MICROPROCESSORS &MICROCONTROLLERS

NAME:D.SAJID ALI

ROLL NO:123EC0008

Assignment-5: 4-taps FIR filter implementation in 8086 assembly program

Implement the 4-taps FIR filter using 8086 emulator. The four co-efficients (or taps) are h[0], h[1], h[2], and h[3]. The input samples are x[0], x[1], x[2], x[3], and x[4]. The output samples are y[0], y[1], y[2], y[3], and y[4]. The FIR filter equation is as follows

y[n]=∑k=4k=0h[k].x[n−k]∑k=0k=4h[k].x[n−k], where n is varied from 0 to 4

Here, the 8-bit input samples are stored at the consecutive memory locations starting from 2000h. The 8-bit filter co-efficients (or taps) are stored at the consecutive memory locations starting from 2100h. The 16-bit output samples should stored at the consecutive memory locations starting at 2200h. The output samples are computed as follows.

y[0]=h[0].x[0]

y[1]=h[0].x[1]+h[1].x[0]

y[2]=h[0].x[2]+h[1].x[1]+h[2].x[0]

y[3]=h[0].x[3]+h[1].x[2]+h[2].x[1]+h[3].x[0]

y[4]=h[0].x[4]+h[1].x[3]+h[2].x[2]+h[3].x[1]

Write 8086 assembly program to implement the above equations. Prepare a .pdf by including this question, code, and screenshots of the input samples/filter taps/output samples at the memory.

Assembly Code:

```
org 100h
```

```
.MODEL SMALL
.STACK 100h
.DATA
    PROMPT_INPUT DB 'Enter 5 input samples x[0] to x[4] (8-bit values 0-255):$'
    PROMPT_COEFF DB 'Enter 4 filter coefficients h[0] to h[3] (8-bit values 0-
255):$'
    PROMPT_SAMPLE DB 'Enter x[$'
    PROMPT_COEFF_SAMPLE DB 'Enter h[$'
    NEWLINE DB 0DH, 0AH, '$'
    PROMPT_CLOSE DB ']:$'
    PROMPT_OUTPUT DB 'Output samples y[0] to y[4]:$'
    PROMPT_OUTPUT_SAMPLE DB 'y[$'
    PROMPT_EQUALS DB '] = $'
.CODE
MAIN PROC
    MOV AX, @DATA
    MOV DS, AX
    MOV AH, 00H
    MOV AL, 03H
    INT 10H
    MOV AH, 09H
    LEA DX, PROMPT_INPUT
    INT 21H
    MOV AH, 09H
    LEA DX, NEWLINE
    INT 21H
    MOV SI, 2000H
```

```
    MOV CX, 5
INPUT_SAMPLES:
    PUSH CX
    MOV AH, 09H
    LEA DX, PROMPT_SAMPLE
    INT 21H
    POP CX
    PUSH CX
    MOV AL, 5
    SUB AL, CL
    ADD AL, 30H
    MOV DL, AL
    MOV AH, 02H
    INT 21H
    MOV AH, 09H
    LEA DX, PROMPT_CLOSE
    INT 21H
    CALL READ_NUMBER
    MOV [SI], AL
    INC SI
    MOV AH, 09H
    LEA DX, NEWLINE
    INT 21H
    POP CX
    LOOP INPUT_SAMPLES
    MOV AH, 09H
```

```
    LEA DX, PROMPT_COEFF
    INT 21H
    MOV AH, 09H
    LEA DX, NEWLINE
    INT 21H
    MOV SI, 2100H
    MOV CX, 4
INPUT_COEFFS:
    PUSH CX
    MOV AH, 09H
    LEA DX, PROMPT_COEFF_SAMPLE
    INT 21H
    POP CX
    PUSH CX
    MOV AL, 4
    SUB AL, CL
    ADD AL, 30H
    MOV DL, AL
    MOV AH, 02H
    INT 21H
    MOV AH, 09H
    LEA DX, PROMPT_CLOSE
    INT 21H
    CALL READ_NUMBER
    MOV [SI], AL
    INC SI
```

```
    MOV AH, 09H
    LEA DX, NEWLINE
    INT 21H
    POP CX
    LOOP INPUT_COEFFS
    MOV DI, 2200H
    MOV CX, 5
    MOV AX, 0
CLEAR_OUTPUT:
    MOV [DI], AX
    ADD DI, 2
    LOOP CLEAR_OUTPUT
    MOV AL, [2000H]
    MOV BL, [2100H]
    MUL BL
    MOV [2200H], AX
    MOV AX, 0
    MOV AL, [2000H]
    MOV BL, [2101H]
    MUL BL
    ADD [2202H], AX
    MOV AL, [2001H]
    MOV BL, [2100H]
    MUL BL
    ADD [2202H], AX
    MOV AX, 0
```

```
MOV AL, [2000H]
MOV BL, [2102H]
MUL BL
ADD [2204H], AX
MOV AL, [2001H]
MOV BL, [2101H]
MUL BL
ADD [2204H], AX
MOV AL, [2002H]
MOV BL, [2100H]
MUL BL
ADD [2204H], AX
MOV AX, 0
MOV AL, [2000H]
MOV BL, [2103H]
MUL BL
ADD [2206H], AX
MOV AL, [2001H]
MOV BL, [2102H]
MUL BL
ADD [2206H], AX
MOV AL, [2002H]
MOV BL, [2101H]
MUL BL
ADD [2206H], AX
MOV AL, [2003H]
```

```
MOV BL, [2100H]
MUL BL
ADD [2206H], AX
MOV AX, 0
MOV AL, [2001H]
MOV BL, [2103H]
MUL BL
ADD [2208H], AX
MOV AL, [2002H]
MOV BL, [2102H]
MUL BL
ADD [2208H], AX
MOV AL, [2003H]
MOV BL, [2101H]
MUL BL
ADD [2208H], AX
MOV AL, [2004H]
MOV BL, [2100H]
MUL BL
ADD [2208H], AX
MOV AH, 09H
LEA DX, PROMPT_OUTPUT
INT 21H
MOV AH, 09H
LEA DX, NEWLINE
INT 21H
```

