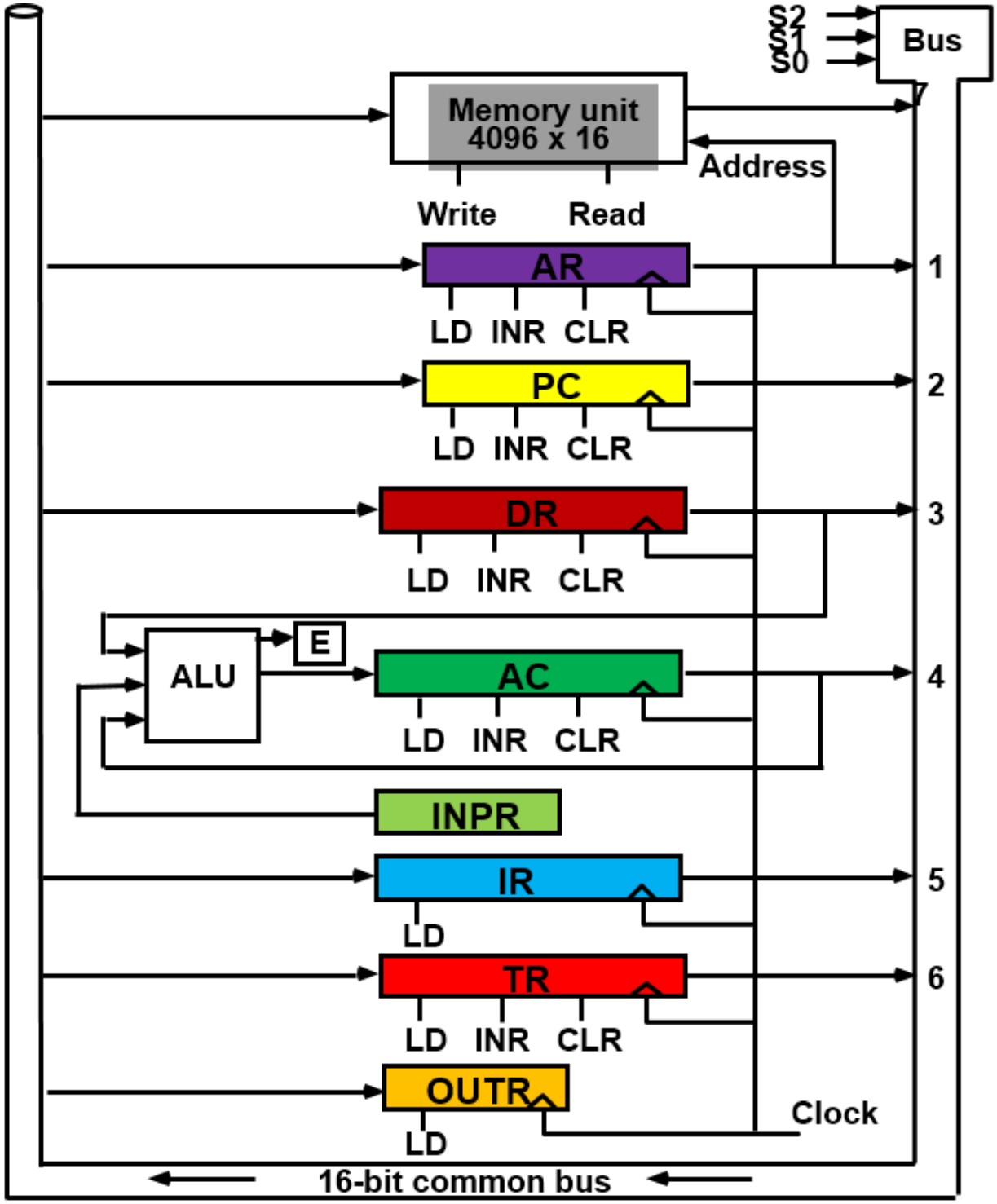
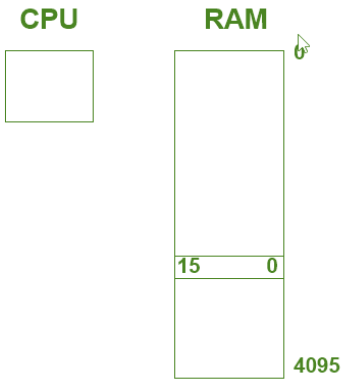
خب اول اینکه باید بدونیم هر کامپیوتر باید یک RAM داشته باشه تا بتونیم دستورات و داده هامونو داخلش ذخیره کنیم و ثبات هایی نیز داریم تا بتونیم از طریق آن ها کارامون رو انجام بدیم.

شکل زیر نمای اصلی کامپیوتر پایه را نشان می دهد که در ادامه به توضیحاتی راجب آن می پردازیم.



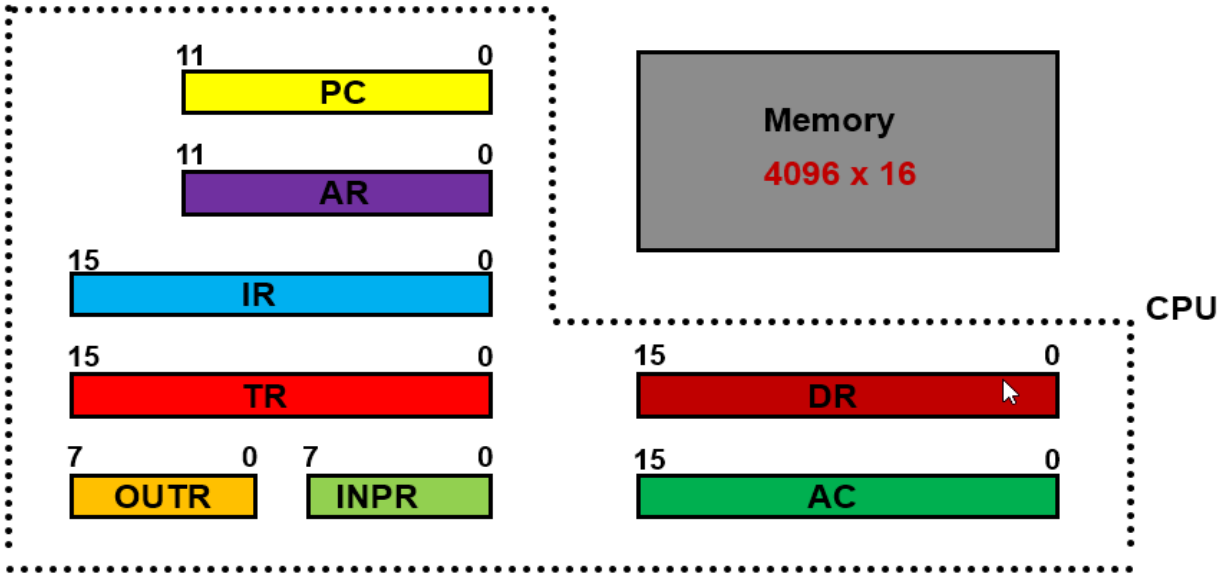
خب از RAM شروع کنیم , اگر دقت کنین می بینین که روی RAM نو شته شده ۴۰۹۶ * ۱۶ که اگر ما RAMرا مانند یک ماتریس دو بعدی در نظر بگیریم این اعداد به این معنی هســـتند که این ماتریس دارای ۴۰۹۶ سطر و هر سطر دارای ۱۶ ستون می باشد.



حالا باید ارتباط RAM با ثبات هارو برقرار کنیم , ابتدا باید بدونیم که برای خوندن یا نوشـــتن در RAM ابتدا باید آدرس خونه ای که میخوایم عمل خوندن یا نوشتن رو انجام بدیم رو به RAM بدیم.

وظیفه آدرس دادن به RAM فقط با AR اســـت پس اگر میخوایم به RAMآدرس بدیم که از اون خو نه چیزی بخونیم یا در اون خو نه چیزی

Registers in the Basic Computer



List of BC Registers

DR	16	Data Register	Holds memory operand
AR	12	Address Register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction Register	Holds instruction code
PC	12	Program Counter	Holds address of instruction
TR	16	Temporary Register	Holds temporary data
INPR	8	Input Register	Holds input character
OUTR	8	Output Register	Holds output character

بنویسیم , باید اون آدرس رو بدیم به AR و بعد AR اون آدرس رو به RAM بده و بعد ما با استفاده از handleهای write و read تعیین کنیم که میخوایم چه عملی رو انجام بدیم.

ثبات PC همیشه داره بیان می کنه که آدرس دستور بعدی ای که باید اجرا بشه چیه یعنی اگر ما داریم دستور ۱۰ رو اجرا می کنیم pc , داره عدد ۱۱ رو نشون می ده(البته اگر پرش در کار نباشه که در ادامه گفته می شه).

خب این دو ثبات فقط با آدرس دهی حافظه سر و کار دارن که این ثبات ها در این جا ۱۲ بیتی هستند.چرا !؟

بقیه ثبات ها با نام های TR , IR , AC , DR ثبات های ۱۶ بیتی ما هستند.چرا !؟

دلیل این تعداد بیت ها در این ثبات ها همین عدد داخل RAM یعنی ۴۰۹۶ *۱۶ می باشد.

یعنی ۴۰۹۶ سطر که در هر سطر ۱۶ ستون داریم.

۴۰۹۶ یعنی ۲ به توان ۱۲ , و همان طور که می دونین برای آدرس دادن به ۲ به توان i سطر نیاز به i بیت می باشد پس اینجا برای آدرس دادن به هر سطر از RAM ما نیاز به ۱۲ بیت داریم که این دلیل ۱۲ بیتی بودن ثبات های آدرس می باشد.

