Name: Manthan M. Sonawane
Panel: ~~I~~ E   Roll No: 17

Assignment - 7

Problem Statement: write X86/64 ALP to add an array of N hexadecimal.

Objectives:
1. Understand the concept of unpacked and packed HEX number and the need of packing the accepted number from user.
2. Repetitive addition.

Theory:

- Explain unpacked and packed number with example.

Packed Numbers: Packed Numbers are stored in two digits to a byte, in 4 bit groups referred to as nibbles. ALU is capable of performing only binary addition and substraction.

Unpacked Numbers: In unpacked numbers, there is only one digit per byte and because of this, unpacked multiplication and division can be done.

· Why packing is required.

1) While accepting an array of numbers from user, all numbers are stored in unpacked form and need to be packed for further arithmetic operations.

2) Packing is required for displaying a number taken from user or displaying addition result of the initialized 2-digit hex numbers.

· New instructions and system calls used.

i) Macro instruction - a request to assemble program to process a predefined sequence of instructions called a macro defination.

```
%macro rw,4
mov rax, %1
mov rdi, %2
mov rsi, %3
mov rdx, %4
syscall
%end macro
```

ii) Array times 10 db 0.

iii) mov rbp, array

iv) Mov rsi, num

Algorithm / Implementation:

Attached print outs.

Platform
- Editor - gedit, a GNU editor.
- Assembler - NASM (Netwide Assembler)
- Linker - LD, a GNU Linker

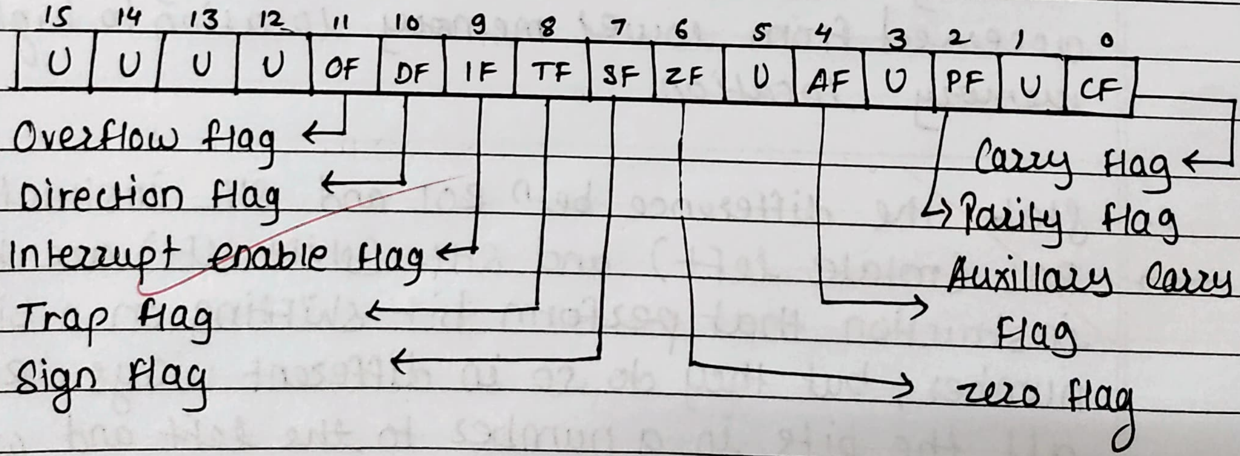
Input
Array elements

Output:
Sum of array elements

Conclusion: Hence, understood the concept of unpacking and packed HEX numbers and ALP to add an array of N Hexadecimal numbers.

FAQ's

1. Explain flag register of 8086 with neat diagram.



The flag register in 8086 microprocessor is a 16-bit register that stores various flags. We can divide flag bits into 2 sections:-

1. Status Flags:
   - Carry Flag (CF) - set if there is carry out of MSB of result
   - Parity Flag (PF) - set if no. of set bits is even
   - Auxillary carry flag (AF) - set if there is carry out of bit 3.
   - zero Flag (ZF) - set if result is zero
   - Sign flag (SF) - set is MSB of result is set.
   - Overflow flag (OF) - set if result of signed operation is too large to fit in destination register.

2. Control Flags:
   - Direction flag (DF) - If set, then string data is accessed from higher memory location to lower memory location, If reset (0), then string data is accessed from lower memory location to higher memory location.

2. State the difference betn ROL and SHL instructions.

→ ROL (rotate left) and SHL (shift left) are both instruction that perform bit shifting on a binary number, but they do so in different ways. ROL shifts all the bits in a number to the left and wraps around the bits that overflow on the left to the right side. whereas SHL simply shifts all the bits to the left, discarding the bits that overflow on the left.

3. Why 30H/37H is subtracted from the number?
Explain with example of each.

→ The subtraction of 30H or 37H from a number is typically done in order to convert the number from its hexadecimal representation to its ASCII representation.

