African Virtual University

Applied Computer Science: CSI 3100

ADVANCED COMPUTER ORGANISATION AND ARCHITECTURE

Harrison Njoroge

Foreword

The African Virtual University (AVU) is proud to participate in increasing access to education in African countries through the production of quality learning materials. We are also proud to contribute to global knowledge as our Open Educational Resources are mostly accessed from outside the African continent.

This module was developed as part of a diploma and degree program in Applied Computer Science, in collaboration with 18 African partner institutions from 16 countries. A total of 156 modules were developed or translated to ensure availability in English, French and Portuguese. These modules have also been made available as open education resources (OER) on oer.avu.org.

On behalf of the African Virtual University and our patron, our partner institutions, the African Development Bank, I invite you to use this module in your institution, for your own education, to share it as widely as possible and to participate actively in the AVU communities of practice of your interest. We are committed to be on the frontline of developing and sharing Open Educational Resources.

The African Virtual University (AVU) is a Pan African Intergovernmental Organization established by charter with the mandate of significantly increasing access to quality higher education and training through the innovative use of information communication technologies. A Charter, establishing the AVU as an Intergovernmental Organization, has been signed so far by nineteen (19) African Governments - Kenya, Senegal, Mauritania, Mali, Cote d'Ivoire, Tanzania, Mozambique, Democratic Republic of Congo, Benin, Ghana, Republic of Guinea, Burkina Faso, Niger, South Sudan, Sudan, The Gambia, Guinea-Bissau, Ethiopia and Cape Verde.

The following institutions participated in the Applied Computer Science Program: (1) Université d'Abomey Calavi in Benin; (2) Université de Ougagadougou in Burkina Faso; (3) Université Lumière de Bujumbura in Burundi; (4) Université de Douala in Cameroon; (5) Université de Nouakchott in Mauritania; (6) Université Gaston Berger in Senegal; (7) Université des Sciences, des Techniques et Technologies de Bamako in Mali (8) Ghana Institute of Management and Public Administration; (9) Kwame Nkrumah University of Science and Technology in Ghana; (10) Kenyatta University in Kenya; (11) Egerton University in Kenya; (12) Addis Ababa University in Ethiopia (13) University of Rwanda; (14) University of Dar es Salaam in Tanzania; (15) Universite Abdou Moumouni de Niamey in Niger; (16) Université Cheikh Anta Diop in Senegal; (17) Universidade Pedagógica in Mozambique; and (18) The University of the Gambia in The Gambia.

Bakary Diallo

The Rector

African Virtual University

Production Credits

Author

Harrison Njoroge

Peer Reviewer

Pelagie Houngue

AVU - Academic Coordination

Dr. Marilena Cabral

Overall Coordinator Applied Computer Science Program

Prof Tim Mwololo Waema

Module Coordinator

Robert Oboko

Instructional Designers

Elizabeth Mbasu

Benta Ochola

Diana Tuel

Media Team

Sidney McGregor	Michal Abigael Koyier
Barry Savala	Mercy Tabi Ojwang
Edwin Kiprono	Josiah Mutsogu
Kelvin Muriithi	Kefa Murimi
Victor Oluoch Otieno	Gerisson Mulongo

Copyright Notice

Supported By

AVU Multinational Project II funded by the African Development Bank.

Table of Contents

Course Overview

Welcome to Advanced Computer Organization and Architecture

This course introduces students to Advanced Computer Organization and Architecture. Where architecture is a term used to describe the attributes of a system as seen by the programmer. Its concerned with designs and operations of a computer. Computer organization is the way the system is structured so that all those cataloged tools can be used, and that in an efficient fashion. This course has its emphasis on system design and performance.

Prerequisites

- Fundamentals of computer organization and architecture

Materials

The materials required to complete this course are:

Textbooks:

- Mostafa Abd-El-Barr , Hesham El-Rewini; Fundamentals of Computer Organization and Architecture and Advanced Computer Architecture and Parallel Processing; Edition 2: ISBN-13: 978-0471703808
- William Stallings; Computer Organization and Architecture; 5th Edition, 2000
- Manoj Franklin; Computer Architecture and Organization: From Software to Hardware; 2007
- David Patterson, John L. Hennessy; Computer Organization and Design: the Hardware/Software Interface; 1994

http://www.ece.umd.edu/~manoj/350/notes/book.pdf

http://iiusatech.com/murdocca/CAO/SlidesPDF/Ch10CAO.pdf

http://ir.nmu.org.ua/bitstream/handle/123456789/125912/03e2069ba199dcfd9990cb0d2c1e23ba.pdf

http://www.cse.iitm.ac.in/~vplab/courses/comp_org/LEC_INTRO.pdf

Computer System

Course Goals

Upon completion of this course the learner should be able to;

- Describe different computer architectures
- Describe the operation of different hardware subsystems
- Analyze high performance computer system.
- Compare different high performance computer system.

Units

- Unit 0: Fundamentals of computer organization and architecture
  Recap of this module

- Unit 1: Functional Organization_

Internal operations in a computer; Micro-architectures; Instructions plumbing and Instruction level parallelism (ILP); Performance Processor and system; Multilevel cache, and cache coherency

- Unit 2: multiprocessing

The Amdahl's law; short vector processing (multimedia operations); The multi-core and multi-processor -segments; Taxonomy Flynn: Structures and multiprocessor architectures; Scheduling multiprocessor systems

- Unit 3:Organization and low-level programming

Structure of low-level programs; Limitations of low-level architectures; Architecture support from low level to high level languages;

- Unit 4:· Strategies and Interface I / O

Fundamentals I / O: handshake and buffering; Mechanisms of interruption: recognition of vector, and interrupt priority; Vehicles (Buses): protocols, arbitration, interrupts, direct memory access (DMA)

- Unit 5: The peripheral devices

Representation of digital and analog values - sampling and quantization; Standards multimedia (audio, music, graphics, image, telephony, video, TV); Coding and decoding multimedia systems; Compression and decompression of data; Input devices: mice,

Assessment

Formative assessments, used to check learner progress, are included in each unit.

Summative assessments, such as final tests and assignments, are provided at the end of each module and cover knowledge and skills from the entire module.

Summative assessments are administered at the discretion of the institution offering the course. The suggested assessment plan is as follows:

1	Class assignments	20%
2	Continuous Assessment Tests	30%
4	End of semester exams	50%
Total		100%

Schedule

Unit	Activities	Estimated time
0	Preview of the pre-requisite 1. Architecture computer system 2. organization of computer components and systems	6 hours
1	Functional Organization • Activity 1.1 - Revision of language to describe the transfer of registration of internal operations in a computer • Activity 1.2 - Micro-architectures - Achievements connections by wires and micro-programmed • Activity 1.3 - Instructions plumbing and Instruction level parallelism (ILP) • Activity 1.4 - Processor and system Performance • Unit Assessment	22 hours
2	Multiprocessing • Activity 2.1 - The Amdahl's law • Activity 2.2 - Short Vector Processing (Multimedia Operations) • Activity 2.3 - The Multi-core and Multi-processor-Segments • Activity 2.4 - Taxonomy Flynn: Structures and multiprocessor architectures • Activity 2.5 - Scheduling Multiprocessor Systems	22 hours
3	Organization and low-level programming • Activity 3.1 - Structure of Low-level Programs • Activity 3.2 Architecture Support from Low-level to High-level Languages	20 hours
4	Strategies and Interface I / O • Activity 4.1 - Fundamentals I/O: handshake and buffering • Activity 4.2 Mechanisms of Interruption: Recognition of Vector, and Interrupt Priority • Activity 4.3 Direct Memory Access (DMA)	25 hours

5	The Peripheral Devices • Activity 5.1 - Representation of Digital and Analog Values - Sampling and Quantization • Activity 5.2 - Sound and Audio, Image and Graphics, Animation and Video • Activity 5.3 - Coding and Decoding Multimedia Systems	25 hours
Total		120 hours

Readings and Other Resources

The readings and other resources in this course are:

Unit 0

Required readings and other resources:

- David L. Tarnoff, Computer Organization and Design Fundamentals, publisher: Lulu.com , 2005
- Bjarne Stroustrup, Programming: Principles and Practice Using C++, ISBN 0321992784, Publisher: Addison Wesley, 2014
- Computer Organization & Design: The Hardware/Software Interface, by Patterson and Hennessy Fourth Edition, Morgan-Kaufman (2009). ISBN: 978012374493
- Computer Organization & Design: The Hardware/Software Interface, by Patterson and Hennessy; Revised Fourth Edition, Morgan-Kaufman (2009). ISBN: 9780123747501

Unit 1

Required readings and other resources:

- Hwang, Kai. Advanced computer architecture: parallelism, scalability, programmability. 1993
- Shiva, Sajjan G., Computer organization, design, and architecture. 2014

Unit 2

Required readings and other resources:

- George W. Zobrist, Kallol Bagchi and Kishor Trivedi. Advanced computer system design. 1998
- Greg Astfalk, Applications on advanced architecture computers.1996
- El-Rewini, Hesham, Mostafa Abd-El-Barr. Advanced computer architecture and parallel processing. 2005

Unit 3

Required readings and other resources:

- Hyde, Randal, The art of assembly language. 2010. 9781593272074 (pbk.)
- Patterson, David A., Computer organization and design: the hardware/software interface. 2014
- Andrew S. Tanenbaum. Structured Computer Organization, (5th Edition). 2005; ISBN-13: 978-0131485211

Unit 4

Required readings and other resources:

- Patterson, David A., Computer organization and design: the hardware/software interface. 2014
- Abd-El-Barr, Mostafa, Fundamentals of computer organization and architecture. 2005
- David Patterson, John L. Hennessy; Computer Organization and Design: the Hardware/Software Interface; 1994

Unit 5

Required readings and other resources:

- Patterson, David A. John L. Hennessy. Computer organization and design: the hardware/software interface. 2012
- Hong Lin. Architectural design of multi-agent systems: technologies and techniques. 2007
- David Patterson, John L. Hennessy; Computer Organization and Design: the Hardware/Software Interface; 1994

Unit 0. Pre-Assessment

Unit Introduction

The purpose of this unit is to determine your grasp of prior knowledge related to this course. The units' expectations are that the students have prior knowledge on basic architecture and organization of computers. A recap of this course/s will prepare and enable the students have a grasp of computer architecture and organization.

Unit Objectives

Upon completion of this unit you should be able to:

- Explain how computer systems work & computer systems basic principles
- Analyze computer system performance.
- Describe concepts behind advanced pipelining techniques.
- Survey the current state of art in memory system design
- Describe how I/O devices are accessed and principles applied

Key Terms

Computer system: A term that describes the computer along with any software and peripheral devices that are necessary to make the computer function.

Computer Architecture: is the attribute which can be recognized by a programmer and which has a direct effect to program execution. Describes the capabilities and programming model of a computer but not a particular implementation

Computer organization: Term used to describe the way the hardware components operate and the way they are connected together to form the computer system. The various components are assumed to be in place and the task is to investigate the organizational structure to verify that the computer parts operate.

Computer performance: is characterized by the amount of useful work accomplished by a computer system or computer network compared to the time and resources used I/O: Accessories that allow computer to perform specific tasks;-

Receive information for processing, Return the results of processing, Store information

Memory: is a collection of cells, each with a unique physical address.

Unit Assessment

Check your understanding!

Class Assignment

Learners are invited to read on basic computer architecture and organization which is the pre-requisite courses for this module.

Instructions

The following questions will help gauge how much you already know about basic architecture and organization of computers which are the pre-requisites of architecture and organization of advanced computers

1. Define the following terms

   a). Computer architecture

   b). Computer organization

   c). Peripheral devices

2. What is the concept of layers in architectural design?

3. What processor performs the fetching or decoding of different instruction during the execution of another instruction.

Grading Scheme

The marks will be awarded as shown below

Question	Sub-question	Marks Awarded
1	(a).	2
	(b).	2
	(c).	2
2	Each term will be awarded not more than 2 marks	8
3	Naming (2 marks); explanation (4 marks)	5
Total		20

Feedback

1. Define the following terms

(i). Computer architecture:- is concerned with structure and behavior of computer as seen by the user. It includes the information formats, the instruction set, and techniques for addressing the architectural design of a computer system. it is also concerned with the specifications of the various functional modules, such as processors and memories and structuring them together into a computer system

(ii).Computer organization;- is concerned with the way the hardware components operate and the way they are connected together to form the computer system. The various components are assumed to be in place and the task is to investigate the organizational structure to verify that the computer parts operate.

(iii). Peripheral devices;- is defined as a computer device, such as a keyboard or printer, that is not part of the essential computer (i.e., the memory and microprocessor).

2. What is the concept of layers in architectural design?

The concepts of layers in architectural design is described as below:

(i). Complex problems can be segmented into smaller and more manageable form

(ii). Each layer is specialized for specific functioning.

(iii). Upper layers can share the services of a lower layer. Thus layering allows us to reuse functionality.

(iv). Team development is possible because of logical segmentation. A team of programmers will build. The system, and work has to be sub-divided of along clear boundaries

3. Explain the processor that performs the fetching or decoding of different instruction during the execution of another instruction.

Pipe-lining

Explanation: Pipe-lining is the process of improving the performance of the system by processing different instructions at the same time, with only one instruction performing one specific operation

Unit Readings and Other Resources

The readings in this unit are to be found at the course-level section “Readings and Other Resources”.

Unit 1. Functional Organization

Unit Introduction

This unit introduces learners to functional organization of a computer. The unit provides you with basic concepts and techniques that will get you started in understanding and analysis of hardware and software interaction in computer systems.

Unit Objectives

Upon completion of this unit the learner should be able to:

- Describe functional organization of the computer
- Explain the basic principles of organization, operation and performance of modern-day computer systems
- Outline the architectural design of the computer system

Key Terms

Computer performance: is characterized by the amount of useful work accomplished by a computer system or computer network compared to the time and resources used.

Micro architecture: is a description of the electrical circuitry of a computer, central processing unit, or digital signal processor that is sufficient for completely describing the operation of the hardware.

Cache memory: is random access memory (RAM) that a computer microprocessor can access more quickly than it can access regular RAM. This memory is typically integrated directly with the CPU chip or placed on a separate chip that has a separate bus interconnect with the CPU

Learning Activities

Activity 1.1 - Revision of Language to Describe the Transfer of Registration of Internal Operations in a Computer

<u>Introduction</u>

In this learning activity section, the learner will be able to learn languages that are used to describe the transfer of registration of internal operations in the computer.

Activity Details

Register Transfer language

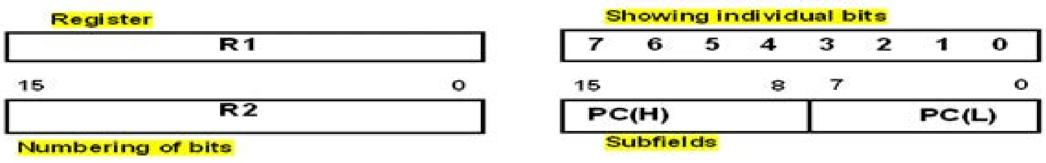
A register transfer language is a notation used to describe the micr-operation transfers between registers. It is a system for expressing in symbolic form the micro-operation sequences among register that are used to implement machine-language instructions. For any function of the computer, the register transfer language can be used to describe the (sequence of) micro-operations

- Register transfer language
- A symbolic language
- A convenient tool for describing the internal organization of digital computers
- Can also be used to facilitate the design process of digital systems.

Registers and Register Transfer

- Registers are designated by capital letters, sometimes followed by numbers (e.g., A, R13, IR)
- Registers are denoted by capital letters and are sometimes followed by numerals, e.g.,
    - MAR – Memory Address Register (holds addresses for the memory unit)
    - PC – Program Counter (holds the next instruction's address)
    - IR – Instruction Register (holds the instruction being executed)
    - R1 – Register 1 (a CPU register)
- We can indicate individual bits by placing them in parentheses, e.g., PC(8-15), R2(5), etc.
- Often the names indicate function:
    - Registers and their contents can be viewed and represented in various ways
    - A register can be viewed as a single entity
    - Registers may also be represented showing the bits of data they contain
- Designation of a register
    - a register
    - portion of a register
    - a bit of a register

Common ways of drawing the block diagram of a register



Register Transfer

- Copying the contents of one register to another is a register transfer
- A register transfer is indicated as

R2 ¬ R1

- ○ In this case the contents of register R1 are copied (loaded) into register R2
- ○ A simultaneous transfer of all bits from the source R1 to the destination register R2, during one clock pulse
- ○ Note that this is a non-destructive; i.e. the contents of R1 are not altered by copying (loading) them to R2
- ○ A register transfer such as

R3 ¬ R5

- ❖ Implies that the digital system has
- ❖ the data lines from the source register (R5) to the destination register (R3)
- ❖ Parallel load in the destination register (R3)
- ❖ Control lines to perform the action

Register Transfer Language Instructions

●Register Transfer

R2 ¬R1

●Simultaneous Transfer

R2 ¬R1, R1 ¬R2

●Conditional Transfer (Control Function)

P: R2 ¬R1

or

If (P = 1) Then R2 ¬R1

●Conditional, Simultaneous Transfer

T: R2 ¬R1, R1 ¬R2

Basic Symbols For Register Transfer

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	MAR, R2
Parentheses ( )	Denotes a part of a register	R2(0-7), R2(L)
Arrow ←	Denotes Transfer of information	R2 ←R1
Comma ,	Separates 2 micro-operations	R2 ←R1, R1 ←R1

Conclusion

The learner was introduced to the register transfer language. In particular, how specific notations (symbols) are used to specify digital systems, rather than in words. Learners were also introduced to how registers can be viewed and represented.

