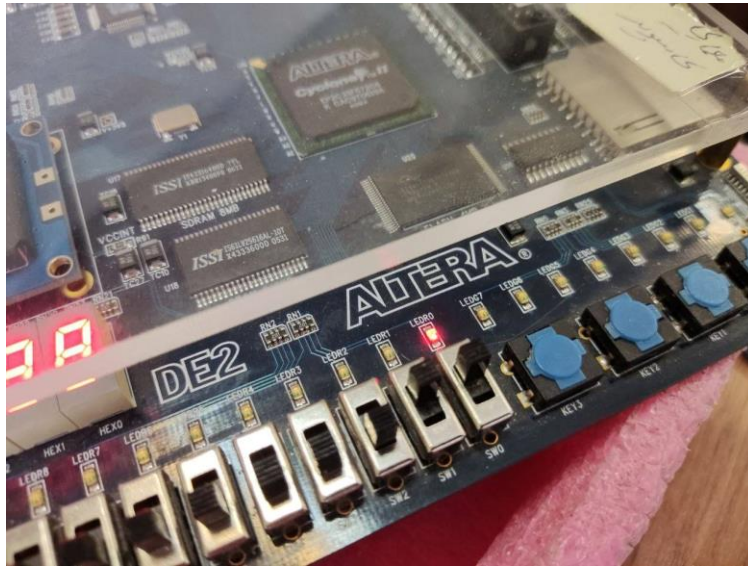


گزارش پایانی آزمایشگاه معماری کامپیوتر

علی حسینی اخوان - 810199406

محمد مشرقی - 810199492

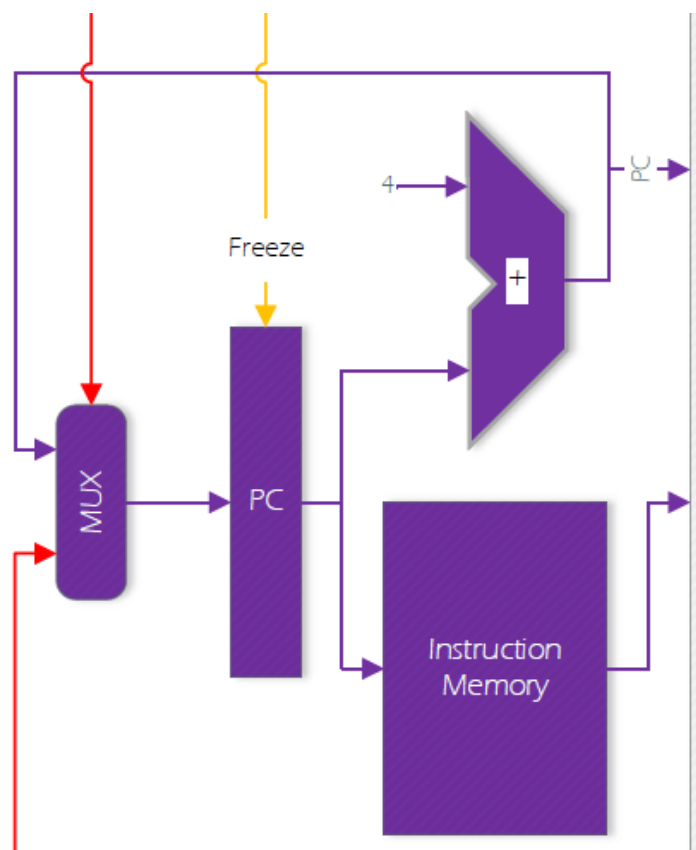
بخش اول) LED روشن :



```
assign LEDR[0] = SW[0] | SW[1];
```

### بخش IF :

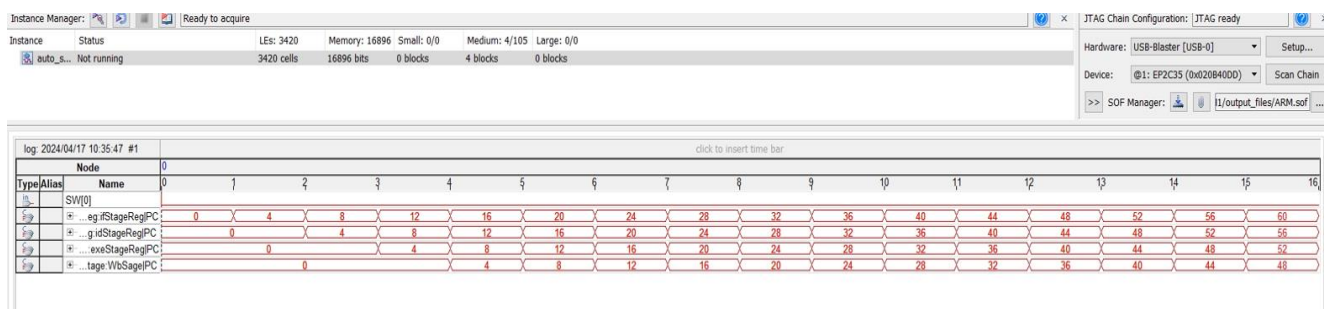
در این مرحله، دستورالعمل‌ها از حافظه دستورالعمل (Instruction Memory) بارگذاری می‌شوند و مقدار شمارنده برنامه (PC) تغییر می‌یابد تا به آدرس بعدی که باید اجرا شود، برسیم. شمارنده برنامه (PC Register) آدرس لازم را برای حافظه دستورالعمل (Instruction Memory) فراهم می‌کند تا دستورالعمل مورد نظر را دریافت کند. افزونگر (PC Adder) عدد ۴ را به مقدار فعلی PC اضافه می‌کند تا آدرس دستورالعمل بعدی تهیه شود. در صورتی که دستورالعمل موجود یک دستور شاخه‌ای (Branch) باشد، آدرس شاخه توسط یک مالتیپلکسر (Multiplexer) انتخاب می‌شود تا شمارنده برنامه (PC) در چرخه بعدی به‌روزرسانی شود. این مرحله شامل اجزایی مانند حافظه دستورالعمل (Instruction Memory)، شمارنده برنامه (PC Register)، افزونگر PC (PC Adder) و مالتیپلکسر آدرس برنامه (Program Address Mux) است که اتصالات آن‌ها به یکدیگر طبق شکل زیر می‌باشد:



مشاهده می‌شود که برای هر بخش، قسمتی مجزا در نظر گرفته شده است و بر اساس توضیحات ارائه شده پیش از این، کد مرتبط با هر بخش نگاشته شده است. جالب است که اگر شرایط خاصی مانند فرمان شاخه‌ای (Branch) پیش آید، مالتیپلکسر ورودی دوم خود را که با نام BranchAddr شناخته می‌شود، بر روی خروجی قرار می‌دهد و در غیر این صورت، مقدار فعلی PC را قرار می‌دهد که در ادامه، به آن عدد ۴ افزوده می‌شود و همین امر سبب ادامه چرخه می‌گردد. در قسمت حافظه دستورالعمل (Instruction Memory) نیز شاهد این هستیم که کد ۳۲ بیتی هر دستور با توجه به مقادیر متفاوت PC نوشته شده است که بر اساس مقدار PC، آن دستور اجرا می‌شود.

