Aditi Tapariya
U21EE081

MICROCONTROLLER AND MICROPROCESSOR LAB

EXPERIMENT 1

AIM: Assembly language program for arithmetic operations of signed and unsigned numbers.

SOFTWARE USED: Keil uVision5

Question-1:

Case-1: Write an assembly language problem for adding 2, 8-bit numbers (unsigned) stored in the internal memory for 30h and 31h. Store the result in internal memory locations 40h and 41h.

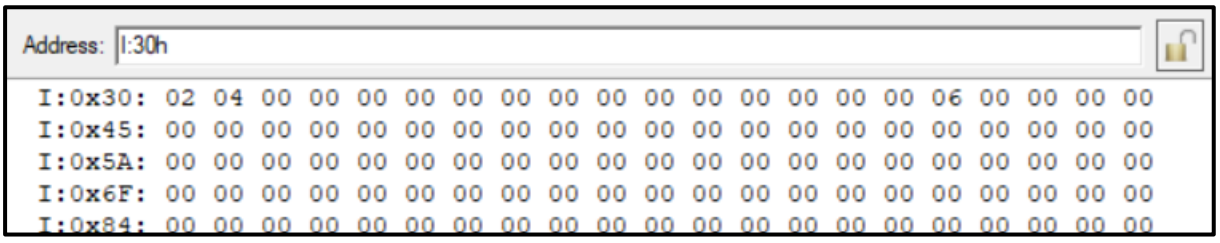
Code:

```
ORG 0000H
MOV A, 30H
MOV R0,31H
ADD A, R0
MOV 40H, A
JNC LOC1
MOV 41H, #01H
LOC1:
END
```

Algorithm:

1. Set the origin address to 0000H.
2. Load the content of memory location 30H into register A.
3. Load the content of memory location 31H into register R0.
4. Add the contents of registers A and R0. The result is stored in register A.
5. Store the lower byte of the result (register A) in memory location 40H.
6. Jump to the label Loc1 if there is no carry (JNC).
7. If Carry is Present set the higher byte (memory location 41H) to 01H.
8. This label marks the end of the code.
9. End the program.

Result:



Conclusion:

The assembly code successfully adds two 8-bit unsigned numbers stored in memory and handles carry if present, ensuring accurate results. It demonstrates fundamental arithmetic operations and conditional branching in assembly language programming.

Case-2: Write an assembly language problem for adding 2, 8-bit signed numbers stored in the internal memory for 30h and 31h. The data in 30h is positive and 31h is negative. The magnitude of data stored in 30h is greater than 31h. Store the result in internal memory locations 40h and 41h.

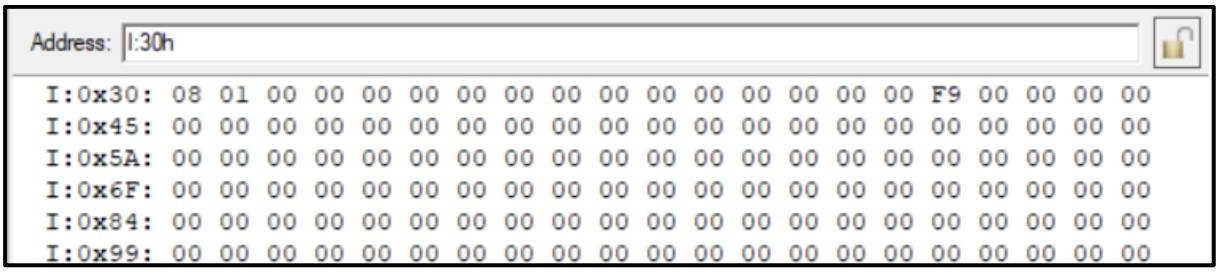
Code:

```
ORG 0000H
MOV A,30H
MOV R1, 31H
CPL A
ADD A, #01H
ADD A, R1
MOV 40H, A
END
```

Algorithm:

1. Set the origin address to 0000H.
2. Load content of memory location 30H (positive number) into register A.
3. Load content of memory location 31H (negative number) into register R1.
4. Complement the bits in register A. Add 1 to the complemented value in register A. This essentially performs a two's complement operation, converting the positive number in register A to its negative equivalent.
5. Add the content of register R1 (negative number) to the result in the register.
6. Store the result (sum of the two numbers) in memory location 40H.
7. End of the program.