Assessment

1. Briefly explain what can be used to store one or more bits of data, also accept and/or transfer information serially?

Shift registers

Shift registers are group of flip-flops. each flip-flop in the register store one bit only i.e 1 or 0.

2. What addressing mode has its address part pointing to the address of actual data.

The addressing mode is direct addressing: In direct addressing operand is stored in memory and the memory address of the operand is specified in the instruction

3. Which addressing mode does not require the fetch operations?

Fetch operations are not required in immediate addressing. Because in immediate addressing the data is part of the instruction.

4. What addressing mode used an instruction of the form ADD X, Y? Absolute or direct addressing is used

5. Which is the register used as a working area in CPU?

An accumulator is register used in computer's central processing unit in which intermediate arithmetic and logic results are stored.

6. What addressing mode is used in the instruction PUSH B?

In register addressing mode the operand is held in memory. The address of the operand location is held in a register which is specified in instruction.

Activity 1.2 - Micro-architectures - Achievements Connections by Wires and Micro-programmed;

Introduction

This section introduces the learners to the micro-architecture of the computer. That is the resources and methods used to achieve specifications.

Activity Details

Micro-architecture is the term used to describe the resources and methods used to achieve architecture specification. The term typically includes the way in which these resources are organized as well as the design techniques used in the processor to reach the target cost and performance goals. The micro-architecture essentially forms a specification for the logical implementation.

Also called computer organization and sometimes abbreviated as µarch or uarch, is the way a given instruction set architecture (ISA) is implemented in a particular processor. A given ISA may be implemented with different micro-architectures; implementations may vary due to different goals of a given design or due to shifts in technology.

The micro-architecture is related to, but not the same as, the instruction set architecture. Micro-architectural elements may be everything from single logic gates, to registers, lookup tables, multiplexers, counters, etc., to complete ALUs, floating point units (FPU) and even larger elements.

A few important points:

- A single micro-architecture can be used to implement many different instruction sets, by means of changing the control store.
- Two machines may have the same micro-architecture, and so the same block diagram, but completely different hardware implementations. This manages both the level of the electronic circuitry and even more the physical level of manufacturing (of both ICs and/or discrete components).
- Machines with different micro-architectures may have the same instruction set architecture, and so both are capable of executing the same programs. New micro-architectures and/or circuitry solutions, along with advances in semiconductor manufacturing, are what allow newer generations of processors to achieve higher performance.

The pipelined datapath is the most commonly used datapath design in micro-architecture today. This technique is used in most modern microprocessors, microcontrollers, and DSPs. The pipelined architecture allows multiple instructions to overlap in execution, much like an assembly line.

The pipeline includes several different stages which are fundamental in micro-architecture designs. Some of these stages include instruction fetch, instruction decode, execute, and write back. Some architectures include other stages such as memory access. The design of pipelines is one of the central micro-architectural tasks.

Execution units are also essential to micro-architecture. Execution units include arithmetic logic units (ALU), floating point units (FPU), load/store units, branch prediction, and SIMD. These units perform the operations or calculations of the processor. The choice of the number of execution units, their latency and throughput is a central micro-architectural design task. The size, latency, throughput and connectivity of memories within the system are also micro-architectural decisions.

System-level design decisions such as whether or not to include peripherals, such as memory controllers, can be considered part of the micro-architectural design process. This includes decisions on the performance-level and connectivity of these peripherals.

Unlike architectural design, where achieving a specific performance level is the main goal, micro-architectural design pays closer attention to other constraints. Since micro-architecture design decisions directly affect what goes into a system, attention must be paid to such issues as:

- Chip area/cost.
- Power consumption.
- Logic complexity.
- Ease of connectivity.
- Manufacturability.
- Ease of debugging.
- Testability.

Conclusion

This section has highlighted the micro-architecture. It discussed the resources and methods used to achieve the architecture.

Assessment

1. Outline with an explanation the micro-architecture

Micro-architecture is used to describe the units that were controlled by the micro-program words. Micro-architecture is related to, but not the same as, the instruction set architecture. The instruction set architecture is near to the programming model of a processor as seen by an assembly language programmer or compiler writer, which includes the execution model, processor registers, address and data formats etc. The micro-architecture (or computer organization) is mainly a lower level structure and therefore manage a large number of details that are hidden in the programming model. It describes the inside parts of the processor and how they work together in order to implement the architectural specification.

Micro-architectural elements may be everything from single logic gates, to registers, lookup tables, multiplexers, counters, etc., to complete ALUs, FPUs and even larger elements. The electronic circuitry level can, in turn, be subdivided into transistor-level details, such as which basic gate-building structures are used and what logic implementation types (static/dynamic, number of phases, etc.) are chosen, in addition to the actual logic design used built them.

Activity 1.3 - Instructions plumbing and Instruction level parallelism (ILP)

Introduction

This section introduces the learners to the Instruction plumbing and Instruction level parallelism (ILP). Basically this all about how many of the operations in a computer program can be performed simultaneously

Activity Details

Instruction plumbing is a measure of how many of the operations in a computer program can be performed simultaneously. The potential overlap among instructions is called instruction level parallelism.

There are two approaches to instruction level parallelism:

- Hardware
- Software

Hardware level works upon dynamic parallelism whereas, the software level works on static parallelism

The Pentium processor works on the dynamic sequence of parallel execution but the Itanium processor works on the static level parallelism.

Example;

Consider the following program:

1.e = a + b

2.f = c + d

3.m = e * f

Operation 3 depends on the results of operations 1 and 2, so it cannot be calculated until both of them are completed. However, operations 1 and 2 do not depend on any other operation, so they can be calculated simultaneously. If we assume that each operation can be completed in one unit of time then these three instructions can be completed in a total of two units of time, giving an ILP of 3/2.

A goal of compiler and processor designers is to identify and take advantage of as much ILP as possible.

Ordinary programs are typically written under a sequential execution model where instructions execute one after the other and in the order specified by the programmer. ILP allows the compiler and the processor to overlap the execution of multiple instructions or even to change the order in which instructions are executed.

How much ILP exists in programs is very application specific. In certain fields, such as graphics and scientific computing the amount can be very large. However, workloads such as cryptography may exhibit much less parallelism.

Micro-architectural techniques that are used to exploit ILP include:

- Instruction pipelining where the execution of multiple instructions can be partially overlapped.
- Superscalar execution, VLIW, and the closely related explicitly parallel instruction computing concepts, in which multiple execution units are used to execute multiple instructions in parallel.
- Out-of-order execution where instructions execute in any order that does not violate data dependencies. Note that this technique is independent of both pipelining and superscalar. Current implementations of out-of-order execution dynamically (i.e., while the program is executing and without any help from the compiler) extract ILP from ordinary programs. An alternative is to extract this parallelism at compile time and somehow convey this information to the hardware. Due to the complexity of scaling the out-of-order execution technique, the industry has re-examined instruction sets which explicitly encode multiple independent operations per instruction.
- Register renaming which refers to a technique used to avoid unnecessary serialization of program operations imposed by the reuse of registers by those operations, used to enable out-of-order execution.
- Speculative execution which allow the execution of complete instructions or parts of instructions before being certain whether this execution should take place. A commonly used form of speculative execution is control flow speculation where instructions past a control flow instruction (e.g., a branch) are executed before the target of the control flow instruction is determined. Several other forms of speculative execution have been proposed and are in use including speculative execution driven by value prediction, memory dependence prediction and cache latency prediction.
- Branch prediction which is used to avoid stalling for control dependencies to be resolved. Branch prediction is used with speculative execution.

Dataflow architectures are another class of architectures where ILP is explicitly specified. In recent years, ILP techniques have been used to provide performance improvements in spite of the growing disparity between processor operating frequencies and memory access times (early ILP designs such as the IBM System/360 Model 91 used ILP techniques to overcome the limitations imposed by a relatively small register file). Presently, a cache miss penalty to main memory costs several hundreds of CPU cycles. While in principle it is possible to use ILP to tolerate even such memory latencies the associated resource and power dissipation costs are disproportionate. Moreover, the complexity and often the latency of the underlying hardware structures results in reduced operating frequency further reducing any benefits. Hence, the aforementioned techniques prove inadequate to keep the CPU from stalling for the off-chip data. Instead, the industry is heading towards exploiting higher levels of parallelism that can be exploited through techniques such as multiprocessing and multithreading.

Superscalar architectures

Is a CPU that implements a form of parallelism called instruction-level parallelism within a single processor or a computer designed to improve the performance of the execution of scalar instructions. A scalar is a variable that can hold only one atomic value at a time, e.g., an integer or a real. A scalar architecture processes one data item at a time - the computers we discussed up till now.

Examples of non-scalar variables:

- Arrays
- Matrices
- Records

In a superscalar architecture (SSA), several scalar instructions can be initiated simultaneously and executed independently. Pipelining allows also several instructions to be executed at the same time, but they have to be in different pipeline stages at a given moment. SSA includes all features of pipelining but, in addition, there can be several instructions executing simultaneously in the same pipeline stage. SSA introduces therefore a new level of parallelism, called instruction-level parallelism.

Conclusion

This section covered the Instruction plumbing and Instruction level parallelism (ILP), that is, how many of the operations in a computer program can be performed simultaneously.

Assessment

1. Outline give an example of an Instruction level parallelism (ILP)

is a measure of how many of the operations in a computer program can be performed simultaneously. The potential overlap among instructions is called instruction level parallelism.

basic idea is to execute several instructions in parallel. Parallelism exists in that we perform different operations (fetch, decode, ...) on several different instructions in parallel.

Mostly determined by the number of true (data) dependencies and procedural (control) dependencies in relation to the number of other instructions.

e.g.

• A: ADD R1 = R2 + R3

• B: SUB R4 = R1 – R5

ILP is traditionally "extracting parallelism from a single instruction stream working on a single stream of data".

Activity 1.4 Processor and System Performance

Introduction

This section introduces the learners to the processor and system performance. Material on the characteristics of the processor and system performance as well as the components that determine the system performance is provided .

Activity Details

The performance of a computer is dependent on how well it works together as a whole. Continually upgrading one part of the computer while leaving outdated parts installed will not improve performance much, if at all. The processor, memory and video card are the most important components when determining performance inside a computer.

The following are some of the most important parts of the computer regarding its speed and computing power;

1. Clock speed (Processor speed);

is often played up to be the major factor in a computer's overall performance. In rare cases this is true, but an average user rarely uses 100 percent of his Central Processing Unit's power (CPU). Things like encoding video or encrypting files, or anything that computes large, complex, numbers requires a lot of processor power. Most users spend most of their time typing, reading email or viewing web pages. During this time, the computer's CPU is probably hovering around 1 or 2 percent of it's total speed. Startup time is probably the only time the CPU is under stress, and even then it's often limited due to the hard drive speed.

2. System RAM speed and size;

The amount and speed of the RAM in your computer makes a huge difference in how your computer performs. If you are trying to run Windows XP with 64 MB of RAM it probably won't even work. When the computer uses up all available RAM it has to start using the hard drive to cache data, which is much slower. The constant transfer of data between RAM and virtual memory (hard drive memory) slows a computer down considerably. Especially when trying to load applications or files. The two types differ in the technology they use to hold data, dynamic RAM being the more common type. Dynamic RAM needs to be refreshed thousands of times per second.

Static RAM does not need to be refreshed, which makes it faster; but it is also more expensive than dynamic RAM. Both types of RAM are volatile, meaning that they lose their contents when the power is turned off.

3. Disk speed and size;

Is the biggest factor in your computer's performance is the hard disk speed. How fast the hard drive can find (average seek time), read, write, and transfer data will make a big difference in the way your computer performs. Most hard drives today spin at 7,200 RPMS, older models and laptops still spin at 5,200 RPMS, which is one reason laptops often appear sluggish to a desktop equivalent. The size of your hard drive plays a very little role in the performance of a computer. As long as you have enough free space for virtual memory and keep the disk defragmented it will perform well no matter what the size.

4. Video card - (onboard video RAM, chip type and speed);

Whenever your computer puts an image on the screen something has to render it. If a computer is doing this with software it is often slow and will affect the performance of the rest of the computer. Also, the image will not be rendered as crisp or as smoothly in the case of video. Even a low-end video card will significantly improve the performance of the computer by taking the large task of rendering the images on the screen from the CPU to the graphics card. If you work with large image files, video or play games you will want a higher end video card. Video cards use their own RAM called Video RAM. The more Video RAM a computer has the more textures and images the card can remember at a time. High end graphics cards for desktops now come with up to 64 megabytes of Video RAM, Laptops often only have 8 or 16 megabytes of Video RAM.

5. Others include memory and system buses

Latency memory, performance and efficiency

Is a time delay between the cause and the effect of some physical change in the system being observed. Latency is a result of the limited velocity with which any physical interaction can take place. This velocity is always lower or equal to speed of light. Therefore, every physical system that has spatial dimensions different from zero will experience some sort of latency. The precise definition of latency depends on the system being observed and the nature of stimulation. In communications, the lower limit of latency is determined by the medium being used for communications. In reliable two-way communication systems, latency limits the maximum rate that information can be transmitted, as there is often a limit on the amount of information that is "in-flight" at any one moment. In the field of human-machine interaction, perceptible latency (delay between what the user commands and when the computer provides the results) has a strong effect on user satisfaction and usability.

Computers run sets of instructions called a process. In operating systems, the execution of the process can be postponed if other processes are also executing. In addition, the operating system can schedule when to perform the action that the process is commanding.

·For example, suppose a process commands that a computer card's voltage output be set high-low-high-low and so on at a rate of 1000 Hz.. The operating system may choose to adjust the scheduling of each transition (high-low or low-high) based on an internal clock.

The latency is the delay between the process instruction commanding the transition and the hardware actually transitioning the voltage from high to low or low to high. System designers building real-time computing systems want to guarantee worst-case response. That is easier to do when the CPU has low interrupt latency and when it has deterministic response.

Caches

Cache memory, also called CPU memory, is random access memory (RAM) that a computer microprocessor can access more quickly than it can access regular RAM. This memory is typically integrated directly with the CPU chip or placed on a separate chip that has a separate bus interconnect with the CPU. The basic purpose of cache memory is to store program instructions that are frequently re-referenced by software during operation. Fast access to these instructions increases the overall speed of the software program. Most programs use very few resources once they have been opened and operated for a time, mainly because frequently re-referenced instructions tend to be cached. This explains why measurements of system performance in computers with slower processors but larger caches tend to be faster than measurements of system performance in computers with faster processors but more limited cache space.

Conclusion

This section covered the processor and system, the characteristics of the components that determine system performance.

Assessment

1. Discuss system performance

Computer performance is characterized by the amount of useful work accomplished by a computer system or computer network compared to the time and resources used. Depending on the context, high computer performance may involve one or more of the following: Short response time for a given piece of work.

- Short response time for a given piece of work
- High throughput (rate of processing work)
- Low utilization of computing resource(s)
- High availability of the computing system or application
- Fast (or highly compact) data compression and decompression
- High bandwidth
- Short data transmission time

Unit Summary

At the end of this unit, you will be conversant with the advanced functional organization of computer. This is by learning how data transfers occurs in the computer, the architecture of the microprocessor, the types of transfer of instructions within the computer and the processor and system performance of the computer.

Unit Assessment

The following section will test your understanding of this unit

Instructions

Answer the following questions

1. Differentiate between computer architecture and computer organization.
2. Explain the significance of layered architecture.
3. Explain the various types of performance metrics.

Grading Scheme

The marks will be awarded as shown below

question	sub-question	marks awarded
1	Any difference award 2 mark	8
2	any significance listed award 1.5 mark, maximum 4 significances	6
3	Explanation only 2 marks.	6
	Each significance listed award 1 mark(maximum 4)	
Total		20

Feedback

1. Difference between computer architecture and computer organization:

Computer architecture	Computer organization
It includes emphasis on logical design, computer design and the system design	It includes emphasis on the system components, circuit design, logical design, structure of instructions, computer arithmetic, processor control, assembly language, programming methods and of performance enhancement
It is concerned with structure and behaviour of computer as seen by the user	Computer organization is concerned with the way the hardware components operate and the way they are connected together to form the computer system

2. Explain the significance of layered architecture

Significance of layered architecture: In layered architecture, complex problems can be segmented into smaller and more manageable form. Each layer is specialized for specific functioning. Team development is possible because of logical segmentation. A team of programmers will build. The system and work has to be sub-divided of along clear boundaries.

3. Explain the various types of performance metrics.

Performance metrics include availability, response time, Channel capacity, latency, Completion time.

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources.

Unit 2. Multiprocessing

Unit Introduction

This unit introduces you to multiprocessing, which is the ability of a system to support more than one processor and/or the ability to allocate tasks between them. This will make you be in a position to understand how several programs can run concurrently in the same computer.

Unit Objectives

Upon completion of this unit the learner should be able to:

1. Describe multiprocessing
2. Explain how a processor can support concurrent running of programs
3. Describe how Amdahls law determines the maximum improvement of a system

Key Terms

Amdahls law: is a law or argument used to find the maximum expected improvement to an overall system when only part of the system is improved.

Multicore: is a type of architecture where a single physical processor contains the core logic of two or more processors or packaged into a single integrated circuit

Multiprocessor: term also refers to the ability of a system to support more than one processor and/or the ability to allocate tasks between them.

Learning Activities

Activity 2.1 - The Amdahl's law

Introduction

The section introduces the learners to Amdahl's law; this law introduces the concept of how to find the maximum expected improvement to an overall system when only part of the system is improved.

