AS Computing
COMP 2
Revision Notes

COMP 2 - Revision Notes

Hardware	Physical components of the computer
Software	Sequences of instructions or programs which run on the computer

Generations of Programming Language

First Generation – Machine code

00101000 00111100 00111000 00001010 01100000 00101001	Machine code is the actual sequence of zeros and ones that the computer can directly decode and run.
Advantages	It is the only language a computer can actually understand. All other languages must be translated into machine code to run. Machine code needs no translation.
Disadvantages	It's pretty much impossible for human beings to read.

Basic Machine Code instructions (operators)

Machine code instructions are a single number that has two parts, an operator (the first part of the number) that tells the computer what to do, and the operand (the second part) that tells it what to do it with. You need to know about three operators:

Load – Take the operand (a number) and put it in the accumulator
Add– Take the operand and add it to the number currently in the accumulator
Store– put the number currently in the accumulator in the address given in [operand]

Second Generation – Assembly Language

LD #60 ADD #10 ST Total	Uses simple letter abbreviations to make machine codes easier to remember, read and write
Advantages	Simpler for humans to understand. A simple one-to-one translation of machine code instructions allows direct manipulation of the computer.
Disadvantages	Needs to be translated (*assembled*), in order to run. This isn't difficult, but it's an extra step. Because it's a one-to-one translation of machine code, it's very long-winded and machine rather than problem focused.

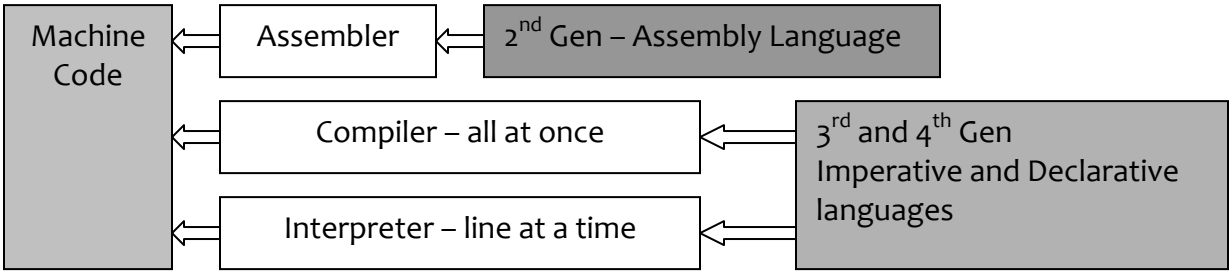
Third Generation – Imperative Programming Languages

`Total := 60 + 10;`	Instructions are executed in a sequence defined by you, the programmer.
Examples	Python, Pascal, C++, Visual Basic
Advantages	Problem focused rather than machine focused, "platform independent" - a program can run on different machines (after translation). Quick and easy to write, understand and debug.
Disadvantages	Requires translation by complex software (compiler/ interpreter). Translation is often inefficient resulting in larger and slower code than assembly language.

Fourth Generation – Declarative Languages

`SELECT column_name(s)` `FROM table_name`	Instructions define what is to be done, but not the detail of how to do it.
Examples	Prolog, SQL
Advantages	Can simplify complex tasks by setting out facts and rules, then letting algorithms work out a solution.
Disadvantages	Not suitable for all types of problem. Requires complex analysis and translation software.

