

**FURTHER EDUCATION AND TRAINING CERTIFICATE:
INFORMATION TECHNOLOGY: SYSTEMS DEVELOPMENT**

ID 78965 LEVEL 4 – 165 CREDITS

LEARNER GUIDE

SAQA: 14917

EXPLAIN COMPUTER ARCHITECTURE CONCEPTS

Learner Information:






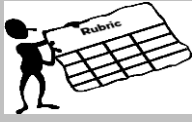



Details	Please Complete this Section
Name & Surname:	
Organisation:	
Unit/Dept:	
Facilitator Name:	
Date Started:	
Date of Completion:	

Copyright

All rights reserved. The copyright of this document, its previous editions and any annexures thereto, is protected and expressly reserved. No part of this document may be reproduced, stored in a retrievable system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior permission.

Key to Icons

The following icons may be used in this Learner Guide to indicate specific functions:

 <p>Books</p>	<p>This icon means that other books are available for further information on a particular topic/subject.</p>
 <p>References</p>	<p>This icon refers to any examples, handouts, checklists, etc...</p>
 <p>Important</p>	<p>This icon represents important information related to a specific topic or section of the guide.</p>
 <p>Activities</p>	<p>This icon helps you to be prepared for the learning to follow or assist you to demonstrate understanding of module content. Shows transference of knowledge and skill.</p>
 <p>Exercises</p>	<p>This icon represents any exercise to be completed on a specific topic at home by you or in a group.</p>
 <p>Tasks/Projects</p>	<p>An important aspect of the assessment process is proof of competence. This can be achieved by observation or a portfolio of evidence should be submitted in this regard.</p>
 <p>Workplace Activities</p>	<p>An important aspect of learning is through workplace experience. Activities with this icon can only be completed once a LEARNER is in the workplace</p>
 <p>Tips</p>	<p>This icon indicates practical tips you can adopt in the future.</p>
 <p>Notes</p>	<p>This icon represents important notes you must remember as part of the learning process.</p>

Learner Guide Introduction

About the Learner Guide...	This Learner Guide provides a comprehensive overview of the EXPLAIN COMPUTER ARCHITECTURE CONCEPTS , and forms part of a series of Learner Guides that have been developed for FURTHER EDUCATION AND TRAINING CERTIFICATE: INFORMATION TECHNOLOGY: SYSTEMS DEVELOPMENT ID 78965 LEVEL 4 – 165 CREDITS . The series of Learner Guides are conceptualized in modular's format and developed FURTHER EDUCATION AND TRAINING CERTIFICATE: INFORMATION TECHNOLOGY: SYSTEMS DEVELOPMENT ID 78965 LEVEL 4 – 165 CREDITS . They are designed to improve the skills and knowledge of learners, and thus enabling them to effectively and efficiently complete specific tasks. Learners are required to attend training workshops as a group or as specified by their organization. These workshops are presented in modules, and conducted by a qualified facilitator.
Purpose	The purpose of this Unit Standard is to Explain computer architecture concepts
Outcomes	Explain computer architecture concepts
Assessment Criteria	The only way to establish whether a learner is competent and has accomplished the specific outcomes is through an assessment process. Assessment involves collecting and interpreting evidence about the learner's ability to perform a task. This guide may include assessments in the form of activities, assignments, tasks or projects, as well as workplace practical tasks. Learners are required to perform tasks on the job to collect enough and appropriate evidence for their portfolio of evidence, proof signed by their supervisor that the tasks were performed successfully.
To qualify	To qualify and receive credits towards the learning programme, a registered assessor will conduct an evaluation and assessment of the learner's portfolio of evidence and competency
Range of Learning	This describes the situation and circumstance in which competence must be demonstrated and the parameters in which learners operate
Responsibility	The responsibility of learning rest with the learner, so: <ul style="list-style-type: none"> • Be proactive and ask questions, • Seek assistance and help from your facilitators, if required.