خب پس از اون جایی که ما داریم به هر سطر جدا آدرس می دیم نه به هرخانه جدا , ما نیاز داریم که بتونیم یک سطر از RAM را بخوینم و در ثباتی ذخیره کنیم و یا در یک سطر از RAM چیزی بنویسیم که هر سطر ۱۶ بیت دارد پس ثبات هایی که با داده های RAM سر و کار دارن و می خوان مقادیر آن را بخوانند یا در آن بنویسند باید اینجا ۱۶ بیتی باشند.

ثبات های INPUT , OUTPUT اینجا ۸ بیتی هستند که در ادامه به آن ها می پردازیم.

قبل از اینکه به چگونگی اجرای دستورات در این کامپیوتر بپردازیم می خوام با چند نکته در مورد نحوه ارتباط این ثبات ها و RAM آشناتون کنم.

اول اینکه همونطور که می بینید ثبات ها و RAM از طریق باس یا همون گذرگاه داده به هم وصل هستند به طوری که هر کدوم از آن ها می تونن مقدار شون رو بر روی باس قرار داده و ثبات بعدی)یا (RAM آن را دریافت کند.

ولی نکته ا صلی ای که اینجا باید توجه کرد اینه که ورودی ثبات AC به این باس و صل نیست و این ثبات فقط از طریق مدار محاسبه منطق می تونه مقدار بگیره ولی میتونه مقدارش رو بذاره رو باس و به هرکسی بده.

یادآوری : باید بدونیم که باس چیه و چطوری کار میکنه که من در حد لازم برای این فصل توضیح میدم . فکر کنین تمام این ثبات ها به یک سیم به هم وصل هستند که برای اینکه تداخلی پیش نیاد فقط یکی از اون ها در هر لحظه حق داره بر روی سیم مقدارشو قرار بده . حالا این مقدار روی سیم در دسترس همه ثبات ها هست و هرکی بخواد میتونه اون رو برداره . حالا چطوری بر میداره!؟ باید بدونیم هر ثبات ۳ تا handleیا همون کنترل کننده به اسم های load , inc , clear داره که

هر کدوم از این ها یک بیت می گیرن که اگر ۰ باشه غیر فعال می شن و اگر ۱ باشه فعال می شن . حالا اگر clear فعال باشه هر مقداری داخل ثبات باشه به ۰ تغییر میکنه . اگر INC فعال باشه هر مقداری در ثبات باشه یکی بهش اضافه می شه . نکته اصلی در load وجود داره . خیلی وقتا ممکنه مقداری برای ورود به ثبات وجود دا شته با شه اما اون ثبات این مقدار رو نمیگیره و وارد خودش نمیکنه که ذخیره بشه . هروقت load فعال باشه اون مقدار وارد ثبات میشه . یعنی load اجازه ورود مقادیر به ثبات رو میده . اینجا هم درسته تمام ثبات ها به باس متصل هستند اما تنها ثبات هایی مقدار روی باس رو میگیرند که load آن ها فعال بشه.

حالا تا حدودی با ارتباطات کلی آشنا شدید حالا به نکات مهمی که معمولا سوالات از آن ها مطرح می شود میپردازیم.

برای درک بیشتر نکات رو در قالب مثالی مطرح میکنم.

آیا میشود مستقیما مقدار RAM را در ثبات DR قرار داد !؟

بله RAM . مقدار خونه ی مورد نظر (M[AR]) را بر روی باس قرار داده و ثبات DR آن را از روی باس بر میدارد.

آیا میشود مستقیما مقدار RAM را در ثبات AC قرار داد !؟

شاید پیش خودتون بگین مثل مثال قبل RAM مقدارشو میذاره رو باس و AC بر میداره اما همونطور که قبلا هم گفتم AC نمیتونه از باس مقدار برداره و فقط از مدار محاسبه منطق میتونه مقدار بگیره.

مدار محاسبه منطق هم فقط از AC , DR , input مقدار میگیره input . که بحثش جداست و از بیرون یعنی از کاربر مقدار میگیره پس فقط می مونه . DR یعنی برای اینکه یک مقدار رو به AC برسونیم تنها راهمون اینه که اون مقدار رو بدیم به DR و در کلاک بعدی اون مقدار رو بدیم به. AC

آیا میشود مقدار DR رو بدیم به TR و همزمان مقدار AC رو بدیم به! RAM ؟

خب طبق نکته ای که در مورد باس گفته شد این عمل میسر نیست چون هر دو عمل باید طرف اول مقدارش رو بر روی باس قرار بده و طرف دوم مقدار رو از روی باس برداره که این عمل در یک کلاک یعنی در یک لحظه به طور همزمان میسر نیست و روی باس تداخل پیش میاد.

آیا می شود مقدار PC را در AR نوشت و همزمان مقدار DR را در AC نوشت !؟
جواب بله است . چون اگر DR بخواد مقدار شو در AC قرار بده نیازی به باس نداره و به طور مستقیم از طریق مدار محاسبه منطق به AC مقدار میده پس باس اشغال نمیشه و میشه یک عمل همزمان با آن نیز انجام داد که اینجا PC مقدارشو روی باس میذاره و IR نیز برمیداره.

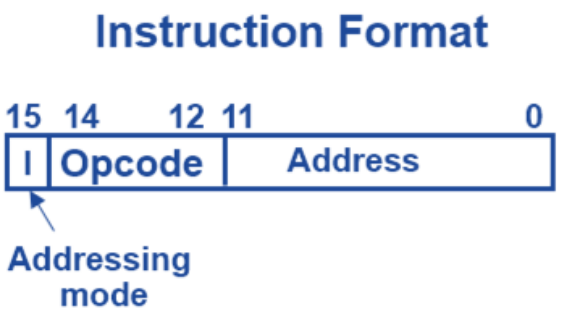
اگر بخوایم DR رو بریزیم در AC و به طور همزمان AC رو نیز در DR قرار بدیم چه اتفاقی میافته !؟
خب اینجا بحث همزمانی پیش میاد . باید بدونین وقتی میگیم همزمان یعنی کسی منتظر نیست یه مقدار جدید بگیره و بعد کارشو انجام بده بلکه همه با مقداری که الان دارن کارشنو انجام میدن. مثلا اینجا DR مقدار شو از طریق مدار محاسبه منطق میده به AC و AC نیز مقدار شو روی باس میذاره و DR از روی باس برمیداره . اینجا دیگه AC منتظر نمی مونه که از DR مقدار جدید بگیره و بعد ارسال کنه بلکه همون مقدار قبلیش رو روی باس قرار میده . یعنی همزمان با اینکه مقدار DR داره به AC میرسه , مقدار قبلی AC نیز در راه رسیدن به DR است و اینگونه جای مقادیر این دو ثبات عوض می شه.

امیدوارم تونسته باشم با این مثال ها نکته های این ارتباط بین ثبات ها رو بهتون نشون داده باشم چون یکی از اصلی ترین نکات این فصل تسلط بر روی نحوه ارتباط بین اجزای این کامپیوتر میباشد که در قسمت کدنویسی از آن ها بیشتر استفاده می کنیم.
حالا که با نحوه ارتباط بیشتر آشنا شدیم بریم ببینیم هر ثبات به چه دلیلی گذاشته شده و چیکار میکنه.
همونطور که قبلا هم گفته شده AR یا همون address register وظیفه آدرس دادن به RAM رو داره و تنها ثباتی است که میتونه به RAM آدرس بده.
PCیا program counter وظیفه نشان دادن دستور بعدی برای اجرا رو داره که میگه وقتی دستور فعلی اجراش تموم شد دستور بعدی که باید اجرا بشه کیه که معمولا خط بعدی است.
ثبات های (data register) DR و (accumulator) AC برای انجام عملیات های محاسباتی با استفاده از مدار محاسبه و منطق استفاده میشن , پس برای انجام یک عملیات مثلا جمع دو مقدار باید از AC و DR استفاده بشه و ثبات دیگری نمیتونه کاری انجام بده پس اگر میخواین دو مقدار رو مثلا جمع کنین باید یکیش در AC باشه و دیگری در DR و سپس دستور جمع رو صادر کنین.
ثبات TR یا template register یک ثبات کمکی است در این کامپیوتر که اگر جایی نیاز دارین مقداری رو جایی نگه دارین و تو کلاک های بعدی دوباره ازش استفاده کنین به عنوان یک ثبات کمکی به شما کمک می کند که توصیه من به شما اینه که از این ثبات تا جایی که میتونین استفاده نکنین تا در فصل ۷ که این ثبات

رو ندارین مشکل نداشته باشین و به جای استفاده از TR میتونین با تکنیک های همزمانی که یک مثال هم از آن زده شده جای آن را پر کنید.

و اما ثبات IR یا instruction register که دستورات در آن تحلیل میشن و بعد اجرا میشن.

ابتدا باید بدونین که دستورات در RAM قرار دارن که به صورت باینری هستند که هر دستور در یک سطر میتونه قرار بگیره پس هر دستور ۱۶ بیت داره که در هر مرحله برای اینکه بفهمیم دستور چیه این عدد ۱۶ بیتی رو از RAMمیگیریم و در IR قرار می دیم و سپس از قوانین از پیش تعیین شده متوجه میشیم که معنی این عدد ۱۶ بیتی چیه و چی از ما میخواد تا براش اجرا کنیم



خب حالا می خوام در مورد IR و نحوه دیکد کردن دستور بگم که چطوری IR می فهمه دستور یا همون عدد ۱۶ بیتی چیه و چی میخواد .

بعد از اینکه دستور از RAM خونده میشه و به IR می رسه , این ثبات ۱۶ بیتی از سه قسمت تشکیل شده , قسمت اول یک بیت به اسم I می باشد که در ادامه به توضیح آن در شرایط مختلف می پردازیم.

