دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

پروژه دوم درس ریزپردازنده و زبان اسمبلی

عنوان پروژه:

شبیه‌ساز حافظه نهان (cache simulator)

نگارش :

لیلاالسادات محسنی

شماره دانشجویی : ۴۰۱۳۱۰۴۴

استاد درس :

دکتر حامد فربه

تاریخ تحویل :

۱۴۰۴/۰۵/۰۷

فهرست مطالب

در این پروژه به شبیه‌سازی حافظه نهان با استفاده از الگوریتم‌های جایگزینی می‌پردازیم.حالت مدنظر در این پروژه 2-way set associative می‌باشد.

در ادامه به توضیح بخش بخش پیاده‌سازی می‌پردازیم.

بخش Data :

حافظه اصلی را طبق داکیومنت پروژه برابر با ۲۵۶ ورد ۴ بایتی در نظر می‌گیریم یعنی آدرس های اصلی بین ۰ تا ۲۵۵ قرار می‌گیرند.

دنباله آدرس های ورودی برابر (5, 12, 13, 17, 4, 12, 13, 17, 2, 13, 19, 43, 61, 19) درنظر گرفته شده است و همچنین تست و اسکرین شات های پروژه بر مبنای آن انجام شده است.

تعداد بلاک های cache برابر ۸ است و چون پیاده سازی کد براساس 2-way set associative است پس ما یک cache با ۴ ست داریم و هرست دارای دو way هست. همچنین دیتای cache به اندازه ۳۲ بایت درنظر گرفته است چون ۸ بلاک است و هر بلاک ۴ بایت.

MAU_Arr یک آرایه برای نگهداری آدرس آخرین استفاده‌شده از هر set در الگوریتم‌های MRU و LRU می‌باشد. ۴ست داریم و هر ست ۴ بایت می‌باشد.

MRU_Index برای هر ست مشخص می‌کند که آخرین way استفاده‌شده کدوم way بوده است یعنی درواقع ۴ خانه برای ۴ ست داریم. هر ست مقدار یک یا دو دارد که مثلا way1 (۱) یا way2(۲) آخرین بار استفاده شده است.

LFU_Arr برای الگوریتم‌هایLFU/MFU ، این آرایه شمارنده‌ای برای تعداد استفاده از هر way ذخیره می‌کند . هر ست دوشمارنده دارد یکی برای way1 و یکی way2 .۴ست داریم و هرست ۴ بایت و دو شمارنده برای همین ۳۲ بایت فضا درنظر گرفته شده است.(یا به عبارتی ۸ بلاک cache داریم و هربلاک ۴ بایت است)

lfsr_seed مقدار اولیه برای تولید اعداد تصادفی در الگوریتم random می‌باشد که تصمیم بگیرد آدرس در way1 باشد یا way2.

بخش start :

در این بخش مقداردهی اولیه رجیسترها می‌پردازیم.رجیستر R0 برای شمارش تعداد hit ها ، R1 برای شمارش تعداد miss ها، R2 برای درصد hit_rate، R3 برای پیمایش آرایه آدرس ها (اندیس) ،R11 فلگ مشخص‌کننده نوع الگوریتم یعنی برای FIFO مقدار ۰، LRU مقدار ۱،MRU مقدار ۲،LFU مقدار ۳، MFU مقدار ۴ و RANDOM هم مقدار ۵ در نظر گرفته شده است. همچنین آدرس شروع آرایه در R5 است .

بخش main_loop و continue_loop :

در این قسمت بررسی می‌کنیم که به انتهای آرایه رسیدیم یا نه. اگر رسیده باشیم که به سراغ بخش محاسبه hit_rate می‌رویم در غیر این صورت به ادامه پیمایش در لیبل continue_loop می‌رویم.در این لیبل R3 را در ۴ ضرب می‌کند چون هر ورد ۴ بایت است(یعنی دو بیت شیفت چپ منطقی) و با مقدار R5 که آدرس شروع آرایه بود جمع می‌شود و در R4 لود می‌شود.سپس به بخش check_address می‌رود تا بررسی کند این آدرس hit است یا miss. بعد مقدار R3 را یکی زیاد می‌کند و به سراغ آدرس بعدی می‌رویم.

بخش calc_hit_rate :

در این بخش دو رجیستر R0 و R1 را جمع می‌کنیم و در رجیستر R6 قرار می‌دهیم (تعداد کل Miss وHit) اگر مقدار R6 برابر صفر بود یعنی دنباله آدرس تهی داشتیم به بخش zero_hit_rate می‌رویم که در این بخش مقدار رجیستر R2 را صفر می‌گذاریم یعنی hit_rate ای نداریم. اگر غیر این باشد تعداد hitها را در رجیسترR7 کپی می‌کنیم.عدد ۱۰۰ را در رجیستر R8 قرار می‌دهیم و تعداد hitها را در ۱۰۰ ضرب میکنیم (در واقع صورت کسر را داریم محاسبه می‌کنیم).