Activity Details

Amdahls law is also known as Amdahl's argument. It is used to find the maximum expected improvement to an overall system when only part of the system is improved. It is often used in parallel computing to predict the theoretical maximum speed up using multiple processors. The law is named after computer architect Gene Amdahl.

Amdahl's law: is law used to find the maximum expected improvement to an overall system when only part of the system is improved. It is often used in parallel computing to predict the theoretical maximum speed up using multiple processors.

The speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program. E.g. if a program needs 20 hours using a single processor core, and a particular portion of the program which takes one hour to execute cannot be parallelized, while the remaining 19 hours (95%) of execution time can be parallelized, then regardless of how many processors are devoted to a parallelized execution of this program, the minimum execution time cannot be less than that critical one hour. Hence, the theoretical speedup is limited to at most 20×.

Equation

A task which can be parallelized can be split up into two parts:

- A non-parallelizable part (or serial part) which cannot be parallelized;
- A parallelizable part which can be parallelized.

Example

A program that processes files from disk, A small part of that program may scan the directory and create a list of files internally in memory. After that, each file is passed to a separate thread for processing. The part that scans the directory and creates the file list cannot be parallelized, but processing the files can.

The time taken to execute the whole task in serial (not in parallel) is denoted T. The time T includes the time of both the non-parallelizable and parallelizable parts. The portion of time to execute in serial the parallelizable part is denoted P. The portion of time to execute in serial the non-parallizable part is then 1 − P.

From this follows that : $T=(1-P)T+PT$

It is the parallelizable part P that can be sped up by executing it in parallel. How much it can be sped up depends on how many subtasks are executed in parallel. The theoretical execution time of the parallelizable part on a system capable of executing N subtasks in parallel .

Amdahl's law gives the theoretical execution time T(N) of the whole task on a system capable of executing N subtasks in parallel:

$$\mathbf{T(N)=((1\text{-}P)+\frac{\square}{\square})T}$$

Consequently, the best (with an infinite number of subtasks) theoretical execution time of the whole task is

$$T_{min} = \lim(N\to\infty)\Box T(TN) = (1-P)T$$

In terms of theoretical overall speedup, Amdahl's law is given as

$$\mathbf{S\ (N)} = \frac{\Box}{\Box(\Box)} = \frac{\Box}{(\Box-\Box)+\frac{\Box}{\Box}}$$

and the best theoretical overall speedup is

$$\mathbf{S_{max} = \lim_{(N\to\infty)} S\ (N)} = \frac{\Box}{(\Box-\Box)}$$

As an example, if P is 90%, then 1 − P is 10%, and the task can be sped up by a maximum of a factor of 10, no matter how large the value of N used. For this reason, parallel computing is only useful for small numbers of processors and problems with very high values of P (close to 100%): so-called embarrassingly parallel problems. A great part of the craft of parallel programming consists of attempting to reduce the component 1 – P to the smallest possible value.

In parallel computing, P can be estimated by using the measured speedup S(N) on a specific number of processors N using

$$P_{estimated} = \frac{1}{\Box(\Box)} - 1\ /((1/N-1)\ )$$

P estimated in this way can then be used in Amdahl's law to predict speedup for a different number of processors.

Conclusion

This unit introduced the learner to the Amdahl's law. Examples were used to teach the learner how to find the maximum expected improvement to an overall system when only part of the system is improved.

Assessment

1. Briefly describe Amdahls law on parallel computing

In computer architecture, Amdahl's law (or Amdahl's argument) gives the theoretical speedup in latency of the execution of a task at fixed workload that can be expected of a system whose resources are improved. It is named after computer scientist Gene Amdahl, and was presented at the AFIPS Spring Joint Computer Conference in 1967.

Amdahl's law can be formulated the following way:

$$S_{\text{latency}}(s) = \frac{1}{1-p+\frac{p}{s}},$$

where

- Slatency is the theoretical speedup in latency of the execution of the whole task;
- s is the speedup in latency of the execution of the part of the task that benefits from the improvement of the resources of the system;
- p is the percentage of the execution time of the whole task concerning the part that benefits from the improvement of the resources of the system before the improvement.

Activity 2.2 - Short Vector Processing (Multimedia Operations)

Introduction

This section introduces the learner to the short vector processing (multimedia operations) which was initially developed for super-computing applications. Today its important for multimedia operations.

Activity Details

Vector processor

Is also called the array processor and is a central processing unit (CPU) that implements an instruction set containing instructions that operate on one-dimensional arrays of data called vectors, compared to scalar processors, whose instructions operate on single data items. Vector processors can greatly improve performance on certain workloads, notably numerical simulation and similar tasks.

They are commonly called supercomputers, the vector processors are machines built primarily to handle large scientific and engineering calculations. Their performance derives from a heavily pipelined architecture which operations on vectors and matrices can efficiently exploit.

Vector processors have high- level operations that work on linear arrays of numbers: "vectors".

Properties of Vector Processors

the vector processors have the following properties;

1. Single vector instruction implies lots of work (loop), i.e. it has fewer instruction fetches
2. Each result independent of previous result.
    a. Multiple operations can be executed in parallel
    b. Simpler design, high clock rate
    c. Compiler (programmer) ensures no dependencies
3. Reduces branches and branch problems in pipelines
4. Vector instructions access memory with known pattern

   a. Effective prefetching
   b. Amortize memory latency of over large number of elements
   c. Can exploit a high bandwidth memory system
   d. No (data) caches required!

Styles of Vector Architectures

They are of two styles

1. Memory- memory vector processors
   a. All vector operations are memory to memory
2. Vector-register processors
   a. All vector operations between vector registers (except vector load and store)
   b. Vector equivalent of load-store architectures
   c. Includes all vector machines since late 1980s
   d. We assume vector-register for rest of the lecture

Components of a Vector Processor

- Scalar CPU: registers, data paths, instruction fetch logic
- Vector register
  - Fixed length memory bank holding a single vector
  - Typically 8-32 vector registers, each holding 1 to 8 Kbits
  - Has at least 2 read and 1 write ports
  - MM: Can be viewed as array of 64b, 32b, 16b, or 8b elements
- Vector functional units (FUs)
  - Fully pipelined, start new operation every clock
  - Typically 2 to 8 FUs: integer and FP
  - Multiple data paths (pipelines) used for each unit to process multiple elements per cycle
- Vector load-store units (LSUs)
  - Fully pipelined unit to load or store a vector
  - Multiple elements fetched/stored per cycle
  - May have multiple LSUs

Cross-bar to connect FUs , LSUs, registers

Basic Vector Instructions

This are in Table 1 below;

Instr.	Operands	Operation	Comment
VADD.VV	V1,V2,V3	V1=V2+V3	vector + vector
VADD.SV	V1,R0,V2	V1=R0+V2	scalar + vector
VMUL.VV	V1,V2,V3	V1=V2xV3	vector x vector
VMUL.SV	V1,R0,V2	V1=R0xV2	scalar x vector
VLD	V1,R1	V1=M [R1..R1+63]	load, stride=1
VLDS	V1,R1,R2	V1=M [R1..R1+63*R2]	load, stride=R2
VLDX	V1,R1,V2	V1=M[R1+V2i,i=0..63]	indexed("gather")
VST	V1,R1	M [R1..R1+63]=V1	store, stride=1
VSTS	V1,R1,R2	V1=M [R1..R1+63*R2]	store, stride=R2
VSTX	V1,R1,V2	V1=M[R1+V2i,i=0..63]	indexed("scatter")

+ all the regular scalar instructions (RISC style)…

Table 1

Vector Memory Operations

- Load/store operations move groups of data between registers and memory
- Three types of addressing
- ○ Unit stride
- Fastest
- ○ Non-unit (constant) stride
- ○ Indexed (gather-scatter)
- Vector equivalent of register indirect
- Good for sparse arrays of data

- Increases number of programs that vectorize
- compress/expand variant also
- Support for various combinations of data widths in memory
    - {.L,.W,.H.,.B} x {64b, 32b, 16b, 8b}

Vector Code Example

Y[0:63] = Y[0:653] + a*X[0:63]

See the table 2 & 3 below;

Table 2

	64 element SAXPY: scalar		64 element SAXPY: vector	
	LD	R0,a	LD R0,a	#load scalar a
	ADDI	R4,Rx,#512	VLD V1,Rx	#load vector X
loop:	LD	R2, 0(Rx)	VMUL.SV V2,R0,V1	#vector mult
	MULTD	R2,R0,R2	VLD V3,Ry	#load vector Y
	LD	R4, 0(Ry)	VADD.VV V4,V2,V3	#vector add
	ADDD	R4,R2,R4	VST Ry,V4	#store vector Y

Table 3

SD	R4, 0(Ry)
ADDI	Rx,Rx,#8
ADDI	Ry,Ry,#8
SUB	R20,R4,Rx
BNZ	R20,loop

Vector Length

vector register can hold some maximum number of elements for each data width (maximum vector length or MVL). What to do when the application vector length is not exactly MVL? Vector-length (VL) register controls the length of any vector operation, including a vector load or store E.g. vadd.vv with VL=10 is for (I=0; I<10; I++) V1[I]=V2[I]+V3[I]

The VL can be anything from 0 to MVL

Conclusion

The section introduced the learner to the short vector processing (multimedia operations) which is used for the operations on high level linear array of numbers.

Assessment

1. State the properties of the vector processors

- Each result independent of previous result

  => Long pipeline, compiler ensures no dependencies

  => High clock rate

- Vector instructions access memory with known pattern

  => Highly interleaved memory

  => Amortize memory latency of over≈64 elements

  => No (data) caches required! (Do use instruction cache)

- Reduces branches and branch problems in pipelines
- Single vector instruction implies lots of work (≈loop)

  => Fewer instruction fetches

Activity 2.3 - The Multicore and Multiprocessor Segments;

Introduction

This section introduces the learner to the multicore and multiprocessor. It also highlights why computer architecture is moving towards multiprocessor architecture

Activity details

Central Processing Unit is what is typically referred to as a processor. A processor contains many discrete parts within it, such as one or more memory caches for instructions and data, instruction decoders, and various types of execution units for performing arithmetic or logical operations.

Multicore: is a type of architecture where a single physical processor contains the core logic of two or more processors.

A multicore CPU has multiple execution cores on one CPU. This can mean different things depending on the exact architecture, but it basically means that a certain subset of the CPU's components is duplicated, so that multiple "cores" can work in parallel on separate operations. This is called CMP, Chip-level Multiprocessing.

A multiprocessor system contains more than one such CPU, allowing them to work in parallel. This is called SMP, or Simultaneous Multiprocessing. That is Multi-processing simply means putting multiple processors in one system.

For example, a multicore processor may have a separate L1 cache and execution unit for each core, while it has a shared L2 cache for the entire processor. That means that while the processor has one big pool of slower cache, it has separate fast memory and arithmetic/logic units for each of several cores. This would allow each core to perform operations at the same time as the others.

Single-core CPU Chip

Figure 1below illustrates the single-core CPU chip. it has the components register files, ALU, bus interface and the system bus. The figure also shows the single core clearly marked out

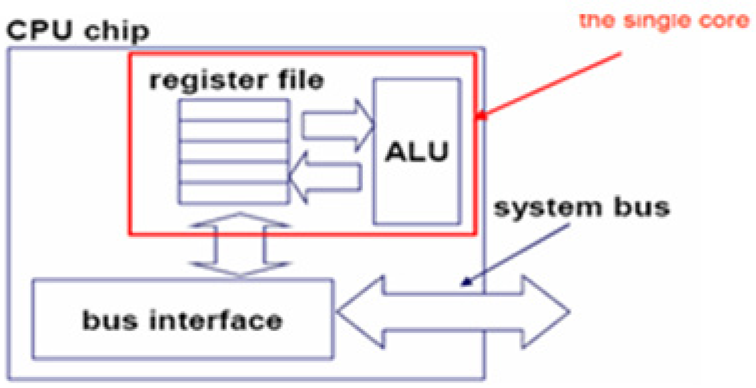


Figure 1

Multi-core architectures

The cores fit on a single processor socket, also called CMP (Chip Multi-Processor)

Multicore System:

A Multicore system usually refers to a multiprocessor system that has all its processors on the same chip. It could also refer to a system where the processors are on different chips but use the same package (i.e., a multichip module). Multicore systems were developed primarily to enhance the system performance while limiting its power consumption. It consists of

1. General - purpose programmable cores,
2. Special - purpose accelerator cores,

3. Shared memory modules,
4. NoC (interconnection network), and
5. I/O interface.

Figure 2 below illustrates the Multi-core CPU chip

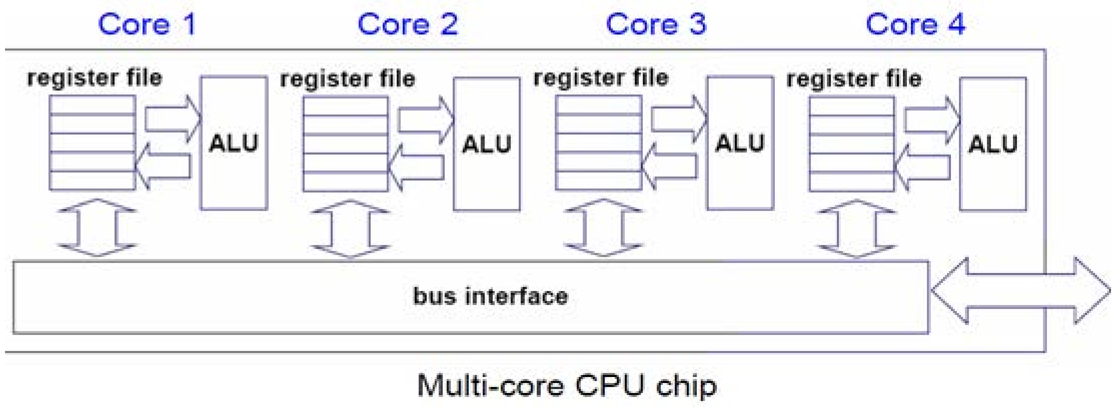


Figure 2

In Figure 2 all processors are on the same chip. The Multi-core processors are MIMD, i.e. different cores execute different threads (Multiple Instructions), operating on different parts of memory (Multiple Data). Also Multi-core is a shared memory multiprocessor, i.e. all cores share the same memory

NB

The main reason for which computer architecture is moving towards multicore systems is scalability. That is, as we increase the number of processors to enhance performance, multicore systems allow limiting power consumption and interprocessor communication overhead. A Multicore system can be scaled by adding more CPU cores and adjusting the interconnection network. More system programming work has to be done to be able to utilize the increased resources. It is one thing to increase the number of CPU resources. It is another to be able to schedule all of them to do useful tasks.

Multiprocessor

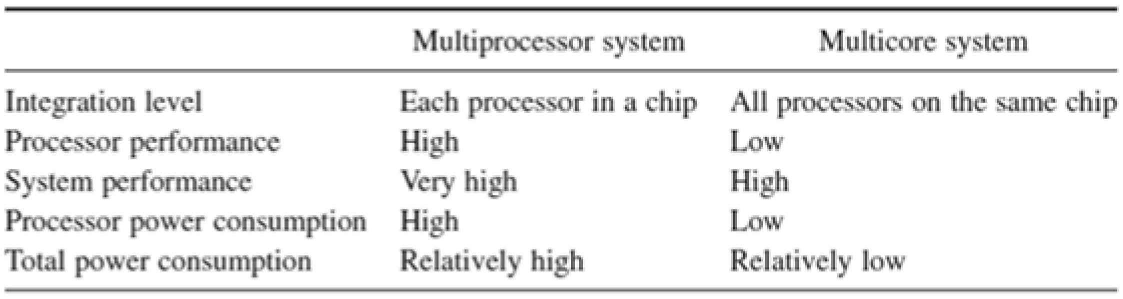
Multiprocessor. is the use of two or more central processing units (CPUs) within a single computer system.

Multiprocessor System

A multiprocessor system has its processors residing in separate chips and processors are interconnected by a backplane bus. Multiprocessor systems were developed to enhance the system performance with little regard to power consumption. A multiprocessor system has good performance and its constituent processors are high - performing processors.

Main Differences between Multicore Systems and Multiprocessor Systems:

The differences are as expressed in Table 4



	Multiprocessor system	Multicore system
Integration level	Each processor in a chip	All processors on the same chip
Processor performance	High	Low
System performance	Very high	High
Processor power consumption	High	Low
Total power consumption	Relatively high	Relatively low

Table 4

Conclusion

This section introduced the main reason why computer architecture is moving towards multicore systems which is scalability. It enables an increase in the number of processors. This has enhanced performance, multicore systems allow limiting power consumption and interprocessor communication overhead

Assessment

Distinguish the multicore and multiprocessor architectures

A CPU, or Central Processing Unit, is what is typically referred to as a processor. A processor contains many discrete parts within it, such as one or more memory caches for instructions and data, instruction decoders, and various types of execution units for performing arithmetic or logical operations.

A multiprocessor system contains more than one such CPU, allowing them to work in parallel. This is called SMP, or Simultaneous Multiprocessing.

A multicore CPU has multiple execution cores on one CPU. Now, this can mean different things depending on the exact architecture, but it basically means that a certain subset of the CPU's components is duplicated, so that multiple "cores" can work in parallel on separate operations. This is called CMP, Chip-level Multiprocessing.

Activity 2.4 - Flynn Taxonomy : Structures and Multiprocessor Architectures

Introduction

This section introduces the learner to the designs of modern processors and their functionalities. Flynn Taxonomy is also discussed in the section.

