EE2016 : Lab Report
Experiment 7

Putta Shravya EE22B032
Maadhav Patel EE22b033

September 2023

1 Aim

To implement arithmetic and logical manipulation programs using Atmel Atmega8 micro controller in assembly program emulation.

Google Drive link

Click here to view all codes and videos of problems.

1.1 Common 8-bit Addition

1.1.1 Code

```
.CSEG                  ; define memory space to hold program - code segment
 LDI ZL,LOW(2*NUM)     ; load byte addrss of LSB of word addrss
 LDI ZH,HIGH(2*NUM)    ;load byte addrss of MSB of word addrss
 LDI XL,0x60           ; load SRAM LSB of 16-bit address in X-register
 LDI XH,0x00           ; above's MSB|word address can be line number here
 LDI R16,00            ; clear R16, used to hold carry

 LPM R0,Z+             ;Z now follows byte addrssng. points MSB of NUM addr
 LPM R1,Z              ; Get second number (LSB of NUM addr) into R1,
 ADD R0,R1             ; Add R0 and R1,result in R0,carry flag affected
 BRCC abc              ; jump if no carry,
 LDI R16,0x01          ; else make carry 1

 abc:
 ST X+,R0              ; store result in given address in SRAM ie 0x60
 ST X,R16              ; store carry in next location 0x61
 NOP                   ; End of program, No operation

 NUM: .db 0xD3,0x5F    ; bytes to be added
```

1.1.2 Inferences and Learning Outcomes

Inferences:
Clock Counter=19. Inference regarding Register Transfers, Click cycles per instruction, latency and throughput are provided in the video in above gdrive link.
Learning Outcomes:
Memory Segmentation and Addressing
Loading Addresses and Registers
Loading Program Memory
Arithmetic Operations
Conditional Branching
Conditional Flag Manipulation

