آزمایش 2

اعضای گروه : مهدیه مقیسه (99243068)، آرین جلالیان (99243026)

بخش A)

سوالات تحلیلی :

1) پردازنده 8086 دارای 20 بیت آدرس و 16 بیت دیتا است. برای کاهش pin ها و line ها و همچنین فضا و هزینه کمتر این پین ها با هم ترکیب شده اند. برای تفکیک آدرس و دیتا در یک clock cycle آدرس و در بعدی دیتا را روی پین ها قرار می دهیم. برای جداسازی آن ها از هم پین های این bus را روی یک Latch مانند 74373 استفاده می کنیم و enable این latch را به ALE پردازنده در سیکلی که آدرس در bus قرار داشت، وصل می کنیم. برای تعیین جهت حرکت دیتاها از 8086 به ممووری یا I/O از یک data transceiver استفاده می کنیم و به پین DT/R پردازنده وصل می کنیم که مشخص شود دیتاهای موجود روی کدام پین باید به طرف دیگر منتقل شود.
2) تراشه های 6264 و 62256 RAM هستند. در این تراشه ها پیش CE برای فعال کردن آن است که به range آدرس decoder وصل می شود. پین های D[0..7] برای قرار گرفتن داده های RAM است و در 6264 پین های آدرسی که می خواهیم از آن ها بخوانیم یا در آن ها بنویسم به صورت A[0..12] و در 62256 به صورت A[0..14] است. در هر دو تراشه پین OE برای خواندن از حافظه استفاده می شود.هنگام خواندن از حافظه این پین فعال می شود و مقدار آدرس های روی پین ها روی پین های دیتا قرار میگیرد. در هنگام نوشتن باید پین WE فعال شود که داده ها از روی پین های دیتا به پین های آدرس منتقل شوند و داده ها در حافظه ذخیره شوند.

// عکس 6264 و 62256

تراشه های 27128 و 27256 ROM هستند. در 27128 پین های آدرس به صورت A[0..15] و در 27256 به صورت A[0..14] است. پین های دیتا و CE و OE مانند قسمت قبل هستند. در ROM پین WE برای نوشتن نداریم. یک پین VPP هم دارد که آن را باید به POWER وصل کنیم. در 27128 یک پین PGM هم داریم که برای CONFIG به کار می رود.

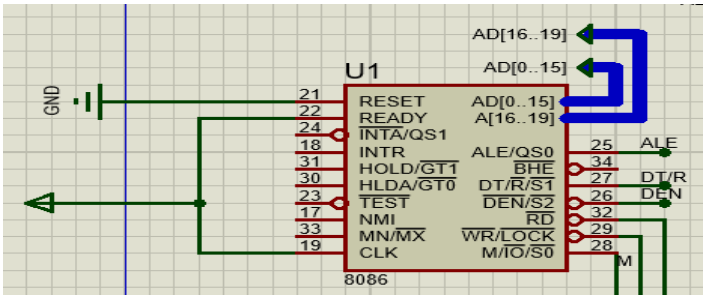
3) شکل سوال عملیات نوشتن در حافظه را نشان می دهد که به شرح زیر است :

cycle	توضیحات
T1	در این سیکل آدرس فرستاده می شود و سیگنال های ALE و DT/R فعال اند. ALE خروجی پردازنده را latch می کند. DT/R با استفاده از transceiver جهت انتقال دیتا را مشخص می کند.
T2	سیگنال ALE غیر فعال می شود و WR و DEN در صورت وجود transceiver فعال می شوند. در این سیکل دیتا خارج شده از پردازنده latch نمی شود تا آدرس خراب نشود و آن را بنویسیم.
T3	در این سیکل تمامی سیگنال ها مقداردهی شده اند تا دیتا در حافظه نوشته شود.
TW	دیتا در این سیکل روی حافظه نوشته می شود.

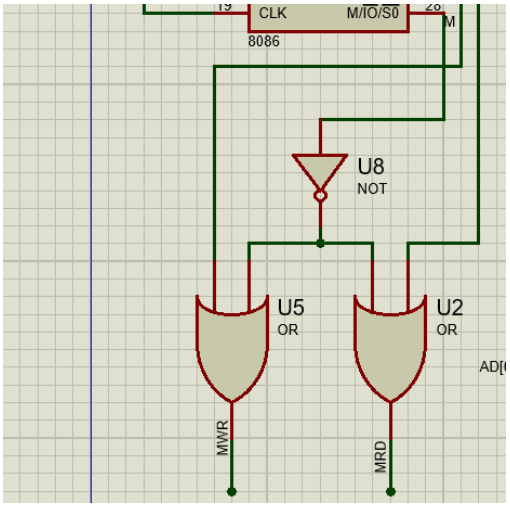
T4	بعد از نوشتن دیتا کار تمام می شود. سیگنال های کنترلی reset می شوند.
T1	عملیات نوشتن دوباره شروع می شود.

سوالات کدی :

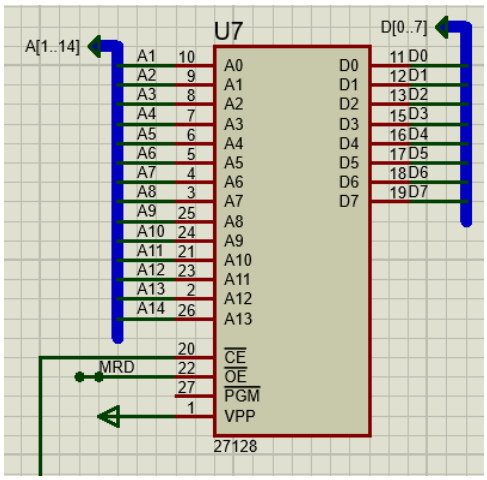
1) برای این قسمت ابتدا پردازنده، ROM و RAM و decoder را قرار می دهیم. ساختار پردازنده 8086 به صورت زیر است :



