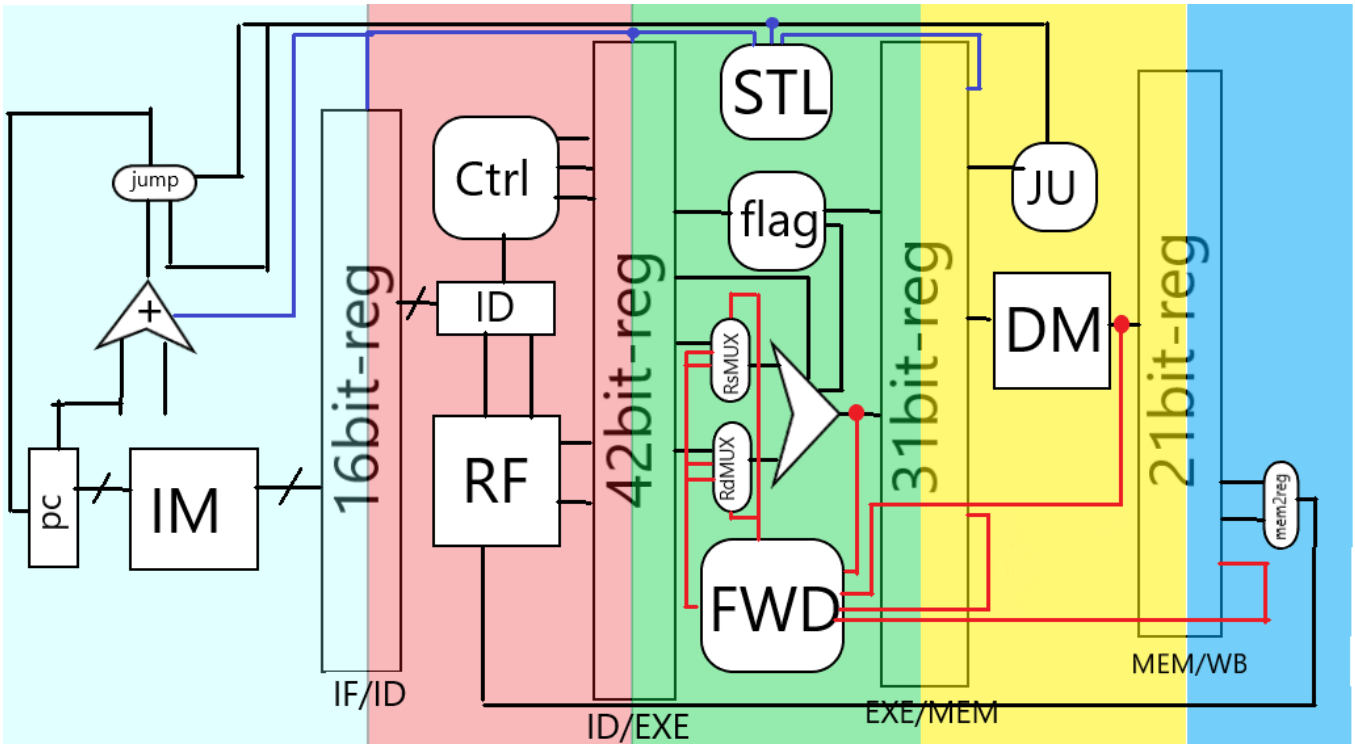
گزارش کار گروه هفتم :

کوروش خاوری، زهرا مختاری، علی ایوبی

پردازنده‌ی ۸ بیتی خط لوله

اجزای به کار رفته در قطعه و شرح کارکرد آن‌ها :



۱- قطعات IF/ID, ID/EXE, EXE/MEM, MEM/WB رجیسترهایی هستند که برای حالت بندی پردازنده به ۵ حالت Instruction Fetch, Instruction Decode, Execution, Memory, Write Back به کار میروند.

۲- قطعه‌ی Ctrl همانند واحد کنترل پردازنده‌ی تک‌حلقه به کار می‌رود.(هر چند این بار نمونه‌ی بسیار ساده شده‌ای از آن را طراحی کردیم.)

۳- قطعه‌ی ID برای استخراج قسمت های مختلف دستور همانند شماره رجیستر ها و مقدار ثابت دستور و آپ کد دستور به کار می‌رود.

۴- قطعه‌ی RF همان Register File است که برای نگه‌داری و مدیریت رجیسترهای پردازنده به کار می‌رود. بر خلاف فایل رجیستر پردازنده تک حلقه، این قطعه با لبه‌ی پایین رونده اطلاعات را درون رجیسترها می‌نویسد، همچنین به صورت آسنکرون رجیسترها را می‌خواند.

