برنامه‌ها چگونه در پردازنده اجرا می‌شوند ؟

در این بخش می‌خواهیم چگونگی اجرای برنامه‌های مختلف در یک پردازنده را به صورت گام به گام بررسی کنیم. مثالهای این بخش عملکرد عمومی پردازنده‌ها را تشریح خواهند کرد.

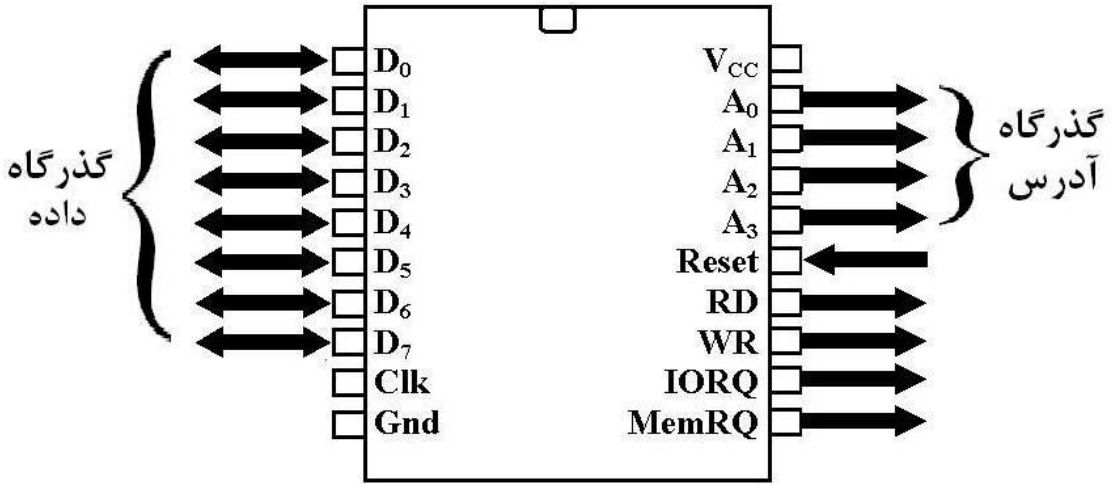
برای طراحی یک سیستم مبتنی بر پردازنده دو کار اساسی باید انجام شود :

۱- **شناخت سخت افزار** که شامل شناخت عملکرد پینهای پردازنده مورد نظر، نحوه عملکرد حافظه‌ها و اتصال آنها به پردازنده و سخت‌افزارهای ورودی/خروجی و ... می‌شود.

۲- **شناخت نرم افزار** که شامل شناخت ساختار داخلی پردازنده به ویژه ثباتهای آن و نیز آشنایی با قواعد زبان اسمبلی پردازنده مورد نظر می باشد.

معرفی یک پردازنده فرضی

برای طراحی مثالهای این بخش از یک پردازنده فرضی استفاده خواهیم کرد که دارای ۴ خط آدرس و ۸ خط داده (شکل ۱-۹)، ثبات انباره A و ثباتهای همه‌منظوره B و C و D است. واضح است که این ثباتها ۸ بیتی هستند (چرا؟).



شکل ۱-۹ - پینهای پردازنده فرضی

در برنامه نویسی به زبان اسمبلی پردازنده فرضی ما چند قاعده باید رعایت شود (این قواعد در زبان اسمبلی اکثر پردازنده ها وجود دارد) :

- یکی از عملوند های عمل جمع، حتماً باید در ثبات A باشد.
- حاصل عمل جمع در ثبات A قرارمی گیرد.

- نمی توان در دستور جمع مستقیماً به حافظه مراجعه کرد.
- تنها رابط پردازنده با خارج آن، ثبات A است.

در ادامه با این نکات برخورد خواهیم کرد.

به طور کلی برای طراحی سیستمها روال زیر را در پیش خواهیم گرفت :

الف) نوشتن برنامه مورد نظر.

ب) بررسی آنچه در پردازنده برای اجرای برنامه انجام می شود.

ج) طراحی سخت افزار سیستم بر اساس گام (ب).

اتصال حافظه به پردازنده

حافظه محل ذخیره برنامه است؛ بنابراین هر سیستمی حتماً دارای یک حافظه است و ضروریست قبل از وارد شدن به جزییات طراحی سیستم، با نحوه ذخیره و بازیابی اطلاعات در حافظه‌ها آشنا شویم.

عملکرد حافظه

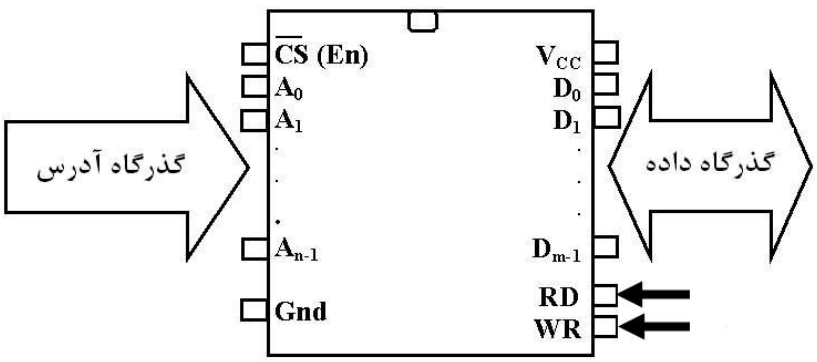
همانطور که قبلاً ذکر شد، حافظه محل ذخیره دستورات و نیز بعضی داده‌های ورودی و خروجی است.

مشخصات مهم یک حافظه عبارتند از :

الف) ظرفیت حافظه: بیانگر حجم اطلاعاتی است که می‌تواند در حافظه ذخیره شود و معمولاً بر حسب بیت یا بایت بیان می‌شود.

ب) ساختار حافظه: حافظه از تعدادی ثبات یکسان (خانه) تشکیل می‌شود که در هر خانه یک قلم داده ذخیره می‌شود. منظور از ساختار حافظه، حجم هر یک از خانه‌ها و تعداد خانه‌هاست.

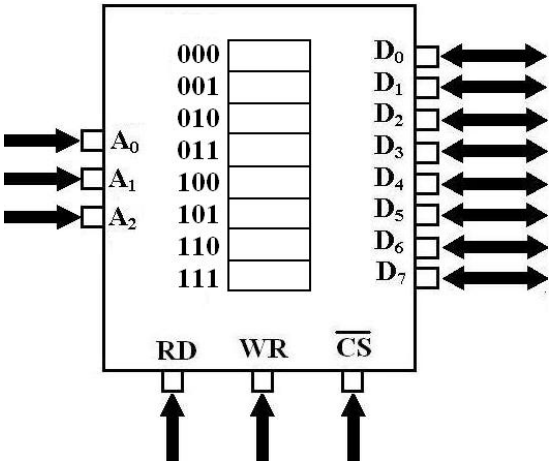
حجم هر خانه حافظه از یک تا ۸ و گاهی ۱۶ بیت معمول است. تعداد خانه‌های یک حافظه نیز معمولاً توانی از ۲ است. شکل کلی یک حافظه را در شکل ۱-۱۰ می‌بینید.



شکل ۱-۱۰- پینهای یک تراشه حافظه

خطوط A_0 تا A_{n-1} که گذرگاه آدرس حافظه نامیده می‌شود، آدرس خانه‌ای از حافظه که باید اطلاعات آن خوانده یا در آن اطلاعاتی نوشته شود را مشخص می‌کند. طول هر قلم اطلاعاتی (حجم هر خانه) که اطلاعات آن از طریق خطوط داده D0 تا D_{m-1} رد و بدل می‌شود، m بیت است.

پینهای RD و WR برای تعیین نحوه تبادل اطلاعات با حافظه (خواندن یا نوشتن) به‌کارمی‌روند. اگر سیگنال RD را فعال کنیم یعنی می‌خواهیم از حافظه اطلاعاتی بخوانیم. با فعال کردن سیگنال WR به حافظه اعلام می‌کنیم که اطلاعاتی در یکی از خانه‌هایش نوشته می‌شود. در هر دو عمل خواندن و نوشتن، تبادل اطلاعات با خانه‌ای از حافظه که شماره (آدرس) آن روی گذرگاه آدرس حافظه گذاشته می‌شود انجام می‌گیرد.



به عنوان مثال حافظه نوعی روبرو را در نظر بگیرید. این حافظه ۸ خط داده (D_0 تا D_7) و ۳ خط آدرس (A_0 تا A_2) دارد؛ چون در این حافظه می‌توان 2^3 خانه حافظه را آدرس‌دهی کرد (با ۳ بیت می‌توان 2^3 عدد دودویی مختلف را مشخص کرد) و ظرفیت هر خانه نیز ۸ بیت است (چون ۸ پین برای تبادل اطلاعات با خارج دارد)، پس این حافظه دارای ۸ (2^3) خانه یک بایتی (۸ بیتی) است. پس ظرفیت این تراشه ۸ بایت یا ۶۴ بیت است.

به عنوان یک مثال عملی، تراشه 2732 ، ۸ خط داده (D_0 تا D_7) و ۱۲ خط آدرس (A_0 تا A_{11}) دارد؛ چون در این حافظه می‌توان 2^{12} خانه حافظه را آدرس‌دهی کرد (با ۱۲ بیت می‌توان 2^{12} عدد دودویی مختلف را مشخص کرد)، پس این حافظه دارای ۱۰۲۴ × ۴ = $2^{10} \times 2^2$ (یا ۴ کیلو) خانه یک بایتی (۸ بیتی) است. پس ظرفیت این تراشه ۴ کیلوبایت یا ۳۲ کیلوبیت (۸ × ۱۰۲۴ × ۴) است که دو رقم آخر شماره آن نیز این مطلب را نشان می‌دهد.

پرسش : ظرفیت حافظه شکل ۱-۱۰ بر حسب بیت چقدر است؟

ذخیره و بازیابی اطلاعات در حافظه

اکنون فرض کنید می‌خواهیم در خانه شماره ۷ حافظه 2732، عدد دودویی ۸ بیتی **10110001** را ذخیره کنیم. کافی است شماره خانه فوق یعنی معادل دودویی ۱۲ بیتی

عدد ۷ (000000000111) را روی پینهای A_0 تا A_{11} و عدد مورد نظر (**10110001**) را روی پینهای D_0 تا D_7 تراشه قرار داده و پین WR (Write) را فعال کنیم تا عدد فوق در خانه شماره ۷ ذخیره شود. اگر این حافظه از نوع RAM باشد تا زمان قطع برق و اگر از نوع ROM باشد به طور دائم این داده در این خانه باقی خواهد ماند.

برای خواندن محتویات یک خانه مثلاً همان خانه شماره ۷ ، کافی است عدد ۱۲ بیتی معادل ۷ یعنی 000000000111 را روی پینهای آدرس حافظه قرار داده و پین RD (Read) را فعال کنیم. عدد ذخیره شده در خانه ۷ که در اینجا 10110001 است روی پینهای D_0 تا D_7 (گذرگاه داده حافظه) قرار می‌گیرد.

توجه کنید که چه برای نوشتن در حافظه و چه برای خواندن از آن، پین فعال ساز تراشه (CS یا En) باید فعال باشد.

مراحل خواندن و نوشتن در حافظه را می‌توان به صورت زیر خلاصه کرد؛

برای نوشتن عدد a در آدرس b حافظه:

- آدرس b را روی پینهای آدرس حافظه (گذرگاه آدرس حافظه) قرار می‌دهیم.
- عدد a را روی پینهای داده حافظه (گذرگاه داده حافظه) قرار می‌دهیم.
- پین WR حافظه را فعال می‌کنیم تا عدد a در آدرس b حافظه ثبت شود.

برای خواندن محتویات خانه حافظه به آدرس x:

- آدرس x را روی پینهای آدرس حافظه (گذرگاه آدرس حافظه) قرار می‌دهیم.
- پین RD حافظه را فعال می‌کنیم تا محتویات خانه حافظه به آدرس x روی پینهای داده حافظه (گذرگاه داده حافظه) قرار گیرد.
- پینهای داده حافظه (گذرگاه داده حافظه) را می‌خوانیم.

پرسش) راجع به دو مفهوم گسترش طول کلمه و گسترش طول آدرس حافظه تحقیق کنید.

ج) زمان دسترسی : مشخصه سرعت حافظه و عبارتست از زمانی که بعد از گذاشتن آدرس خانه مورد نظر روی خطوط آدرس و فعال کردن پین RD به طول می‌انجامد تا داده خانه فوق روی خطوط داده تراشه حافظه به صورت پایدار ظاهر شود.

اکنون که با نحوه خواندن و نوشتن در حافظه آشنا شدیم، به مثالهایی از طراحی سیستم می‌پردازیم. اولین مثال راجع به یک سیستم بسیار ساده است.

مثال ۱) اولین کاری که می‌خواهیم پردازنده فرضی ما انجام دهد این است که مقدار ثباتهای B و C خودش را با هم جمع کند و در ثبات D قرار دهد.

یادآوری می‌کنیم برای طراحی سیستمها به ترتیب زیر عمل می‌کنیم:
الف) نوشتن برنامه مورد نظر.
ب) بررسی آنچه در پردازنده برای اجرای برنامه انجام می شود.
ج) طراحی سخت افزار سیستم بر اساس گام (ب).
به عنوان گام اول طراحی این سیستم، برنامه آن را می نویسیم.
این برنامه می تواند به سادگی یک خط برنامه باشد :

D ⟵ B + C

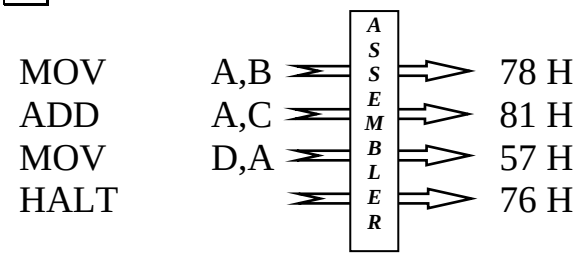
اما طبق قواعد (الف) و (ب) که مطرح شد، نمی‌توان جمع دو ثبات را به‌طور مستقیم انجام داد؛ یعنی دستور فوق نامعتبر است و با توجه به اینکه ثبات A ،انباره یعنی برگه موقت کار است، مراحل زیر باید انجام شود :

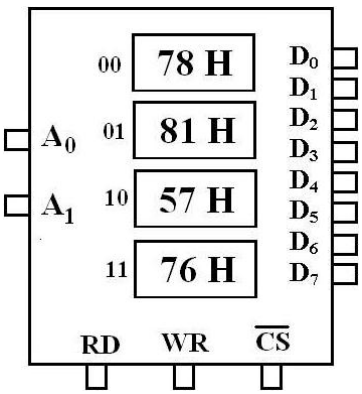
A ⟵ B
A ⟵ A + C
D ⟵ A

مراحل ذکر شده باید به زبانی که پردازنده متوجه شود، یعنی زبان صفر و یک یا همان زبان ماشین به پردازنده دیکته شود. در عمل برنامه هیچگاه مگر در موارد خاص به زبان ماشین نوشته نمی‌شود و این کار در پایین‌ترین سطح با زبان اسمبلی انجام می‌شود. معادل اسمبلی فرضی دستورات بالا به صورت زیر است (مشابه این دستورات در زبان اسمبلی پردازنده‌های مختلف وجود دارد) :

```
MOV     A,B
ADD     A,C
MOV     D,A
HALT
```

قبلاً گفته شد که زبان اسمبلی شباهتهایی به زبان انگلیسی دارد؛ مثلاً دستور MOV A,B یعنی **MOV**e **B** to **A** . همچنین دستور HALT به معنای **قطع** اجرای برنامه است.
این دستورات با کمک اسمبلر پردازنده ما به زبان ماشین تبدیل می‌شوند. کدهای زبان ماشین معمولاً در قالب هگزادسیمال نشان داده می‌شوند.





