ریزپردازنده

Microprocessor

Books

- ***The Z80 Microprocessor , Hardware , Software programming & interfacing***
- ***Author:*** *Burry B. Brey*
- ***Translator:*** *Hossein Nia*
- **Publisher:** Astane Ghodse Razavi(Beh Nashr

Books

Microcompiuter and Microprocessor : the 8080, 8085, Z-80 Programming , interfacing and trubleshooting

Publisher: Nass •
Pub.Date: 1381
Edition Turn: 3
ISBN: 964-6264-43-4-3
Pages: 719
Author: John E . Uffenbeck
Translator: Mahmmod Dayani •

Integration levels

* **SSI** (small scale integration)
» Introduced in late 1960s
» 1-10 gates

* **MSI** (medium scale integration)
» Introduced in late 1960s
» 10-100 gates

* **LSI** (large scale integration)
» Introduced in early 1970s
» 100-10,000 gates

* **VLSI** (very large scale integration)
» Introduced in late 1970s
» More than 10,000 gates

مقدمه

- با پیدایش و پیشرفت مدارهای مجتمع کم کم این گرایش بوجود آمد که کامپیوترهای *کوچک و ارزان* ساخته شود.

- پیشرفت تکنولوژی ساخت IC ها و پیدایش تکنولوژی MOS در دهه ۶۰ امکان ساخت مدارهای مجتمع با مقیاس بزرگ (LSI) را در اوایل دهه ۷۰ بوجود آورد.

- استفاده از تکنولوژی LSI باعث میشد که بتوان یک پردازنده (processor) را بطور کامل به صورت یک تراشه ساخت .

- پردازنده بخش مرکزی یک کامپیوتر است که با گرفتن دستورات مختلف میتواند عملیات جمع و تفریق و مقایسه و یا عملیات منطقی را انجام دهد.

مقدمه ادامه:

- **پردازنده هایی که بصورت تک تراشه ساخته میشوند بدلیل اندازه کوچکی که دارند ریزپردازنده (microprocessor) نامیده میشوند.**

- **اولین ریزپردازنده ایی که به بازار آمد تراشه ۴۰۰۴ شرکت اینتل بود که بعنوان یک تراشه قابل برنامه ریزی بروی ماشین های حساب طراحی گردید.**

- **این تراشه بدلیل قابلیت برنامه ریزی که داشت توانست به سرعت در بخش های مختلف صنعتی بعنوان بخش اصلی بعضی از سیستمهای کنترلی مورد استفاده قرارگیرد.**

- **استفاده از یک ریزپردازنده باعث میشود که بتوان از یک تراشه تنها با تغییر برنامه آن در بخشهای مختلف استفاده نمود.**

- **ریزپردازنده جای خود را در صنعت (مانند کنترل ماشین های صنعتی) و در مصارف خانگی ، علمی و آموزشی باز کرده است.**

-

کاربرد ریزپردازنده ها:

۱- جایگزین نمودن سیستمهای دیجیتالی قدیمی بوسیله سیستم های مبتنی بر ریزپردازنده .

۲- ساختن کامپیوترهای بسیار کوچک (میکروکامپیوترها)

کاربرد ادامه:

- در طراحی یک سیستم دیجیتال، عمده ترین هدف به حداقل رساندن هزینه میباشد.
  این خود به حداقل رساندن تراشه هارا به دنبال خواهد داشت.

- برای به حداقل رساندن تعداد تراشه ها ، مناسبترین راه استفاده از تراشه های قابل برنامه ریزی میباشد.

- تراشه ها ی قابل برنامه ریزی :
  ۱- prom
  ۲- PLA
  ۳- microprocessors

uP به عنوان یک تراشه قابل برنامه ریزی :

- برنامه های مربوط به یک uP در یک یا چند تراشه حافظه که در کنار uP قرار دارند ذخیره میشود.

- دستورات هر برنامه متوالیا" توسط uP خوانده و به اجرا در میآید.

- با تغییر برنامه موجود در حافظه ، ازیک uP می توان در سیستم های مختلف استفاده نمود.
- استفاده از uP ها جهت ساختن سیستم های دیجیتالی ساده تر و سریعتر ازساخت سیستم های دیجیتال قدیمی است.

- زیرا مهارت های ساخت مدار های چاپی و لحیم کاری تراشه های مختلف بروی آنها جای خود را به وارد نمودن برنامه های نوشته شده به درون حافظه میدهد.

بزرگترین عیب سیتم های مبتنی بر ریزپردازنده سرعت کم آنها نسبت به سرعت سیستم های دیجیتال مشابه می باشد.

- تعداد تراشه های لازم برای ساخت یک ریز کامپیوتر :

۱- uP ۲- ROM ۳- RAM ۴- I/O

- استفاده از ریز کامپیوتر ها برای ساخت سیستم هایی که از حد مشخصی ساده تر هستند مقرون به صرفه نمی باشد.

- در این گونه موارد می توان از PROM و PLA و یا حتی از تراشه های MSI و SSI استفاده نمود.

اوایل دهه ۷۰ تکنولوژی LSI امکان ساخت ریزپردازنده ها را بوجود آورد.

- قراردادن واحد کنترل و واحد ریاضی و منطقی در یک تراشه باعث شکل گرفتن ریزپردازنده ها گردید.
- از نظر ساختمان داخلی ریزپردازنده ها را به سه دسته عمده تقسیم می کنند.

۱- ریزپردازنده های همه منظوره

General Purpose Microprocessors

۲- ریزکنترل کننده ها یا ریزکامپیوتر

Unit chip – Microcontrollers

۳- پردازنده های لایه ایی

Bit – Slice microprocessors

History

Company	4 bit	8 bit	16 bit	32 bit	64 bit
intel	4004 4040	8008 8080 8085	8088/6 80186 80286	80386 80486	80860 pentium
zilog		Z80	Z8000 Z8001 Z8002		
Motorola		6800 6802 6809	68006 68008 68010	68020 68030 68040	

پارامترهای تعیین کننده قدرت یک ریز پردازنده:

۱- word size :

- زمانی که طول ثباتها بزرگتر باشد، سرعت عملکرد پردازنده ها نیز بیشتر است.
- مدیریت و کار با دستورالعمل های بزرگتر و اجزای اطلاعاتی بزرگتر ممکن می شود.

۲- تعداد خطوط گذرگاه داده :

- از طریق گذرگاه داده است که داده ها می توانند از RAM به وسایل ورودی و خروجی و بلعکس انتقال داده شوند.
- هرچه این مقدار بیشتر باشد سرعت کامپیوتر بیشتر است زیرا که هر بار اطلاعات بیشتری جهت پردازش بازیابی می شوند.

پارامترهای تعیین کننده قدرت یک ریز پردازنده: ادامه

۳ – تعداد خطوط گذرگاه آدرس :

- بزرگ بودن گذرگاه آدرس بدین معنا است که مقدار حافظه قابل آدرس دهی پردازنده بیشتر است.

- قابلیت آدرس دهی حافظه های بزرگ دو مزیت مهم دارد.

۱- کاربرد برنامه های پیچیده و کارآمدتر آسان میشود.

۲- Multitasking آسانتر میشود.

عملکرد ریزپردازنده ها

- ریزپردازنده قطعه مرکزی هر سیستم میتنی برریزپردازنده میباشد.
- ریزپردازنده عملیات انجام شده توسط وسایل جانبی یک سیستم را طبق برنامه ذخیره شده کنترل و قابلیت عملیات ریاضی و منطقی را دارد.
- uP دستورالعمل های یک برنامه را فراخوانی (fetch) ، رمزگشایی (decode) و درنهایت اجرا (execute) میکند.
- uP به حافظه و I/O رجوع میکند تا تبادل داده ایی انجام دهد وهم چنین به سیگنالهای کنترلی دیگر وسایل پاسخ میدهد.

سیگنال های دیگر وسایل می تواند uP را به یکی از عملیات زیر وادارد :

۱- Reset : اجرای برنامه از یک محل اولیه شروع میشود.

۲- Wait : uP را وادار به منتظر ماندن جهت دسترسی به محلی از حافظه میکند

۳- Interrupt : باعث ایجاد وقفه در اجرای برنامه جاری وشاخه شدن به محلی از حافظه می شود که زیر برنامه مورد نظر جهت سرویس دادن به وسایل خروجی که درخواست وقفه نموده است قراردارد.

۴- suspention : باعث توقف عملیات و شناور نمودن سرهای گذرگاه آدرس و داده شده و اجازه ارتباط مستقیم به حافظه (DMA) جهت خواندن و نوشتن را توسط وسایل جانبی را می دهد.

معماری ریزپردازنده ها

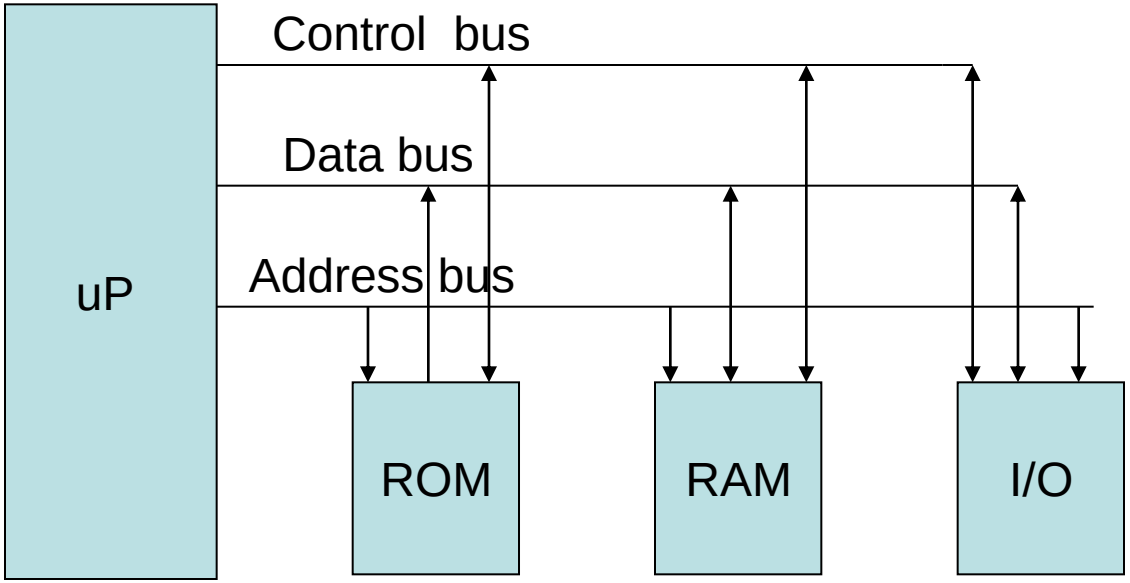
- یک سیستم مبتنی بر ریزپردازنده شامل :

uP ، RAM، ROM و I/O میباشد که به عنوان مجموعه ایی از ثبات های قابل آدرس دهی در نظر گرفته میشود.

- آن دسته از ثباتهایی که در داخل ریزپردازنده قرار دارند ثباتهای داخلی یا internal registers نامیده میشوند.

- ثباتهایی که در داخل ROM,RAM و I/O قرار دارند را ثباتهای خارجی یا external registers می نامند.

Microprocessor based system



معماری ریزپردازنده ادامه:

- به مجموعه ثباتها و مراودات داده ایی ممکن بین این ثباتها را معماری سیستم (system Architecture) می نامند.

- انواع ثباتهای داخل ریزپردازنده و ارتباط داده ایی بین این ثباتها را معماری ریزپردازنده (up Architecture) می نامند.

- یک سیستم مبتنی بر ریزپردازنده وظایف خود را با انتقال داده ها (transfer) و تغییر شکل داده ها (transform) در درون ثباتها انجام می دهد.

- عموما" تغییر شکل داده ها در ثباتهای داخلی که معمولا از نوع عملیاتی (operational) هستند شکل می گیرید.

- ثبات های عملیاتی برخلاف ثبات های حافظه ایی عملیات ریاضی و منطقی را بروی داده های مورد نظر انجام می دهند و باعث تغییر شکل می شوند.

معماری ریزپردازنده ادامه:

- یک ریزپردازنده عمل انتقال داده ها و تغییر شکل داده ها را طبق دستورالعمل های برنامه های کاربردی که از داخل حافظه ROM سیستم خوانده می شود کنترل وهمزمان می نماید.

- ثبات های دیگر وسایل یک سیستم مبتنی برریزپردازنده از طریق گذرگاه های خارجی با گذرگاه سیستم اتصال پیدا می کنند. این گذرگاه ها شامل گذرگاه آدرس ، داده و کنترل میباشند.

دو مجموعه معادل سیگنال های کنترلی برای خواندن ونوشتن ثباتهای خارجی وجود دارد :

- یک مجموعه شامل سیگنال های ’(RD) و ’(WR) و (IO/M’) .

- سیگنال ’RD زمانی از ریزپردازنده تولید میشود که بخواهد داده ایی را از حافظه یا I/O بخواند.
- سیگنال ’WR بعدا زمانی که داده مورد نظر بر روی گذرگاه داده قرار گرفت تولید تا در حافظه و یا I/O نوشته شود.
- سیگنال ’IO/M مشخص کننده این است که وسیله آدرس شده توسط ریزپردازنده جهت خواندن و نوشتن حافظه است یا I/O.

مجموعه معادل سیگنال های قبل که توسط ریزپردازنده تولید میشوند عبارتند از :

- سیگنال خواندن از حافظه MEMR’
- سیگنال نوشتن در حافظه MEMW’
- سیگنال خواندن از I/O I/OR’
- سیگنال نوشتن در I/O I/OW’

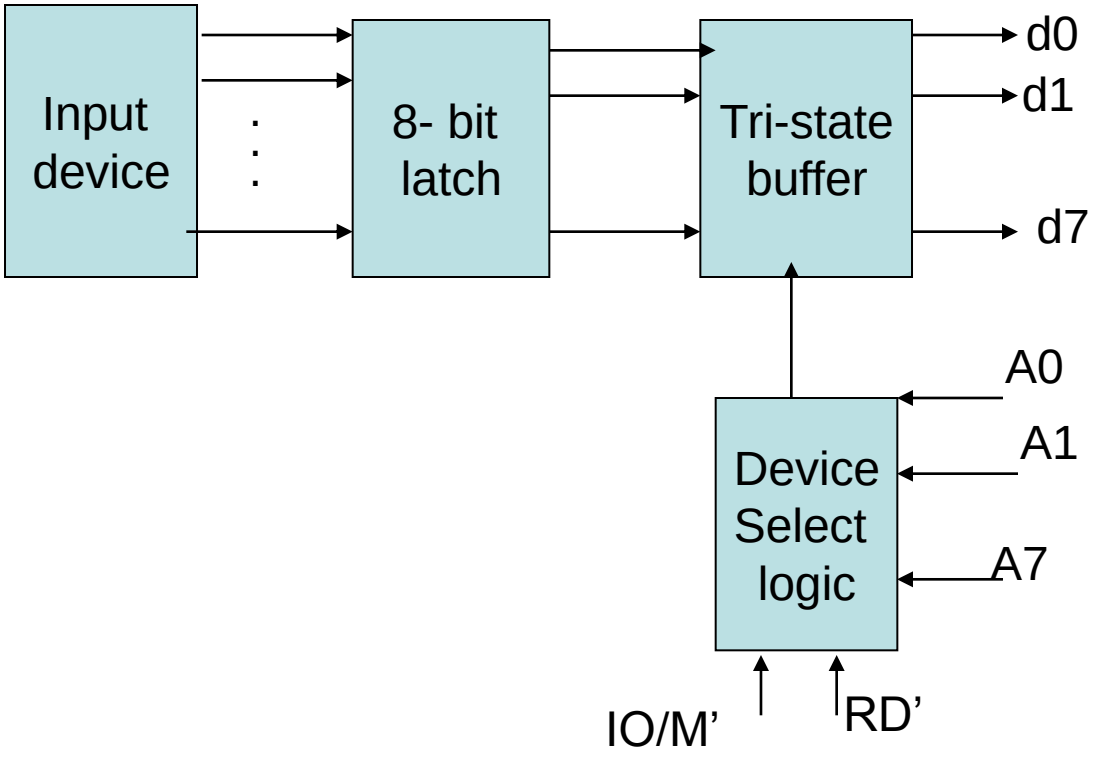
وسایل ورودی input device

- وسایل خارجی یا جانبی که برای یک ریزپردازنده داده های ورودی تولید میکند را وسایل ورودی می نامند.
- وسایل ورودی شامل محدوده وسیعی از وسایل الکترونیکی و یا الکترومکانیکی می باشند.
- از یک کلید ساده تا یک سیستم مبتنی بر ریزپردازنده دیگر می تواند به عنوان وسایل ورودی یک ریزپردازنده بکار گرفته شوند.
- داده تولید شده توسط یک وسیله ورودی به صورت موقت در یک ثبات (پورت ورودی) ذخیره می گردد تا توسط ریزپردازنده خوانده شود.

وسایل ورودی ادامه:

- جهت وارد نمودن محتویات ثبات ورودی(پورت ورودی) به ریزپردازنده خروجی آن از طریق بافر سه حالته به گذرگاه داده وصل میشود.
- بافر سه حالته از طریق یک مدارکنترلی (device select logic) که خود توسط سیگنال های RD’ و IO/M’ خطوط کنترل ریزپردازنده و آدرس وسیله مورد نظر از گذرگاه آدرس تحریک میشود فعال میگردد.
- سیگنال RD’ زمانی توسط ریزپردازنده تولید میگردد که دستورالعمل مورد اجرا یک دستور وارد کردن داده از وسیله ورودی باشد.

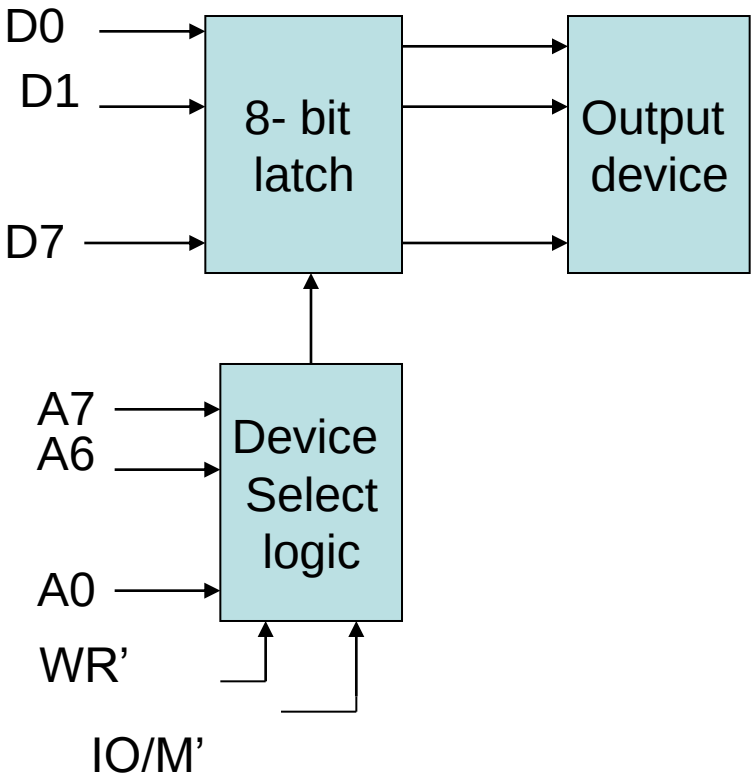
Input Port


Input device
8- bit latch
Tri-state buffer
d0
d1
d7
A0
A1
A7
Device Select logic
IO/M'
RD'

وسایل خروجی Out Put Device

- داده ایی که قرار است از یک سیستم مبتنی بر ریزپردازنده خارج شود، در یک ثبات که به گذرگاه داده متصل شده است قرار میگیرد.
- این ثبات پورت خروجی نام دارد که با سیگنال کنترلی که از device select logic خارج می گردد تنظیم میشود.
- سیگنال کنترلی توسط اعمال آدرس پورت خروجی مورد نظر و سیگنال های WR’ و IO/M’ تولید می گردد.
- آدرس پورت خروجی که قرار است داده به آن ارسال گردد قسمتی از دستورالعمل می باشد.

Out Put Port

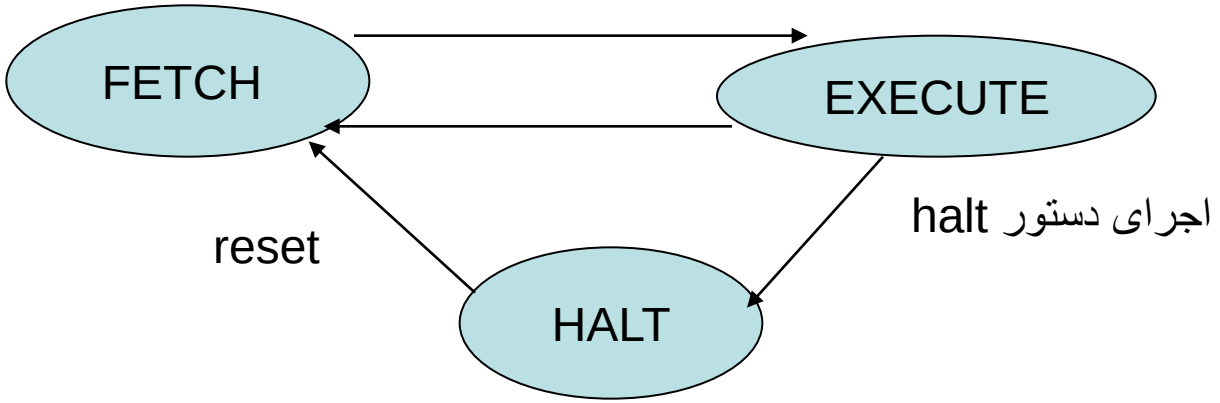

D0
D1
D7
8- bit
latch
Output
device
A7
A6
A0
Device
Select
logic
WR’
IO/M’

واحد کنترل control unit

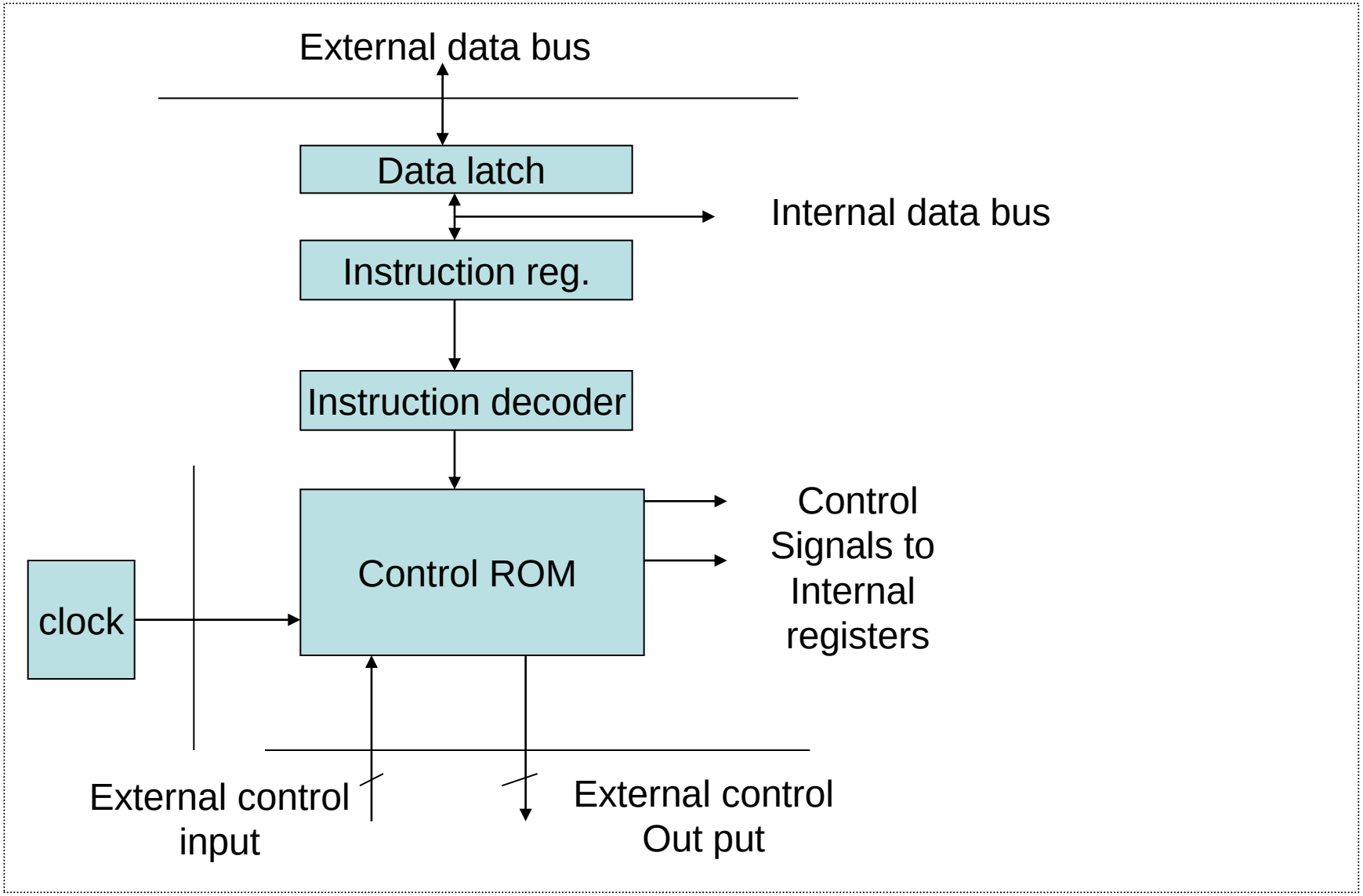
- واحد کنترل یک ریزپردازنده عمل تنظیم (synchronizes) و کنترل تمامی انتقال داده و تغییرات داده ایی را به عهده دارد.
- تمامی عملیاتی که به ریزپردازنده نسبت داده می شود اعمالی است که توسط واحد کنترل انجام میشود.
- عملکرد اصلی ریزپردازنده توسط واحد کنترل تنظیم می گردد و شامل عملیات چرخشی (cyclical) فراخوانی ترتیبی و اجرای دستورالعمل ها میباشد.
- واحد کنترل با دریافت سیگنال clock اصلی سیستم سیگنال ها ی مربوط به انتقال داده ها و تغییر شکل داده ها را تولید می کند.

واحد کنترل ادامه :

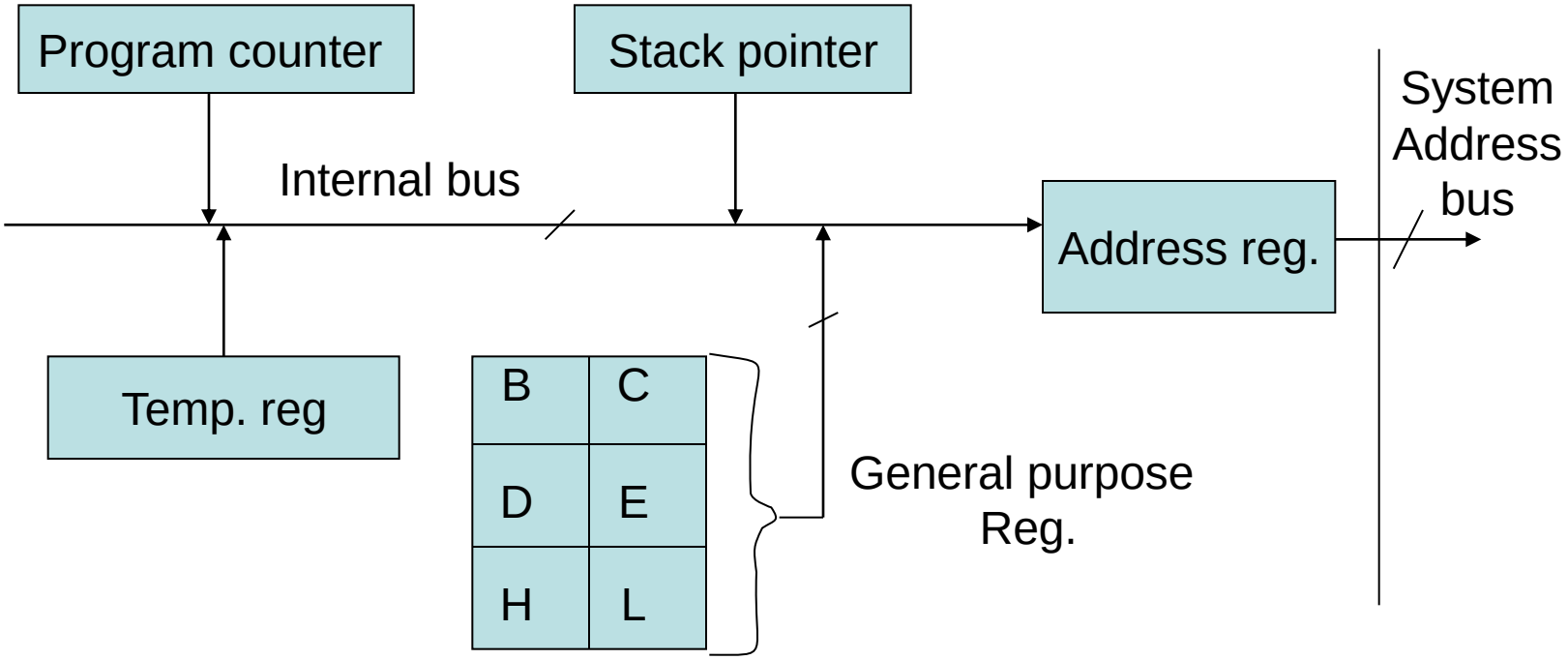
- واحد کنترل به صورت معمولی بین سیکل فراخوانی و اجرا چرخش می کند مگر اینکه دستور HALT را اجرا نماید ، که در این صورت در حالت توقف قرار میگیرد. جهت خارج شدن از حالت توقف سیستم باید reset شود.



واحد کنترل


External data bus
Data latch
Internal data bus
Instruction reg.
Instruction decoder
Control ROM
clock
Control Signals to Internal registers
External control input
External control Out put

ثبات های داخلی



اجرای دستورالعمل

- واحد کنترل از ثباتی بنام program counter استفاده میکند تا پی ببرد که چه دستورالعملی باید اجرا گردد.
- PC یک ثبات عملیاتی است که همیشه دارای آدرس دستور بعدی است که باید اجرا گردد. یا حاوی آدرس بعدی کلمه ایی از یک دستور چند کلمه ایی است.
- عملیاتی بودن این ثبات بدلیل قابلیت increment شدن با دستور واحد کنترل می باشد.
- وقتی یک ریزپردازنده reset میشود، واحد کنترل ثبات pc را صفر میکند تا به آدرس شروع برنامه اشاره کند.

اجرای دستورالعمل ادامه:

- برای بدست آوردن اولین کلمه دستورالعمل آدرس موجود در pc برروی گذرگاه آدرس قرار میگیرد.

- جهت انجام ابن کار واحد کنترل محتوی pc را به ثبات آدرس (.address reg) انتقال میدهد و به pc یکی اضافه میکند تا به دستورالعمل بعدی اشاره کند.

(خروجی ثبات آدرس پایه های آدرس یک uP می باشد)

- واحد کنترل سپس سیگنال خواندن را تولید میکند تا داده آدرس شده از محل حافظه به uP انتقال یابد.

