به نام خدا

دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر

آزمایشگاه سیستم های دیجیتال ۲

آزمایش پیاده سازي پردازنده **ARM**

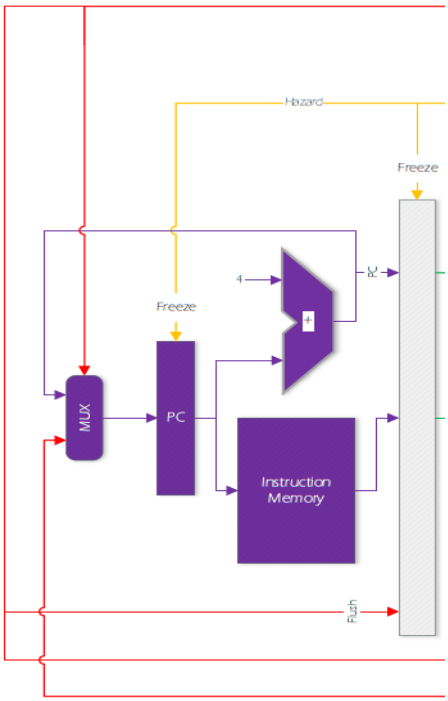
نام و نام خانوادگی	امیرمهدی جودی عرفان ایروانی
شماره دانشجویی	۸۱۰۱۹۷٤۸۷ ۸۱۰۱۹۷۴۶۲
تاریخ ارسال گزارش	۱۴۰۰/۴/۴

فهرست گزارش مطالب

پیاده سازی مرحله واکشی	۳
پیاده سازی مرحله کدگشایی	۴
پیاده سازی مرحله اجرا	۵
پیاده سازی مرحله حافظه	۶
پیاده سازی مرحله بازنویسی	۷
پیاده سازی واحد مخاطره	۸
خروجی نهایی در Modelsim	۸
خروجی نهایی در Quartus	۹
پیاده‌سازی تکنیک ارسال به جلو	۱۰
پیاده‌سازی SRAM و SRAM Controller	۱۱
پیاده‌سازی CACHE و CACHE Controller	۱۳

پیاده سازی مرحله واکشی

در این قسمت، حرکت دستورات در خط لوله بررسی می شود. این قسمت شامل پیاده سازی مرحله IF و رجیستر بعد از آن است. سایر مراحل و رجیستر ها فقط دستور را عبور می دهند.

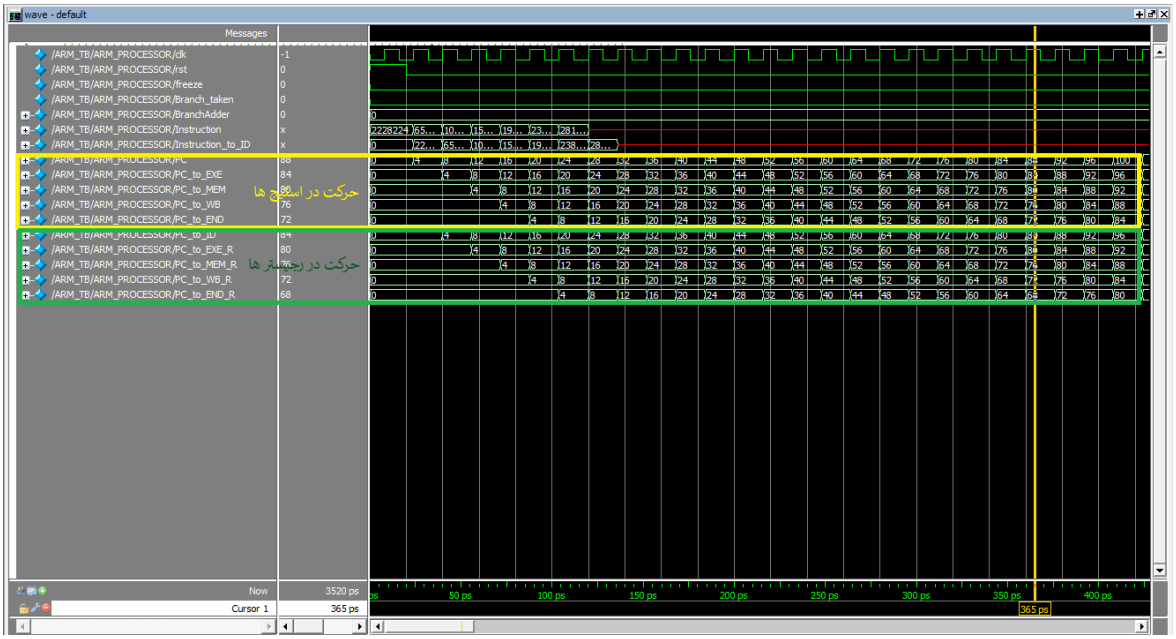


IF Stage and Register

این قسمت از یک جمع کننده، یک رجیستر، یک مالتی پلکسر و یک حافظه برای ذخیره دستورات است.

رجیستر پایپ لاین، خروجی شمارنده و خود دستورات رو به مرحله بعد انتقال می دهد. در هنگاه مخاطره، رجیستر فریز شده و در هنگام رخ دادن branch هم فلاش می شود تا دستورات نادرست از پردازنده حذف شوند.

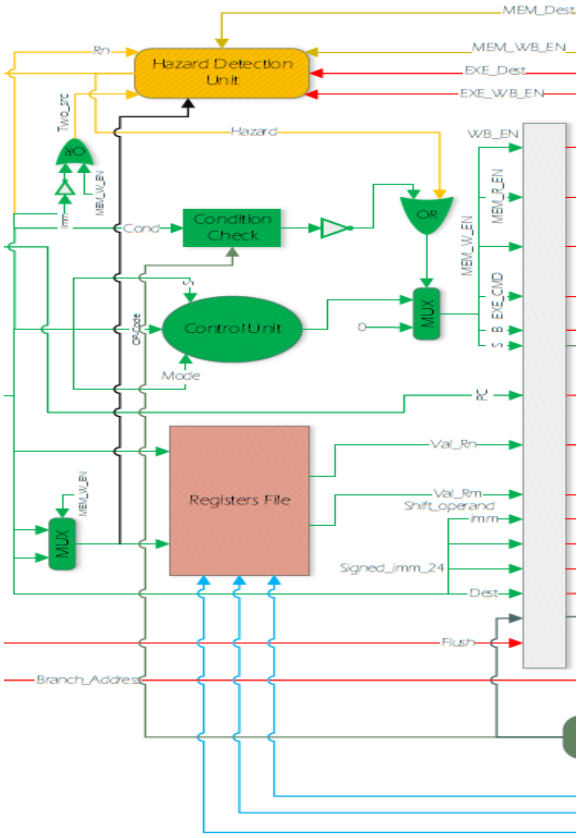
نتیجه به صورت زیر است:



خروجی بخش اول

پیاده سازی مرحله کدگشایی

در این قسمت، مرحله ID (بدون Hazard Unit) پیاده شده و ۱۸ دستور اول برنامه محک اجرا می شود.



به کمک condition check شرط اجرای دستور چک می شود. اگر شرط برقرار نبود دستور اجرا نمی شود(سیگنال های کنترلی صفر می شوند).

واحد control unit با بررسی opcode و mode و s، سیگنال های کنترلی را تولید می کند.