پس از تبدیل دستورات برنامه به زبان ماشین، آنها را در یک تراشه حافظه ذخیره می‌کنیم (شکل بعد را ببینید) تا پردازنده یکی یکی آنها را واکشی و اجرا کند؛ مثلاً وقتی پردازنده کد یک بایتی 78 H (یا 01111000 B) را از حافظه دریافت می‌کند، با رمزگشایی آن متوجه می‌شود که باید محتویات ثبات B را در ثبات A قرار دهد و بهمین ترتیب بقیه دستورات را اجرا می‌کند تا به کد 76 H (کد HALT) برسد. این کد به پردازنده می‌گوید که اجرای برنامه به پایان رسیده است. به کدهای فوق که پردازنده را به اجرای عملیات‌های گوناگون وامی‌دارند، کد عمل[1] گفته می‌شود. چون برنامه فوق چهار بایتی است (تعداد کدهای عمل آن ۴ بایت است)، از یک حافظه ۴ بایتی که دو خط آدرس دارد استفاده می‌کنیم.

طراحی سخت افزار

اکنون گام (الف) به پایان رسیده است؛ برنامه (نرم‌افزار) سیستم آماده است و نوبت به طراحی سخت‌افزار می‌رسد. در این گام باید عملکرد پردازنده برای اجرای این برنامه را بررسی کنیم تا در گام بعد بتوانیم بر اساس این دانسته‌ها، سخت افزار نهایی را طراحی کنیم.

سخت‌افزار سیستم فوق بسیار ساده است و فقط باید مکانیسمی در آن پیش‌بینی شود که برنامه زبان ماشین را که در حافظه ذخیره شده، به موقع به پردازنده برساند. برای طراحی این مکانیسم باید با بررسی نحوه اجرای برنامه (گام **ب**) ببینیم پردازنده با چه روشی بایتهای محتوی برنامه را از حافظه درخواست می‌کند؟

پردازنده چگونه کدهای عمل را از حافظه می‌خواند؟

همانطور که قبلاً گفته شد، در پردازنده‌ها ثباتی به نام شمارنده برنامه یا PC وجود دارد که وظیفه آن، نگهداری شماره دستوری است که قرار است اجرا شود. این شماره دستور در بسیاری از پردازنده‌ها ازجمله پردازنده فرضی ما، شماره خانه‌ای از حافظه است که دستور

[1] ***Operational Code : OP-Code***

بعدی را در خود نگه می‌دارد[۱]. چون در پردازنده مورد بررسی ما گذرگاه آدرس ۴ بیتی است، بنابراین تمام شماره‌ها (از جمله شماره دستور بعدی که در ثبات PC ذخیره می‌شود) نیز ۴ بیتی است؛ بنابراین ثبات PC در این پردازنده یک ثبات چهار بیتی است.
اولین اصل طراحی سیستم در مورد نحوه درخواست کدهای عمل از حافظه توسط پردازنده است:

اصل اول طراحی سیستم: پردازنده برای درخواست یک کد عمل از حافظه، مقدار ثبات PC را روی خطوط آدرسش گذاشته و سیگنال RD خود را فعال می کند؛ در این لحظه پردازنده انتظار دارد که کد عمل مورد نظر که همان دادهٔ ذخیره شده در PCامین خانه حافظه است، روی گذرگاه داده اش بیاید تا بتواند آن را بخواند و اجرا کند؛ وظیفه طراح سیستم برآوردن این انتظار پردازنده است.

فرض می‌کنیم در پردازنده فرضی ما، بعد از روشن شدن سیستم، PC برابر صفر می‌شود[۲].
بنابراین پردازنده به خانه صفر حافظه مراجعه و اولین دستور را از آنجا درخواست می‌کند.
این درخواست توسط پردازنده با قرار دادن آدرس 0000 (یعنی مقدار PC) روی گذرگاه آدرس و فعال کردن سیگنال خواندن (RD) و سپس خواندن گذرگاه داده به هدف خواندن اولین کد عمل برنامه انجام می‌شود.
پس از خواندن اولین دستور برنامه (78 H) از حافظه و اجرای آن، PC به طور خودکار یک واحد افزایش می‌یابد و برای خواندن خانه بعدی از حافظه به کار می‌رود؛ به این صورت که پردازنده محتوای PC یعنی 0001 را روی گذرگاه آدرس قرار داده و پس از فعال کردن سیگنال RD ، گذرگاه داده را که انتظار دارد محتوی دستور بعدی (81 H) باشد می‌خواند.
کد HALT (76 H) به پردازنده می‌گوید که باید افزایش PC و درخواست دستور بعدی از حافظه را متوقف کند.

در زیر مراحل اجرای این برنامه را می بینید :

مقداری که در این لحظه باید روی **گذرگاه داده** پردازنده ظاهر شود	عددی که پردازنده روی **گذرگاه آدرس** خود قرار می‌دهد	**PC**
78 H (کد عمل صفر)	0000	0000

---- اجرای کد 78 h ----

[۱] بعدها خواهید دید که در پردازنده‌های سری 80xx اینتل دقیقاً این‌گونه نیست.
[۲] مقدار هر ثبات بعد از روشن شدن سیستم، یکی از مشخصات خاص پردازنده‌هاست که *Register Wake up Value* نام دارد.

0001 0001 81 H (کد عمل یک)

---- اجرای کد 81 h ----

0010 0010 57 H (کد عمل دو)

---- اجرای کد 57 h ----

0011 0011 76 H (کد عمل سه)

---- اجرای کد 76 h ----

با اجرای کد 76h (دستور HALT) پردازنده دیگر به مقدار PC اضافه نمی‌کند تا دستور بعدی را بخواند؛ بنابراین اجرای برنامه قطع می‌شود.

بنا بر آنچه گفته شد، مکانیسم پردازنده برای دریافت دستورات از حافظه، قرار دادن PC روی گذرگاه آدرس و فعال کردن سیگنال RD است؛ پس باید سیستمی طراحی شود که وقتی پردازنده اعمال فوق را انجام داد، در پاسخ آن بایت ذخیره شده در PCامین خانه حافظه که در واقع کد عمل دستوری است که باید اجرا شود را روی گذرگاه داده پردازنده قرار دهد. به بیان دیگر، باید اتصالات بین پردازنده و حافظه را به نحوی برقرار کنیم که وقتی پردازنده یک آدرس را روی گذرگاه آدرس گذاشته و سیگنال RD را فعال می کند، بایت مورد نظر از حافظه به گذرگاه داده پردازنده متقل شود.

با گذرگاههای داده چه باید کرد؟ کدهای عمل باید از طریق گذرگاه داده پردازنده به آن وارد شوند. از طرف دیگر حافظه کدهای عمل را از طریق گذرگاه داده‌اش به خارج ارسال می‌کند. بنابراین طبیعی است که:

گذرگاه داده پردازنده بیت به بیت (پین به پین) به گذرگاه داده حافظه متصل می‌شود.

این موضوع با گفته قبلی ما که در سیستم یک گذرگاه داده بیشتر وجود ندارد مطابقت دارد. توجه کنید اگر هر دو جزء سیستم برای ارتباط با هم بخواهند گذرگاههای داده مجزا داشته باشند، سیستم ما مملو از سیمهای ارتباطی می‌شود که هزینه و خطاپذیری سیستم را بسیار افزایش می‌دهد.

برای دانستن چگونگی اتصال گذرگاههای آدرس پردازنده و حافظه به هم، به مراحل اجرای برنامه به ویژه آنچه روی گذرگاه آدرس پردازنده می‌آید، توجه کنید. اولین نکته‌ای که جلب

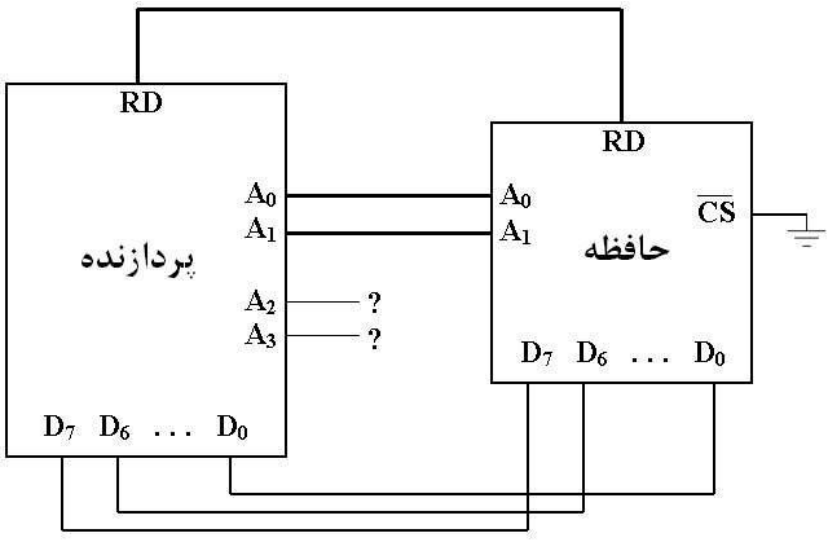
توجه می‌کند این است که خطوط آدرس A_2 و A_3 در طول اجرای برنامه صفر هستند و تغییر نمی‌کنند. بنابراین دقت خود را به بیتهای A_0 و A_1 معطوف می‌کنیم.

اکنون روند واکشی دستور اول را ببینید؛ هنگامی که پردازنده روی خطوط A_0 و A_1 عدد 00 را گذاشته و سیگنال RD خود را فعال می‌کند، انتظار دارد کد 78h روی گذرگاه داده‌اش بیاید. از طرف دیگر وقتی روی خطوط آدرس A_0 و A_1 حافظه عدد 00 گذاشته شده و سیگنال RD آن فعال شود، حافظه محتویات خانه صفر خود (78h) را روی گذرگاه داده‌اش می‌گذارد.

واکشی دستور دوم نیز همین روند را دنبال می‌کند؛ هنگامی که پردازنده روی خطوط A_0 و A_1 عدد 01 را گذاشته و سیگنال RD خود را فعال می‌کند، انتظار دارد کد 81h روی گذرگاه داده‌اش بیاید. از طرف دیگر وقتی روی خطوط آدرس A_0 و A_1 حافظه عدد 01 گذاشته شده و سیگنال RD آن فعال شود، حافظه محتویات خانه صفر خود (81h) را روی گذرگاه داده‌اش می‌گذارد.

بررسی آنچه گفته شد نشان می‌دهد اتصالات لازم برای این سیستم به شرح زیر است:

- اتصال بیت به بیت گذرگاه داده پردازنده به گذرگاه داده حافظه
- اتصال خطوط A_0 و A_1 پردازنده به خطوط A_0 و A_1 حافظه
- اتصال سیگنال RD حافظه به سیگنال RD پردازنده



شکل ۱-۱۲- سخت افزار مثال ۱

به شکل ۱-۱۲ و نحوه عملکرد سیستم توجه کنید. در ابتدا پردازنده مقدار PC یعنی 0000 را روی خطوط آدرس خود یعنی A_0 تا A_3 می‌گذارد؛ اما تنها خطوط A_0 و A_1 به حافظه متصل هستند که مقدار آنها 00 است. سپس پردازنده برای خواندن کد عمل، پین RD

خود که به پین RD حافظه متصل است را فعال می‌کند. با فعال شدن پین RD حافظه توسط پردازنده، محتویات خانه 00 حافظه (که قبلاً مقدار 78 H در آن ذخیره شده) روی گذرگاه داده حافظه که به گذرگاه داده پردازنده متصل است، قرار می‌گیرد و پردازنده آن را می‌خواند. چون پردازنده می‌داند که اولین بایتی که از حافظه می‌خواند، یک کد عمل است، این بایت را مستقیماً به ثبات دستورالعمل ارسال می‌کند. با افزایش PC ، بایتهای بعدی برنامه که در حافظه ذخیره شده‌اند، خوانده و سپس اجرا می‌شوند.

چون خطوط A_2 و A_3 پردازنده تا پایان اجرای برنامه صفر هستند، تنها اتصال خطوط پایینی آدرس پردازنده یعنی A_0 و A_1 به حافظه کافی است. با خطوط A_2 و A_3 چه باید کرد ؟ فعلاً نیازی به اتصال آنها نیست؛ اما بعداً برای تکمیل کار به آنها نیاز خواهیم داشت.

پرسش) حافظه مورد استفاده در این مثال ۴ بایت ظرفیت دارد؛ اما حتی اگر ظرفیت آن بیشتر هم بود، باز برنامه را با شروع از خانه صفر حافظه ذخیره می کردیم. چرا؟

پرسش) اگر کد HALT در انتهای برنامه استفاده نشود، چه می شود؟

زمانبندی سیستم

ذکر یک نکته ضروری به نظر می رسد؛ وقتی پردازنده می خواهد یک دستور را از حافظه بخواند، شماره آن دستور را روی گذرگاه آدرس خود گذاشته و سیگنال RD را فعال می‌کند و سپس گذرگاه داده خود را می خواند. پردازنده در این لحظه **انتظار دارد** که محتوی PC-امین خانه حافظه روی گذرگاه داده آن آمده باشد؛ در حالی که اگر حافظه سیستم کندتر از پردازنده باشد (زمان دسترسی حافظه طولانی باشد که معمولاً چنین است)، عمل خواندن PC-امین خانه حافظه به طول می انجامد و ممکن است باعث شود اطلاعات خوانده شده از گذرگاه داده توسط پردازنده معتبر نباشد. بعدها راجع به چگونگی لحاظ کردن مسایل زمانبندی در طراحی یک سیستم بحث خواهیم کرد.

مثال ۲) هر بار که برنامه مثال ۱ اجرا شود، یک عدد ثابت در ثبات D ذخیره می‌شود. چون در ابتدای کار B و C همیشه یک مقدار دارند (Wake up Value).

پس برنامه را به نحوی اصلاح می‌کنیم که B و C ابتدا با مقادیر دلخواه (مثلاً ۴۲ و ۶۸) مقداردهی و سپس با هم جمع شوند. برنامه مورد نظر به همراه کدهای عمل آن در زیر آمده است.

MOV	B,42	ASSEMBLER	06 h	2A h
MOV	C,68		0E h	44 h
MOV	A,B		78 h	
ADD	A,C		81 h	
MOV	D,A		57 h	
HALT			76 h	

توجه کنید که در زبان ماشین معمولاً کدها در قالب هگزادسیمال نشان داده می‌شوند؛ به‌همین دلیل عدد ۴۲ به صورت 2A H نشان داده شده است.

می‌بینید که کد عمل بار کردن یک مقدار ثابت در ثبات B ، دو بایتی است که بایت اول (06 H) که کد عمل اصلی نام دارد، نشان می‌دهد که بایت بعدی (2A H) باید در ثبات B ذخیره شود. همچنین توجه کنید که کد بارگذاری ثبات B با مقدار ثابت (06 H) با کد بارگذاری ثبات C با مقدار ثابت (0E H) متفاوت است.

این برنامه ۸ بایت کد عمل دارد که باید در حافظه ذخیره شود؛ بنابراین از یک حافظه ۸ بایتی استفاده می‌کنیم که ۳ خط آدرس دارد.

روند اجرای این برنامه با برنامه قبل تفاوت مهمی دارد؛ در برنامه قبلی هنگامی که کد 78h از حافظه خوانده می‌شود، پردازنده آن را اجرا می‌کند (محتویات ثبات B را به ثبات A منتقل می‌کند). اما در این برنامه وقتی پردازنده کد 06 را از حافظه می‌خواند، متوجه می‌شود باید عدد ثابتی را در ثبات B کپی کند؛ این عدد چند است؟! چون پردازنده هنوز نمی‌داند این عدد ثابت چقدر است، باید کد عمل بعدی که این عدد را نشان می‌دهد، از حافظه بخواند. بنابراین اجرای این دستور تا خواندن کد عمل بعدی به تأخیر می‌افتد.

پرسش) کد عمل دستور MOV B, 43 و MOV C, 29 را بنویسید.

قسمتی از روند اجرای این برنامه در ذیل دیده می‌شود.