قسمت دوم آن ۳ بیت برای opcode دستور است , این ۳ بیت بیانگر نوع عملیاتی است که باید انجام بدیم مانند جمع و ... ولی خب در یک حالتش معنی دیگری دارد که توضیح داده خواهد شد.

۱۲ بیت بعدی هم در بسیاری از حالات معنی آدرس خانه ای از حافظه را دارد و در حالت های دیگر بیانگر عملی است که باید صورت بپذیرد که این نیز بیان خواهد شد.

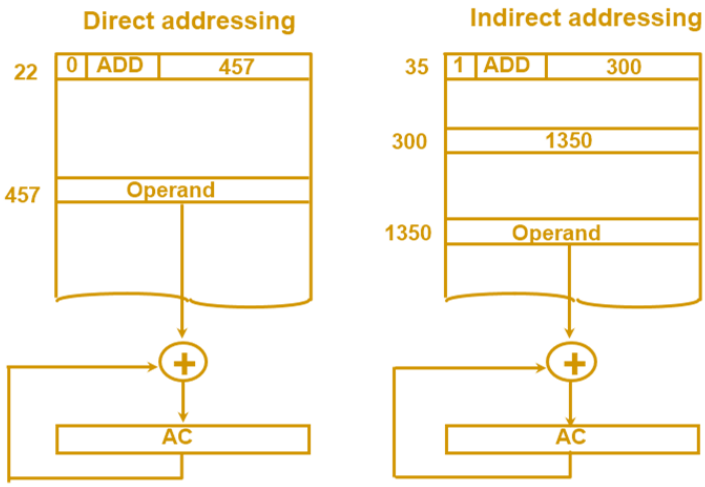
۳ بیت opcode به یک دیکدر داده میشه و ۸ خروجی داره که در هر لحظه یکیشون فعاله یا به زبان ساده تر با ۳ بیت میشه ۸ حالت ساخت , به این ۸ حالت از d0 تا d7 اسم میدیم.

حالا اگر از d0 تا d6 یکی فعال باشه) از d0 تا d7 فقط یکیشون میتونه فعال باشه (به این معناست که دستوری که در IR قرار داره یک دستور حافظه ای است بدین معنی که ۳ بیت opcode تعیین می کنه که چه عملی باید انجام بشه و ۱۲ بیت آخر بیانگر آدرسی در RAM است (سمت راست یعنی از ۰ تا ۱۱ یعنی باید در اصل بگیم ۱۲ بیت اول ولی چون از سمت چپ توضیح دادم به همین روش ادامه میدم) و تک بیت I مستقیم یا غیر مستقیم بودن آدرس را مشخص می کند.

اول بگم مستقیم و غیر مستقیم چیه و بعد توضیحاتم رو در مورد دستورات حافظه ای تکمیل کنم.

شما یک آدرس RAM میدین و میگین این آدرس مستقیم است , من باید برم به همون آدرسی که شما دادین و مقداری که داخلش قرار داره رو بهتون بدم مثلا شما میگین خونه ۱۰۰ رو به من بده من میرم خونه

۱۰۰ می بینم عدد داخلش ۱۴ اســـت , حالا من ۱۴ رو به شـــما تحویل می دم که این میشـــه آدرس دهی مستقیم.



حالا آدرس دهی غیر مســتقیم چیه : شــما میگین خونه ۱۰۰ رو غیر مستقیم میخواین , من میرم خونه ۱۰۰ می بینم درونش ۱۴ قرار داره , ولی چون غیر مستقیم خوا ستین دیگه ۱۴ رو به شما نمیدم , حالا من میرم خونه ۱۴ می بینم درونش ۳۰ قرار داره , من عدد ۳۰ رو به شما می دم . انگار آدرسی که شما دادین , مقدار داخلش میشـــه آدرس جدید و من میرم مقدار این آدرس جدید رو میارم.

خب حالا دوباره به صورت کلی میگم که دستورات حافظه ای چیکار میکنن : از طریق opcode می فهمیم که چیکار قراره بکنیم فرض کنیم عملیات جمع باشـــه , میریم ۱۲ بیت ســـمت راســـت IR رو برمی داریم و به عنوان آدرس ازش اســـتفاده می کنیم , حالا I رو چک میکنیم اگر مستقیم بود که هیچ اما اگر غیر مستقیم بود تبدیل به آدرس مستقیمش می کنیم به این حالت که میریم به اون آدرس و هرچی توش بود میشه آدرس جدید

مستقیم شده ی ما , حالا باید به آدرس مستقیم شده و مقدار شو به مقداری که از قبل در AC بود جمع کنیم و درون AC قرار بدیم.

بله درســـت دارین فکر میکنین , در مورد جمله آخر باید بگم که در این نوع دســـتورات یک طرف ماجرا همیشه AC قرار داره و جواب هم همیشه در AC قرار میگیره.

• **Basic Computer Instruction Format**

Memory-Reference Instructions (OP-code = 000 ~ 110)

Register-Reference Instructions (OP-code = 111, I = 0)

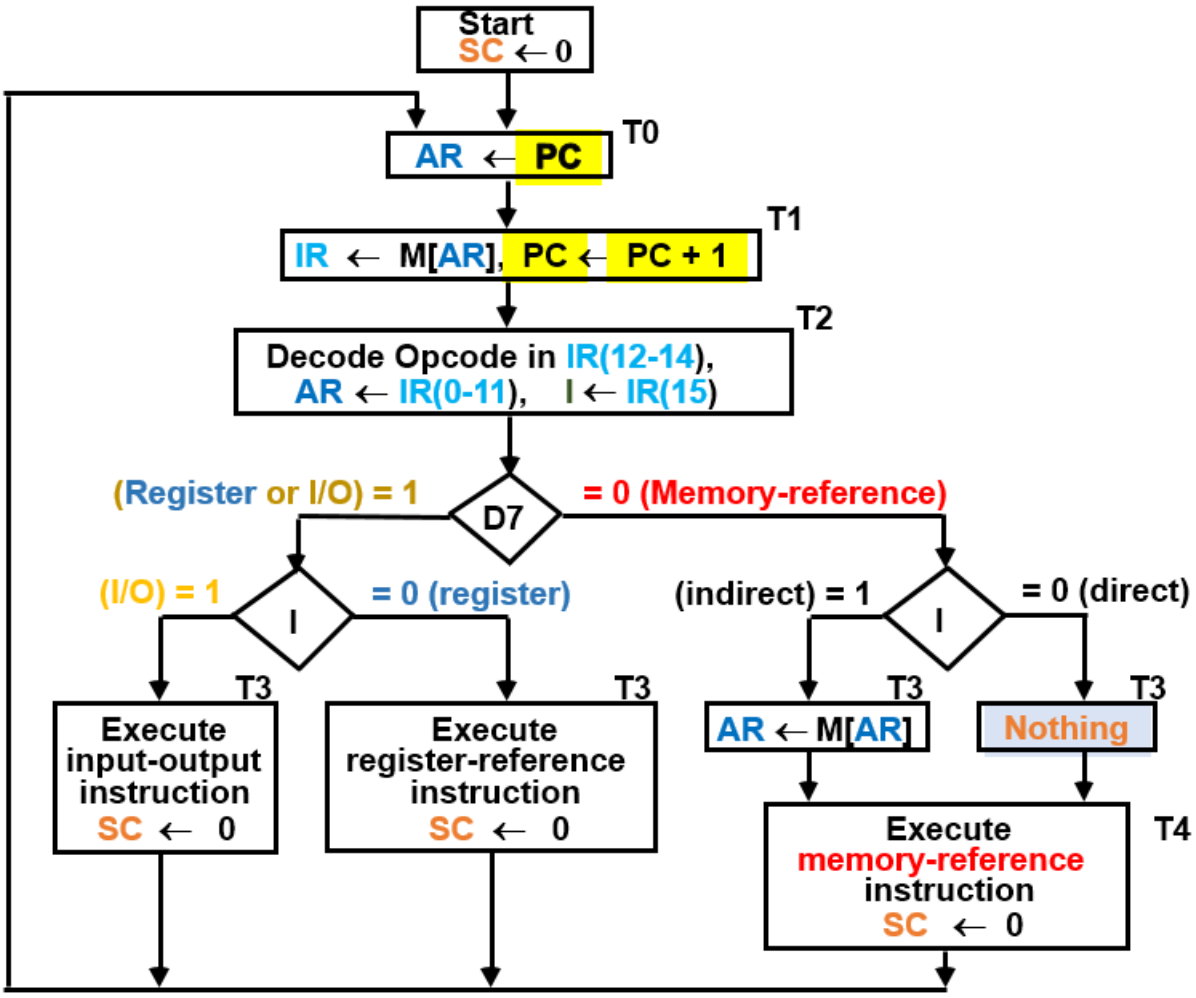
Input-Output Instructions (OP-code =111, I = 1)

حالت دوم زمانی پیش می آید که d7 فعال باشد)یعنی opcodeبرابر با ۱۱۱ باشد (و تک بیت I که در IR است ۰ باشد که در این حالت دستور از نوع دستورات ثباتی است . در این حالت ۴ بیت ثابت است و تغییری نمیکند پس سوال پیش می آید که چگونه انواع مختلف دستورات ثباتی از هم تفکیک می شوند . باید بگم که دیگه در این نوع دستورات چیزی به اسم آدرس نداریم یعنی ۱۲ بیت آزاد داریم که می تونیم با اون نوع دستور رو مشخص کنیم . اینجا طوری برنامه ریزی شده و قوانین حاکم است که از این ۱۲ بیت فقط یک بیت باید یک باشد که هر کدام از آن ها که یک باشد معنی یک عملیات ثباتی جدا را می دهد یعنی ۱۲ حالت (اگر چند بیت به اشتباه یک شد اولین بیت از سمت چپ مورد قبول واقع می شود).