Activity Details

Flynn's taxonomy is a classification of computer architectures. i.e. is a classification of computer architectures. It has been used as a tool in design of modern processors and their functionalities. Flynn classification definitions are based upon the number of concurrent instruction (or control) streams and data streams available in the architecture. They are;-

1. SISD (Single instruction stream, single data stream)

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) to operate on single data stream (DS) i.e. one operation at a time.

Examples

The traditional uniprocessor machines like a PC (currently manufactured PCs have multiple cores) or old mainframes.

2. Single instruction stream, 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 graphics processing unit (GPU)

3. Multiple instruction streams, single data stream (MISD)

Multiple instructions operate on one 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.

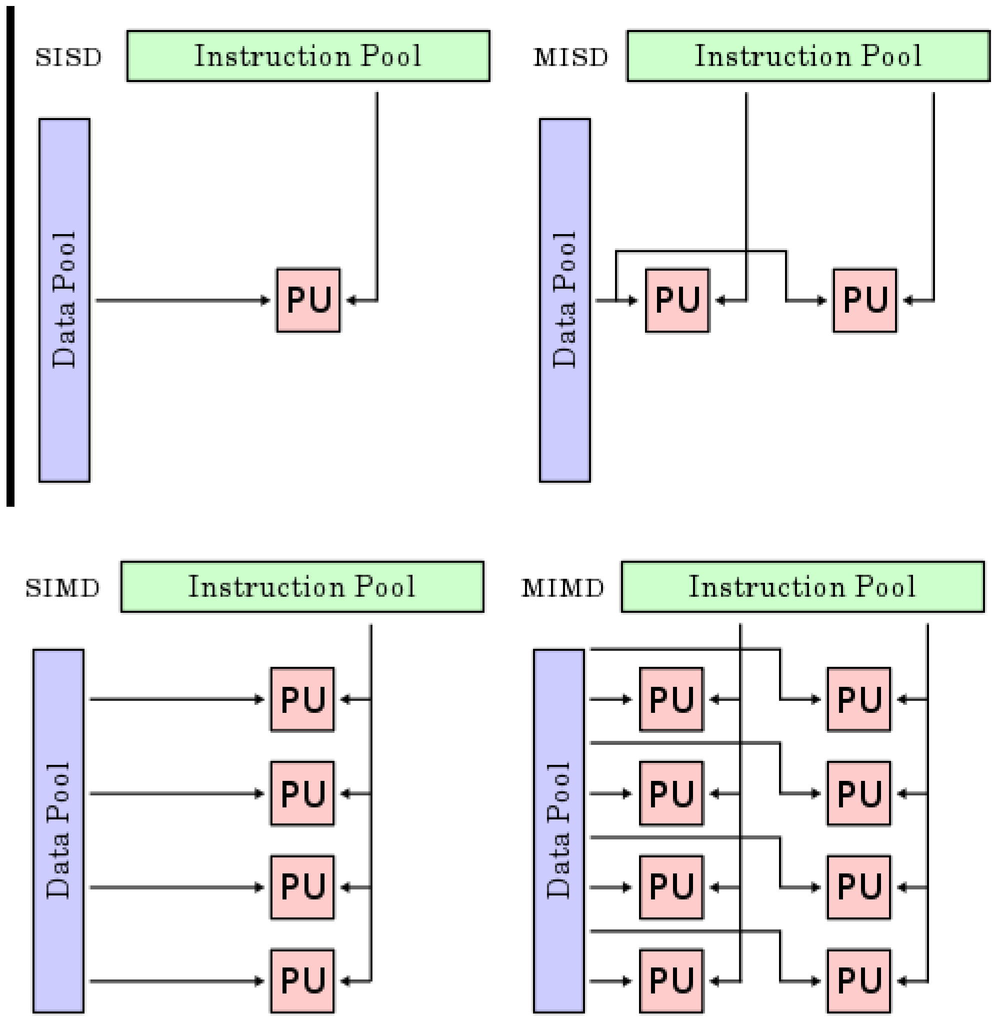
Examples include the Space Shuttle flight control computer.

4. Multiple instruction streams, multiple data streams (MIMD)

Multiple autonomous processors simultaneously executing different instructions on different data. MIMD architectures include multi-core superscalar processors, and distributed systems, using either one shared memory space or a distributed memory space.

Diagram comparing classifications

These four architectures are shown below visually. Each processing unit (PU) is shown for a unicore or multi-core computer:



Note

As of 2006, all the top 10 and most of the TOP500 supercomputers are based on a MIMD architecture.

Further divide the MIMD category into the two categories below, and even further subdivisions are sometimes considered.

5. Single program, multiple data streams (SPMD)

Multiple autonomous processors simultaneously executing the same program (but at independent points, rather than in the lockstep that SIMD imposes) on different data. Also termed single process, multiple data the use of this terminology for SPMD is technically incorrect, as SPMD is a parallel execution model and assumes multiple cooperating processes executing a program. SPMD is the most common style of parallel programming. The SPMD model and the term was proposed by Frederica Darema.

Gregory F. Pfister was a manager of the RP3 project, and Darema was part of the RP3 team.

6. Multiple programs, multiple data streams (MPMD)

Multiple autonomous processors simultaneously operating at least 2 independent programs. Typically such systems pick one node to be the "host" ("the explicit host/node programming model") or "manager" (the "Manager/Worker" strategy), which runs one program that farms out data to all the other nodes which all run a second program. Those other nodes then return their results directly to the manager. An example of this would be the Sony PlayStation 3 game console, with its SPU/PPU processor.

Conclusion

This sections introduced the learner the classification of computer architecture and has been used as a tool in design of modern processors and their functionalities

Assessment

Discuss the Multiple instruction streams data stream (MISD) architecture

This implies that several instructions are operating on a single piece of data. The same data flows through a linear array of processors executing different instruction streams. This architecture is also known as systolic array for pipelined execution of specific algorithms.

- Not much used in practice.

Activity 2.5 - Scheduling multiprocessor systems

Introduction

This section introduces the learner to the multiprocessor scheduling. In this the workload called tasks can be spread across processors and thus be executed much faster.

Activity Details

In computer science, multiprocessor scheduling is an NP-hard optimization problem. The problem statement is: "Given a set J of jobs where job ji has length li and a number of processors m, what is the minimum possible time required to schedule all jobs in J on m processors such that none overlap?"

The applications of this problem are numerous, but are, as suggested by the name of the problem, most strongly associated with the scheduling of computational tasks in a multiprocessor environment.

Multiprocessor schedulers have to schedule tasks which may or may not be dependent upon one another. For example take the case of reading user credentials from console, then use it to authenticate, then if authentication is successful display some data on the console. Clearly one task is dependent upon another. This is a clear case of where some kind of ordering exists between the tasks. In fact it is clear that it can be modelled with partial ordering. Then, by definition, the set of tasks constitute a lattice structure.

The general multiprocessor scheduling problem is a generalization of the optimization version of the number partitioning problem, which considers the case of partitioning a set of numbers (jobs) into two equal sets (processors).

Processors purpose-specific graphics and GPU

General-purpose computing on graphics processing units (GPGPU, rarely GPGP or GPU) is the use of a graphics processing unit (GPU), which typically handles computation only for computer graphics, to perform computation in applications traditionally handled by the central processing unit (CPU). The use of multiple graphics cards in one computer, or large numbers of graphics chips, further parallelizes the already parallel nature of graphics processing. In addition, even a single GPU-CPU framework provides advantages that multiple CPUs on their own do not offer due to the specialization in each chip.

GPGPU pipeline is a kind of parallel processing between one or more GPUs and CPUs that analyzes data as if it were in image or other graphic form. While GPUs generally operate at lower frequencies, they usually have many times more cores to make up for it (up to hundreds at least) and can, thus, operate on pictures and graphical data effectively much faster, dozens or even hundreds of times faster than a traditional CPU, migrating data into graphical form and then using the GPU to "look" at it and analyze it can result in profound speedup. GPGPU pipeline is a kind of parallel processing between one or more GPUs and CPUs that analyzes data as if it were in image or other graphic form. While GPUs generally operate at lower frequencies, they usually have many times more cores to make up for it (up to hundreds at least) and can, thus, operate on pictures and graphical data effectively much faster, dozens or even hundreds of times faster than a traditional CPU, migrating data into graphical form and then using the GPU to "look" at it and analyze it can result in profound speedup.

Reconfigurable Logic and Purpose-specific Processors

Reconfigurable computing is a computer architecture combining some of the flexibility of software with the high performance of hardware by processing with very flexible high speed computing fabrics like field-programmable gate arrays (FPGAs). The principal difference when compared to using ordinary microprocessors is the ability to make substantial changes to the data path itself in addition to the control flow. On the other hand, the main difference with custom hardware, i.e. application-specific integrated circuits (ASICs) is the possibility to adapt the hardware during runtime by "loading" a new circuit on the reconfigurable fabric.

The concept of reconfigurable computing has existed since the 1960s, when Gerald Estrin's paper proposed the concept of a computer made of a standard processor and an array of "reconfigurable" hardware. The main processor would control the behavior of the reconfigurable hardware. The latter would then be tailored to perform a specific task, such as image processing or pattern matching, as quickly as a dedicated piece of hardware. Once the task was done, the hardware could be adjusted to do some other task. This resulted in a hybrid computer structure combining the flexibility of software with the speed of hardware. Reconfigurable architectures can bring unique capabilities to computational tasks. They offer the performance and energy efficiency of hardware with the flexibility of software.

<u>Conclusion</u>

This section introduced the learner to the multiprocessor scheduling applied in the computer architecture by partitioning the jobs to be performed.

Assessment

1. Briefly describe multiprocessor scheduling

Multiprocessor scheduling is an NP-hard optimization problem. The problem statement is: "Given a set J of jobs where job ji has length li and a number of processors m, what is the minimum possible time required to schedule all jobs in J on m processors such that none overlap?" The applications of this problem are numerous, but are, as suggested by the name of the problem, most strongly associated with the scheduling of computational tasks in a multiprocessor environment.

Multiprocessor schedulers have to schedule tasks which may or may not be dependent upon one another. For example take the case of reading user credentials from console, then use it to authenticate, then if authentication is successful display some data on the console. Clearly one task is dependent upon another. This is a clear case of where some kind of ordering exists between the tasks. In fact it is clear that it can be modelled with partial ordering. Then, by definition, the set of tasks constitute a lattice structure.

The general multiprocessor scheduling problem is a generalization of the optimization version of the number partitioning problem, which considers the case of partitioning a set of numbers (jobs) into two equal sets (processors)

Unit Summary

At the end of this unit, the learners will be describe Amadahl's law, Flynn Taxonom, multiprocessing and Scheduling. Short vector processing and multicore and multiprocessor is also covered. This is by learning how several processors can be integrated into one system to solve and allocate given tasks among themselves.

Unit Assessment

The following section will test the learners understanding of this unit

Instructions

Answer the following questions

1. State Amdahls law and what is used for?
2. Explain two properties of vector processors
3. What is multicore processor?

Grading Scheme

The marks will be awarded as shown below

Question	Sub-question	marks Awarded
1	Stating and explaining award a mark each, maximum 6	6
2	Any two and their explanations 2mark, maximum 4 significances	4
3	Stating only 2 marks.	5
	Each subsequent explanation listed award 1 mark(maximum 3)	
Total		15

Feedback

1. Also known as Amdahl's argument. It is used to find the maximum expected improvement to an overall system when only part of the system is improved, is law used to find the maximum expected improvement to an overall system when only part of the system is improved. It is often used in parallel computing to predict the theoretical maximum speedup using multiple processors.

2. Single vector instruction implies lots of work (loop)

-Fewer instruction fetches

- Each result independent of previous result

– Multiple operations can be executed in parallel

– Simpler design, high clock rate

– Compiler (programmer) ensures no dependencies

- Reduces branches and branch problems in pipelines

- Vector instructions access memory with known pattern

– Effective prefetching

– Amortize memory latency of over large number of elements

– Can exploit a high bandwidth memory system

– No (data) caches required!

3. It is a type of architecture where a single physical processor contains the core logic of two or more processors. A multicore CPU has multiple execution cores on one CPU.

This can mean different things depending on the exact architecture, but it basically means that a certain subset of the CPU’s components is duplicated, so that multiple “cores” can work in parallel on separate operations.

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources.

Unit 3. Computer Organization and low-level Programming

Unit Introduction

This unit introduces the learner to the organization and low-level programming that provides little or no abstraction which is basically about the machine and assembly language programming.

Unit Objectives

At the end of the unit the learner should be able to;-

1. Describe low-level programming
2. Distinguish between assembly and machine language
3. Understand the fundamental concepts of machine organization and the requirements of low-level language programming.

Key Terms

Compiler: Computer program (or a set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language), with the latter often having a binary form known as object code.

Machine Language: Set of instructions executed directly by a computer's central processing unit (CPU).

Assembly Language: Low-level programming language for a computer, or other programmable device, in which there is a very strong (generally one-to-one) correspondence between the language and the architecture's machine code instructions.

Assembler: Computer program which translates assembly language to an object file or machine language format.

Learning Activities

Activity 3.1 - Structure of low-level programs

Introduction

This section introduces the learner to the low-level programming languages. The learners should distinguish the low-level languages and their different uses in computer programming

Activity Details

In computer science, a low-level programming language is a programming language that provides little or no abstraction from a computer's instruction set architecture—commands or functions in the language map closely to processor instructions. This refers to either machine code or assembly language. The word "low" refers to the small or nonexistent amount of abstraction between the language and machine language; because of this, low-level languages are sometimes described as being "close to the hardware." Because of the close relationship between the language and the hardware architecture programs written in low-level languages tend to be relatively non-portable.

Low-level languages can convert to machine code without a compiler or interpreter— second-generation programming languages use a simpler processor called an assembler— and the resulting code runs directly on the processor. A program written in a low-level language can be made to run very quickly, with a small memory footprint. An equivalent program in a high-level language can be less efficient and use more memory. Low-level languages are simple, but considered difficult to use, due to numerous technical details that the programmer must remember. By comparison, a high-level programming language isolates execution semantics of computer architecture from the specification of the program, which simplifies development.

Machine codes

Machine code is the only language a computer can process directly without a previous transformation. Currently, programmers almost never write programs directly in machine code, because it requires attention to numerous details that a high-level language handles automatically, requires memorizing or looking up numerical codes for every instruction, and is extremely difficult to modify.

True machine code is a stream of raw, usually binary, data. A programmer coding in "machine code" normally codes instructions and data in a more readable form such as decimal, octal, or hexadecimal which is translated to internal format by a program called a loader or toggled into the computer's memory from a front panel.

Assembly language

Second-generation languages provide one abstraction level on top of the machine code. In the early days of coding on computers like the TX-0 and PDP-1, the first thing MIT hackers did was write assemblers. Assembly language has little semantics or formal specification, being only a mapping of human-readable symbols, including symbolic addresses, to opcodes, addresses, numeric constants, strings and so on.

Typically, one machine instruction is represented as one line of assembly code. Assemblers produce object files that can link with other object files or be loaded on their own. Most assemblers provide macros to generate common sequences of instructions.

Limitations of low-level architectures

- Very hard to read or learn for the uninitiated.
- Not very self-documenting like higher level languages.
- Harder to modify and maintain.
- Less support, than high level languages, in development and debug environments.

Conclusion

The learner should distinguish the various levels of low-level programming used in programming. They can also state the limitations arising from comparing the two low-level of programming languages

Assessment

Distinguish between machine and assembly languages

Machine language is the actual bits used to control the processor in the computer, usually viewed as a sequence of hexadecimal numbers (typically bytes). The processor reads these bits in from program memory, and the bits represent "instructions" as to what to do next.

Thus machine language provides a way of entering instructions into a computer (whether through switches, punched tape, or a binary file).

While assembly language is a more human readable view of machine language. Instead of representing the machine language as numbers, the instructions and registers are given names (typically abbreviated words, or mnemonics, like ld means "load"). Unlike a high level language, assembler is very close to the machine language. The main abstractions (apart from the mnemonics) are the use of labels instead of fixed memory addresses, and comments.

An assembly language program (ie a text file) is translated to machine language by an assembler. A disassembler performs the reverse function (although the comments and the names of labels will have been discarded in the assembler process).

Machine language faster than assembly language even than assembly language depend upon machine language

Activity 3.2 - Architecture support from Low level to High level languages

Introduction

This section introduces learners to the various supports offered by programming languages starting with the low-level to the high level programming languages.