PC	عددی که پردازنده روی **گذرگاه آدرس** خود قرار می‌دهد	مقداری که در این لحظه باید روی **گذرگاه داده** پردازنده ظاهر شود
0000	0000	06 H (کد عمل صفر)
0001	0001	2A H (کد عمل یک)

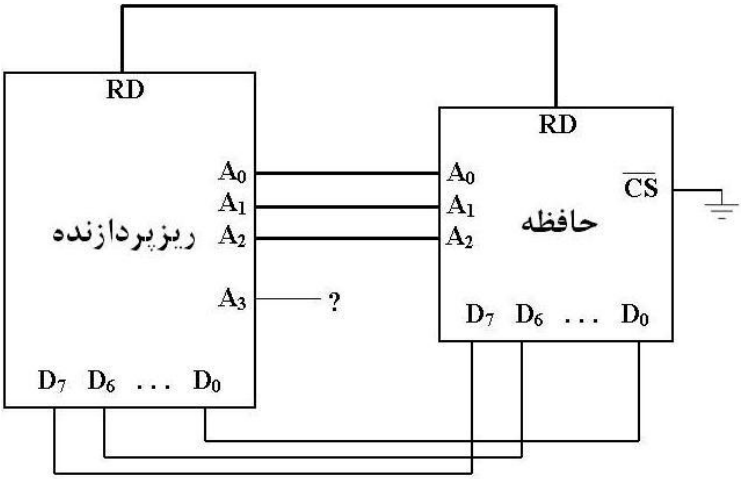
---- اجرای کد 2A h 06 h ----

0010	0010	0E H (کد عمل دو)
0011	0011	44 H (کد عمل سه)

---- اجرای کد 44 h 0E h ----

....

پرسش) روند اجرای برنامه را تکمیل کنید و به کمک آن نشان دهید سخت‌افزار سیستم به شکل زیر است:



در سخت‌افزار مثال اول که حافظه دو خط آدرس A_0 و A_1 دارد، این دو خط به خطوط پایین آدرس پردازنده (A_0 و A_1) متصل شده‌اند. در سخت‌افزار مثال دوم که حافظه سه خط آدرس A_0 و A_1 و A_2 دارد، باز این خطوط به خطهای پایین آدرس پردازنده (A_0 و A_1 و A_2) متصل شده‌اند. این یک قانون کلی است:

خطوط آدرس حافظه به خطوط پایین آدرس پردازنده متصل می‌شوند.

دلیل این موضوع ساده است؛ روند تغییرات خطوط پایین آدرس پردازنده (A_1A_0 در مثال ۱ و $A_2A_1A_0$ در مثال ۲) با روند تغییرات خطوط آدرس حافظه یکسان است. مثلاً در مثال ۲ پردازنده برای خواندن ۸ کد عمل ابتدای برنامه از آدرس 000 تا آدرس 111 را روی خطوط آدرس $A_2A_1A_0$ قرار می‌دهد. از سوی دیگر، برای خواندن ۸ کد عملی که در حافظه ذخیره شده است، باید از آدرس 000 تا آدرس 111 روی خطوط آدرس حافظه قرار گیرد. بنابراین برای هماهنگ شدن عملکرد حافظه با پردازنده، باید خطوط آدرس حافظه به خطوط پایین آدرس پردازنده متصل شوند.

با خطوط بالای آدرس پردازنده چه باید کرد؟ در مثالهای بعدی این موضوع را بررسی خواهیم کرد.

بنابراین تاکنون از مثالهای اول و دوم، قوانینی برای طراحی سیستم یافته‌ایم که در مثالهای بعدی بدون بحث از آنها استفاده می‌کنیم:

گذرگاه داده پردازنده بیت به بیت (پین به پین) به گذرگاه داده حافظه متصل می‌شود.
خطوط آدرس حافظه به خطوط پایین آدرس پردازنده متصل می‌شوند.
سیگنال RD پردازنده باید به سیگنال RD حافظه متصل شود.

مثال ۳) این مثال ما را با نحوه خواندن و نوشتن در حافظه آشنا می‌کند. برنامه‌های مثال ۱ و ۲ به درستی اجرا می‌شوند، اما نمی‌توانیم نتیجه اجرا را مشاهده کنیم؛ چون نتیجه اجرا در ثبات D که یکی از ثباتهای داخلی پردازنده است ذخیره می‌شود. به علاوه اعداد ورودی را نمی‌توان جز با تغییر برنامه، تغییر داد. در حالی که سیستمی مطلوب است که بتوان با تغییر ورودیها، خروجیهای مختلفی از آن گرفت. برنامه مثال ۲ را به نحوی اصلاح می‌کنیم که محتویات خانه‌های شماره ۱۳ و ۱۴ حافظه را جمع و حاصل را در خانه شماره ۱۵ حافظه ذخیره کند:

```
MOV     A,[13]   ASSEMBLER   3A h  0D h
MOV     B,A                  47 h
MOV     A,[14]               3A h  0E h
ADD     A,B                  80 h
MOV     [15],A               32 h  0F h
HALT                         76 h
```

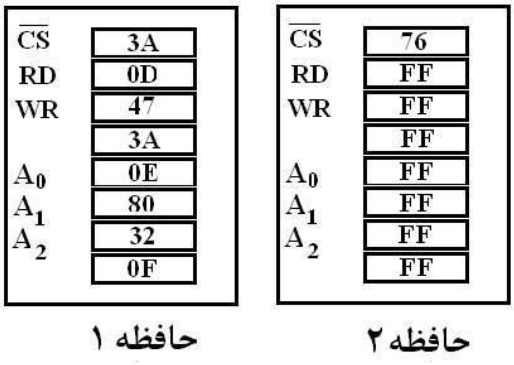
توجه کنید که دستور MOV A, 13 عدد ۱۳ را در ثبات A ذخیره می‌کند؛ در حالی که دستور MOV A, [13]، عددی که در خانه شماره ۱۳ حافظه ذخیره شده را در ثبات A قرار می‌دهد. با مقایسه این برنامه با برنامه قبلی واضح است کد عمل این دو دستور نیز با هم متفاوت است. این موضوع یکی از قوانین معمول زبانهای اسمبلی است[۱].

کد عمل دستور **MOV A,[13]** دو بایتی است که بایت اول آن (کد عمل اصلی که 3A h است) نشان می‌دهد عددی که در خانه حافظه‌ای به شماره کد عمل دوم (0D H یا ۱۳دهدهی) قرار دارد باید در ثبات A ذخیره شود. همچنین توجه کنید که مطابق آنچه در قواعد زبان اسمبلی پردازنده فرضی ما گفته شد، تنها یک ثبات (معمولاً ثبات انباره) می‌تواند برای تبادل اطلاعات با حافظه مورد استفاده قرار گیرد که در پردازنده فرضی ما ثبات A است؛ بنابراین دستور MOV B,[13] نامعتبراست و به جای آن باید از دو دستور اول برنامه استفاده کرد.

[۱] در بعضی زبانهای اسمبلی (مانند اسمبلی پردازنده Z80)، به جای علامت [ ] از علامت ( ) استفاده می‌شود.

پرسش) چرا کد عمل اصلی دستورات اول و پنجم با هم متفاوت است؟ چرا کد عمل اصلی دستورات اول و سوم مشابه است؟

تعداد کدهای عمل، ۹ بایت است که در یک حافظه ۸ بایتی جا نمی‌گیرد. بنابراین باید از یک حافظه ۱۶ بایتی با چهار خط آدرس استفاده کنیم. فرض کنید فقط حافظه‌های ۸ بایتی در اختیار داریم؛ بنابراین باید از دو تراشه حافظه ۸ بایتی استفاده کنیم که هر کدام ۳ خط آدرس دارند (در انتهای این مثال خواهیم گفت چرا چنین فرضی را در نظر گرفته‌ایم). نحوه ذخیره سازی اطلاعات را در شکل مقابل می‌بینید. معمولاً خانه‌های خالی حافظه‌ها حاوی 11111111 (FF h) هستند.

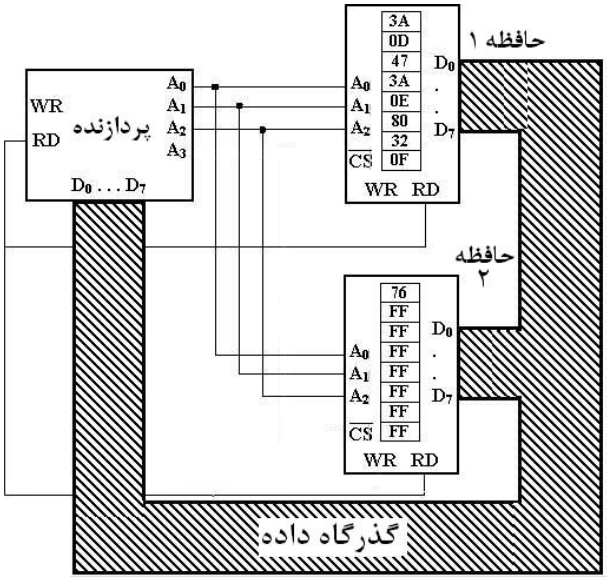


چگونه این دو حافظه باید به پردازنده متصل شوند؟

طبق قرار قبلی، گذرگاههای داده این دو حافظه باید به هم متصل و مشترک شده و به گذرگاه داده پردازنده متصل شوند.

آنچه باقی می‌ماند نحوه اتصال خطوط آدرس پردازنده برای آدرس‌دهی مناسب به این حافظه هاست. پردازنده ۴ خط آدرس و حافظه‌ها هرکدام ۳ خط آدرس دارند. طبق قرار قبلی، خطوط آدرس حافظه‌ها باید به خطوط پایین آدرس پردازنده و سیگنالهای RD نیز به هم متصل می‌شوند.

شکل زیر اتصال ابتدایی بین این دو حافظه و پردازنده را نشان می‌دهد:



برای فهم چگونگی تکمیل اتصالات، باید عملکرد پردازنده برای اجرای این برنامه را بررسی کنیم. قبلاً گفته شد که پردازنده برای دریافت کد عمل از حافظه مقدار PC را روی گذرگاه آدرس خود گذاشته و سیگنال RD را فعال می کند.

دستورات خواندن اطلاعات از حافظه (مانند دستور MOV A, [13]) چگونه اجرا می‌شوند؟

اصل دوم طراحی سیستم: پردازنده برای خواندن محتویات یکی از خانه‌های حافظه، شماره (آدرس) آن خانه را روی گذرگاه آدرس خود گذاشته و سیگنال RD خود را فعال می کند. در این لحظه پردازنده انتظار دارد محتویات خانه حافظه ذکرشده روی گذرگاه داده‌اش ظاهر شود.

دستورات نوشتن اطلاعات در حافظه (مانند دستور MOV [15], A) چگونه اجرا می‌شوند؟

اصل سوم طراحی سیستم: پردازنده برای نوشتن مقداری در یکی از خانه‌های حافظه، شماره (آدرس) آن خانه را روی گذرگاه آدرس خود و آن مقدار را روی گذرگاه داده خود گذاشته و سیگنال WR خود را فعال می کند.

مراحل اجرای این برنامه به صورت زیر است:

PC	عددی که پردازنده روی **گذرگاه آدرس** خود قرار می‌دهد	مقداری که در این لحظه باید روی **گذرگاه داده** پردازنده ظاهر شود
0000	* 0000	3A (دریافت کد عمل)
0001	* 0001	0D (دریافت کد عمل)

---- اجرای کد 3A 0D ----

0001	**1101**	**محتویات خانه ۱۳ حافظه**
0010	* 0010	47 (دریافت کد عمل)

---- اجرای کد 47 h ----

0011	* 0011	3A (دریافت کد عمل)
0100	* 0100	0E (دریافت کد عمل)

---- اجرای کد 0E 3A ----

0100	**1110**	**محتویات خانه ۱۴ حافظه**
0101	* 0101	80 (دریافت کد عمل)

---- اجرای کد 80 h ----

0110	* 0110	32 (دریافت کد عمل)
0111	* 0111	0F (دریافت کد عمل)

---- اجرای کد 0F 32 ----

0111	**1111**	**محتویات ثبات A** **(که پردازنده روی گذرگاه داده قرار می دهد.)**
1000	1000	76 (دریافت کد عمل)

---- اجرای کد 76 h ----

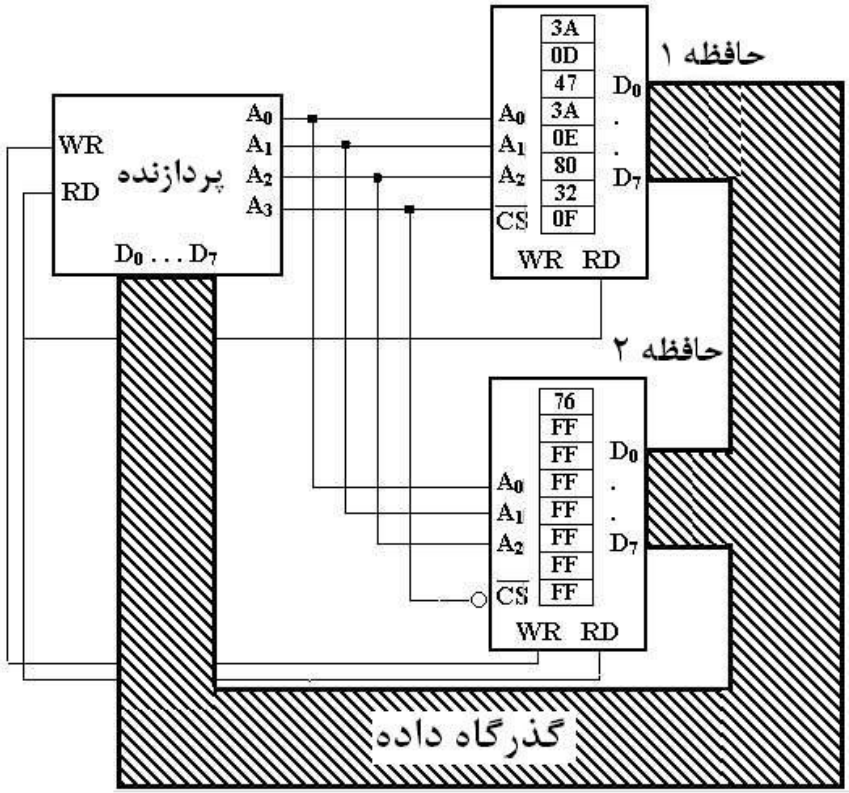
پرسش) با توجه به اصول دوم و سوم طراحی سیستم، نحوه اجرای دستور 0E 3A و دستور 0F 32 را توضیح دهید.

با دقت در روند اجرای برنامه، مشکل مهمی در مدار رسم شده به چشم می‌آید. وقتی پردازنده می‌خواهد دستور اول و دستور آخر را از حافظه واکشی کند، در هر دو حالت روی خطوط پایین آدرس خود ($A_2A_1A_0$) آدرس 000 قرار می‌دهد. چون این خطوط به خطوط آدرس هر دو حافظه متصل است، هر دو حافظه می‌خواهند محتویات اولین خانه خود را روی گذرگاه داده قرار دهند که طبیعتاً مشکل ایجاد می‌کند که نشان می‌دهد هنوز اتصالات مربوط به حافظه تکمیل نشده است.

برای حل این مشکل به آنچه برای اجرای این برنامه روی گذرگاه آدرس می‌آید دقت کنید. در کنار تعدادی از این آدرسها علامت * گذاشته شده است. با دقت در این آدرسها می‌توان دریافت که اولاً در تمام این آدرسها بیت پرارزش آدرس (A_3) صفر است؛ ثانیاً تمام دستوراتی که با این آدرسها مورد مراجعه قرار گرفته‌اند در حافظه اول قرار دارند.

اکنون به آدرسهایی توجه کنید که علامت * ندارند. بیت A_3 در تمام این آدرسها برابر 1 بوده و ضمناً خانه‌هایی که به کمک این آدرس مورد مراجعه قرار می‌گیرند، همگی در حافظه دوم قرار دارند.

