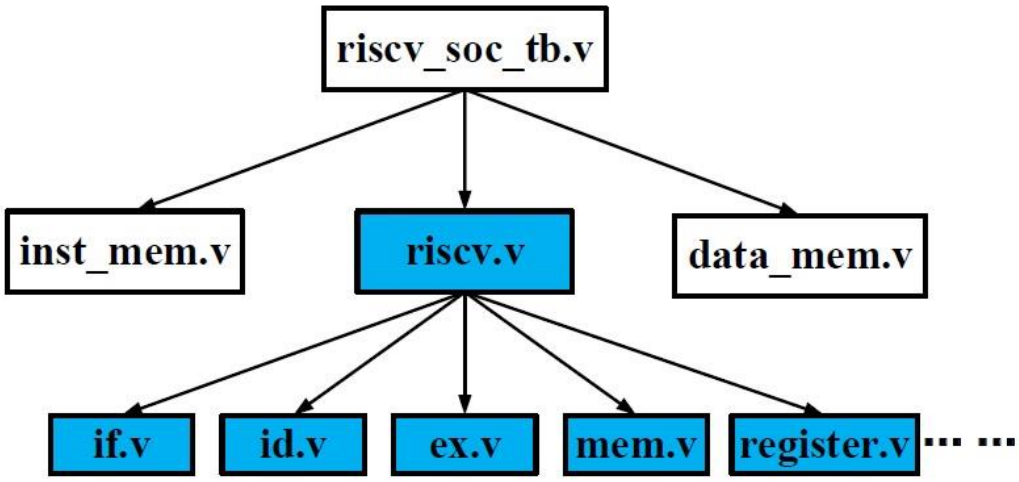
برای افزایش سرعت اجرای انواع دستورها از معماری Pipelined بهره میبریم.

در این پیاده سازی CPU به ۵ بخش تقسیم می شود :

۱. Instruction Fetch
۲. Instruction Decode
۳. Execution
٤. Memory Access
۵. Register Access



همینطور با استفاده از ماژول staller.v میتوان برای جلوگیری از ایجاد هازارد با ایجاد stall، از ایجاد آن جلوگیری کرد :

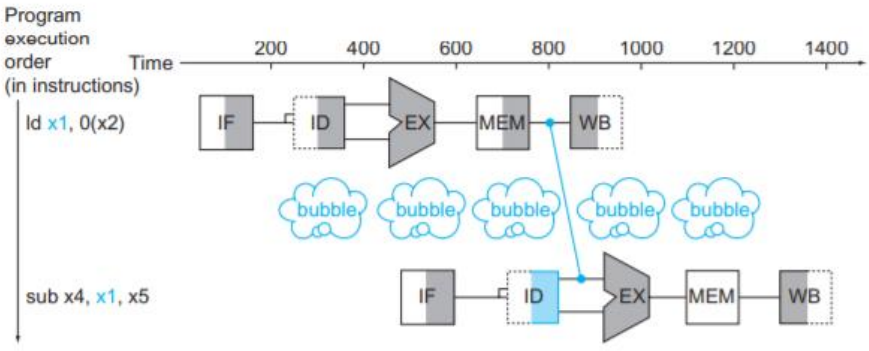


FIGURE 4.28 We need a stall even with forwarding when an R-format instruction following a load tries to use the data. Without the stall, the path from memory access stage output to execution stage input would be going backward in time, which is impossible. This figure is actually a simplification, since we cannot know until after the subtract instruction is fetched and decoded whether or not a stall will be necessary. Section 4.7 shows the details of what really happens in the case of a hazard.

در CPU پیاده سازی شده با Verilog دستور های زیر قابل انجام است :

Instruction	Opcode	Funct۳	Funct۷
add	۰۱۱۰۰۱۱	۰۰۰	۰۰۰۰۰۰۰
addi	۰۰۱۰۰۱۱	۰۰۰	
sub	۰۱۱۰۰۱۱	۰۰۰	۰۱۰۰۰۰۰
and	۰۱۱۰۰۱۱	۱۱۱	۰۰۰۰۰۰۰
or	۰۱۱۰۰۱۱	۱۱۰	۰۰۰۰۰۰۰
xor	۰۱۱۰۰۱۱	۱۰۰	۰۰۰۰۰۰۰
blt	۱۱۰۰۱۱۱	۱۰۰	
beq	۱۱۰۰۱۱۱	۰۰۰	
jal	۱۱۰۱۱۱۱		
sll	۰۱۱۰۰۱۱	۰۰۱	۰۰۰۰۰۰۰
srl	۰۱۱۰۰۱۱	۱۰۱	۰۰۰۰۰۰۰
lw	۰۰۰۰۰۱۱	۰۱۰	
sw	۰۱۰۰۰۱۱	۰۱۰	۰۰۰۰۰۰۰

دیاگرام نهایی CPU نیز به شکل زیر خواهد بود :

