|  |  |
| --- | --- |
| **EXPERIMENT NO. 5** | |
| **Title** | Develop a 8086 Assembly Language Program for Conversions– (a) Packed to Unpacked BCD (b) Unpacked to Packed BCD (c) Packed BCD to ASCII |
| **Name** | Deep Salunkhe |
| **Roll No** | 21102A0014 |
| **Outcome** | **CO2** |
| **Algorithm** | **Algorithm to Unpack the Packed BCD Number**  **Step I         :**Initialize the data memory.  **Step II        :**Load number into register AL.  **Step III       :**Mask the lower nibble.  **Step IV       :**Rotate 4 times left to make MSB digit = LSB.  **Step V        :**Display the digit.  **Step VI       :**Load number in AL.  **Step VII      :**   Mask upper nibble.  **Step VIII      :**   Display the result.  **Step IX        :**Stop. |
| **Code** | 1. Packed to Unpacked BCD   .model small  .stack 100H  .data cinf  a db 35h ; store 1st no at location a  ab1 ? ; define ub1 location location to save unpacked version of a ab2 ? ; define ub1 location location to save unpacked version of a ends  .code  start: mov ax,@data mov ds,ax  mov ax,000h ;clear content of accumulator mov bl, 00fh ;move 0fh into reg bl  mov bh, 0f0h ;move f0h in reg bh  mov al, a ; move the first ASCII in al AND al,bl ; AND the content of al and bl  mov ub1,al ; move the result of ANDing at location ub1 mov al,a ;move the first ASCII no in al  AND al,bh ; AND the content of al and bh mov cl,04h ; move 04h in reg cl  ror al,cl ; shift the content of ch reg to left by 4 bits mov ub2,al ; move the result at location ub1  mov ah, 4ch int 21h  end start  end   1. Unpacked to Packed BCD   .model small  .stack 100H  .data  a db 05h ; store 1st no at location a  b db 03h ; store 2nd no at location b  pb1 db ? ; define pb1 locatio to save packed version of a and b  ends  .code  start: mov ax,@data mov ds,ax  mov ax,000h ;clear content of accumulator  mov al,b ; move the second unpacked in al mov cl,04h ; move 04h in reg cl  ror al,cl ; shift the content of ch reg to left by 4 bits mov ah,a ; move the first number into register ah OR al,ah ; or the content of al with ah  mov pb1,al mov ah, 4ch |

|  |  |
| --- | --- |
|  | int 21h end start  end |
| **Output** |  |
| **Conclusion** | The ASCII number can be unpacked or converted into HEX form by masking its upper nibble.  This masking can be done by two methods. One method is we can AND the ASCII number by 0FH or we can perform SAL/SHL operation 4 times.  This SHL instruction shifts the content of the source to left and appends 0s in LSB.  The unpacked number can be packed by using OR operation.  For e.g., 01H and 02H are two numbers to be packed, we can shift one of the number 4 times towards left by using SAL/SHL instruction and OR shifted number with the another number i.e. if 01H is shifted left four times, it becomes 10H. Then OR it with 02H.  The result will be 12H which is packed version of those two numbers. |

|  |  |
| --- | --- |
| **EXPERIMENT NO. 6** | |
| **Title** | Develop Assembly language program using 8086 microprocessor for block copy, block exchange with and without string instructions. |
| **Name** | Deep Salunkhe |
| **Roll No** | 21102A0014 |
| **Outcome** | **CO2** |
| **Algorithm** | **Copying Byte array from one location to another without using string instructions**   1. Start. 2. Allocate some space for the input array, output array and length of array in data segment. 3. Initialize CX with the length of the array. 4. Load the effective address of input string in SI and that of output string in DI. 5. Move the contents from the location whose offset is currently pointed by SI in Data segment to the location whose offset is currently pointed by DI in Data segment. 6. Increment SI,DI. 7. Decrement CX. 8. Repeat steps 5-7 until CX = 0. 9. Stop.   **Copying Byte array from one location to another using string instructions**   1. Start. 2. Allocate some space for the input array, output array and length of array in data segment. 3. Initialize CX with the length of the array. 4. Load the effective address of input string in SI and that of output string in DI. 5. Overlap the Data Segment and Extra Segment in memory. 6. Using the REPNZ prefix for MOVSB instruction, the array is copied into destination location. 7. Stop.   **Copying Word array from one location to another without using string instructions**   1. Start. 2. Allocate some space for the input array, output array and length of array in data segment. 3. Initialize CX with the length of the array. 4. Load the effective address of input string in SI and that of output string in DI. 5. Move the contents from the location whose offset is currently pointed by SI in Data segment to the location whose offset is currently pointed by DI in Data segment. 6. Increment SI and DI by 2. 7. Decrement CX by 2. 8. Repeat steps 5 & 6 until CX = 0. 9. Stop.   **Copying Word array from one location to another using string instructions** |