نتیجه مهمی به دست می‌آید: «هنگامی که پین A_3 صفر است پردازنده با خانه‌های حافظه اول کار می‌کند. وقتی این پین یک می‌شود، پردازنده با خانه‌های حافظه دوم تبادل اطلاعات می‌کند». به عبارت دیگر، هنگامی که پردازنده پین A_3 خود را صفر می‌کند باید حافظه اول فعال و حافظه دوم غیرفعال شود. وقتی پردازنده پین A_3 خود را یک می‌کند باید حافظه اول غیرفعال و حافظه دوم فعال شود. فعالسازی حافظه‌ها از طریق پین $\overline{CS}$ انجام می‌شود. شکل زیر اتصال مناسب برای استفاده از این ویژگی را نشان می دهد.



در این سیستم از گسترش طول آدرس استفاده شده است.

شاید این سؤال برای شما پیش آمده باشد که وقتی در اینجا فقط با حافظه‌های ۸ خانه ای سروکار داریم، معنای خانه شماره ۱۳ حافظه چیست ؟

معمولاً تمام بایتهای فضای آدرس‌دهی به دنبال هم شماره‌گذاری و خوانده می شوند؛ مثلاً در این پردازنده که چهار خط آدرس (۱۶ مکان آدرس‌دهی شده) داریم، فرض می‌کنیم فضای آدرس‌دهی از صفر تا ۱۵ پشت سر هم قرار دارد که حافظه ۸ خانه‌ای اول، ۸ آدرس ابتدایی و حافظه ۸ خانه دوم، ۸ آدرس دوم را پوشش می‌دهد. مثلاً منظور از خانه شماره ۱۳ حافظه، **خانه شماره ۵ حافظه دوم** است. پردازنده برای درخواست ارتباط با این خانه عدد 1101 (۱۳) را روی گذرگاه آدرسش قرار می‌دهد.

پرسش) اگر به جای دو حافظه هشت خانه‌ای، از چهار حافظه چهارخانه‌ای استفاده کنیم، مکان خانه شماره ۱۳ حافظه در این پیکربندی را تعیین کنید.

چگونه این سیستم با وجود اتصال گذرگاههای داده حافظه‌ها بهم درست کار می‌کند؟ از بیت A_3 برای انتخاب حافظه استفاده شده است؛ وقتی پردازنده ۸ بایت اول برنامه که در حافظه ۱ ذخیره شده را درخواست می‌کند، PC که روی گذرگاه آدرس قرار می‌گیرد از **0**000 تا 111**0** تغییر می‌کند. می‌بینید که برای خواندن ۸ بایت اول برنامه، بیت A_3 پردازنده صفر است؛ بنابراین حافظه ۱ فعال و حافظه ۲ غیرفعال است (به $\overline{CS}$ها دقت کنید) و محتویات حافظه ۱ روی گذرگاه داده مشترک قرار می‌گیرد؛ اما وقتی پردازنده بایت شماره ۸ (**1**000) را درخواست می‌کند (یعنی این آدرس را روی گذرگاه آدرس خود می‌گذارد) یا با خانه‌های ۱۳ (**1**101) یا ۱۴ (**1**110) یا ۱۵ (**1**111) فضای آدرس‌دهی کار می‌کند، بیت A_3 «یک» بوده و حافظه اول فعال و حافظه دوم غیرفعال می‌شود.

پرسش) چون می‌خواهیم در خانه شماره ۱۵ بنویسیم، حافظه استفاده شده در این آدرس باید از نوع RAM باشد. مکان خانه شماره ۱۵ را در حافظه‌های متصل شده مشخص کنید.

پرسش) با توجه به اینکه حافظه دوم از نوع RAM است، اگر برق سیستم قطع شود، کد 76 H نیز پاک می‌شود و با وصل مجدد برق باید راهی برای اضافه کردن این کد به انتهای برنامه اندیشید. یک راه نرم‌افزاری این است که در ابتدای برنامه دستوراتی بنویسیم که کد 76h را در انتهای برنامه (در خانه مناسب حافظه RAM) بنویسد. روش سخت‌افزاری، اضافه کردن یک تراشه ثبات با محتویات 76h به سیستم است. در این صورت باید اتصالات سیستم به نحوی انجام شود که با وصل شدن برق به سیستم، محتویات ثبات فوق در خانه مناسب حافظه RAM کپی شود. این دو راه را پیاده‌سازی و با هم مقایسه کنید.

پرسش) چرا سیگنال WR پردازنده فقط به حافظه ۲ متصل شده است ؟

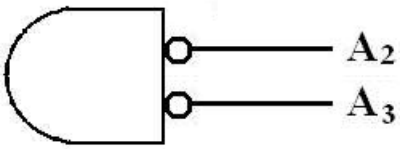
پرسش) اگر آخرین کد برنامه این سیستم، کد HALT نباشد چه می شود؟

پرسش) اگر به جای دو حافظه ۸ خانه‌ای، چهار حافظه ۴ خانه‌ای داشته باشیم، به کمک مراحل اجرای برنامه شرح دهید چگونه این چهار حافظه باید به پردازنده متصل شوند؟ **راهنمایی:** دو خط آدرس حافظه‌ها را به خطوط پایین آدرس پردازنده (A_1A_0) متصل و از خطوط بالای آدرس پردازنده (A_3A_2) برای انتخاب یک حافظه در

A3	A2	A1	A0	
0	**0**	0	0	= 0
0	**0**	0	1	= 1
0	**0**	1	0	= 2
0	**0**	1	1	= 3
0	1	0	0	= 4
0	1	0	1	= 5
0	1	1	0	= 6
0	1	1	1	= 7
1	**0**	0	0	= 8
1	**0**	0	1	= 9
1	**0**	1	0	= 10
1	**0**	1	1	= 11
1	1	0	0	= 12
1	1	0	1	= 13
1	1	1	0	= 14
1	1	1	1	= 15

هر لحظه استفاده کنید. برای این کار، به نمایش دودویی آدرسهای 0000 تا 1111 در کادر مقابل توجه کنید.

می‌بینید که وقتی پردازنده با چهار خانه اول حافظه تبادل داده می‌کند، A_3A_2 برابر 00 است؛ بنابراین هنگامی که A_3A_2 برابر 00 است باید حافظه اول فعال و بقیه حافظه‌ها غیرفعال شوند. با اتصال خروجی گیت AND زیر به ورودی $\overline{CS}$ حافظه اول این موضوع تضمین خواهد شد:



این طراحی را برای سه حافظه بعدی تکمیل کنید..

***

شاید این پرسش برای شما پیش آمده باشد که وقتی با انتخاب یک حافظه ۱۶ بایتی می‌توان به راحتی خطوط آدرس حافظه را به خطوط آدرس پردازنده متصل کرد، چرا دو حافظه ۸ بایتی انتخاب کرده‌ایم تا به مشکلات انتخاب حافظه به کمک CS و خطوط بالای آدرس پردازنده برخورد کنیم؟

این موضوع در سیستمهای مبتنی بر پردازنده بسیار پیش می‌آید. مثلاً به جای یک حافظه RAM یک گیگابایتی در کامپیوتر می‌توان از دو حافظه ۵۱۲ مگابایتی یا ۴ حافظه ۲۵۶ مگابایتی (که ممکن است بیشتر در دسترس باشند) استفاده نمود. بنابراین باید با تکنیک استفاده از حافظه‌های کم‌حجم‌تر به منظور پوشش فضای یک آدرس‌دهی بزرگ آشنا باشیم.

تاکنون مجموعه قوانین زیر را برای طراحی سیستم دیده‌ایم:

گذرگاه داده پردازنده بیت به بیت (پین به پین) به گذرگاه داده حافظه متصل می‌شود.
خطوط آدرس حافظه به خطوط پایین آدرس پردازنده متصل می‌شوند.
از خطوط بالای آدرس پردازنده برای انتخاب حافظه مناسب استفاده می‌شود.
سیگنال RD پردازنده باید به سیگنال RD حافظه متصل شود.
سیگنال WR پردازنده باید به سیگنال WR حافظه متصل شود.

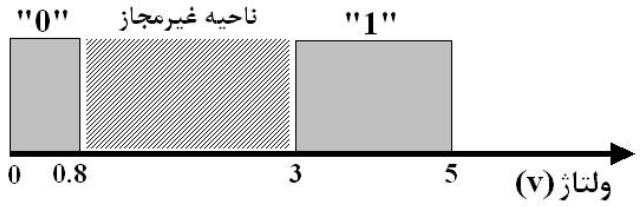
اتصال وسایل ورودی/خروجی به پردازنده

همانطور که قبلاً ذکر شد، مهمترین وظیفه پردازنده، پردازش اطلاعات ورودی به آن و ارسال نتایج به خارج است. در مثالهای قبلی داده‌های ورودی و خروجی در حافظه ذخیره

می‌شدند. در مثالهای بعدی در پی این هستیم که ورودی و خروجی سیستم را نه از طریق حافظه که از راههای قابل‌لمس‌تر پیاده کنیم.

اطلاعاتی که در سیستمهای مبتنی بر پردازنده و مدارهای منطقی وجود دارد، همه در قالب «صفر و یک منطقی» است. بنابراین اگر بتوانیم یک بیت (یک رقم دودویی) را تولید کنیم، می‌توانیم اطلاعات چند بیتی را نیز به عنوان ورودی به سیستم بدهیم. به همین ترتیب اگر بتوان یک بیت را نمایش داد، می‌توان اطلاعات چندبیتی را نیز به راحتی به نمایش درآورد.

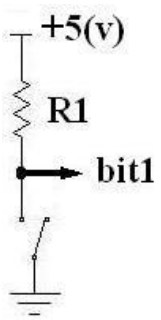
در مدارهای منطقی و سیستمهای مبتنی بر پردازنده، از «صفر و یک منطقی» تعبیر ولتاژی می‌شود:



شکل بالا نشان می‌دهد، اگر ولتاژی بین صفر تا ۰/۸ ولت باشد از آن به «صفر منطقی» و اگر بین ۳ تا ۵ ولت باشد از آن به «یک منطقی» تعبیر می‌شود[۱]. مثلاً اگر به پین ورودی یک گیت NOT (پین ۱ تراشه 7404) ولتاژی حدود صفر (مثلاً ۰/۲ ولت) اعمال کنیم، روی پین خروجی این گیت (پین ۲ تراشه 7404) ولتاژی حدود ۵ ولت (مثلاً ۴/۸ ولت) به کمک ولت‌متر قابل مشاهده است. ولتاژ بین ۰/۸ تا ۳ ولت در منطق ولتاژی فوق غیرمجاز است؛ یعنی اگر خروجی یک مدار منطقی مثلاً ۲ ولت باشد، مدار خراب است و به درستی کار نمی‌کند. اگر به ورودی یک مدار منطقی ولتاژ ۲ ولت داده شود، مدار ممکن است از آن تعبیر «صفر» یا «یک» داشته باشد.

پس برای ایجاد یک بیت، باید مداری دوحالته داشته باشیم که دو ولتاژ صفر و ۵ ولت را بتواند تولید کند. برای نمایش یک بیت نیز باید مداری ببینیم که در مقابل دو ولتاژ صفر و ۵ ولت، دو واکنش متفاوت ایجاد کند.

چگونه یک بیت را تولید کنیم ؟



ساختار مقابل با استفاده از یک کلید دوحالته این کار را انجام می دهد.

در این ساختار اگر کلید باز باشد، سیگنال bit1 توسط مقاومت R1 به ۵ ولت متصل است؛ بنابراین «یک منطقی» ایجاد می شود.

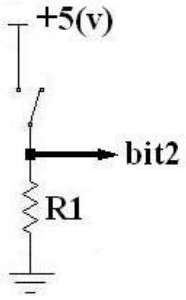
[۱] این منطق ولتاژ موسوم به TTL است. در منطق‌های ولتاژی دیگر مانند CMOS، ولتاژهای دیگری به «صفر» و «یک» منطقی نسبت داده می‌شوند.

اگر کلید بسته باشد، سیگنال bit1 به زمین متصل شده و «صفر منطقی» ایجاد می شود.

به طور خلاصه :

bit1	کلید
"1"	باز
"0"	بسته

یک بیت با استفاده از ساختار روبرو نیز قابل ایجاد است.



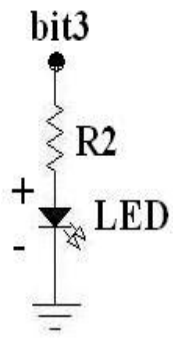
در این ساختار اگر کلید باز باشد، سیگنال bit2 توسط مقاومت R1 به زمین متصل است؛ بنابراین «صفر منطقی» ایجاد می شود.
اگر کلید بسته شود، سیگنال bit2 به ۵ ولت متصل شده و «یک منطقی» ایجاد می‌شود.

به طور خلاصه :

bit2	کلید
"0"	باز
"1"	بسته

مقاومت R1 در هردو ساختار برای جلوگیری از اتصال کوتاه شدن منبع تغذیه هنگام بسته شدن کلید به کار رفته و مقدار نوعی آن 10 KΩ است.
همانطور که مشاهده می کنید **bit1** در هنگام **باز بودن کلید** برابر «یک منطقی» است و **bit2** در هنگام **بسته بودن کلید**.
دو ساختار ذکر شده از نظر فنی متفاوت هستند که بررسی این تفاوت به عنوان تحقیق به شما واگذار شده است.

چگونه یک بیت را نمایش دهیم ؟



ساده ترین راه برای نمایش یک بیت، استفاده از دیود نوری (LED[1]) است. دیود نوری یک عنصر الکترونیکی شبیه یک لامپ کوچک با دوپایه مثبت (آند) و منفی (کاتد) است. اگر ولتاژ پایه مثبت دیود نوری از ولتاژ پایه منفی آن از حدی بیشتر شود، دیود نوری روشن و در غیر این صورت خاموش می‌شود. ساختار روبرو برای نمایش یک بیت با دیود نوری به کار می رود.

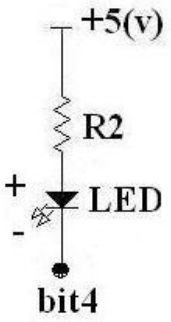
[1] **L**ight **E**mitting **D**iode

اگر bit3، «صفر منطقی» باشد، ولتاژی حدود صفر دارد و با توجه به صفر بودن ولتاژ سر منفی، دیود نوری روشن نخواهد شد. اگر bit3، «یک منطقی» باشد دارای ولتاژی بالاتر از ۳ ولت است و باعث روشن شدن دیود نوری می شود.

به طور خلاصه :

bit3	وضعیت دیود نوری
"0"	خاموش
"1"	روشن

ساختار دیگر نمایش یک بیت را در شکل روبرو می بینید.

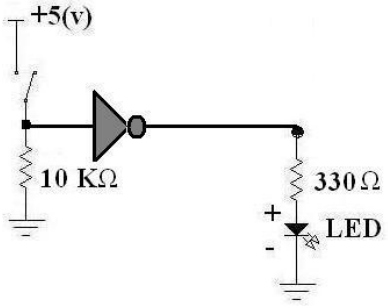


پرسش) با توجه به ساختار مقابل، جدول زیر را کامل کنید.

bit4	وضعیت دیود نوری
"0"	
"1"	

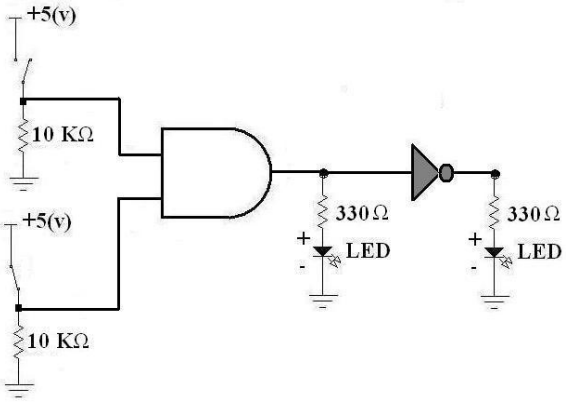
مقاومت R2 در هر دو ساختار برای جلوگیری از سوختن دیود نوری در اثر عبور جریان زیاد به کار رفته و مقدار نوعی آن Ω 330 است.

