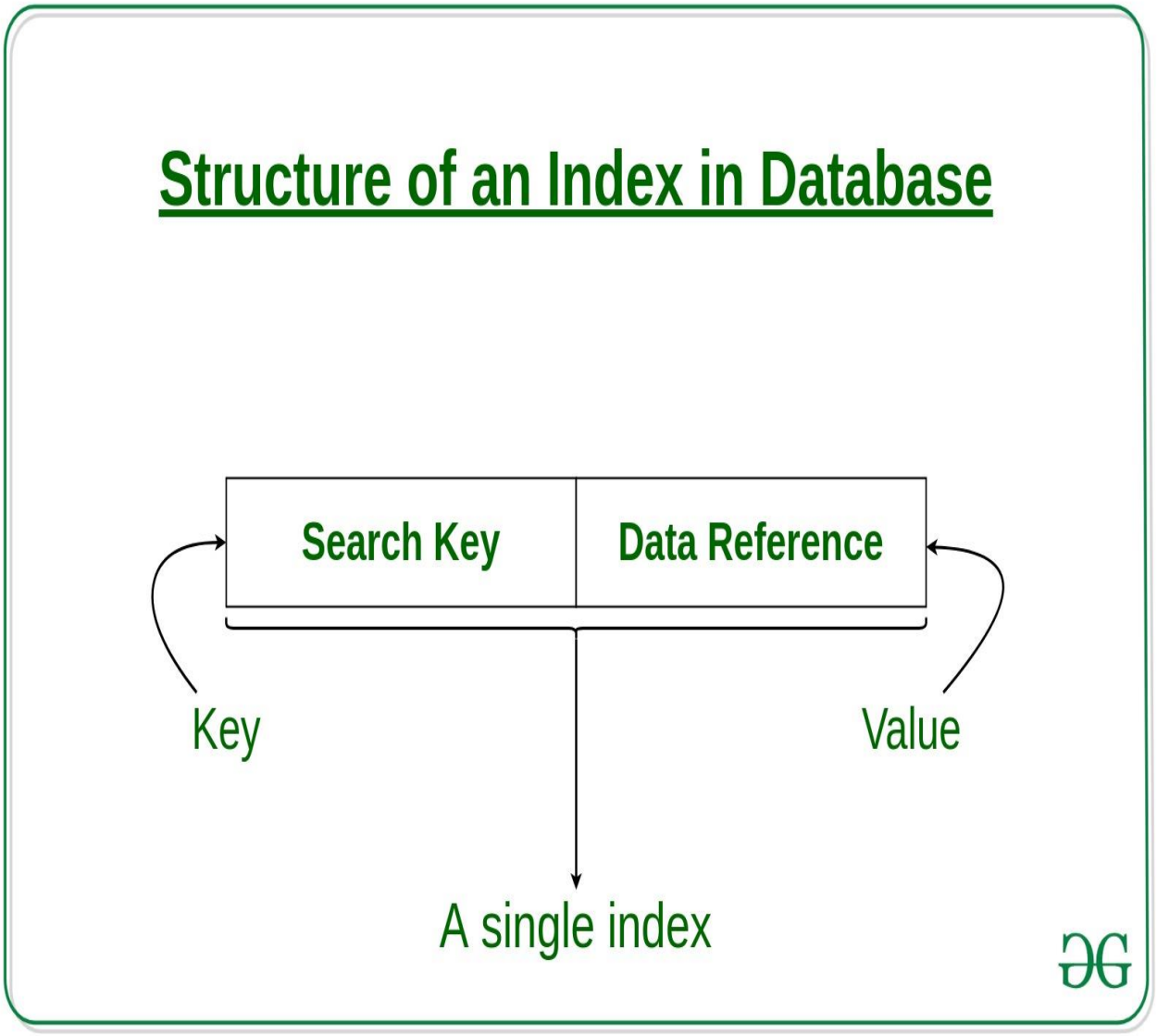
}

Indexing is a way to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed. It is a data structure technique which is used to quickly locate and access the data in a database.

Indexes are created using a few database columns.

The first column is the Search key that contains a copy of the primary key or candidate key of the table. These values are stored in sorted order so that the corresponding data can be accessed quickly.
Note: The data may or may not be stored in sorted order.

The second column is the Data Reference or Pointer which contains a set of pointers holding the address of the disk block where that particular key value can be found.



The indexing has various attributes:

Access Types: This refers to the type of access such as value based search, range access, etc.

Access Time: It refers to the time needed to find particular data element or set of elements.

Insertion Time: It refers to the time taken to find the appropriate space and insert a new data.

Deletion Time: Time taken to find an item and delete it as well as update the index structure.

Space Overhead: It refers to the additional space required by the index.