For example, if the hexadecimal number is 30H, represents the decimal number 48. The ASCII code for the digit "0" is 48. Therefore, by subtracting 30H from the hexadecimal number, we can convert it to the ASCII representation of the digit "0".

```
section .data
array db 11H, 12H, 13h, 14h, 15h
msg db "Sum of array:", 10
msgLen equ $-msg

section .bss
sum resw 1
temp resw 2
temp1 resb 1

%macro rw 4
mov rax, %1
mov rdi, %2
mov rsi, %3
mov rdx, %4
syscall
%endmacro

section .text
global _start
_start:
mov rsi, array
mov ax, 0h
mov bx, 0h
mov cx, 5

up2:
mov bl, byte[rsi]
add ax, bx
jnc skip
inc ah

skip:
inc rsi
dec cx
jnz up2

mov word[sum], ax
rw 1, 1, msg, msgLen;

call disp

rw 60,0,0,0;
```

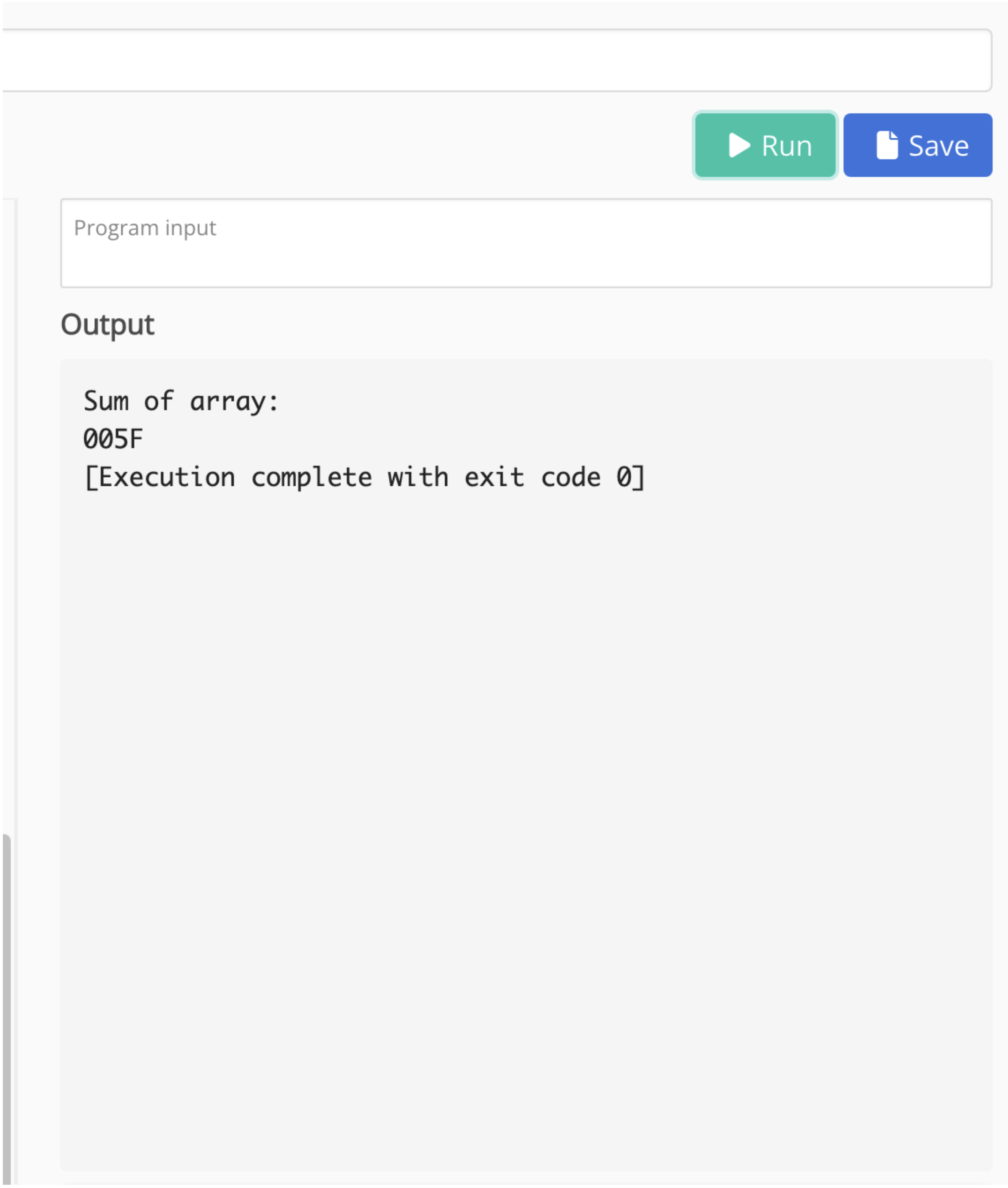
```
disp:
mov bp, 4
mov ax, word[sum]

up1:
rol ax, 4
mov [temp], ax
and ax, 0fh
cmp al, 09
jbe down1
add al, 07

down1:
add al, 30h
mov [temp1], al
rw 1, 1, temp1, 1;

mov ax, word[temp]
dec bp
jnz up1
ret
```

OUTPUT


Run
Save
Program input
Output
Sum of array:
005F
[Execution complete with exit code 0]