اجرای دستورالعمل ادامه:

- داده مورد نظر از طریق بافر داده (data buffer) به داخل uP و سپس به ثبات دستورالعمل (IR) انتقال میابد.
- ثبات های داخل ریزپردازنده از طریق گذرگاه داده داخلی با هم مرتبط میشوند.
- اولین کلمه یک دستورالعمل opcode آن می باشد.
- Opcode عملیات مورد نظر جهت اجرای یک دستورالعمل را به واحد کنترل می فهماند.
- خروجی IR رمزگشایی گردیده و توسط واحد کنترل مورد استفاده قرار میگیرد تا توالی عملیاتی که سبب اجرای یک دستورالعمل میگردد به وجود آید.

اجرای یک دستورالعمل ادامه:

- opcode موجود در IR به آدرس شروع محلی واقع در ROM کنترلی که درآن محل توالی از ریز دستورالعمل ها قرار دارد اشاره میکند.
- هر دستورالعمل در یک مجموعه ایی از دستورالعمل های ثابت یک uP با اجرای متوالی ریز دستورالعمل هایی که در رابطه با آن دستورالعمل بخصوص می باشد توسط واحد کنترل اجرا می گردد.
- در uP تک تراشه ایی ریز دستورالعمل ها (micro instructions) و مجموعه دستورالعمل ها توسط سازنده جایگزین یا قرار میگیرد.

اجرای دستورالعمل ادامه :

- یک دستورالعمل اغلب ممکن است از یک یا چند کلمه تشکیل شده باشد.
- کلمه اول همیشه مبین opcode و کلمات دوم و سوم مشخص کننده آدرس یا داده ثابتی می باشد.
- بعد ازاینکه opcode فراخوانی و در داخل ثبات دستورالعمل قرار گرفت تفسیر(decode) آن مشخص میکند که به کلمات یا بایتهای اضافی دیگر نیاز دارد یا نه.
- بایتهای (کلمات) یک دستورالعمل چند بایتی در حافظه بصورت پشت سرهم قرار میگیرند.

اجرای یک دستورالعمل ادامه :

- جهت فراخوانی کامل یک دستورالعمل چند بایتی، بایتهای اضافی آن دستور که در حافظه باقی ماندند، باید به داخل ریزپردازنده انتقال تا درنهایت اجرا شوند.
- هر عمل مراجعه به حافظه جهت فراخوانی (read) یک کلمه از دستور یا نوشتن (write) داده ایی در حافظه را memory reference می نامند.
- اگر دستوری دو بایتی باشد دومین رجوع به حافظه نیاز بوده تا بایت دوم دستورالعمل به داخل ریزپردازنده آورده شود.
- مقصد بایت دوم دستورالعمل بستگی به نوع دستور دارد که اغلب در ثبات موقتی ذخیره می شود.
- ثبات های موقتی توسط واحد کنترل جهت ذخیره کردن داده ها یا آدرس هایی که قسمتی از یک دستورالعمل میباشند استفاده می گردد تا اینکه آنها به ثبات های دیگر منتقل یا به عنوان عملوند در محاسبات شرکت کنند.

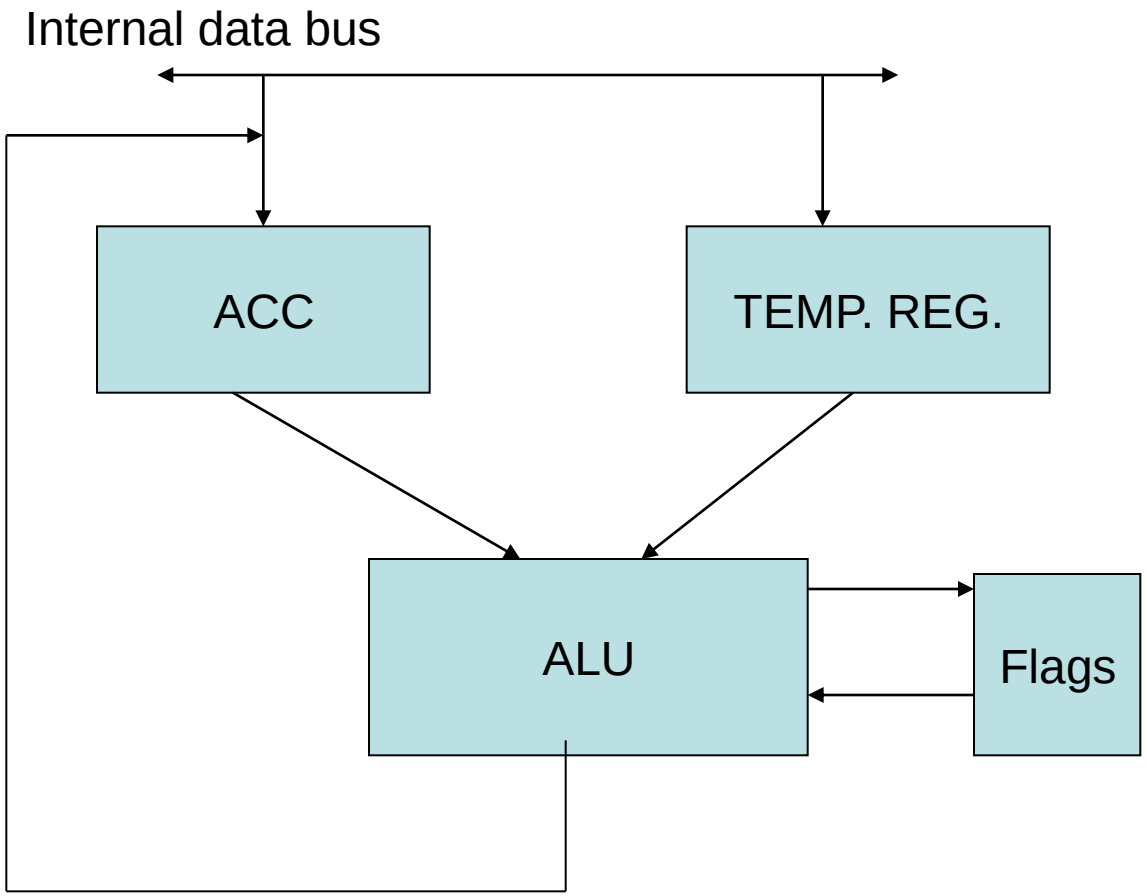
ثبات های همه منظوره general purpose reg.

- از این ثبات ها جهت نگهداری نتایج حاصل از عملیات بصورت موقت یا حتی آدرس دیگر محل هایی که ریزپردازنده باید با آنها ارتباط برقرار کند استفاده میشود.
- نگهداری نتایج در این ثبات ها بر خلاف بردن آنها به حافظه RAM از لحاظ صرفه جویی در وقت بهتر است.
- سرعت اجرایی دستورالعمل هایی که با ثبات های همه منظوره در ارتباط هستند بیشتر است ، زیرا آن دستور العمل جهت اجرا نیازی به رجوع به حافظه ندارد.

واحد ریاضی و منطقی
Arithmetic and Logic Unit

- عملیات ریاضی و منطقی که بروی یک و یا دو عملوند صورت میگیرد عمل transformation داده را که کار اصلی ریزپردازنده است را شکل میدهد.
- ALU قادر به انجام عملیات زیر بروی داده های باینری می باشد.
- ۱- جمع وتفریق باینری
- ۲- عملیات منطقی مانند AND ,OR, XOR
- ۳- مکمل گیری Complement
- ۴- چرخش به راست وچپ

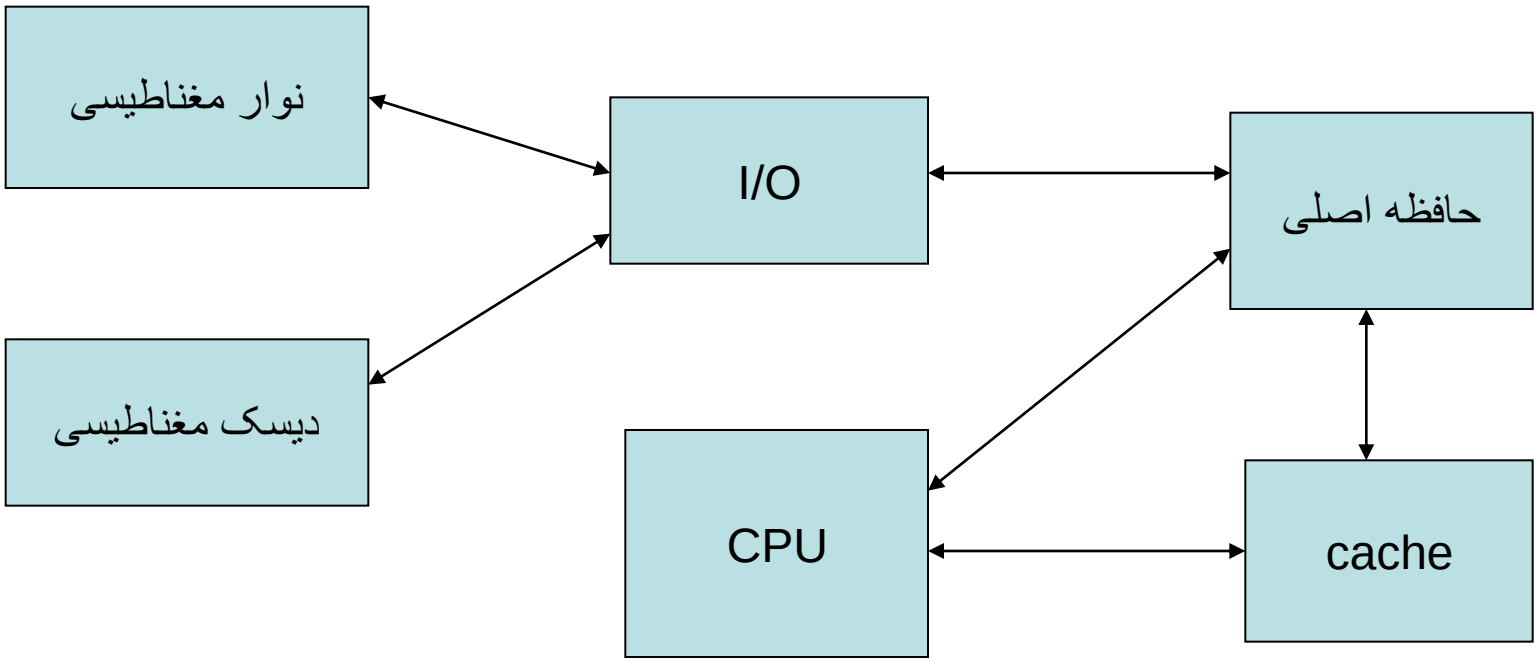
واحد ریاضی و منطقی
Arithmetic and Logic Unit



حافظه های سلسله مراتبی
memory hierarchy

- حافظه سلسله مراتبی شامل تمام وسایل ذخیره کننده اطلاعات سیستم کامپیوتری میباشد.
- از حافظه های کمکی با ظرفیت زیاد وسرعت کم تا حافظه اصلی نسبتا" سریع و بلاخره حافظه کوچکتر و بسیار سریع حافظه نهان (cache) میباشد.
- دلیل بکار بردن دو یا سه نوع حافظه با سرعت های مختلف یک مسله اقتصادی است.
- هرچه ظرفیت حافظه بیشتر باشد بهای واحد بیت آنها کمتر میشود و زمان دستیابی به اطلاعات افزایش می یابد.

حافظه های سلسله مراتبی



حافظه های سلسله مراتبی ادامه :

- هر چه سرعت حافظه بیشتر باشد، بهای واحد بیت آن نیز بالاتر است.

- حافظه نهان بخشی از برنامه و اطلاعات را ذخیره می کند که در جریان اجرای برنامه ، CPU بیشترین مراجعه به آنها را دارا است. در صورتی که حافظه های کمکی آن قسمت از برنامه را ذخیره می کنند، که فعلا مورد نیاز CPU نمیباشد.

- معمولا نسبت سرعت بین حافظه اصلی و حافظه نهان نسبت ۱ به ۷ است و زمان متوسط دسترسی به اطلاعات در حافظه های کمکی معمولا حدود ۱۰۰ برابر بیشتر از حافظه اصلی است.

- *هدف از حافظه های سلسله مراتبی این است که با حداقل هزینه حداکثر سرعت دستیابی به اطلاعات در یک سیستم کامپیوتری حاصل شود.*

حافظه memory

- وظایف کلی حافظه عبارتند از :

۱- ذخیره نمودن برنامه.
۲- فراهم نمودن داده برای cpu در صورت درخواست.
۳- دریافت نتیجه از cpu جهت ذخیره نمودن.

حافظه ادامه:

- حافظه های یک میکرو کامپیوتر بر اساس کلمه (word) ساخته شده اند. (1k کلمه یا 4k کلمه)

- برای دستیابی به محتوی یک حافظه به دو ثبات نیاز میباشد:

۱- MAR (memory address register) که دارای آدرس کلمه ایی است که نیاز به دست یافتن به آن میباشد.

۲- MDR (memory data register) که دارای داده ایی است که باید در داخل حافظه نوشته یا از داخل حافظه خوانده شود.

مثال : یک حافظه با ظرفیت 1k*8 دارای ۱۰۲۴ کلمه ۸ بتی میباشد.

حافظه ادامه:

- بطور کلی حافظه به دو صورت عمل میکند:
  ۱- READ (خواندن) ۲- WRITE (نوشتن)
- یک حافظه در حال خواندن است اگر اطلاعاتی در یک آدرس بخصوص در حافظه مورد نیاز باشد.
- برای خواندن حافظه :

۱- آدرس محلی که باید داده از آن خوانده شود در MAR قرار میگیرد.

۲- فرمان Read توسط واحد کنترل صادر میشود.

۳- داده ایی که باید خوانده شود از حافظه به MDR منتقل میگردد که در این حالت داده از طریق گذرگاه داده در اختیار سیتم قرار میگیرد.

توجه: عمل خواندن باعث تخریب محتوی محل خوانده شده نمی شود.

حافظه ادامه:

- برای نوشتن مراحل زیر صورت میگیرد:

۱- آدرس محلی از حافظه که داده باید در آن نوشته شود در MAR قرار میگیرد

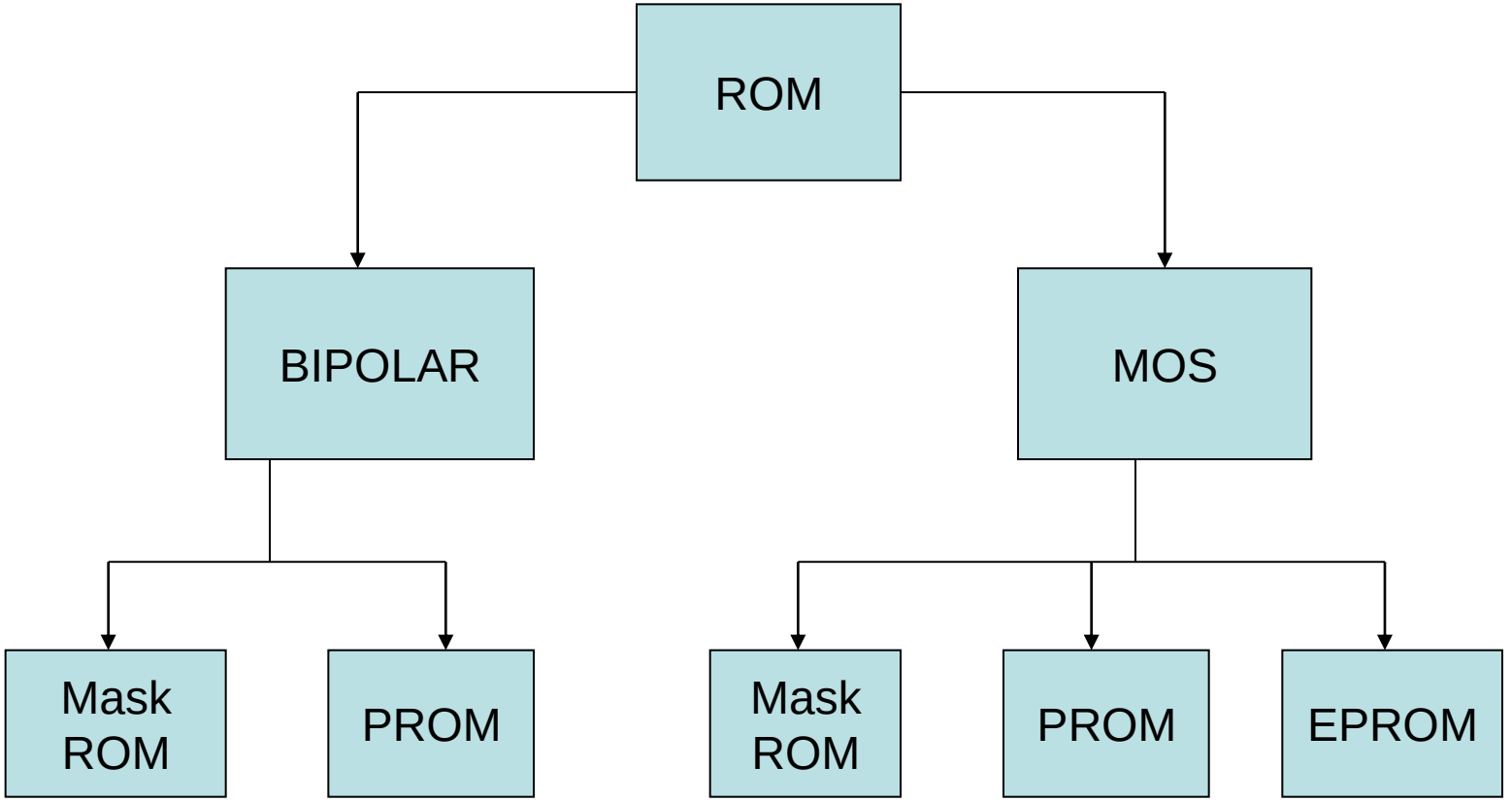
۲- داده ایی که باید نوشته شود در MDR قرار میگیرد.

۳- فرمان نوشتن از طریق خط R/W صادر میگردد و داده در

محل آدرس شده منتقل میگردد.

توجه: عمل نوشتن محتویات قبلی یک حافظه را از بین می برد.

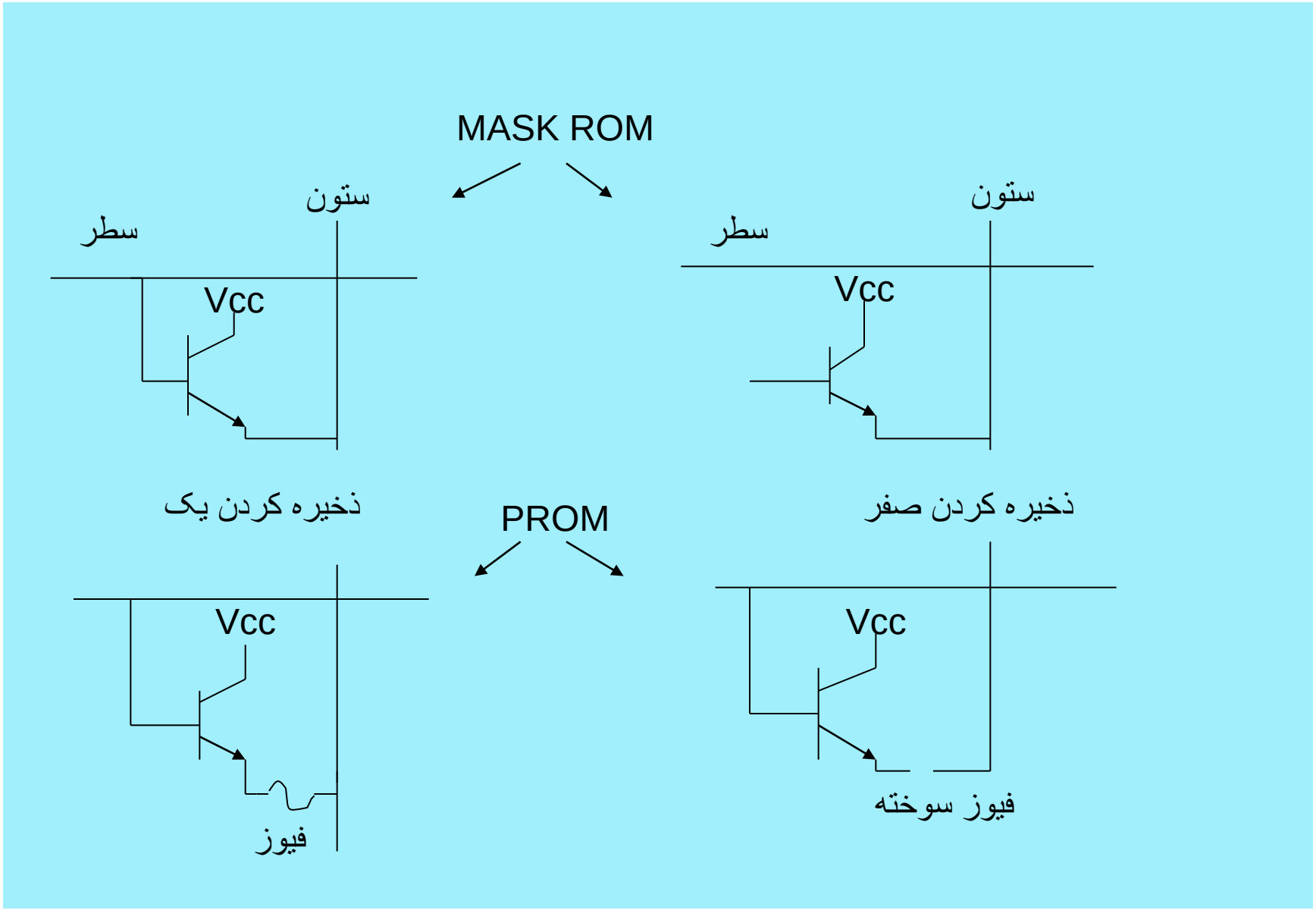
ROM حافظه نیمه هادی



Read-Only Memory

- uP can read instructions from ROM quickly
- Cannot write new data to the ROM
- ROM remembers the data, even after power cycled
- Typically, when the power is turned on, the microprocessor will start fetching instructions from the still-remembered program in ROM (bootstrap)

حافظه ROM ادامه:



حافظه ROM ادامه :

- یک ROM با ظرفیت ۱ کیلو بیت بصورت 256*4 را درنظر بگیرید.
- در حالت معمولی ساختمان این حافظه به یک دیکودر 8*256 نیاز دارد و این حافظه از ۲۵۶ سطر (کلمه) و ۴ ستون (داده) تشکیل می گردد.
- پیچیده گی (complexity) این حافظه از حاصل جمع خطوط خارج شده از دیکودر و تعداد ستونها (داده ها) بدست می اید. ۲۵۶ +۴ =۲۶۰

ROM عملی (ماتریسی)

- برای کاهش پیچیدگی و کاهش اندازه دیکودر به کار گرفته شده در یک حافظه ROM از طراحی ROM ماتریسی استفاده میشود.

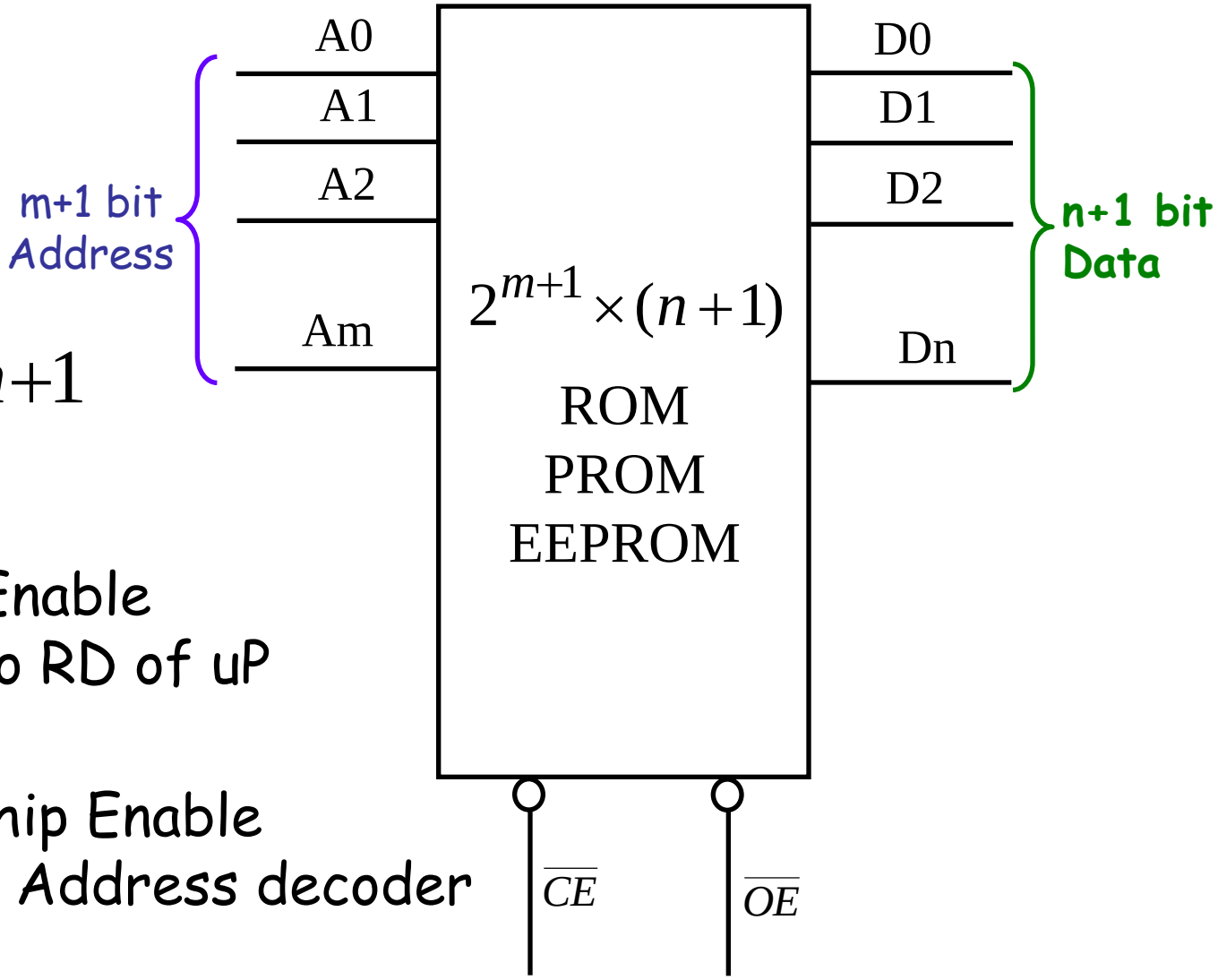
- یک ROM با ظرفیت ۱ کیلو بیت (256*4) را بصورت ماتریسی طراحی کنید.

در این ROM به وسایل زیر نیاز میباشد:

- یک مخزن 32*32
- یک دیکودر 5*32
- ۴ مالتی پلکسر 8*1

پیچیده گی این سیستم برابر است با ۳۲+۳۲ =۶۴

ROM

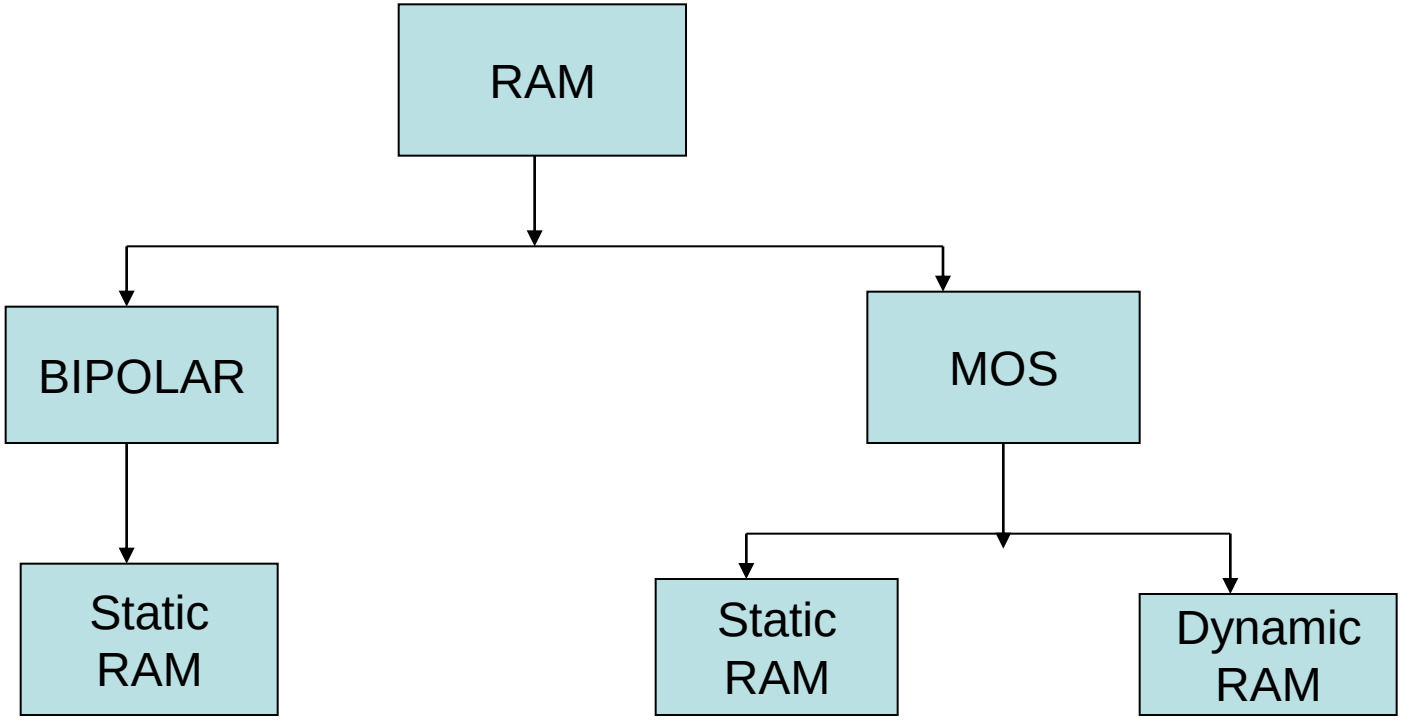


Capacity : 2^{m+1}

$\overline{OE}$: Output Enable
connect to RD of uP

$\overline{CE}(\overline{CS})$: Chip Enable
to Address decoder

RAM حافظه نیمه هادی



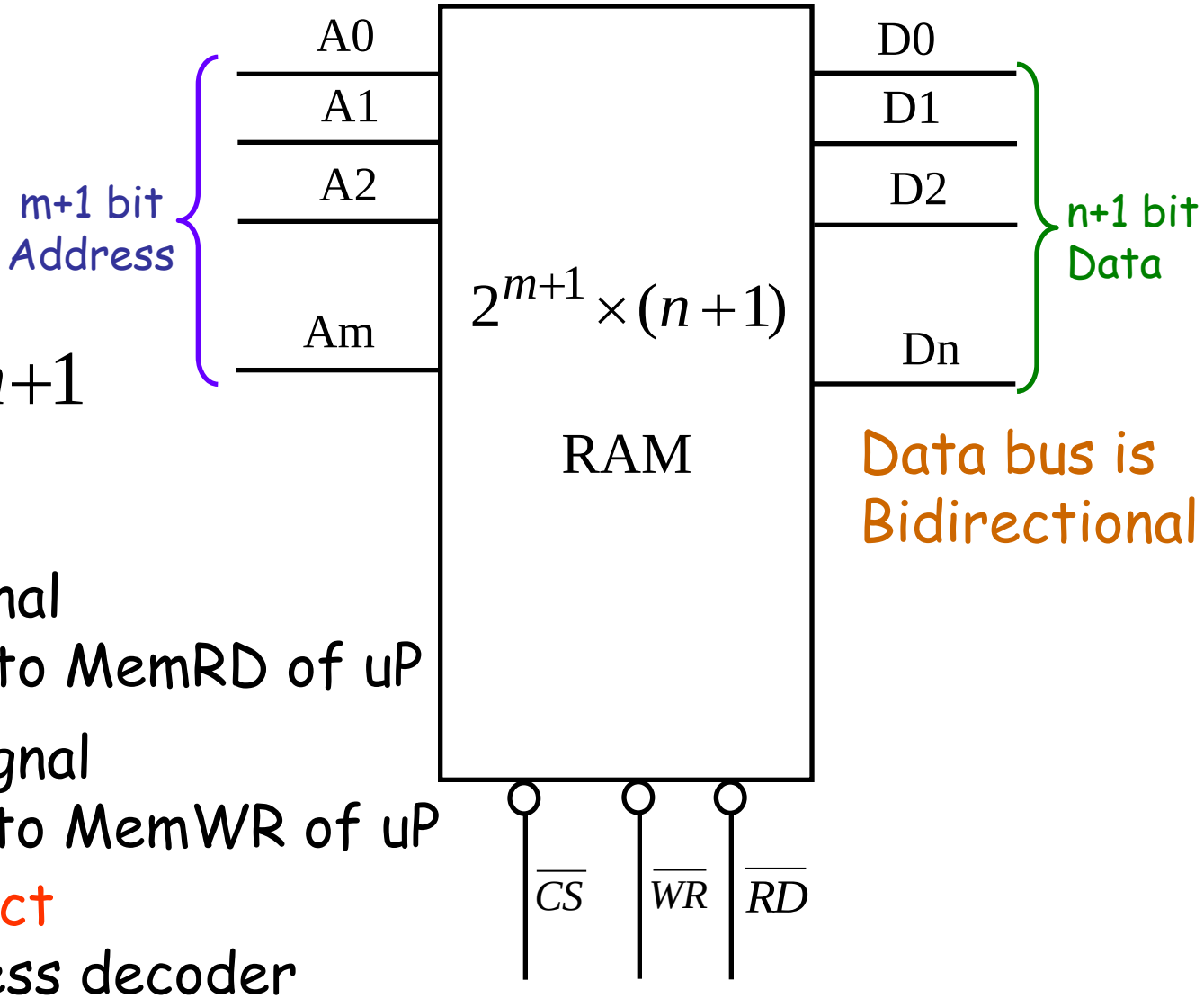
RAM (Random Access Memory)

- The uP can read the data from RAM quickly,
- The uP can write new data quickly to RAM
- RAM forgets its data if power is turned off
- Two type of is available :

Dynamic RAM (DRAM): cap base, slow , low cost – high capacity/volume , applied for main memory(pc) need refresh.