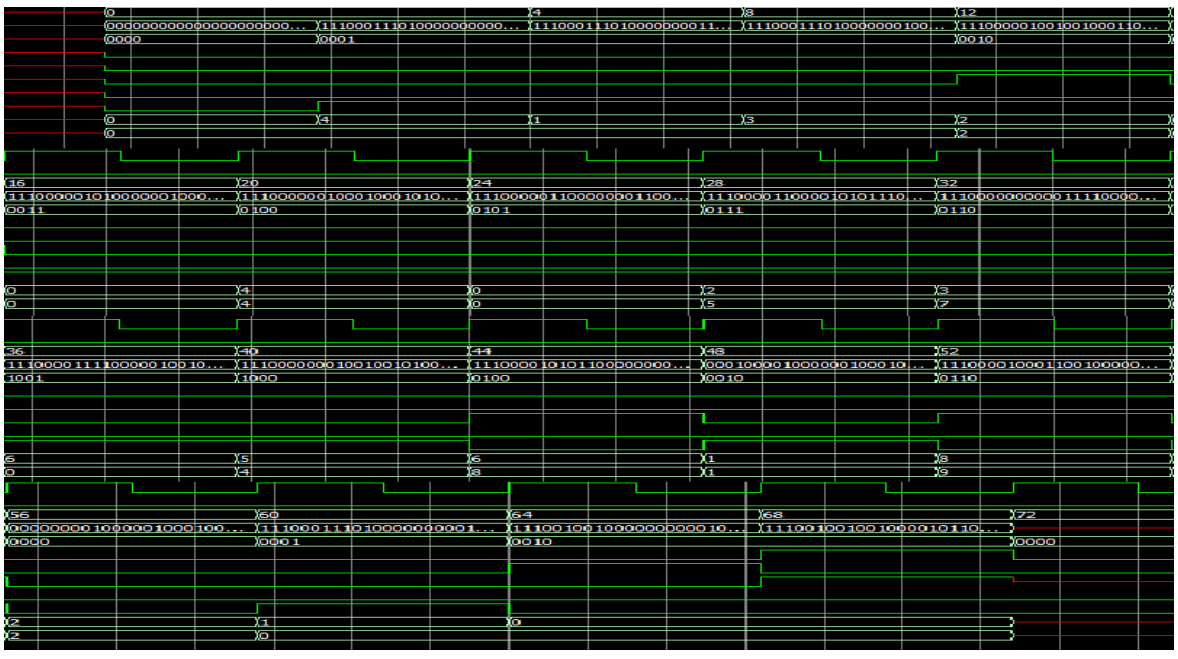
واحد two source برای واحد مخاطره، یک یا دو رجیستره بودن دستور را مشخص می کند.

واحد register file شامل رجیستر های پردازنده می باشد. نوشتن به لبه ی بالارونده کلاک همگام است ولی خواندن بدون کلاک می باشد.

برای دستور پرش، اگر شرط پرش برقرار باشد، رجیستر فلاش می شود.

ID Stage and Register

نتیجه به صورت زیر است:



خروجی بخش دوم

پیاده سازی مرحله اجرا

در این قسمت، مرحله EXE پیاده شده و ۱۸ دستور اول برنامه محک اجرا می شود.

این قسمت شامل واحد ALU است که دستورات محاسباتی را انجام می دهد.

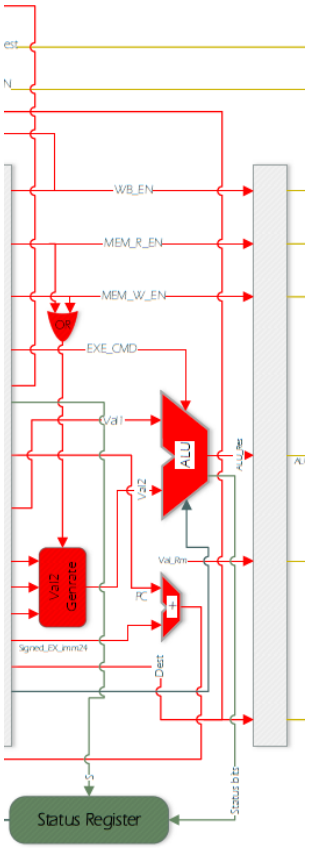
قسمت Val2Generate وظیفه ی ساخت ورودی دوم واحد ALU را دارد. اگر دستورات مربوط به حافظه باشد، مقدار offset گسترش علامت داده می شود. اگر دستور از نوع فوری باشد، rotate داده می شود و اگر از نوع فوری نباشد، شیفت اعمال می شود.

رجیستر وضعیت با لبه ی پایین رونده کلاک همگام شده است. این رجیستر حاوی فلگ های n، z، c و v است.

- فلگ n همان بیت ۳۱ خروجی است.
- فلگ z هم از or شدن بیت های خروجی به دست می آید.
- فلگ c در جمع carry و در تفریق borrow است.
- فلگ v هم overflow است.

در جمع: اگر ورودی ها مثبت و خروجی منفی شود یا بر عکس.

در تفریق: اگر عملوند دوم و خروجی هم علامت شوند.



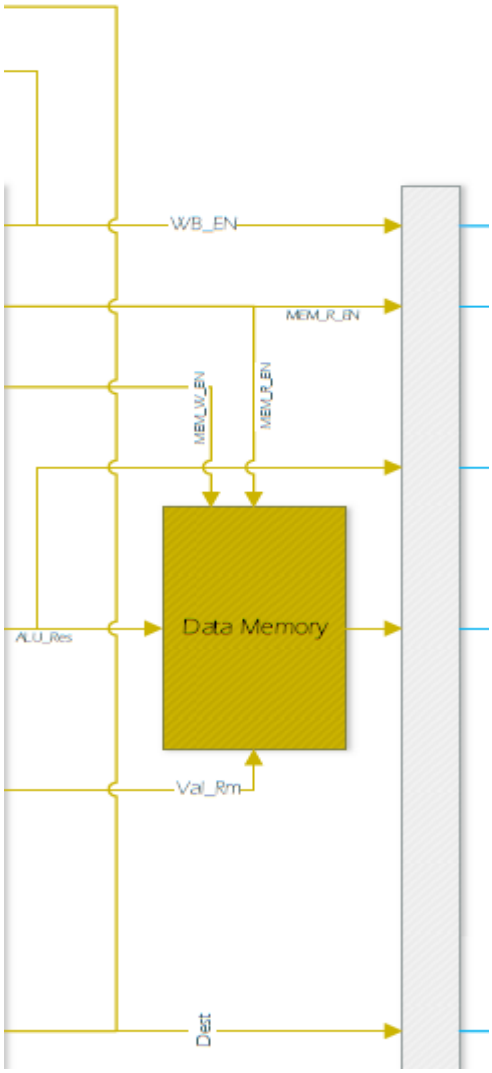
EXE Stage and Register

پیاده سازی مرحله حافظه

در این قسمت مرحله MEM پیاده سازی می شود.

این مرحله شامل حافظه ی داده می باشد. نوشتن با کلاک همگام بوده ولی خواندن این طور نیست.

سیگنال های رجیستر مقصد، خروجی مرحله اجرا و نوشتن در مقصد نیز به مرحله ی بعد فرستاده می شوند.