Types of Translation Software

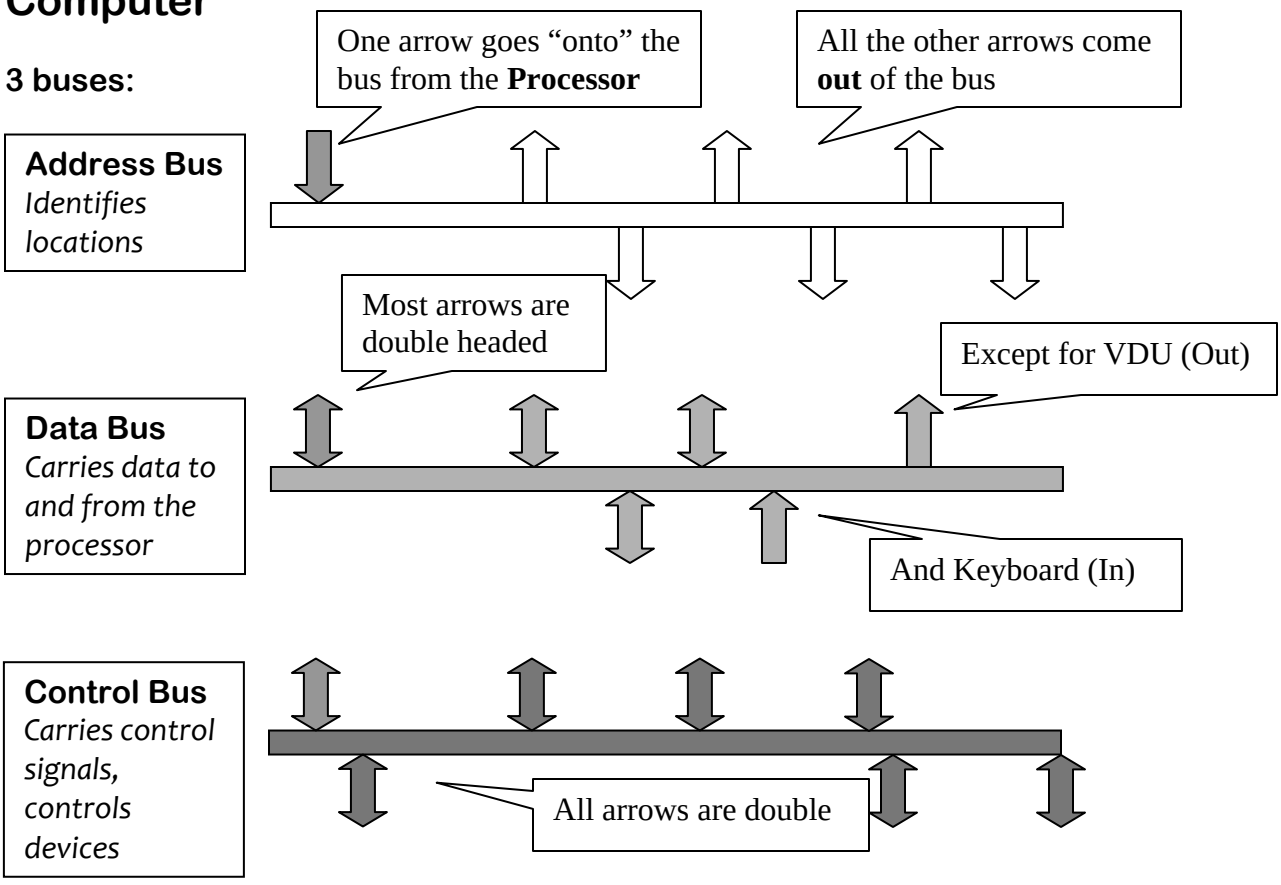


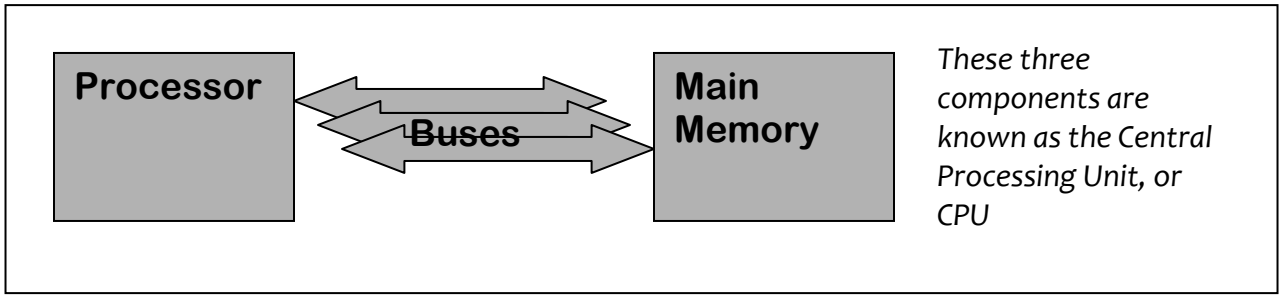
Assembler	**Process**	Translates second generation Assembly language into machine code in a one-to-one operation
	Use	Running an Assembly Language program
	Reason	Require to run Assembly Language. There are no other options for Assembly Language
Compiler	**Process**	Translates third/fourth generation languages into Machine Code in a one-to-many relationship. Checks the syntax of the program, then translates all of the code in one go and produces an output file which can then be run directly without the compiler being present.