حالت سوم نیز زمانی است که d7 فعال باشد و بیت I نیز یک باشد که این نوع دستورات ورودی خروجی است که این نوع دستورات نیز با استفاده از آن ۱۲ بیت کارهای مختلفی را انجام می دهند.

نمودار زیر مباحث گفته شده را به صورت فلوچارت نمایش داده است.

توضیحات بیشتر در مورد نمودار در زیر نمودار آمده است پس اگر جایی برایتان جدید بود نگران نشین.



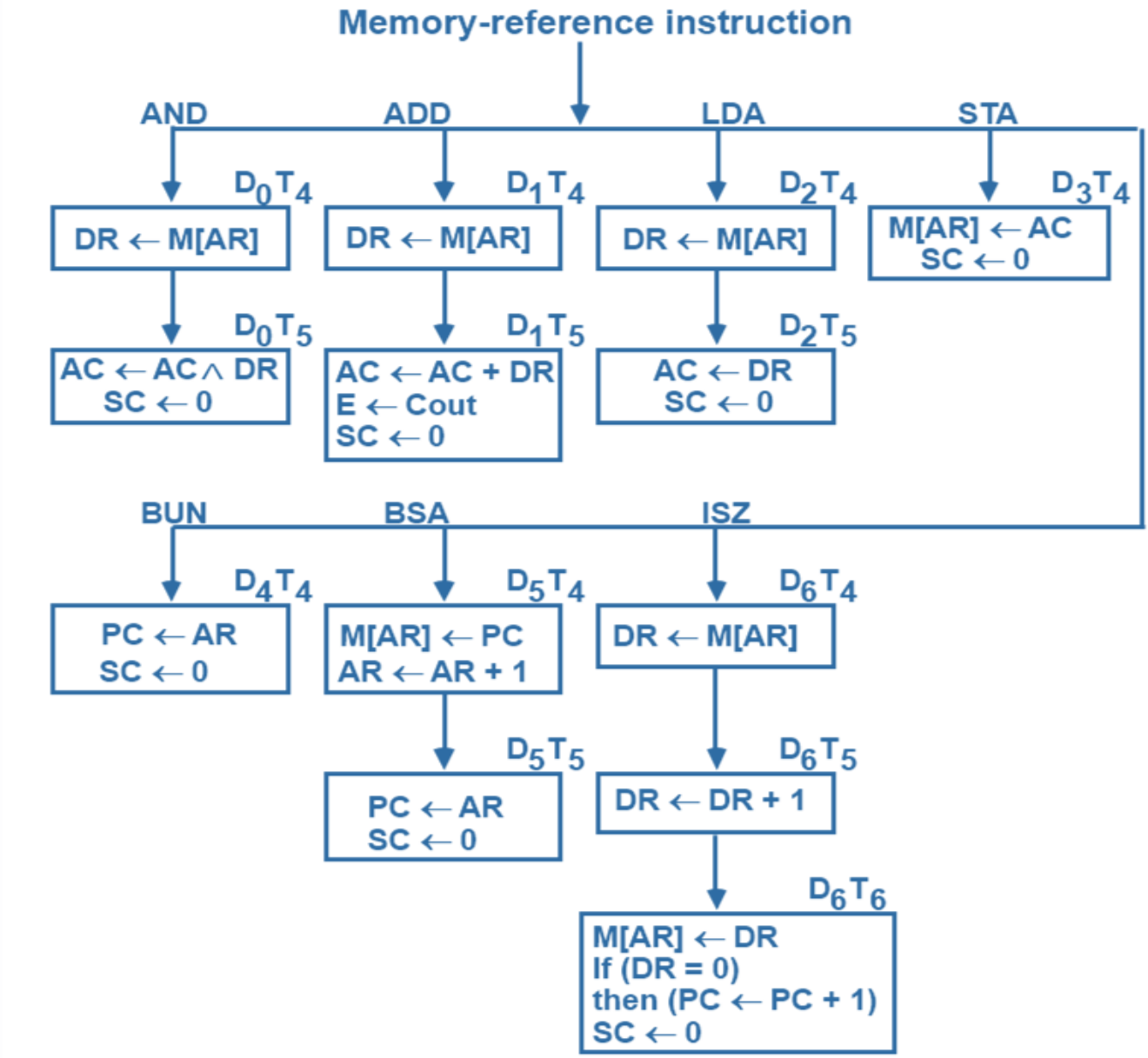
در ابتدا sc صفر شده برا شروع برنامه است که از sc برای کلاک زدن استفاده میشه یعنی اینکه الان در کدوم کلاک یا همون در کدوم لحظه قرار داریم.

PCوارد AR شـــده , گفته بودیم که PC میگه خط بعدی که باید اجرا بشـــه کیه , خط بعدی خب طبیعتا تو RAMقرار داره یعنی همه دسـتورات تو RAM هسـتند , پس ما باید بریم تو RAM خط بعدی یا همون خطی که PC آدرس میده رو بیاریم , اما گفتیم که فقط AR است که میتونه به RAM آدرس بده , برای اینکه خونه ای که PC داره نشون میده رو بخونیم باید ابتدا مقدار PC رو بدیم به. IR

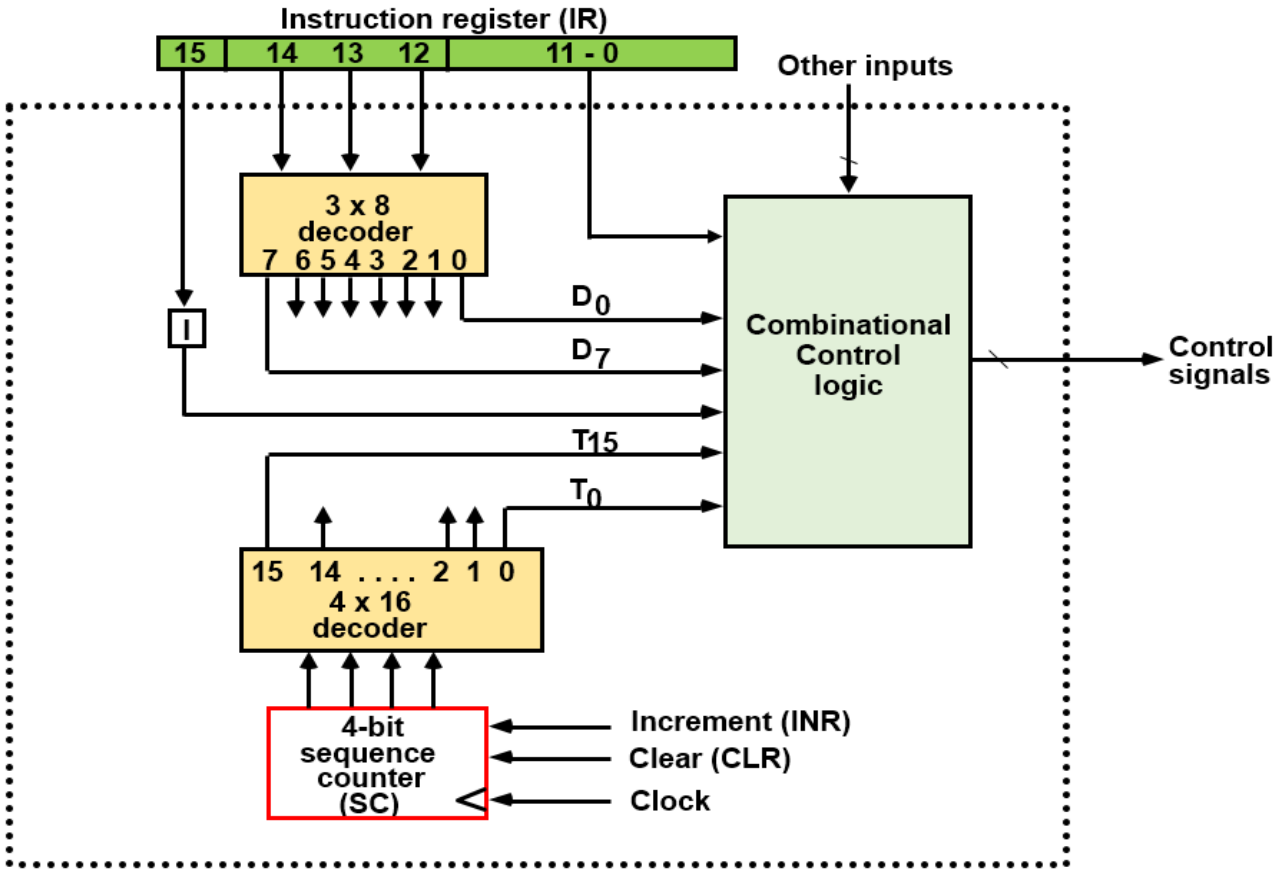
در خط سـوم می بینید که IR <– M[AR] نوشـته شـده یعنی خونه ای از RAM که AR داره نشـون میده , مقدارش رو وارد IR کن و همچنین PC یک عدد اضافه شده برای اینکه وظیفه PC اینه که بگه خط بعدی کیه و الان وظیفشو انجام داد حالا وقتشه به خط بعدی اشاره کنه یعنی یه خط دیگه بره جلو.

در خط بعد داره عملیات دیکد و انتقال آدرس ۱۲ بیتی به AR رو انجام می ده) البته ممکنه این ۱۲ بیت آدرس نباشه اونموقع دیگه بیخیال مقداری که وارد AR شده میشه. (

در قسمت لوزی داره چک میکنه که d7 فعال است یا حالت دیگه ای اتفاق افتاده) اگر d7 فعال نباشه یعنی یکی دیگه فعاله و یعنی دستور از نوع حافظه ای است. (



حالا اگر d7 فعال نباشه یعنی دستور حافظه ای باشه میره سمت راست اگر I برابر با ۰ باشه که کاری نداره اما اگر ۱ باشه آدرس رو مستقیم میکنه به این روش که خونه که داریم بهش الان آدرس میدیم یعنی M[AR]رو میاره و به عنوان آدرس جدید در نظر میگیره)یعنی میریزه در AR (, بعد از اطمینان از مستقیم بودن آدرس به انجام عملیات میپردازه.



