آزمایش دوازدهم: کامپیوتر پایه – ادامه (VHDL)

در ادامه قصد داریم با استفاده از اطلاعاتی که در آزمایش قبل بدست آوردیم اقدام به طراحی کامپیوتر پایه با استفاده از VHDL کنیم.

نکته: فایل VHDL مربوطه در پیوست موجود است (برگرفته از صفحه درس)

ابتدا پورت های ورودی ماژول را تعریف میکنیم:

- Reset: همان ریست است
- CLK: کلاک پردازنده
- Data: لاین دیتا، ۱۶ بیت (Word Size)
- Address: لاین آدرس، ۱۲ بیت (2^{12})
- Memory Read: یک تک بیت منطقی برای مشخص کردن حالت خواندن از رم
- Memory Write: یک تک بیت منطقی برای مشخص کردن حالت نوشتن روی رم
- Inport: پورت ورودی ۸ بیتی
- Outport: پورت خروجی ۸ بیتی
- Intr_in: وقفه برای ورودی (تک بیت منطقی)
- Intr_out: وقفه برای خروجی (تک بیت منطقی)

در ادامه کد به توصیف Architecture کامپیوتر پایمون میپردازیم:

ابتدا ثبات های خاص منظوره را تعریف میکنیم:

- Dr: همان Data Register است که باتوجه به نکات بالا باید ۱۶ بیتی باشد (چون خط دیتا هم ۱۶ بیتی است). این ثبات دیتا را نگه میدارد.
- AR: همان Address Register است که با توجه به نکات بالا باید ۱۲ بیتی باشد (چون لاین آدرس هم ۱۲ بیتی است). این ثبات وظیفه نگه داری آدرس را برعهده دارد.

- AC: همان accumulator است و باتوجه به آنکه از جنس دیتاست، ۱۶ بیتی تعریفش میکنیم. این ثبات در ارتباط با ALU مورد استفاده قرار میگرد. در واقع ثباتی است که نتایج محاسبات را در آن ذخیره و حساب میشوند.
- IR: همان Instruction Register است و با توجه به آنکه از روی مموری خوانده میشود، ۱۶ بیتی تعریفش میکنیم. این رجیستر وظیفه نگه داری ادرس دستوری که درحال حاضر میخواهیم اجرا کنیم را برعهده دارد.
- PC: همان Program Counter است و از آنجایی که از جنس آدرس است، ۱۲ بیت تعریفش میکنیم. وقتی آدرس را میخوانیم و در IR میریزیم، بلافاصله PC را یکدونه اضافه میکنیم و در واقع به خط بعدی که میخواهیم اجرا کنیم اشاره میکند:

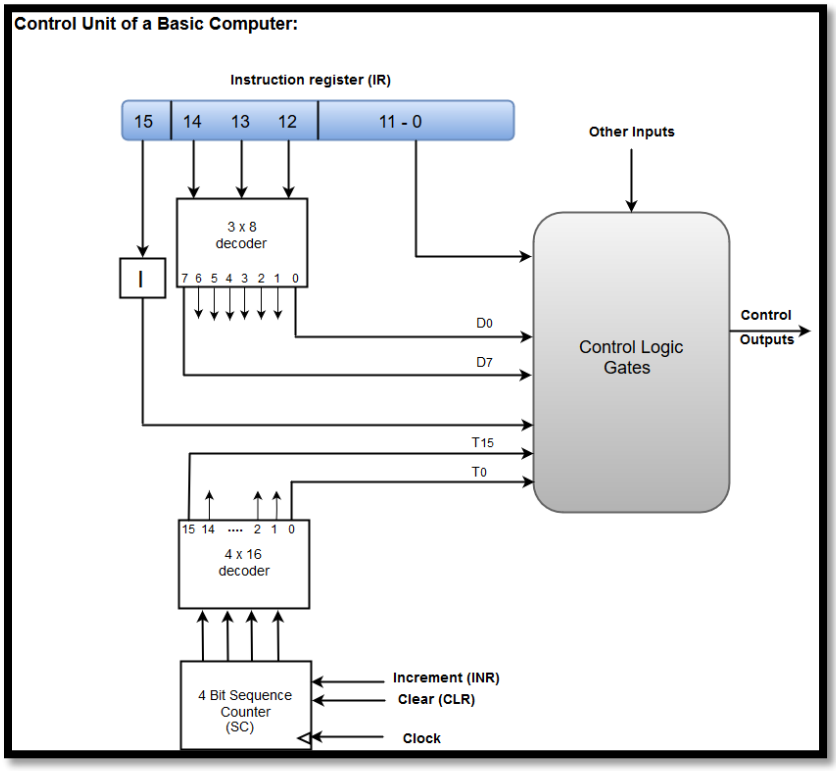
IR <- M[PC]