برای فهمیدن توضیحات لطفاً به کد بخش IF مراجعه کنید.

## نتایج:



## بخش ID:

در مرحله‌ی تفسیر دستورات (Instruction Decode)، اطلاعات از دستور استخراج شده و سیگنال‌های کنترلی مورد نیاز برای مراحل بعدی تولید می‌شوند. در این مرحله، فرایندهای زیر به صورت خلاصه و نکته‌وار انجام می‌پذیرد:

1. واحد کنترل (Control Unit): بر اساس کد عملیاتی (opcode) استخراج شده از دستور، فرمان اجرایی مورد نیاز برای واحد ALU در مرحله‌ی اجرا (Execution Stage) تولید می‌شود. این opcode در مراحل بعدی دیگر نیازی نیست.

آورده شده است. دستور NOP به عنوان یک دستور پیاده‌سازی نمی‌شود.

R-type Instructions	Description	Bits							
		31:28	27:26	25	24:21	20	19:16	15:12	11:00
		Cond.	Mode	I	OP-Code	S	Rn	Rd	shifter operand
0	NOP <sup>11</sup>	No Operation	1110	00	0	0000	0	0000	000000000000
1	MOV	Move	cond	00	I	1101	S	0000	Rd
2	MOVN <sup>12</sup>	Move NOT	cond	00	I	1111	S	0000	Rd
3	ADD	Add	cond	00	I	0100	S	Rn	Rd
4	ADC	Add with Carry	cond	00	I	0101	S	Rn	Rd
5	SUB	Subtraction	cond	00	I	0010	S	Rn	Rd
6	SBC	Subtract with Carry	cond	00	I	0110	S	Rn	Rd
7	AND	And	cond	00	I	0000	S	Rn	Rd
8	ORR	Or	cond	00	I	1100	S	Rn	Rd
9	EOR	Exclusive OR	cond	00	I	0001	S	Rn	Rd
10	CMP	Compare	cond	00	I	1010	1	Rn	0000
11	TST <sup>13</sup>	Test	cond	00	I	1000	1	Rn	0000
12	LDR	Load Register	cond	01	0	0100	1	Rn	Rd
13	STR	Store Register	cond	01	0	0100	0	Rn	Rd
14	B	Branch	cond	10	1	0	signed_immed_24		

جدول 2- لیست دستورهای پردازنده

2. تنظیم سیگنال‌های کنترلی:

- برای دستورات شاخه‌ای (Branch)، سیگنال کنترلی B به 1 تنظیم می‌شود.
- برای دستور LDR، سیگنال MEM\_R\_EN به 1 تنظیم می‌گردد تا امکان خواندن از حافظه فراهم شود.
- برای دستور STR، سیگنال MEM\_W\_EN به 1 تنظیم می‌شود تا امکان نوشتن در حافظه مهیا شود.

- سیگنال WB\_EN برای تمام دستورات به جز CMP و TST به 1 تنظیم می‌شود، زیرا این دو دستور هیچ مقداری را به Register File باز نمی‌گردانند.

### 3. بروزسانی Register File:

- بر اساس نتیجه ALU در لبه‌ی نزولی کلاک، مقادیر به‌روز می‌شوند.

- مقادیر وضعیت‌های (N (Negative), V (Overflow), C (Carry), B (Branch) در Register File ذخیره شده و برای بررسی شرایط مورد استفاده قرار می‌گیرند تا تعیین شود که آیا همه سیگنال‌های کنترلی باید صفر شوند یا خیر.

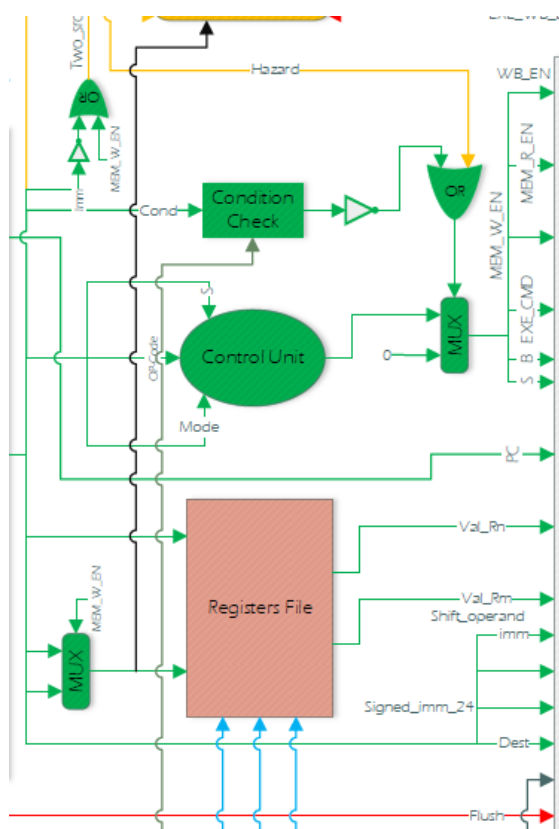
### 4. پورت‌های Register File:

- این فایل شامل 16 رجیستر است و دارای دو پورت خواندن و یک پورت نوشتن می‌باشد.

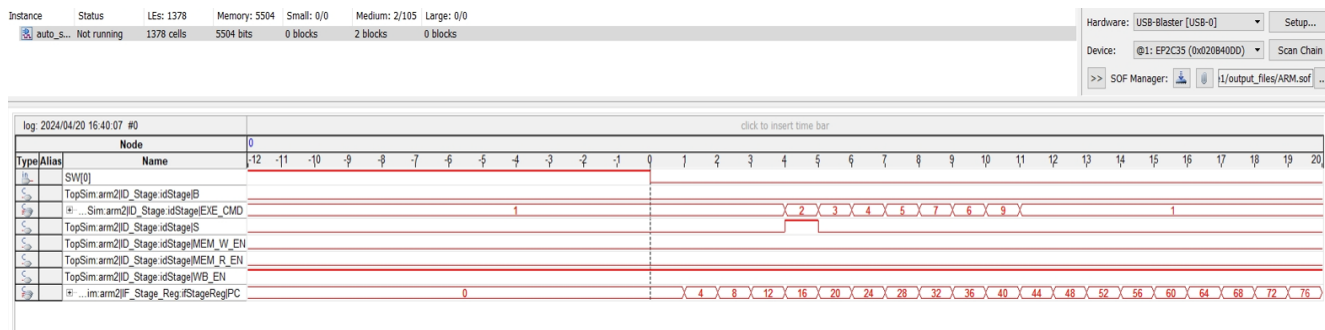
- نوشتن در Register File به صورت همزمان با کلاک در لبه پایینی و خواندن به صورت آسنکرون انجام می‌شود.