Storing Results and Data
Defining Constants
Comments and Code Structure

1.1.3 Observations

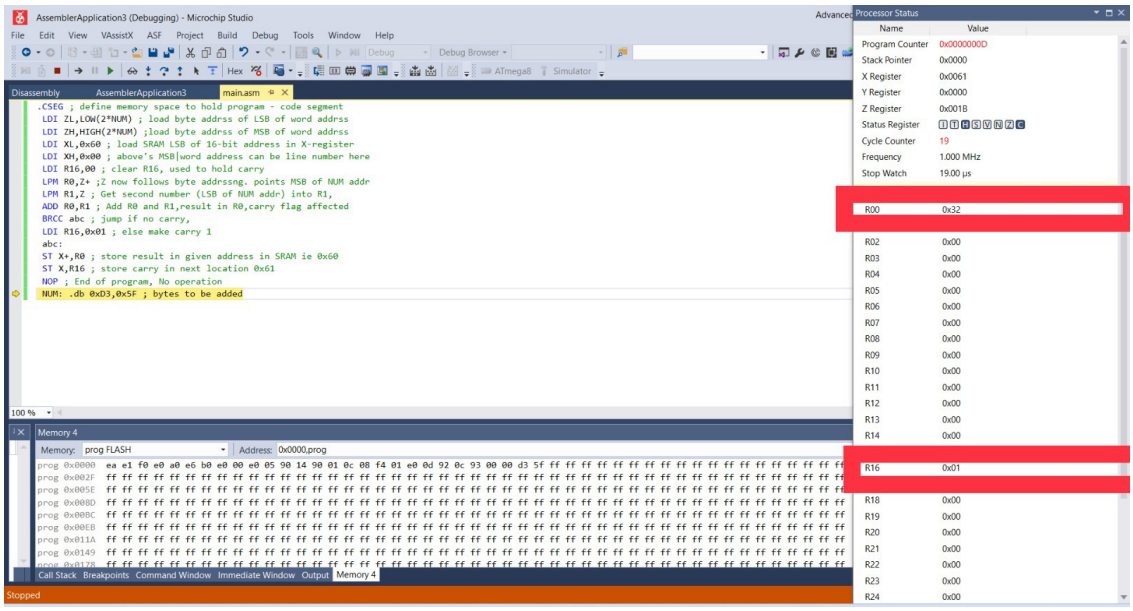


Figure 1: Addition

1.1.4 Flow Chart

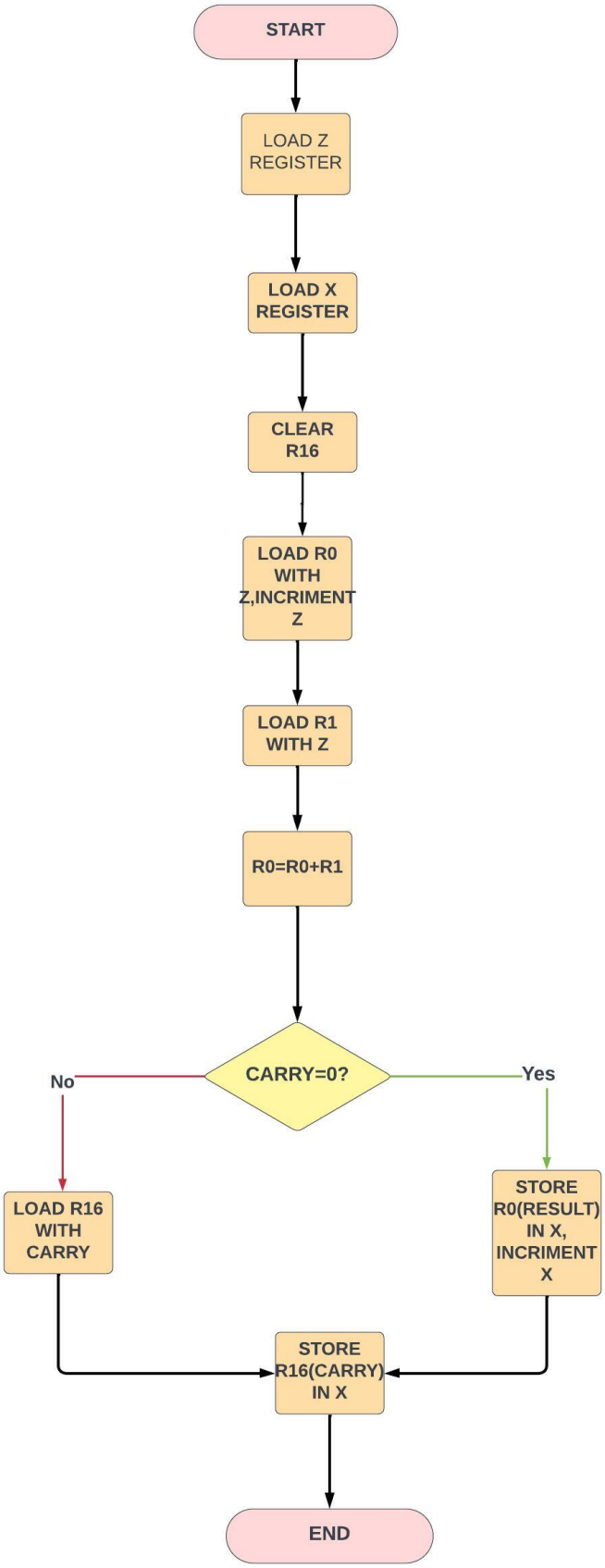


Figure 2: Addition

1.2 Division on AVR

1.2.1 Code

```
; Define memory locations
.EQU DIVIDEND = 0x0E        ; Memory location for dividend
.EQU DIVISOR = 0x10         ; Memory location for divisor
.EQU QUOTIENT = 0xF0        ; Memory location for quotient
.EQU REM = 0xFF             ; Memory location for remainder

; Load dividend and divisor from memory
LDI R16, DIVIDEND           ; Load dividend from memory into r16
LDI R17, DIVISOR            ; Load divisor from memory into r17
CLR R18                     ; Initialize quotient (r18) to 0
CLR R19                     ; Initialize remainder (r19) to 0

; Check for division by zero (divisor should not be zero)
CPSE R17, R1                ; Compare r17 (divisor) with 0
RJMP division_start         ; If not zero, start division
; Handle division by zero here (error condition)

division_start:
; Loop to perform division
LOOP:
 CP R16, R17              ; Compare dividend (r16) with divisor (r17)
 BRCC remainder           ; If dividend < divisor, remainder is in r16 (carry is not set)
 SUB R16, R17             ; Subtract divisor from dividend
 INC R18                  ; Increment quotient
 RJMP LOOP                ; Repeat the loop

remainder:
; Store quotient and remainder in memory
STS QUOTIENT, R18         ; Store quotient in memory
STS REM, R16              ; Store remainder in memory
```