در بخش div_loop مقدار رجیستر R6 و R7 را مقایسه می‌کنیم اگر R7 کوچکتر ازR6 باشد اتمام فرآیند تقسیم تمام می‌شود به عبارتی تا وقتی که تعداد hit ها ضرب در ۱۰۰ بیشتر ازtotal است ازش کم می‌کنیم و شمارش می‌کنیم چندبار کم کرده‌ایم در این صورت درصد محاسبه می‌شود و درR2 ذخیره می‌شود.

$$hit\_rate = \frac{hit\_count}{hit\_count + miss\_count} * 100$$

بخش check_address :

در ابتدا همه رجیسترهایی که قرار است تو این تابع تغییر کنند را ذخیره می‌کنیم تا بعدا با pop برگردند. در رجیستر R5 آدرس شروع cache را لود می‌کنیم. چون cache ۴ ست دارد باید باقی مانده آدرس را بر ۴ محاسبه می‌کنیم که این کار را برای راحتی با and کردن آدرس و عدد ۳ انجام می‌دهیم. در ادامه چون هر ست ۸ بایت دارد مقدار رجیستر R7 را ۸ قرار می‌دهیم. شماره ستی که در مرحله قبل محاسبه کردیم را در ۸ ضرب می‌کنیم تا متوجه شویم چند بایت باید از ابتدای cache جلو برویم تا به ست موردنظر برسیم و آن را درR7 ذخیره می‌کنیم. سپس همین مقدار را با آدرس ابتدای cache جمع می‌کنیم و مجددا درR7 ذخیره می‌کنیم که این اشاره گر به آدرس اولین way ست موردنظر است. در ادامه way1 را در R8 لود می‌کنیم و با آدرس موردنظر مقایسه می‌کنیم اگر برابر بود یعنی آدرس پیدا شده است و باید مقدارR9 را یکی زیاد کنیم که این بعدا برای ثبت شماره way که hit شده است می‌باشد و در ادامه آن به بخش calc_hit می‌رود در غیر این صورت باید way2 و الگوریتم جایگزینی را بررسی کنیم. در همه این الگوریتم ها قبل از اینکه داده ای درج بشه way2 هم بررسی می‌شود.

بخش miss_fifo :

اگر آدرس در way1 پیدا نشد باید بررسی کنیم شاید آدرس در way2 باشد و اگر نه miss حساب کنیم. مقدارR7 را ۴ بایت زیاد می‌کنیم چون قبلا R7 اشاره گر به آدرس way1 بود می‌خواهیم به way2 برویم(یعنی بلوک دوم از همین set). سپس مقدار آن را در R2 لود می‌کنیم با R4 مقایسه می‌کنیم اگر برابر بود یعنی آدرس پیدا شده و hit می‌شود و به بخش calc_hit می‌رویم تا شمارنده hit اضافه شود اگر نبود miss کامل در کل ست است و به بخش miss_plus می‌رویم.

بخش miss_MRU_LRU :

در ابتدا مقدار R7 را ۴ بایت جلو می‌بریم سپس آن را در R2 لود می‌کنیم مقایسه می‌کنیم آدرس فعلی درway2 است یا نه.اگر برابر بود بعنی آدرس فعلی در way2 است و hit داریم و مقدار R9 را ۲ قرار می‌دهیم چون در ادامه می‌خواهیم بدانیم که این hit در way1 بوده یا way2 (برای الگوریتم های LRU، MRU، LFU، MFU). اگر hit در R2 باشد به بخش calc_hit می‌رویم.

در ادامه آدرس آرایه MRU_Arr را در R10 لود می‌کنیم این آرایه برای هر ست آدرس بلوکی که اخیرا استفاده شده را نگه میدارد. آدرس MRU_Index را در R2 لود می‌کنیم. MRU_Index یک آرایه است که ۴ خونه دارد چون cache ۴ ست دارد وبرای هر ست نگه میدارد که آخرین way استفاده شده در آن ست کدام بوده است. از مقدار R7 ۴ بایت کم می‌کنیم چون قبلا اضافه شده بود حالا می‌خواهیم به ابتدای ست برگردیم. مقدار آن را در R9 لود می‌کند و مقایسه می‌کند اگر مقدار آن صفر باشد یعنی استفاده نشده است و آدرسش را در MRU_Arr ذخیره می‌کند به عبارتی می‌گوییم آخرین way استفاده شده در این ست همین است. اگر way1 خالی بود شماره آن را ۱ در نظر می‌گیریم و در R9 قرار می‌دهیم. شماره way را در MRU_Index ذخیره می‌کنیم یعنی در این ست آخرین way استفاده شده way1 بوده است. اگر way1 خالی باشد داده جدید در همان way1 درج می‌شود.

آدرس way1 را در R7 لود کرده بودیم ۴ بایت اضافه می‌کنیم تا به آدرس way2 برویم. مقدار آن را در R9 لود می‌کنیم بررسی می‌کنیم آیا این بلاک خالی هست یا نه اگر نباشد از آرایه MRU_Arr مقدار قدیمی یا همان آخرین آدرس را در R7 بارگذاری می‌کنیم. چون MRU_Arr به ازای هر ست یک مقدار ۴ بایتی ذخیره کرده، با ضرب در۴ به مکان دقیق ست R6 می‌رویم. اگر بلاک دوم خالی باشد آدرس فعلی یعنیR7 را را به‌عنوان مقدار MRU در ست فعلی ذخیره می‌کنیم . همچنین اگر بلاک دوم خالی بود مقدار ۲ را درR9 می‌ریزیم این مقدار یعنی اینکه از بلاک دوم به عنوان MRU در نظر گرفته شود. اگر بلاک دوم خالی بود، مقدار MRU_Index ست R6 را برابر ۲ قرار می‌دهیم. (یعنی الان بلاک دوم دارای داده شده) و در آخر به بخش miss_plus می‌رویم.

بخش miss_LFU_MFU :

به R7 ۴بایت اضافه می‌شود چون می‌خواهیم به way2 از ست فعلی برویم.مقدار ذخیره شده در این آدرس را در R2 لود می‌کنیم. این مقدار را با آدرس درخواستی R4 مقایسه می‌کند.بررسی می‌کند آیا این بلاک در cache هست یا نه. اگر برابر بودند یعنی hit رخ داده است و مقدار ۲ در R9 قرار داده می‌شود و سپس به بخش calc_hit می‌رود تا شمارش hit را انجام دهد.

