# گزارش‌کار آزمایش دوم – آزمایشگاه ریزپردازنده

نگار هنرور صدیقیان-99243076

امین احسانی مهر-99243009

## سوالات تحلیلی

1-در زمان ارائه پردازنده 8086 چرا سه باس آدرس، داده ورودی و داده خروجی در پایه‌های تراشه با هم ترکیب شده‌اند؟

-دلیل آن صرفه‌جویی در تعداد پین‌هایی است که روی پردازنده قرار میگیرند،با این کار تعداد چیپ‌ها کاهش یافته و از نظر اقتصادی به صرفه است.

هنگام اتصال به حافظه خارجی چگونه این سه گذرگاه از هم تفکیک میشوند؟

-تفکیک این گذرگاه‌ها به وسیله latch و به این صورت انجام میشود که در سیکل اول آدرس باس جداسازی شده در سیکل دوم داده‌های ورودی خروجی یا همان دیتا باس جداسازی شده و با توجه به read یا write بودن آن به حافظه یا cpu منتقل میشوند.

2-با بررسی دیتا شیت تراشههای 6264 و 62256 و 27128 و 27256 سیگنالهای آدرس، داده و کنترلی هرکدام را مشخص کنید. عملکرد کلی تراشه و وظیفه هرکدام از پایه‌ها را شرح دهید.

-6264: یک رم استاتیک 8 بیتی با ظرفیت 8k*8 است و برای work ram و save ram استفاده میشود. در به کارگیری این نوع از رم رعایت میزان ولتاژ از اهمیت بالایی برخوردار است.

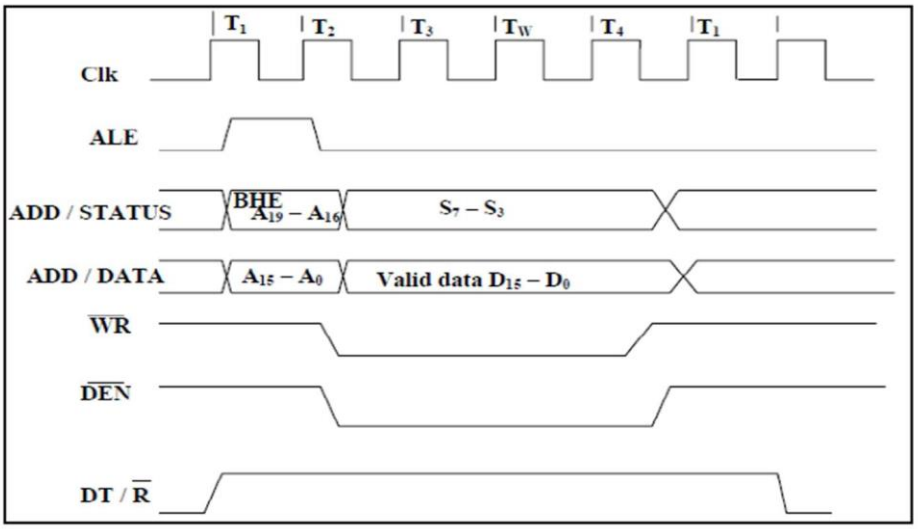
-62256:یک CMOS STATE RAM هشت بیتی بوده که به دلیل داشتن خاصیت خاموش شدن خودکار در صورت عدم نیاز موجب کاهش مصرف انرژی در نتیجه افزایش کارآیی و بهینه بودن شده است.

دارای 8 پین I/O بوده و در صورت low بودن دو سیگنال کنترلی CE و OE دیتا در خانه های حافظه نوشته میشود که در این زمان سیگنال WE به صورتHIGH فعال میشود.

-27128: رام 8 بیتی قابل پاک شدن و نوشتن مجدد یا همان EPROM میباشد. به وسیله نور ماورا بنفش این امکان وجود دارد که بیتهای قبلی نوشته شده روی چیپ پاک شده و مجددا نوشته شود.

