CST809: SECURITY ARCHITECTURE AND DESIGN

AFRICA CENTRE OF EXCELLENCE ON TECHNOLOGY ENHANCED LEARNING (ACETEL)

NATIONAL OPEN UNIVERSITY OF NIGERIA

Enhanced Learning (ACETEL)

Course Guide

Introduction

This course discusses the fundamental components of security architecture. Topics treated include Components design; Principles of secure component design; Component identification; Security Design Principle; Principle of Secure Design; Principle of Software Security, Design Principle for Protection Mechanism; Trusted Computing Base and protection mechanism; formal security models and evaluation criteria; Project on modeling secure system

Course Competencies

To develop overall architecture and is developed to provide guidance during the design of the product. It outlines the level of assurance that is required and potential impacts that this level of security could have during the development stages on the product overall.

Course Objectives

- Explain the concept of security architecture analysis.
- Value addition points in security architecture and design
- Expatiate on the forms of security models, open and distributed systems.

Working Through this Course

In order to have a thorough understanding of the course units, you will need to read and understand the contents, practice the steps by designing and implementing a prototype architecture for a PC and evaluating the performance. This 2unit course designed to cover approximately thirteen weeks, and it will require 65 hours of study.

There are two forms of assessments – formative and summative. The formative assessments are designed to support the students to learn. This include the in-text questions and self-assessment exercises and major Assignments which may be Tutor Marked Assignments (TMAs) or Computer Marked Assignments (CMAs). The summative assessment is the final exam.

Study Units

Module 1: Fundamental component of Design Architecture
- Unit 1: Architecture development and style
- Unit 2: Technological Development
- Unit 3: Performance Measure

Module 2: Instructional Set Architecture and Design
- Unit 1: Memory Location and Operations
- Unit 2: Addressing Modes
- Unit 3: Instruction Types

Module 3: Secure Component Design
- Unit 1: Processing Unit Design
- Unit 2: Memory System Design
- Unit 3: Input and Output Design

Module 4: Security Design Principle
- Unit 1: Principle of Secure Design
- Unit 2: Principle of Software Security
- Unit 3: Design Principle for Protection Mechanism
- Unit 4: Trusted Computing Base

References and Further Readings

Von Neumman, J. (1945) *First Draft of a Report on the EDVAC. Moore School of Electrical Engineering*, University of Pennsylvania.

Wang, S.P. and Ledley, R.S. (2013) *Computer Architecture and Security: Fundamentals of Designing Secure Computer Systems*. Available here

What is The Difference Between RISC and CISC Architecture? Available here

Mano, M.M. (2014) *Computer System Architecture*. 3rd Ed. Available here

Null, L. and Lobur, J. (2003) *The Essentials of Computer Organization and Architecture*. Jones and Bartlett Publishers, Sudbury, Massachusetts.

Hennessey, J.L and Jouppi, N.P. (1991) Computer architecture and technology – An evolving interaction. *Computer*, pp. 18-29, September 1991. Available here

Fernandez, E.B. (2013) *Security Patterns in Practice*. Wiley and Sons Ltd., United Kingdom.

Obaidat, M.S. and Boudriga, N.A. (2010) *Fundamentals of Performance Evaluation of Computer and Communication Systems*. John Wiley & Sons, Inc. New Jersey.

Obaidat, M.S. and Papadimitriou, G.I. (Eds.) *Applied System Simulation: Methodologies and Applications*. Springer, Norwell, MA, 2003.

Papadimitriou, G.I., Sadoun, B. and Papazoglou, C. (2003) *Fundamentals of Systems Simulation*. in: Obaidat, M.S. and Papadimitriou, G.I. (eds.) *Applied System Simulation: Methodologies and Applications*. Springer, Norwell, MA, 2003.

Law, A.M. (1999) Simulation, Modeling and Analysis. McGraw Hill Inc., New York, pp. 187-213.

Vincent, J-M. and Legrand, A. (2015). Performance measurements of computer systems: Tools and analysis. Available here

Boudec, J-Y. (2017) Performance Evaluation of Computer and Communication Systems. Available here

Memory Operations. Available: Memory Operations. Available here

Presentation Schedule

The Presentation Schedule included in your course materials gives you the important dates for the completion of tutor marked assignments and attending tutorials. Remember, you are required to submit all your assignments by the due date. You should guard against lagging behind in your work.

Assessment

Table 3 presents the mode you will be assessed.

Table 3: Assessment

S/N	Method of Assessment	Score (%)
1	Portfolios	10
2	Mini Projects with presentation	20
3	Laboratory Practical	20
4	Assignments	10
5	Final Examination	40
Total		100

Portfolio

A portfolio has been created for you tagged "**My Portfolio**". With the use of Microsoft Word, state the knowledge you gained in every Module and in not more than three sentences explain how you were able to apply the knowledge to solve problems or challenges in your context or how you intend to apply the knowledge. Use this Table format:

Application of Knowledge Gained

Module	Topic	Knowledge Gained	Application of Knowledge Gained

You may be required to present your portfolio to a constituted panel.

Mini Projects with presentation

You are to work on the project according to specification. You may be required to defend your project. You will receive feedback on your project defence or after scoring. This project is different from your thesis.

Laboratory Practical

The laboratory practical may be virtual or face-to-face or both depending on the nature of the activity. You will receive further guidance from your facilitator.

Assignments

Take the assignment and click on the submission button to submit. The assignment will be scored, and you will receive a feedback.

Examination

Finally, the examination will help to test the cognitive domain. The test items will be mostly application, and evaluation test items that will lead to creation of new knowledge/idea.

How to get the Most from the Course

To get the most in this course, you:

- Need a personal laptop. The use of mobile phone only may not give you the desirable environment to work.
- Need regular and stable internet.
- Need to install the recommended software.
- Must work through the course step by step starting with the programme orientation.
- Must not plagiarise or impersonate. These are serious offences that could terminate your studentship. Plagiarism check will be used to run all your submissions.
- Must do all the assessments following given instructions.
- Must create time daily to attend to your study.

Facilitation

There will be two forms of facilitation – synchronous and asynchronous. The synchronous will be held through video conferencing according to weekly schedule. During the synchronous facilitation:

- There will be two hours of online real time contact per week making a total of 26 hours for thirteen weeks of study time.
- At the end of each video conferencing, the video will be uploaded for view at your pace.
- You are to read the course material and do other assignments as may be given before video conferencing time.
- The facilitator will concentrate on main themes.
- The facilitator will take you through the course guide in the first lecture at the start date of facilitation

For the asynchronous facilitation, your facilitator will:

- Present the theme for the week.
- Direct and summarise forum discussions.
- Coordinate activities in the platform.
- Score and grade activities when need be.
- Support you to learn. In this regard personal mails may be sent.
- Send you videos and audio lectures, and podcasts if need be.

Read all the comments and notes of your facilitator especially on your assignments, participate in forum discussions. This will give you opportunity to socialise with others in the course and build your skill for teamwork. You can raise any challenge encountered during your study. To gain the maximum benefit from course facilitation, prepare a list of questions before the synchronous session. You will learn a lot from participating actively in the discussions.

Finally, respond to the questionnaire. This will help ACETEL to know your areas of challenges and how to improve on them for the review of the course materials and lectures.

Learner Support

You will receive the following support:

- Technical Support: There will be contact number(s), email address and chatbot on the Learning Management System where you can chat or send message to get assistance and guidance any time during the course.

- 24/7 communication: You can send personal mail to your facilitator and the centre at any time of the day. You will receive answer to you mails within 24 hours. There is also opportunity for personal or group chats at any time of the day with those that are online.

- You will receive guidance and feedback on your assessments, academic progress, and receive help to resolve challenges facing your stuides.

Course Information

Course Code:	CST809
Course Title:	Security Architecture and Design
Credit Unit:	2
Course Status:	Elective
Course Blub:	This course discusses the fundamental components of security architecture. Topics treated include Components Design; Principles of Secure Component Design; Component Identification; Security Design Principle; Principle of Secure Design; Principle of Software Security, Design Principle for Protection Mechanism; Trusted Computing Base and Protection Mechanism; Formal Security Models and Evaluation Criteria; Project on Modeling Secure System.
Semester:	First
Course Duration:	13
Required Hours for Study:	65

Course Team

Course Developer:	ACETEL
Course Writer:	Dr. Abayomi Jegede and Dr. Ahmed Aliyu
Content Editor:	Dr. Ismaila Idris
Instructional Designers:	Inegbedion Juliet O. (Ph.D) & Dr. Lukman Bello
Learning Technologists:	Mr. Awe Olaniyan Joseph

Module 1: Fundamental Components of Design Architecture

Module Introduction

This module presents the basics of architectural design of computer systems. The module explores the traditional and modern models of computer architecture alongside their structures, features, capabilities and limitations. It also explores technological advances in the areas of pipelining, caching, memory design and microprocessors and their impact on the development of improved architectures for modern computer systems. Finally, the module discusses the concept of performance measurement, its objectives and characteristics.

Unit 1: Architecture Development and Style
Unit 2: Technological Development
Unit 3: Performance Measure

Unit 1: Architecture Development and Style

Contents

1.0 Introduction

In this unit, you will learn acquire knowledge and skills to distinguish between different architecture styles and select which architecture is most suitable for a particular application. To achieve this, you will learn about the various approaches to computer architecture design, their characteristics, advantages and disadvantages.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- use different architectures and style in project development.
- select the appropriate computer architecture for different applications.

3.0 Main Content

3.1 Internal Structure of the Computer

Have you ever considered the consequences of building a skyscraper consisting of several floors and facilities which are linked together and are expected to function as a unit without having an architectural design for the project? Can you imagine the chaos and frustration that the workers will encounter? What about the time, energy and money that the project team will expend trying to figure things out? How about the high chances that that the project will eventually fail?

A computer is an electronic machine that manipulates binary coded data and instructions (or program). A computer is made up of two main parts, namely memory and central processing unit (CPU). A computer uses the memory to store data and instructions which processes. The central processing unit (usually referred to as the brain or heart of the computer) controls all processing activities of the computer system. Communication between the CPU and other components of the computer takes place via a data path known as the bus. The CPU performs the tasks of fetching, decoding and execution of program instructions on the correct data. The CPU consists of three parts: registers, arithmetic and logic unit (ALU and control unit. A register is a hardware device that stores a wide variety of data such as addresses, program counters, or data required for program execution. Registers are usually located on the processor to provide a quick access to stored data. The ALU performs arithmetic operations

(such as addition, subtraction and multiplication) and logic operations (comparisons). The control unit monitors the execution of al instructions and transfer of all information. The control unit retrieves instructions from memory, decodes the instructions, places data in the appropriate location and provides the ALU with signals which indicate the operations to carry out.

How is the internal structure of the computer designed?
Is the design of internal structure of all computers based on the same or different styles?

3.2 An Overview of Computer Architecture

Computer architecture is a branch of computer engineering which focuses of the development of computer systems or platforms based on widely accepted software and hardware technology standards. (Wang and Ledley, 2013). Computer architecture focuses on the design of computer system and compatible technologies.

There are two distinct architectural styles used for the development of computer systems. These include the traditional architecture and modern architectural styles. Computer systems developed using the traditional style were based on the Von Neumann model and its different variants. The concept of instruction set is the foundation of modern architecture styles. This forms the basis for two common types of architectures, namely Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC).

3.3 Traditional Architecture Styles

Traditional approaches use the Von Neumann model and its different variants to develop computer systems.

3.3.1 Von Neumann Architecture

The design of modern computer systems is based on Von Neumann architecture. This architecture, which consists of CPU, memory, input/output (I/O) unit and storage, performs all processing activities under the control of a stored program. These components are connected via a single bus. The computer performs all its operations as directed by a stored program. The CPU is made up of the control unit (CU) and arithmetical logical unit (Von Neumann, 1945). Most modern computers are based on the Neumann model, using a single bus to connect control, data and address circuits. Figure 1.1 illustrates the Neumann model (Wang and Ledley, 2003).

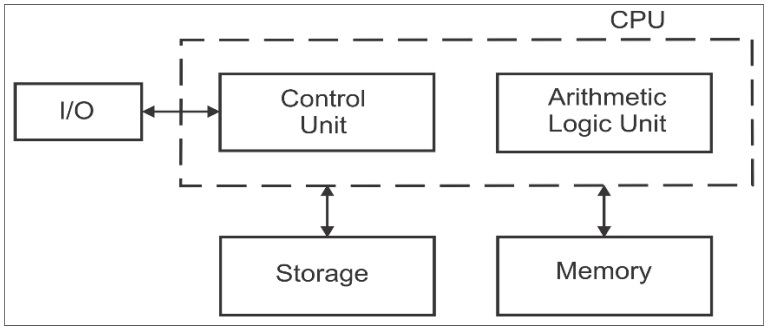

Fig. 1.1: Von Neumann model

A system bus modification of the original Neumann architecture is presented in Figure 1.2 (Wang and Ledley, 2003).

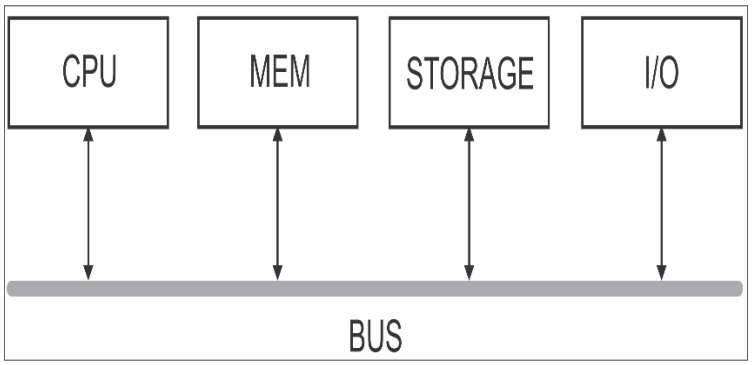

Fig. 1.2: System bus modification of Von Neumann Model

The modified model relies on the concept of direct memory access (DMA). DMA allows direct communication between the memory and I/O. This facilitates direct communication and data exchange between the memory and I/O devices without the involvement of the CPU.

The Von Neumann model does not provide a secure architecture because the CPUs, memory, I/O, auxiliary storage, and network interface are connected to a single bus consisting of the address bus, data bus and control bus. An intruder who breaks into the system can easily gain access to the system bus and take control of the computer system and its resources. The attacker can sniff information being transmitted among the various components of the computer system via the bus.

3.3.1 Modified Von Neumann Architecture

The original Neumann architecture focused primarily on the computer as a standalone system, with all the hardware, software, data and other resources resident on a single machine. The model did not envisage the concept of computer networking which allows a set of computers and other devices to be connected together for the purpose of sharing resources. The approach did not foresee the exchange of data and

information via networks and Internet, which is crucial to modern information processing. This limitation necessitated the need for the modification of the generic architecture to include the idea of a network interface. The network interface provides a means of connecting multiple computers and devices together. It also facilitates the exchange of information between the interconnected computers. Figure 1.3 illustrates a modified Von Neumann architecture, whereby a network interface (unit) is added to the system bus (Wang and Ledley, 2003) This allows the I/O unit to handle only input and output devices such as keyboard, mouse and printer.

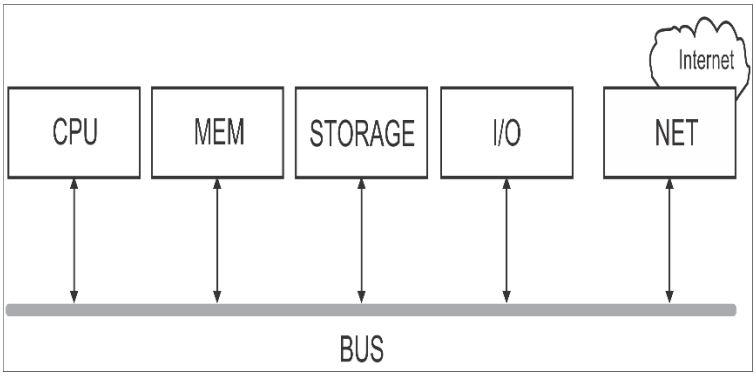

Fig. 1.3: Modified Von Neumann model with network interface

Modified Von Neumann model addressed security concerns by using a separate system bus to demarcate the network unit from other components of the computer system. A computer user must issue an explicit command for the bus controller to allow the two system buses to exchange data. In this approach, the integrity of communications is provided by a secure micro-operating system, which runs on a microprocessor and resides in a special location called the "red zone'. The booting involves the loading of a firmware image to the microprocessor memory during the booting process. A special purpose process monitors the system and resets it whenever it reaches a preset threshold. Memory protection and virtualization techniques are also used to enhance runtime security and protect against code injection.

Is there any difference in the architectural designs of first generation computers such as ENIAC and EDVAC and the designs of modern desktops and laptops?

What are the main architectural styles used in modern computers? Do they have similarities and differences?

3.4 Modern Architecture Styles

The concept of instruction set forms the basis of modern architecture styles. Typical examples of modern architectures are Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC).

3.4.1 Complex Instruction Set Computer

A compiler translates programs containing complex mathematical functions and complex subroutines into long sequences of machine language instructions. This makes compiler development a challenging and time consuming. The goal of CISC was to simplify the development of compilers which can handle complex instructions. CISC processors are designed mainly to provide more complex instruction set. CISC-based processors simplify programming by matching a machine instruction to high-level language statements. Figure 1.4 illustrates the CISC architecture.

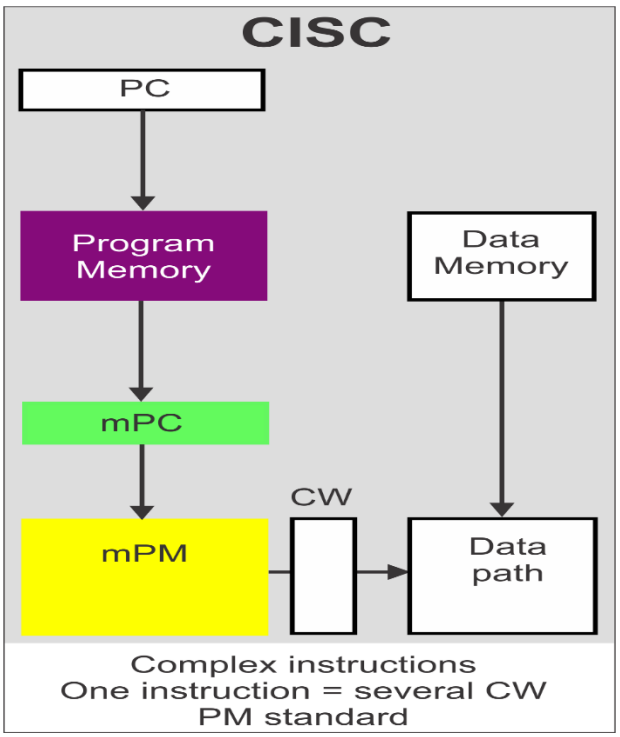

Fig. 1.4: Schematic diagram for CISC

CISC processors such as x86 Intel architectures have a large number of variable length instructions with complex layouts. The CISC architecture comprises of complex instruction set, whereby the execution of a single instruction can trigger the performance of multiple operations. For example, it is possible to perform a loop operation using a single assembly language instruction. The instructions used to manipulate register operands requires only two bytes while the manipulation of two memory addresses may require five bytes. This makes it imperative for

CISC to support variable length instructions and use variable clock cycles to execute the instructions. Many systems based of CISC architecture perform input and output operations using the memory system instead of a register file.

CISC computers store both data and instructions in the same cache memory, and use the same path for both instructions and data. The processors use a microprogram (a sequence of microinstructions) to generated control signals for executing a variety of memory resident instructions. The sequential execution of microinstruction may lead to a significant reduction in the speed of instruction execution.

Advantages of CISC Architecture

- Provides easy and cost effective implementation of microprogramming, instead of using expensive hard wired control unit.

- Uses a general-purpose hardware to execute instructions such that new instructions can be incorporated into the chip without modifying structure of the instruction set.

- Uses a fewer instructions to achiee a given task because of its support for efficient use of main memory.

- Facilitates simple implementation of the compiler by providing microprogram instruction sets that match the high-level programming language constructs.

Disadvantages of CISC Architecture

- Increasingly complex chip hardware and instruction set as new versions of CISC processors incorporates previous generation processors

.

- Reduced overall performance due varying clock times used by different instructions.

- Necessitates continuous reprogramming of on-chip hardware.

- Many functions performed by CISC architecture require complex hardware and on-chip software.

Have the limitations/problems associate with CISC been addressed? Do the new architectural style(s) offer any significant improvement?

3.4.2 Reduced Instruction Set Computer

The reduced instruction set computer (RISC) architecture uses simplified instruction set to provide an overall reduction in the time required for program execution. The instruction set of RISC processors are small in size and highly optimal for the execution of operations. Each of these instructions is designed to perform one operation. RISC architectures carry out only register to register operations. This implies that memory resident data cannot be used as operands. The instructions are all of the same length and have only a few different formats.

Figure 1.5 illustrates the structure of RISC architecture.

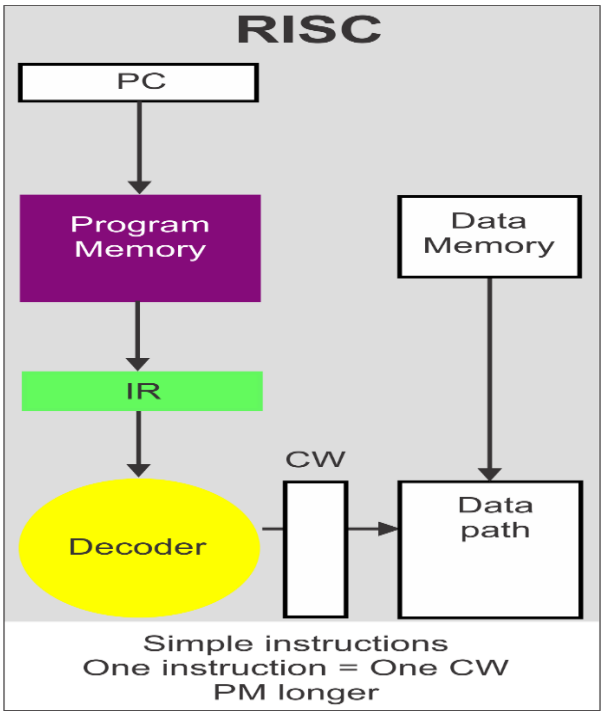

Fig. 1.5: Schematic diagram for RISC

The use of fewer and simplified instructions increases the speed of execution compared. The processor uses pipelining to optimize of each instruction which allows the simultaneous implementation of tasks such as like fetch, decode and execute. This results in an overlap of the execution cycles (fetch, decode and execute) of one or more instructions. This enables the processor to execute more instructions within a shorter period of time. RISC processors optimizes the usage of multiple registers usage to avoid frequent interactions with the memory or to reduce access time. This approach ensures that frequently used operands are placed in storage locations that can be easily accessed by the processor. The chip of most RISC based systems contains large memory cache that provides instructions with fast access to the memory.

Almost all new instruction sets for any architecture since 1982 have either been CISC or a hybrid of CISC and RISC. These systems use one clock cycle to execute most instructions, which makes it necessary to replace microprogrammed control by hardwired control in order to support faster execution of instructions. A machine cycle refers to the amount of time required to retrieve two operands from registers, execute ALU operation and store the result in a register. Generally, the speed of execution of one-cycle instructions is higher than that of microinstructions
used by the CISC processor. The use fewer instructions in RISC processors simplifies the design of the control unit.

The following are the main features of RISC architecture:

- Fewer and con6strained set of instructions.
- Uses fewer addressing modes.
- The instructions have simple and uniform format and are executed in one cycle.
- Reduced external memory access time due to the availability of larger number of registers.
- Only load and store instructions are allowed access to the memory.
- Used hard-wired control instead of micro programmed control.
- The architecture provides support for pipelining.

Power architecture, Alpha, PA-RISC, PIC, AVR, ARM are typical examples of RISC processors.

Advantages of RISC Architecture

- Its simplified instruction set provides improved performance of two to four higher times than that of CISC processors.
- Reduced instruction set enables RISC to use less chip space. This enables designers to use the same chip to implement additional functions such as floating point arithmetic units or memory management.
- The architecture offers reduced cost of each chip because it uses small sized chips which allows the integration of more components on a single silicon panel.
- Simple architecture makes the time required to design RISC processors lower than that for CISC processors.
- RISC processors use many registers to support execution of instructions at a rate faster than CISC processors.