در ادامه در R10 آدرس آرایه شمارنده‌های LFU لود می‌کنیم . این آرایه تعداد استفاده هر بلاک در هر ست را نگه می‌دارد.سپس ۴ بایت از R7 کم می‌کنیم چون می‌خواهیم به way1 برگردیم و بررسی کنیم خالی هست یا نه.مقدار بلاک را در R2 لود می‌کنیم با صفر مقایسه می‌کنیم اگر برابر بود یعنی خالی است و مقدار R2 را برابر یک قرار می‌دهیم یعنی شمارنده فرکانس برای این بلاک را از ۰ به ۱ تنظیم می‌کنیم. اگر way1 خالی بود مقدار۱ را در موقعیت مربوطه در LFU_Arr می‌نویسیم.فرکانس way1 ست مربوطه را می‌نویسیم. اگر way0 خالی بود نیاز به بررسی way1 نیست و به miss_plus برای درج داده جدید می‌رویم. اگرway1 خالی نبود به way2 می‌رویم یعنی R7 را ۴ بایت اضافه می‌کنیم سپس مقدار آن را در R2 لود می‌کنیم بررسی می‌کنیم که خالی است یا نه اگر خالی بود مقدار شمارنده فرکانس آن را ۱ قرار می‌دهیم. سپس آدرس R10 را به حالت اول برمی‌گرداند درنهایت به بخش misss_plus می‌رود تا عملیات درج داده ادامه یابد.

بخش miss_plus :

این بخش پس از یک miss در cache کد باید تصمیم بگیرد که با استفاده از کدام الگوریتم جایگزینی یک بلاک cache را با یک بلاک cache جدید جایگزین کند.در این بخش شمارنده R1 را برای شمارش تعداد miss ها زیاد می‌کنیم سپس مقدار R11 را با مقادیر مختلف ۰تا ۵ مقایسه می‌کنیم تا ببینیم باید کدام الگوریتم را اجرا کنیم.

بخش calc_hit :

این بخش زمانی اجرا می‌شود که در برنامه تشخیص داده شده است داده موردنظر در cache موجود بوده است حالا باید بر اساس نوع الگوریتم جایگزینی اطلاعات cache به روزرسانی بشود.اگر مقدار R11 برابر ۳ یا ۴ باشد باید به بخش update_lfu_mfu برویم که در این بخش آدرس LFU_Arr که شمارنده فرکانس استفاده برای هر way از هر ست رو نگه می‌دارد در R10 لود می‌کنیم (هم برای way1 و هم برای way2)

رجیستر R9 را با مقدار یک مقایسه می‌کنیم اگر hit درway1 بود به بخش inc_way1_count می‌رویم در غیر این صورت آدرس شمارنده را ۴ بایت جابه‌جا می‌کنیم. در بخش inc_way2_count مقدار شمارنده فعلی را می‌خواند

یکی به آن اضافه می‌کند و دوباره ذخیره می‌شود. برای way1 هم مشابه قسمت قبل. در مرحله بعد به hit_finalize می‌رویم که شمارنده تعداد hit ها را زیاد می‌کند. سپس به بخش push_pop می‌رویم و رجیسترها را pop می‌کنیم.

برای الگوریتم‌های MRU/LRU :

R10 با آدرس پایه آرایه MRU_Arr لود می‌شود سپس R7 که آدرس بلوک فعلی است که hit شده است در MRU_Arr ذخیره می‌شود.سپس آدرس پایه MRU_Index در R2 لود می‌شود و بعد شماره way که hit در آن رخ داده ذخیره می‌شود. اینکار باعث می‌شود که الگوریتم MRU/LRU بدونه آخرین بلوک استفاده‌شده در این set چه بوده و در کدام way قرار داشته است.

بخش insert_fifo :

مقدار موجود در آدرس حافظه‌ای که در رجیستر R7 قرار دارد را در R9 بارگذاری می‌کنیم سپس مقدار R7 را ۴ بایت کاهش می‌دهیم یعنی به آدرس way قبلی در همان ست اشاره می‌کند مقداری که در R9 است در آدرس جدید R7 ذخیره می‌کند.دوباره R7 را ۴ واحد اضافه می‌کنیم تا به آدرس way اول برگردد مقدار آدرس جدیدی که قرار است درج شود (که در R4 است) را در این آدرس (way اول) ذخیره می‌کنیم.

بخش insert_MRU :

در این بخش ابتدا آدرس بلوک cache مناسب برای جایگزینی از آرایه MRU در شاخص ست R6 لود می‌شود و در رجیستر R7 قرار می‌گیرد. سپس مقدار آدرس جدید (R4) در آن موقعیت خاص از cache ذخیره می‌شود.

بخش insert_LRU و LRU_OK :

در این بخش ابتدا از رجیستر R2 اندیسی گرفته می‌شود تا مقدار قبلی استفاده‌شده از کش در R9 لود شود. سپس آدرس آرایه LRU (در R10) برای ست مربوطه لود می‌شود و داده داخل آن در R7 قرار می‌گیرد. اگر مقدار فعلی حافظه cache صفر باشد ، مقدار جدیددر همان جا قرار می‌گیرد. در غیر این صورت، با توجه به مقدارR9 ، بررسی می‌کنیم که آیا این اولین استفاده بوده یا نه؛ در حالت اول داده به مکان بعدی (آدرس بزرگ‌تر) و در غیر این صورت به مکان قبلی (آدرس کوچک‌تر) نوشته می‌شود.

