```
            section .data

L1:         db      'Original # :'
L2:         db      'Bad 2 bit# :'
L2a:        db      'Bad 1 bit# :'
L3:         db      'Hamming #  :'
L4:         db      'In Error    ',0xa
L5:         db      'Match OK    ',0xa
L6:         db      'Can Correct ',0xa
L7:         db      'Cant Correct',0xa
L9:         db      'Test Number:'
L10:        db      'Corrected #:'
L11:        db      'Bad Bit at : '

L20:        db      '========================',0xa
L21:        db      'Test With Orignal Number ',0xa
L22:        db      'Test With Bad Data Bit   ',0xa
L23:        db      'Test With Bad Parity Bit ',0xa

ORIG:       dq      01010101b           ; Original good number  ->> Good number with Hamming (8,4) Set --
>>       0001101001010
GOODHAM:    dq  0001101001010b          ; Original Number Hamming encoded
BADHAM:     dq      0001101001110b      ; Bad number with good Hamming parity set and good Extra Partiy
(above)...This can be error corrected.
BADPAR:     dq      0001101001011b      ; Bad number with good Hamming parity set and Bad Extra parity
(above)...This CAN NOT be error corrected.

OUTPUT:     db      '                 ' ;output buffer
WCODE:      dq      0                   ;                   ; buffer for good number with partiy
RESULT:     dq      0                   ; Working space (use for output)
PBIT:       dq      0                   ;work space for extra parity bit
TEST1:      dq      0                   ;for detecting match or errors
TEST2:      dq      0
TEMP:       dq      0
LEN:        db      0

            section .text
            global  main                ;Tell linker about main
            extern  write, exit
main:
            mov     rbp, rsp            ; for correct debugging
            push    rbp
            mov     rbp, rsp

                                        ;Display Hamming (8,4) encoding of good number..
            xor     rax,rax             ; clear things to start
            mov     rax,[ORIG]          ; get original value
            mov     [RESULT],rax        ; store it for display
            lea     rsi,[L1]            ; load label
            call    MYWRITE             ; write value out in binary
            mov     rax,[ORIG]
            mov     [WCODE],rax         ; setup Hamming Call
            call    HAMMING             ; create the Hamming value
            mov     rax,[RESULT]        ; Store in test1 for comparison
            lea     rsi,[L3]            ; load label
            call    MYWRITE             ; write value out in binary

                                        ;Test with Original Number (Hamming encoded) as test
            lea     rsi,[L20]
            call    MYWRITE3
            lea     rsi,[L21]
            call    MYWRITE3

            mov     rax,[GOODHAM]       ; load value with bad bit but good extra parity
            mov     [RESULT],rax
            lea     rsi,[L9]            ; load label
            call    MYWRITE             ; write value out in binary
```

```
            mov     rax,[GOODHAM]       ; load value with bad bit but good extra parity
            mov     [WCODE],rax         ; setup call
            call    CORR                ; call correct...


                                        ;Test with BADHAM - Bad Data Bit, but good hamming & parity
            lea     rsi,[L20]
            call    MYWRITE3
            lea     rsi,[L22]
            call    MYWRITE3

            mov     rax,[BADHAM]        ; load value with bad bit but good extra parity
            mov     [RESULT],rax
            lea     rsi,[L9]            ; load label
            call    MYWRITE             ; write value out in binary
            mov     rax,[BADHAM]        ; load value with bad bit but good extra parity
            mov     [WCODE],rax         ; setup call
            call    CORR                ; call correct...

                                        ;Test with BADPAR - Good Data, but bad parity
            lea     rsi,[L20]
            call    MYWRITE3
            lea     rsi,[L23]
            call    MYWRITE3

            mov     rax,[BADPAR]        ; load value with bad bit but good extra parity
            mov     [RESULT],rax
            lea     rsi,[L9]            ; load label
            call    MYWRITE             ; write value out in binary
            mov     rax,[BADPAR]        ; load value with bad bit but good extra parity
            mov     [WCODE],rax         ; setup call
            call    CORR                ; call correct...