Result:



Conclusion:

The assembly code successfully adds two 8-bit signed numbers, handling positive and negative values using two's complement arithmetic, ensuring proper addition and storage of the result.

Case-3: Write an assembly language problem for adding 2, 8-bit signed numbers stored in the internal memory for 30h and 31h. The data in 30h is positive and 31h is negative. The magnitude of data stored in 30h is less than 31h. Store the result in internal memory locations 40h and 41h.

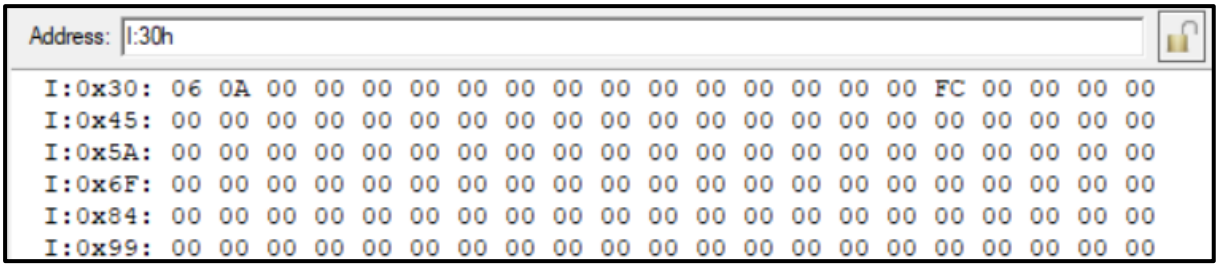
Code:

```
ORG 0000H
MOV A,30H
MOV R1, 31H
CPL A
ADD A, #01H
```

```
ADD A, R1
CPL A
ADD A, #01H
MOV 40H, A
END
```

Algorithm:
1. Set the origin address to 0000H.
2. Load the value at memory address 30H into register A, and the value at memory address 31H into register R1.
3. Complement the bits in register A. Add 1 to the value in register A.
4. Add the value in register R1 to the value in register A.
5. Complement the bits in register A. Add 1 to the value in register A.
6. Move the value in register A to memory address 40H.
7. End of the program.

Result:



Conclusion:
The assembly code accurately adds two 8-bit signed numbers, considering positive and negative values with proper handling of magnitudes, ensuring correct addition and storage of the result.

Case-4: Write an assembly language problem for adding 2, 8-bit signed numbers stored in the internal memory for 30h and 31h. The data stored in both 30h and 31h is negative. Store the result in internal memory locations 40h and 41h.

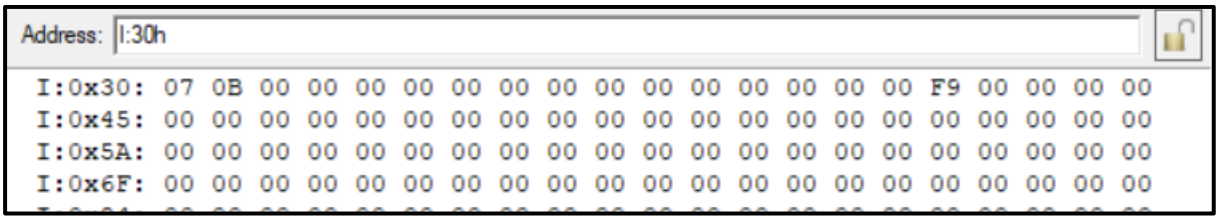
Code:

```
ORG 0000H
MOV A,30H
MOV R1, 31H
CPL A
ADD A, #01H
MOV R2, A
MOV A, R1
CPL A
ADD A, #01H
ADD A, R2
MOV 40H, A
END
```

Algorithm:
1. Set the origin address to 0000H.

2. Load the value at memory address 30H into register A, and the value at memory address 31H into register R1.
3. Complement the bits in register A. Add 1 to the value in register A.
4. Move the content of register A to register R2.
5. Load the original value of R1 into register A.
6. Complement (invert) the bits in register A. Add 1 to the complemented value in register A.
7. Add the original value of R1 to the complemented value in A.
8. Move the content of register A to memory address 40H.
9. End of the program.

Result:



Conclusion:
The assembly code accurately adds two 8-bit signed negative numbers, handling two's complement arithmetic properly to ensure correct addition and storage of the result in memory addresses 40H and 41H.

Question-2:

Case-1: Write an assembly-level program to add 2 n-byte numbers. Take n=10. Store the first no at external RAM, at location 2500h onwards, and the second no at location 2600h and store the result of addition at 2500h.