MEM Stage and Register

پیاده سازی مرحله بازنویسی

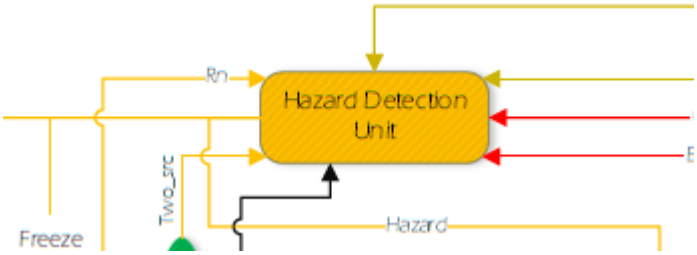
در این قسمت مرحله WB پیاده سازی می شود.

WB Stage

این قسمت شامل یک مالتی پلکسر می باشد که خروجی حافظه یا مرحله اجرا را برای بازنویسی در رجیستر مقصد می فرستد.

پیاده سازی واحد مخاطره

در این قسمت پیاده سازی واحد مخاطره انجام می شود.

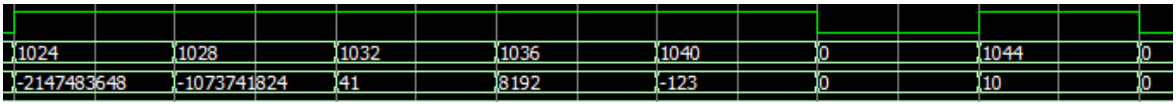


Hazard Unit

اگر در مرحله ID دستوری وارد شود که به داده ای نیاز داشته باشد که طی یک یا دو دستور قبل تغییر داده شده است، پایپ لاین باید متوقف شده تا دستورات قبلی به اتمام برسند. برای این منظور، حباب ایجاد شده و پایپ لاین متوقف می شود.

خروجی نهایی در Modelsim

حافظه ی داده به صورت زیر پر شده است:



اجرای این دستورات ۲۸۴ سیکل طول کشیده و دستور آخر در پردازنده می ماند.

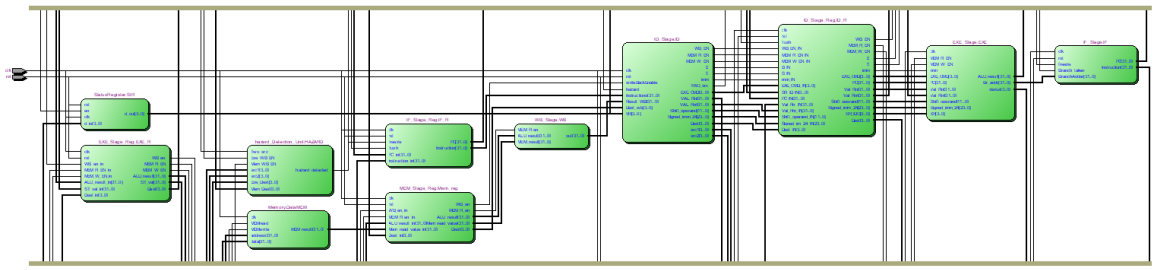
خروجی نهایی در Quartus

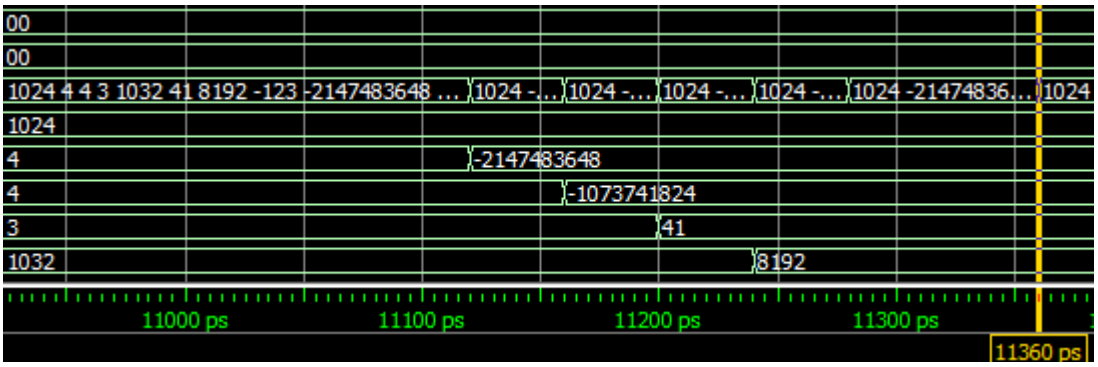
در signal tap analyzer کلاک را وصل کرده و کامپایل می کنیم. نتیجه به صورت زیر است:

Analysis & Synthesis Summary	
Analysis & Synthesis Status	Successful - Thu Jun 24 23:28:44 2021
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	arm
Top-level Entity Name	ARM
Family	Cyclone IV E
Total logic elements	1,139
Total combinational functions	545
Dedicated logic registers	905
Total registers	905
Total pins	34
Total virtual pins	0
Total memory bits	17,408
Embedded Multiplier 9-bit elements	0
Total PLLs	0

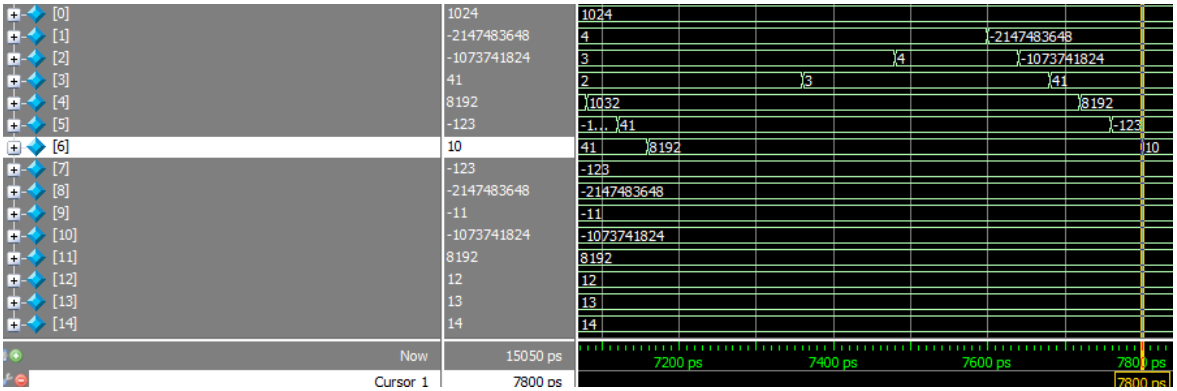
Slow 1200mV 125C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name	Note
1	77.81 MHz	77.81 MHz	altera_reserved_tck	





بدون این تکنیک، کارآیی به صورت بالا است. این قسمت ۲۸۴ کلاک طول کشیده است(طول کلاک ۴۰ می‌باشد).



با این تکنیک، کارآیی به صورت بالا است. این قسمت ۱۹۵ کلاک طول کشیده است.

یعنی کارآیی ۱.۴۶ برابر شده است.