-27256: کاملا مشابه تراشه قبلی بوده و تنها تفاوت آن در این است که این تراشه فضای حافظه بیشتری دارد.

3-نمودار زمانی زیر مربوط به عمل خواندن از حافظه پردازنده 8086 میباشد. در یک جدول اتفاقی که در هرکلاک زمانی رخ میدهد را شرح دهید.




| clock | Description |
|---|---|
| T1 | آدرسی که قصد داریم دیتا در آن نوشته شود روی local bus قرار میگیرد |
| T2 | سیگنال ALE در حال LOW شدن است پس نتیجه میگیریم داده در حال قرارگرفتن روی باس است.<br>محتوای local bus که آدرس بوده در خروجی قرار گرفته و local bus خالی میشود.<br>سیگنال WR شروع به high شدن کرده که به معنای فعال شدن برای نوشتن است.<br>پردازنده آدرس مورد نظر که روی local bus بوده را دریافت کرده و دیتا را برای نوشته شدن به این آدرس هدایت میکند. |
| T3 | داده منتقل میشود |
| T4 | پایان فرآیند انتقال داده |

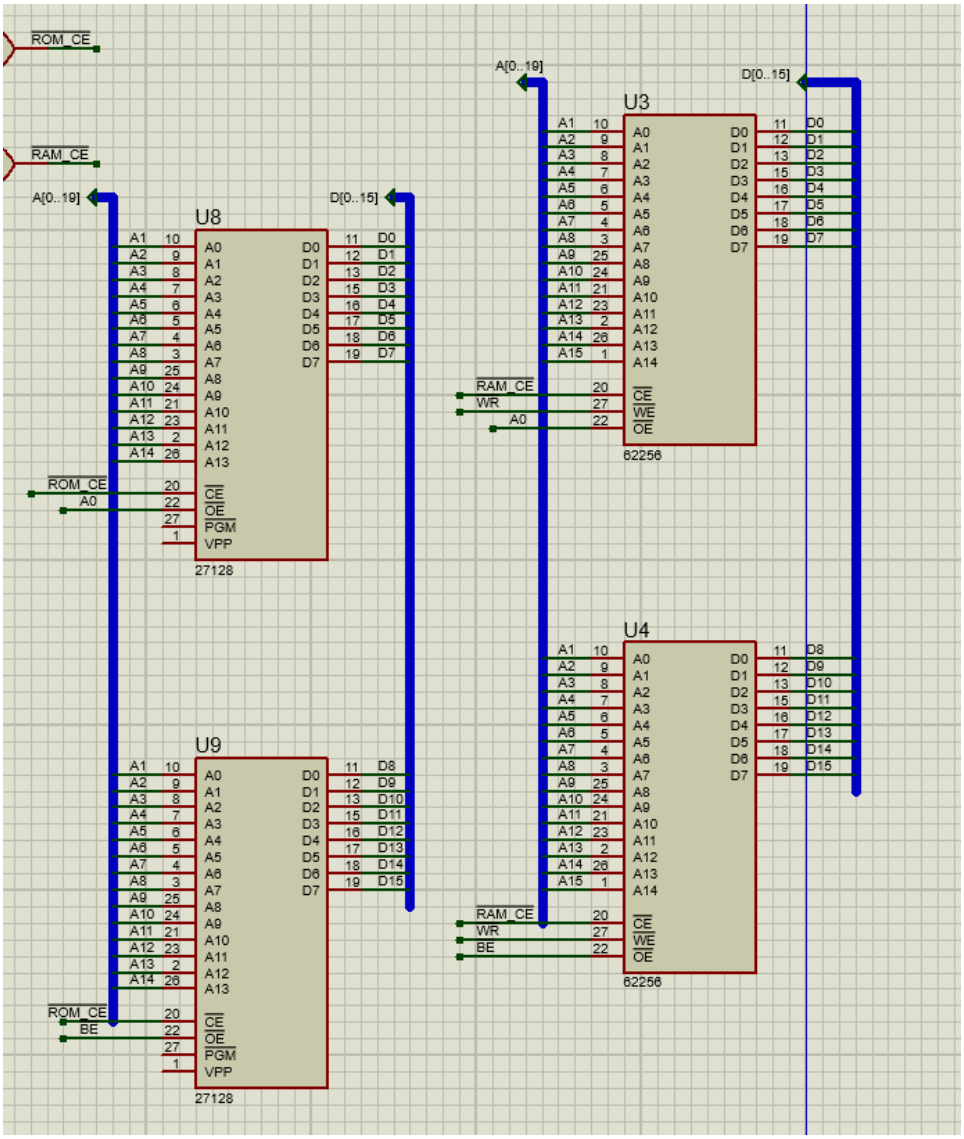
گزارش کد

## بخش A

در این بخش به دنبال پیاده‌سازی یک رام ۶۴ کیلوبایتی به صورت k*16b4 به کمک تراشه EPROM(16k*8) و یک رم ۶۴ کیلوبایتی به صورت k*16b4 به کمک تراشه RAM(32k*8)هستیم.