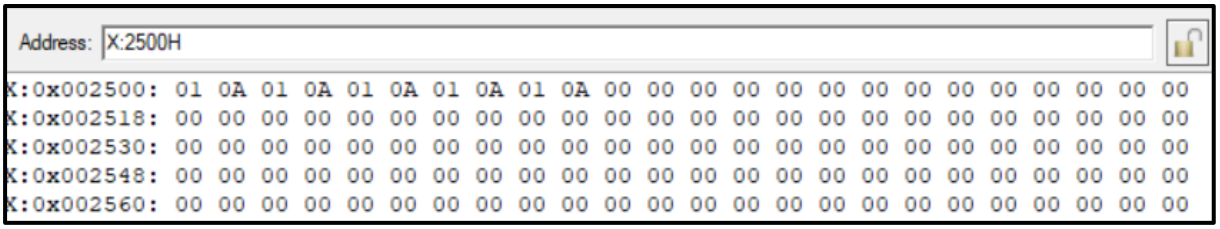
Code:

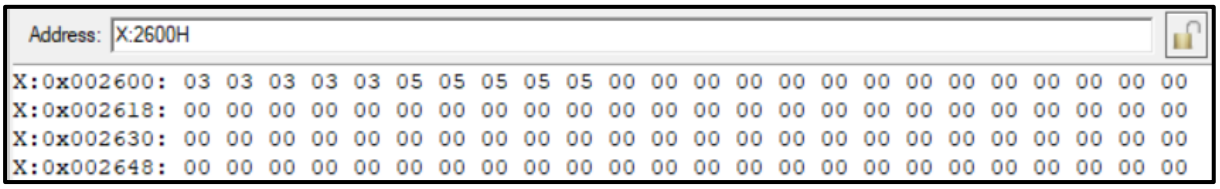
```
ORG 0000H
MOV DPTR, #2500H
CLR C
MOV R2, #0AH
DO: MOVX A, @DPTR
MOV R3, A
INC DPH
MOVX A, @DPTR
ADD A, R3
DEC DPH
MOVX @DPTR, A
INC DPL
DJNZ R2, DO
JNC STOP
MOV A, #01H
MOVX @DPTR, A
STOP:
END
```

Algorithm:

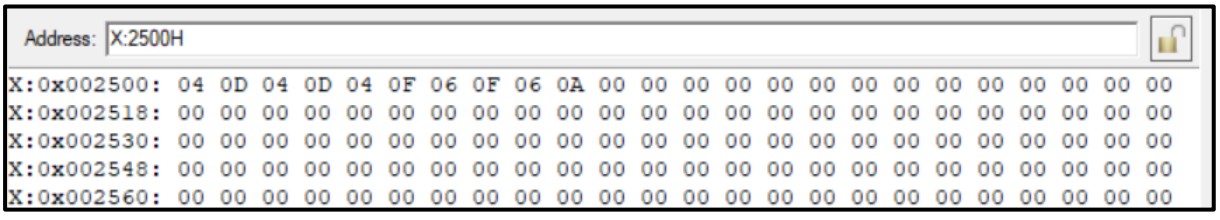
1. Set the origin address to 0000H.
2. Set the data pointer to memory address 2500H.
3. Clear the carry flag.
4. Initialize register R2 with the value 0AH.
5. Move the value at the memory location addressed by DPTR to register A.
6. Copy the value from A to R3.
7. Increment the high byte of the data pointer (DPH).
8. Move the value at the updated memory location addressed by DPTR to A.
9. Add the value in R3 to A with the carry.
10. Decrement the high byte of the data pointer (DPH).
11. Move the result back to the memory location addressed by DPTR.
12. Increment the low byte of the data pointer (DPL).
13. Decrement R2 and jump to DO if it's not zero.
14. Jump to D01 if the carry flag is not set (no carry).
15. Move the immediate value 01H to register A.
16. Move the value in A to the memory location addressed by DPTR.
17. End the loop (DO1).
18. End the program.

Result:
INPUT:





OUTPUT:



Conclusion:
The assembly code efficiently adds two n-byte numbers stored in external RAM, handling carry and ensuring correct addition, then stores the result at the specified memory location, demonstrating effective memory manipulation techniques.

Case-2: Write an assembly-level program to add 2 n-byte numbers. Take n=10. Store the first no at code memory, at location 2500h onwards, and the second no also at code memory at location 2600h, and store the result of addition at 2500h in external memory.