بخش insert_LFU و insert_direct_LFU :

در این بخش ابتدا مقدار way1 بررسی می‌شود؛ اگر خالی باشد، آدرس جدید مستقیماً در آن نوشته می‌شود. اما اگر way1 پر باشد، ابتدا به سراغ آرایه‌ی LFU_Arrمی‌رود تا شمارش تعداد دفعات استفاده‌ی دو way را بخواند. سپس

با مقایسه‌ی این دو فرکانس، تصمیم می‌گیرد که داده جدید در کدام way نوشته شود: اگر $freq1 \leq freq2$باشد، جایگزینی در way1 انجام می‌شود؛ در غیر این صورت، داده در way2 نوشته خواهد شد.

بخش insert_MFU و insert_MFU_direct :

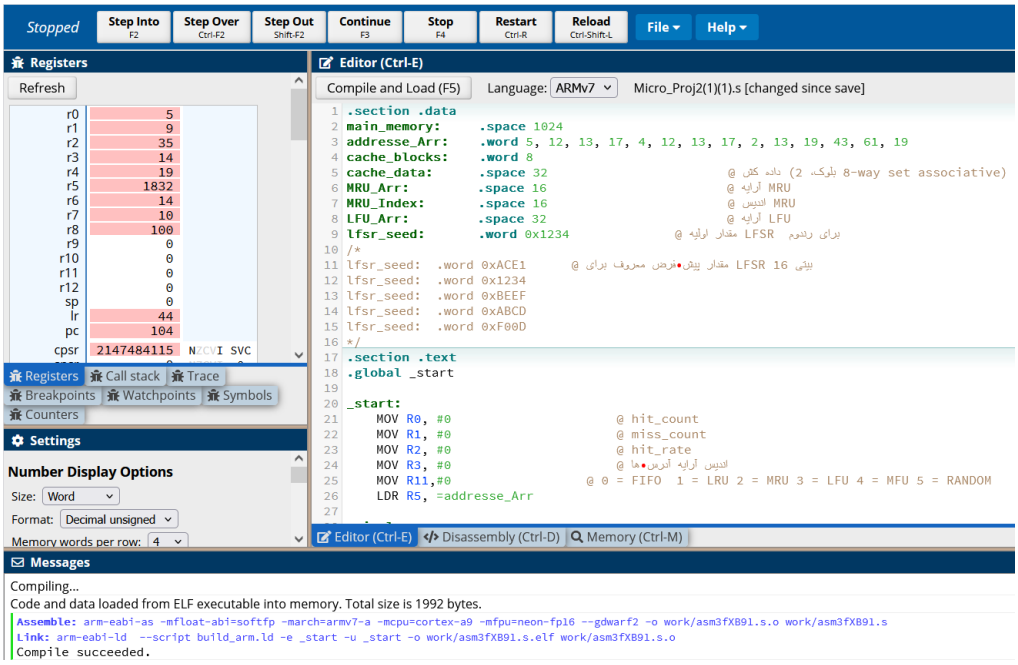
در این بخش درابتدا مقدار way1 بررسی می‌شود؛ اگر خالی باشد ، آدرس جدید مستقیماً در آن درج می‌گردد (insert_MFU_direct)در غیر این صورت، تعداد دفعات استفاده‌ی هر way از آرایه‌ی LFU_Arr خوانده می‌شود. سپس اگر $freq1 \geq freq2$ باشد، چون way1 بیشتر استفاده شده است، مقدار جدید در آن قرار می‌گیرد؛ در غیر این صورت، درج در way2 انجام می‌شود.

بخش insert_random و write_way1 و write_way0 :

در ابتدا شمارنده‌ی miss افزایش می‌یابد. سپس با استفاده از آدرس داده اندیس ست cache محاسبه می‌شود ، و محل دقیق ست در cache_data تعیین می‌شود. برای تولید تصمیم تصادفی، یک مقدار از متغیر lfsr_seed به‌عنوان بیت اولیه (Seed) بارگذاری می‌شود و با اعمال عملیات XOR و چرخش بیت‌ها، یک بیت تصادفی ساخته می‌شود. این بیت مشخص می‌کند که داده در way1 ذخیره شود یا way2. اگر بیت ۰ باشد، در way1 درج می‌شود؛ در غیر این صورت به آدرس بعدی (way2) می‌رود و در آن نوشته می‌شود.

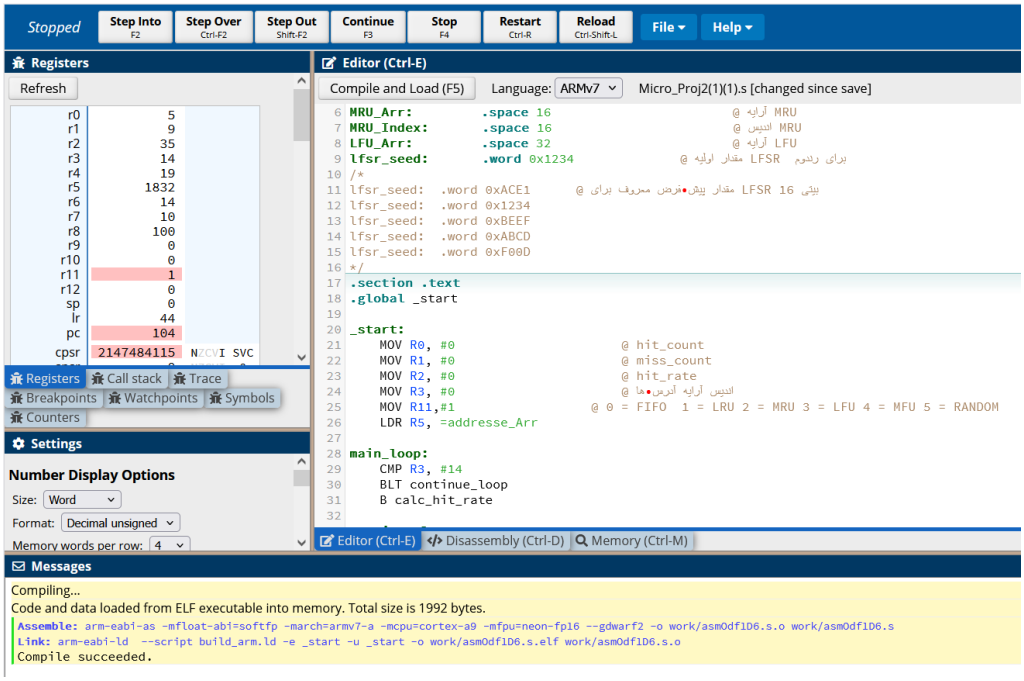
اسکرین شات ها :