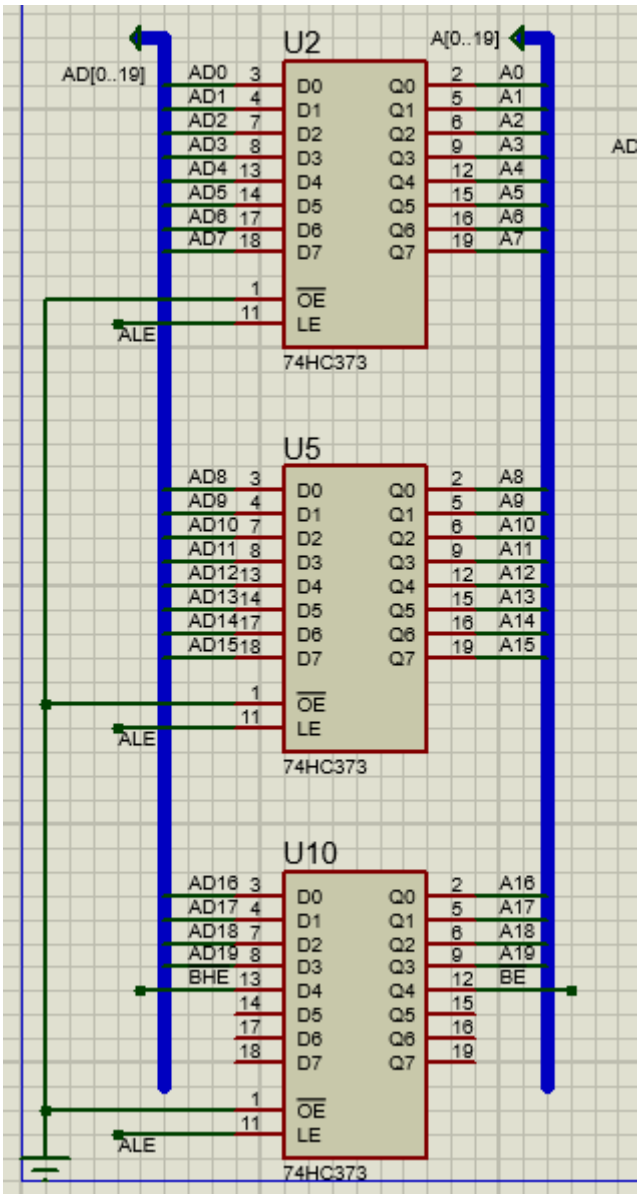
ROM:00000-07FFF

RAM:70000-77FFF

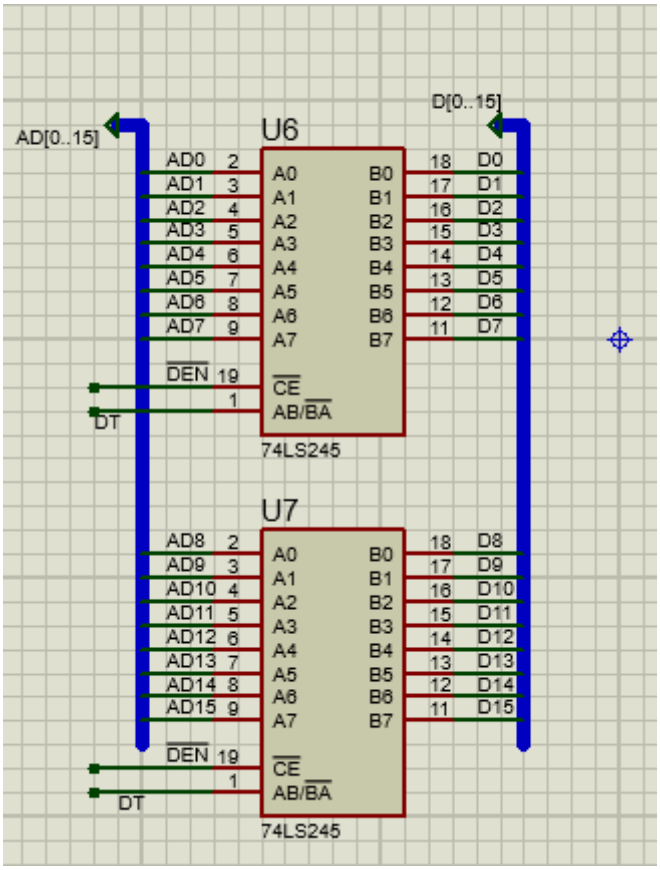



تعدادی data و address bus داریم که از میکروپروسسور ما خارج می‌شوند.ما این bus ها را یکبار به لچ برده تا بررسی کنیم اگر محتوای bus آدرس است ، در صورتی که محتوای bus ما آدرس نباشد هیچ مقداری برای خروجی نخواهیم داد. حال در

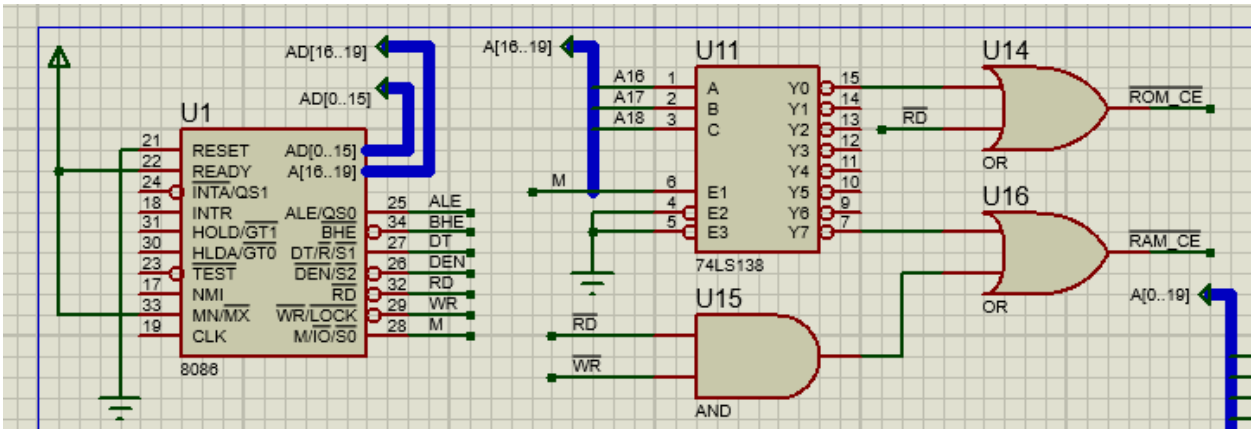
صورتی که محتوای آدرس باشد به کمک سیگنال‌های کنترلی مقدار آدرس را برمیگردانیم؛ توجه کنید که صرفا مقدار آدرس برمیگردد و نه دیتا آن.




