CO224 : Lab 06 Building a Memory Hierarchy

Group 02: Dilshan D.M.T. E/20/069
Karunarathne K.N.P. E/20189

In part 2 of lab 6, we enhanced the CPU's data memory hierarchy by integrating a data cache module between the CPU and the data memory to reduce data access delays. To evaluate the performance improvement, we executed sample programs on both the original cache-less setup and the modified setup, comparing the differences in access delays.

The diagram below illustrates the finite state machine for the cache controller:

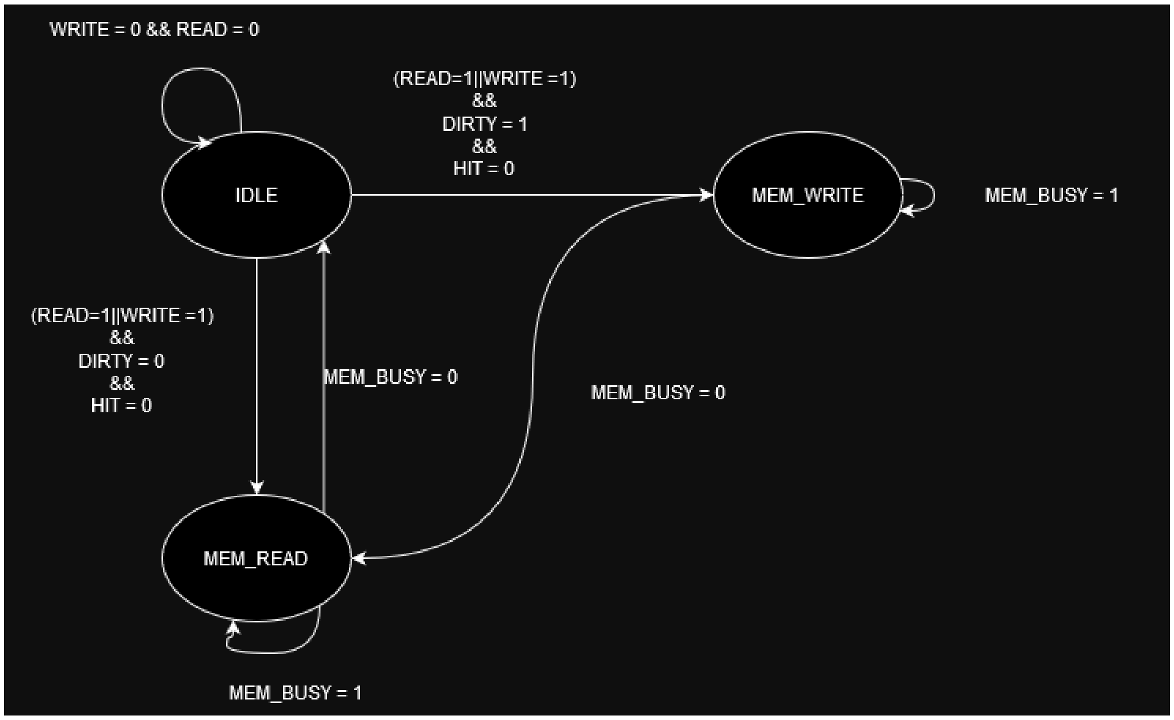


FIGURE : FSM

We have used three states for the FSM:

IDLE: The default state when no data requests are being processed.
MEM_READ: Activated when data needs to be fetched from the main memory.
MEM_WRITE: Activated when data needs to be written to the main memory

When performing read or write operations for the first time, a system with cache is generally more efficient. This is because the system with cache must load entire data blocks from memory to execute the operation, which is more time-consuming than reading a single word. For the initial read, a system with cache takes 21 clock cycles while a system without cache takes only 6 clock cycles. The timing diagrams below illustrate the behavior of the program counter (PC) for both systems during an initial read/write operation.