۵- قطعه‌ی STL یا همان Stall Unit برای رفع Load Hazard تعبیه شده است: همانطور که می‌دانید اطلاعات مربوط به دستور load در خروجی DM حاضرخواهد شد، بنابراین، این قطعه با متوقف کردن دستورهای پشت دستور load – در صورت لزوم- به کمک قطعه‌ی Forwarding Unit این مشکل را مدیریت خواهد کرد.(یک حباب در خط لوله وارد می‌کند)

۶- قطعه‌ی FWD یا Forwarding Unit برای رفع مشکل هماهنگی رجیسترها به کار می‌رود: اگر بخواهیم قبل از مرحله‌ی Write Back و به روز رسانی رجیستر مقصد در دستورهای نوعR، از رجیستر مقصد استفاده کنیم، باید محتوای آن را از خروجی واحد پردازش یا قسمت mem به محل های قبلی Forward کنیم که این کار وظیفه‌ی این قطعه است.

۷- JU یا Jump Unit برای بررسی محتوای رجیستر flag ها(که هم اکنون بخشی از EXE/MEM رجیستر است) به کار می‌رود و با توجه به نوع دستور Jump و محتوای رجیستر مربوطه، تعیین می‌کند که پرش انجام شود یا خیر: در صورت انجام پرش، محتوای دو رجیستر قبل از پرش، Flush خواهد شد، که این به معنی ورود دو حباب به خط لوله می‌باشد. (توجه به این نکته لازم است که پردازنده‌ی ما، Jump Not Taken Prediction انجام می‌دهد، یعنی به طورپیشفرض درنظر می‌گیرد که پرش انجام نشده است و دستور های بعد از آن‌را وارد میکند، در صورت انجام نشدن، هیچ حبابی وارد پردازنده نخواهد شد.)

```
data[00] = 16'b10110_001_00000011;// $1 <= 3
data[01] = 16'b0000001110_001_010;// rotate $1, 2 units right :: forwarding type 1 :: $1 <= 192(00000011 -> 11000000)
data[02] = 16'b10110_010_00000101;// $2 <= 5
data[03] = 16'b0000000001_111_001;// $7 <= $7 + $1 :: $7 <= 192
data[04] = 16'b0000000001_111_010;// $7 <= $7 + $2 :: $7 <= 197 :: forwarding type 2
data[05] = 16'b10101_000_00001010;// jump to line 10
data[06] = 16'b10110_101_00010101;// this instruction will be flushed out! if not it will cause $5 <= 21
data[07] = 16'b10110_110_00011011;// this instruction will be flushed out! if not it will cause $6 <= 27
data[08] = 16'b10101_000_00000001;// this instruction is unreachable ! if not it will jump to line 1 (loop)
data[09] = 16'b10101_000_00000001;// this instruction is unreachable ! if not it will jump to line 1 (loop)
data[10] = 16'b10110_111_00000111;// $7 <= 7
```

برای آزمایش پردازنده، از کد بالا برای دستور های موجود در حافظه‌ی دستورات بهره برده ایم :

=دستور خط اول، مقدار ۳ را در رجیستر ۱ قرار میدهد.

=دستور خط دوم، مقدار رجیستر ۳ را ۲ واحد به سمت راست، rotate می‌کند، بنابراین انتظار داریم 00000011 (مقدار اولیه رجیستر ۳) به 11000000 یا همان ۱۹۲ تبدیل شود. دقت شود که اگر ساز و کار مناسبی برای Forwarding در نظر نگیریم، در این خط دچار مشکل خواهیم شد، چرا که با مقدار ثانویه‌ی رجیستر ۱که۳ است باید کار کنیم و نه مقدار اولیه و در این جا باید Forwardingنوع۱ انجام دهیم : انتقال مقدار جدید رجیستر ۳ از MEM به EXE

دستور خط سوم، مقدار ۵ را در رجیستر ۲ قرار می‌دهد.

دستور خط چهارم، مقدار رجیستر۷ را با رجیستر ۱ جمع کرده در رجیستر ۷ قرار می‌دهد.

