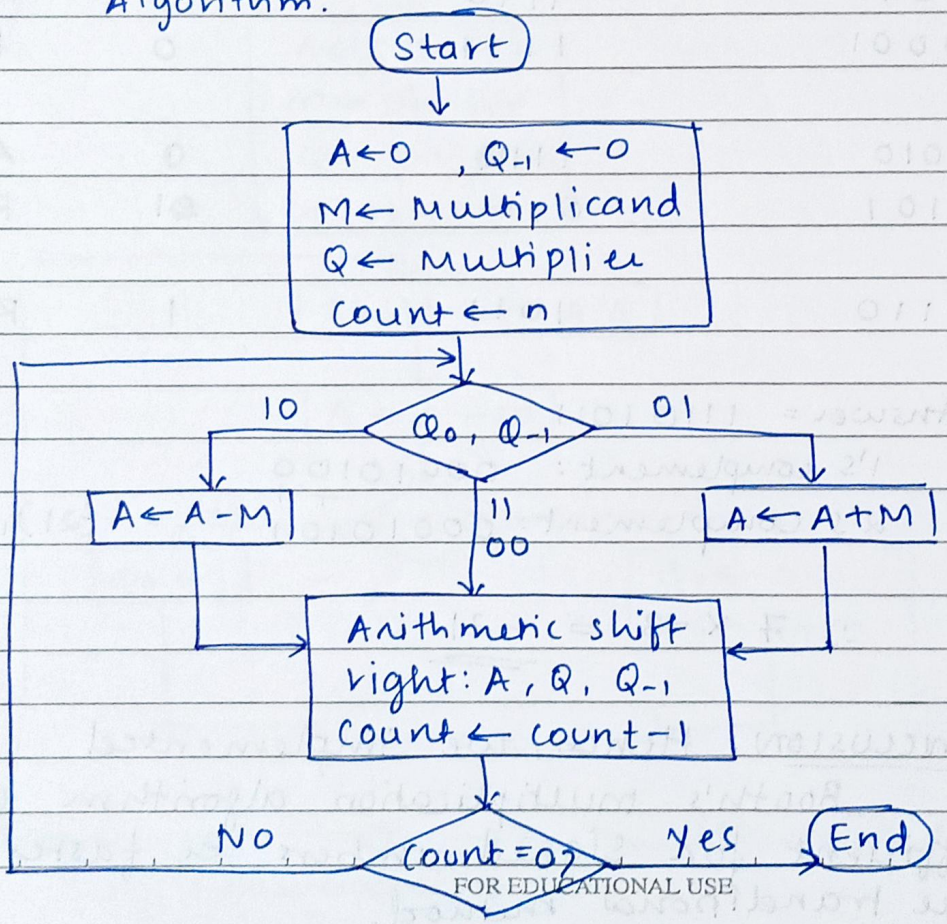
NAME: PRERNA S. JADHAV
SAP ID: 60004220127
BATCH: C2-2
POA EXPERIMENT 01 : BOOTH's ALGORITHM

AIM : To implement Booth's Multiplication Algorithm.

THEORY : Booth algorithm gives a procedure for multiplying binary integers in signed 2's complement representation in efficient way.

Algorithm:



Q] 7 x -3

M = 7 → 0111 $\quad -m = 1001$

Q = -3 → 0011 ⇒ 1's complement = 1100

2's complement = 1100 + 1 = 1101

∴ Q = -3 = 1101

count	A	Q	Q_{-1}	Operation
4	0000	1101	0	Initialize
	1001	1101	0	A ← A - m
3	1100	1110	1	Right shift
	0011	1110	1	A ← A + m
2	0001	1111	0	Right shift
	1010	1111	0	A ← A - M
1	1101	0111	1	Right shift
0	1110	1011	1	Right shift

