**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

**KHOA ĐIỆN – ĐIỆN TỬ**

**🙦-----🕮-----🙥**

A blue and black logo

Description automatically generated

**Báo cáo thực tập tốt nghiệp**

**Thiết kế bộ nhớ đệm và thuật toán thay thế LRU cho pipeline RV32I CPU**

Lớp: EE3365\_L01

GHVD: Trần Hoàng Linh

Trợ giảng: Cao Xuân Hải

|  |  |
| --- | --- |
| Huỳnh Bảo Duy | 2110078 |
| Nguyễn Đắc Tâm | 2114714 |

TP.HCM, tháng 8 năm 2024

1. **Lý do chọn đề tài.**

Ở thiết kế pipeline processor trong thực tế sẽ là một system off chip, cụ thể là bộ memory sẽ nằm bên ngoài so với bộ xử lý. Do đó việc truy cập vào bộ nhớ trong thực tế sẽ tốn rất nhiều time cycle dẫn đến tốc độ làm việc của pipeline processor sẽ rất chậm. Để khắc phục việc đó, bộ nhớ đệm được thiết kế kết hợp với bộ xử lý như là một system on chip. Bộ nhớ đệm có kích thước nhỏ hơn so với bộ nhớ chính và có chức năng lưu trữ và truy xuất dữ liệu cho bộ nhớ chính.

Ở cấp độ mô phỏng, việc truy cập vào dữ liệu từ bộ nhớ đệm và bộ nhớ chính sẽ cùng tốn 1 xung cycle. Do đó về mặt kiểm tra sự cải thiện về tốc độ của chương trình là không chuẩn xác. Ở đề tài này, chúng em sẽ kiểm tra tính đúng đắn của bộ Cache khi được thiết kế chèn vào CPU pipeline 5 tầng, kết hợp với

1. **Thiết kế bộ Direct Mapped Cache.**
2. **Tổng quan thiết kế bộ Cache**

Bộ nhớ cache có vai trò lưu và đọc dữ liệu từ các địa chỉ cấp từ EX\_stage, thay thế cho Memory. Khi một địa chỉ được truy xuất, dữ liệu tại địa chỉ đó sẽ được lưu lại trong bộ nhớ đệm, khi địa chỉ đó được truy xuất lại thì bộ nhớ đệm đã lưu sẵn dữ liệu và sử dụng mà không cần phải tiếp cận bộ nhớ chính, việc truy xuất khi đã có sẵn dữ liệu là gọi là hit. Ngược lại khi truy xuất một địa chỉ chưa có dữ liệu được lưu trữ trong bộ nhớ đệm thì câu lệnh truy xuất sẽ đi đến bộ nhớ chính để lấy dữ liệu, sau đó lưu trữ lại trong bộ nhớ đệm, lần truy xuất này gọi là miss.

Ở miss event, có hai trường hợp xảy ra.