Activity Details

Architecture support from low level to high level languages is in the following ways;

Abstraction in software design;

1. Assemble-level abstraction

A programmer who writes directly with the raw machine instruction set. Expressing the program in terms of instructions, addresses, registers, bytes and words

2. High-level languages

Allows the programmer to think in terms of abstractions that are above the machine level. The programmer may not even know on which machine the program will ultimately run. The RISC philosophy focusing instruction set design on flexible primitive operations from which the complier can build its high level operations

3. Data types

a) ARM support for characters

For handling characters is the unsigned byte load and store instruction

b) ANSI (American National Standards Institute) C basic data types

Defines the following basic data types –

- Signed and unsigned characters of at least eight bits – Signed and unsigned short integers of at least 16 bits – Signed and unsigned integers of at least 16 bits – Signed and unsigned long integers of at least 32 bits – Floating-point, double and long double floating-point numbers – Enumerated types – Bit fields (sets of Boolean variables).
- The ARM C compiler adopts the minimum sizes for each of these types
- The standard integer uses 32-bit values

c) ANCI C derived data types

- Defines derived data types – Arrays, Functions, Structures, Pointers, Unions
- ARM pointers are 32 bits long and resemble unsigned integers

- The ARM C compiler aligns characters on byte boundaries, short integers at even addresses and all other types on word boundaries

d) ARM architectural support for C data types

- Provides native support for signed and unsigned 32-bit integers and for unsigned bytes, covering the C integer, long integer and unsigned character types
- For arrays and structures: base plus scaled index addressing
- Current versions of the ARM include signed byte and signed and unsigned 16-bit loads and stores, providing some native support for short integer and signed character types

4. Expressions

a) Register use

- The key to the efficient evaluation of a complex expression is to get the required values into the registers in the right order and to ensure that frequently used values are normally resident in registers
- Optimizing this trade-off between the number of values that can be held in registers and the number of registers remaining is a major task for the complier

b) ARM support

- The 3-address instruction format used by the ARM gives the compiler the maximum flexibility
- Thumb instructions (generally 2-address) – restricts the compiler's freedom to some extent – smaller number of general registers also makes its job harder

c) Accessing operands

●A procedure will normally work with operands that are presented in one of the following ways, and can be accessed as indicated as an argument passed through a register – The value is already in a register, so no further work is necessary as a argument passed on the stack – Stack pointer (r13) relative addressing with an immediate offset known at compile-time allows the operand to be collected with a single LDR

●As a constant in the procedure's literal pool – PC-relative addressing, again with an immediate offset known at compile-time, gives access with a single LDR

●As a local variable – Local variables are allocated space on the stack and are accessed by

- ○ a stack pointer relative LDR
- ○ As a global variable – Global (and static) variables are allocated space in the static area and are accessed by static base (is usually in r9) relative addressing

d) Pointer arithmetic

- Arithmetic on pointers depends on the size of the data types that the pointers are pointing to
- If a variable is used as an offset it must be scaled at run-time
- If p is held in r0 and i in r1, the change top may be compiled as: ADD r0, r0, r1, LSL #2 ; scale r1 to int

e) Arrays

- The declaration: int a[10]; – a reference to a[i] is equivalent to the pointer-plus-offset form *(a+i)

Conclusion

The section lists several supports that can be got by using the various programming level operations. This can include abstraction, pointer arithmetic, arrays etc.

Assessment

1. Explain the following supports as obtainable in application of the various programming levels

- Abstraction; is a technique for managing complexity of computer systems. It works by establishing a level of complexity on which a person interacts with the system, suppressing the more complex details below the current level.
- Pointer arithmetic: is another way to traverse through an array.

2. Describe Low-level programming language; is a programming language that provides little or no abstraction from a computer's instruction set architecture—commands or functions in the language map closely to processor instructions. it refers to refers to either machine code or assembly language. The word "low" refers to the small or nonexistent amount of abstraction between the language and machine language; because of this, low-level languages are sometimes described as being "close to the hardware." Because of the close relationship between the language and the hardware architecture programs written in low-level languages tend to be relatively non-portable. Low-level languages can convert to machine code without a compiler or interpreter. A program written in a low-level language can be made to run very quickly, with a small memory footprint.

Unit Summary

At the end of this unit, the learners will be conversant with Low-level programming, which is

about machine and Assembly programming. It also looked at the language support offered.

Unit Assessment

The following section will test the learners understanding of this unit which is on low and high-level programming and its architecture

Instructions

Answer the following questions

1. Differentiate between low-level and high-level programming
2. Give 3 limitations of low-level architecture
3. Explain machine code

Grading Scheme

The marks will be awarded as shown below

question	sub-question	marks awarded
1	Any difference award 2 mark maximum 4	8
2	any limitation listed award 2 mark, maximum 4	8
3	explanation (maximum marks 4)	4
Total		20

<u>Feedback</u>

1. A low-level programming language is a programming language that provides little or no abstraction from a computer's instruction set architecture—commands or functions in the language map closely to processor instructions. it refers to either machine code or assembly language. while high-level programming languages are those closer to human languages and further from machine languages.

2. Very hard to read or learn for the uninitiated.

- Not very self-documenting like higher level languages.
- Harder to modify and maintain.
- Less support, than high level languages, in development and debug environments.

3.Is a set of instructions executed directly by a computer's central processing unit (CPU). Each instruction performs a very specific task, such as a load, a jump, or an Arithmetic logic unit (ALU) operation on a unit of data in a CPU register or memory.

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources.

Unit 4. Strategies and Interface I/O

Unit Introduction

This section introduces the learners to the strategies of I/O interfaces. They include;- polled, interrupt driven and DMA.

Unit Objectives

At the end of this unit, the learners will

1. Explain the Strategies of Interface I/O
2. Distinguish between handshaking and buffering
3. Understand the programmed IO mode of data transfer
4. Describe a DMA transfer.

Key Terms

Polling: refers to actively sampling the status of an external device by a client program as a synchronous activity. Polling is most often used in terms of input/output (I/O), and is also referred to as polled I/O or software-driven I/O.

Interrupt: a hardware signal that breaks the flow of program execution and transfers control to a predetermined storage location so that another procedure can be followed or a new operation carried out.

Direct memory access (DMA): is a method that allows an input/output (I/O) device to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations. The process is managed by a chip known as a DMA controller (DMAC).

Input / Output: the process of input or output, encompassing the devices, techniques, media, and data used

Learning Activities

Activity 4.1 - Fundamentals I/O: handshake and buffering

Introduction

This section introduces the learner to the various strategies used in I/O interfaces and other operations possible on an interface

Activity details

The computer is useless without some kind of interface to to the outside world. There are many different devices which we can connect to the computer system; keyboards, VDUs and disk drives are some of the more familiar ones. Irrespective of the details of how such devices are connected we can say that all I/O is governed by three basic strategies.

- Programmed I/O
- Interrupt driven I/O
- Direct Memory Access (DMA)

In programmed I/O all data transfers between the computer system and external devices are completely controlled by the computer program. Part of the program will check to see if any external devices require attention and act accordingly. This process is known as polling. Programmed I/O is probably the most common I/O technique because it is very cheap and easy to implement, and in general does not introduce any unforeseen hazards.

Programmed I/O

Is a method of transferring data between the CPU and a peripheral, such as a network adapter or an ATA storage device. In general, programmed I/O happens when software running on the CPU uses instructions that access I/O address space to perform data transfers to or from an I/O device.

The PIO interface is grouped into different modes that correspond to different transfer rates. The electrical signaling among the different modes is similar — only the cycle time between transactions is reduced in order to achieve a higher transfer rate

The PIO modes require a great deal of CPU overhead to configure a data transaction and transfer the data. Because of this inefficiency, the DMA (and eventually UDMA) interface was created to increase performance. The simple digital logic required to implement a PIO transfer still makes this transfer method useful today, especially if high transfer rates are not required like in embedded systems, or with FPGA chips where PIO mode can be used without significant performance loss.

Interrupt driven I/O

Is a way of controlling input/output activity in which a peripheral or terminal that needs to make or receive a data transfer sends a signal that causes a program interrupt to be set. At a time appropriate to the priority level of the I/O interrupt, relative to the total interrupt system, the processor enters an interrupt service routine (ISR). The function of the routine will depend upon the system of interrupt levels and priorities that is implemented in the processor.

In a single-level single-priority system there is only a single I/O interrupt – the logical OR of all the connected I/O devices. The associated interrupt service routine polls the peripherals to find the one with the interrupt status set.

Handshaking

Handshaking is a I/O control method to synchronize I/O devices with the microprocessor. As many I/O devices accepts or release information at a much slower rate than the microprocessor, this method is used to control the microprocessor to work with a I/O device at the I/O devices data transfer rate.

Handshaking is an automated process of negotiation that dynamically sets parameters of a communications channel established between two entities before normal communication over the channel begins. It follows the physical establishment of the channel and precedes normal information transfer. The handshaking process usually takes place in order to establish rules for communication when a computer sets about communicating with a foreign device. When a computer communicates with another device like a modem, printer, or network server, it needs to handshake with it to establish a connection.

Handshaking can negotiate parameters that are acceptable to equipment and systems at both ends of the communication channel, including information transfer rate, coding alphabet, parity, interrupt procedure, and other protocol or hardware features. Handshaking is a technique of communication between two entities. However, within TCP/IP RFCs, the term "handshake" is most commonly used to reference the TCP three-way handshake. For example, the term "handshake" is not present in RFCs covering FTP or SMTP. One exception is Transport Layer Security, TLS, setup, FTP RFC 4217. In place of the term "handshake", FTP RFC 3659 substitutes the term "conversation" for the passing of commands.

A simple handshaking protocol might only involve the receiver sending a message meaning "I received your last message and I am ready for you to send me another one." A more complex handshaking protocol might allow the sender to ask the receiver if it is ready to receive or for the receiver to reply with a negative acknowledgement meaning "I did not receive your last message correctly, please resend it" (e.g., if the data was corrupted en route).

Handshaking facilitates connecting relatively heterogeneous systems or equipment over a communication channel without the need for human intervention to set parameters.

Example: Supposing that we have a printer connected to a system. The printer can print 100 characters/second, but the microprocessor can send much more information to the printer at the same time. That's why, just when the printer gets it enough data to print it places a logic 1 signal at its Busy pin, indicating that it is busy in printing. The microprocessor now tests the busy bit to decide if the printer is busy or not. When the printer will become free it will change the busy bit and the microprocessor will again send enough amounts of data to be printed. This process of interrogating the printer is called handshaking.

Buffering

Is the process of transferring data between a program and an external device, The process of optimizing I/O consists primarily of making the best possible use of the slowest part of the path between the program and the device. The slowest part is usually the physical channel, which is often slower than the CPU or a memory-to-memory data transfer. The time spent in I/O processing overhead can reduce the amount of time that a channel can be used, thereby reducing the effective transfer rate. The biggest factor in maximizing this channel speed is often the reduction of I/O processing overhead.

A buffer is a temporary storage location for data while the data is being transferred. A buffer is often used for the following purposes:

- Small I/O requests can be collected into a buffer, and the overhead of making many relatively expensive system calls can be greatly reduced.
- A collection buffer of this type can be sized and handled so that the actual physical I/O requests made to the operating system match the physical characteristics of the device being used.
- Many data file structures, such as the f77 and cos file structures, contain control words. During the write process, a buffer can be used as a work area where control words can be inserted into the data stream (a process called blocking). The blocked data is then written to the device. During the read process, the same buffer work area can be used to examine and remove these control words before passing the data on to the user (deblocking).
- When data access is random, the same data may be requested many times. A cache is a buffer that keeps old requests in the buffer in case these requests are needed again. A cache that is sufficiently large and/or efficient can avoid a large part of the physical I/O by having the data ready in a buffer. When the data is often found in the cache buffer, it is referred to as having a high hit rate. For example, if the entire file fits in the cache and the file is present in the cache, no more physical requests are required to perform the I/O. In this case, the hit rate is 100%.
- Running the disks and the CPU in parallel often improves performance; therefore, it is useful to keep the CPU busy while data is being moved. To do this when writing, data can be transferred to the buffer at memory-to-memory copy speed and an asynchronous I/O request can be made. The control is then immediately returned to the program, which continues to execute as if the I/O were complete (a process called write-behind). A similar process can be used while reading; in this process, data is read into a buffer before the actual request is issued for it. When it is needed, it is already in the buffer and can be transferred to the user at very high speed. This is another form or use of a cache.

Conclusion

This section introduced the learner to the various ways, interfaces access and pass data. They include polling, interrupt and DMA. In them, speed between the different devices connected to the CPU are synchronized to be able to communicate effectively

Assessment

1.What is the difference between programmed-driven I/O and interrupt-driven I/O?

Programmed-driven I/O means the program is polling or checking some hardware item e.g. mouse within a loop.

For Interrupt driven I/O, the same mouse will trigger a signal to the program to process the mouse event.

2.What is one advantage and one disadvantage of each?

Advantage of Programmed Driven: easy to program and understand

Disadvantages: slow and inefficient

Advantage of Interrupt Driven: fast and efficient

Disadvantage: Can be tricky to write if you are using a low level language.

Can be tough to get the various pieces to work well together. Usually done by the hardware manufacturer or the OS maker e.g. Microsoft.

Activity 4.2 - Mechanisms of interruption: recognition of vector, and interrupt priority

Introduction

The following section introduces the learner to Interruptions that occur in Programmed I/O

Activity Details

In system programming, an interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt is a signal from a device attached to a computer or from a program within the computer that causes the main program that operates the computer (the operating system) to stop and figure out what to do next. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing. The processor responds by suspending its current activities, saving its state, and executing a function called an interrupt handler (or an interrupt service routine, ISR) to deal with the event. This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities. There are two types of interrupts: hardware interrupts and software interrupts.

Hardware interrupts

Hardware interrupts are used by devices to communicate that they require attention from the operating system. Internally, hardware interrupts are implemented using electronic alerting signals that are sent to the processor from an external device, which is either a part of the computer itself, such as a disk controller, or an external peripheral.

For example, pressing a key on the keyboard or moving the mouse triggers hardware interrupts that cause the processor to read the keystroke or mouse position. Unlike the software type (described below), hardware interrupts are asynchronous and can occur in the middle of instruction execution, requiring additional care in programming. The act of initiating a hardware interrupt is referred to as an interrupt (IRQ).

Software interrupt

A software interrupt is caused either by an exceptional condition in the processor itself, or a special instruction in the instruction which causes an interrupt when it is executed. The former is often called a trap or exception and is used for errors or events occurring during program executions that are exceptional enough that they cannot be handled within the program itself.

For example, if the processor's arithmetic logic unit is commanded to divide a number by zero, this impossible demand will cause a divide-by-zero exception, perhaps causing the computer to abandon the calculation or display an error message. Software interrupt instructions function similarly to subroutine calls and are used for a variety of purposes, such as to request services from low-level system software such as device drivers. For example, computers often use software interrupt instructions to communicate with the disk controller to request data be read or written to the disk.

Each interrupt has its own interrupt handler. The number of hardware interrupts is limited by the number of interrupt request (IRQ) lines to the processor, but there may be hundreds of different software interrupts. Interrupts are a commonly used technique for computer multitasking, especially in real-time computing. Such a system is said to be interrupt-driven

Interrupts can be categorized into these different types:

- Maskable interrupt (IRQ): a hardware interrupt that may be ignored by setting a bit in an interrupt mask register's (IMR) bit-mask.
- Non-maskable interrupt (NMI): a hardware interrupt that lacks an associated bit-mask, so that it can never be ignored. NMIs are used for the highest priority tasks such as timers, especially watchdog timers.
- Inter-processor interrupt (IPI): a special case of interrupt that is generated by one processor to interrupt another processor in a multiprocessor system.
- Software interrupt: an interrupt generated within a processor by executing an instruction. Software interrupts are often used to implement system calls because they result in a subroutine call with a CPU ring level change.
- Spurious interrupt: a hardware interrupt that is unwanted. They are typically generated by system conditions such as electrical interference on an interrupt line or through incorrectly designed hardware.

Conclusion

This section has introduced learners to the different categories of I/O interrupts, that is the hardware and software interrupts

Assessment

Briefly describe hardware and software Interruptions

Hardware interruptions

Hardware interruptions are generated by certain events which come up during the execution of a program. This type of interruptions is managed on their totality by the hardware and it is not possible to modify them.

A clear example of this type of interruptions is the one which actualizes the counter of the computer internal clock, the hardware makes the call to this interruption several times during a second in order to maintain the time up to date.

Hardware Interruptions

External interruptions are generated by peripheral devices, such as keyboards, printers, communication cards, etc. They are also generated by coprocessors. It is not possible to deactivate external interruptions.

These interruptions are not sent directly to the CPU but they are sent to an integrated circuit whose function is to exclusively handle this type of interruptions

Software Interruptions

Software interruptions can be directly activated by the assembler invoking the number of the desired interruption with the INT Instruction.

The use of interruptions helps us in the creation of programs and by using them our programs gets shorter. It is easier to understand them and they usually have a better performance mostly due to their smaller size. This type of interruptions can be separated in two categories: the operative system DOS interruptions and the BIOS interruptions.

Activity 4.3 - Direct Memory Access (DMA)

Introduction

This section introduces the learners to the DMA programmed I/O which provides access to the microprocessor between devices operating at different speeds

Activity Details

Direct Memory Access and DMA-controlled I/O

The DMA I/O technique is used in personal computer systems including those using Intel family of microprocessors. The direct memory access (DMA) I/O technique provides direct access to the memory while the microprocessor is temporarily disabled. A DMA controller temporarily borrows the address bus, data bus, and control bus from the microprocessor and transfers the data bytes directly between an I/O port and a series of memory locations. The DMA transfer is also used to do high-speed memory-to memory transfers. Two control signals are used to request and acknowledge a DMA transfer in the microprocessor-based system. The HOLD signal is a bus request signal which asks the microprocessor to release control of the buses after the current bus cycle. The HLDA signal is a bus grant signal which indicates that the microprocessor has indeed released control of its buses by placing the buses at their high-impedance states. The HOLD input has a higher priority than the INTR or NMI interrupt inputs.

Special hardware writes to / reads from memory directly (without CPU intervention) and saves the timing associated with op-code fetch and decoding, increment and test addresses of source and destination. The DMA controller may both stop the CPU and access the memory (cycle stealing DMA) or use the bus while the CPU is not using it (hidden cycle DMA). The DMA controller has some control lines (to do a handshake with the CPU negotiating to be a bus master and to emulate the CPU behaviour while accessing the memory), an address register which is auto-incremented (or auto-decremented) at each memory access, and a counter used to check for final byte (or word) count.

