



پروژه درس معماری کامپیوتر

عنوان پروژه:

Simulating virtual memory, TLB, Cache and main memory

استاد درس:

آقای دکتر بیت‌الهی

تهیه کنندگان:

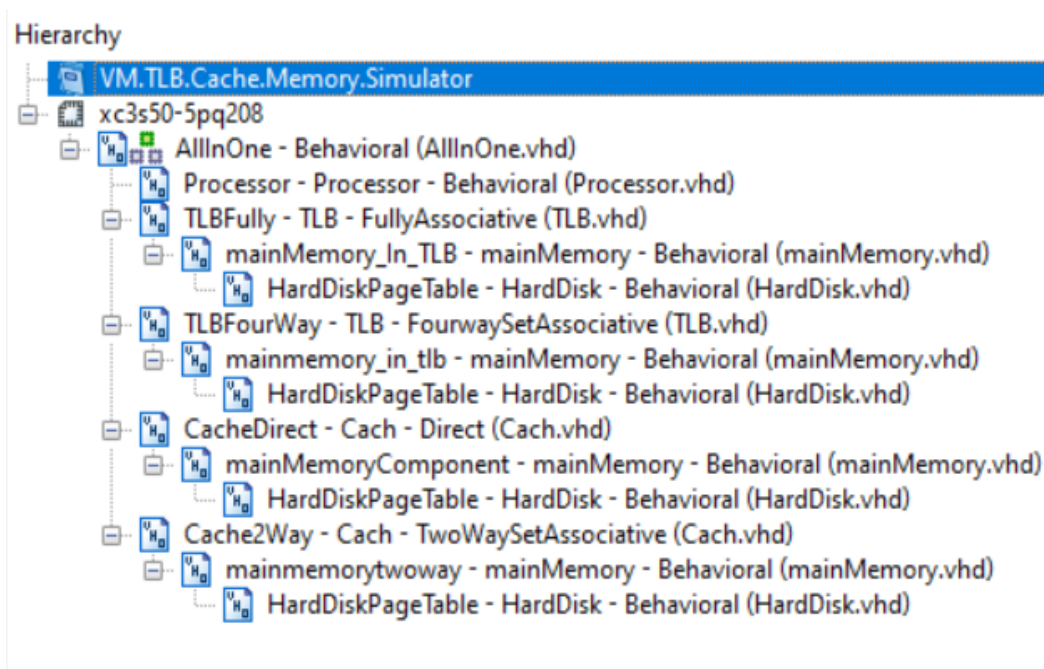
آیسا میاهی‌نیا

آیلین نائب‌زاده

تابستان 1401

شرح پروژه:

این پروژه با استفاده از زبان vhdل پیاده سازی شده است. عملکرد و هدف اصلی به اینصورت است که ۶ بخش از اجزاء یک سیستم کامپیوتری از جمله main memory, CPU, virtual memory, و TLB را شبیه سازی می کند. در جهت بهبود کد و خوانایی راحت تر، کد مربوط به هر بخش در یک فایل جدا قرار گرفته است. و در صورت نیاز به هر component پیاده سازی شده، با استفاده از دستورات موجود، آن بخش را در فایل بالاتر صدا می زنیم. حال به توضیحات مربوط به هر فایل می پردازیم.

