

**Department of Electronics & Telecommunication Engineering****Experiment No.: SIX****Microcontroller & Applications****Name:****Batch/Rollno:****SAP ID:****Date:**

Objective:	Assembly language programming in simulation environment for AVR's microcontroller ATmega328p.
Outcome:	Programs that demonstrate the understanding and use of arithmetic, logical, rotate/shift and branching instructions in light of accomplishing specific tasks.
Tasks/Problem Statement:	<p>Students Perform following tasks:</p> <ol style="list-style-type: none"> Write a program to add two 8-bit numbers stored in at data memory location \$200 and \$201, data memory location and store the result at memory location \$300 and \$301, MSB first and then LSB. Verify the program working for worst case data condition. Repeat the above program to subtract two numbers ie [\$200] - [\$201] (contents of location 201 subtracted from location 200) Write a program to logically AND, OR and XOR two numbers stored at memory locations \$300 and \$400 and store the result of operation at location \$500. Write a program that inspects the contents of memory location \$200. If the content is zero (\$00) then store \$00 at memory location \$300, else store \$ff at location \$300. Write a program that inspects the lower nibble of the number stored at memory location \$200. If the lower nibble is \$F (ie all ones) then store \$ff at memory location \$300, if its \$0 (ie all zeros) then store \$00 at memory location \$300, else store \$7f at location \$300. Write a program to toggle specific bits in a register and / or logically shift (rotate the bits in register
Programs,	A)



Department of Electronics & Telecommunication Engineering

comments, brief
explanation and
output:

INPUT:-

```
PrathamGopani_6000221086_053_MCAExp6.asm
; Name: Pratham Gopani
; ID: 6000221086
; Date: 06/03/25

; Task (x): Add two 8-bit numbers stored at 0x00 and 0x01, store result at 0x02 (LSB) and 0x03 (MSB)
; Task (y): Subtract 0x00 - 0x01, store result at 0x02 (LSB), 0x03 (MSB)
; Task (z): AND, OR, XOR of 0x00 and 0x01, result at 0x04, 0x05, 0x06

; Task (x)
LDR R10, #0x00 ; Load first number from 0x00 into R10
LDR R11, #0x01 ; Load second number from 0x01 into R11
ADD R10, R11 ; Add R10 to R11, result in R10
STR R10, #0x02 ; Store R10 (carry) of result at 0x02
STR R10, #0x03 ; Store R10 (carry) of result at 0x03

; Task (y)
LDR R10, #0x00 ; Load subtrahend from 0x00 into R10
LDR R11, #0x01 ; Load minuend from 0x01 into R11
SUB R10, R11 ; Subtract R11 from R10
STR R10, #0x02 ; Store R10 (LSB) of result at 0x02
STR R10, #0x03 ; Store R10 (MSB) of result at 0x03

; Task (z)
AND R10, R11 ; AND of 0x00 and 0x01, result at 0x04
OR R10, R11 ; OR of 0x00 and 0x01, result at 0x05
XOR R10, R11 ; XOR of 0x00 and 0x01, result at 0x06
```

OUTPUT:-

```
PrathamGopani_6000221086_053_MCAExp6.asm
; Name: Pratham Gopani
; ID: 6000221086
; Date: 06/03/25

; Task (x): Add two 8-bit numbers stored at 0x00 and 0x01, store result at 0x02 (LSB) and 0x03 (MSB)
; Task (y): Subtract 0x00 - 0x01, store result at 0x02 (LSB), 0x03 (MSB)
; Task (z): AND, OR, XOR of 0x00 and 0x01, result at 0x04, 0x05, 0x06

; Task (x)
LDR R10, #0x00 ; Load first number from 0x00 into R10
LDR R11, #0x01 ; Load second number from 0x01 into R11
ADD R10, R11 ; Add R10 to R11, result in R10
STR R10, #0x02 ; Store R10 (carry) of result at 0x02
STR R10, #0x03 ; Store R10 (carry) of result at 0x03

; Task (y)
LDR R10, #0x00 ; Load subtrahend from 0x00 into R10
LDR R11, #0x01 ; Load minuend from 0x01 into R11
SUB R10, R11 ; Subtract R11 from R10
STR R10, #0x02 ; Store R10 (LSB) of result at 0x02
STR R10, #0x03 ; Store R10 (MSB) of result at 0x03

; Task (z)
AND R10, R11 ; AND of 0x00 and 0x01, result at 0x04
OR R10, R11 ; OR of 0x00 and 0x01, result at 0x05
XOR R10, R11 ; XOR of 0x00 and 0x01, result at 0x06
```

