CO224 Computer Architecture

Lab 6 - Building a Memory Hierarchy
Part 2 – Data Cache

Group No: 09
E/20/148 Hewawasam A.K.L.
E/20/157 Janakantha S.M.B.G.

Cache Controller

A finite state machine was designed according to the following cases and the cache controller which handles cache misses was implemented.

	WRITE	READ	Dirty Bit	Valid Bit	Tag Match	
Read	0	1	0	0	0	mem_read
	0	1	0	0	1	mem_read
	0	1	0	1	0	mem_read
	0	1	0	1	1	cache_read
	0	1	1	0	0	Forbidden
	0	1	1	0	1	Forbidden
	0	1	1	1	0	mem_write→ mem_read → cache_write
	0	1	1	1	1	cache_read
Write	1	0	0	0	0	cache_write
	1	0	0	0	1	cache_write
	1	0	0	1	0	cache_write
	1	0	0	1	1	cache_write
	1	0	1	0	0	Forbidden
	1	0	1	0	1	Forbidden
	1	0	1	1	0	mem_write→ cache_write
	1	0	1	1	1	cache_write

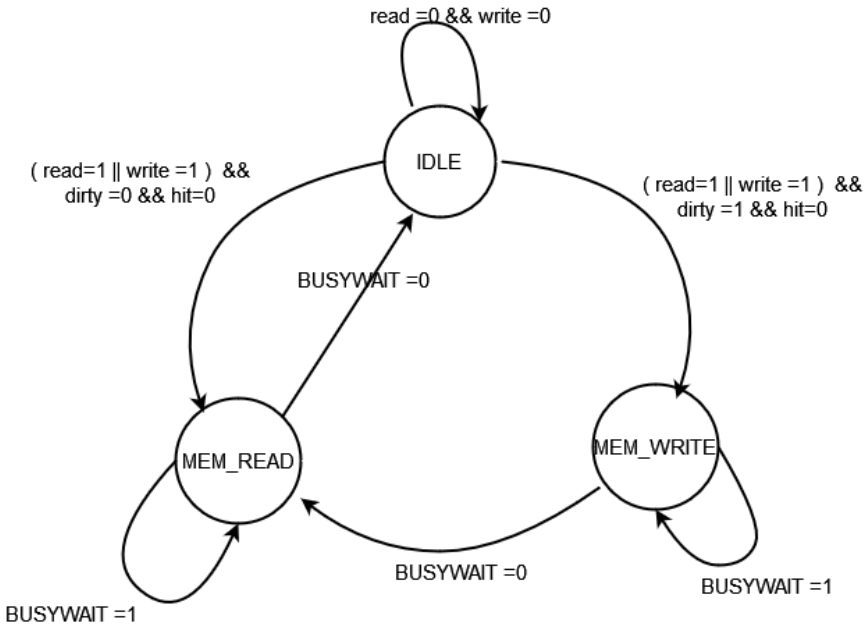

Figure 1 **:** Finite State Machine

Comparison between the cache-less and the cached

In the Lab 06 part 1, the memory hierarchy for the CPU was initialized by implementing a main memory module. In this part, the setup was modified by adding a data cache between the CPU and the data memory aiming to enhance the performance of the system.

Cache is basically a small and fast memory at the top level, which tricks the CPU to see the memory as large and fast. Cache is built according to two principles of locality.

1. Temporal Locality – Recently accessed data will likely to be accessed again soon.
2. Spatial Locality – Closely located data to the recently accessed ones will likely to be accessed soon

The cached setup and the cache-less setup were tested by executing the following sample program. It took 412 time units with cache-less system , whereas it took 620 units with the cached system. In this scenario, we have used 9 instructions which deal with memory, and two of them are misses. Since a relatively larger miss rate is encountered in this example, the expected performance boost of the cache was not observed.

```
ASM sample_program.s
 1    loadi 0 0x09
 2    loadi 1 0x01
 3    swd 0 1
 4    swi 1 0x00
 5    lwd 2 1
 6    lwd 3 1
 7    sub 4 0 1
 8    swi 4 0x02
 9    lwi 5 0x02
10    swi 4 0x20
11    lwi 6 0x20
```

Figure 2 **:** Sample Program

Cashe-less Setup

A latency of 40-time units is noticed in every instruction since the memory is accessed each time. The CPU is stalled for 5 CPU clock cycles whenever the read or write operation occurs and this reduces its performance.