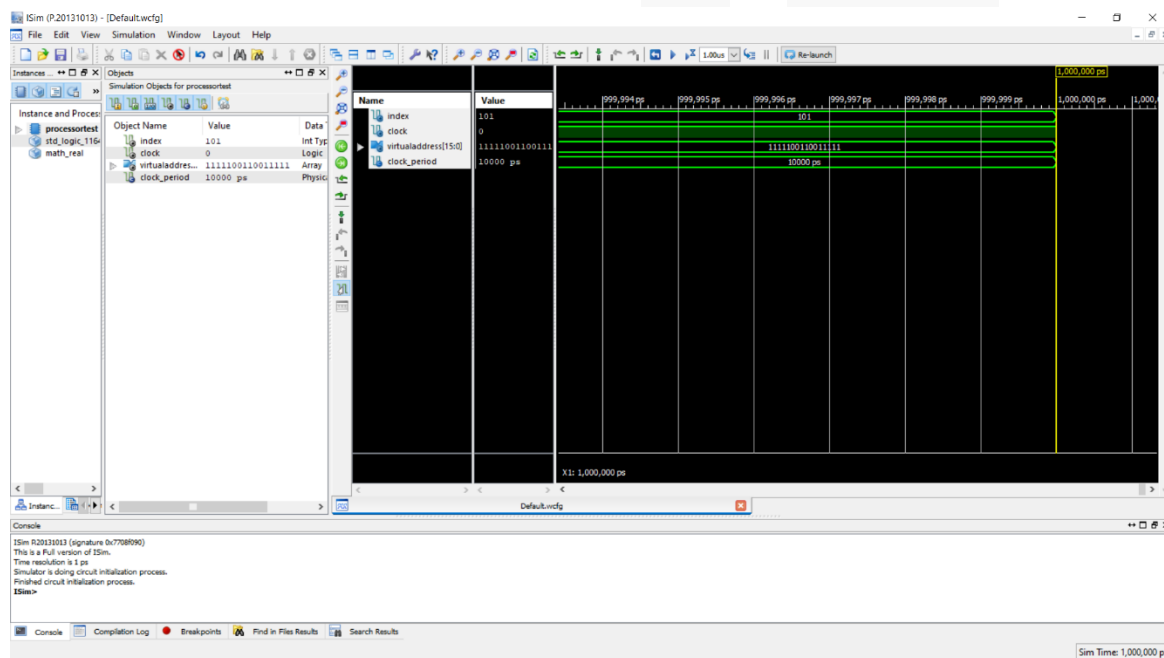


Processor -1

در این بخش یک آرایه ۱۰۰ عضوی از virtual address ها داریم، بطوریکه هر عضو آن ۱۶ بیت (۹ بیت برای VPN و ۷ بیت برای PageOffset) می‌باشد. همچنین این آرایه بصورت رندوم پر شده‌است و هیچ قاعده و نظم خاصی برای آن وجود ندارد.

باتوجه به اینکه وظیفه این بخش برگرداندن یک virtual address از آرایه موجود است، بعنوان ورودی یک عدد صحیح بعنوان index و یک clock از نوع std_logic تعریف می‌کنیم و خروجی نیز تنها یک virtual address از نوع std_logic_vector هست که ۱۶ بیت ظرفیت دارد.

در بخش منطق و architecture عضو index ام آرایه virtual address ها را درون خروجی می‌ریزیم.



2- TLB

در این بخش ابتدا نیاز به یک سیگنال کنترلی داریم تا مشخص کنیم که TLB از چه نوعی است. ورودی این بخش یک Virtual address از نوع std_logic_vector 16 بیتی و یک clock از نوع std_logic است. خروجی این بخش نیز شامل یک متغیر به اسم miss_hit از نوع std_logic و یک PPN(Physical Page Number) است که در واقع ترجمه شده virtual address ورودی می‌باشد. خروجی TLB در ادامه بعنوان ورودی به cache داده می‌شود.

1- TLB Fully Associative

در architecture مربوط به این بخش ابتدا چند signal تعریف کردیم. ابتدا نیاز داریم که virtual address ورودی را به اجزاء آن که PageOffset و VPN هستند تجزیه کنیم. پس برای هر یک از این بخش‌ها std_logic_vector هایی به ترتیب با اندازه های ۷ و ۹ بیتی تعریف می‌کنیم. همچنین یک سیگنال بعنوان controllerMissOrHit تعریف می‌کنیم که در ابتدا آن را با '0' مقدار دهی می‌کنیم.

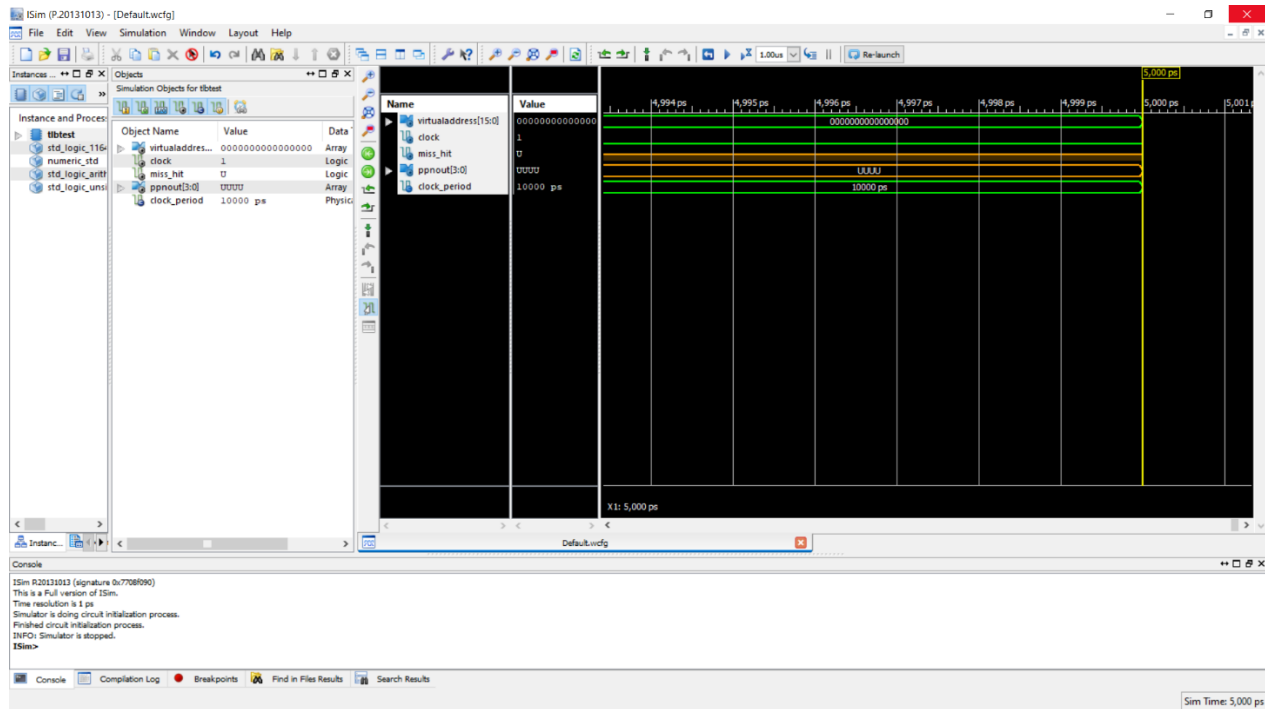
سپس یک آرایه 48 عضوی (TLBmem) طبق خواسته صورت سوال از داده های ۱۴ بیتی (۴ بیت برای PPN + ۹ بیت برای VPN + ۱ بیت برای Valid) تعریف می‌کنیم. (اعضاء این آرایه نیز کاملاً بصورت رندوم پر شده اند).