Disadvantages of RISC Architecture

- The nature of program being executed determines the performance of a RISC processor. The processor needs to wait for the output of the current instruction result before executing the next instruction. This occurs in situations where a compiler performs poor scheduling of instruction execution.

3.4.3 Comparison Between CISC and RISC

Table 1 presents a comparison between CISC and RISC architectures.

Table 1: CISC vs RISC

CISC	RISC
Has a large number of variable-length instructions	Has a small set of fixed-length instructions
Uses more addressing modes (between 16 and 24)	Uses fewer addressing modes (3 to 5)
Provides hardware support for complex addressing modes	Uses software to synthesize complex addressing modes
Uses fewer (8 to 24) general-purpose registers and a single cache for instructions and data.	Has a large number of general-purpose registers, (between 32 and 192) with split data cache and instruction cache.
Generic CISC processors use micro-coded control memory (modern CISC processor are based on hardwired control.)	The processor is based on a hardwired control and does not use control memory.
Uses complex instructions which are executed in multiple cycles.	Consists of simple, single cycle instructions.
Minimal or no support for pipelining.	Full support for pipelining
The microprogram determines the degree of complexity of operation	The complexity of instruction execution depends on the compiler

You should consider the following factors before taking a decision on an appropriate architecture for the new computer installation.

i. The number and format of instructions in the instruction set. That is, whether the instruction set consists of large numbers of variable length instructions or small number of fixed length instructions.

ii. The choice between fewer general-purpose registers with a single cache for instructions and data, or large number of general-purpose registers and with split data cache and instruction cache.

iii. The number of addressing modes.

iv. The choice between hardware support for complex addressing modes and the use of software to synthesize complex addressing modes.

v. The choice between a processor controlled by a micro-coded control memory and that which relies on a hardwired control and without control memory.

vi. Whether the architecture will use complex instructions which are executed in multiple cycles or simple, single cycle instructions.

vii. Whether or not the proposed architecture will provide support for pipelined instructions.

viii. Whether the complexity of operation will lie with the microprogram or the compiler.

Portfolios:
Visit your computer hardware lab or a computer technician workshop nearby. Identify the components of different computers there. Also compare and contrast the internal structures and features of the motherboards of the computers there. Take some pictures of your observation and se d them to your tutor during online facilitation.

Take a visit to your local phone repair workshop. Study the internal structure of the mobile phones and tablets under repair. Compare these to those of the computers.

Lab Activity

Implement the Von Neumann machine shown in Figure 1.1.

Apparatus (1) Preassembled components Control unit, Memory and Arithmetic and logic unit. (2) External storage device (e.g. hard disk drive or flash memory). (3) I/O terminal consisting of keyboard and monitor, (4) Circuit board (5) Pieces of wires

Method: Set up your I/O to terminal by connecting the keyboard and monitor. Plug the control unit and arithmetic and logic unit on the circuit board to make the CPU. Connect the memory unit and external storage using bidirectional cables as shown in the diagram. Finally, connect the I/O terminal to the CPU to make a complete computer system. Connect the setup to a power source and boot the system. Install basic system and application software and test the system.

Discussion

Which factors will you consider if you want to carry out a new computer installation?

4.0 Self-Assessment Exercise(s)

1. Which of the following is an advantage of RISC architecture?

   A. Uses more addressing modes (typically 16 to 24)
   B. Better performance of up to two or four times than CISC processors because of simplified instruction set.
   C. Little or no support for pipelined instructions.
   D. Complexity lies in microprogram.

   Answer: B

2. The original Neumann architecture focused primarily on
   A. The computer as a standalone system, with all the hardware, software, data and other resources resident on a single machine.
   B. The number of addressing modes.
   C. Complexity.
   D. Addressing mode.

Answer: A

Lab Exercise

1. Implement a system bus modification of the original Neumann architecture as shown in Figure 1.2 (10 marks)
2. Implement the modified Von Neumann model with network interface shown in Figure 1.3.

5.0 Conclusion

Knowledge of computer architecture provides computer scientists and engineers with an in-depth understanding of the internal structure of the computer. It also enables them to know the relevance of architectural styles to the design and functionality of computer systems. This knowledge will enable system analysts, programmers and managers to select the appropriate computer for different applications.

6.0 Summary

This unit presented an overview of the internal structure of the computer system. It also provided a coverage of the concept of computer architecture development and style. It discusses the common architectures found in traditional and modern computer systems as well as their characteristics, advantages and disadvantages. This will help prospective system analysts, programmers and system implementers to determine the suitable architecture for a specific application. Unit 2 focuses on technological developments and advances in computer architecture.

7.0 References/Further Reading

Von Neumman, J. (1945). *First Draft of a Report on the EDVAC. Moore School of Electrical Engineering*, University of Pennsylvania.

Wang, S.P. & Ledley, R.S. (2013). *Computer Architecture and Security: Fundamentals of Designing Secure Computer Systems*. Available here

What Is The Difference Between RISC and CISC Architecture? Available here

Mano, M.M. (2014). *Computer System Architecture*. (3rd ed.). Available here

Null, L. & Lobur, J. (2003).*The Essentials of Computer Organization and Architecture*. Sudbury, Massachusetts: Jones and Bartlett Publishers.

Unit 2: Technological Development

Contents

1.0 Introduction

You will learn from this unit, the technological developments of different components and features of the computer system. After studying this unit, you will acquire skills for identifying recent improvements in computer architecture; as well as designing and selecting appropriate machines for implementing innovative solutions to real life problems.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to design and select suitable architecture for different real life problems.

3.0 Main Content

3.1 Advances in Computer Architecture

Can you compare cars manufacture in the 1960s with the ones which were produced recently? Have you also noticed the differences in different grades and specification of cars from the same manufacture? All the modifications and changes that have occurred over time are as a result of technological developments in the field of automobiles.

The design of a new computer is determined by how the machine is to be implemented and how it will be used (Hennessey and Jouppi, 1991). Continuous improvements in integrate circuit (IC) technology has played the most a vital role in the design modern computer architectures. The technological breakthroughs in Integrated Circuit (IC) also has a significant impact on the choice of implementation on the choice of implementation techniques for computer systems. We will explore technological developments of computer architecture in the areas of performance improvements, memory, multiprocessors and programmable solid state drive.

Are you aware that modern computers are faster and can process large amounts of data compared to older generations of computer systems? What do you think makes this possible?

3.2 Impact of Technological Development on Performance

The two major developments that have improved the performance of computer systems are pipelining and caching. The use of pipelining and cache has led to significant improvements in the performance of computer systems. Both techniques use more devices to achieve a higher throughput. These have led to developments of machines with improved processing capabilities, high speed of execution and low response time. These techniques were predominantly used in mainframes and supercomputers. Improvements in IC technology made their applications feasible in microprocessor-based computer systems.

3.2.1 Pipelining

Pipelining uses a uses an approach known as instruction-level parallelism to increase the performance of machines. (Hennessey and Jouppi, 1991). Instruction-level parallelism provides for overlapping; that is, the execution of a sequence of independent instructions simultaneously. The concept of pipelining is illustrated in the Figure 1.6 below.

Ideally, a pipelined machine should be able to complete the execution of an instruction within a clock cycle. This requires an efficient implementation of instruction level parallelism to ensure that the pipeline is full of independent instructions.

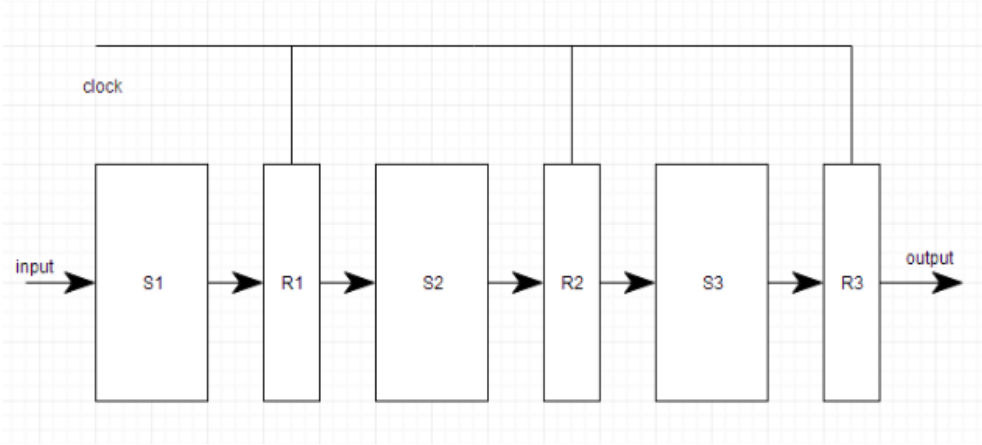

Fig. 1.6: Concept of Pipelining (studytonight.com)

An enhanced technique known as super-pipelining supports higher performance improvements than the conventional pipelining technology. A similar approach called superscalar places more than one instruction per clock into the pipeline.

3.2.2 Caching

The CPU uses a hardware cache to reduce the average cost (time or energy) to access data from the main memory. A cache is a smaller, faster processor-based memory used to store frequently used addresses and data. The purpose of cache memory is to store instructions and data that are currently used by the processor in a location that is easily accessible to the CPU. The concept of caching is depicted in the figure below.

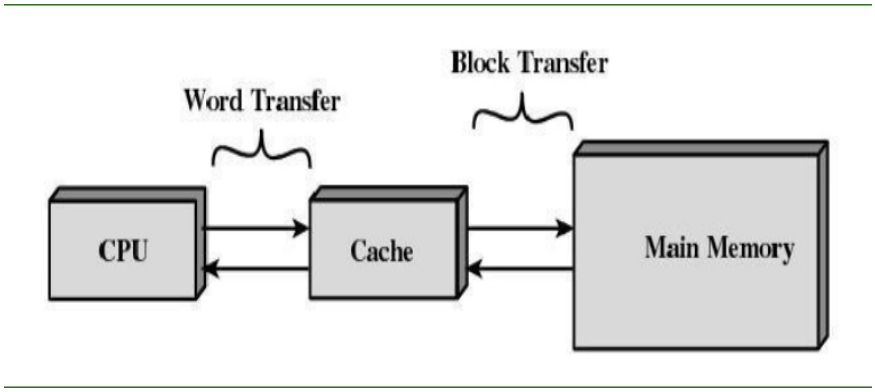

Fig. 1.7: Concept of caching (biovolttech.com)

Figure 1.7 above shows how the processor first checks the cache to see if the required data or instruction is available there. Usually, the instructions or data are placed in the cache during a previous memory fetch. The processor reads the data or instructions from the cache instead of performing more time-consuming operations to retrieve the data or instruction from the memory or other data storage devices. Caching is used to speed up computer operations and processing. The reduction in

latency between the processor and memory speeds results in significant improvements in the performance of computer systems.

Do you observe that the internal memory (RAM/ROM) available in older laptops are lower than the ones in modern laptops?

Have you also observed that there is relative decrease in prices of computer systems despite the increase in memory capacity?

3.3 Advances in Memory Technology

Advances in integration technology has led to a progressive decrease in the cost of computer memory. This had had several important effects on the design of computer. It has also minimized the emphasis placed of the importance of conserving memory. Nowadays, designers reduce design complexity by increasing processor speeds at the expense of memory costs. For example, RISC architectures have lower design complexity, but higher memory requirements and processor speeds.

Improvements in integrated circuit technology also led to breakthroughs in the dynamic random memory access (DRAM) and static random memory access (SRAM). DRAM typically contains data or program used by a computer processor. It uses a memory cell consisting of a tiny capacitor and transistor to store each bit of data. The capacitor can be in either of two states: charged (represented as 1) or discharged (represented as 0). The data on the chip may be lost if the capacitors lose their electric charge. Hence, there is a need for periodic rewriting of data in the capacitors using an external memory refresh circuit. DRAM is made of multiple individual IC chips, consisting of a large array of *storage elements* which are small sub-circuits of 1 transistor each. The sub-circuits can store billions of binary bits, which can be accessed randomly. DRAM is a commonly used in servers, workstations and personal computers (PCs).

Static RAM or SRAM) is a semiconductor memory that uses flip-flop (bi-stable latching circuitry) to store each bit. SRAM is volatile and requires continuous power supply to retain data bits in its memory. However, SRAM does not need periodic refreshing unlike DRAM. Static RAM provides faster access to data and is more expensive than DRAM. It is used for cache memory and on a video card as part of the random access memory digital-to-analog converter.

Modern computers are faster, lighter and more powerful than their older generation counterparts. What do you think is responsible for this?

3.4 Advances in Microprocessor

A multiprocessor is a component that executes instructions and carries out other tasks involved in computer processing. It is usually built on a silicon microchip and has more advanced architectural design than a conventional CPU. A pictorial representation of a microprocessor can be seen in Figure 1.8 below.

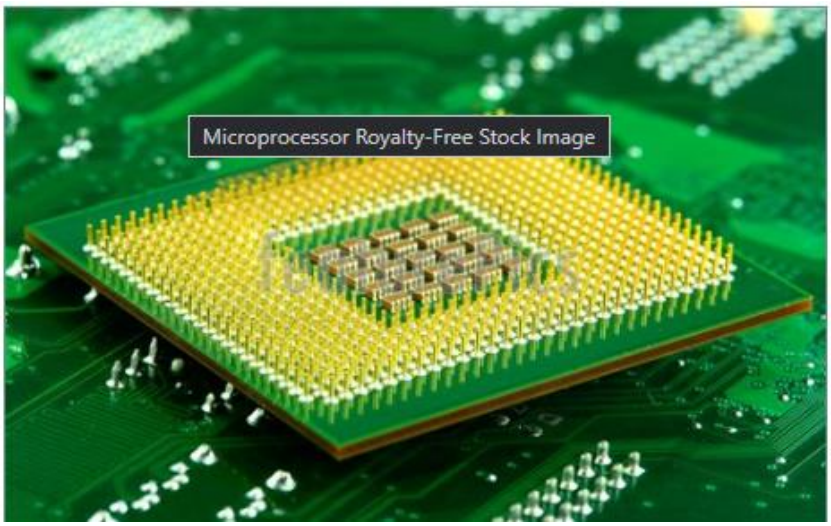

Fig. 1.8: Microprocessor

A microprocessor controls all the functions of the CPU of a computer or other digital device. The microprocessor is built on a single integrated circuit and functions as an artificial brain of the CPU.

The following are some of the functions of the microprocessor:

- It uses timing signals to control all other parts of the computer.
- It transfers data between memory and I/O devices.
- It fetches data and instructions from memory.
- It decodes instruction.
- It performs arithmetical and logical operations.
- It executes programs stored in memory.
- It performs communication among I/O devices

Advancements in microprocessor technology produced high performance computers at reduced costs. Microprocessor-based computers currently have performance rates of millions of instructions per second.
Are you aware that the hard disk of windows system is not as efficient and reliable as that of Mac computers?

Do you know that it is possible to program to control the way an external memory function?

3.5 Programmable Solid State Storage

A solid state drive (SSD), sometimes referred to as flash drives or solid-state disks, is a storage media which uses solid-state flash memory consisting of a flash controller and NAND flash memory chips to store data permanently. An SSD uses microchip to store information, unlike a hard disk drive which uses magnetic technology to store information. The SSD controller is optimized to perform high performance sequential and random data access.

Modern SSDs contain several low frequency CPUs running firmware that perform low level functions such as error-correction, wear levelling, read/write caching and encryption. Developers can use some newly discovered interfaces to program SSDs. These interfaces allow application developers to make the SSD to implement functions which would normally have been dedicated to the applications. This leads to efficient bandwidth utilization, reduced access latency and lower power consumption because of the elimination of data transfer between the computer and the SSD. This also the SSD to implement functions such as high performance data storage, data intensive computing and kernel bypass for user applications and semantic extensions.

Lab Activity

Use bidirectional buses to connect CPU, Cache and Memory to obtain the setup shown in Figure 1.7. Connect an I/O unit to the setup and simulate word transfer and block transfer modes of caching.

Discussion

A multiprocessor is a component that executes instructions and carries out other tasks involved in computer processing. What is your own opinion?

4.0 Self-Assessment Exercise(s)

1. The two major developments that have improved the performance of computer systems are:
   A. Fetching and Decoding
   B. Executing and Storing
   C. Caching and Storing
   D. Pipelining and Caching

Answer: D

2. The following are some of the functions of microprocessor except
   A. Controlling all other parts of the machine and sending timing signals.
   B. Transferring data between memory and I/O devices.
   C. Fetching data and instructions from memory.
   D. Pipelining the data

Answer: D

Assignment 1
Do you think pipelining and caching have any significant contribution to performance in modern computer systems? Provide justifications for your answer.

Mini Project
Write a seminar paper on the effects of advances in microprocessor technology on the size, cost and performance of your modern computers and mobile phones.

5.0 Conclusion

Technological advancements in performance, memory, microprocessors and solid-state storage have led to the development of high performance, lightweight and more affordable computer systems. These systems have large memory capacity, low power consumption and the ability to handle complex processing tasks.

6.0 Summary

This unit covered technological developments in the field of computer architecture. It explored advances in performance, memory technology, microprocessor and solid-state storage, and their impacts on the developments of modern computer systems. The next unit discusses relevance, objectives and techniques for evaluating the performance of computer systems.

7.0 References/Further Reading

Fernandez, E.B. (2013). *Security Patterns in Practice*. United Kingdom: Wiley and Sons Ltd.

Hennessey, J.L & Jouppi, N.P. (1991). "Computer architecture and technology – An evolving interaction." *Computer*, pp. 18-29, September 1991. Available here

Mano, M.M. (2014). *Computer System Architecture*. (3rd ed.). Available here

Null, L. & Lobur, J. (2003). *The Essentials of Computer Organization and Architecture*. Sudbury, Massachusetts: Jones and Bartlett Publishers. Available here

Wang, S.P. & Ledley, R.S. (2003). *Computer Architecture and Security: Fundamentals of Designing Secure Computer Systems*. Available here

Unit 3: Performance Measure

Contents

1.0 Introduction

In this unit, you will acquire skills for evaluating the performance of different computer architectures. To achieve this, you will learn about the concept of performance metric as well as the different techniques and parameters for evaluating the performance of computer systems

2.0 Intended Learning Outcomes (ILOs)

By the end of this, you will be able to demonstrate the use of performance measure in providing solutions to problem-based scenarios.

3.0 Main Content

3.1 What is Computer Performance Measure?

How do you determine whether a computer system is efficient, reliable and suitable for your application? What are the yardsticks you would consider when you want to choose an appropriate architecture for real-time and delay sensitive applications?

Computer performance measurement is the evaluation of the amount of work carried out by a computer system. The main goal of performance measurement is to use quantitative techniques to predict the behavior of

a computer system (Obaidat and Bouderiga, 2010). This depends on parameters such as throughput, response time and execution time of the computer system. Throughput is measured in terms of the amount of data processed or transferred by a computer system over a period of time. The amount of data processed or transferred depends on factors such as the speed of the CPU, memory capacity, operating systems performance and the tool used for the measurement. Response time refers to the total time a computer takes to respond to a request for service such as memory read/write, disk I/O, database query or access to a web page. The execution time is a measure of how long it takes the CPU to execute a program instruction. This includes the actual amount of time taken to process the instruction as well as the time for carrying out necessary system services required for the execution of the instruction.

The three major techniques used for evaluating performance of computer systems are (a) analytical modeling, (b) simulation and (c) measurement and testing. The arrangement of these methods is based on increasing order of cost and accuracy. Analytical models are the cheapest to use and are the least accurate. Simulations are more flexible, accurate and reliable, but it takes a lot of time to derive the model, design and code the simulator and verify and validate the model. Measurement is the most accurate and reliable, but it consumes a greatest amount of time and effort.

3.1.1 Objectives of Computer Performance Measurement

The major objectives of performance evaluation are as follows (Obaidat and Papadimitriou, 2003; Papadimitriou et al, 2003; Jain 1991):

- To compare alternative system designs: This is a comparison of the performance of different hardware or software systems or component designs and choosing the best alternative for a specific application. This is used for a quantitative evaluation and determination of the best configuration for the actual operating environment.
- Procurement: Performance evaluation assists in determining the system that is most cost-effective for a specific application.
- Capacity Planning: Data centre administrators and managers use performance evaluation to ensure that adequate resources are available to meet system objectives without compromising performance goals.
- System Tuning: This is concerned with finding a set of parameter values such as disk and network buffer sizes that will produce optimal system performance.
- Performance Debugging: The goal here is to determine why a system or an application fails or meet performance expectations.

Performance analysis helps to identify the major cause of the problem so that the system administrator can remedy the situation.

- Set expectation: Performance measurement enables system users to set appropriate expectations of the minimum performance requirements that should be met by the system.
- Recognize relative performance: The goal is to achieve a quantitative evaluation of change in performance of a system relative to previous systems, customer expectations or competing systems.

Which relevant characteristics of the computer would you measure to determine whether the computer has good performance or not?
How would you ensure that a measured characteristic gives an unbiased result irrespective of the kind of computer involved or the operating environment?

3.2 Metrics

A metric is a criterion used to measure the performance of computer systems. Metrics usually have an underlying relationship with the accuracy, speed, availability and reliability of the operations or services carried out by the system. Performance analysts measure the following basic features of a computer system:

i. the number of times an event occurs,
ii. the amount of time it takes for an event to occur, and
iii. the size of some parameter.

The values obtained from such measurements can be used to derive the actual value that a performance analyst wishes to use to describe the system. The actual value is referred to as performance metric.

Computer performance metrics are divided into the following main categories [Obaidat and Papadimitriou (2003); Papadimitriou et al, 2003]:

- Higher better metrics (HB): The higher the value of the metric, the better is the performance of the system. Productivity is an example of HB metric.

- Lower better metrics (LB): Metrics which have lower values indicate better performance than those with higher values. An example of LB metric is response time.

- Nominal better metrics (NB): This requires that the value should not be too high or too low. It is desirable that metrics in this category have values between 0.5 and 0.75. A typical example of NB metric is utilization.

Availability and reliability are other measures of performance used by analysts. Availability is measured in terms of mean time to failure (MTTF) and mean time between failure (MTBF) [Obaidat and Papadimitriou (2003); Papadimitriou et al, 2003, Law, 1999]. MTTF is a measure of the duration a device or component is reasonably expected to function before a failure occurs. It is computed as the total hours of operation, divided by the total number of devices being tracked. MTBF is defined as the actual time between two successive failures. It is calculated as the sum of MTTF and mean time to repair (MTTR). That is, the total time it takes a device to experience failure and for that failure to be repaired.

3.2.1 Desirable Characteristics of Computer Performance Metric

The desirable characteristics of performance metrics include [Vincent, J-M. and Legrand, A. (2015):

- Reliability: The performance metric should indicate that A has a better performance than B if a system A always performs better than a system B.

- Repeatability: The repeated performance of the same experiment should produce the same value of the metric.

- Consistency: This implies that different systems and different configurations of the same system have the same units of a metric and same precise definition for the metric.

- Linearity: An improvement in the actual performance of the machine should result in a corresponding increase in the value of the metric.

- Ease of measurement: A metric should be easy to measure to ensure that it is correctly determined and to encourage people to actually use it will actually use it.

- Independence: A metric should be defined in such a way that is not in favour of any particular system.

How would you measure the processing speed and power of the computer?

What does a 1.8GHz value mean in a PC configuration?

3.3 Measuring Computer Speed and Processing Power

Million **i**nstructions per second (MIPS) is a well-known (but outdated) parameter for measuring the speed and processing power of a computer. MIPS measures gives a rough estimate of the number of machine instructions that a computer can execute in one second. MIPS does not give an accurate measure of computer performance because different instructions require different amounts of time for execution. Complex instructions require more time, while simple instructions will execute within a shorter period of time.

Moreover, there is no standard method for measuring MIPS. While the CPU measures MIPS only in terms of processor speed, real life applications consider other factors such as the speed of transfers between the processor and I/O devices. Thus, a machine with a high MIPS rating, may not outperform a low MIPS-rated counterpart even if they execute the same application. Despite these limitations, MIPS provides a rough idea of the speed of a computer. For example, an older computer such as IBM PC/XT has a rating of ¼ MIPS, while Pentium-based PCs are rated 100 MIPS and above.

Nowadays, a computer's processing speed is expressed in terms of the clock speed, measured in cycles per second. One cycle/second equals to 1 hertz. This implies that a processor which executes two thousand million (or two billion) cycles per second has a clock speed of 2 gigahertz (GHz). A CPU with high will execute instructions faster than a CPU with low i clock speed.