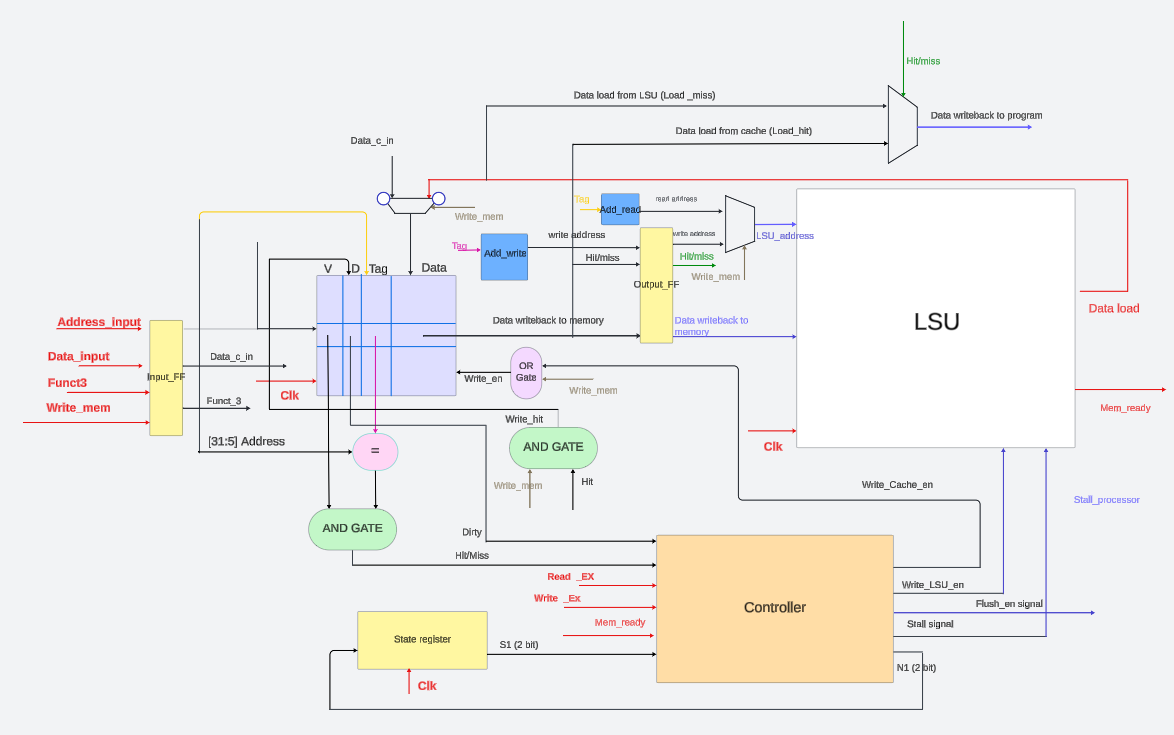
* Dữ liệu tại địa chỉ đang có trong bộ nhớ đệm chưa bị thay đổi từ khi truy xuất ra từ bộ nhớ chính, hoặc là chưa có dữ liệu tại vùng địa chỉ đó ở bộ nhớ đệm. Khi đó chỉ cần cập nhật dữ liệu mới và địa chỉ mới tại bộ nhớ đệm mà không cần cập nhật lại dữ liệu vào bộ nhớ chính
* Dữ liệu tại địa chỉ có trong bộ nhớ đệm đã bị thay đổi từ khi truy xuất ra từ bộ nhớ chính. Khi đó cần có cơ chế cập nhật lại dữ liệu vào bộ nhớ chính trước khi đi vào bộ nhớ đệm.

Để làm được những điều đó, bộ cache bao gồm 1 table chứa các dữ liệu bao gồm 32-bit data, bit tag, 1 bit valid, 1 bit dirty. Trong đó:

* 32-bit data dùng để lưu dữ liệu
* Bit tag dung để lưu các bit giá trị lớn từ 32-bit địa chỉ truy xuất, số lượng bit tag tùy vào độ lớn của table mà mình mong muốn. Ví dụ: khi tạo ra table 8 ngõ vào (3 bit) thì cần dùng 27-bit cao của địa chỉ làm tag (do offset địa chỉ lệnh là 4 nên bỏ qua 2 bit cuối của địa chỉ). Các bit tag sẽ được so sánh với số lượng bit cao tương ứng của địa chỉ truy xuất để phát hiện là hit/miss event.
* 1 bit valid có chức năng cập nhật việc ghi thành công dữ liệu vào bộ nhớ đệm, bit valid kết hợp với bit so sánh của phần tag sẽ cho ra tín hiệu hit hoặc miss
* 1 bit dirty để cập nhật sự thay đổi của dữ liệu so với dữ liệu ở bộ nhớ chính. Bit dirty là 1 thì khi có miss event, cần cập nhật lại giá trị vào bộ nhớ chính trước khi vào hoạt động của bộ nhớ đệm và ngược lại.

Ngoài ra để kiểm soát tốt hoạt động của bộ nhớ đệm, cần có bộ cache controller để kiểm soát các tín hiệu. Khi có lệnh truy xuất bộ nhớ, hệ thống CPU cần có tín hiệu tạm dừng hoạt động của CPU để thực hiện các thao tác ở bộ nhớ đệm, sau đó sẽ hoạt động lại như bình thường. Bộ cache controller sẽ làm nhiệm vụ đó bằng cách thiết kế theo máy trạng thái hoạt động thay đổi theo xung clock cạnh lên.