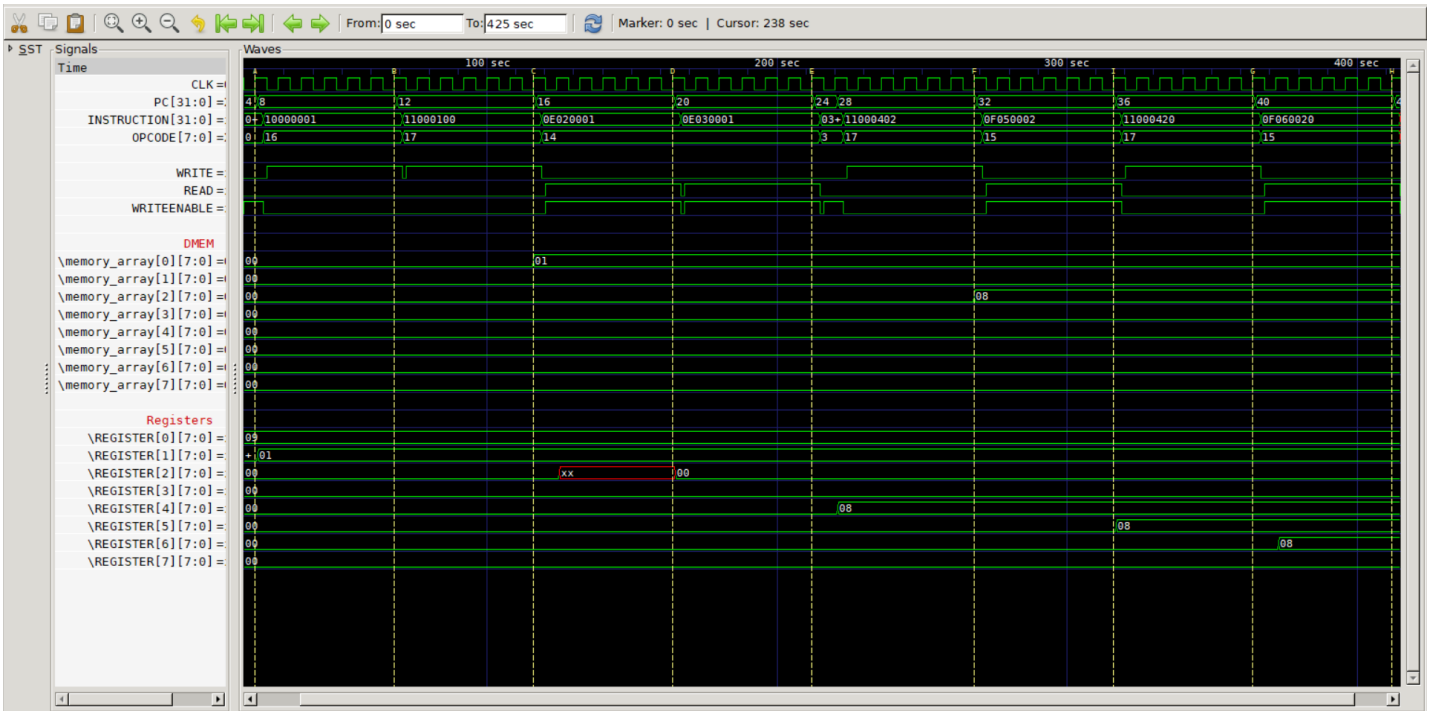

Figure 3 **:** GTKWave obtained by executing the sample program in the cache-less setup

Cashed Setup

1. Case 1 :

```
loadi 0 0x09
loadi 1 0x01
swd 0 1
swi 1 0x00
lwd 2 1
lwd 3 1
sub 4 0 1
swi 4 0x02
lwi 5 0x02
swi 4 0x20
lwi 6 0x20
```

`swd 0 1` : Write the value from register 0 to the memory address given in register 1. Since register 0 and 1 are loaded with values `0x09` and `0x01`, it instructs to write the value `0x09` in memory at the address `0x01`.