Therefore, Cache less system is more efficient when reading or writing for the first time. You can see it in the timing diagrams given below:

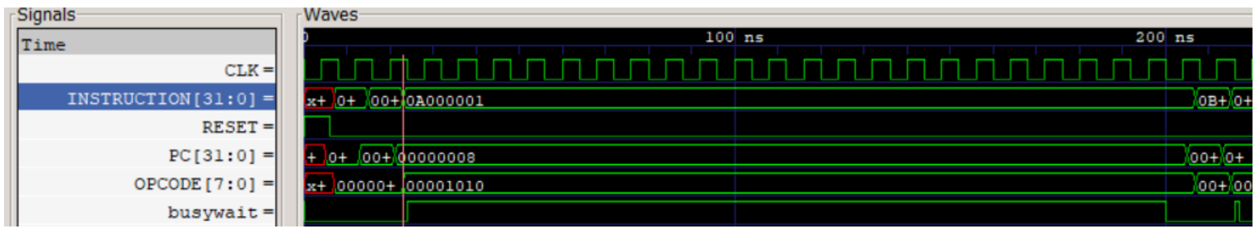


Figure: System with the cache

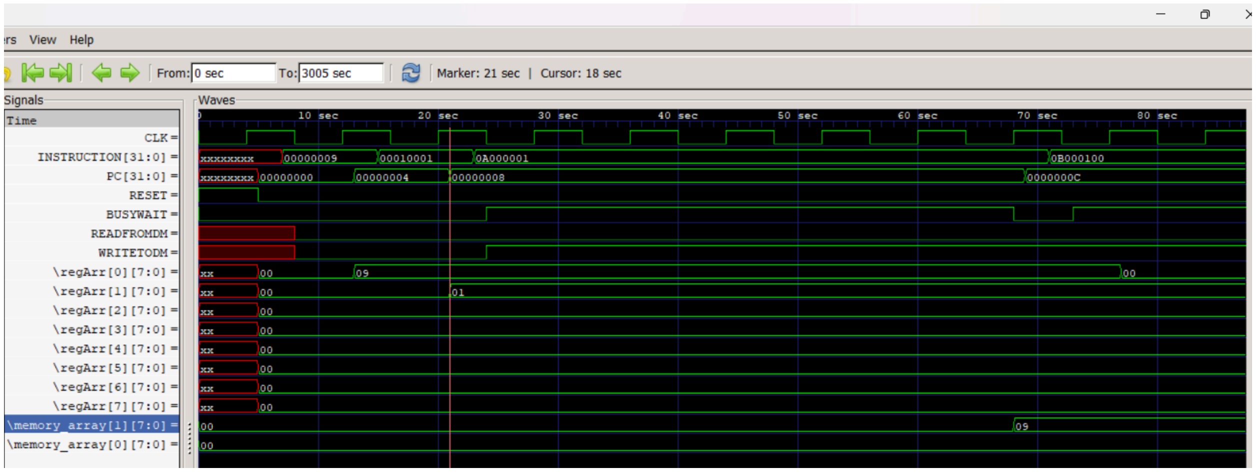


Figure :System without the cache

Once the cache is loaded, if a read or write operation hits the cache, the output can be provided within a single clock cycle. This efficiency is due to the streamlined process that occurs when accessing data from the cache. Specifically, the following steps are involved:

1. **Extract Tag, Valid Bits, and Dirty Bits**: The system quickly retrieves the necessary metadata (tags and status bits) from the cache to determine if the data is present and valid.
2. **Tag Comparison**: The system compares the tag of the requested data with the tags stored in the cache to confirm a cache hit.
3. **Write Data to Cache**: If the operation is a write, the data is promptly written to the cache.