Direct map cache có 32 entries tương đương với 5 bit -index mỗi entry chứa được 32 bit dữ liệu. Như vậy kích thước của bộ cache nhóm chọn là 128Kb.



*Datapath của MEM\_STAGE CPU khi có bộ nhớ đệm*

1. **Thiết kế theo kiểu Moore**

Từ sơ đồ datapath trên, chọn thiết kế bộ cache controller theo phong cách máy trạng thái Moore, lấy các tín hiệu control đồng bộ với xung clock, gồm các trạng thái như sơ đồ dưới đây:

A diagram of a process

Description automatically generated

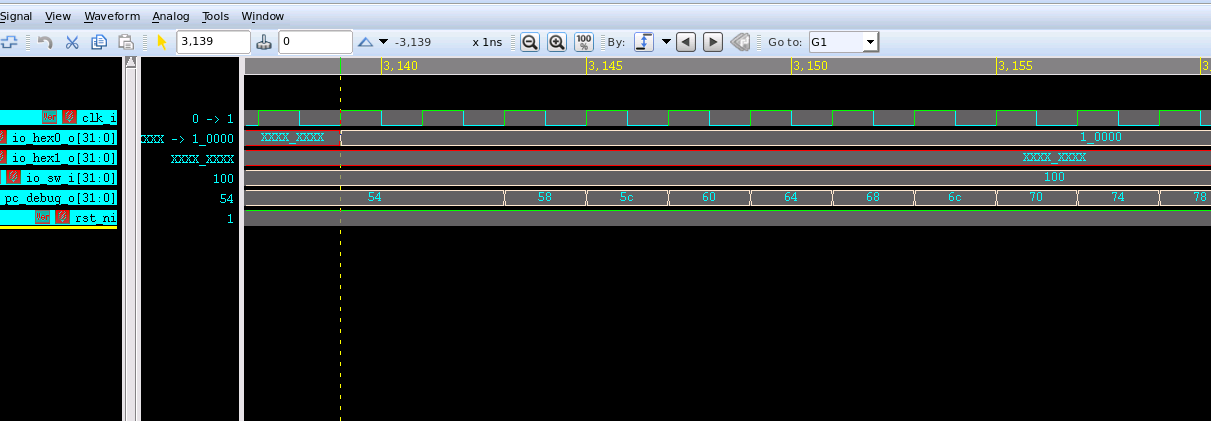
Trong đó:

Idle: Trạng thái chờ các câu lệnh hợp lệ để truy xuất bộ nhớ (load và store instruction).

Check and update: Trạng thái kiểm tra dữ liệu của địa chỉ truy xuất có trong bộ nhớ đêm hay không và cập nhật trạng thái của địa chỉ đó ở bộ nhớ đệm, xác định hit/miss và bit dirty. Ở trạng thái này cache controller cấp tín hiệu cho phép viết vào table cache để cập nhật các giá trị valid, dirty, ngoài ra tín hiệu stall để dừng hoạt động CPU cũng được cập nhật.

Write clean: Trạng thái này xảy ra khi miss event và dirty bit =0, cho phép ghi dữ liệu mới vào bộ cache và chờ bộ cache cập nhật để quay lại trạng thái chờ. Ở trạng thái này, bộ controller cấp chân cho phép ghi cho bộ nhớ đệm, tín hiệu dừng cho CPU.

Write dirty: Trạng thái này xảy ra khi miss event và dirty bit =1, cho phép ghi dữ liệu hiện có ở bộ nhớ đệm vào bộ nhớ chính trước khi ghi dữ liệu mới từ bộ nhớ chính vào bộ nhớ đệm, do trong thực tế bộ nhớ chính sẽ cần nhiều thời gian để cập nhật dữ liệu hơn nên ở trạng thái này sẽ chờ tín hiệu sẵn sàng từ bộ nhớ chính để tiếp tục hoạt độn. Ở trạng thái này, bộ controller cấp chân cho phép ghi cho bộ nhớ đệm và bộ nhớ chính, tín hiệu dừng cho CPU.