1.2.2 Explanation

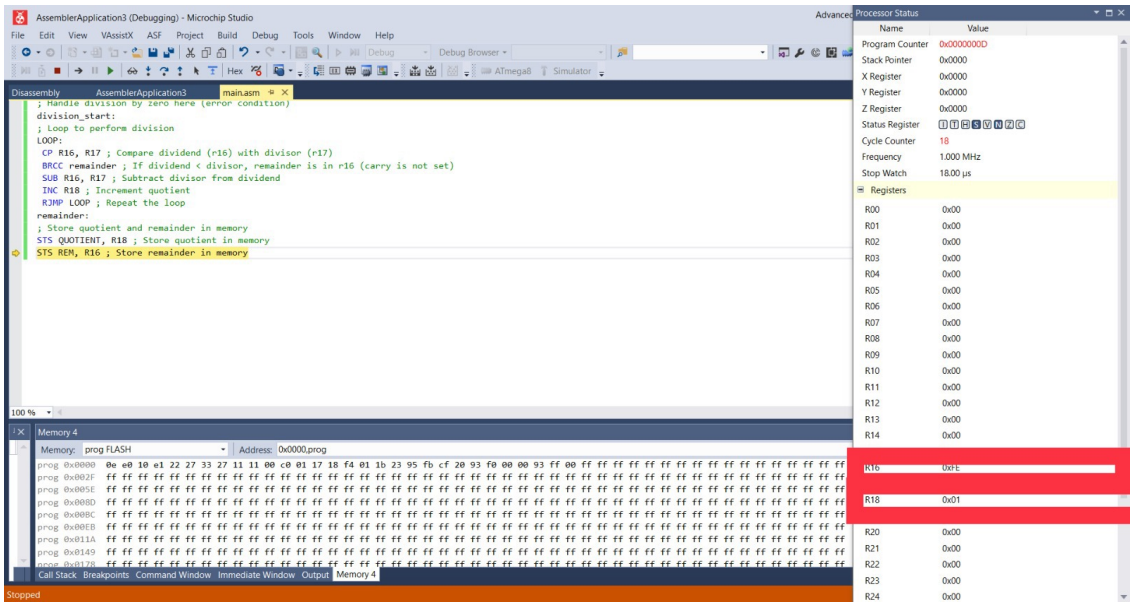
Memory Location Definitions:
DIVIDEND: Memory location for the dividend (initialized to 0x0E)
DIVISOR: Memory location for the divisor (initialized to 0x10)
QUOTIENT: Memory location for the quotient (initialized to 0xF0)
REM: Memory location for the remainder (initialized to 0xFF)
Loading Dividend and Divisor:
Load the dividend from memory (DIVIDEND) into register R16.
Load the divisor from memory (DIVISOR) into register R17.
Initialize Quotient and Remainder:
Initialize the quotient (register R18) to 0.
Initialize the remainder (register R19) to 0.
Check for Division by Zero:
Check if the divisor is zero (by comparing R17 with 0). If it's zero, handle the division by zero error condition.
Division Start:
If the divisor is not zero, proceed to the division.
Division Loop:
Compare the dividend (R16) with the divisor (R17).
If the dividend is less than the divisor, set the remainder to the dividend and exit the loop.
Subtract the divisor from the dividend and increment the quotient.
Repeat the loop.
Store Quotient and Remainder:
Store the quotient (in register R18) in memory at the location QUOTIENT.
Store the remainder (in register R16) in memory at the location REM.

End of Division Algorithm.

1.2.3 Inferences and Learning Outcomes

Inferences:
Cycle counter=18. Inference regarding Register Transfers, Click cycles per instruction, latency and throughput are provided in the video in above gdrive link.
Learning Outcomes:
Memory Locations and Equates
Loading Data from Memory
Register Initialization
Conditional Branching
Looping and Control Structures
Arithmetic Operations
Storing Results in Memory
Error Handling
Comments and Code Structure