Since the cache is not yet loaded, the *valid bit* is 0 , *dirty bit* is 0 for the index `000` and there may be garbage values for tag and data block. Therefore, a write-miss has occurred. With the *write-back* policy, the corresponding block of data (4 Bytes) is fetched from the memory which takes 5 CPU clock cycles per a Byte (which is a total of additional 20 clock cycles). Then the value `0x09` is stored in the 1st word of the cache block.

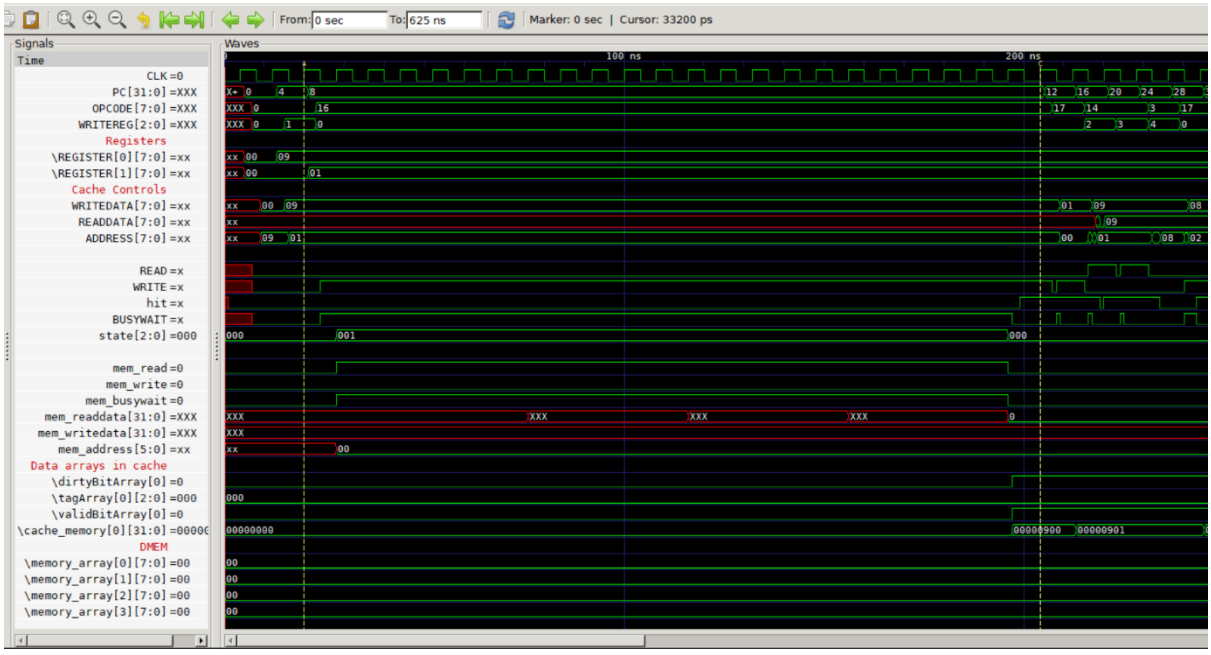

Figure 4 : GTKWave obtained by executing the sample program in the cached setup : For instruction `swd 0 1`

Case 2 :

```
loadi 0 0x09
loadi 1 0x01
swd 0 1
swi 1 0x00
lwd 2 1
lwd 3 1
sub 4 0 1
swi 4 0x02
lwi 5 0x02
swi 4 0x20
lwi 6 0x20
```