MX:         xor     edi, edi            ; 0 return = success
            call    exit



; USAGE: load global WCODE with value in error
CORR:       mov rax,0                   ;clear temp,Parity Bit & result
            mov [TEMP],rax
            mov [PBIT],rax
            mov [RESULT],rax

            mov rax,[WCODE]             ;get value
            shr rax,1                   ;get rid of extra parity
            jnc C0
            mov rbx,1                   ;record 1 parity bit (zero'd above)
            mov [PBIT],rbx

C0:         mov [WCODE],eax             ;store number

            and rax,001010101010b       ;Bits for Position 1
            call PARITY
            jz  C1
            mov rax,1                   ;mark bit position 1 as bad
            mov [TEMP],rax


C1:         mov rax,[WCODE]
            and rax,001011101111b       ;Bits for position 2
            call PARITY
            jz  C2
            mov rax,[TEMP]              ;mark bit position 2 as bad
            add rax,2
            mov [TEMP],rax
```

```
C2:         mov rax,[WCODE]
            and rax,000111100001b       ;Bits for position 4
            call PARITY
            jz  C3
            mov rax,[TEMP]              ;mark bit position 4 as bad
            add rax,4
            mov [TEMP],rax

C3:         mov rax,[WCODE]
            and rax,000000001111b       ;Bits for Position 8
            call PARITY
            jz  C4
            mov rax,[TEMP]              ;mark bit position 8 as bad
            add rax,8
            mov [TEMP],rax

C4:         cmp rax,0                   ;alls ok, nothing needs to be done.
            mov rax,[WCODE]
            jz  CPACK

            lea     rsi,[L11]
            call    MYWRITE2
            mov     rax,[TEMP]
            call    TODEC

            mov     rax,12              ;get value again
            cmp     rax,[TEMP]
            jng     CNOT                ;Bit Position greater that data..meaning more than one error
            sub     rax,[TEMP]          ; we are working with 12 bit values...
            mov     rbx,rax             ;store shift value
            mov     rax,[WCODE]         ;get main value again

            mov     rcx,rbx             ;setup loop counter
C3a:        ror     rax,1               ;rotate bit into position
            loop    C3a
            xor     rax,1               ;toggle bit
            mov     rcx,rbx
C3b:        rol     rax,1               ;rotate back
            loop    C3b
            mov     [WCODE],rax         ;store corrected value



CPACK:                                  ; now that we corrected, lets check extra parity bit
            and     rax,001011101111b   ;get rid of parity bits... (_ _ 1 _ 1 1 1 _ 1 1 1 1)
            mov     rbx,rax             ;store
            and     rax,000000001111b   ;get chunk
            mov     rcx,rax             ;store
            mov     rax,rbx
            and     rax,000011100000b   ;get chunk
            shr     rax,1               ;shift to left 1
            add     rax,rcx             ;add 2 chunks
            mov     rcx,rax             ;store
            mov     rax,rbx
            and     rax,001000000000b   ;get last chunk
            shr     rax,2               ;shift left 2
            add     rax,rcx             ;add in remaining chunk
            mov     [RESULT],rax        ;stor in result.

            lea    rsi,[L10]            ;write out number
            call   MYWRITE


                                        ;now that we have packed number, check extra parity
            mov    rax,[RESULT]
            call    PARITY
```

```
            jz      CPACK2
            mov     rax,1
            cmp     rax,[PBIT]
            jne     COK
            jmp     CNOT

CPACK2:     mov     rax,0
            cmp     rax,[PBIT]
            je      COK
            jmp     CNOT

COK:
            lea     rsi,[L5]            ; load label
            call    MYWRITE2            ; Alls OK
            ret


CNOT:       lea     rsi,[L7]            ; load label
            call    MYWRITE2            ; Write CAN NOT CORRECT
            ret




; USAGE: Load Global WCODE with number...
HAMMING:    mov rax,0                   ;                ; clear RESULT
            mov [RESULT],rax

                                        ;first calc the extra parity bit.
            mov rax,0
            mov [PBIT],rax              ;clear parity bit
            mov rax,[WCODE]             ;Get value
            call PARITY
            jz  H0
            mov rax,1                   ;set parity bit for last
            mov [PBIT],rax

                                        ;make room for parity bits...