- TR: همان Temp Register است و صرفا یک رجیستر برای نگه داری موقتی دیتا هاست. چون از جنس دیتاست، ۱۶ بیت است
- INPR: همان Input Register است که فرمان ورودی را نگه داری میکند. همانند پورت ورودی آن ۸ بیتی است.
- OUTR: همان Output Register است که فرمان خروجی را نگه داری میکند. همانند پورت خروجی آن ۸ بیتی است

بعد از تعریف ثبات های خاص منظوره بایستی یکسری موجودیت دیگر نیز تعریف کنیم که در طراحی باس پیشتر به آنها اشاره کرده بودیم:

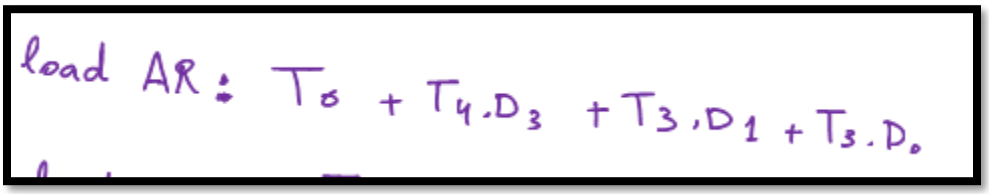
- Instruction Decoder: یک دکودر ۸ بیتی که درواقع مشخص میکند کدام Operation Code مدنظر ماست. (از آنجایی که در طراحیمان تعداد بیت های OPCODE را ۳ بیت درنظر گرفتیم، دکود شده آن ۸ بیت خواهد بود)

- Sequence Counter: برای درک بهتر این موجودیت بهتر است به تصویر زیر دقت کنید:



همانطور که در تصویر فوق پیداست، ۴ بیت برای شمارنده اختصاص میدهیم که میتواند تا ۲ به توان ۴ یعنی ۰ تا ۱۵ شمارش کند. این شمارنده برای اینکه تشخیص دهیم در اجرای دستورالعمل دقیقا درکدام مرحله هستیم ضروری است.

- Timing Signals: این سیگنال ها به ما کمک میکنند تا Micro Operation های خود را به گونه ای تعریف کنیم که در زمانی خاص اجرا شوند و معنادار باشند. برای مثال:



- بیت I: بیت مد که شیوه آدرسدهی را مشخص میکند (Direct Or Indirect)
- بیت E: بیتی که اضافه (سر ریز) خروجی ALU در آن ریخته میشود
- بیت S: بیت شروع-پایان فلیپ فلاپ (تک بیت)
- بیت Enable Decoder: بیت Enable برای دکودر دستورالعمل

- بیت Clear: بیت Clear برای پاک کردن رجیستر Sequence Counter (تک بیت)
- بیت FGI: بیت Flag برای ورودی (تک بیت)
- بیت FGO: بیت Flag برای خروجی (تک بیت)
- بیت IEN: بیت Interrupt Enable (وقفه)
- بیت r: بیت فلیپ فلاپ وقفه

حال به سراغ Process ها میرویم:

ابتدا امر رفتار Instruction Decoder را توصیف میکنیم:

۸ حالت مختلف داریم، که برای هرکدام نیاز است که یک بیت مشخص خروجی Decoder (معادل دسیمال آن عدد کد شده) روشن شود. نکته مهم در این بخش این است که اینکار را با استفاده از بیت ۱۴ الی ۱۲ دستورالعمل انجام میدهیم. (همانند تصویر صفحه قبل)

در عین حال شرط این Process، فعال بودن Enable Decoder است.

در Process بعدی، کاری میکنیم که در هر Clock، درصورتی که بیت Clear روشن نبود آن را یک عدد اضافه میکنیم (از صفر تا ۱۵ میشماریم). درصورتی که یک بود، صفر اش میکنیم

در Process بعدی با توجه به مقدار فعلی Sequence Counter اقدام به تعیین Time Signal میکنیم.

درصورتی که پورت Reset، ۱ باشد، تایم صفر میشود

Process بعدی که اصل کار است و در واقع همان بخش Control Unit است، به شرح زیر صورت میگیرد (بیس Time Signal):

ابتدا ۴ متغیر تعریف میکنیم که استفاده هرکدام را در ادامه توضیح خواهیم داد:

- Temp: تک بیت
- Sum: ۱۷ بیت

- AC_EXT: ۱۷ بیت
- DR_EXT: ۱۷ بیت

نکته: اگر Reset یک بود، تمامی سیگنال ها را حالت پایه شان برمیگردانیم.

اگر در T0 بودیم و همچنین Interrupt نداشتیم، مقدار Memory Read را یک و محتوای PC را درون AR میریزیم.

در مرحله بعد اگر در T1 بودیم و همچنین Interrupt نداشتیم، دستورالعمل را واکشی (Fetch) کرده و PC را یک عدد افزایش میدهیم(پوینت کند به خط بعدی)

در مرحله بعد اگر در T2 بودیم و همچنین Interrupt نداشتیم، Opcode را دکود میکنیم. برای اینکار Enable دکودر را یک و سپس همچنین ۱۲ بیت (کم ارزش) را در Address Register و بیت ۱۵ را در I ذخیره میکنیم.

