Low Level Languages

Explain the concepts and, using examples, demonstrate an understanding of the use of the accumulator, registers, and program counter;
Describe immediate, direct, indirect, relative and indexed addressing of memory when referring to low-level languages;
Discuss the concepts and, using examples, show an understanding of mnemonics, opcode, operand and symbolic addressing in assembly language to include simple arithmetic operations, data transfer and flow-control.

Notes

- Opcodes is the part of the binary string that represent the operations that the computer can understand and carry out. They are easier to remember.
- They can be represented by mnemonics which are the pseudo names given to the different operations that make it easier. E.g. ADD.
- The operand is the data to be manipulated, there's no point telling the computer what to ADD if there's no data to apply it to. It can hold the address of the data, or just the data.
- Address Labels are used as a symbolic representation of the operands
- Symbolic addressing is the use of characters to represent the address of a store location

There are three different operations that can be done in assemble language, which have different effects on the processor. These are: arithmetic or logic operations, data transfer and flow control.

Arithmetic Operations

- When opcode is decoded, data is collected and placed in the memory data register
- It is then manipulated in the ALU. Part of this is the accumulator, where results are temporarily stored until they are needed for the next operation.

Data Transfer

- Some operations (like GET and GTO) just move data in and out of the memory.

Flow Control

- This is where the order which instructions are executed may be changed by a jump instruction or a conditional jump instruction.

The special Registers

The program counter (PC) is used to keep track of the location of the next instruction to be executed. (This register is also known as the Sequence Control Register (SCR)). It is used so that the processor always knows where the next instruction is. Note that the content can be changed by some instructions as well as simple incrimination.

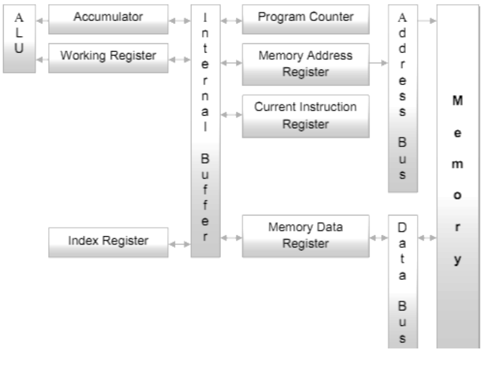
The memory address register (MAR) holds the address of the instruction or data that is to be fetched from memory. This address can come directly from the PC if the data to be fetched is the next instruction, or it can come from the CIR when the address of the data to be used is found in the decoding process.

The current instruction register (CIR) holds the instruction that is to be executed, ready for decoding. This instruction will consist of (at least) an operation code which will be looked up in the table of codes so that the processor knows what actions are necessary and the address which will be sent to the MAR.

The memory data register (MDR) holds data to be transferred to memory and data that is being transferred from memory, including instructions on their way to the CIR. The computer cannot distinguish between data and instructions. Both are held as binary numbers. How these binary numbers are interpreted depends on the registers in which they end up. The MDR is the only route between the other registers and the main memory of the computer.

The accumulator is where results are temporarily held and is used in conjunction with a working register (ALU) to do calculations.

The index register is a special register used to adjust the address part of an instruction. If this is to be used then the simple case of two parts to the instruction which was detailed above, must become three parts because there must be a part of the instruction which defines which sort of addressing must be used.



Addressing

Each instruction in machine code is represented by a series of binary digits. There are two parts to each instruction, an operation (the opcode) and the data. The data is what the operation is being applied to, there are a number of different ways in which this data can be represented, and this is known as addressing. Usually the address of the data is given rather than just the data.

Direct Addressing

This is where the address part of the instruction holds the address of where the data is. For example if an instruction was 00110111 and the first 3 bits were the instruction e.g. ADD and the last 5 bits were the address, then the computer would know to add whatever is in the data location 10111 to the accumulator. Most bytes are bigger than 8 bits, more like 32 or 64 bits. However it does not give access to enough memory, and needs supplementing with other methods.

