Victoria University of Bangladesh

Department of Computer Science & Engineering

Name: Ashit Kumar (Abir)

Student ID: 2221220011.

Final Assessment

Program: B. Sc. in (CSE)

Semester: Spring-2024

Batch: 22nd (Evening)

Course Code: CSE-233

Course Title: Computer Organization & Assembly Language.

Ans to the ques No- 1 (a)

1. Number of operands in assembly language:

The number of operands in assembly language instructions can very depending on the specific intruction and the architecture of the processor being used.

2. Zero - operand Instruction (No operands):

Some assembly language instructions don't require any operands. These instructions typically perform simple operations or control flow actions without the need for additional data. Example include NOP (no operation), HALT (Halt execution).

3. One - Operand Instruction (Unary operation):

These instructions operate on a single operand. The operand could be register, a memory location, or an immediate value. Examples include ~~be a register~~ MOV (move data), INC (increment), and DEC (decrement)

4. Two-Operand Instructions (Binary operations):

These instructions operands on two opereands. The operands can be registers, memory locations or immudiate values. Examples include ADD (addition), SUB (subtraction), and AND (bitwise AND)

Some assemble languages also support instructions with more than two operands, but they are less common and typically involve specialized operations.

Ans to the ques No-1 (b)

Differenee between programming between programing - Low-level, Higher-level, language:

Certainly! Here's a breakdown of the differences between programing languages, loco-level languages, and higher-level languages:

#Programming Language: A programming language is a formal language with aset of syntax and rules used to instruct a computer to perfor specific takes on operations. programming languges provide a way for humans to communicate instructions to computers to.

computers in a structured and understandable manner. Examples of programming languages include C, Java, Python, and JavaScript.

*Low-level Language:

A low-level language is a programming language that provides little or no abstraction from the hardware of the computer. Low-level languages are closer to the binary machine code that the computer understands and directly executes

* Higher-level Language:

A higher-level language is a programming language that provides a higher level of abstraction from the hardware of the computer. Higher-level languages are designed to be more user-friendly and easier to read, write, and easier to read, write and maintain Compared to low-level languages. They typically used more natural languages constructs and libararies for common tasks. Examples of Higher-level languages include Python, Java, C++, and JavaScript.

Ans to the ques No-1(c)

To calculate the amount of memory required to store the graphic, we need to first determine the total number of pixels in the image. We can do this by multiplying the width and height of the image by the resolution (dpi) and then converting it to bytes.

Given:
- Image width = 7 inches
- Image height = 5 inches
- Resolution (dpi) = 6000

First, let's convert the image dimensions from inches to pixels using the given dpi:

- Image width in pixels = Image width (inches) * Resolution (dpi) = 7 inches * 6000 dpi = 42000 pixels
- Image height in pixels = Image height (inches) * Resolution (dpi) = 5 inches * 6000 dpi = 30000 pixels

Now, we calculation the total number of pixels in the image.

· Total number of pixels = ~~in the image~~:
Image width in pixels * Image height in pixels = 42000 pixels * 30000 pixels = 1,260,000,0
Pixels.

Next, we need to determine the number of bytes required to store each pixel, Since of the image resolution is given in dpi (dots per pixel to find the number of bytes per pixel;

1 inch = 2.54 centimeters (conversion factor)

· Dots per inch (dpi) = 6000 dpi

· Dots per centimeter (dpcm)

Ans to the ques No-2(a)

DMA (Direct memory Access) Controllers are specialized hardware components within a Computer system that facilitate data between peripherals and the main memory (RAM) without involving the cpu.

Here's how they work:

* when a peripheral device such as a hard

drive, network adapter, or graphics card needs to transfer data to or from memory, it sends a request to the DMA controller.

* The DMA controller device such as a hard drive, network adapter, and memory, bypassing the CPU. This frees up the CPU to perform otherpert taks while the data transfer occurs.

* DMA controllers typically have their own registers, control logic, and memory access circuits to manage the data tranfer process efficiently.

* one the data transfer is complete, the DMA controller may generate an interrupt to notify the CPU, allowing is to process the transfered data or perform any Aecessary follow-up-taks.

DMA Controllers are essential for achieving high-speed data transfers in computer systems, particularly in scenarios where large amounts of data need to be moved between peripherals and memory quickly, network communications or video processing.

Ans to the ques No-2(b)

Design simple units of ALU characteristics of ALU:

Certainly! Here are the fundamental characteristics of a simple Arithmetic Logic Unit (ALU):

1. Operations supported: An ALU typically performs arithmetic Logic operations. These operations may include addition, AND, OR, XOR, Shift left, Shift right, etc.