Fifo :



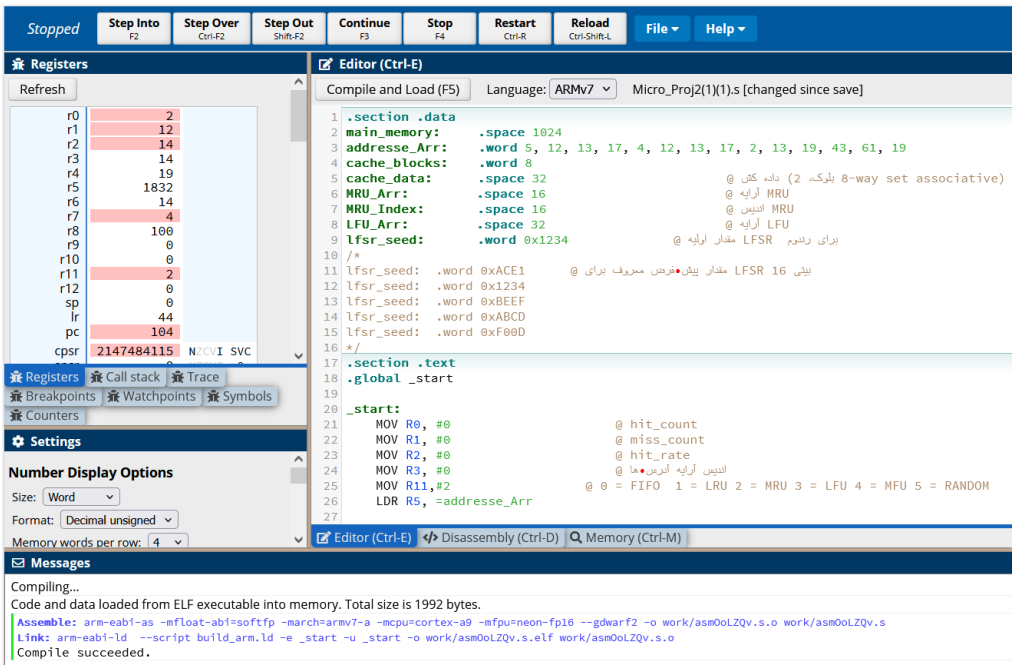
Miss/hit	%4	آدرس
M	۱	۵
M	۰	۱۲
M	۱	۱۳
M	۱	۱۷
M	۰	۴
H	۰	۱۲
H	۱	۱۳
H	۱	۱۷
M	۲	۲
H	۱	۱۳
M	۳	۱۹
M	۳	۴۳
M	۱	۶۱
H	۳	۱۹

:LRU



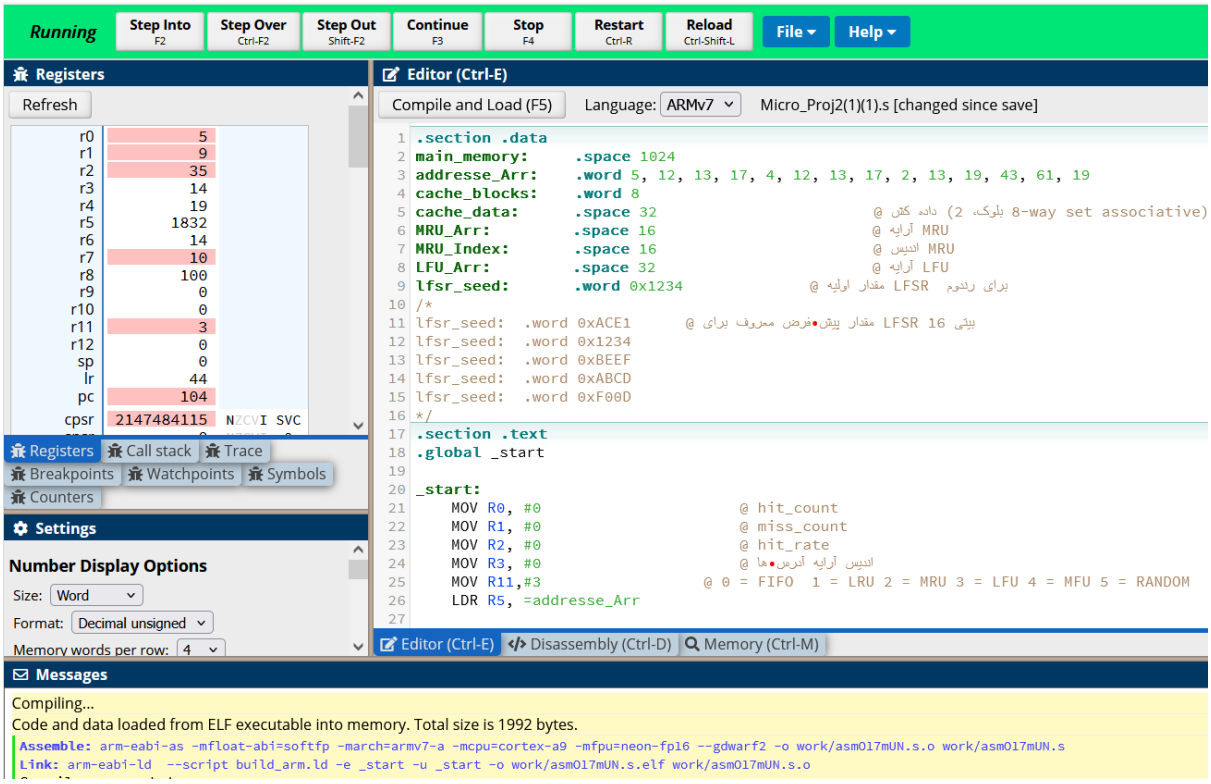
Miss/hit	%4	آدرس
M	۱	۵
M	۰	۱۲
M	۱	۱۳
M	۱	۱۷
M	۰	۴
H	۰	۱۲
H	۱	۱۳
H	۱	۱۷
M	۲	۲
H	۱	۱۳
M	۳	۱۹
M	۳	۴۳
M	۱	۶۱
H	۳	۱۹