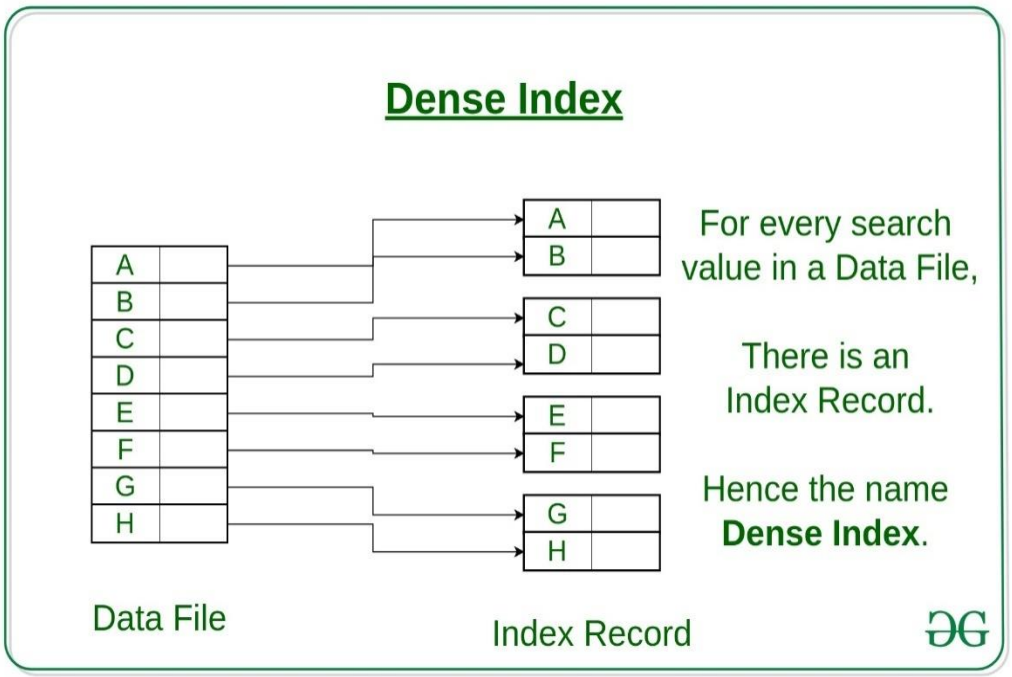
In general, there are two types of file organization mechanism which are followed by the indexing methods to store the data:

Sequential File Organization or Ordered Index File: In this, the indices are based on a sorted ordering of the values. These are generally fast and a more traditional type of storing mechanism. These Ordered or Sequential file organization might store the data in a dense or sparse format:

Dense Index:

For every search key value in the data file, there is an index record.

This record contains the search key and also a reference to the first data record with that search key value.

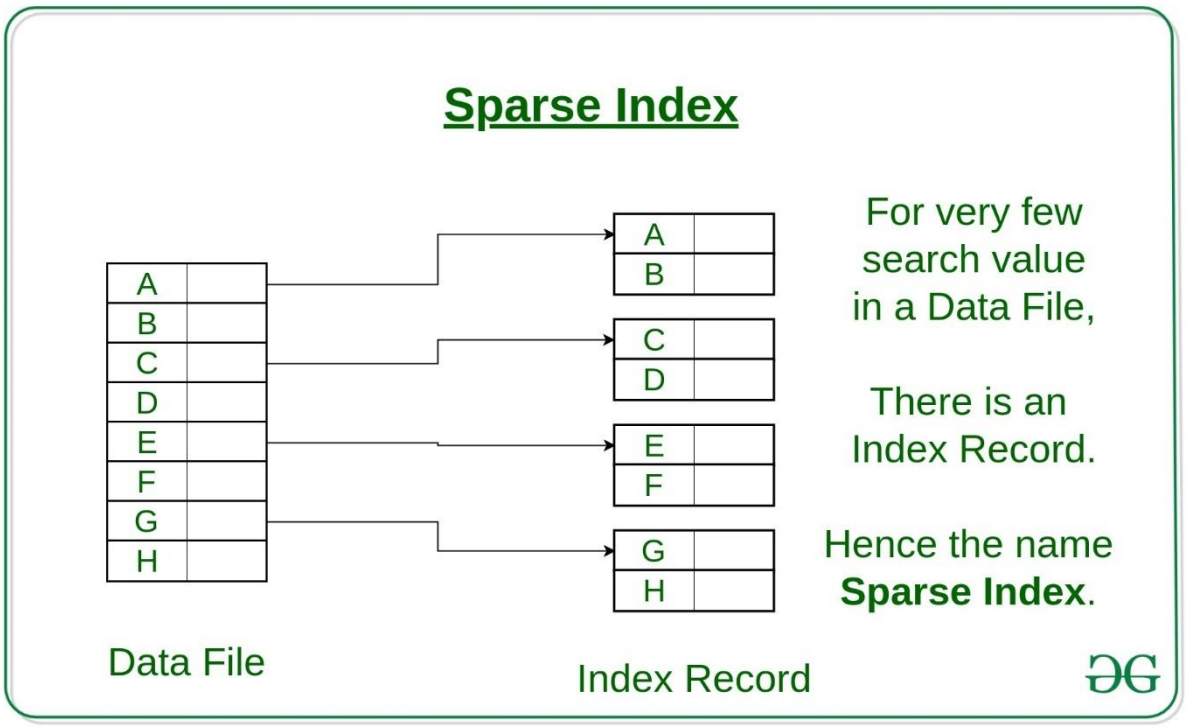


Sparse Index:

The index record appears only for a few items in the data file. Each item points to a block as shown.

To locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for.