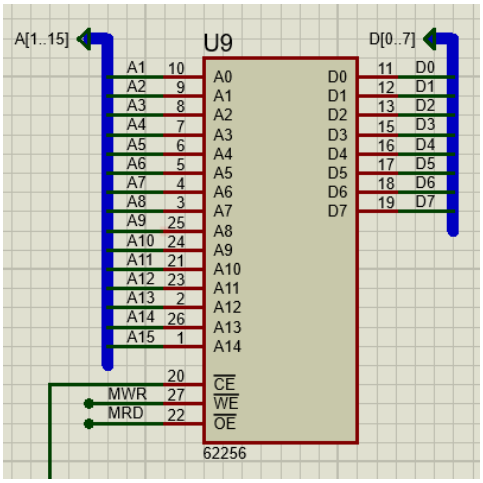
برای وصل کردن port های read و write پردازنده به حافظه ها باید ترکیب این پین ها را با پین پردازنده (M/IO) که مشخص می کند داریم با مموری کار می کنیم یا I/O. ترکیب آن ها به صورت زیر است :



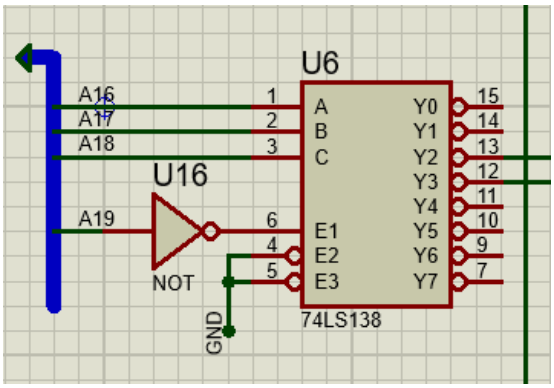
برای ROM همانطور که در صورت سوال گفته است از 27128 (8 * 16K)EPROM استفاده می کنیم. در سمت چپ port های آن را به bus آدرس (از 1 تا 15) وصل می کنیم و port های سمت راست را به bus دیتا وصل می کنیم. CE که enable این حافظه است را باید به خروجی ترکیب decoder و M/IO وصل کنیم. OE که همان output enable است را هم باید به MRD که در قسمت قبل ایجاد شده است، وصل کنیم :



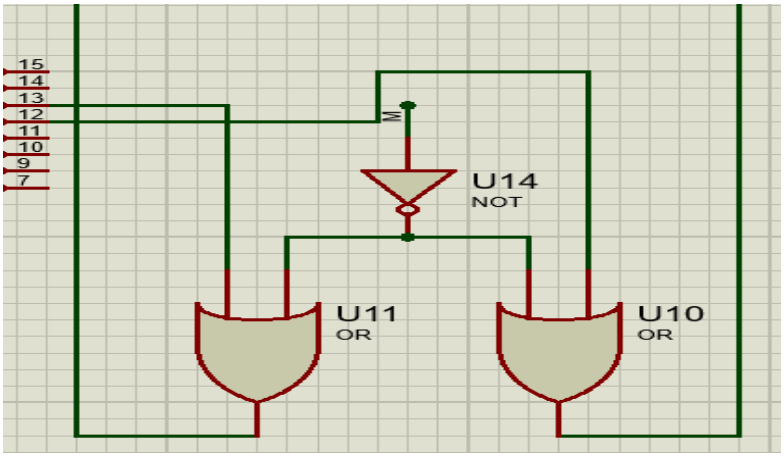
برای RAM همانطور که در صورت سوال گفته شده است از 62256 (8 * 32k)RAM استفاده می کنیم. Port های چپ و راست مموری را مانند ROM به bus ها وصل می کنیم. CE که enable مموری است را به خروجی ترکیب decoder و M/IO وصل می کنیم. OE را به MRD و WE را به MWR وصل می کنیم :



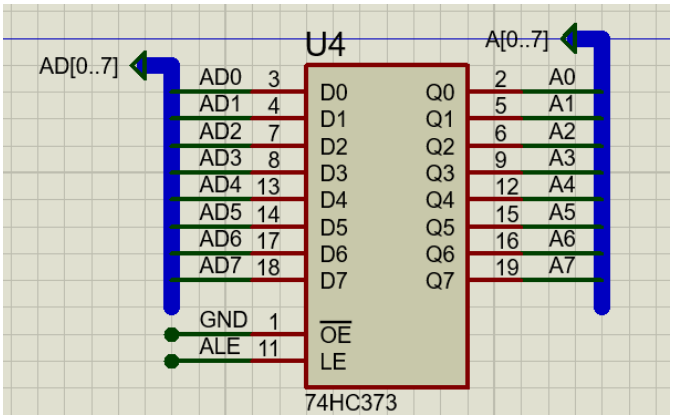
ساختار decoder به صورت زیر است. این decoder سه ورودی دارند که بیت های 16 تا 18 ادرس هستند. سه enable هم در آن وجود دارد که دوتا از آن ها را به زمین و یکی را به not بیت 19 ادرس وصل می کنیم. خروجی آن range آدرس های مموری ها را مشخص می کند :



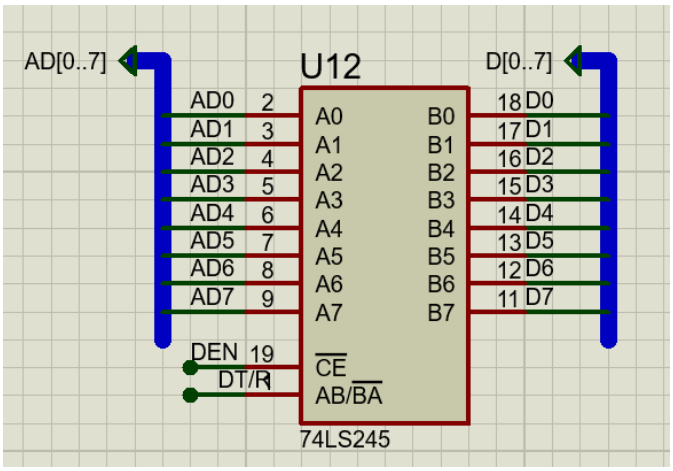
برای وصل کردن خروجی decoder به مموری ها ابتدا آن را با M/IO پردازنده or می کنیم تا مشخص شود که داریم با مموری کار می کنیم :