: MRU



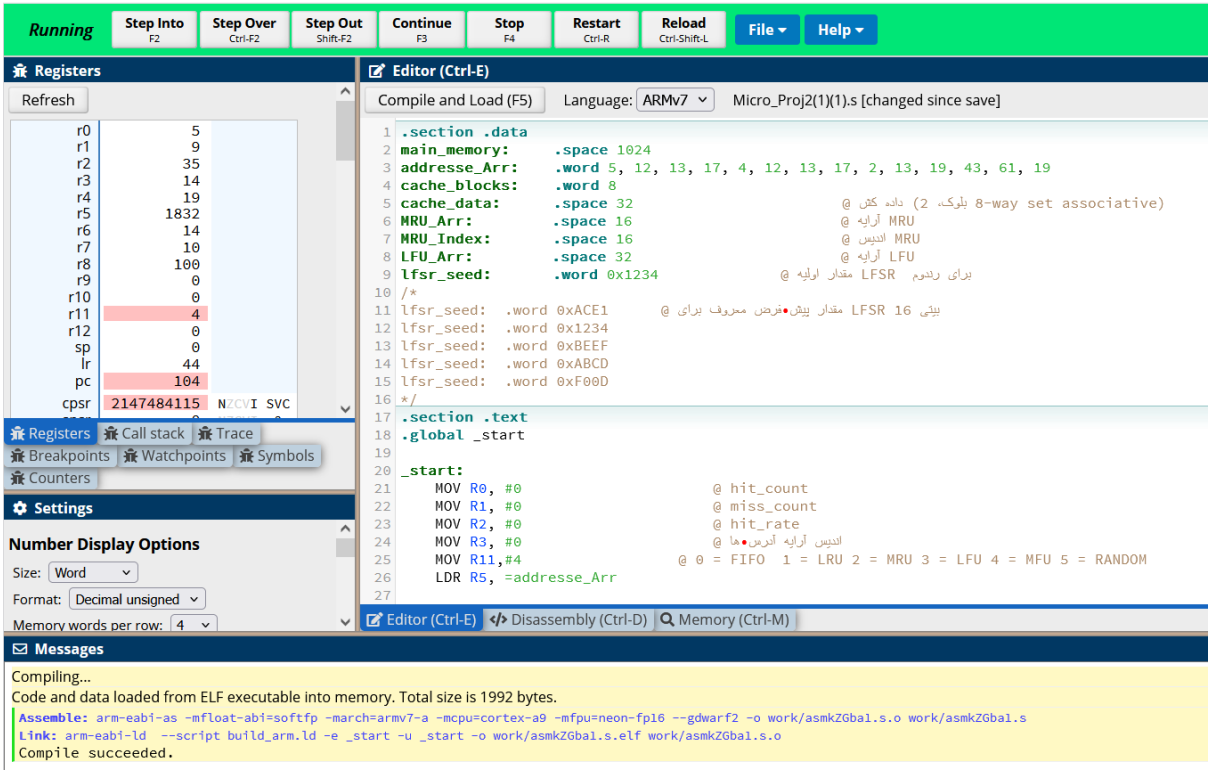
Miss/hit	%4	آدرس
M	۱	۵
M	۰	۱۲
M	۱	۱۳
M	۱	۱۷
M	۰	۴
H	۰	۱۲
M	۱	۱۳
M	۱	۱۷
M	۲	۲
M	۱	۱۳
M	۳	۱۹
M	۳	۴۳
M	۱	۶۱
H	۳	۱۹