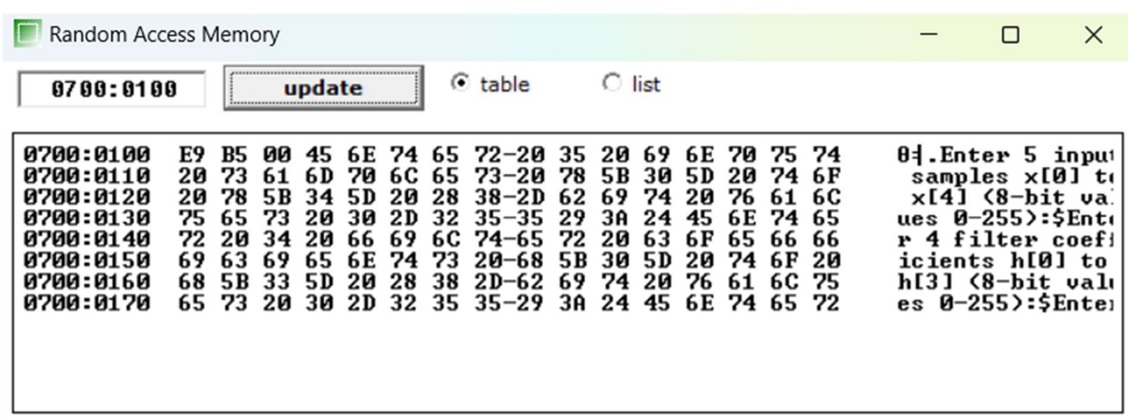
```
    MOV SI, 2200h
    MOV CX, 5
DISPLAY_OUTPUTS:
    PUSH CX
    MOV AH, 09H
    LEA DX, PROMPT_OUTPUT_SAMPLE
    INT 21H
    POP CX
    PUSH CX
    MOV AL, 5
    SUB AL, CL
    ADD AL, 30H
    MOV DL, AL
    MOV AH, 02H
    INT 21H
    MOV AH, 09H
    LEA DX, PROMPT_EQUALS
    INT 21H
    MOV AX, [SI]
    CALL DISPLAY_NUMBER
    MOV AH, 09H
    LEA DX, NEWLINE
    INT 21H
    ADD SI, 2
    POP CX
    LOOP DISPLAY_OUTPUTS
```

```
    MOV AH, 4Ch
    INT 21h
MAIN ENDP
READ_NUMBER PROC
    MOV AH, 01H
    MOV DL, 0
    INT 21H
    SUB AL, 30H
    MOV BL, AL
    MOV AH, 01H
    INT 21H
    CMP AL, 0DH
    JE SINGLE_DIGIT
    SUB AL, 30H
    MOV CL, 10
    MUL CL
    ADD AL, BL
    RET
SINGLE_DIGIT:
    MOV AL, BL
    RET
READ_NUMBER ENDP
DISPLAY_NUMBER PROC
    MOV CX, 0
    MOV BX, 10
CONVERT_LOOP:
```

```
    MOV DX, 0
    DIV BX
    PUSH DX
    INC CX
    CMP AX, 0
    JNE CONVERT_LOOP
DISPLAY_LOOP:
    POP DX
    ADD DL, 30H
    MOV AH, 02H
    INT 21H
    LOOP DISPLAY_LOOP
    RET
DISPLAY_NUMBER ENDP
END MAIN
```