### 5. بررسی شرایط (Condition Check):

- بر اساس مقدار Cond، خروجی Condition Check تعیین می‌شود که این امر نشان‌دهنده‌ی ضرورت اجرای دستور یا عدم اجرای آن است.

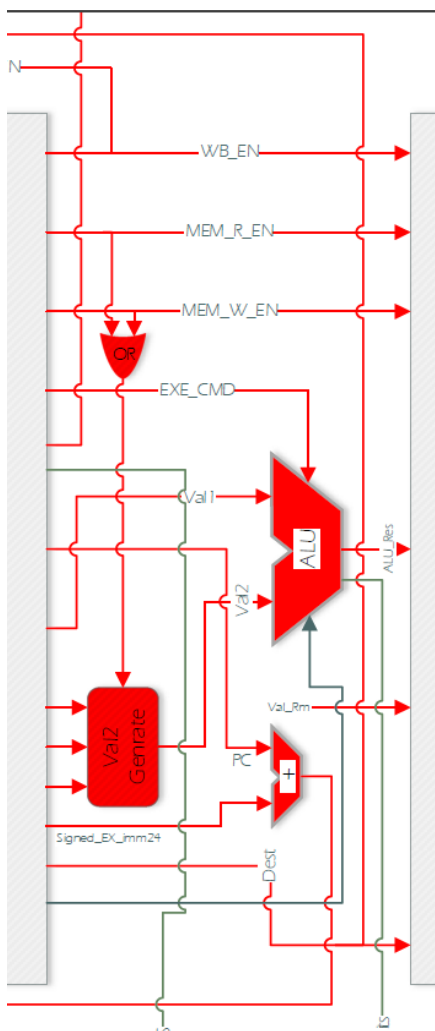


## نتایج:



## بخش EXE:

در مرحله اجرا (EXE Stage) پردازشگر ARM، واحد حساب و منطق (ALU) نقش کلیدی ایفا می‌کند. ALU دو ورودی دریافت می‌کند و بر اساس فرمان‌های اجرایی (EXE\_CMD)، خروجی‌های مربوطه از جمله carry out و فلگ‌های حالت مختلف (N: منفی، V: سرریز، Z: صفر) را تولید می‌کند که به Status Register منتقل می‌شوند که داریم:



## 1. ALU (واحد حساب و منطق):

- عملیات‌ها: ALU بر اساس دستورالعمل‌هایی که از فاز تفسیر دستورات (ID Stage) دریافت می‌کند، عملیات‌های مختلفی را اجرا می‌کند. این عملیات‌ها می‌توانند شامل جمع، تفریق، و دیگر دستورات منطقی باشند.

- تولید فلگ‌ها: ALU همچنین فلگ‌های حالت مانند N, Z, C, و V را تولید می‌کند که بر اساس نتایج عملیات‌های انجام شده بر روی داده‌ها می‌باشند.

Instruction	ALU Command	Operation
MOV	0001	result = in2
MVN	1001	result = ~in2
ADD	0010	result = in1 + in2
ADC	0011	result = in1 + in2 + C
SUB	0100	result = in1 - in2
SBC	0101	result = in1 - in2 - ~C
AND	0110	result = in1 & in2
ORR	0111	result = in1   in2
EOR	1000	result = in1 ^ in2
CMP	0100	result = in1 - in2
TST	0110	result = in1 & in2
LDR	0010	result = in1 + in2
STR	0010	result = in1 + in2
B	XXXX	

جدول 5- ریز دستوره‌ای واحد حساب و منطق

## 2. Val2 Generator (تولیدکننده Val2):

- این ماژول مسئول تولید دومین عملوند برای ALU است. Val2 می‌تواند بر اساس مقادیر imm (مقدار فوری)، Shift\_operand (عملوند شیفت)، و Val\_Rm (مقدار از رجیستر Rm) تعیین شود.

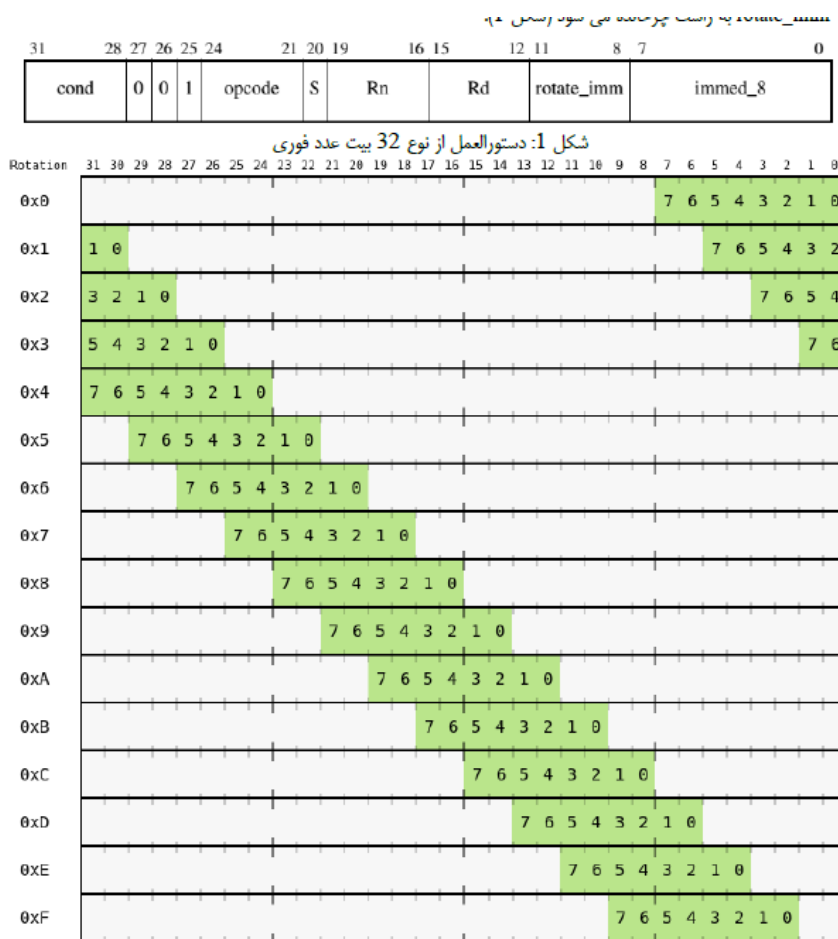
Rd: نشاندهنده آدرس ثبات مقصد است. این آدرس در دستور STR به عنوان یکی از مقادیری که باید در حافظه ذخیره شود مورد استفاده قرار می‌گیرد.