|  |  |
| --- | --- |
|  | 1. Start. 2. Allocate some space for the input array, output array and length of array in data segment. 3. Initialize CX with the length of the array. 4. Load the effective address of input string in SI and that of output string in DI. 5. Overlap the Data Segment and Extra Segment in memory. 6. Using the REPNZ prefix for MOVSW instruction, the array is copied into destination location. 7. Stop. |
| **Code** | 1. Copying Byte array from one location to another without using string instructions   Labels Mnemonics Operands START MOV AX,@DATA REPEAT LEA DS  INC CX LOOP LEN INT ARR1  ARR2 SI  DI 21H,4CH BL  .model small  .stack 100h  .data  arr1 db 02h,09h,06h,10h,07h arr2 db ?  len dw $-arr1 ends  .code  start: mov ax,@data  mov ds,ax mov cx,len lea si,arr1 lea di,arr2  repeat: mov bl,[si]  mov ds:[di],bl inc si  inc di  loop repeat mov ah,4ch int 21h  ends end start   1. Copying Byte array from one location to another using string instructions Labels Mnemonics Operands   START MOV AX,@DATA LEA DS  MOVSBCX REPNZ LEN INT ARR1  ARR2 SI  DI 21H,4CH ES  .model small  .stack 100h  .data  arr1 db 02h,09h,06h,10h,07h arr2 db ?  len dw $-arr1 ends  .code  start: mov ax,@data  mov ds,ax mov es,ax mov cx,len lea si,arr1 lea di,arr2 repnz movsb mov ah,4ch int 21h  ends end start |

|  |  |
| --- | --- |
|  | 1. Copying Word array from one location to another without using string instructions   Labels Mnemonics Operands START MOV AX,@DATA REPEAT LEA DS  ADD CX  SUB LEN  JNZ ARR1  INT ARR2 SI  DI 21H,4CH,2H BX  .model small  .stack 100h  .data  arr1 dw 0212h,0933h,0621h,1023h,0798h  arr2 dw ?  len dw $-arr1 ends  .code  start: mov ax,@data  mov ds,ax mov cx,len lea si,arr1 lea di,arr2  repeat: mov bx,[si]  mov ds:[di],bx add si,2h  add di,2h sub cx,2h jnz repeat mov ah,4ch int 21h  ends end start |

|  |  |
| --- | --- |
|  | 4) Copying Word array from one location to another using string instructions  Labels Mnemonics Operands  START MOV AX,@DATA LEA DS MOVSW CX REPNZ LEN  INT ARR1 ARR2 SI  DI 21H,4CH ES  .model small  .stack 100h  .data  arr1 dw 0212h,0933h,0621h,1023h,0798h  arr2 dw ?  len dw $-arr1 ends  .code  start: mov ax,@data  mov ds,ax mov es,ax mov cx,len lea si,arr1 lea di,arr2 repnz movsw mov ah,4ch int 21h  ends  end start |

|  |  |
| --- | --- |
| **Output** | Copying Byte array from one location to another without using string      **Copying Byte array from one location to another using string** |

|  |  |
| --- | --- |
|  | **Copying Word array from one location to another without using string** |

|  |  |
| --- | --- |
|  | **Copying Word array from one location to another using string** |

|  |  |
| --- | --- |
| **Conclusion** | 1. Segment override prefix is used to change the default segment when physical address is calculated internally by the microprocessor. 2. String instructions by default use the Data segment for input and Extra segment for output. 3. The instruction prefix REPNZ can only be used with string instructions andmakes   the processor execute the following string instruction till the whole input string is processed. |