Static RAM(SRAM): ff base, fast, expensive, low – cap/vol, applied for cache , no refresh

RAM(Static)



Capacity : 2^{m+1}

Data bus is Bidirectional

$\overline{RD}$: Read signal
connect to MemRD of uP

$\overline{WR}$: Write signal
connect to MemWR of uP

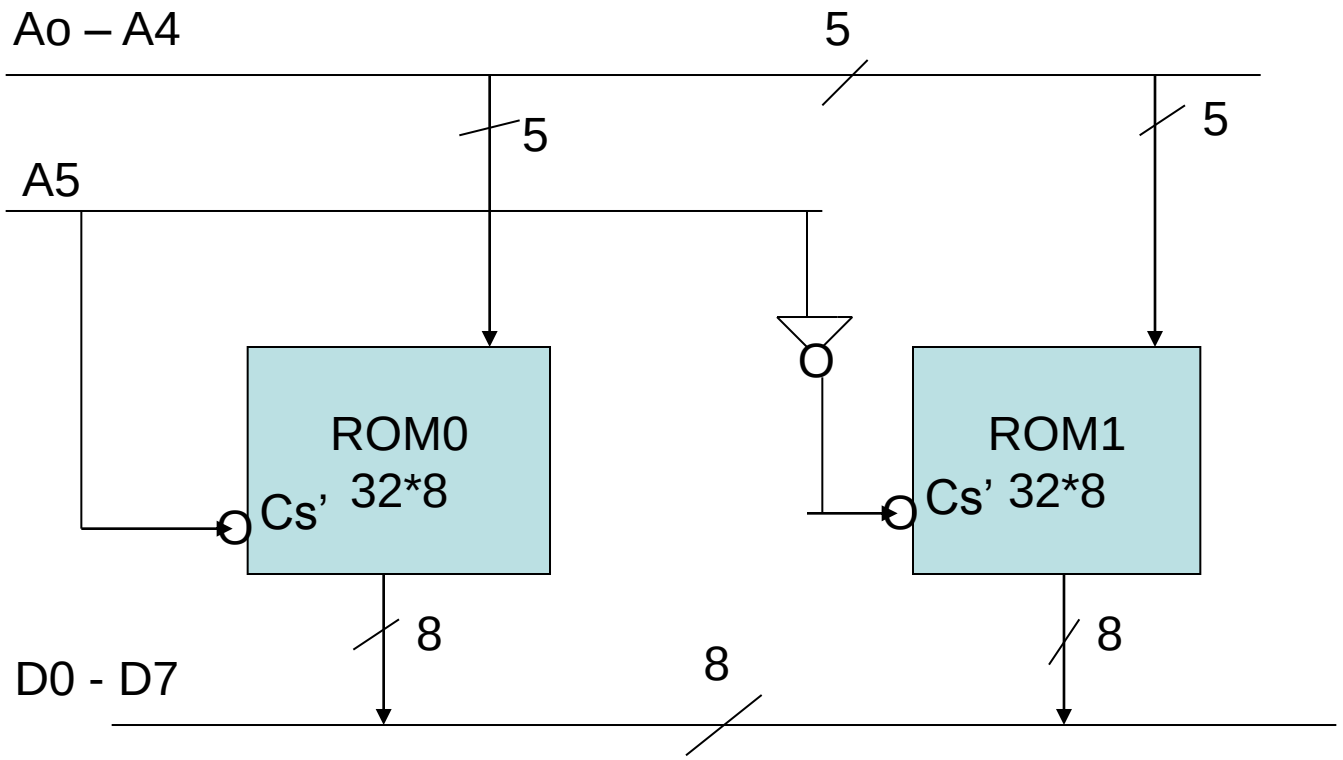
$\overline{CS}$: Chip Select
to Address decoder

Memory expansion

- بسیار مطلوب می بود اگر تمام ظرفیت های مورد نیاز حافظه بصورت تراشه های مجزا موجود می بود.ولی چنین نیست:
- مثال:

اگر یک ROM 8*64 نیاز باشد، ولی فقط تراشه های ROM 8*32 موجود باشد با استفاده از دو تراشه 8*32 میتوان یک ROM 8*64 ساخت.

Memory expansion ادامه (افزایش تعداد کلمات حافظه)



Memory expansion ادامه

- اگر A5=0 باشد ROM0 انتخاب میشود .
- اگر A5=1 باشد ROM1 انتخاب میشود .

- محدوده آدرس ROM0 00h تا 1Fh (۳۲ بایت)
- محدوده آدرس ROM1 20h تا 3Fh (۳۲ بایت)
- محدوده کل حافظه ۶۴ بایتی 00h تا 3Fh میباشد.

- تمرین : یک ROM 8 * 128 را با استفاده از ROMهای 32*8 بسازید.

Memory expansion ادامه: (افزایش تعداد بیتهای هر محل حافظه)

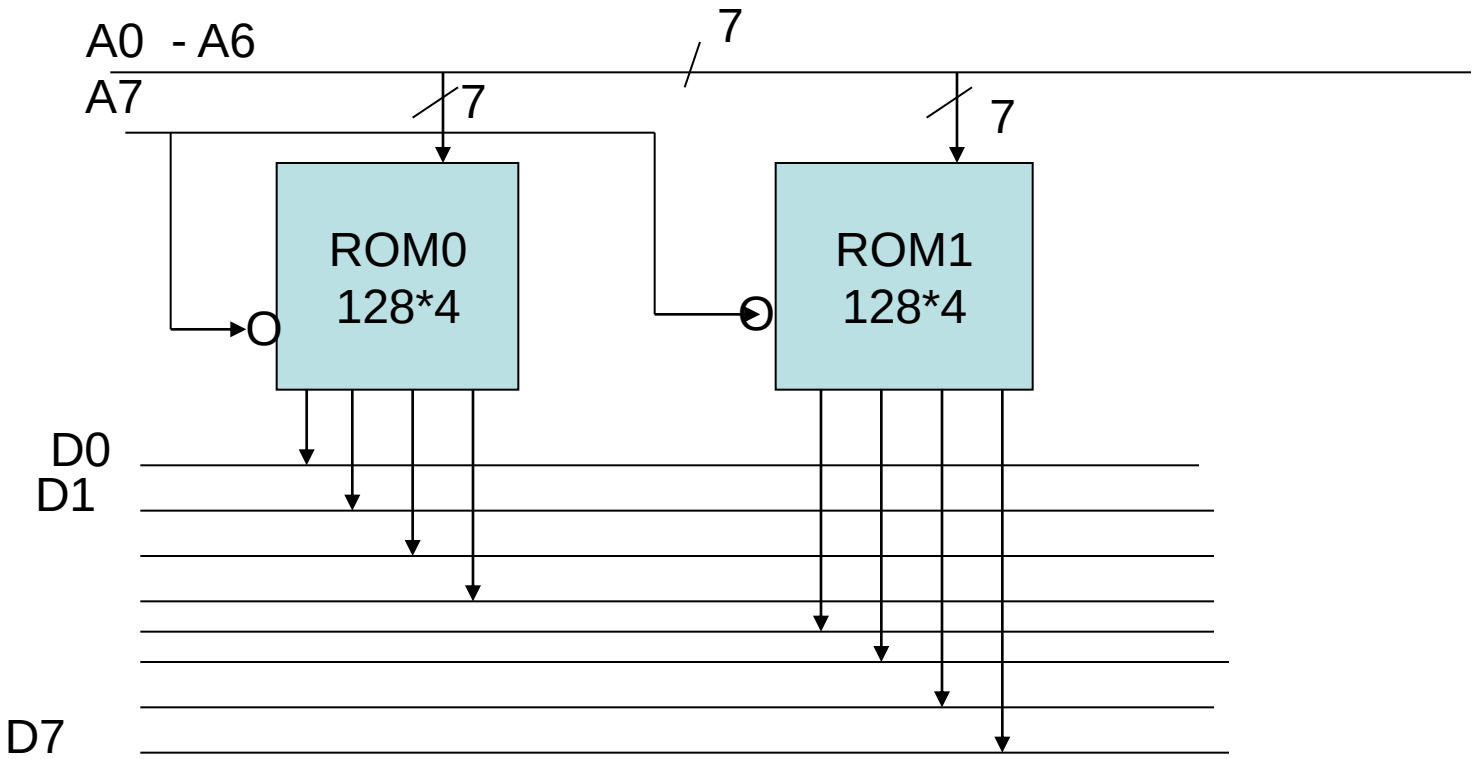
- گاهی اوقات حافظه هایی با طول کلمه ۸ بیتی ممکن است موجود نباشد..

  مثال:

  یک ROM 128*8 نیاز بوده ولی فقط تراشه های 128*4 موجود می باشند..

Memory expansion ادامه :

- اگر A7=0 باشد بطور همزمان محتوی آدرس شده هر یک از کلمات حافظه برروی گذرگاه داده قرار میگیرند.
- محدوده آدرس کل حافظه : 00h تا 7Fh



Memory address decoding

- یک ریزپردازنده ۸ بیتی معمولا دارای ۱۶ خط آدرس میباشد و در نتیجه می تواند به ۶۵۵۳۶ (۶۴ کیلو) محل از حافظه دسترسی داشته باشد.

- معمولا در یک عمل همه خطوط آدرس ممکن است لازم نباشد .

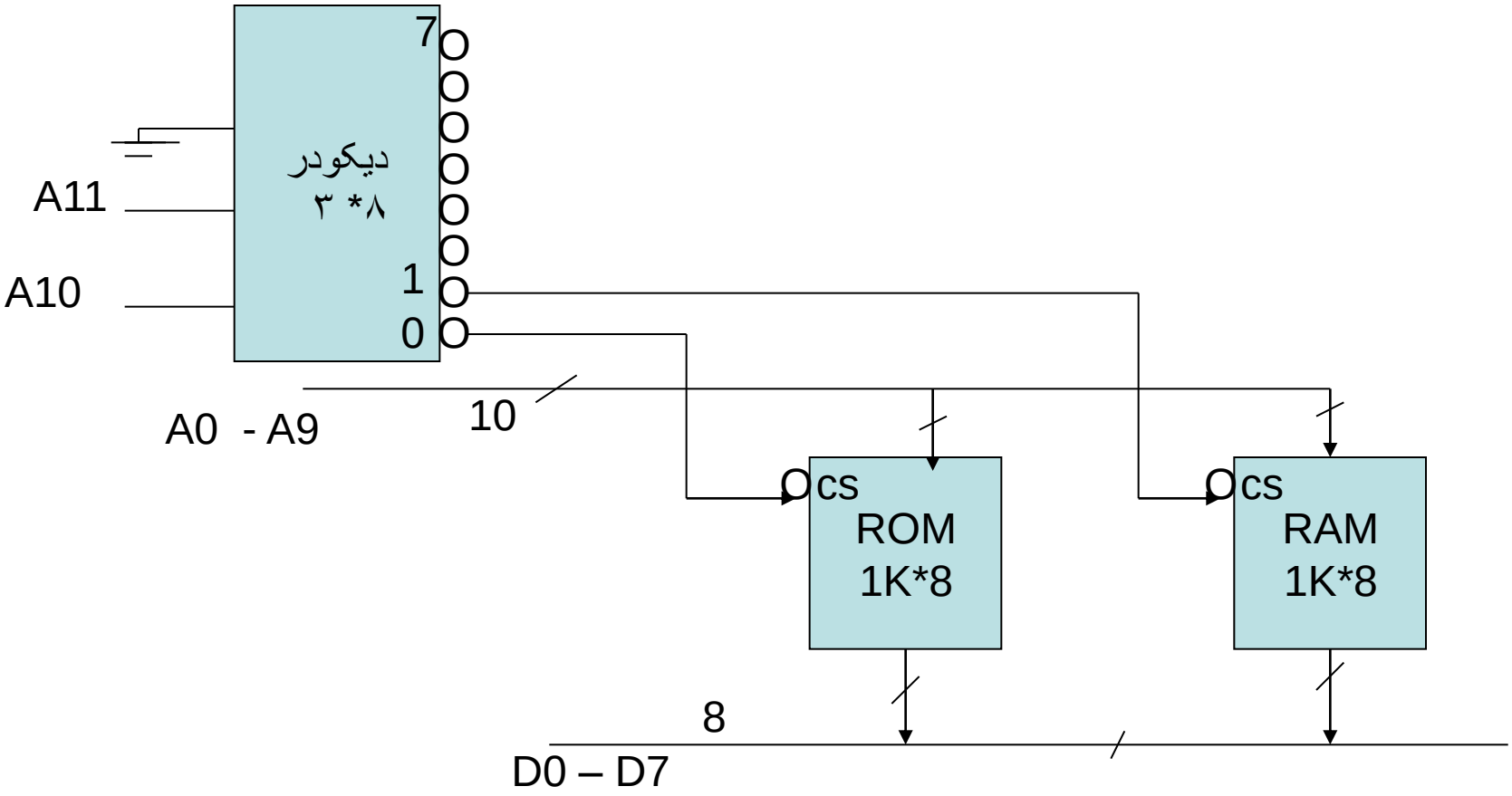
- حافظه مورد نیاز هم به صورت چندین تراشه مجزا بوده که باید از طریق CS هریک از تراشه ها به آنها دسترسی پیدا نمود.

- معمولا برای فعال نمودن csها از خطوط آدرس استفاده میشود.

- معمولا خطوط ارزش پایین گذرگاه آدرس را برای آدرس محل های حافظه و خطوط ارزش بالا گذرگاه آدرس را برای انتخاب تراشه ها استفاده میکنند.

Memory address decoding ادامه:

یک شیوه آدرس دهی نمودن یک حافظه ROM و RAM یک کیلو بایتی .



Memory address decoding ادامه:

- اگر A11=0 , A10=0 باشد حافظه ROM انتخاب میشود.
- اگر A11=0 , A10= 1 باشد حافظه RAM انتخاب میشود.

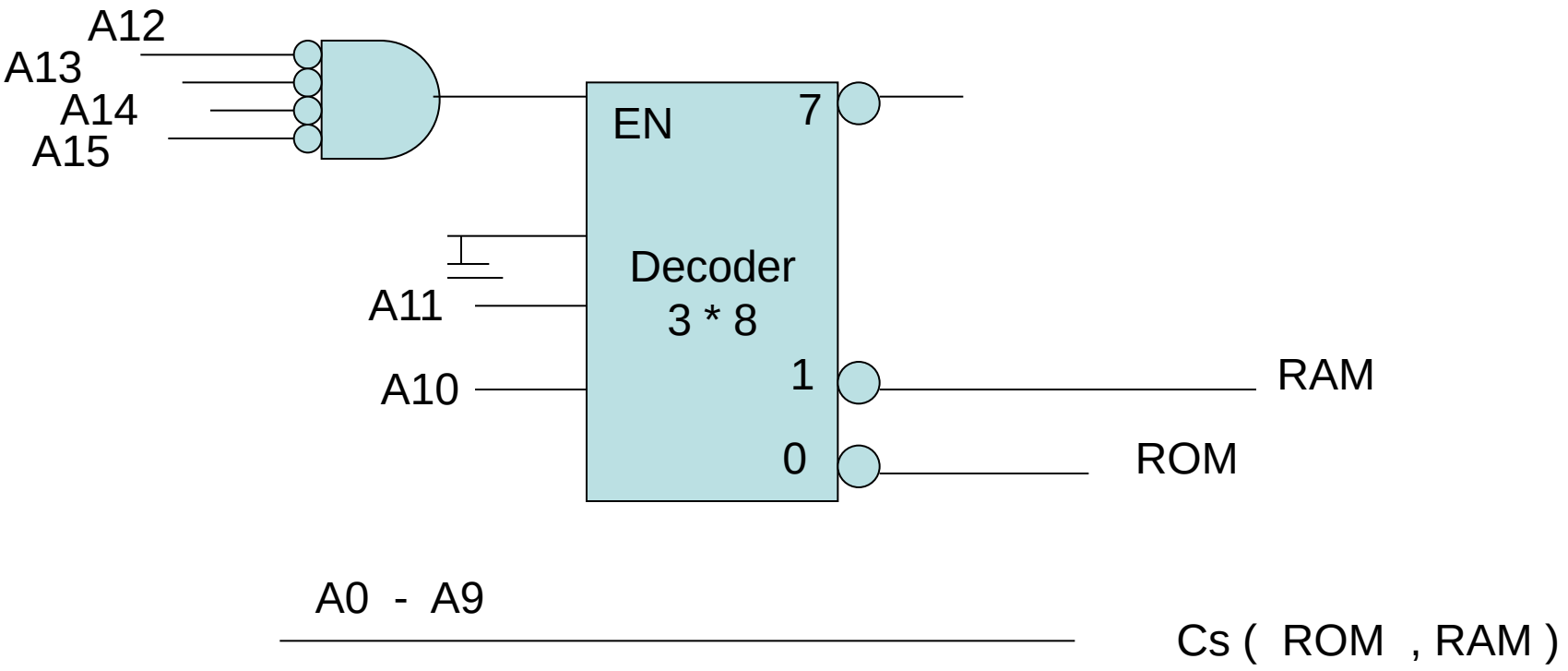
- محدوده آدرس ROM : 0000h تا 03FFh
- محدوده آدرس RAM : 0400h تا 07FFh

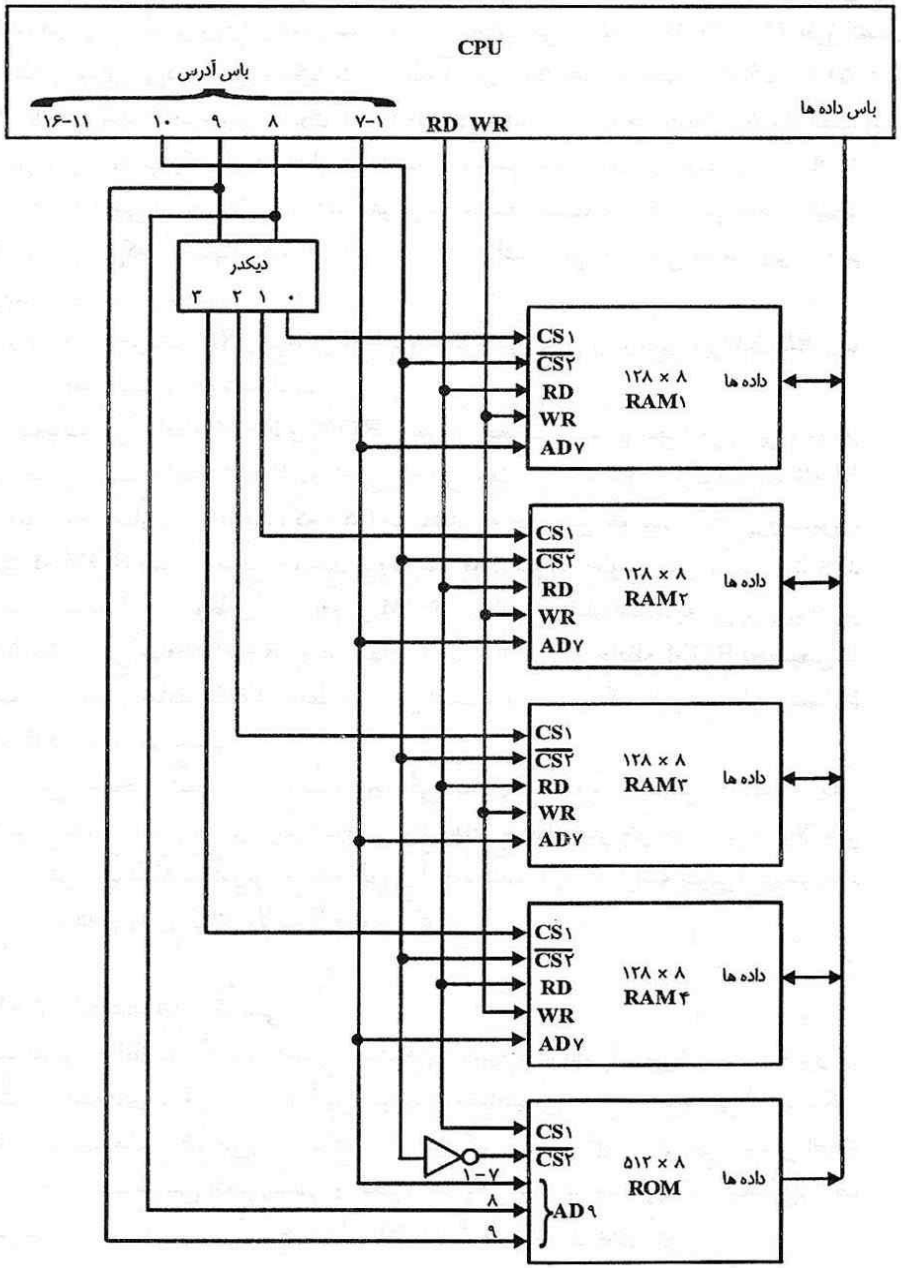
- از آنجاییکه چهار خط آدرس A12 تا A15 بجایی وصل نشده اند و میتوانند ۱۶ حالت مختلف بخود بگیرند، در نتیجه هر محل از حافظه ROM یا RAM می تواند توسط ۱۶ آدرس مختلف مورد دستیابی قرار گیرد.

  در این صورت برای هر محل حافظه یک آدرس ثابت وجود ندارد و به چنین روشی آدرس دهی non absolute addressing (آدرس دهی غیر مطلق)گفته میشود.

Memory address decoding ادامه:

آدرس دهی مطلق

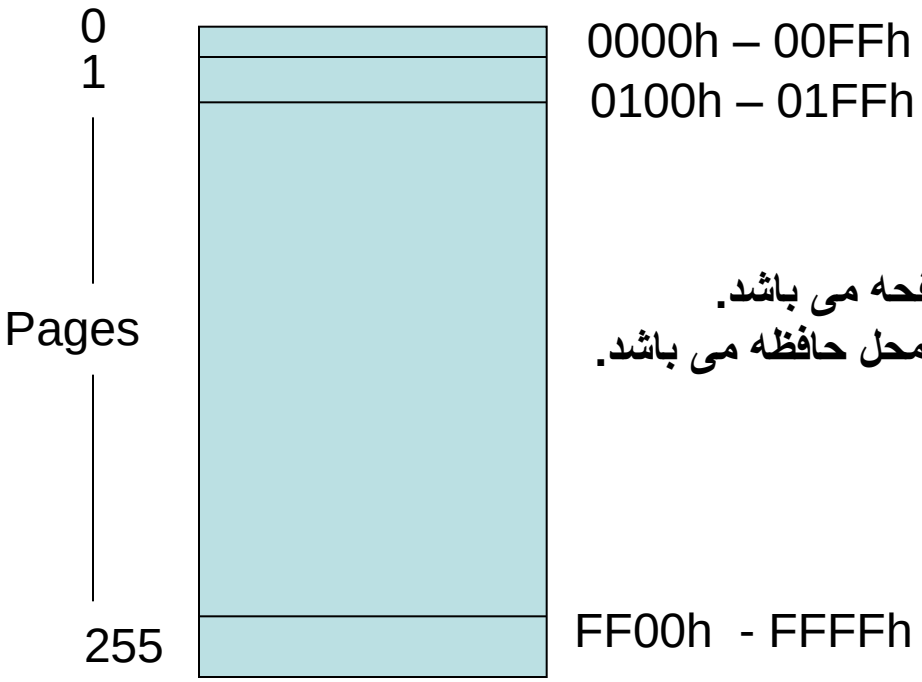




شکل (۱۲-۴) اتصال حافظه به CPU

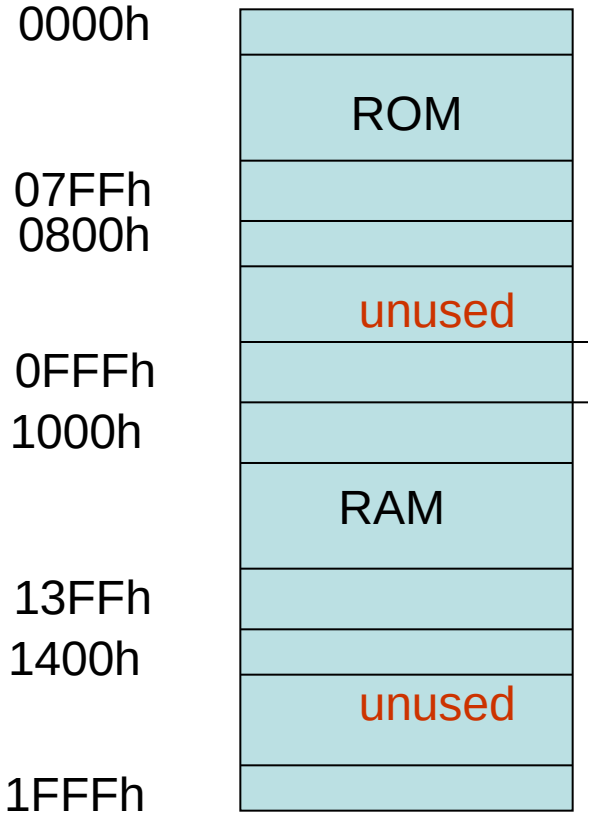
Memory pages
صفحات حافظه

- **در بعضی از ریزپردازنده ها که دارای ۱۶ خط آدرس میباشند، فضای آدرس دهی (۶۵۵۳۶محل) به ۲۵۶ بلوک (صفحه) که هر یک دارای ۲۵۶ محل (آدرس) می باشند تقسیم می شوند. (۶۵۵۳۶ = ۲۵۶ * ۲۵۶)**



دو رقم با ارزش بالا مشخص کننده صفحه می باشد.
دو رقم با ارزش پایین مشخص کننده محل حافظه می باشد.

Memory map
نقشه حافظه



- در یک سیستم مبتنی بر ریزپردازنده یک سازمان حافظه ۸ کیلو بایتی وجود دارد. در این سیستم یک حافظه ROM ۲ کیلو بایتی که از آدرس 0000h شروع میشود و یک حافظه RAM ۱کیلو بایتی که از آدرس 1000h شروع میشود وجود دارد. نقشه حافظه سیستم مورد نظر را ترسیم کنید.

Over flow

- در جمع دو عدد یک کلمه ایی ممکن است حاصل جمع قابل نمایش در یک کلمه نباشد . در این صوت ، حاصل را نمی توان در انباره که طول کلمه آن بیش از یک کلمه نیست نگهداری نمود که در این حالت می گوین سرریز (over flow) رخ داده است.

- بر حسب اینکه از چه روشی برای نمایش اعداد منفی استفاده میکنیم نتیجه را باید ترمیم کنیم.

- در ریزپردازنده ها پس از هر عمل جمع یا تفریق وقوع یا عدم وقوع سرریز در یک بیت ، که با حرف V نشان داده میشود ،ضبط می گردد.

قوانین حاکم بر جمع اعداد چند بیتی مکمل دو مورد بحث قرار میگیرند.
در جمع دو عدد هشت بیتی ۵ حالت متفاوت وجود داد.

۱- هر دو عدد مثبت و نتیجه حاصل قابل نمایش در یک کلمه میباشد.

۱۱۹ = ۷ + ۱۱۲ 01110000 +
00000111

01110111 C=0 و V=0

۲- هر دو عدد مثبت هستند ولی نتیجه حاصل قابل نمایش در انباره نیست.

۱۶۰ = ۴۸ + ۱۱۲ 01110000 +
00110000

10100000 , V = 1, C=0

نتیجه حاصل باید مثبت ۱۶۰ باشد در صورتی که ۹۶- است.
برای تصحیح نتیجه حاصل باید بیت نقلی را به عنوان بیت علامت در کنار انباره قرار داد تا نتیجه درست حاصل شود.

Over flow ادامه :

۳- هر دو عدد منفی ولی انباره سرریز نشده است:

-107 + -10 = -117

10010101 +
11110110

10001011 C=1 و V=0

۴- هر دو عدد منفی و نتیجه حاصل قابل نمایش در انباره نیست.

-111 + -39 = -150

10010001 +
11011001

01101010 C= 1 و V=1

Over flow ادامه :

۵- یک عدد مثبت ودیگری منفی است. در این حالت هرگز سرریز رخ نمی دهد.

A) عدد مثبت بزرگتر از قدر مطلق عدد منفی است.

-112 + 120 = 8

10010000 +

01111000

00001000

C= 1 و V= 0

B) عدد مثبت کوچکتر از قدر مطلق عدد منفی است.