Answer = 1110 1011

1's complement: 0001 0100

2's complement: 0001 0101 + 1 $= (21)_{10}$

∴ 7 x -3 = -21

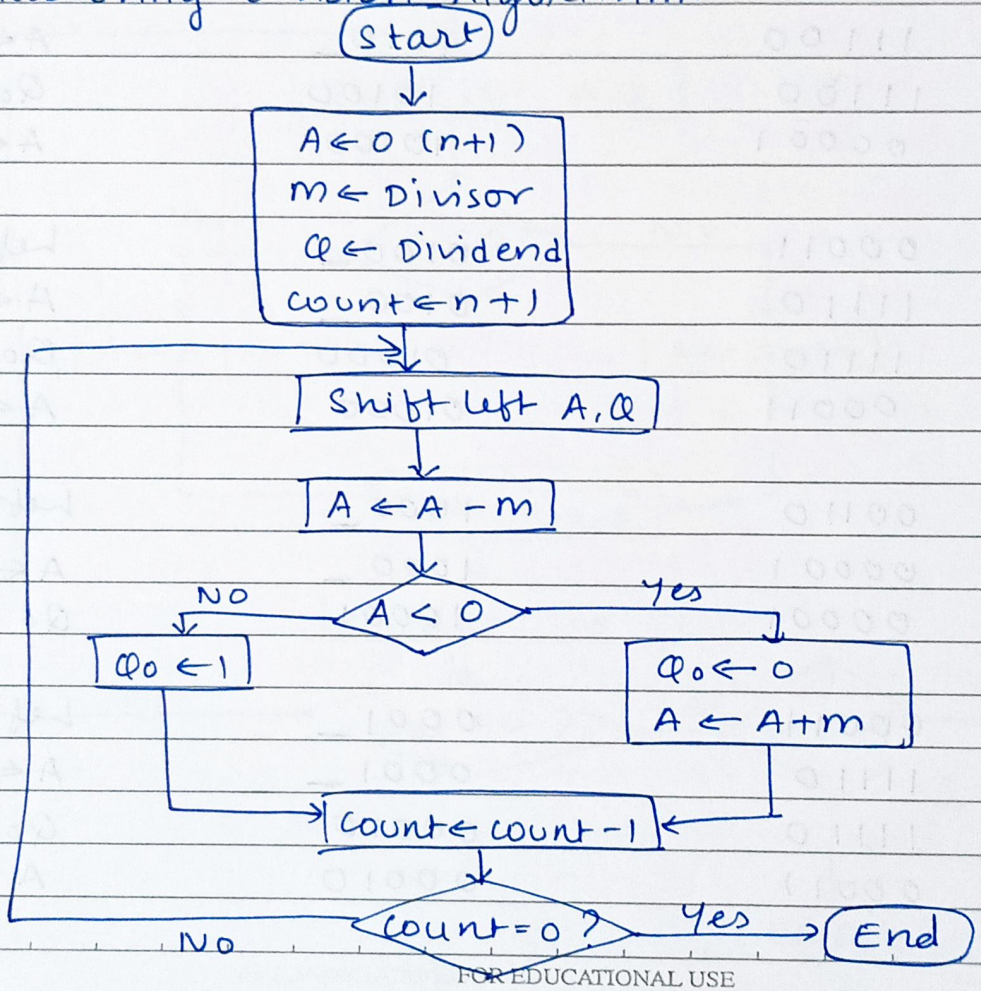
CONCLUSION: Hence, we implemented Booth's multiplication algorithm which is efficient for signed numbers & faster than the traditional method.

NAME: PRERNA S. JADHAV
SAP ID: 60004220127
BATCH: C2-2
POA EXPERIMENT 02: DIVISION ALGORITHM

AIM: To implement and study Restoring/ Non-restoring division algorithm.

THEORY :

i) Restoring division Algorithm:



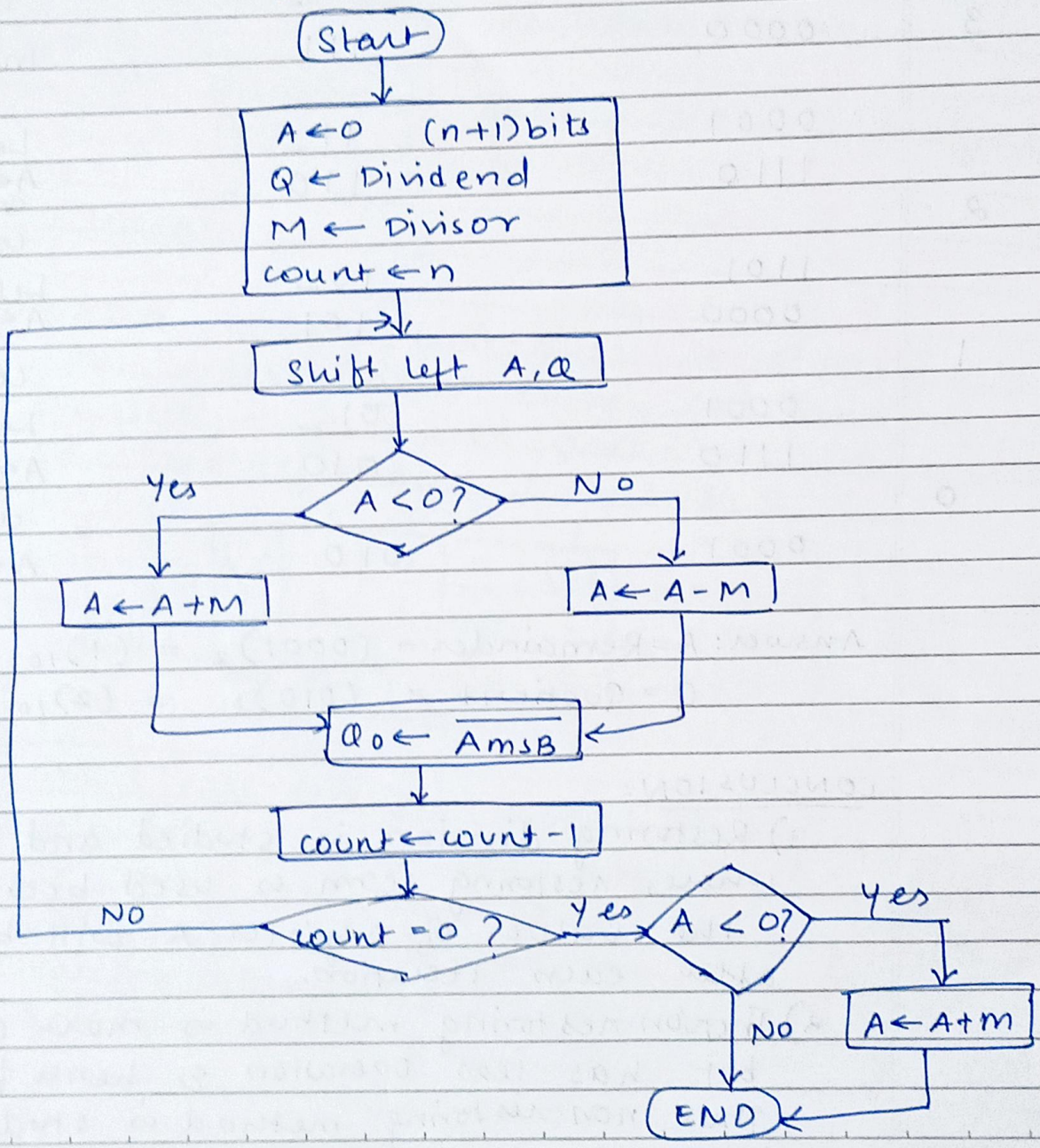
Q Divide: 13/5

$Q = (13)_{10} = (1101)_2$
$M = (5)_{10} = (0101)_2$
} 4 bits so use n+1 bits.

Count	A	Q	Operation
5	00000	01101	Initialization
	00000	1101_	Left shift
	11011	1101_	A ← A − M
	11011	11010	$Q_0 \Leftarrow 0$
4	00000	11010	A ← A + m
	00001	1010_	Left shift
	11100	1010_	A ← A − M
	11100	10100	$Q_0 \Leftarrow 0$
3	00001	10100	A ← A + m
	00011	0100_	Left shift
	11110	0100_	A ← A − M
	11110	01000	$Q_0 \Leftarrow 0$
2	00011	01000	A ← A + m
	00110	1000_	Left shift
	00001	1000_	A ← A − M
1	00001	10001	$Q_0 \Leftarrow 1$
	00011	0001_	Left shift
	11110	0001_	A ← A − M
	11110	00010	$Q_0 \Leftarrow 0$
0	00011	00010	A ← A + M