حالا اگر d7 فعال باشه میاد سمت چپ , اگر Iبرابر با ۰ باشه میره راست و دستورات ثباتی اجرا میشن و

CLA	7800	Clear AC
CLE	7400	Clear E
CMA	7200	Complement AC
CME	7100	Complement E
CIR	7080	Circulate right AC and E
CIL	7040	Circulate left AC and E
INC	7020	Increment AC
SPA	7010	Skip next instr. if AC is positive
SNA	7008	Skip next instr. if AC is negative
SZA	7004	Skip next instr. if AC is zero
SZE	7002	Skip next instr. if E is zero
HLT	7001	Halt computer

اگر I برابر با ۱ باشه میره چپ و دستورات ورودی خروجی اجرا میشن.

INP	F800	Input character to AC
OUT	F400	Output character from AC
SKI	F200	Skip on input flag
SKO	F100	Skip on output flag
ION	F080	Interrupt on
IOF	F040	Interrupt off

حالا ببینیم این دستوراتی که میگیم چی هستن.

تصویر زیر انواع مختلف دستورات ثباتی را نمایش می دهد که در آن منظور از r یک کلاک خاص (نوشته شده در تصویر) و منظور از مثلا B11 یعنی بیت شماره ۱۱ یک شده است(مربوط به زمانی که ۱۲ بیت سمت راست دیگه آدرس نیستن و از طریق اونا می فهمیم دستور چیه).

یه نکته دیگه اینکه یک بیت E در کنار مدار محاسبه منطق است که برای شیفت در اینجا استفاده میشه.

پس از این به بعد اگر میخواین مثلا AC مقدارش صفر بشه کافیه در کد بنویسین CLA یا هرکاری که در این تصویر می بینین با کد مخفف آن قابل اجرا است.

شما مینویسین CLA امادر اصل کدی که وجود دارد اینگونه است:

۱۱۱	۱۰۰۰۰۰۰۰۰۰۰۰

دستوراتی که if دارن بعدش PC یکی زیاد شده برای فرار از اجرای خطی بعدی است مثلا میخواین اگر AC صفر نشده خط بعد یعنی خطی که PC میگه اجرا بشه ولی اگر صفر شده بره دو خط بعد یعنی خط PC+1 اجرا بشه . معمولا این کار در اجرای حلقه ها مورد استفاده قرار میگیره که مثلا تا وقتی که AC صفر نشده میخواین یک حلقه اجرا بشه و وقتی صفر شد از حلقه بیاد بیرون.

به مثال زیر توجه کنید:

```
AC= – 10
T : …
.
.
.
AC <- AC + 1
SZA
BUN T   // jump T
HLT
```

اینجا ده بار که به SZA میرسه و بعدش چون AC هنوز ۰ نشده خط بعد یعنی BUN یا همون دستور پرش به خط T اتفاق میافته و حلقه تکرار میشه اما دفعه آخر چون AC صفر شده به دو خط بعد میره یعنی HLT و در این کد برنامه تموم میشه HLT) برنامه رو تموم میکنه.)

خب دستورات ثباتی گفته شد حالا دستورات ورودی خروجی گفته میشه که در تصویر زیر انواع آن را ملاحظه میکنین:

$D_7IT_3 = p$
$IR(i) = B_i,\ i = 6, \ldots, 11$

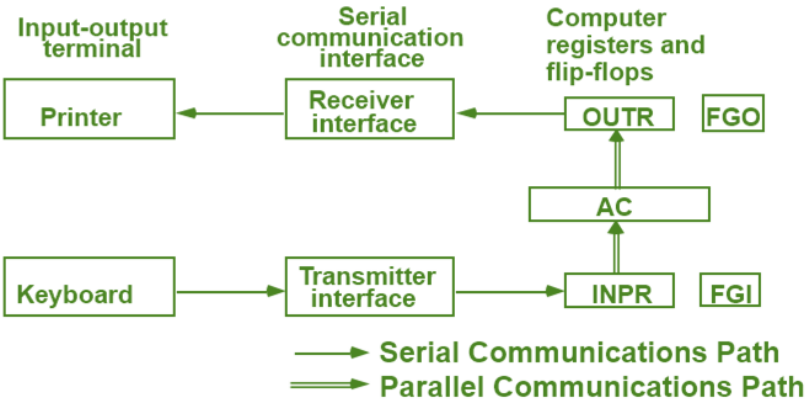
	p:	SC ← 0	Clear SC
INP	pB_{11}:	AC(0-7) ← INPR, FGI ← 0	Input char. to AC
OUT	pB_{10}:	OUTR ← AC(0-7), FGO ← 0	Output char. from AC
SKI	pB_9:	if(FGI = 1) then (PC ← PC + 1)	Skip on input flag
SKO	pB_8:	if(FGO = 1) then (PC ← PC + 1)	Skip on output flag
ION	pB_7:	IEN ← 1	Interrupt enable on
IOF	pB_6:	IEN ← 0	Interrupt enable off

IENیک بیت است که اگر ۰ باشد به هیچ وجه اجازه وقفه داده نمی شود و اگر ۱ باشد اجازه می دهد اگر وقفه ای آمد اجرا شود.

FGIو FGO فلگ های ورودی خروجی هستن . همونطور که در شکل اصلی کامپیوتر پایه هم دیدین , ثبات های ورودی و خروجی فقط با AC در ارتباط هستن یعنی ورودی پس از گرفته شدن در INPUT میره و بعد فقط میتونه در AC قرار بگیره و از طرفی فقط AC است که میتونه در خروجی مقدار بذاره.

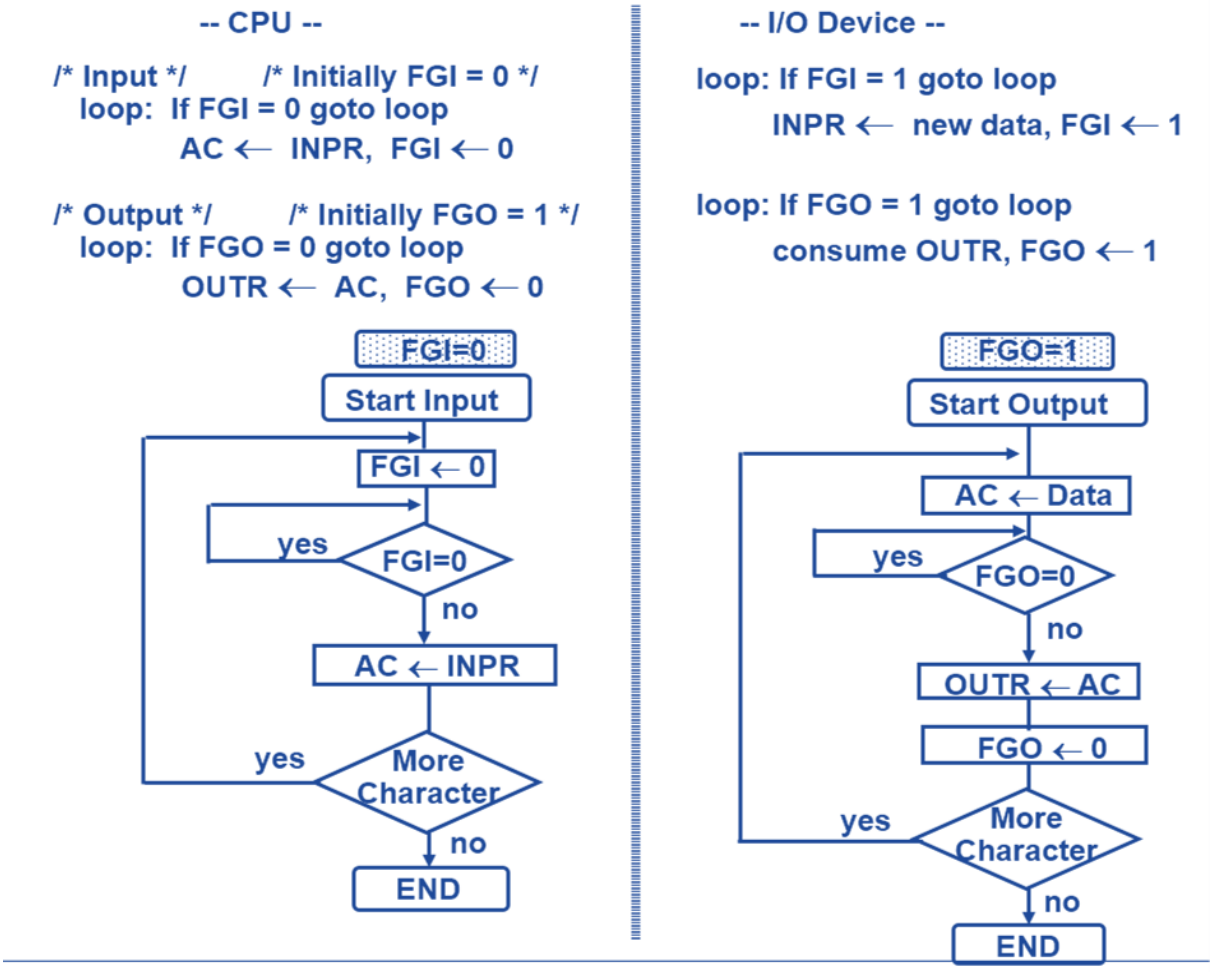
برای درک بهتر FGI , FGO یعنی بیت ورودی و بیت خروجی بهتر است خودتان را جای AC قرار دهید . هروقت یکی از این بیت ها ۱ شد یعنی AC میتونه کارشو انجام بده . یعنی اگر FGI = 1 شد یعنی ورودی اومده و آمادست که وارد AC بشه و اما اگر ۰ بود یعنی ورودی ای وجود ندارد که وارد AC بشه یا بهتره بگیم ورودی قبلی برداشته شده و ورودی جدیدی وجود ندارد . برای خروجی نیز اگر FGO = 1 شد یعنی ثبات خروجی آماده است که AC مقدار خودشو در ثبات خروجی قرار بده و وقتی ۰ شد یعنی ثبات خروجی پر شده و هنوز خالی نشده و AC نباید کاری بکنه در قسمت خروجی.

