****

**SUBMITTED BY:**

**UQBA GULZAR (067)**

**ZUNAIRA NOOR (075)**

**ZUNAIRA KHATOON (074)**

**HADIA ZAKIR (019)**

**SUBMITTED TO:**

**Engr. AHSEN ILYAS**

**COURSE TITLE:**

**CALD LAB**

**DEPARTMENT:**

**SOFTWARE ENGINEERING**

**SECTION:**

**BSE II-B**

**LAB # 9**

**To get familiar with different Architectures of computer**

**Questions:**

**1. Comment on Architecture of the following CPUs**

IBM Power2, Itanium, EPIC, Intel i860, Transmeta Crusoe, StarCore (Lucent/Motorolla), TigerSHARC (ADI), Infenon

**IBM Power2:**

In the early 1990s, IBM Power2 was an advanced RISC microprocessor architecture, based on the original POWER architecture.

Key features included:

* **Superscalar design:** Increased parallelism by allowing multiple instructions to be issued per cycle.
* **Increased functional units:** Highlighted six integer and four Floating Point Units, allowing for more complete operations.
* **Improved floating-point performance:** Supported fused multiply-add (FMA) instructions, which combine multiplication and addition operations in a single operation, greatly improving the efficiency of scientific calculations.
* **Branch Prediction:** Implemented advanced branch prediction to minimize pipeline bottlenecks and improve instruction flow.
* **Larger caches:** Include**s** larger caches that improve data access speed and overall performance.
* **Scalability:** Designed to support multiprocessor configurations, making it suitable for high-performance computing environments.

**Itanium (Intel)**

The Itanium processor, developed by Intel in collaboration with HP, was based on the EPIC (Explicitly Parallel Instruction Computing) architecture.

Key features include:

* **EPIC Architecture:** Focuses on exploiting instruction-level parallelism through compile-time timing**.**
* **VLIW (Very Long Instruction Word):** Used long instruction words that can encode multiple operations, facilitating parallel execution.
* **Predictability and Hypothesis:** Lowered branch performance and enabled speculative execution to increase pipeline efficiency.
* **Advanced Memory Architecture:** Designed for high-bandwidth memory subsystems to support large data sets and complex computations.
* **64-bit processing:** Designed for enterprise servers and high-performance computing applications.

**EPIC :**

EPIC is an architectural approach mainly used on Itanium processors.

Key considerations include:

* **Instruction-level parallelism:** Relies on the compiler to schedule instructions for parallel execution, minimizing hardware complexity.
* **Explicit** **Inference:** Supports speculative execution of instructions to improve performance.
* **Predicate execution:** Reduces branching by allowing conditional execution **in** the command stream.

**Intel i860:**

Introduced in the late 1980s, the Intel i860 was an early RISC microprocessor with notable features:

* **Concurrency:** Includes both scalar and vector processing units that support concurrent operations.
* **Graphics and Scientific Computing:** Targeted applications with significant floating point and graphics processing requirements.
* **High Bandwidth Memory Interface:** Improved data transfer capacity for memory intensive operations.

**Transmeta Crusoe:**

The Transmeta Crusoe processor was known for its innovative approach to power efficiency and x86 compatibility:

* **Code Morphing Software (CMS):** Dynamically translated x86 instructions into native VLIW instructions, enabling compatibility with x86 -software.
* **Low power consumption:** Targeted mobile devices and embedded devices with strict power requirements.
* **Long-term power management:** Adjust CPU operating parameters in real time to save energy.

**StarCore (Lucent/Motorola):**

Developed by Lucent and Motorola, StarCore was a high-performance DSP (digital signal processor) architecture:

* **High-performance DSP:** Optimized for signal processing tasks such as telecommunications and multimedia applications. .
* **Scalability:** Multiple cores are supported for parallel processing, improving the performance of complex tasks.
* **Energy Efficiency:** Designed to be energy efficient and compatible with embedded systems.
* **Flexible instruction:** Provides versatile instruction for various signal processing algorithms.

**TigerSHARC (ADI):**

The TigerSHARC DSP from Analog Devices is designed for demanding digital signal processing and boasts high computational density.