A → Remainder → $(00011)_2$ = $(3)_{10}$
Q → Quotient → $(00010)_2$ = $(2)_{10}$

ii) Non-Restoring division Algorithm:



Q Divide: $7 \div 3$

$Q = 7 = 111$

$M = 3 = 011 = 0011$

count	A	Q	Operation
3	0000	111	Initialization
	0001	11_	Left shift
	1110	110	$A \leftarrow A - M$, $Q \leftarrow 0$
2			count - 1
	1101	10_	Left shift
	0000	101	$A \leftarrow A + M$, $Q \leftarrow 1$
1			count - 1
	0001	01_	Left shift
	1110	010	$A \leftarrow A - M$, $Q \leftarrow 0$
0			count - 1
	0001	010	$A \leftarrow A + M$

Answer: A = Remainder = $(0001)_2 = (1)_{10}$

Q = Quotient = $(010)_2 = (2)_{10}$

CONCLUSION:

1) Restoring division is studied and implemented where, restoring term is used because the value of register A will be restored after each iteration.

2) The non-restoring method is more complex but has less operation & hence faster. Thus non-restoring method is studied & implemented.

NAME: PRERNA S. JADHAV
SAP ID: 60004220127
BATCH: C2-2
POA EXPERIMENT 03: PAGE REPLACEMENT

AIM: To implement Page replacement algorithm.
FIFO, LRU, OPTIMAL

THEORY:

1) FIFO (First-In First-Out): It states that the oldest (first) entry, or "head" of the queue is processed first just like queues.

Page Reference: 1, 3, 0, 3, 5, 6, 3

1	3	0	3	5	6	3
		(0)	0	0	0	(3)
	(3)	3	3	3	(6)	6
(1)	1	1	1	(5)	5	5
Miss	Miss	Miss	Hit	Miss	Miss	Miss

- No. of frames = 3
- No. of pages = 7
- No. of Misses = 6
- No. of Hits = 1

2) LRU (Least-Recently used): Its a greedy algorithm where the page to replaced is least recently used. So, the page not utilized for the longest time in the memory (compared to all other pages) gets replaced.

Page Reference: 1, 2, 1, 0, 3, 0, 4, 2, 4

1	2	1	0	3	0	4	2	4
			(0)	0	0	0	0	0
	(2)	2	2	(3)	3	3	(2)	2
(1)	1	1	1	1	1	(4)	4	4
Miss	Miss	Hit	Miss	Miss	Hit	Miss	Miss	Hit

- No. of frame: 3
- No. of pages: 9
- No. of misses: 6
- No. of hits: 3

3) OPTIMAL: It minimizes the no. of page faults by predicting future accesses & replacing the page that will not be needed for the longest period in the future.

Page Reference: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3

7	0	1	2	0	3	0	4	2	3
		(1)	1	1	(3)	3	3	3	3
	(0)	0	0	0	0	0	(4)	4	4
(7)	7	7	(2)	2	2	2	2	2	2
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit

- No. of frame: 3
- No. of pages: 10
- No. of misses: 6
- No. of Hits: 4

CONCLUSION: All the three algorithms are implemented

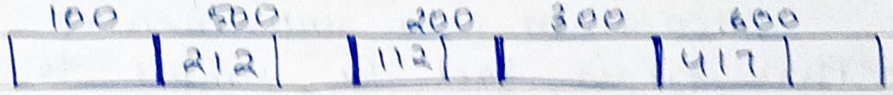
	FIFO	optimal	LRU
concept	oldest page removed	future page usage	Least recently used
Pros	Simple to implement	Provides the best theoretical performance	Effective in many practical cases
cons	Doesn't consider page usage	Impractical, as it needs future knowledge	Overhead to track usage for all pages

NAME: PRERNA S. JADHAV
SAP ID: 60004220127
BATCH: C2-2
POA EXPERIMENT 04: MEMORY ALLOCATION

AIM: To implement Memory Allocation (First Fit, Best Fit, Worst Fit)

THEORY:

1) FIRST FIT: The partition is allocated which is the first sufficient block from the top of Main memory. It scan memory from the beginning & chooses the first available block that is large enough. Thus it allocates the first hole that is large enough.
Eg: Consider memory blocks: 100, 500, 200, 300, 600 KB
Fit the process: P1 → 212, P2 → 417, P3 → 112, P4 → 426

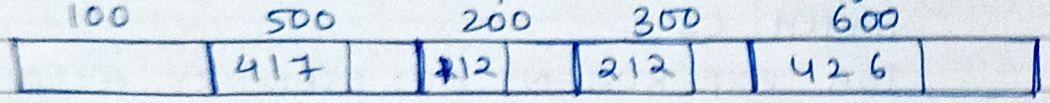


P4 → 426 is not able to fit in.