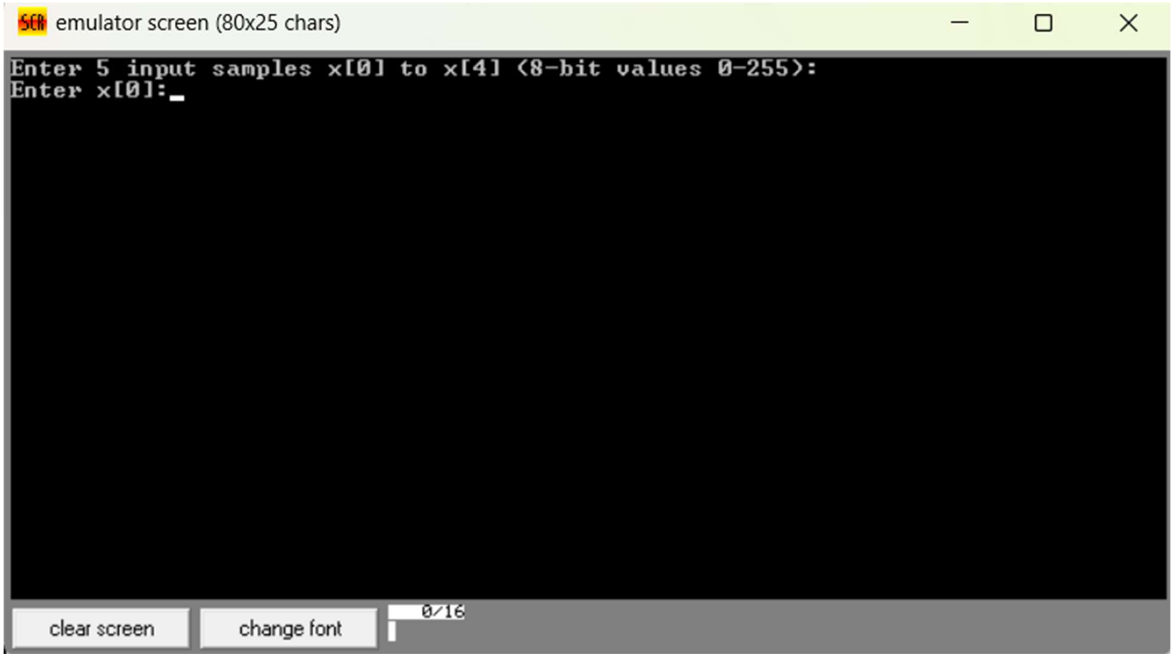
Screenshots:

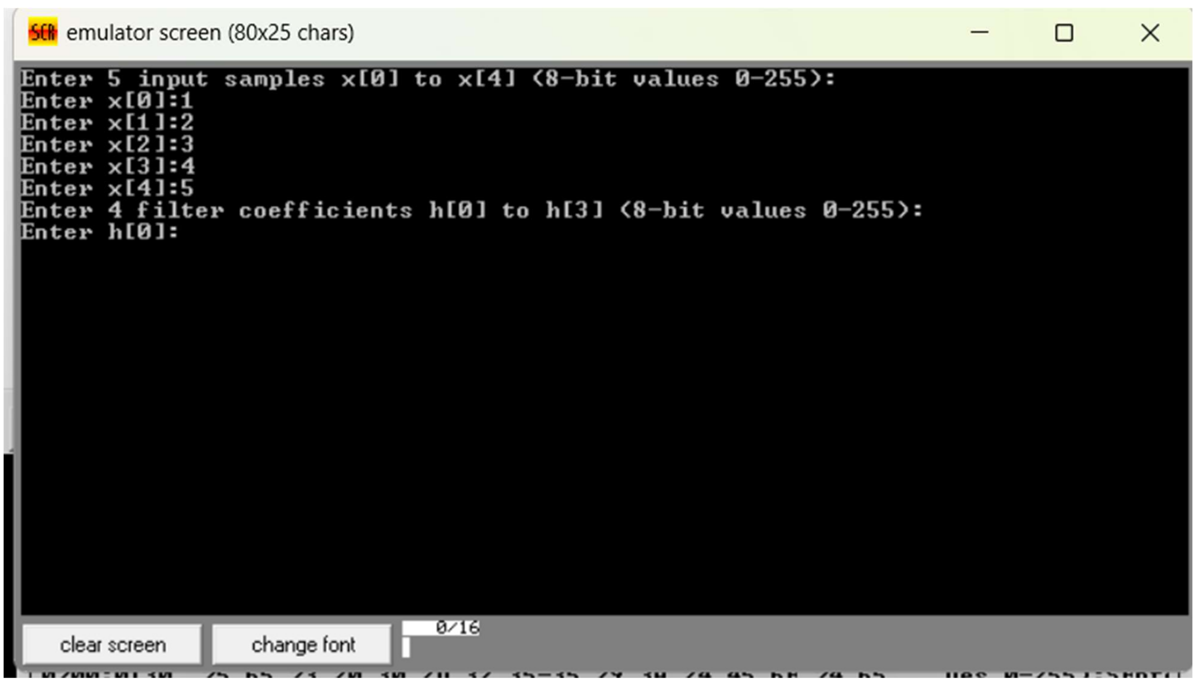
Before Execution:

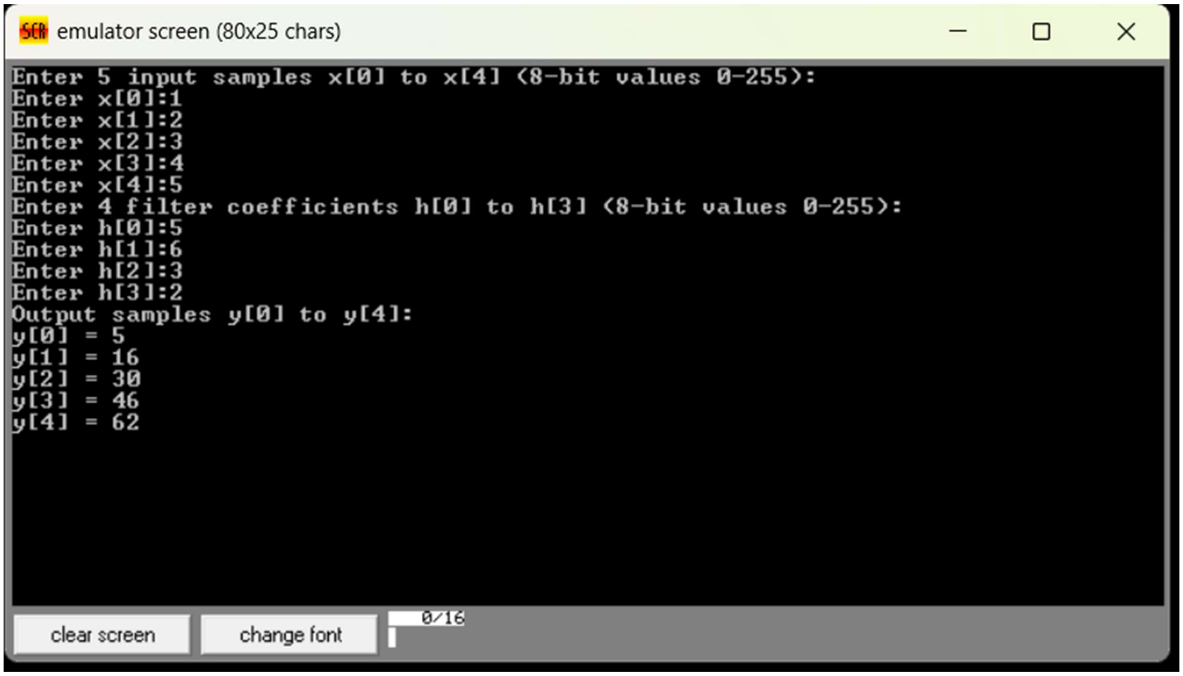


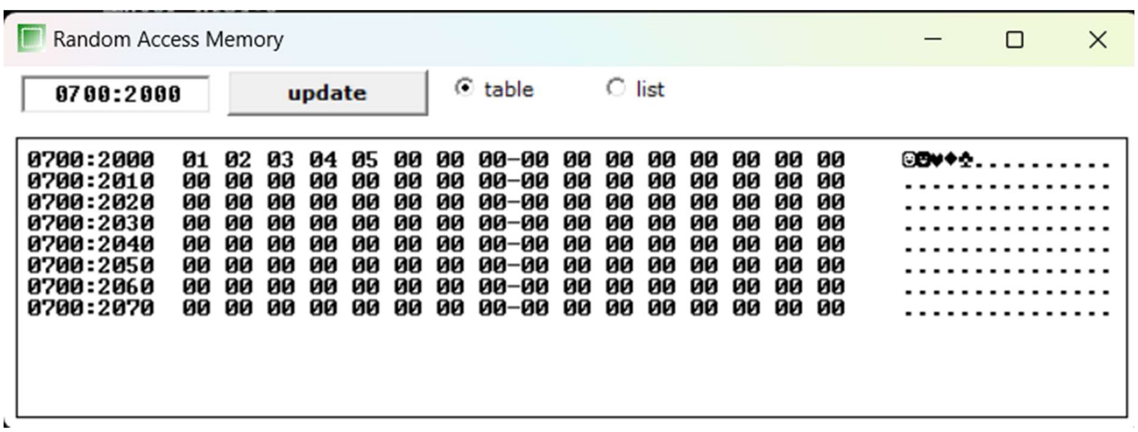
After Execution:

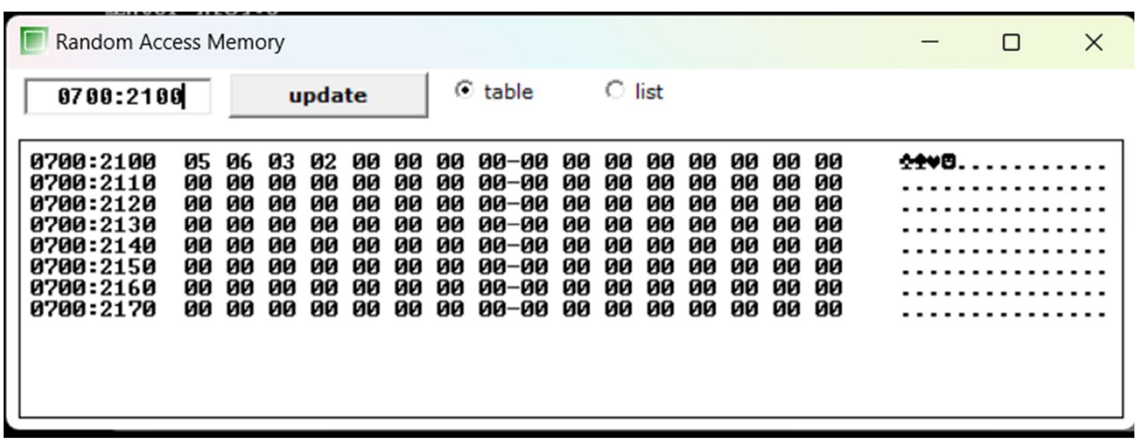
AT MEMORY:

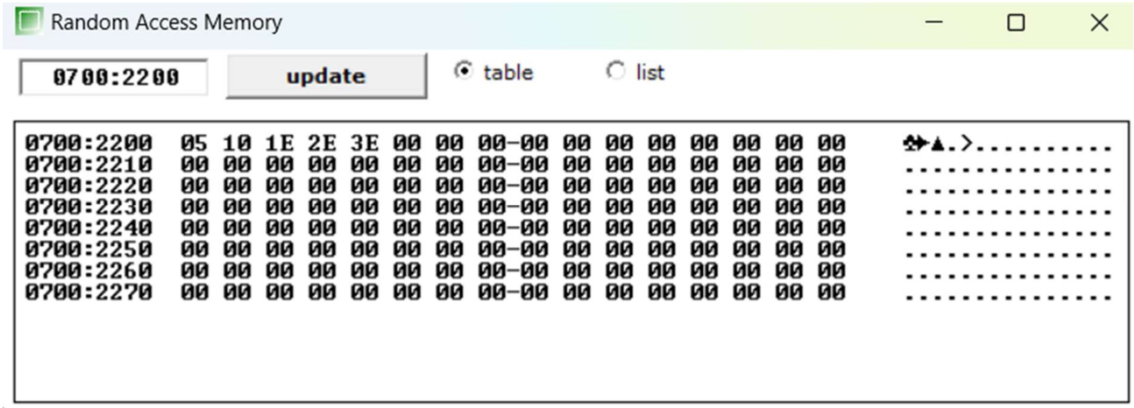

emulator screen (80x25 chars)
Enter 5 input samples x[0] to x[4] (8-bit values 0-255):
Enter x[0]:_
clear screen
change font
0/16


