Victoria University of Bangladesh

Final -Term Examination– summer 2022

Submitted To:

Syeda Tithi

Lecturer

Victoria University of Bangladesh

Submitted By:

Name: Anny Konika Das,

ID: 2120190011

Program: B.Sc in CSE.

Semester: summer-2022.

Course Title: Computer Organization & Assembly Language.

Course Code: CSE-233.

1.NO.Qus.Ans:

a)ans: Number of operands: Instruction sets may be categorized by the maximum number of operands explicitly specified in instructions.(In the examples that follow, a, b, and c are (direct or calculated) addresses referring to memory cells, while reg1 and so on refer to machine registers.)

1. 0-operand (zero-address machines), so called stack machines: All arithmetic operations take place using the top one or two positions on the stack: push a, push b, add, pop c. For stack machines, the terms "0-operand" and "zero-address" apply to arithmetic instructions, but not to all instructions, as 1-operand push and pop instructions are used to access memory.
2. 1-operand (one-address machines), so called accumulator machines, include early computers and many small microcontrollers: most instructions specify a single right operand(that is, constant, a register, or a memory location), with the implicit accumulator as the left operand (and the destination if there is one): load a, add b, store c. A related class is practical stack machines which often allow a single explicit operand in arithmetic instructions: push a, add b, pop c.
3. 2-operand — many CISC and RISC machines fall under this category:
4. CISC — often load a,reg1; add reg1,b; store reg1,c on machines that are limited to one memory operand per instruction; this may be load and store at the same location.
5. CISC — move a->c; add c+=b.
6. RISC — Requiring explicit memory loads, the instructions would be: load a,reg1; load b,reg2; add reg1,reg2; store reg2,c
7. 3-operand, allowing better reuse of data:[4]
8. CISC — It becomes either a single instruction: add a,b,c, or more typically: move a,reg1; add reg1,b,c as most machines are limited to two memory operands.
9. RISC — arithmetic instructions use registers only, so explicit 2-operand load/store instructions are needed: load a,reg1; load b,reg2; add reg1+reg2->reg3; store reg3,c; unlike 2-operand or 1-operand, this leaves all three values a, b, and c in registers available for further reuse

10. more operands—some CISC machines permit a variety of addressing modes that allow more than 3 operands (registers or memory accesses), such as the VAX "POLY" polynomial evaluation instruction.

Due to the large number of bits needed to encode the three registers of a 3-operand instruction, RISC processors using 16-bit instructions are invariably 2-operand machines, such as the Atmel AVR, the TI MSP430, and some versions of the ARM Thumb. RISC processors using 32-bit instructions are usually 3-operand machines, such as processors implementing the Power Architecture, the SPARC architecture, the MIPS architecture, the ARM architecture, and the AVR32 architecture.

B)ans: <u>The difference between programming Low level and High level language :</u>

The difference between programming Low level and High level language are given below:

Low-level languages:

Machine languages and the assembly languages that represent them (collectively termed low-level programming languages) tend to be unique to a particular type of computer. For instance, an ARM architecture computer (such as may be found in a PDA or a hand-held videogame) cannot understand the machine language of an Intel Pentium or the AMD Athlon 64 computer that might be in a PC.

Higher-level languages:

Though considerably easier than in machine language, writing long programs in assembly language is often difficult and is also error prone. Therefore, most practical programs are written in more abstract high-level programming languages that are able to express the needs of the programmer more conveniently (and thereby help reduce programmer error). High levellanguages are usually —compiled‖ into machine language (or sometimes into assembly language and then into machine language) using another computer program called a compiler. High level languages are less related to the workings of the target computer than assembly language, and more related to the language and structure of the problem(s) to be solved by the final program.

Summing up the differences between low level and high level programming language.

Low level language:

1. They are faster than high level language.
2. Low level languages are memory efficient.
3. Low level languages are difficult to learn.
1. Programming in low level requires additional.
2. knowledge of the computer architecture.
3. They are machine dependent and are not portable.
4. They provide less or no abstraction from the hardware.
5. They are more error prone.

High level language:

1. They are comparatively slower.
2. High level languages are not memory efficient.
3. High level languages are easy to learn.
4. Programming in high level do not require any additional.
5. knowledge of the computer architecture.
6. They are machine independent and portable.
7. They provide high abstraction from the hardware.
8. They are less error prone.

C)ans:

Given that,

(7 x 600) x (5 x 600) = 4200 x 3000 pixels

= 12600000 pixels (÷ 8)

= 1575000 bytes (÷ 1024)

= 1538.08 Kb (÷ 1024)

= 1.5 Mb.