* **SIMD architecture:** Single Instruction, Multiple Data (SIMD) supported for simultaneous processing of multiple data points.
* **Flexible memory architecture:** Includes onboard memory and high-speed external memory interfaces to support real-time processing.
* **Low Latency:** Designed for applications that require minimal latency, such as real-time signal processing.

**Infineon:**

Infineon Technologies is known for its microcontrollers and embedded processors, especially for automotive and industrial applications:

* **Automotive Microcontrollers:** Specializes in processors for safety-critical and real-time control systems.
* **Energy efficiency:** Optimized for low energy consumption, very important for batteries and energy efficient devices.
* **Scalability and Integration:** Provides high integration with various peripherals and scalable performance to meet different application needs.
* **Security features:** Includes advanced security measures for applications that require secure data processing, such as automotive and industrial control systems.

**2.** **Daily life examples of embedded systems around us (Atleast 40)**

Embedded systems are integrated into various devices and applications we interact with daily.

Here are 40 examples of embedded systems around us:

1. **Smartphones:** Control everything from communication to internet browsing.
2. **Digital watches:** Offer timekeeping functions and additional features such as alarms and exercise tracking.
3. **Microwave ovens:** Adjust the cooking time and power level.
4. **Washing machines:** Check wash cycles, water levels and temperatures.
5. **Televisions:** Process and display broadcast signals and manage user interfaces.
6. **Remote controls:** You can control TVs, air conditioners and other devices.
7. **Refrigerators:** Adjust temperature, manage defrost cycles and control smart functions.
8. **Air conditioners:** Adjust temperature settings and modes.
9. **Dishwashers:** Check wash cycles and water temperature.
10. **Home security systems:** Monitor and control security cameras, alarms and sensors.
11. **Thermostats:** Controls heating and cooling systems.
12. **Smoke Detectors:** Monitors smoke and fire and triggers alarms.
13. **Carbon Monoxide Detector:** Detects harmful levels of carbon monoxide.
14. **Fitness Trackers:** Track physical activity, heart rate and sleep patterns.
15. **Digital Cameras:** Controls imaging and processing functions.
16. **Printers:** Manage print jobs and interact with computers.
17. **Intelligent home assistants:** Manage home automation systems and provide information services.
18. **Game consoles:** Process game data and manage user inputs.
19. **Smart Locks:** Provides keyless entry and security features.
20. **Automated lighting systems:** Control lighting based on schedules or sensor inputs.
21. **Bluetooth Headsets:** Control wireless audio streaming.
22. **Vehicle infotainment systems:** Controls audio, navigation and connectivity functions.
23. **Engine control units (ECU):** Controls the functions and performance of a vehicle's engine.
24. **Anti-lock braking systems (ABS):** Prevent the wheels from locking when braking.
25. **Airbag control systems:** Airbags deploy in the event of a collision.
26. **Tire Pressure Monitoring Systems (TPMS):** Monitors tire pressure and alerts drivers to problems.
27. **Cruise control systems:** Automatically maintain the set speed of the vehicle.
28. **Parking Sensors:** Assists in parking by detecting obstacles.
29. **ATMs:** manage banking transactions.
30. **POS Systems:** Retail Transaction Management.
31. **Elevators:** Check floor selection, door operation and security systems.
32. **Smart meters:** Monitor and report usage of utilities such as electricity and water.
33. **Medical equipment:** Includes insulin pumps, pacemakers and diagnostic equipment.
34. **Hearing aids:** Amplify the sound for the hearing impaired.
35. **Decompilers:** Decoding and displaying TV signals.
36. **Home routers:** Check your internet connection and network traffic.
37. **Electronic toys:** Provide interactive gaming experiences.
38. **Digital thermometers:** Measurement of body temperature and readings on screen.
39. **Electric toothbrushes:** Optimize brushing patterns and times.
40. **Smart speakers:** Stream audio and control smart home devices with voice commands.

**3. Name and Comment on Architecture of the processor of your PC and Mobile.**

**Name of the Architecture of the processor of PC:**

64-bit Operating System, x64-based processor.