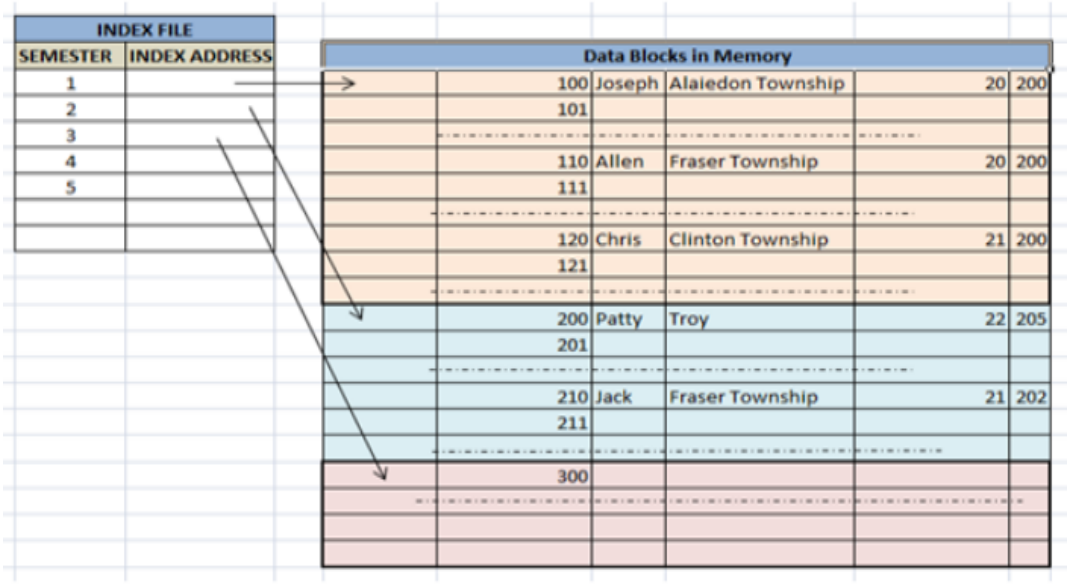
We start at that record pointed to by the index record, and proceed along with the pointers in the file (that is, sequentially) until we find the desired record.



Clustered Indexing

When more than two records are stored in the same file these types of storing known as cluster indexing. By using the cluster indexing we can reduce the cost of searching reason being multiple records related to the same thing are stored at one place and it also gives the frequent joing of more than two tables(records). Clustering index is defined on an ordered data file. The data file is ordered on a non-key field. In some cases, the index is created on non-primary key columns which may not be unique for each record. In such cases, in order to identify the records faster, we will group two or more columns together to get the unique values and create index out of them. This method is known as the clustering index. Basically, records with similar characteristics are grouped together and indexes are created for these groups.

For example, students studying in each semester are grouped together. i.e. 1st Semester students, 2nd semester students, 3rd semester students etc are grouped.

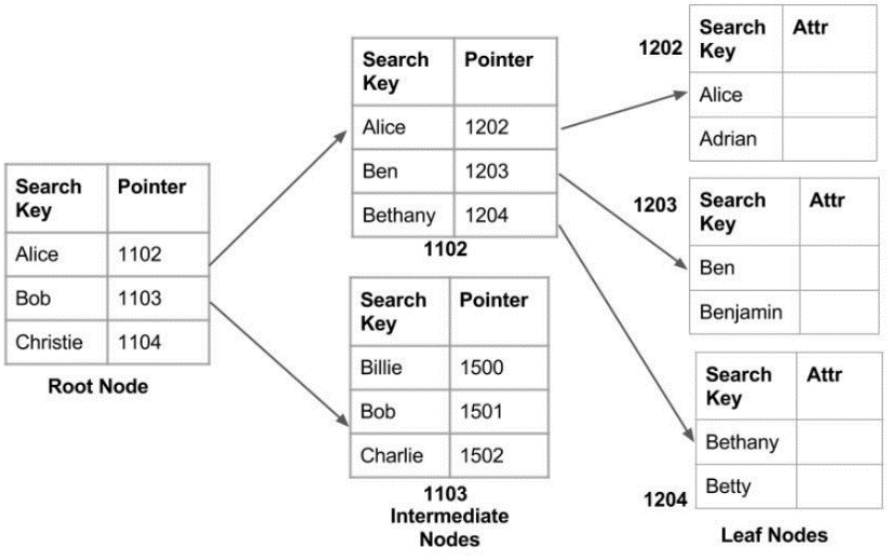


Primary Indexing:

This is a type of Clustered Indexing wherein the data is sorted according to the search key and the primary key of the database table is used to create the index. It is a default format of indexing where it induces sequential file organization. As primary keys are unique and are stored in a sorted manner, the performance of the searching operation is quite efficient.

Non-clustered or Secondary Indexing

A non clustered index just tells us where the data lies, i.e. it gives us a list of virtual pointers or references to the location where the data is actually stored. Data is not physically stored in the order of the index. Instead, data is present in leaf nodes. For eg.the contents page of a book. Each entry gives us the page number or location of the information stored. The actual data here(information on each page of the book) is not organized but we have an ordered reference(contents page) to where the data points actually lie. We can have only dense ordering in the non-clustered index as sparse ordering is not possible because data is not physically organized accordingly. It requires more time as compared to the clustered index because some amount of extra work is done in order to extract the data by further following the pointer. In the case of a clustered index, data is directly present in front of the index.



Non clustered index

Multilevel Indexing

With the growth of the size of the database, indices also grow. As the index is stored in the main memory, a single-level index might become too large a size to store with multiple disk accesses. The multilevel indexing segregates the main block into various smaller blocks so that the same can stored in a single block. The outer blocks are divided into inner blocks which in turn are pointed to the data blocks. This can be easily stored in the main memory with fewer overheads.

Programming Techniques

Programming is a complex matter. Therefore, every technique that helps to simplify it is very useful. In this article, we will consider simple and effective programming techniques which will help to simplify the work of the programmer.

Nowadays, it’s often being discussed the programming language which will be better for students who learn how to program. I fact, the programming language is not important because learning to program is learning to develop algorithmic thinking.

Programming techniques are almost in all aspects of our lives, starting from subconscious mind programming techniques to computer programming. So, here is a list of the most basic computer programming techniques. Some of them are obvious and don’t require to be described in detail.

In digital logic and computing, a counter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock. The most common type is a sequential digital logic circuit with an input line called the clock and multiple output lines. The values on the output lines represent a number in the binary or BCD number system. Each pulse applied to the clock input increments or decrements the number in the counter.