حال در منطق پروژه از یک حلقه for استفاده می‌کنیم که بر روی تک تک اعضاء آرایه TLBmem پیمایش می‌کند و چک می‌کنیم اگر بیت های ۴م تا ۱۲م عضو نام این آرایه با VPN آدرس ورودی برابر بودند، پس Hit رخ می‌دهد و هر دو سیگنال controllerMissOrHit و miss_hit را با '1' مقدار دهی می‌کنیم و بیت ۱۰م تا ۱۳م همین عضو را در سیگنال PPNOut می‌ریزیم.

ولی در خارج از این حلقه لازم است که شرایطی را که Miss رخ بدهد را نیز چک کنیم. یعنی اگر هیچ یک از اعضاء TLBmem شرط خواسته شده در بدنه حلقه را نداشته باشند. در این شرایط miss_hit را با '0' مقدار دهی می‌کنیم و باید به Page Table را بررسی کنیم، پس از بررسی یکی از اعضاء TLBmem را بطور رندوم پر می‌کنیم (با استفاده از عملگر & که برای concatenation استفاده می‌شد).

باتوجه به اینکه Page Table درون memory قرار گرفته است، لازم است با استفاده از port map آن را صدا بزنیم و خروجی این بخش را در یک سیگنال موقت مانند PPN_out_tmp بریزیم. و این مقدار را به PPNOut که خروجی TLB هست بدهیم.

دقت شود که به هنگام port map به این دلیل که تنها به بخش Page Table از memory احتیاج داریم پس کافی است که متغیر controller موجود در memory را با '1' مقدار دهی کرده، cache_type را 'u' گذاشته و virtual Address موجود در TLB را به memory پاس بدهیم و از خروجی های memory نیز، تنها PPN برای ما اهمیت دارد.



2-2 TLB Four Way Set Associative

در این حالت ۱۲ مجموعه (set) داریم و در هر گروه چهار عضو وجود دارد. در این بخش منطق برنامه بسیار شبیه به حالت قبلی می باشد ولی لازم است که چند سیگنال جدید در جهت ذخیره کردن Tag و Index و همچنین باقی مانده index بر ۱۲ به برنامه اضافه کنیم. همانند حالت قبلی PageOffset ۷ بیت را به خود اختصاص می دهد ولی VPN به دو بخش تقسیم می شود به اینصورت که ۴ بیت ابتدایی برای index (integer) و ۵ بیت باقی مانده برای ذخیره کردن tag جدا می شوند. در این بخش نیز یک آرایه با ۴۸ عضو از std_logic_vector های ۱۴ بیتی داریم، بطوریکه هر عضو آن از ۴ بیت PPN، ۴ بیت index، ۵ بیت tag، ۱ بیت valid تشکیل شده است. برای ذخیره کردن index یک سیگنال از نوع integer تعریف می کنیم و پس از تبدیل بیت های ۱۰ تا ۳ VPN به عدد صحیح آن را در index می ریزیم. همچنین سیگنال indexMod12 را نیز با ۴ برابر index در پیمان ۱۲ پر می کنیم.

در این بخش نیز با استفاده از یک حلقه که تنها ۴ بار تکرار می‌شود دو شرط زیر را چک می‌کنیم:

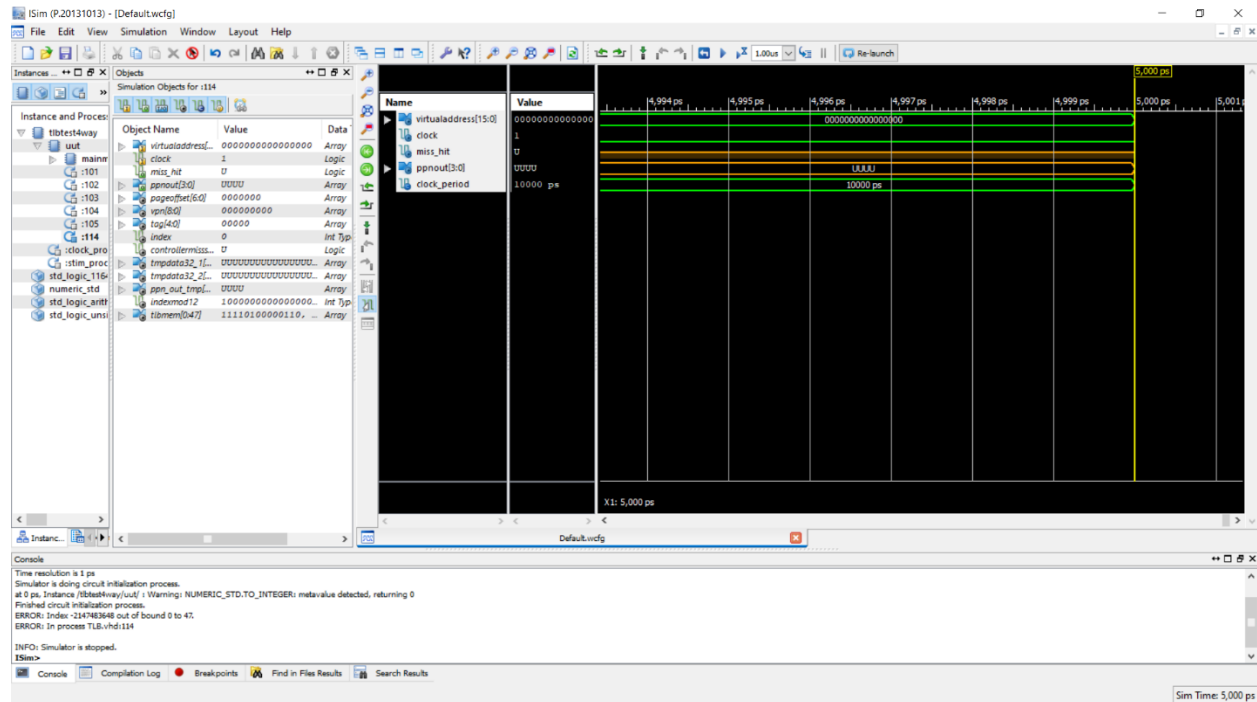
```
TLBmem(indexMod12 + i)(12 downto 4) = VPN AND TLBmem(indexMod12 + i)(13) = '1'
```

اگر برقرار بودند یعنی Hit رخ داده، هر دو سیگنال `controllerMissOrHit` و `miss_hit` را با '1' مقدار دهی می‌کنیم و `TLBmem(indexMod12 + i)(3 downto 0)` را در `PPNOut` می‌ریزیم. در خارج از این حلقه چک می‌کنیم اگر `controllerMissOrHit` همچنان '0' بود، پس باید به Page Table مراجعه کنیم، پس دقیقاً مانند آنچه که در Fully TLB داشتیم، memory را صدا می‌زنیم و با تعیین

ورودی

`controller = '1'` از Page Table آن استفاده می‌کنیم.

خروجی memory را در `ppn_out_tmp` می‌ریزیم و `TLBmem(indexMod12+ 0)` را با استفاده از آن مقدار دهی می‌کنیم.



3- Main Memory

در این بخش Page Table و Data Memory را پیاده سازی کردیم. پس بعنوان ورودی یک سیگنال کنترلی تعریف کردیم که اگر '1' باشد، یعنی به Page Table احتیاج داریم و اگر '0' باشد به Data Memory. لازم است یک سیگنال کنترلی دیگر نیز داشته باشیم تا نوع cache را که می‌خواهیم از آن استفاده کنیم نگه داریم. همچنین یک ورودی بعنوان clock و دو ورودی دیگر بعنوان PhysicalAddressInput و VirtualAddress به ترتیب به اندازه ۱۱ و ۱۶ بیت تعریف کردیم. درحالی که بعنوان Page Table از این بخش استفاده شود، یک خروجی PPN (Physical Page Number) خواهیم داشت.

در غیر اینصورت دو خروجی ۳۲ بیتی نیاز داریم. (outputData, outputData2)

حال چند سیگنال جدید تعریف می‌کنیم.

ابتدا برای هر بخش به یک آرایه نیاز داریم:

یک آرایه بعنوان dataMemory شامل ۳۸۵ عضو ۳۲ بیتی تعریف می‌کنیم و سپس یک آرایه به اسم PageTable شامل ۵۱۲ عضو ۵ بیتی خواهیم داشت. (در ابتدا تمام عضوهای این دو آرایه صفر هستند). دو سیگنال دیگر نیز به نام‌های VPN و PageOffset تعریف می‌کنیم تا Virtual Address را تجزیه و در آن‌ها نگه داریم.