-112 + 96 = -16

10010000 +

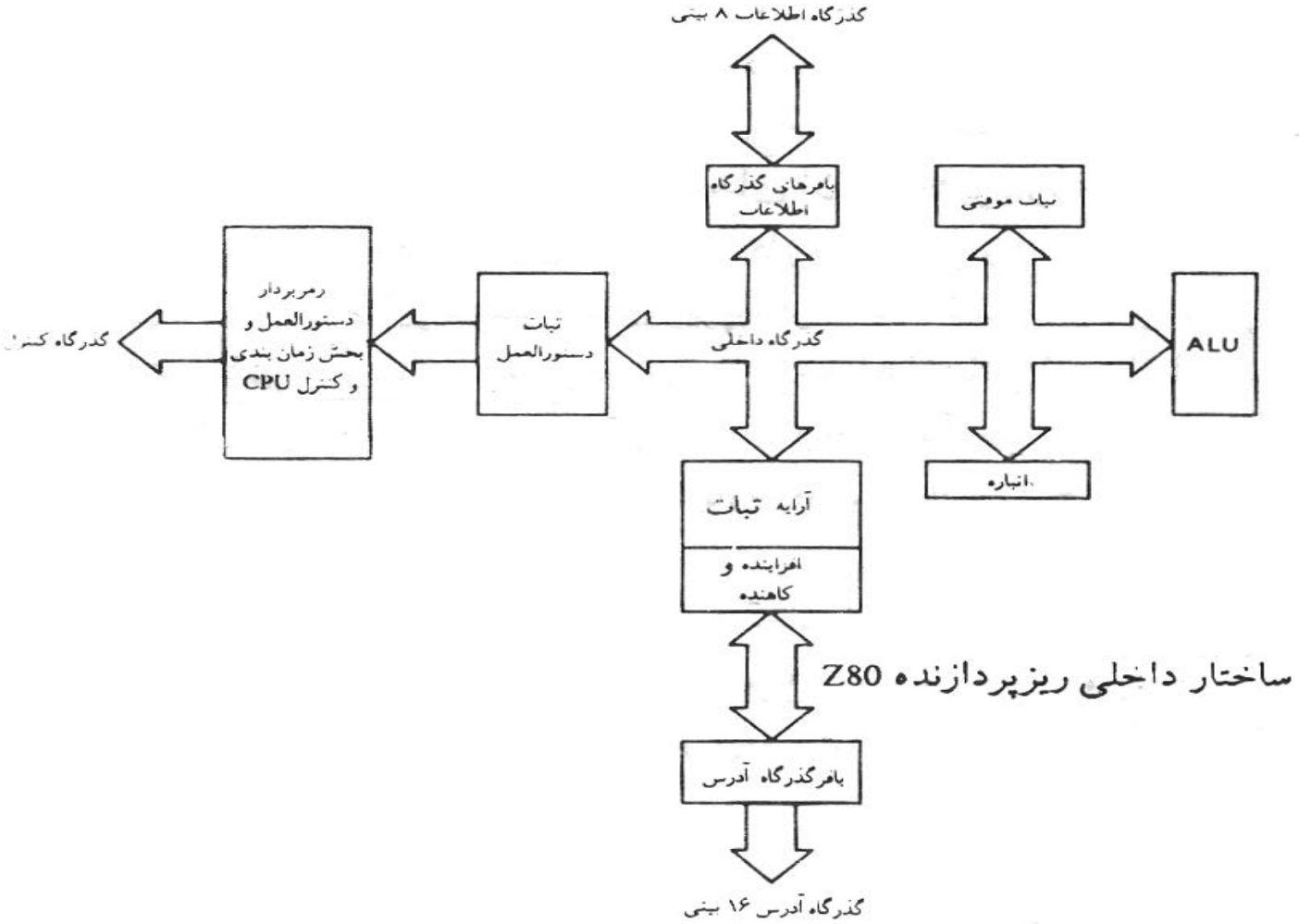
01100000

11110000

C= 0 و V= 0

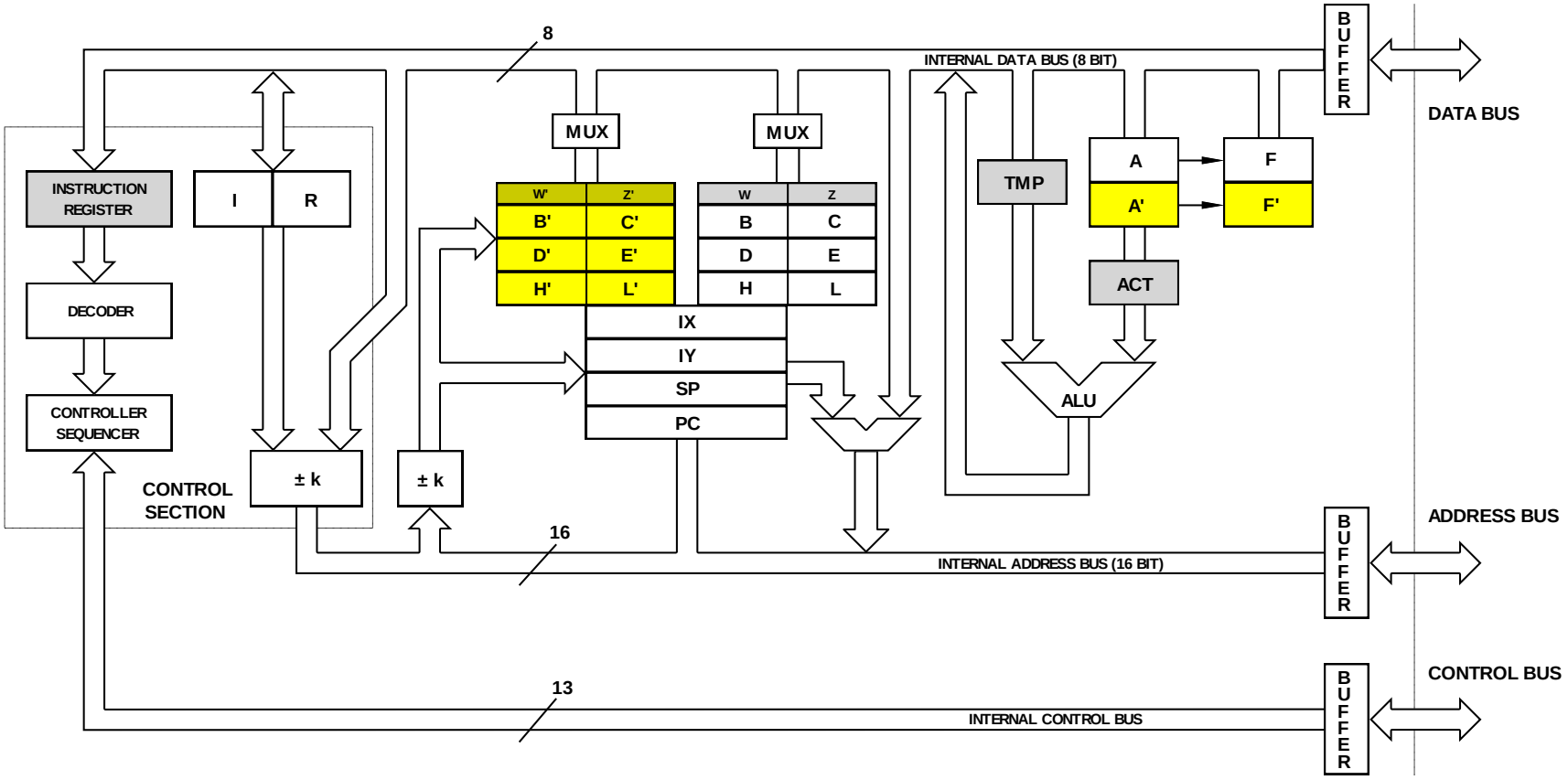
Z80 Microprocessor

ريزپردازنده Z80 يك ريزپردازنده ۸ بيتي همه منظوره است كه براي بسياري از كاربردهاي كنترلي مناسب است ريزپردازنده Z80 داراي يك آرايه ثبات، يك بخش زمان‌بندي و كنترل، يك واحد حسابي و منطقي (ALU) و اتصالات گذرگاهي به محيط خارج است.

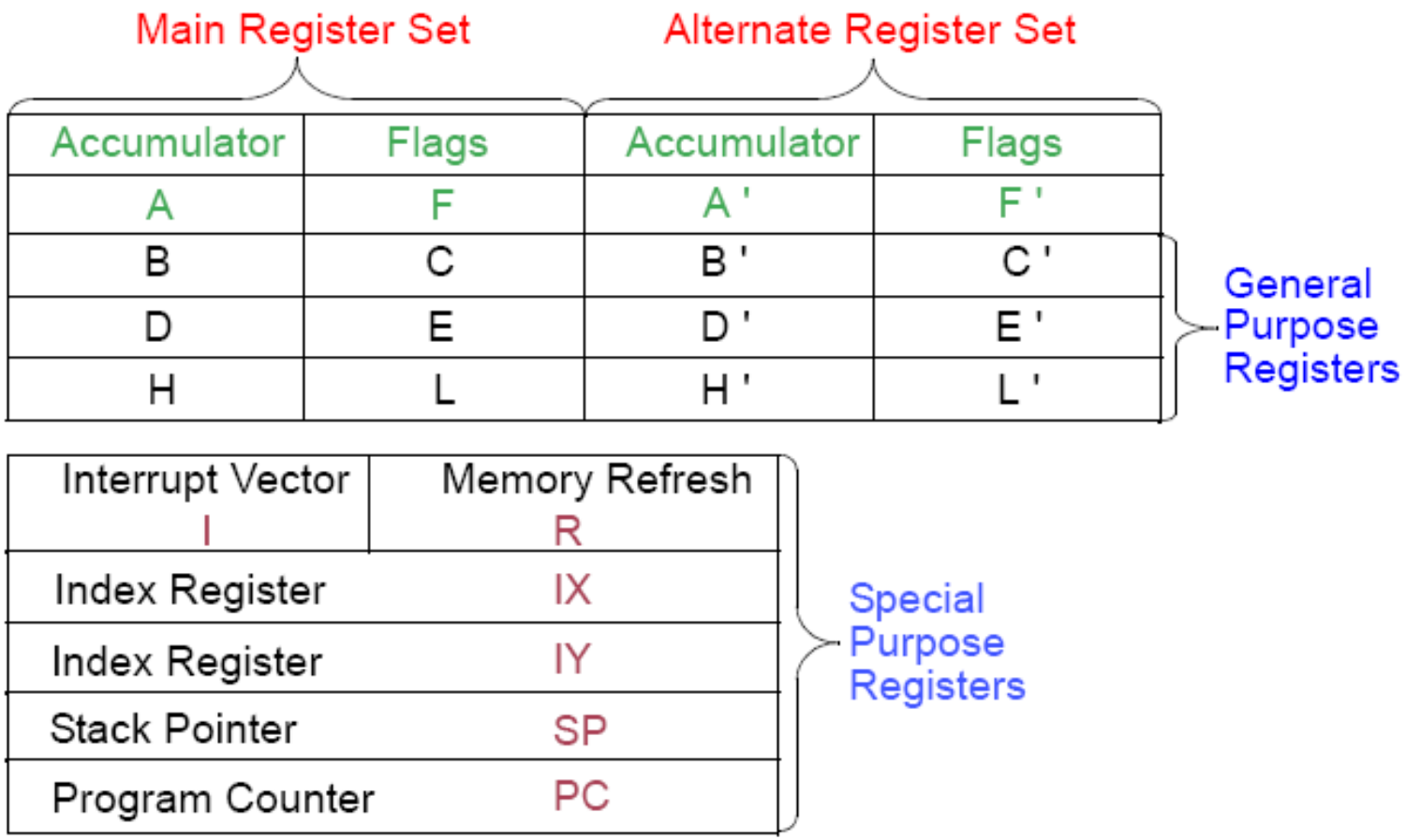


ساختار داخلی ريزپردازنده Z80

ساختمان داخلی Z80

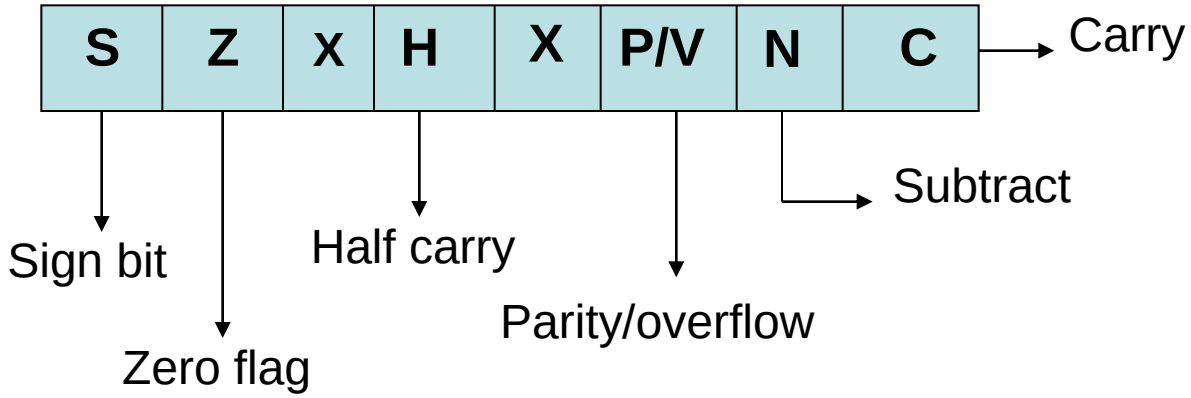


مدل برنامه نویسی Z80



ثبات پرچم Flag Register

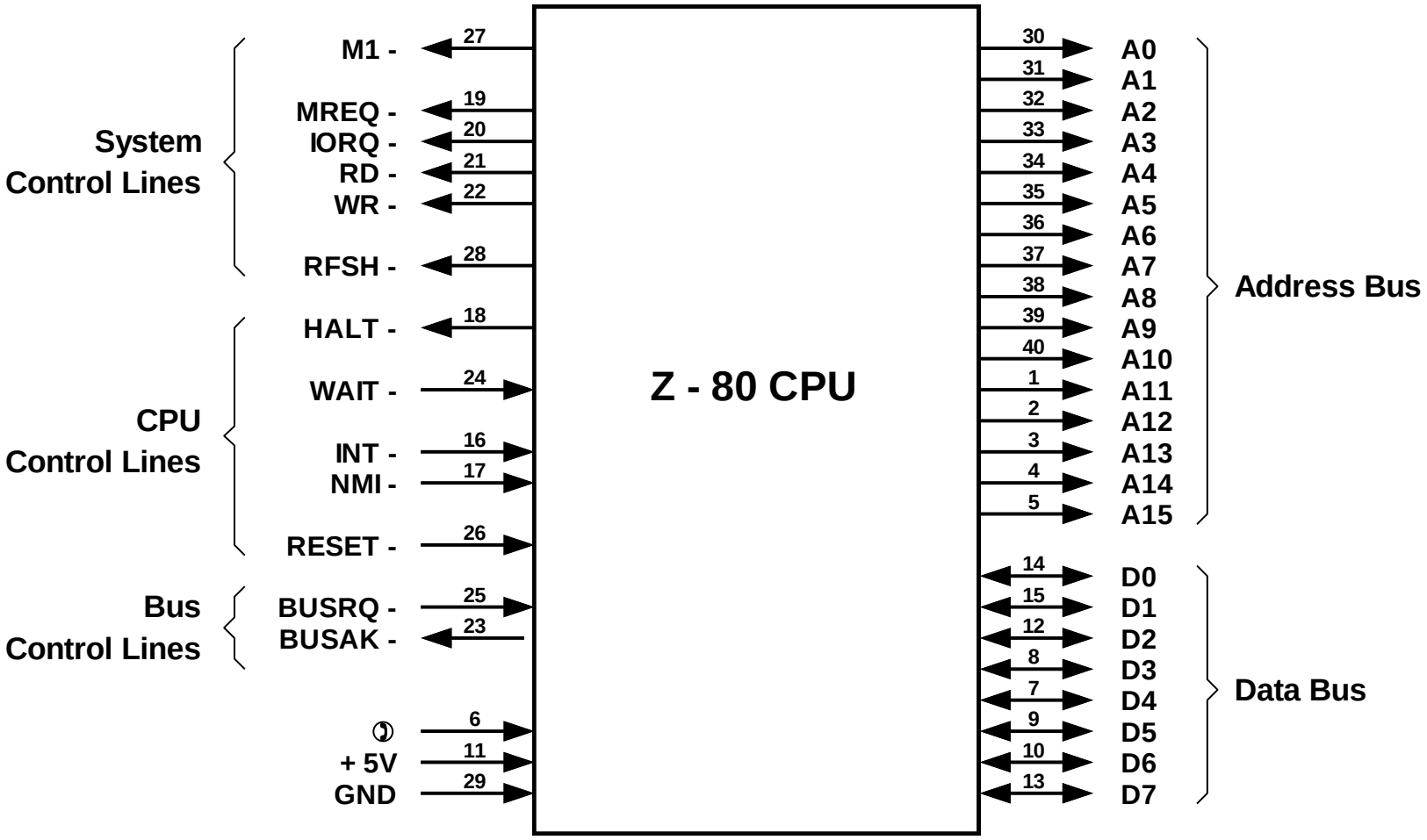
این ثبات برای حفظ و نگهداری اطلاعات مربوط به نتایج حاصل از عملیات مختلف ریزپردازنده می باشد.



کاربرد عمده بیتهای پرچم در دستورات پرش شرطی می باشد.

کاربرد بیتهای H و N در محاسبا ت BCD میباشد.

Z80 – pin out


Z - 80 CPU
System Control Lines
CPU Control Lines
Bus Control Lines
M1 - 27
MREQ - 19
IORQ - 20
RD - 21
WR - 22
RFSH - 28
HALT - 18
WAIT - 24
INT - 16
NMI - 17
RESET - 26
BUSRQ - 25
BUSAK - 23
6
+ 5V 11
GND 29
30 A0
31 A1
32 A2
33 A3
34 A4
35 A5
36 A6
37 A7
38 A8
39 A9
40 A10
1 A11
2 A12
3 A13
4 A14
5 A15
Address Bus
14 D0
15 D1
12 D2
8 D3
7 D4
9 D5
10 D6
13 D7
Data Bus

Pin Function Summary

Address A0 –A15: output, active high• •
16 bit address for memory (64KB) •
•8 bit address for I/O (256) •
•Refresh address during M1 (A0 –A6 : 7 bit) •

Data D0 –D7: input/output, active high •
•8 bit exchange with memory & I/O. •
•Interrupt vector •

RD / WR: output, active low •
•Indicates that the CPU wants to read / write data from / to memory or I/O. •

M1: output, active low •
•together with MREQ, indicates that the current machine cycle is the opcode fetch cycle •
•together with IORQ, indicates an interrupt acknowledge cycle •

MREQ : output, active low **LD** •

•indicates that the address bus holds a valid address for a **memory** • read/write operation.

•Also active low during memory refresh (*RFSH*). •

IORQ : output, active low **IN, OUT** •

•indicates that the lower half of the address bus *(A0 –A7)*holds a • valid I/O address for an **I/O** read / write operation.

•also generated concurrently with *M1*during an interrupt • acknowledge cycle.

RFSH : output, active low •together with *MREQ*duringM1, • indicates that the lower seven bits of the system's address bus *(A0 – A7)*can be used as a refresh address to the system's dynamic memories

HALT : output, active low •

indicates that the CPU has executed a HALT instruction.. •

During HALT, the CPU executes NOPs to maintain memory refresh. •

HALT exit : INT, NMI, RESET •

WAIT : input, active low • •

This signal indicates to the CPU that the addressed memory or I/O • device is not ready for data transfer.

Always sampled at T2. •

NMI : input, active low •

higher priority than INT. •

always recognized at the end of the current instruction, independent • of the status of the interrupt enable flip-flop (IFF).

automatically forces the CPU to restart at location 0066H. •

INT : input, active low •

•Acknowledged a INT request at the end of the current instruction if the internal software-controlled interrupt enable flip-flop (IFF) is enabled.

•EI / DI Instruction •

•Mode 0, 1, 2 •

BUSREQ : input, active low •

•higher priority than NMI and is always recognized at the end of the current machine cycle.

•BUSREQ forces the CPU address bus, data bus, and control signals MREQ, IORQ, RD, and WR to go to a high-impedance state so that other devices can control these lines.

BUSACK : output, active low •indicates to the requesting device that the CPU address bus, data bus, and control signals MREQ, IORQ RD, and WR have entered their high-impedance states.

•The external circuitry can now control these lines. •

RESET : input, active low •

•initializes the CPU as follows •

–Reset the interrupt enable flip-flop (IFF = 0) •

–Clear the PC (PC = 0000h) •

–Clear registers I and R •

•sets the interrupt status to Mode 0. •

•the address and data bus go to a high-impedance state •

•all control output signals go to the inactive state. •

•RESET must be active for a minimum of three full clock cycles before the reset operation is complete. •

CLK : input •

•Normally 5 –20 Mhz according to data sheet •

VCC / GND : input •

Pin Functions

- **M1’** indicates that the current machine cycle is the opcode fetch cycle of an instruction execution.
- **MREQ’** **m***emory Request (output, active Low, tristate).* MREQ indicates that the address bus holds a valid address for a memory read or memory write operation

IORQ’ *Input/Output Request (output, active Low,tristate).* IORQ indicates that the lower half of the address bus holds a valid I/O address for an I/O read or write operation.

Pin Functions

- **RD’** *Read (output, active Low, tristate).* RD indicates that the CPU wants to read data from memory or an I/O device. The addressed I/O device or memory should use this signal to gate data onto the CPU data bus.
- **WR’** *Write (output, active Low, tristate).* WR indicates that the CPU data bus holds valid data to be stored at the addressed memory or I/O location

RFSH’ *Refresh (output, active Low).* RFSH, together with MREQ indicates that the lower seven bits of the system’s address bus can be used as a refresh address to the system’s dynamic memories.

Pin functions

HALT' *HALT State (output, active Low).* HALT indicates that the CPU has executed a HALT instruction and is waiting for either a non-maskable or a maskable interrupt (with the mask enabled) before operation can resume. During HALT, the CPU executes NOPs to maintain memory refresh.]

WAIT' *WAIT (input, active Low).* WAIT communicates to the CPU that the addressed memory or I/O devices are not ready for a data transfer. The CPU continues to enter a WAIT state as long as this signal is active. Extended WAIT periods can prevent the CPU from properly refreshing dynamic memory.

RESET' *Reset (input, active Low).* RESET initializes the CPU as follows: it resets the interrupt enable flip-flop, clears the PC and registers I and R, During reset time, the address and data bus go to a high-impedance state, and all control output signals go to the inactive state.

Pin functions

- **INT’** *Interrupt Request (input, active Low).* Interrupt Request is generated by I/O devices. The CPU honors a request at the end of the current instruction if the internal software-controlled interrupt enable flip-flop (IFF) is enabled.

- **NMI’** *Non-Maskable Interrupt (input, negative edge-triggered).* NMI has a higher priority than INT. NMI is always recognized at the end of the current instruction, independent of the status of the interrupt enable flip-flop, and automatically forces the CPU to restart at location 0066H.

Pin functions

BUSREQ’

Bus Request (input, active Low). Bus Request has a higher priority than NMI and is always recognized at the end of the current machine cycle .

BUSREQ forces the CPU address bus, data bus, and control signals MREQ , IORQ, RD, and WR to go to a high-impedance state so that other devices can control these lines.

BUSACK’

Bus Acknowledge (output, active Low). Bus Acknowledge indicates to the requesting device that the CPU address bus, data bus, and control signals MREQ, IORQ RD, and WR have entered their high-impedance states. The external circuitry can now control these lines.

Pin functions

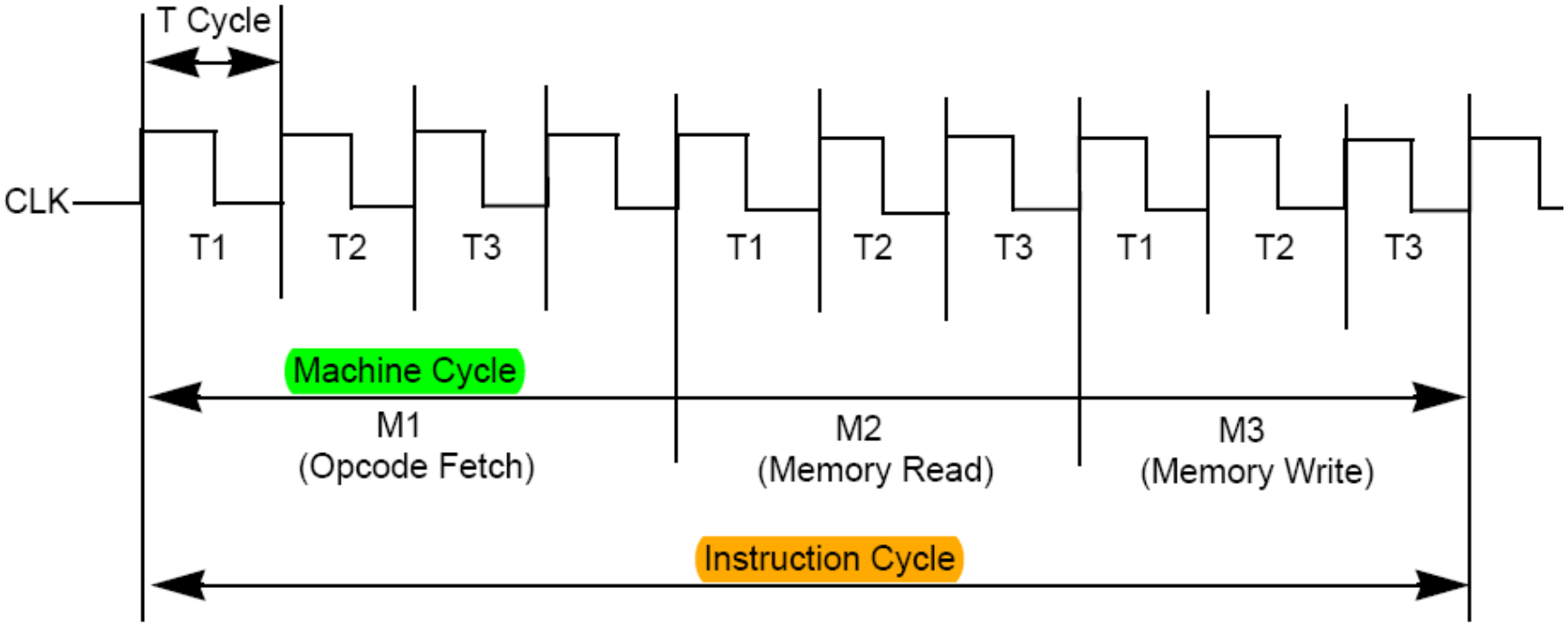
A15–A0 •

Address Bus (output, active High, tristate). A15-A0 form a 16-bit • address bus. The Address Bus provides the address for memory data bus exchanges (up to 64 Kbytes) and for I/O device exchanges

D7–D0

Data Bus (input/output, active High, tristate). D7–D0 constitute an 8-bit bidirectional data bus, used for data exchanges with memory and I/O.