A counter circuit is usually constructed of a number of flip-flops connected in cascade. Counters are a very widely used component in digital circuits, and are manufactured as separate integrated circuits and also incorporated as parts of larger integrated circuits.

Electronic counters

An electronic counter is a sequential logic circuit which has a clock input signal and a group of output signals that represent an integer "counts" value. Upon each qualified clock edge, the circuit will increment (or decrement, depending on circuit design) the counts. When the counts have reached the end of the counting sequence (maximum counts when incrementing; zero counts when decrementing), the next clock will cause the counts to overflow or underflow and the counting sequence will start over. Internally, counters use flip-flops to represent the current counts and to retain the counts between clocks. Depending on the type of counter, the output may be a direct representation of the counts (a binary number) or it may be encoded. Examples of the latter include ring counters and counters that output Gray codes.

Many counters provide additional input signals to facilitate dynamic control of the counting sequence, such as:

Reset - sets counts to zero. Some IC manufactures name it "clear" or "master reset (MR)".

Enable - allows or inhibits counting.

Direction - determines whether counts will increment or decrement.

Data - parallel input data which represents a particular counts value.

Load - copies parallel input data to the counts.

Some counters provide a Terminal Count output which indicates that the next clock will cause overflow or underflow. This is commonly used to implement counter cascading (combining two or more counters to create a single, larger counter), by connecting the Terminal Count output of one counter to the Enable input of the next counter.

The modulus of a counter is the number of states in its count sequence. The maximum possible modulus is determined by the number of flip-flops. For example, a four-bit counter can have a modulus of up to 16 (2^4).

Counters are generally classified as either synchronous or asynchronous. In synchronous counters, all flip-flops share a common clock and change state at the same time. In asynchronous counters, each flip-flop has a unique clock and the flip-flop states change at different times.

Synchronous counters are categorized in various ways. For example:

Modulus counter - counts through a particular number of states.

Decade counter – modulus 10 counter (counts through ten states).

Up/down counter – counts both up and down, as directed by a control input.

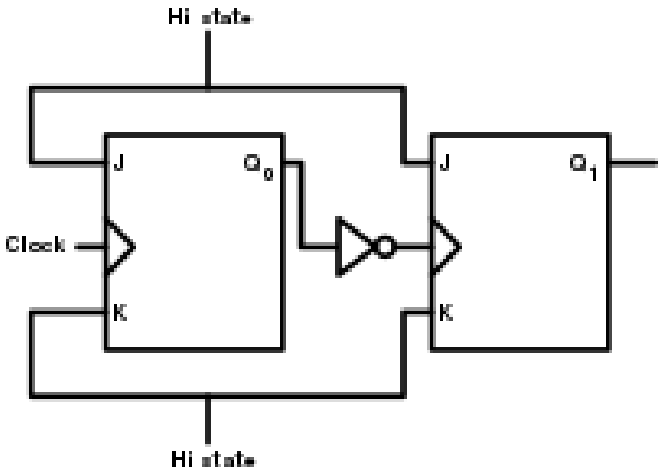
Ring counter – formed by a "circular" shift register.

Johnson counter – a twisted ring counter.

Gray code counter - outputs a sequence of Gray codes.

Counters are implemented in a variety of ways, including as dedicated MSI and LSI integrated circuits, as embedded counters within ASICs, as general-purpose counter and timer peripherals in microcontrollers, and as IP blocks in FPGAs.

Asynchronous (ripple) counter



Asynchronous counter created from two JK flip-flops

An asynchronous (ripple) counter is a "chain" of toggle (T) flip-flops wherein the least-significant flip-flop (bit 0) is clocked by an external signal (the counter input clock) and all other flip-flops are clocked by the output of the nearest, less significant flip-flop (e.g., bit 0 clocks the bit 1 flip-flop, bit 1 clocks the bit 2 flip-flop, etc.). The first flip-flop is clocked by rising edges; all other flip-flops in the chain are clocked by falling clock edges. Each flip-flop introduces a delay from clock edge to output toggle, thus causing the counter bits to change at different times and producing a ripple effect as the input clock propagates through the chain. When implemented with discrete flip-flops, ripple counters are commonly implemented with JK flip-flops, with each flip-flop configured to toggle when clocked (i.e., J and K are both connected to logic high).

In the simplest case, a one-bit counter consists of a single flip-flop. This counter will increment (by toggling its output) once per clock cycle and will count from zero to one before overflowing (starting over at zero). Each output state corresponds to two clock cycles, and consequently the flip-flop output frequency is exactly half the frequency of the input clock. If this output is then used as the clock signal for a second flip-flop, the pair of flip-flops will form a two-bit ripple counter with the following state sequence:

Clock cycle	Q1	Q0	(Q1:Q0) decimal
0	0	0	0
1	0	1	1
2	1	0	2
3	1	1	3
4	0	0	0

Additional flip-flops may be added to the chain to form counters of any arbitrary word size, with the output frequency of each bit equal to exactly half the frequency of the nearest, less significant bit.

Ripple counters exhibit unstable output states while the input clock is propagating through the circuit. The duration of this instability (the output settling time) is proportional to the number of flip-flops. This makes ripple counters unsuitable for use in synchronous circuits that require the counter to have a fast output settling time. Also, it is often impractical to use ripple counter output bits as clocks for external circuits because the ripple effect causes timing skew between the bits. Ripple counters are commonly used as general-purpose counters and clock frequency

dividers in applications where the instantaneous count and timing skew is unimportant.