Learning Unit 1 Explain computer architecture concepts

UNIT STANDARD NUMBER	:	14917
LEVEL ON THE NQF	:	4
CREDITS	:	7
FIELD	:	Physical, Mathematical, Computer and Life Sciences
SUB FIELD	:	Construction Information Technology and Computer Sciences

PURPOSE:	<p>This unit standard is intended:</p> <p>to provide a fundamental knowledge of the areas covered for those working in, or entering the workplace in the area of Information Systems & Technology Management.</p> <p>as additional knowledge for those wanting to understand the areas covered</p> <p>People credited with this unit standard are able to:</p> <p>Explain computer architecture elements</p> <p>Explain the organisation of a computer</p> <p>Describe the design constraints in the design of instruction sets for computers</p> <p>The performance of all elements is to a standard that allows for further learning in this area.</p>
LEARNING ASSUMED TO BE IN PLACE:	
<p>The credit value of this unit is based on a person having prior knowledge and skills to:</p> <p>Demonstrate an understanding of fundamental mathematics (at least NQF level 3)</p> <p>Demonstrate PC competency skills (End-User Computing unit Standards, at least up to NQF level 3.)</p> <p>Demonstrate an understanding of types of computer systems. (SGB-ID= IST004).</p>	

SESSION 1.

Explain computer architecture elements.

Learning Outcomes

- 1. The explanation identifies the functions of elements which make up computer architecture.
- 2. The explanation outlines the functions of elements which make up computer architecture.
- 3. The explanation distinguishes categories of each element and outlines their features.
- 4. The explanation identifies examples of the application of architecture elements.

Computer Architecture

Computer architecture involves the design of computers. Processor design involves the **instruction set** design and the **organisation** of the processor.

Instruction set architecture (ISA) describes the processor in terms of what the assembly language programmer sees, i.e. the instructions and registers. **Organisation** is concerned with the internal design of the processor, the design of the bus system and its interfaces, the design of memory and so on. Two machines may have the same ISA, but different organisations. The organisation is implemented in hardware and in turn, two machines with the same organisation may have different hardware implementations, for example, a faster form of silicon technology may be used in the fabrication of the processor

Introduction

The first point that must be made about computer architecture is that there is no standard computer architecture, in the same way as there is no such thing as a standard house architecture or standard motor car design

However, just as all cars have some basic features in common, so too do computers. In this section, we take a high level look at the components of computer architecture that are common to all computers, noting that any particular computer will differ in various details from the general model presented.

1. The Operation of a computer

Computer: An integrated set of algorithms and data structures capable of storing and executing programs.

Components:

1. Data - types of elementary data items and data structures to be manipulated
2. Primitive Operations - operations to manipulate data
3. Sequence Control - mechanisms for controlling the sequence of operations
4. Data Access - mechanisms to supply data to the operations
5. Storage Management - mechanisms for memory allocation
6. Operating Environment - mechanisms for I/O (i.e. communication)

2. Implementation of the components

Hardware

Firmware

Software

A. Hardware

CPU (Central processing Unit)

Inboard memory

I/O devices - keyboard, mouse, disk drives, etc

Outboard storage - disks, CDs

1. CPU

ALU (arithmetic-logic unit)

Control Unit

Registers (fast memory)

PC - program counter, location counter, program address register, instruction counter

IR - instruction register

2. Inboard memory

Registers (listed above)

Cache memory

Main memory

3. Hardware-level operations

Fetch-execute cycle:

Processor **fetches** an instruction from memory.

Processor **executes** the fetched instruction (performs the instruction cycle)

Steps in fetch-execute cycle:

1. **Program counter**- contains the address of the next instruction to be fetched.
2. The fetched instruction is loaded into **Instruction Register**
3. Program counter is **incremented** after fetching.
4. Processor interprets the bits stored in the IR and **performs the required action**.

Instruction cycle - processing that is required for a single instruction.

The instructions executed by CPU are machine language instructions. Some of them are implemented in hardware - e.g. increment a register by 1. Some are implemented in **Firmware** - e.g. add the contents of two registers.

B. Firmware

Firmware: A set of machine-language instructions implemented by programs, called *micro programs*, stored in programmable read-only memory in the computer (PROM).