CPU Timing



1 Instruction Cycle =1 ~ 6 Machine Cycle

- 1 Machine Cycle =3 ~ 6 T Cycle

Instruction Cycle

- 1. Op. Code Fetch *= M1*
- 2. Memory Read
- 3. Memory Write
- 4. I/O Read
- 5. I/O Write
- 6. Bus Request / Acknowledge
- 7. INT Request / Acknowledge
- 8. NMI Request / Acknowledge
- 9. HALT Exit

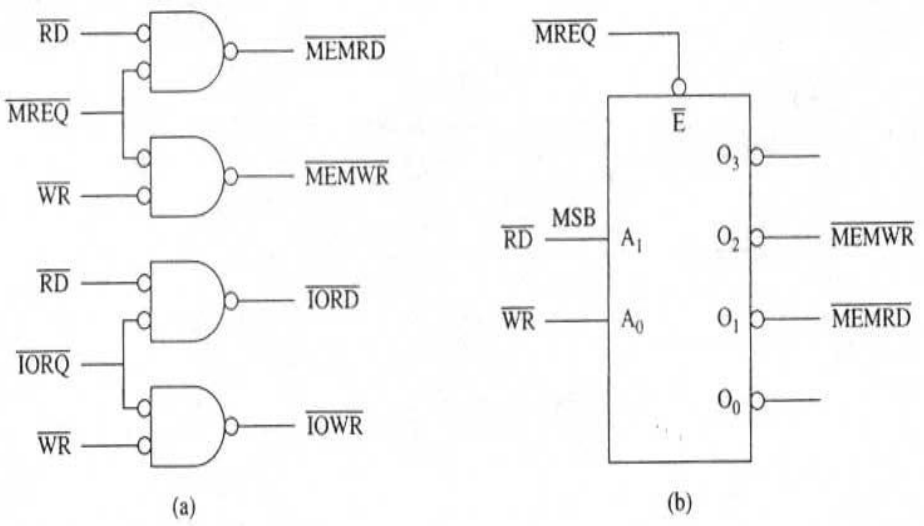
Machine Cycles and Control Signals

TABLE
The Z80 Machine Cycles and Control Signals

Machine Cycle	$(\overline{M_1})$	$\overline{MREQ}$	$\overline{IORQ}$	$\overline{RD}$	$\overline{WR}$
Opcode Fetch (M_1)	0	0	1	0	1
Memory Read	1	0	1	0	1
Memory Write	1	0	1	1	0
I/O Read	1	1	0	0	1
I/O Write	1	1	0	1	0
Interrupt Acknowledge	0	1	0	1	1
Nonmaskable Interrupt	0	0	1	0	1
Bus Acknowledge ($\overline{BUSAK} = 0$)	1	Z	Z	Z	Z

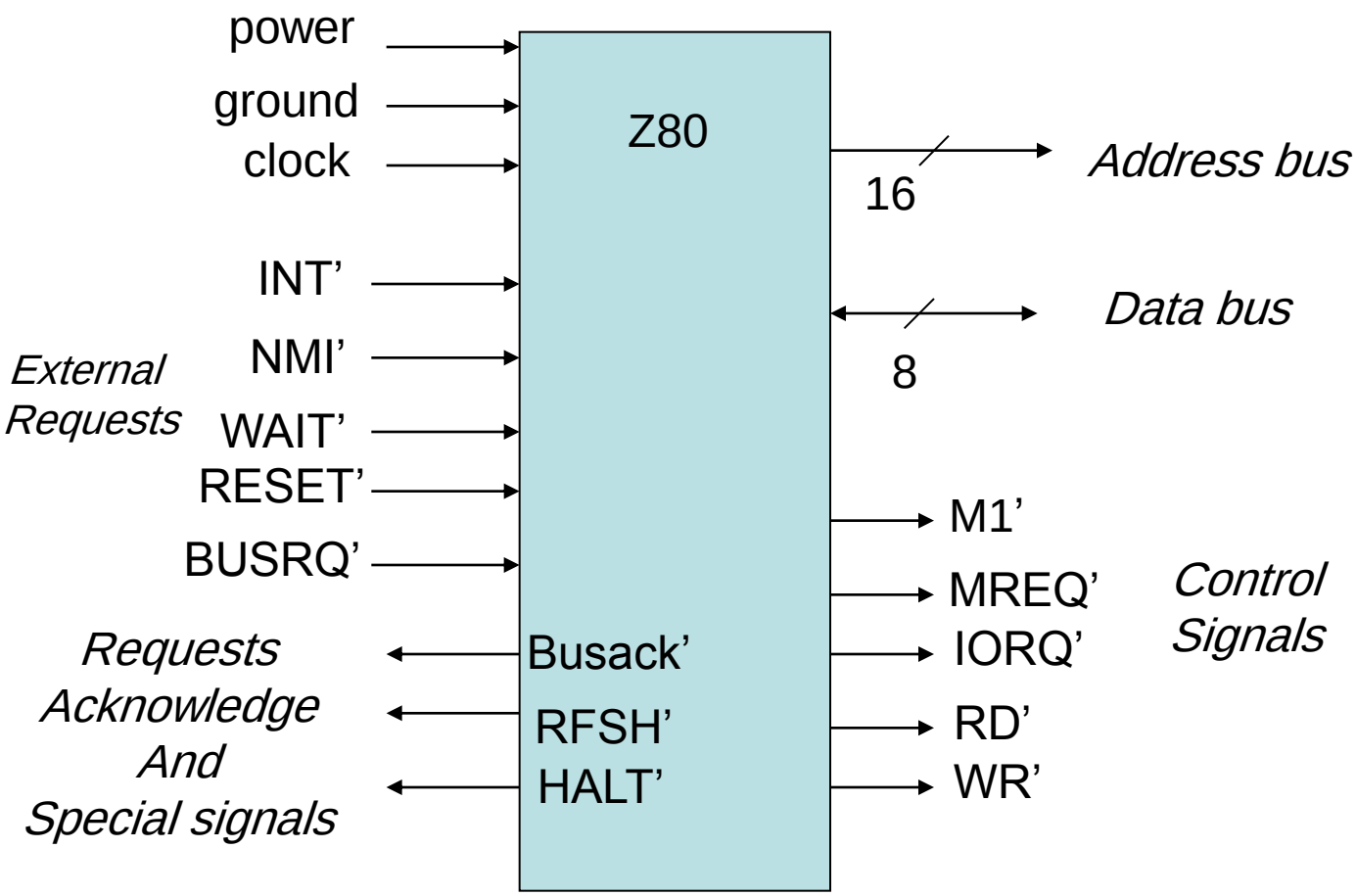
NOTE: Logic 0 = Active, Logic 1 = Inactive, Z = High Impedance

Generating Control Signals

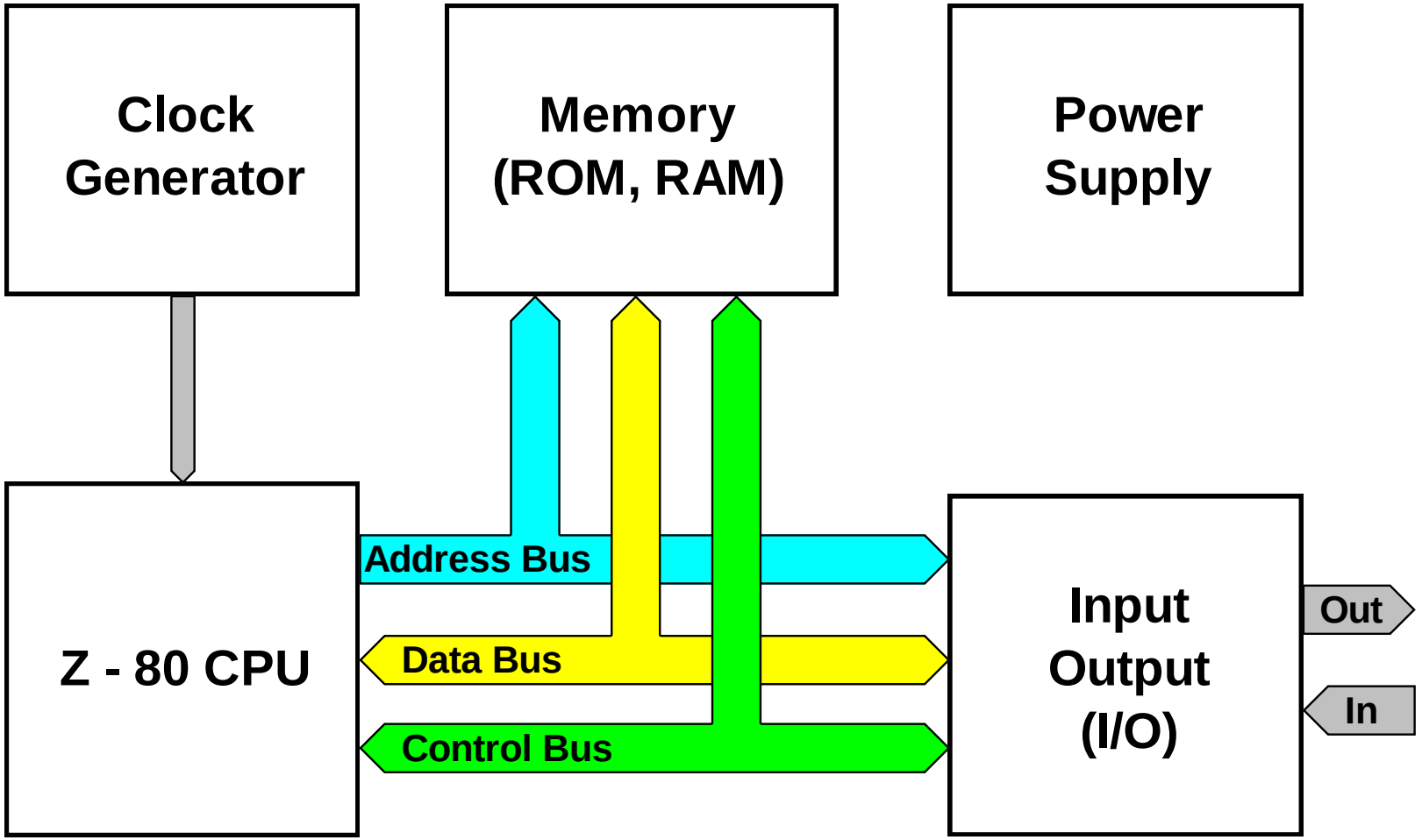


کلیه سیگنالهای z80 در ۶ گروه تقسیم بندی میشوند:

1- address bus •
2- data bus •
3- control bus •
4- external requests •
5- request acknowledge and special signals •
6- power and frequency signals •


power
ground
clock
Z80
16
Address bus
INT'
Data bus
External
Requests
NMI'
8
WAIT'
RESET'
M1'
BUSRQ'
MREQ'
Control
Signals
Requests
Acknowledge
And
Special signals
Busack'
IORQ'
RFSH'
RD'
HALT'
WR'

Minimal Configuration of a Z80 Microcomputer



Z80 Memory connection

CPU 16 bit address bus ➔ 64 k memory(max)

CPU 8 bit data bus ➔ 8 bit data width

Generally should be connected

Data to data

Address to address

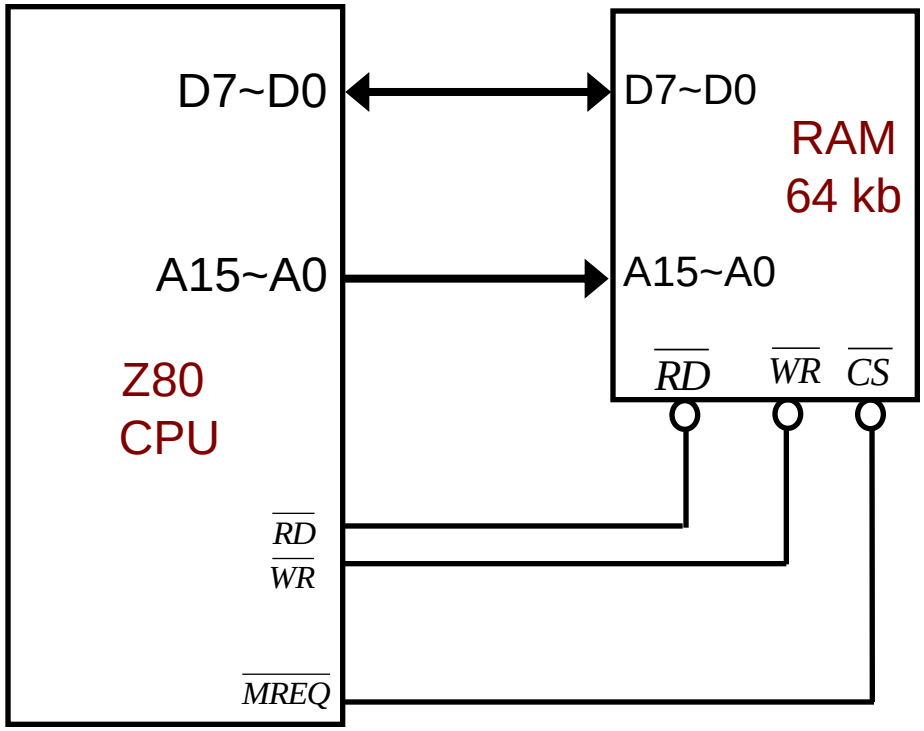
Wr to wr

Rd to rd

Mreq to cs

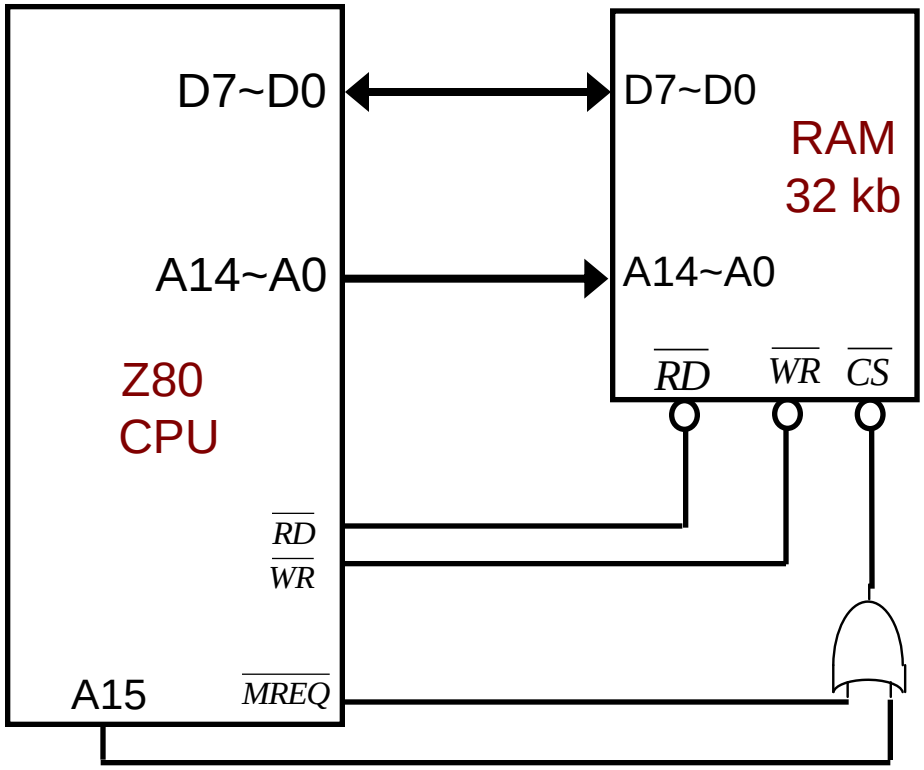
Memory connection (cont.)

If only one RAM chip Full size (64 kb capacity)❑



Memory connection (cont.)

If RAM capacity was 32 kb
A15 composed with MREQ
RAM area is from 0000h to 7FFFh



Memory connection (cont.)

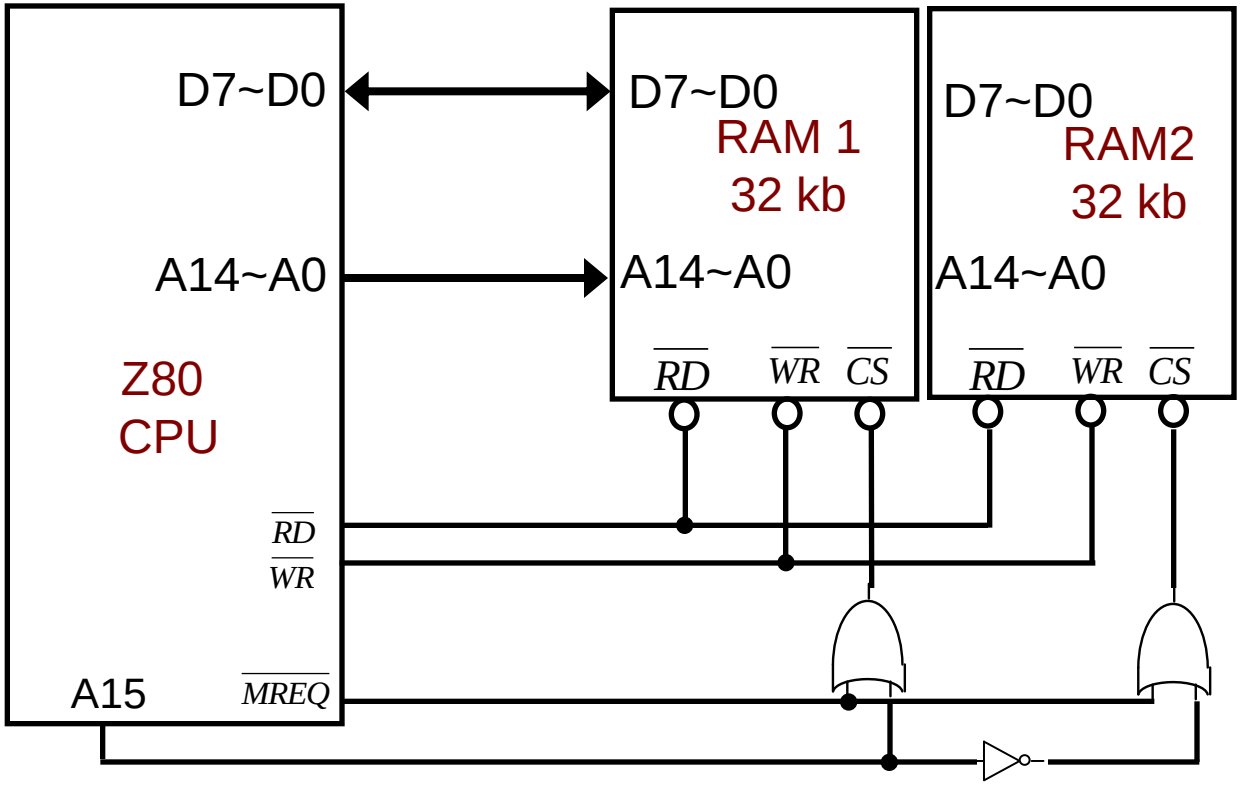
There is two 32 kb RAM

Problem: Bus Conflict. The two memory chips will provide data at the same time when microprocessor performs a memory read.

Solution: Use address line A15 as an “arbiter”. If A15 outputs a logic “1” the upper memory is enabled (and the lower memory is disabled) and vice-versa.

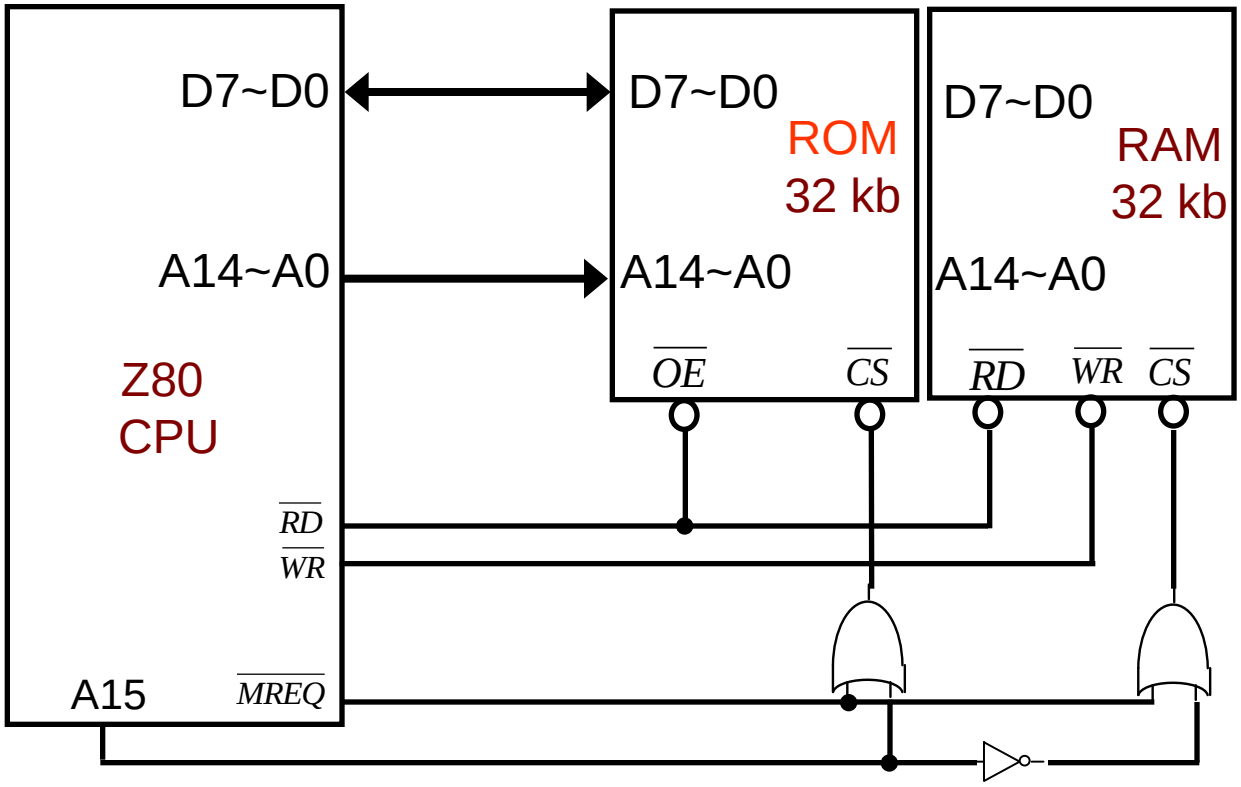
Memory connection (cont.)

There is two 32 kb RAM
A15 applied to select one RAM chip
Two RAM area is from 0000h to 7FFFh (RAM1)
and 8000h to FFFFh (RAM2)



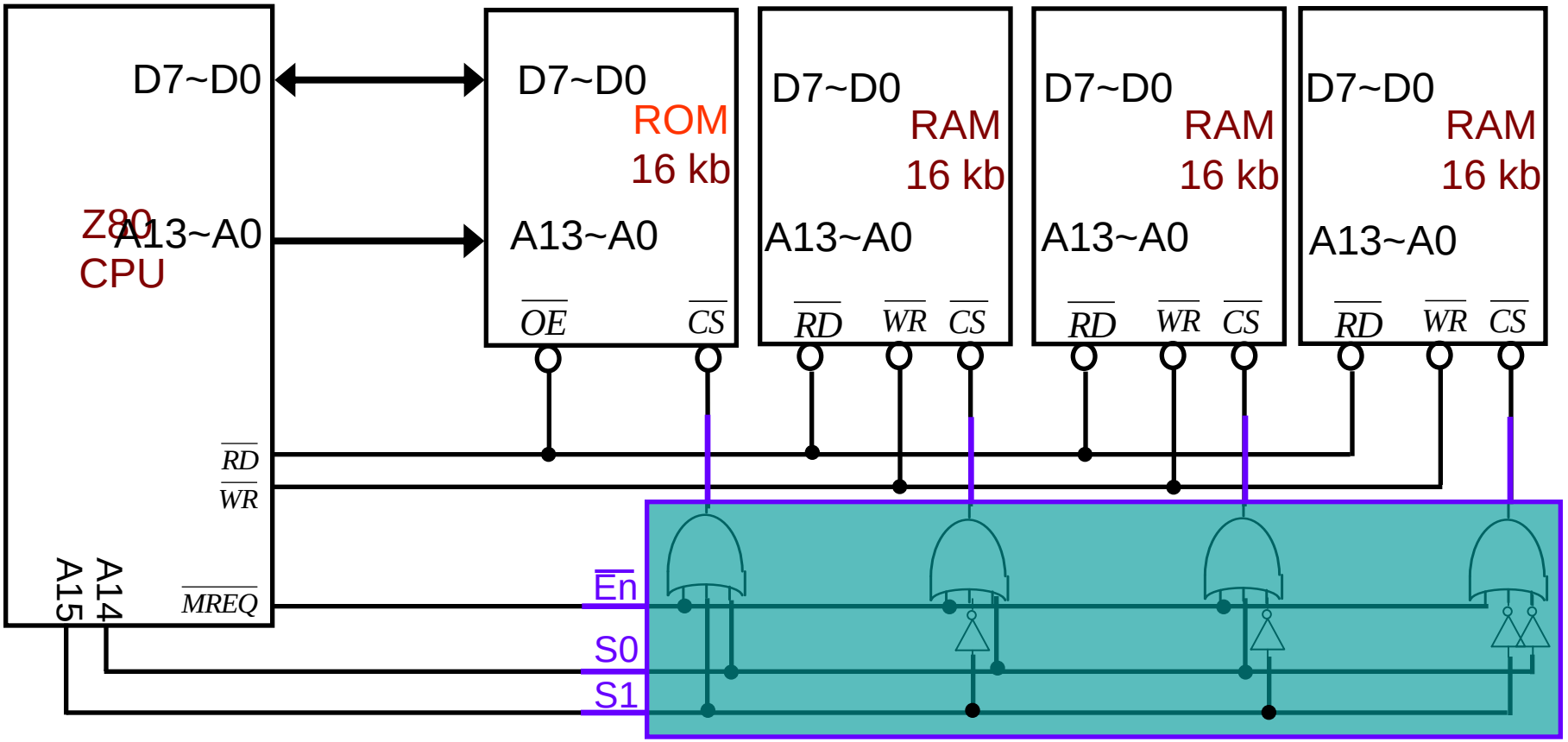
Memory connection (cont.)

32 kb ROM and 32 kb RAM
ROM doesn't have wr signal



Memory connection (cont.)

There is 4 memory chip
A14 and A15 applied to chip selection



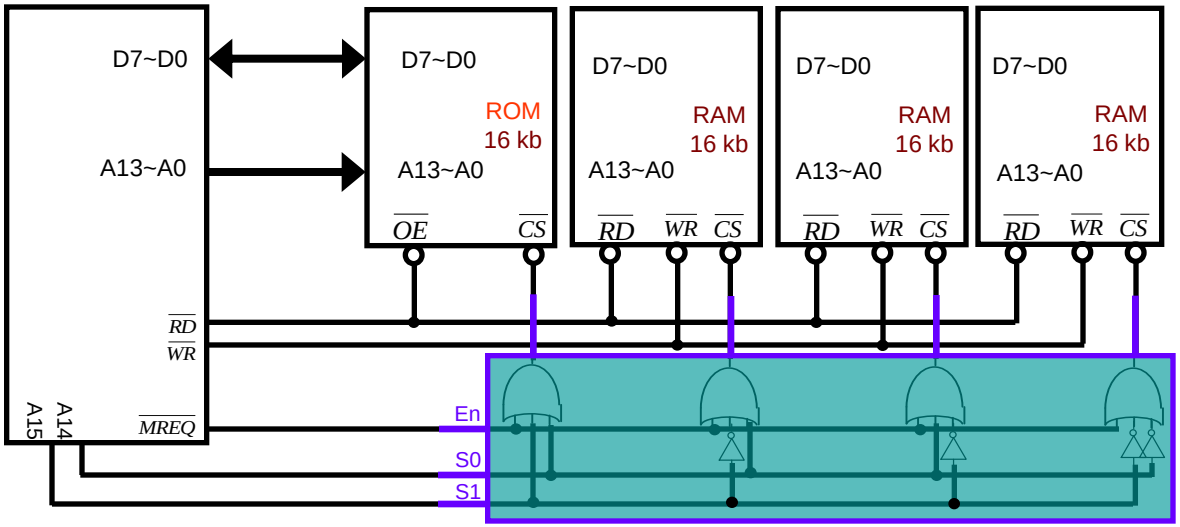
Address Bit Map

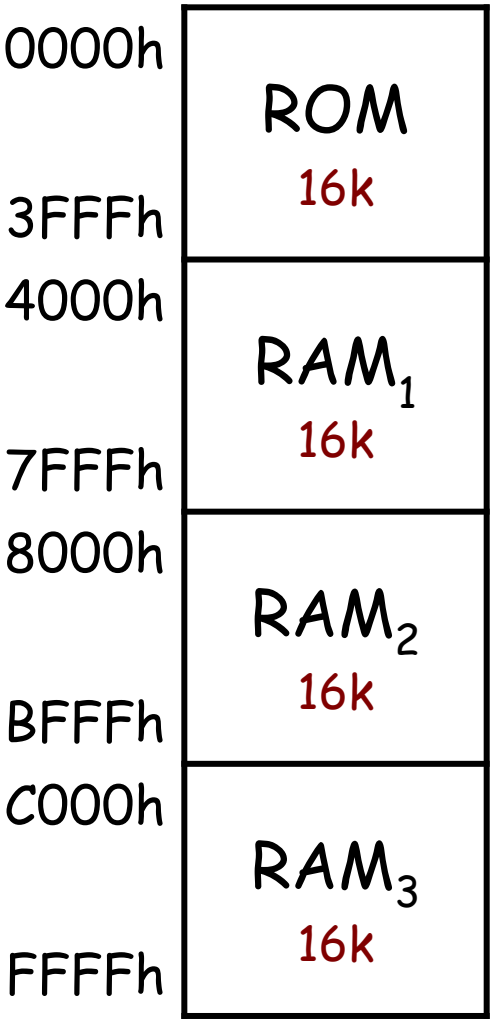
Selects chip Selects location within chips

A15 to A0 (HEX)	AA 11 54	AA 11 32	AAAA 1198 10	AAAA 7654	AAAA 3210	Memory Chip
0000h	00	00	0000	0000	0000	ROM
3FFFh	00	11	1111	1111	1111	
4000h	01	00	0000	0000	0000	RAM_1
7FFFh	01	11	1111	1111	1111	
8000h	10	00	0000	0000	0000	RAM_2
BFFFh	10	11	1111	1111	1111	
C000h	11	00	0000	0000	0000	RAM_3
FFFFh	11	11	1111	1111	1111	

Memory Map

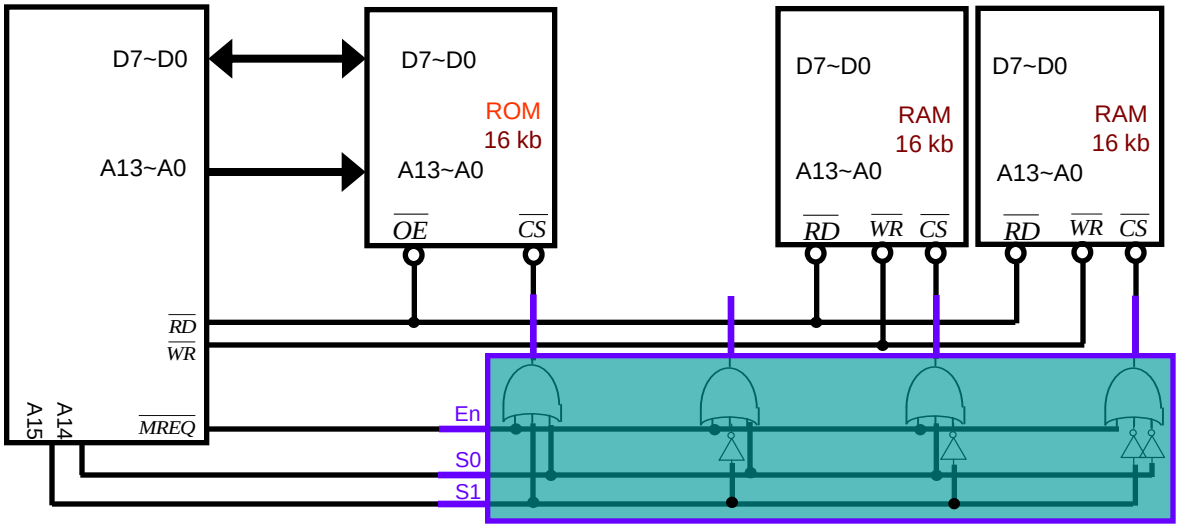
- Represents the memory type
- Address area of each memory chip
- Empty area

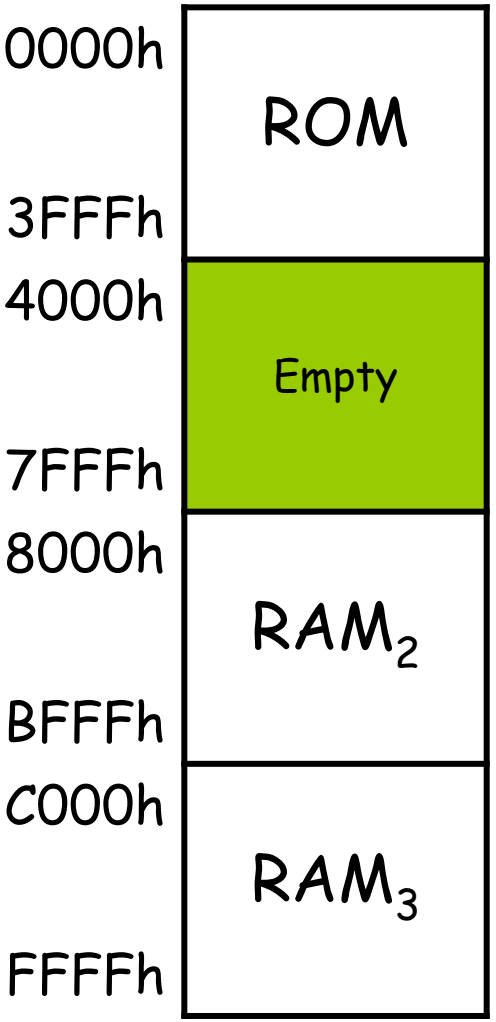




Memory Map

- Empty Area cann't write and read
- Read op. returns FFh value (usualy)
- Write op. cann't store any value on it