• Input-Output Configuration



INPR	Input register - 8 bits
OUTR	Output register - 8 bits
FGI	Input flag - 1 bit
FGO	Output flag - 1 bit
IEN	Interrupt enable - 1 bit

SKI , SKOنیز همانند if هایی که در مرحله ثباتی دیدین عمل میکنه یعنی برای فرار از حلقه اما اینبار SKIمیگه تا زمانی خط بعدی رو اجرا کن ورودی نیومده اگر ورودی اومد برو ۲ خط بعدی و اجراش کن و SKOنیز همین کار رو برای خروجی انجام میده.



حالا دستورات حافظه ای رو مشاهده می کنیم که انواع مختلف اونا چی هستن و چیکار میکنن:

Symbol	Operation Decoder	Symbolic Description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1$, if $M[AR] + 1 = 0$ then $PC \leftarrow PC+1$

که البته برای اجرای این دستورات یک کلاک کافی نیست و در ادامه به نحوه پیاده سازی و کد زنی آن وارد میشیم.

ISZدستوری است که مقدار یک خونه از حافظه رو میگیره و یکی زیاد میکنه میزاره سر جاس بعد چک میکنه اگر صفر به PC یکی اضافه میکنه که برای بیرون رفتن از حلقه از آن استفاده میشه. مثلا میخواین ۱۰ بار یک حلقه تکرار بشه , داخل یک خونه از RAM عدد –۱۰ رو قرار بده و آخر هر حلقه یک ISZ به اون خونه انجام بده مثلا) ISZ 250 ۲۵۰ خونه RAM است که عدد –۱۰ در اون قرار داره . (کد کامل شده در ادامه وجود داره.

فقط در مورد BUN و BSA توضیحاتی رو باید بدم چون در ادامه مورد نیاز است.

BUNهمون عمل پرش رو انجام میده یعنی داری یک خطی رو اجرا میکنی مثلا خط ۱۰ حالا دستور BUN ۲۵۰ رو میبینی یعنی باید بپری به خط ۲۵۰ یعنی خط بعدی که باید اجرا کنی باید ۲۵۰ باشه , کی میگه خط بعدی که باید اجرا کنی کجا ست , بله , PC پس عدد ۲۵۰ رو بده به . PC حالا چرا AR رو داده به . PC چون وقتی دارین دستور رو دیکد میکنین این عدد ۲۵۰ در همون ۱۲ بیت سمت راست IR قرار داره که هنگتم دیکد کردن اون ۱۲ بیت رو داده به AR یعنی الان ۲۵۰ به AR منتقل داده شده که حالا فقط کافیه AR رو وارد PC کنیم تا دستور بعدی که اجرا میشه ۲۵۰ باشه.

حالا میخوام BSA رو تو ضیح بدم که شباهت زیادی به BUN داره با فرق اینکه شما قراره بعد از پرش و انجام د ستورات دوباره برگردین به جایی که بودین . یعنی خط ۱۰ ه ستین می بینین نو شته BSA 250 یعنی بپر خط ۲۵۰ اجراش کن تا تموم بشه بعد برگرد به خط ۱۱) در سته ۱۱ چون خط ۱۰ همون BSA 250 ا ست و اجرا شده نباید دوباره اجرا بشه باید خط بعدیش اجرا بشه (که اینجا لازمه موقعی که از خط ۱۰ داری

میپری به خط ۲۵۰ (PC , که الان ۱۱ هست) رو یه جا ذخیره کنیم و وقتی خواستیم برگردیم دوباره برش داریم که بفهمیم باید به کجا برگردیم.

برای این کار هنگام BSA کردن یا ساده تر بگم فراخوانی یک تابع , خط اول تابع یعنی مثلا خط ۲۵۰ در اینجا خالی می باشد یعنی تابع از خط ۲۵۱ شروع می شود . ما در خط ۲۵۰ مقدار PC یعنی ۱۱ را ذخیره می کنیم بعد از اجرای تابع دوباره پرش میکنیم به خط ۲۵۰ می بینیم داخلش چند هست که ۱۱ هست و پرش می کنیم به خط ۱۱ یعنی برمیگردیم.

به مثال زیر توجه کنید:

خط آخر تابع نوشته ۱ BUN ۱۳۵ یعنی به طور غیر مستقیم بپر به خط ۱۳۵ یعنی برو خط ۱۳۵ ببین درونش چه مقداری هست یعنی ۲۱ بعد بپر به خط ۲۱.

درکل ما میریم به تابعی , آدرس برگشت رو در خط اول تابع میذاریم و موقع برگشت از همونجا بر میداریمش و برمیگردیم.

روش های دیگری هم برای استفاده از توابع و برگشت وجود داره مثلا استفاده از پشته یا یک ثبات برای ذخیره کردن آدرس برگشت توابع که هرکدام مشکلات خاص خودشو داره.

مثلا پشته به کنترل زیادی نیاز داره که آدرس ها و دیتا ها با هم قاطی نشه و بدونی آدرس کجاست و دیتا کجاست.

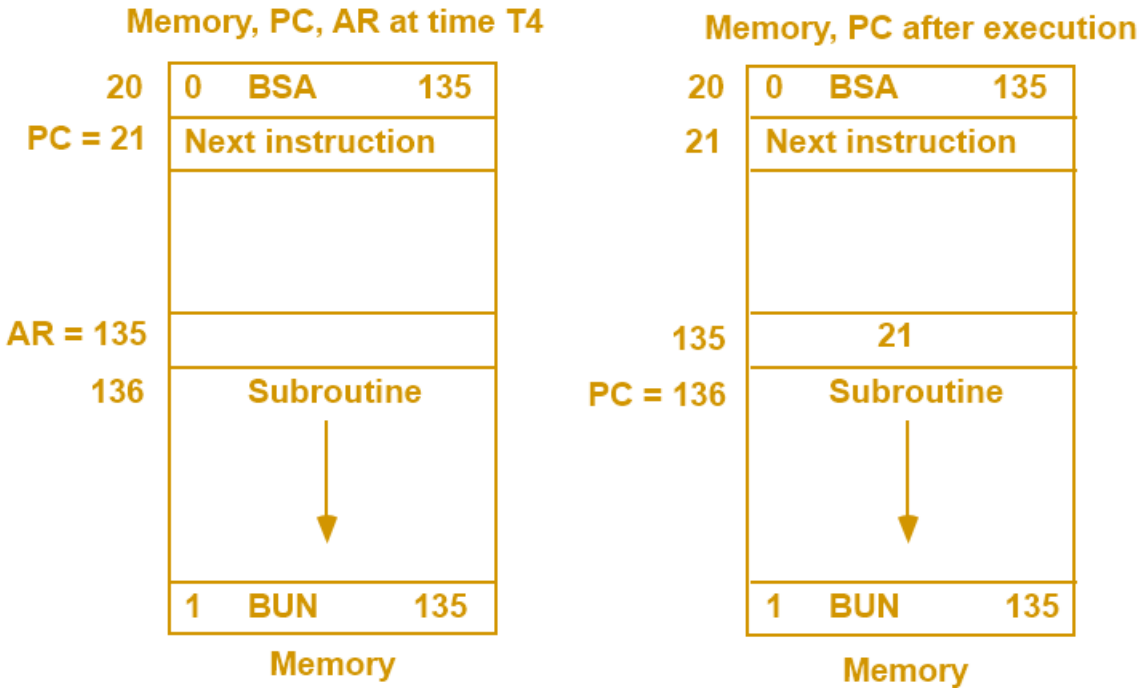
وجود ثبات خب یک ثبات کمه یعنی فقط میتونی یک تابع فراخوانی کنی و در یک تابع نمیتونی تابع دیگری رو فراخونی کنی چون اگر از تابع اول بری به تابع دوم فقط آدرس برگشت به تابع اول رو داری نه آدرس برگشت به کد اصلی رو یا به عبارتی پیاده سازی تابع های تودرتو و بازگشتی امکان پذیر نیست.

نوشتن آدرس برگشت در خونه اول هم این مشکل رو داره که نمیشه باهاش توابع بازگشتی رو پیاده سازی کرد چون یه آدرس بازگشت دیگه جای آدرس بازگشت قبلی می شینه و آدرس قبلی گم میشه.

خب فراخوانی تابع رو هم متوجه شدیم چطوریه حالا میخوام بگم وقتی وقفه میاد چه اتفاقی میفته.