Example: The language in problem #4, p.42 consists of very low-level instructions. It would be implemented in hardware (hardwired). The instruction $a = a + b$, could be implemented in firmware.

Advantages:

- flexibility - by replacing the PROM component we can increase the set of machine instructions, e.g. we can implement vector multiplication as a machine operation.
- less cost of the hardware - the simpler the instructions, the easier to hardwire them.

C. Software

Translators and virtual architectures

6. Translation (compilation):

Input: high-level language program

Output: machine language code

Types of translators (high to low level):

Preprocessor:

Input: extended form of high-level language

Output: standard form of high-level language

Compiler:

Input: high-level language program

Output (object language): assembler code

Assembler:

Input: assembly language

Output: one-to-one correspondence to a machine language code

Loader/Link editor:

Input: assembler/machine language reloadable program

Output: executable program (absolute addresses are assigned)

7. Software simulation - Interpreters

The program is not translated to machine language code. Instead, it is executed by another program.

Example: Prolog interpreter written in C++

Advantages of interpreted languages:

- very easy to be implemented
- easy to debug
- flexibility of language - easy to modify the interpreter

- portability - as long as the interpreter is portable, the language is also portable.

Disadvantages of interpreted languages: slow execution.

8. Virtual machines

Program: data + operations on these data

Computer: implementation of data structures + implementation of operations

Hardware computer: elementary data items, very simple operations

Firmware computer: elementary data items, machine language instructions
(we may have specialized processors e.g. vector multiplication)

Software computer: each programming environment defines a specific software computer.

E.G - the operating systems is one specific virtual computer. A programming language also defines a virtual computer.

Basic Hierarchy:

Software

Firmware

Hardware

Software sub-hierarchy - depends on how a given programming environment is implemented

Example of virtual machines hierarchy:

Java applets

Java virtual machine - used to implement the Java applets

C virtual machine - used to implement Java

Operating system - used to implement C

Firmware - used to implement machine language

Hardware - used to implement firmware micro programs

9. Binding and binding times

Binding - fixing a feature to have a specific value among a set of possible values.

E.G. - your program may be named in different ways and when you choose a particular name you have done a binding. Different programming features have different binding times, depending on:

- the nature of the feature, e.g. you choose the names of the variables
in the source code, the operating system chooses the physical address of the variables.
- the implementation of the feature - in certain cases the programmer has a choice
to specify the binding time for a given feature.

Binding occurs at:

At language definition - concerns available data types and language structures, e.g in C++ the assignment statement is `=`, while in Pascal it is: `:`

At language implementation - concerns representation of data structures and operations,

e.g. representation of numbers and arithmetic operations

At translation -

Chosen by the programmer - variable types and assignments

Chosen by the compiler - relative locations of variables and arrays

Chosen by the loader - absolute locations

At execution -

Memory contents

On entry to a subprogram (copying arguments to parameter locations)

At arbitrary points (when executing assignment statements)

Recursive programs

Dynamic libraries

Example: `X = X+10` (see p. 62) Discuss what happens at each stage listed above.

Importance of binding times

If done at translation time - more efficiency is gained

If done at execution time - more flexibility is gained.

Learning goals

- Understand the concept of hierarchical levels of virtual computers and the interplay

between hardware, firmware and software. Be able to discuss translation vs simulation and binding.

Exam-like questions

1. Explain the concept of firmware. What are the advantages of firmware?
2. List the types of translators discussed in class. Describe briefly the input and output of each of them.
3. What is the idea of software simulation? Discuss the advantages and disadvantages.
4. Discuss the concept of virtual machines.
5. What is binding?
6. What types of binding occur at language definition, language implementation?
7. What types of binding occur at program translation and program execution?
8. What is the advantage of bindings performed at translation time? At execution time?

SESSION 2.

Explain the organisation of a computer.