1.2.4 Observations and Flow Charts



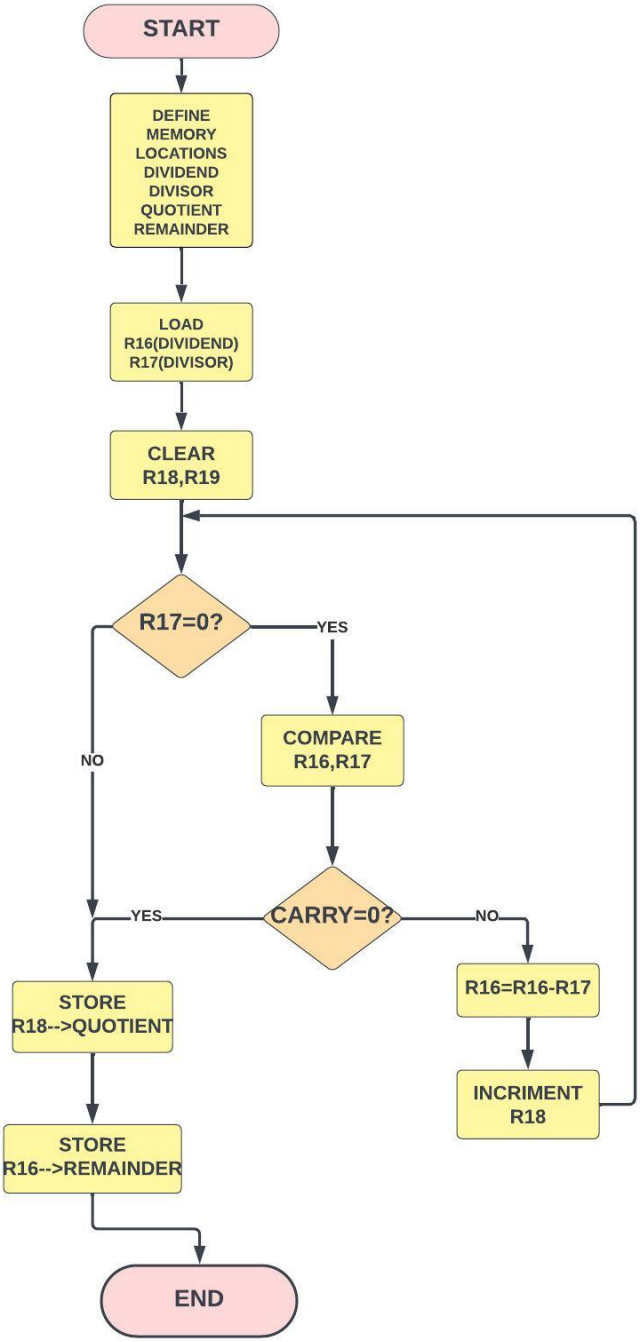


Figure 3: Division

1.3 Parity Detection

1.3.1 Code

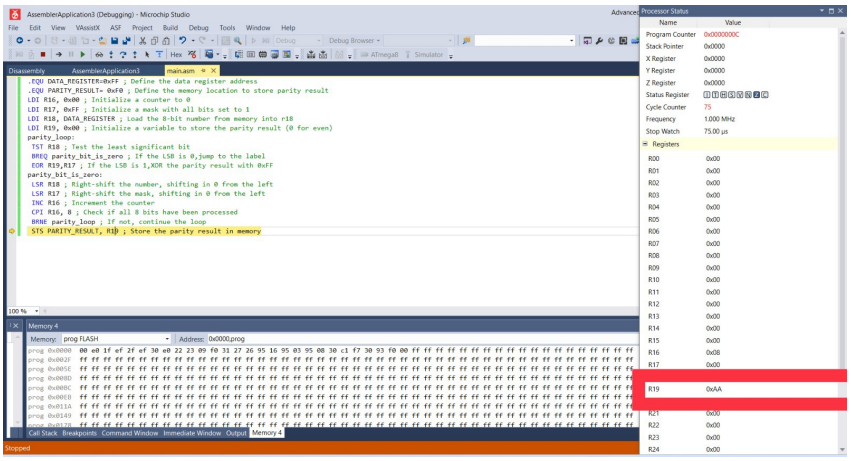
```
.EQU DATA_REGISTER=0xFF      ; Define the data register address
.EQU PARITY_RESULT= 0xF0     ; Define the memory location to store parity result

LDI R16, 0x00                ; Initialize a counter to 0
LDI R17, 0xFF                ; Initialize a mask with all bits set to 1
LDI R18, DATA_REGISTER       ; Load the 8-bit number from memory into r18
LDI R19, 0x00                ; Initialize a variable to store the parity result (0 for even)

parity_loop:
 TST R18                     ; Test the least significant bit
 BREQ parity_bit_is_zero     ; If the LSB is 0,jump to the label
 EOR R19,R17                 ; If the LSB is 1,XOR the parity result with 0xFF

parity_bit_is_zero:
 LSR R18                     ; Right-shift the number, shifting in 0 from the left
 LSR R17                     ; Right-shift the mask, shifting in 0 from the left
 INC R16                     ; Increment the counter
 CPI R16, 8                  ; Check if all 8 bits have been processed
 BRNE parity_loop            ; If not, continue the loop
 STS PARITY_RESULT, R19      ; Store the parity result in memory
```

1.3.2 Inferences and Learning Outcomes

Inferences:
Cycle Counter=75. Inference regarding Register Transfers, Click cycles per instruction, latency and throughput are provided in the video in above gdrive link.