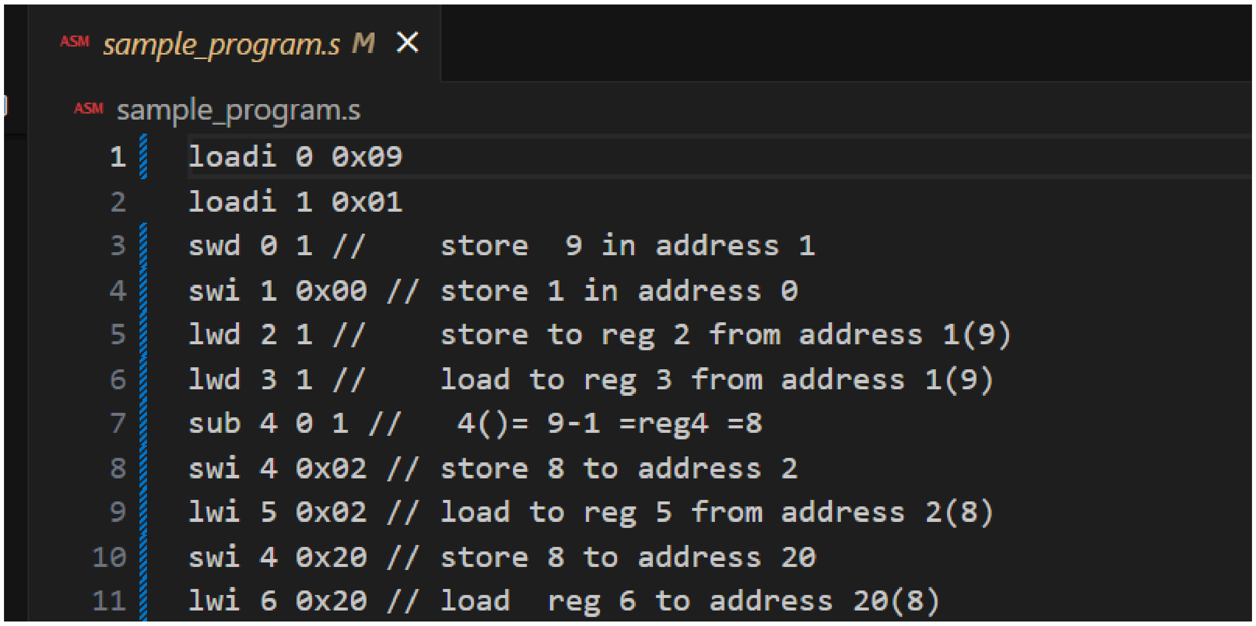
Because these operations are designed to be extremely fast, they can be completed within a single clock cycle if the data is found in the cache (a cache hit).

In contrast, a system without cache must always fetch data directly from the main memory. This process is essentially slower because it involves multiple steps such as locating the data in memory, retrieving it, and then processing it.

Therefore, if a program achieves a high cache hit rate—meaning that most of the data accesses are successfully served by the cache—the overall efficiency of the system with cache is significantly higher.

The timing diagram below illustrates the behavior of the system with cache during cache hits, showing how it can quickly provide outputs within one clock cycle.

Instruction set:


sample_program.s M

sample_program.s

```
loadi 0 0x09
loadi 1 0x01
swd 0 1 //     store  9 in address 1
swi 1 0x00 // store 1 in address 0
lwd 2 1 //     store to reg 2 from address 1(9)
lwd 3 1 //     load to reg 3 from address 1(9)
sub 4 0 1 //    4()= 9-1 =reg4 =8
swi 4 0x02 // store 8 to address 2
lwi 5 0x02 // load to reg 5 from address 2(8)
swi 4 0x20 // store 8 to address 20
lwi 6 0x20 // load  reg 6 to address 20(8)
```