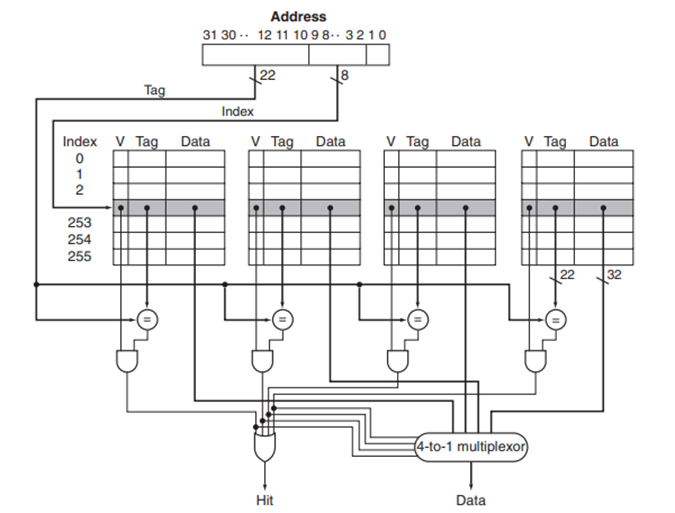
****

1. **Thiết kế máy trạng thái theo kiểu Mealy**

Ở thiết kế này, bộ cache controller được thiết kế theo kiểu máy trạng thái Mealy, lấy các tín hiệu control bất đồng bộ. Sơ đồ máy trạng thái như sau:

1. **Thiết kế bộ 4-set Associativity Cache và thuật toán thay thế LRU.**
2. **4-set Associativity Cache**

Đối với bộ direct map, mỗi giá trị index ngõ vào chỉ chứa được data tại một địa chỉ. Do đó, miss rate còn khá cao. Để khắc phục vấn đề đó, ta cần thay đổi cấu trúc bộ cache bằng cách gia tăng độ lớn của một ngõ vào và giảm số lượng ngõ vào bộ table. Như vậy kích thước tổng của bộ Cache không đổi nhưng độ lớn của một dòng được tăng lên. Đó là thiết kế N-set , M-way cache, với N là số ngõ vào, M là số vùng dữ liệu ứng với M địa chỉ được lưu tại một ngõ vào.



Để làm rõ hơn ta lấy một ví dụ sau:

Một chương trình truy cập lần lượt các địa chỉ như sau: 0,32

**A screenshot of a computer

Description automatically generated**

1. **Thuật toán thay thế LRU**
2. **Ý tưởng của thuật toán**

Ở bộ 4-set associativity 8 ways, có thể làm giảm miss rate so với bộ direct-mapped, tuy nhiên khi có địa chỉ mới truy vào cùng vùng dòng set và cột way, miss event vẫn xảy ra.

A grid of white squares

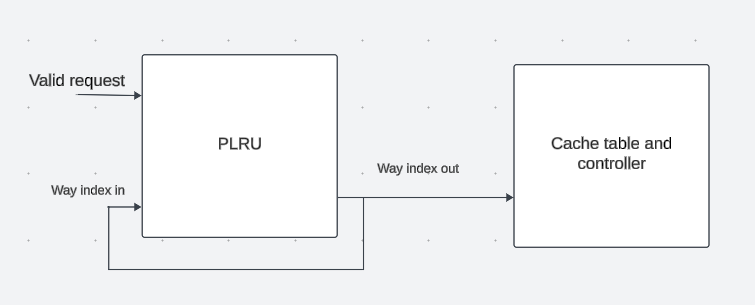
Description automatically generated

Giả sử chương trình truy xuất vị trí (0,1) hai lần khi đó miss event xảy ra

Để khắc phục tình trạng đó cần thiết kế một bộ tính toán way index có cơ chế tự lựa chọn vị trí way. Bộ tính toán này được đặt trước table cache và ngõ vào của bộ cache chỉ cần 2-bit lựa chọn index set. Bộ tính toán cụ thể có hai chức năng chính như sau:

* Tận dụng tối đa số way trong một set để lưu trữ dữ liệu
* Khi đã dùng hết số way trong một set mà vẫn có ngõ vào là set đó thì sẽ thay thế dữ liệu mới vào khu vực way đã được sử dụng lần xa nhất (Least Recently Used)