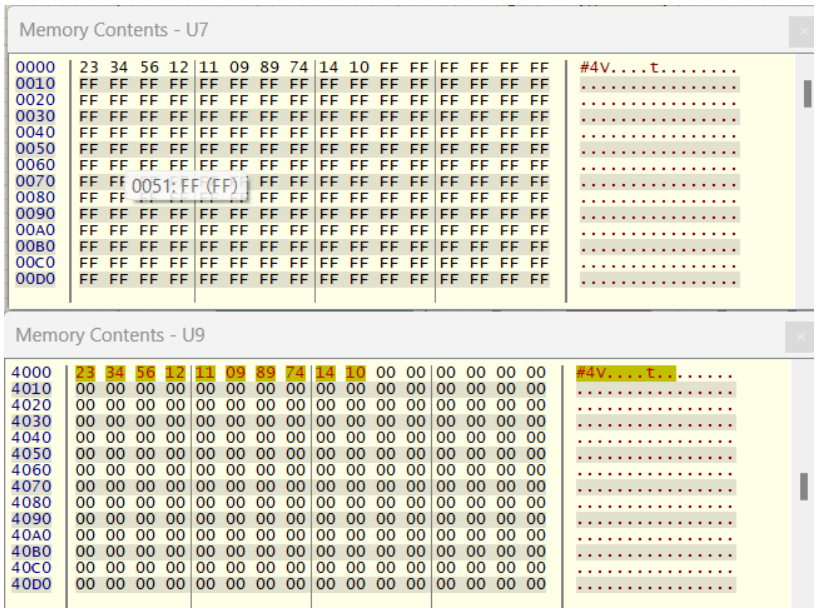
روی پردازنده bus وصل شده که هم دیتا و هم آدرس را منتقل می کند. برای اینکه مقدار دیتا در سیکل بعدی آدرس سیکل قبل را خراب نکند از LATCH استفاده می کنیم تا به بیت 19 برسیم. Enable آن را به ALE پردازنده که همان address latch enable است، وصل می کنیم :



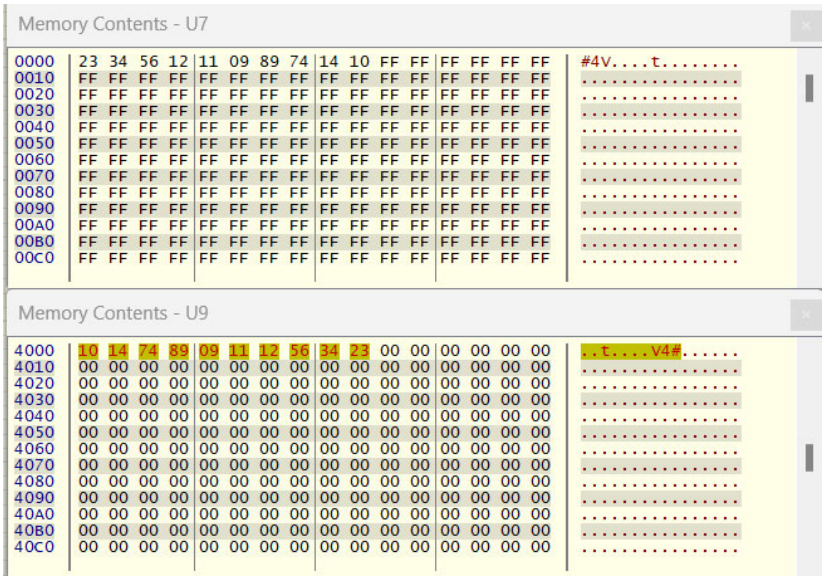
برای انتقال دیتاها باید از data transceiver استفاده کنیم. در این سوال چون دیتاها 8 بیتی هستند یکی کافی است. Enable آن را به DEN پردازنده وصل می کنیم و port پایینی را هم به DT/R پردازنده وصل می کنیم :



سپس فایل bin. را روی ROM قرار می دهیم و مدار را شبیه سازی می کنیم. Memory content این شبیه سازی به صورت زیر است :



سپس سورس کد دوم که انتقال معکوس بود را قرار می دهیم و دوباره شبیه سازی می کنیم. Memory content به صورت زیر است :



2) در این سوال باید یک روتین بنویسیم که N عدد حافظه ROM را به ابتدای حافظه RAM منتقل کند. برای این کار یک procedure به نام CPY قرار می دهیم.در این سوال به دلخواه عدد N را 10 در نظر گرفتیم. در ابتدای کار و قبل از شروع حلقه تعداد انتقال ها و index های مبدا و مقصد را ست می کنیم :

```
cpy proc near

    mov cx, 10   ;10 transfer needed
    mov si, 0 ;set source index
    mov di, 0 ;set destination index
```

سپس یک loop ایجاد می کنیم. در ابتدای حلقه ادرس شروع ROM که از decode کردن به دست آوردیم را در data segment قرار می دهیم. برای مثال decode ادرس برای ROM و RAM به این صورت به وجود آمده است (تقریبا 64 کیلوبایت حافظه برای حافظه داخلی پردازنده است پس ادرس ها را به گونه ای قرار دادیم که تداخل نداشته باشد) :