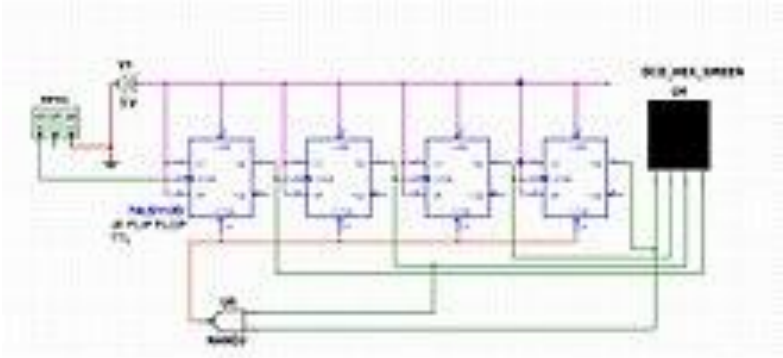
Synchronous counter

A 4-bit synchronous counter using JK flip-flops

In a synchronous counter, the clock inputs of the flip-flops are connected together and all flip-flops are simultaneously triggered by the common clock. Consequently, all of the flip-flops change state at the same time (in parallel).

For example, the circuit shown to the right is an ascending (up-counting) four-bit synchronous counter implemented with JK flip-flops. Each bit of this counter is allowed to toggle when all of the less significant bits are at a logic high state. Upon clock rising edge, bit 1 toggles if bit 0 is logic high; bit 2 toggles if bits 0 and 1 are both high; bit 3 toggles if bits 2, 1 and 0 are all high.

Decade counter



A circuit decade counter using JK Flip-flops (74LS112D)

A decade counter is one that counts in decimal digits, rather than binary. A decade counter may have each (that is, it may count in binary-coded decimal, as the 7490 integrated circuit did) or other binary encodings. A decade counter is a binary counter that is designed to count to 1010 (decimal 10). An ordinary four-stage counter can be easily modified to a decade counter by adding a NAND gate as in the schematic to the right. Notice that FF2 and FF4 provide the inputs to the NAND gate. The NAND gate outputs are connected to the CLR input of each of the FFs."[1] . It counts from 0 to 9 and then resets to zero. The counter output can be set to zero by pulsing the reset line low. The count then increments on each clock pulse until it reaches 1001 (decimal 9). When it increments to 1010 (decimal 10) both inputs of the NAND gate go high. The result is that the NAND output goes low, and resets the counter to zero. D going low can be a CARRY OUT signal, indicating that there has been a count of ten.

A ring counter is a circular shift register which is initiated such that only one of its flip-flops is the state one while others are in their zero states.

A ring counter is a shift register (a cascade connection of flip-flops) with the output of the last one connected to the input of the first, that is, in a ring. Typically, a pattern consisting of a single bit is circulated so the state repeats every n clock cycles if n flip-flops are used.

Johnson counter

A Johnson counter (or switch-tail ring counter, twisted ring counter, walking ring counter, or Möbius counter) is a modified ring counter, where the output from the last stage is inverted and fed back as input to the first stage.[2][3][4] The register cycles through a sequence of bit-patterns, whose length is equal to twice the length of the shift register, continuing indefinitely. These counters find specialist applications, including those similar to the decade counter, digital-to-analog conversion, etc. They can be implemented easily using D- or JK-type flip-flops.

Computer science counter

In computability theory, a counter is considered a type of memory. A counter stores a single natural number (initially zero) and can be arbitrarily long. A counter is usually considered in conjunction with a finite-state machine (FSM), which can perform the following operations on the counter:

Check whether the counter is zero

Increment the counter by one.

Decrement the counter by one (if it's already zero, this leaves it unchanged).

The following machines are listed in order of power, with each one being strictly more powerful than the one below it:

Deterministic or non-deterministic FSM plus two counters

Non-deterministic FSM plus one stack

Non-deterministic FSM plus one counter

Deterministic FSM plus one counter

Deterministic or non-deterministic FSM.

Variables. Variables can be considered as the most essential programming techniques. The number and their type depend on the language you are using.

Repetition or Loops. «For» is the most widely spread type of repetition. Most languages apply «for» to convey the idea of counting.

Decisions or Selection. To make the program flexible, we must make it respond to user input. Most algorithmic languages use a select method to control the program flow.

Arrays. Arrays are helpful for collections of similar items.

Modular Arithmetic. It will help to limit the number of program’s outputs or to make the things to “wrap around”. It’s one of the most plain and helpful programming techniques.

Manipulating Text. The text is being stored as numbers. The ability to convert the symbols to ASCII and vice-versa is a very helpful technique. You can apply it in many cases. One of them is to check for upper or lower case, for example. Chopping up strings is also very useful. You can display initials or create anagrams in such a way.

Random Numbers and Scaling. Scaling numbers is a helpful skill so try to master it. Randomness will also help if you want things to look more natural. For example, applying recursive programming techniques, you've made a tree in Scratch but it looks unnatural. Add some randomness to create a natural-looking effect.

WHAT PROFESSIONS REQUIRE PROGRAMMING SKILLS

The world is changing rapidly. New gadgets and electronic reviews appear almost every day. And today many professions require coding and programming skills – and not only those which are directly related to IT.

Programmer. Programmers make good money, and universal computerization increases the demand for IT-specialists.

Web-developer. The popularity of social networks, unlimited possibilities of the Internet for business, education, entertainment – all this makes the profession of web-programmer interesting, popular and highly paid.

Copywriter, SMM – and SEO-optimizer. People of these professions are directly related to the field of IT – their task is promoting sites on the world wide web and social platforms. They don't develop web resources and databases but the people of these professions must understand how they work.

Accountant, financier. People of these professions work a lot in specialized programs which are often very complex and require competent configuration. That's when they will need basic coding and programming skills.