	Use	Where tested software is to be shipped out to a user, where the program needs to be as fast as possible or needs to be run without additional program support. Cannot be easily changed.
	Reason	Speed of execution, simplicity of product, can't be changed
Interpreter	**Process**	Translates third/fourth generation languages into Machine Code one line at a time. Checks the syntax of each line and then translates into machine code and executes the program on the computer.
	Use	Software under development, software run in controlled environment.
	Reason	Software debugging, line by line analysis.

Internal and External Hardware Components of a Computer

3 buses:





Anything **not** part of the CPU is called a 'Peripheral'. Peripherals are usually considered external devices. An example of a peripheral is Secondary storage, the Keyboard, the Monitor.

Secondary storage is storage that is not Main Memory. An example would be the Hard Disk.

Main memory is 'addressable', in other words, each location in memory has a unique code – its address. To read or write to memory, the processor *writes* the address to the **address bus**, then uses the **control bus** to indicate a read or write operation. The data is transferred to or from the selected address over the **data bus**. All three buses work together – together they are called the **'System Bus'**.

The Processor and the Fetch-Execute Cycle

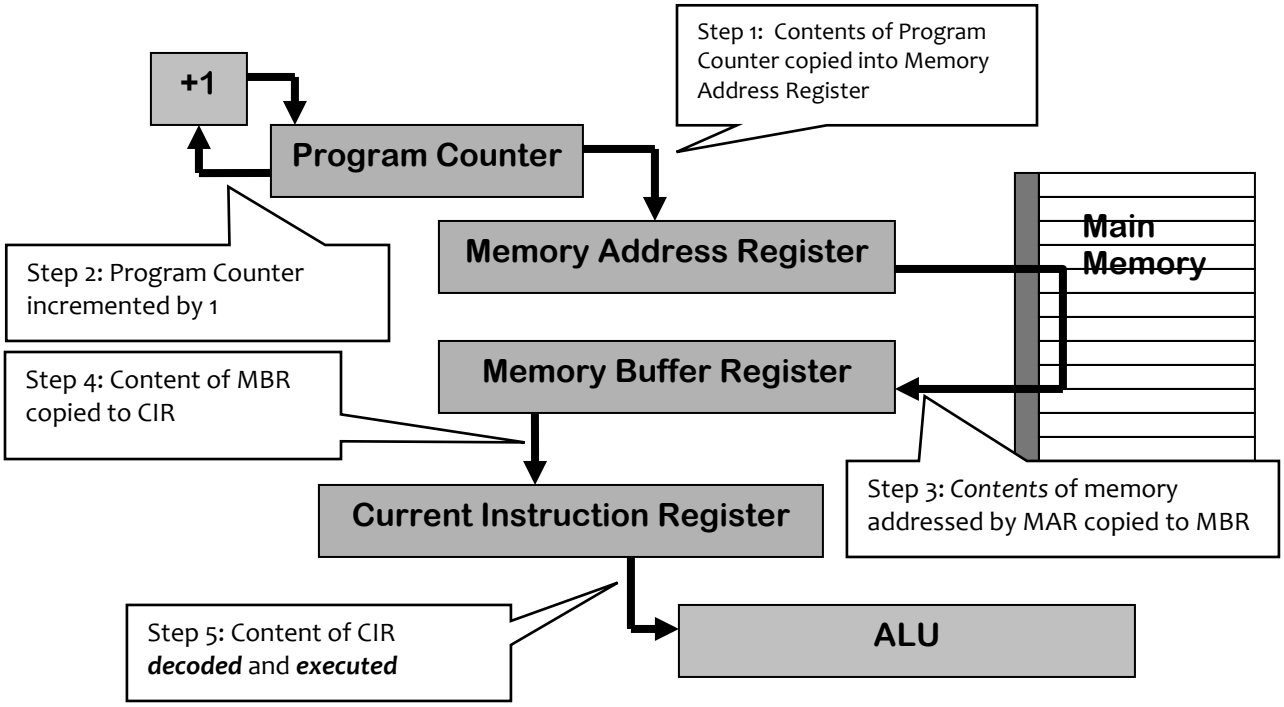
Component parts of the Processor	
Program control unit	Fetches program instructions from memory, decodes them and then executes them one at a time.
Arithmetic and Logic Unit	Performs arithmetic and logical operations on data (for example, adding, ANDing)
Clock	Each pulse is the signal for another cycle of operations. The clock co-ordinates all devices in a computer, making sure they all act together.
General purpose and dedicated registers	Fast, temporary storage locations inside the processor.