2.NO.Qus.Ans:

a)Ans. DMA Controllers:

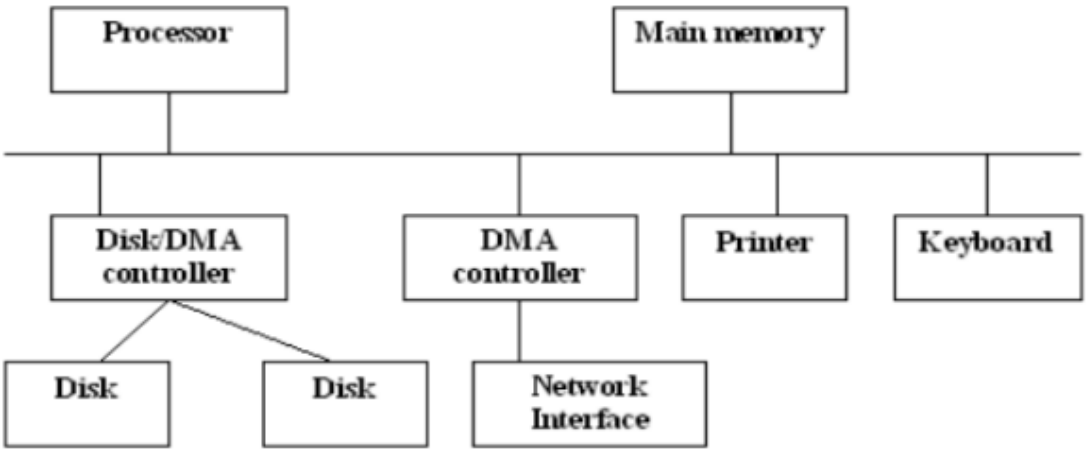
DMA or Direct Memory Access Controller is an external device that controls the transfer of data between I/O device and memory without the involvement of the processor. DMA controller is a control circuit that performs DMA transfers, is a part of the I/O deviceinterface. It performs functions that normally be carried out by the processor. DMA controller must increment the memory address and keep track of the number of transfers. The operations of DMA controller must be under the control of a program executed by the processor. To initiate the transfer of block of words, the processor sends the starting address, the number of words in the block and the direction of the transfer. On receiving this information, DMA controller transfers the entire block and informs the processor by raising an interrupt signal. While a DMA transfer is taking place, the processor can be used to execute another program. After the DMA transfer is completed, the processor can return to the program that requested the transfer.

• Three registers in a DMA interface are:

• Starting address

• Word count

• Status and control flag.



Use of DMA controllers in a computer system

A conflict may arise if both the processor and a DMA controller or two DMA controllers try to use the bus at the same time to access the main memory. To resolve this, an arbitration procedure is implemented on the bus to coordinate the activities of all devices requesting memory transfers.

b)ans: ALU:

Short for Arithmetic Logic Unit, ALU is one of the many components within a computer processor. The ALU performs mathematical, logical, and decision operations in a computer and is the final processing performed by the processor. After the information has been processed by the ALU, it is sent to the computer memory.In some computer processors, the ALU is divided into two distinct parts, the AU and the LU. The AU performs the arithmetic operations and the LU performs the logical operations. In computing, an arithmetic and logic unit (ALU) is a digital circuit that

Design of simple units of ALU:

In ECL, TTL and CMOS, there are available integrated packages which are referred to as arithmetic logic units (ALU). The logic circuitry in this units is entirely combinational (i.e. consists of gates with no feedback and no flip-flops).The ALU is an extremely versatile and useful device since, it makes available, in single package, facility for performing many different logical and arithmetic operations. Arithmetic Logic Unit (ALU) is a critical component of a microprocessor and is the core component of central processing unit.

