

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
	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_read
	1	0	1	1	1	cache_write

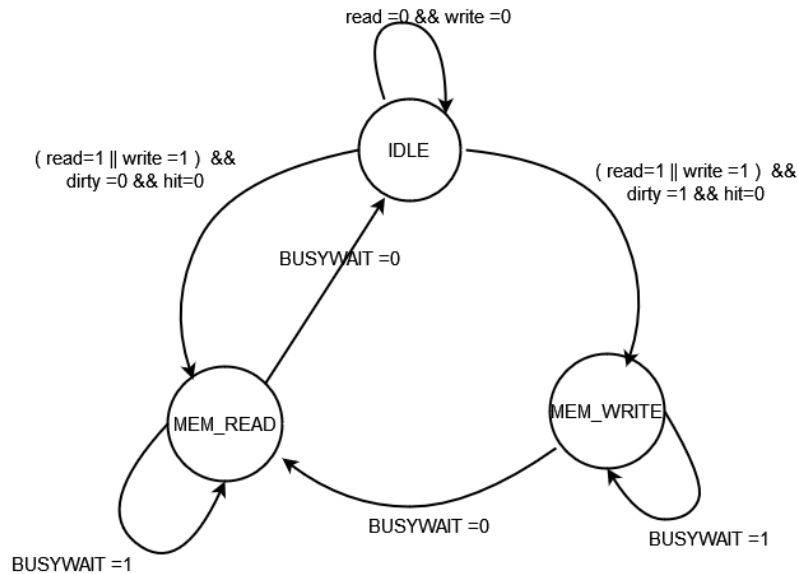


Figure 1 : Finite State Machine

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.

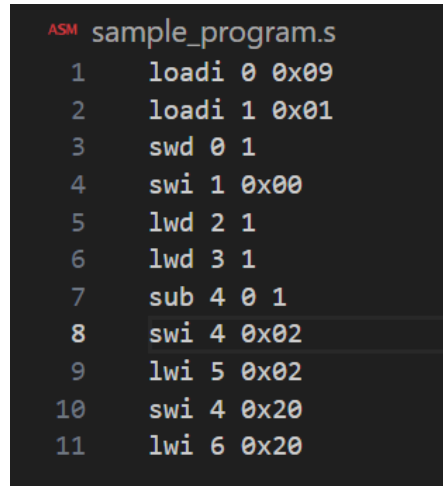
Cache is basically a small and fast memory at the top level, which tricks the CPU to see the memory as large and fast. The time taken to read data from a cache is quite fast than from the memory. Cache is built according to two principles of locality.

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.

A latency of 40 time units is noticed in each read and write instructions in the cache-less data memory. Therefore, the CPU is stalled for 5 CPU clock cycles whenever the read or write operation occurs.

In contrast, the setup with a cache memory stalls the CPU only if a miss is encountered. If the access is a hit, memory is not accessed, 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 a case of a miss, latencies of much larger values can be noticed since 20 CPU cycles are still required to access a block of data in the 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. So, the cached memory system is highly efficient, only if the hit rate is large.

The cached setup and the cache-less setup was 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 performs read or write operations, and two of them are misses. Since a relatively larger miss rate is encountered, the expected performance boost of the cache was not observed in this example.

A screenshot of a code editor showing assembly code for a file named 'sample_program.s'. The code consists of 11 instructions, each preceded by a line number from 1 to 11. The instructions are: 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, and 11 lwi 6 0x20. The text is white on a dark background.

```
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

In conclusion, if the hit rate is significantly high, the cache will perform effectively. In case, if the miss rate is way higher, the cached setup will be even less efficient than the cache-less setup. 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