: LFU



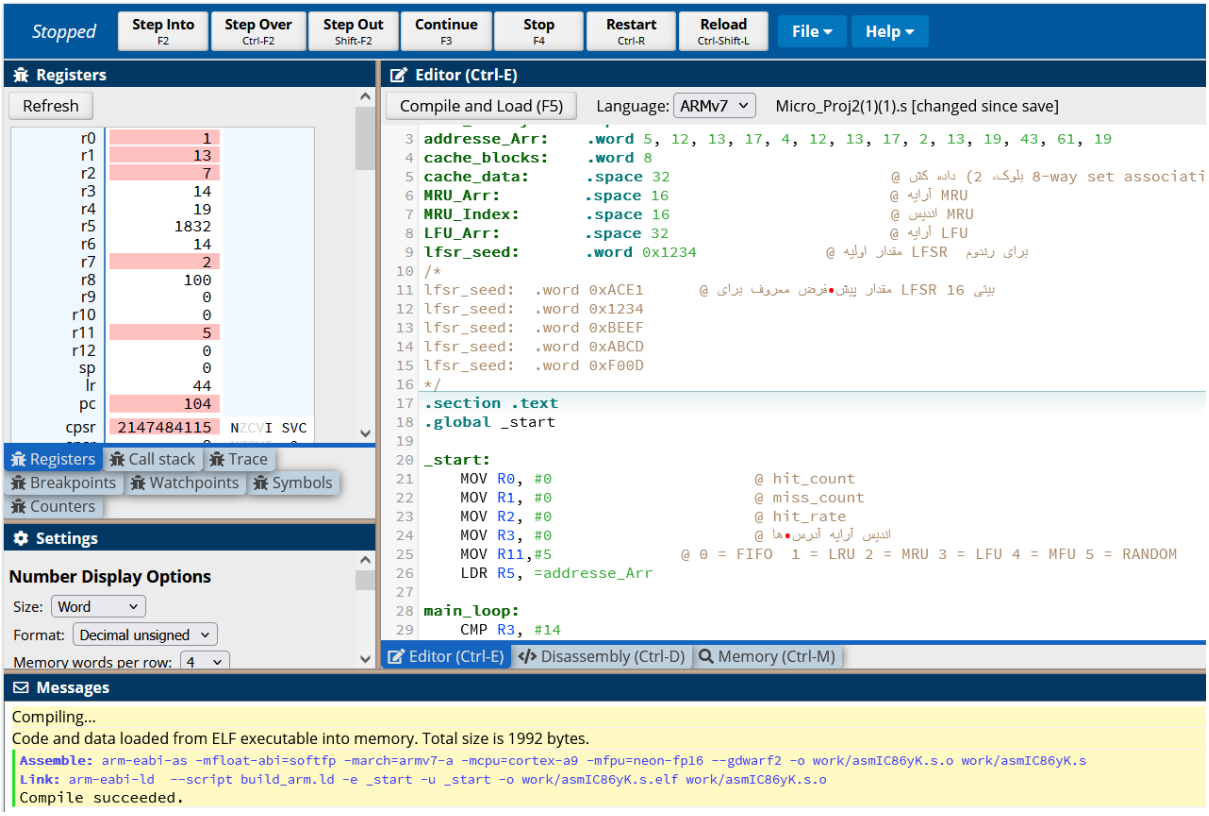
Miss/hit	%4	آدرس
M	۱	۵
M	۰	۱۲
M	۱	۱۳
M	۱	۱۷
M	۰	۴
H	۰	۱۲
H	۱	۱۳
H	۱	۱۷
M	۲	۲
H	۱	۱۳
M	۳	۱۹
M	۳	۴۳
M	۱	۶۱
H	۳	۱۹

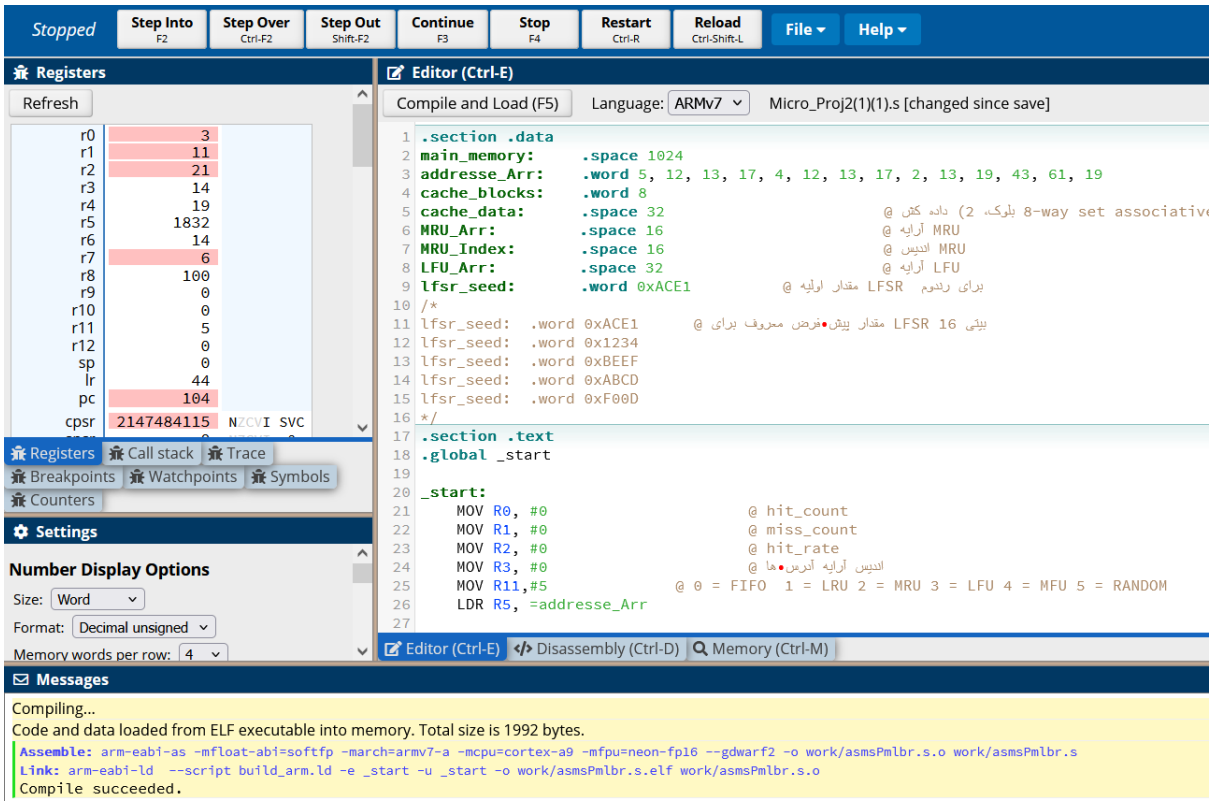
: MFU



Miss/hit	%4	آدرس
M	۱	۵
M	۰	۱۲
M	۱	۱۳
M	۱	۱۷
M	۰	۴
H	۰	۱۲
H	۱	۱۳
H	۱	۱۷
M	۲	۲
H	۱	۱۳
M	۳	۱۹
M	۳	۴۳
M	۱	۶۱
H	۳	۱۹