Immediate Addressing

This is when the value in the instruction is not an address at all but the actual data. This is very simple, although not often used because the program parameters cannot be changed. This means that the data being operated on can’t be adjusted and only uses constants.

Indirect Addressing

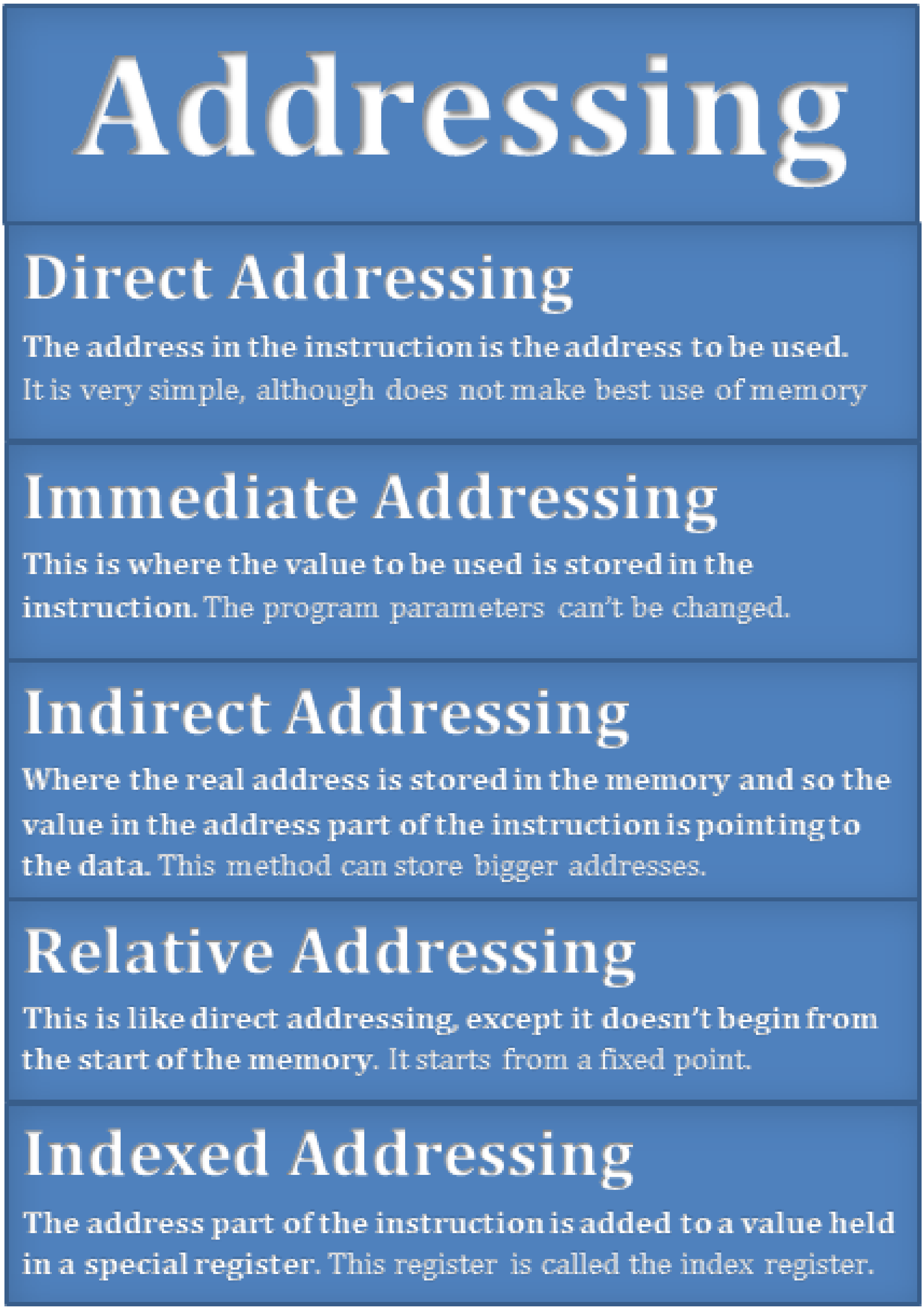
This is where the real address is stored in the memory so the value in the address part of the instruction is pointing to the address of the data. This method is useful because the amount of space in a location is much bigger than the space in the address part of the instruction. Therefore we can store larger addresses and use more memory.