Learning Outcomes:
Memory Locations and Equates
Loading Immediate Values and Data from Memory
Bitwise Operations
Conditional Branching
Register Manipulation
Storing Results in Memory
Looping and Control Structures
Logic and Parity Calculation
Comments and Code Structure

1.3.3 Observations and Flow Chart



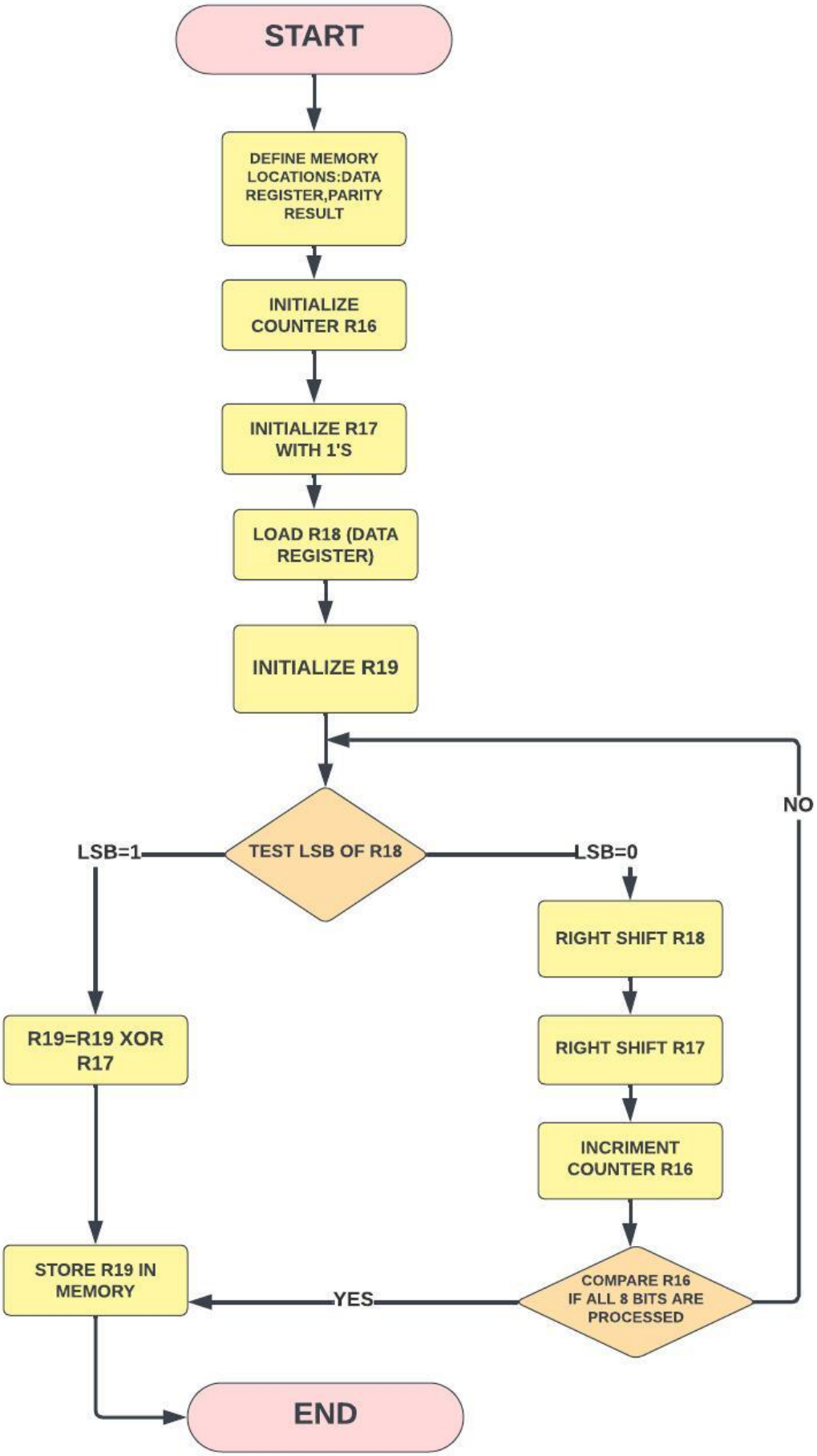


Figure 4: Parity

1.4 Largest and Smallest of a number set given

1.4.1 Code

```
.CSEG
LDI ZL,LOW(2*NUM)          ; load byte addrss of LSB of word addrss
LDI ZH,HIGH(2*NUM)         ; load byte addrss of MSB of word addrss
LDI XL,0x60                ; load SRAM LSB of 16-bit address in X-register
LDI XH,0X60                ; above's MSB|word address can be line number here

LPM R16,Z+                 ; Z now follows byte addrssng. points MSB of NUM addrss
LPM R17,Z                  ; Get second number (LSB of NUM addr) into R17

CP R16,R17                 ; Compare to find the greater number
BRCC forward               ; Jump if R16 is smaller number
MOV R16,R17                ; Move smaller value into R16
forward: ST X,R16          ; Store smaller value in X
NOP                        ; End of program,No operation

NUM: .db 0xD3,0x5F         ; bytes to be compared
```

1.4.2 Inferences and Learning Outcomes

Inferences:
Cycle counter=16.Inference regarding Register Transfers, Click cycles per instruction, latency and throughput are provided in the video in above gdrive link.
Learning Outcomes:
Memory Segmentation and Addressing
Loading Addresses and Registers
Loading Program Memory (LPM)
Arithmetic and Logic Operations
Conditional Branching
Register Manipulation
Storing Results in Memory
Data Representation
Comments and Code Structure

