E/16/388

Weerasundara W.M.T.M.P.B.

CO-224

Lab 6 part 2 Report

Performance comparison with cache less design

- In lab 6 part 1 memory was implemented without a cache memory. Therefore any memory access instruction will take the same time to interact with the memory (5 clock cycles).
- In this part cache was added. For a small number of memory access instructions, modules with cache takes more time than the cache less design because it deals with memory blocks with 4 words. Therefore if small number of memory access instructions with accessing different parts of the memory, this module will take lots of time to load whole blocks but cache less module can just load/store the word in 5 cycles.
- The real advantages in the cache comes when we have lots of memory access instructions in a program which tries to access memory locations that are close to each other or the same location over and over again.

As an simple example,

If we modify the same memory location multiple times in a program,
Cache less design will take 5 cycles per instruction but the design with a cache will take lots of cycles for updating the cache at the 1st time we
Access that location but for the next instructions it handles them within
A single cycle. Also we can access locations nearby that location within
A single cycle too.

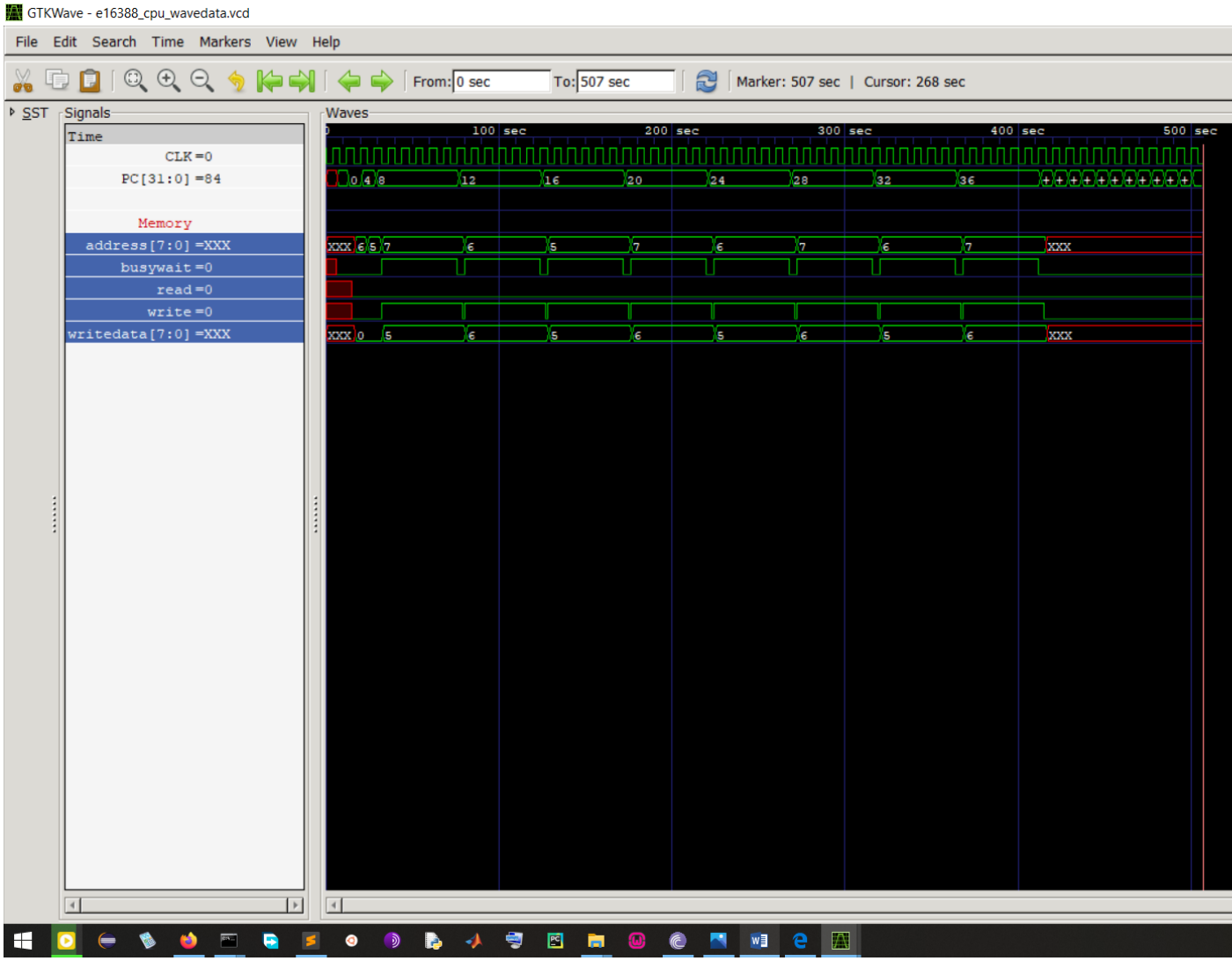
Consider below set of instructions,

This is a simple store type instructions which access nearby memory locations and keep update them. For the convenience only store
Is considered.

```
loadi 1 0x06
loadi 5 0x05
swi 5 0x07
swi 1 0x06
swi 5 0x05
swi 1 0x07
swi 5 0x06
swi 1 0x07
swi 5 0x06
swi 1 0x07
```

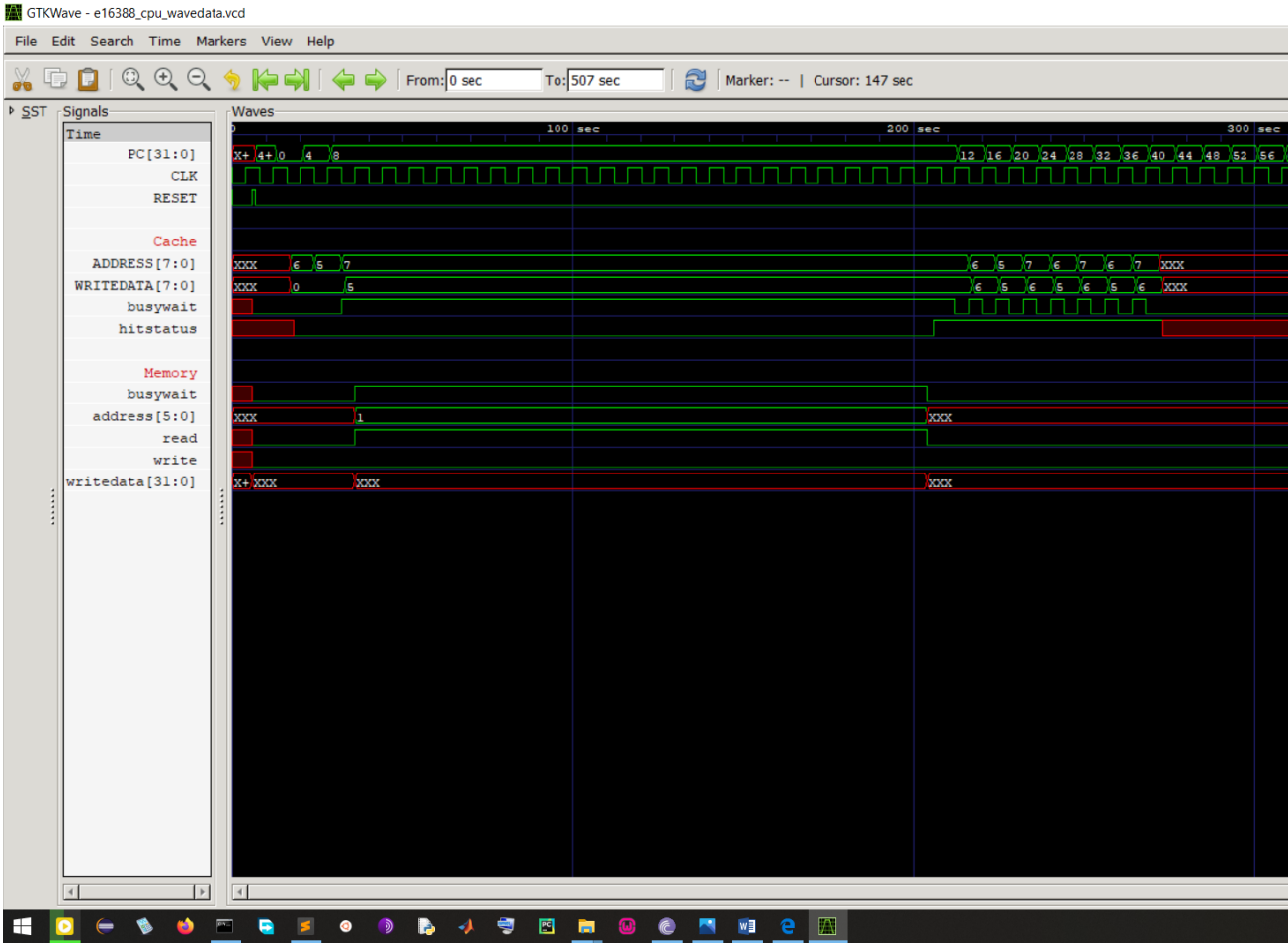
Let’s consider their signal diagrams to consider efficiency of these instruction set
When they run in design without cache and with cache.