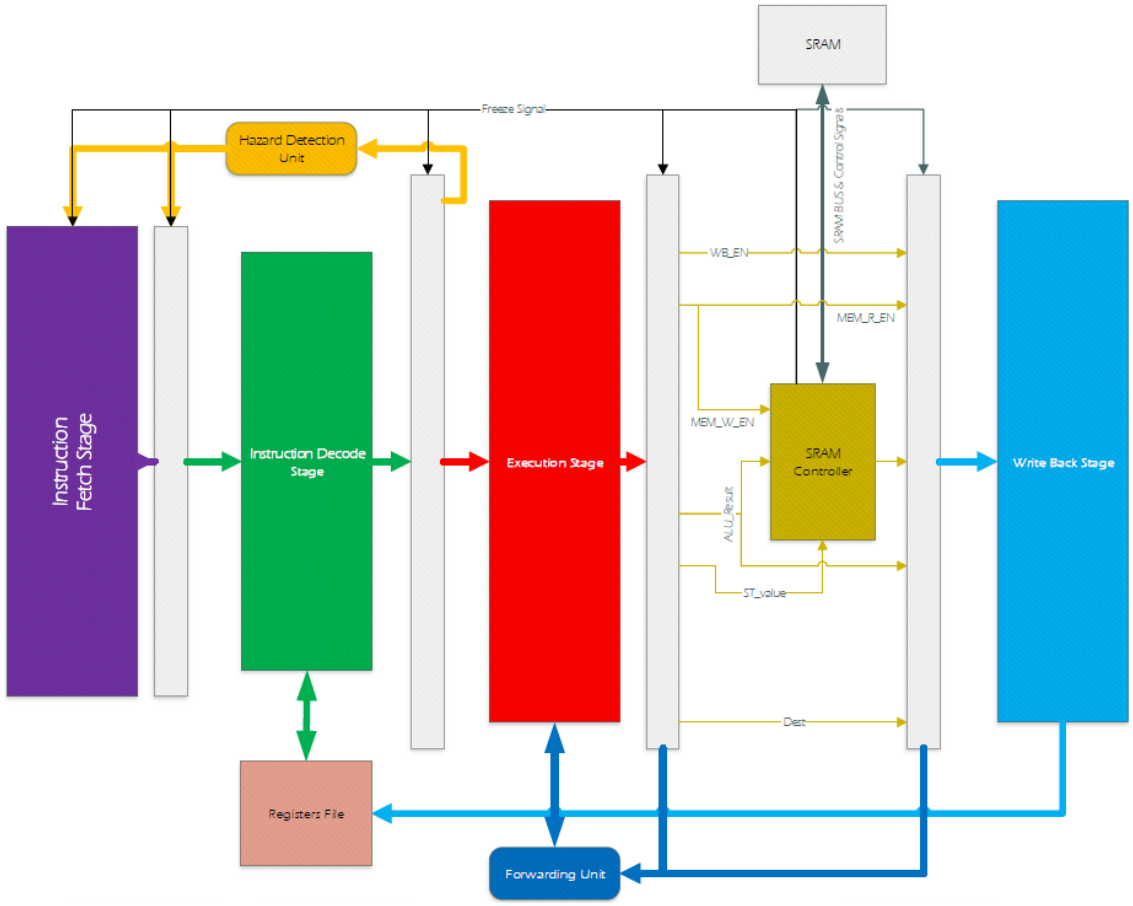
Analysis & Synthesis Summary	
Analysis & Synthesis Status	Successful - Fri Jun 25 00:16:21 2021
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	arm
Top-level Entity Name	ARM
Family	Cyclone IV E
Total logic elements	1,157
Total combinational functions	549
Dedicated logic registers	922
Total registers	922
Total pins	35
Total virtual pins	0
Total memory bits	17,920
Embedded Multiplier 9-bit elements	0
Total PLLs	0

تعداد کل المان‌های منطقی زیاد شده است.

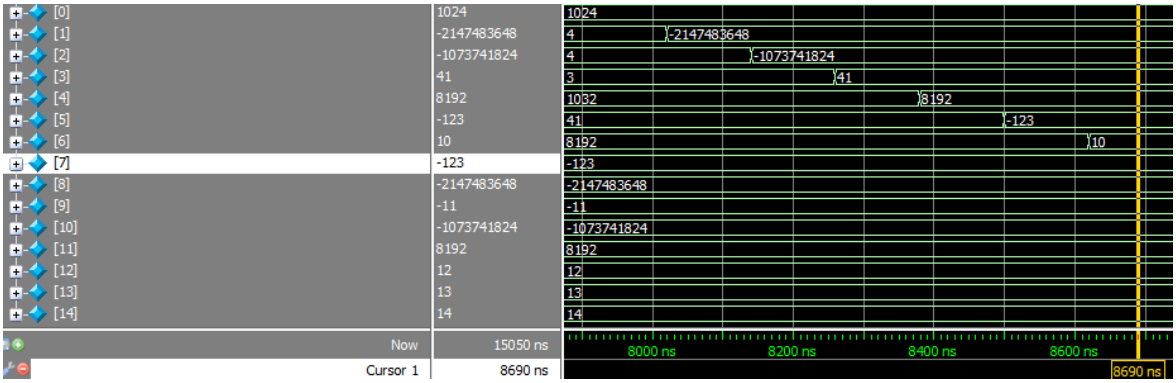
Performance/Cost (1) = 1/284/1139*40*10E9= 123656.78

Performance/Cost (2) = 1/195/1157*40*10E9= 121732.99

پیاده‌سازی SRAM و SRAM Controller



به کمک یک حافظه SRAM که به واقعیت نزدیک است، پردازنده را مجددا اجرا می کنیم. این حافظه asynchronous بوده و به کمک یه Controller به صورت synchronous و مشابه مراحل قبل پیاده می‌شود. آز آنجایی که کار حافظه به بیش از یک سیکل نیاز دارد، رجیسترهای میانی همگی stall شده تا داده‌های نادرست وارد پردازنده نشوند.



به این ترتیب، برنامه در ۴۳۵ کلاک انجام می‌شود، یعنی کارآیی نسبت به قسمت اول ۰.۶۵ و نسبت به قسمت دوم ۰.۴۵ شده است. برای رفع این مشکل از حافظه‌ی سریع cache استفاده می‌شود.

Analysis & Synthesis Summary	
Analysis & Synthesis Status	Successful - Fri Jun 25 00:21:29 2021
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	arm
Top-level Entity Name	ARM
Family	Cyclone IV E
Total logic elements	1,154
Total combinational functions	546
Dedicated logic registers	922
Total registers	922
Total pins	35
Total virtual pins	0
Total memory bits	17,920
Embedded Multiplier 9-bit elements	0
Total PLLs	0

Slow 1200mV 125C Model Fmax Summary				
	Fmax	Restricted Fmax	Clock Name	Note
1	68.42 MHz	68.42 MHz	altera_reserved_tck	

تعداد کل المان‌های منطقی زیاد شده است. در کل نیز وضع بدتر شده است(فرکانس کاری کم شده است).