2. Input Data width: The ALU takes input data in the form of binary numbers. The width of these input data determines the range of values that can be processed. For example, a 4-bit ALU can operate on 4-bit binary numbers.

3. Out-put Data width: Similarly, the output data width determines the size of the result prod by the ALU. It's typically the same width as the input data, but it can be wider if necessary.

1. Control Signals: The ALU requires control signals to specify the operation to be performed. These control signals select the particular arithmetic or logic operation the ALU should execute.

Status Flags: Many ALUs also include status flags to indicate the result of an operations. Common flags include zero flag (Z), carry flag (C), overflow flag (V), and Sign flag (S). If there was an overflow during arithmetic operations.

Ans to the ques No-2(c)

Convert 5G.AB216 into an equivalent binary number:

To convert the decimal number 5G.AB216 into its equivalent binary representation, we can follow these steps:

1. Convert the integer part (5G) into binary.
2. Convert the fractional part (AB216) into binary. Let's convert each part separately:

(9)

1. Converting the integer part (5G) into binary:

* 5 in decimal is 101 in binary.

* G is equivalent to 6 in decimal (since G represents the 6th digit after the decimal point)

* Therefore, the integer part 5G in binary is 101.6.

2. Converting the fractional part (AB216) into binary.

* To convert the fractional part, we need to convert each digit to binary.

* A = 10, B = 11, 2 = 0010, 1 = 0001, 6 = 0110.

* Concatenation these binary representation of we get AB216 in binary 1011.0001000101 10.

Combining both part, the equivalent binary representation of 5G.AB216 is.

Copy code

```
101.61011.0001000101 10
```

Ans to the ques No-3(a)

Describe current usage of Assembly Language

Assembly language, is despite being low-level and somewhat cryptic compared to higher-level programming languages, is still utilized in various contexts today. Here are some of its current uses:

1. System programming: Assembly language is commonly used in system programming tasks where direct hardware manipulation or interaction with the operating system is required, BIOS firmware, and operating system components.

2. Embedded systems: In embedded systems programming, where resources are often limited and performance is critical, assembly language is sometimes used to optimize critical sections of code.

3. Reverse Engineering and Vulnerability Research: Assembly language is essential for reverse engineering software and analyzing security vulnerabilities. Security researchers and

analyze binary executables identify vulnerabilities, and develop exploits.

4. Performance Critical Applications: In certain performance-critical applications such as real-time systems, signal processing, game developments many resort to writing critical sections of code in assembly language to squeeze out the maximum performance from the hardware.

Legacy System Maintenance: Many legacy systems and applications written in assembly language still exist today. Organizations may need to maintain or update these systems, requiring programmers with assembly language expertise.

Ans to the ques NO-3(b)

Discuss input in assembly language program:
In assembly language programming, handling input and output (I/O) involves interacting with external devices such as keyboard, display, files and network interfaces. Here's a discussion on input and output operations in assembly language programs.

1. Device I/O Instructions: Assembly language provides instructions to perform I/O operations directly with hardware devices. Examples of such instructions include IN and OUT in x86 assembly language for performing port I/O.

2. Interrupts: Many I/O operations are handled through interrupts. When an external device needs attention (e.g, a key press on a keyboard), it generates an interrupt handler routine, which can then read or write data to/from the device.

3. Memory-Mapped I/O: Some I/O device are memory-mapped, meaning that they appear as memory locations in the address space of the processor. Memory-mapped I/O simplifies I/O operations, as they can be performed using standard memory access instructions.

Direct memory Access (DMA): DMA Controllers can be utilized to transfer data between memory and external devices without involving the CPU.

Ans to the ques No-3(c)

Describe different types of registers

Registers are small, high-speed storage location within a CPU (central processing unit) that hold ~~that~~ data temparacity during processing. They come in various types, each serving specific function within a computer architecture.

1. Program Counter (PC): Also knows as the instruction pointer, it holds the memory address of the next instruction to be ~~fetched~~ executed.

2. Instruction Register (IR): Stores the current instruction being executed by the CPU. It holds the opcode, which specifies the operation to be performed, and may also hold operands or addresses depending on the instruction format.

3. Memory Address Register (MAR): Contains the memory address of the data to be read from or written to in the main memory.

4. Memory Buffer Register (MBR): Holds data being transferred to or from the main memory. It temporarily to the main memory.

5. Accumulator (Acc): Often used in arithmetic and logic operations, it temporarily stores the results of computation performed by the CPU.

6. Index Register (IX): Used for indexing operations, particularly in addressing memory locations. It holds an offset value that is added to a base address to calculate the effective address of operands.