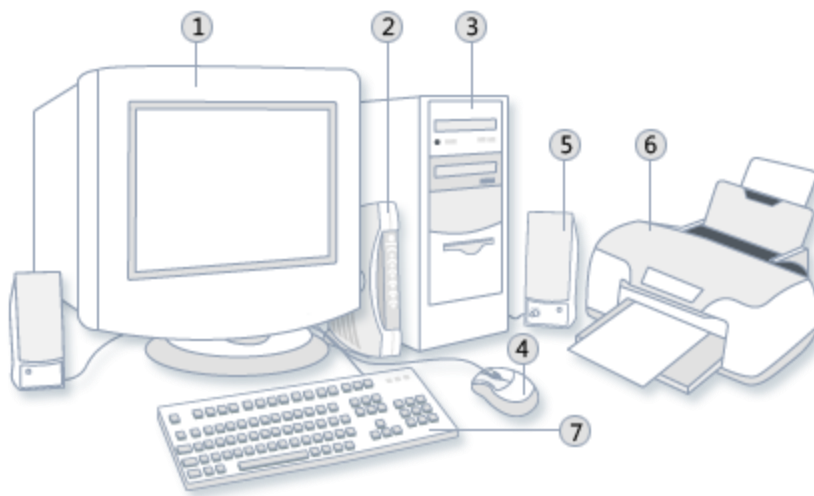
Learning Outcomes

- 1. The explanation identifies the purpose of computer components.
- 2. The explanation outlines how components achieve their outcomes in terms of their relationships, and the structure of the computer.

Parts of a computer

- **System unit**
- **Storage**
- **Mouse**
- **Keyboard**
- **Monitor**
- **Printer**
- **Speakers**
- **Modem**

If you use a desktop computer, you might already know that there isn't any single part called the "computer." A computer is really a system of many parts working together. The physical parts, which you can see and touch, are collectively called **hardware**. (**Software**, on the other hand, refers to the instructions, or programs, that tell the hardware what to do.) The illustration below shows the most common hardware in a desktop computer system. Your system may look a little different, but it probably has most of these parts. A laptop computer has similar parts but combines them into a single notebook-sized package.



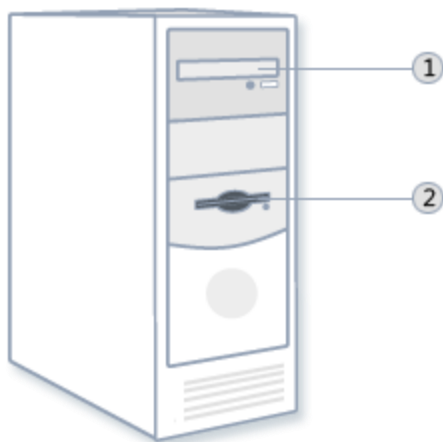
- | | | | |
|-----------|---------------|-----------|------------|
| 1 Monitor | 3 System unit | 5 Speaker | 7 Keyboard |
| 2 Modem | 4 Mouse | 6 Printer | |

Desktop computer system

Let's take a look at each of these parts.

System unit

The **system unit** is the core of a computer system. Usually it's a rectangular box placed on or underneath your desk. Inside this box are many electronic components that process information. The most important of these components is the **central processing unit (CPU)**, or **microprocessor**, which acts as the "brain" of your computer. Another component is **random access memory (RAM)**, which temporarily stores information that the CPU uses while the computer is on. The information stored in RAM is erased when the computer is turned off. Almost every other part of your computer connects to the system unit using cables. The cables plug into specific **ports**(openings), typically on the back of the system unit. Hardware that is not part of the system unit is sometimes called **peripherals device** or **device**.



1 CD/DVD drive 2 Floppy disk drive

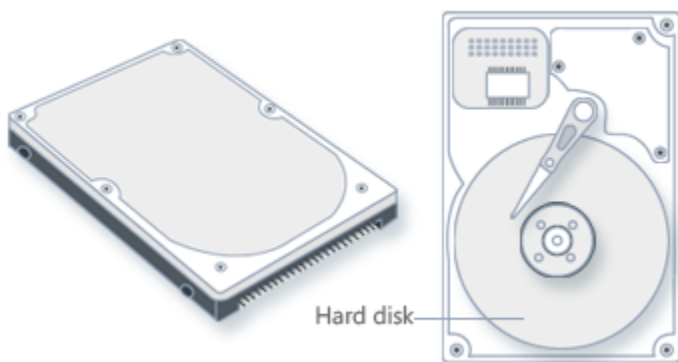
System unit

Storage

Your computer has one or more **disk drives**—devices that store information on a metal or plastic disk. The disk preserves the information even when your computer is turned off.