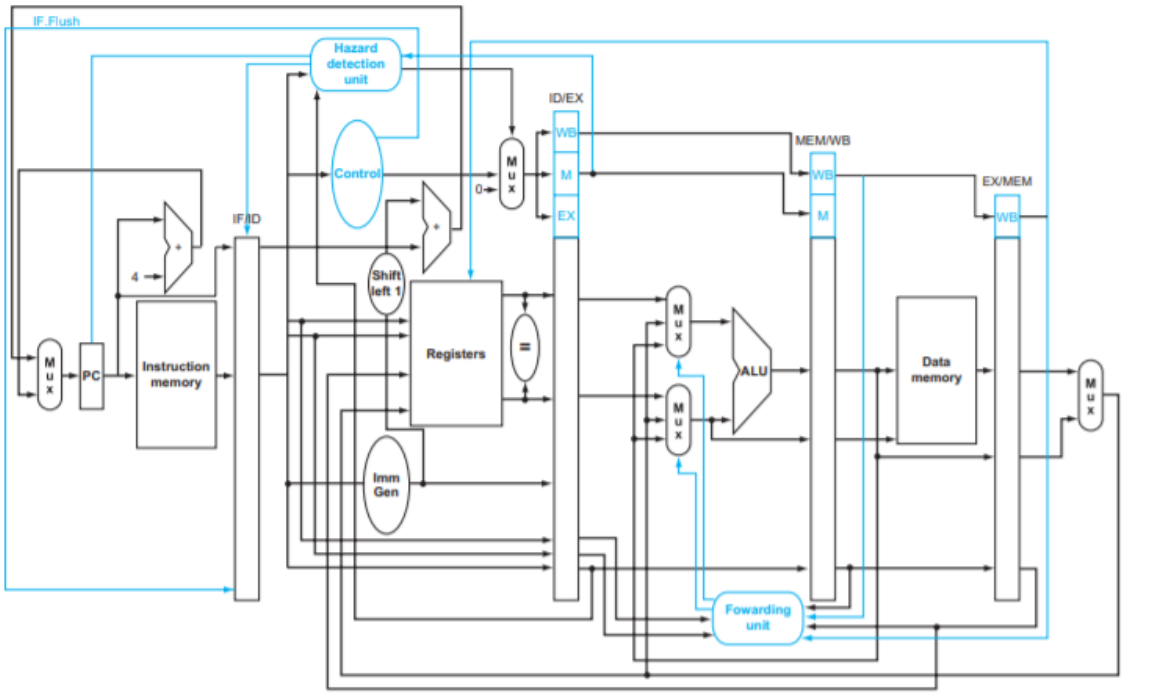


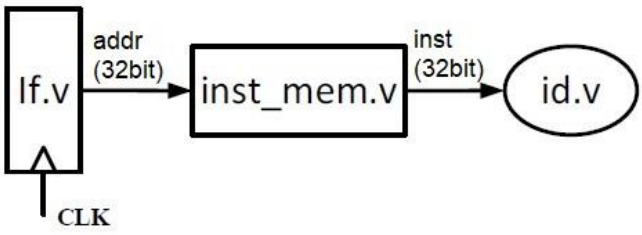
FIGURE 4.62 The final datapath and control for this chapter. Note that this is a stylized figure rather than a detailed datapath, so it's missing the ALUsrc Mux from Figure 4.55 and the multiplexor controls from Figure 4.49.

در قسمت پایین به توضیح مرحله به مرحله اجرای دستور ها در ماژول ها میپردازیم :

۱- If.v

این ماژول ProgramCounter را هر بار بعلاوه ی ۴ میکند تا دستور بعدی اجرا شود. اما اگر دستور یکی از سه دستور beq, blt , jal باشد، PC را سریعا آپدیت میکند و به ادرسی که در دستور به دست آمده تغییر میدهد.

سپس PC به ماژول inst_mem (Instruction Memory) بعنوان ادرس دستور فرستاده میشود و بعد از آن به ماژول استخراج دستور (id.v) فرستاده میشود.



Id.v -۲

در این ماژول دستوری که از inst_mem.v گرفته شده است را دریافت و از آن funct۳ , ALUop و funct۷ را استخراج میکنیم.

در این ماژول نیاز به خواندن یا نوشتن در رجیستر ، نیاز به پرش در دستور های branch و انجام Sign Extension برای اعداد به دست آمده از دستور ها در دستور های شامل immediate .

بعد مشخص شدن اینکه آیا رجیسترها نیاز به خواندن دارند یا خیر ، شماره های رجیستر به دست آمده از دستور را به Register File بعنوان ورودی های rs۲ , rs۱ میدهیم.

همینطور بعد از مشخص شدن نیاز به نوشتن در رجیستر ، دیتای به دست آمده را به rd میدهیم.