Rn: همواره به عنوان یکی از عملوندهای دستورات مورد استفاده قرار می‌گیرد.

shifter operand: برای عملوند شیفت در پردازنده ARM به سه شکل زیر پیاده‌سازی شده است:

1- 32 بیت عدد فوری (32-bit immediate):

در این حالت مقدار بیت ۱ برابر یک است. عدد ۸ بیتی imm8 در یک ظرف ۳۲ بیت قرار می‌گیرد سپس به اندازه دو برابر rotate\_imm به راست چرخونده می‌شود:



شکل ۲: نحوه چرخش عدد فوری

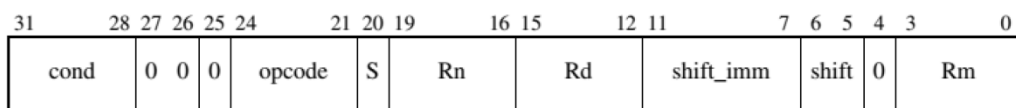
## ۲- شیفت فوری (Immediate shifts):

در این حالت بیت ۱ و بیت چهارم دستورالعمل نیز صفر برابر صفر است. عملوند دوم از رجیستر خوانده می‌شود. سپس عدد خوانده شده

براساس حالت shift به مقدار shift\_imm شیفت داده می‌شود حالت های شیفت در جدول زیر قرار دارد.

مقدار	توضیحات	وضعیت شیفت
00	Logical shift left	LSL
01	Logical shift right	LSR
10	Arithmetic shift right	ASR
11	Rotate right	ROR

جدول ۴- وضعیت شیفت در دستورات شیفت فوری



شکل ۳: دستورالعمل از نوع شیفت فوری



### 3. Status Register (رجیستر وضعیت):

- تنظیم فلگ‌ها: در صورتی که سیگنال S فعال باشد (یعنی نیاز به ذخیره‌سازی فلگ‌ها در رجیستر وضعیت داشته باشیم)، فلگ‌های تولید شده توسط ALU در Status Register ذخیره می‌شوند. در غیر این صورت، این فلگ‌ها نادیده گرفته می‌شوند.

Opcode [31:28]	Mnemonic extension	Meaning	Condition flag state
0000	EQ	Equal	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set/unsigned higher or same	C set
0011	CC/LO	Carry clear/unsigned lower	C clear
0100	MI	Minus/negative	N set
0101	PL	Plus/positive or zero	N clear
0110	VS	Overflow	V set
0111	VC	No overflow	V clear
1000	HI	Unsigned higher	C set and Z clear
1001	LS	Unsigned lower or same	C clear or Z set
1010	GE	Signed greater than or equal	N set and V set, or N clear and V clear ( $N == V$ )
1011	LT	Signed less than	N set and V clear, or N clear and V set ( $N != V$ )
1100	GT	Signed greater than	Z clear, and either N set and V set, or N clear and V clear ( $Z == 0, N == V$ )
1101	LE	Signed less than or equal	Z set, or N set and V clear, or N clear and V set ( $Z == 1$ or $N != V$ )
1110	AL	Always (unconditional)	-
1111	-	See Condition code 0b1111	-

جدول 3- کد شرط دستورات

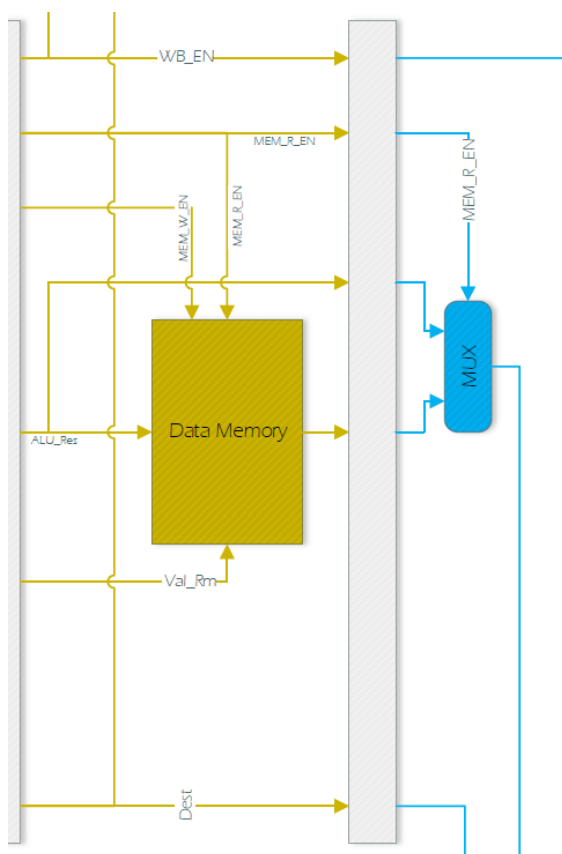
### 4. محاسبه آدرس شاخه‌ای (Branch Address Calculation):

- آدرس‌های شاخه‌ای بر اساس مقدار PC فعلی و مقادیر امضا شده‌ای که به آن اضافه می‌شوند (Signed\_imm)، محاسبه می‌گردند.

### نتایج:

[illegible]

تکمیل ARM (مرحله حافظه و بازنویسی):



در مرحله حافظه (Memory Stage) پردازنده ARM، داده‌ها از یک حافظه RAM شبیه‌سازی شده خوانده یا در آن نوشته می‌شوند. این عملیات تحت کنترل سیگنال‌های MEM\_R\_EN برای خواندن و MEM\_W\_EN برای نوشتن انجام می‌پذیرد. این سیگنال‌ها توسط واحد کنترل (Control Unit) تولید شده و همراه با دستورالعمل‌ها در پایپ‌لاین به جلو منتقل می‌شوند.

1. آغاز حافظه و آدرس‌دهی:

- حافظه داده از آدرس 1024 شروع می‌شود و دسترسی به حافظه بر اساس بایت امکان‌پذیر نیست؛ دسترسی‌ها بر اساس ۳۲ بیت یا ۴ بایت انجام می‌گیرد.

- بدین ترتیب، خواندن یا نوشتن فقط از آدرس‌هایی که مضربی از ۴ هستند، ممکن است. مثلاً خواندن از آدرس‌های 1024, 1028, ... امکان‌پذیر بوده و هر خواندن، ۳۲ بیت داده را بازمی‌گرداند.

2. عملیات خواندن و نوشتن:

- در صورت فعال بودن سیگنال MEM\_R\_EN، داده‌ها از حافظه خوانده می‌شوند و در صورت فعال بودن MEM\_W\_EN، داده‌ها در حافظه نوشته می‌شوند.

- این سیگنال‌ها توسط واحد کنترل بر اساس نوع دستور (حافظه‌ای یا محاسباتی) تنظیم می‌شوند.

### 3. مرحله بازنویسی (Write-Back Stage):

- در این مرحله، سیگنال WB\_EN فعال شده و داده خوانده شده از حافظه یا مقدار محاسبه شده توسط ALU به ثبات‌های مقصد نوشته می‌شود.

- این فرایند اطمینان می‌دهد که نتایج عملیات‌های قبلی برای استفاده در دستورات بعدی در دسترس هستند.

## نتایج:

Node		0
Type	Alias	Name
File	...	ter_file/reg_file[0]
File	...	ter_file/reg_file[1]
File	...	ter_file/reg_file[2]
File	...	ter_file/reg_file[3]
File	...	ter_file/reg_file[4]
File	...	ter_file/reg_file[5]
File	...	ter_file/reg_file[6]
In		SWI01

در نهایت پس از تکمیل بخش های write back و memory خروجی بصورت فوق است.