Hard disk drive

Your computer's **hard disk drive** stores information on a **hard disk**, a rigid platter or stack of platters with a magnetic surface. Because hard disks can hold massive amounts of information, they usually serve as your computer's primary means of storage, holding almost all of your programs and files. The hard disk drive is normally located inside the system unit.



Hard disk drive

CD and DVD drives

Nearly all computers today come equipped with a CD or DVD drive, usually located on the front of the system unit. CD drives use lasers to read (retrieve) data from a CD, and many CD drives can also write (record) data onto CDs. If you have a recordable disk drive, you can store copies of your files on blank CDs. You can also use a CD drive to play music CDs on your computer.



CD

DVD drives can do everything that CD drives can, plus read DVDs. If you have a DVD drive, you can watch movies on your computer. Many DVD drives can record data onto blank DVDs.

Tip

- If you have a recordable CD or DVD drive, periodically back up (copy) your important files to CDs or DVDs. That way, if your hard disk ever fails, you won't lose your data.

Floppy disk drive

Floppy disk drives store information on **floppy disks**, also called **floppies** or **diskettes**. Compared to CDs and DVDs, floppy disks can store only a small amount of data. They also retrieve information more slowly and are more prone to damage. For these reasons, floppy disk drives are less popular than they used to be, although some computers still include them.



Floppy disk

Why are floppy disks "floppy"? Even though the outside is made of hard plastic, that's just the sleeve. The disk inside is made of a thin, flexible vinyl material.

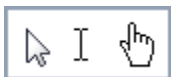
Mouse

A mouse is a small device used to point to and select items on your computer screen. Although mice come in many shapes, the typical mouse does look a bit like an actual mouse. It's small, oblong, and connected to the system unit by a long wire that resembles a tail. Some newer mice are wireless.



Mouse

A mouse usually has two buttons: a primary button (usually the left button) and a secondary button. Many mice also have a wheel between the two buttons, which allows you to scroll smoothly through screens of information.



When you move the mouse with your hand, a pointer on your screen moves in the same direction. (The pointer's appearance might change depending on where it's positioned on your screen.) When you want to select an item, you point to the item and then **click** (press and release) the primary button. Pointing and clicking with your mouse is the main way to interact with your computer.

Keyboard

A keyboard is used mainly for typing text into your computer. Like the keyboard on a typewriter, it has keys for letters and numbers, but it also has special keys:

- The **function keys**, found on the top row, perform different functions depending on where they are used.
- The **numeric keypad**, located on the right side of most keyboards, allows you to enter numbers quickly.
- The **navigation keys**, such as the arrow keys, allow you to move your position within a document or webpage.

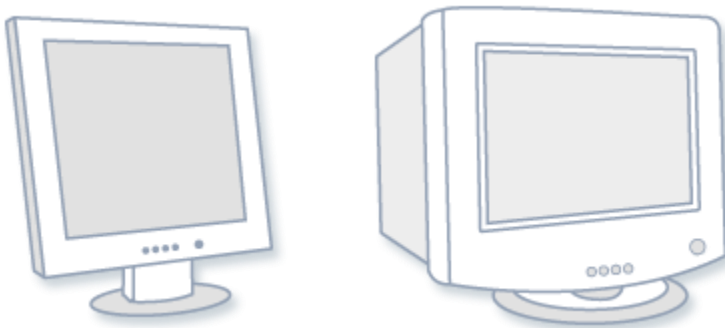


Keyboard

You can also use your keyboard to perform many of the same tasks you can perform with a mouse.

Monitor

A **monitor** displays information in visual form, using text and graphics. The portion of the monitor that displays the information is called the **screen**. Like a television screen, a computer screen can show still or moving pictures. There are two basic types of monitors: **CRT** (cathode ray tube) monitors and **LCD** (liquid crystal display) monitors. Both types produce sharp images, but LCD monitors have the advantage of being much thinner and lighter. CRT monitors, however, are generally more affordable.