**Comment:**

A 64-bit operating system running on an x64-based processor offers significant advantages in terms of performance, memory capacity, and security. With the ability to address substantially more memory than 32-bit systems, a 64-bit OS can efficiently manage large applications and multitask with ease, enhancing overall system responsiveness and user experience. The x64 architecture supports multiple cores and threads, allowing for improved parallel processing and better utilization of modern multi-core CPUs. Enhanced security features, such as Data Execution Prevention (DEP) and Address Space Layout Randomization (ASLR), protect against various exploits, making 64-bit systems more robust and secure. This combination has become the standard for modern computing, ensuring compatibility with a wide range of software and providing a future-proof foundation for technological advancements.

**Name of the Architecture of Mobile:**

ARM implementer 65 architecture 8 variant 0 part 3331 revision 4.

**Comment:**

The ARM Implementer 65 Architecture 8 Variant 0 Part 3331 Revision 4 refers to a specific configuration of an ARM Cortex-A series processor, known for its efficiency and performance in various computing environments. This particular implementation follows the ARMv8 architecture, which introduces 64-bit processing capabilities, enhancing both computational power and memory addressing capabilities compared to earlier 32-bit versions. The Cortex-A series is designed to deliver high performance while maintaining energy efficiency, making it ideal for a wide range of applications from mobile devices to embedded systems and IoT devices. The revision and variant details indicate ongoing refinements and optimizations, ensuring improved stability, security features, and performance enhancements. Overall, this ARM implementation exemplifies the balance of power efficiency and performance that ARM processors are renowned for, contributing to their widespread adoption across diverse technological landscapes.

**LAB # 10**

**To get familiar with emulator 8086**

**Exercice 1.1 :**

**Comment on the Architecture of intel 8086 Processor.**

**Intel 8086 Processor**

**Architecture: 16-bit x86 (CISC - Complex Instruction Set Computing)**

**Key Features:**

* **16-bit Microprocessor: Handles 16-bit data and instructions.**
* **Registers: Includes 14 16-bit registers, such as general-purpose (AX, BX, CX, DX), segment (CS, DS, SS, ES), pointer (SP, BP), index (SI, DI), instruction pointer (IP), and flags register.**
* **Segmented Memory: Can address up to 1 MB of memory using a segmented approach, with each segment being 64 KB.**
* **Bus Interface Unit (BIU) and Execution Unit (EU):**
  + **BIU: Manages instruction fetching, memory addressing, and I/O.**
  + **EU: Executes instructions and performs arithmetic and logic operations.**
* **Pipelining: Features a simple 6-byte prefetch queue for instruction fetching, improving execution speed.**
* **Instruction Set: CISC design with a wide variety of instructions and addressing modes.**
* **Clock Speed: Operates at speeds ranging from 5 MHz to 10 MHz.**
* **Interrupts: Supports both hardware and software interrupts with an interrupt vector table.**

**Comment:**

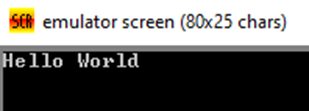
**The Intel 8086, released in 1978, is a foundational processor in the x86 family. Its 16-bit architecture, segmented memory model, and CISC instruction set enabled complex computing tasks and efficient memory usage. The BIU and EU separation facilitated basic pipelining, enhancing performance. The 8086's design has significantly influenced modern CPU architectures, making it a pivotal development in computing history.**

**Exercise 1.2 :**

**Hello world Program**

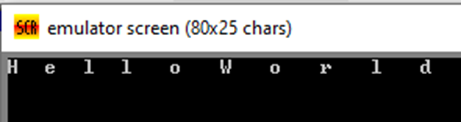
|  |
| --- |
| **; Title Hello World Program**  **org 100h**  **.data ; Declare variables**  **hello\_message db 'Hello World','$'**  **.code ; Write code**  **main proc**  **mov ax,@data ; Copy the address of data**  **mov ds,ax ; Segment into DS register**  **mov dx,offset hello\_message**  **mov ah,9 ; MS-Dos Function to display string**  **INT 21H  ret**    **mov ax,4C00h ; Halt the program and return control to OS**  **INT 21H**  **main endp**    **end main ; Mark the end of the source file** |