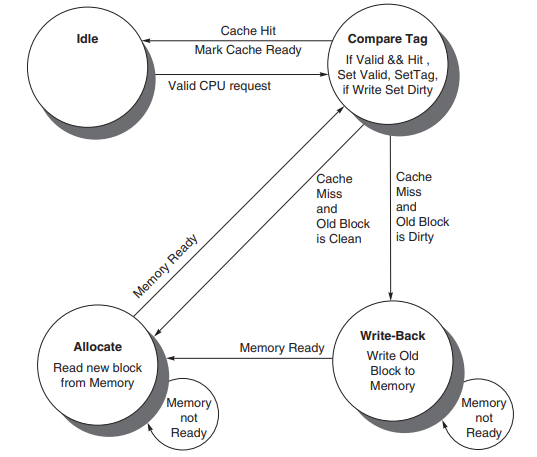
Sơ đồ bộ cache khi có thêm bộ xử lý LRU như sau.



Bộ LRU có chức năng như một flipflop tận dụng hoạt động của bộ cache từ 2-4 xung clock để cập nhật way index mới cho lần sử dụng sau.

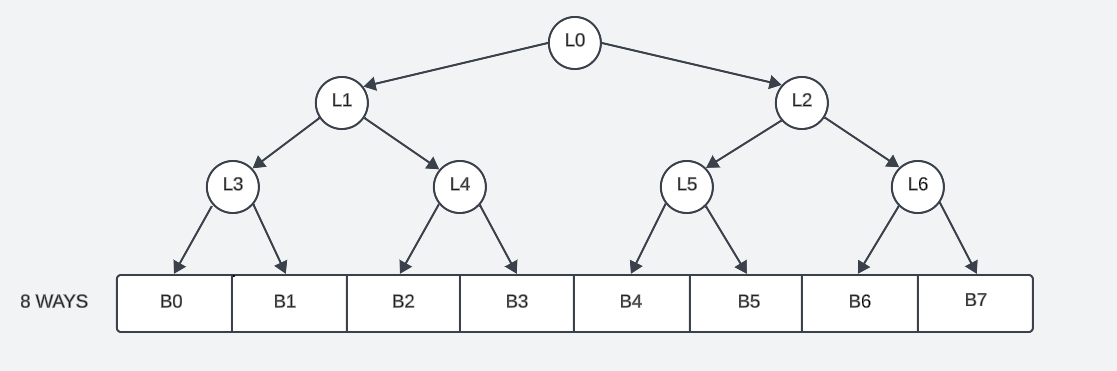
1. **Thiết kế bộ PLRU**

Giản đồ trạng thái của bộ cache controller.



Sơ độ mạch so sánh của bộ set associative cache.

Pseudo LRU 8 ways



Hình 1.1. Psuedo LRU tree

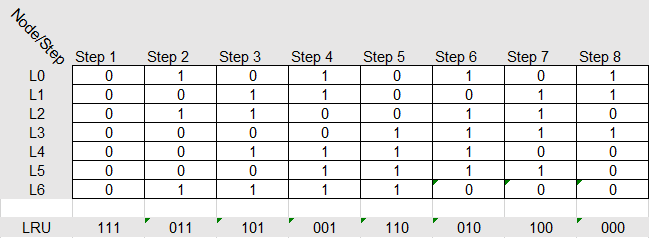
Trong thiết kế 8-way set associative cache, LRU tree có cấu trúc như hình trên. LRU tree có thể được mô tả như 3 tầng, tầng đầu tiên có L0, tầng thứ hai là L1 và L2, tầng thứ ba chứa L3, L4, L5, L6. Gọi mỗi way là một block (trong block chứa tag và data), ví dụ khi truy cập block B5, giá trị nhị phân là 101, lúc này LRU tree sẽ được cập nhật lại giá trị mỗi lần truy cập, như vậy L0 sẽ bằng 1, L2 bằng 0 và L5 sẽ bằng 1. Như quy tắc trên, mỗi lần truy cập cache, LRU tree sẽ cập nhật lại giá trị và lưu trữ trạng thái ở từng node (L0, L1, L2,…).