Thus, the ability to program is extremely important for a person, no matter what profession he or she has chosen. Learning programming trains memory, thinking, logic and helps in achieving professional and life goals

The stack is a LIFO (last in, first out) data structure implemented in the RAM area and is used to store addresses and data when the microprocessor branches to a subroutine. Then the return address used to get pushed on this stack. Also to swap values of two registers and register pairs we use the stack as well.

In the programmer‘s view of 8085, only the general purpose registers A, B, C, D, E, H, and L, and the Flags registers were discussed so far. But in the complete programmer's view of 8085, there are two more special purpose registers, each of 16-bit width. They are the stack pointer, SP, and the program counter, PC. The Stack Pointer register will hold the address of the top location of the stack. And the program counter is a register always it will hold the address of the memory location

from where the next instruction for execution will have to be fetched. The complete programmer's view of 8085 is shown in the following figure.

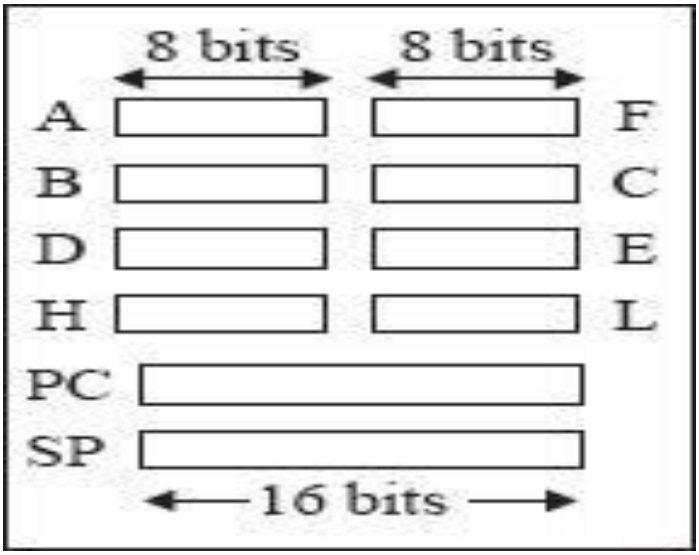


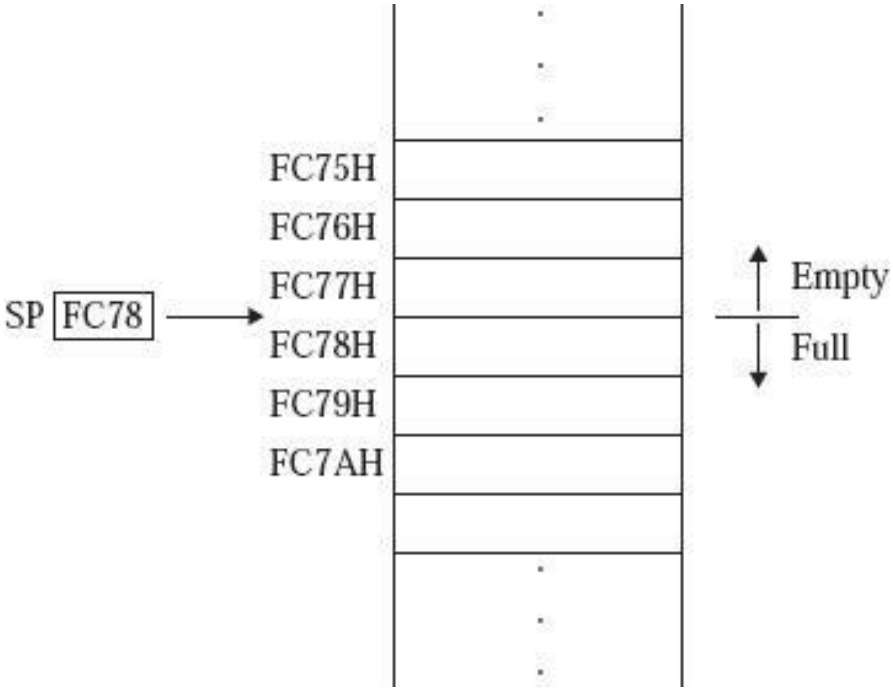
Fig. Programmer's view of 8085

SP is a special purpose 16-bit register. It contains a memory address. Suppose SP contents are FC78H, then the 8085 interprets it as follows.

Memory locations FC78H, FC79H, ..., FFFFH are having useful information. In other words, these locations are treated as filled locations. Memory locations FC77H, FC76H, ..., 0000H are not having any useful information. In other words, these locations are treated as empty locations.

On a stack, we can perform two operations. PUSH and POP. In case of PUSH operation, the SP register gets decreased by 2 and new data item used to insert on to the top of the stack. On the other hand, in case of POP operation, the data item will have to be deleted from the top of the stack and the SP register will get increased by the value of 2.

Thus, the contents of SP specify the top most useful location in the stack. In other words, it indicates the memory location with the smallest address having useful information. This is pictorially represented in the following figure –



Tutorialskeyboard_arrow_down

Studentkeyboard_arrow_down

Courses

Jobskeyboard_arrow_down

Top of Form

Bottom of Form

Sign In

Sign In

Home

Courses

Algorithmskeyboard_arrow_down

Data Structureskeyboard_arrow_down

Languageskeyboard_arrow_down

Interview Cornerkeyboard_arrow_down

GATEkeyboard_arrow_down

CS Subjectskeyboard_arrow_down

Studentkeyboard_arrow_down

Jobskeyboard_arrow_down

GBlog

Puzzles

What's New ?

