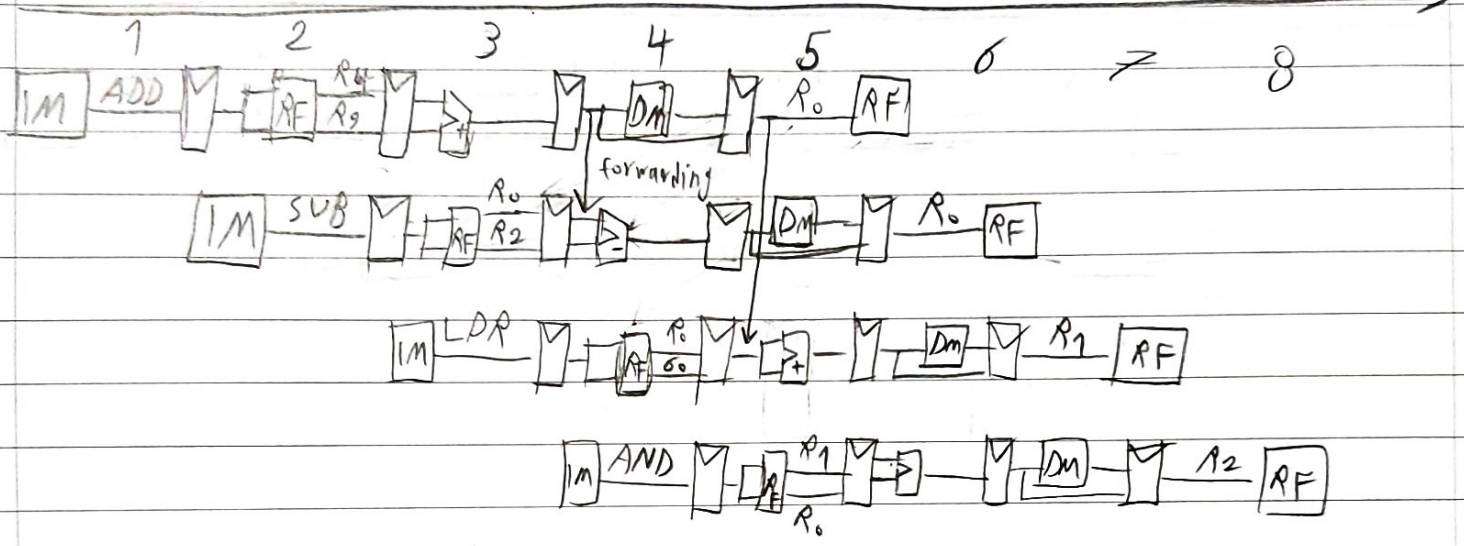
(1

ADD R_0, R_4, R_7

SUB R_1, R_0, R_2

LDR $R_1, [R_0, \#60]$

AND R_2, R_1, R_0



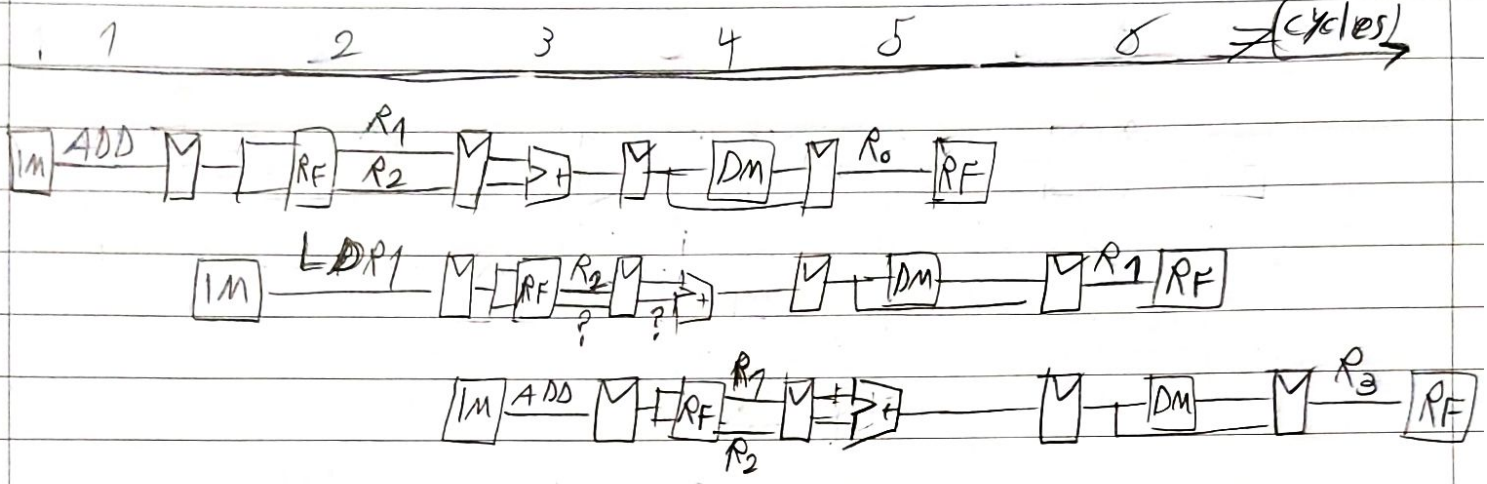
دستور ADD باید R_0 را تولید کند تا در SUB از آن استفاده شود. پس هنگامی که در استیج Memory یا Writeback است forward می کنیم به استیج Execution در دستور SUB، همچنین به دستور LDR.