A diagram of a machine

Description automatically generated

Hình 1.2. Psudeo LRU circuit

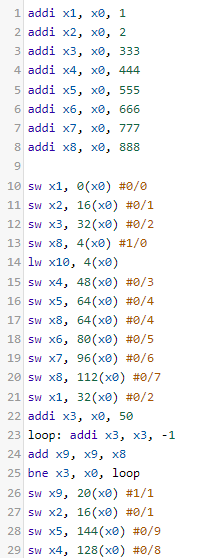
Từ những giá trị đã được lưu trong các node L0, L1, … giá trị LRU sẽ được tính như hình trên, Hình dưới đây sẽ giải thích trực quan hơn các trường hợp cụ thể cách hoạt động của bộ LRU.



Hình 1.3. Giá trị LRU tree ở mỗi lần truy cập

Ở hình trên, giá trị của mỗi node được lưu lại qua từng step (hoặc gọi là lần truy cập), ở step 1 giá trị của các node đều bằng 0, do đó giá trị LRU sẽ có giá trị là 111 như sơ đồ 1.2. Tiếp đến step 2, giá trị LRU ở step 1 sẽ được sử dụng để cập nhật lại LRU tree, ở bước này, giá trị LRU trước đó là 111, do đó L0 được cập nhật bằng 1, vì L0 bằng 1 nên L2 sẽ được cập nhật bằng LRU[1] và bằng 1, và vì L2 bằng 1 nên L6 sẽ được cập nhật bằng LRU[0] và bằng 1. Tương tự với các step sau đó.

Đoạn code assembly như hình dưới sẽ được dùng để kiểm tra hoạt động của bộ 8-way set associative cache.

Ban đầu, các giá trị thanh ghi từ x1 đến x8 được khởi tạo để thuận tiện cho quá trình kiểm tra kết quả.

Ở dòng 10, giá trị thanh ghi x1 sẽ được lưu vào set thứ nhất (mỗi set chứa 8 ways). Tiếp đến x2, x3 cũng được lưu vào set thứ nhất, vì mỗi set chứa 8 block nên lúc này 3 block chứa giá trị của x1, x2, x3 và còn 5 block chưa có giá trị.

Ở dòng 13, giá trị thanh ghi x8 được lưu và set thứ 2. Dòng 14, giá trị ở set thứ 2 sẽ được load vào x10.

Tương ứng các dòng sau đó, x4, x5, x8 (vì x8 được lưu vào địa chỉ 64 đã lưu trước đó nên block có địa chỉ 64 sẽ thay thế x5 bằng x8), x6, x7, x8 lưu vào set thứ nhất. Lúc này set thứ nhất đã lưu đủ 8 giá trị (set thứ nhất đầy).

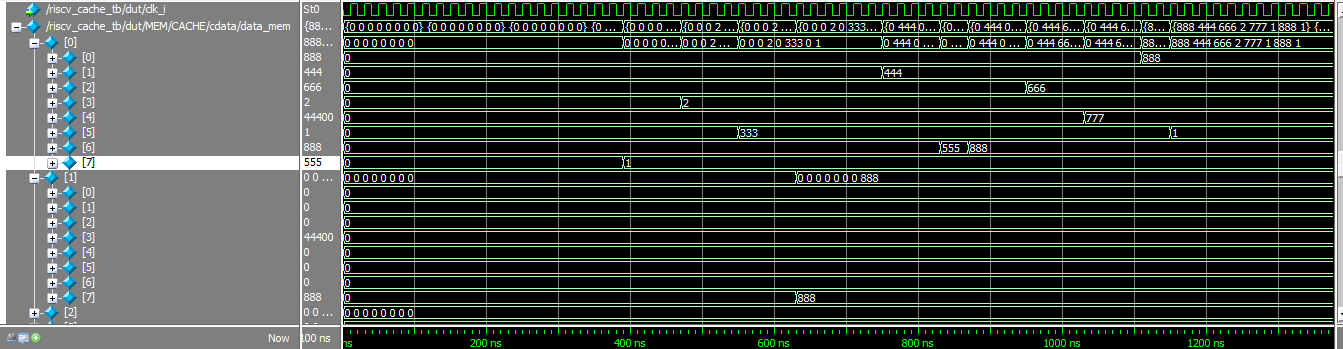
Do đó, lệnh ở dòng 21, ô có địa chỉ 32 chứa giá trị 333 (thanh ghi x3) sẽ được thay thế bằng 1 (thanh ghi x1).