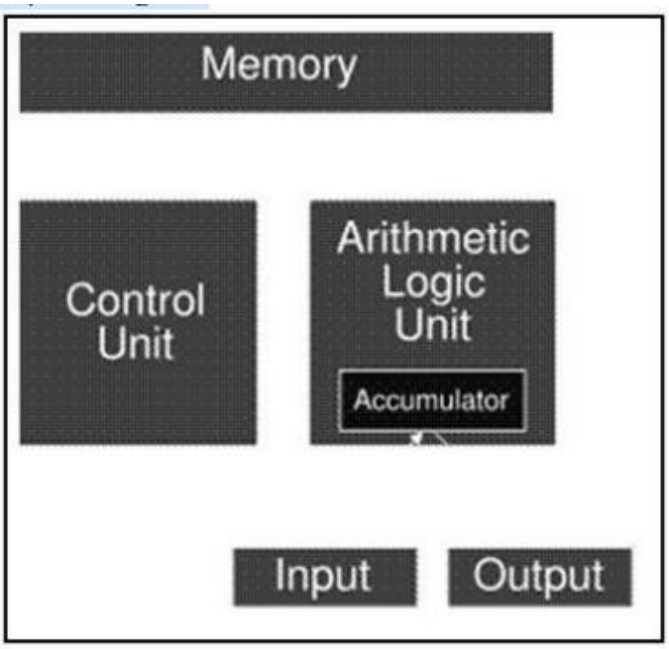
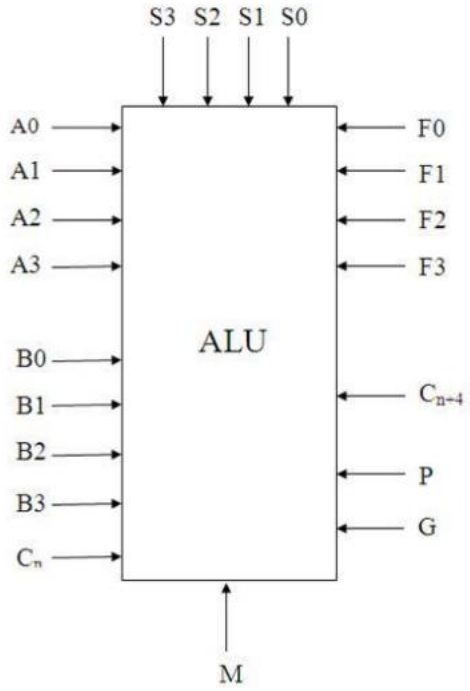


Fig.1 *Central Processing Unit (CPU)*

ALU‘s comprise the combinational logic that implements logic operations such as AND, OR and arithmetic operations, such as ADD, SUBTRACT.

Functionally, the operation of typical ALU is represented as shown in diagram below,



Characteristics Of ALU

The ALU is responsible for performing all logical and arithmetic operations.

- Some of the arithmetic operations are as follows: addition, subtraction, multiplication and division.

- Some of the logical operations are as follows: comparison between numbers, letter and or special characters.

- The ALU is also responsible for the following conditions: Equal-to conditions, Less-than condition and greater than condition.

C)ans: The hexadecimal number given is 5 G.A B 216

4-bit binary equivalent 1111110.101111011000

Hence the equivalent binary number is (1111110.101111011000)2.

3.No.Qus.Ans:

A)ans: Assembly Language Programming:

An assembly language is a low-level programming language for a computer, or other programmable device, in which there is a very strong (generally one-to-one) correspondence between the language and the architecture's machine code instructions. Each assembly language is specific to a particular computer architecture, in contrast to most high-level programming languages, which are generally portable across multiple architectures, but requireinterpreters or compiling. Assembly language is converted into executable machine code by a utility program referred to as an assembler; the conversion process is referred to as assembly, or assembling the code.

Current usage of Assembly Language:

There have always been debates over the usefulness and performance of assembly language relative to high-level languages. Assembly language has specific niche uses where it is important; see below. But in general, modern optimizing compilers are claimed to render high-level languages into code that can run as fast as hand-written assembly, despite the counter-examples that can be found. The complexity

of modern processors and memory sub-systems makes effective optimization increasingly difficult for compilers, as well as assembl programmers. Moreover, and to the dismay of efficiency lovers, increasing processor performance has meant that most CPUs sit idle most of the time, with delays caused by predictable bottlenecks such as I/O operations and paging. This has made raw code execution

speed a non-issue for many programmers. There are some situations in which developers might choose to use assembly language:

1. A stand-alone executable of compact size is required that must execute without recourse to the run-time components or libraries associated with a high-level language; this is perhaps the most common situation. For example, firmware for telephones, automobile fuel and ignition systems, air-conditioning control systems, security systems, and sensors.
2. Code that must interact directly with the hardware, for example in device drivers and interrupt handlers.
3. Programs that need to use processor-specific instructions not implemented in a compiler. A common example is the bitwise rotation instruction at the core of many encryption algorithms.
4. Programs that create vectorized functions for programs in higher-level languages such as C. In the higher-level language this is sometimes aided by compiler intrinsic functions which map directly to SIMD mnemonics, but nevertheless result in a one-to-one assembly conversion specific for the given vector processor.
5. Programs requiring extreme optimization, for example an inner loop in a processor-intensive algorithm. Game programmers take advantage of the abilities of hardware features in systems, enabling games to run faster. Also large scientific simulations require highly optimized algorithms, e.g. linear algebra with BLAS or discrete cosine transformation (e.g. SIMD assembly version from x264).
6. Situations where no high-level language exists, on a new or specialized processor, for example.
7. Programs that need precise timing such as
8. real-time programs such as simulations, flight navigation systems, and medical equipment. For example, in a fly-by-wire system, telemetry must be interpreted and acted upon within strict time constraints. Such systems must

eliminate sources of unpredictable delays, which may be created by (some) interpreted languages, automatic garbage collection, paging operations, or preemptive multitasking. However, some higher-level languages incorporate run-time components and operating system interfaces that can introduce such delays. Choosing assembly or lower-level languages for such systems gives programmers greater visibility and control over processing details.