Conclusion

This section has introduced the learners to the DMA and its operations of synchronizing the I/O devices with the microprocessor

Assessment

Describe how DMA helps in the synchronization of different devices in accessing the microprocessor

The direct memory access (DMA) I/O technique provides direct access to the memory while the microprocessor is temporarily disabled. A DMA controller temporarily borrows the address bus, data bus, and control bus from the microprocessor and transfers the data bytes directly between an I/O port and a series of memory locations. The DMA transfer is also used to do high-speed memory-to-memory transfers. Two control signals are used to request and acknowledge a DMA transfer in the microprocessor-based system. The HOLD signal is a bus request signal which asks the microprocessor to release control of the buses after the current bus cycle. The HLDA signal is a bus grant signal which indicates that the microprocessor has indeed released control of its buses by placing the buses at their high-impedance states. The HOLD input has a higher priority than the INTR or NMI interrupt inputs.

Unit Summary

At the end of this unit, the learners will be conversant with the strategies of I/O interfaces. This involves accessibility of devices connected to the processor and where I/O transfers must take place between them and the processor. the various access methods, e.g. polling, interrupt and DMA. The interrupt process is also learned in this section.

Unit Assessment

The following section will test the learners understanding of this unit

Instructions

Answer the following questions

1.Explain two strategies that govern I/O transfers

2.What is handshaking and how is it carried out?

Grading Scheme

The marks will be awarded as shown below

question	sub-question	marks awarded
1	explanations of any two @ 4 mark	8
2	definition award 2 marks, explanation of how it works 4 marks.	6
Total		14

Feedback

1. Expalin any two from the following

a. Programmed I/O

Programmed I/O (PIO) refers to data transfers initiated by a CPU under driver software control to access registers or memory on a device. The CPU issues a command then waits for I/O operations to be complete. As the CPU is faster than the I/O module, the problem with programmed I/O is that the CPU has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data. The CPU, while waiting, must repeatedly check the status of the I/O module, and this process is known as Polling. As a result, the level of the performance of the entire system is severely degraded.

b. Interrupt driven I/O

The CPU issues commands to the I/O module then proceeds with its normal work until interrupted by I/O device on completion of its work.

For input, the device interrupts the CPU when new data has arrived and is ready to be retrieved by the system processor. The actual actions to perform depend on whether the device uses I/O ports, memory mapping.

For output, the device delivers an interrupt either when it is ready to accept new data or to acknowledge a successful data transfer. Memory-mapped and DMA-capable devices usually generate interrupts to tell the system they are done with the buffer.

Although Interrupt relieves the CPU of having to wait for the devices, but it is still inefficient in data transfer of large amount because the CPU has to transfer the data word by word between I/O module and memory.

c. Direct Memory Access (DMA)

Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module controls exchange of data between main memory and the I/O device. Because of DMA device can transfer data directly to and from memory, rather than using the CPU as an intermediary, and can thus relieve congestion on the bus. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.

DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory busses. Hardware design is complicated because the DMA controller must be integrated into the system, and the system must allow the DMA controller to be a bus master. Cycle stealing may also be necessary to allow the CPU and DMA controller to share use of the memory bus.

2. Handshaking is a I/O control method to synchronize I/O devices with the microprocessor. this method is used to control the microprocessor to work with a I/O device at the I/O devices data transfer rate. Handshaking is an automated process of negotiation that dynamically sets parameters of a communications channel established between two entities before normal communication over the channel begins.

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources

Unit 5. The Peripheral Devices

Unit Introduction

This unit introduces peripheral device which are connected to the computer. It mainly deals on what needs to be done to the data that flows between the processor and the Peripheral devices

The peripherals include computer mouse, keyboard, image scanners, tape drives, microphones, loudspeakers, webcams, and digital cameras.

Unit Objectives

At the end of the lesson the learners should be able to;-

1. Define peripheral devices
2. Understand the meaning of digitization, coding-decoding and compression
3. understand how to convert analog to digital data

Key Terms

Peripheral devices: is any auxiliary device such as a computer mouse or keyboard that connects to and works with the computer in some way. Other examples of peripherals are image scanners, tape drives, microphones, loudspeakers, webcams, and digital cameras.

Sampling: is the reduction of a continuous signal to a discrete signal. A common example is the conversion of a sound wave (a continuous signal) to a sequence of samples (a discrete-time signal).

Quantization: is the process of mapping a large set of input values to a (countable) smaller set.

Analog data: Analog data is data that is represented in a physical way.

Digital data: are discrete, discontinuous representations of information or works, as contrasted with continuous, or analog signals which behave in a continuous manner, or represent information using a continuous function.

Learning Activities

Activity 5.1 - Representation of digital and analog values - sampling and quantization

Introduction

This section introduces the learner to sampling and quantization. This is where signals are represented into horizontal and vertical values (axes).

Activity Details

Analog and Digital Signals

Digitalization of an analog signal involves two operations:

1) Sampling, and
2) Quantization

Analog signals consist of continuous values for both axes. Consider an electrical signal whose horizontal axis represents time in seconds and whose vertical axis represents amplitude in volts. The horizontal axis has a range of values from zero to infinity with every possible value in between. This makes the horizontal axis continuous. The vertical axis is also continuous allowing the signal's amplitude to assume any value from zero to infinity. For every possible value in time there is a corresponding value in amplitude for the analog signal.

An analog signal exists throughout a continuous interval of time and/or takes on a continuous range of values. A sinusoidal signal (also called a pure tone in acoustics) has both of these properties.

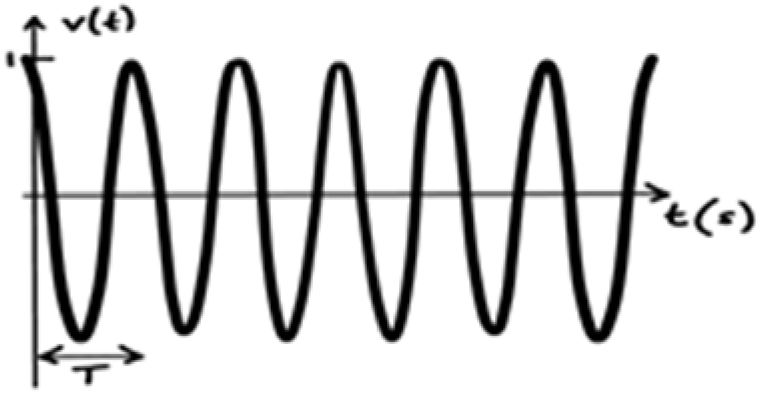


Fig. 1: Analog signal. This signal v(t)=cos(2πft) could be a perfect analog recording of a pure tone of frequency f Hz. If f=440 Hz, this tone is the musical note A above middle C, to which orchestras often tune their instruments. The period T=1/f is the duration of one full oscillation.

In reality, electrical recordings suffer from noise that unavoidably degrades the signal. The more a recording is transferred from one analog format to another, the more it loses fidelity to the original.

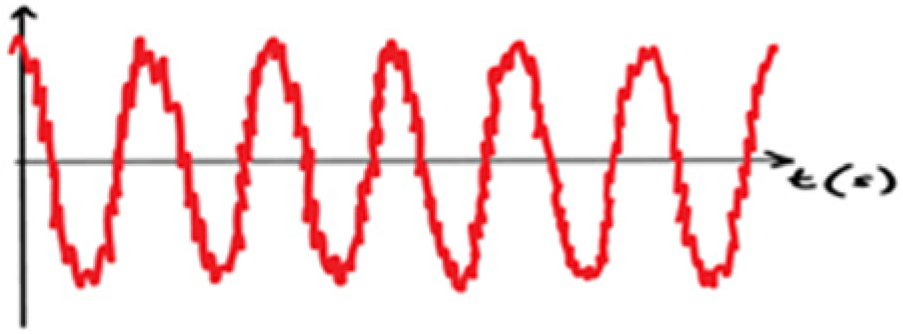


Fig. 2: Noisy analog signal. Noise degrades the sinusoidal signal in Fig. 1. It is often impossible to recover the original signal exactly from the noisy version.

Digital signals on the other hand have discrete values for both the horizontal and vertical axes. The axes are no longer continuous as they were with the analog signal. In this discussion, time will be used as the quantity for the horizontal axis and volts will be used for the vertical axis.

A digital signal is a sequence of discrete symbols. If these symbols are zeros and ones, we call them bits. As such, a digital signal is neither continuous in time nor continuous in its range of values. And, therefore, cannot perfectly represent arbitrary analog signals. On the other hand, digital signals are resilient against noise.

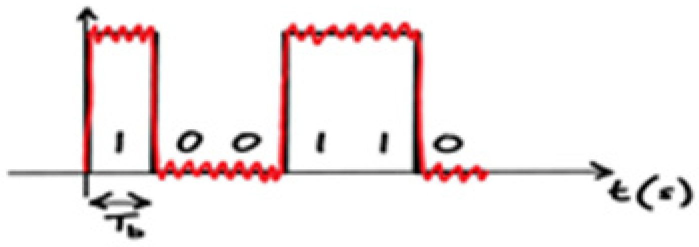


Fig. 3: Analog transmission of a digital signal. Consider a digital signal 100110 converted to an analog signal for radio transmission. The received signal suffers from noise, but given sufficient bit duration Tb, it is still easy to read off the original sequence 100110 perfectly.

Digital signals can be stored on digital media (like a compact disc) and manipulated on digital systems (like the integrated circuit in a CD player). This digital technology enables a variety of digital processing unavailable to analog systems. For example, the music signal encoded on a CD includes additional data used for digital error correction. In case the CD is scratched and some of the digital signal becomes corrupted, the CD player may still be able to reconstruct the missing bits exactly from the error correction data. To protect the integrity of the data despite being stored on a damaged device, it is common to convert analog signals to digital signals using steps called sampling and quantization.

Introduction to Sampling

The motivation for sampling and quantizing is the need to store a signal in a digital format. In order to convert an analog signal to a digital signal, the analog signal must be sampled and quantized. Sampling takes the analog signal and discretizes the time axis. After sampling, the time axis consists of discrete points in time rather than continuous values in time. The resulting signal after sampling is called a discrete signal, sampled signal, or a discrete-time signal. The resulting signal after sampling is not a digital signal. Even though the horizontal axis has discrete values the vertical axis is not discretized. This means that for any discrete point in time, there are an infinite number of allowed values for the signal to assume in amplitude. In order for the signal to be a digital signal, both axes must be discrete.

Sampling is the process of recording an analog signal at regular discrete moments of time. The sampling rate fs is the number of samples per second. The time interval between samples is called the sampling interval Ts=1/fs.

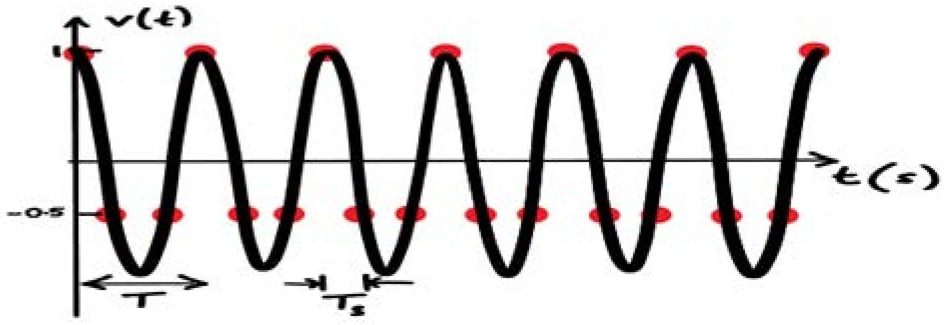


Fig. 4: Sampling. The signal v(t)=cos(2πft) in Fig. 1 is sampled uniformly with 3 sampling intervals within each signal period T. Therefore, the sampling interval Ts=T/3 and the sampling rate fs=3f. Another way see that fs=3f is to notice that there are three samples in every signal period T.

To express the samples of the analog signal v(t), we use the notation v[n] (with square brackets), where integer values of n index the samples. Typically, the n=0 sample is taken from the t=0 time point of the analog signal. Consequently, the n=1 sample must come from the t=Ts time point, exactly one sampling interval later; and so on. Therefore, the sequence of samples can be written as v[0]=v(0), v[1]=v(Ts), v[2]=v(2Ts),…

v[n]=v(nTs) for integer n …………………………………………1

In the example of Fig. 4, v(t)=cos(2πft) is sampled with sampling interval Ts=T/3

to produce the following v[n].

v[n]=cos(2πfn/Ts) by substituting t=nTs ……………………………………………………..2

=cos(2πfnT/3) since Ts=T/3 ………………………….3

=cos(2πn/3) since T=1/f …………………………..4

This expression for v[n] evaluates to the sample values depicted in Fig. 4 as shown below.

v[0=cos(0)=1

]v[1] =cos(2π3)=−0.5

v[2] =cos(4π3)=−0.5

v[3] =cos(2π)=1

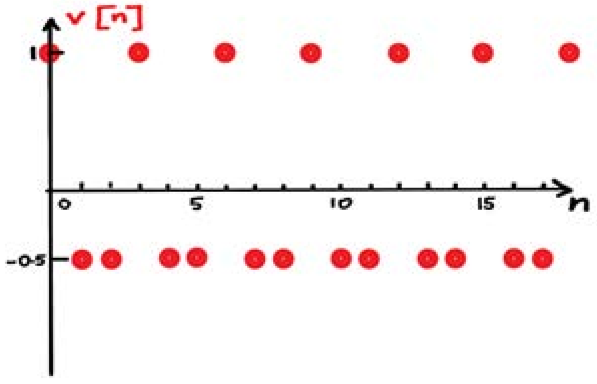


Fig. 5: Samples. The samples from Fig. 4 are shown as the sequence v[n] indexed by integer values of n.

Quantization

Since a discrete signal has discrete points in time but still has continuous values in amplitude, the amplitude of the signal must be discretized in order to store it in digital format. The values of the amplitude must be rounded off to discrete values. If the vertical axis is divided into small windows of amplitudes, then every value that lies within that window will be rounded off (or quantized) to the same value.

For example, consider a waveform with window sizes of 0.5 volts starting at –4 volts and ending at +4 volts. At a discrete point in time, any amplitude between 4.0 volts and 3.5 volts will be recorded as 3.75 volts. In this example the center of each 0.5-volt window (or quantization region) was chosen to be the quantization voltage for that region.

In this example the dynamic range of the signal is 8 volts. Since each quantization region is 0.5 volts there are 16 quantization regions included in the dynamic range. It is important that there are 16 quantization regions in the dynamic range. Since a binary number will represent the value of the amplitude, it is important that the number of quantization regions is a power of two. In this example, 4 bits will be required to represent each of the 16 possible values in the signal's amplitude.

A sequence of samples like v[n] in Fig. 5 is not a digital signal because the sample values can potentially take on a continuous range of values. In order to complete analog to digital conversion, each sample value is mapped to a discrete level (represented by a sequence of bits) in a process called quantization. In a B-bit quantizer, each quantization level is represented with B bits, so that the number of levels equals 2B

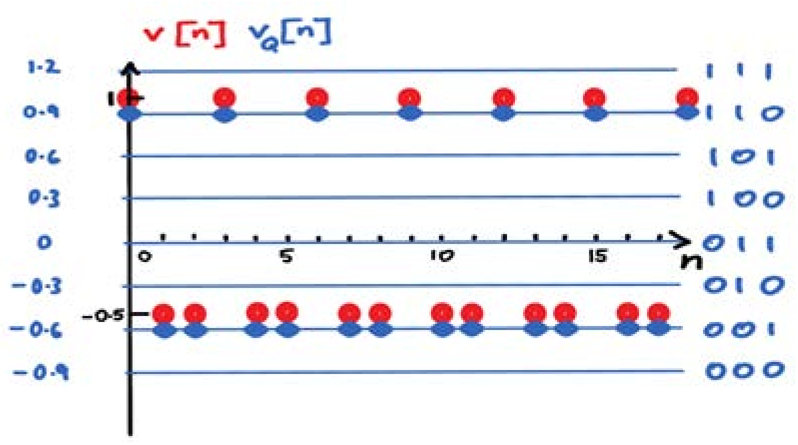


Fig. 6: 3-bit quantization. Overlaid on the samples v[n] from Fig. 5 is a 3-bit quantizer with 8 uniformly spaced quantization levels. The quantizer approximates each sample value in v[n] to its nearest level value (shown on the left), producing the quantized sequence vQ[n]. Ultimately the sequence vQ[n] can be written as a sequence of bits using the 3-bit representations shown on the right.

Observe that quantization introduces a quantization error between the samples and their quantized versions given by e[n]=v[n]−vQ[n]. If a sample lies between quantization levels, the maximum absolute quantization error le[n]l is half of the spacing between those levels. For the quantizer in Fig. 6, the maximum error between levels is 0.15 since the spacing is uniformly 0.3. Note, however, that if the sample overshoots the highest level or undershoots the lowest level by more than 0.15, the absolute quantization error will be that difference larger than 0.15.

The table below completes the quantization example in Fig. 6 for n=0,1,2,3. The 3-bit representations in the final row can be concatenated finally into the digital signal 110001001110.

Table 1: Quantization example.

Sequence	n=0	n=1	n=2	n=3
Samples v[n]	1	-0.5	-0.5	1
Quantized samples vQ[n]	0.9	-0.6	-0.6	0.9
Quantization error e[n]=v[n]−vQ[n]	0.1	0.1	0.1	0.1
3-bit representations	110	1	1	110

Conclusion

This section has made the learners learn how analog (continuous) data can be digitized

Assessment

1.What is the difference between analogue and digital data?