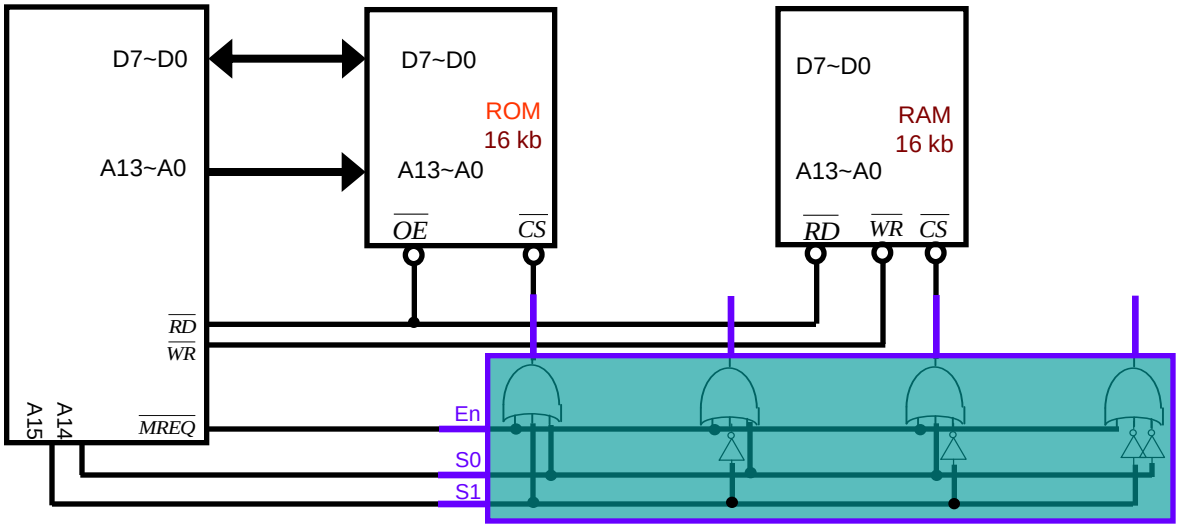


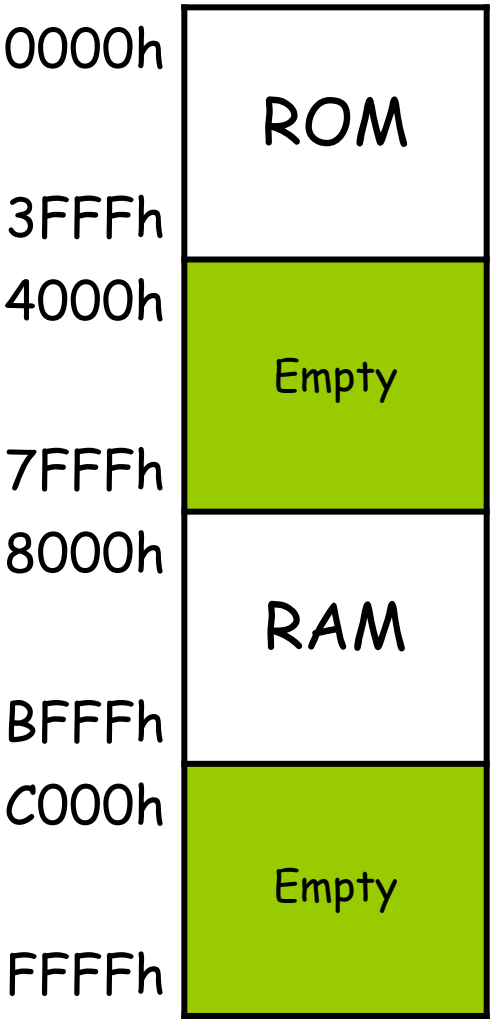
Memory Map

- Empty Area cann't write and read

-Read op. returns FFh value (usualy)

-Write op. cann't store any value on it





Full and Partial Decoding

Full (exhaust) Decoding

- All of the address lines are connected to any memory/device to perform selection

-Absolute address : any memory location has one address

Partial Decoding

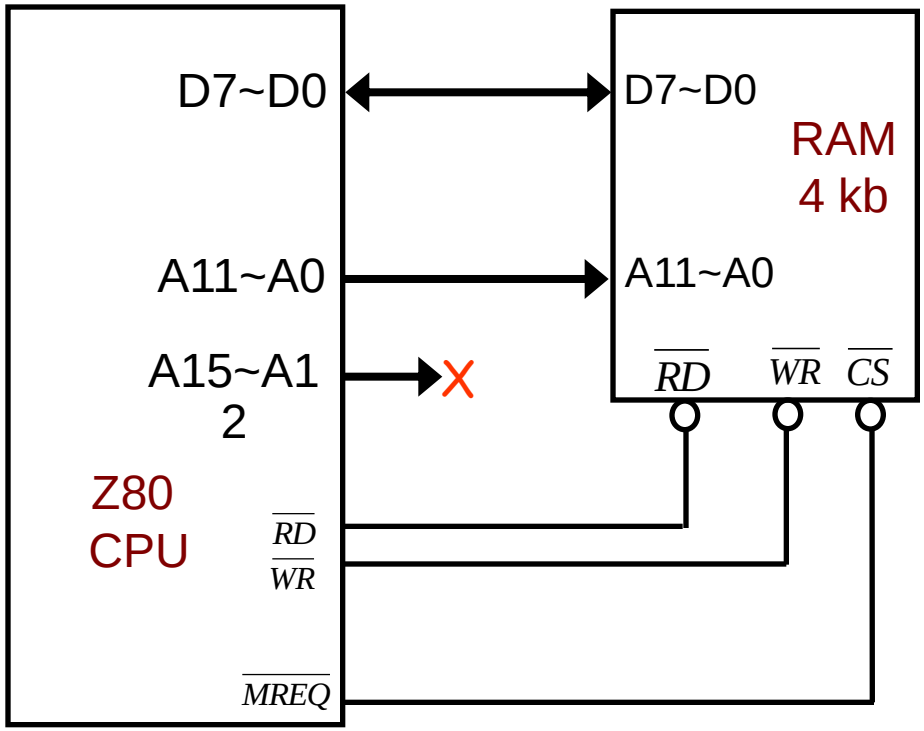
- When some of the address lines are connected the memory/device to perform selection

- Using this type of decoding results into roll-over addresses (fold back or shading).

- roll-over address : any memory location has more than one address

Partial Decoding

-A15~A12 has no connection
-Then doesn't play any role in addressing
-What is the Memory and Address Bit map?



Partial Decoding

Every memory location has more than one address

For example first RAM location has addresses:

0000h

1000h

2000h

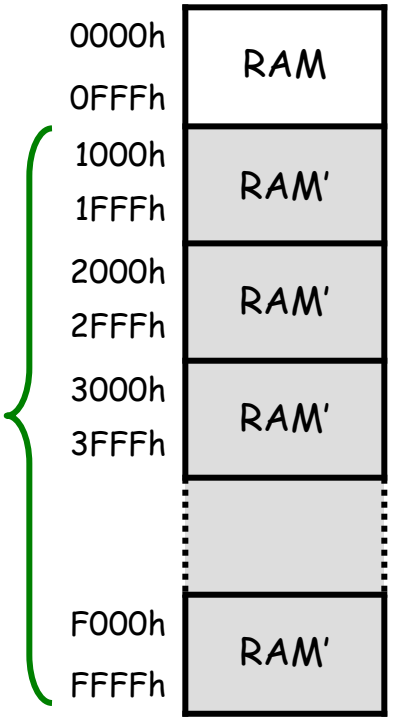
3000h

.....

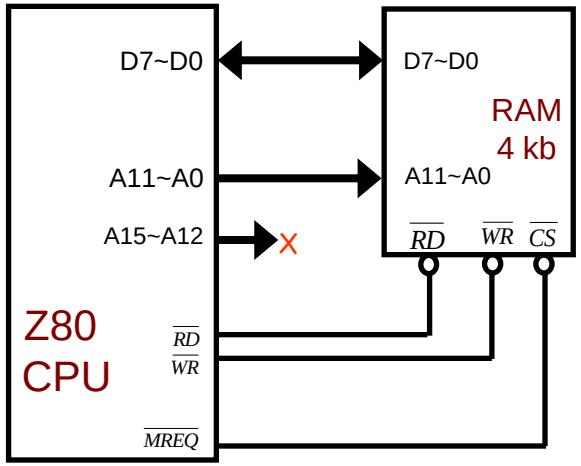
.....

F000h

Roll-over Address

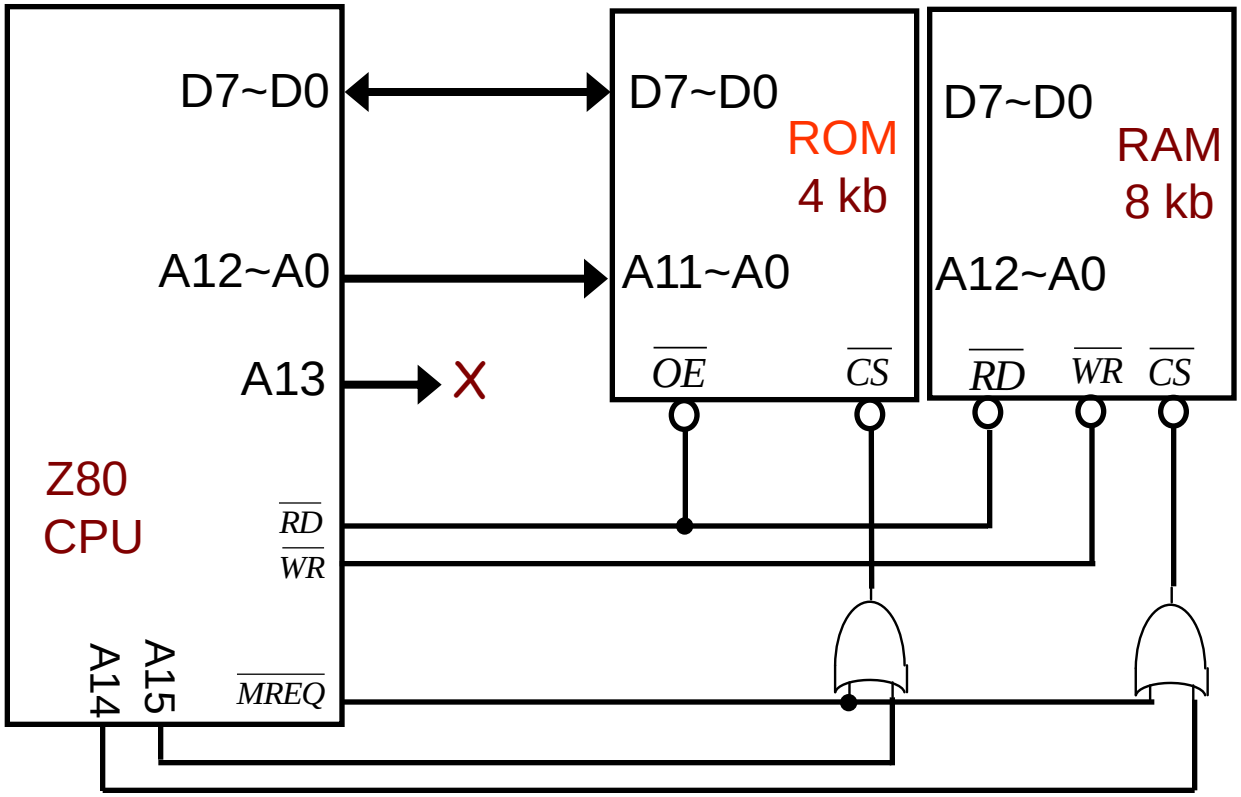


A15 to A0 (HEX)	AAAA 1111 5432	AAAA 1198 10	AAAA 7654	AAAA 3210	Memory Chip
X000h	xxxx	0000	0000	0000	RAM
XFFFh	xxxx	1111	1111	1111	



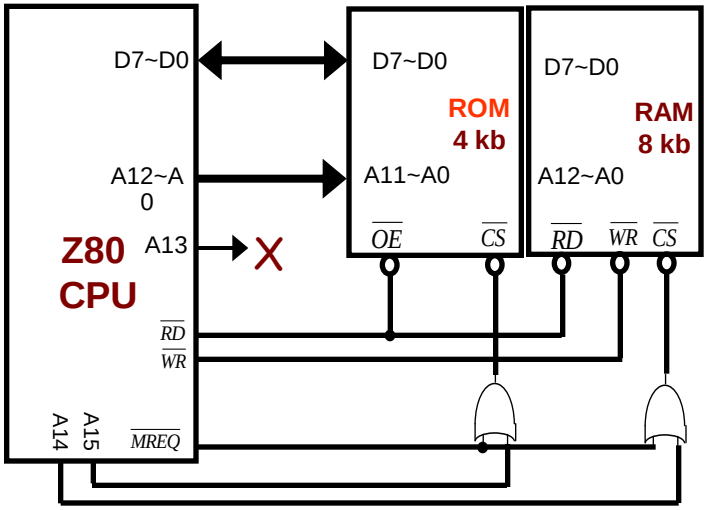
Partial Decoding

A12 only connected to RAM
A13 has no connection
What is the memory map?



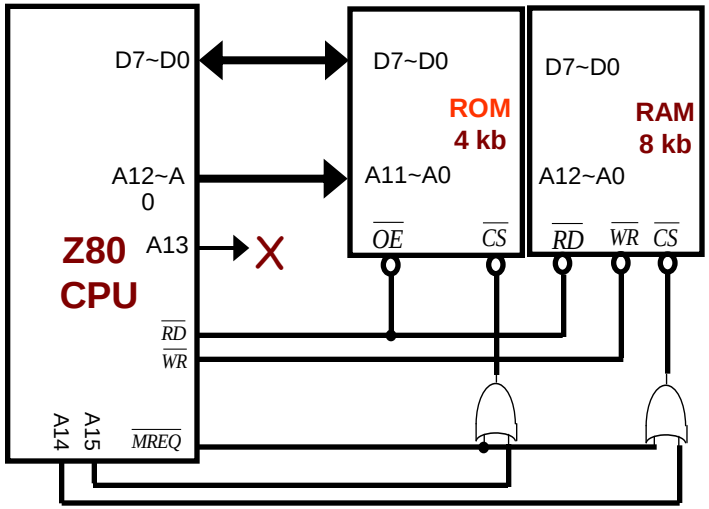
Partial Decoding

- 8 roll-over address for ROM
- 4 roll-over address for RAM

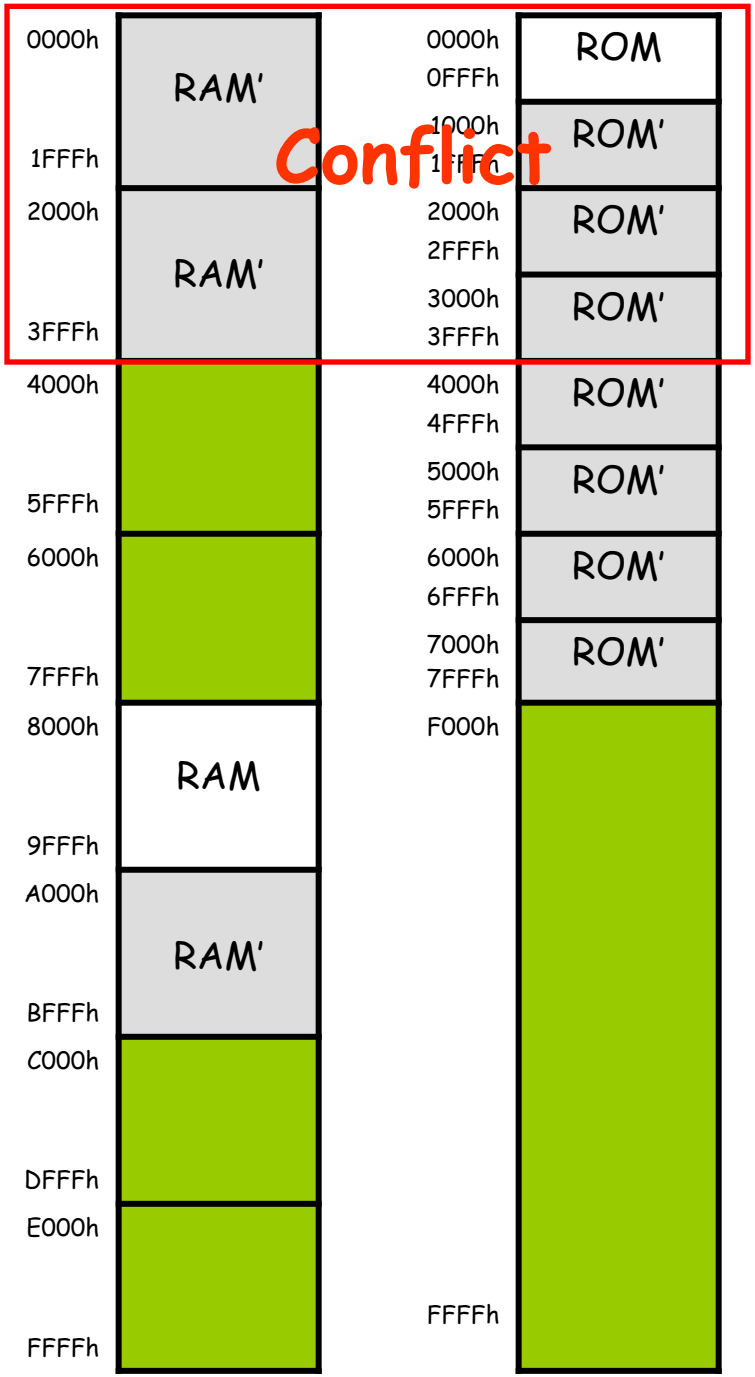


AAAA 1111 5432	AAAA 1198 10	AAAA 7654	AAAA 3210	Memory Chip
0xxx	0000	0000	0000	ROM
0xxx	1111	1111	1111	
X0x0	0000	0000	0000	RAM
X0x1	1111	1111	1111	

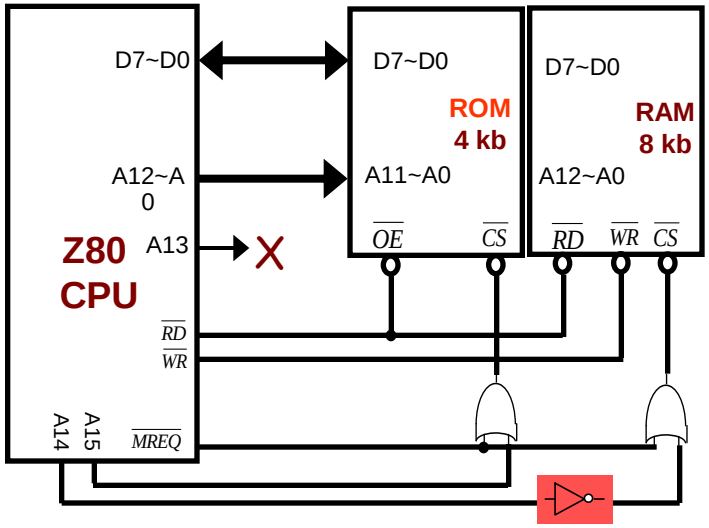
Partial Decoding



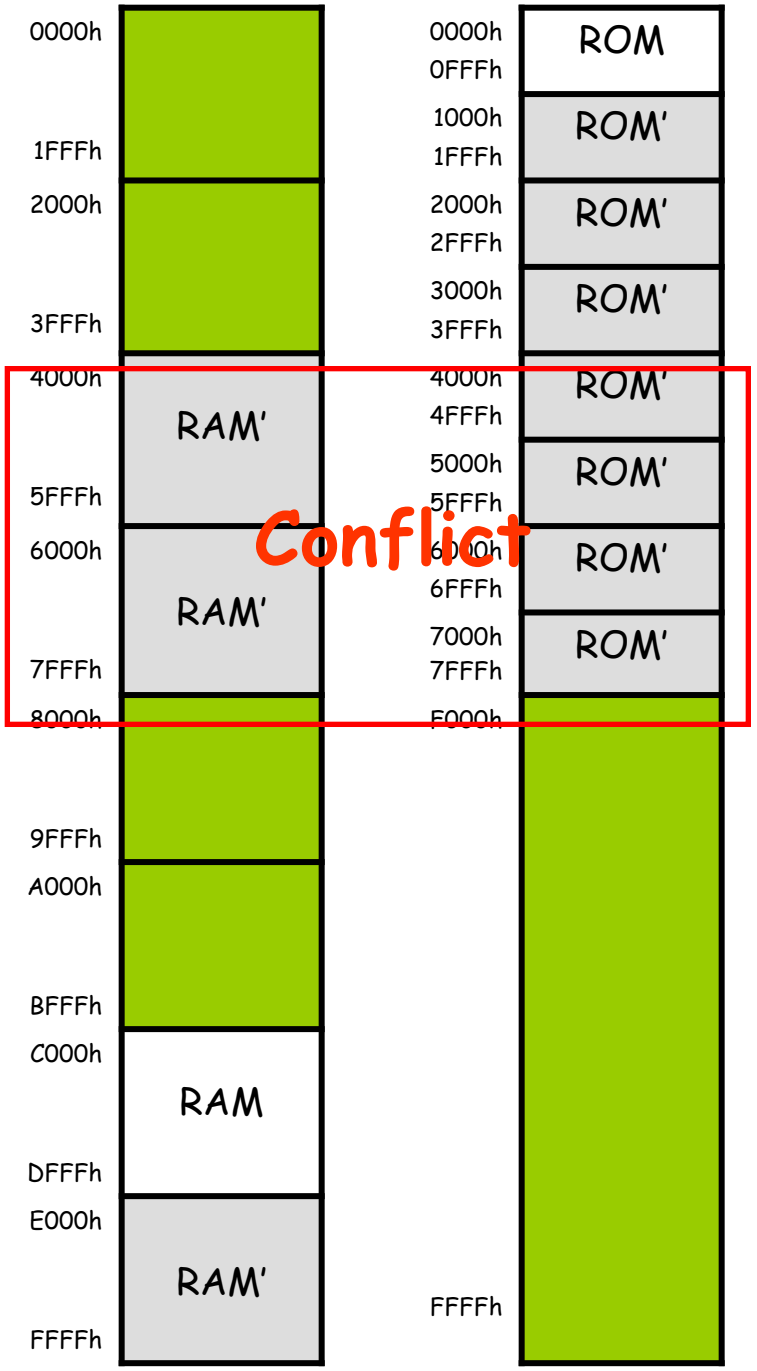
AAAA 1111 5432	AAAA 1198 10	AAAA 7654	AAAA 3210	Memory Chip
0xxx	0000	0000	0000	4k ROM
0xxx	1111	1111	1111	
X0x0	0000	0000	0000	8k RAM
X0x1	1111	1111	1111	



Partial Decoding

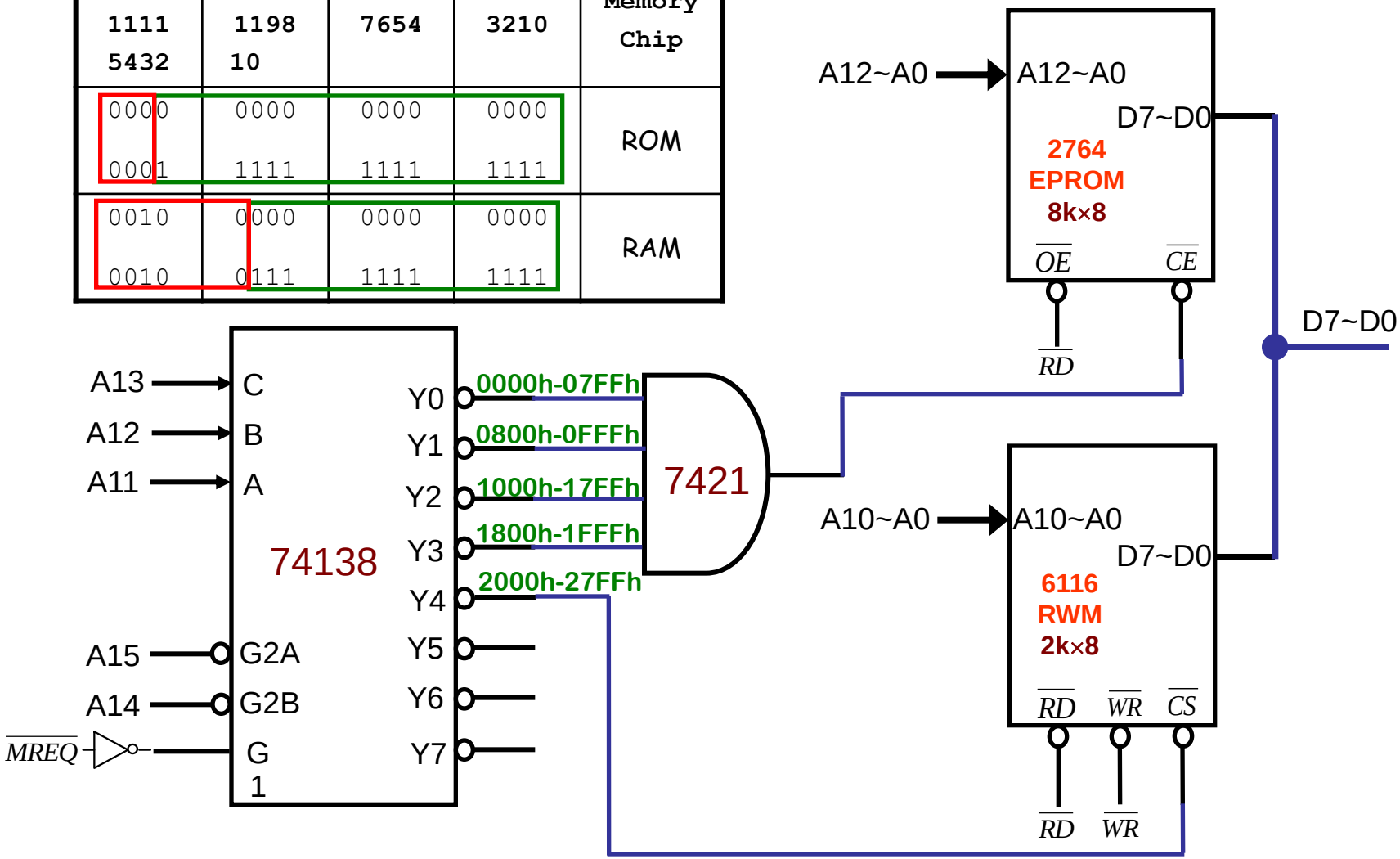


AAAA 1111 5432	AAAA 1198 10	AAAA 7654	AAAA 3210	Memory Chip
0xxx	0000	0000	0000	4k ROM
0xxx	1111	1111	1111	
X1x0	0000	0000	0000	8k RAM
X1x1	1111	1111	1111	



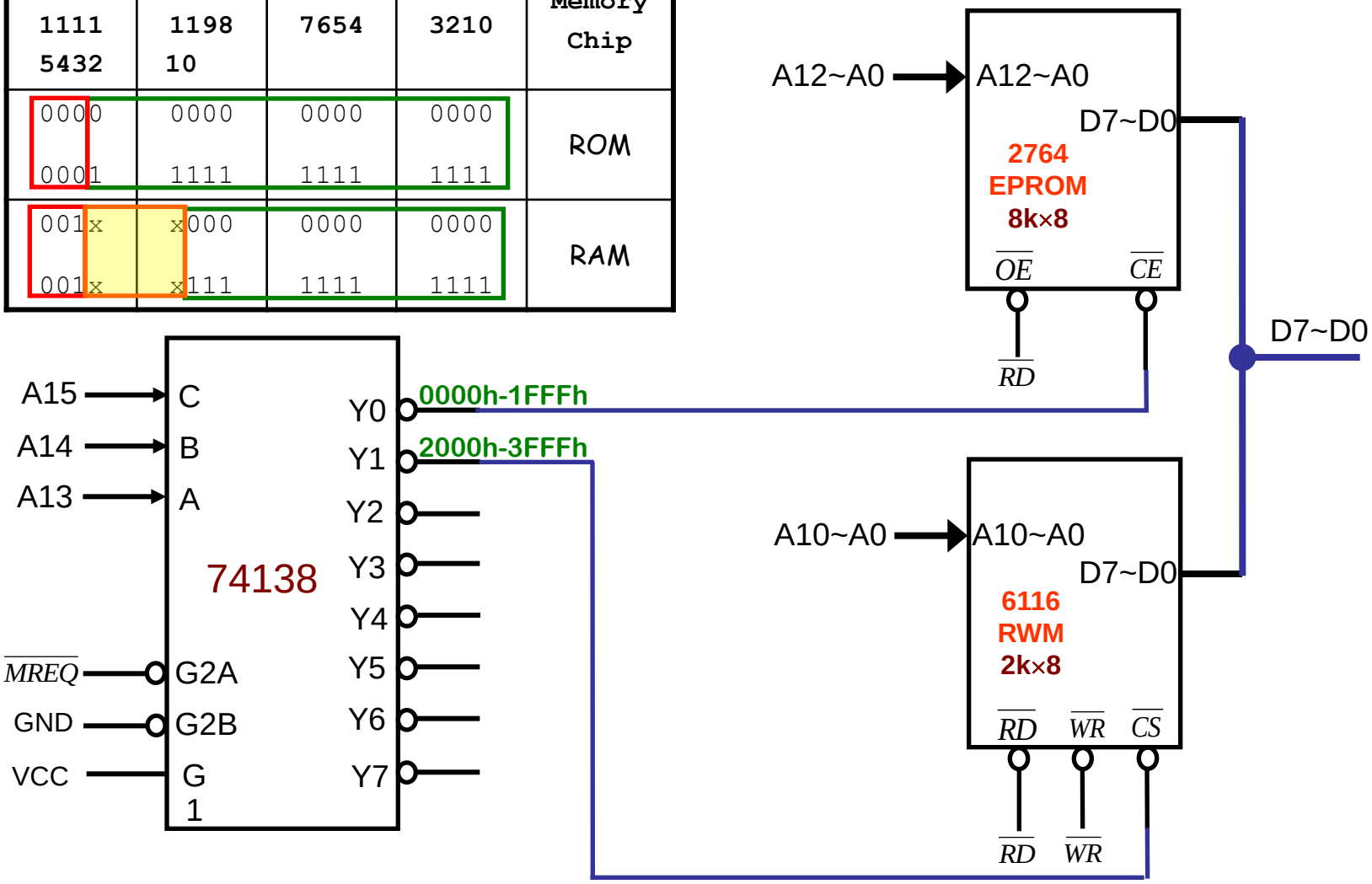
Full (exhaustive) decoding

AAAA 1111 5432	AAAA 1198 10	AAAA 7654	AAAA 3210	Memory Chip
0000 0001	0000 1111	0000 1111	0000 1111	ROM
0010 0010	0000 0111	0000 1111	0000 1111	RAM