Relative Addressing

This is direct addressing that does not commence from the start of the address of the memory. It begins from a fixed point, and all addresses are relative to that point.

Indexed Addressing

This is where the address part of the instruction is added to the value held in a special register. The special register use is the Index Register (IR). The result of this is then the required address.



Addressing

Direct Addressing

The address in the instruction is the address to be used. It is very simple, although does not make best use of memory

Immediate Addressing

This is where the value to be used is stored in the instruction. The program parameters can't be changed.

Indirect Addressing

Where the real address is stored in the memory and so the value in the address part of the instruction is pointing to the data. This method can store bigger addresses.

Relative Addressing

This is like direct addressing, except it doesn't begin from the start of the memory. It starts from a fixed point.

Indexed Addressing

The address part of the instruction is added to a value held in a special register. This register is called the index register.

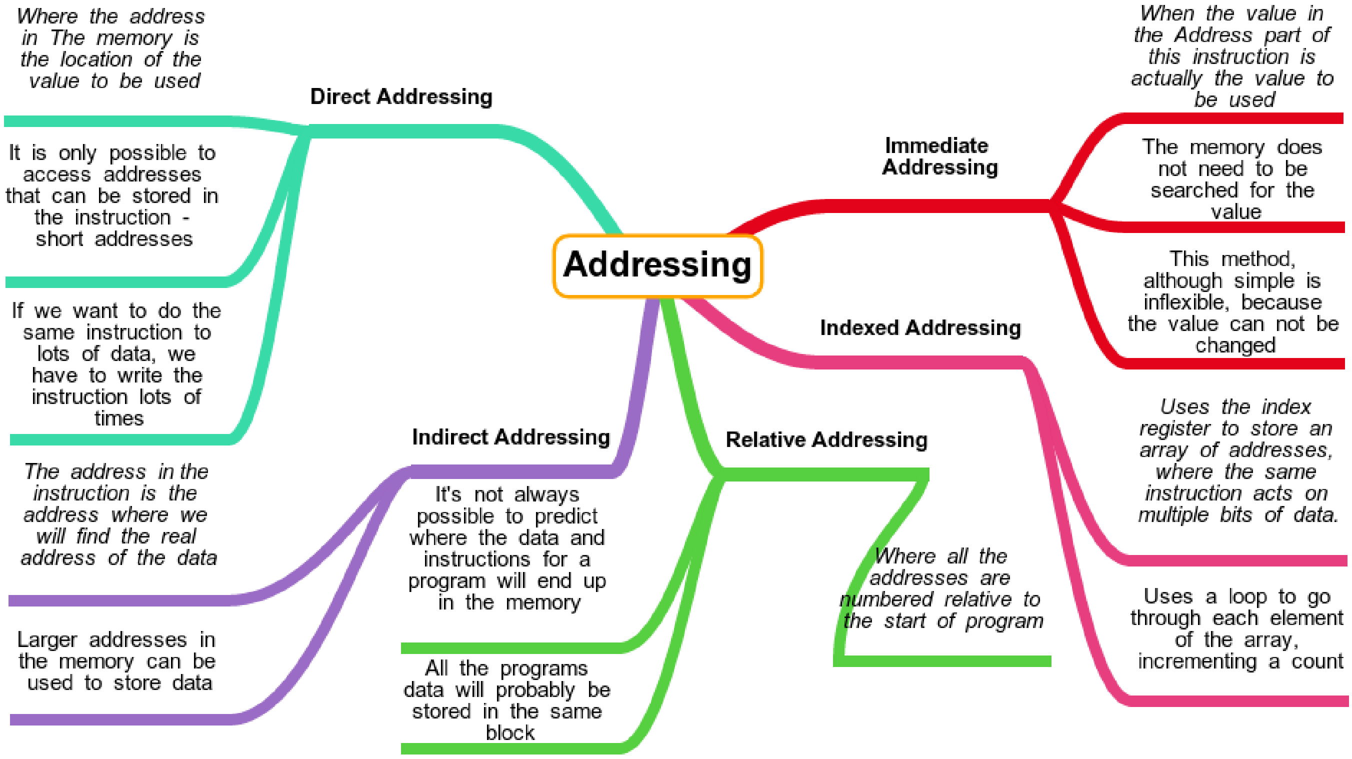
Features of Low Level Languages

Computers work with machine code instructions which are written as a series of binary digits, each instruction is in two parts, as operation and an address.