2) Best FIT: Allocate the process to the partition which is the first smallest suffiest partition among the free available partition. It searches the entire list of holes to find the smallest hole whose size is greater than or equal

to the size of the process.

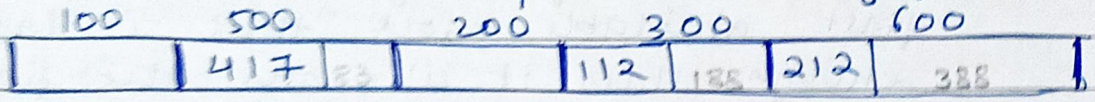
Eg: Consider same question as of first fit



we are able to fit in all the processes.

3) WORST FIT: Allocate the process to the partition which is largest sufficient among the freely available partitions available in main memory. It is opposite to the best fit algorithm. It searches the entire list of holes to find the largest hole & allocate it to process.

Eg: Consider same question as of first fit



we are not able to fit P4 in the memory

CONCLUSION: All the memory allocation algorithm are implemented & studied.

→ First fit is fast & simple to implement but suffers external fragmentation

→ Best fit reduces external fragmentation but requires more time to search for the appropriate partition

→ Worst fit reduces external fragmentation by leaving largest free partition but can lead to inefficient use of memory.

NAME: PRERNA SUNIL JADHAV
SAPID: 60004220127
BATCH: C2-2
POA EXPERIMENT 05: 16-BIT ADDITION / SUBTRACTION

AIM: Assembly program for 16-bit addition/ subtraction using direct, Immediate & Register Addressing mode

THEORY:

1) DIRECT ADDRESSING MODE: The instruction specifies the memory address directly to access or manipulate the data. The value at the specified memory location is used in operation
→Example: Adding the value at memory location 0x1000 to the AX register
MOV AX, [0x1000]

2) IMMEDIATE ADDRESSING MODE: The instruction contains the actual data or constant value to be used directly in the operation. The value is not stored in a register or memory location; it is a part of the instruction itself
→Example: loading the value 42 into BX register
MOV BX, 42

3) REGISTER ADDRESSING MODE: The operation is performed using data stored in registers. The instruction specifies one or more registers as operands for the operation.

→Eg: Suppose AX has 1234h value & BX has 5678h value we can add those by following instruction set

```
MOV  AX, 1234h
MOV  BX, 5678h
ADD  CX, AX
ADD  CX, BX
```

CONCLUSION: The assembly program demonstrates 16-bit addition and subtraction using different addressing modes. It gives a practical example of how to perform these operations in x86 assembly language.

NAME: PRERNA S. JADHAV
SAP ID: 60004220127
BATCH: C2-2
POA EXPERIMENT 06: PROGRAM TO SORT NUMBERS

AIM: Assembly program to sort numbers in ascending and descending order.

THEORY: The algorithm flow of the program can be stated as:

Step 1:
- → Initialize variable
- → Create an array (data) with unsorted no.s
- → Define the no. of elements in array (N)
- → Define a temporary variable (temp)

Step 2:
- → Outer loop for sorting
- → Use an index (SI) to point to current element
- → Inside outer loop user another index (DI) to point to first element
- → Compare the current element with next element
- → Swap the values if its smaller than the current element
- → Continue the process for all elements in the array
- → Repeat the inner loop for each elements

Step 3:
→ Continue outer loop until all elements are in order

Step 4:
→ End the program

CONCLUSION: Hence we have implemented the sorting algorithm (Bubble sort) in assembly language

NAME : PRERNA S. JADHAV
SAP ID: 60004220127
BATCH : C2-2
POA EXPERIMENT 07 : TRANSFER N BLOCKS

AIM: Assembly program to transfer n block of data from one segment to another segment

THEORY:
To transfer n blocks of data from one segment to another segment in 8086, you can use a simple loop that iterates through each block & copies the data.
Step 1: set up the source & destination segment register with appropriate values.
Step 2: Initialize a counter (eg. CX) with value n to represent the no. of blocks to be transferred
Step 3: calculate the offset addresses for source & destination segment. Use an index to keep track of current block's offset
Step 4: Loop:
→ Load a word from source into a register
→ Store the value from register into the destination segment
→ Increment offset index for both source & destination segments to point to next block.

