Cambridge International Examinations
Cambridge International Advanced Subsidiary and Advanced Level

CPU ARCHITECTURE

QUESTION 1

Explain how the width of the data bus and system clock speed affect the performance of a computer system.

ANSWER 1

Data Bus Width

- the width of the data bus determines the number of bits that can be simultaneously transferred
- increasing the width of the data bus increases the number of bits/amount of data that can be moved at one time (or equivalent)
- …hence improving processing speed as fewer transfers are needed
- By example: e.g. double the width of the data bus moves 2x data per clock pulse

Clock Speed

- determines the number of cycles the CPU can execute per second
- increasing clock speed increases the number of operations/number of fetch-execute cycles that can be carried out per unit of time
- ...however, there is a limit on clock speed because the heat generated by higher clock speeds cannot be removed fast enough

QUESTION 2

Most computers use **Universal Serial Bus (USB) ports** to allow the attachment of devices. Describe two benefits of using USB ports

ANSWER 2

- devices automatically detected and configured when first attached/plug and play
- it is nearly impossible to wrongly connect a device
- USB has become an industrial standard
- supported by many operating systems
- USB 3.0 allows full duplex data transfer
- later versions are backwards compatible with earlier USB systems
- allows power to be drawn to charge portable devices

QUESTION 3

Name and describe three **buses** used in the **von Neumann** model.

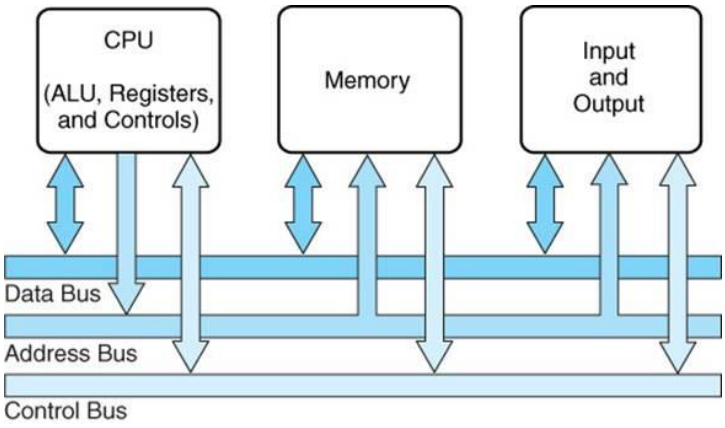
ANSWER 3

Address Bus

- lines used to transfer address of memory or input/output location
- unidirectional bus

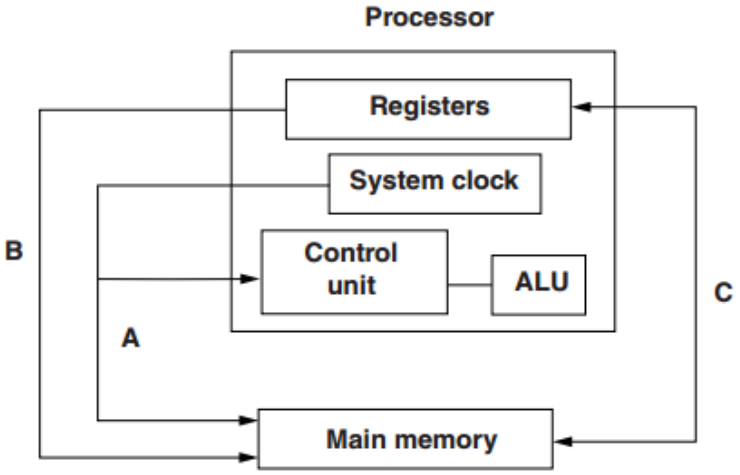
Data Bus
- used to transfer data between the processor and memory/input and output devices
- bidirectional bus

Control Bus
- used to transmit control signals
- e.g. read/write/fetch/ …
- dedicated bus since all timing signals are generated according to control signal



QUESTION 4

(a)
The diagram above shows a simplifie
form of processor architecture.
Name the three buses
labelled A, B and C.



(b) State the role of each of the following special purpose registers used in a typical processor.
- Program Counter
- Memory Data Register
- Current Instruction Register
- Memory Address Register

ANSWER 4

(a) A = **control** bus
B = **address** bus
C = **data** bus

(b) **Program Counter** – stores the address of next instruction to be executed

Memory Data Register – stores the data in transit between memory and other registers // holds the instruction before it is passed to the CIR

Current Instruction Register – stores the current instruction being executed

Memory Address Register – stores the address of the memory location which is about to be accessed

QUESTION 5

(a) Describe the **stored program** concept for the basic **Von Neumann** model for a computer system.