3-1 controller = '0'

در صورتی که controller = '0' باشد و به data memory احتیاج داشته باشیم، لازم است چک کنیم که cache_type، ۰ است یا ۱. اگر ۱ باشد یعنی از نوع 2 way است. پس از تبدیل PhysicalAddressInput به Integer آن را به آرایه data memory بعنوان index پاس می‌دهیم و خروجی را در outputData می‌ریزیم. و چون به outputData2 احتیاجی نداریم، آن را با 'u' مقدار دهی می‌کنیم.

ولی در غیر اینصورت لازم است outputData2 را نیز مقداردهی کنیم. و عبارت زیر را بعنوان index به data memory پاس می‌دهیم:

```
physicalAddressInput(10 downto 3) + not physicalAddressInput(2) + physicalAddressInput(1 downto 0)
```

3-2 controller = '1'

در صورتی که controller = '1' باشد و به Page Table احتیاج داشته باشیم. ابتدا شرط زیر را چک می‌کنیم:

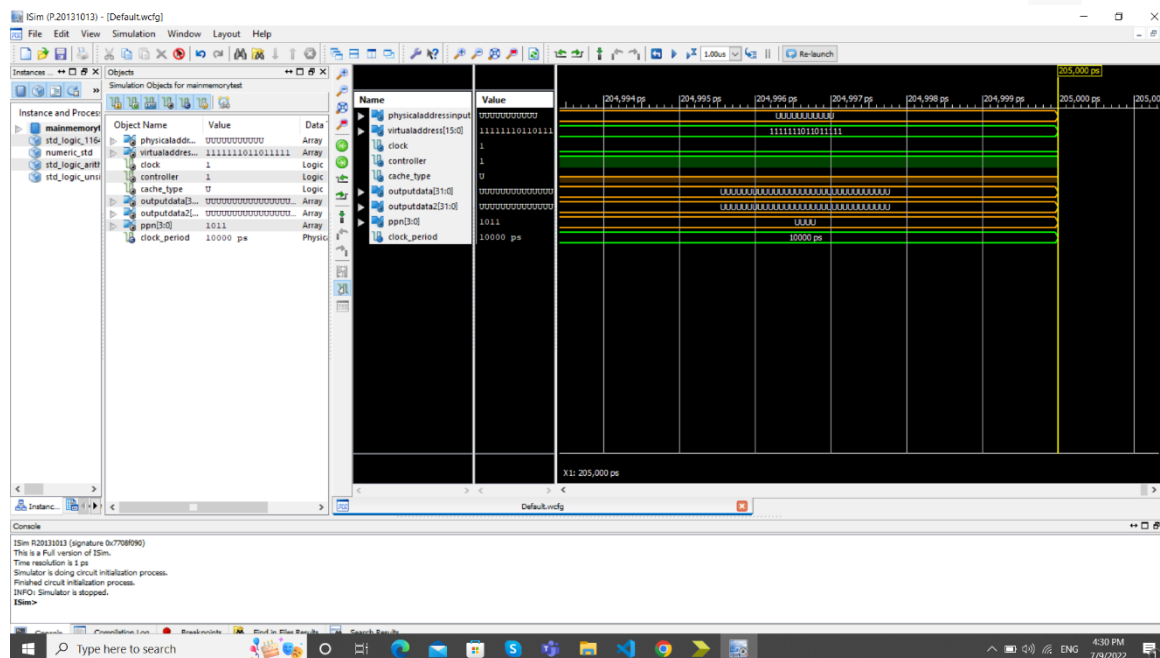
```
pageTable(to_integer(unsigned(VPN))) = "UUUUU" or pageTable(to_integer(unsigned(VPN)))(4) = '0'
```

در صورتی که برقرار باشد یعنی آدرس مورد نظر در Page Table هم وجود ندارد، پس لازم هست که حالا به Hard Disk مراجعه کنیم و با استفاده از Port map آن را صدا بزنیم.

حال 3 بیت آخر tmpPhysicalAddressOut خروجی Hard Disk را به PPN می‌دهیم و عنصر متناظر با VPN درون PageTable و آدرس متناظر با tmpPhysicalAddressOut درون Data Memory را بصورت زیر مقدار دهی می‌کنیم:

```
PPN<= tmpPhysicalAddressOut(10 downto 7);
pageTable(to_integer(unsigned(VPN)))<= '1' & tmpPhysicalAddressOut(10 downto 7);
dataMemory(to_integer(unsigned(tmpPhysicalAddressOut))) <= tmpoutputData(31 downto 0);
```

ولی در صورتی که آدرس مورد نظر در Page Table وجود داشته باشد، مقدار PPN همان ۴ بیت کم ارزش تر عنصر متناظر با VPN موجود در PageTable می‌شود.



Cache - 4

در این بخش دو ورودی داریم، یک clock و یک PhysicalAddress که درواقع Physical Address حاصل خروجی TLB می‌باشد. همچنین ۳ خروجی داریم. ابتدا یک سیگنال miss_hit تعریف می‌کنیم و سپس دو خروجی دیگر از نوع std_logic_vector های ۳۲ بیتی (output32) و ۶۴ بیتی (output 64) تعریف می‌کنیم، چرا که دو نوع cache داریم. و وابسته به اینکه از چه نوعی استفاده می‌کنیم، یکی باید کاملاً Undefined بشود.