Logical instructions in 8085 microprocessor

Data transfer instructions in 8085 microprocessor

Branching instructions in 8085 microprocessor

Reset Accumulator (8085 & 8086 microprocessor)

Difference between CALL and JUMP instructions

Simplified Instructional Computer (SIC)

Instruction Set used in simplified instructional Computer (SIC)

Instruction Set used in SIC/XE

Computer Organization | RISC and CISC

Difference between RISC and CISC processor | Set 2

Vector processor classification

Introduction of Single Accumulator based CPU organization

Computer Organization | Problem Solving on Instruction Format

Computer Organization | Instruction Formats (Zero, One, Two and Three Address Instruction)

Addressing Modes

Addressing modes in 8085 microprocessor

Flag register in 8085 microprocessor

Flag register of 8086 microprocessor

Addressing modes in 8086 microprocessor

Memory Segmentation in 8086 Microprocessor

General purpose registers in 8086 microprocessor

Registers of 8085 microprocessor

Pin diagram of 8085 microprocessor

Interrupts in 8085 microprocessor

Interrupts in 8086 microprocessor

Program for Decimal to Binary Conversion

Cache Memory in Computer Organization

Program for Binary To Decimal Conversion

Random Access Memory (RAM) and Read Only Memory (ROM)

Branching instructions in 8085 microprocessor

Last Updated: 29-07-2020

Branching instructions refer to the act of switching execution to a different instruction sequence as a result of executing a branch instruction.

The three types of branching instructions are:

Jump (unconditional and conditional)

Call (unconditional and conditional)

Return (unconditional and conditional)

1. Jump Instructions – The jump instruction transfers the program sequence to the memory address given in the operand based on the specified flag. Jump instructions are 2 types: Unconditional Jump Instructions and Conditional Jump Instructions.

(a) Unconditional Jump Instructions: Transfers the program sequence to the described memory address.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
JMP	address	Jumps to the address	JMP 2050

(b) Conditional Jump Instructions: Transfers the program sequence to the described memory address only if the condition in satisfied.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
JC	address	Jumps to the address if carry flag is 1	JC 2050
JNC	address	Jumps to the address if carry flag is 0	JNC 2050
JZ	address	Jumps to the address if zero flag is 1	JZ 2050
JNZ	address	Jumps to the address if zero flag is 0	JNZ 2050
JPE	address	Jumps to the address if parity flag is 1	JPE 2050
JPO	address	Jumps to the address if parity flag is 0	JPO 2050
JM	address	Jumps to the address if sign flag is 1	JM 2050
JP	address	Jumps to the address if sign flag 0	JP 2050

2. Call Instructions – The call instruction transfers the program sequence to the memory address given in the operand. Before transferring, the address of the next instruction after CALL is pushed onto the stack. Call instructions are 2 types: Unconditional Call Instructions and Conditional Call Instructions.

(a) Unconditional Call Instructions: It transfers the program sequence to the memory address given in the operand.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
CALL	address	Unconditionally calls	CALL 2050

(b) Conditional Call Instructions: Only if the condition is satisfied, the instructions executes.

OPCODE	OPERAND	EXPLANATION	EXAMPLE

OPCODE	OPERAND	EXPLANATION	EXAMPLE
CC	address	Call if carry flag is 1	CC 2050
CNC	address	Call if carry flag is 0	CNC 2050
CZ	address	Calls if zero flag is 1	CZ 2050
CNZ	address	Calls if zero flag is 0	CNZ 2050
CPE	address	Calls if parity flag is 1	CPE 2050
CPO	address	Calls if parity flag is 0	CPO 2050
CM	address	Calls if sign flag is 1	CM 2050
CP	address	Calls if sign flag is 0	CP 2050

3. Return Instructions – The return instruction transfers the program sequence from the subroutine to the calling program. Return instructions are 2 types: Unconditional Jump Instructions and Conditional Jump Instructions.

(a) Unconditional Return Instruction: The program sequence is transferred unconditionally from the subroutine to the calling program.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
RET	none	Return from the subroutine unconditionally	RET

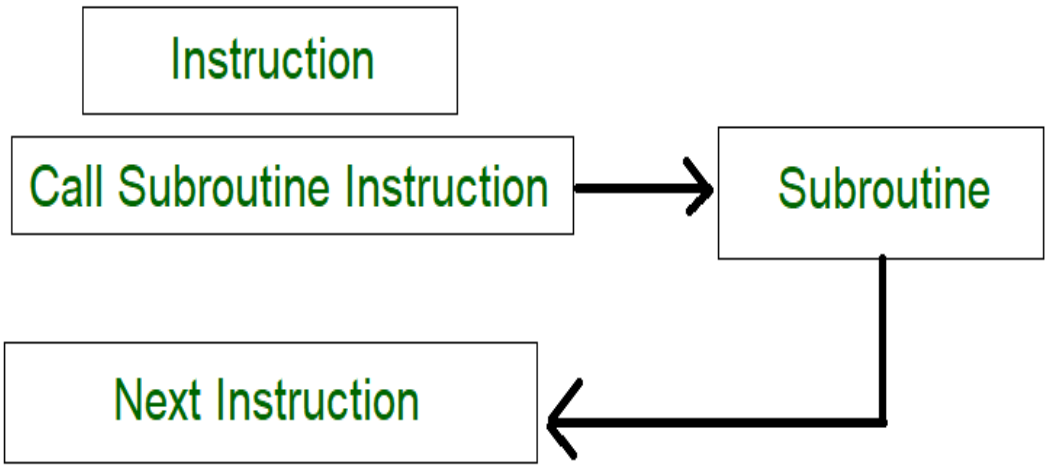
(b) Conditional Return Instruction: The program sequence is transferred unconditionally from the subroutine to the calling program only is the condition is satisfied.