(b) (i) Name the three types of **bus** used by a processor.
(ii) State the function of the **system clock** in a processor

ANSWER 5

(a) – program must be resident in (main) memory to be executed
– program consists of a sequence of instructions
– which occupy a (contiguous) block of main memory
– instructions and data are indistinguishable
– each instruction is fetched, (decoded) and then executed
– instruction fetch and data operation cannot occur at the same time

(b) (i) – control bus
– data bus
– address bus

(ii) generates the timing signals/generates the signals to synchronise events in the processor / fetch–(decode)–execute cycle

QUESTION 6

(a) Briefly describe the **Von Neumann** architecture.

(b) Define all **registers** used by Von Neumann architecture.

ANSWER 6

(a) - The Von Neumann architecture uses a single processor which follows a linear sequence of fetch-decode-execute.

- In order to do this, the processor has to use some special registers, which are discrete memory locations with special purposes attached.

(b) - **Program counter (PC)** : Keeps track of where to find the next instruction so that a copy of the instruction can be placed in the current instruction register.

- **Memory Address Register (MAR)**: Used to hold the memory address that contains either the next piece of data or an instruction that is to be used.

- **Memory Data Register (MDR)**: Acts like a buffer and holds anything that is copied from the memory ready for the processor to use it.

- **Current Instruction Register (CIR)**: Holds the instruction that is to be executed.

- **Index Register (IR)**: A microprocessor register used for modifying operand addresses during the run of a program, typically for doing array operations. Index registers are used for a special kind of indirect addressing where an immediate constant (which is part of the instruction) is added to the contents of the index register to form the address to the actual operand or data.

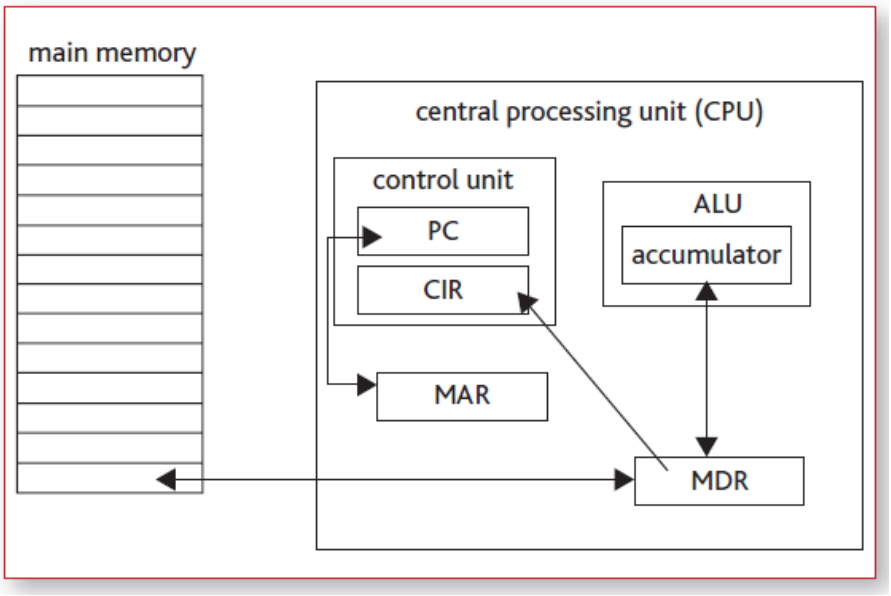
- **Status Register (SR)**: It sits inside ALU and has the following 3 purposes during a program's instructions execution. It holds:
    - Results of comparisons to decide later for action
    - Interim results of arithmetic performed
    - Any errors occurred during the arithmetic operations.

QUESTION 7

What is the purpose of **ALU** and **Control Unit**?

ANSWER 7

- The central processor contains the **ALU** (Arithmetic Logic Unit) and the **Control Unit**.
- The **ALU** is where data is processed.
- This involved arithmetic and logical operations.
- Arithmetic operations are those that add and subtract numbers.
- Logical operations involve comparing binary patterns and making decisions.
- The **Control Unit** fetches instructions from memory, decodes them and synchronizes the operations before sending signals to other parts of the computer.
- The **accumulator** is in the **ALU**.
- The **Program Counter (PC)** and the **Current Instruction Register (CIR)** are in the **Control Unit**.
- The **Memory Data Register (MDR)** and **Memory Address Register (MAR)** are in the **processor**.



QUESTION 8

Describe the following terms"

(a) **Peripherals**
(b) **I/O Controllers**
(c) **I/O Port**

ANSWER 8

(a) Input/Output devices are used by the system to get information in and out, as they are not internal but are connected to the CPU, we refer to them as **peripherals**.