|  |  |
| --- | --- |
| **EXPERIMENT NO. 7** | |
| **Title** | Develop assembly language program using 8086 microprocessor for I/O using INT N. |
| **Name** | Deep Salunkhe |
| **Roll No** | 21102A0014 |
| **Outcome** | **CO2** |
| **Algorithm** | Algorithm  **Perform addition of 2 numbers using I/O operations**   1. Start 2. Allocate some space for the messages to be displayed in the datasegment. 3. Display message to accept 1st input. 4. Accept the input from user and store it in some register. 5. Display message to accept 2nd input. 6. Accept the 2nd input from user and store in in some register. 7. Perform addition and store the result in DL register. 8. Stop. |
| **Code** | **Perform addition of 2 numbers using I/O operations**   |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  |  |  |  |  | | --- | --- | --- | | **Labels** | **Mnemonics** | **Operands** | | START | MOV | AX,@DATA | |  | LEA | DS | |  | INC | AH | |  | INT | msg1,msg2,msg3 | |  | SUB | CL | |  | ADD | BL | |  | AAA | DL | |  |  | DX | |  |  | AL | |  |  | 21H,4CH | |  |  | 09H,01H,30H,13H,10,02H |   .model small  .stack 100h  .data  msg1 db 'Enter the 1st number: $' msg2 db 'Enter the 2nd number: $' msg3 db 'Result: $'  ends  .code  start: mov ax,@data mov ds,ax mov ah,09h lea dx,msg1 int 21h  mov ah,01h int 21h  sub al,30h mov bl,al  mov DL, 10 ;printing new line mov AH, 02h |

|  |  |
| --- | --- |
|  | int 21h mov DL, 13  mov AH, 02h int 21h  mov ah,09h lea dx,msg2 int 21h  mov ah,01h int 21h  sub al,30h mov cl,al  mov DL, 10 ;printing new line mov AH, 02h  int 21h mov DL, 13  mov AH, 02h int 21h  mov ah,09h lea dx,msg3 int 21h  add cl,bl mov al,cl aaa  add al,30h mov dl,al  mov ah,02h int 21h  mov ah,4ch int 21h  end start  end |

|  |  |
| --- | --- |
| **Output** |  |
| **Conclusion** | 1. DX is used to store the offset of the message which is to be displayed on the output screen. 2. By default the input taken from the user is stored in AL. 3. Combination of various data stored in AH and interrupts allow us to makeuse of different services like display message, display data in register, display new line etc. |

|  |  |
| --- | --- |
| **EXPERIMENT NO. 8** | |
| **Title** | 8086 Assembly Language Program to Count Odd and Even Numbers from the given block of Numbers |
| **Name** | Deep Salunkhe |
| **Roll No** | 21102A0014 |
| **Outcome** | **CO2** |
| **code** | .model small  .stack 100H  .data  array db 02h,04h,05h,10h,11h oddcnt db ? evencnt db ?  ends  .code  start: mov ax,@data  mov ds,ax  mov cx,05h  lea si,array  next: mov al, [ si ]  roral,01h ; rotate right array element present in alsoi ts LSB goes into CF  jnc even ; if CF =0 goto lable even  inc dl ; else due to CF=1increment odd count jmp ahead ; goto next element  even:incdh ;due to CF=0 increment even count in reg dh mov oddcnt,dl; copy countindlreg into variable ‘oddcount  mov evencnt,dh ; copy count in dh reg into variable ‘evencoun  ahead: incsi ; increment array pointer to address next element loop next ;dec byte counter cx and if cx not=0 go to lable next  mov ah,4ch  int 21h  end start  end |

|  |  |
| --- | --- |
| **Output** |  |
| **Conclusion** | To find the number of even and odd numbers in given string ,we can make use of RCR instruction which rotates the all the bits in a specified word or byte some number of bit positions to the right.  The operation is circular because the LSB of the Operand is rotated into the carry flag and the bit in the carry flag is rotated around into the MSB of the operand.  Thus by checking carry flag we can increment the content of location where number of even numbers is to be saved if carry flag is reset .  Then we can also find then number of odd numbers in string by subtracting number of even numbers from the string length. |