B:-

INPUT:-

```
PrathamGopani_6000221086_053_MCAExp6.asm
; Name: Pratham Gopani
; ID: 6000221086
; Date: 06/03/25

; Task (x): Add two 8-bit numbers stored at 0x00 and 0x01, store result at 0x02 (LSB) and 0x03 (MSB)
; Task (y): Subtract 0x00 - 0x01, store result at 0x02 (LSB), 0x03 (MSB)
; Task (z): AND, OR, XOR of 0x00 and 0x01, result at 0x04, 0x05, 0x06

; Task (x)
LDR R10, #0x00 ; Load first number from 0x00 into R10
LDR R11, #0x01 ; Load second number from 0x01 into R11
ADD R10, R11 ; Add R10 to R11, result in R10
STR R10, #0x02 ; Store R10 (carry) of result at 0x02
STR R10, #0x03 ; Store R10 (carry) of result at 0x03

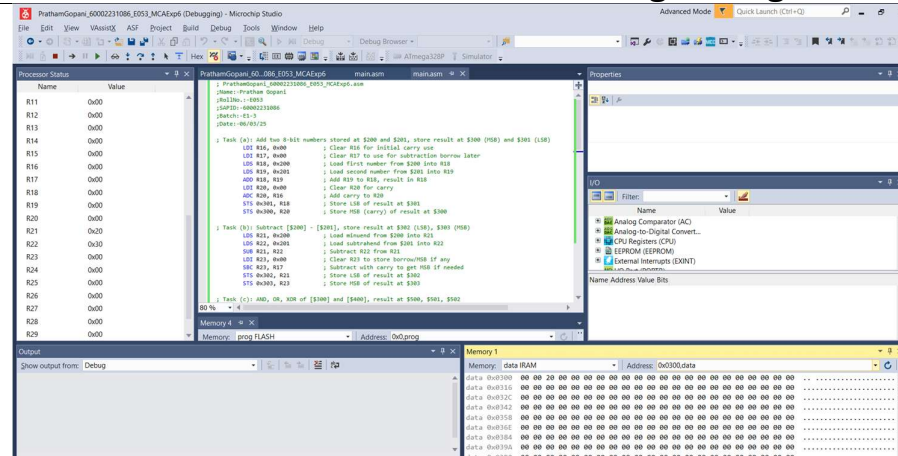
; Task (y)
LDR R10, #0x00 ; Load subtrahend from 0x00 into R10
LDR R11, #0x01 ; Load minuend from 0x01 into R11
SUB R10, R11 ; Subtract R11 from R10
STR R10, #0x02 ; Store R10 (LSB) of result at 0x02
STR R10, #0x03 ; Store R10 (MSB) of result at 0x03

; Task (z)
AND R10, R11 ; AND of 0x00 and 0x01, result at 0x04
OR R10, R11 ; OR of 0x00 and 0x01, result at 0x05
XOR R10, R11 ; XOR of 0x00 and 0x01, result at 0x06
```