Direct Cache -1-4

در این حالت خروجی ۶۴ بیتی و شامل 2 word است. همچنین byteoffset دو بیت، word offset یک بیت، tag سه بیت و index پنج بیت را به خود اختصاص می‌دهد.

ابتدا چند سیگنال تعریف می‌کنیم تا بتوانیم هر آدرس را به `index` , `tag` , `byte offset` و `word offset` تجزیه کنیم. یک سیگنال به اسم `TagIndex` نیز تعریف می‌کنیم تا بتوانیم بعداً حاصل `tag & index` را در آن ذخیره کنیم. در این بخش یک آرایه ۳۲ عضوی (`buffermem`) از `std_logic_vector` های ۷۳ بیتی خواهیم داشت. به اینصورت که ۶۴ بیت ابتدایی برای ذخیره کردن `data`، ۵ بیت برای `index`، ۳ بیت برای `tag`، ۱ بیت برای `valid` خواهیم داشت.

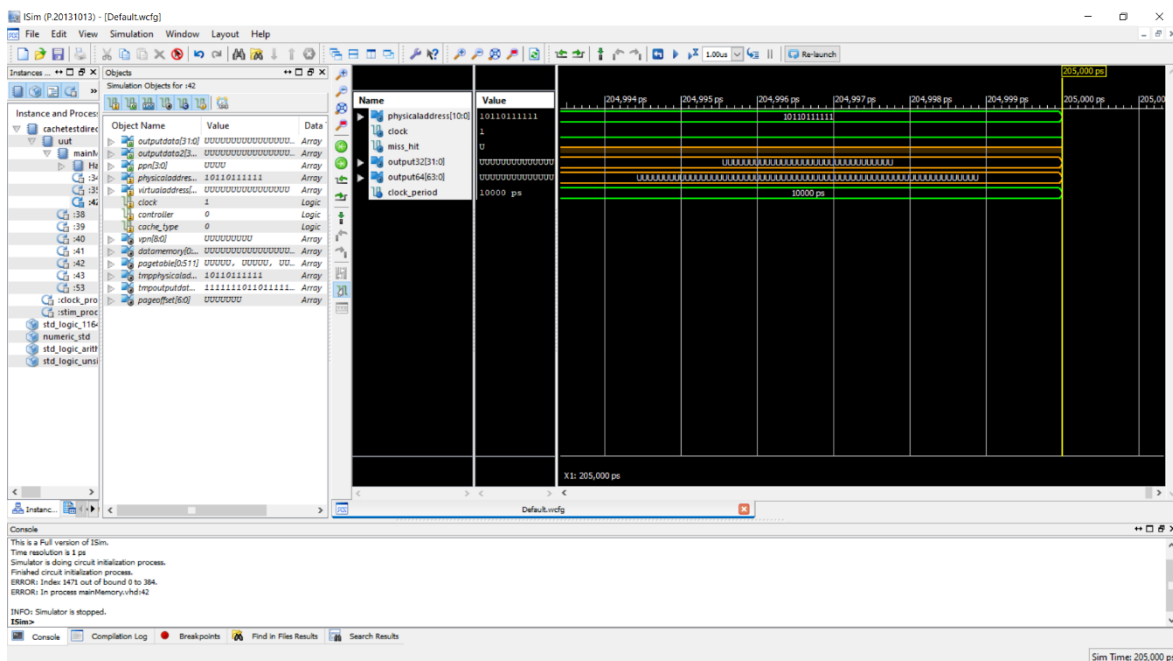
(*تمام اعضاء این آرایه در ابتدا صفر هستند.)

حال در بخش منطق نیز یک حلقه به طول ۳۲ خواهیم داشت که روی تک تک اعضاء آرایه دو شرط زیر را بررسی می‌کند.

```
buffermem(i)(71 downto 64) = TagIndex and buffermem(i)(72) = '1'
```

اگر این دو شرط رخ بدهند، یعنی `Hit` اتفاق افتاده، پس ۶۴ بیت ابتدایی این عضو از مجموعه را در خروجی می‌ریزیم. ولی خارج از این حلقه چک می‌کنیم، اگر `controllerMissOrHit` همچنان '۰' بود (Miss رخ داده است) لازم است از بخش `data memory` قرار گرفته شده در `memory` مورد نظر را پیدا کنیم، پس با استفاده از `port map` آن را صدا می‌زنیم. (توجه شود که ورودی `controller` موجود در `memory` باید با '۰' مقداردهی شود و `virtual address` نیز تعریف نشده باشد، چرا که احتیاجی به آن نداریم.)