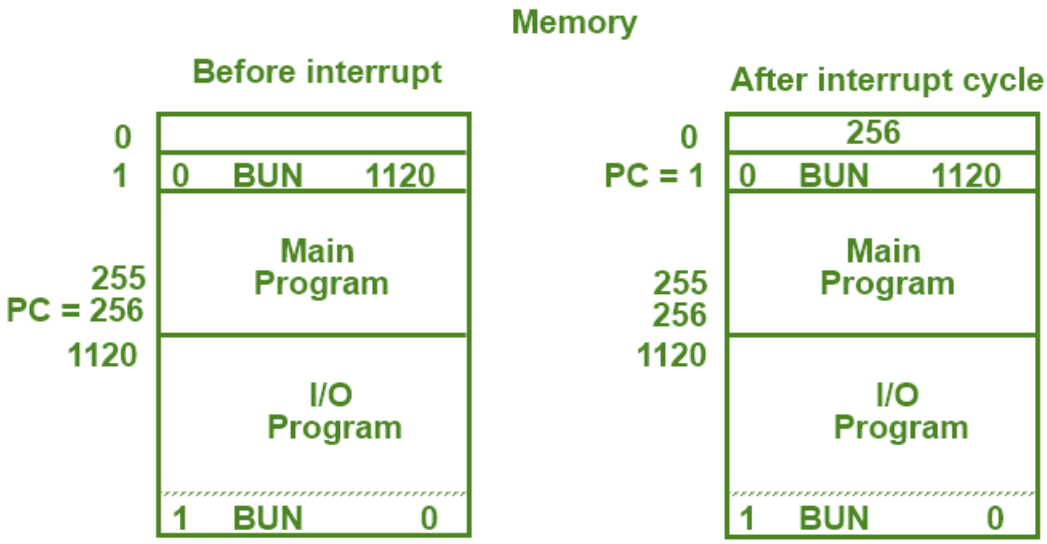
در وقفه در اصل ما یک فراخوانی تابع داریم به خط ۰ و بعد از اونجا هم یک پرش داریم به جایی که زیرروال ما وجود داره و میخوایم اجراش کنیم برای وقفه و پس از اجرا به محل اولیه برمیگردیم.

به تصویر زیر توجه کنید:



بیانگر این است که R نشان میدهد که وقفه داریم یا نه که اگر باشه به سمت راست میره و وقفه رو آماده اجرا میکنه و در مرحله بعد اجراش میکنه.

به شکل زیر توجه کنید:

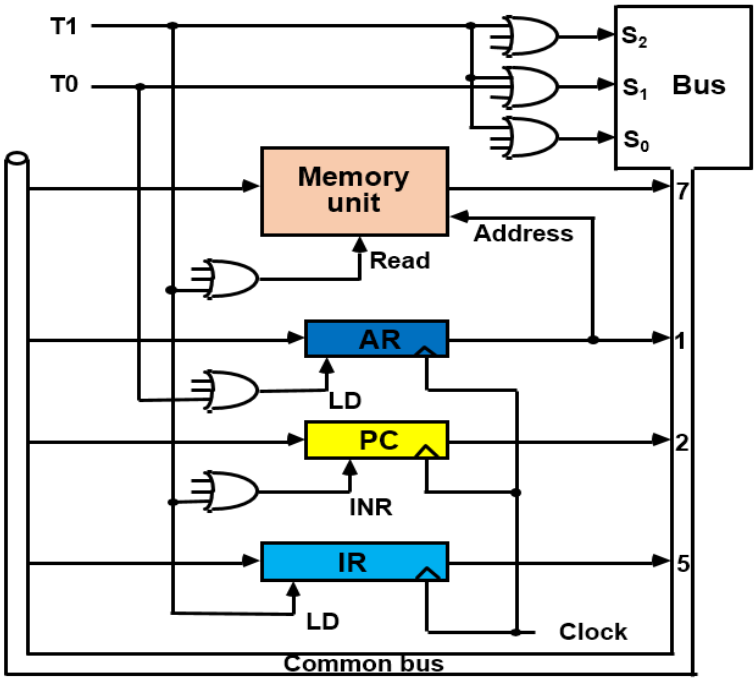


وقفه اومده ما داشتیم خط ۲۵۵ رو اجرا می کردیم , خط ۰ رو فراخونی میکنیم و بعدشم خطی که کد مربوط به وقفه داره پرش میکنیم (خط ۱۱۲۰) و بعد یک راست برمیگردیم به خطی که از آن جا خط ۰ را فراخوانی کردیم(۲۵۶).

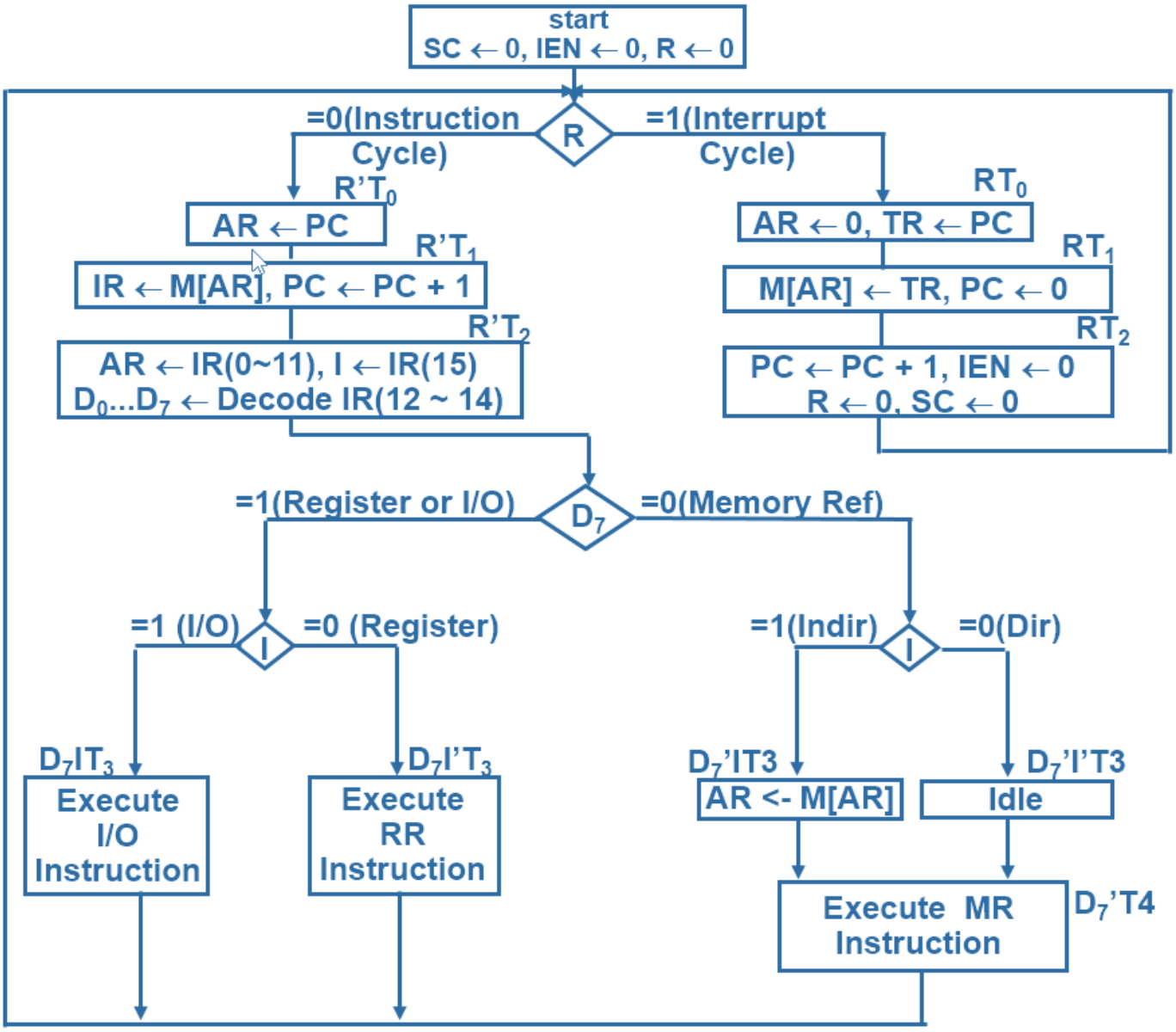
خب حالا باید فلوچارت اصلی رو با این وقفه پیوند بزنیم و یک فلوچارت کامل ارایه بدیم .

- **Fetch and Decode**

T0: AR ← PC ($S_0S_1S_2$=010, T0=1)
T1: IR ← M [AR], PC ← PC + 1 (S0S1S2=111, T1=1)
T2: D0, . . . , D7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)



به شکل زیر توجه کنید:



در اینجا می بینید که در صورت وجود وقفه سمت راست فلوچارت و در صورت نبودن وقفه سمت چپ فلوچارت اجرا خواهد شد.

دقت کنید دستورات در سمت راست به شرط یک بودن R اجرا میشن که نوشتن RT0 , RT1 , RT2 ولی در سمت چپ فقط در ۳ مرحله اول R' لحاظ شده یعنی اگر در ۳ مرحله اول اجرای سیکل دستور باشیم و ناگهان وقفه بیاد بیخیال میشه دستور رو و میره وقفه رو اجرا میکنه اما اگر اون ۳ مرحله رد کنه دیگه وقفه هم بیاد در مرحله بعد چک میشود.

خب حالا میخوایم کم کم وارد بحث پیاده سازی بشیم لطفا ابتدا به تصویر زیر توجه کنید:

Symbol	Hex Code I = 0	Hex Code I = 1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load AC from memory
STA	3xxx	Bxxx	Store content of AC into memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instr. if AC is positive
SNA	7008		Skip next instr. if AC is negative
SZA	7004		Skip next instr. if AC is zero
SZE	7002		Skip next instr. if E is zero
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

در این تصویر تمام کد هایی که تا الان گفتم نوشته شده و پیاده سازی شده مثلا ADD که ابتدا مقدار درون RAM رو در DR نوشته و در کلاک بعد AC رو با DR جمع کرده.

کدهای دیگه هم پیاده سازی شدن.

اینجا مبحثی از فصل ۴ هست که اشاره ای به آن می کنیم که آن هم پیدا کردن ورودی های load , inc , clear ثبات هاست . خب ما گفتیم مثلا اگر clear برای یک ثبات ۱ بشه ثبات مقدارش ۰ میشه ولی نگفتیم چه موقع و چطوری clear برابر ۱ خواهد شد که اینجا به آن می پردازیم.