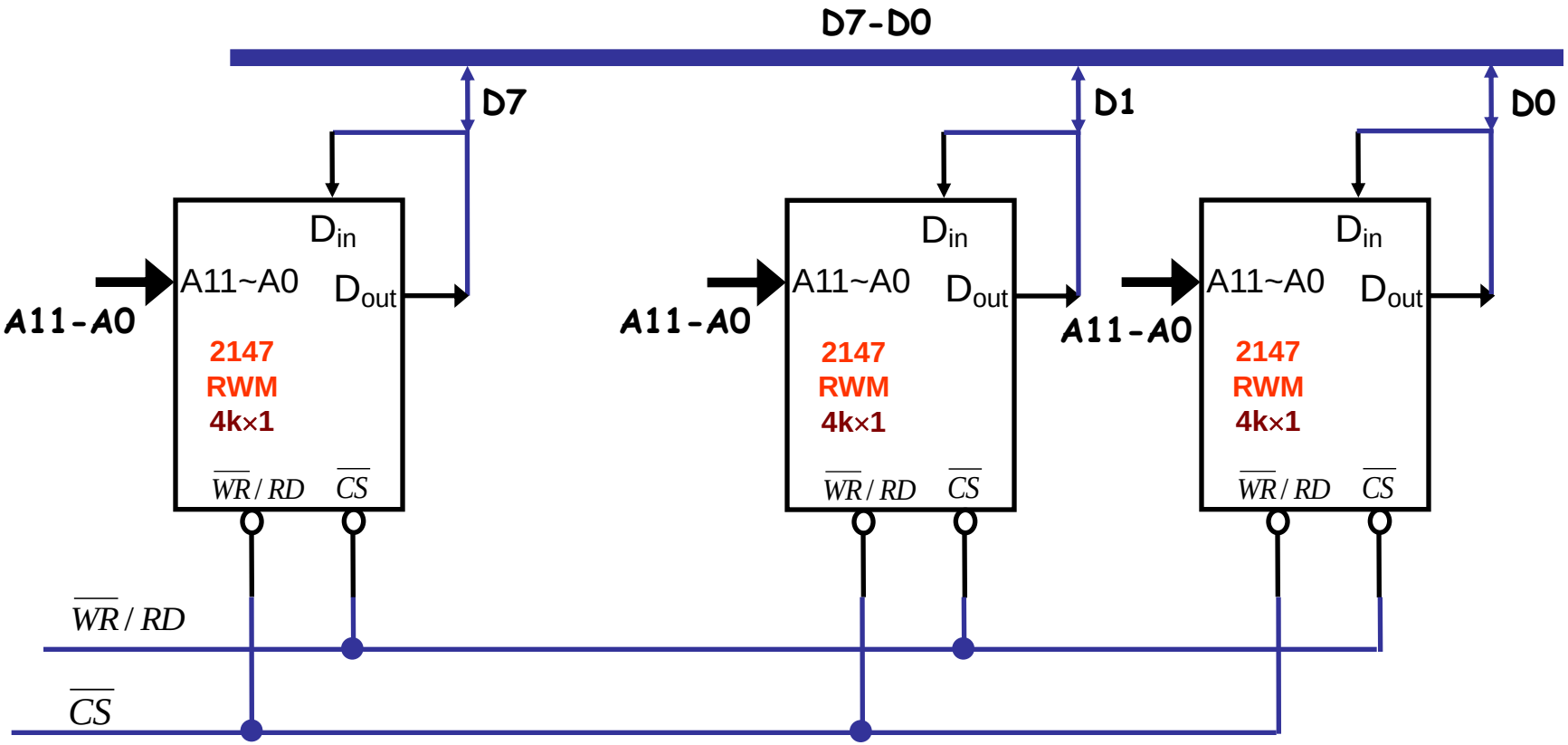


Partial decoding

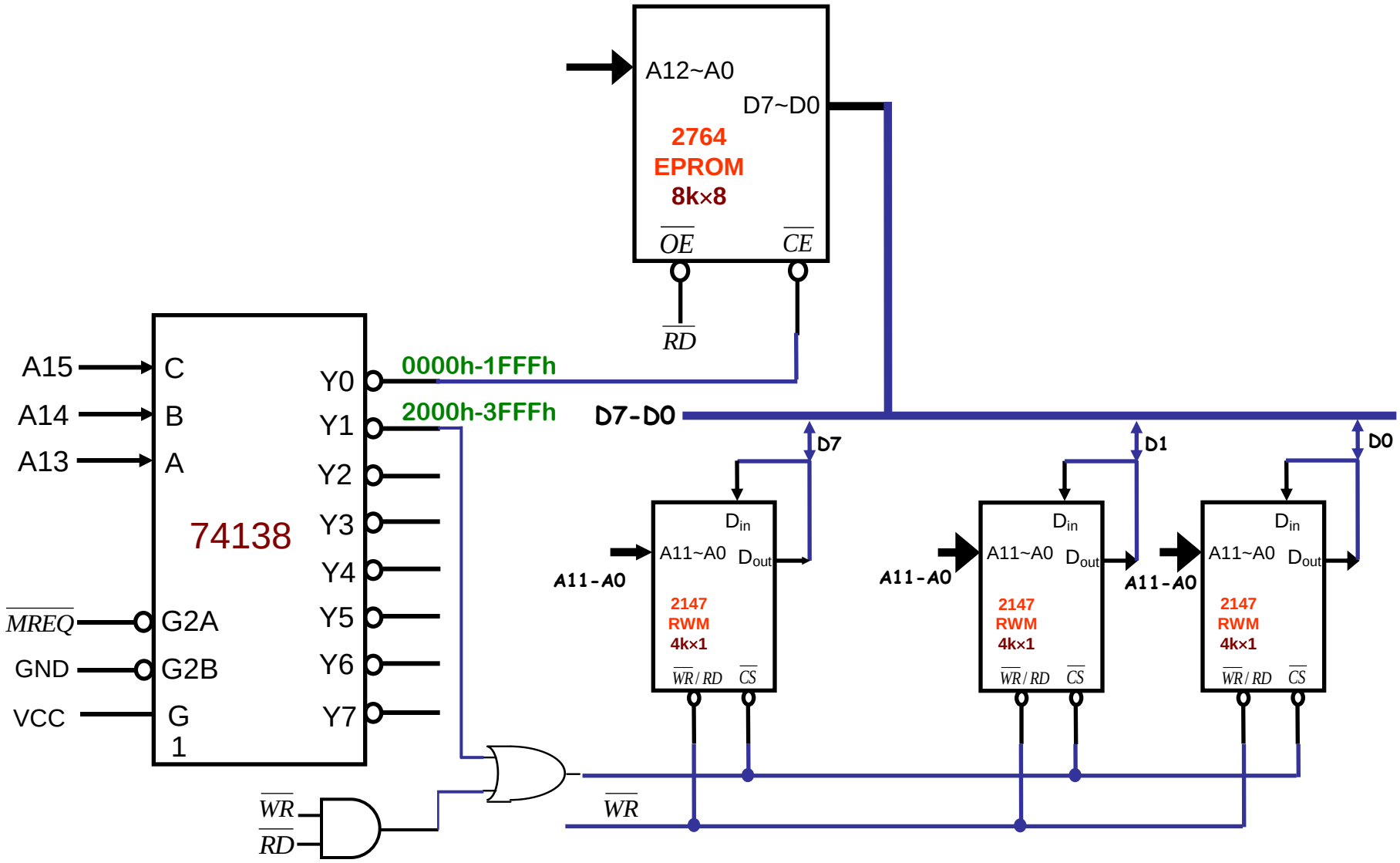
AAAA 1111 5432	AAAA 1198 10	AAAA 7654	AAAA 3210	Memory Chip
0000 0001	0000 1111	0000 1111	0000 1111	ROM
001x 001x	x000 x111	0000 1111	0000 1111	RAM



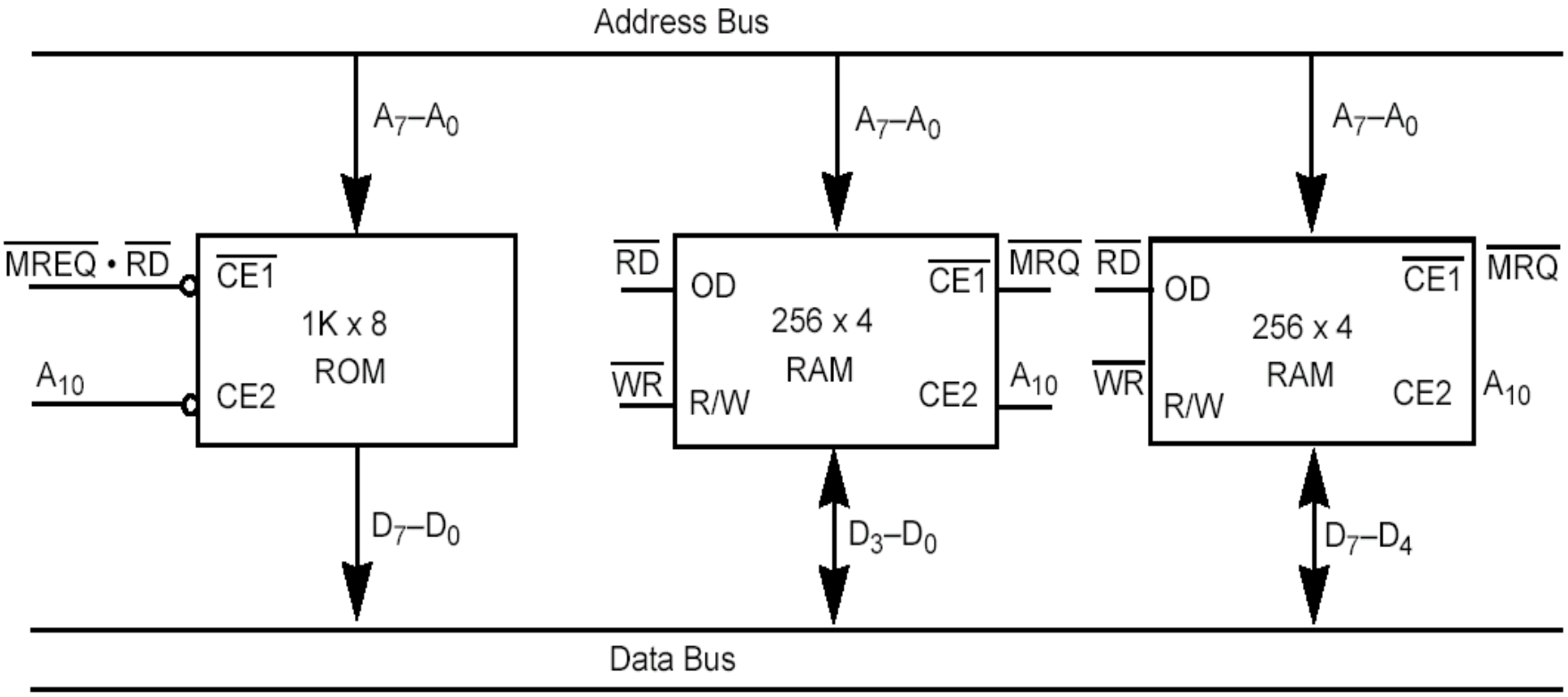
1 Bit Memory With Separated I/O



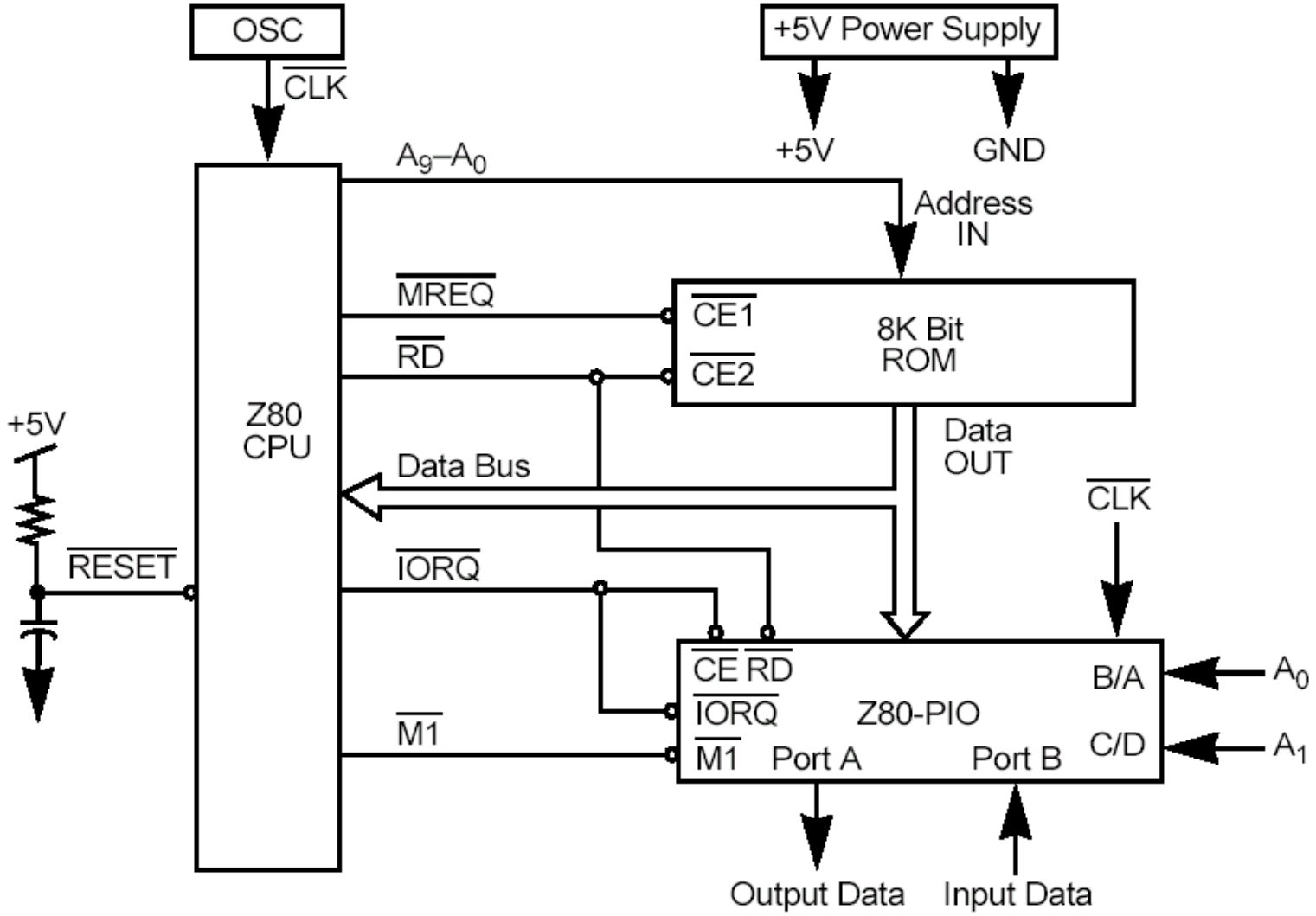
What is the memory(addr. bit) map



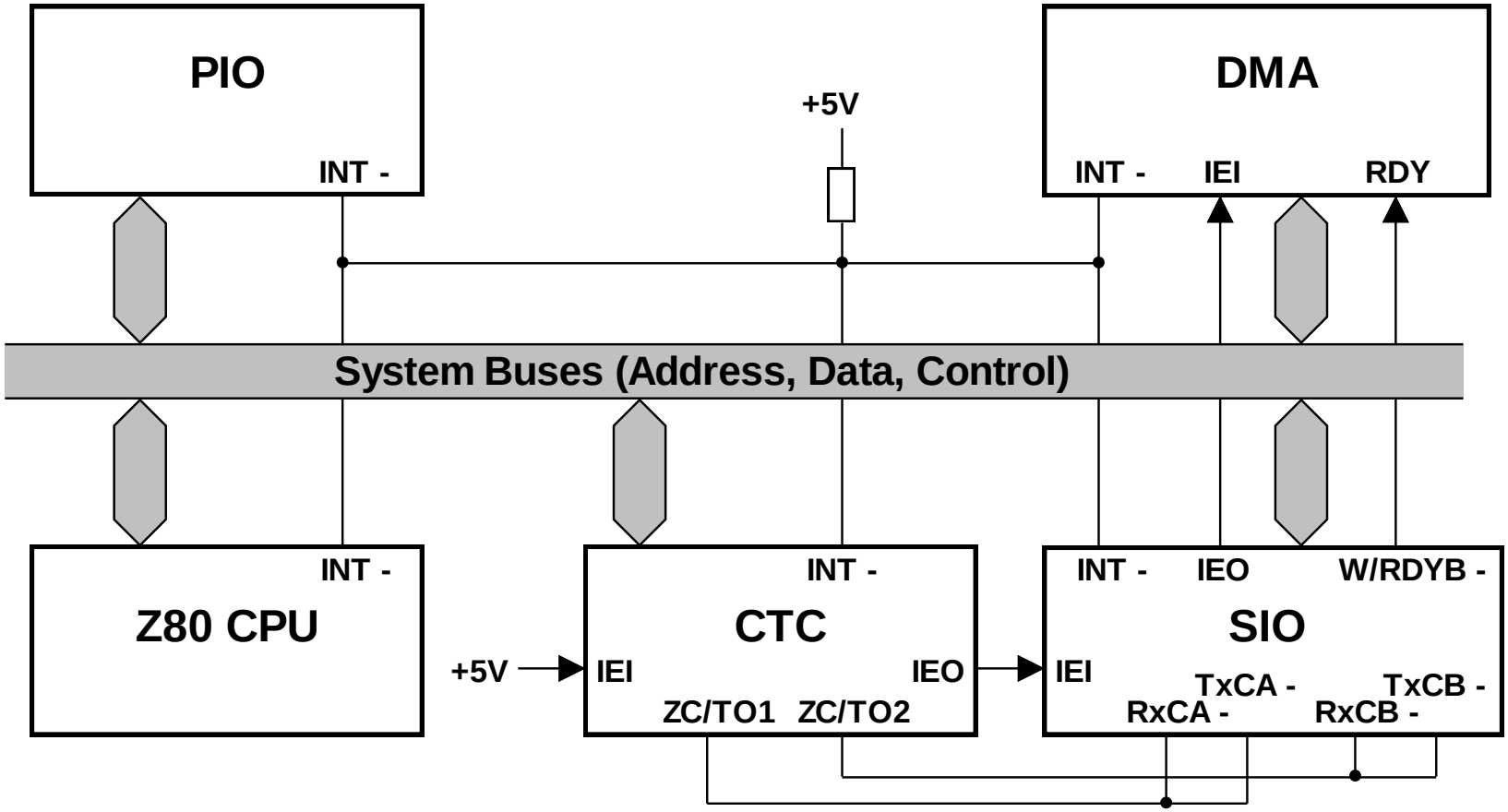
Adding RAM & ROM



Minimum Z80 Computer System



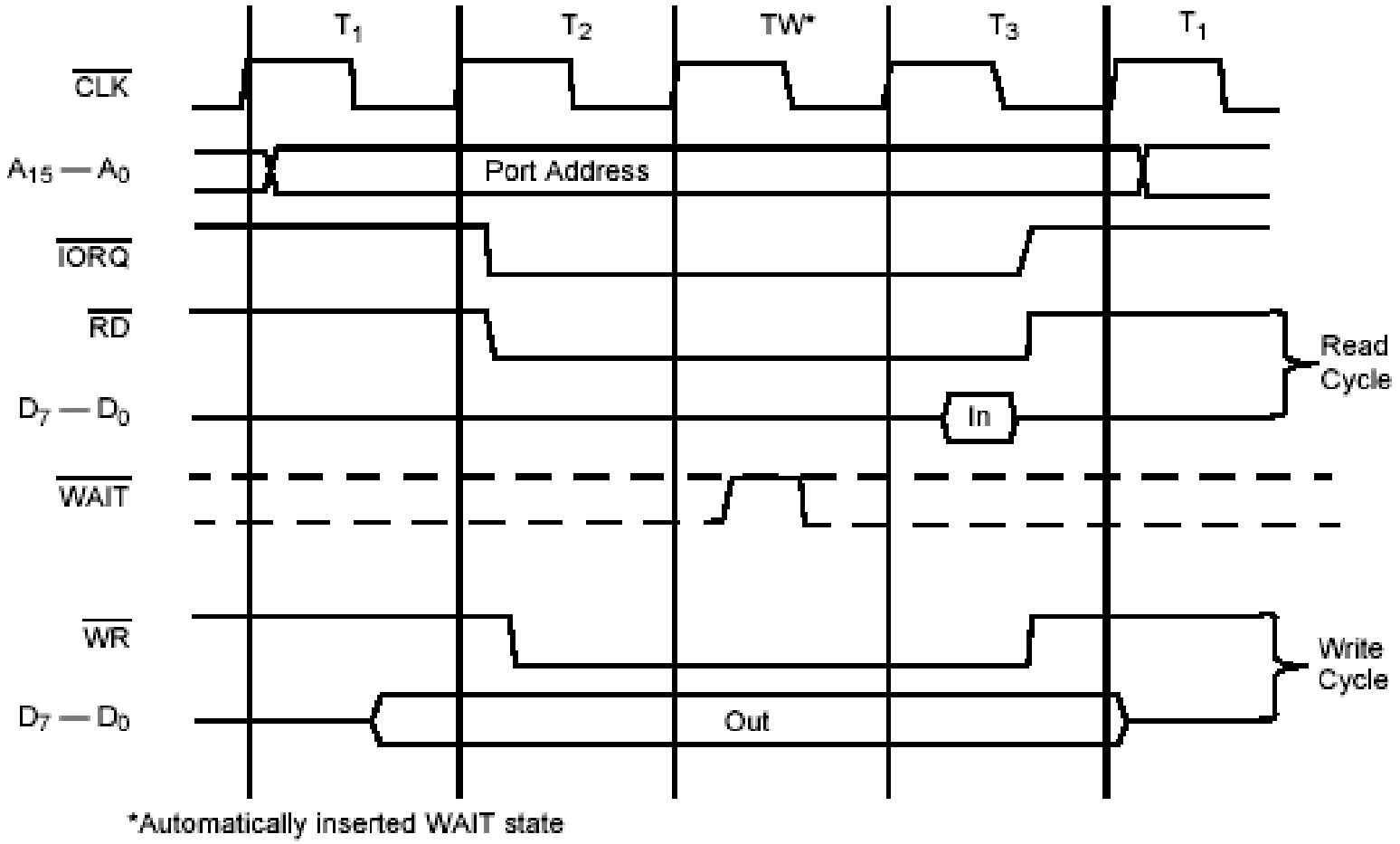
Z80-µP-Family (Typical Environment)



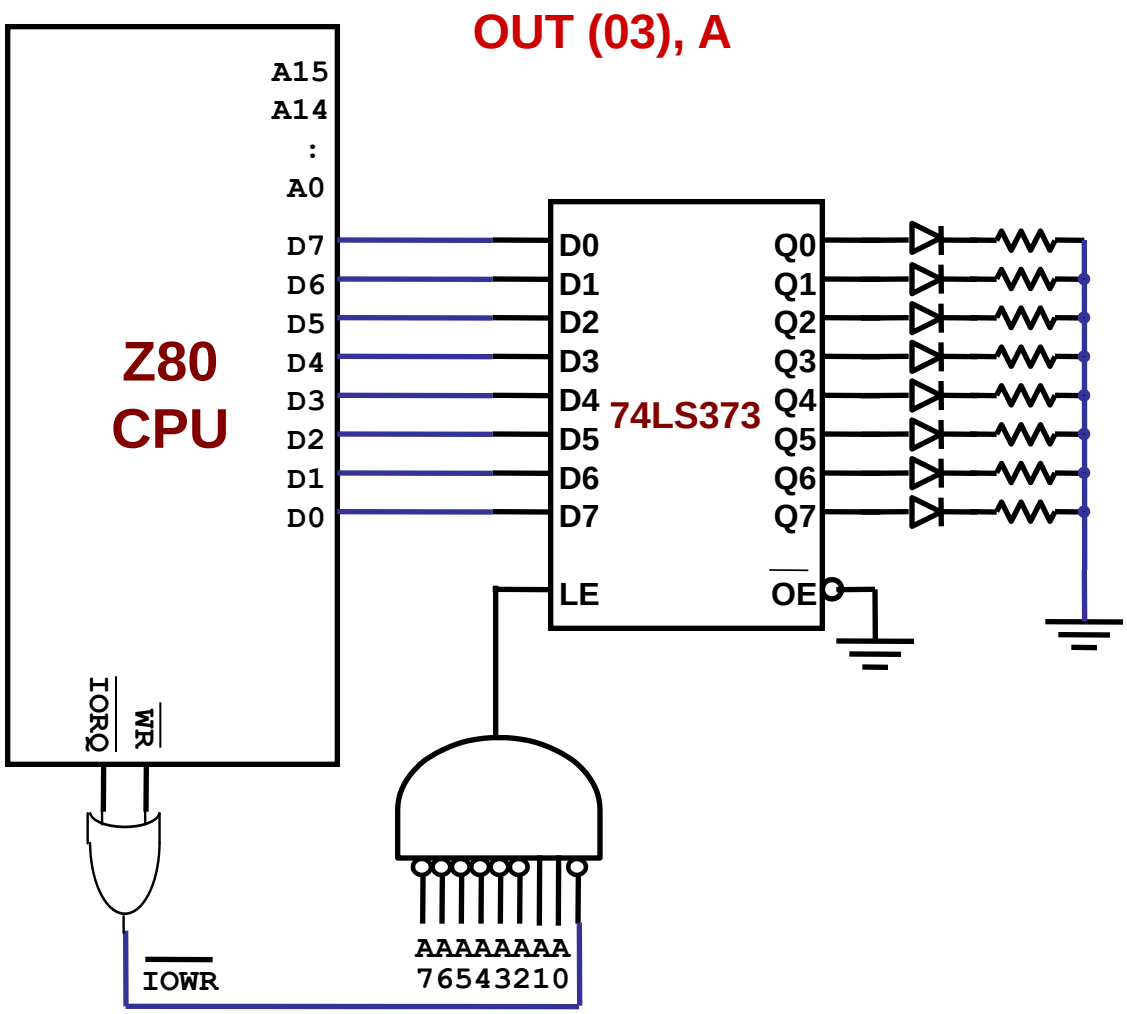
Z80 Input Output

- Z80 at most could have 256 input port and 256 output
- 8 bit port address is placed on A7-A0 pin to select the I/O device
- **OUT (n), A**
  - n is 8 bit port address
  - Content of A is data
- **OUT (C), r**
  - Content of C is a port address
  - r is a data register
- **IN A, (n)**
  - n is 8 bit port address
  - Data is transfered to A
- **IN r (C)**
  - Content of Reg C is a port address
  - Input data is transfered to r (data reg)

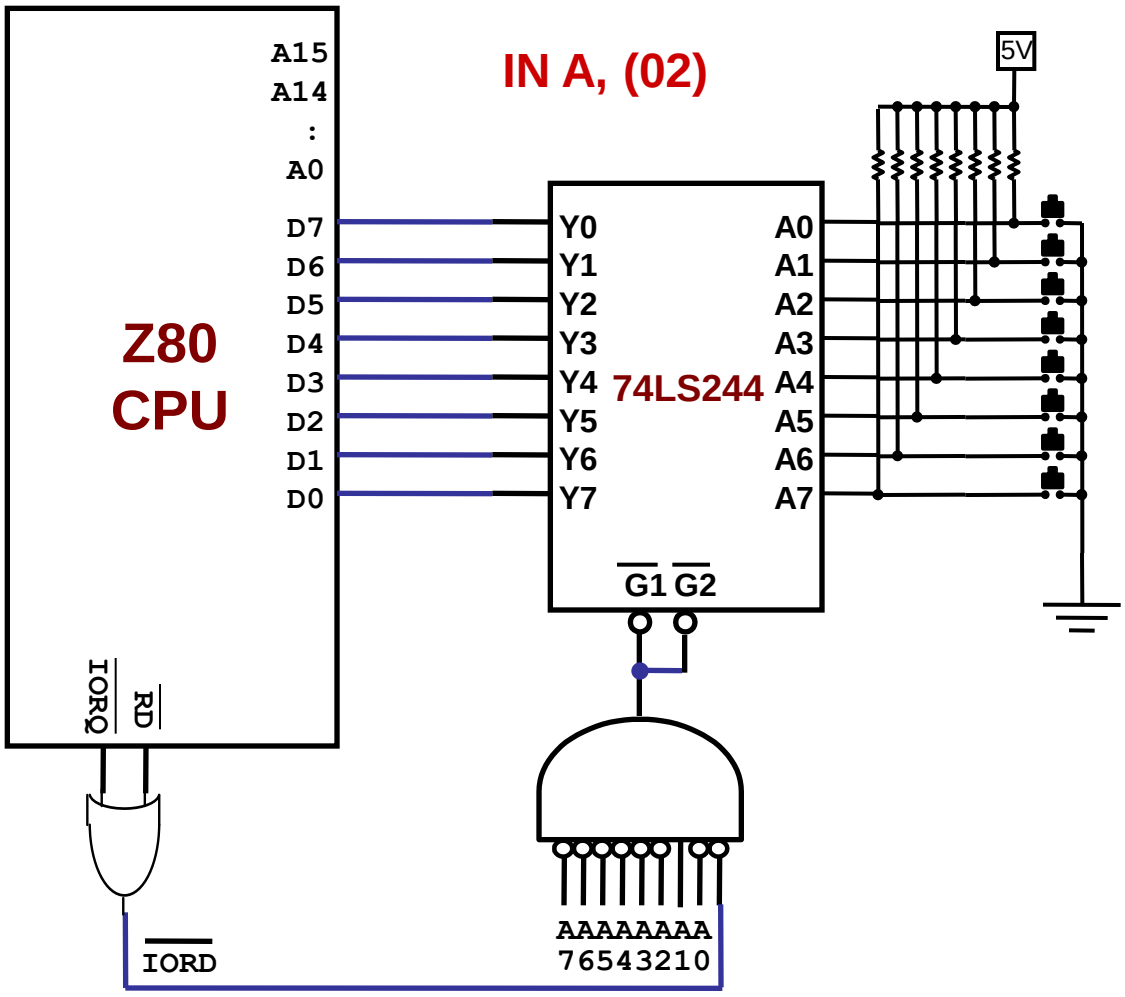
Remember IO read/write cycle



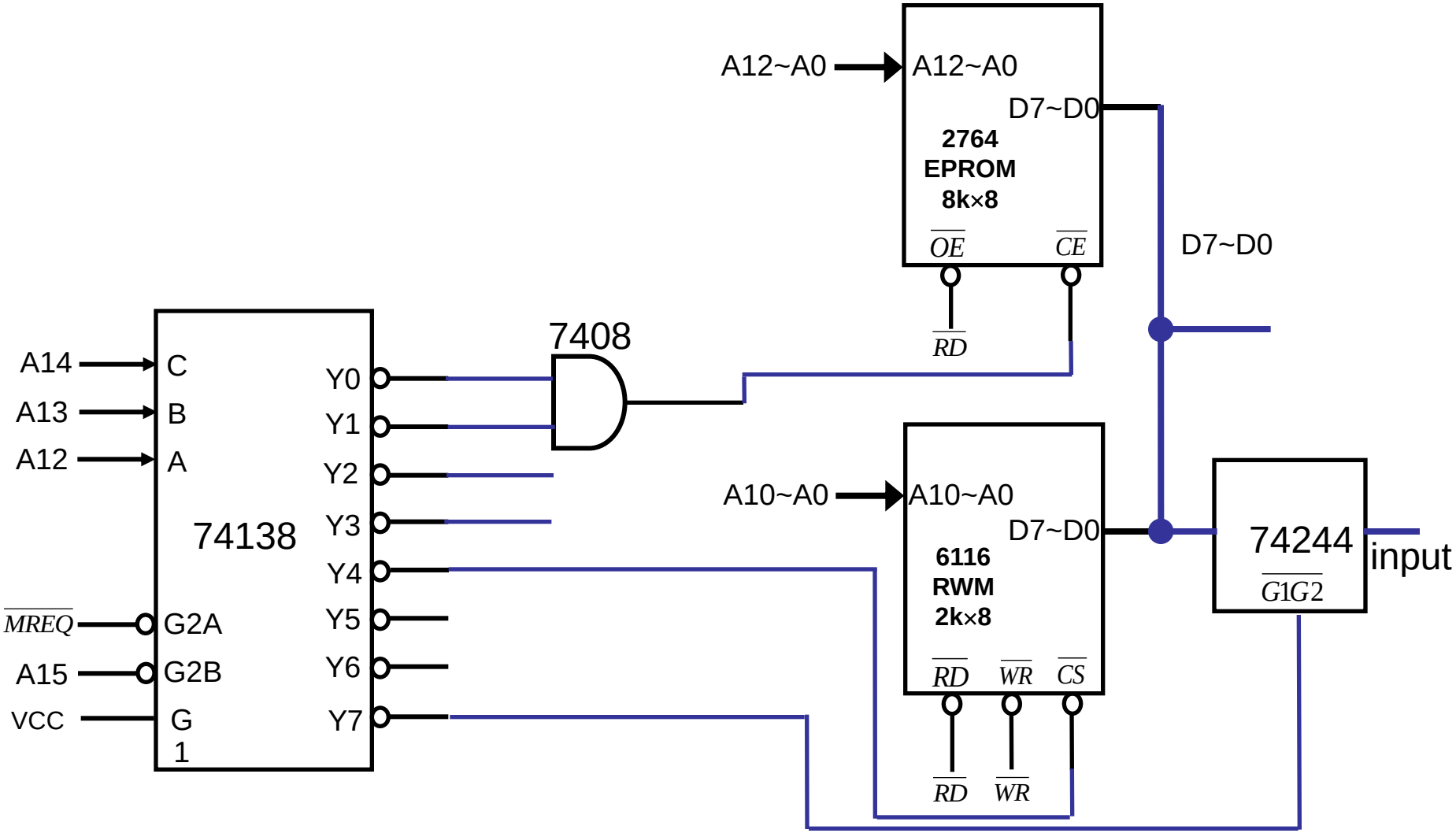
Z80 and simple output port



Z80 and simple input port



Memory map & Address Bit map



نرم افزار

- برنامه : یک سری دستورات مجزا که بطور متوالی پشت سرهم قرار میگیرند.