حال خروجی‌های حاصل از `memory` که دو `std_logic_vector` ۳۲ بیتی هستند را در دو متغیر موقت ذخیره می‌کنیم. و اولین متغیر را در ۳۲ بیت اول یک عضو رندوم `buffermem` می‌ریزیم و ۳۲ عضو دیگر را با دومین متغیر پر می‌کنیم. همچنین لازم است که `valid bit` این عضو را با '۱' مقدار دهی کرده و در نهایت ۶۴ بیت ابتدای آن را در خروجی می‌ریزیم.



2-4- 2-Way Set Associative Cache

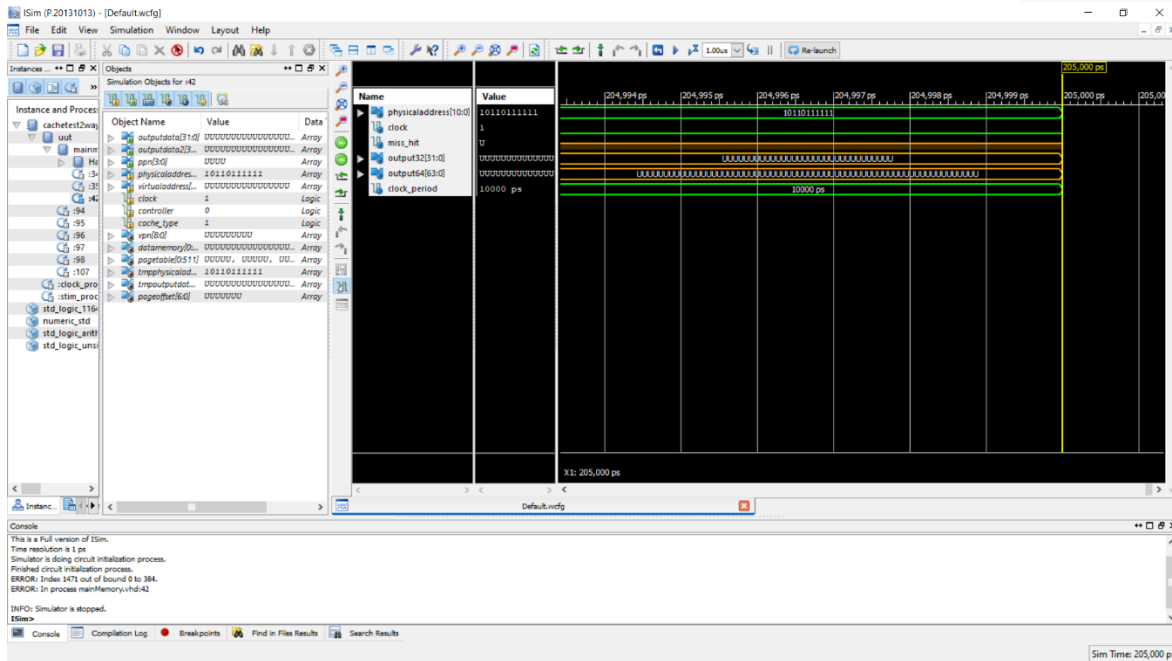
در این حالت خروجی ۳۲ بیتی و تنها شامل یک word است، همچنین index , tag و byteOffset هر کدام به ترتیب ۵، ۴ و ۲ بیت را به خود اختصاص می‌دهند.

آرایه‌ای که در این بخش تعریف می‌کنیم شامل ۱۶ عضو، ۴۲ بیتی (۳۲ بیت دیتا، ۴ بیت index، ۵ بیت tag و ۱ بیت valid) هست.

سایر منطق این حالت بسیار شبیه به Direct Cache است. با این تفاوت که در اینجا حلقه لازم است ۱۶ بار تکرار شود و برای چک کردن حالت Hit دو شرط زیر بررسی شوند:

```
buffermem(i)(40 downto 32) = TagIndex AND buffermem(i)(41) = '1'
```

و در حالتی که miss رخ بدهد، با مراجعه به data memory و خواندن ۳۲ بیت دیتا از آن، یک عضو رندوم از buffermem را پر می‌کنیم.



5 – Hard Disk

در این بخش یک ورودی به اسم VirtualAddress داریم، همچنین یک متغیر به اسم cache_type که نوع cache‌ی که از آن استفاده کنیم را نگه می‌دارد و یک clock نیز داریم.

بعنوان خروجی نیز یک PhysicalAddressOut داریم که درواقع متناظر VirtualAddress ورودی است و همچنین یک outputData.

حال لازم است چند آرایه جدید به برنامه اضافه کنیم. ابتدا یک آرایه به اسم PAddress می‌سازیم، بطوریکه متشکل از ۱۰۰۰ آدرس ۱۱ بیتی است. و همچنین یک آرایه به اسم VAddress که شامل ۱۰۰۰ آدرس ۱۶ بیتی است. و دو آرایه ۱۰۰۰ عضوی متشکل از داده‌های ۳۲ بیتی نیز تعریف می‌کنیم. (DataOutput1 , DataOutput2)

(*تمامی عناصر این آرایه های را بطور رندوم پر کردیم.)