- Input Devices – used to feed information to the system (e.g. keyboard)
- Output Devices – used to display the results of data processing on the input data (e.g. screen)

(b) **I/O Controllers** – an electronic circuit that connects to a system bus and an I/O device. It provides the correct voltages and currents for the system bus and the I/O device. Examples are:

- Keyboard controller, attached to a keyboard
- Disk controller for a Hard Disk
- Video display controller, attaching a video display unit (monitor)

(c) **I/O port** – a method of performing input/output between the CPU and peripheral devices in a computer. This allows I/O devices to be connected to the CPU without having to have specialized hardware for each one.

FETCH-EXECUTE CYCLE

QUESTION 1

The table shows six stages in the **von Neumann fetch-execute cycle**. Put the stages into the correct sequence by writing the numbers 1 to 6 in the right hand column.

ANSWER 1

Description of stage	Sequence number
the instruction is copied from the Memory Data Register (MDR) and placed in the Current Instruction Register (CIR)	3
the instruction is executed	6
the instruction is decoded	5
the address contained in the Program Counter (PC) is copied to the Memory Address Register (MAR)	1
the value in the Program Counter (PC) is incremented so that it points to the next instruction to be fetched	4
the instruction is copied from the memory location contained in the Memory Address Register (MAR) and is placed in the Memory Data Register (MDR)	2

QUESTION 2

The sequence of operations shows, in **register transfer notation**, the fetch stage of the fetch execute cycle.

1. MAR ← [PC]
2. PC ← [PC] + 1
3. MDR ← [[MAR]]
4. CIR ← [MDR]

- [register] denotes contents of the specified register or memory location
- step 1 above is read as “the contents of the Program Counter are copied to the Memory Address Register”

(a) (i) Describe what is happening at step 2.
(ii) Describe what is happening at step 3.
(iii) Describe what is happening at step 4.

(b) Describe what happens to the registers when the following instruction is executed:
LDD 35

(c) (i) Explain what is meant by an **interrupt**.
(ii) Explain the actions of the **processor** when an interrupt is detected.

ANSWER 2

(a) (i) the program counter is incremented

(ii) the data stored at the address held in MAR is copied into the MDR

(iii) the contents of the Memory Data Register is copied into the Current Instruction Register

(b) • the MAR is loaded with the operand of the instruction // loaded with 35
• the Accumulator is loaded with the contents of the address held in MAR // the Accumulator is loaded with the contents of the address 35

(c) (i) • a signal
• from a device/program that it requires attention from the processor

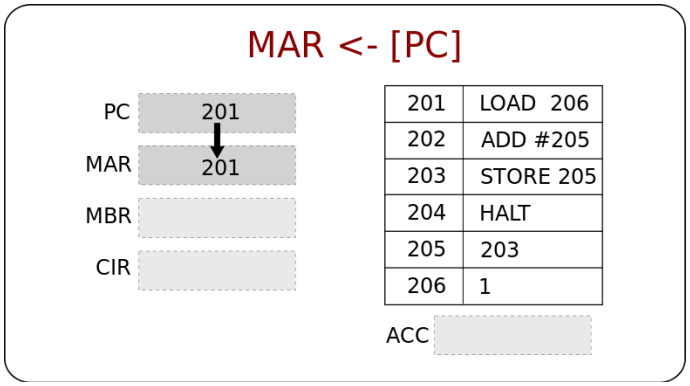
(ii) • at a point during the fetch-execute cycle …
• check for interrupt
• if an interrupt flag is set/ bit set in interrupt register
• all contents of registers are saved
• PC loaded with address of interrupt service routine

QUESTION 3

Describe the Fetch-Decode-Execute cycle.

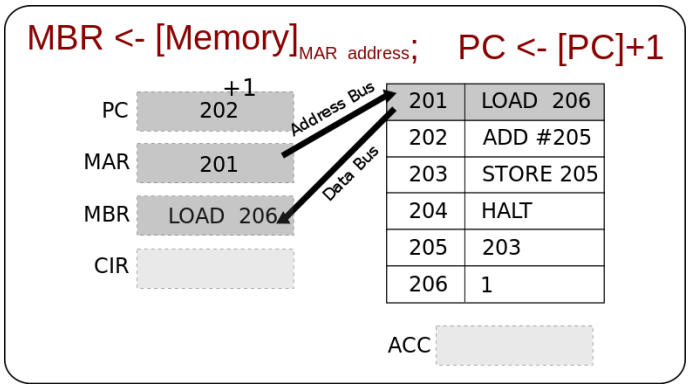
ANSWER 3

1. The contents of the **Program Counter (PC)**, the address of the next instruction to be executed, is place into the **Memory Address Register (MAR)**.