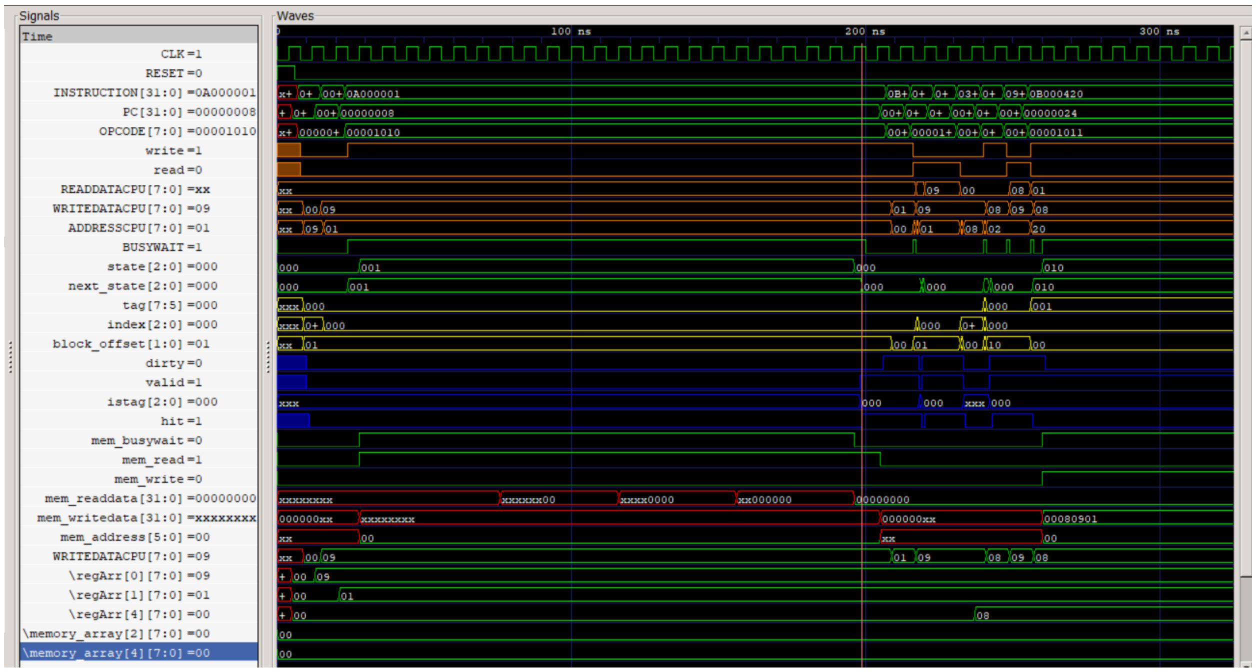
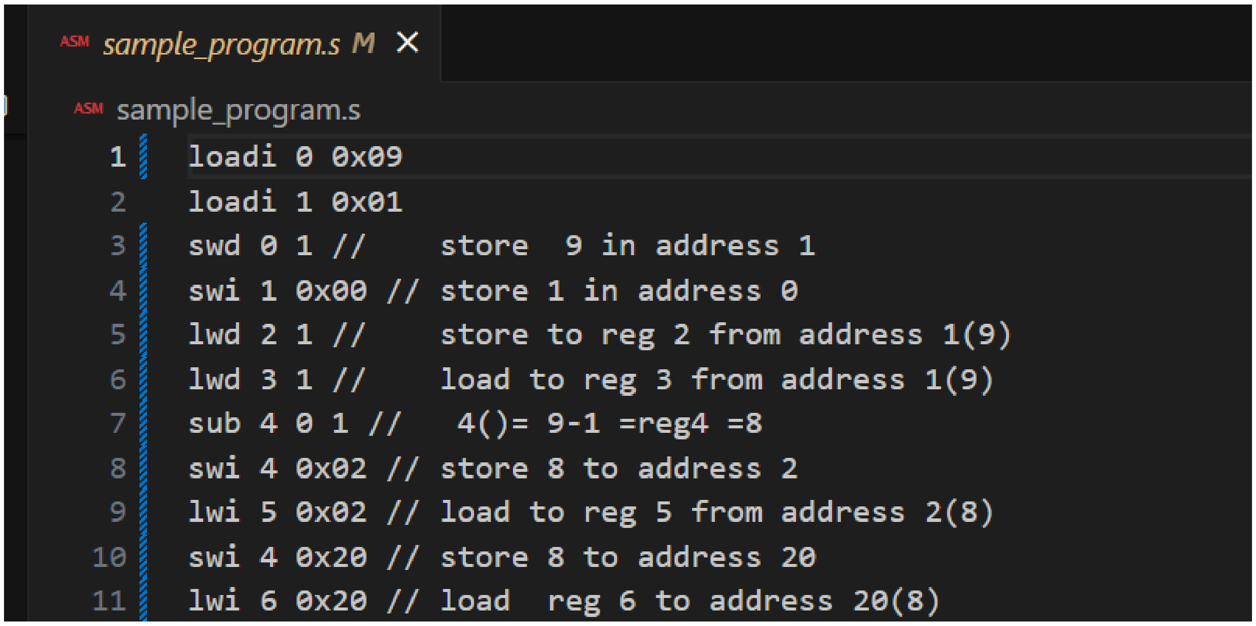
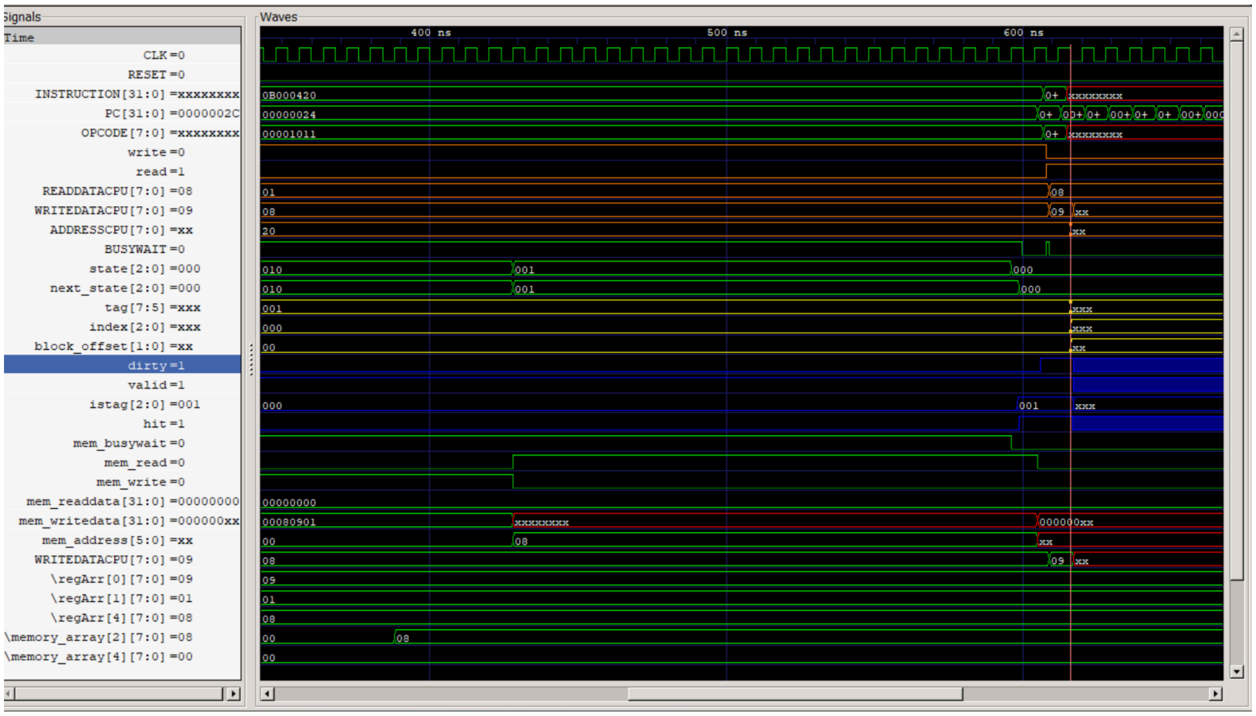

```
Time
CLK=1
RESET=0
INSTRUCTION[31:0]=0A000001
PC[31:0]=00000008
OPCODE[7:0]=00001010
write=1
read=0
READDATACPU[7:0]=xx
WRITEDATACPU[7:0]=09
ADDRESSCPU[7:0]=01
BUSYWAIT=1
state[2:0]=000
next_state[2:0]=000
tag[7:5]=000
index[2:0]=000
block_offset[1:0]=01
dirty=0
valid=1
istag[2:0]=000
hit=1
mem_busywait=0
mem_read=1
mem_write=0
mem_readdata[31:0]=00000000
mem_writedata[31:0]=xxxxxxxx
mem_address[5:0]=00
WRITEDATACPU[7:0]=09
\regArr[0][7:0]=09
\regArr[1][7:0]=01
\regArr[4][7:0]=00
\memory_array[2][7:0]=00
\memory_array[4][7:0]=00
```