در بخش منطق برنامه ابتدا یک متغیر به اسم `index_Input` از نوع `integer` تعریف می‌کنیم و در طی یک حلقه بطول ۱۰۰۰ و پس از بررسی و مقایسه تمامی عناصر موجود در `VAddress`، اگر شرط زیر برقرار بود، آنگاه `index_Input` را با `i` مقدار دهی می‌کنیم.

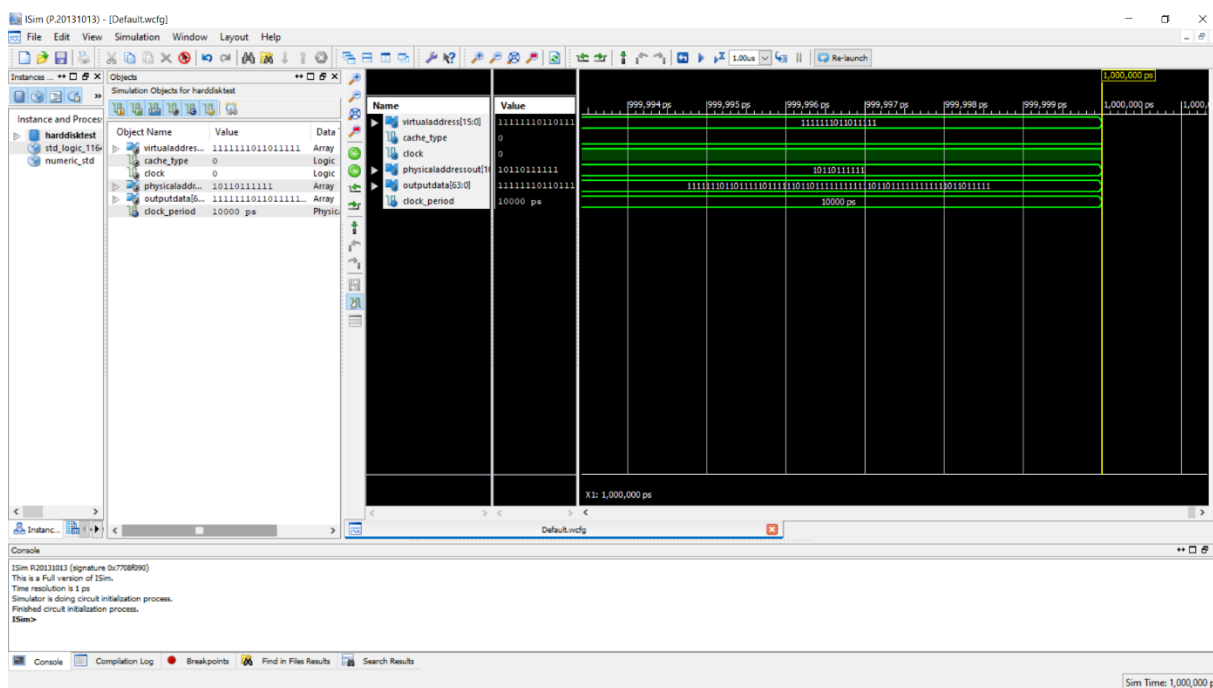
```
VAddress(i) = VirtualAddress
```

حال در خارج از این حلقه باتوجه به اینکه `cache_type` چه مقداری داشته باشد، متغیر `outputData` را مقدار دهی می‌کنیم.

اگر `Direct Cache` باشد، لازم است ۳۲ بیت اول آن را با استفاده از `DataOutput1` پر کنیم و ۳۲ بیت بعدی آن را با استفاده از `DataOutput2` پر کنیم.

در غیر اینصورت تنها ۳۲ بیت اول آن را با استفاده از `DataOutput1` پر می‌کنیم و ۳۲ بیت بعدی آن را با 'u' مقدار دهی می‌کنیم.

```
if(cache_type = '0') then
    outputData(31 downto 0) <= DataOutput1(index_Input);
    outputData(63 downto 32) <= DataOutput2(index_Input);
else
    outputData(31 downto 0) <= DataOutput1(index_Input);
    outputData(63 downto 32) <= "UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU";
end if;
```



این فایل درواقع عملکردی مشابه تابع Main در زبان های نرم افزاری دارد. به اینصورت که مانند نقطه شروع برنامه ما عمل می کند و بخش های مختلف به استثناء Hard disk در این بخش با استفاده از کلید واژه port map صدا زده شده اند.

ابتدا بعنوان ورودی یک عدد (index) از نوع عدد صحیح و یک clock تعریف کردیم. همچنین دو سیگنال کنترلی به اسم cache_type و TLB_type. و بعنوان خروجی نیز دو سیگنال OutputData و OutputData2 را از نوع std_logic_vector به ترتیب با اندازه های ۳۲ و ۶۴ بیت تعریف کردیم. حال باتوجه به عکس زیر لازم است که component های مورد نیاز را به ترتیب صدا بزنیم و باتوجه به ورودی ها و خروجی ها، در بدنه architecture سیگنال های جدید تعریف کنیم.