0010100000000000000 : 28000H

0011100000000000000 : 38000H

در ادامه بعد از ست کردن ادرس ROM مقدار موجود در خانه SI را به DX منتقل می کنیم.

```
loop1:
  mov ax, 2800h  ;set datasegment for read data from rom
  mov ds, ax
  mov dx, [si]         ;read and move data from rom to dx
```

در ادامه ادرس RAM را در data segment ست می کنیم و مقدار رجیستر DX را در خانه DI قرار می دهیم.

```
mov ax, 3800h  ;set datasegment for write data to ram
mov ds, ax
mov [di], dx         ;move data in dx to ram
```

در ادامه چون 8086، word addressable است، هر کدام از index ها را دو واحد زیاد می کنیم و از counter یکی کم می کنیم و اگر صفر نبود به ابتدای حلقه برمیگردیم.

```
inc di   ;incease di
inc di
inc si   ;increase si
inc si

dec cx    ;decrease the number of transfer needed
cmp cx, 0 ;if number of transfers equals to 0, exit the loop
jnz loop1
```

سپس در main این تابع را فراخوانی می کنیم و به دلیل روتین بودن آن دائما باید تکرار شود :

```
main proc near

    mainL :

      call cpy ;call cpy to tranfer data from ROM to RAM



      jmp mainL



    ret
main endp
```

3) در این سوال باید N عدد از حافظه ROM را به طور معکوس به RAM منتقل کنیم. قبل از شروع حلقه تعداد انتقال ها که 10 است و همچنین index های مبدا و مقصد را ست می کنیم. در این سوال SI برابر 18 است چون 10 عدد داریم و ادرس ها 16 بیتی هستند و از صفر شروع می شوند .

```
invcpy proc near

    mov cx, 10 ;10 transfer needed
    mov si, 18 ;set source index
    mov di, 0 ;set destination index
```

مانند قسمت قبل ابتدا ادرس های ROM و RAM را در data segment ست می کنیم سپس محتوای خانه SI را در DX قرار می دهیم و در خانه DI میریزیم.

```
loop2:
  mov ax, 2800h  ;set datasegment for read data from rom
  mov ds, ax
  mov dx, [si]
  ;;;;;;;;
  ;mov dx, 6754h   ;testing
  ;;;;;;;;;;;
  mov ax, 3800h
  mov ds, ax
  mov [di], dx
```

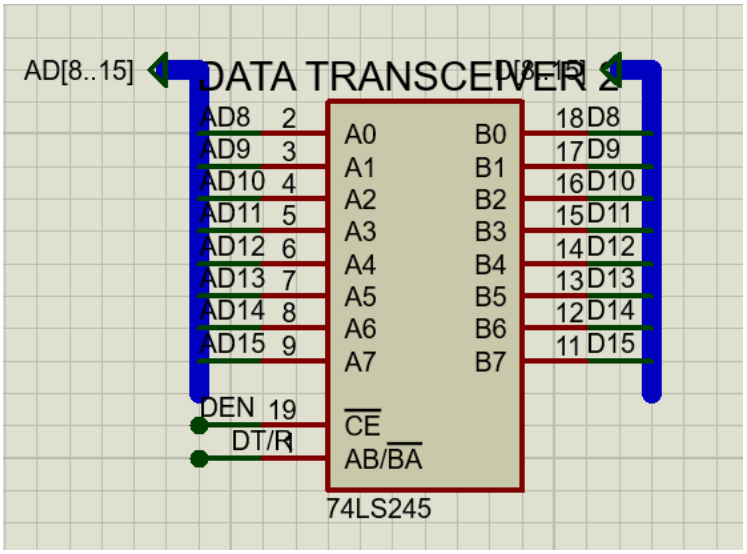
سپس SI را دو واحد کم و DI را دو واحد زیاد می کنیم و از counter یکی کم می کنیم و شرط صفر نبودن آن را بررسی می کنیم :

```
inc di
inc di
dec si
dec si

dec cx
cmp cx, 0
jnz loop2
```

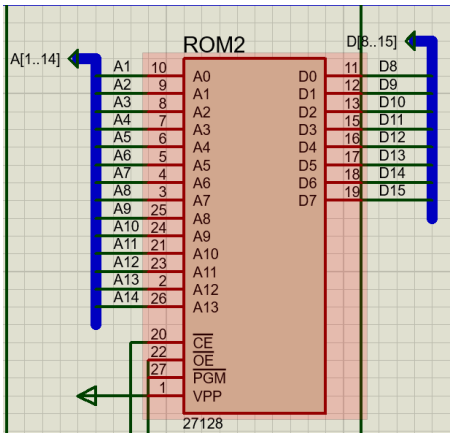
سپس در main این تابع را دائما فراخوانی می کنیم :

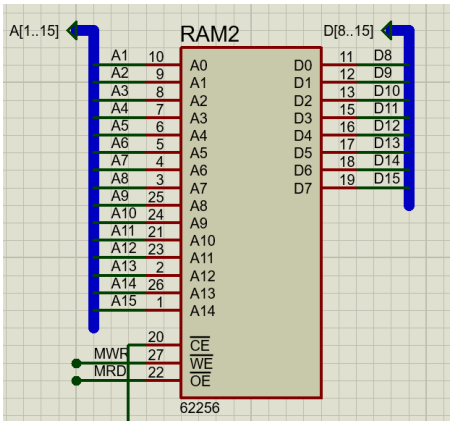
```
main proc near

    mainL :


      call invcpy ;call invcpy to tranfer data from ROM o RAM in revers

      jmp mainL


    ret
main endp
```

4) برای این قسمت باید تغییراتی در مدار سوال یک ایجاد کنیم. ابتدا یک data transceiver به مدار اضافه می کنیم که انتقال 8 بیت بعدی هم انجام بگیرد و port های سمت راست آن را بیت های دیتای 8 تا 15 وصل می کنیم :