 **Discussion**

Based on your experience from this course, why do you think computer measurement is important?

4.0 Self-Assessment Exercise(s)

1. Which of these best describe computer performance measurement?
   A. It is the evaluation of the amount of work carried out by a computer system.
   B. Systems or component designs and deciding the best alternative for a specific application.
   C. The execution time is defined as the amount of time taken by the CPU to execute a program instruction.

D. The basic features of a computer system that performance analysis.

Answer: A

2. MIPS stands for
   A. Mili-Instructions Per Seconds
   B. Million Instructions Per Second
   C. Message Instruction Per Second
   D. Million Instructions Per Seconds

Answer: B

5.0 Conclusion

Computer performance measurement enables computer architect and designers to evaluate the speed, accuracy, reliability and reliability of a proposed computer architecture and design. It also helps system implementers to assess the performance of different hardware or software systems or component designs and choose the best alternative for a specific application.

6.0 Summary

In this unit, you have learnt the goal and objectives of performance measure. This unit also provided you with the knowledge of computer performance metric, its desirable characteristics and the criteria for choosing a suitable metric for evaluating computer architecture and design. The unit also introduced you to the parameters used to measure computer speed and processing power.

7.0 References/Further Reading

http://polaris.imag.fr/arnaud.legrand/teaching/2013/EP_02_measurements.pdf

https://www.researchgate.net/publication/325551949_Performance_Evaluation_of_Computer_and_Communication_Systems

Obaidat, M.S. & Boudriga, N.A. (2010). *Fundamentals of Performance Evaluation of Computer and Communication Systems*. New Jersey: John Wiley & Sons, Inc.

Saastamoinen J, Khan S, Tiensyrjä K & Taipale T (2011). "Multi-threading support for system-level performance simulation of multi-core architectures." Proceedings of the 24th International Conference on Architecture of Computing Systems, Como, Italy, 169–177.

Saastamoinen, J. & Kreku, J. (2011). "Workload model generation for system-level design exploration." Proceedings of the Conference on Design and Architectures for Signal and Image Processing (DASIP), 2011.

Throughput Definition. Available here

Module 2: Instructional Set Architecture and Design

Module Introduction

This module focuses on the architecture and design of instruction set for the computer. Topics covered in this module include memory location and operations, addressing modes and various types of computer instructions.

Unit 1: Memory Location and Operations
Unit 2: Addressing Modes
Unit 3: Instruction Types

Unit 1: Memory Location and Operations

Contents

1.0 Introduction

In this unit, you will learn the fundamentals of computer memory, address space and the interaction between the memory interacts and other components of the computer. After studying this unit, you will acquire skills for identifying, designing and selecting appropriate memory structure and device for implementing computer architecture.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to justify the use of memory location and operations in design.

3.0 Main Content

3.1 Computer Memory: Definition and Types

Where and how does the computer store data and instructions it processes? How do we keep the result of data processed by the computer permanently?

A computer memory is a device used for temporary or permanent storage of information. Primary memory and secondary memory are the two basic types of computer memory.

3.1.1 Primary Memory

Primary memory contains data and instructions which are currently being processed by the computer. This makes it difficult for a computer to operate without a primary memory. Memory devices built using semiconductor or integrated circuits and are accessed by operating systems, software, and hardware. Primary memory may be volatile or non-volatile. A primary memory is said to be volatile if the data stored in it remains as long as the computer is connected to a power source. A volatile memory loses its data once the power supply to the computer system is switched off or interrupted. Random access memory (RAM) is an example of volatile primary memory. Non-volatile primary memory provides data permanence even in the event of power loss interruption. Data stored in a non-volatile primary emory such as read only memory is preserved even when the computer is disconnected from a power source. Read only memory (ROM) is a non-volatile memory which does n6ot lose its contents even when the computer is disconnected from a power source.

3.1.2 Secondary Memory

Secondary memory or auxiliary storage are non-volatile and preserves stored data even when there is power loss or interruption. Secondary memory such as the hard disk may be located internally within the computer system. Other examples of secondary storage such as USB memory sticks, CD, DVD and Blu-ray discs are usually external to the system. Secondary storage support large amount of data and enables permanent data storage.

Computer systems use three main types of secondary memory:

i. Solid state storage devices, such as USB memory sticks.
ii. Optical storage devices, such as CD, DVD and Blu-ray discs.
iii. Magnetic storage devices, such as hard disk drives.

The main difference between primary and secondary memory is that primary memory is accessible to the CPU via the system bus, while access to secondary memory takes place via the input/output channels.

The following table provides a comparison between primary and secondary memory.

Table 2: Primary Memory vs Secondary Memory

Parameter for comparison	Primary memory	Secondary memory
Accessibility	Directly accessible to the CPU	Not directly accessible to the CPU
Volatility	Volatile	Non-volatile
Means of production	Semiconductors	Magnetic or optical materials
Modes of access	Via the system bus	Via the I/O channels'
Size	Small	Large
Cost	Less expensive	More expensive
Location	Internally (on the processor board	Externally (off the processor board or outside of the computer

3.1.3 Cache Memory

A cache is a special type of memory used to store data ad programs currently being executed by the computer or those frequently used by the processor. A cache memory consists of registers which are located close to the processor. It provides faster access to data and instructions compared to main memory. The use of cache reduces the latency between processor and memory speeds. This implies a reduction in the amount of time the CPU spends waiting for data and instructions from the memory.

Why do you think a town or city is divided into streets and each street assigned a name? Why do you think each house in your city is assigned an address?

Which activities does the computer carry out to retrieve data and instructions in its memory during program execution?

3.2 Memory Location and Address Space

A memory location is any space within the memory that is used to store data. Each location in the memory is referenced by an address, which acts as a pointer to that location. A memory address is a unique identifier which the CPU or a device uses to keep track of stored programs or data. Memory addresses are ordered and fixed-length sequences of digits which are usually represented and manipulated as unsigned integers. The size of each memory address is a machine word, which represents the amount of memory used by the CPU to store numbers in cache, registers or RAM. A mmory word may be 16 bits (2 bytes), 32 bits (4 bytes) or 64 bits (8 bytes) depending on the configuration of CPU.

An address space is a range of valid memory addresses that a program or process can access (for example, 0 to 4GB on a 32-bit machine). This refers to the actual physical memory that is available to a program or process. Machines based on older hardware architectures had one single address space, shared by all programs and the operating system. This memory space is also the actual physical memory. Modern computers are byte-addressable. That is, each RAM cell can hold binary values up to the size of one byte. The computer segments data that are more than one byte into multiple bytes and stores them in a consecutive memory addresses.

3.3 Basic Memory Operations

Figure 2.1 illustrates the instruction execution cycle, which presents the details of the two major operations on memory. These operations are:

- Fetch: Used to retrieve a value from a memory location referenced by a unique address. The format is fetch(address); the fetch keyword refers to the function that is to be executed, while the address is the argument. The operation produces an output without changing the value stored at that memory location.

- Store: Places a new value into the memory cell specified by the address. Usually written as store(address, value); the store designates a processor specific operation; while the two arguments (address and value) identify the actual value to be placed in the given address.

Access to the memory may be sequential or random. Sequential access means the CPU locates stored data or programs in a predetermined, ordered sequence. Magnetic tapes use sequential access method. Random access, on the otherhand, allows the CPU to access data in any location directly without requiring that other locations be accessed first. Hard disk

and optical disks (such as CD, DVD or blue ray) are examples of random access memory devices.

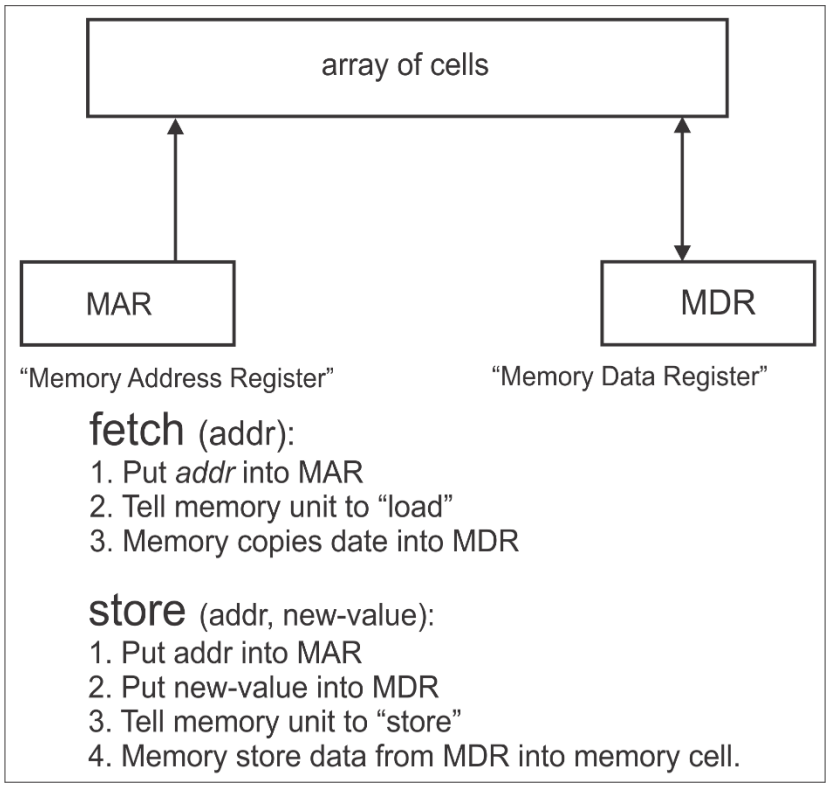

Figure 3.1 Instruction execution cycle

See source link

Discussion

Which of the considerations will determine your choice of primary and secondary storage when you are designing architecture or assembling a machine?

4.0 Self-Assessment Exercise(s)

1. A computer memory is any physical device used to store information _____or _____

   A. Temporarily or otherwise
   B. Physically or spiritually
   C. Temporarily or permanently
   D. Permanently or physically.

Answer: C

2. Secondary memory is also referred to as

- Auxiliary storage or non-volatile.
- Volatile storage
- Compact storage
- Hard disk

Answer: A

5.0 Conclusion

Data and programs must be stored in unique memory locations prior to processing by the CPU. The bus determines a fixed number of CPU memory addresses assigned based on the processing requirements of the CPU. The CPU then accesses and manipulates data in physical memory in individual segments. The allocation of memory addresses takes place during the boot process in order to assign physical addresses to data and instructions.

6.0 Summary

In this unit, you have learnt about the structure of the computer memory and how the computer allocates memory addresses during program execution. The unit also introduced you the two modes of memory access and the basic operations which the computer performs on memory. In the next unit, you will learn how to use addressing modes to access operands in specified memory and register locations.

7.0 References/Further Reading

https://www.wiley.com/en-us/Security+Patterns+in+Practice%3A+Designing+Secure+Architectures+Using+Software+Patterns-p-9781119998945

Mano, M.M. (2014) *Computer System Architecture*. 3rd Ed. Available here

Hennessey, J.L & Jouppi, N.P. (1991). "Computer Architecture and Technology – An Evolving Interaction." *Computer*, pp. 18 -29, September 1991. Available here

Mano, M.M. (2014). *Computer System Architecture*. (3rd ed.).Available here

Memory Operations. Available here

Wang, S.P. & Ledley, R.S. (2003) *Computer Architecture and Security: Fundamentals of Designing Secure Computer Systems.* Available here

Unit 2: Addressing Modes

Contents

1.0 Introduction

In this unit, you will acquire the skills for writing assembly program instructions for different addressing modes used by modern computers. To achieve this, you will learn the assembly language convention for different addressing modes and the corresponding register transfer operations which they implement.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to perform programming tasks using addressing modes, present addressing modes and writing sample segment codes.

3.0 Main Content

How does the computer locate operands in its memory?
Does it use the same or different methods to find where the operands are stored?

3.1 Basic Terminologies

- Address: An address is basically a reference to a place where a person or an object resides. The link will provide further understanding of what an address is about. An address may refer to any of the following:

- In computer data storage, an address is a reference to a location where data may be accessed.

- Computer networks refer to address as Internet Protocol (IP) address or other unique network location.

- An Internet address is a synonym for a web address that references a specific resource on the Internet.

- An address my also refer to the physical location of a building.

A computer address usually refers to a location of data in memory or on an external storage device. It is a mean used by the CPU or devices for tracking data stored in the computer.

- **Addressing:** In basic computer operation, addressing is a method used by the CPU to identify the location where data or information is stored. A computer network uses addressing to identify sending and receiving devices on the network.

- **Addressing mode:** The term addressing modes refers to the method used to specify the operand part of an instruction. The instruction code contains an actual data value or the address of the result/operand. An addressing mode uses information contained in registers and/or constants in a machine language instruction or other location within the computer to calculate the effective memory address of an operand.

3.2 The Purpose of Addressing Modes

Computers use addressing mode techniques to accomplish either or both of the following purposes (Mano, 2014):

- To provide users with programming versatility using facilities such as pointers to memory, counters for loop, indexing of data and program relocation.

- To minimize the number of used to store the operands in an instruction.

- To give assembly language programmers the flexibility to write programs that can execute a large number of instructions within a short period of time.

An understanding of the concept of instruction format will provide you with a good knowledge of the addressing modes presented in this unit. The instruction format defines the various fields contained in instructions. The figure below represents the format of assembly language instructions.

Opcode	Mode	Address

The opcode specifies the operation to be executed, while mode field helps to locate the operands. The operand refers to the actual data or the memory location used to store the data to be manipulated.

Is it possible for an instruction to have an empty address field?
How such instruction determine operand to manipulate or process?

3.3 Zero Address Modes

The opcode defines the operation carried out on the address field of the instruction. However, instructions in zero address modes do not have address field at all. Zero address modes are of the categories:

- Implied mode
- Immediate mode.

3.3.1 Implied mode

This mode gives an explicit specification of the operand in the definition of the instruction. A typical example is the instruction, "complement accumulator" (CMA) which has an empty address field. The instruction computes the 1's complement of the current contents of the accumulator. If the accumulator contains a binary value 0101 1010, CMA will change that value to 1010 0101. Generally, implied mode is found in all instructions that use the accumulator to reference registers.

3.3.2 Immediate Mode

The address field of this type of instruction contains the actual operand, rather than a reference to a register or memory location. This mode

manipulates data values based on the operation specified in the instruction. This addressing mode is used to assign numeric values to a register or memory location. The assembly language command

LD #NBR

will transfer a numeric value, NBR into the accumulator; that is, $AC \leftarrow \#NBR$. For example, LD 1000 will initialize the accumulator with the actual value 1000.

Which group of instructions do not contain in their address fields?
How does this group of instructions work?

3.4 Non-zero Address Modes

The address field of non-zero mode instruction is not empty. It usually contains reference to a register or memory location. The operation specified in the instruction manipulates the contents of the register or memory location in the address field.

3.4.1 Register Mode

A register mode instruction uses its address field to identify a register in CPU whose content is to be manipulated by the operation specified in the instruction. Generally, the processor registers are used to store the operands contained in the instruction. For example,

LD R1

will transfer the data contained in register R1 into the accumulator; that is, $AC \leftarrow R1$.

3.4.2 Register Indirect Mode

Here, the instruction references a CPU register which contains the address of the operand in memory. The register stores the location of the operand rather than the actual operand to be manipulated. The instruction

LD (R1)

loads the accumulator with an operant whose memory address is stored in R1. That is, $AC \leftarrow M[R1]$.

The programmer uses a previous instruction to store the address of the operand in a register. The memory address is specified by referencing the register. This technique is efficient because the number of bits used to select a register is lower than the number of bits required for direct specification of a memory address.

3.4.3 Autoincrement or Autodecrement Mode

In this mode, the register is incremented or decremented after (or before) using its value to access the memory. The instruction

LD (R1)+

increments the register R1 after using its contents to reference the memory. That is, $AC \leftarrow M[R1],\ R1 \leftarrow R1 + 1$.

This type of instruction is used to increment or decrement the register used to store a reference to a table of data in memory. This enables a programmer to increase or reduce the value in the register after every access to the table. The high importance of autoincrement and autodecrement operations makes computers designers provide a special mode for automatic increment or decrement operations after data access.

3.4.4 Direct Addressing Mode

The effective address of the operand is given directly in the address field of the instruction. The instruction

LD ADR

loads the accumulator with an operand whose location in memory is specified by ADR. That is, $AC \leftarrow M[ADR]$. The address, ADR refers to memory location which contains the operand that load instruction manipulates. Direct mode is used in branch type instruction to specify the actual address of the location to which the control unit should transfer program execution.

3.4.5 Indirect Addressing Mode

Here, the address field contains a reference to the memory address location which holds the actual address of the operand. The instruction

LDA @ADR

directs the control unit to use the address part, ADR to access the memory location which contains the actual address of the operand to be transferred into the accumulator. That is, $AC \leftarrow M[M[ADR]]$. This instruction performs memory reference twice; firstly, to access the location containing the actual address of the operand and secondly to use the actual address to retrieve the operand from the memory.

3.4.6 Relative Addressing Mode

The actual address of the operand is sum of the contents of the program counter and the address part of the instruction. This is written in assembly language format as

LD $ADR

which is implemented as $AC \leftarrow M[PC + ADR]$. The operand (usually a signed 2's complement value) may either be positive or negative. The memory location of the actual address is calculated based on the address of the next instruction. For example, if the program counter points to location 500 and the operand of the instruction contains 44. The fetch phase reads the instruction at location 500 from memory and increments the program counter (by one) to 501. The actual address is calculated as 501 + 44= 545. This translate to 44 memory locations after the address of the next instruction. Instruction formats based on relative addressing produces shorter address field because they require fewer number of bits compared to instructions that specify entire memory addresses.

3.4.7 Indexed Addressing Mode

This mode calculates the effective address by adding the contents of a special register in the CPU known as index register, to the address part of the instruction. The instruction

LD ADR(X)

is implemented as $AC \leftarrow M[ADR + XR]$. This instruction loads the accumulator with an operand whose effective address is the sum of index register and the value specified in the address part of the instruction. The operand field designates the start address of an array of data in memory. The memory stores each array operand in locations relative to the start address. The index value contained in the index register designates the interval between the start address and the address of the operand. The same instruction can be used to access any operand in the array once the index register is initialized with the correct index value. The control unit accesses consecutive operands by incrementing index register.

3.4.8 Base Register Addressing Mode

This is a technique which adds the contents of a base register, *BR* to the operand provided in the instruction in order to obtain the actual address of a data. The address field contains an offset relative to this base address. The only difference between this technique and index addressing mode is that it uses an offset register known as a base register, rather than an index register. The two modes differ based on usage rather ho they calculate the actual address. The instruction

LD ADR(X)

is implemented as $AC \leftarrow M[ADR + BR]$. Computers use base register addressing modes to implement program relocation in memory. The addresses of instructions must reflect the change in position when control transfers programs and data (in multiprogramming systems) from one memory segment to another. The use of a base register ensures there is

no need to change the displacement values of instructions. The value of the base register is updated by the control unit to indicate the where a new memory segment starts from.

Addressing modes such as relative mode, index mode and base address mode computes the actual address by computing the sum of the value stored in a specific CPU register and the operand field of the instruction. That is,

effective address = operand part of instruction + value in CPU register

The special-purpose CPU register which may be the program counter, an index register or base register determines the addressing mode for the application.

Table 3 uses the LOAD instruction to summarize the eight addressing modes and the corresponding register transfer operations (Mano, 2014).

Table 3: Addressing Modes

Mode	Assembly Mnemonic	Register Transfer
Direct addressing	LD ADR	AC ← M[ADR]
Indirect addressing	LD @ADR	AC ← M[M[ADR]]
Relative address	LD $ADR	AC ← M[PC + ADR]
Immediate address	LD #NBR	AC ← NBR
Index addressing	LD ADR(X)	AC ← M[ADR + XR]
Register addressing	LD R1	AC ← R1
Register indirect	LD (R1)	AC ← M[R1]
Autoincrement	LD (R1) +	AC ← M[R1], R1 ← R1+1

The table presents the assembly mnemonic and the actual implementation for each instruction. ADR, NBR, X, Rl and AC represent an address, a number (or operand), an index register, a processor register, and the accumulator register respectively. The @ character is the symbol for an indirect address, while the $ sign preceding an address indicates that the address is relative to the program counter PC. In immediate mode instruction, the # character which precedes the operand indicates that the operand is a numeric value. An indexed mode instruction places a register in parentheses after the symbolic address. The register indirect mode encloses the reference to the register containing the memory address in parentheses. The autoincrement mode places a positive sign after the register in parenthesis. The autodecrement mode uses a negative sign instead. You will need to know the available instructions and the addressing modes before you can write assembly language programs for a particular computer.

More details on the characteristics and applications of the various address modes are presented as follows.

Data register direct
The source or destination of data is referenced by data register (D0-D7).

For example, if you append '.B' to MOVE, the operation will affect only the low byte of the destination register.

INSTRUCTION	MOVE.B D0,D3			
	MEMORY		REGISTER	
	ADDRESS	CONTENTS	NAME	CONTENTS
BEFORE			D0 D3	10204FFF 1034F88A
AFTER			D0 D3	10204FFF 1034F8FF

Address register direct
The destination of data is referenced by the address register (A0-A7). This mode can only specify word or longword operands. A signed operation is performed on a word operand to fit the register. The example shows the transfer of the
contents of A3 into A0.

INSTRUCTION	MOVEA.L A3,A0			
	MEMORY		REGISTER	
	ADDRESS	CONTENTS	NAME	CONTENTS
BEFORE			A0 A3	00200000 0004F88A
AFTER			A0 A3	0004F88A 0004F88A

Absolute short
A one extension word of the bits 16-23 of the instruction specifies the source or destination address. The full address is obtained by performing a signed operation of the 16-bit short address

For example, the source address is specified using the immediate mode, destination is specified using absolute short address. The operation is '.W', such that a signed operation extends the source address to two bytes

INSTRUCTION	MOVE.W #$1E,$800			
	MEMORY		REGISTER	
	ADDRESS	CONTENTS	NAME	CONTENTS
BEFORE	000800 000801	12 34		
AFTER	000800 000801	00 1E		

Absolute long
This mode uses two extension words to specify source or destination address in an instruction.

For example, the source address is specified using the immediate mode, destination is specified using absolute short address n. The operation modifies only one memory byte and is specified as '.B'.

INSTRUCTION	MOVE.B #$1E,$8F000			
	MEMORY		REGISTER	
	ADDRESS	CONTENTS	NAME	CONTENTS
BEFORE	08F000	FF		
AFTER	08F000	1E		

Register indirect
An address register references the source or destination address of the operand.

The example below shows an instruction which transfers a longword from data register D0 to the memory location referenced by address A0.

INSTRUCTION	MOVE.L D0,(A0)			
	MEMORY		REGISTER	
	ADDRESS	CONTENTS	NAME	CONTENTS
BEFORE	001000 001001 001002 001003	55 02 3F 00	A0 D0	00001000 1043834F
AFTER	001000 001001 001002 001003	10 43 83 4F	A0 D0	00001000 1043834F

Post-increment register indirect

This technique increments the address register by the number of bytes transferred after a read or write operation.

byte: [Ai]←[Ai]+1 g word: [Ai]←[Ai]+2 g longword: [Ai]←[Ai]+4

INSTRUCTION	MOVE.W (A5)+,D0			
	MEMORY		REGISTER	
	ADDRESS	CONTENTS	NAME	CONTENTS
BEFORE	001000 001001 001002 001003	45 67 89 AB	A5 D0	00001000 0000FFFF
AFTER	001000 001001 001002 001003	45 67 89 AB	A5 D0	00001002 00004567

Pre-decrement register indirect
Uses a '-' sign before (Ai)
A technique which decrements the address register by the number of bytes transferred after a read or write operation.

byte: [Ai]←[Ai]+1 g word: [Ai]←[Ai]+2 g longword: [Ai]←[Ai]+4

INSTRUCTION	MOVE.W D0,(A7)			
	MEMORY		REGISTER	
	ADDRESS	CONTENTS	NAME	CONTENTS
BEFORE	001000 001001 001002 001003	10 12 83 47	A7 D0	00001002 00000143
AFTER	001000 001001 001002 001003	01 43 83 47	A7 D0	00001000 00000143

Register indirect with offset

Uses a 16-bit signed offset to extend the memory word specified in the instruction. It calculates the actual address of the source or destination by appending a sign-extended offset to the address register. For example, the effective address, n is computed by adding 6 to the address register. The content of the address register remains the same.