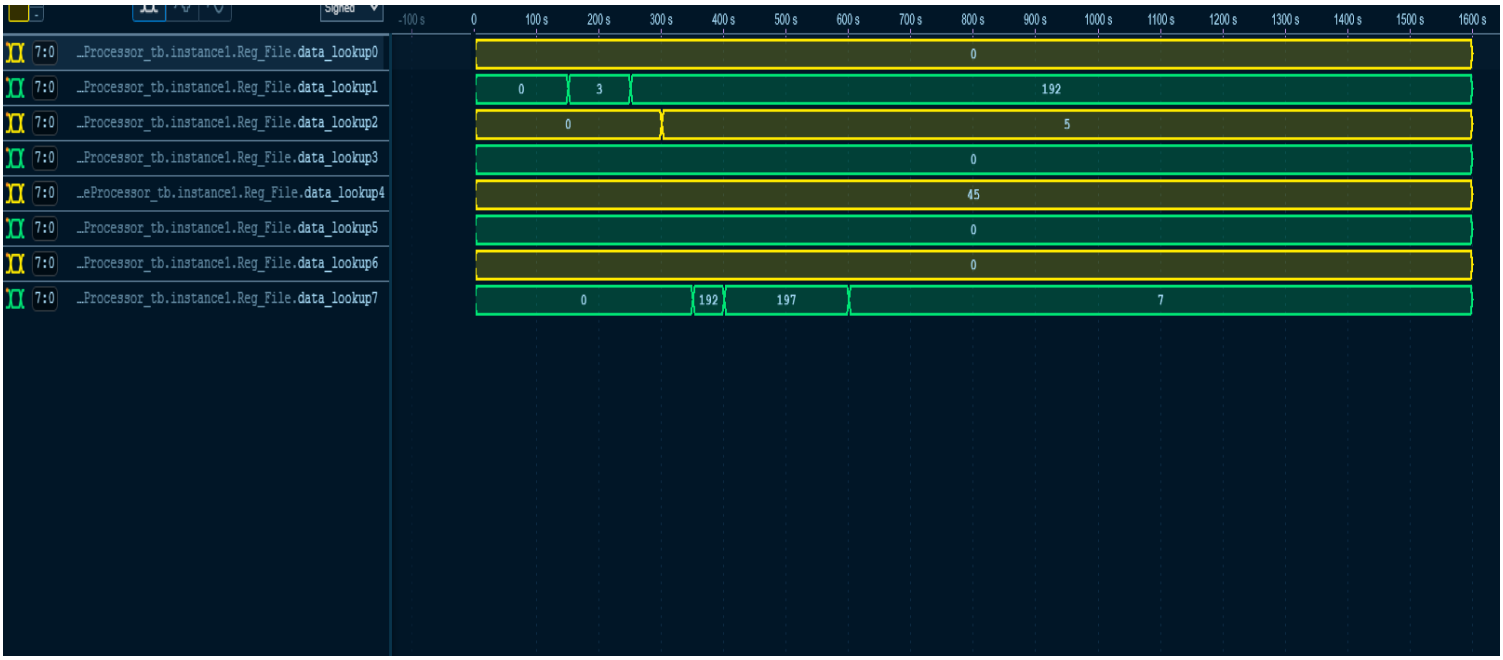
دقت شود که مقدار اولیه ی رجیستر ۷ برابر صفر است. پس انتظار داریم مقدار آن ۱۹۲ شود.

دستور خط پنجم، مقدار رجیستر ۷ را با رجیستر ۲ جمع کرده و در رجیستر ۷ زخیره خواهد کرد، بنابراین انتظار داریم در پایان این عمل، مقدار رجیستر ۷ برابر با ۱۹۷ شود، دقت شود که اگر از Forwarding نوع ۲ استفاده نکنیم، مقدار قدیمی برای رجیستر ۲ را با رجیستر ۷ جمع خواهیم کرد که نتیجه‌ی نادرست ۱۹۲ به ما میدهد، ولی با انتقال مقدار درست این رجیستر از قسمت WB به قسمت EXE، می توانیم به نتیجه‌ی مطلوبمان که همان ۱۹۷ است برسیم.

دستور خط پنجم، پرشی بدون قید و شرط به خط ۱۰ انجام خواهد داد، باید دستور های خط ۶ و ۷ را که بعد از دستور ۵ وارد پردازنده کرده ایم را فلاش کنیم، دقت کنید که اگر سازوکار مناسب برای فلاش کردن در نظر نگرفته باشیم، دو دستور بعدی به صورت کامل اجرا خواهند شد و مقدار های رجیستر های ۵ و ۶ را به ترتیب برابر ۲۱ و ۲۷ خواهند کرد، اما اگر انجام نشوند با مقدار رجیستر های نام برده شده برابر ۰ باقی بماند.

دستور خط ۱۱ ام، برای بررسی پرش درست مورد استفاده قرار گرفته است، و مقدار رجیستر ۷ را برابر با ۷ قرار خواهد داد.

ما در زیر، صرفا خروجی های Register File را بررسی می کنیم تا درستی کارکرد پردازنده را نشان دهیم:



نمودار بالا، نتایج ذکر شده‌ی مورد انتظار بالا را برآورده کرده است، در ضمن نمودار ها به ترتیب مقدار های رجیستر های ۰ تا ۷ را نشان می‌دهند. بنابراین پردازنده به درستی Forwardingو Flush و Stall را مدیریت می کند.