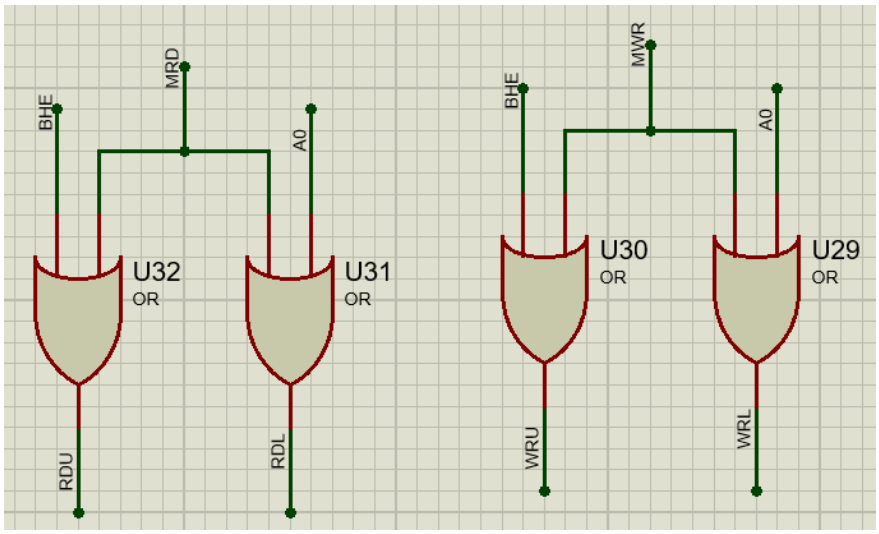


سپس یک ROM و یک RAM برای 8 بیت اضافه شده در مدار قرار می دهیم و بیت های 8 تا 15 را روی آن ها قرار می دهیم و enable های آن ها را به ترکیب خروجی decoder وصل می کنیم :

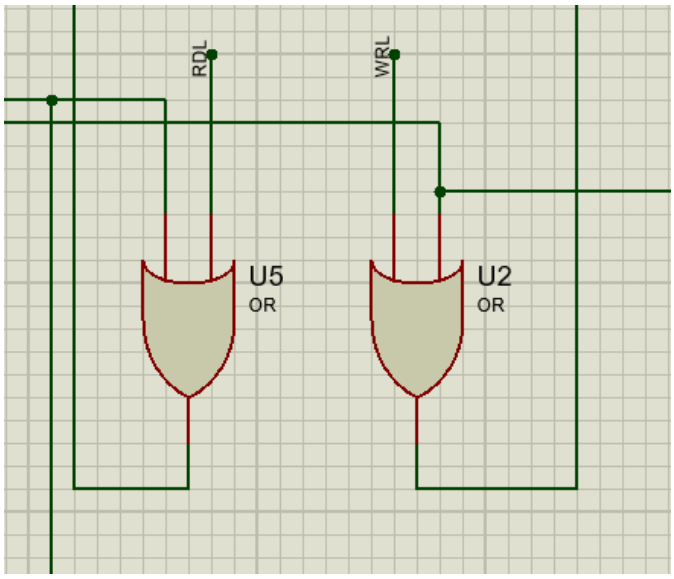


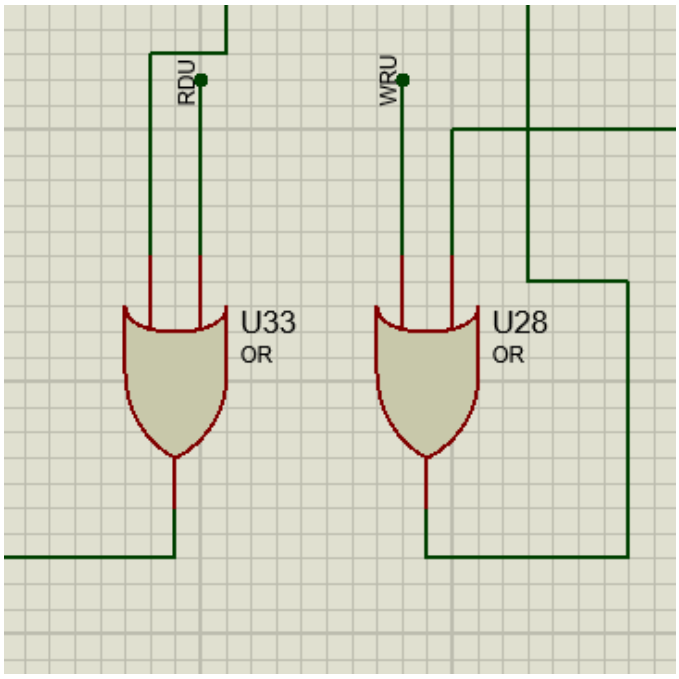


همانطور که صورت سوال گفته شده است 8 بیت بالا باید در بانک زوج و 8 بیت پایین باید در بانک فرد قرار بگیرند. پس باید حاصل ترکیب MRD و MWR با BHE پردازنده که همان bank high enable است را با خروجی decoder ترکیب کنیم و به مموری ها وصل کنیم :



به این ترتیب 8 بیت بالا در بانک زوج و 8 بیت پایین در بانک فرد ذخیره می شوند :