OUTPUT:-

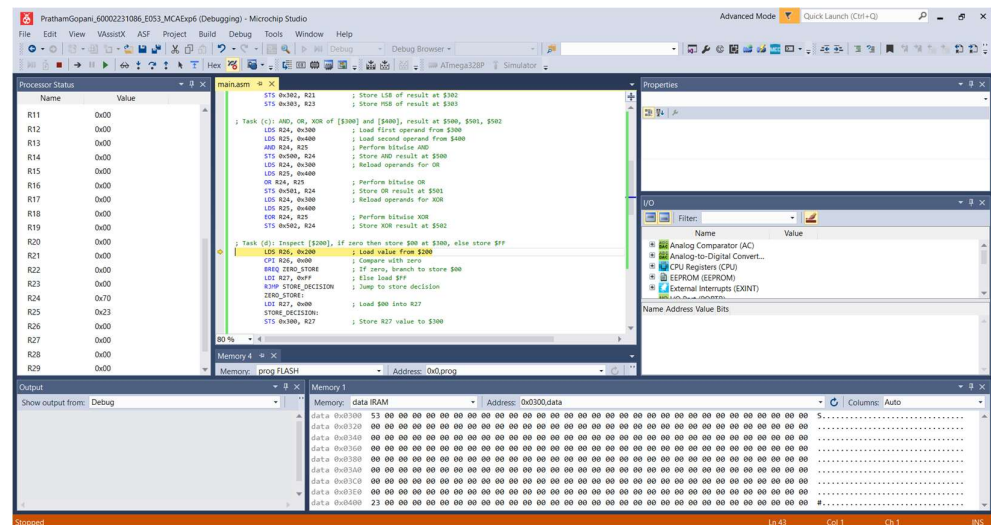


Department of Electronics & Telecommunication Engineering

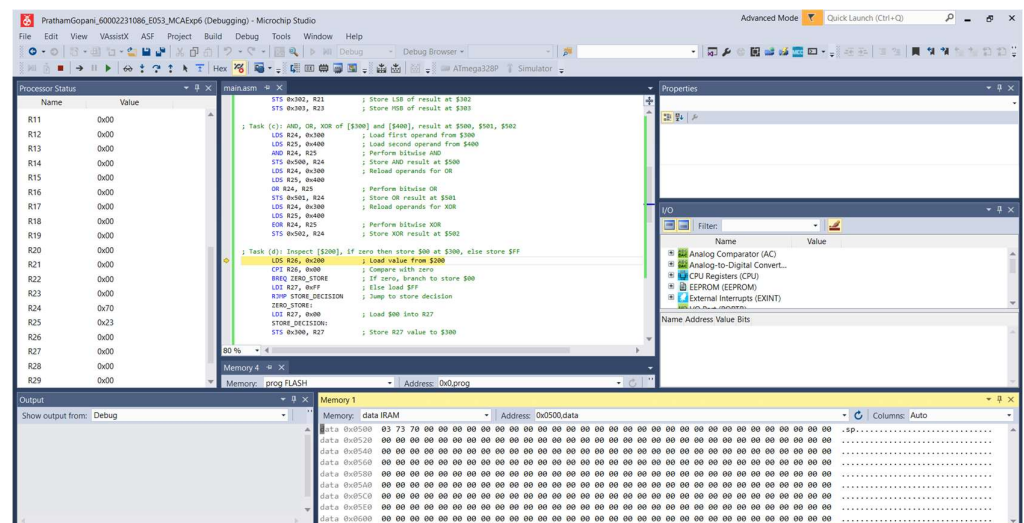


C:-

INPUT:-



OUTPUT:-



D:-

The screenshot displays the AVR Studio IDE with the following components:

- Top Menu Bar:** File, Edit, View, VAssistX, ASf, Project, Build, Debug, Tools, Window, Help.
- Toolbar:** Standard IDE tools including file operations, editing, and debugging.
- Processor Status Window:**
 - Registers:** R11 (0x00), R12 (0x00), R13 (0x00), R14 (0x00), R15 (0x00), R16 (0x00), R17 (0x00), R18 (0x00), R19 (0x00), R20 (0x00), R21 (0x00), R22 (0x00), R23 (0x00), R24 (0x00), R25 (0x00), R26 (0x00), R27 (0x00), R28 (0x00), R29 (0x0F).
 - Memory:** prog_FLASH (Address: 0x0prog).
- Assembly Code (main.asm):**

```

; Task (v): Check lower nibble of [R20], store conditional result to $300
LDI R20, 0x200          ; Load value from $200
ANDI R20, 0x0F          ; Mask upper nibble to isolate lower nibble
CPI R20, 0x0F           ; Check if lower nibble is $F
BRNG STORE_0F           ; If $F, branch to store $F
CPI R20, 0x00           ; Check if lower nibble is $0
BRNG STORE_00           ; If $0, branch to store $00
; Else store $FF
LDI R20, 0xFF           ; Load $FF into R20
RORF STORE_0F           ; Rotate $FF into R20
STORE_R0:               ; Load $00 into R20
LDI R20, 0x00
STORE_00:               ; Store result at $300
STI 0x300, R20

; Task (f): Toggle bits and rotate register
LDI R20, 0x00000000     ; Load test pattern in R20
COM R20                 ; Complement (toggle) all bits
ROL R20                 ; Rotate right through carry
ROR R20                 ; Rotate left through carry
HERE: JMP HERE          ; Infinite loop to end program