جدول زیر انواع اعداد به دست آمده برای انواع دستورها را نشان میدهد :

Instruction	Opcode	Funct۳	Funct۷	ALUop
add	۰۱۱۰۰۱۱	۰۰۰	۰۰۰۰۰۰۰	۰۰۰۰۰۱
sub	۰۱۱۰۰۱۱	۰۰۰	۰۱۰۰۰۰۰	۰۰۰۰۱۰
sll	۰۱۱۰۰۱۱	۰۰۱	۰۰۰۰۰۰۰	۰۰۰۰۱۱
jal	۱۱۰۱۱۱۱			۰۰۰۱۰۰

۳- Ex.v

در این قسمت از اطلاعات به دست آمده درقسمت قبل از دستور استفاده میشود تا دستور اجرا شود.

برای مثال اگر ALUop مشخص کند که دستور نیاز به جمع دارد، اعداد با هم جمع میشوند و اگر تفریق باشد با استفاده از مکمل این عمل انجام میشود.

Operation	ALUop
+	۰۰۰۰۰۱
<< (shift left)	۰۰۰۰۱۱
-	۰۰۰۰۱۰

٤- Mem.v

این ماژول خواندن و نوشتن در Data Memory را کنترل میکند.

اگر دستور lw باشد دیتا (۳۲ بیت) از data_mem (Data Memory) خوانده میشود.

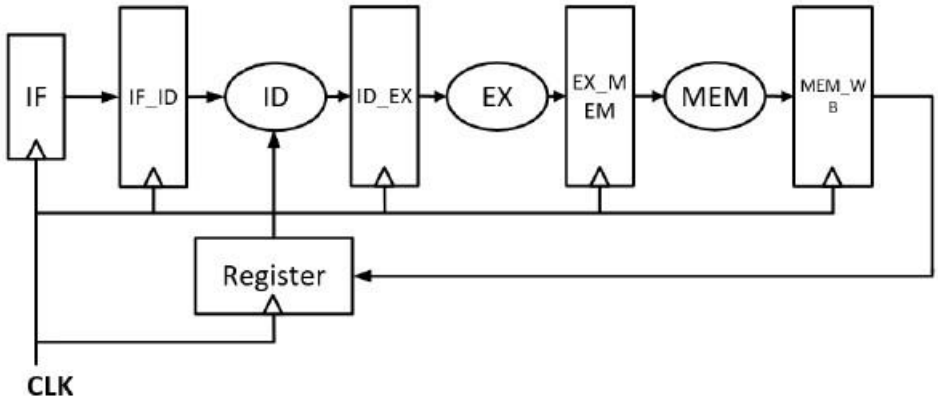
و اگر دستور sw باشد، دیتا داخل data_mem نوشته میشود.

۵- Register.v

شامل رجیسترهای CPU می باشد.

با توجه به اطلاعات به دست آمده در مراحل قبل، اگر نیاز به خواندن از رجیستر ها باشد با توجه به ورودی rs۱ , rs۲ که شامل شماره رجیسترها هستند، عملیات خواندن انجام میشود. همینطور اگر دستور شامل نوشتن در رجیستر بود، دیتای آن از rd به دست می آید.

اگر هر دو سیگنال خواندن و نوشتن در رجیستر موجود باشند، برای جلوگیری از ایجاد هازاردها اگر هر دو شماره رجیستر برای خواندن و نوشتن یکی باشد، دیتایی که برای نوشته شدن وارد ماژول شده بعنوان خروجی از آن خارج میشود (Data Forwarding).



۶- Stall.v

اگر دیتا نتواند توسط دیتا فورواردینگ به دست بیاید، اگر دیتا در دستور بعدی نیاز باشد باعث به وجود آمدن هازارد خواهد شد. پس نیاز به یک وقفه در اجرای دستور نیاز خواهد بود.

این ماژول وظیفه ایجاد این وقفه ها در زمان نیاز را بر عهده دارد.

۷- If_id.v

مداری ترتیبی بین PC و ID است. وظیفه فرستادن PC و دستور را برعهده دارد. اگر Pipeline متوقف شده بود ، ماژول این دو مقدار را نگه میدارد.

و اگر دستور شامل branch باشد، این دو مقدار را پاک میکند.

۸- Id_ex.v

مداری ترتیبی است که در بین ID و ex قرار دارد. وظیفه فرستادن سیگنال های به دست آمده در Instruction Decode را دارد. اگر Pipeline متوقف شده باشد این سیگنال ها بدون تغییر خواهند ماند.

۹- Ex_mem.v

مدار ترتیبی بین Ex و Mem است که سیگنال های به دست آمده در ex.v , ادرس رجیستر های مورد نظر , سیگنال نوشتن در رجیستر و سیگنال های دیگر را به ماژول بعدی میفرستد.

اگر Pipeline متوقف شده باشد این سیگنال ها بدون تغییر خواهند ماند.

۱۰- Mem_wb.v

مداری ترتیبی بین Mem و WB است که دیتای نوشتن در رجیستر ، ادرس رجیستر مقصد و سیگنال های دیگر میفرستد.