H0:         mov rax,[WCODE]             ;get original value
            and rax,00001111b           ;get first chunk
            mov [RESULT],rax            ;store it

            mov rax,[WCODE]             ;get original value
            and rax,01110000b           ;get next chunk to shift
            shl rax,1                   ;make room for parity 3
            add rax,[RESULT]            ;add in first chunk
            mov [RESULT],rax            ;store it

            mov rax,[WCODE]             ;get original value
            and rax,10000000b           ;get next chunk to shift
            shl rax,2                   ;make room for parity 2
            add rax,[RESULT]            ;add in first chunk
            mov [RESULT],rax            ;store it

                                        ;now calulate parity and place bits

            mov rax,[RESULT]            ;Get value to calc position 1
            and rax,101010101010b       ;Bits for Position 1

            call PARITY
            jz  H1
            mov rax,[RESULT]
            xor rax,100000000000b       ;set flag only if 1 cause its already zero
            mov [RESULT],rax            ;store it for prosperities sake
```

```
H1:         mov rax,[RESULT]
            and rax,011001100110b       ;Bits for position 2
            call PARITY
            jz  H2
            mov rax,[RESULT]
            xor rax,010000000000b       ;set flag only if 1 cause its already zero
            mov [RESULT],eax

H2:         mov rax,[RESULT]
            and rax,000111100001b       ;Bits for position 4
            call PARITY
            jz  H3
            mov rax,[RESULT]
            xor rax,000100000000b       ;set flag only if 1 cause its already zero
            mov [RESULT],rax            ;store it for prosperities sake

H3:         mov rax,[RESULT]
            and rax,000000011111b       ;Bits for Position 8
            call PARITY
            jz  H4
            mov rax,[RESULT]
            xor rax,000000010000b       ;set flag only if 1 cause its already zero
            mov [RESULT],rax

H4:         mov eax,[PBIT]              ;get extra parity bit value again
            jz H5
            mov rax,[RESULT]
            shl rax,1
            xor rax,0000000000001b      ;set bit 1 (Extra Parity)
            mov [RESULT],rax
            ret

H5:         mov eax,[RESULT]
            shl rax,1
            and rax,1111111111110b      ;Clear bit 1  (Extra Parity)
            mov [RESULT],rax
            ret



;calc parity on 16 bits (1 word)
;Usage, load RAX with value...Sets zero flag if even
PARITY:     xor bx,bx                   ;clear counter
            mov ecx,16                  ;testing for 16 bits
            and ax,0xFFFF               ;make sure!
PLOOP:      shr ax,1                    ;shift right bit -> CF
            jnc PLOOP1                  ;loop if zero
            inc bx                      ;inc bit counter
PLOOP1:     loop PLOOP                  ;loop

POUT:       and bx,1                    ;test bit 1 ... if so, always odd, ZF = 1 for even, 1 for odd
            ret

; Usage: Load RSI with label
MYWRITE3:
            mov     edx, 26             ; write label
            mov     edi, 1
            call    write
            ret

; Usage: Load RSI with label
MYWRITE2:
            mov     edx, 13             ; write label
            mov     edi, 1
            call    write
            ret
```

```
; Usage: Load RSI with label
MYWRITE:
            mov     edx, 12             ; write label
            mov     edi, 1
            call    write

            mov     rax,[RESULT]        ; get Data to output
            call    TOBIN               ; convert to binary for output

            mov     edx, 17             ; default length for binary word
            lea     rsi, [OUTPUT]       ; load buffer
            mov     edi, 1
            call    write
            ret


;Usage : Load value into EAX
TOBIN:
            mov bx,0x8000               ;load divisor...
            mov ecx,OUTPUT              ;point to output string
TB1:
            xor edx,edx                 ;clear things
            div bx                      ;eax = quotient, edx = remainder
            add ax,48                   ;ascii adjust
            mov [ecx],al                ;store number...
            inc ecx                     ;inc String Pointer
            mov ax,dx                   ;get remainer
            ror bx,1                    ;rotate divisor
            cmp bx,0x8000               ;have we gone thru whole thing?
            jne TB1                     ;loop if not
            mov [ecx], byte 0xa         ;Line Feed
            ret


;Usage : Load value into EAX
TODEC:
            mov ebx,0
            mov [LEN],ebx               ;clear length
            mov ebx,10                  ;load divisor...
            mov ecx,OUTPUT              ;point to output string
TD1:
            xor edx,edx                 ;clear things
            div ebx                     ;eax = quotient, edx = remainder
            add eax,48                  ;ascii adjust
            mov [ecx],al                ;store number...
            mov eax,[LEN]               ;Inc Length
            inc eax
            mov [LEN],eax
            inc ecx                     ;inc string ptr
            mov edi,edx                 ;store remainer
            mov eax,ebx                 ;mov divisor for divide
            mov ebx,10                  ;setup divide
            xor edx,edx                 ;clear things
            div ebx                     ;reduce divisor
            mov ebx,eax                 ;get divsor
            mov eax,edi                 ;restore remainer
            cmp ebx,0                   ;have we gone thru whole thing?
            jg TD1                      ;loop if not

            mov [ecx], byte 0xa         ;null term
            mov eax,[LEN]               ;Inc Length
            inc eax
            mov [LEN],eax
```

```
        lea     rsi,[OUTPUT]
        mov     edx, [LEN]          ; write value
        mov     edi, 1
        call    write

        ret
```