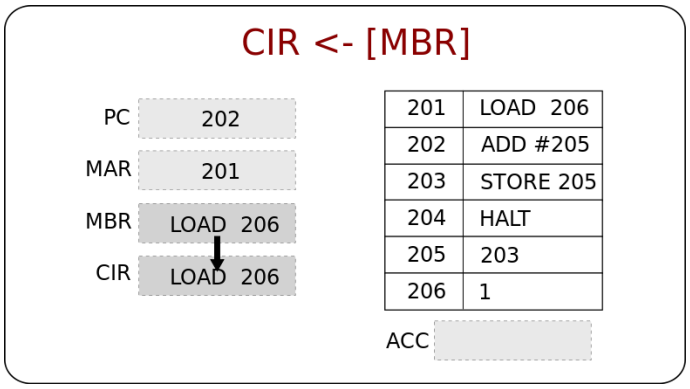


2. The address is sent from the **MAR** along the address but to the **Main Memory**. The instruction at that address is **found and returned** along the **data bus** to the **Memory Buffer Register (MBR)**.

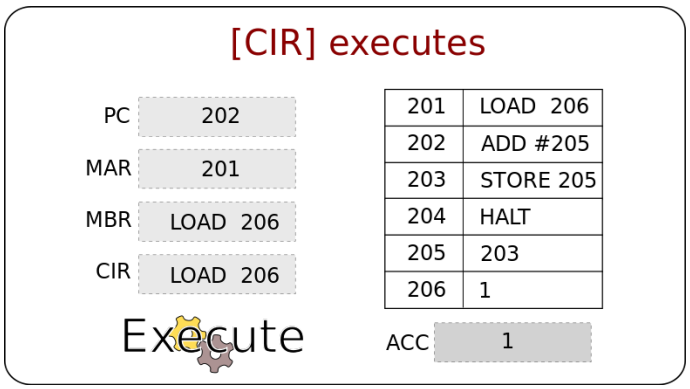
At the same time the contents of the **Program Counter (PC)** is increase by 1, to reference the next instruction to be executed.