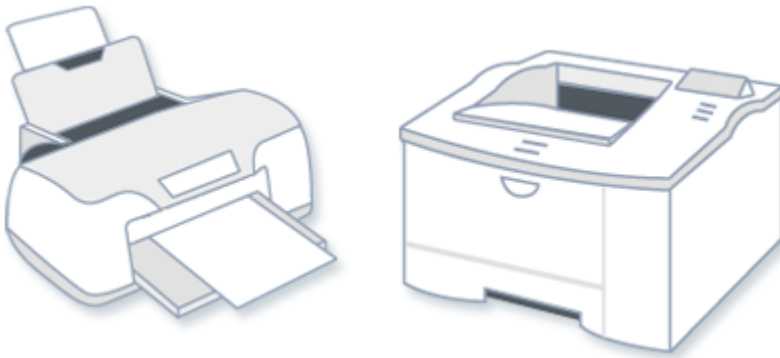


LCD monitor (left); CRT monitor

(right)

Printer

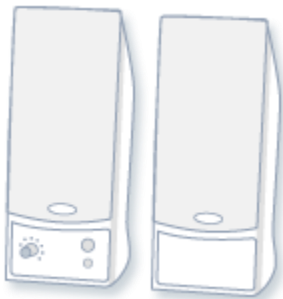
A printer transfers data from a computer onto paper. You don't need a printer to use your computer, but having one allows you to print e-mail, cards, invitations, announcements, and other materials. Many people also like being able to print their own photos at home. The two main types of printers are **inkjet printers** and **laser printers**. Inkjet printers are the most popular printers for the home. They can print in black and white or in full color and can produce high-quality photographs when used with special paper. Laser printers are faster and generally better able to handle heavy use.



Inkjet printer (left); laser printer (right)

Speakers

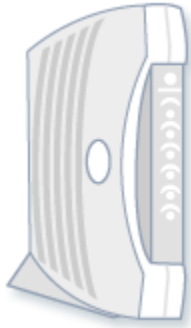
Speakers are used to play sound. They may be built into the system unit or connected with cables. Speakers allow you to listen to music and hear sound effects from your computer.



Computer speakers

Modem

To connect your computer to the Internet, you need a **modem**. A modem is a device that sends and receives computer information over a telephone line or high-speed cable. Modems are sometimes built into the system unit, but higher-speed modems are usually separate components.



Cable modem

SESSION 3.

Describe the design constraints in the design of instruction sets for computers.

Learning Outcomes

- 1. The description identifies the constraints, and outlines the issues involved.
- 2. The description outlines how the constraints have been accommodated, by using examples.

Describe the design constraints in the design of instruction sets for computers.

An **instruction set**, or instruction set architecture (ISA), is the part of the computer architecture related to programming, including the native data types, instructions, registers, addressing modes, memory architecture, interrupt and exception handling, and external I/O. An ISA includes a specification of the set of opcodes (machine language), and the native commands implemented by a particular processor

Classification of instruction sets

A complex instruction set computer (CISC) has many specialized instructions, which may only be rarely used in practical programs. A reduced instruction set computer (RISC) simplifies the processor by only implementing instructions that are frequently used in programs; unusual operations are implemented as subroutines, where the extra processor execution time is offset by their rare use. Theoretically, important types are the minimal instruction set computer and the one instruction set computer, but these are not implemented in commercial processors. Another variation is the very long instruction word (VLIW) where the processor receives many instructions encoded and retrieved in one instruction word.