این بخش data transceiver نام دارد. در این بخش برعکس کاری که در بخش قبل انجام دادیم را انجام می‌دهیم به این ترتیب که bus را بررسی میکنیم در صورتی که data bus باشد data را خروجی میگیریم . از آن‌جایی که data transceiver یک بافر دوطرفه است در این بخش می‌توانیم دیتا را نیز دریافت کرده و آن را به address bus پردازنده برگردانیم.




بخش بعدی شامل یک دیکودر است. این دیکودر مشخص میکند که در ادامه قصد داریم از حافظه رام بخوانیم یا از حافظه رم.

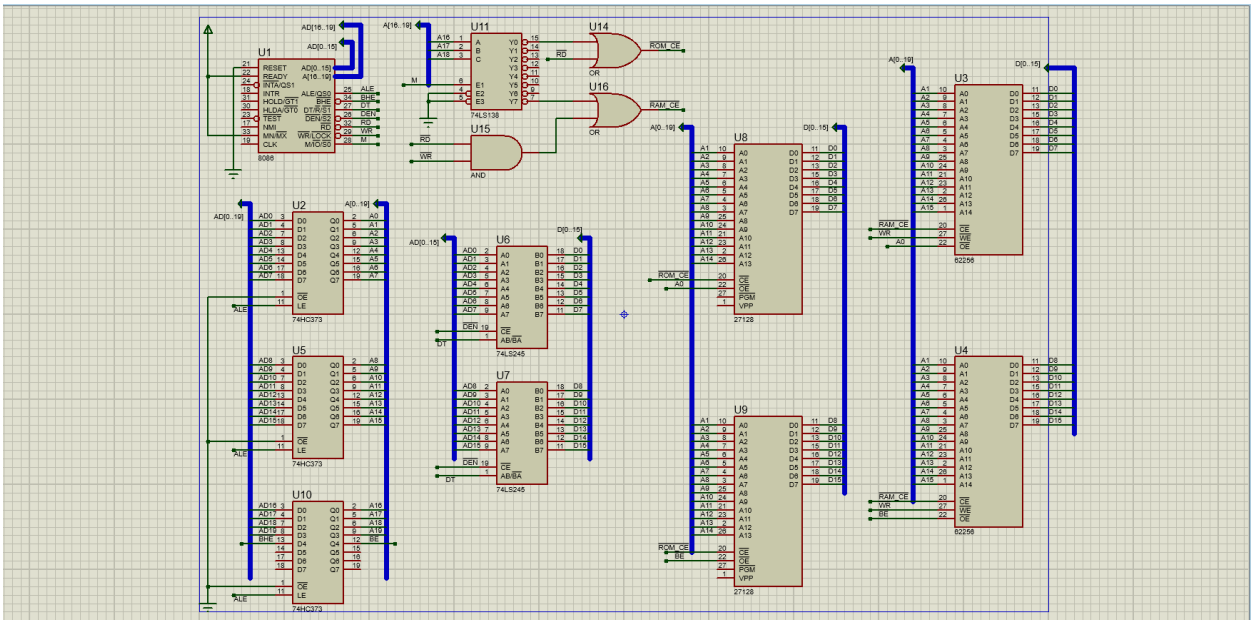



در بخش رام، ابتدا باید یک فایل باینری به آن بدهیم . به کمک این فایل باینری مجموعه‌ای از آدرس‌ها دریافت شده و سپس دیتای مربوط به این آدرس‌ها را برمی‌گرداند.در اینجا به دلیل استفاده از بانک زوج و فرد دو تا رام پیاده سازی شده است.به همین

دلیل ما یک سیگنال ورودی کنترلی به رام به نام outputEnable داریم که این سیگنال مشخص میکند به دنبال خواندن از کدام یک از رام ها هستیم و یا میخواهیم عملیات خواندن روی هردو رام انجام شود.