Making the computer Faster

Increase the clock speed	Should directly increase the performance of a computer by speeding up the cycle, the time it takes the processor to complete an instruction.
Increase the bus width	The system bus is a significant bottleneck. Large numbers or complex instructions must be split into smaller parts to fit across the bus. Widening the bus means less splitting and better performance.
Increase the word length	Word length affects the size of the registers and reduces the need to split large numbers or complex operations

	across multiple registers, thus improving performance.

The Fetch-Execute Cycle



Note: It's essential to remember the name and Mnemonic of the four special purpose registers here. PC, MAR, MBR, CIR.

Machine code instructions are often represented in Hexadecimal

The stored program concept

Machine code instructions are stored in **main memory,** from which they are **fetched** and **executed.** These instructions can be replaced by others at any time.

Logic Gates

AND		A B Q 0 0 **0** 0 1 **0** 1 0 **0** 1 1 **1**	**A.B**	Output is on **only** when both inputs are on, otherwise its off.

OR		A B Q 0 0 **0** 0 1 **1** 1 0 **1** 1 1 **1**	$A+B$	Output is on if either of the inputs is on, or both.
NOT		A Q 0 **1** 1 **0**	$\overline{A}$	Output is on if the input is off, and off if the input is on.
XOR		A B Q 0 0 **0** 0 1 **1** 1 0 **1** 1 1 **0**	$A \oplus B$	Output is on if one input is on and the other is off, otherwise the output is off.
NAND		A B Q 0 0 **1** 0 1 **1** 1 0 **1** 1 1 **0**	$\overline{A.B}$	The reverse of an AND gate, output is on unless both inputs are on, in which case it is off.
NOR		A B Q 0 0 **1** 0 1 **0** 1 0 **0** 1 1 **0**	$\overline{A+B}$	The reverse of an OR gate, output is on only if both inputs are off. If any input is on, the output is off.
XNOR		A B Q 0 0 **1** 0 1 **0** 1 0 **0** 1 1 **1**	$\overline{A \oplus B}$	The reverse of an XOR, output is on only if both inputs are the same. If they are different, output is off.

Boolean Algebra and Simplifications

$A.B=B.A$
$A+B=B+A$
$(A+B)+C=A+(B+C)$
$A.(B+C)=A.B+A.C$

De Morgan's Law
$\overline{(X+Y)} = \overline{X}.\overline{Y}$
$\overline{X.Y} = \overline{X}+\overline{Y}$

$X.X=X$
$X.\overline{X} = 0$

$1+A = 1$
$0+A=0$
$0.A=A$
$1.A=A$

Steps for De Morgan's Law

1. **Swap the expression – from and to or - or or to and**

2. **Invert each side of the expression**
3. **Invert the entire expression**
4. **Remove any duplicate inversions**

Think of it this way …..