در مقایسه زیر، چون فرکانس کلاک ها تغییر کرده، performance دو قسمت اول با فرکانس ۵۰ مگاهرتز مجددا بازنویسی شدند:

Performance/Cost (1) = 1/284/1139*20*10E9= 90047.50

Performance/Cost (2) = 1/195/1157*20*10E9= 88646.59

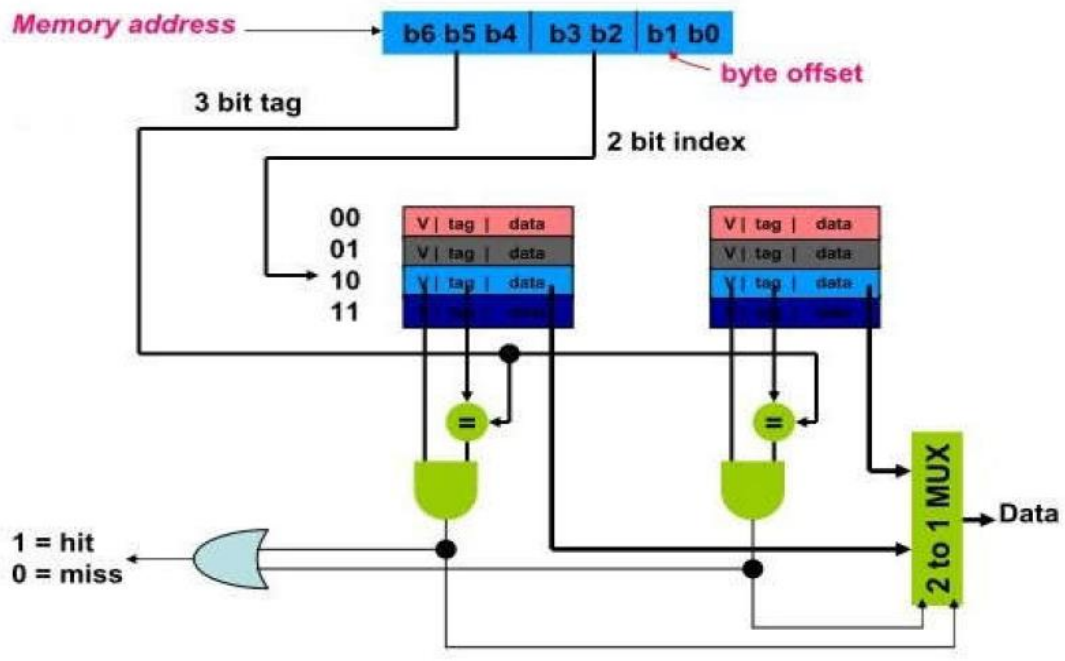
Performance/Cost (3) = 1/435/1154*20*10E9 = 39841.43

پیاده‌سازی CACHE و CACHE Controller

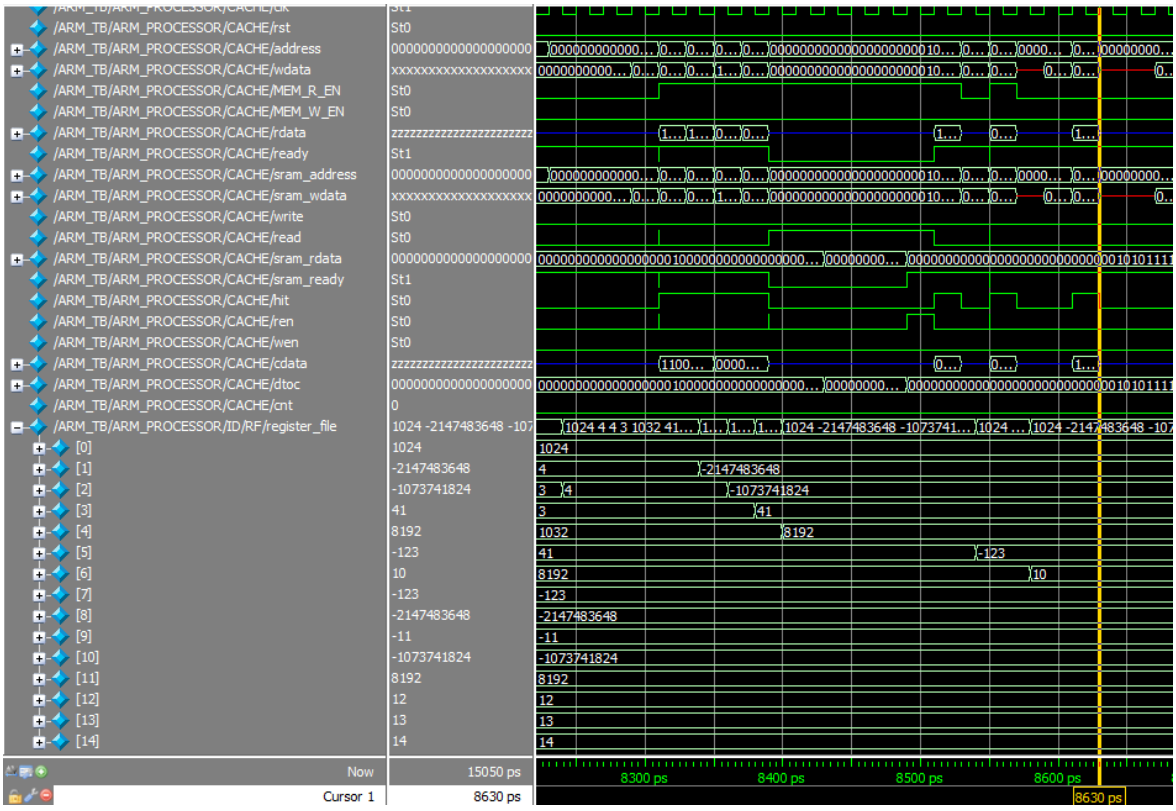
برای جبران کندی ناشی از SRAM میتوان از حافظه CACHE استفاده کرد.

حافظه های CACHE به صورت‌های مختلف پیاده‌سازی می‌شوند. در این آزمایش یک CACHE به صورت 2-Way Set Associates با مشخصات زیر می‌سازیم:

اندازه هر کلمه: ۳۲ بیت
اندازه هر بلاک: ۶۴ بیت (۲ کلمه)
تعداد مجموعه ها (set): ۶۴
اندازه حافظه نهان: ۱ کیلو بایت برای داده (همچنین حدود ۱ کیلو بایت هم برای نشانه ها و غیره مورد نیاز است.)
گذرگاه آدرس: ۱۹ بیت
تعداد بیت مورد نیاز برای نشانه (tag): ۱۰ بیت
تعداد بیت شاخص (index): ۶ بیتن
دارای بیت اعتبار (valid) برای هر بلاک
هر مجموعه نیاز به یک بیت (used) برای سیاست جایگزینی LRU دارد.



مثالی از cache



به این ترتیب، برنامه در ۴۳۲ کلاک انجام می‌شود، یعنی کارآیی نسبت به قسمت اول ۰.۶۶ و نسبت به قسمت دوم ۰.۴۵ و نسبت به قسمت سوم ۱ شده است.

Analysis & Synthesis Summary	
Analysis & Synthesis Status	Successful - Fri Jun 25 00:32:17 2021
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	ARM
Top-level Entity Name	ARM
Family	Cyclone IV E
Total logic elements	1,156
Total combinational functions	548
Dedicated logic registers	922
Total registers	922
Total pins	35
Total virtual pins	0
Total memory bits	17,920
Embedded Multiplier 9-bit elements	0
Total PLLs	0

Slow 1200mV 125C Model Fmax Summary				
	Fmax	Restricted Fmax	Clock Name	Note
1	67.0 MHz	67.0 MHz	altera_reserved_tck	