Machine language

- Machine language is built up from discrete statements or instructions. On the processing architecture, a given instruction may specify:
- Particular registers for arithmetic, addressing, or control functions
- Particular memory locations or offsets
- Particular addressing modes used to interpret the operands
- More complex operations are built up by combining these simple instructions, which (in a von Neumann architecture) are executed sequentially, or as otherwise directed by control flow instructions

Instruction types

Examples of operations common to many instruction sets include:

Data handling and Memory operations

- **set** a register to a fixed constant value
- **move** data from a memory location to a register, or vice versa. Used to store the contents of a register, result of a computation, or to retrieve stored data to perform a computation on it later.
- **read** and **write** data from hardware devices

Arithmetic and Logic operations

- **add**, **subtract**, **multiply**, or **divide** the values of two registers, placing the result in a register, possibly setting one or more condition codes in a status register
- perform bitwise operations, e.g., taking the **conjunction** and **disjunction** of corresponding bits in a pair of registers, taking the **negation** of each bit in a register
- **compare** two values in registers (for example, to see if one is less, or if they are equal)

Control flow operations

- **branch** to another location in the program and execute instructions there
- **conditionally branch** to another location if a certain condition holds
- **indirectly branch** to another location, while saving the location of the next instruction as a point to return to

Complex instructions

CISC processors include "complex" instructions in their instruction set. A single "complex" instruction does something that may take many instructions on other computers. Such instructions are typified by instructions that take multiple steps, control multiple functional units, or otherwise appear on a larger scale than the bulk of simple instructions implemented by the given processor. Some examples of "complex" instructions include:

- saving many registers on the stack at once
- moving large blocks of memory
- complex and/or floating-point
- performing an atomic test-and-set instruction
- instructions that combine ALU with an operand from memory rather than a register

A complex instruction type that has become particularly popular recently is the SIMD or Single-Instruction Stream Multiple-Data Stream operation or vector instruction, that is an operation that performs the same arithmetic operation on multiple pieces of data at the same time. SIMD have the ability of manipulating large vectors and matrices in minimal time. SIMD instructions allow easy parallelization of algorithms commonly involved in sound, image, and video processing. Various SIMD implementations have been brought to market under trade names such as MMX, 3DNow! and AltiVec.

Specialised processor types like GPUs for example also provide complex instruction sets. Nonetheless many of these specialised processor complex instruction sets do not have a publicly available native instruction set and native assembly language for proprietary hardware related reasons and are usually only accessible to software developers through standardized higher level languages and APIs. The OpenGL virtual instruction set and virtual assembly language ARB assembly language and CUDA are examples of such hardware abstraction layers on top of the specialised processor native instruction set.

Parts of an instruction

MIPS32 Add Immediate Instruction

001000	00001	00010	0000000101011110
OP Code	Addr 1	Addr 2	Immediate value

Equivalent mnemonic: **addi \$r1,\$r2,350**



One instruction may have several fields, which identify the logical operation to be done, and may also include source and destination addresses and constant values. This is the MIPS "Add Immediate" instruction which allows selection of source and destination registers and inclusion of a small constant. On traditional architectures, an instruction includes an opcode specifying the operation to be performed, such as "add contents of memory to register", and zero or more operand specifiers, which may specify registers, memory locations, or literal data. The operand specifiers may have addressing modes determining their meaning or may be in fixed fields. In very long instruction word (VLIW) architectures, which include many microcode architectures, multiple simultaneous opcodes and operands are specified in a single instruction.

Some exotic instruction sets do not have an opcode field (such as Transport Triggered Architectures (TTA) or the Forth virtual machine), only operand(s). Other unusual "0-operand" instruction sets lack any operand specifier fields, such as some stack machines including.

Instruction length

The size or length of an instruction varies widely, from as little as four bits in some microcontrollers to many hundreds of bits in some VLIW systems. Processors used in personal computers, mainframes, and supercomputers have instruction sizes between 8 and 64 bits. The longest possible instruction on x86 is 15 bytes (120 bits). Within an instruction set, different instructions may have different lengths. In some architectures, notably most reduced instruction set computers (RISC), instructions are a fixed length, typically corresponding with that architecture's word size. In other architectures, instructions have variable length, typically integral multiples of a byte or a half word.

Representation

The instructions constituting a program are rarely specified using their internal, numeric form (machine code); they may be specified by programmers using an assembly language or, more commonly, may be generated from programming languages by compilers.