**OUTPUT:**

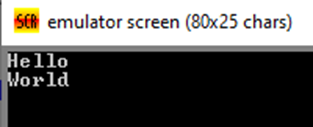
****

**Exercise 1.3 : Load all example of assembly language one by one in Emu8086 and execute them. Practice all these examples and analyze output.**

|  |
| --- |
| **org 100h**  **mov ah, 02h ; Function to display character**  **mov dx, 'H' ; Display 'H'**  **int 21h**  **mov dx, 0Dh ; Move to a new line**  **int 21h**  **mov dx, 0Ah ; Move to a new line**  **int 21h**  **mov dx, 'e' ; Display 'e'**  **int 21h**  **mov dx, 0Dh ; Move to a new line**  **int 21h**  **mov dx, 0Ah ; Move to a new line**  **int 21h**  **mov dx, 'l' ; Display 'l'**  **int 21h**  **mov dx, 0Dh ; Move to a new line**  **int 21h**  **mov dx, 0Ah ; Move to a new line**  **int 21h**  **mov dx, 'l' ; Display 'l'**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, 'o' ; Display 'o'**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, 'W' ; Display 'W'**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, 'o' ; Display 'o'**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, 'r' ; Display 'r'**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, 'l' ; Display 'l'**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, 'd' ; Display 'd'**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov dx, ' ' ; Display space**  **int 21h**  **mov ah, 4Ch ; Halt the program and return control to OS** |

****

|  |
| --- |
| **org 100h**  **mov ah, 09h ; Function to display string**  **mov dx, offset hello\_message ; Display 'Hello' on the first line**  **int 21h**  **mov dx, offset newline ; Move to a new line**  **int 21h**  **mov dx, offset world\_message ; Display 'World' on the second line**  **int 21h**  **mov ah, 4Ch ; Function to exit the program**  **int 21h**  **hello\_message db 'Hello$' ; 'Hello' text**  **world\_message db 'World$' ; 'World' text**  **newline db 0Dh, 0Ah, '$' ; Newline characters with a terminating dollar sign** |

****

|  |
| --- |
| **org 100h**  **mov ah, 09h ; Function to display string**  **mov dx, offset hello\_message ; Display 'Hello' on the first line**  **int 21h**  **mov dx, offset newline ; Move to a new line**  **int 21h**  **mov dx, offset world\_message ; Display 'World' on the second line**  **int 21h**  **mov ah, 4Ch ; Function to exit the program**  **int 21h**  **hello\_message db 'Hello$' ; 'Hello' text**  **world\_message db 'World$' ; 'World' text**  **newline db 0Dh, 0Ah, '$' ; Newline characters with a terminating dollar sign** |

**LAB # 11**

**Practice how to Input, Output and Display Characters as well as strings**

**Questions:**

**1.** Write a program that input a character from the user 2 times. The program will display the

character entered by user 2 times on screen in newline(without using a loop).

Sample input vs output:

Please enter the character? A

you have entered the character A

you have entered the character A

Please enter another character? B

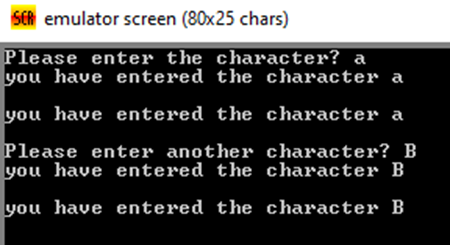
you have entered the character B

you have entered the character B

**PROGRAM:**

|  |
| --- |
| .model small  .stack 100h  .data  msg1 db 'Please enter the character? ', '$'  msg2 db 13, 10, 'you have entered the character ', '$'  msg3 db 13, 10, 'Please enter another character? ', '$'  msg4 db 13, 10, 'you have entered the character ', '$'  newline db 13, 10, '$'    .code  main proc  mov ax, @data  mov ds, ax    ; Prompt for first character  lea dx, msg1  mov ah, 09h  int 21h    ; Read first character  mov ah, 01h  int 21h  mov bl, al ; Store character in bl    ; Display the first character twice  lea dx, msg2  mov ah, 09h  int 21h    mov dl, bl  mov ah, 02h  int 21h    lea dx, newline  mov ah, 09h  int 21h    lea dx, msg2  mov ah, 09h  int 21h    mov dl, bl  mov ah, 02h  int 21h    lea dx, newline  mov ah, 09h  int 21h    ; Prompt for second character  lea dx, msg3  mov ah, 09h  int 21h    ; Read second character  mov ah, 01h  int 21h  mov bl, al ; Store character in bl    ; Display the second character twice  lea dx, msg4  mov ah, 09h  int 21h    mov dl, bl  mov ah, 02h  int 21h    lea dx, newline  mov ah, 09h  int 21h    lea dx, msg4  mov ah, 09h  int 21h    mov dl, bl  mov ah, 02h  int 21h    lea dx, newline  mov ah, 09h  int 21h    ; Terminate program  mov ah, 4Ch  int 21h  main endp  end main |