OPCODE	OPERAND	EXPLANATION	EXAMPLE
RC	none	Return from the subroutine if carry flag is 1	RC

OPCODE	OPERAND	EXPLANATION	EXAMPLE
RNC	none	Return from the subroutine if carry flag is 0	RNC
RZ	none	Return from the subroutine if zero flag is 1	RZ
RNZ	none	Return from the subroutine if zero flag is 0	RNZ
RPE	none	Return from the subroutine if parity flag is 1	RPE
RPO	none	Return from the subroutine if parity flag is 0	RPO
RM	none	Returns from the subroutine if sign flag is 1	RM
RP	none	Returns from the subroutine if sign flag is 0	RP

1. Subroutine –
A set of Instructions which are used repeatedly in a program can be referred to as Subroutine. Only one copy of this Instruction is stored in the memory. When a Subroutine is required it can be called many times during the Execution of a Particular program. A call Subroutine Instruction calls the Subroutine. Care Should be taken while returning a Subroutine as Subroutine can be called from a different place from the memory.

The content of the PC must be Saved by the call Subroutine Instruction to make a correct return to the calling program.



2. Subroutine Nesting –
Subroutine nesting is a common Programming practice In which one Subroutine call another Subroutine.

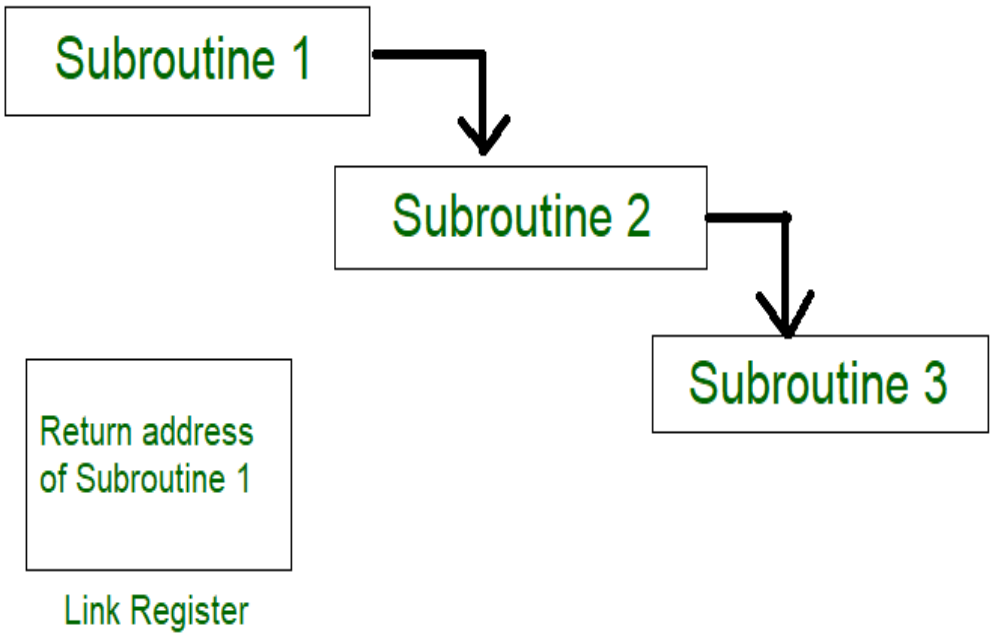


Figure – Subroutine calling another subroutine

From the above figure, assume that when Subroutine 1 calls Subroutine 2 the return address of Subroutine 2 should be saved somewhere. So if link register stores return address of Subroutine 1 this will be (destroyed/overwritten) by return address of Subroutine 2. As the last Subroutine called is the first one to be returned (Last in first out format). So stack data structure is the most efficient way to store the return addresses of the Subroutines.

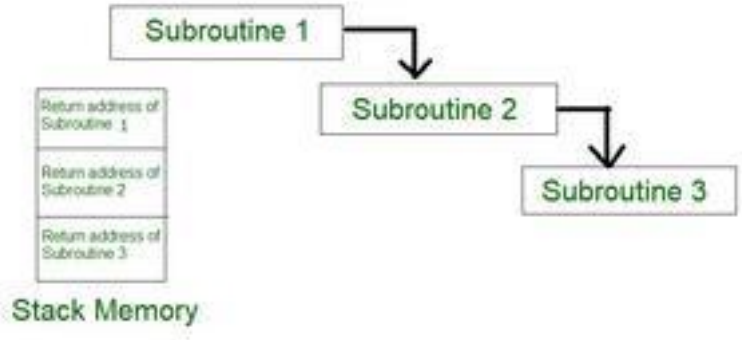


Figure – Return address of subroutine is stored in stack memory

3. Stack memory –
Stack is a basic data structure which can be implemented anywhere in the memory. It can be used to store variables which may be required afterwards in the program Execution. In a stack, the first data put will be last to get out of a stack. So the last data added will be the First one to come out of the stack (last in first out).

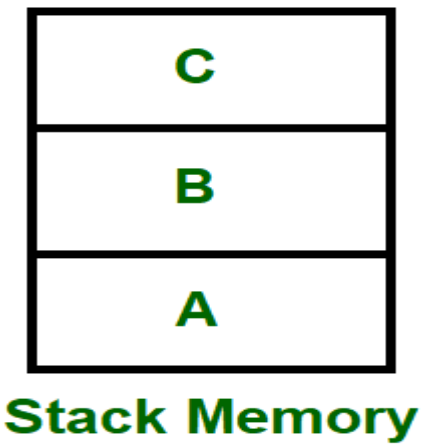


Figure – Stack memory having data A, B & C

So from the diagram above first A is added then B & C. While removing first C is Removed then B & A.