**بخش FORWARDING و Hazard:**

در پردازنده‌ها، سه نوع اصلی مخاطره وجود دارد که می‌توانند بر عملکرد و توالی اجرای دستورات تأثیر بگذارند:

الف) مخاطره ساختاری:

این نوع مخاطره مربوط به محدودیت‌های سخت‌افزاری در پردازنده است، مانند تعارض در دسترسی به منابع سخت‌افزاری. به‌طور مثال، اگر دو فاز مختلف (مانند WB و ID) همزمان بخواهند از یک قسمت خاص از سخت‌افزار مانند ثبت‌های عمومی استفاده کنند، مخاطره ساختاری رخ می‌دهد. برای رفع این نوع مخاطره، ممکن است عملیات نوشتن در ثبت‌ها به لبه پایین‌رونده کلاک منتقل شود تا از تداخل با مراحل دیگر جلوگیری شود.

(ب) مخاطره کنترلی:

این مخاطره زمانی رخ می‌دهد که تصمیم‌گیری در مورد شاخه‌های برنامه به تأخیر بیفتد. مثلاً، اگر یک دستور شاخه پیش از محاسبه و تأیید آدرس شاخه در خط لوله وارد شود، ممکن است دستورات نادرستی به خط لوله وارد شوند. برای مقابله با این مخاطره، معمولاً از سیگنال‌های Flush استفاده می‌شود تا دستورات نامربوط از خط لوله پاک شوند.

که هر دو آنها رفع شده و نیاز به حل این دو نمی باشد.

(ج) مخاطره داده‌ای:

این نوع مخاطره شامل سه زیرمجموعه است:

1. خواندن پس از نوشتن (Read After Write - RAW): این مخاطره زمانی اتفاق می‌افتد که یک دستور به داده‌ای نیاز دارد که هنوز توسط دستور قبلی نوشته نشده است. برای جلوگیری از این مخاطره، از واحد تشخیص هازارد استفاده می‌شود تا اطمینان حاصل شود که داده‌ها پیش از استفاده، به‌درستی نوشته و آیدیت شده‌اند.

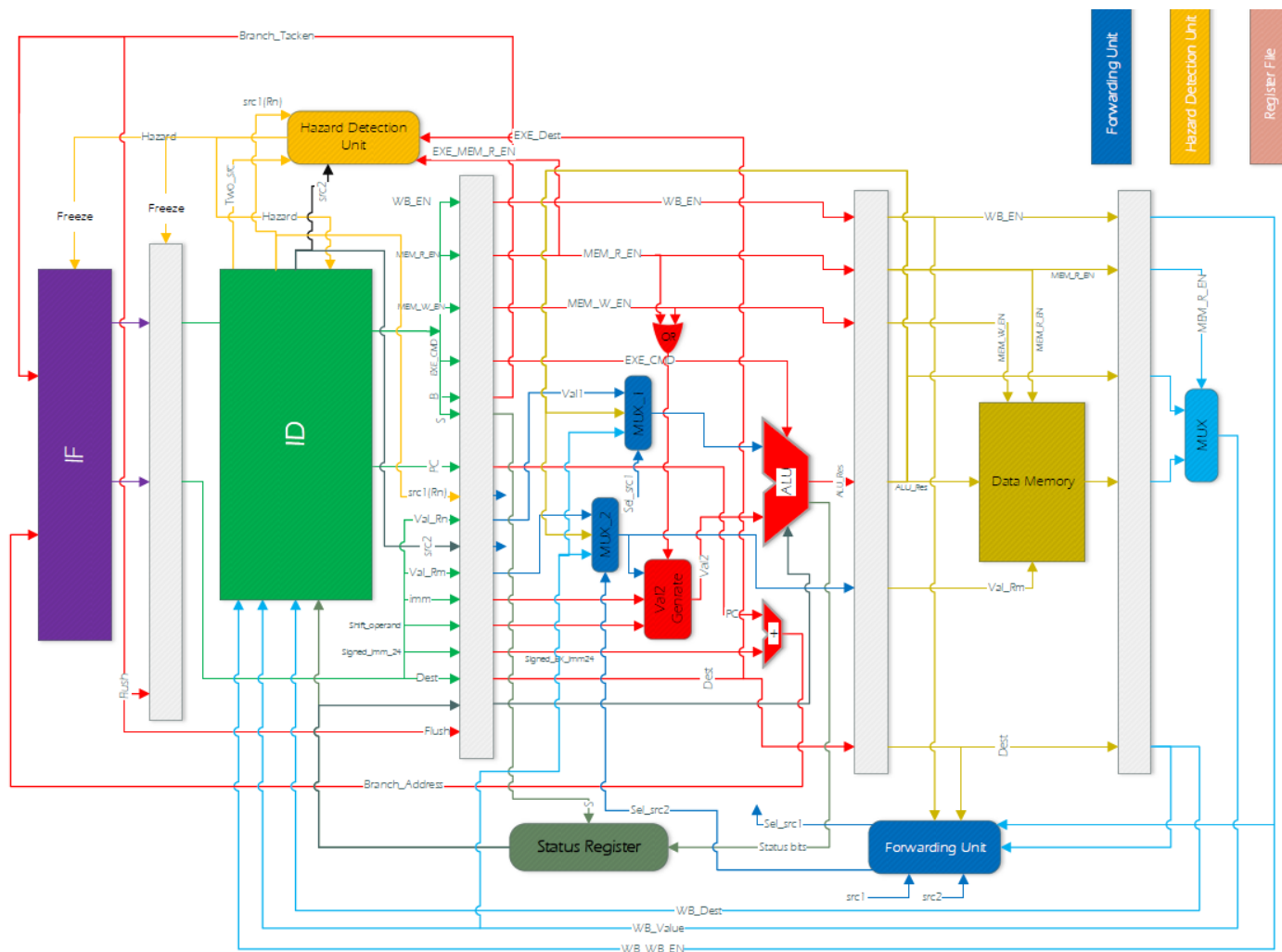
2. نوشتن پس از خواندن (Write After Read - WAR): این مخاطره زمانی رخ می‌دهد که یک دستور می‌خواهد مقدار یک رجیستر را تغییر دهد در حالی که دستور قبلی همان رجیستر را خوانده است. در پردازنده‌هایی که به ترتیب دستورات عمل می‌کنند، این نوع مخاطره کمتر اتفاق می‌افتد.

3. نوشتن پس از نوشتن (Write - WAW Write After): این مخاطره زمانی اتفاق می‌افتد که دو دستور می‌خواهند به طور همزمان به یک رجیستر بنویسند. این مخاطره عمدتاً در پردازنده‌هایی که دستورات را به ترتیب غیرمتوالی اجرا می‌کنند، مهم است.