Analogue data is continuous, allowing for an infinite number of possible values. Digital data is discrete, allowing for a finite set of values

2.Why is it difficult to save analogue sound waves in a digital format?

Analogue is continuous data, converting continuous data to discrete values may lose some of the accuracy

3.Differentiate between anlog and digital data

Analog refers to circuits in which quantities such as voltage or current vary at a continuous rate. When you turn the dial of a potentiometer, for example, you change the resistance by a continuously varying rate. The resistance of the potentiometer can be any value between the minimum and maximum allowed by the pot. In digital electronics, quantities are counted rather than measured. There's an important distinction between counting and measuring. When you count something, you get an exact result. When you measure something, you get an approximate result.

Activity 5.2 - Sound and Audio, Image and Graphics, Animation and Video

Introduction

The following sections describe various types of data that you might find, in addition to static graphics data, in multimedia files.

Activity Details

Computer animation lies somewhere between the motionless world of still images and the real-time world of video images. All of the animated sequences seen in educational programs, motion CAD renderings, and computer games are computer-animated (and in many cases, computer-generated) animation sequences.

Traditional cartoon animation is little more than a series of artwork cells, each containing a slight positional variation of the animated subjects. When a large number of these cells is displayed in sequence and at a fast rate, the animated figures appear to the human eye to move.

A computer-animated sequence works in exactly the same manner, i.e a series of images is created of a subject; each image contains a slightly different perspective on the animated subject. When these images are displayed (played back) in the proper sequence and at the proper speed (frame rate), the subject appears to move.

Computerized animation is actually a combination of both still and motion imaging. Each frame, or cell, of an animation is a still image that requires compression and storage. An animation file, however, must store the data for hundreds or thousands of animation frames and must also provide the information necessary to play back the frames using the proper display mode and frame rate.

Animation file formats are only capable of storing still images and not actual video information. It is possible, however, for most multimedia formats to contain animation information, because animation is actually a much easier type of data than video to store.

The image-compression schemes used in animation files are also usually much simpler than most of those used in video compression. Most animation files use a delta compression scheme, which is a form of Run-Length Encoding that stores and compresses only the information that is different between two images (rather than compressing each image frame entirely). RLE is relatively easy to decompress on the fly.

Storing animations using a multimedia format also produces the benefit of adding sound to the animation (what's a cartoon without sound?). Most animation formats cannot store sound directly in their files and must rely on storing the sound in a separate disk file which is read by the application that is playing back the animation.

Animations are not only for entertaining kids and adults. Animated sequences are used by CAD programmers to rotate 3D objects so they can be observed from different perspectives; mathematical data collected by an aircraft or satellite may be rendered into an animated fly-by sequence. Movie special effects benefit greatly by computer animation.

Digital Video

One step beyond animation is broadcast video. Your television and video tape recorder are a lot more complex than an 8mm home movie projector and your kitchen wall. There are many complex signals and complicated standards that are involved in transmitting those late-night reruns across the airwaves and cable. Only in the last few years has a personal computer been able to work with video data at all.

Video data normally occurs as continuous, analog signals. In order for a computer to process this video data, we must convert the analog signals to a non-continuous, digital format. In a digital format, the video data can be stored as a series of bits on a hard disk or in computer memory.

The process of converting a video signal to a digital bitstream is called analog-to-digital conversion (A/D conversion), or digitizing. A/D conversion occurs in two steps:

1. Sampling captures data from the video stream.
2. Quantizing converts each captured sample into a digital format.

Each sample captured from the video stream is typically stored as a 16-bit integer. The rate at which samples are collected is called the sampling rate. The sampling rate is measured in the number of samples captured per second (samples/second). For digital video, it is necessary to capture millions of samples per second.

Quantizing converts the level of a video signal sample into a discrete, binary value. This value approximates the level of the original video signal sample. The value is selected by comparing the video sample to a series of predefined threshold values. The value of the threshold closest to the amplitude of the sampled signal is used as the digital value.

A video signal contains several different components which are mixed together in the same signal. This type of signal is called a composite video signal and is not really useful in high-quality computer video. Therefore, a standard composite video signal is usually separated into its basic components before it is digitized.

The composite video signal format defined by the NTSC (National Television Standards Committee) color television system is used in the United States. The PAL (Phase Alternation Line) and SECAM (Sequential Coleur Avec Memoire) color television systems are used in Europe and are not compatible with NTSC. Most computer video equipment supports one or more of these system standards.

The components of a composite video signal are normally decoded into three separate signals representing the three channels of a color space model, such as RGB, YUV, or YIQ. Although the RGB model is quite commonly used in still imaging, the YUV, YIQ, or YCbCr models are more often used in motion-video imaging. TV practice uses YUV or similar color models because the U and V channels can be downsampled to reduce data volume without materially degrading image quality.

Once the video signal is converted to a digital format, the resulting values can be represented on a display device as pixels. Each pixel is a spot of color on the video display, and the pixels are arranged in rows and columns just as in a bitmap. Unlike a static bitmap, however, the pixels in a video image are constantly being updated for changes in intensity and color. This updating is called scanning, and it occurs 60 times per second in NTSC video signals (50 times per second for PAL and SECAM).

A video sequence is displayed as a series of frames. Each frame is a snapshot of a moment in time of the motion-video data, and is very similar to a still image. When the frames are played back in sequence on a display device, a rendering of the original video data is created. In real-time video the playback rate is 30 frames per second. This is the minimum rate necessary for the human eye to successfully blend each video frame together into a continuous, smoothly moving image.

A single frame of video data can be quite large in size. A video frame with a resolution of 512 x 482 will contain 246,784 pixels. If each pixel contains 24 bits of color information, the frame will require 740,352 bytes of memory or disk space to store. Assuming there are 30 frames per second for real-time video, a 10-second video sequence would be more than 222 megabytes in size! It is clear there can be no computer video without at least one efficient method of video data compression.

There are many encoding methods available that will compress video data. The majority of these methods involve the use of a transform coding scheme, usually employing a Fourier or Discrete Cosine Transform (DCT). These transforms physically reduce the size of the video data by selectively throwing away unneeded parts of the digitized information.

Transform compression schemes usually discard 10 percent to 25 percent or more of the original video data, depending largely on the content of the video data and upon what image quality is considered acceptable.

Usually a transform is performed on an individual video frame. The transform itself does not produce compressed data. It discards only data not used by the human eye. The transformed data, called coefficients, must have compression applied to reduce the size of the data even further. Each frame of data may be compressed using a Huffman or arithmetic encoding algorithm, or even a more complex compression scheme such as JPEG. This type of intraframe encoding usually results in compression ratios between 20:1 to 40:1 depending on the data in the frame. However, even higher compression ratios may result if, rather than looking at single frames as if they were still images, we look at multiple frames as temporal images.

In a typical video sequence, very little data changes from frame to frame. If we encode only the pixels that change between frames, the amount of data required to store a single video frame drops significantly. This type of compression is known as interframe delta compression, or in the case of video, motion compensation. Typical motion compensation schemes that encode only frame deltas (data that has changed between frames) can, depending on the data, achieve compression ratios upwards of 200:1. This is only one possible type of video compression method. There are many other types of video compression schemes, some of which are similar and some of which are different.

Digital Audio

All multimedia file formats are capable, by definition, of storing sound information. Sound data, like graphics and video data, has its own special requirements when it is being read, written, interpreted, and compressed. Before looking at how sound is stored in a multimedia format we must look at how sound itself is stored as digital data. All of the sounds that we hear occur in the form of analog signals. An analog audio recording system, such as a conventional tape recorder, captures the entire sound wave form and stores it in analog format on a medium such as magnetic tape.

Because computers are now digital devices it is necessary to store sound information in a digitized format that computers can readily use. A digital audio recording system does not record the entire wave form as analog systems do (the exception being Digital Audio Tape [DAT] systems). Instead, a digital recorder captures a wave form at specific intervals, called the sampling rate. Each captured wave-form snapshot is converted to a binary integer value and is then stored on magnetic tape or disk.

Storing audio as digital samples is known as Pulse Code Modulation (PCM). PCM is a simple quantizing or digitizing (audio to digital conversion) algorithm, which linearly converts all analog signals to digital samples. This process is commonly used on all audio CD-ROMs.

Differential Pulse Code Modulation (DPCM) is an audio encoding scheme that quantizes the difference between samples rather than the samples themselves. Because the differences are easily represented by values smaller than those of the samples themselves, fewer bits may be used to encode the same sound (for example, the difference between two 16-bit samples may only be four bits in size). For this reason, DPCM is also considered an audio compression scheme.

One other audio compression scheme, which uses difference quantization, is Adaptive Differential Pulse Code Modulation (ADPCM). DPCM is a non-adaptive algorithm. That is, it does not change the way it encodes data based on the content of the data. DPCM uses the sample number of bits to represent every signal level. ADPCM, however, is an adaptive algorithm and changes its encoding scheme based on the data it is encoding. ADPCM specifically adapts by using fewer bits to represent lower-level signals than it does to represent higher-level signals. Many of the most commonly used audio compression schemes are based on ADPCM.

Digital audio data is simply a binary representation of a sound. This data can be written to a binary file using an audio file format for permanent storage much in the same way bitmap data is preserved in an image file format. The data can be read by a software application, can be sent as data to a hardware device, and can even be stored as a CD-ROM.

The quality of an audio sample is determined by comparing it to the original sound from which it was sampled. The more identical the sample is to the original sound, the higher the quality of the sample. This is similar to comparing an image to the original document or photograph from which it was scanned.

The quality of audio data is determined by three parameters:

- Sample resolution
- Sampling rate
- Number of audio channels sampled

The sample resolution is determined by the number of bits per sample. The larger the sampling size, the higher the quality of the sample. Just as the apparent quality (resolution) of an image is reduced by storing fewer bits of data per pixel, so is the quality of a digital audio recording reduced by storing fewer bits per sample. Typical sampling sizes are eight bits and 16 bits.

The sampling rate is the number of times per second the analog wave form was read to collect data. The higher the sampling rate, the greater the quality of the audio. A high sampling rate collects more data per second than a lower sampling rate, therefore requiring more memory and disk space to store. Common sampling rates are 44.100 kHz (higher quality), 22.254 kHz (medium quality), and 11.025 kHz (lower quality). Sampling rates are usually measured in the signal processing terms hertz (Hz) or kilohertz (kHz), but the term samples per second (samples/second) is more appropriate for this type of measurement.

A sound source may be sampled using one channel (monaural sampling) or two channels (stereo sampling). Two-channel sampling provides greater quality than mono sampling and, as you might have guessed, produces twice as much data by doubling the number of samples captured. Sampling one channel for one second at 11,000 samples/second produces 11,000 samples. Sampling two channels at the same rate, however, produces 22,000 samples/second.

The amount of binary data produced by sampling even a few seconds of audio is quite large. Ten seconds of data sampled at low quality (one channel, 8-bit sample resolution, 11.025 samples/second sampling rate) produces about 108K of data (88.2 Kbits/second).

Adding a second channel doubles the amount of data to produce nearly a 215K file (176 Kbits/second). If we increase the sample resolution to 16 bits, the size of the data doubles again to 430K (352 Kbits/second). If we now increase the sampling rate to 22.05 Ksamples/second, the amount of data produced doubles again to 860K (705.6 Kbits/second). At the highest quality generally used (two channels, 16-bit sample resolution, 44.1 Ksamples/second sampling rate), our 10 seconds of audio now requires 1.72 megabytes (1411.2 Kbits/second) of disk space to store.

Consider how little information can really be stored in 10 seconds of sound. The typical musical song is at least three minutes in length. Music videos are from five to 15 minutes in length. A typical television program is 30 to 60 minutes in length. Movie videos can be three hours or more in length. We're talking a lot of disk space here.

One solution to the massive storage requirements of high-quality audio data is data compression. For example, the CD-DA (Compact Disc-Digital Audio) standard performs mono or stereo sampling using a sample resolution of 16 bits and a sampling rate of 44.1 samples/second, making it a very high-quality format for both music and language applications. Storing five minutes of CD-DA information requires approximately 25 megabytes of disk space--only half the amount of space that would be required if the audio data were uncompressed.

Audio data, in common with most binary data, contains a fair amount of redundancy that can be removed with data compression. Conventional compression methods used in many archiving programs (zoo and pkzip, for example) and image file formats don't do a very good job of compressing audio data (typically 10 percent to 20 percent). This is because audio data is organized very differently from either the ASCII or binary data normally handled by these types of algorithms.

Audio compression algorithms, like image compression algorithms, can be categorized as lossy and lossless. Lossless compression methods do not discard any data. The decompression step produces exactly the same data as was read by the compression step. A simple form of lossless audio compression is to Huffman-encode the differences between each successive 8-bit sample. Huffman encoding is a lossless compression algorithm and, therefore the audio data is preserved in its entirety.

Lossy compression schemes discard data based on the perceptions of the psychoacoustic system of the human brain. Parts of sounds that the ear cannot hear, or the brain does not care about, can be discarded as useless data.

An algorithm must be careful when discarding audio data. The ear is very sensitive to changes in sound. The eye is very forgiving about dropping a video frame here or reducing the number of colors there. The ear, however, notices even slight changes in sounds, especially when specifically trained to recognize audial infidelities and discrepancies. However, the higher the quality of an audio sample, the more data will be required to store it. As with lossy image compression schemes, at times you'll need to make a subjective decision between quality and data size.

Audio

There is currently no "audio file interchange format" that is widely used in the computer-audio industry. Such a format would allow a wide variety of audio data to be easily written, read, and transported between different hardware platforms and operating systems.

Most existing audio file formats, however, are very machine-specific and do not lend themselves to interchange very well. Several multimedia formats are capable of encapsulating a wide variety of audio formats, but do not describe any new audio data format in themselves.

Many audio file formats have headers just as image files do. Their header information includes parameters particular to audio data, including sample rate, number of channels, sample resolution, type of compression, and so on. An identification field ("magic" number) is also included in several audio file format headers.

Several formats contain only raw audio data and no file header. Any parameters these formats use are fixed in value and therefore would be redundant to store in a file header. Stream-oriented formats contain packets (chunks) of information embedded at strategic points within the raw audio data itself. Such formats are very platform-dependent and would require an audio file format reader or converter to have prior knowledge of just what these parameter values are.

Most audio file formats may be identified by their file types or extensions. Some common sound file formats are:

.AU	Sun Microsystems
.SND	NeXT
HCOM	Apple Macintosh
.VOC	SoundBlaster
.WAV	Microsoft Waveform
AIFF	Apple/SGI
8SVX	Apple/SGI

A multimedia format may choose to either define its own internal audio data format or simply encapsulate an existing audio file format. Microsoft Waveform files are RIFF files with a single Waveform audio file component, while Apple QuickTime files contain their own audio data structures unique to QuickTime files.

MIDI Standard

Musical Instrument Digital Interface (MIDI) is an industry standard for representing sound in a binary format. MIDI is not an audio format, however. It does not store actual digitally sampled sounds. Instead, MIDI stores a description of sounds, in much the same way that a vector image format stores a description of an image and not image data itself.

Sound in MIDI data is stored as a series of control messages. Each message describes a sound event using terms such as pitch, duration, and volume. When these control messages are sent to a MIDI-compatible device (the MIDI standard also defines the interconnecting hardware used by MIDI devices and the communications protocol used to interchange the control information) the information in the message is interpreted and reproduced by the device.

MIDI data may be compressed, just like any other binary data, and does not require special compression algorithms in the way that audio data does.

Conclusion

The activity introduced the various data formats that are possible in a multimedia, it also explained the conversions possible, e.g. sampling, quantization and animations

Assessment

1.What is digital conversion? is a very useful feature that converts an analog voltage on a pin to a digital number. By converting from the analog world to the digital world, we can begin to use electronics to interface to the analog world around us.

e.g. Analog-to-digital conversion is an electronic process in which a continuously variable (analog) signal is changed, without altering its essential content, into a multi-level (digital) signal.

The input to an analog-to-digital converter (ADC) consists of a voltage that varies among a theoretically infinite number of values. Examples are sine waves, the waveforms representing human speech, and the signals from a conventional television camera. The output of the ADC, in contrast, has defined levels or states. The number of states is almost always a power of two -- that is, 2, 4, 8, 16, etc. The simplest digital signals have only two states, and are called binary. All whole numbers can be represented in binary form as strings of ones and zeros.

2.Explain MIDI

MIDI (Musical Instrument Digital Interface) is a protocol designed for recording and playing back music on digital synthesizers that is supported by many makes of personal computer sound cards. Originally intended to control one keyboard from another, it was quickly adopted for the personal computer. Rather than representing musical sound directly, it transmits information about how music is produced. The command set includes note-ons, note-offs, key velocity, pitch bend and other methods of controlling a synthesizer. The sound waves produced are those already stored in a wavetable in the receiving instrument or sound card.

Activity 5.3 - Coding and decoding multimedia systems

Introduction

This section introduces the learner to the fact that Multimedia data which is voluminous in nature needs to be coded and decoded so that it can be transmitted fast on the existing media

Activity Details

In multimedia system design, storage and transport of information play a significant role. Multimedia information is inherently voluminous and therefore requires very high storage capacity and very high bandwidth transmission capacity. There are two approaches that are possible - one to develop technologies to provide higher bandwidth (of the order of Gigabits per second or more) and the other to find ways and means by which the number of bits to be transferred can be reduced, without compromising the information content. There are two approaches that are possible - one to develop technologies to provide higher bandwidth (of the order of Gigabits per second or more) and the other to find ways and means by which the number of bits to be transferred can be reduced, without compromising the information content, i.e. data compression.