- Design without cache



1. To run these instructions about 400 time units have spent, for each instruction cpu is being stalled.
2. Each memory access takes 5 clock cycles to complete.
3. To access previously accessed locations, also takes 5 cycles.
4. All the memory instructions are treated equally.

- Design with cache



1. To run these instructions less than 300 time units were spent.
2. Only for the 1st memory access instruction, several clock cycles were used because whole word block is imported to cache from memory. (words 4,5,6,7 in this program)
3. Following instructions are completed within the clock cycle.
4. Cpu is not stalled after the 1st memory access.
5. Therefore this design takes slightly less time to deal with memory access instructions when they access nearby memory locations.
6. Cache stores nearby location data, their addresses therefore cpu deals with much faster cache rather than the much slower memory.
7. This happens as long as we access the same/nearby memory locations. Cache uses combinational logic to decide whether accessed memory location is at the cache or not.
8. Cache divides incoming address to Tag, Index, Offset to get required word location and checks them with currently stored fields in the cache. Also cache checks the validity of the data stored in the cache as well and the consistency with the data in the cache and memory..
9. Therefore without losing any data, cache runs above tests and efficiently deals between cpu and memory.

Design issues and ideas

- Designing using cache memory is much more complex than designing without using cache memory.
- Because of issues in simulation, expected timings are not coming.
- For the memory access instruction when a miss happens, extra clock cycle is spent than the expected number of cycles.
- By using a write buffer we can increase the efficiency of the write back operation. We can use a first in first out queue for store limited number of write backs and avoid stalling cpu for some time.

Summary

- Cache is used for increase efficiency for the instructions that works with memory.
  Cache separates addresses into 3 fields. According to this design in lab 6,

Tag (7-5)	Index (4 – 2 bits)	Offset (0 – 1 bits)

- Cache stores 4 blocks of data with Tag, Validity bit to detect validity and,
  Dirty bit to detect it's consistency with it's memory location and 4 word data according to the Index of the memory address locations.

Valid bit(1 bit)	Dirty bit(1 bit)	Tag bits (3 bits)	Data (32 bits)

- Inside the cache input address's fields are checked with stored fields in the cache according to the index, If requesting memory is in the cache operation finishes within the clock cycle.
  If not if the data in cache is overwritten by the required data from memory if the data in cache is consistent with the data in memory, otherwise those data will be written to the memory before overwrite them with the data block from the required memory address.

This design has the ability to handle memory write or read instructions within 1 clock cycle rather than stalling the cpu for each time memory write or read instruction runs. Therefore this design with the cache has better performance than the design without a cache.