```
- I/O Window:** Filtered to show Name and Value.
 - Analog Comparator (AC)
 - Analog-to-Digital Convert...
 - CPU Registers (CPU)
 - EEPROM (EEPROM)
 - External Interrupts (EXTINT)
- Memory Window:**
 - Memory:** prog_FLASH (Address: 0x0prog).
 - Memory 1:** data RAM (Address: 0x200data).
- Output Window:** Shows output from: Debug.

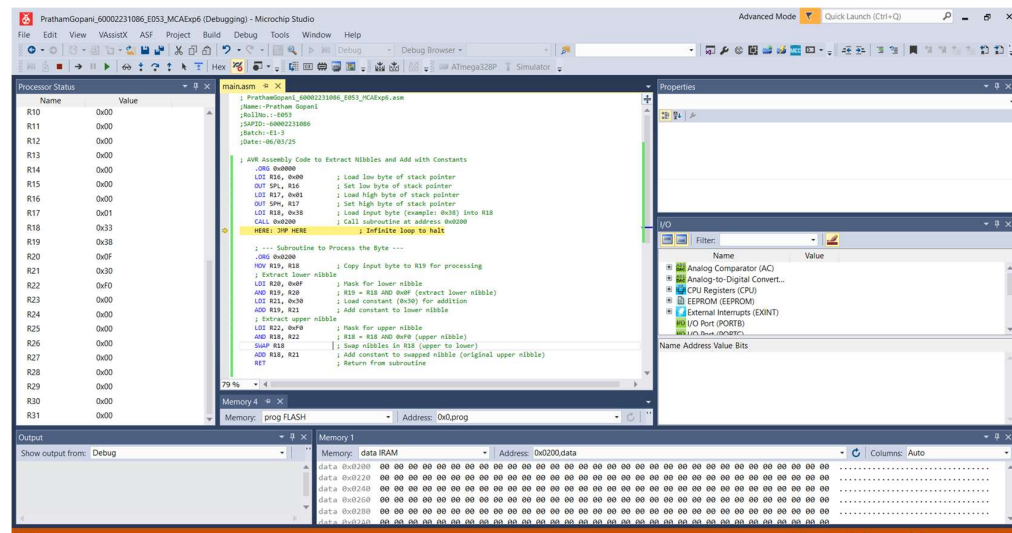
The screenshot displays the Proteus IDE interface with the following components:

- Assembly Editor (main.asm):** Contains assembly code for a simple calculator. The code uses the `LDI` (Load Immediate) instruction to load values into registers, the `ADD` (Add) instruction to perform addition, and the `ST` (Store) instruction to save the result in RAM. Comments in English explain each step. A yellow highlight is placed on the instruction `HERE: JMP HERE`, which is an infinite loop used to keep the processor running after the calculation is complete.
- Processor Status Window:** Shows the state of the 8051 microcontroller, including the program counter (PC) and the status of various flags.
- Properties Window:** Displays the properties of the selected component, such as the microcontroller, including its name, value, and various configuration options.
- Memory Window:** Shows the memory map of the system, including the RAM and ROM. The memory address 0x200 is highlighted, indicating the location where the result of the calculation is stored.



Department of Electronics & Telecommunication Engineering

1) Write an AVR assembly language program to extract and add the nibbles of a byte. The input byte is initially loaded into a register. The lower and upper nibbles are separated, added individually with given constants, and then processed through bit manipulation. Show the result in a R18 & R19.



Attach screen shots of the output

NOTE: Each code output should display student SAP id as well as Name of student along with date of performance.

Conclusion :

This experiment successfully demonstrates the implementation and simulation of arithmetic, logical, conditional branching, and bit manipulation instructions in AVR assembly for the ATmega328P microcontroller. Through a series of structured tasks, gained hands-on experience in handling memory operations, evaluating register content, and controlling program flow based on specific data conditions—validating both normal and edge-case scenarios.



Department of Electronics & Telecommunication Engineering