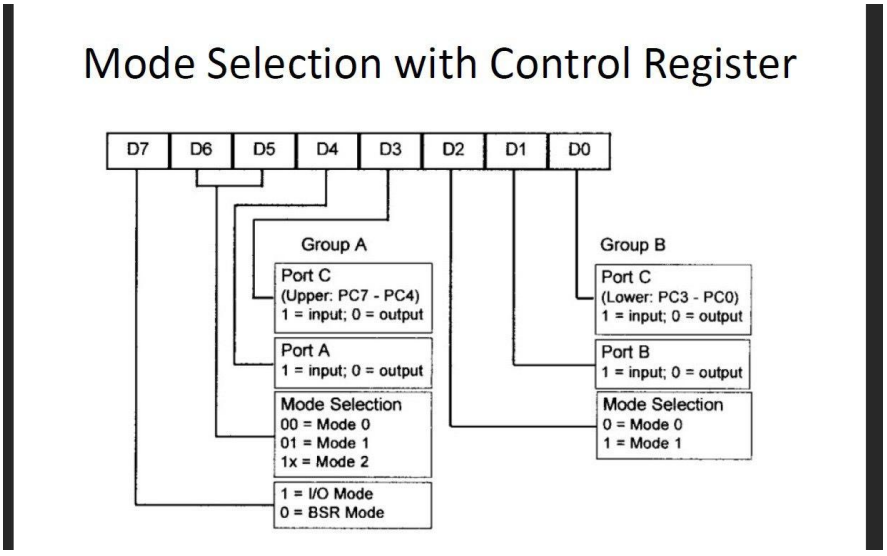
5) برنامه های اسمبلی ایجاد شده در قسمت 2 و 3 را در مدار جدید قرار می دهیم و شبیه سازی را انجام می دهیم. Memory content برای 8 بیت پایین در بانک های فرد ROM , RAM و برای 8 بیت بالا در بانک زوج، به صورت زیر است :

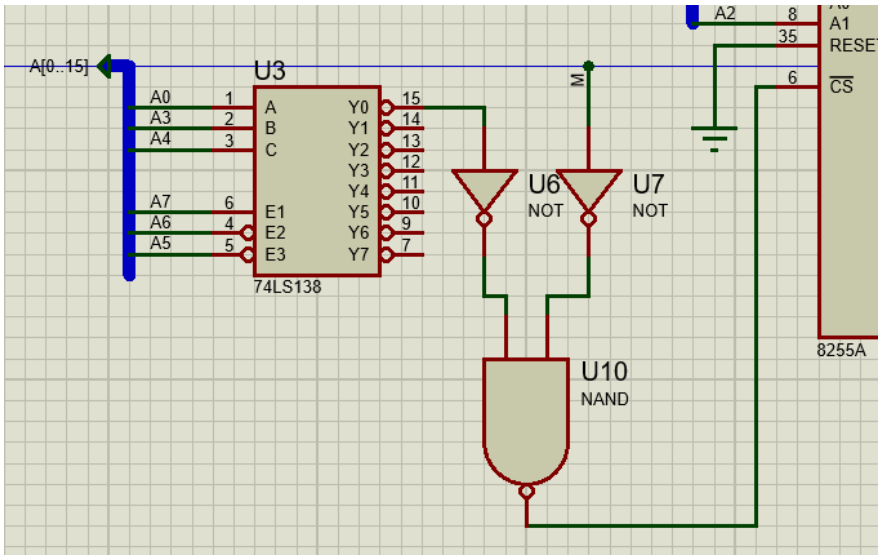
بخش B)

سوالات تحلیلی :

در صورت سوال از ما خواسته شده تا 8255 را در 80h دیکود کنیم و A0 و A1 در 8255 به A1 وA2 پردازنده وصل هستند که مشخص می کنند در هر لحظه کدام port فعال باشد. برای اینکه شروع از 80h باشد، بیت 7 و 6 و 5 آدرس latch شده را به enable های decoder وصل می کنیم. A3 وA4 و A0 ورودی های اصلی decoder هستند. در نتیجه اگر A1 وA2 صفر باشند، PA با 80h انتخاب می شود (10000000). با مقادیر متفاوت برای A2 , A1 به ترتیب PB، PC و config با آدرس های 84h و82h و 86h انتخاب می شوند.

1 0 0 0 0 0(A1) 0(A0) 0 => 80h => PA
1 0 0 0 0 0(A1) 1(A0) 0 => 82h => PB
1 0 0 0 0 1(A1) 0(A0) 0 => 84h => PC
1 0 0 0 0 1(A1) 1(A0) 0 => 86h => Config

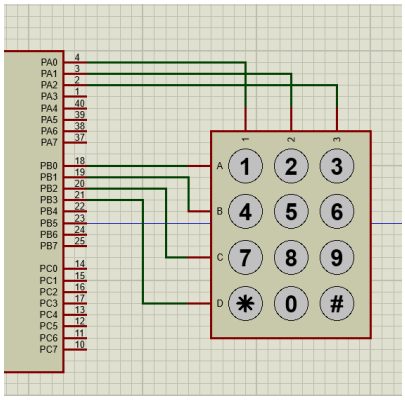




همانطور که در شکل بالا مشاهده می کنیم، در صورتی که بیت 0 از دیکدرفعال باشد (0 عبور داد شود) خروجی اینورتر 1 می شود و در صورتی که بیت I/O هم فعال شود (0 عبور داده شود) خروجی دیگر inverter هم صفر می شود. در نهایت خروجی nand صفر است که باعث فعال شدن 8255 می شود. (active low).

سوالات عملی :

1) در این بخش از یک صفحه کلید ماتریسی به شکل زیر استفاده کردیم و آن را به 8255 وصل کردیم :



8255، یک interface بین پردازنده 8086 و I/O است. سه port دارد که از این port ها برای خواندن و نوشتن استفاده می شود. یک port هم داخل آن قرار دارد که config است و مشخص می کند از کدام از آن سه port برای خواندن یا نوشتن استفاده می شود.