بدون فوروارڈینگ:

[illegible]

## :Forwarding



### ماژولاش Forwarding Unit

1. تعیین منبع داده‌ها:

- این دو سیگنال مشخص می‌کنند که داده‌های مورد نیاز برای اجرای دستورات از کدام مرحله خط لوله باید به واحد اجرا (EXE stage) منتقل شوند. بر اساس وضعیت اجرای دستورات در مراحل قبلی، این سیگنال‌ها تنظیم می‌شوند تا اطمینان حاصل شود که داده‌های به‌روز در دسترس هستند.

2. انتخاب داده‌ها توسط Mux:

- داده‌ها بر اساس سیگنال‌های sel\_src1 و sel\_src2 از مراحل مختلف خط لوله (مانند ID, EXE, MEM یا WB) گرفته شده و به ورودی‌های EXE stage فرستاده می‌شوند.

3. شرایط مختلف Forwarding:









- اگر یک دستور نیاز به داده‌ای دارد که توسط دستور قبلی نوشته شده و هنوز در مرحله MEM قرار دارد، سیگنال sel\_src برابر با ۱ تنظیم می‌شود.

- اگر داده در مرحله WB است، سیگنال sel\_src برابر با ۲ تنظیم می‌شود.

- در غیر این صورت، هیچ Forwarding ای رخ نمی‌دهد و sel\_src برابر با ۰ خواهد بود.

ماژول Hazard نیز به‌روزرسانی شده است تا با استفاده از Forwarding، مخاطرات احتمالی را شناسایی و مدیریت کند. در شرایطی که Forwarding نتواند پوشش دهد مخاطره داده‌ای را (مثلاً زمانی که دستور مربوط به بارگیری و محاسبه داده‌ها در همان سیکل اتفاق می‌افتد) که تنها زمانی اتفاق می‌افتد در این بخش که بخواهیم از مموری چیزی داخل رجیستر بریزیم و همزمان بخواهیم از همان رجیستر بخوانیم

با فورواردینگ :

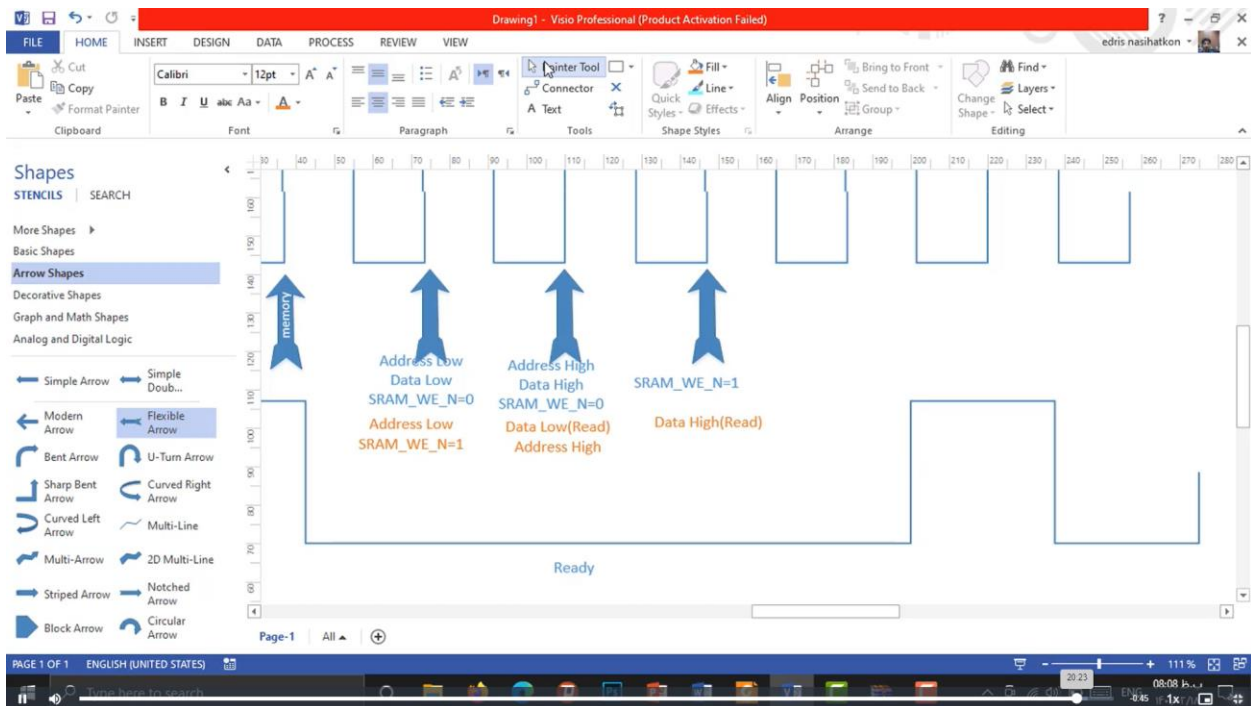
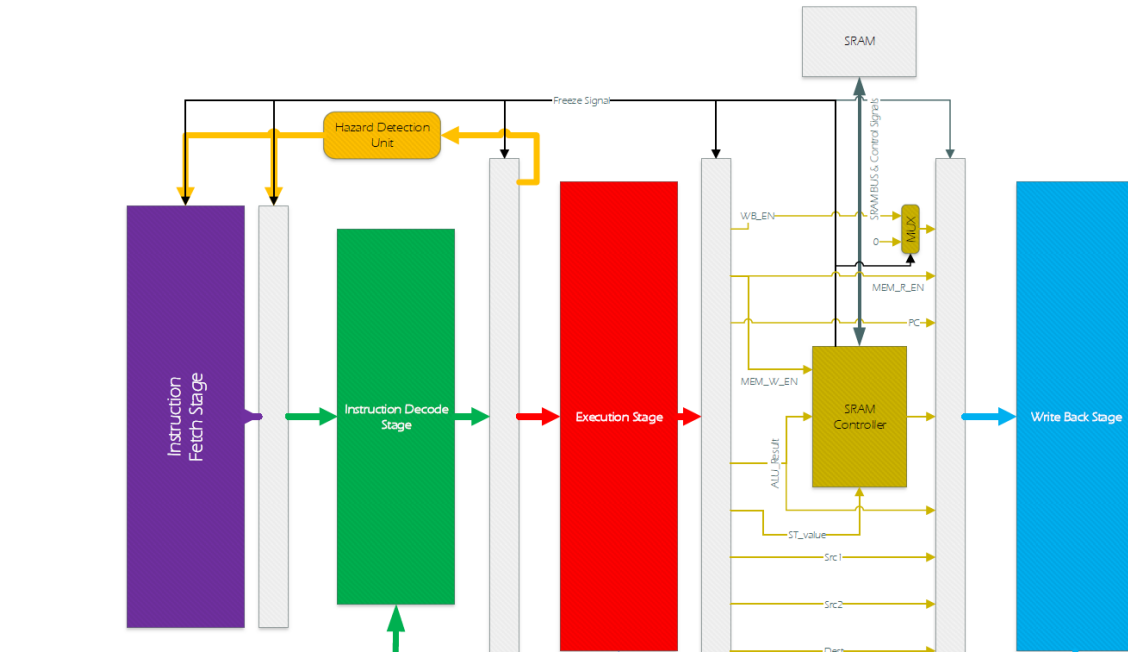
log 2024/05/15 10:09:07 #0			click to insert time bar																	
Node			0																	
Type	Alias	Name	-128	-64	0	64	128	192	256	320	384	448	512	576	640	704	768	832	896	
	#	...ter_flelreg_flel[0]	0		120		1024							20						
	#	...ter_flelreg_flel[1]	1				4							-2147483648						
	#	...ter_flelreg_flel[2]	2			0	1	2	3											
	#	...ter_flelreg_flel[3]	3			10	1	2	3	0	1	2	3							
	#	...ter_flelreg_flel[4]	4			41														
	#	...ter_flelreg_flel[5]	5			-123	8192		141		141		141							
	#	...ter_flelreg_flel[6]	6			10		141	141		141		141							
		SW[0]																		