INSTRUCTION	MOVE.W 6(A0),D0			
	MEMORY		REGISTER	
	ADDRESS	CONTENTS	NAME	CONTENTS
BEFORE	001026 001027	07 BF	A0 D0	00001020 00000000
AFTER	001026 001027	07 BF	A0 D0	00001020 000007BF

Register indirect with index and offset

A variation of register indirect mode which uses an index register and an 8-bit signed offset (displacement) n. The effective address is the sum of the displacement, the value in the index register and the value in the address register.
For example, n =$10+$100A+$2=$101C

INSTRUCTION	MOVEA $10(A0,D0.L),A1			
	MEMORY		REGISTER	
	ADDRESS	CONTENTS	NAME	CONTENTS
BEFORE	00101C 00101D	EF 10	A0 A1 D0	0000100A 00000000 00000002
AFTER	00101C 00101D	EF 10	A0 A1 D0	0000100A FFFFEF10 00000002

PC-relative with offset

This approach calculates only the effective source address by adding a 16-bit displacement to the contents of the PC. It uses a position-independent code, which enables the assembler to calculate the displacement by subtracting PC from label.

INSTRUCTION	MOVE.W COUNT(PC),D5			
	MEMORY		REGISTER	
	ADDRESS	CONTENTS	NAME	CONTENTS
BEFORE	001026 001027	AB CD	PC D5	00001000 12345678
AFTER	001026 001027	AB CD	PC D5	00001004 1234ABCD

PC-relative with index and offset

This mode calculates the relative address to the PC by adding an 8-bit signed displacement to an index register.

INSTRUCTION	MOVE.B TABLE(PC,D0.W),D0			
	MEMORY		REGISTER	
	ADDRESS	CONTENTS	NAME	CONTENTS
BEFORE	001015	19	PC D0	0000100A ABCD0005
AFTER	001015	19	PC D0	0000100E 00000019

Immediate

Specifies the source operand using two extension words and may be expressed in:
decimal (& prefix or none)
hexadecimal ($ prefix)
octal (@ prefix)
binary (% prefix)
ASCII (string within ‘’)

Immediate quick

This is an optimized version of immediate addressing which uses one word binary representation for the instruction and data. The operand is stored in a 32-bit sign-extended destination address. This mode can be used with the following instructions
MOVEQ (operand must be an 8-bit signed integer)
ADDQ (operand must lie within the range of 1 to 8)
SUBQ (operand must lie within the range of 1 to 8)

Discussion

Discuss the various addressing mode and their differences.

4.0 Self-Assessment Exercise(s)

1. In basic computer operation, _____ is a method used by the CPU to identify the location where data or information is stored.
   A. Addressing
   B. Addressing mode
   C. Indirect Addressing
   D. Immediate Addressing.
   Answer: A

2. Addressing modes gives assembly language programmers the flexibility to write programs that are more efficient with respect to the number of
   A. Lines of codes
   B. Instructions and execution time.
   C. Programs
   D. All of the above.
   Answer: B

5.0 Conclusion

The processor uses addresses to find operands in memory or register locations. Addressing modes enable the programmer to write programs to manipulate operand in computer memory or register location in a predefined way.

6.0 Summary

In this unit, you have learned how to use different addressing modes to write instructions for manipulating memory or register based operands. In the next unit, you will learn how to write program segments in assembly language using different instruction types.

7.0 References/Further Reading

Address. Available here

Mano, M.M. (2014). *Computer System Architecture*. (3rd ed.). Available here

Wang, S.P. & Ledley, R.S. (2003). *Computer Architecture and Security: Fundamentals of Designing Secure Computer Systems*. Available here

Unit 3: Instruction Types

Contents

1.0 Introduction

In this unit, you will learn about the major categories of instructions used by the computer to carry out its operations and different types of instructions under each category. After studying this unit, you will be able to write instructions to achieve different purposes and be able to combine several instructions into useful assembly language programs.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- Interpret and modify instruction types of programming tasks.
- Evaluate instruction types of programming tasks.

3.0 Main Content

3.1 What is a computer instruction?

Have you given instructions to anyone today? Did you get instructions from your facilitator or boss? Can you identify some reasons why people give instructions to or receive instructions from others?

A computer instruction is a set of commands type by a human using a keyboard (or another input device) and given to a computer to enable it carry out a specific task. The computer uses an operating system and a compiler to interpret the instructions and carry out the specified tasks.

Now, does the computer determine which operation to carry out and how to carry it out?

Which classes of instructions are available in modern computer systems?

3.2 Classification Based on Type of Operation

There are three categories instructions based on this parameter:

- Data transfer instructions
- Data manipulation instructions
- Program control instructions

3.2.1 Data Transfer Instruction

The instructions transfer data or information from one location in the computer to another without changing its value. The transfers may take place between on processor register and another, between the memory and processor registers, or between processor registers and I/O. The commonly used data transfer instructions are presented in Table 4 (Mano, 2014). Also presented is the opcode (mnemonic symbol) for each instruction.

Table 4: Basic Data Transfer Instructions

Name	Opcode (Mnemonic)
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Note that the mnemonics used by different computers may differ even if the instruction name is the same. However, the operations performed are basically the same. The load instruction is widely used to transfer data from memory to a processor register (usually the accumulator). The store instruction transfers data from a processor register into memory. Computers with multiple CPU registers use the move instruction to transfer data between registers. It is also used to transfer data between two memory locations or between CPU registers and memory. The exchange instruction swaps data between a register and memory location or between two registers is accomplished using the exchange instruction. The input and output instructions are used for data transfers among processor registers and input or output terminals. Data transfers between processor registers and a memory stack are carried out using the push and pop instructions.

Some versions of assembly language use the opcode (mnemonic symbol) to highlight the distinctions between different addressing modes. This makes the symbol LDI and MOVI become the mnemonic for load immediate and move immediate respectively. Other variations of assembly language indicate the addressing mode by using a special symbol. For example, placing a pound sign # before the operand indicates that the instruction uses the immediate mode. Note that each instruction can be used with different addressing modes.

3.2.2 Data Manipulation Instructions

These refer These refer to a group of instructions which provides the processor with computational capabilities so that it can perform various operations on data. Data manipulation instructions perform arithmetic, logic and shift operations. Such instructions are usually of three basic types:

i. Arithmetic instructions
ii. Logical and bit manipulation instructions
iii. Shift instructions

Arithmetic Instructions

Arithmetic instructions perform basic arithmetic operations such as addition, subtraction, multiplication, and division. Modern computers support these operations. Some small computers, however, can implement only addition and (possibly) subtraction instructions. The multiplication and division operations are implemented as special forms of addition and subtraction and are carried out using software subroutines.

Table 5 presents commonly used arithmetic instructions (Mano, 2014).

Table 5: Commonly Used Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

The increment instruction is used to increase the value of a register or memory operand by 1. The execution of an increment operation on a register containing all 1's produces an all 0's result. The decrement instruction reduces the binary contents of a register or memory location by 1. A decrement of a number containing all 0's, produces a result containing only 1's. The add, subtract, multiply, and divide instructions can be applied on different data types (integer, floating point, binary or decimal data, single-precision or double-precision data). The operations are carried out on data that are in processor registers during program execution.

A computer may have instructions that specify the data types of the operands it manipulates. Such computers may have different add instructions binary integers, floating-point operands, and decimal operands. The following add instructions use different data types.

ADD I Add two binary integer numbers
ADD F Add two floating - point numbers
ADD D Add two decimal numbers in BCD

The computer stores the carry bit from an operation using a special carry flip-flop. The "add with carry" instruction sums two operands and adds the carry bit from the previous operation. In a similar way, "subtract with borrow" instruction computes the difference of two words and a borrow which may occur as a result of a previous subtract operation. The negate instruction calculates the 2's complement of a number, and chages the sign of an integer that is represented in the signed-2's complement form.

Logical or Bit Manipulation Instructions

These instructions are used to manipulate individual bits or a group of bits stored in registers. The logical instructions process each bit separately and manipulates it as a Boolean variable. Hence, it is possible to change bit values, to clear a single bit or a group of bits, or to add new bits into registers or memory operands.

Table 6 lists common logical and bit manipulation instructions (Mano, 2014).

Table 6: Logical and bit manipulation instructions

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

The clear instruction replaces the specified operand by 0's. The complement instruction (1's complement) is takes a binary operand as an input and produces an output containing the inverse of the input bits. The AND, OR, and XOR instructions performs the corresponding logical operations on individual bits of the operands. The Boolean operations performed by logical instructions are commonly implemented as bit manipulation operations. There are three possibilities when it comes to bit manipulation:

- a selected bit can be cleared to 0,
- or can be set to 1, or
- can be complemented.

Clearing a bit or a selected group of bits can be done using the AND instruction. ANDing a binary value to 0 produces a 0; while ANDing a binary value to 1 will produce no change in the value of the variable. That is, $xbo = 0$, and $xb1 = x$. You can AND operands containing 0's in specific bit positions with another operand in order to a selectively clear the bits in the target operand. The AND instruction also does masking (in fact AND instruction is called a mask) by inserting 0's in specific position(s) in an operand.

The OR instruction performs a selective set operation on a bit or a selected group of bits of an operand. ORing any Boolean variable x with 1 produces a 1, but the value remains the same when ORed with a 0. That is, $x + 1 = 1$ and $+0 = x$. You can set the individual bits of a target operand by performing an OR operation on an operand containing 1's in the specific bit positions and the target operand with the corresponding bits positons to be changed to 1. The XOR instruction performs a selective complement operation on the bits of an operand. It is based on the Boolean relationships $x \oplus 1 = x'$ and $x \oplus 0 = x$. Thus, XORing a binary

variable with a 1 produces its complement, without changing its value when XORed with a 0. The instructions allow you to set, clear or complement individual bits. Bit manipulation instructions can also be used to manipulate a flip-flop that controls the interrupt operations in the computer.

Shift Instructions
Shift instructions are used to move the bits of an operand to the left or right. The operation may be either logical shifts, arithmetic shifts, or rotate-type operations. The type of shift is determined by the bit inserted at the end of the operand. Table 7 lists four types of shift instructions.

Table 7: Typical Shift Instructions

Name	Mnemonic
Arithmetic shift left	SHLA
Arithmetic shift right	SHRA
Logical shift left	SHL
Logical shift right	SHR
Rotate left	ROL
Rotate right	ROR
Rotate left through carry	RORC
Rotate right through carry	ROLC

The rotate instructions shift out the bits at one end of an operand into the other end in a circular fashion. In rotate through carry instruction, a carry bit is seeing as a part of the register holding the operand being rotated. The instruction inserts the carry bit into the least significant bit position of the operand in the register, transfers the leftmost bit position into the carry, and shifts the entire bits of the operand to the left at the same time.

The logical shift places 0 at the end of a binary operand. The last bit is the leftmost bit for shift right and the rightmost bit position for the shift left. Arithmetic shifts are usually implemented as signed-2's complement operations. The sign bit in arithmetic shift-right instruction is shifted to the right alongside the rest of the number. The sign bit itself remains unchanged and the end bit also does not change. In arithmetic shift-left instruction, a 0 is inserted at the end position. This is similar to the logical shift-left instruction. Arithmetic shift-left instruction is not available as a separate instruction in many computers.

3.2.3 Program Control Instructions
Program control instructions are used to make decision and alter the direction of program execution. The execution of a program control instruction changes the contents of the program counter and redirects the

flow of control. This resells in an alteration of the normal sequence of program execution. Digital computers use this mechanism to control the flow of instruction processing and provide branching to different portions of the program.

Table 8 provides a list of commonly used program control instructions (Mano, 2014).

Table 8: Program Control Instructions

Name	Mnemonic
Branch	BR
Call	JMP
Skip	SKP
Call	CALL
Return	RET
Compare (by subtraction)	CMP
Test (by adding)	TST

The branch and jump instructions perform the same operation and one can be used in place of the other. However, they may sometimes be used to indicate distinct addressing modes. The jump (or branch) usually contains a single operand in the address part of the instruction. The assembly language format is

BR ADR

where ADR is a symbolic name for a memory location. A branch instruction transfers the value of ADR into the program counter. This makes the instruction at location ADR the next instruction to be executed.

Branch and jump instruction may be conditional or unconditional. A conditional branch transfers control does not require the satisfaction of any conditions before shifting control to a specified address. The conditional branch requires the satisfaction of specific before the branch can take place. A conditional branch instruction may be branch if positive or branch if zero. If the condition is satisfied, the address specified in the branch instruction is transferred into the program counter and that become the address of the next instruction. If the condition is not met, contents of the program counter remains the same does and the program continues with normal sequential execution of instructions.

The skip instruction usually has an empty address field. A conditional skip instruction transfers control to the next instruction based of the satisfaction of the predefined condition. This is achieved by a simultaneous increment of the PC during both the retrieval and execution

phases of an instruction. The processor transfers control to the next instruction If the condition is not met.

The compare and test instructions listed in Table 8 do not cause a direct alteration of the sequence of program execution. They only set conditions for the next set of conditional branch instructions by computing the difference of two operands, and without retaining the result of the operation. However, the operation results in the setting of certain status bit conditions. Similarly, the test instruction executes a logical AND on two operands and updates certain status bits, but does modify the operands or retain the result of the computation. Commonly used status bits are carry bit, sign bit, a zero indication, and an overflow condition.

Conditional Branch Instructions

Table 9 presents a summary of commonly used branch instructions (Mano, 2014).

Table 9: Common Branch Instructions

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	$Z = 0$
BNZ	Branch if not zero	$Z = 1$
BC	Branch if carry	$C = 1$
BNC	Branch if no carry	$C = 0$
BP	Branch if plus	$S = 0$
BM	Branch if minus	$S = 1$
BV	Branch if overflow	$V = 1$
BNV	Branch if no overflow	$V = 0$
Unsigned Compare Conditions $(A - B)$		
BHI	Branch if higher	$A > B$
BHE	Branch if higher or equal	$A \geq B$
BLO	Branch if lower	$A < B$
BLOE	Branch if lower or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$
Signed Compare Conditions $(A - B)$		
BGT	Branch if greater than	$A > B$
BGE	Branch if greater or equal	$A \geq B$
BLT	Branch if less than	$A < B$
BLE	Branch if less or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$

Note that each mnemonic is made up of the letter B (for branch) and an abbreviation of the condition name. The letter N (for No or Not) indicates that the opposite of the condition state is used. Thus, BC and BNC are interpreted as Branch on Carry, and BNC is Branch on No Carry

respectively. Control of program execution is transferred to the address specified by the instruction if the stated condition is met. Otherwise, program execution continues with the instruction in sequence.

The zero status bit determines whether or not an ALU operation produces a zero or non-zero result. The carry bit indicates if the most significant bit of the contents of the ALU is a carry bit. It is can be combined with the rotate instructions to check determine the bit shifted from the end of a register operand into the carry position. The sign bit reflects whether the most significant bit of the output of ALU is positive or negative. That is, S = 0 denotes a positive sign and S = 1, a negative sign. Therefore, a branch on positive checks for if the sign bit is 0 and a branch on negative checks if the sign bit is 1. Generally, these two conditional branch instructions can be used to determine the sign and magnitude of the most significant bit. The overflow bit is used to check if the result of an arithmetic operation causes an overflow. That is, if the number of bits in the output exceeds the size of the register or memory word allocated to store the result. It is used in conjunction as part ofarithmetic operations performed on signed (2's complement) numbers.

Subroutine Call and Return

A subroutine is a self-contained part of the program that performs a specific function. The main program issues repetitive calls to the subroutine as may be required during execution. Each call to a subroutine transfers control of execution to the first address of the subroutine and triggers the execution of its instructions. Control is returned to the main program after the execution of the subroutine.

Call and return instructions such as call subroutine, jump to subroutine, branch to subroutine, or branch and save address can be combined with subroutines. The location where a subroutine begins from can be specified by the call and return instruction consisting of an operation code and an address. The execution of this instruction involves two operations: (1) creating a temporary location used to store a memory address to which control is transferred after the execution of the subroutine (the return address (i.e. the return address), and (2) the transfer of control to the beginning of the subroutine. The return from subroutine retrieves the return address from the temporary location and loads it in the program counter. This transfers control to the instruction referenced by the return address.

Why do some computers have only one operand in their address field, while others have up to two or three operands?
Does the number of operands in the address field have any effect of the ease of writing programs and the execution of such programs by the computer?

3.3 Classification Based on Number of Operands

This approach classifies instructions based on the number of operands in the address field. Here, there are three main types of instructions:

- Zero-address instruction
- One-address instruction
- Two-address instruction
- Three-address instruction

3.3.1 Zero Address Instructions

Zero-address instruction commonly found in stack-organized computers have empty address field for arithmetic operations such as addition (ADD), subtraction (SUB), multiplication (MUL) and division (DIV). However, the PUSH and POP instructions, uses an address field to reference the operand that communicates with the stack. The following program implements the arithmetic expression V = (W + X) ∗ (Y+ Z) for a stack organized computer. Note that TOS stands for top of stack.

```
        PUSH   W      TOS ← W
        PUSH   X      TOS ← X
        ADD           TOS ← (W + X)
        PUSH   Y      TOS ← Y
        PUSH   Z      TOS ← Z
        ADD           TOS ← (Y + Z)
MUL     TOS ← (Y + Z) ∗ (W + X)
        POP    V      M [V] ← TOS
```

3.3.2 One Address Instructions

A one-address instruction as the name suggests contains only one operand in the address part of the instruction. These instructions carry out all data manipulations using the accumulator (AC). It also stores the result of all operations in the accumulator.

Example 1
The following program is used implement the arithmetic instruction X = (A + B)

```
LOAD A    AC ← M [A]
ADD B     AC ← A [C] + M [B]
STORE X   M [X] ← AC
```

All operations are performed on the contents of the accumulator and data stored in a memory address. The intermediate result is stored in temporary memory address.

Example 2
The following program uses one address instruction to computer the perimeter of a rectangle. Recall that Perimeter = 2*(L+B)

```
LOAD L          // Load the length into the accumulator
CLC  // Clear the carry flag so it does not get added into the result
ADD B       // Add the breadth and store in the accumulator
MULT #2         // Multiply the contents of accumulator by 2
STORE P     //Transfer the result in accumulator to memory location P
```

Example 3
Add x and y together and storing the result in x; that is, x = x + y

```
LOAD x      // Load one operand into the accumulator.
CLC  // Clear the carry flag so it does not get added into the result
ADD y       // Add the other operand
STA x// Store the operand back to x
```

3.3.3 Two Address Instructions

A two-address instruction contains two operands in the address portion of the instruction. One of the operands is as the source, while the other is the destination. Two-address instructions are the most common in computers used for commercial operations. The instruction

ADD R1, R2

denotes the operation R1 ← R1 + R2. This instruction directs the control to add the contents of register R2 to that of R1 place the result in R1. Here, R2 and R1 are the source and destination respectively.

The instruction

MOV R1, R2

denotes the register operation R1 ← R2 (or R2 ← R1, depending on the kind of computer you use).

Example 4
We rewrite the program in Example 2 using two address instructions.

```
MOV R1, L       // Load the length into register R1
MOV R2, B       // Load the breadth into register R2
ADD R1, R2// Add the contents of R1 and R2 and store the result in R1
MULT R1, #2     // Multiply the result in R1 by numeric value 2 and store
the result in R1
STORE P, R1     // Transfer the contents of R1 into memory location P
```

3.3.4 Three Address Instructions

A two-address instruction These instructions feature three operands in the address field. Two of the operands designate the sources, while the third operand is the destination. The contents of the address field may specify either a processor register or a memory operand. The following assembly language program evaluates X = (A + B) ∗ (C + D). The register transfer operation is also indicated alongside the corresponding instruction.

ADD R1, A, B R1 ← M [A] + M [B]
ADD R2, C, D R2 ← M [C] + M [D]
MUL X, R1, R2 M [X] ← R1 ∗ R2

The computer uses two processor registers, R1 and R2. The symbols, M [A], M [B], M [C] and M [C] denote the operands at memory locations A, B, C AND D respectively. Three-address format produce short programs when applied to arithmetic operations. However, the binary-coded instructions used to specify three-address formats require too many bits. The format of instructions in Cyber 170 (an example three-address instruction commercial computer) is based on either three register address fields or two register fields and one memory address field.

Mini project

Study sample programs written using one-address, two-address and three-address instructions. Compare them in terms of size, complexity, ease of understanding, etc. Discuss your findings in your study group.

4.0 Self-Assessment Exercise(s)

1. Data manipulation instructions can be seen as
   A. A group of instructions which provides the processor with computational capabilities so that it can perform various operations on data.
   B. The instruction set that provides instructions to move data from/to these two registers to/from integer registers.

Answer: A

5.0 Conclusion

The availability different instruction types allow an assembly language programmer to select an instruction that is suitable for a specific operation. This enables the programmer to translate algorithms into

functional computer programs that process data stored in memory and register locations and to accomplish the operations specified in algorithms.

6.0 Summary

In this unit, you have learned the various types of instruction, their uses and format. The knowledge you have acquired will enable you to write functional assembly language programs for real world problems.

7.0 References/Further Reading

Mano, M.M. (2014). *Computer System Architecture*. (3rd ed.). Available: here

Null, L. & Lobur, J. (2003). *The Essentials of Computer Organization and Architecture*. Sudbury, Massachusetts: Jones and Bartlett Publishers.

Wang, S.P. & Ledley, R.S. (2003). *Computer Architecture and Security: Fundamentals of Designing Secure Computer Systems*. Available here

Module 3: Secure Component Design

Module Introduction

This module introduces you to the secure component design of the computer system. It discusses components design that includes processing unit design, memory system design and input and output design.

Unit 1: Processing unit design
Unit 2: Memory system design
Unit 3: Input and output design
Unit 1: Processing Unit Design

Unit 1: Processing unit design

Contents

1.0 Introduction

In this unit, you will learn the concept of processing unit design of a computer system. The processing unit design is divided into seven

sections including instruction set, registers, program counter and flow control, Reduced Instruction Set Computer (RISC) processors, pipelining, CPU security and virtual CPU. The instruction set is subdivided into instruction classifications, logic instructions, arithmetic instructions, and Intel 64/32 instructions.

2.0 Intended Learning Outcomes (ILOs)

In this unit, you are expected to be able to:

1. Design a processing unit

3.0 Main Content

3.1 Instruction set

Instruction set offers a platform that enables users to carry out full use of the processor. The choice of instructions is decided by the apparatus designed for by the processor. For instance, a data-intensive data can pick processors that are built with greater word size, extra math functions and an excellent number of memory management features. Conversely, a processor designed/developed for implanted devices like remote controls for TV and DVD player could have simple or fewer instructions, little word size and easy addressing characteristics (Layton, 2016).

3.1.1 Instruction classification

Instructions often comprise 2 components namely, operands and operation code (opcode). Each instruction may contain one opcode and zero or more than one operands. Figure 3.1 depicts regular instruction formats. Operands are situated either in a memory location or in registers with fast speed. Table 3.1 highlights some well-known categories of instructions. Logic and arithmetic instructions are considered as a sub-part of the register reference instructions and memory.

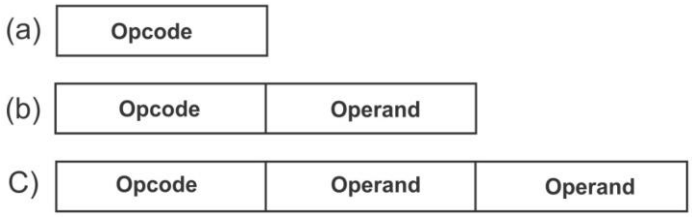

Figure 3.1 Opcode and Operands

3.1.1.1 Logic Instructions

In the instructionset, every processorcontains logical operation instructions. Regular logic operations include NOT, OR AND, and Exclusive OR (XOR). NOT operation turns upside down an input AND requires that both inputs are 1 so as to obtain 1. OR requires that one or more value of the inputs is 1 so as to obtain 1. Conversely, OR brings out 0 when both inputs are zero (0). Last is the XOR, which works by adding both inputs and removing the carry if it occurs. Thus, when two inputs are dissimilar, the output is 1 else is zero (0). Figure 9 depicts the instances of the logic operation pertaining to XOR (Patterson and Hennessy, 2013).