Performance/Cost (1) = 1/284/1139*20*10E9= 90047.50

Performance/Cost (2) = 1/195/1157*20*10E9= 88646.59

Performance/Cost (3) = 1/435/1154*20*10E9 = 39841.43

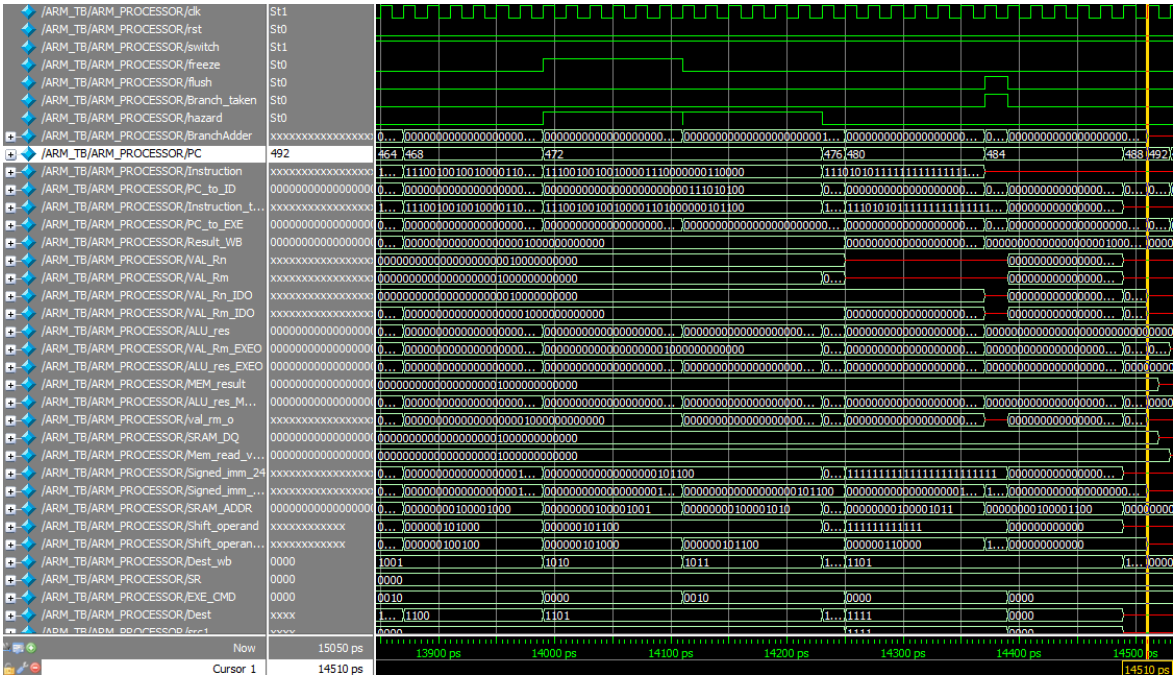
Performance/Cost (3) = 1/432/1156*20*10E9 = 40048.70

با توجه به نوع دستورات، مزیت استفاده از cache به خوبی دیده نشده‌است. پس دستورات زیر را در دو حالت بدون کش و با کش اجرا می‌کنیم:

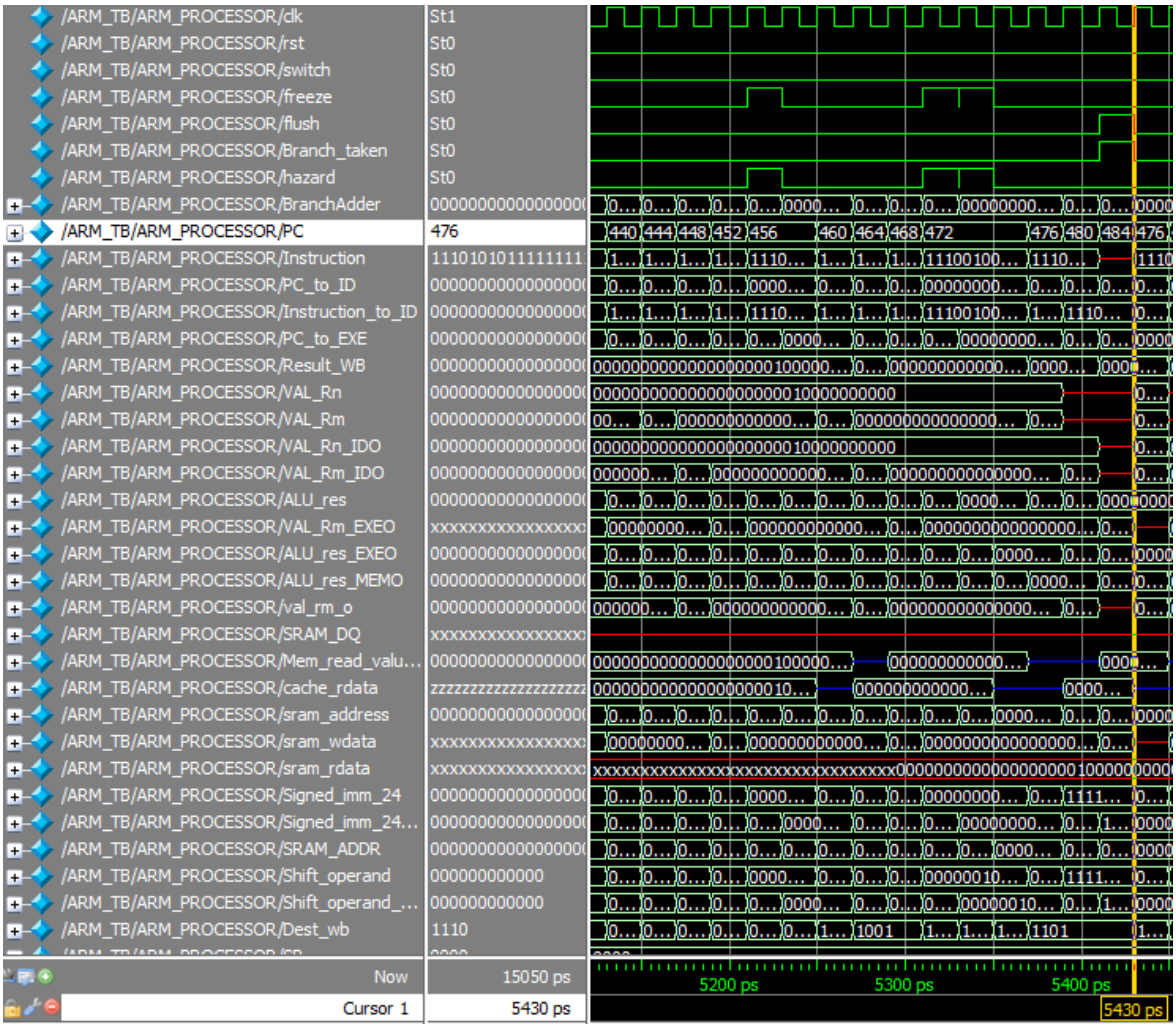
```
1110_00_1_1101_0_0000_0000_101100000001; //MOV     R0,#1024
1110_00_1_1101_0_0000_0001_101000000001; //MOV     R1,#4096
1110_01_0_0100_0_0000_0001_000000000000; //STR     MEM[1024] = 4096
1110_01_0_0100_0_0000_0001_000000000100; //STR     MEM[1028] = 4096
1110_01_0_0100_0_0000_0001_000000001000; //STR     MEM[1032] = 4096
1110_01_0_0100_0_0000_0001_000000001100; //STR     MEM[1036] = 4096
1110_01_0_0100_0_0000_0001_000000010000; //STR     MEM[1040] = 4096
1110_01_0_0100_0_0000_0001_000000010100; //STR     MEM[1044] = 4096
1110_01_0_0100_0_0000_0001_000000011000; //STR     MEM[1048] = 4096
1110_01_0_0100_0_0000_0001_000000011100; //STR     MEM[1052] = 4096
1110_01_0_0100_0_0000_0001_000000100000; //STR     MEM[1056] = 4096
1110_01_0_0100_0_0000_0001_000000100100; //STR     MEM[1060] = 4096
1110_01_0_0100_0_0000_0001_000000101000; //STR     MEM[1064] = 4096
1110_01_0_0100_0_0000_0001_000000101100; //STR     MEM[1068] = 4096
1110_01_0_0100_0_0000_0001_000000110000; //STR     MEM[1072] = 4096
1110_01_0_0100_1_0000_0010_000000000000; //LDR     R2 = MEM[1024]
1110_01_0_0100_1_0000_0011_000000000100; //LDR     R3 = MEM[1028]
1110_01_0_0100_1_0000_0100_000000001000; //LDR     R4 = MEM[1032]
1110_01_0_0100_1_0000_0101_000000001100; //LDR     R5 = MEM[1036]
1110_01_0_0100_1_0000_0110_000000010000; //LDR     R6 = MEM[1040]
1110_01_0_0100_1_0000_0111_000000010100; //LDR     R7 = MEM[1044]
1110_01_0_0100_1_0000_1000_000000011000; //LDR     R8 = MEM[1048]
1110_01_0_0100_1_0000_1001_000000011100; //LDR     R9 = MEM[1052]
1110_01_0_0100_1_0000_1010_000000100000; //LDR     R10 = MEM[1056]
1110_01_0_0100_1_0000_1011_000000100100; //LDR     R11 = MEM[1060]
1110_01_0_0100_1_0000_1100_000000101000; //LDR     R12 = MEM[1064]
1110_01_0_0100_1_0000_1101_000000101100; //LDR     R13 = MEM[1068]
1110_01_0_0100_1_0000_1110_000000110000; //LDR     R14 = MEM[1072]
1110_01_0_0100_1_0000_0010_000000000000; //LDR     R2 = MEM[1024]
1110_01_0_0100_1_0000_0011_000000000100; //LDR     R3 = MEM[1028]
1110_01_0_0100_1_0000_0100_000000001000; //LDR     R4 = MEM[1032]
1110_01_0_0100_1_0000_0101_000000001100; //LDR     R5 = MEM[1036]
```

```
1110_01_0_0100_1_0000_0110_000000010000; //LDR   R6 = MEM[1040]
1110_01_0_0100_1_0000_0111_000000010100; //LDR   R7 = MEM[1044]
1110_01_0_0100_1_0000_1000_000000011000; //LDR   R8 = MEM[1048]
1110_01_0_0100_1_0000_1001_000000011100; //LDR   R9 = MEM[1052]
1110_01_0_0100_1_0000_1010_000000100000; //LDR   R10 = MEM[1056]
1110_01_0_0100_1_0000_1011_000000100100; //LDR   R11 = MEM[1060]
1110_01_0_0100_1_0000_1100_000000101000; //LDR   R12 = MEM[1064]
1110_01_0_0100_1_0000_1101_000000101100; //LDR   R13 = MEM[1068]
1110_01_0_0100_1_0000_1110_000000110000; //LDR   R14 = MEM[1072]
1110_01_0_0100_1_0000_0010_000000000000; //LDR   R2 = MEM[1024]
1110_01_0_0100_1_0000_0011_000000000100; //LDR   R3 = MEM[1028]
1110_01_0_0100_1_0000_0100_000000001000; //LDR   R4 = MEM[1032]
1110_01_0_0100_1_0000_0101_000000001100; //LDR   R5 = MEM[1036]
1110_01_0_0100_1_0000_0110_000000010000; //LDR   R6 = MEM[1040]
1110_01_0_0100_1_0000_0111_000000010100; //LDR   R7 = MEM[1044]
1110_01_0_0100_1_0000_1000_000000011000; //LDR   R8 = MEM[1048]
1110_01_0_0100_1_0000_1001_000000011100; //LDR   R9 = MEM[1052]
1110_01_0_0100_1_0000_1010_000000100000; //LDR   R10 = MEM[1056]
1110_01_0_0100_1_0000_1011_000000100100; //LDR   R11 = MEM[1060]
1110_01_0_0100_1_0000_1100_000000101000; //LDR   R12 = MEM[1064]
1110_01_0_0100_1_0000_1101_000000101100; //LDR   R13 = MEM[1068]
1110_01_0_0100_1_0000_1110_000000110000; //LDR   R14 = MEM[1072]
1110_01_0_0100_1_0000_0010_000000000000; //LDR   R2 = MEM[1024]
1110_01_0_0100_1_0000_0011_000000000100; //LDR   R3 = MEM[1028]
1110_01_0_0100_1_0000_0100_000000001000; //LDR   R4 = MEM[1032]
1110_01_0_0100_1_0000_0101_000000001100; //LDR   R5 = MEM[1036]
1110_01_0_0100_1_0000_0110_000000010000; //LDR   R6 = MEM[1040]
1110_01_0_0100_1_0000_0111_000000010100; //LDR   R7 = MEM[1044]
1110_01_0_0100_1_0000_1000_000000011000; //LDR   R8 = MEM[1048]
1110_01_0_0100_1_0000_1001_000000011100; //LDR   R9 = MEM[1052]
1110_01_0_0100_1_0000_1010_000000100000; //LDR   R10 = MEM[1056]
1110_01_0_0100_1_0000_1011_000000100100; //LDR   R11 = MEM[1060]
1110_01_0_0100_1_0000_1100_000000101000; //LDR   R12 = MEM[1064]
1110_01_0_0100_1_0000_1101_000000101100; //LDR   R13 = MEM[1068]
1110_01_0_0100_1_0000_1110_000000110000; //LDR   R14 = MEM[1072]
1110_01_0_0100_1_0000_0010_000000000000; //LDR   R2 = MEM[1024]
1110_01_0_0100_1_0000_0011_000000000100; //LDR   R3 = MEM[1028]
1110_01_0_0100_1_0000_0100_000000001000; //LDR   R4 = MEM[1032]
1110_01_0_0100_1_0000_0101_000000001100; //LDR   R5 = MEM[1036]
1110_01_0_0100_1_0000_0110_000000010000; //LDR   R6 = MEM[1040]
```

```
1110_01_0_0100_1_0000_0111_000000010100; //LDR     R7 = MEM[1044]
1110_01_0_0100_1_0000_1000_000000011000; //LDR     R8 = MEM[1048]
1110_01_0_0100_1_0000_1001_000000011100; //LDR     R9 = MEM[1052]
1110_01_0_0100_1_0000_1010_000000100000; //LDR     R10 = MEM[1056]
1110_01_0_0100_1_0000_1011_000000100100; //LDR     R11 = MEM[1060]
1110_01_0_0100_1_0000_1100_000000101000; //LDR     R12 = MEM[1064]
1110_01_0_0100_1_0000_1101_000000101100; //LDR     R13 = MEM[1068]
1110_01_0_0100_1_0000_1110_000000110000; //LDR     R14 = MEM[1072]
1110_01_0_0100_1_0000_0010_000000000000; //LDR     R2 = MEM[1024]
1110_01_0_0100_1_0000_0011_000000000100; //LDR     R3 = MEM[1028]
1110_01_0_0100_1_0000_0100_000000001000; //LDR     R4 = MEM[1032]
1110_01_0_0100_1_0000_0101_000000001100; //LDR     R5 = MEM[1036]
1110_01_0_0100_1_0000_0110_000000010000; //LDR     R6 = MEM[1040]
1110_01_0_0100_1_0000_0111_000000010100; //LDR     R7 = MEM[1044]
1110_01_0_0100_1_0000_1000_000000011000; //LDR     R8 = MEM[1048]
1110_01_0_0100_1_0000_1001_000000011100; //LDR     R9 = MEM[1052]
1110_01_0_0100_1_0000_1010_000000100000; //LDR     R10 = MEM[1056]
1110_01_0_0100_1_0000_1011_000000100100; //LDR     R11 = MEM[1060]
1110_01_0_0100_1_0000_1100_000000101000; //LDR     R12 = MEM[1064]
1110_01_0_0100_1_0000_1101_000000101100; //LDR     R13 = MEM[1068]
1110_01_0_0100_1_0000_1110_000000110000; //LDR     R14 = MEM[1072]
1110_01_0_0100_1_0000_0010_000000000000; //LDR     R2 = MEM[1024]
1110_01_0_0100_1_0000_0011_000000000100; //LDR     R3 = MEM[1028]
1110_01_0_0100_1_0000_0100_000000001000; //LDR     R4 = MEM[1032]
1110_01_0_0100_1_0000_0101_000000001100; //LDR     R5 = MEM[1036]
1110_01_0_0100_1_0000_0110_000000010000; //LDR     R6 = MEM[1040]
1110_01_0_0100_1_0000_0111_000000010100; //LDR     R7 = MEM[1044]
1110_01_0_0100_1_0000_1000_000000011000; //LDR     R8 = MEM[1048]
1110_01_0_0100_1_0000_1001_000000011100; //LDR     R9 = MEM[1052]
1110_01_0_0100_1_0000_1010_000000100000; //LDR     R10 = MEM[1056]
1110_01_0_0100_1_0000_1011_000000100100; //LDR     R11 = MEM[1060]
1110_01_0_0100_1_0000_1100_000000101000; //LDR     R12 = MEM[1064]
1110_01_0_0100_1_0000_1101_000000101100; //LDR     R13 = MEM[1068]
1110_01_0_0100_1_0000_1110_000000110000; //LDR     R14 = MEM[1072]
1110_10_1_0_11111111111111111111111 ;    //B       #-1
```