1.4.3 Observations

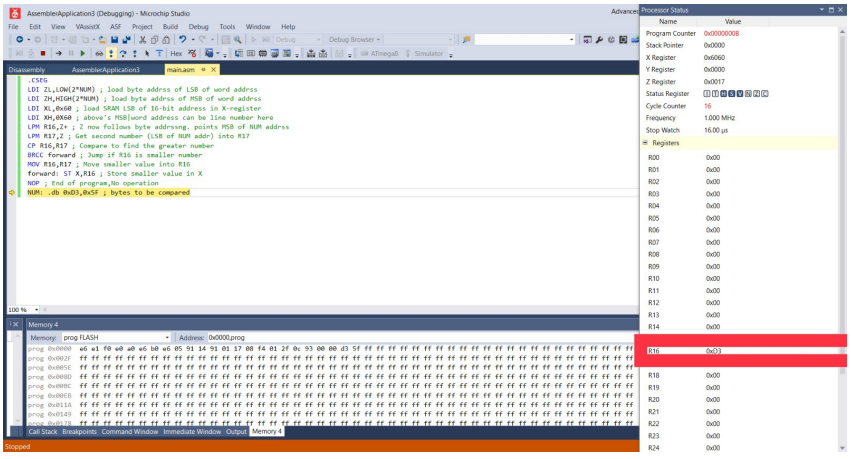


Figure 5: Largest,Smallest

1.4.4 Flow Chart

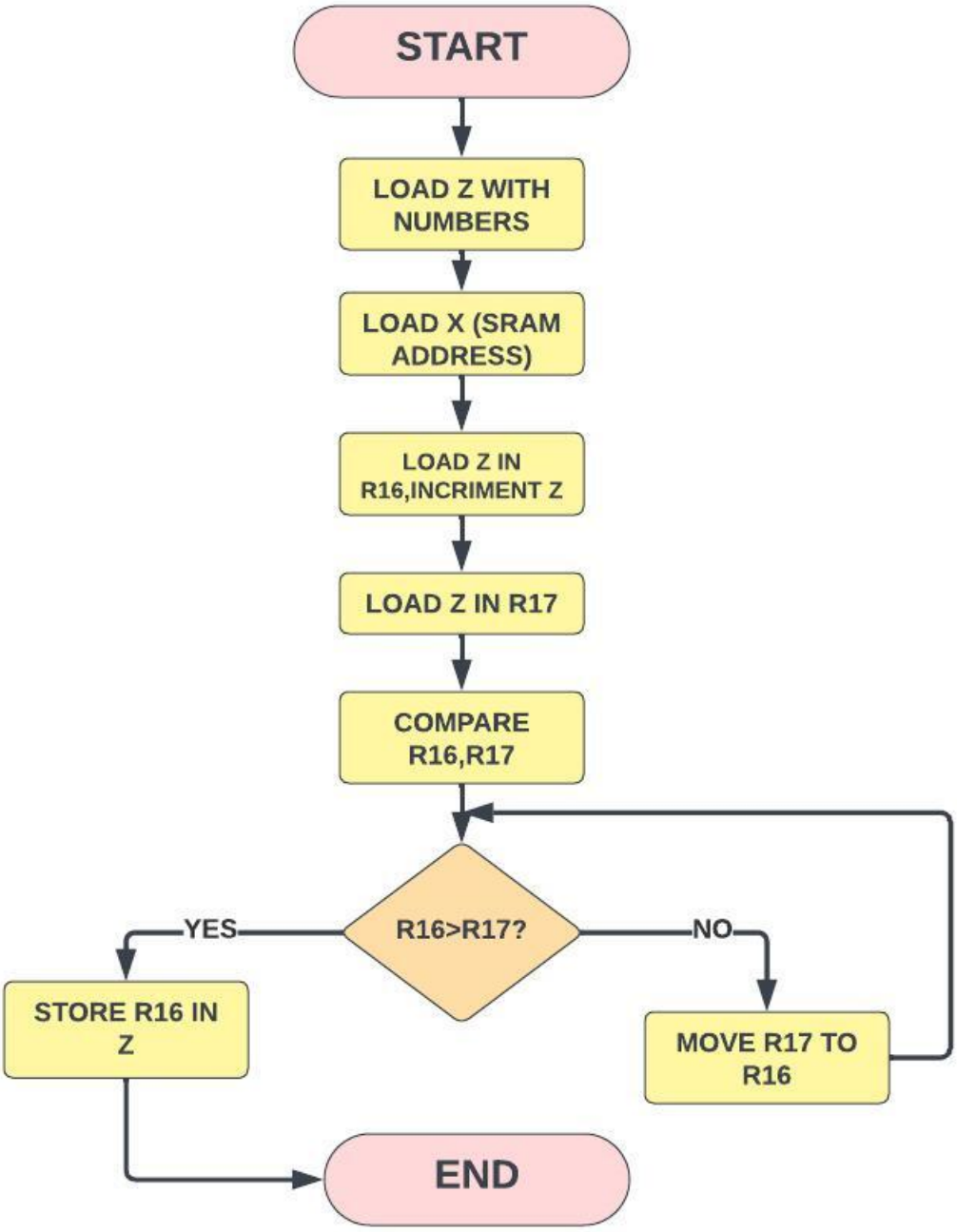


Figure 6: Largest,Smallest

1.5 Fibonacci Sequence

1.5.1 Code

```
.EQU N=10          ; Change this value to the desired Nth term

LDI R16, 0         ; Initialize a counter for the Fibonacci sequence
LDI R17, 0         ; Initialize a variable for the (N-1)th term (a(n-1))
LDI R18, 1         ; Initialize a variable for the Nth term (a(n))

fib_loop:
 CPI R16, N        ; Compare the counter with N
```

```
 BREQ fib_done    ; If counter equals N, exit the loop
 MOV R19, R18     ; Copy the Nth term to r19
 ADD R18, R17     ; Calculate the next term (a(n) = a(n-1) + a(n-2))
 MOV R17, R19     ; Update (N-1)th term to be the current Nth term
 INC R16          ; Increment the counter
 RJMP fib_loop    ; Repeat the loop

fib_done:
 MOV R0, R17 ; Store the Nth term in register R0
```

1.5.2 Inferences and Learning Outcomes

Inferences:
Cycle counter=86,for N=10. Inference regarding Register Transfers, Click cycles per instruction, latency and throughput are provided in the video in above gdrive link.
Learning Outcomes:
Symbolic Constants
Loading Immediate Values
Looping and Control Structures
Arithmetic Operations
Register Manipulation
Incrementing Counters
Conditional Branching
Storing Results
Comments and Code Structure