3.1.1.2 Arithmetic Instructions

The arithmetic instructions comprise SUB, DIV, ADD, MUL and the rest. Design and development of arithmetic instructions differ based on the type of processor. Meanwhile, it typically relies on the extent of size of the instruction. Depending on the amount of operands, the instructions are ordered as 1) three addresses operands 2) two addresses operand 3) one address and zero address operands. Table 10 depicts regular classes of instructions.

Table 10 Regular classes of instructions

Instruction Type	Operation Process
Memory to memory instructions	The two operands reside in memory
Memory to register instructions	Just only single operand resides in memory, while the other one resides in register
Register reference instructions	Operation is conducted on constituent of one or extra registers
Memory reference instructions	Operation is conducted on constituent of the memory location
Control instructions	Pause, halt, branching, and the rest.
Input and output instructions	Perform input or output
Macroinstruction	Group of instructions

Considering three address machine, it contains 2 operands and the outcome of the result is kept separately. A two address machine re-use single address, which is one of the operands and afterward retain the outcome. The one-address machine often employs the accumulator (ACC) with one address, that is, usually a register to carry out the execution.

Address in instruction set is often mentioned as a memory unit, register or both. Because registers are considered to be more speedy than memory, instructions that employs registers as operands are faster than computers that employs only memory. Ordinary processors employ registers to carryout arithmetic operations,. A number of embedded chips to make use of memory to do so (Coppola et al., 2018).

The following are instances of instructions involving varied operands

i. Three addresses machine

$$ADD\ A, B, C \qquad C \leftarrow A + B$$

Every instruction requires three memory access or registers during every execution. In reality, the greater parts of operations are centered on two operands with the resulting outcome residing in the location of one of the operand. Hence, a two address instruction could be suitable for this kind of processor.

ii. Two addresses machine

$$ADD\ A, B \qquad A \leftarrow A + B$$

The two operands instruction are a type of processors. Further to fetch execution cycle, the two addresses put in a write-back cycle in order to keep the outcome to the position/location of an individual operands.

iii. One address machine

$$ADD\ B \qquad ACC \leftarrow ACC + B$$

A one-address processor might be considered as a lookalike to a two-address processor excluding if by default it uses the ACC and registers in CPU.

The addressing approach for operands could be register, indirect or direct. Direct concept for addressing is considered as when an operand is a value based on numbers. The Indirect concept denotes that the operand is an address, which points to a memory location. While the register addressing concept is considered as when the operand is kept in a register.

Direct addressing $ADD\ A, 3 \qquad A \leftarrow A + 3$

Indirect addressing $ADD\ A, Pi \qquad A \leftarrow A + Pi$

Considering indirect concept of addressing, the Pi (π) operand is considered as a pointer or label referring to the memory position/location of the main number.

3.1.1.3 Intel 64/32 Instructions

Instructions for transferring data transmit between the installed memory and the general purpose and section registers. The data transfer instructions also carry out particular operations for example; stack access, data conversion, and conditional moves. There are several instructions related to the Intel 64/32, which includes move, stack manipulation, shift, rotate, control transfer, call and return, loop, random

number generator, and program environment instructions (Andrade et al., 2014).

3.2 REGISTERS

Registers are developed for particular purposes and are wired deliberately for diverse functions. A processor has one or more registers to store data and instruction. A number of unique registers are employed to hold the condition of the present operation. Do you know the advantage of registers in a system? Well, registers are the memory with the most rapid processing in a computer system. The rationale for using registers is to make storage and computation process faster. Through minimizing the time for accessing external and internal memory. Thus, the machine with huge amount of registers for enhancing speed is named, reduced instruction set computer (RISC). Processors often allocate unique functions to a particular registers, as well as the following registers:

i. An Accumulator (ACC): gather the outcome of computations (von Neumann,2006).
ii. Address registers: monitor where a specified instruction or portion of data is kept in memory. Every memory location is recognized by an address.
iii. Data registers: do not permanently store data collected from or that is about to be forwarded to a memory.
iv. Status Registers: performs the function of storing present CPU status.
v. Program Counter (PC): is employed to point to the present command being executed.

Recently built processors often contain higher sophisticated control unit and devoted control bits for security. Thus, Intel Core i7-900 is centered on 45 ns process paradigm. It comprises an executable bit, which enables the classification of memory into non-executable or executable. This classification is used when incorporated with an augmenting operating system. Whenever a code tries to run in a non-executable memory, the processor triggers a fault message to the operating system. This characteristic can evade a number of classes of viruses or worms, which take advantage of buffer overrun vulnerabilities. Thus, assist to enhance the complete security of the system (Daswani et al., 2007).RISC often employs registers for every operand in specific function, for instance, ADD. It enables programs execute faster. Consequently, programmers must load values from the stored memory to registers hence causing the program to be large in size. The different registers include the general-purpose, segment, and EFLAGS register.

3.3 Program Counter and Flow Control

A program counter (PC), in another word instruction pointer, serve as a unique register that keeps the present program running location. PC is added up to point to the subsequent instruction by adding up a single word after fetching every instruction (Hennessy and Patterson, 2013).

$$PC \leftarrow PC + 1$$

In the expression, 1 denotes the subsequent instruction. In a 32 bit processor, subsequent instruction is located at PC+4. Thus, PC often points to the subsequent command (instruction) after the present command is fetched, translated, and executed. Program does not often execute linearly. It can halt at a number of points to execute task interrupts and jump to a different instruction; this is called branching. We have two kinds of branches namely, unconditional and conditional. The *goto* statement is a popular method in developing branches. Nevertheless, there is a divergent view on whether not to use or to use *goto*. A lot of programming languages used *goto* statement although; a number of others did not, for example, Java. The structured program rule justify that the *goto* declaration is not required to code programs (Knuth, 1998). The *goto* statement is usually joined with *If* statement in the pattern of:

IF *condition* THEN *goto label*;

3.4 RISC Processors

A reduced instruction set computer (RISC) is termed as a kind of microprocessor, which recognizes a comparatively constrained amount of commands. The major advantage of the RISC is the fact that it runs the commands/instructions extremely fast because the instructions are reduced and now simple. In addition, another major benefit is that RISC chips need less number of transistors, which position them to be cheaper to develop. From the time when RISC was discovered, traditional systems have been named to as complex instruction set computers (CISCs). The design characteristics for the majority of RISC processors are pipelining, one cycle execution at a time and a huge amount of registers. Considering the architecture and programming of the RISC, only easy commands, which can be run in one clock cycle is employed. The predecessor of the RISC is the CISC architecture. The key aim of CISC architecture is to finish a whole task in a few lines of assembly language. Figure 3.2 depicts MULT instruction executions in RISC architecture

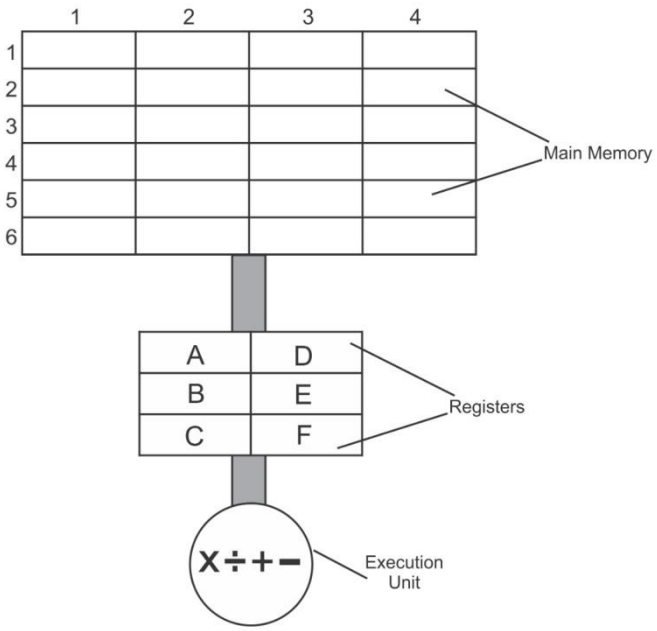

Figure 3.2 MULTInstruction executions in RISC architecture

A CISC processor is a normal processor, which is embedded with a multiply (MULT) instruction. If MULT is executed, it loads the numbers (values) into distinctive registers, then carry out the product of the operands in the run unit, and then keeps the product outcome in the suitable register (Andrade et al., 2014). Hence, the whole work of multiplying two values can be attained with just a single instruction:

MULT 2:3, 5:2

3.5 Pipelining

Conventional pipelining is a classic characteristic in RISC processors, which is similar to assembly line. Since the processor functions on diverse level of the instruction at same period, additional command can be run in a little period.

3.5.1 Types of Pipelines

The various types of pipelines in computing include:

i. Instruction pipelines: permit overlapping execution of more than one instruction with the equivalent circuitry. It is often categorized into phases namely, register fetching, arithmetic phase and instruction decoding, in which at every phase processes single instruction at a given duration.
ii. Graphics pipelines: It exist in almost all graphics cards, that comprise of more than one arithmetic unit, or whole CPUs, which apply the different phases of regular rendering operations that is,

light calculation, window clipping, perspective projection, rendering, and color.

iii. Software pipelines: commands are written in such a way that the outcome of single operation is spontaneously employed as the input to the subsequent, operation. The UNIX system call pipe as a standard instance of the explained concept; even though other OS also take advantage of pipes (Hockney *et al*., 1988).

3.6 Central Processing Unit Security

A number of tamper resistant CPUs are put together such that physical efforts to modify or read the data within the CPU may not be achieved with no difficulties. The CPU's tamper-resistant package guards its internal mechanisms, for instance, cache and register from hardware attacks. CPU's inner cache is large sufficiently, for instance, 10x megabytes to accommodate a kernel and the operational set of information for almost all programs. However, not large enough to accommodate entire programs. Programs that need additional memory employ un-trusted exterior memory. What do you think a kernel trap handler is? The CPU attached to a kernel while ejecting or fetching a cache line. Therefore, kernel attached handler can guard data kept in auxiliary (external) memory. This kind of CPU has a matching public and private key pair. Meanwhile, the private key is concealed in the CPU and not exposed to any user such as software, which executes on the CPU.

The CPU discloses the public key in a CPU certificate that is endorsed by the computer company. A consumer trusts a CPU if its certificate is endorsed by a trusted computer company. ACPU employs the private key to sign and decrypt data, and consumers employ the public key to confirm signatures and encrypt information for the CPU.

The framework Build and schedule processes using a kernel on the CPU, taking into account the level of the program. In the kernel, traps and interrupts are dealt. The kernel operates in privileged mode; it can write or read all locations of physical memory. The kernels are separated into distinctive address spaces using page tables. In order to avoid command instructions affecting or accessing data in a specific area, the CPU used traditional melody defense. This is only possible if the kernel updates the page table of all address spaces. The kernel text and some of its data can be avoided live in the protected CPU to prevent the kernel being abused (Wang and Ledley, 2012; Patterson and Anderson, 2013).

The cost of trapping every cache event is dependent on the memory access existence and abnormal rate of a program, the costs of cryptographic procedures and the level of safety needed by a program. The average size of a CPU cache and the hardware-supposed cryptographic mechanism is assumed to have decreased. A secure

processor is shown in Figure 3.3. A full operating system is the kernel and a collection of user-level servers that enforce OS abstractions. The device user can also run VMWare virtualized operating systems in user space. The system can report the software and hardware settings of a computer, thus a user can make choice on whether the software and hardware can be relied on for protecting the system for customers. The CPU determines every kernel via the kernel's text content hash and the initialized data segment. When the kernel is updated before the device boots, it will change the content hash

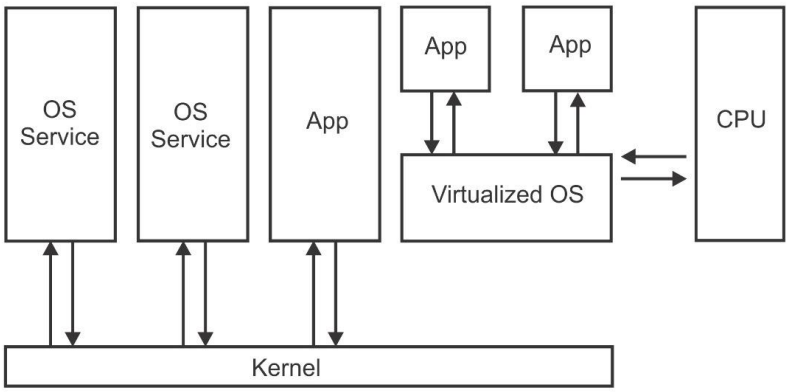

Figure 3.3 A secure processor

That is because the text and data of a kernel inside the tamper-resistant CPU cannot be altered after the boots of the computer, for example, via a DMA unit. The entity hash for the kernel is known as the kernel signature. In several levels, a stable processor device boots up. The Processor calculates the content hash of the BIOS when there is a hardware reset, and jumps to the BIOS file. The BIOS then tests the boot loader's content hash, which is located in the first sector of the computer's master hard drive, and transfers to the boot loader disk. Finally, the kernel signature is determined by the boot loader code, and jumps into the corresponding kernel. A privileged CPU instruction is used for every stage. A privileged CPU instruction is used in each stage for calculating the hash material. The instruction preserves the Hash material within the CPU register. The registers are held so as not to change their content in malicious programs (Wang and Ledley, 2012).

3.7 Virtual Central Processing Unit

Virtual machine is termed as "computer on a computer," which, as if on a real physical device, is a completely separated software container that runs its own operating system and applications. Unlike a real computer device, a virtual machine works with its own virtual RAM, hard drive, CPU, and network interface card (NIC). Figure 3.4 shows a virtual machine architecture diagram

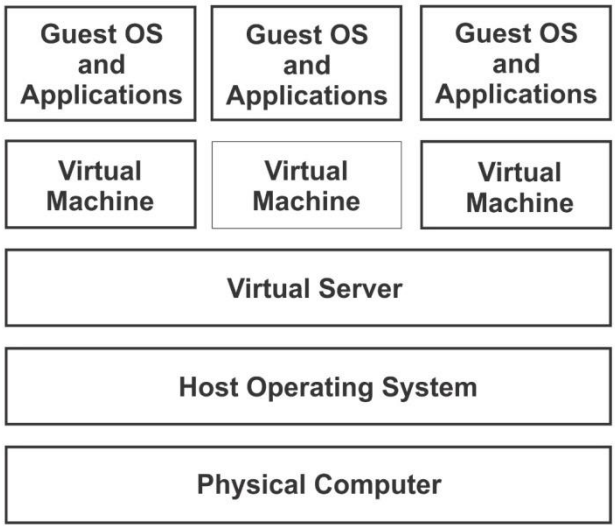

Figure 3.4 Virtual Machine Architecture Diagram

A virtual Processor is a Processor that uses software-centric host CPU resources and functions as a standard CPU. The combination between a virtual machine/CPU and a real machine/CPU cannot be accomplished by an operating system and software programs or other devices may not be reached over the network. Yet a virtual machine is entirely case of a vi software and does not contain hardware components. Virtual machines therefore offer some unique advantages over physical hardware (Wang and Ledley, 2012).

In the virtual machine, many OS environments will co-exist with each other on a single computer. The virtual machine will generate a set of instructions (ISA) for maintenance, availability, program delivery, and disaster recovery that is slightly different from the actual system. VM's main features are as follows.

i. It can run multiple operating systems, such as Debian, Windows, Mac and the rest, on a single computer.
ii. It helps in reducing money costs by improving energy efficiency through the use of less hardware while also improving your admin to server ratio.
iii. It also guarantees enterprise applications work with maximum availability and performance.
iv. It helps in building up business continuity via enhanced disaster recovery resolutions and achieves increased availability all through the datacenter.
v. Enhance enterprise desktop control and management with more rapid deployment of desktops and lesser support calls due to application disagreement.
vi. Enhance company system management by speeding up system delivery and reducing service calls due to program inconsistencies.

Since virtual machines share the host 's general resources, it requires more memory and can cause performance problems.

Case Studies *CMOS VLSI Design Case study: Intel Processor*

4.0 Self-Assessment Exercise(s)

1. Which of the following is not a type of instruction?
   A. Memory to memory instructions
   B. Memory to register instructions
   C. Register reference instructions
   D. Register to cache instructions

Answer: D

5.0 Conclusion

In this unit, we have studied the processing unit design. From the discussion, instruction set, registers, program counters, and flow control, reduced instruction set computer, pipelining CPU security and virtual CPU has been discussed. Therefore, we have learned the basic design concept of the processor unit.

6.0 Summary

A CPU contains control signals, data, and address. The data buses and size ofthe address is often 64 in a 64-bit CPU. Thus, a 32-bit CPU will get the highest memory of $2^{32}= 2^2$ x $2^{30}= 4$ GB.

A CPU can take several and varied instructions. A number of instructions for example logic instructions can use less time to run than the arithmetic instruction. A typical instruction has two parts namely opcode and operand. In addition, the addressing method for the operands can be indirect and direct. Registers are considered as the fastest memory in computer systems. The important registers are; address register, accumulator, data registers, status registers, and program counters.

RISC contains very fewer instructions and each and every instruction takes one cycle. The aim here is to make instructions to be easy so that they can easily be pipelined. This is to achieve a single clock throughput at greater frequencies.

Several researches have been carried out for securing CPUs to evade kernel being tampered with. One method to do this is to employ encryption by adding a public-private key pair. Virtual CPUs are software-centered CPUs that employ the resources of the host CPU and behave like normal CPUs. The virtual machine can offer an instruction set architecture (ISA), which operate likean actual machine.

7.0 References/Further Reading

Coppola, M., Grammatikakis, M. D., Locatelli, R., Maruccia, G., &Pieralisi, L. (2018). *Design of cost-efficient interconnect processing units: SpidergonSTNoC*. CRC press.

Patterson, D. A., & Hennessy, J. L. (2013). *Computer organization and design MIPS edition: the hardware/software interface*. Newnes.

Wang, S. P., &Ledley, R. S. (2013). First Edition. Computer architecture and security: Fundamentals of designing secure computer systems. John Wiley & Sons.

Unit 2: Memory System and Design

Contents

1.0 Introduction

In this unit, you will learn the concept of a memory system design of a computer. The memory system design includesone-bit memory circuit, register (MAR, MDR and Main Memory), cache memory, virtual memory, non-volatile memory, external memory, memory access security.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- Design memory system to solve a specific problem

3.0 Main Content

3.1 Memory System Design

Memory is the major constituent in a computer system. The speed of computer relies greatly on the memory capacity of the system. Cost and speed are the two essential parameters of memory. A memory system's "entry time" requires writing memory time and deleting memory time. If you need to create your memory quickly, add a reasonable amount of

circuit to your main memory and use the fastest memory circuit. Therefore the memory is costly. The systems' not immediately needed programs and data are usually stored in less costly secondary memory, such as a hard drive to reduce memory costs. Secondary storage is often seen as a storage. When the CPU requires it, data stored in the storage is transmitted to the main memory. In this unit we switch from designing and creating a one-bit memory to ordering and connecting a big memory structure through the memory interface. This unit ends with a review of possible threats to memory protection and how to store memory storage data (Kolodner, 2014).

3.1.1 One Bit Memory Circuit

Flash memory is employed for storing documents and photos. Fewer individuals have investigated how flash data is stored. Would you like a single memory unit and why can it store data without battery requirement? Let us first study the memory of a computer. Any device such that a data can be read and written is taken as a memory.

In addition, even after reading/writing has been finished, data must be processed and stored. Figure 3.5 depict a light bulb, which is one of Edison's major findings. Turning K is employed for activating the light, which is enabled until the switch has disabled, and then it is disabled before the light is re-activated. The analogy is to remember command entry from the bulb and switch.

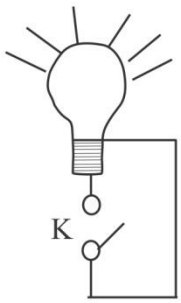

Figure 3.5 Light bulb and a switch

However, this device cannot be used in building up a computer because every operation requires human contact. Therefore, an electronic switch can be used as transistor. Figure 3.5 portrays a circuit which uses a transistor as an electronic switch. If the input is considered small, zero volts are, thus, no electrical current occurs from point *b* to *a*. The transfer *c* to *a* is disconnected, and the high performance is +5 V. When a high voltage is used at the input, then the current from point *b* to *a* would be available. The switch is therefore ON when there is a relation between *c* and the output is therefore *0V*, which is small

The conventional transistor-based electronic switch utilizes an electrical current to trigger the component. Thus, it is named the current-based

component. Current-based component depletes a large amount of power, it also has heart issues and hence, cannot be carried out on a large scale. The subsequent step employs metal oxide semiconductor (MOS), which was substituted by complementary metal-oxide semiconductor (CMOS) component. Figure 3.6 depicts a simple MOS switch.

Hypothetically, the component of CMOS depletes no current. It is a component that is dependent on voltage. Considering CMOS system, it is possible to integrate a large number of circuits into a single circuit. Thus, it is possible to create an integrated circuit (LSI) of large scale and an integrated circuit (VLSI) of very large scale. A great number of processors have recently been developed and produced using VLSI technology.

Figure 3.7 depicts a distinct memory cell. It can be considered as one-bit memory equivalent to the switch and light bulb. In addition, it could be seen as a circuit that neither joins two NOR gates together. Therefore, this circuit is named as a flip-flop or an RS trigger. There is set (S) and reset (R) as inputs, then Q is the output. The $\bar{Q}$ is the inverse output.

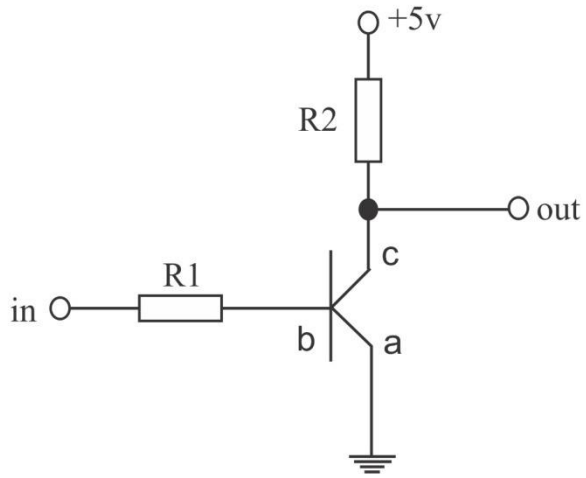

Figure 3.6 Electronic switch made of a transistor and resistors

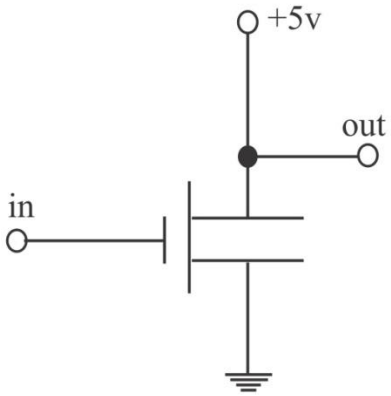

Figure 3.7 Electronic switch made of a MOS component

Assuming that $S = 1$ and $R = 0$, because $S = 1$ in that case $\bar{Q}$ should be 0. R and $\bar{Q}$is now $Q = 1$. When S now moves to be 1, in order for both S

and *R* to be 0. Thus, the circuit stays in the state explained above. When *R* becomes 1, then it will force *Q* to be 0 however, the two inputs *S* and *Q* to the top NOR gates are zero, in order to make $\bar{Q}$ turn into a 1. This situation will also continue. Now we have seen this flip-flop "remembers" all the two input signals, *S* and *R*. It works just like the aforementioned switch circuit and light bulb however, its entire constituent is electronic together with the control signal. Hence, this is the concept of a computer memory system. Computer memory constitute of billions of this kind of memory unit with supplementary circuits signaling and the processor (Wang and Ledley, 2012; Ohmaru, 2014).