/ARM_TB/ARM_PROCESSOR/clk
/ARM_TB/ARM_PROCESSOR/rst
/ARM_TB/ARM_PROCESSOR/switch
/ARM_TB/ARM_PROCESSOR/freeze
/ARM_TB/ARM_PROCESSOR/flush
/ARM_TB/ARM_PROCESSOR/Branch_taken
/ARM_TB/ARM_PROCESSOR/hazard
/ARM_TB/ARM_PROCESSOR/BranchAdder
/ARM_TB/ARM_PROCESSOR/PC
/ARM_TB/ARM_PROCESSOR/Instruction
/ARM_TB/ARM_PROCESSOR/PC_to_ID
/ARM_TB/ARM_PROCESSOR/Instruction_t...
/ARM_TB/ARM_PROCESSOR/PC_to_EXE
/ARM_TB/ARM_PROCESSOR/Result_WB
/ARM_TB/ARM_PROCESSOR/VAL_Rn
/ARM_TB/ARM_PROCESSOR/VAL_Rm
/ARM_TB/ARM_PROCESSOR/VAL_Rn_IDO
/ARM_TB/ARM_PROCESSOR/VAL_Rm_IDO
/ARM_TB/ARM_PROCESSOR/ALU_res
/ARM_TB/ARM_PROCESSOR/VAL_Rm_EXEO
/ARM_TB/ARM_PROCESSOR/ALU_res_EXEO
/ARM_TB/ARM_PROCESSOR/MEM_result
/ARM_TB/ARM_PROCESSOR/ALU_res_M...
/ARM_TB/ARM_PROCESSOR/val_rm_o
/ARM_TB/ARM_PROCESSOR/SRAM_DQ
/ARM_TB/ARM_PROCESSOR/Mem_read_v...
/ARM_TB/ARM_PROCESSOR/Signed_imm_24
/ARM_TB/ARM_PROCESSOR/Signed_imm_...
/ARM_TB/ARM_PROCESSOR/SRAM_ADDR
/ARM_TB/ARM_PROCESSOR/Shift_operand
/ARM_TB/ARM_PROCESSOR/Shift_operan...
/ARM_TB/ARM_PROCESSOR/Dest_wb
/ARM_TB/ARM_PROCESSOR/SR
/ARM_TB/ARM_PROCESSOR/EXE_CMD
/ARM_TB/ARM_PROCESSOR/Dest
Now
15050 ps
Cursor 1
14510 ps
13900 ps
14000 ps
14100 ps
14200 ps
14300 ps
14400 ps
14500 ps
14510 ps

بدون کش


/ARM_TB/ARM_PROCESSOR/clk
/ARM_TB/ARM_PROCESSOR/rst
/ARM_TB/ARM_PROCESSOR/switch
/ARM_TB/ARM_PROCESSOR/freeze
/ARM_TB/ARM_PROCESSOR/flush
/ARM_TB/ARM_PROCESSOR/Branch_taken
/ARM_TB/ARM_PROCESSOR/hazard
/ARM_TB/ARM_PROCESSOR/BranchAdder
/ARM_TB/ARM_PROCESSOR/PC
/ARM_TB/ARM_PROCESSOR/Instruction
/ARM_TB/ARM_PROCESSOR/PC_to_ID
/ARM_TB/ARM_PROCESSOR/Instruction_to_ID
/ARM_TB/ARM_PROCESSOR/PC_to_EXE
/ARM_TB/ARM_PROCESSOR/Result_WB
/ARM_TB/ARM_PROCESSOR/VAL_Rn
/ARM_TB/ARM_PROCESSOR/VAL_Rm
/ARM_TB/ARM_PROCESSOR/VAL_Rn_IDO
/ARM_TB/ARM_PROCESSOR/VAL_Rm_IDO
/ARM_TB/ARM_PROCESSOR/ALU_res
/ARM_TB/ARM_PROCESSOR/VAL_Rm_EXEO
/ARM_TB/ARM_PROCESSOR/ALU_res_EXEO
/ARM_TB/ARM_PROCESSOR/ALU_res_MEMO
/ARM_TB/ARM_PROCESSOR/val_rm_o
/ARM_TB/ARM_PROCESSOR/SRAM_DQ
/ARM_TB/ARM_PROCESSOR/Mem_read_valu...
/ARM_TB/ARM_PROCESSOR/cache_rdata
/ARM_TB/ARM_PROCESSOR/sram_address
/ARM_TB/ARM_PROCESSOR/sram_wdata
/ARM_TB/ARM_PROCESSOR/sram_rdata
/ARM_TB/ARM_PROCESSOR/Signed_imm_24
/ARM_TB/ARM_PROCESSOR/Signed_imm_24...
/ARM_TB/ARM_PROCESSOR/SRAM_ADDR
/ARM_TB/ARM_PROCESSOR/Shift_operand
/ARM_TB/ARM_PROCESSOR/Shift_operand_...
/ARM_TB/ARM_PROCESSOR/Dest_wb
Now
15050 ps
Cursor 1
5430 ps
5200 ps
5300 ps
5400 ps
5430 ps

با کش