3. The **MBR** loads the **Current Instruction Register (CIR)** with the instruction to be executed.



4. The instruction is decoded and executed using the ALU if necessary.



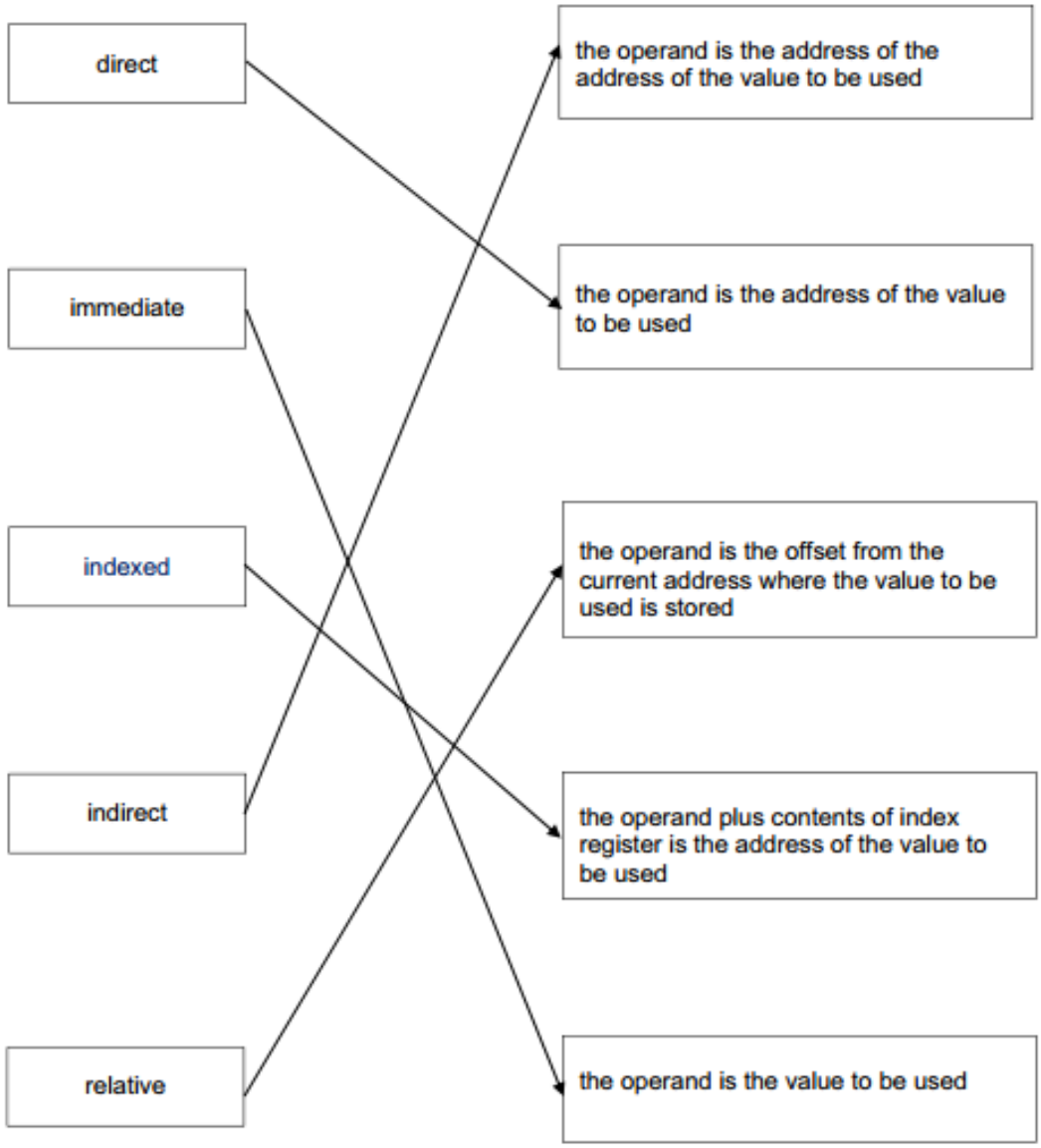
The cycles starts again…

PROCESSOR'S INSTRUCTION SET

QUESTION 1

Five modes of addressing and five descriptions are shown below. Draw a line to connect each mode of addressing to its correct description.

ANSWER 1

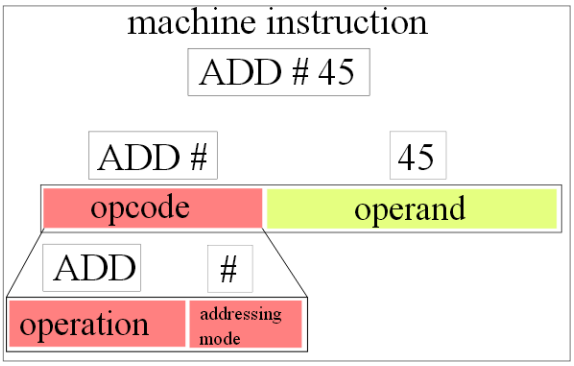


QUESTION 2

Show an understanding between **OpCode** and **Operand**.

ANSWER 2

OpCode Instruction name
Operand Data or Address

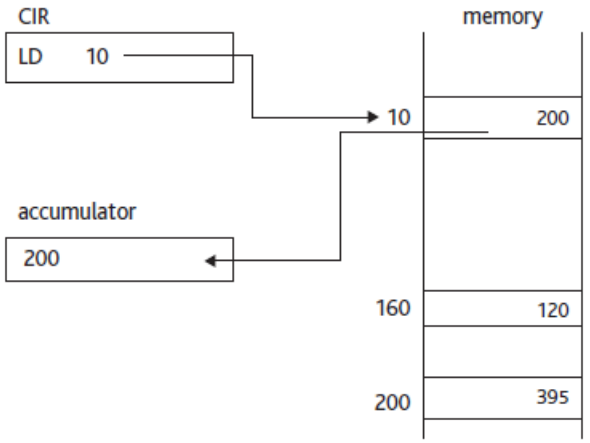


QUESTION 3

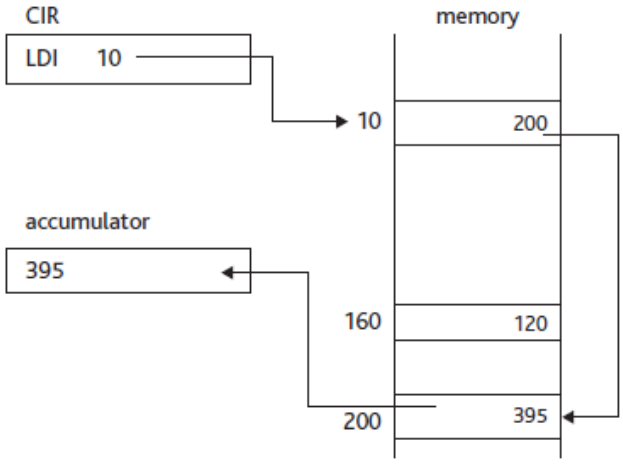
Show an understanding of the following addressing modes:
Direct, Indirect, Indexed, Relative