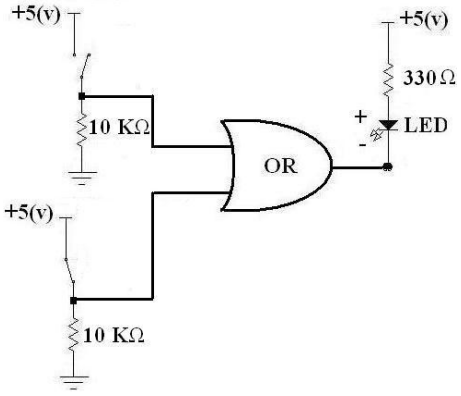
مثال) در ساختار روبرو مشخص کنید دیود نوری خاموش است یا روشن ؟



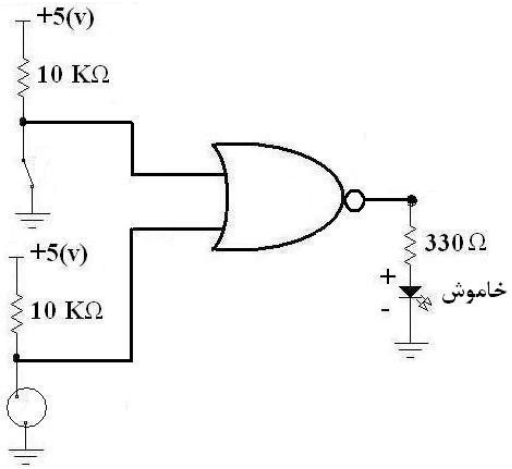
چون کلید باز است، ورودی گیت NOT با مقاومت 10KΩ به زمین متصل شده است و «صفر منطقی» است. اگر ورودی گیت NOT «صفر» باشد، خروجی آن «یک» (یعنی حدود ۵ ولت) است؛ چون ولتاژ سر مثبت دیود نوری از سر منفی آن بیشتر است، دیود نوری روشن می شود.

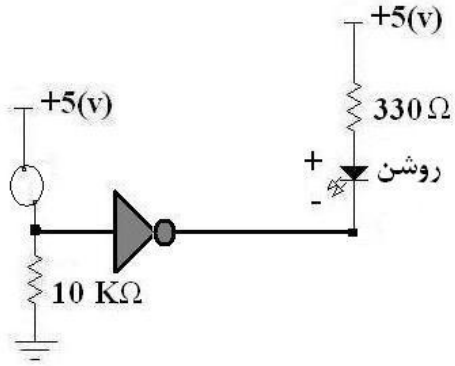
پرسش) در هریک از ساختارهای زیر مشخص کنید هر یک از دیودهای نوری روشن است یا خاموش؟





پرسش) در هریک از شکلهای زیر معلوم کنید کلید مشخص شده باز است یا خیر؟





مثال ۴) این مثال ما را با نحوه اتصال خروجی به سیستم آشنا می‌کند. می‌خواهیم سیستم مثال ۳ را به نحوی تغییر دهیم که محتوای خانه‌های ۱۳ و ۱۴ حافظه را با هم جمع کند و نتیجه که یک عدد ۸ بیتی است را روی ۸ عدد دیود نوری نشان دهد؛ به نحوی که روشن بودن هر دیود نوری نشان‌دهنده "یک منطقی" و خاموش بودن آن نشانه "صفر منطقی" باشد.

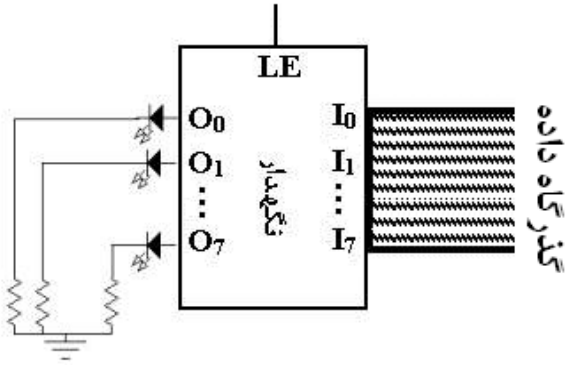
این ۸ عدد دیود نوری باید به کجا متصل شوند؟ اولین پاسخی که به ذهن می‌رسد این است که چون گذرگاه داده محل دسترسی به اطلاعات است، این ۸ دیود نوری را به سیمهای گذرگاه داده متصل کنیم. اما این عقیده خوبی نیست؛ چون می‌خواهیم از روشن و خاموش بودن دیودهای نوری نتیجه جمع را بدانیم؛ در حالی که اگر دیودهای نوری به گذرگاه داده متصل شوند، در هر لحظه محتویات گذرگاه داده را نشان خواهند داد که ممکن است کد عمل، آدرس مکانی از حافظه و ... باشد!

مشکل دیگری که وجود دارد این است که نتیجه عمل جمع تنها یک لحظه کوتاه روی گذرگاه داده ظاهر می‌شود و سپس گذرگاه داده به اطلاعات دیگر اختصاص می‌یابد؛ اگر

دیودهای نوری مستقیماً به گذرگاه داده متصل باشند، نمی‌توانند این لحظه کوتاه را ثبت کنند.

بنابراین باید سیستم به گونه‌ای طراحی شود که اولاً تنها لحظه‌ای که گذرگاه داده محتوی حاصل جمع است، دیودهای نوری را به گذرگاه داده متصل کند، ثانیاً وقتی نتیجه جمع از روی گذرگاه داده برداشته شد، اتصال بین دیودهای نوری و گذرگاه داده را قطع کند و نتیجه جمع را روی دیودهای نوری حفظ کند. بخشی از سیستم که این کار را انجام می‌دهد، تراشه‌ای به نام نگهدار[1] است.

تراشه نگهدار (مثلاً تراشه 74LS273) دارای ۸ پین ورودی، ۸ پین خروجی و یک پین فعال‌ساز به نام LE[2] است. هرگاه پین LE فعال شود، خروجیهای نگهدار به ورودیهای آن متصل خواهند شد و پس از غیرفعال شدن LE خروجیهای نگهدار در حالت قبلی خود باقی می‌مانند (قفل می‌شوند). بنابراین برای طراحی سیستم، گذرگاه داده را به ورودی نگهدار و ۸ دیود نوری را به خروجی آن متصل می‌کنیم و ترتیبی اتخاذ می‌کنیم که در لحظه‌ای که گذرگاه داده محتوی حاصل جمع مورد نظر است، پین LE نگهدار فعال و پس از قفل کردن حاصل جمع در خروجی (دیودهای نوری) غیرفعال شود. شکل تراشه نگهدار و نحوه اتصال آن به سیستم را در زیر می‌بینید:



به عنوان اولین گام برای طراحی این سیستم، برنامه را می‌نویسیم. پیش از این آموختیم که هرکدام از ورودیها و خروجی‌ها در سیستم مبتنی بر پردازنده دارای یک آدرس هستند. به همان شیوه که خانه‌های حافظه را آدرس‌دهی می‌کردیم، به خروجی (در اینجا تراشه نگهدار) هم یک آدرس (مثلاً آدرس صفر) نسبت می‌دهیم.

طراح سیستم باید به هر خروجی یک آدرس نسبت دهد.

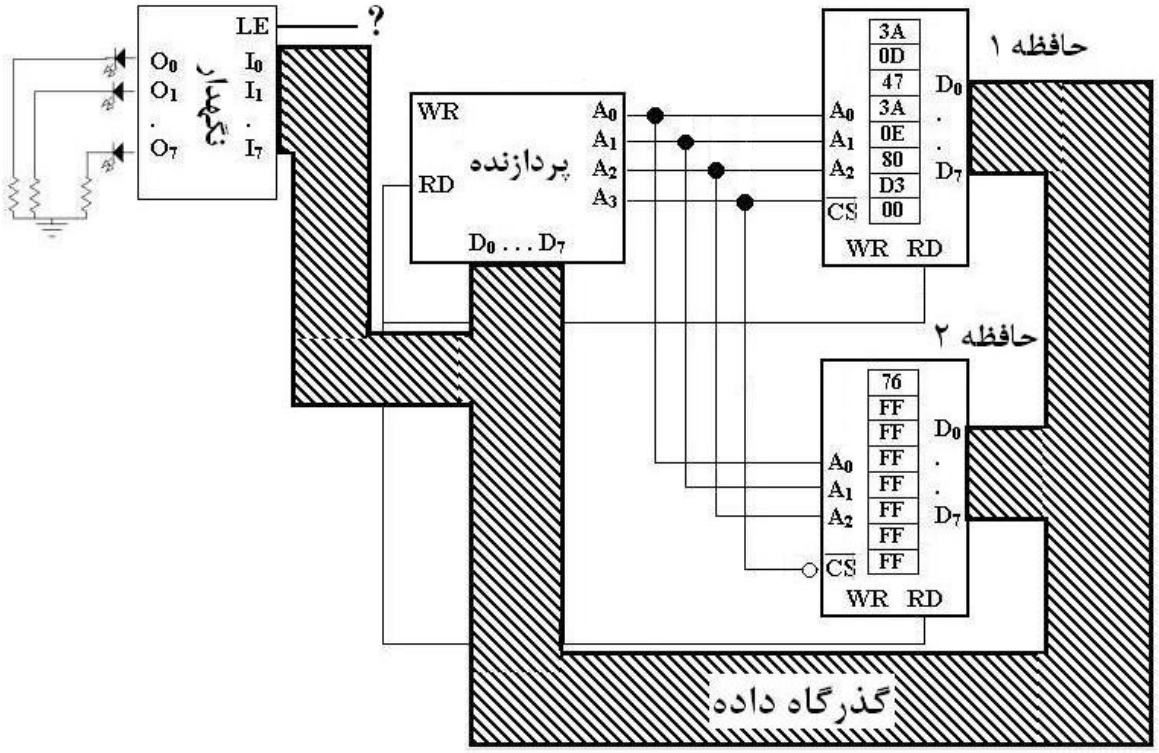
[1] ***Latch***

[2] ***Latch Enable : LE***

برنامه زیر را ببینید:

MOV	A,[13]	ASSEMBLER	3A	0D
MOV	B,A		47	
MOV	A,[14]		3A	0E
ADD	A,B		80	
OUT	[0],A		D3	00
HALT			76	

این برنامه را در دو حافظه ذخیره و به انضمام تراشه نگهدار به پردازنده متصل می‌کنیم. یک اتصال ابتدایی برای این سیستم را در شکل زیر می‌بینید:



تنها کاری که باقی‌مانده این است که پین LE را در لحظه مناسب فعال کنیم. این لحظه مناسب، وقتی است که پردازنده نتیجه عمل جمع را روی گذرگاه داده قرار می‌دهد. دقت در برنامه بالا نشان می‌دهد که این اتفاق در هنگام اجرای دستور Out [0], A رخ می‌دهد. این دستور مقدار ثبات A که حاصل جمع مورد نظر ماست را به **خروجی** با آدرس صفر ارسال می‌کند؛

بنابراین پین LE باید در لحظه اجرای این دستور فعال شود. نشانه‌های اجرای این دستور چیست؟ به بیان دیگر، این دستور چگونه توسط پردازنده اجرا می‌شود؟

اصل چهارم طراحی سیستم: پردازنده برای ارسال داده‌ای به یک خروجی، آدرس آن خروجی را روی گذرگاه آدرس و داده مورد نظرش را روی گذرگاه داده قرار داده و پین WR را فعال می‌کند.

این اصل شباهت عجیبی به اصل سوم (مربوط به نوشتن در حافظه) دارد! در واقع تفاوتی اساسی بین نوشتن در حافظه و ارسال داده به خروجی وجود دارد که چون نمی‌خواهیم در حال حاضر بحث را بیهوده پیچیده کنیم، در اصل چهارم ذکر نشده است. بعداً این اصل را تکمیل می‌کنیم.

مراحل اجرای این برنامه در زیر نشان داده شده است :

PC	عددی که پردازنده روی **گذرگاه آدرس** خود قرار می‌دهد	مقداری که در این لحظه باید روی **گذرگاه داده** پردازنده ظاهر شود
0000	0000 (آدرس حافظه)	3A (دریافت کد عمل)
0001	0001	0D (دریافت کد عمل)
	---- اجرای کد 3A 0D ----	
0001	**1101**	**محتویات خانه ۱۳ حافظه**
0010	0010	47 (دریافت کد عمل)
	---- اجرای کد 47 ----	
0011	0011	3A (دریافت کد عمل)
0100	0100	0E (دریافت کد عمل)
	---- اجرای کد 3A 0E ----	
0100	**1110**	**محتویات خانه ۱۴ حافظه**
0101	0101	80 (دریافت کد عمل)
	---- اجرای کد 80 ----	
0110	0110	D3 (دریافت کد عمل)
0111	0111	00 (دریافت کد عمل)
	---- اجرای کد D3 00 ----	
0111	**0000** (آدرس خروجی)	**محتویات ثبات A**

پین LE در این لجظه باید فعال شود.

1000 1000 76

---- اجرای کد 76 ----

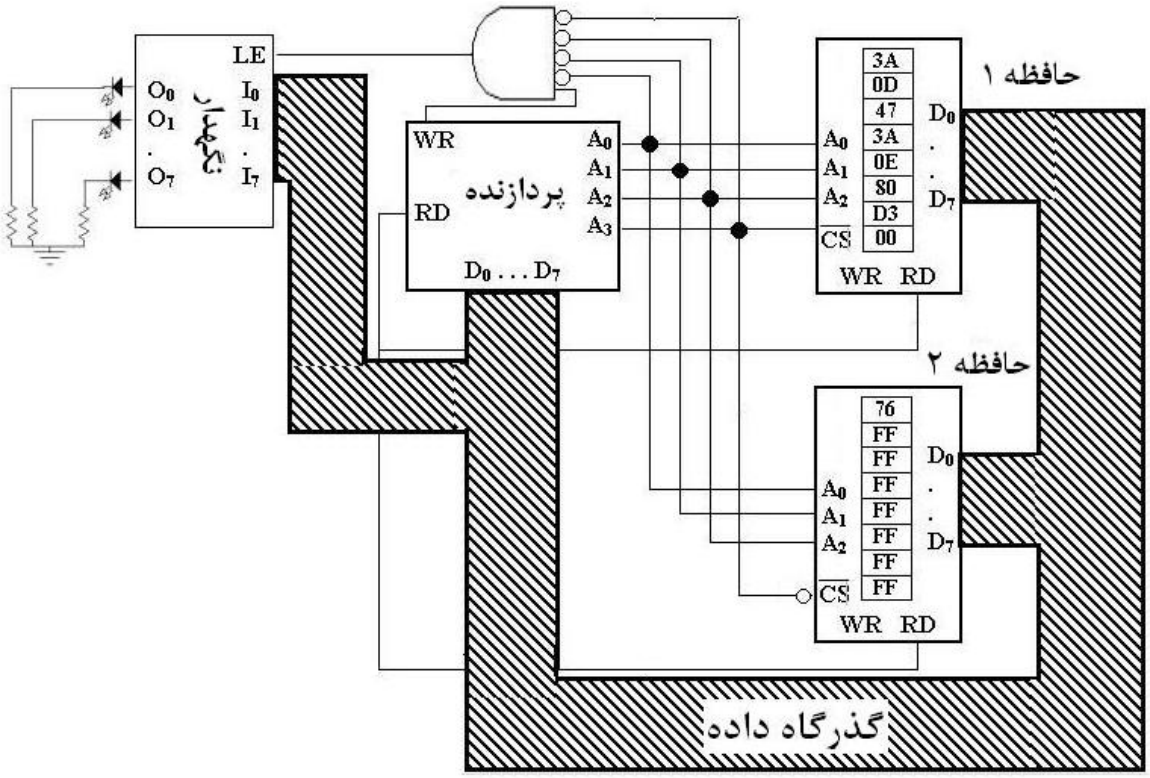
به لحظه‌ای که پین LE باید فعال شود توجه کنید. به نظر می‌آید، در لحظه‌ای که آدرس 0000 روی گذرگاه آدرس می‌آید باید پین LE فعال شود. این ایده کاملاً درست نیست؛ چون همانطور که در روند اجرای برنامه مشاهده می‌کنید وقتی پردازنده می‌خواهد اولین کد عمل برنامه را بخواند نیز آدرس 0000 را روی گذرگاه آدرسش قرار می‌دهد. تفاوت این دو حالت این است پردازنده هنگام خواندن اولین کد عمل برنامه سیگنال RD و هنگام ارسال داده به خروجی سیگنال WR را فعال می‌کند.