`swi 1 0x00` : Write the value from register 1 (value `0x01`) to the memory at address `0x00`
`lwd 2 1` : Read memory at the address given in register 1 (address `0x01`) and store the result (value `0x01`) in register 2.
`lwd 3 1` : Read memory at the address given in register 1 (address `0x01`) and store the result (value `0x01`) in register 3.
`sub 4 0 1` : Subtract the value in register 1 (`0x01`) from the value in register 0 (`0x09`) and store the result (value `0x08`) in register 4.
`swi 4 0x02` : Write the value from register 4 (value `0x08`) to the memory at address `0x02`
`lwi 5 0x02` : Read memory at address `0x02` and store the result (value `0x08`) in register 5.

For the highlighted instructions, the required data block is already in the cache entry for the index `000` . Therefore, all those cache accesses are hits, and the corresponding data block is handled within the same clock cycle itself.

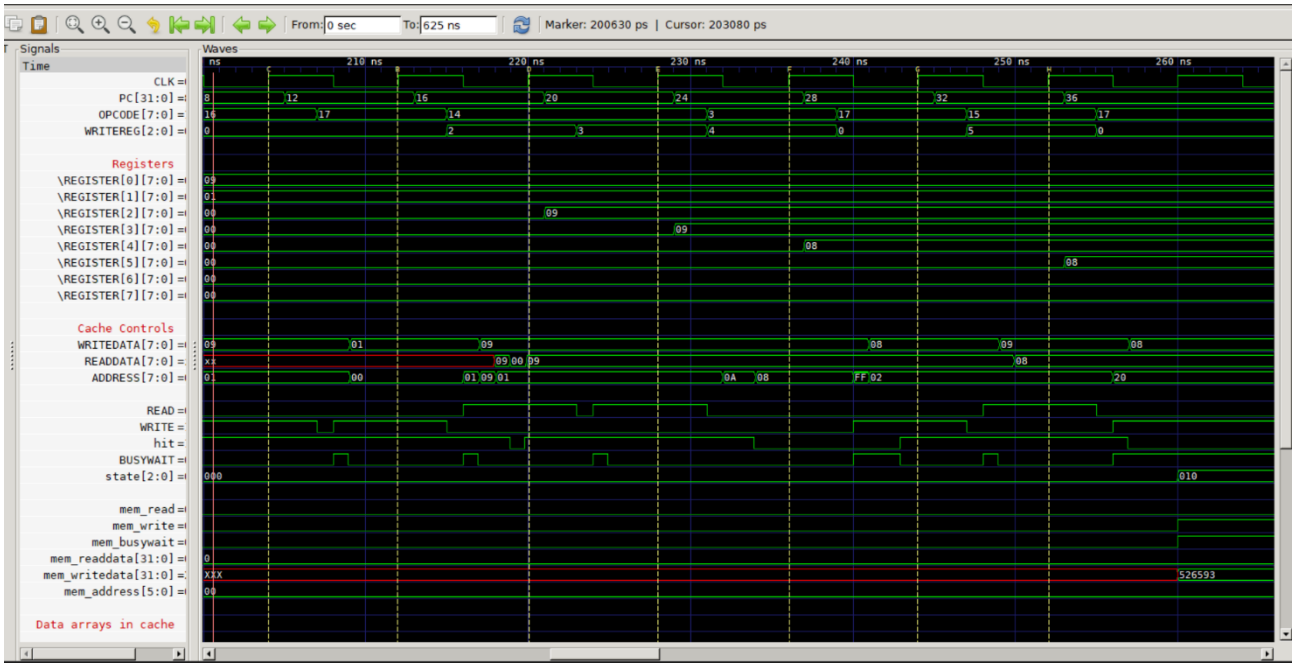

Figure 4 : GTKWave obtained by executing the sample program in the cached setup :
For instructions `swi 1 0x00 , lwd 2 1, lwd 3 1, sub 4 0 1, swi 4 0x02, lwi 5 0x02`

Case 3 :

```
loadi 0 0x09
loadi 1 0x01
swd 0 1
swi 1 0x00
lwd 2 1
lwd 3 1
sub 4 0 1
swi 4 0x02
lwi 5 0x02
swi 4 0x20
lwi 6 0x20
```