رم نیز مشابه با رام بوده و دو رم داریم چون از بانک زوج و فرد استفاده می‌کنیم. سیگنال کنترلی outputEnable در اینجا مشابه با سیگنال کنترلی‌ای به همین نام که در رام گفته شد بوده و یک سیگنال کنترلی جدید به نام write داریم که در صورت نیاز به نوشتن در رم فعال می‌شود.

در نهایت شماتیک کلی به شکل فوق است:



**کد این بخش به شرح زیر است:**

در بخش .DATA آدرس شروع رم و رام طبق آنچه در کد نوشته شده مقداردهی شده‌اند.مقدار N که پیشتر آن را توضیح دادیم که تعداد دفعاتی است که از رام خوانده و به رم کپی میکنیم نیز در این بخش مقداردهی شده است و مقدار آن برابر ۱۰ می‌باشد.

```
12  .MODEL SMALL
13  .STACK 64
14
15  .DATA
16  ROM EQU 0000H
17  RAM EQU 7000H
18  N EQU 10
19
```

در ادامه در بخش .CODE و در پراسس CPY در ابتدا مقدار رجیستر SI که قرار است به ما در شمارش تعداد دفعات کپی کردن کمک کند را صفر کرده و در ادامه در اولین خط از لیبل mainloop با مقایسه مقدار رجیستر SI با مقدار N بررسی میکنیم که شرط خاتمه عملیات و در نتیجه حلقه ایجاد شده است یا خیر؛تا در صورت پایان از حلقه به بیرون جامپ کنیم.

```
20
21  .CODE
22
23  CPY PROC
24     MOV SI, 0
25     MAINLOOP:
26        CMP SI, N
27        JZ ENDLOOP
```

در ادامه دیتا سگمت را روی رام ست کرده و آدرسی که SI در خود دارد را از روی رام میخوانیم که به ترتیب اجرا مقادیر 0,1,2,...,10 هستند. در مرحله بعدی دیتا سگمنت را روی رم ست میکنیم تا مقادیری که از رام خوانده شده و در رجیستر BX ذخیره شده است را به رم منتقل کرده و مقدار رجیستر SI را یک‌واحد افزایش میدهیم تا حلقه ادامه یابد.

```
29        ; Set data segment register to point to ROM
30        MOV AX, ROM
31        MOV DS, AX
32
33        MOV BX, [SI]
34
35        ; Set data segment register to point to RAM
36        MOV AX, RAM
37        MOV DS, AX
38
39        MOV [SI], BX
40
41        INC SI
42
43        JMP MAINLOOP
44     ENDLOOP:
45     RET
46  CPY ENDP
```

در پراسس INVCPY دقیقا برعکس پراسس CPY عمل می‌کنیم، یعنی اول مقدار CX که شمارنده حلقه ماست را به اندازه N ست میکنیم و در هر مرحله اجرای حلقه آن را یک واحد کاهش میدهیم؛ سپس در لیبل MAINLOOP مقدار رجیستر SI را به اندازه CX ست میکنیم و دیتا سگمنت را روی رام ست میکنیم ، در اینجا تعداد دفعات خواندن N را به SI منتقل کرده و یک واحد از آن کمک میکنیم و اختلاف آن را با CX پیدا میکنیم تا با کمک حاصل این اختلاف آدرس مورد نظر در رم را پیدا کند، یعنی مثلا داده دوم را در آدرس نهم ذخیره کند.

```
INVCPY PROC
   MOV CX, N
   DEC CX

   MAINLOOP:
      MOV SI, CX

      MOV AX, ROM
      MOV DS, AX

      MOV BL, [SI]

      MOV SI, N
      DEC SI
      SUB SI, CX

      MOV AX, RAM
      MOV DS, AX

      MOV [SI], BL

      LOOP MAINLOOP

   RET
INVCPY ENDP

MAIN PROC FAR
   INFLOOP:
      CALL CPY
      JMP INFLOOP
MAIN ENDP

END MAIN
```