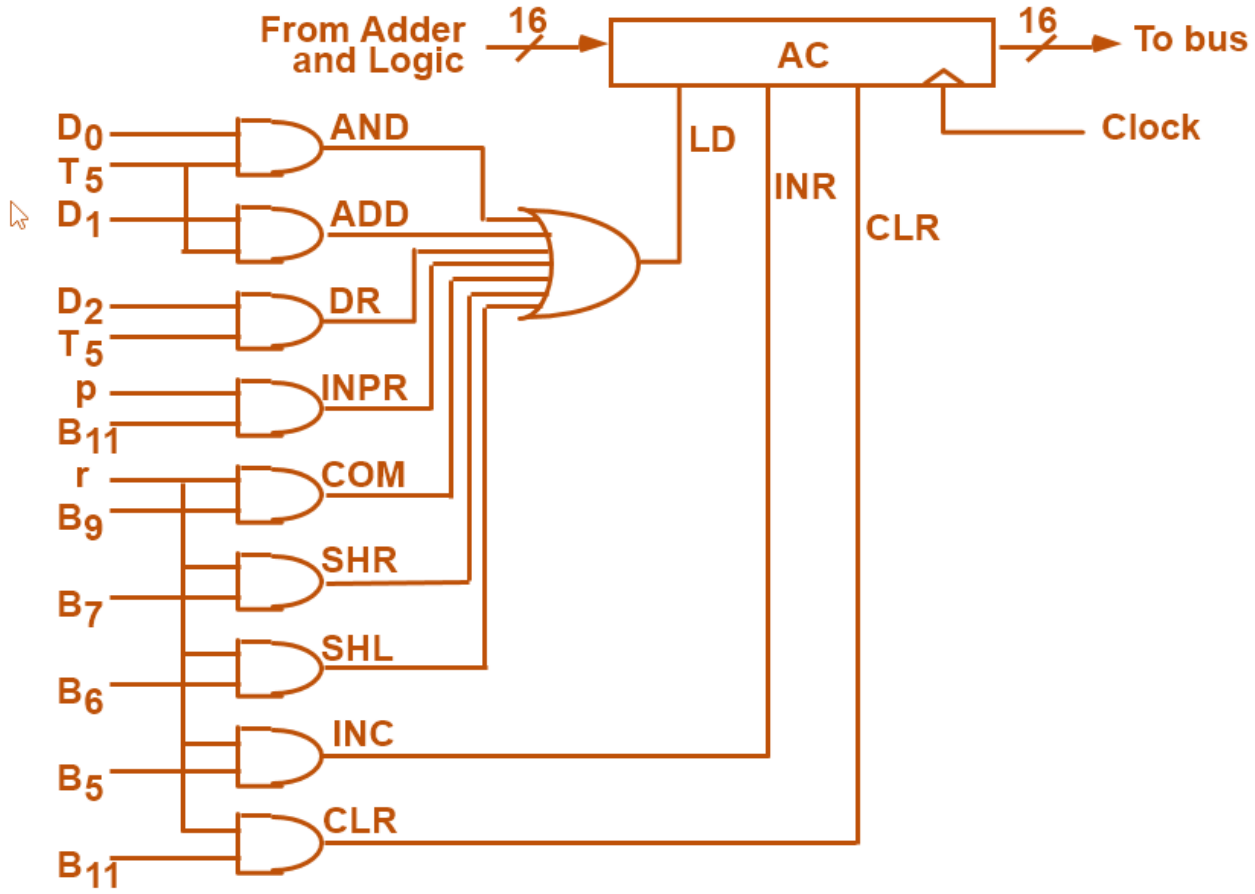
اگر دقت کنین در تصویر بالا کنار هر کد یک شرط هم نوشته شده مثلا RT0 یا DT1 و ... که این ها بیانگر شرط اجرا شدن اون کد است یعنی اگر اون شرط برقرار باشه کد مقابلش اجرا میشه.

حالا مخوایم بدونیم چه زمانی مثلا AC , INC شده است یعنی کجا AC+1 شده.

حالا باید برید در این کد بگردید هرجا که AC <- AC + 1 نو شته شده رو پیدا کنید , شرط ها شو با هم orکنید به handle inc برای AC متصل کنید.

به شکل زیر توجه کنید:

Gate structures for controlling the LD, INR, and CLR of AC



در این شکل ما تمام handle های AC رو مقدار دادیم اینطوری که هرجا AC یکی زیاد شده شرطشو دادیم به , INC هرجا ۰ شده شرطشو دادیم به clear و هرجا مقداری وارد AC شده شرطش رو دادیم به loadکه هرجا تعداد شرط ها از یکی بیشتر شده همه رو با هم or کردیم.

این مبحث بیشتر مربوط به فصل ۴ است و بیشتر از این اینجا توضیح نمیدم

مبحث اصلی آخری که میخوام راجبش صحبت کنم کد نویسی در این جا هست که خیلی مهمه و باید تسلط کاملی روی اون داشته باشید که من سعی دارم با چند مثال ساده و سخت این آشنایی رو در شما به وجود بیارم.

از مثال ساده ADD شروع میکنیم. دستور اینه که ما باید AC رو با M[AR] جمع کنیم و در AC قرار بدیم.

AC <- AC + M[AR]

برای جمع کردن می دونیم که باید یه سر ماجرا AC باشه که هست و سر دیگر ماجرا باید DR باشه پس باید M[AR] رو بدیم به DR و سپس عملیات جمع رو انجام بدیم :

۱) DR <- M[AR]

۲) AC <- AC + M[AR]

مثال بعدی:

M[AR] <- AC , AC <- M[AR]

یعنی مقدار داخل RAM با مقدار AC جابه جا بشن.

برای اینکه مقدار RAM وارد AC بشه شکی نیست که باید اون رو بده به DR , سپس DR رو بدیم به AC اما با اینکار مقدار قبلی AC که باید بره در RAM قرار بگیره خراب میشه)مقدار قبلی AC خراب میشه.(

راه اول اینه که AC رو بذاریم در TR و ازش کمک بگیریم و بعدش TR رو بدیم به: RAM

۱) DR <- M[AR]

۲) TR <- AC

۳) AC <- DR

۴) M[AR] <- TR

که اگر دقت کنید می بینید که کد شماره ۲ و ۳ رو میشه همزمان انجام داد یعنی:

- ۱) DR <- M[AR]
- ۲) TR <- AC , AC <- DR
- ۳) M[AR] <- TR

راه دوم : همونطور که اول هم گفتم سعی کنین از TR استفاده نکنین , ببینید:

- ۱) DR <- M[AR]
- ۲) AC <-DR , M[AR] <- AC

همزمان که مقدار DR از طریق مدار محاسبه منطق و بدون دخالت باس داره وارد AC میشه AC , مقدار قبلیشو از طریق باس وارد RAM میکنه.

مثال بعدی رو میخوام طوری حل کنم که بعد از اجرای دستور AC , همون مقداری رو داشته باشه که قبل از اجرای دستور داشت یعنی مثلا قبل از اجرا درون AC عدد ۲۰ بود بعد از اجرا هم داخلش ۲۰ باشه یعنی مقدارش خراب نشه.

M[AR] <- M[AR] + AC

ابتدا مثال را با استفاده از TR حل میکنم:

- ۱) DR <- M[AR]
- ۲) TR <- AC , AC <- AC + DR
- ۳) M[AR] <- AC
- ۴) DR <- TR
- ۵) AC <- DR

در خط دوم برای خراب نشدن مقدار AC اون رو در TR گذاشتم و در خط ۴ و ۵ به AC برگردونده شد.

حالا بدون استفاده از: TR

- ۱) DR <- M[AR]
- ۲) AC <- AC+DR , DR <- AC
- ۳) AC <- DR , M[AR] <- AC

در خط دوم همزمان با جمع کردن با مدار محا سبه منطق , از طریق باس مقدار قبلی AC به DR داده شد و در خط سوم با استفاده از مدار محاسبه منطق مقدار قبلی AC که در DR بود به AC برگردونده شد و حاصل جمع هم از طریق باس به RAM رسید.

مثال بعدی:

IF(M[AR]=AC)
PC <- PC+1

ما نمیتونیم مستقیم دو عدد رو مقایسه کنیم . برای اینکه مقایسشون کنیم باید منها کنیم و مقدار نهایی رو در AC قرار بدیم . حالا ما میتونیم با استفاده از دستورات ثباتی AC را چک کنیم مثلا چک کنیم AC صفر است یا خیر) با استفاده از(SZA

میشـه فهمید AC مثبت اسـت یا خیر) با (SPA و میشـه فهمید منفی اسـت یا نه) با . (SNA ما اینجا مقدار RAM را با AC منها میکنیم و نتیجه را در AC قرار می دهیم و سپس چک میکنیم که صفر شده یا نه در نظر داشته باشید که میخوام از خراب شدن AC در انتهای برنامه نیز جلوگیری کنم:

- ۱) DR <- M[AR]
- ۲) AC <- AC – DR , DR <- AC
- ۳) SZA
- ۴) HLT
- ۵) PC <- PC +1 , AC <-DR

در خط سوم چک شده اگر AC صفر نشده برو خط ۴ و برنامه تموم میشه اما اگر AC صفر شده برو خط ۵ و اجراش کن چیزی که خواسته شده که اینجا خواسته شده به مقدار PC اضافه بشه.

فقط در نظر داشته باشید که هیچوقت نمیشه همزمان دوتا handle از یک ثبات فعال باشند.

مثال : آیا کد زیر درست است!؟

- AC <-0 , AC <- AC +1

خیر چون هم CLEAR و هم INC در AC فعال شده است.

- AC <- AC+1 , AC <- AC +DR

خیر چون LOAD و INC با هم فعال شده اند.