طبق config که در کد (10000010) انجام دادیم، مشخص می کنیم که PA ها را خودمان ورودی می دهیم و از PB می خوانیم.

```
keypad_input proc near

        ; CONFIG 8255
        mov al, 10000010B
        out 86H, al
```

در صورت ورودی دادن از طریق صفحه کلید ماتریسی، دو حالت به وجود می آید. حالت اول به این صورت است که ستون های اول یا سوم فعال می شوند. که در این صورت اگر ورودی به صورت 1011 باشد، یا عدد 4 و یا عدد 6 فعال است. پس ورودی زوج است و در حافظه عدد 2 را در حافظه می نویسیم و اگر به صورت 0111 و یا 1101 باشد، ورودی فرد است و عدد 1 را در حافظه می نویسیم.

```
two_columns_enabled:
    mov al, 010B
    out 80H, al ; outputing the enable two cols
    in al, 82H ; inputing from rows

    cmp al, 11111111B ; if it is all 1s then no button is pressed
    je middle_column_enabled ; the middle column contains the selected button

    cmp al, 11111101B ; check if the second button is selected
    je handling_the_even

    jmp handling_the_odd
```

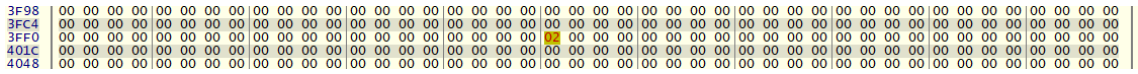
حالت دوم به این صورت است که ستون وسط فعال شود یعنی ورودی ما 101 باشد. پس اگر بعد از فشردن صفحه کلید 0111 و یا 1101 باشد عدد زوج و اگر 1011 باشد عدد فرد است و مطابق بالا در حافظه عدد 2 یا 1 را قرار می دهیم.

```
middle_column_enabled:
        mov al, 101B
        out 80H, al ; outputing the enable middle col

        IN al, 82H ; input from rows


        cmp al, 11111111B ; if it is all 1s then no button is pressed
        je two_columns_enabled ; the other columns contains the selected button

        cmp al, 11111101B ; check if the second button is selected
        je handling_the_odd

        jmp handling_the_even
```

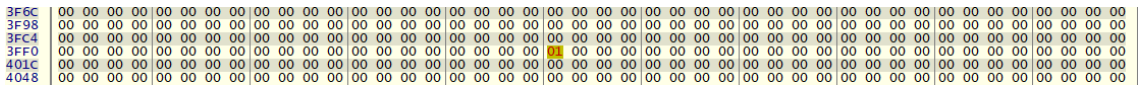
بعد از تعیین زوج یا فرد بودن از توابع زیر برای نوشتن در حافظه استفاده می کنیم :

```
handling_the_odd:
        mov bx, 1h ; if it is odd then 1 is stored
        MOV ds:[8], bx
        RET
handling_the_even:
        mov bx, 2h ; if it is even then 2 is stored
        MOV ds:[8], bx
        RET
```

اگر عدد زوج باشد memory content به صورت زیر است :

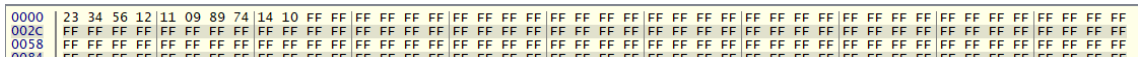


اگر عدد فرد باشد memory content به صورت زیر است :

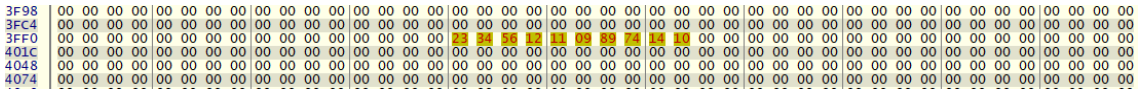


به دلیل active low بودن port ها اگر 1 ورودی باشد port ها غیر فعال هستند و اگر 0 ورودی باشد فعال هستند.

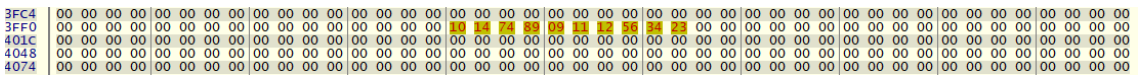
2) برای این قسمت مانند قسمت قبل زوج یا فرد بودن ورودی صفحه کلید را مشخص می کنیم و اگر زوج بود، با استفاده از isEven تابع CPY و اگر فرد بود با استفاده از isOdd تابع INVCPY را فراخوانی می کنیم.
عدد های نوشته شده در ROM این صورت هستند :



اگر عدد زوج وارد شود :

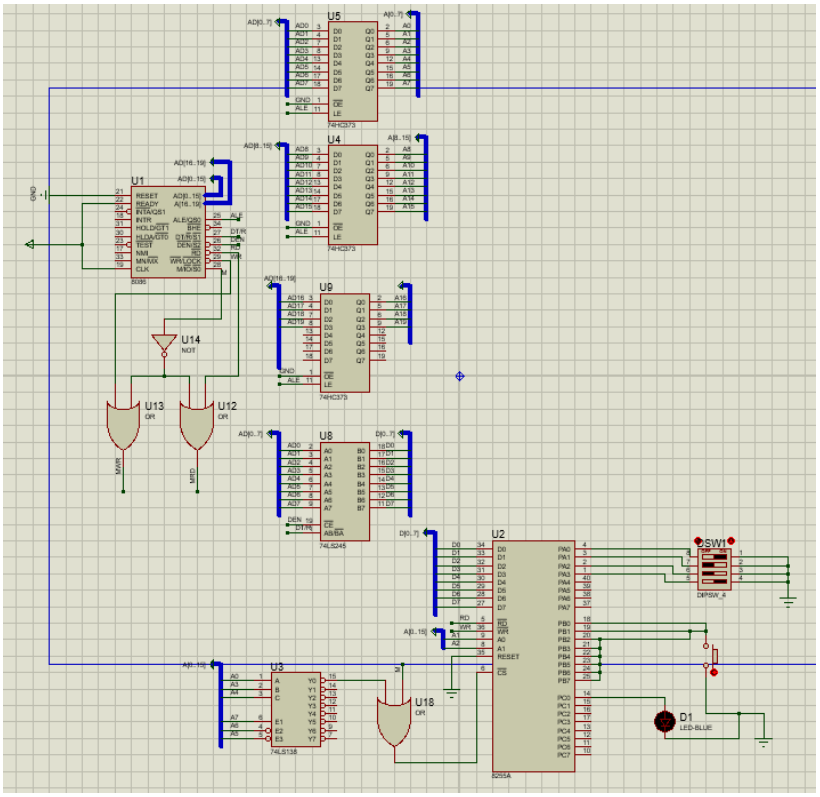


اگر عدد فرد وارد شود :



توجه شود که کد این قسمت و قسمت قبل کاملا شبیه هم است و در یک فایل اسمبلی قرار گرفته اند و برای قسمت 1، بخش isEven و isOdd کامنت شده اند.