: RANDOM


Stopped
Step Into F2
Step Over Ctrl-F2
Step Out Shift-F2
Continue F3
Stop F4
Restart Ctrl-R
Reload Ctrl-Shift-L
File
Help
Registers
Refresh
r0 1
r1 13
r2 7
r3 14
r4 19
r5 1832
r6 14
r7 2
r8 100
r9 0
r10 0
r11 5
r12 0
sp 0
lr 44
pc 104
cpsr 2147484115 NZCVI SVC
Registers Call stack Trace
Breakpoints Watchpoints Symbols
Counters
Settings
Number Display Options
Size: Word
Format: Decimal unsigned
Memory words per row: 4
Editor (Ctrl-E)
Compile and Load (F5)
Language: ARMv7
Micro_Proj2(1)(1).s [changed since save]
addresse_Arr: .word 5, 12, 13, 17, 4, 12, 13, 17, 2, 13, 19, 43, 61, 19
cache_blocks: .word 8
cache_data: .space 32 @ داده کش (2 بلوک، 8-way set associati
MRU_Arr: .space 16 @ آرایه MRU
MRU_Index: .space 16 @ اندیس MRU
LFU_Arr: .space 32 @ آرایه LFU
lfsr_seed: .word 0x1234 @ مقدار اولیه LFSR برای رندوم
/*
lfsr_seed: .word 0xACE1 @ بیتی 16 مقدار پیش‌فرض محروف برای
lfsr_seed: .word 0x1234
lfsr_seed: .word 0xBEEF
lfsr_seed: .word 0xABCD
lfsr_seed: .word 0xF00D
*/
.section .text
.global _start
_start:
MOV R0, #0 @ hit_count
MOV R1, #0 @ miss_count
MOV R2, #0 @ hit_rate
MOV R3, #0 @ اندیس آرایه آدرس‌ها
MOV R11,#5 @ 0 = FIFO 1 = LRU 2 = MRU 3 = LFU 4 = MFU 5 = RANDOM
LDR R5, =addresse_Arr
main_loop:
CMP R3, #14
Editor (Ctrl-E) Disassembly (Ctrl-D) Memory (Ctrl-M)
Messages
Compiling...
Code and data loaded from ELF executable into memory. Total size is 1992 bytes.
Assemble: arm-eabi-as -mfloat-abi=softfp -march=armv7-a -mcpu=cortex-a9 -mfpu=neon-fp16 --gdwarf2 -o work/asmIC86yK.s.o work/asmIC86yK.s
Link: arm-eabi-ld --script build_arm.ld -e _start -u _start -o work/asmIC86yK.s.elf work/asmIC86yK.s.o
Compile succeeded.


Stopped
Step Into F2
Step Over Ctrl-F2
Step Out Shift-F2
Continue F3
Stop F4
Restart Ctrl-R
Reload Ctrl-Shift-L
File
Help
Registers
Refresh
r0 3
r1 11
r2 21
r3 14
r4 19
r5 1832
r6 14
r7 6
r8 100
r9 0
r10 0
r11 5
r12 0
sp 0
lr 44
pc 104
cpsr 2147484115 NZCVI SVC
Registers Call stack Trace
Breakpoints Watchpoints Symbols
Counters
Settings
Number Display Options
Size: Word
Format: Decimal unsigned
Memory words per row: 4
Editor (Ctrl-E)
Compile and Load (F5)
Language: ARMv7
Micro_Proj2(1)(1).s [changed since save]
.section .data
main_memory: .space 1024
addresse_Arr: .word 5, 12, 13, 17, 4, 12, 13, 17, 2, 13, 19, 43, 61, 19
cache_blocks: .word 8
cache_data: .space 32 @ داده کش (2 بلوک، 8-way set associative
MRU_Arr: .space 16 @ آرایه MRU
MRU_Index: .space 16 @ اندیس MRU
LFU_Arr: .space 32 @ آرایه LFU
lfsr_seed: .word 0xACE1 @ مقدار اولیه LFSR برای رندوم
/*
lfsr_seed: .word 0xACE1 @ بیتی 16 مقدار پیش‌فرض محروف برای
lfsr_seed: .word 0x1234
lfsr_seed: .word 0xBEEF
lfsr_seed: .word 0xABCD
lfsr_seed: .word 0xF00D
*/
.section .text
.global _start
_start:
MOV R0, #0 @ hit_count
MOV R1, #0 @ miss_count
MOV R2, #0 @ hit_rate
MOV R3, #0 @ اندیس آرایه آدرس‌ها
MOV R11,#5 @ 0 = FIFO 1 = LRU 2 = MRU 3 = LFU 4 = MFU 5 = RANDOM
LDR R5, =addresse_Arr
Editor (Ctrl-E) Disassembly (Ctrl-D) Memory (Ctrl-M)
Messages
Compiling...
Code and data loaded from ELF executable into memory. Total size is 1992 bytes.
Assemble: arm-eabi-as -mfloat-abi=softfp -march=armv7-a -mcpu=cortex-a9 -mfpu=neon-fp16 --gdwarf2 -o work/asmsPmlbr.s.o work/asmsPmlbr.s
Link: arm-eabi-ld --script build_arm.ld -e _start -u _start -o work/asmsPmlbr.s.elf work/asmsPmlbr.s.o
Compile succeeded.