Each computer has a different operation set, which means different computers understand different operations – the machine code is unique to that machine.

For example, if instructions were 8 bits, and there were 3 bits used to stand for the operation code (8 possible combinations) and 5 bits used to represent the identifier of the address, where the data is stored. (Note: this is very simplified, and the binary instructions are usually considerably longer that 8 bits). If 001 means ADD then 00101001 would mean add whatever is in location 01001. This value will then be added to the accumulator.

Next, read about addressing, there are five different types in the specification.



Addressing

Direct Addressing

Where the address in The memory is the location of the value to be used

It is only possible to access addresses that can be stored in the instruction - short addresses

If we want to do the same instruction to lots of data, we have to write the instruction lots of times

Immediate Addressing

When the value in the Address part of this instruction is actually the value to be used

The memory does not need to be searched for the value

This method, although simple is inflexible, because the value can not be changed

Indexed Addressing

Uses the index register to store an array of addresses, where the same instruction acts on multiple bits of data.

Uses a loop to go through each element of the array, incrementing a count

Indirect Addressing

The address in the instruction is the address where we will find the real address of the data

Larger addresses in the memory can be used to store data

Relative Addressing

Where all the addresses are numbered relative to the start of program

It's not always possible to predict where the data and instructions for a program will end up in the memory

All the programs data will probably be stored in the same block

Past Exam Questions and Answers

Explain the term opcode

the mnemonic part of the instruction/that indicates what it is to do/code for the operation

What is symbolic addressing?

the use of characters to represent the address of a store location

What is the purpose of the accumulator?

temporary storage (within ALU)
holds data being processed/used during calculations
deals with the input and output in the processor

What is indexed addressing?

uses an index register/IR
...and an absolute address...
...to calculate addresses to be used

What is direct addressing?

the instruction gives the address to be used

What is relative addressing?

allows a real address to be calculated...
...from a base address...
...by adding the relative address
relative address is an offset
can be used for arrays

can be used for branching

What’s immediate addressing?

used in assembly language
uses data in address field…
… as a constant

Why is it not possible to use only direct addressing in assembly languages?

number of addresses available is limited…
...by the size of the address field
code is not relocatable/code uses fixed memory locations

Explain Mnemonics

a code that is easily remembered…
...used to give the opcode/instruction
e.g. ADD

Explain flow controll

the order in which instructions are executed
the order may be changed by a jump
instruction/conditional jump instruction

How and why is the the index register (IR) used?

used in indexed addressing
stores a number used to modify an address…
… which is given in an instruction
allows efficient access to a range of memory locations/by incrementing the value in the IR
eg used to access an array

What are the differences between machine code and assembly language?

Machine Code:
written in binary or hex
no translation needed
very difficult to write
Assembly language:
includes mnemonics
includes names for data stores
translated by an assembler
easier to write than machine code, but more difficult than high level language

What's the use of an operand, in an assembly language instruction?

address field (in an instruction)
it holds data…
to be used by the operation given in the opcode
eg in ADD 12, "12" is the operand

What's the difference between direct and indirect addressing?

direct:
the simplest/most common method of addressing
uses the (data in) the address field…without modification
eg In ADD 23, use the number stored in address 23 for
the instruction (accept any valid example)
limits the memory locations that can be addressed *
indirect:
uses the address field as a vector/pointer… to the address to be used
used to access library routines
eg In ADD 23, if address 23 stores 45, address 45 holds the number to be used increases the memory locations that can be addressed