Code:

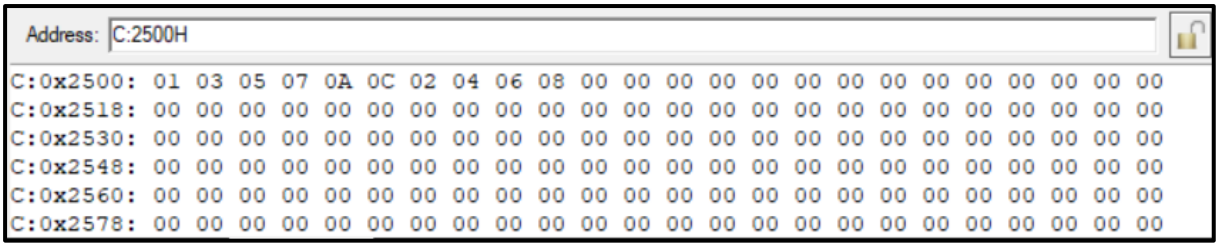
```
ORG 0000H
MOV DPTR, #2500H
CLR C
MOV R2, #0AH
DO: MOV A, #00H
MOVC A, @A+DPTR
MOV R3, A
INC DPH
MOV A, #00H
MOVC A, @A+DPTR
ADD A, R3
DEC DPH
MOVX @DPTR, A
INC DPL
DJNZ R2, DO
JNC STOP
MOV A, #01H
MOVX @DPTR, A
STOP:
END
```

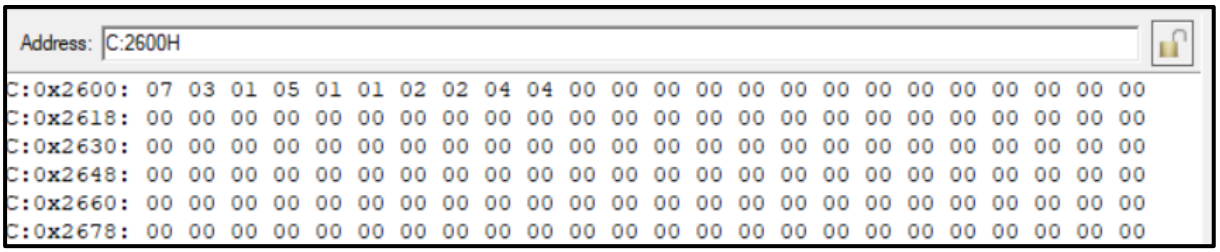
Algorithm:

1. Set the origin address to 0000H.
2. Set the data pointer to memory address 2500H.
3. Clear the carry flag.
4. Initialize register R2 with the value 0AH.
5. Start a loop (DO).
6. Move the immediate value 00H to register A.
7. Move the value from the memory location addressed by (A + DPTR) to A using the code memory as the source.
8. Copy the value from A to R3.
9. Increment the high byte of the data pointer (DPH).
10. Move the value from the updated memory location addressed by (A + DPTR) to A.
11. Add the value in R3 to A with carry.
12. Decrement the high byte of the data pointer (DPH).
13. Move the result back to the memory location addressed by (A + DPTR).
14. Increment the low byte of the data pointer (DPL).
15. Decrement R2 and jump to DO if it's not zero.
16. Jump to D01 if the carry flag is not set (no carry).
17. Move the immediate value 01H to register A.
18. Move the value in A to the memory location addressed by (A + DPTR).
19. End the loop (DO1).
20. End the program.

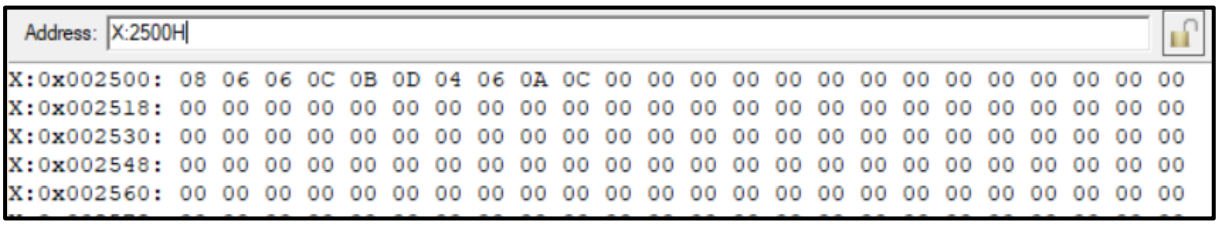
Result:

INPUT:





OUTPUT:



Conclusion:
The assembly code effectively adds two n-byte numbers stored in code memory, handling carry and ensuring accurate addition, then stores the result in external memory, showcasing efficient memory access and manipulation techniques.