3.1.2 Register, MAR, MDR and Main Memory

A unique kind of memory, which is employed to keep a binary value for a short term in order to perform basic storage and calculations, is called register. It is embedded in the CPU to carry out a particular role (Dumas, 2006). Contrary to normal memory, every register handles a specific operation depending on how it is configured. In relation to speed, the register is the fastest probable memory on a CPU-operated computer system. Registers are used inside a network in a variety of different ways. It can be used to keep this instruction's address running, this register is known as the program counter (PC). The Instruction Register (IR) retains the key instruction that the machine is running at the time. Status register contains various 1-bit registers and each tracks a particular CPU state. A Flag comprises of information including negative sign, overflow, arithmetic carry, and other vital data that is observed by the control unit. A one bit memory constructed using a flip-flop is shown in Figure 3.8

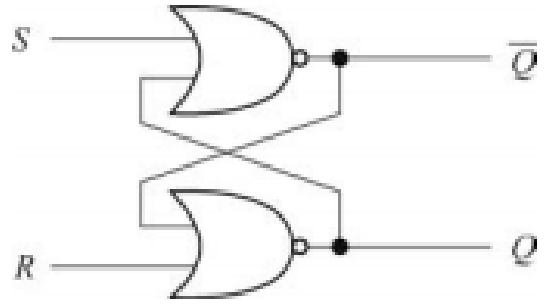

Figure 3.8 A 1-bit memory cell constructed by a flip-flop

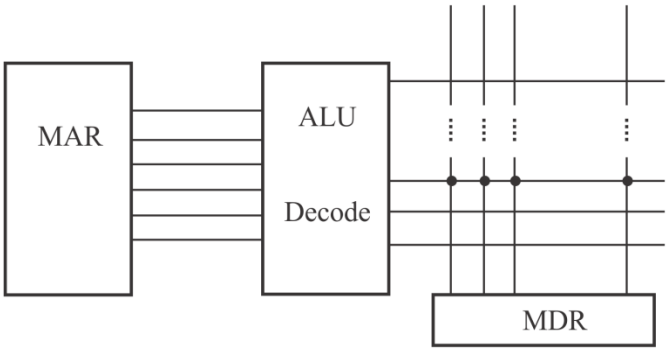

Figure 3.9 Relationship between the MAR, MDR, and memory

Random-access memory (RAM) or memory can be separately accessed in a random pattern. As mentioned earlier, there are often billions of single 1-bitmemory units in a computer memory system. A processor uses two separate registers to access certain memory modules, namely the Memory Data Register (MDR) and the Memory Address Register (MAR) (Foster and Iberall, 1985). A memory data register is also referred to as a memory buffer register because it houses the data being processed or retrieved from the memory location that the memory address register actually manages. Figure 3.9 demonstrates the relationship amid the MDR, MAR, and memory itself. An address decoder is normally employed to decrease the number of addresses needed for the huge amount of memory units. For instance, a 32-bit address bus will address 2^{32}= 4G memory.

For MAR, the maximum bit is named as the most important bit (MSB) and the minimum bit is named as the least significant bit (LSB). For MDR, the amount of bits that can be accessed simultaneously is known based on the register width, which is connected to the processor. Considering 32-bit data bus, a single instruction can run 32-bit data. Though 64-bit data can be processed in a 64-bit data bus in one instruction. A 64-bit machine is usually speedier.

Communication between the memory registers and CPU operate as follows: the CPU transfer (load) address and data to MAR and MDR respectively to store data at a common memory location. In the decoder the address is encoded in such a way that based on the MAR address only one output is allowed. Then the CPU transmits a signal that enables the write line to the memory. The data is then transmitted to the selected memory from MDR

The CPU forwards the address to the MAR in order to access and read data from a particular location. Interpreting the address, then choose it. Afterward the CPU sends a read command, then the data is read to MDR from the selected memory position. We may now note that MDR is a dual-way register with both in and out. Whereas MAR is a single way register that also transmits the signal out. You know a 32-bit machine can contain up to 4 GB of memory. Generally, a computer that has *k* bits address in width, then the overall amount of memory addresses is as expressed below:
$M = 2^k$

M represents the total likely amount of memory in a computer. For instance, an 8-bit processor like Intel 8080 or Z80, the total likely memory is 2^8 = 256. A 16-bit computer should have 2^{16} = 65,536 (64K) memory. Additionally, the 32-bit computer should contain a memory address, which is up to:

$2^{32} = 2^2 \times 2^{10} \times 2^{10} \times 2^{10}$

$4 \times 10^3 \times 10^3 \times 10^3 = 4 \times 10^9$

In a 1GB memory, which is about 109 bytes, the entire memory is equal to 4G for a 32-bit system. Current computer systems often employ RAM as the actual memory. A RAM called Dynamic RAM (DRAM) is used mainly on personal systems because it is cheaper and is heavily integrated as it has little power usage. DRAM requires being refreshed from time to time to evade data loss. However, another RAM called static RAM (SRAM) is typically faster than DRAM, with lower incorporation rate. Additionally, it more costly thus, it is often observed in servers or unique fast computers (Foster and Iberall, 1985; Wang and Ledley, 2012).

3.1.3 Cache Memory

Knowing that registers are a special type of memory, however, they are limited in numbers, which provides the fastest speed. RAM, on the other hand, is significantly less expensive compared to registers, and can be implemented with easy access in massive quantities. It does have a lower speed, though. To bridge the void, the computer system employs another form of memory called a cache. Figure 3.10 illustrates computer systems memory hierarchy. The memory cache is a small amount of fast memory placed between the memory and the processor to bridge the speed limit between the main memory and the CPU.

(Hwang, 1993). The cache is comparatively inferior to the primary memory. The cache memory operating procedure is that it pre-fetches the data from the main memory and keeps them accessible when needed by the processor. If the prediction is correct, the processor receives the data directly from the memory of the cache without having to access the main memory. It may not be surprising if people ask why cache memory is needed and how it works, and how to estimate the data required before running a program. For example, look at the "block" concept. Assuming you want to add two matrices of M row and N column, you will be required to add M N times. If all data is in the cache, it is referred to as call read hit, this will save extra time to collect data using the cache as opposed to forwarding the address to the address buffer (MAR) and then waiting for MDR data with each addition operation.

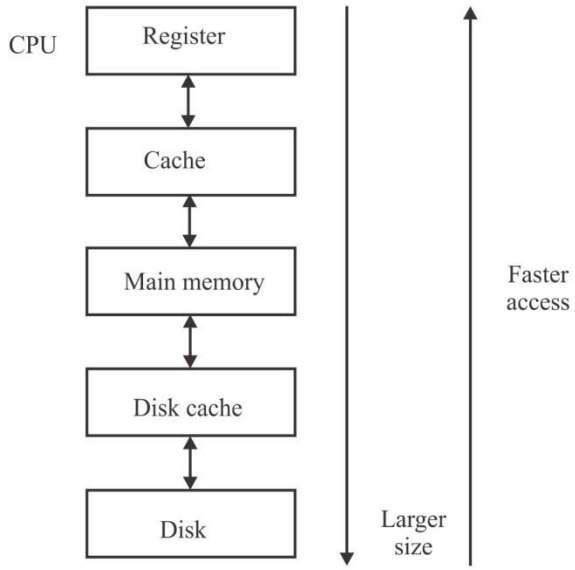

Figure 3.10 Memory hierarchy of the computer system

If the data, for which the processor request is not in the cache, happens, it is called a read miss. The MAR and MDR are therefore enabled, and the data is now read from the actual memory. The estimation algorithm then transfers additional adjacent memory position into cache assuming that there may also be an immediate request for future data. Statistics show that the block data that navigates from the actual memory to the cache when a read miss occurs minimizes the overall access time and thus increases the speed of the computer. Taking the "block" idea into consideration, you can observe the movement of data between cache memory and main memory is actually block-based. Whereas moving data between cache memory and register is word-based, which depends on the data and instructions' actual length. A number of processors, namely primary and secondary, contain two-level caches. Some recent processors have 3 level caches labelled with L1, L2, and L3. At first the processor attempts to reach L1 for the data required, if there is a mistake, then it transfers to L2, if there is also a mistake, then the data should be retrieved from the memory itself. The block chosen from the main memory is also inserted in both caches. Figure 3.11 shows a 2 level cache system.

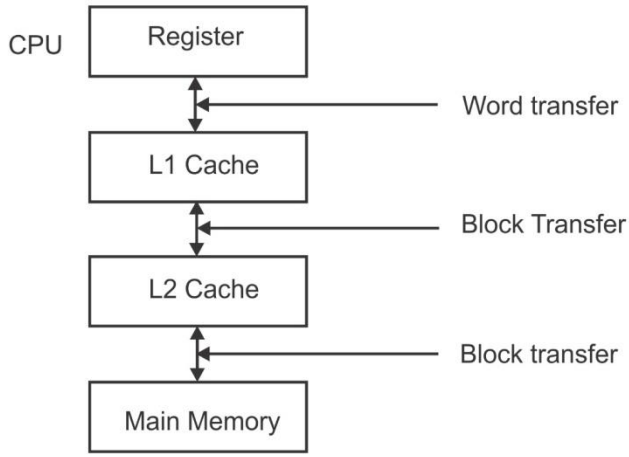

Figure 3.11 Two-level cache memory and data transfer

The memory cache increases the performance of a computer program. By calculating and moving certain blocks from the main memory to the memory cache, the memory access time is greatly improved.

3.1.4 Virtual Memory

By now, you should have understood the significance of the quantity of main memory relating to the system speed. It is common to have a lot of machine programs to run simultaneously. Thus a great quantity of memory is required. However, no matter how big the main memory is, the amount of memory required for executing programs that still not be enough. Digital memory thus provides programs with a large amount of memory as opposed to their physical memory. Occupying part of the storage disk, which is slower. As with memory cache, virtual memory is a change between the main memory and disk storage. In opposite to the memory of the cache, virtual memory does not rely on it and does exist alone. It actually used most of the storage disk. The idea of virtual memory in a computer system is depicted in Figure 3.12.Virtual memory has some common features with a cache memory in terms of improving the access speed. Furthermore, it virtually enlarges the primary memory in other that programs can possibly execute on a system simultaneously without any fear regarding memory constraints.

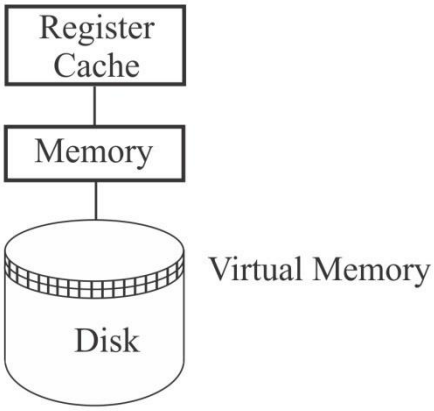

Figure 3.12 Computer memory, virtual memory, and disk storage

Meanwhile, the virtual memory is categorized into 2namely, paged virtual memory and segmented virtual memory (Wang and Ledley, 2012; Dumas, 2006).

3.1.5 Non-Volatile Memory

Majority of the existing computers use RAM as their main memory. RAM is a form of write / read memory that has rapid speed of retrieval and can be easily implemented on a large scale. Conversely, when power is off, RAM is called volatile, which is, it loses data. Even when power is off, the non-volatile memory still holds data. Electrically programmable erasable ROM (EEPROM) and flash memory are called non-volatile memories. Flash memory card with digital circuits was integrated on a regular bus board. Figure 3.13 shows a photo of the non-volatile memory card.


Figure 3.13 Non-volatile memory card invented in 1985

Read-only memory (ROM) is considered as a non volatile however, it is not possible to be modified or re-written. A ROM is written one time only using a special device is named as Programmable ROM (PROM). A PROM in which its data can be erased using ultraviolet light is called Erasable PROM (EPROM). Thus, it is not seen to be random write/read. Another version of the EPROM is the one that uses electricity to clear the whole chip and set to be blank is named EEPROM or E^2PROM, which is the latest modern flash memory.

The existing memory cards for mobile phone or digital cameras and USB flash drive are the new kinds of non volatile memory centered on E^2PROM technology with speed between disk drives and primary memory (Nishi and Magyari-Kope, 2019; Wang and Ledley, 2012).

3.1.6 External Memory

Another external memory name is supplementary memory, or secondary memory. This is storage with high capacities. The external memory is typically greater than the principal memory. One can see the normal external memory on a computer, often called a hard drive. Network storage, USB flash drives and external memory backup drives are all considered. In most situations we call storage external memory. Hard drives are connected via a System Bus to the CPU. Others are connected via serial interface, such as USB flash drives. It takes a lot of time to prepare your address and the data needed to communicate with the CPU. Hence, the hard drive connected to the CPU is not as fast as the one. Offline storage, hard drives, tertiary storage, small computer system interface (SCSI), serial advanced technology attachment (SATA), serial attached SCSI (SAS), network attached storage (NAS), cloud storage and storage area network (SAN) are categorized as external memory (Shiva, 2000).

3.1.7 Memory Access Security

Memory space protection is carried out by enabling memory regions to be non-executable. In such a way that, any effort to execute machine code in those regions will lead to an exception. It employs hardware characteristics, for example, NX (non-executable) bit. If the whole or part of the writable region of the operating system can be marked as nonexecutable, then it will prevent the heap and stack memory areas from being run. This principle helps prevent the occurrence of particular buffer overflow exploits. Specifically, those attacks such as Blaster worms and Sasser which inject and run code. Their attack method relies on some memory part, usually the stack, because they are both writable and executable. If not, otherwise the attack fails. As stated earlier, in an integrated circuit, DRAM keeps data in distinct condensers. To prevent data loss, it needs regular refreshments, at about every 64ms. In action, memory cell condensers typically keep their values very lengthily, particularly at low temperatures. The data in DRAM can be recovered in most cases, even if the DRAM happens not refreshed for quite a few minutes.

The condenser properties can be exploited by the hackers to recover data stored in memory by quickly rebooting the system and depositing RAM content. By cooling the chips and transferring them to a completely new computer. USB flash drives, another technique is the most useful use of external memory for data storage, even sensitive data. The biggest downside of this memory is in physical protection. To enforce data security, users often employ software/hardware solutions to protection data.In the software solution, the protection of the data is usually based on encryption. For instance, in the case of USB drive, several programs are available for encrypting data transparently and automatically. In a

hardware solution, hardware encryption is embedded. In addition, a hardware solution might provide extra attribute By overwriting the memory drive data when more than a specific number of trials are inputted with the wrong password

Segmentation is an additional method that does not only offer extra virtual address however, it offer protection of the information/data kept in the memory segments. Thus, the dynamic data structure used for some arrays and stacks. However, increasing data structure may bump into the subsequent one. When OS uses a single address or in other words assign addresses linearly, it could overwrite one another. Consequently, the overwritten memory could cause system crashes or breach system security. A systematically programmed, overwritten memory that allow hackers to control the computer system by stopping services and then modifying or stealing data. It's usually seen as a buffer overflow attack.

A buffer is a part of the idea of memory and can be used to store user data. For example. In general, buffers have a range of fixed maximum sizes. If the user provides extra input that the buffer can handle, then additional input may end up in unintended memory locations, so the buffer is considered to be in a state of overflow. An execution stack monitors, which type of function programs execute at any given time. Additionally, what function they require after completing the present function to return to. The single address space of the memory is shown in Figure 3.14a. This could overwrite the current software as the stack grows up. Keeping in mind Figure 3.14b, stack, software, and data are held in separate parts. The system segment can be configured to run only and the data segment configured to be non-executable. This will fix the memory overwriting problem and a variety of issues with buffer overflow. Figure 3.15 shows an example. The actual program verifies the password, then open ATM if it is correct. The password is now presumed to be 16 characters long.

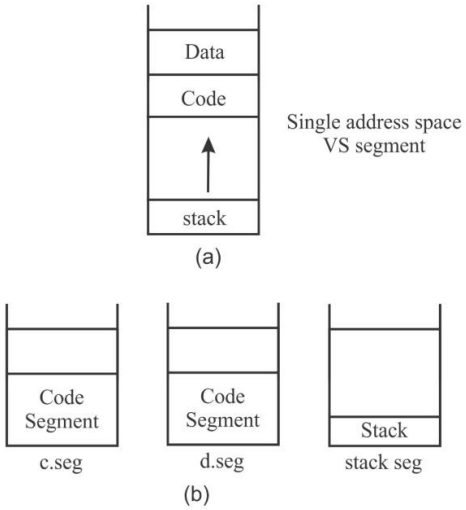

Figure 3.14 Memory Segmentation and Memory Access Security

If the users key-in a password that is greater than 16 characters, then the return address will be overwritten back to the actual system. So hackers might build a 17th to 20th character in such a way that they are exactly the same as the open ATM subroutine address. Hence the ATM will still be opened even if the password were keyed-in incorrect. The 17th to 20th characters created are named as strings of attack

.

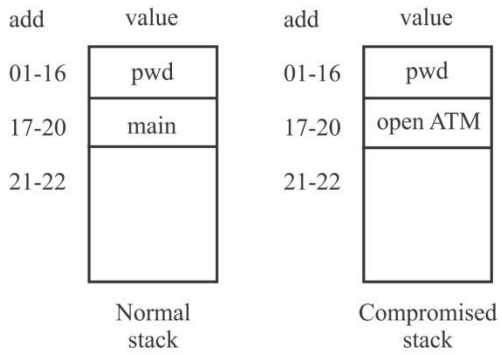

Figure 3.15 Buffer Overflow Attack

Most organizations, for electronic mail and non-vital applications, employ cloud services. Marketing and financial data are important data that are not normally stored in the cloud. The biggest challenge here is data security. Cloud data protection is therefore a very critical and changing topic which needs to be addressed (Wang and Ledley, 2012; Chin and Older, 2010). However, several solutions have been proposed by the researchers in the domain. Yet, there is still a need to explore the problems and provide improved solutions.

 Discussion

The employment of cache is very important in memory system design, discuss.

4.0 Self-Assessment Exercise(s)

1. A unique kind of memory, which is employed to keep a binary value for a short term in order to perform basic storage and calculations, is called?
   A. Cache
   B. Cookies
   C. Register
   D. Buffer

Answer: C

2. A …… is a part of memory concept that can be employed to hold user input before saving?
   A. Register
   B. Cache
   C. Buffer
   D. RAM

Answer: D

5.0 Conclusion

You have learned how to design a one-bit memory circuit in this unit, and have understood how to structure and attach a large memory module through the memory interface. We have researched different memories of their big benefits for each of them as well. In addition, a number of memory protection concerns and how to secure the data stored in memory were addressed in this unit.

6.0 Summary

Different memories were addressed, and a flip-flop was used to create the method for designing a simple memory unit.

DRAM and SRAM are normally located within the computer system. They are volatile but they deliver rapid pace. However, cache memory is typically faster than main memory but low in quantity. It provides useful data access to the CPU by predicting and pre-finding the blocks of data. Virtual memory is a term that shares some similarities to memory cache to expand memory address space and increase access speed. By using virtual memory, programs can run at the same time without fear of memory limitations.

Data encryption can guard non-permitted data access stored on USB flash drive and computers. Hardware protection and encryption can take away data when a hacker is attempting to crack the password at any certain time. Public clouds offer software, infrastructure, and platforms to customers. When moving critical data to the cloud, studies must be carried out to guarantee the security of data.

7.0 References/Further Reading

Kolodner, J. L. (2014). Retrieval and organizational strategies in conceptual memory (PLE: memory): a computer model. Psychology Press.

Ohmaru, T. (2014). U.S. Patent No. 8,773,906. Washington, DC: U.S. Patent and Trademark Office.

Wang, S. P., &Ledley, R. S. (2013). First Edition. Computer architecture and security: Fundamentals of designing secure computer systems. John Wiley & Sons.

Unit 3: Input and Output Design

Contents

1.0 Introduction

Having gone through the memory system design, you will further learn the concept of input and output design in a computer system. The unit involves discussion of the overview on input/output design, which includes direct memory access, interrupts and programmed input/output. The Universal serial bus and IEEE 1394, and network interface card are discussed in relation to memory design. Afterward, the basic NIC architecture, data transmission, and input/output security concepts are explained.

2.0 Intended Learning Outcomes (ILOs)

By the end of this unit, you will be able to:

- Evaluate an input and output design

3.0 Main Content

3.1 Input and Output Design

Input and output (I/O) devices are considered to be important as the central processing unit (CPU). In respective of fastness of the CPU, it is required to take data in and out. In most situations, the destination of the data is remote. Thus, data is often moving in and out of a system via a network interface that manages the exchange of communication.

The I/O devices involve a number of internal registers. Do you know how the internal registers works? These registers are employed for receiving data and command from the computer core processor. Conversely, I/O devices convey data in a distinct manner. Some data are at very slow speed, while others are at high speed. Keyboard feedback is very sluggish in velocity. Thus, it will be built in such a way that the CPU does not waste time waiting for keyboard input (Haris and Haris, 2019; Wang and Ledley, 2012). A universal I/O device diagram is depicted in Figure 3.16.

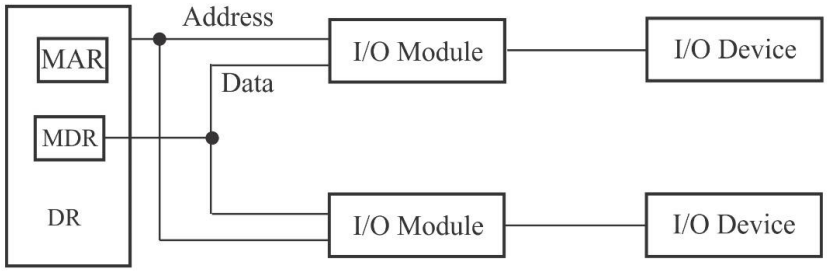

Figure 3.16 Universal I/O Interface Illustration

3.1.1 Direct Memory Access

When you transport large chunks of data from I/O equipment to main memory, consideration must be given to the overhead for any step. Conventionally then, data is read into registers, and further written from the CPU to the memory. Those processes require two stages. Using direct memory access (DMA), data could be written/read straight-away to/from memory without the need for communication with the CPU during transportation (Dumas, 2006). The DMA manager is controlled by the CPU. The processor usually sends data transfer instructions to the DMA. In generality, CPU forwards I/O device numbers, number of bytes and memory start address to the DMA controller. Once this is done the remaining task is accomplished by the DMA controller. Once all the data is read it will cause the forwarding pending. The I/O memory writing process is the same except data transmitted in a reverse direction. In the case, if the forwarding is finished by the DMA controller then there is an interrupt signal to inform the CPU. A typical DMA system also includes an address

register (AR), a data register buffer (DR) and a word counting register (WCR). Figure 3.17 depicts the different I/O diagrams.

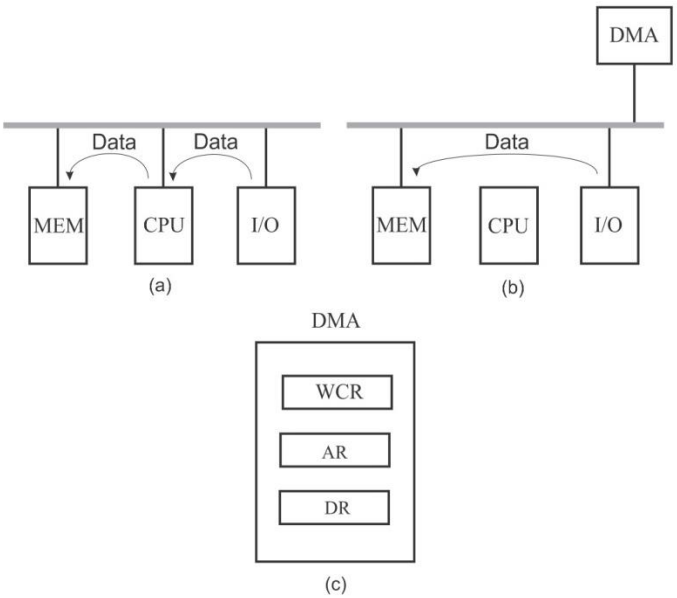

Figure 3.17 I/O Diagrams (a) I/O Data Exchange without DMA, (b) I/O Data Exchange with DMA and (c) Architecture Diagram of a DMA

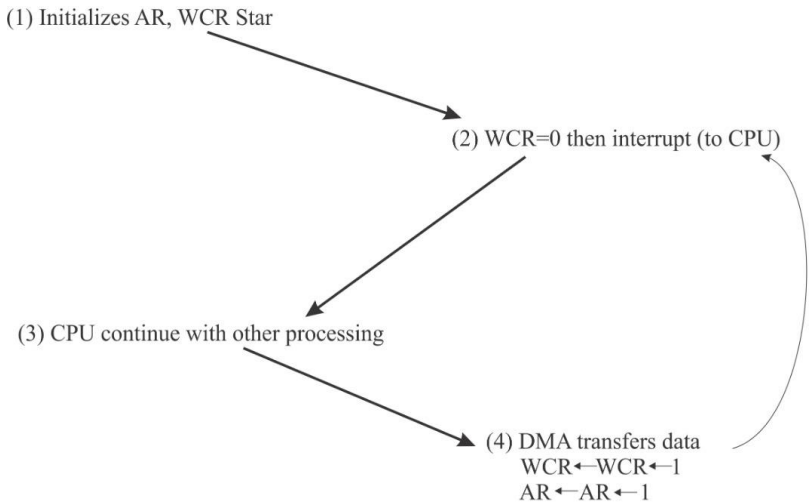

Figure 3.18 DMA Handshake Process

(a) Demonstrate an I/O data swap with no DMA (b) display I/O data swap with DMA and (c) DMA architecture; Transportation of DMA data involves a handshaking procedure. At first, on the DMA side, the CPU initializes the AR and WCR, first, it verifies whether WCR=0. If yes, this means the transition is complete. DMA must forward a message to the CPU for an interrupt. If WCR ≠ 0, then the DMA starts transmitting data. The word-count at every iteration Register reduces by one, and the list of addresses decreases by three. WCR←WCR-1, for example. AR←AR + 1. The data

continues to move until WCR = 0. Figure 5.18 indicates a handshake method with DMA. .