3) مدار این سوال به طور کلی به صورت زیر است :



ابتدا برای config عدد 92h را قرار می دهیم که مشخص می کند PA و PB برای ورودی و PC برای خروجی است.

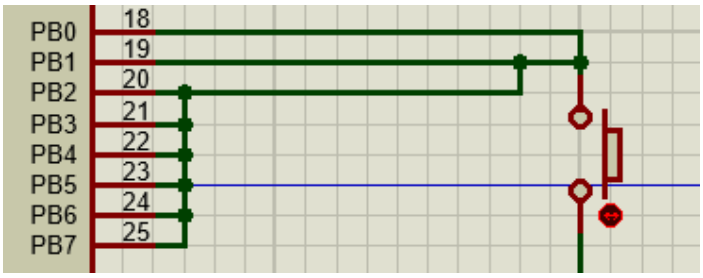
```
mov al, 92h
out 86H, al ; config 8255
```

سپس LED را مطابق کد زیر خاموش می کنیم.

```
mov al, 0feH
out 84H, al ; turn of the LED at first
```

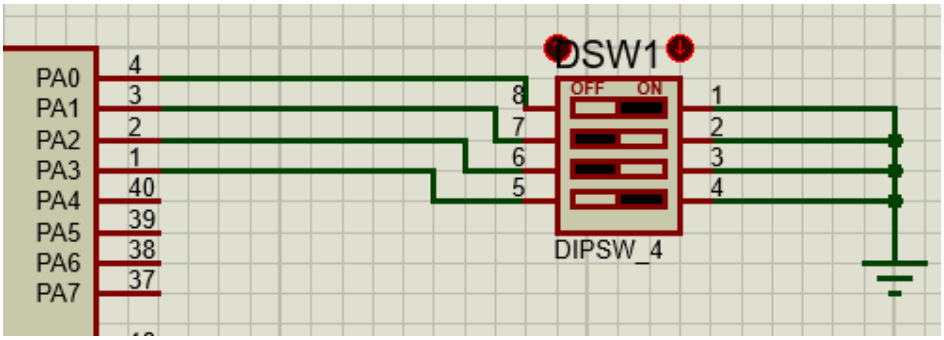
یک کلید در مدار قرار می دهیم و آن را به PB وصل می کنیم و با قرار دادن 82h در al آن را فعال می کنیم و چک می کنیم اگر وصل شد، LED ها را با استفاده از حلقه blink به تعداد دفعات تعیین شده روشن می کنیم وگرنه تا وقتی کلید وصل شود در این حلقه می چرخیم.

```
button_is_pushed:

    in al, 82H
    cmp al, 0 ; if it is 0 then button is pushed and is transfering the 0 voltage from gnd
    jne button_is_pushed
```



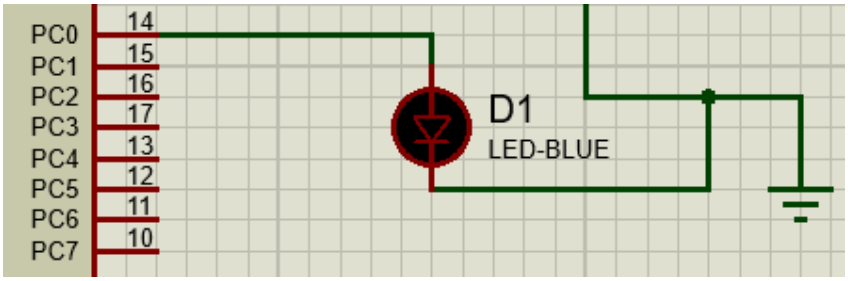
سپس یک dip switch قرار می دهیم و آن را به PA ها وصل می کنیم. از آنجایی که port ها active low هستند، اگر dip switch در یک بیت فعال شود باید بیت ها را invert کنیم. از طریق این dip switch تعداد دفعات روشن شدن LED را مشخص می کنیم.

```
get_n:

    in al, 80H
    mov bl, 11111111B ; input the port A which is connected to dip switch
    XOR al, bl ; the input is active low so we should invert it
    MOV cl, al
    mov ch, 0 ; cx now is the N presented by dip switch

jmp blink
```



یک حلقه blink داریم که در آن PC را با 84h فعال می کنیم و LED را روشن می کنیم و تابع delay را صدا می زنیم.تا یک ثانیه صبر کند. سپس LED را خاموش می کنیم و دوباره یک ثانیه صبر می کنیم. این فرایند به تعداد دفعات تعیین شده در قسمت قبل انجام می شود.

```
blink :


    mov al, 1
    out 84h, al ; turn on the LED

    call delay ; wait 1 second

    mov al, 0
    out 84h, al ; turn off the LED

    call delay ; wait 1 second

    loop blink ; decrement the cx, check if it is not 0 and repeat

jmp button_is_pushed ; job is done
```



تابع delay به صورت زیر است که با توجه به مدل پردازنده 7000h به اندازه یک ثانیه طول می کشد.

```
delay proc near

   ; a simple loop that does nothing but iteration 7000h
   ; times to delay the process by approximatley 1 second

   mov bx, 7000h
   delay_loop:
      sub bx, 1
      cmp bx, 0
      jne delay_loop

    ret

delay endp
```