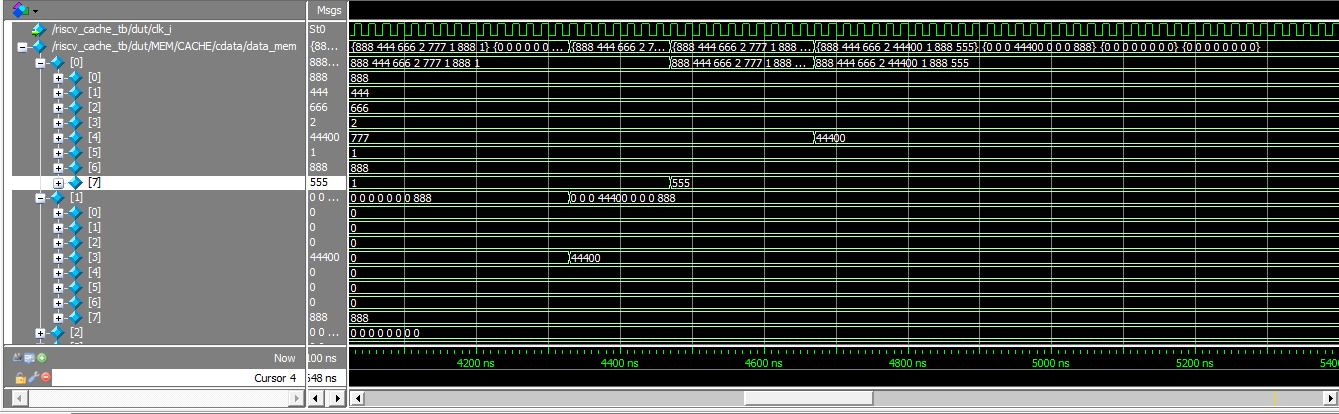
Từ dòng 23 đến dòng 25, giá trị x9 sẽ được tính toán và có giá trị sau cùng là 888 nhân 50 và bằng 44400. Sau đó x9 được lưu vào set thứ 2.

Dòng 27, ô có địa chỉ 16 sẽ cập nhật giá trị của x2.

Ở dòng 28, vì set thứ nhất đã đầy và không có block nào chứa địa chỉ khớp với địa chỉ 144, do đó 1 block nào đó (được quyết định bởi khối LRU) sẽ được lưu lại giá trị vào memory và cập nhật giá trị mới.. Tương tự với dòng lệnh 29.

Dưới đây là waveform mô tả đoạn code assembly phía trên.





Hình 1.4. Waveform cho đoạn test assembly

Như dữ liệu đã tính toán từ hình 1.3, giá trị ban đầu sẽ được lưu vào block 7 tiếp đến lần lượt là block 3, 5, 1, 6, 2, 4, 0. Và mỗi lần truy cập sau đó, LRU tree sẽ được cập nhật và đưa ra địa chỉ của block tiếp theo cần được ghi ở từng set.

1. **Lời nhận xét và định hướng trong tương lai.**

Đúng với dự đoán về tính tối ưu của thiết kế, bộ cache 4-way associativity có thời gian chạy chương trình tốt hơn bộ cache direct map cùng data size. Thuật toán thay thế LRU phát huy được tối đa thiết kế của bộ 4-way associativity. Tuy nhiên, ở dự án lần này, chúng em vẫn còn có thể tối ưu hơn nữa bộ Cache khi kết hợp thêm yếu tố lân cận trong không gian của bộ nhớ ( spatially and locality ) vào bộ Cache. Khi đó miss rate được dự đoán sẽ giảm đáng kể hơn nhờ vào cơ chế dự đoán được chương trình có xu hướng truy cập các vùng dự liệu gần với địa chỉ đang truy xuất.

Trong tương lai, chúng em dự định sẽ tiếp tục phát triển tốc độ bộ Cache bằng cách giảm miss rate của nó bằng một bộ đệm trung gian có khả năng phân loại câu lệnh và xử lý câu lệnh song song với hoạt động của cache.

1. **Nguồn tham khảo.**