Data compression is often referred to as coding, whereas coding is a general term encompassing any special representation of data that achieves a given goal. Information coded (compression) done at the source end has to be correspondingly decoded at the receiving end. Coding can be done in such a way that the information content is not lost; that means it can be recovered fully on decoding at the receiver. However, media such as image and video (meant primarily for human consumption) provide opportunities to encode more sufficiently but with a loss. Coding (consequently, the compression) of multimedia information is subject to certain quality constraints. For example, the quality of a picture should be the same when coded and, later on, decoded.

Coding and compression techniques are critical to the viability of multimedia at both the storage level and at the communication level. Some of the multimedia information has to be coded in continuous (time dependent) format and some in discrete (time independent) format. In multimedia context, the primary motive in coding is compression. By nature, the audio, image, and video sources have built-in redundancy, which make it possible to achieve compression through appropriate coding. As the image data and video data are voluminous and act as prime motivating factors for compression, our references in this chapter will often be to images even though other data types can be coded (compressed).

Conclusion

This section has informed the learner the reasons why compression is necessary in multimedia data, that is why coding and decoding are necessary in multimedia data transmission. processes like coding, compression and decoding were learned.

Assessment

1.What is multimedia? is content that uses a combination of different content forms such as text, audio, images, animation, video and interactive content

2.Define the terms;

Coding; is the process of putting a sequence of characters (letters, numbers, punctuation, and certain symbols) into a specialized format for efficient transmission or storage.

Decoding; the conversion of an encoded format back into the original sequence of characters.

Compression; is a reduction in the number of bits needed to represent data. Compressing data can save storage capacity, speed file transfer, and decrease costs for storage hardware and network bandwidth.

Unit Summary

This unit introduced the learner to the peripheral device that can be connected to the computer. It dealt on the conversions of analog to digital data. the different wave forms representing conversions of the analog to digital data were also introduced. Also the multimedia data and what needs to be done for it to be transmitted across networks. The peripherals include computer mouse, keyboard, image scanners, tape drives, microphones, loudspeakers, webcams, and digital cameras.

Unit Assessment

The following section will test the learners understanding of this unit

Instructions

Answer the following questions

1.Explain digitization of an analog signal?

2.What is sampling in digitization?

3.Explain the term animation

Grading Scheme

The marks will be awarded as shown below

Question	Sub-question	marks awarded
1	Explanation award 2 mark. extars like examples and diagrams award 2 extra marks; maximum 2 giving total 6	6
2	Definition award 2marks,	2
3	Explanation only 2 marks.	2
Total		10

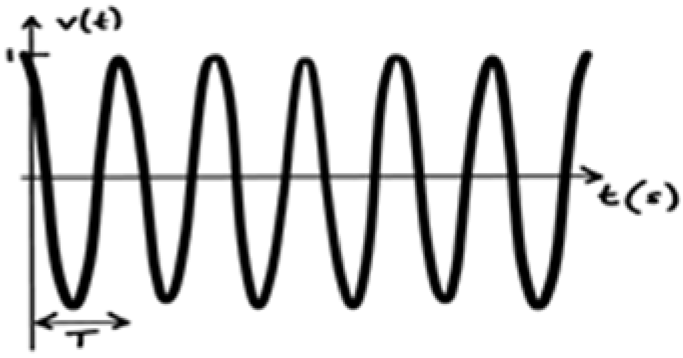
Feedback

1. Digitalization of an analog signal involves two operations:

1) Sampling, and

2) Quantization

Analog signals consist of continuous values for both axes. An analog signal exists throughout a continuous interval of time and/or takes on a continuous range of values. A sinusoidal signal (also called a pure tone in acoustics) has both of these properties.



2.Sampling is the process of recording an analog signal at regular discrete moments of time.

3.Computer animation lies somewhere between the motionless world of still images and the real-time world of video images.

Unit Readings and Other Resources

The readings in this unit are to be found at course level readings and other resources.

Course Assessment

Introduction

This assessment consists of an assignment, a sitting CAT and a final exam. The assiment has a weight of 20 while the CAt has a weight of 30. The final exam has a weighting of 50, giving a total of 100 percent.

Mid Term Exam 1(20 %)

Instructions

Answer the following question be as detailed as possible

Question: Explain the concept of interrupts and DMA.

Grading Scheme

Marks to be awarded based on key issues mentioned in the explanation. maximum is 10 marks for correct answers

Feedback

When an interrupt occurs the CPU issues commands to the I/O module then proceeds with its normal work until interrupted by I/O device on completion of its work.

if an interrupt occurs due to the input device, the device interrupts the CPU when new data has arrived and is ready to be retrieved by the system processor. The actual actions to perform depend on whether the device uses I/O ports, memory mapping.

if it occurs due to the output device, the device delivers an interrupt either when it is ready to accept new data or to acknowledge a successful data transfer. Memory-mapped and DMA-capable devices usually generate interrupts to tell the system they are done with the buffer.

An Interrupt relieves the CPU of having to wait for the devices, but it is still inefficient in data transfer of large amount because the CPU has to transfer the data word by word between I/O module and memory. Below are the basic operations of Interrupt:

- CPU issues read command
- I/O module gets data from peripheral whilst CPU does other work
- I/O module interrupts CPU
- CPU requests data
- I/O module transfers data

Direct Memory Access (DMA)

Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module controls exchange of data between main memory and the I/O device. Because of DMA device can transfer data directly to and from memory, rather than using the CPU as an intermediary, and can thus relieve congestion on the bus. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.

DMA increases system concurrency by allowing the CPU to perform tasks while the DMA system transfers data via the system and memory busses. Hardware design is complicated because the DMA controller must be integrated into the system, and the system must allow the DMA controller to be a bus master. Cycle stealing may also be necessary to allow the CPU and DMA controller to share use of the memory bus.

Mid term Exam 2 (30 %)

Instructions

Answer all the questions in this paper

1.(a). Draw the block diagram of a DMA controller (10 marks)

(b). explain its functioning (5 marks)

2.Describe the architecture of a shared memory multiprocessor (5 marks)

Grading Scheme

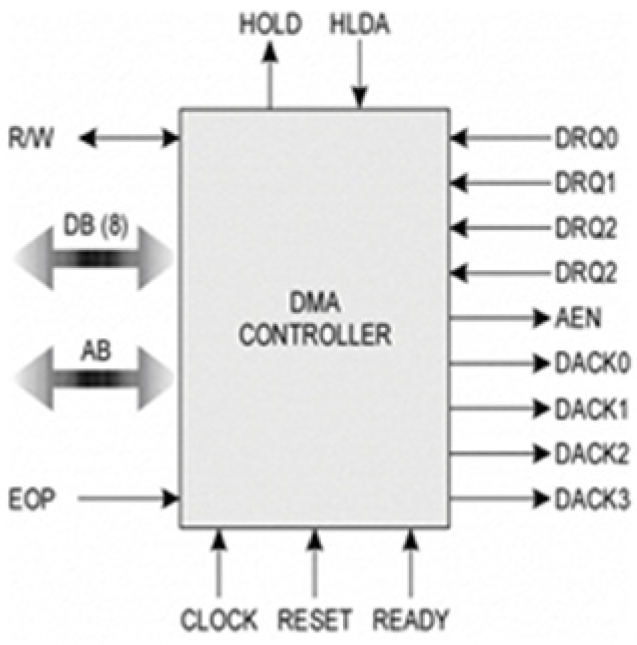
1 (a) Correct diagram 10 marks

1 (b)Correct function 2 marks each (max 10)

2 Any correct answer 2 marks each (max 10)

Feedback

1.(a). Draw the block diagram of a DMA controller



A typical DMA controller.

(b). Explain its functioning

During any given bus cycle, one of the system components connected to the system bus is given control of the bus. This component is said to be the master during that cycle and the component it is communicating with is said to be the slave. The CPU with its bus control logic is normally the master, but other specially designed components can gain control of the bus by sending a bus request to the CPU. After the current bus cycle is completed the CPU will return a bus grant signal and the component sending the request will become the master.

2. Describe the architecture of a shared memory multiprocessor

- Processors have their own connection to memory
- Processors are capable of independent execution and control
- Have a single OS for the whole system, support both processes and threads, and appear as a common multiprogrammed system
- Can be used to run multiple sequential programs concurrently or parallel programs
- Suitable for parallel programs where threads can follow different code (task-level-parallelism)

Final Exam (50 %)

Instructions

1. Answer question one and any other two
2. Question one carries 30 marks
3. Other questions carry 20 marks each

Questions

1. (a). What is Cache memory- Explain working of a Cache memory. (10 marks)

   (b). What is pipelining? Explain instruction pipelining. (10 marks)

   (c). Describe interrupt driven I/O. (10 marks)

2. Explain working of DMA data transfer. Compare it with programmed I/O and interrupt driven data transfer. (20 Marks)

3. Explain the difference between hardwired control and micro programmed control. Is it possible to have a hardwired control associated with a control memory? (20 Marks)

4. Explain the block diagram of an I/O interface unit. (20 marks)

5. Using a diagram, explain the following steps executing of a program

a. Fetch

b. Decode

c. Execute (20 marks)

Grading Scheme

Marks to be distributed as indicated against each question in the answers students will provide

Feedback

1.(a). Cache memory

Is memory that stores program instructions that are frequently re-referenced by software during operation. Fast access to these instructions increases the overall speed of the software program.

(b). Pipelining

A form of computer organization in which successive steps of an instruction sequence are executed in turn by a sequence of modules able to operate concurrently, so that another instruction can be begun before the previous one is finished.

Instruction pipelining is a technique that implements a form of parallelism called instruction-level parallelism within a single processor. It therefore allows faster CPU throughput (the number of instructions that can be executed in a unit of time) than would otherwise be possible at a given clock rate.

(c) . Interrupt driven I/O

The CPU works on its given tasks continuously. When an input is available, such as when someone types a key on the keyboard, then the CPU is interrupted from its work to take care of the input data.

2. Working of DMA data transfer

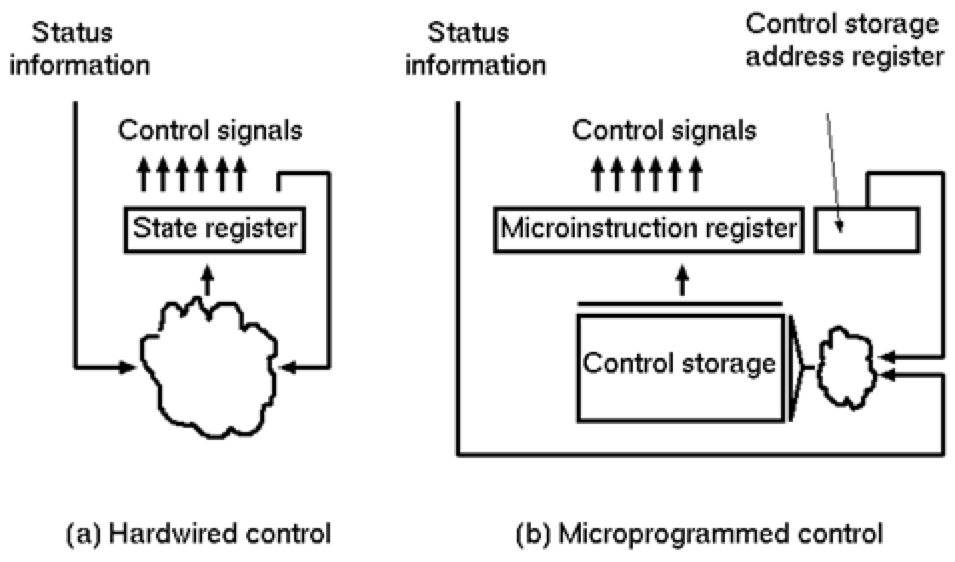
- First the CPU programs the DMA controller by setting its registers so it knows what to transfer where
- It also issues a command to the disk controller telling it to read data from the disk into its internal buffer and verify the checksum.
- When valid data are in the disk controller's buffer, DMA can begin. The DMA controller initiates the transfer by issuing a read request over the bus to the disk controller . This read request looks like any other read request, and the disk controller does not know (or care) whether it came from the CPU or from a DMA controller. Typically, the memory address to write to is on the bus' address lines, so when the disk controller fetches the next word from its internal buffer, it knows where to write it. The write to memory is another standard bus cycle.
- When the write is complete, the disk controller sends an acknowledgement signal to the DMA controller, also over the bus. The DMA controller then increments the memory address to use and decrements the byte count. If the byte count is still greater than 0, steps 2 through 4 are repeated until the count reaches 0.
- At that time, the DMA controller interrupts the CPU to let it know that the transfer is now complete. When the operating system starts up, it does not have to copy the disk block to memory; it is already there.

Comparing DMA with programmed I/O and interrupt driven data transfer

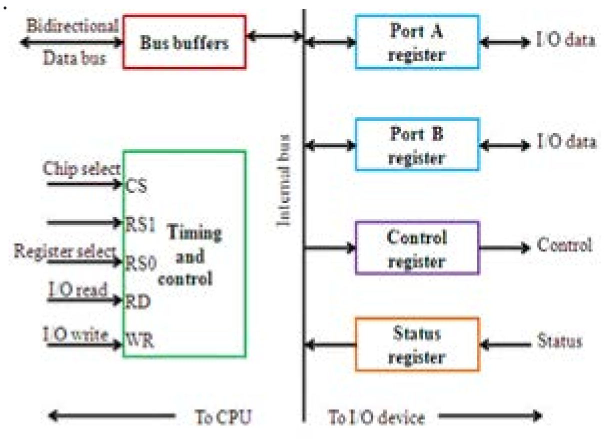
Programmed I/O (PIO) refers to data transfers initiated by a CPU under driver software control to access registers or memory on a device. while the device interrupts the CPU when new data has arrived and is ready to be retrieved by the system processor. The actual actions to perform depend on whether the device uses I/O ports, memory mapping.

3. Micro programmed control is a control mechanism to generate control signals by using a memory called control storage (CS), which contains the control signals. Although micro programmed control seems to be advantageous to CISC machines, since CISC requires systematic development of sophisticated control signals, there is no intrinsic difference between these 2 control mechanisms.

Hardwired control is a control mechanism to generate control signals by using appropriate finite state machine (FSM). The pair of "microinstruction-register" and "control storage address register" can be regarded as a "state register" for the hardwired control. Note that the control storage can be regarded as a kind of combinational logic circuit. We can assign any 0, 1 values to each output corresponding to each address, which can be regarded as the input for a combinational logic circuit.



4.



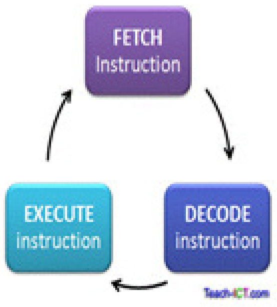
5. Is the process by which a computer or a virtual machine performs the instructions of a computer program. The instructions in the program trigger sequences of simple actions on the executing machine.

Fetch; first step the CPU carries out is to fetch some data and instructions (program) from main memory then store them in its own internal temporary memory areas. These memory areas are called 'registers'. The computer fetches the instruction from its memory and then executes it. This is done repeatedly from when the computer is booted up to when it is shut down. When the instruction has been decoded, the CPU can carry out the action that is needed. This is called executing the instruction. The CPU is designed to understand a set of instructions - the instruction set.

Decode; The next step is for the CPU to make sense of the instruction it has just fetched. This process is called 'decode'. The CPU is designed to understand a specific set of commands.

These are called the 'instruction set' of the CPU. Each make of CPU has a different instruction set. The CPU decodes the instruction and prepares various areas within the chip in readiness of the next step.

Execute;   This is the part of the cycle when data processing actually takes place. The instruction is carried out upon the data (executed). The result of this processing is stored in yet another register. Once the execute stage is complete, the CPU sets itself up to begin another cycle once more.



Module Summary

In this module, in unit 1 you learnt about advanced functional organization of computer, which included how data transfers occurs in the computer, the architecture of the microprocessor, the types of transfer of instructions within the computer and the processor and system performance of the computer.. In unit 2, Amadahl's law, Flynn Taxonom, multiprocessing and Scheduling were introduced. Also Short vector processing and multicore and multiprocessor was also learned. In unit 3, Low-level programming was introduced. This included machine and Assembly programming. In unit 4, strategies of I/O interfaces were learnt together with the various access methods, e.g. polling, interrupt and DMA. Finally in unit 5, the peripheral devices that can be connected to the computer were introduced. Wave forms representing conversions of the analog to digital data was also introduced as well as multimedia data.

Course References

John L. Hennessy and David A. Patterson; Computer Architecture: A Quantitative Approach

John Paul Shen and Mikko H. Lipasti; Modern Processor Design: Fundamentals of Superscalar Processors

M. J. Flynn; Computer Architecture: Pipelined and Parallel Processor Design

Kai Hwang; Advanced Computer Architecture: Parallelism, Scalability, Programmability

David A. Patterson and John L. Hennessy; Computer Organization and Design:The Hardware/ Software Interface

The African Virtual University Headquarters

Cape Office Park

Ring Road Kilimani

PO Box 25405-00603

Nairobi, Kenya

Tel: +254 20 25283333

contact@avu.org

oer@avu.org

The African Virtual University Regional Office in Dakar

Université Virtuelle Africaine

Bureau Régional de l'Afrique de l'Ouest

Sicap Liberté VI Extension

Villa No.8 VDN

B.P. 50609 Dakar, Sénégal

Tel: +221 338670324

bureauregional@avu.org