`swi 4 0x20` :Write the value from register 4 (value `0x08`) to the memory at address `0x20`

Though the *valid bit* is 1 and *dirty bit* is 1 of the index `000`, the tags mismatch, indicating a write-miss. Since that block is *dirty* (inconsistent), it is written back to the memory and then the new block is fetched. Those two actions take 20 CPU clock cycles each, which leads to a miss penalty of 40 cycles. Then, for the index `000` , a new entry with the tag `001` is updated with the value stored in register 4.

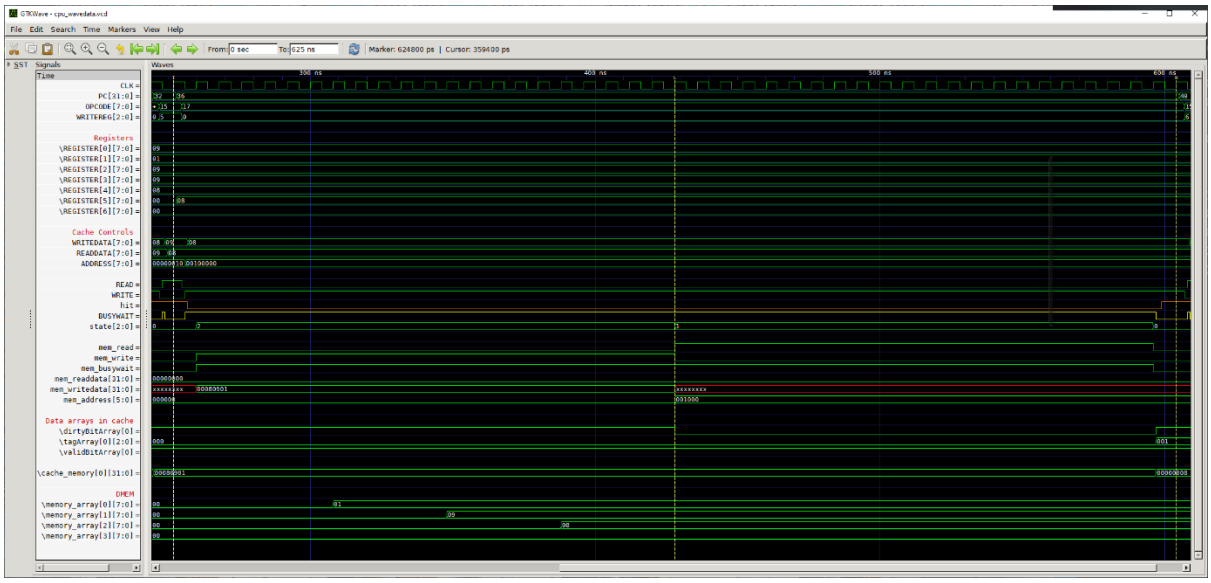

Figure 5 : GTKWave obtained by executing the sample program in the cached setup : For instruction `swi 4 0x20`

Case 4 :

```
loadi 0 0x09
loadi 1 0x01
swd 0 1
swi 1 0x00
lwd 2 1
lwd 3 1
sub 4 0 1
swi 4 0x02
lwi 5 0x02
swi 4 0x20
lwi 6 0x20
```

`lwi 6 0x20` :Read memory at address `0x20` and store the result (value `0x08`) in register 6.

In the cache entry with index `000`, the *valid bit* is 1 and *dirty bit* is 1, and it is the correct memory block since the tags are equal. As it is a read-hit, the value is stored in register 6 within the same cycle.