بنابراین در لحظه‌ای که:

- پردازنده آدرس 0000 را روی گذرگاه آدرس خود می‌گذارد،
- سیگنال WR را فعال می‌کند،

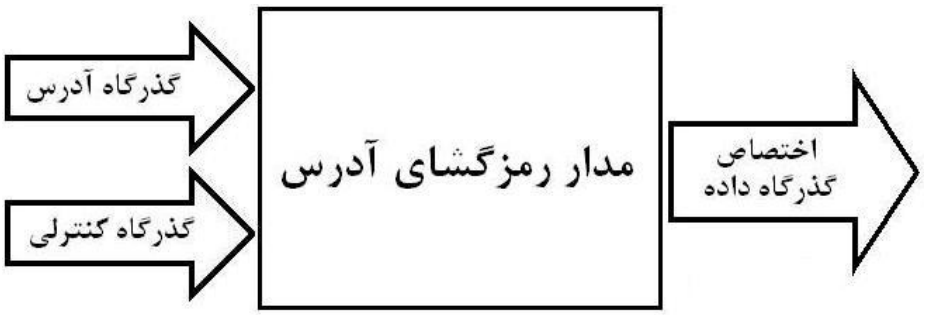
پین LE باید فعال شود.

مدار مورد نظر را در شکل زیر می‌بینید.



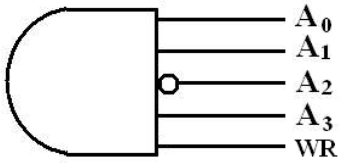
همانطور که گفته شد، پردازنده برای اجرای دستور OUT [0],A ، محتویات A (حاصل جمع مورد نظر) را روی گذرگاه داده و آدرس 0000 را روی گذرگاه آدرس خود گذاشته و سیگنال WR را فعال می‌کند. در این لحظه خروجی گیت AND در مدار بالا «یک» شده و نگهدار فعال می‌شود و مقدار گذرگاه داده را در خروجی خود (روی دیودهای نوری) نشان می‌دهد. وقتی پردازنده می‌خواهد دستور بعد را اجرا کند، PC دستور بعدی (آدرس ۹ یعنی 1001) را روی گذرگاه آدرس می‌گذارد که طبعاً نگهدار را غیرفعال خواهد کرد (چون خروجی گیت AND صفر می شود)؛ اما جای نگرانی نیست، چون عدد مورد نظر ما روی خروجی نگهدار (دیودهای نوری) قفل شده است.

گیت AND استفاده شده در این مثال، شکل ساده ای از مدار رمز گشای آدرس[1] است. وظیفه این مدار در سیستمهای مبتنی بر پردازنده، اختصاص گذرگاه داده به متقاضی مجاز با استفاده از اطلاعات گذرگاه آدرس و کنترل است.



در واقع گیت AND در این مثال، آدرس 0000 را رمزگشایی می‌کند.

پرسش) اگر بخواهیم نتیجه عمل جمع علاوه بر نمایش روی دیودهای نوری در خانه شماره ۱۵ حافظه هم ذخیره شود، چه تغییراتی در سخت افزار و نرم افزار سیستم باید انجام شود؟



پرسش) گفتیم طراح سیستم باید به هر خروجی یک آدرس نسبت دهد. اگر آدرس خروجی را به جای صفر، ۱۱ در نظر بگیریم (از دستور OUT [11], A استفاده کنیم) نشان دهید که ورودیهای مدار رمزگشای آدرس (گیت AND) به شکل مقابل خواهند بود. توجه کنید که در اینجا ورودیهای گیت AND باید به نحوی به گذرگاه آدرس متصل شوند که تنها وقتی عدد ۱۱ روی گذرگاه آدرس می آید، خروجی گیت AND برابر ۱ شود. در اینجا چون غیر از خروجی فوق آدرس ۱۱ دیگری در سیستم وجود ندارد، استفاده از سیگنال

[1] ***Address Decoder Circuit***

WR لزومی ندارد؛ اما بهتر است عادت کنیم در ورودی مدار رمزگشای آدرس خروجی از سیگنال WR استفاده نماییم.

به کمک مثال چهار، قوانین طراحی سیستم خود را کاملتر می‌کنیم:

- گذرگاه داده پردازنده بیت به بیت (پین به پین) به گذرگاه داده حافظه و ورودی تراشه نگهدار متصل می‌شود.
- خطوط آدرس حافظه به خطوط پایین آدرس پردازنده متصل می‌شوند.
- از خطوط بالای آدرس پردازنده برای انتخاب حافظه مناسب استفاده می‌شود.
- سیگنال RD پردازنده باید به سیگنال RD حافظه متصل شود.
- سیگنال WR پردازنده باید به سیگنال WR حافظه متصل شود.
- برای اتصال خروجی به سیستم، باید از تراشه نگهدار استفاده کنیم که گذرگاه داده به ورودی این تراشه متصل می‌شود. تراشه نگهدار وقتی باید فعال شود که پردازنده آدرس آن خروجی (که توسط طراح سیستم به آن نسبت داده می‌شود) را روی گذرگاه آدرس خود قرار داده و سیگنال WR خود را فعال کند.
- برای اتصال خروجی به سیستم، باید به هر خروجی یک آدرس نسبت داده و این آدرس را هم در نوشتن برنامه و هم در طراحی مدار رمزگشای آدرس (گیت AND فعال‌کننده تراشه نگهدار) مورد توجه قرار دهیم.

پرسش) دستور NOT A محتویات ثبات A را معکوس می‌کند و کد عمل آن 73h است. نرم‌افزار و سخت‌افزار سیستم بالا را به نحوی تغییر دهید که مجموع خانه‌های ۱۳ و ۱۴ حافظه را به خروجی با آدرس صفر و معکوس آن را به خروجی با آدرس ۱۱ ارسال کند.

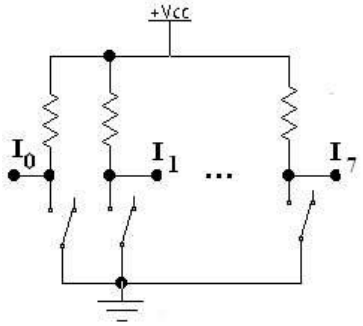
برخلاف وعده‌ای که کرده بودیم، در مدار رسم‌شده سیگنال WR پردازنده را به پین WR حافظه‌ها متصل نکرده‌ایم. دلیل این موضوع این است که وقتی پردازنده آدرس 0000 را روی گذرگاه آدرسش قرار داده و پین WR را فعال می‌کند، داده باید به خروجی شماره صفر ارسال شود؛ در حالی که اگر سیگنال WR پردازنده به پین WR حافظه اول متصل باشد، در این لحظه تلاشی برای نوشتن در خانه صفر حافظه اول هم انجام می‌شود که صحیح نیست.

دلیل این خلف وعده این است که هنوز تفاوت دقیق اصل سوم (نوشتن در حافظه) و اصل چهارم (ارسال داده به خروجی) که به دو سیگنال MemRQ و IORQ برمی‌گردد را بیان نکرده‌ایم. خوشبختانه چون در این سیستم نوشتن در حافظه نداریم، این موضوع مشکلی

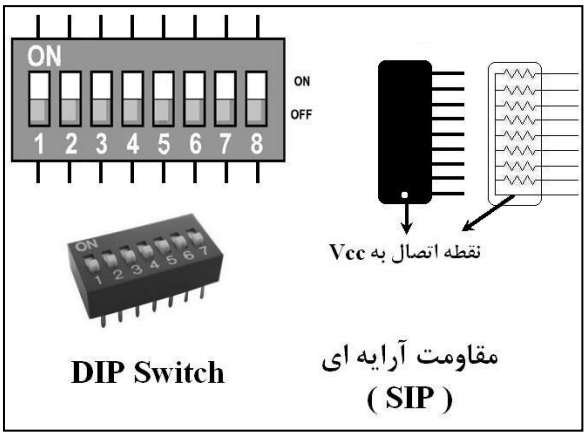
پیش نمی‌آورد. اما در مثال بعد با تکمیل اصول طراحی پردازنده‌ها، این خلف وعده را جبران خواهیم کرد.

در اینجا لازم است باز به مسأله زمانبندی سیستم توجه کنیم. هنگامی که پردازنده آدرس خروجی (0000) را روی گذرگاه آدرس خود قرار داده و سیگنال WR را فعال می‌کند، گیت AND رمزگشای آدرس، تراشه نگهدار را فعال کرده و گذرگاه داده سیستم را به LEDها متصل می‌کند. توجه داشته باشید که زمان اجرای دستور OUT [0],A (که طی آن آدرس 0000 روی گذرگاه آدرس قرار دارد و سیگنال WR فعال است) به دلیل سرعت بالای پردازنده، بسیار کوتاه است. معمولاً سرعت تراشه نگهدار از سرعت پردازنده بسیار پایین‌تر است و به همین لحاظ در طی زمان کوتاه اجرای دستور، تراشه نگهدار موفق به قفل کردن داده‌ها نمی‌شود. بنابراین باید مکانیسمی اندیشیده شود تا اجرای این دستور چند برابر حالت عادی به طول بیانجامد تا تراشه نگهدار فرصت کافی برای قفل کردن داده‌ها داشته باشد.

مثال ۵) در مثال ۴ متوجه شدید که چگونه می‌توان حاصل پردازش داده‌های ورودی را روی تعدادی دیود نوری نشان داد.
اکنون می‌خواهیم پا را فراتر نهاده و داده‌های ورودی را نیز به جای حافظه، از خارج پردازنده بخوانیم؛ به بیان دیگر می‌خواهیم سیستمی طراحی کنیم که دو عدد را از کاربر سیستم دریافت کرده و آنها را با هم جمع کند و نتیجه را روی تعدادی دیود نوری نشان دهد.
سؤال اول این است که چگونه می‌توان اعداد را به سیستم وارد کرد ؟ ساده‌ترین راه، ورود اعداد به صورت دودویی با استفاده از ۸ عدد کلید به صورت شکل زیر است.



همانطور که می‌بینید، به‌عنوان مثال اگر کلید اول بسته باشد، ولتاژ I_0 برابر صفر ولت («صفر منطقی») و اگر کلید باز باشد، این ولتاژ حدود ۵ ولت («یک منطقی») است. بنابراین با باز و بسته کردن این کلیدها می‌توان یک عدد ۸ بیتی (بین صفر تا ۲۵۵) تولید کرد.

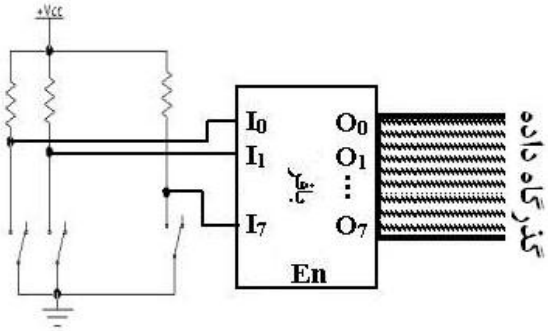


معمولاً برای اتصال ساختار فوق (۸ کلید دوحالته)، از کلیدهای کوچک کنار هم که در تعداد ۴ تایی یا ۸ تایی ساخته می شوند و به نام میکروسوییچ یا DIP Switch معروفند، استفاده می‌شود. همچنین برای خودداری از به کار بردن ۸ مقاومت مجزا که باعث شلوغی مدار می شود، از مقاومتهای آرایه ای که به نام SIP معروفند، استفاده می شود که هم جای کمتری اشغال می کنند و هم اتصال آنها ساده تر است.

سؤال دوم این است که این کلیدها باید به کجا متصل شوند؟ به گذرگاه داده؟ واضح است که این راه عملی نیست؛ چون اگر کلیدها را مستقیماً به گذرگاه داده متصل کنیم، عدد روی گذرگاه داده همیشه همان عددی است که کلیدها نشان می‌دهند و گذرگاه داده از انجام وظایف دیگر خود مانند انتقال کدهای عمل و آدرسهای حافظه و آدرسهای ورودی/خروجی باز می‌ماند.

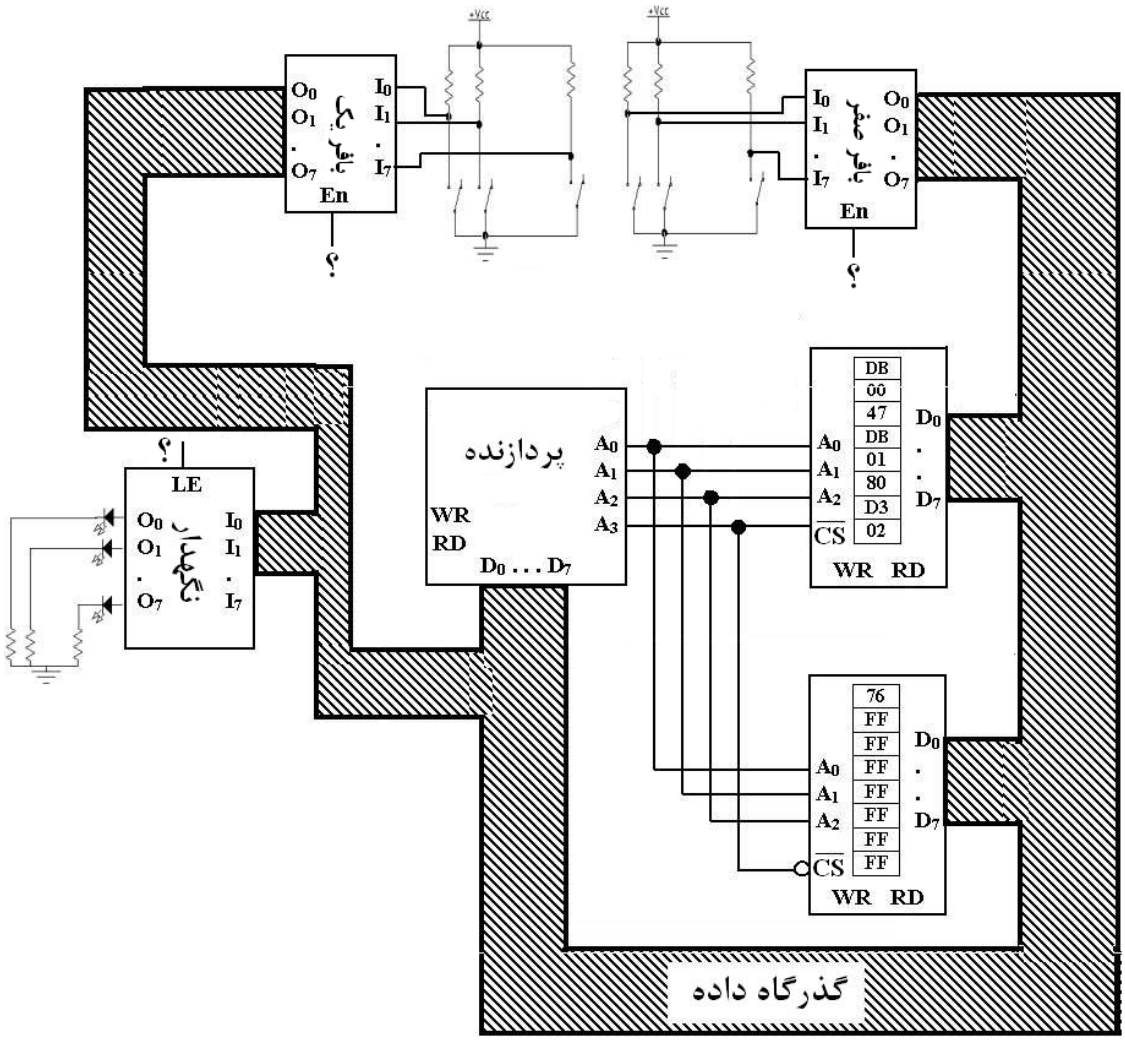
در واقع تنها در لحظه‌ای که پردازنده می‌خواهد داده‌های روی گذرگاه داده را بخواند، باید این گذرگاه به کلیدها متصل شود. قسمتی از سیستم که این کار را انجام می‌دهد، تراشه‌ای به نام بافر سه‌حالته[1] است.