- مجموعه دستورات یک ریزپردازنده با ساختمان داخلی آن رابطه مستقیم دارد.

- زبان ماشین : بیان دستورات قابل اجرای ریزپردازنده با استفاده از صفرو یک.

- - زبان اسمبلی : برای سهولت کار برنامه نویسی از عبارات یا
- واژه های مفهوم تر، نظیر add,sub,...
- که بخاطر سپردن آنها ساده ترمی باشد استفاده میکند.
-
- اسمبلر : مترجم زبان اسمبلی به زبان ماشین است.
- ورودی اسمبلر ، کد برنامه بزبان اسمبلی است که به آن
- کد منبع (source code) گفته میشود.
- خروجی اسمبلر کد برنامه ای است بزبان ماشین که به آن
- کد هدف (object code) گفته میشود.

- - Resident assembler and cross- assembler

دستورالعمل ها

- دستورازدوبخش operand,opcode تشکیل میشود.
- Opcode (operation cod)- این بخش از دستور مبین عملی است که باید انجام گیرید.
- Operand(عملوند) – این بخش از دستورمشخص کننده:

-اطلاعات

-آدرس اطلاعات

-آدرس محل قرارگرفتن نتیجه عمل

آدرس دهی
addressing

- دستورالعمل ها بروی اطلاعاتی که در ثباتهای داخلی یا حافظه های جانبی و یا تراشه های ورودی و خروجی میباشد عملیاتی را انجام میدهند.
- هر دستورالعمل علاوه بر تعیین نوع کار مشخص میکند که دستورالعمل مربوط به ثباتهای درون ریزپردازنده یا تراشه های حافظه و یا تراشه های ورودی و خروجی است. تعیین این موضوع را آدرس دهی گویند.

- آدرس دهی به روشی گفته میشود که ریزپردازنده محلی که عملیات باید صورت گیرد را در موقع عملیات تولید میکند.

روش های آدرس دهی
Addressing mode

- در Z80 ده روش آدرس دهی وجود دارد.

۱- روش آدرس دهی انباره یا ضمنی :

اگر حافظه مورد آدرس انباره باشد به آن ادرس دهی انباره گویند.
در این گونه موارد اپرند مورد نظر انباره می باشد.

- در بعضی از دستورات اپرند A بطور ضمنی در کد عمل وجود دارد که به آن آدرس دهی ضمنی نیز گفته می شود.

CPL CP s

۲- روش آدرس دهی ثباتی :

- اپرند بسیاری ازدستورهای ریزپردازنده یک یا چند عدد از ثباتها میباشد.
  دراین گونه موارد چون حافظه مورد آدرس یکی (چند عدد) ازثبات ها بوده، به آن روش آدرس دهی ثباتی گفته میشود.
- LD r , r’
  در این قالب دستوری r , r’ میتوانند هریک از ثباتهای
- B,C,D,E,H,L,A باشند.

- این دستورات از کد 01xxxyyy استفاده میکنند. LD A ,C

- کد ماشینی دستور LD A,C 01 111 001 یا 79 h میباشد.

روش آدرس دهی ثباتی دارای دو مزیت بسیار مهم میباشد:

۱- ثباتها در درون ریزپردازنده میباشند و در نتیجه سرعت تبادل اطلاعات بین آنها نسبت به تراشه های بیرونی بیشتر خواهد بود.

۲- تعداد ثباتها محدود و در نتیجه تعداد بیت لازم برای مشخص کردن یک ثبات کم بوده و در نتیجه طول یک دستور از نظر تعداد بایت نیز کم خواهد شد و این خود سرعت اجرای این گونه دستورات را بالا میبرد.

۳- روش آدرس دهی فوری یا بی واسطه
Immediate Addressing

- بعضی از دستورات اطلاعات مورد نیاز خود را در خود دارند.
- در این حالت عین اطلاعات و نه آدرس آنها در دستور مورد نظر وجود دارد.
- LD A , 35 h
- این دستور چند بایتی میباشد؟

۴- روش آدرس دهی فوری توسعه یافته

extended immediate addressing

- در صورتیکه آدرس دهی فوری باشد ولی اطلاعات مورد نیاز بیش از یک کلمه از حافظه را اشغال کند به آن آدرس دهی فوری توسعه یافته گویند.
- مجموعه دستورات LD dd,nn نمونه مشخصی از این نوع آدرس دهی در z80 میباشند. این دستور در حالت کلی دارای کد زیر می باشد.
- بایت اول 00dd0001
- بایت دوم n(low)
- بایت سوم n(high)
- مثال : LD BC, 4587h
- کد ماشینی مثال فوق را نشان دهید.

۵- روش آدرس دهی مستقیم
Direct Addressing

- در این روش آدرس اطلاعات مورد نظر جزئی از دستور میباشد.
- LD r , (nn) این دستور محتوی محلی از حافظه که آدرس آن nn میباشد را به ثبات r انتقال میدهد.
- کد ماشینی این دستور در سه بایت به شکل زیر است :
- 00xxx010
- n(low)
- n(high)
- مثال : LD A,(4565h)
- کد ماشینی : 3A
- 65
- 45

۶- روش آدرس دهی غیر مستقیم
Indirect Addressing

- در این روش آدرس اطلاعات مورد نظر خود محتوی یکی از ثباتها (یا زوج ثباتها) میباشد.
- مثال : LD A,(BC) این دستور محتوی محلی از حافظه که آدرس آن محتوی جفت ثبات BC میباشد را به ثبات A انتقال میدهد.
- کاربرد این دستور :

امکان دسترسی به یک سری اطلاعات که بطور متوالی در حافظه قرار دارند.

مثال کاربردی این گونه دستورات

```
        LD BC,0100h
        LD DE,0150h
        LD H,06h
loop:   LD A ,(BC)
        LD (DE) , A
        INC BC
        INC DE
        DEC H
        JPNZ,loop
        HALT
```

۷- روش آدرس دهی شاخص دار
Index Addressing

- از ثبات شاخص جهت نگهداری آدرس در بعضی از دستورات ریزپردازنده استفاده میشود.
- محتوی ثبات شاخص با اطلاعات موجود در دستورات مزبور جمع میشود تا آدرس مورد نظر بدست آید.
- LD A, (Ix + n)
- nیک عدد ۸بیتی مکل ۲ میباشد. (۱۲۷ تا ۱۲۸-)

- از این روش آدرس دهی نیز جهت دسترسی به یک سری اطلاعات که بطور متوالی در حافظه ذخیره گردیده است نیز استفاده میشود.

مثال کاربردی این دستور

- LD Ix ,0100h
- LD H, 06h
- loop: LD A, (Ix+00h)
- LD (Ix+50h) , A
- INC Ix
- DEC H
- JPNZ, loop
- HALT

۸- روش آدرس دهی نسبی
Relative Addressing

- مورد استفاده عمده این روش در دستورات پرش است.
- اطلاعات موجود در این دستور به شمارنده برنامه (PC) اضافه میشود تا آدرس دستور بعدی که باید اجرا گردد مشخص شود.
- دستور JRe یک نمونه ساده از این دستورات در z80 است.
- کد این دستور بصورت :
- بایت اول 00011000
- بایت دوم e-2
- مقدار pc پس از خواندن دوبایت دستور به داخل ریزپردازنده به میزان دو واحد افزایش میابد.به همین دلیل در بایت دوم e با e-2 جایگزین شده است.
- e-2 یک عدد ۸ بیتی مکمل ۲ میباشد.

۹- روش آدرس دهی بیت
Bit Addressing

- در بعضی از ریزپردازنده ها دستورهایی وجود دارند که بوسیله آنها می توان به تک تک بیتهای یک بایت از حافظه یا یکی از تباتها دست یافت و مقدار ان را آزمایش نمود و یا مقدار آن را تغییر داد.

- این نوع دستورها از روش آدرس دهی بیت استفاده می کنند

- Set b ,r این دستور بیت شماره b از ثبات r را یک میکند.

۱۰ – روش آدرس دهی مستقیم صفحه صفر
Zero Page Direct Addressing

- دستورات ورودی وخروجی در Z80 :
- IN A,(n)
- OUT (n) ,A
- n یک عدد هشت بیتی است که مشخص کننده آدرس پورت مورد نظر میباشد.
- n دارای محدوده آدرس FFh تا 00h میباشد .

(محدوده n در صفحه صفر قرار دارد)

دستورات انتقال ۸ بیتی

8- bit load group

این دستورالعمل ها مربوط به انتقال اطلاعات ۸ بیتی بین ثباتهای داخلی و نوشتن از ثباتهای داخلی به حافظه و بلعکس میباشد.

Mnemonic: LD r , s

Symbolic

Operation: r ⟵ s

Flags :

S	Z	X	H	X	P/N	N	C

Op- code : 01 r s

No. of byte : 1

No. of M cycle : 1

سیکل ماشینی یعنی مراجعه به حافظه در هنگام اجرای دستور

No. of T cycle : 4

دستورات انتقال ۸ بیتی ادامه:

Ld r , n	
Ld r , (HL)	
Ld r , (IX + d)	Ld r , (IY + d)
Ld (HL) , r	
Ld (IX + d) , r	Ld (IY + d) , r
Ld (HL) , n	
Ld (IX + d) , n	Ld (IY + d) , n
Ld A , (BC)	Ld (BC), A
Ld A , (DE)	Ld (DE) ,A
Ld A , (nn)	Ld (nn) , A

مثال: بار گذار ۸ بیتی

```
2000  LD E, 64h
2002  LD C, 100
2004  LD A ,-1
2006  LD L , 33h
```

- 2000 1E
- 2001 64
- 2002 0E
- 2003 64
- 2004 3E
- 2005 FF
- 2006 2E
- 2007 33

16 – bit load group گروه انتقال ۱۶ بیتی

این گروه وظیفه خواندن و نوشتن داده ها از حافظه را دارند بااین تفاوت که داده ها در اینجا ۱۶ بیتی هستند.

LD dd , nn	Ld Ix , nn	Ld IY , nn	
Ld HL , (nn)	Ld dd , (nn)	Ld Ix , (nn)	Ld IY , (nn)
Ld (nn) , HL	Ld (nn) , dd	Ld (nn) , Ix	Ld (nn) , IY
Ld SP , HL	Ld Sp , IX	Ld SP , IY	
PUSH qq	PUSH IX	PUSH IY	
POP qq	POP IX	POP IY	

مثال : بارگذاری ۱۶ بیتی
برنامه های زیر چه کاری انجام میدهند.

- **LD IX,1000h**
- **LD IY,1200h**
- **LD A, (IX)**
- **LD (IY), A**

LD BC ,1000h
LD DE , 1200h
LD A , (BC)
LD (DE) , A

- **LD HL ,1000h**
- **LD A, (HL)**
- **LD H , 12h**
- **LD (HL), A**
- **----------**

دستورات تعویض و انتقال و جستجو :

دستورات تعویض:

این دستورات محتوی ثبات داخلی را با یکدیگر تعویض میکند.

این دستورات تاثیری بروی پرچم ندارند.

EX DE , HL	DE ⟷ HL			
EX AF , AF’	AF ⟷ AF’			
EXX	BC ⟷ BC’	DE ⟷ DE’	HL ⟷ HL’	
EX (SP) , HL	H ⟷ (sp + 1)	L ⟷ (sp)		
EX (SP) , IX	IXH ⟷ (sp +1)	IXL ⟷ (sp)		
EX (SP) , IY	IYH ⟷ (sp +1)	IYL ⟷ (sp)		

دستورات تعویض و انتقال و جستجو ادامه:

- دستورات انتقال :
- انتقال یک سری داده از یک قسمت حافظه به محل دیگر حافظه با استفاده از ثبات های زیر:
- (HL) : اشاره دارد به مبدا حافظه (آدرس جایی که داده ها هم اکنون وجود دارد)
- (DE) : اشاره دارد به مقصد حافظه (آدرس جایی که داده ها باید انتقال پیدا کنند)
- (BC) : اشاره دارد به تعداد داده هایی که باید انتقال یابند.
- بعد از اینکه برنامه نویس ثبات های بالا را مقدار دهی نمود هر یک از چهار دستور انتقال را میتواند بکار گیرد.
- LDIR : (LOAD INCREMENT AND REPEAT)
- این دستور تعمیم یافته دستور LDI است و برای انتقال گروهی از داده ها از یک محل حافظه به محل دیگر استفاده میشود.

  در این دستور عمل مشابه LOAD و INCREMENT تکرار میگردد تا زمانیکه محتوی شمارنده BC به صفر برسد.
- دستورات LDD و LDDR مانند دو دستور قبل میماند فقط بعد از هر انتقال جفت ثبات های HL و DE یک واحد کاهش میابند.

دستورات تعویض و انتقال و جستجو ادامه:

دستورات جستجو :

این دستورات برای جستجو و پیدا کردن هر کاراکتر ۸ بیتی بکار میرود.

دستور CPIR (COMPARE , INCREMENT AND REPEAT)

این دستور محتوی ثبات A را با محتوی محلی از حافظه که آدرسش در HL میباشد مقایسه میکند اگر یکی نبودند (HL) را یکی اضافه میکند و (BC) را یکی کم میکند و محتوی آدرس بعدی را با A مقایسه میکند . این عمل تا جایی ادامه میابد که یا بایت مورد نظر پیدا شود یا شمارنده (BC) به صفر برسد.

مثال کاربردی دستور LDIR

برنامه زیر ۲۵۶ بایت داده را از یک محل حافظه به ترتیب به محل دیگری از حافظه انتقال میدهد.

```
LD HL , 1000h
LD DE , 2000h
LD BC , 0100h
LDIR
```

مثال : کاربرد دستور CPIR

```
LD HL ,1200h
LD A, 23h
LD BC, 50h
CPIR
```

برنامه زیر چه کاری انجام میدهد.

```
LD HL , 1000h
LD DE , 1200h
LD A , (HL)
EX DE , HL
LD (HL) , A
```

دستورات ریاضی و منطقی ۸ بیتی

- دستورات ریاضی و منطقی ۸ بیتی بصورت باینری بین داده واقع در مبدا و داده موجود در انباره انجام میشود و نتیجه در انباره قرار میگیرد.
- تنها در دستور CP s (مقایسه) نتیجه به انباره نمی روید.
- تمامی دستورات این گروه بر حسب نوع عمل برروی پرچم ها تاثیر میگذارد.

ADD A , r

ADD A , n

ADD A , (HL)

ADD A , (IX + d)

ADD A , (IY + d)

==========

ADC A , s

SUB s

SBC A , s

دستورات ریاضی و منطقی ۸ بیتی ادامه :

دستورات منطقی :

AND s

Or s

XOR s

دستور مقایسه :

CP s

INC r **INC (HL)** **INC (IX + d)** **INC (IY + d)**

DEC S

مثال : جمع ۸ بیتی

```
LD     A,15h          LD IX, 0700h          LD A,C
ADD  A, 7             LD A, (IX + 0)        ADD A, B
LD    HL , 1A01h      ADD A, (IX+01)        LD D, A

LD   (HL) , A         LD (IX + 2 ), A
```

محتوی ثبات A و ثبات F را درخاتمه بیابید.

```
LD A, 53h             LD A, 23h
LD B, 3Ah             LD B , B8h
ADD A,B               ADD A, B
SUB 8Ch               SUB DBh
DEC A                 DEC A
```

دستورات ریاضی ۱۶ بیتی :

ADD HL ,ss **SS= BC ,DE , HL , SP**

ADC HL , ss

SBC HL , ss

ADD IX , pp **pp = BC , DE , IX , SP**

ADD IY , rr

INC SS

INC IX

INC IY

DEC ss DEC IX DEC IY

مثال : جمع ۱۶ بیتی و ۳۲ بیتی

جمع ۳۲ بیتی

```
LD BC , 0101h
LD DE , 0FFFh
LD  HL , 0200h
LD IY , 3303h
ADD IY, DE
ADC HL ,BC
```

جمع ۱۶ بیتی

```
LD BC , 0110h
LD HL , 01FFh
ADD HL , BC
----------------------
LD HL ,(2000H)
LD BC ,(2002H)
ADD HL , BC
LD (2004H) , HL
```

عمل ضرب با استفاده از جمع تکراری
(دستورات جمع ۱۶ بیتی)

- LD HL ,10
- ADD HL ,HL 10 * 2
- ADD HL , HL 10 * 4
- ADD HL , HL 10 * 8

HL * 40 =

(HL * 32) + (HL * 8)

ADD HL ,HL

ADD HL, HL

ADD HL ,HL

LD D,H

LD E, L

ADD HL ,HL

ADD HL ,HL

ADD HL ,DE

HL * 15=
(HL * 16) – (HL * 1)

LD D, H

LD E, L

ADD HL ,HL

ADD HL ,HL

ADD HL ,HL

ADD HL ,HL

SCF

CCF

SBC HL ,DE

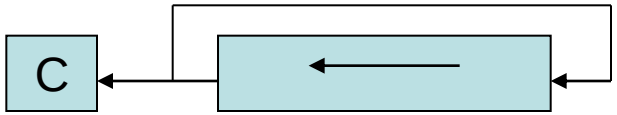
ROTATE AND SHIFT GROUP

قابلیت اصلی Z80 در این است که قادر به به ROTATE و SHIFT دادن داده در A و یا هر ثبا ت همه منظوره و در هر محلی از حافظه میباشد.

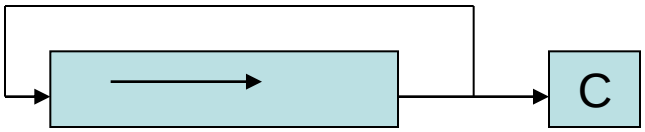
دستورات SHIFT ریاضی و منطقی در بسیاری از موارد از جمله عملیات ضرب و تقسیم اعداد صحیح کاربرد دارد.

دستورالعمل های RLC و RRC محتوی یک ثبا ت ۸ بیتی یا مکانی از حافظه را یک بیت به چپ یا براست میچرخانند و هر انچه که در حلقه چرخانده میشود در پرچم بیت نقلی نیز قرار میگیرد.

RLC:



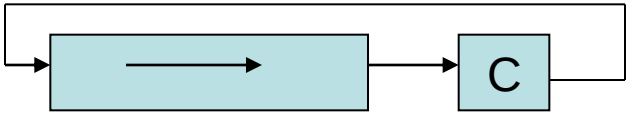
RRC :



دستورالعمل های RL و RR ثبا ت یا مکانی از حافظه را به همراه پرچم بیت نقلی به صورت یک ثبات ۹ بیتی بکار میگیرند.

RL :

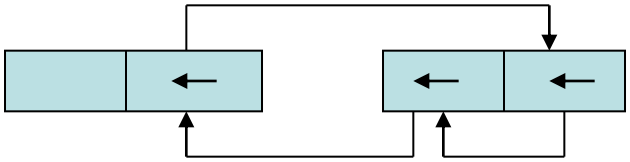
RR:



دستورالعمل های RLD و RRD اطلاعات را چهار بیت بین نیم بایت طرف راست انباره و مکانی از حافظه که بوسیله HL آدرس دهی میشوند می چرخاند.

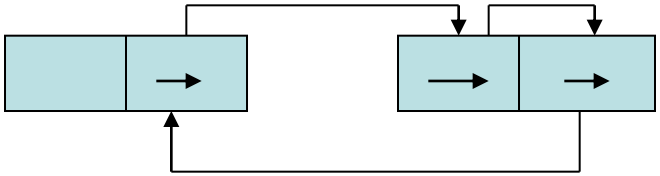
RLD:

A



RRD:

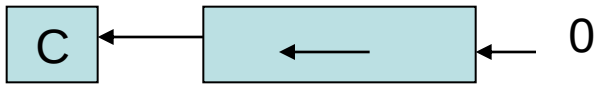
A



دستورالعمل های SRL و SLA برای انتقال محتوی یک ثبات یا مکانی از حافظه به راست یا چپ بکار میروند.

اگر از دستور SLA استفاده شود یک 0 به داخل آخرین بیت طرف راست و اگر از دستور SRL استفاده گردد یک 0 به داخل آخرین بیت طرف چپ وارد میشود.

SLA:



SRL :

دستورالعمل SRA یک دستور تغییر مکان به راست است که بیت علامت را از طرف راست نسخه برداری میکند.

SRA:

دستورات همه منظوره ریاضی و کنترل Z80

DAA **decimal adjust accumulator**

مثال :

```
LD A ,11
ADD A, 19
DAA
```

CPL **Complement Accumulator**

NEG **Tow’s complement accumulator**

CCF **Complement carry flag**

SCF **Set carry flag**

Nop **No operation**

HALT **Cpu halted**

گروه Set و Reset و Test

Z80 با یک دستور قابلیت set و reset و آزمایش کردن هریک از بیتهای موجود در انباره و یا هریک از ثباتهای همه منظوره و یا هر محلی از حافظه را دارد.

گروه آزمایش :

BIT b, r BIT b , (HL) BIT b, (IX + d) BIT b, (IY + d)

گروه SET :

SET b, r SET b , (HL) SET b , (IX + d) SET b, (IY + d)

گروه Reset :

RES b , S s= (HL) , (IX + d) , (IY + d)

Jump Group گروه پرش

دستور پرش غیر شرطی :

Jp nn nn= 16 bit address

دستور پرش شرطی :

JP cc , nn

condition code

NZ not zero

Z zero

NC not carry

C carry

PO parity odd

PE parity Even

P sign positive

M sign negative

دستورات پرش ادامه :

دستور پرش نسبی غیر شرطی:

JR e pc ← pc + e

دستورات پرش نسبی شرطی:

JR C ,e **JR NC, e** **JR Z , e** **JR NZ , e**

دستورات پرش غیر شرطی (غیر مستقیم):

JP (HL) **JP(IX)** **JP(IY)**

دستورات پرش ادامه :

دستور **DJNZ , e**

این دستور دو بایتی جهت کنترل **loop** یک برنامه استفاده میشود.

این دستور از نوع نسبی بوده ثبات **B** را **decrement** کرده و پرش زمانی صورت میگیرد که اگر محتویات ثبات **B** صفر نشده باشد.

مثال: علامت سوال چه عددی باشد تا برنامه به آدرس ستاره پرش کند.

آدرس	دستورالعمل
2000	LD B , 7
* * * *	----------
-------	----------
-------	----------
200A	DJNZ, ?
-------	----------

مثال :

```
        LD B , 6
Loop:   ADD HL ,HL
        DJNZ , Loop
ENDP    JP  ENDP
```

A َ را آزمایش کن اگر مثبت بود به آدرس 3000 پرش کن.

```
LD A, (0100H)
OR A
JP P , 3000H
```

برنامه زیر چه کاری انجام میدهد.

```
CP 50H
JP NC , 3000H
```

اگر A 50 به آدرس 3000 پرش کن

دستور مقایسه (نکته) :

دستور	CP s	A-S
اگر	A=S باشد	C=0 و Z= 1
آگر	A> S باشد	C=0 و Z= 0
اگر	A< S باشد	C=1 و Z=0

مثال: عدد موجود در آدرس 2000h از حافظه را از عدد موجود در آدرس 2001h کم کرده و در آدرس 2002h قرار میدهد. (از روش مکمل دو انجام دهید)

```
1000   LD A , (2000H)
1003   LD B , A
1004   LD A, (2001H)
1007   CPL
1008   INC A
1009   ADD A,B
100A   LD A , (2002H)
100B   HALT
```

چند عدد که تعداد آن در خانه 2000H قرار دارد را بایکدیگر جمع و حاصل ۸بیتی آن را در همان خانه قرار دهید.

```
        LD HL ,2000H
        LD B , (HL)
        XOR A
LOOP:   INC HL
        ADD A, (HL)
        DEC B
        JP NZ , LOOP
        LD (2002) , A
        HALT
```

دو عدد موجود در خانه های 2000H و 2001H را با یکد یگر مقایسه و عدد بزرگتر را در خانه 2002H ذخیره کنید.

```
        LD A, (2001H)
        LD B , A
        LD A , (2000H)
        CP B
        JP NC , EXIT
        LD A , B
EXIT :  LD (2002H) , A
        HALT
```

دستورات CALL و RETURN

دستور CALL حالت خاصی از دستورات پرش است . جایی که آدرس بعدی دستور CALL در داخل STACK ذخیره میشود.

دستور RETURN برعکس دستور CALL عمل میکند . زیرا که داده ایی که در بالای STACK قراردارد به داخل PC میرود (P0P) تا عمل برگشت صورت گیرد.

CALL nn

(SP-1) ← PCH

(SP-2) ← PCL

PC ← nn

SP به SP-2 اشاره میکند.

RET

PCL ← (SP)

PCH ← (SP + 1)

SP به SP+2 اشاره میکند.

دستورات ورودی و خروجی :

Z80 دارای مجموعه وسیعی از دستورات ورودی و خروجی میباشد.
نحوه آدرس دهی وسایل ورودی و خروجی یا بصورت فوری است و یا بصورت ثباتی غیر مستقیم که از ثبا ت C استفاده میشود. از ثبات C برای مشخص کردن آدرس ۸ بیتی پورت استفاده میشود.

دستورات ورودی :

IN A , (n)
IN r , (C)

دستورات خروجی :

OUT (n) , A
OUT (C) , r

IN A, (n)
Operation: A⟵ (n)
Op Code: IN
Operands: A, (n)

Description: The operand n is placed on the bottom half (A0 through A7) of the address bus to select the I/O device at one of 256 possible ports. The contents of the Accumulator also appear on the top half (A8 through A15) of the address bus at this time. Then one byte from the selected port is placed on the data bus and written to the Accumulator (register A) in the CPU.

M Cycles	T States	4 MHz LT.
3	11 (4, 3, 4)	2.75

Condition Bits Affected: None

Example: If the contents of the Accumulator are 23H, and byte 7BH is available at the peripheral device mapped to I/O port address 01H. At execution of INA, (01H) the Accumulator contains 7BH.

مثال: قطعه برنامه ایی بنویسید که محتوی پورت شماره 05H را خوانده ودر صورت صفر یا مثبت بودن محتوی به آدرس 3000H پرش کند

```
IN A, (05H)
CP 00H
JP P , 3000H
```

برنامه ایی بنویسد که اطلاعات موجود در پورت شماره 00H را خوانده و تعداد یکهای موجود در آن را چک نماید. اگر این تعداد زوج باشد کلیه چراغ های متصل به پورت خروجی 10H را روشن ودر غیر این صورت کلیه چراغ ها را خاموش نماید.

```
START:    IN A , (00H)
          AND A
          JP PE , PEVN
          LD A, 00H
OUTPUT:   OUT  (10H), A
          JR  START
PEVN :    LD A, FFH
          JR  OUTPUT
```

در بخشی از یک برنامه میخواهیم رقم یا وزن کمتر یک عدد دو رقمی BCD که در انباره ذخیره گردیده است را با عدد 8 مقایسه کنیم و در صورتیکه رقم BCD مورد نظر مساوی 8 باشد به آدرس ALI پرش نماید.

```
AND   0FH
CP    08H
JP Z  ALI
```

برای جدا سازی ارقام (MASKING) از دستور AND استفاده میکنیم

چهار بیت وزن کمتر را جدا کن	AND 0F
مقدار 0AH را از نتیجه کم کن	SUB 0AH
در صورتیکه محتوی انباره بیشتر	JP P , ALI
یا مساوی عد 0AH است به آدرس ALI	
پرش کن	

برنامه ایی بنویسید که یک عدد ۳۲ بیتی موجود در حافظه از آدرس 1A00H را با عدد ۳۲ بیتی دیگر که از آدرس 1A04 H در حافظه قرار دارد را جمع وحاصل ۳۲ بیتی آنرا در حافظه آز آدرس 1A08 H ذخیره نماید.

```
LD HL , (1A00H)
LD DE , (1A04H)
ADD HL ,DE
LD (1A08H) , HL
LD HL , (1A02H)
LD DE , (1A06H)
ADC HL , DE
LD (1A0AH) , HL
HALT
```

برنامه ایی بنویسید که محتویات محل های حافظه 187AH تا 1889H را پاک نماید.

1800	LD C , 20H	0E 20
1802	LD HL , 187AH	21 7A 18
1805	XOR A	AF
1806	LD (HL) , A	77
1807	INC HL	34
1808	DEC C	0D
1809	JPNZ ,1806H	C2 06 18
180C	HALT	76

برنامه ایی بنویسید که محتویات محل های حافظه 1900H و 1901H را با محتویات ۱۶ بیتی جفت ثبات DE جمع و حاصل را درجفت ثبات DE قرار دهد.

LD A , (1900H)	7E 00 19
ADD A ,E	83
LD E , A	5F
LD A, (1901H)	7E 01 19
ADC A , D	82
LD D , A	57
HALT	76

مدت زمان اجرای یک برنامه و ایجاد تاخیر زمانی (DELAY)

در بسیاری از موارد عملی کاربرد ریزپردازندها لازم است که اجرای بخش های مختلف یک برنامه با فواصل زمانی مشخص از هم جدا شوند.
در این فواصل زمانی عملا" ریزپردازنده باید بیکار بماند. برای اینکار از زیر برنامه هایی استفاده میشود که در کار ریزپردازنده تاخیر ایجاد میکند . بزبانی دیگر ریزپردازنده به کاری مشغول میشود که نتیجه آن با بیکار بودن تفاوت ندارد.
قطعه برنامه زیر را در نظر بگیرید.

```
        LD DE , nn
LOOP: DEC DE
        LD A , D
        OR E
        JPNZ , LOOP
```

مدت زمان اجرای برنامه نشان داده شده از رابطه زیر بدست میاید.

$$T = 10T + nn\,(\,6T+4T+\,4T+10T) = 10T + nn(24T)$$

اگر فرض کنیم فرکانس برابر با ۵/۲ مگا هرتز باشد $T=0.4\ \mu s$ میشود.

اگر هدف اجرای تاخیر 0.1 ثانیه باشد با قراردادن مقادیر T و T کل در رابطه بالا مقدار nn مشخص میشود .

$$nn= 10416.25 = 10416 = 28B0H$$

و با قرار دادن مقدار هگزا دسیمال nn در برنامه قبل تاخیر 0.1 ثانیه ایجاد میشود.

با استفاده از زیر برنامه زیر میتوان تاخیر ها ی 0.1 ثانیه به بالا را با دقت 0.1 ثانیه ایجاد نمود. (F = 2.5 mhz)

```
Delay : LD DE , 2B80H
LOOP: DEC DE
        LD A , D
        OR E
        JPNZ , LOOP
         RET
```