حال در LDR برای controll Hazard نیاز به stall داریم پس AND در سایکل 5 و 6 stall می شود تا مقدار R_1 در سایکل 7 مشخص شود تا به استیج Execute در دستور AND، forward شود.

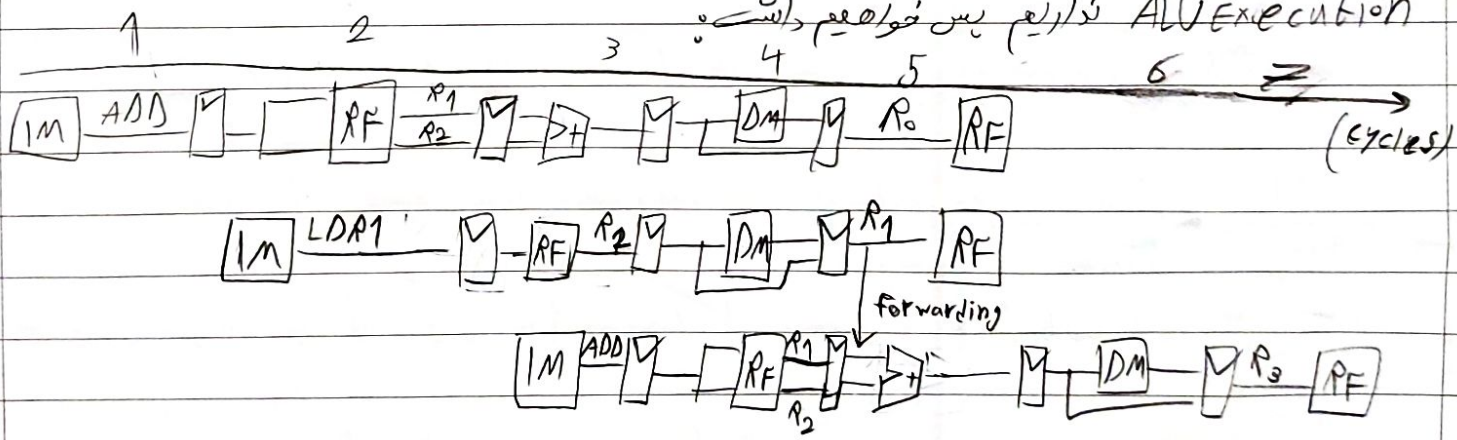
ADD R_0, R_1, R_2

LDR1 R_1, R_2

ADD R_3, R_1, R_2



چون دستورات قبل از LDR1 دستوری از نوع LDR1 نیستند و دستوراتی که از این جنس نیستند نیازی به DataMemory در مرحله memory ندارند می‌توانیم آن را حذف کنیم و در دستور LDR1 نیز چون برای آدرس دهی فقط یک رجیستر داریم دیگر نیازی به بلوک ALUExecution نداریم پس خواهیم داشت:



همان طور که مشخص است در سایکل 4 مقدار آدرس R_2 در R_1 قرار داده می‌شود و در Hazard ، ADD با write back کردن مقدار R_1 به Execution در اینسترکشن forward کنترل می‌شود و دیگر نیازی به stall نیست.

برای تغییر در پردازنده ابتدا یک سیگنال LDR1 در control پردازنده ایجاد می‌کنیم که اگر صفر باشد LDR1 و اگر 1 باشد LDR عادی فعال می‌شود.
سپس نیاز به سه MUX 2:1 داریم که به این صورت عمل می‌کند:
اگر سیگنال 1 باشد به صورت قبل عمل می‌کند ولی اگر صفر باشد source A وارد MUX می‌شود و خروجی آن وارد MUX دوم می‌شود که خروجی آن نیز وارد Data Memory می‌شود و پس از آن با گرفتن یک اشتباه از به MUX سوم وصل می‌کنیم تا پردازنده به کار خود ادامه دهد ولی با این تفاوت که وقتی نیازی به ALU Execution نباشد دیگر stall رخ نمی‌دهد.