این تراشه دارای تعدادی ورودی و خروجی و یک پین فعالساز En است. خروجی این تراشه در صورت غیرفعال بودن پین En ، امپدانس بالا[2] (مثل حالت قطع) و در صورت فعال بودن این پین برابر مقدار ورودی است. بنابراین ۸ عدد کلید را به ۸ پین ورودی بافر سه‌حالته و گذرگاه داده را به خروجی آن متصل می‌کنیم و ترتیبی می‌دهیم که این تراشه فقط هنگامی که پردازنده می‌خواهد عددی که کلیدها روی آن تنظیم شده‌اند را بخواند فعال شود و کلیدها را به گذرگاه داده متصل کند. نحوه اتصال بافر سه‌حالته به پردازنده را در شکل مقابل می‌بینید. چون می‌خواهیم دو عدد را بخوانیم، به دو مجموعه کلید و دو تراشه بافر سه‌حالته ۸ ورودی نیاز داریم.



[1] *Tri-State Buffer*
[2] *High Impedance*

اتصال اولیه این سیستم را در شکل زیر می‌بینید:



سه علامت سؤال در شکل وجود دارد که باید تکمیل شود؛ به عبارت دیگر با توجه به روند اجرای برنامه، باید لحظه‌ای که پینهای En بافرها و پین LE نگهدار باید فعال شوند را پیدا کرده و به کمک مدار مناسب بازسازی کنیم.

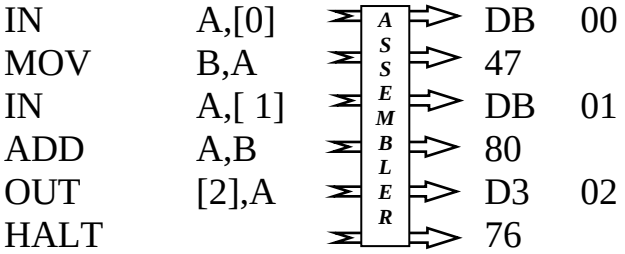
مطابق قرار قبلی، برای تکمیل مدار فوق برنامه مورد نظر را نوشته و طبق روندی که پردازنده برای اجرای برنامه در پیش می‌گیرد، اتصالات مدار را کامل می‌کنیم.

همانطور که در مورد خروجی و تراشه نگهدار عمل کردیم، به ورودیها نیز آدرس می‌دهیم؛ مثلاً به مجموعه کلید اول که قرار است عدد اول را تولید کند، آدرس «صفر» و به مجموعه کلید دوم (تراشه بافر سه‌حالته دوم) که عدد دوم را از آن می‌خوانیم، آدرس «یک» اختصاص می‌دهیم.

طراح سیستم باید به هر ورودی یک آدرس نسبت دهد.

بدیهی است که خروجی سیستم یعنی تراشه نگهدار هم باید دارای آدرس باشد که آدرس آن را در این سیستم «دو» درنظر می‌گیریم. توجه کنید که آدرس ورودی/خروجی‌ها دلخواه است و توسط طراح سیستم انتخاب می‌شود.

به برنامه زیر دقت کنید:



دستور IN A, [0] محتویات ورودی شماره صفر را خوانده و به ثبات A منتقل می‌کند. در لحظه اجرای این دستور، باید گذرگاه داده سیستم به مجموعه کلید اول (ورووددی شماره صفر) متصل شود. نشانه‌های اجرای این دستور چیست؟ به بیان دیگر، این دستور چگونه توسط پردازنده اجرا می‌شود؟

> **اصل پنجم طراحی سیستم:** پردازنده برای دریافت داده‌ای از یک ورودی، آدرس آن ورودی را روی گذرگاه آدرس خود قرار داده و پین RD را فعال می‌کند و محتویات گذرگاه داده را (که باید در این لحظه مقدار ورودی را روی خود داشته باشد) می‌خواند.

این اصل نیز شبیه اصل سوم (خواندن یکی از خانه‌های حافظه) شده است! به زودی تفاوت را بیان خواهیم کرد.

مراحل اجرای این برنامه را در زیر مشاهده می کنید :

PC	عددی که پردازنده روی **گذرگاه آدرس** خود قرار می‌دهد	مقداری که در این لحظه باید روی **گذرگاه داده** پردازنده ظاهر شود
0000	0000 (آدرس حافظه)	DB (دریافت کد عمل)
0001	0001	00 (دریافت کد عمل)

---- اجرای کد DB 00 ----

محتویات بافر شماره صفر	**0000** (آدرس ورودی)	**0001**

پین EN تراشه بافر صفر در این لجظه باید فعال شود.

47 (دریافت کد عمل)	0010	0010

---- اجرای کد 47 ----

0011	0011	(دریافت کد عمل) DB
0100	0100	(دریافت کد عمل) 01

---- اجرای کد 01 DB ----

0100	**0001**	**محتویات بافر شماره ۱**

پین EN تراشه بافر ۱ در این لحظه باید فعال شود.

0101	0101	(دریافت کد عمل) 80

---- اجرای کد 80 ----

0110	0110	(دریافت کد عمل) D3
0111	0111	(دریافت کد عمل) 02

---- اجرای کد 02 D3 ----

0111	**0010**	**محتویات ثبات A**

(که پردازنده روی گذرگاه داده قرار می دهد.)

پین LE در این لحظه باید فعال شود.

1000	1000	76

---- اجرای کد 76 ----

به لحظه‌ای که پین En بافر صفر باید فعال شود توجه کنید. به نظر می‌آید، در لحظه‌ای که آدرس 0000 روی گذرگاه آدرس می‌آید باید پین En فعال شود. این ایده کاملاً درست نیست؛ چون همانطور که در روند اجرای برنامه مشاهده می‌کنید وقتی پردازنده می‌خواهد اولین کد عمل برنامه را بخواند نیز آدرس 0000 را روی گذرگاه آدرسش قرار می‌دهد. متأسفانه چون در هر دو حالت پردازنده مشغول عملیات خواندن گذرگاه داده است، دیگر برای تمایز دو حالت فوق نمی‌توانیم مانند مثال قبل از سیگنالهای RD و WR استفاده کنیم. تنها تفاوت این دو حالت این است که پردازنده در هنگام دریافت کد عمل با **حافظه** و هنگام خواندن بافر با **ورودی/خروجی** در ارتباط است.

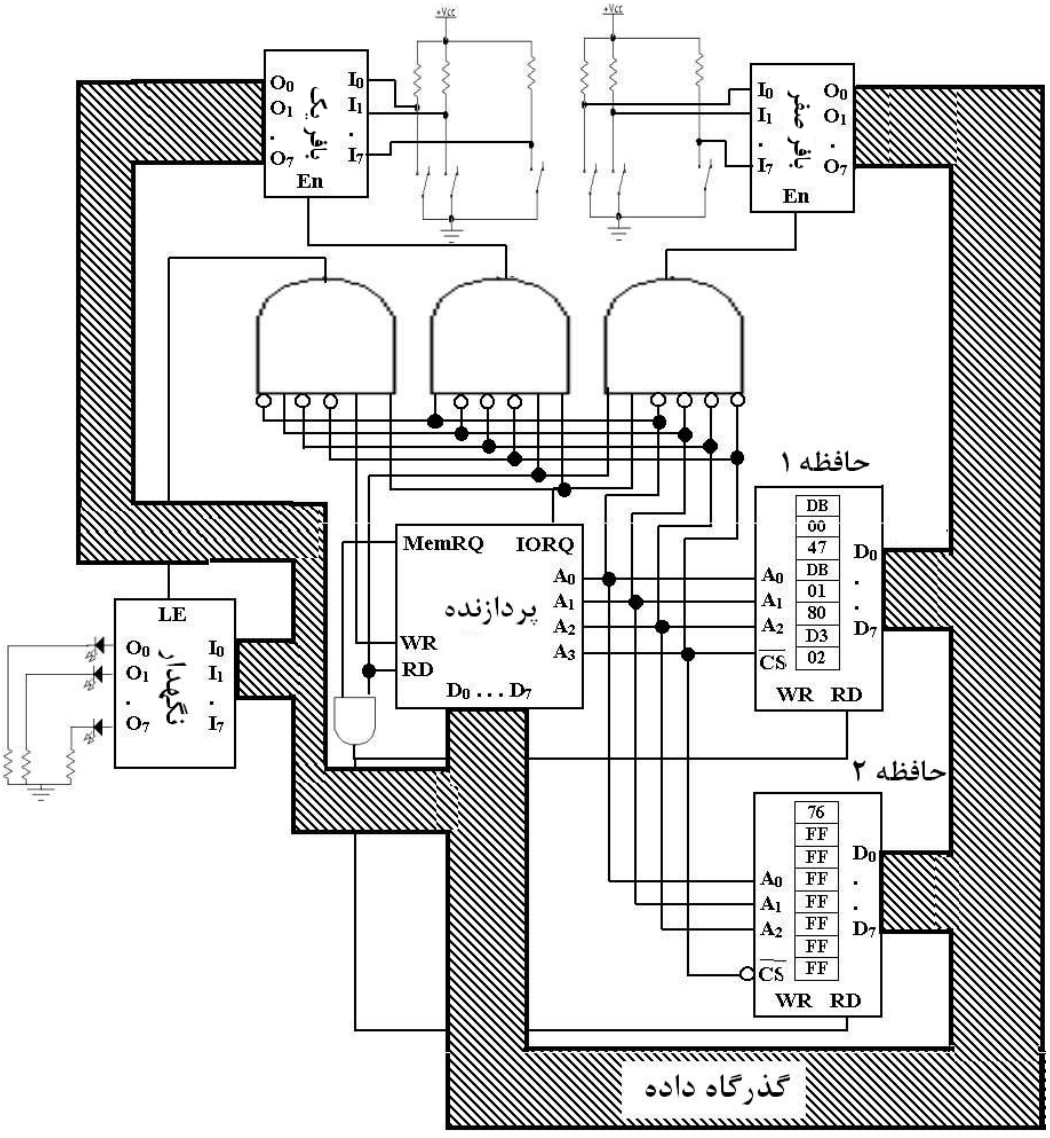
دو سیگنال مهم در پردازنده‌ها این مشکل را حل می‌کنند. هنگامی که پردازنده در حال تبادل اطلاعات با حافظه است **سیگنال MemRQ** و در هنگام تبادل اطلاعات با ورودی/خروجی **سیگنال IORQ** را فعال می‌کند.

بنابراین پردازنده در هنگام واکشی کد عمل از آدرس صفر به خانه صفر حافظه اول (MemRQ فعال و IORQ غیرفعال) و هنگام خواندن عدد اول به بافر سه‌حالته اول (MemRQ غیرفعال و IORQ فعال) مراجعه می کند.

به بیان دیگر، سیستم باید به نحوی طراحی شود که تنها در هنگام اجرای دستور IN کلیدها به گذرگاه داده پردازنده متصل شوند؛ این لحظه دو نشانه دارد که به کمک آن نشانه‌ها سیستم را طراحی می‌کنیم:

- فعال شدن سیگنالهای RD و IORQ
- قرار گرفتن آدرس ورودی روی گذرگاه آدرس پردازنده

جزییات مدار را در شکل زیر می‌بینید.



نکته دیگری نیز در این شکل وجود دارد. سیگنال RD حافظه به سیگنال RD پردازنده متصل نشده است؛ بلکه از خروجی یک گیت AND گرفته شده که ورودیهای آن سیگنالهای RD و MemRQ هستند. یعنی حافظه باید هنگامی برای خواندن فعال شود که سیگنالهای RD و MemRQ فعال باشند. اگر سیگنال RD حافظه به سیگنال RD پردازنده متصل شود، هنگامی که پردازنده می‌خواهد از یک ورودی با آدرسی مشترک با حافظه (مثلاً از ورودی شماره صفر) نیز اطلاعات بخواند، حافظه برای خواندن فعال خواهد شد که درست نیست.

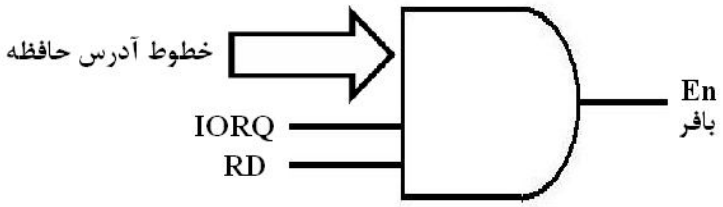
طراحی یک مدار رمزگشای آدرس کامل

پرسشی که ممکن است پیش آید این است که چرا در ورودی گیتهای AND فعالساز بافرها از سیگنال RD استفاده کرده‌ایم؟ در این سیستم دو آدرس صفر داریم که یکی برای خواندن از حافظه و یکی برای خواندن از ورودی استفاده می‌شود که به کمک سیگنالهای MemRQ و IORQ از هم جدا شده‌اند. در این سیستم از آدرس صفر برای هیچگونه خروجی (حافظه یا نگهدار) استفاده نشده است که برای تفکیک آن از آدرسهای صفر ورودی ناچار به استفاده از سیگنالهای RD و WR باشیم. بنابراین به نظر می‌آید استفاده از سیگنال RD در ورودی گیتهای AND فعالساز بافرها لزومی ندارد.

اصل زیر را همیشه (چه لازم باشد و چه لازم نباشد) رعایت می‌کنیم:

> در ورودی گیت AND فعالساز بافرهای ورودی، همیشه از سیگنالهای IORQ و RD استفاده کنید.

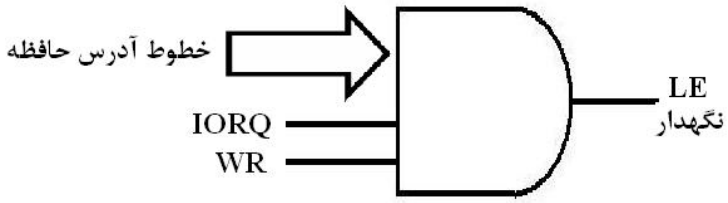
شکل کلی گیت AND فعالساز بافرهای ورودی به صورت زیر است:



پرسش دیگر این است که با وجود اینکه ما در این برنامه فقط در خروجی (نگهدار) می‌نویسیم و نوشتن در حافظه نداریم، چرا در ورودی گیت AND فعالساز نگهدار هم از سیگنال IORQ استفاده کرده‌ایم، در حالیکه به نظر می‌آید جایی که یک مکان حافظه و یک ورودی یا خروجی، آدرس مشابه داشته باشند و باید هر دو خوانده یا هردو نوشته شوند، از سیگنالهای MemRQ و IORQ برای تمایز آنها استفاده می‌کنیم؟

در ورودی گیت AND فعالساز تراشه‌های نگهدار خروجی، همیشه از سیگنالهای IORQ و WR استفاده کنید.