**OUTPUT:**

****

**2.** Write a program that inputs a character from the user in lowercase, the program will convert it to uppercase and will display it on console after conversion.

Hint: - The ASCII codes for lowercase letters (a-z) are 97-122. In order to convert a lowercase letter to uppercase letter, just subtract 32 from its ASCII code.

Sample input vs output:

Please enter an alphabet in lowercase? a

Upper case of the input is : A

**PROGRAM:**

.model small

.stack 100h

.data

msg1 db "Please enter an alphabet in lower case? $"

msg2 db 0DH, 0AH, "Upper case of the input is : $"

input db ?

output db ?

.code

main proc

mov ax, @data

mov ds, ax

; Display message asking for input

lea dx, msg1

mov ah, 09h

int 21h

; Read character from user

mov ah, 01h

int 21h

mov input, al

; Check if input is lowercase

cmp input, 'a'

jl not\_lowercase

cmp input, 'z'

jg not\_lowercase

; Convert lowercase to uppercase

sub input, 32

not\_lowercase:

; Display message for output

lea dx, msg2

mov ah, 09h

int 21h

; Display the converted character

mov dl, input

mov ah, 02h

int 21h

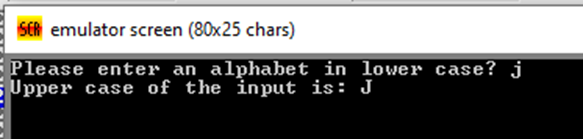
mov ah, 4Ch

int 21h

main endp

end main

**OUTPUT:**



**3.** Write an Assembly code which take an alphabet from user and print the next and previous

alphabets on the screen.

Sample input vs output:

Please enter an alphabet ? d

Previous alphabet in English grammar is : c

You have entered alphabet : d

Next alphabet in English grammar is : e

**PROGRAM:**

.model small

.stack 100h

.data

msg1 db "Please enter an alphabet: $"

msg2 db "Previous alphabet in English grammar is : $"

msg3 db "You have entered alphabet : $"

msg4 db "Next alphabet in English grammar is : $"

input db ?

prev\_alpha db ?

next\_alpha db ?

newline db 0Dh, 0Ah, '$'

.code

main proc

mov ax, @data

mov ds, ax

; Display message asking for input

lea dx, msg1

mov ah, 09h

int 21h

; Read character from user

mov ah, 01h

int 21h

mov input, al

; Calculate previous alphabet

mov al, input

dec al

mov prev\_alpha, al

; Display previous alphabet message

lea dx, msg2

mov ah, 09h

int 21h

; Display previous alphabet

mov dl, prev\_alpha

mov ah, 02h

int 21h

lea dx, newline

mov ah, 09h

int 21h

; Display user input message

lea dx, msg3

mov ah, 09h

int 21h

; Display user input

mov dl, input

mov ah, 02h

int 21h

lea dx, newline

mov ah, 09h

int 21h

; Calculate next alphabet

mov al, input

inc al

mov next\_alpha, al

; Display next alphabet message

lea dx, msg4

mov ah, 09h

int 21h

; Display next alphabet

mov dl, next\_alpha

mov ah, 02h

int 21h

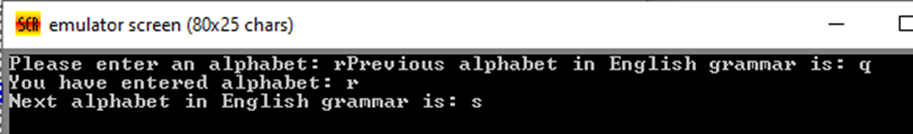
mov ah, 4Ch

int 21h

main endp

end main

**OUTPUT:**

****