Question-3:
Write an assembly-level program to add each byte of a 10-digit number stored in external RAM starting at address 2500h. The result should be stored at address 2600h.

Code:

```
ORG 0000H
MOV DPTR, #2500H
MOVX A, @DPTR
MOV R0, A
MOV R1, #00H
MOV R2, #09H
FIND: INC DPL
MOVX A, @DPTR
ADD A, R0
MOV R0, A
JNC NEXT
INC R1
NEXT: DJNZ R2, FIND
MOV DPTR, #2600H
MOV A, R0
MOVX @DPTR, A
INC DPL
```

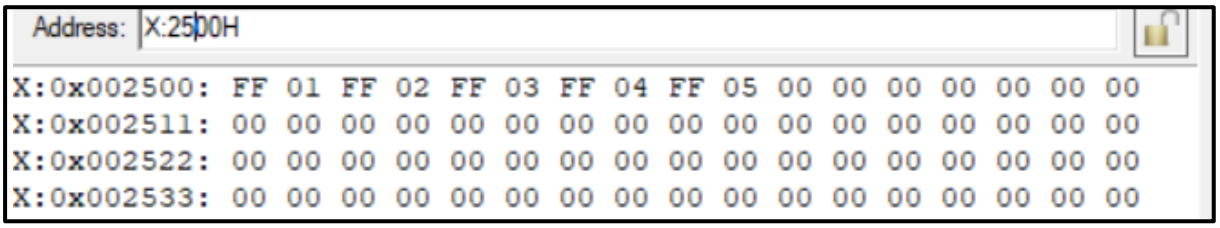
```
MOV A, R1
MOVX @DPTR, A
END
```

Algorithm:

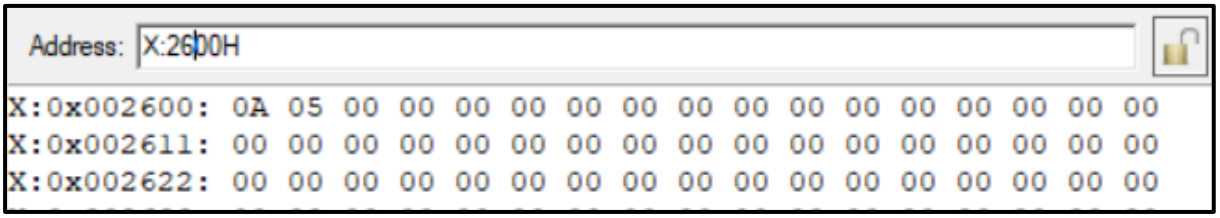
1. Set the origin of the program counter to 0000H.
2. Load the data pointer (DPTR) with the address 2500H.
3. Move the data at the address pointed to by DPTR to the accumulator (A).
4. Move the content of the accumulator (A) to register R0.
5. Initialize register R1 with 00H.
6. Initialize register R2 with 09H.
7. Start a loop labeled as FIND:
   a. Increment the low byte of DPTR (DPL).
   b. Move the data at the address pointed to by DPTR to the accumulator (A).
   c. Add the content of register R0 to the accumulator (A).
   d. Move the result back to register R0.
   e. Check if there is no carry (JNC) from the addition, if so, jump to the label NEXT.
   f. If there is a carry, increment register R1.
   g. Label NEXT and decrement register R2.
   h. Repeat the loop FIND until register R2 becomes zero.
8. Once the loop ends, move the data pointer (DPTR) to the address 2600H.
9. Move the content of register R0 to the accumulator (A).
10. Move the content of the accumulator (A) to the address pointed to by DPTR.
11. Increment the low byte of DPTR (DPL).
12. Move the content of register R1 to the accumulator (A).
13. Move the content of the accumulator (A) to the address pointed to by DPTR.
14. End the program.

Result:

INPUT:



OUTPUT:



Conclusion:

The assembly code effectively sums each byte of a 10-digit number stored in external RAM, storing the result at a specified address, and showcasing efficient memory access and arithmetic operations.