3.1.2 Interrupts

Processor usually run programs without interrupts. In a number of cases, the processor might be interrupted due to the external circumstance to manage several urgent tasks (Haris and Haris, 2019; Foster and Iberall, 1985). Do you know the circumstances that could lead to the trigger of interrupts? The following are unique situations:

- Software requests
- Buffer overflow or underflow
- Power failure
- System reset
- unlawful instruction code
- When DMA data transmission completed.

The interrupt managing process initially keeps the present program counter +1 to a stack and afterwards jumps to the interruption services process. The interruption service schedule often carried out the following functions.

- Handle the interrupt
- Recover from the interrupt
- Explored the stack and fetch the recovered address: PC+1
- Move to PC+1

In the course of keeping the processor condition, the processor might disable further interrupts so as to avoid status change.

3.1.3 Programmed Input/Output

An instance of standard I/O devices includes a keyboard, display, printer, and mouse. The interface, which links to the registers in CPU are developed for a single reason. It is similar to a mouse and keyboard that cannot be placed wrongly on PS/2 interfaces. In several situations, users desire to have an all-purpose I/O module, which could either be configured to input or at other time configured as an output, which relies on the application. CPU first informs the I/O module the type of function it desires it to be, and then send the data.

A programmed input/output module (PIO) is a concept in which all functions are set by instructions. The illustration of PIO connected to a CPU is depicted in Figure 3.19. The I/O address and data registers function alike as MAR and MDR. Both use a single bus, as the memory does. We have two I / O port mapping processes:

i. Input / output coded memory mapping of I/O ports to set aside the memory address space.
   Processors writing to an I/O port in this process are close in writing to the memory

ii. Isolated mapping of input/out does not use any memory space. This employs a distinct speed of I/O emails. A large number of Intel processors, for example, work with isolated mapping. Inside the isolated I/O system, specific I/O instructions are required to access I/O devices. Systems built and developed with processors that support the isolated I/O have the flexibility to use isolated I/O or I/O mapped by memory. For instance, a monitor, keyboard and mouse are often mapped to I/O space whereas a video card might be assigned to block of a memory address via employing memory mapping interface (Wang and Ledley, 2012).

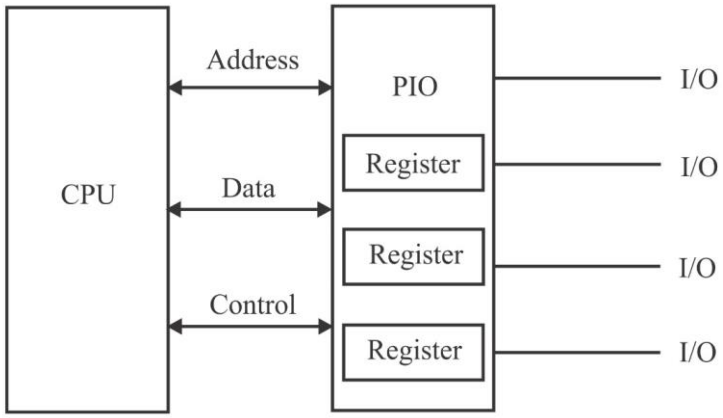

Figure 3.19 Programmed I/O

3.1.3.1 Universal Serial Bus and IEEE 1394

In 1995 the Universal Serial Bus (USB) was created. The main intention of the USB was to define an external expansion bus that would make it as easy to connect peripherals to a device as to link up a phone to a jack.

.Considering the introduction of USB, computer users can connect printer, mouse and keyboard, external memory, hard drives, and camera. The attachment of devices to the USB port can accommodate up to 127 devices. The USB port is designed to avoid setting jumpers and configuring new computers. Users can also hot-plug the USB devices and the program will automatically detect and configure the devices for immediate use. The benefits of USB meanwhile include power delivery, expandable, and power conservation. The USB architecture includes host hardware, consisting of root hub and USB host controller. The host controller generates transmissions over the USB while the root hub provides the points of contact. Figure 3.20 displays USB architecture. A host controller interface (HCI) is known to be a register level interface that enables the hardware of the host controller to communicate with the operating system of a host computer. There exist 3 types of USB host controller interface (Chrysanthakopoulos, 2013; Wang and Ledley, 2012).

- Open HCI (OHCL)
- Universal HCI
- Enhanced HCI

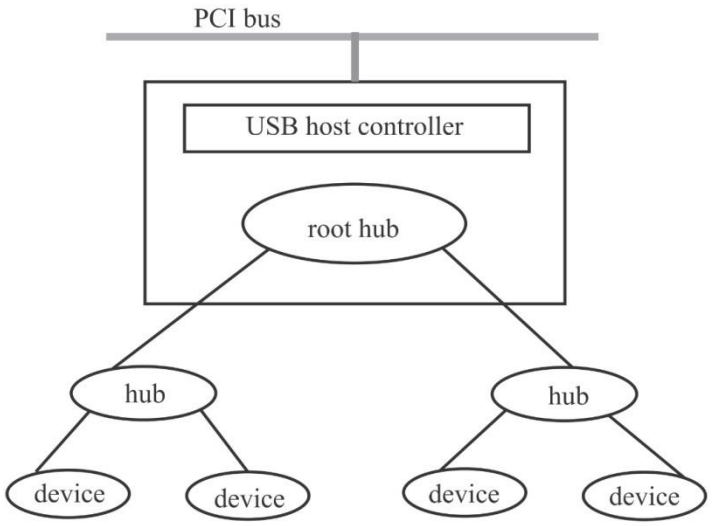

Figure 3.20 USB Host Controller, Root Hub and Hubs

3.1.3.2 Network Interface Card (NIC)

The NIC is considered as a computer hardware component that allows a device to be linked to a machine and a device. Using a specific connection layer and physical layer standard, for example, Ethernet or Wi-Fi, a NIC is embedded with the electronic circuitry required for communication. This provides a foundation for a complete network protocol stack, which facilitates communication between small computer sets on the same local area network. For large computer sets, network communication is done via routable protocols, such as IP.

That and every Ethernet network handler has a specific 48-bit serial number called MAC address which is stored for add-on cards in read-only memory embedded in the card. It's fair to assume that two network controllers with the same address will not exist. The Institute of Electrical and Electronics Engineering (IEEE) stipulates this requirement at the time of fabrication. Ethernet network controllers typically support variations of 10Mbit/s Ethernet, 100Mbit/s Ethernet, and 1,000Mbit/s Ethernet. Those controllers are designated 10/100/1000 which means they can accept a notional maximum rate of transfer of 10, 100 or 1000Megabits per second (Hwang, 1993).

3.5.1 Basic NIC Architecture

As mentioned in Section 3.5, the NIC enables computer communication over an Internet network. It is related in both link and physical layer, which is OSI layer 2 and 1 respectively. It offers physical access to a networking medium, and a low-level addressing system through the use of MAC addresses. It allows computer users to link to each other through wireless or use cables. Most NICs have the device, memory, control logic and medium access control (MAC) DMA interface (Patterson and Hennessy, 2013; Wang and Ledley, 2012).

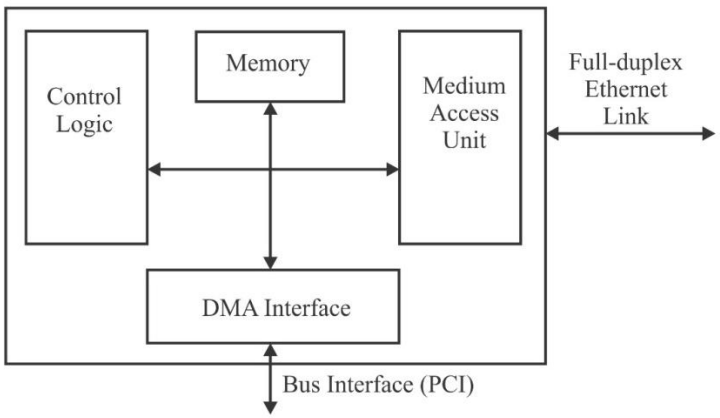

Figure 3.21 NIC Architecture Block Diagram

3.5.2 Data Transmission

NIC allows frames (data) to be transmitted and received between network and host operating system. Frames are received and submitted within a sequence of steps performed by the host and NIC. The host operating system (OS) usually managed a series of buffers which are used for headers and contents of frames. The OS also maintains a queue or ring of descriptors to the buffer. Each buffer descriptor specifies the place a buffer resides in host memory and it size of the buffer is (Patterson and Hennessy, 2013).

3.6 Input/Output Security

What are the three applications of security in I/O devices? I/O security typically varies from security of application to physical protection and security of service. Security threats could negatively impact all devices used for input and output. An unlocked keyboard is an example of physical protection. An unencrypted hard drive can disclose sensitive information to unauthorized users. The following measures can be employed for securing I/O devices (Wang and Ledley, 2012).

- Disabling certain key combinations
- Anti-glare displays
- Adding password to the printer
- Disabling bootable USB port
- Encrypting hard drives

Discussion

List and discuss the input/output devices you use in your office computer or personal computer

4.0 Self-Assessment Exercise(s)

1. What is DMA?
   A. Dynamic Memory Access
   B. Direct Memory Access
   C. Domain Memory Access
   D. Duration Memory Access

Answer: B

2. What are the three applications of security in I/O devices?
   A. Application security
   B. Physical security
   C. Operation security
   D. Technical security

Answer: A, B, C.

5.0 Conclusion

We examined the various types of input and output devices within this package. I/O systems have a set of internal registers. The registers are used to get instruction and process data. When using DMA technology, data transmission between I/O devices and memory can be directly achieved without the need for CPU resources. To address the low-speed problems that can slow down CPU operations in certain cases, interrupts are then used to prevent I/O devices from occupying CPU until the I/O job is complete or their I/O buffer register is complete.

6.0 Summary

The I/O devices include USB (connecting printer, keyboard, etc.) and network interface card. USB is a basic I / O connector and on many I / O applications it has become a touchstone port. The USB 3.0 will send and receive data at a speed of 400 MB/s. The Network Interface Card (NIC) is considered to be a special I/O device that regulates the transmission of data over the network. An MAC address is given to each and every NIC, which is a distinctive address for identifying devices on the Internet. However, some computer expatriate has a reservation on classifying the NIC as an I/O devices, because of its importance and special applicability. Further, NIC is different from regular I/O devices such as a mouse, printer, display and keyboard.

Conventional I/O security problems aren't difficult to handle. Network-related security is, however, a very complicated and detailed subject, which has attracted much interest in the research community.

7.0 References/Further Reading

Wang, S. P., &Ledley, R. S. (2013). First Edition. Computer architecture and security: Fundamentals of designing secure computer systems. John Wiley & Sons.

Chrysanthakopoulos, G. (2013). U.S. Patent No. 8,412,508. Washington, DC: U.S. Patent and Trademark Office..

Patterson, D. A., & Hennessy, J. L. (2013). Computer organization and design MIPS edition: the hardware/software interface. Newnes.

Virtual Lab Activities

The following activities are provided for the 3 units explained in this module

1) Processing unit design

In designing a microprocessor unit, the following general steps for logical process flow need to be considered.

a) Determining the required capabilities of the processor

In determining the required capabilities of the processor, the following questions need to be answered.

i) What are the limitations in terms of speed, price, power and or resources?

ii) Is the chip a general purpose, or an embedded chip or a varied type entirely?

iii) Does the chip have vector and scalar operation capabilities?

iv) Does it have floating point, integer or point arithmetic, or the integration of all three?

v) Will the chip support interrupt?

vi) Is the chip self-contained, or must it interface with several external peripherals?

Further, there is a need to consider if the processor chip will consider wide range of instruction. These instructions might include;

i) Addition/subtraction ii) Multiplication iii) Division iv) Shifting and rotating v) Logical operations: NOT, XOR, NOR, OR AND, etc. vi) Conditional jumps vii) Unconditional jumps viii) Stack operation: pop or push.

b) Layout the data path that will handle the needed capabilities.

In designing the data path, we need to define what the ALU architecture need to use;

i) Accumulator ii) Stack iii) Register iv) A combination of the above 3

c) Determine and state the machine code Instruction Set Architecture (ISA) format

When the basic data path is designed, then we need to create the ISA. Thus, the following need to be considered;

i) What will be the length of the machine word?
ii) How to handle the immediate values?
iii) What types of instructions can take immediate values?
iv) Is the processor CISC or RISC?

d) Build the required logic in order to control the data path.

Once we have our ISA and data path, we can begin to build the logic of the basic control unit. The control unit is actually implemented as finite state machine. Thus, we can try to map the ISA to control unit in a logical way.

- **Memory system design**

In the memory system design, the basic sequential design using flip flop concept need to be employed. In this light, the following steps are considered.

i) Need to obtain circuit state diagram from the design specification
ii) Generate a state table
iii) Select flip flops
iv) Generate circuit excitation table
v) Build K-maps for: flip flop inputs and primary inputs
vi) Get a minimized SOP equations
vii) Draw logic diagram
viii) Simulate to verify design and debug as required
ix) Conduct circuit analysis and logic optimization.

- **Input and output design**

When designing an I/O mechanism the input and output design interface is very important. I/O interface is driven by a processor, which is communication via bus. Thus, the following need to be considered in the design.

i) The required logic need to be built for interpreting the devices address created by the processor.
ii) There is need to implement handshaking for the I/O interface based standard commands e.g. READY, BUSY and WAIT
iii) The interface must be able to convert parallel form to serial data and vice versa, in the case of exchanging different data formats via an I/O interface.

iv) The interrupt mechanism needs to be built for the I/O interface design to avoid large idle time while waiting for data from input device.

There are other I/O concepts including channel I/O, Port-mapped I/O and direct memory access in memory design.

Module 4: Security Design Principles

Module Introduction

This module introduces you to the secure component design of the computer system. discuss thefundamental principle of secure design, the principle ofsoftware security, design principle for protection mechanism and trusted computingbase.

Unit 1: Principles of secure design
Unit 2: Principles of software security
Unit 3: Principles of protection mechanism
Unit 4: Trusted security base

Unit 1: Principles of secure design

Contents

1.0 Introduction

In this unit, you will learn the fundamental principles for secure design in a computer system. The unit includes an overview of the principles of secure design, followed by an explanation on attack surface and elements of the principles.

2.0 Intended Learning Outcomes (ILOs)

At the end of this unit you are expected to be able to carry out the following:

Design security from the start and should be able to structure the security-relevant features

3.0 Main Content

3.1 Overview

It is almost a daily happening that we hear from the news regarding the organization computer attack. The security organization is rapidly rising and awareness is also increasing. However, there are still several cyber attacks occurrence. So how does this keep happening? The possible answer to this question might not be easily integrated into a single concept. Because security is a multifaceted concept, and also the causes of security collapse are complex. Nevertheless, there is a general point that drives the security breakdown, which is ineffective adherence to secure design principles. In the next section, we will discuss the fundamentals of an attack surface.

3.2 Attack surface

Having considered the overview regarding security, we are going to discuss the attack surface in relation to the principles of secure design. Do you know what attack surface is? Well, the attack surface is considered as the set of possible ways an application can be attacked. It is employed in measuring the attack-ability of a system (Chapple *et al.,* 2018).

- The bigger the attack surface of an application, the more probable an attacker is to take advantage of its vulnerabilities and the more damage is likely to result from the attack.
- Contrast to measure vulnerability by keeping track of a number of reported security bugs.
- Both the size of the attack surface and a number of security bugs are useful for measuring security however, they have different meanings.

Attack surface reduction

- Minimize code that executes by default, by disabling features that not all users need.
- Provide restriction to who can access the code, by demanding for authentication to access and request only admin to access unsafe functions.
- Reduce right (privilege) level of code, by preferring code running as the user to admin and prefer to Set Group ID (SETGID) and Set User ID (SETUID)

3.2.1 Elements of principles for secure design

In this subsection, I will take you through those elements that are related to the principles of secure design. These concepts include build protection from the outset, allow potential security changes, reduce and separate security power, use the least privilege, layout the safety-related features, render security friendly and do not rely on security secrecy (Chapple *et al.,* 2018).

i. Design security from the start

At the design stage, you need to consider how to architect defenses. These defenses must be integrated at the foundation level of the design. Emphasis should be made and analyze why a certain system exists, for which subjects, for what task purpose and thus, which resources will be accessible through the system. Then decision is made that best combine the relevant secure design principles into the system design (Conklin et al., 2015).

ii. Allow for future security enhancements

The system needs to be designed in such a way that, it can accommodate continuous security improvement for a long period of time. Security is an ongoing procedure that never ends. Adversaries would invent fresh techniques, previously unknown vulnerabilities in extensively used components will be discovered, and the regular development of fresh features will all integrate to continually change the attack surface. Therefore, you need to design a system that is flexible to modification.

iii. Minimize and isolate security control

In the secure design, you need to reduces the amount shared systems by extra of one user (subject) to gain access to the data resources (objects). In addition, you need to inhibit communication and encapsulate the system components. The security control of the system should be secluded from other component code of the data. For instance, serving an application on the Internet give room to both users and attackers. It enables them to share the Internet in order to achieve access to the system application. If it happens that the attackers launch a distributed denial of service (DDOS) attack and overload the application, the legitimate users will be unable to access the application. In another example, providing the same login page for customers, partners, and employees to login to your company portal, the login page should be an architect to the satisfaction of every user. However, based on this principle, the idea would be to implement different login pages for a different type of users.

iv. Employ the least privilege

It is important to minimize the number of rights to be assigned to a system (subjects). The subject should only have the right required to

finish a task. The default function should lack access. If access is required temporarily, then it should revoke right after task completion.

v. Structure the security-relevant features

The security-related features should be ordered in an advantageous way during the security design. Partially organized dependencies say that synchronization calling and other device dependencies will be structured in part. The fundamental method in the design of the system is layering, whereby the system is organized into functionally interrelated modules or components. And the layers are organized linearly in spite of inter-layer dependencies. However, the structural design concepts influence the core device architecture because of the manner in which the components bind to each other and the nature of their interfaces.

vi. Make security friendly

In the security design of a system, the security should not negatively affect subjects who adhere to the rules. Do you know that a complex security system could lead to unfriendly security system? Well, the system should be simple for subjects to provide and restrict access. The established defaults must be reasonable. A secure system that is not simple will be cumbersome. Thus, the system will be complex and difficult when trying to enhance the system in future.

vii. Do not depend on secrecy for security

You must recognize that system security credibility can not be based on implementation confidentiality. The principle of safe design does not, however, rely on the ignorance of potential attackers. The secure design must be carried out in such a way that, the secure concept is decoupled from the protection keys. This help in allowing the secure system to be assessed by different reviewers without concern that the review conducted might compromise the security (stalling et al., 2012).

Discussion *Discuss on the possible attack surface of Nigeria banking industry.*

4.0 Self-Assessment Exercise(s)

1. The integrity of system security should not depend on?
   A. The secrecy of the design.
   B. Architecture
   C. Program design
   D. Review

Answer: A

5.0 Conclusion

In this unit, we have studied the principles of secure design, which are the concept employed in the secure design of a computer cybersecurity. The main aim of the principles is to enhance the security of a system by achieving the three major keys concepts of secure computing. You can believe with me that, in the process of learning the course unit, we have seen that most of the principles are human perceptional-based ideas, which have proven to be useful in attaining software security design.

6.0 Summary

Design principles for a secure system, are considered as those rules that assist in achieving the three concepts of security. These concepts include integrity, confidentiality, and availability. The attack surface is the possible ways an attack can be carried out on the system, which is based on the level of vulnerabilities. Further, the related elements of the principles for secure design are discussed.

7.0 References/Further Reading

Chapple, M., Stewart, J. M., and Gibson, D. (2018). (ISC) 2 CISSP Certified Information Systems Security Professional Official Study Guide. John Wiley & Sons.

Conklin, W. A., White, G., Cothren, C., Davis, R., & Williams, D. (2015). Principles of computer security. McGraw-Hill Education Group.

Schroeder, M. D., Clarck, D. D., and Saltzer, J. H. (1977). The Multics kernel design project. Proceedings of Sixth A.C.M. Symposium on Operating System Principles pages 43– 56.

Stallings, W., Brown, L., Bauer, M. D., & Bhattacharjee, A. K. (2012). Computer security: principles and practice (pp. 978-0). Upper Saddle River (NJ: Pearson Education.

Unit 2: Principles of software security

Contents

1.0 Introduction

Having understood the principles of secure design in unit 1 of this module, we now need to go through the fundamentals of principles for software security. The unit comprises of overview, what are principles of software security, and the elements of principles of software security.

2.0 Intended Learning Outcomes (ILOs)

At the end of this course unit, you are expected to be able to carry out the following:

Manage real-life problems from security threats.

3.0 Main Content

3.1 Overview

The description of computer security at a high level is seen as a continuous process of dealing with the 3 key concepts of software security on multiple layers of the system. Do you know the major key concepts of software security? Well, this has been mentioned in the introduction of module 4 of this course. By the way, the key elements include availability, confidentiality, and integrity. Integrity in a software program is the concept of ensuring that only permitted subjects can access and operate on software information only by approved methods and processes. For example, the software security concept allows sales personnel to update

only their own leads within the system in a lead management application. Though, sales manager will be able to track and update all leads. If every sales staff can update all the leads or someone else, then there will be integrity violation (Gegick and Barnum, 2005a).

Confidentiality in the software system can be considered as an idea of avoiding unpermitted access to certain software tools or information. If the confidentiality concept is built-in a software system, only the permitted information/tools would be accessible. For example, looking at the sales lead management that is well secured, each sales personnel cannot access or even update someone else's lead, thus we can say that the system is confidential. The other sales personnel should not know regarding the leads let alone accessing it.

The idea of allowable objects being able to reach and use the program at any time is included in the applications. Let's dig at the recruitment system for selling leads. When software system is running on a web server so Internet Protocol (IP) limitation may be placed in order to limit access to the software system depending on the IP address requested. If all sales staff were to access the information network from the 192.168.1.20 IP address in this situation, so refusing access from all other IPs would have to ensure that unauthorized access to the system is stopped until the correct subject may access the system from an approved location. Essentially, if the requesting subject does not come from a authorized IP address then the device may appear to them inaccessible. This is one way of verifying where a program is accessed (McLaughlin and Gogan, 2018).

The number of principles of software safety design has been described as useful when incorporating security concepts into the software framework. These principles of various strategies allow systems to achieve key safety elements based on general architectural models.

3.1.1 What are the principles of software security?

Security of the expression may have many definitions, depending on the sense and perspective in which it is used. Security from the system/software design and development point of view is seen as the ongoing process of managing the integrity, availability and confidentiality of a sub-program, program, and system data.

The description of computer security at a high level is seen as a continuous process of dealing with the 3 key concepts of software security on multiple layers of the system. The term principle is considered a fundamental truth. Thus, principles of software security are those concepts upon which software is implemented for it to be resilient against attack.

3.1.2 Elements of principles of software security

In this subsection, I will take you through those elements that are related to the principles of software security. These principles include secure the weakest link, practice defense in depth, fail securely, follow the principle of least privilege, compartmentalize, keep it simple, promote privacy, remember that hiding secret is hard, be reluctant to trust and use your community resources (Williams, 2019).

i. Secure the weakest link

Typically, experts need to prove that security is seen as a chain; and just as a chain, it's just as strong as the weakest link. So the protection of software is only as strong as its weakest part. Attackers focus on the simple aim, rather than attempting to break encryption by going to endpoints. They'll try to crack an application detectable through the firewall, for example, rather than the firewall itself. Knowing when the weak spots of a software system have been reinforced will mean to a software vendor that the system is well protected enough for use (Gegick and Barnum, 2005b).