مشاهده میشود که تعداد کل کلاک ها از حدود 282 به 192 کاهش یافته است که مطلوب است.

بهبود کارایی برابر است با:

$$\frac{282 - 194}{282} = \frac{88}{282} = 0.312 \rightarrow \text{حدود 30 درصد بهبود داریم}$$

طبیعتاً انتظار داریم که با اضافه شدن ماژول forward سخت افزار بیشتری استفاده شود و باعث افزایش هزینه می شود





SRAM (Static Random Access Memory) یک نوع حافظه است که به دلیل سرعت دسترسی بالا و قابلیت اعتماد برای ذخیره‌سازی داده‌ها، عموماً در حافظه‌های کش پردازنده‌ها مورد استفاده قرار می‌گیرد. در این سیستم، کنترلر SRAM وظیفه مدیریت ارتباط بین CPU و حافظه SRAM را بر عهده دارد و به عنوان یک واسطه عمل می‌کند تا امکان خواندن و نوشتن از و به CPU را فراهم آورد. که در ادامه به آن به مازول های آن حرف می زنیم.

### 1. کنترلر SRAM:

- کنترلر SRAM به عنوان یک واسطه بین پردازنده (CPU) و حافظه SRAM عمل می‌کند. این کنترلر مسئولیت هماهنگ‌سازی دسترسی‌های خواندن و نوشتن را بر عهده دارد.

- به منظور دستیابی به کارآمدی بیشتر و کاهش تأخیر، اطلاعات به صورت 16 بیتی خوانده و نوشته می‌شوند، ولی از آنجایی که دسترسی به حافظه SRAM به صورت 32 بیتی (4 بایتی) است، معمولاً برای خواندن یا نوشتن کامل یک کلمه، دو چرخه 16 بیتی لازم است.

### 2. مدیریت تأخیر:

- زمانی که عملیات خواندن یا نوشتن در حافظه SRAM انجام می‌شود، دستورات دیگر CPU باید متوقف شوند تا عملیات حافظه کامل شود. این توقف توسط سیگنال فریز (freeze) کنترل می‌شود که تا پایان عملیات حافظه فعال می‌ماند.

- عملیات خواندن یا نوشتن در حافظه SRAM ممکن است تا 6 سیکل کلاک به طول انجامد، بسته به اینکه داده در حال خوانده شدن یا نوشته شدن است.

### 3. شیفت داده‌ها:

- به دلیل استفاده از داده‌های 16 بیتی در حین دسترسی به حافظه که طراحی آن برای 32 بیت است، لازم است داده‌های دریافتی به درستی در بین سیکل‌های کلاک شیفت داده شوند تا داده کامل بازسازی شود.

با فورواردینگ :

log: 2024/05/29 10:55:27 #0		click to insert time bar																							
Node		0																	1						
Type	Alias	Name	-128	-64	0	64	128	192	256	320	384	448	512	576	640	704	768	832	896						
		ter_file/reg_file[0]	0		128	1024											20								
		ter_file/reg_file[1]	1			8192			4												-2147483648				
		ter_file/reg_file[2]	2				-1073741824	0		1	2	3	0	1	2	3	0	1	2	3		-1073741824			
		ter_file/reg_file[3]	3				-2147483648	0	1	2	3	0	1	2	3	0	1	2	3		41				
		ter_file/reg_file[4]	4			41	1024	1028	1032	1024	1028	1032	1024	1028	1032	1024	1028	1032		8192					
		ter_file/reg_file[5]	5			-123		8192		-1073741824	41		41		41		41			-123					
		SW[0]																							

بدون فورواردینگ :

Node		0	egmer																												0	
Type	Alias	Name	-60	Value59	520	524	528	532	536	540	544	548	552	556	560	564	568	572	576	580	584	588	592	596	600	604	608	612	616	620	624	628
		SW[0]		1																												
		ter_file/reg_file[0]		0	20																											
		ter_file/reg_file[1]		1	-2147483648																											
		ter_file/reg_file[2]		2	-1073741824																											
		ter_file/reg_file[3]		3	41																											
		ter_file/reg_file[4]		4	8192																											
		ter_file/reg_file[5]		5	-123																											
		er_file/reg_file[10]		10	-1073741824																											
		er_file/reg_file[11]		11	8192																											

برای کاهش کارایی داریم:

$$\frac{430 - 194}{430} = \frac{236}{430} = 0.5476 \rightarrow \text{حدود 54 درصد کارایی کاهش پیدا کرده}$$

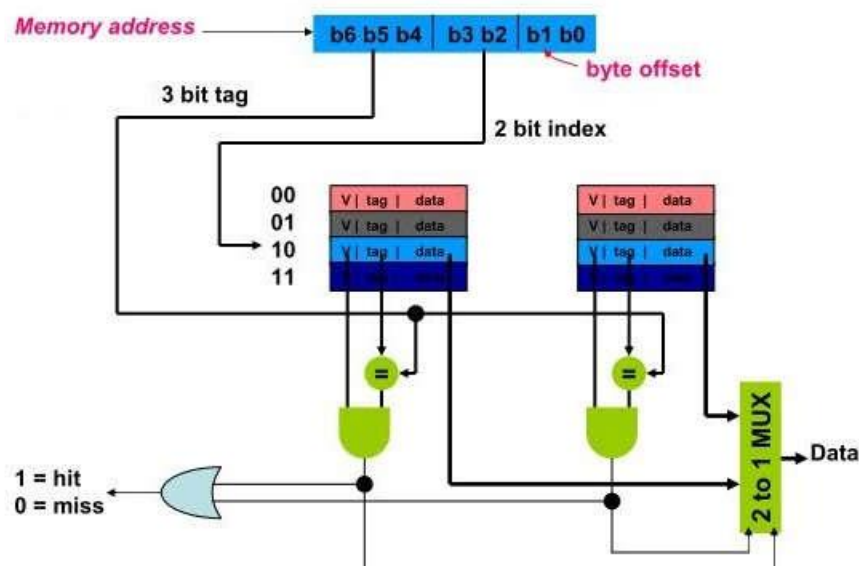
طبیعتا انتظار داریم که با اضافه شدن ماژول های مختلف سخت افزار این بخش هم باعث افزایش هزینه و هم ککاهش کارایی رو داشته باشیم.