ANSWER 3

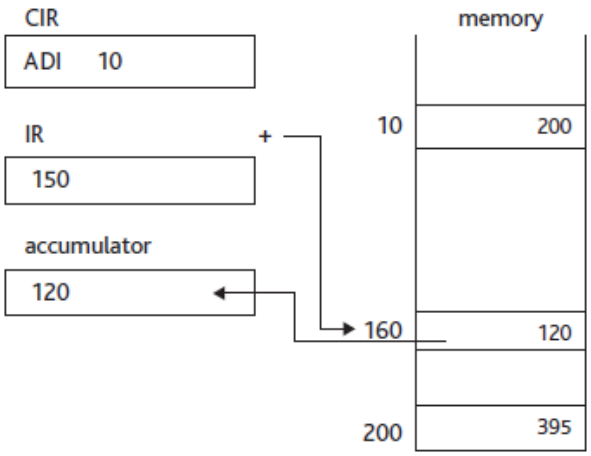
Direct Addressing Mode: means that the operand is the **actual address** to be used.



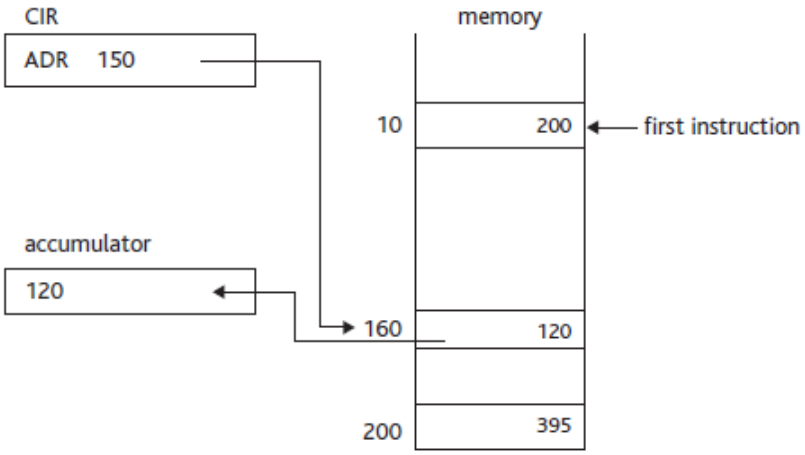
Indirect Addressing Mode: means the operand holds the address of the location that holds the value to be used.



Indexed Addressing Mode: means the operand is added to the IR to calculate the address to be used



Relative Addressing Mode: means the operand is added to the address of the first instruction to calculate the address.



ASSEMBLY LANGUAGE

QUESTION 1

Assemblers translate from assembly language to **machine code**. Some assemblers scan the assembly language program twice; these are referred to as **two-pass assemblers**.

The following table shows five activities performed by two-pass assemblers. Write 1 or 2 to indicate whether the activity is carried out during the first pass or during the second pass

ANSWER 1

Activity	First pass or second pass
any symbolic address is replaced by an absolute address	2
any directives are acted upon	1
any symbolic address is added to the symbolic address table	1
data items are converted into their binary equivalent	1
forward references are resolved	2

QUESTION 2

The table shows assembly language instructions for a processor which has one general purpose register, the Accumulator (ACC).

Instruction		Explanation
Op code	Operand	
LDD	<address>	Direct addressing. Load contents of given address to ACC
STO	<address>	Store the contents of ACC at the given address
LDI	<address>	Indirect addressing. The address to be used is at the given address. Load the contents of this second address to ACC
LDX	<address>	Indexed addressing. Form the address from <address> + the contents of the index register. Copy the contents of this calculated address to ACC
INC	<register>	Add 1 to contents of the register (ACC)
JMP	<address>	Jump to the given address
END		Return control to operating system

The diagram shows the contents of the memory.

Main memory

120	0 0 0 0 1 0 0 1
121	0 1 1 1 0 1 0 1
122	1 0 1 1 0 1 1 0
123	1 1 1 0 0 1 0 0
124	0 1 1 1 1 1 1 1
125	0 0 0 0 0 0 0 1
126	0 1 0 0 0 0 0 1
127	0 1 1 0 1 0 0 1
200	1 0 0 0 1 0 0 0

(a) (i) Show the contents of the Accumulator after execution of the instruction:

LDD 121

Accumulator: | | | | | | | | |

(ii) Show the contents of the Accumulator after execution of the instruction:
Explain how you arrived at your answer.