اگر Interrupt داشتیم و همچنین در T0 بودیم، آدرس خروجی (Return Address) را در Temp Register ذخیره میکنیم.

اگر Interrupt داشتیم و همچنین در T1 بودیم آدرس خروجی را در خانه ۰ موری ذخیره میکنیم

اگر Interrupt داشتیم و در T2 بودیم، Program Counter را یکدونه اضافه و Interrupt و Sequence Counter را ریست میکنیم.

اگر در T3 بودیم (مرحله عملیات ها):

- اگر D7 اکتیو بود (یعنی OPCODE = 111) همچنین I نیز اکتیو بود: یعنی دستور العمل از نوع I/O است. در کد ۶ حالت مختلف دستور تعریف شده است:
  - INPR: وارد کردن کاراکتر (در AC ذخیره میشود – ۸ بیت)
  - OUT: خروجی دادن مقدار AC – ۸ بیتی (در رجیستر OUTR)
  - SKI: درصورتی که FGI اکتیو بود، Program Counter را یکی زیاد میکند
  - SKO: درصورتی که FGO اکتیو بود، Program Counter را یکی زیاد میکند
  - ION: Interrupt را فعال میکند
  - IOF: Interrupt را غیر فعال میکند
- بعد از انجام آن Sequence Counter را ریست میکنیم.

- اگر D7 اکتیو بود و هچنین I غیر اکتیو بود: یعنی به دستورالعمل های ثباتی طرفیم. در اینجا ۱۲ دستورالعمل مختلف ثباتی تعریف شده است:

  - CLA: پاک کردن AC
  - CLE: پاک کردن E
  - CMA: مکمل AC را در خودش میریزد
  - CME: مکمل E را در خودش میریزد
  - Circulate Right: شیفت دایره ای به راست (دقت شود AC و E در اینجا یکی درنظر گرفته میشوند)
  - CIL: شیفتت دایره ای به چپ
  - INC: افزایش AC
  - SPA: اگر عدد در AC مثبت بود، Program Counter را یکی زیاد کن (به بیت ۱۵ ام آن دقت میکنیم)
  - SNA: اگر عدد در AC منفی بود، Program Counter را یکی زیاد کن (به بیت ۱۵ ام آن دقت میکنیم)
  - SZA: اگر AC صفر بود، Program Counter را یکی زیاد کن.
  - SZE: اگر E صفر بود، Program Counter را یکی زیاد کن.
  - HLT: Control Unit را تا وقفه بعدی متوقف کن

  در نهایت Sequence Counter را ریست میکنیم.

- اگر D7 یک نبود و همچنین I یک بود، آدرس را به خاطر Indirect Mode واکشی کن
- در غیر اینصورت کاری نکن

اگر در T4 بودیم و I = 1 بود و همچنین D7 اکتیو بود، آدرس غیر مستقیم را واکش

ی میکنیم و در AR میریزیم. در غیر اینصورت کاری نمیکنیم.

اگر در T5 بودیم چند سناریو مختلف داریم:

- اگر در D0 و D1 و D2 و D6 قرار داشته ایم(یعنی اکتیو بودند) از آنجایی که به ترتیب نماینده عملیات های AND و ADD و LDA و ISZ اند، Memory Read را اکتیو میکنیم.
- اگر در D3 و D5 قرار داشتیم(یعنی اکتیو بودند)، Memory Write را اکتیو میکنیم
- وظیفه D3 یا همان STA ذخیره مقدار AC در موری است
- وظیفه D4 یا همان BUN این امکان را میدهد که مقدار AR را در PC بریزیم. با اینکار برنامه نویس این امکان را دارد که در کامپیوتر ما JUMP انجام دهد(بی قید و شرط)

- The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally. The instruction is executed with one rnicrooperation:
- D4T4: PC <-- AR, SC <-- 0

اگر در T7 بودیم و همچنین D0 اکتیو بود به این معناست که باید عملیات AND را انجام دهیم. برای اینکار صرفا محتوای ثبات AC با محتوای ثبات DR، AND میشوند. در نهایت مثل همیشه Sequence Counter را ریست میکنیم.

اگر در T7 بودیم و همچنین D1 اکتیو بود به این معناست که باید عملیات جمع را انجام دهیم. نکته مهم در اینجاست که بیت ۱۶ ام حاصل جمع مربوط به ثبات E است. در نهایت مثل همیشه Sequence Counter را ریست میکنیم.

اگر در T7 بودیم و همچنین D2 اکتیو بود به این معناست که صرفا باید محتوای ثبات DR را در AC لود کنیم.

اگر در T7 بودیم و همچنین D6 اکتیو بود به این معناست که DR را یکی افزایش میدهیم!

اگر در T8 بودیم و همچنین D6 اکتیو بود، اگر دیتا صفر بود، Program Counter را یکی زیاد میکنیم. و درنهایت Sequence Counter را نیز ریست میکنیم.

موفق باشید