emulator screen (80x25 chars)
Enter 5 input samples x[0] to x[4] (8-bit values 0-255):
Enter x[0]:1
Enter x[1]:2
Enter x[2]:3
Enter x[3]:4
Enter x[4]:5
Enter 4 filter coefficients h[0] to h[3] (8-bit values 0-255):
Enter h[0]:
clear screen
change font
0/16


emulator screen (80x25 chars)
Enter 5 input samples x[0] to x[4] (8-bit values 0-255):
Enter x[0]:1
Enter x[1]:2
Enter x[2]:3
Enter x[3]:4
Enter x[4]:5
Enter 4 filter coefficients h[0] to h[3] (8-bit values 0-255):
Enter h[0]:5
Enter h[1]:6
Enter h[2]:3
Enter h[3]:2
Output samples y[0] to y[4]:
y[0] = 5
y[1] = 16
y[2] = 30
y[3] = 46
y[4] = 62
clear screen
change font
0/16


Random Access Memory
0700:2000
update
table
list
0700:2000 01 02 03 04 05 00 00 00-00 00 00 00 00 00 00 00
0700:2010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:2020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:2030 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:2040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:2050 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:2060 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:2070 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00


Random Access Memory
0700:2100
update
table
list
0700:2100 05 06 03 02 00 00 00 00-00 00 00 00 00 00 00 00
0700:2110 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:2120 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:2130 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:2140 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:2150 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:2160 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0700:2170 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00



RESULT:

1) Input Samples Storage:

- Location: 2000h memory address
- Consecutive memory locations for x[0] to x[4]
- Each sample is an 8-bit value
- Actual inputs: x[0]=1, x[1]=2, x[2]=3, x[3]=4, x[4]=5

2) Filter Coefficients Storage:

- Location: 2100h memory address
- Consecutive memory locations for h[0] to h[3]
- Each coefficient is an 8-bit value
- Actual coefficients:
  - h[0] = 5
  - h[1] = 6
  - h[2] = 3
  - h[3] = 2

3) Output Samples Storage:

- Location: 2200h memory address

- Consecutive memory locations for y[0] to y[4]
- y[0]=5=5H
- y[1]=16=10H
- y[2]=30=1EH
- y[3]=46=2EH
- y[4]=62=3EH
- Outputs are 16-bit to handle potential multiplication overflow

CALCULATION:

Inputs:

- *x[0] = 1*
- *x[1] = 2*
- *x[2] = 3*
- *x[3] = 4*
- *x[4] = 5*

Coefficients:

- *h[0] = 5*
- *h[1] = 6*
- *h[2] = 3*
- *h[3] = 2*

Calculating each output sample:

1. *y[0] = h[0] * x[0] y[0] = 5 * 1 = 5*

2. *y[1] = h[0] * x[1] + h[1] * x[0] y[1] = (5 * 2) + (6 * 1) = 10 + 6 = 16*
3. *y[2] = h[0] * x[2] + h[1] * x[1] + h[2] * x[0] y[2] = (5 * 3) + (6 * 2) + (3 * 1) = 15 + 12 + 3 = 30*
4. *y[3] = h[0] * x[3] + h[1] * x[2] + h[2] * x[1] + h[3] * x[0] y[3] = (5 * 4) + (6 * 3) + (3 * 2) + (2 * 1) = 20 + 18 + 6 + 2 = 46*
5. *y[4] = h[0] * x[4] + h[1] * x[3] + h[2] * x[2] + h[3] * x[1] y[4] = (5 * 5) + (6 * 4) + (3 * 3) + (2 * 2) = 25 + 24 + 9 + 4 = 62*

Final outputs:

- ***y[0] = 5***
- ***y[1] = 16***
- ***y[2] = 30***
- ***y[3] = 46***
- ***y[4] = 62***

Explanation:

- Each output is a weighted sum of input samples
- The weights are the filter coefficients h[0] to h[3]
- The most recent input samples have a larger impact due to their position in the calculation
- Changing coefficients directly changes the output values and the filter's characteristics

Conclusion

In this implementation, we designed a **4-tap FIR filter** using the **8086 emulator** to process 8-bit input samples with 8-bit filter coefficients. The computed **16-bit output samples** were stored in consecutive memory locations.