LDI 124

Accumulator: | | | | | | | | |

(iii) Show the contents of the Accumulator after execution of the instruction:
Explain how you arrived at your answer.

LDX 120

Index Register:	0	0	0	0	0	1	1	0
Accumulator:								

(b) Trace the assembly language program using the trace table.

```
300    LDD    321
301    INC
302    STO    323
303    LDI    307
304    INC
305    STO    322
306    END
307    320
       ↑      ↑
320    49
321    36
322    0
323    0
```

Accumulator	Memory address			
	320	321	322	323
	49	36	0	0

ANSWER 2

(a) (i)

Accumulator:	0	1	1	1	0	1	0	1

(ii)

Accumulator:	0	1	1	0	1	0	0	1

Explanation

- content of 124 is 0 1 1 1 1 1 1 1
- this is equivalent to 127
- contents of 127 are 0 1 1 0 1 0 0 1

(iii)

Accumulator:	0	1	0	0	0	0	0	1

Explanation

- index register value = 6
- 120 + 6 = 126
- contents of 126 placed in the accumulator

(b)

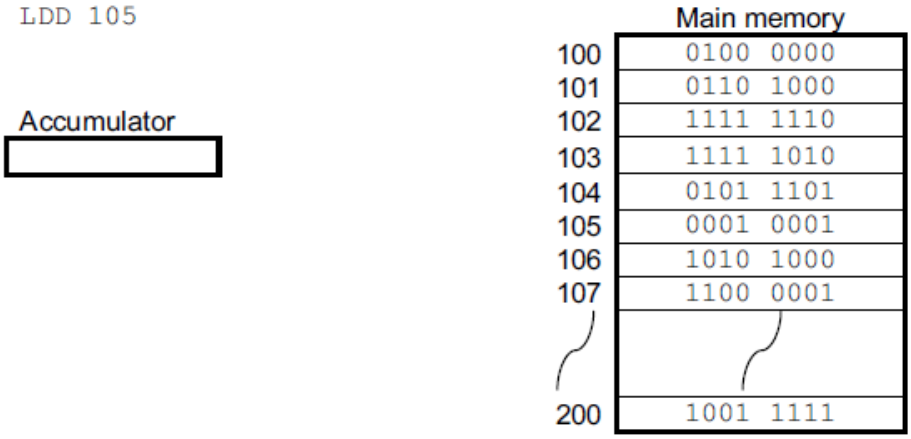
Accumulator	Memory address			
	320	321	322	323
	49	36	0	0
36				
37				
				37
49				
50				
			50	

QUESTION 3

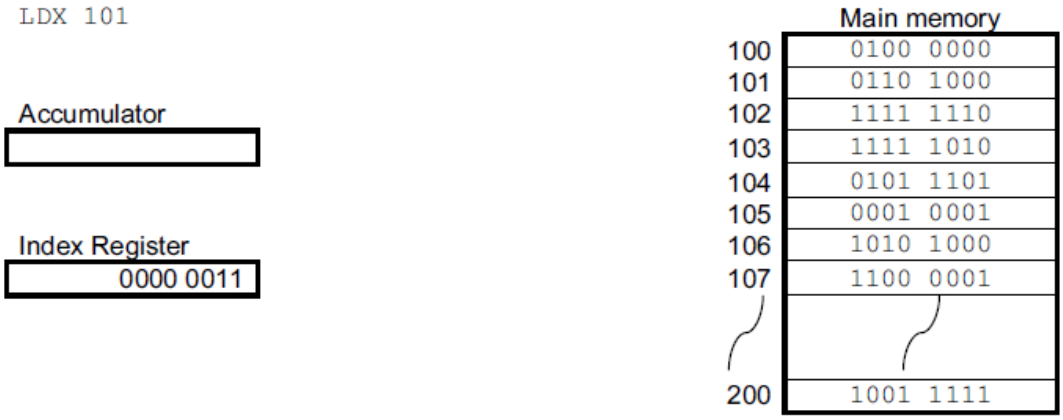
The table shows the assembly language instructions for a processor which has one general purpose register – the Accumulator.

Instruction		Explanation
Op Code	Operand	
LDD	<address>	load using direct addressing
STO	<address>	store the contents of the Accumulator at the given address
LDI	<address>	load using indirect addressing
LDX	<address>	load using indexed addressing
INC		add 1 to the contents of the Accumulator
END		end the program and return to the operating system

(a) Write on the diagram to explain the instruction shown.
Show the contents of the Accumulator after the execution of the instruction.



(b) Write on the diagram to explain the instruction shown.
Show the contents of the registers after the execution of the instruction.