→ Decrease the counter (CX) by 1
→ Repeat steps until CX reaches zero
Step 6: End the loop & complete data transfer

CONCLUSION : Hence, we implemented the assembly language program in 8086 to transfer n blocks using the above algorithm.

NAME : PRERNA S. JADHAV
SAP ID: 60004220127
BATCH : C2-2
POA EXPERIMENT 08: MIN / MAX NUMBER

AIM: Assembly language program to find minimum / maximum number from a given array.

THEORY:

Step 1: Load the address of the array into a register (eg. SI) and initialise 2 register to store minimum & maximum values. Set the minimum value to large +ve no. and maximum value to a large -ve no.

Step 2: Initialize loop counter (eg. CX) to the no. of elements in the array.

Step 3: Loop:
a) Load the current element from array
b) Compare the current element with the value in minimum register. If it is smaller, update the minimum register
c) Compare the current element with the value in maximum register. If it is larger, update maximum register

Step 4: Decrement loop counter (eg. CX) & check if it has reached zero. If not repeat the loop

Step 5: After the loop is completed, the minimum & maximum values are stored in the respective registers.

CONCLUSION: Hence, we implemented the assembly language program in 8086 to find the minimum and maximum number in the array using the above algorithm.

NAME : PRERNA S. JADHAV
SAP ID : 60004220127
BATCH : C2-2
POA EXPERIMENT 09 : FACTORIAL OF A NUMBER

AIM: Assembly language program to find the factorial of a number without and with macros.

THEORY :

a) WITHOUT MACROS:
Step 1: Load the value of 'n' (eg. 5) into CX register
Step 2: Initialize the result (factorial) in the AX register to 1
Step 3: Create a loop labeled 'FactorialLoop':
a) Multiply AX by CX using MUL.
b) Decement CX
c) Repeat the loop until CX reaches 0.
Step 4: The result (factorial) is stored in AX.

b) WITH MACROS:
Step 1: Define a macro named 'CalculateFactorial' that takes 2 arguments, the number 'n'
Step 2: Load the value of 'n' into the CX register and initialize the result (factorial) in the AX register to 1.

Step 3: Create a loop labeled "factorialLoop":
a) Multiply AX by CX using MUL instruction
b) Decrement CX
c) Repeat loop until CX reaches 0

Step 4: Call the "calculateFactorial" macro with the number to calculate factorial for (e.g., 5).

Step 5: Result will be stored in AX register.

CONCLUSION: Both programs use the same algorithm to calculate the factorial but the second program uses a macro to encapsulate the calculation whereas the first program performs calculation without macros.
Macros allow code reusability & simplification of code, which can lead to efficient & maintainable programs.

NAME : PRERNA S. JADHAV
SAP ID: 60004220127
BATCH : C2-2
POA EXPERIMENT 10: DOS INTERRUPTS

AIM: Assembly language program to demonstrate few DOS Interrupts.

THEORY: DOS (Disk operating system) interrupts in 8086 refer to a set of software interrupts that allow programs running in DOS to interact with the operating system.
These interrupts provide a way for application to request various services from the DOS, such as file I/O, memory allocation, program termination.

1) int 21H : It provides a wide range of services, including file operations (eg: opening, reading, writing, closing, files), console I/O, program termination
2) int 10H : This interrupt allows for video services, such as changing video modes, setting cursor position, and writing characters to screen.
3) int 20H : This interrupt is used to request the termination of the program. Its equivalent to exit()

function in C language or the 'exit' system call in Unix-like operating system.

CONCLUSION: These DOS interrupts provide a way for programs various tasks & interact with the DOS. It makes easier to perform common tasks like file operations, memory management, and user interface interaction within a DOS environment.