شکل کلی گیت AND فعالساز تراشه‌های نگهدار به صورت زیر است:

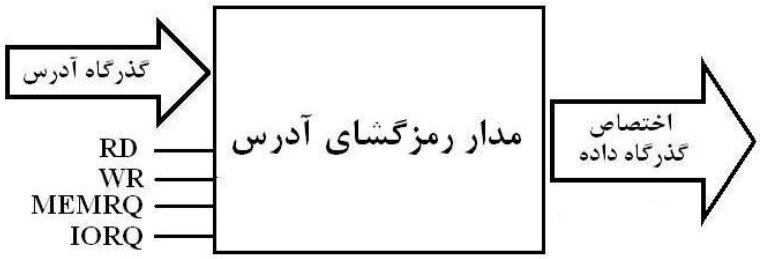


برای اینکه خود را درگیر جزییات مدار (آیا آدرسی مشترک داریم؟ کدام مخصوص خواندن و کدام ویژه نوشتن است؟ کدام مربوط به حافظه و کدام مربوط به ورودی/خروجی است؟ و...) سیگنالهای زیر را می‌سازیم و همیشه (چه لازم باشد و چه نباشد) از آنها استفاده می‌کنیم:

بنابراین از این پس:

- در ورودی گیت AND فعالساز بافرهای ورودی، همیشه از سیگنال IORD استفاده می‌کنیم.
- در ورودی گیت AND فعالساز تراشه‌های نگهدار خروجی، همیشه از سیگنال IOWR استفاده می‌کنیم.
- سیگنال RD حافظه‌هایی که محتویات آنها باید خوانده شود را به سیگنال MemRD متصل می‌کنیم.
- سیگنال WR حافظه‌هایی که در آنها باید اطلاعاتی نوشته شود را به سیگنال MemWR متصل می‌کنیم.

شکل زیر مدار کلی رمزگشای آدرس را نشان می‌دهد.



اصول کامل طراحی سیستم

- **اصل اول:** پردازنده در هنگام واکشی کد عمل یک دستور، محتویات ثبات PC را روی گذرگاه آدرس خود گذاشته و سیگنالهای RD و MemRQ خود را فعال می‌کند و انتظار دارد در پاسخ این کار، کد عمل مناسب روی گذرگاه داده‌اش ظاهر شود تا بتواند آن را بخواند و اجرا کند.
- **اصل دوم:** پردازنده هنگام خواندن محتویات یک خانه حافظه، شماره آن خانه را روی گذرگاه آدرس خود گذاشته و سیگنالهای RD و MemRQ خود را فعال می‌کند و انتظار دارد در پاسخ این کار، محتویات آن خانه حافظه روی گذرگاه داده‌اش ظاهر شود تا بتواند آن را بخواند.
- **اصل سوم:** پردازنده هنگام نوشتن عددی در یک خانه حافظه، شماره آن خانه را روی گذرگاه آدرس خود و عدد مورد نظر را روی گذرگاه داده‌اش قرار می‌دهد و سیگنالهای WR و MemRQ خود را فعال می‌کند.
- **اصل چهارم:** پردازنده هنگام ارسال یک داده به خروجی، آن داده را روی گذرگاه داده و آدرس خروجی مزبور را روی گذرگاه آدرسش قرار می‌دهد و سیگنالهای WR و IORQ خود را فعال می‌کند.
- **اصل پنجم:** پردازنده هنگام خواندن اطلاعات یک دستگاه ورودی، شماره (آدرس) آن ورودی را روی گذرگاه آدرس خود گذاشته و سیگنالهای RD و IORQ خود را فعال می‌کند و انتظار دارد در پاسخ این عمل، دستگاه ورودی مزبور اطلاعاتش را روی گذرگاه داده پردازنده قرار دهد.

اصل اول مربوط به زمان واکشی دستورات و چهار اصل دیگر به زمان اجرای دستورات مربوط می‌شوند.

از این پنج اصل، چند قانون کلی برای طراحی و سیم‌بندی سیستمها نتیجه می‌شود که باید آنها را همیشه به خاطر داشته باشیم:

- تنها یک گذرگاه داده در سیستم وجود دارد. بنابراین گذرگاه داده پردازنده و گذرگاه داده حافظه و ورودی تراشه نگهدار و خروجی تراشه بافر سه‌حالته همگی باید به هم متصل شوند.
- برای اتصال حافظه به پردازنده، خطوط آدرس آن حافظه را به **خطوط پایین آدرس پردازنده** متصل کرده و از خطوط بالای آدرس پردازنده برای انتخاب حافظه استفاده می‌کنیم. خطوط RD و WR حافظه(ها) باید به سیگنالهای MemRD و MemWR متصل شوند.
- برای اتصال خروجی به گذرگاه داده باید از تراشه نگهدار استفاده کنیم. نگهدار باید تنها هنگام اجرای دستور OUT مربوط به خودش یعنی وقتی آدرس آن خروجی روی گذرگاه آدرس پردازنده می‌آید و سیگنالهای WR و IORQ فعال می‌شوند، فعال شود و خروجی را به گذرگاه داده پردازنده متصل کند.
- برای اتصال خروجی به سیستم، باید به هر خروجی یک آدرس نسبت داده و این آدرس را هم در نوشتن برنامه و هم در طراحی مدار رمزگشای آدرس (گیت AND فعال‌کننده تراشه نگهدار) مورد توجه قرار دهیم.
- برای اتصال ورودی به گذرگاه داده باید از بافر سه‌حالته استفاده کنیم. بافر سه‌حالته باید تنها هنگام اجرای دستور IN مربوط به خودش یعنی وقتی آدرس آن ورودی روی گذرگاه آدرس پردازنده می‌آید و سیگنالهای RD و IORQ فعال می‌شوند، فعال شود و ورودی را به گذرگاه داده پردازنده متصل کند.
- برای اتصال ورودی به سیستم، باید به هر ورودی یک آدرس نسبت داده و این آدرس را هم در نوشتن برنامه و هم در طراحی مدار رمزگشای آدرس (گیت AND فعال‌کننده تراشه بافر سه‌حالته) مورد توجه قرار دهیم.

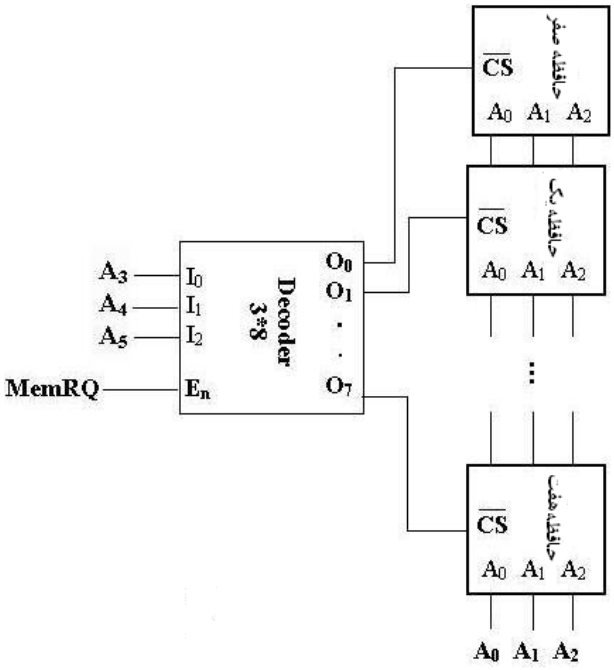
پرسش) اگر آدرس ورودیها ۹ و ۱۰ و آدرس خروجی ۱۱ باشد و بخواهیم نتیجه جمع را در خانه ۱۳ حافظه هم بنویسیم، مدار رمزگشای آدرس به چه شکل در می آید؟ (از سیگنالهای IORD و IOWR و MemRD و MemWR استفاده کنید).

پرسش) در مورد مسأله زمانبندی در طراحی سیستم مثال ۵ بحث کنید.

پرسش) خروجی سیستم ما یک عدد ۸ بیتی است که روی ۸ عدد دیود نوری نشان داده می‌شود و حداکثر آن ۱۱۱۱۱۱۱۱ یا ۲۵۵ است. حال اگر مجموعه کلید اول عدد ۱۵۰ و

مجموعه کلید دوم عدد ۲۰۰ را نشان دهند، چه اتفاقی می‌افتد[1] ؟ با نوشتن معادل دودویی اعداد فوق به پرسش پاسخ دهید.

پرسش) در مثالهای ذکر شده، حداکثر تعداد کدهای عمل برنامه‌ها، ۹ بایت بود. پردازنده فرضی ما چون دارای ۴ خط آدرس است حداکثر می‌تواند تا ۱۶ بایت (0000 تا 1111) را آدرس‌دهی کند. بنابراین اگر تعداد کدهای عمل برنامه‌ای بیش از ۱۶ بایت باشد، نمی‌توان از این پردازنده استفاده کرد.



فرض کنید پردازنده‌ای با ۶ خط آدرس (A_0 تا A_5) داریم. این پردازنده می‌تواند تا 2^6 یعنی ۶۴ بایت را آدرس‌دهی کند. حال فرض کنید برنامه‌ای به حجم ۶۰ بایت داریم و تنها حافظه‌های موجود، حافظه‌های ۸ بایتی است. بنابراین باید ۸ عدد از این حافظه‌ها را به پردازنده متصل کنیم.

الف) توضیح دهید مدار رمز گشای آدرس شکل بالا چگونه آدرس‌دهی مناسب ۶۴ بایت را فراهم می‌کند ؟ (خروجی دکودر، فعال پایین است).

ب) در صورت استفاده از مدار بالا، می‌توان سیگنال RD حافظه را مستقیماً به سیگنال RD پردازنده متصل کرد و نیازی به ساختن سیگنال MemRD نیست. چرا؟

روشهای اتصال ورودی/خروجی به سیستم

شاید تاکنون به این اندیشیده باشید که چرا اصلاً باید آدرسهای ورودی/خروجی با آدرسهای حافظه مشترک باشند تا برای متمایز کردن آنها به زحمت اسفاده از سیگنالهای IORD و IOWR و MemRD و MemWR بیافتیم؟ آیا بهتر نیست برای ورودی/خروجی‌ها از آدرسهایی استفاده کنیم که در آن آدرسها حافظه‌ای نصب نشده باشد تا دیگر نیازی به استفاده از سیگنالهای کنترلی نباشد؟

روشی که در مثال ۵ برای اتصال ورودی/خروجی به پردازنده آموختیم، به روش **اتصال ورودی/خروجی با نگاشت ورودی/خروجی (I/O Mapped I/O)** موسوم است. در این

[1] این مسأله به مشکل سرریز (***Overflow***) معروف است.

روش، آدرسهایی که به ورودی/خروجی‌ها نسبت داده می‌شود ممکن است با آدرسهای حافظه مشترک باشد و از سیگنالهای کنترلی برای متمایز کردن آنها استفاده می‌کنیم. دستورات IN و OUT در برنامه‌نویسی به این روش به کار می‌روند.

روش دیگری برای اتصال ورودی/خروجی به پردازنده وجود دارد که **اتصال ورودی/خروجی با نگاشت حافظه (Memory Mapped I/O)** نام دارد. در این روش، آدرسهای ورودی/خروجی‌ها را به گونه‌ای انتخاب می‌کنند که با آدرس خانه‌های حافظه یکسان نباشد و دیگر از سیگنالهای کنترلی برای رمزگشایی آدرس استفاده نمی کنند؛ به بیان دیگر آدرس ورودی/خروجی‌ها را در ادامه آدرسهای کدهای عمل برنامه در نظر گرفته و با ورودی/خروجی‌ها همانند خانه‌های حافظه رفتار می‌کنند تا سیستم ساده‌تر باشد.

در این حالت آدرس مشترکی در سیستم نخواهیم داشت و صِرف بیان آدرس یک موجودیت برای مشخص کردن آن کافی است و دیگر به سیگنالهای کنترلی برای رمزگشایی آدرس نیازی نخواهد بود (البته طبعاً سیگنالهای RD و WR پردازنده باید به حافظه متصل شوند؛ اما هدف دیگر رمزگشایی آدرس نیست).

به عنوان مثال اگر بخواهیم مثال ۵ را با این روش طراحی کنیم، آدرس ورودیها (بافرها) و خروجی (نگهدار) نمی‌تواند با آدرسهای حافظه (که در این سیستم از آدرس صفر تا ۸ به کدهای عمل برنامه اختصاص یافته است) یکسان باشد؛ مثلاً باید به بافرهای صفر و ۱ به ترتیب آدرسهای ۹ و ۱۰ و به تراشه نگهدار آدرس ۱۱ را اختصاص دهیم.یعنی دستور MOV A,[9] برای خواندن بافر شماره «صفر»، دستور MOV A,[10] برای خواندن بافر شماره «یک» و دستور MOV [11],A برای نوشتن در نگهدار مورد استفاده قرار می‌گیرد. برنامه سیستم در این روش به شکل زیر است:

```
MOV    A, [9] ;input 1
MOV    B, A
MOV    A, [10] ;input 2
ADD    A, B
MOV    [11], A ;output
HALT
```

روش ورودی/خروجی با نگاشت حافظه از نظر طراحی ساده‌تر است؛ چون دیگر نیازی به استفاده از سیگنالهای کنترلی نیست. اما چون در این روش برای ورودی/خروجی هم از دستور MOV باید استفاده شود و تمایزی بین خواندن و نوشتن در حافظه با خواندن و نوشتن در ورودی/خروجی نیست، سیستمهای طراحی شده با این روش از نظر فهم پیچیده‌تر هستند. در این حالت کسی که برنامه این سیستم را می خواند بدون مراجعه به

مدار نمی تواند بفهمد که آیا منظور از مثلاً آدرس ۱۲، آدرس یک خانه حافظه است یا یک واحد ورودی/خروجی؟ این موضوع فهم کلی سیستم را پیچیده می کند.

روش اتصال ورودی/خروجی با نگاشت ورودی/خروجی از نظر طراحی سخت افزاری (به خاطر استفاده از سیگنالهای کنترلی) مشکل‌تر و از نظر نرم افزاری ساده‌تر است. مزیت مهم این روش آن است که بر خلاف روش قبلی، در این روش یک ورودی یا خروجی می‌تواند دارای آدرس مشترک با یک خانه حافظه باشد که تفاوت آن دو از طریق سیگنالهای MemRQ و IOREQ معلوم می‌شود. به همین دلیل یک آدرس در یک سیستم می‌تواند در آنِ واحد آدرس یک ورودی و آدرس یک خروجی و آدرس یک خانه حافظه برای خواندن و آدرس یک خانه حافظه برای نوشتن باشد.

توجه کنید که در مدار مثال ۵ در هر ۱۶ بایت فضای آدرس‌دهی پردازنده حافظه نصب شده است. به همین جهت در این سیستم استفاده از روش اتصال ورودی/خروجی با نگاشت حافظه چندان معنایی ندارد (مگر اینکه سیستم را طوری طراحی کنیم که تنها در ۹ بایت اول فضای آدرس‌دهی، حافظه نصب شده باشد تا بتوان از بقیه آدرسها برای ورودی/خروجی استفاده کرد).

پرسش) علیرغم اینکه پردازنده فرضی ما دارای فضای آدرس دهی ۱۶ بایتی است، اما با استفاده از روش ورودی/خروجی با نگاشت ورودی/خروجی می‌توان به این پردازنده ۱۶ خانه حافظه بعنوان ورودی (دستور یا اطلاعات)، ۱۶ خانه حافظه به عنوان خروجی، ۱۶ ورودی و ۱۶ خروجی متصل شود. با استفاده از عملکرد سیگنالهای RD و WR و MemRQ و IORQ این موضوع را نشان دهید.

پرسش) سیستم مثال ۵ را از نظر سخت افزاری و نرم افزاری به گونه‌ای تغییر دهید که خروجی علاوه بر نمایش روی دیودهای نوری، در خانه شماره ۱۳ حافظه نیز نوشته شود. طراحی خود را با دو روش گفته شده انجام دهید.

پرسش) سیستمی طراحی کنید که محتویات یک ورودی به آدرس ۷ و یک ورودی به آدرس ۸ را بخواند و محتویات آنها را با محتویات خانه شماره ۱۵ حافظه جمع کند و نتیجه را در خانه شماره ۱۴ حافظه و خروجی شماره ۱۰ بنویسد.

با مطالبی که تاکنون گفته شد، عملکرد کلی پردازنده‌ها و چگونگی اتصال حافظه و ورودی/خروجی به آنها تا حدودی روشن شد. در فصول بعد جزییات بیشتری راجع به پردازنده های 8088/8086 را بیان خواهیم کرد.