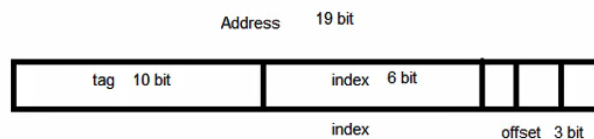
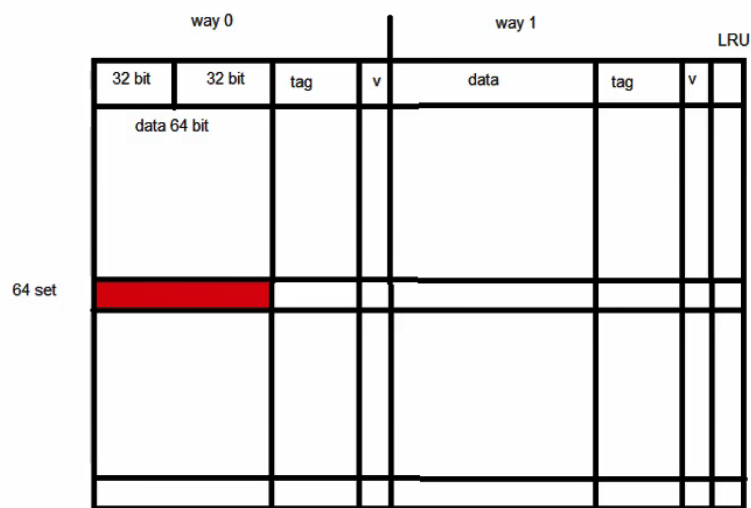
## بخش CACHE :

حافظه Cache نوعی حافظه با سرعت بالا و اندازه نسبتاً کوچک است که برای ذخیره داده‌ها و دستورالعمل‌های پرکاربرد به کار می‌رود تا دسترسی سریع‌تری نسبت به حافظه اصلی داشته باشد. این حافظه به عنوان یک لایه واسطه بین پردازنده و حافظه اصلی عمل می‌کند و از اصل محلیت استفاده می‌کند که شامل محلیت مکانی و زمانی است. محلیت مکانی بیان می‌کند که دسترسی به یک داده معمولاً به دنبال دسترسی به داده‌های مجاور آن خواهد بود، و محلیت زمانی نشان می‌دهد که داده‌هایی که یک بار استفاده شده‌اند، احتمالاً در آینده نزدیک دوباره مورد استفاده قرار خواهند گرفت.

زمانی که پردازنده به داده‌ای نیاز دارد، ابتدا Cache بررسی می‌شود؛ اگر داده مورد نظر در Cache موجود باشد (Hit)، داده با سرعت بسیار بالا بازیابی می‌شود. اما اگر داده در Cache موجود نباشد (Miss)، پردازنده مجبور است داده را از حافظه اصلی که فرایندی کندتر است بازیابی کند. Cache معمولاً به صورت ست-اسوسیاتیو سازماندهی می‌شوند که در آن هر ست شامل چندین خط است و هر خط قادر به ذخیره سازی چندین بلوک داده است. برای مدیریت داده‌هایی که باید از Cache حذف شوند تا جای برای داده‌های جدید باز شود، از الگوریتم Least Recently Used (LRU) استفاده می‌شود.

در نهایت، مدیریت Cache توسط کنترلر Cache انجام می‌شود که وظیفه دارد هر زمان که دسترسی به Cache انجام می‌شود، بررسی کند آیا داده‌ای در Cache موجود است یا خیر. این کنترلر همچنین مسئولیت بازیابی داده‌ها از حافظه اصلی را در صورت Miss در Cache بر عهده دارد. استفاده از حافظه Cache به طور قابل توجهی عملکرد پردازنده‌ها را افزایش داده و تاخیر ناشی از دسترسی به حافظه اصلی را کاهش می‌دهد، که این امر به ویژه در پردازش‌های سنگین و دسترسی‌های مکرر به داده، اهمیت زیادی دارد.

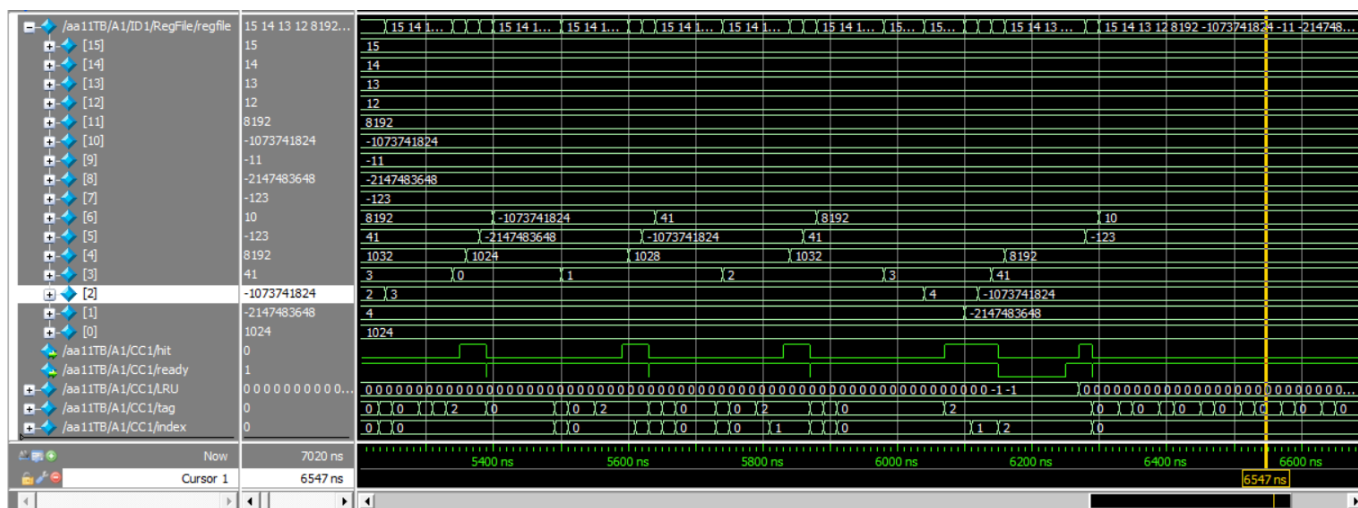




WRITE: LRU == 0 => data way 1 , LRU = 1  
LRU == 1 => data way 0 , LRU = 0

I

## نتایج:



با مقایسه با قسمت های قبل می فهمیم که:

زمان sram بیشتر از زمان sram+cache و هر دو آن ها بیشتر از زمان حافظه داخلی است. که برای کارایی دقیقاً معکوس می شود.