9. cryptographic algorithms that must always take strictly the same time to execute, preventing timing attacks.

B)ans: Input output in assembly Language Program

Input/Output (I/O) instructions are used to input data from peripherals, output data to peripherals, or read/write input/output controls. Early computers used special hardware to handle I/O devices. The trend in modern computers is to map I/O devices in memory, allowing the direct use of any instruction that operates on memory for handling I/O.

- **IN Input;** MIX; initiate transfer of information from the input device specified into consecutive locations starting with M, block size implied by unit
- **OUT Output;** MIX; initiate transfer of information from consecutive locations starting with M to the output device specified, block size implied by unit.
- **IOC Input-Output Control;** MIX; initiate I/O control operation to be performed by designated device
- **JRED** Jump Ready; MIX; Jump if specified unit is ready (completed previous IN, OUT, or IOC operation); if jump occurs, J-register loaded with the address of the instruction which would have been next if the jump had not been taken.
- **JBUS** Jump Busy; MIX; Jump if specified unit is not ready (not yet completed previous IN, OUT, or IOC operation); if jump occurs, J-register loaded with the address of the instruction which would have been next if the jump had not been taken.

C)ans: Registers:

In computer architecture, a processor register is a small amount of storage available as part of a CPU or other digital processor. Such registers are (typically) addressed by mechanisms other than main memory and can be accessed more quickly. Almost all computers, load-store architecture or not, load data from a larger memory into registers where it is used for arithmetic, manipulated, or tested, by some machine instruction. Manipulated data is then often stored back in main memory, either by the same instruction or a subsequent one. Modern processors use either static or dynamic RAM as main memory, the latter often being implicitly accessed via one or more cache levels. A common property of computer programs is locality of reference: the same values are often accessed repeatedly and frequently used values held in registers improves performance. This is what makes fast registers (and caches) meaningful.

Various types of Registers:

Registers are normally measured by the number of bits they can hold, for example, an "8-bit register" or a "32-bit register". A processor often contains several kinds of registers, that can be classified accordingly to their content or instructions that operate on them:

1. **User-accessible registers –** The most common division of user-accessible registers is into data registers and address registers.
2. **Data registers -**Data registers can hold numeric values such as integer and floating-point values, as well as characters, small bit arrays and other data. In some older and low end CPUs, a special data register, known as the accumulator, is used implicitly for many operations.
3. **Address registers -** Address registers hold addresses and are used by instructions that indirectly access primary memory.
4. Some processors contain registers that may only be used to hold an address or only to hold numeric values (in some cases used as an index register whose value is added as an offset from some address); others allow registers to hold either kind of quantity. A wide variety of possible addressing modes, used to specify the effective address of an operand, exist.
5. The stack pointer is used to manage the run-time stack. Rarely, other data

stacks are addressed by dedicated address registers, see stack machine.

6. **Conditional registers-** Conditional registers hold truth values often used to determine whether some instruction should or should not be executed.
7. **General purpose registers (GPRs)** can store both data and addresses, i.e., they are combined Data/Address registers and rarely the register file is unified to include floating point as well.
8. **Floating point registers (FPRs)** store floating point numbers in many architectures.
9. **Constant registers** hold read-only values such as zero, one, or pi.
10. Vector registers hold data for vector processing done by SIMD instructions (Single Instruction, Multiple Data).
11. **Special purpose registers (SPRs)** hold program state; they usually include the program counter (aka instruction pointer) and status register (aka processor status word). The aforementioned stack pointer is sometimes also included in this group. Embedded microprocessors can also have registers corresponding to specialized hardware elements.
12. **Instruction registers** store the instruction currently being executed.
13. In some architectures, model-specific registers (also called machine-specific registers) store data and settings related to the processor itself. Because their meanings are attached to the design of a specific processor, they cannot be expected to remain standard between processor generations.
14. Control and status registers – There are three types: program counter, instruction registers and program status word (PSW).
15. Registers related to fetching information from RAM, a collection of storage registers located on separate chips from the CPU (unlike most of the above, these are generally not architectural registers):
    - Memory buffer register (MBR).
    - Memory data register (MDR)
    - Memory address register (MAR)
    - Memory Type Range Registers (MTRR)

    Hardware registers are similar, but occur outside CPUs.

“THE END”