Design

The design of instruction sets is a complex issue. There were two stages in history for the microprocessor. The first was the CISC (Complex Instruction Set Computer) which had many different instructions. In the 1970s, however, places like IBM did research and found that many instructions in the set could be eliminated. The result was the RISC (Reduced Instruction Set Computer), an architecture which uses a smaller set of instructions. A simpler instruction set may offer the potential for higher speeds, reduced processor size, and reduced power consumption. However, a more complex set may optimize common operations, improve memory/cache efficiency, or simplify programming

CISC (Complex Instruction Set Computer)

What is CISC?

CISC, which stands for Complex Instruction Set Computer, is a philosophy for designing chips that are easy to program and which make efficient use of memory. Each instruction in a CISC instruction set might perform a series of operations inside the processor. This reduces the number of instructions required to implement a given program, and allows the programmer to learn a small but flexible set of instructions.

Hardware architectures

Most CISC hardware architectures have several characteristics in common:

- Complex instruction-decoding logic, driven by the need for a single instruction to support multiple addressing modes.
- A small number of general purpose registers. This is the direct result of having instructions which can operate directly on memory and the limited amount of chip space not dedicated to instruction decoding, execution, and microcode storage.
- Several special purpose registers. Many CISC designs set aside special registers for the stack pointer, interrupt handling, and so on. This can simplify the hardware design somewhat, at the expense of making the instruction set more complex.
- A "Condition code" register which is set as a side-effect of most instructions. This register reflects whether the result of the last operation is less than, equal to, or greater than zero, and records if certain error conditions occur.

The ideal CISC machine

CISC processors were designed to execute each instruction completely before beginning the next instruction. Even so, most processors break the execution of an instruction into several definite stages; as soon as one stage is finished, the processor passes the result to the next stage:

- An instruction is fetched from main memory.
- The instruction is decoded: the controlling code from the microprogram identifies the type of operation to be performed, where to find the data on which to perform the operation, and where to put the result. If necessary, the processor reads in additional information from memory.
- The instruction is executed. the controlling code from the microprogram determines the circuitry/hardware that will perform the operation.
- The results are written to memory.

In an ideal CISC machine, each complete instruction would require only one clock cycle (which means that each stage would complete in a fraction of a cycle.) In fact, this is the maximum possible speed for a machine that executes 1 instruction at a time.

A realistic CISC machine

In reality, some instructions may require more than one clock per stage, as the animation shows. However, a CISC design can tolerate this slowdown since the idea behind CISC is to keep the total number of cycles small by having complicated things happen within each cycle.

CISC and the Classic Performance Equation

The usual equation for determining performance is the sum for all instructions of (the number of cycles per instruction * instruction cycle time) = execution time. This allows you to speed up a processor in 3 different ways --- use fewer instructions for a given task, reduce the number of cycles for some instructions, or speed up the clock (decrease the cycle time.) CISC tries to reduce the number of instructions for a program, and (as we will see) RISC tries to reduce the cycles per instruction.

CISC Pros and Cons

The advantages of CISC

At the time of their initial development, CISC machines used available technologies to optimize computer performance.

- Microprogramming is as easy as assembly language to implement, and much less expensive than hardwiring a control unit.
- The ease of micro coding new instructions allowed designers to make CISC machines upwardly compatible: a new computer could run the same programs as earlier computers because the new computer would contain a superset of the instructions of the earlier computers.
- As each instruction became more capable, fewer instructions could be used to implement a given task. This made more efficient use of the relatively slow main memory.
- Because micro program instruction sets can be written to match the constructs of high-level languages, the compiler does not have to be as complicated.

The disadvantages of CISC

Still, designers soon realized that the CISC philosophy had its own problems, including:

- Earlier generations of a processor family generally were contained as a subset in every new version --- so instruction set & chip hardware become more complex with each generation of computers.
- So that as many instructions as possible could be stored in memory with the least possible wasted space, individual instructions could be of almost any length---this means that different instructions will take different amounts of clock time to execute, slowing down the overall performance of the machine.
- Many specialized instructions aren't used frequently enough to justify their existence --- approximately 20% of the available instructions are used in a typical program.
- CISC instructions typically set the condition codes as a side effect of the instruction. Not only does setting the condition codes take time, but programmers have to remember to examine the condition code bits before a subsequent instruction changes them.