1.5.3 Observations and Flow Charts

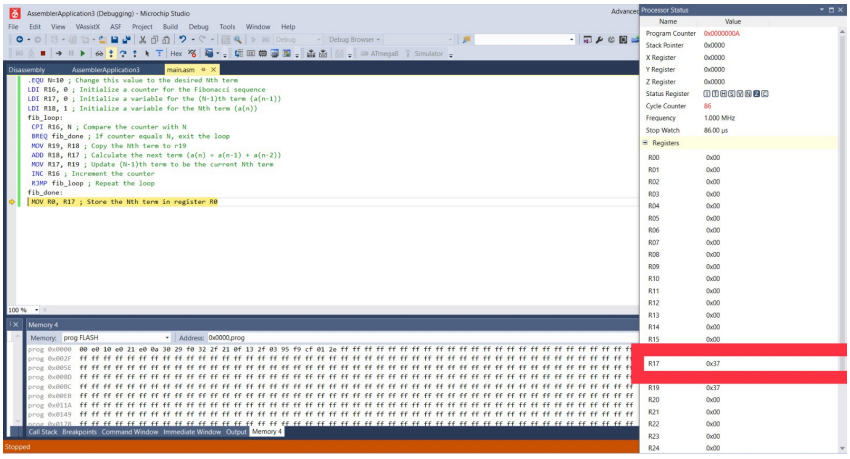


Figure 7: Fibonacci

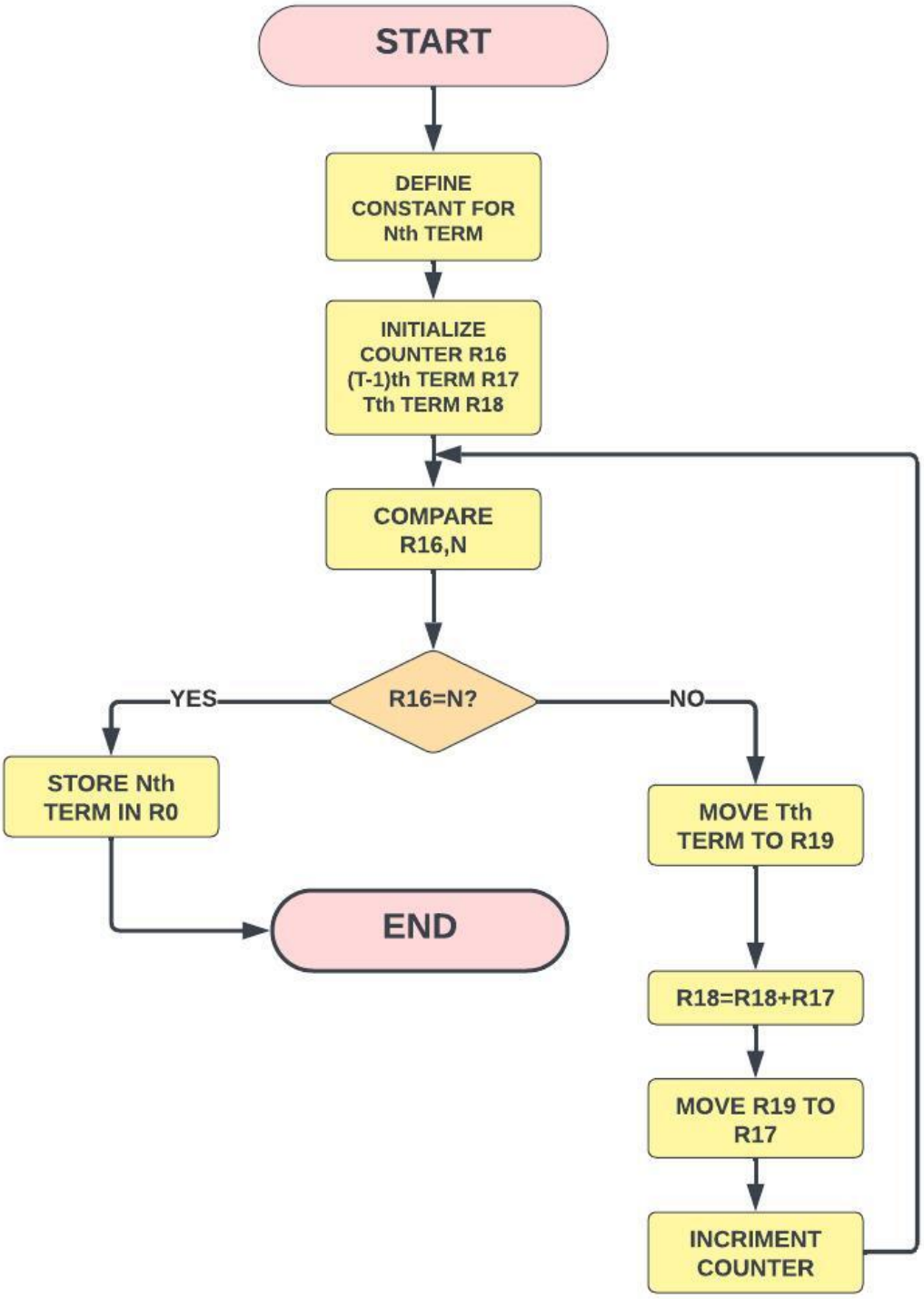


Figure 8: Fibonacci

2 Individual Contribution

The Experiment was done by both of the team members.
Implimentation of codes of addition,multiplication,Division and their results,flow charts,observations,learning outcomes were done by Maadhav Patel[EE22B033].
Implimentation of codes of Parity,Largest and smallest number,Fibonacci and their results,flow charts,observations,learning outcomes were done by Shravya[EE22B032].
Lab Report was written together by both the team members.