(c) Trace this assembly language program using the trace table.

```
500     LDD   507
501     INC
502     STO   509
503     LDD   508
504     INC
505     STO   510
506     END
507      22
508     170
509       0
510       0
```

Accumulator	Memory Address			
	507	508	509	510
	22	170	0	0

ANSWER 3

(a)

Accumulator
0001 0001

(b)

Accumulator
0101 1101

Index Register
0000 0011

(c)

Accumulator	Memory Address			
	507	508	509	510
	22	170	0	0
22				
23				
			23	
170				
171				
				171

QUESTION 4

Trace the assembly language program using the trace table.

ANSWER 4

Assembly code	Main memory start		Main memory end	
	Address	Contents	Address	Contents
	10	9	10	9
	11	2	11	2
LDM #10 **ADD** #12 STORE 12	12	7	12	**22**
	13	10	13	10
	14	12	14	12

This code loads the **number** 10 into the accumulator, then adds the **number** 12, it then stores the result 22 into memory location 12.

QUESTION 5

Trace the assembly language program using the trace table.

ANSWER 5

Assembly code	Main memory start		Main memory end	
	Address	Contents	Address	Contents
	10	9	10	9
	11	2	11	2
LDD 10 **ADD**12 STORE 12	12	7	12	**16**
	13	10	13	10
	14	12	14	12

This code loads the value stored in memory **location** 10 into the accumulator (9), then adds the value stored in memory **location** 12 (7), it then stores the resut into memory location 12 (9+7=16)

QUESTION 6

What do we mean by the term '**mnemonic**'?

ANSWER 6

The first attempt at making programs more understandable was to use a **mnemonic** in place of the binary opcodes.

Opcode	Operation	Mnemonic
001	Add a number	ADD
010	Subtract a number	SUB
011	Input a value	IN
100	Output a value	OUT
101	Store a value in a memory location	STO
110	Get a value from a memory location	GET

QUESTION 7

What do we mean by the term '**symbolic**' addressing?

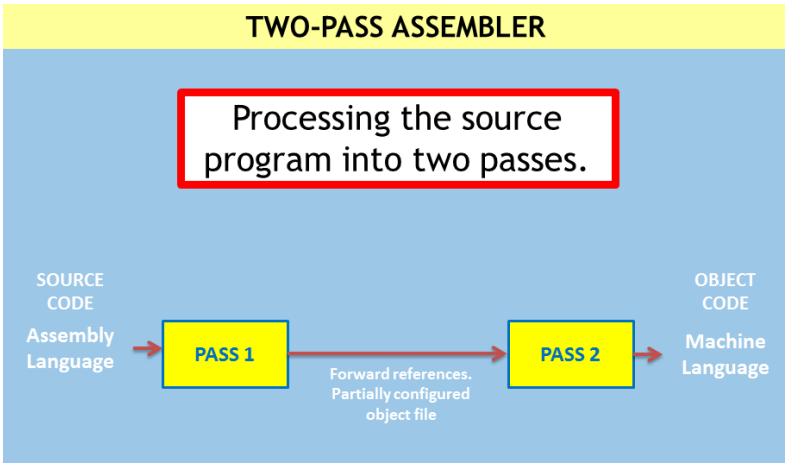
ANSWER 7

Using **mnemonics** to represent **opcodes** and labels to represent the memory locations makes the program code very much easier for the programmer to understand. The use of **labels** for memory addresses is called **symbolic addressing**.

QUESTION 8

Describe the different stages of the assembly process for a 'two-pass' assembler for a given simple assembly language program.

ANSWER 8



PASS 1:

- Assign addresses to all statements in the program.
- Addresses of symbolic labels are stored.
- Some assemble directives will be processed.

PASS 2:

Translate opcode and symbolic operands.

- Generate data values defined by BYTE, WORD, etc.
- Assemble directives will be processed.
- Write the object program and assembly listing.

QUESTION 9

Define the term '**Assembly Directives**'.

ANSWER 9

- **Assembly directives** are instructions that are executed by an **assembler** at assembly time, not by a CPU at run time.
- The names of pseudo-ops often start with a **dot** to distinguish them from machine instructions.

Category	Directives
Compiler-Use-Only Directives	.bgnb .endb .file .gjsrlive .gjsrsaved .lab .livereg .loc .option .ugen .vreg
Location Control Directives	.align .data .rdata .sdata .space .text
Symbol Declaration Directives	.extern .globl .struct symbolic equate .weakext
Routine Entry Point Definition Directives	.aent .ent

QUESTION 10

What is a **Macro**?

ANSWER 10

Macro is a single instruction that expands automatically into a set of instructions to perform a particular task.