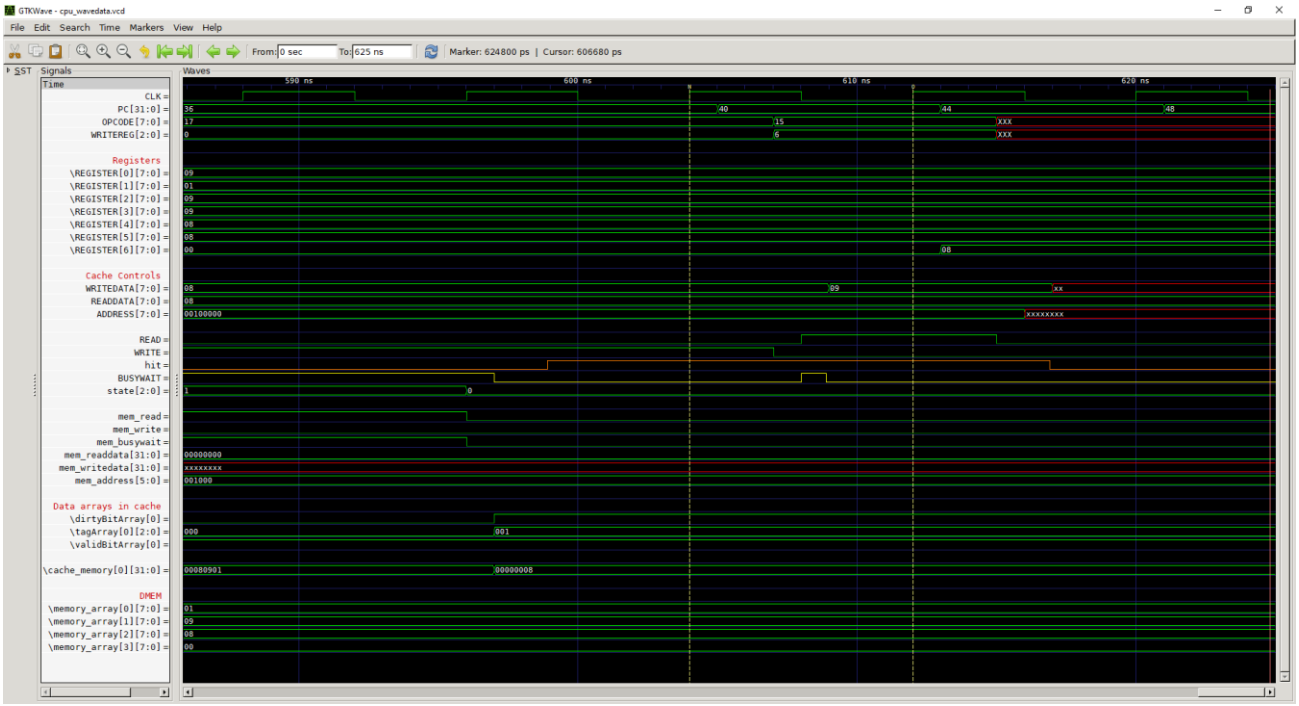

Figure 6 : GTKWave obtained by executing the sample program in the cached setup :
For instruction `lwi 6 0x20`

Conclusion

The cache-less system stalls the CPU for 5 clock cycles for each instruction which reduces its efficiency. In contrast, the setup with a cache accesses the memory and stalls the CPU, only if a miss is encountered. If the access is a hit, data in cache is served within the same clock cycle and the CPU is not stalled. Therefore, in case of a hit, the time taken for execution is reduced and the performance is significantly increased. However, in case of a miss, latencies of much larger values can be noticed since a significant amount of time is taken to read/write the data memory. If the data entry already in the cache is not dirty, 20 clock cycles are taken , while, if the data entry is dirty, 40 clock cycles are taken.

Therefore, if the miss rate is way higher, the cached setup will be even less efficient than the cache-less setup. Only if the hit rate is significantly high, the cache will perform effectively. Higher hit rates can be achieved by increasing the cache size and adjusting the block size accordingly.

The Verilog files including all the modules, testbench and screenshots of timing diagrams are available on :
https://github.com/Bimsara-Janakantha/8-Bit-Single-Cycle-Processor/tree/66bcb5ef23b6e70f0016132bd017cf2e7c2abce6/Group09_Lab06_Part2