Let me give you another example in a real-world security issue. It is generally known that there is more money stored in a bank than in a grocery store, which one did you think arm robbers might likely attack? The grocery store, because the banks are equipped with stronger security precaution in term of tools and personnel. The grocery store, therefore, is seen as a much easier target. The payoff for successful robbery of a grocery store is of course much lower than attacking (robbery) a bank. Yet it could be much easier to run from the crime scene of the grocery store. You then need to find weak links and toughen them before a reasonable level of risk is reached.

ii. Practice defense in depth

Using complex defensive strategies to escape complete compromise if one layer turns out to be inadequate, then another layer would be able to prevent complete compromise. The theory of security in-depth design is an idea of layering resource access permission verification into a software framework that decreases the possibility of a successful attack. In the layered resource system (object), authorization allows unauthorized subjects to bypass any attempt at permission to access a resource (McLaughlin and Gogan, 2018).

For example, firewall to guard subnet, meanwhile, information on the subnet is encrypted. In another example, instead of allowing a user to login with just a username and password, you could use a Captcha system, an IP check, brute force detection logs of their login attempts and so on (Stalling et al., 2012).

iii. Fail securely

The software designed should be secured even when it failed. This will be able to deny unauthorized access that wants to take advantage of the failed software.

There are several reasons that could make a web application to fail when processing transaction. Probably, a database link failed, or the data inserted from a subject is incorrect. This principle of software security states that applications should fail in a secure manner. Failure of the system should not provide the subject with additional rights, and it should not depict the subject critical information such as database queries or logs (Conklin et al., 2015).

iv. Follow the principle of least privilege

In the design principle of secure software, you must make sure that only the minimum access level required for completing a task is assigned to a subject. Further, only the minimum amount time requirement is allotted. For example, a subject who signed up to a blog application as an "author" must not have administrative rights that allow them to remove or add subjects. Subjects should only be allowed to post news to the application (Carroll, 2014).

v. Compartmentalize

The building block of simple access mechanism is not nothing in terms of complex access. Reduce the amount of damage that can be done by breaking the device up into units. Very few operating systems divide the systems into units, because it is difficult to control and manage. Root privilege is an example of how things should not be done (Schumacher et al., 2013).

vi. Keep it simple

In the secure software design, there is a need to make the implementation as simple as possible. Because complex design is often not easy to understand and maintain in the case of error occurrence (Schumacher et al., 2013).

vii. Promote privacy

In the secure software design, attempt not to do anything that compromises the privacy of the subject. The privacy should be promoted because it is often tradeoff against usability. For example, enabling the system to forget credit card number after use. Meanwhile, subjects hate having to type it in every period. The privacy need also to be extended to code and the system. Thus, there is no need to provide more information than necessary.

viii. Remember that hiding secret is hard

Security via obscurity is a poor security control and is close to constantly fail when it happens to be the only control. This does not mean that keeping secrets is a poor concept; it merely means that keeping information confidential does not depend on the reliability of key systems. The protection of a software program, for example, must not be dependent on knowledge of the source code being kept secret. Protection will rely on many other factors, including security depth, business transaction limits, sound network design with password policies, and fraud and audit controls. A practical example is Linux, its source code is widely available and yet Linux is a resilient, secure and robust operating system when properly secured (Conklin et al., 2015).

ix. **Be reluctant to trust**

Instead of making conclusions that must be true, you would be hesitant to expand the trust. In some situations, it is prudent not to trust yourself. Just because a certain security characteristic is a promising standard does not mean that it really makes sense.

A loyalty service company, for example, offers data that Internet Banking uses, makes available the amount of incentive rates, and a short list of potential redeemable goods. Nonetheless, the data must be reviewed to ensure the end subjects are safe to present. The reward values are also a positive figure, and not impossible to be high (Bertino, 2005).

x. Use your community resources

In designing secure software, repeated use of the application promotes trust. Also, public scrutiny promotes trust. Thus, you need to first of all test the software by providing it to the community. So that subjects can use and report on the security loop holes (Saltzer, 2011).

Discussion

Assuming you are a security expert, and you have been employed to study, identify and strengthen the weakest links in a software system. Discuss how will you intend to achieve that?

4.0 Self-Assessment Exercise(s)

1. In the secure software design, there is a need to make the implementation as simple as possible because:
   A. Repeated use of the application promotes trust. Also, public scrutiny promotes trust.
   B. Because complex design is never easy to understand and maintain in the case of error occurrence.
   C. Attempt not to do anything that compromises the privacy of the subject

Answer: B

2. Which of the following is correct about the design principles of secure software?
   A. Repeated use of the application promotes trust. Also, public scrutiny promotes trust.
   B. It is prudent not to trust yourself. Just because a certain security characteristics is a promising standard does not mean that it really makes sense
   C. The privacy should also extend to code and the system.
   D. The security must depend on many other factors.

Answer: A

5.0 Conclusion

In this unit, we have studied those security design principles that are integrated into the software development concept. The complete aim of the principles is to improve or achieve the three major keys of secure computing, which include integrity, confidentiality, and availability. You can believe with me that, in the process of learning the course unit, we have seen that most of the principles are human perceptional-based ideas, which have proven to be promising in achieving software security design.

6.0 Summary

The architects and developers need to consider the perimeter of the security requirement when designing and implementing a system that needs to satisfy a security quality attribute. Even, the least needed protection features need to be considered. However, not every system will continue to follow all the fundamentals of the concepts of security architecture, but can use one or more variations. That is dependent on the standard for software protection set by the client and programmer as the inclusion of protection in a system introduces an additional layer to the whole system, which may have a detrimental effect on performance.

That is the reason it requires the idea of a minimum security requirement when constructing a program. That the quality features must be listed alongside the other quality features of the information program, so that the program in question obeys all values dependent on the quality preferences. The ten principles of information security architecture have also been outlined and deemed to be the frameworks used in the achievement of protected applications.

7.0 References/Further Reading

Carroll, J. M. (2014). Computer security. Butterworth-Heinemann

Conklin, W. A., White, G., Cothren, C., Davis, R., & Williams, D. (2015). Principles of computer security. McGraw-Hill Education Group.

Barnum, S. and Gegick, M. (2005). Least Privilege. Retrieved on August 28, 2011 from https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/principles/351-BSI.html.

Gegick, M. and Barnum, S. (2005). securing the weakest link: revised 2013: digital, Inc. 2005-2007. Digital retains copyrights to this material) (https://www.us-cert.gov/bsi/articles/knowledge/principles/securing-the-weakest-link).

Unit 3: Design principles for protection mechanism

Contents

1.0 Introduction

In unit 2, we have studied those principles that are related to software design. We now need to go through the fundamentals of principles for protection mechanism. The unit comprises of overview, what are the principles of protection mechanism, and the elements of principles for a protection mechanism.

2.0 Intended Learning Outcomes (ILOs)

In this unit, you are expected to be able to do the following:

- Design principles to manage specific real-life scenarios

3.0 Main Content

3.1 Overview

Protection mechanisms of a system are concepts used for regulating rights of a subject for accessing objects of the system. The mechanisms are general expected to have stringent restrictions and to be simple. For the restriction, it is by minimizing interactions/ access and by inhibiting communication. In terms of simplicity, it should be easy to understand, fewer inconsistencies and less wrong. The fundamentals of principles for the protection mechanism are explained subsequently.

3.2 Principles for protection mechanism

Protection mechanism is a security tool that enforces some chosen security policies for various subjects on the system. What do you think will happen if there is no protection mechanism in a system? Well, without a protection mechanism, it is not possible to prevent subjects from gaining full access to the system. Each subject will be able to add content, remove content and also make big changes to the system without restriction. Mechanisms of protecting are the elements that have a straight connection with content and threat. While several other elements of threat are in the whole process, these mechanisms are situated at the business end of technical security (Smith, 2012). The protection mechanism enables the prevention of disastrous problem which might be caused by untrustworthy software. A protection mechanism is the controls system designers can build right (privilege) into their systems. The technical mechanisms involve 5 concepts namely, abstraction, layering, process isolation, data hiding and hardware segmentation.

3.2.1 Elements of principles for a protection mechanism

In this subsection, I will take you through those elements that are related to the principles for a protection mechanism. These principles include least privilege, economy of mechanism, complete mediation, open design, separation of privilege, least common mechanism, psychological acceptability, and fail-safe default (Bishop, 2003).

i. Least privilege

The protection mechanism of a system should have the rights (privileges) necessary to achieve the subject task. Lack of access should be the default setting. Whenever access is needed temporarily, the right after usage should be rescinded. The theory effectively restricts the harm that may occur from a mistake or accident. It also minimizes the amount of potential contacts for correct service between privileged services to the lowest, meaning that unintended, accidental or inappropriate uses of privilege are less likely to occur. Consequently, if a concern occurs about the abuse of a right, it minimizes the amount of resource systems that need to be audited. In another word, where a system may provide "firewalls," the least privilege concept gives a justification for where to install the firewalls. Thus, An example of this principle is the "need-to-know" of the military security rule (Stallings et al., 2012).

ii. Economy of mechanism

The protection mechanism need to be adequately small and simple as to be assessed and developed/implemented, keep it as simple and small as possible (KISS). For example, the security kernel. The detail of the security kernel can be found in unit 4 of this module. A complex mechanism may not be correct:

- Modeled
- Configured
- Understood
- Used
- Implemented

This familiar principle is applicable to every aspect of a system. However, of this purpose, it needs the emphasize on protection mechanisms: architecture and deployment mistakes triggering unintended access paths that not be found during common usage because common usage sometimes does not find attempts to practice inappropriate access pathways. Because of the above factor, techniques like line-by-line software inspection and physical hardware evaluation that establish protective mechanisms are needed. Small and non-complex architecture is necessary for these techniques to thrive.

iii. Complete mediation

Every access to each object must be checked in the protection mechanism, and must be efficient. Further, normal run-time the protection mechanism must be during:

- Initialization
- Shutdown
- Restart

When you apply this principle in a step-by-step order, is the main foundation of the protection system. It forces a system-wide view of access management and control that includes initialization, recovery, restart shutdown, and maintenance in addition to typical operation. It means a foolproof method must be established to establish the source of each order. In fact, it also demands that such initiatives improve efficiency by sceptically evaluating the result of an authority check. If there is a variation in authority, the recalled outcome must be regularly modified in a scientific way.

iv. Open design

Protection mechanism does not rely on design secrecy. "Health by Silence" is not a smart idea. Open concept security system would be open to Public inspection. Additionally, it's better to have a fellow/friend checking for an error than a foe. The structures are not to depend on the stupidity of possible attackers. It may depend, however, on possession of unique, less complicated safe passwords or keys. Thus, the decoupling of security mechanisms from protective keys helps several reviewers to test the mechanisms without worrying that the analysis undertaken can weaken the safety.Furthermore, any skeptical subject may be certified to make himself convinced that the system he is about to use is sufficient for his purpose. At last, trying to maintain confidentiality for any program which receives broad dissemination is essentially not possible.

v. Separation of privilege

The separation of rights in the protection process need to be carried out in such a way that access to objects depends on meeting more than one condition

- Separation of duty
- Two-person rule

Separated keys are valid in a computer system in any situation where two or more requirements have to be met before access is permitted. For instance, systems that provide user-extendible protected data types usually rely on privilege separation for their implementation. This concept is widely used in safe deposit boxes for banks. This is often used in the security framework that a nuclear weapon can be fired only if two separate custodians give the correct order (Gupta et al., 2016).

vi. Least common mechanism

Is good to reduce the amount of mechanism that is specific to more than one user and should be set for all subjects to rely on. Each shared mechanism is thus a potential path to information. Each shared mechanism (particularly one involving shared variables) signifies a possible path of information between subjects and should be designed with great concern to ensure that safety is not compromised by accident. Additionally, every process that covers all subjects should be endorsed to the subject's satisfaction. A job presumably tough than just satisfying one or a few subjects. For example, given the option to implement a new function as a surveillance procedure shared by all subjects. Moreover, it can serve as a library practice which can be done as though it were the subject's own, choosing the latter course. Meanwhile, if one or a few subjects fail to meet the function 's certification level. Then they can offer a replacement, or not use it at all. They will stop being affected by a error in it anyway

vii. Psychological acceptability

The user interface needs to be simple to use in the security system so that subjects plan and apply the mechanism correctly and automatically. Else, they will be bypassed. Security mechanisms must not contribute extra to the difficulties of accessing resources. Furthermore, mistakes should be minimized to the degree that the mental picture of the subject's intent of protection suits the methods that the subject will use as such. If the subject has to translate his picture of his security protection requirements into a language of specifications that is fundamentally different, the subject may make some errors.

viii. Fail-safe default

Whenever an action fails, systems should be as safe as when the act started. Expectedly, the default must be lack of access. There is a need to deliberate on why a user should have access. However, there is no need

to deliberate on why a user should not have access. The substitute, in which protection mechanism seek to identify conditions in which access should be refused, provides the wrong psychological basis for safe system design. A traditional design has to focus on arguments why objects have to be accessible, rather than why they should not. Some objects will be looked into inadequately within a huge system. A default lack of authorization is therefore safer. A design/development or implementation flaw in a system that offers clear authorization appears to fail by refusing permission, thus, a safe situation because it can be detected easily. At the other hand, by requiring entry, a design/creation or execution flaw in a system that explicitly prevents entry appears to fail. It is a failure which in normal use can go unnoticed. This principle refers both to the outward appearance of the protection mechanism and its fundamental implementation

Discussion

Discuss how firewall relates to protection mechanism in a computer system

4.0 Self-Assessment Exercise(s)

1. List all the principles for a protection mechanism
   Ans: i) Least privilege ii) Economy of mechanism iii) Complete mediation iv) Open design v) Separation of privilege vi) Least common mechanism vii) Psychological acceptability viii) Fail-safe default
2. Explain what you understand as default lack permission
   Ans: The term default lack permission means that whenever the system is at default state, then there should be no access to an object in-respective of the privilege level of the subject.
3. In your understanding, explain psychological acceptability
   Ans: In the design of a secured system, the design of user interface must be easy and interactive such that the system will be generally acceptable by the users (subjects).

5.0 Conclusion

In this unit, we have studied the concepts of design principles for protection mechanism in security design. The protection mechanism design principles include least common mechanism, least privilege,

complete mediation, economy mechanism, open design, psychological acceptability, separation of privilege, and fail-safe default. These principles help in achieving optimal protection mechanism in the security design.

6.0 Summary

Design principles for protection mechanism, are those restrictions and policies that are required to achieve a secure design in terms of computing. These principles have been highlighted in the conclusion section. The least privilege principle state that subject (user) must be provided with only those rights that are needed in order to achieve the completion of a task. Economy mechanism principle states that protection mechanisms must be as simple as possible. Complete mediation principle demands that all accesses to object need to be checked to guarantee that they are permitted. The principle of open design states that the protection mechanism should not be dependent upon the secrecy of its design or implementation. The principle of privilege separation states that a system should not grant authorization based on a single condition. The Less Common Mechanism principle states that mechanisms used to access resources should not be shared. In computer protection the concept of psychological acceptability considers the human dimension. It says that protection mechanisms should not make access to the artifacts more difficult than if the security measures were not in operation. Default-safe theory states that even when a subject has clear access to an entity (resource) thus, that object must be denied access.

7.0 References/Further Reading

Gupta, B., Agrawal, D. P., & Yamaguchi, S. (Eds.). (2016). *Handbook of research on modern cryptographic solutions for computer and cyber security*. IGI global.

Smith, R. E. (2012). A contemporary look at Saltzer and Schroeder's 1975 design principles. *IEEE Security & Privacy*, *10*(6), 20-25.

Bishop, M. (2003). What is computer security?.*IEEE Security & Privacy*, *1*(1), 67-69.

Stallings, W., Brown, L., Bauer, M. D., & Bhattacharjee, A. K. (2012). *Computer security: principles and practice* (pp. 978-0). Upper Saddle River (NJ: Pearson Education.

Unit 4: Trusted computing base

Contents

1.0 Introduction

Having gone through the principles of protection mechanism, we need to study the trusted computing base (TCB) concept, which is related to the protection mechanism concept. The unit involves the discussion on TCB, the security perimeter and, reference monitor and kernels.

2.0 Intended Learning Outcomes (ILOs)

In this unit, you are expected to be able to achieve the following,

- Evaluate available trusted computing base system

3.0 Main Content

3.1 Overview

The expatriate of computer security has recognized the significance of human discretion, which depends in the architectural structure proposed in the 1980s named the Trust Computing Base (TCB). The TBC is complete with software, hardware, procedures, and entities whose proper operations and decision-making are deemed necessary for the overall system security. It will include the program and the security officers running the critical safety system at an enterprise. It will also include all structures for the management and preservation of publicly identifiable information (PII) involving employees and customers. In an operating

system (OS), it would involve files, a process in an underlying kernel. Elements that might be excluded from a TCB include any system whose public disclosure or malfunction would not generate a serious or cascading problem. Under existing infrastructure, the TCB applies to provider and affiliate party systems and networks (Candaele et al., 2015). This poses serious difficulties in securing TCB properties, as it stretches the TCB boundary to a more complicated area to handle and control. Subsequently, I would take you through the fundamentals of TCB, which include security perimeter, reference monitor and kernels.

3.2 Trusted computing base

A secured system obeys some designated set of security criteria for protecting system information. A TCB is considered as an integration of software, hardware, and controls that work together in order to form a trustworthy framework for implementing the security strategy that is necessary. TCB is viewed as a subset of a full system of information. In order to allow for a thorough review that will fairly guarantee that the device satisfies design criteria and standards, it should be as few as possible. The TCB is the only component of the device that can be expected to obey the security policy strictly and implement it. Nonetheless, it is not necessary that you have to trust every aspect of the program. However, if you have to consider a program from a security perspective, your evaluation will include all the trustworthy components that characterize the particular system

.

Generally, TCB components in a system are in charge of directing and managing authorization access to the system. It should offer methods for accessing resources both in internal and external components of the TCB. The TCB components also perform the task of ensuring that systems behave properly in all situations and follows the security criteria in all cases. The components generally limit the activities of external components of the TCB. In the subsequent subsections, I will take you through the security perimeters (Stalling et al., 2015).

3.2.1 Security perimeter

Your system's protection perimeter is seen as an artificial border that demarcates the TCB from the rest of the network (see Figure 3.22). The barrier means unsafe connectivity or interference between the TCB and the rest of the operating network will not occur. The TCB needs to establish protected networks, known as the trustworthy path, to communicate with the remaining network. What do you think is a direction of trust? A trustworthy route is a medium generated with a stringent standard to allow required correspondence to take place without exposing vulnerabilities to the TCB. This also prevents device consumers (commonly referred to as subjects) from compromising due to TCB

exchange. In any system that seek to achieve high levels of security to their user, then there is a need for a trusted path.

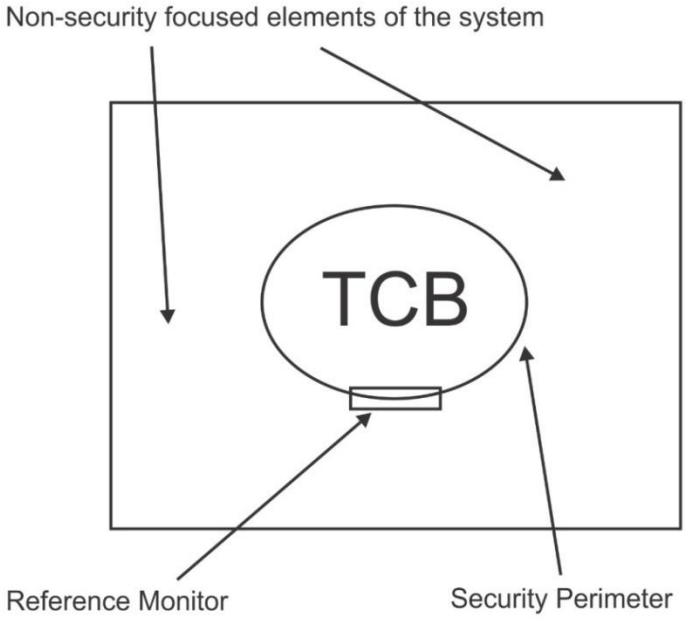

Figure 3.22 TCB, security perimeter, and reference monitor

3.2.2 Reference monitor and kernels

If you want to incorporate a protected program, any part of the TCB must be built and developed to impose access restrictions on network assets and services (referred to as objects). The section of the TCB that validates access to each resource before access requests are allowed is called a comparison check (see Figure 3.22). The reference display is located between each object and subject. Verifying that the credentials of a requesting party meet the access requirements of the entity before the request is permitted to continue. In this case, if the permission requirements are not fulfilled, then the requests are not permitted. The TCB's access control enforcer is essentially the reference controller. It can be a conceptual element of TCB; it may not involve a real, stand-alone, or independent portion of the functioning framework.

The security kernel is referred to as the group of components in the TCB which work together to execute operational functions of the reference monitor. The reference monitor is used as a technique or theory put into motion by applying a security kernel in hardware and software. The security kernel 's purpose is to implement suitable components to enforce operational functionality of the reference monitor and defy all known attacks. The security kernel uses a reliable way of interacting with topics. This also mediates all requests for access to services, allowing only those requests which suit the correct access rules and policies in place for a system The reference monitor has to provide detailed knowledge about every resource it controls. Typically this sort of knowledge includes their recognition and description. If a subject requests for access to an object,

then the reference monitor tests the descriptive detail of the object to decide whether or not access will be permitted (Stewart et al., 2008).

 **Discussion**

Security perimeter of your system is considered to be an imaginary boundary. What do you think ?

4.0 Self-Assessment Exercise(s)

1. The group of components in the TCB that work together to implement reference monitor operational functions is called the?
   A. Reference monitor
   B. Hardware component
   C. Software component
   D. Security Kernel

Answer: D

2. ……is a channel created with a stringent standard to permit necessary communication to occur without revealing the TCB to security vulnerabilities?
   A. Trusted path
   B. Path and Route
   C. None of the above
   D. All of the above

Answer: A

5.0 Conclusion

Within this unit, we have learned the fundamentals of trusted computing base. Such principles include the perimeter of security and the reference point and the kernel of protection. The security perimeter is an artificial line demarcating the TCB from the rest of the network. The reference panel is the TCB enforcer of access controls. The protection kernel's goal is to activate correct components to implement the operational functionality of the reference monitor and to defy all known attacks.

6.0 Summary

The primary components of a Trusted Computing Base (TCB) are the elements of software and hardware used to implement the security laws. The perimeter of protection identifies and demarcates TCB components from non-TCB components and the monitor of reference. This acts as a tool for access control around the protection perimeter. The course unit's goal is to understand the design principles in security design for the trusted computing base.

7.0 References/Further Reading

Candaele, B., Soudris, D., & Anagnostopoulos, I. (Eds.). (2015). *Trusted computing for embedded systems*. Cham: Springer International Publishing.

Stewart, J. M., Tittel, E., & Chapple, M. (2008). *CISSP: Certified information systems security professional study guide*. John Wiley & Sons.

www.sybex.com/go/cissp7e. Retrieved August 2019.

Stallings, W., Brown, L., Bauer, M. D., & Bhattacharjee, A. K. (2012). *Computer security: principles and practice* (pp. 978-0). Upper Saddle River (NJ: Pearson Education.

www.sybex.com/go/cissp7e. Retrieved August 2019

Virtual Lab Activities

The following activities are provided for the 2 units explained in this module

1) Principles of software security

In principles of software security, a virtual lab activity is provided on how to test the security of software in terms of bugs and vulnerability. The following are steps involved in testing open source software security testing methodology manual:

i) Vulnerability scanning
ii) Security scanning
iii) Penetration testing
iv) Risk assessment
v) Security auditing
vi) Posture assessment
vii) Ethical hacking

2) Trusted security base

The focused responsibility of TCB is to maintain confidentiality and integrity of data on system. One of the of the function of TCB is monitoring several system operations namely;

i. Memory protection
ii. Input/out operation
iii. Process activation
iv. Execution domain switching.

For the purpose of virtual lab activities, we will consider the memory protection operation of TCB.

The TCB monitors the following operations for memory protection:

i) Monitor references to the system memory
ii) Monitor calls for verifying the confidentiality and integrity of the systems' data in the storage.