Figure :system with cache in write hit/read hit

Figures below shows the difference of time consumption of two systems for the same instruction set.:



```
loadi 0 0x09
loadi 1 0x01
swd 0 1 //     store  9 in address 1
swi 1 0x00 // store 1 in address 0
lwd 2 1 //     store to reg 2 from address 1(9)
lwd 3 1 //     load to reg 3 from address 1(9)
sub 4 0 1 //    4()= 9-1 =reg4 =8
swi 4 0x02 // store 8 to address 2
lwi 5 0x02 // load to reg 5 from address 2(8)
swi 4 0x20 // store 8 to address 20
lwi 6 0x20 // load  reg 6 to address 20(8)
```



```
Time
CLK =0
RESET =0
INSTRUCTION[31:0] =xxxxxxxx
PC[31:0] =0000002C
OPCODE[7:0] =xxxxxxxx
write =0
read =1
READDATACPU[7:0] =08
WRITEDATACPU[7:0] =09
ADDRESSCPU[7:0] =xx
BUSYWAIT =0
state[2:0] =000
next_state[2:0] =000
tag[7:5] =xxx
index[2:0] =xxx
block_offset[1:0] =xx
dirty =1
valid =1
istag[2:0] =001
hit =1
mem_busywait =0
mem_read =0
mem_write =0
mem_readdata[31:0] =00000000
mem_writedata[31:0] =000000xx
mem_address[5:0] =xx
WRITEDATACPU[7:0] =09
\regArr[0][7:0] =09
\regArr[1][7:0] =01
\regArr[4][7:0] =08
\memory_array[2][7:0] =08
\memory_array[4][7:0] =00
```

Figure : System with cache

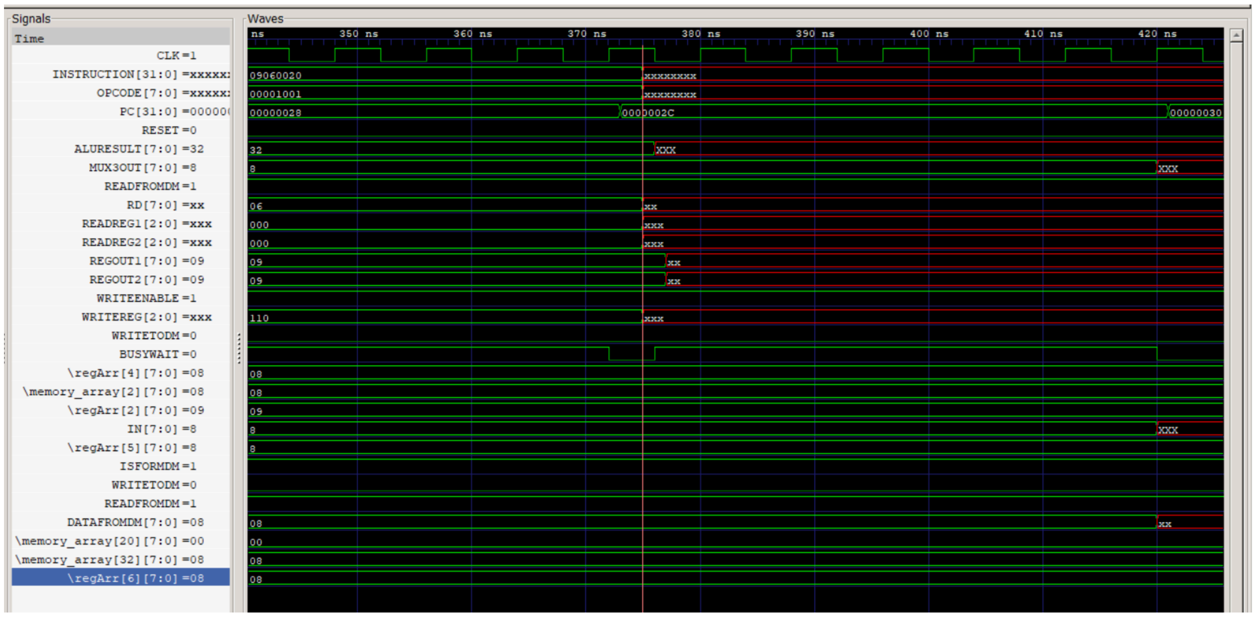


Figure : System without cache

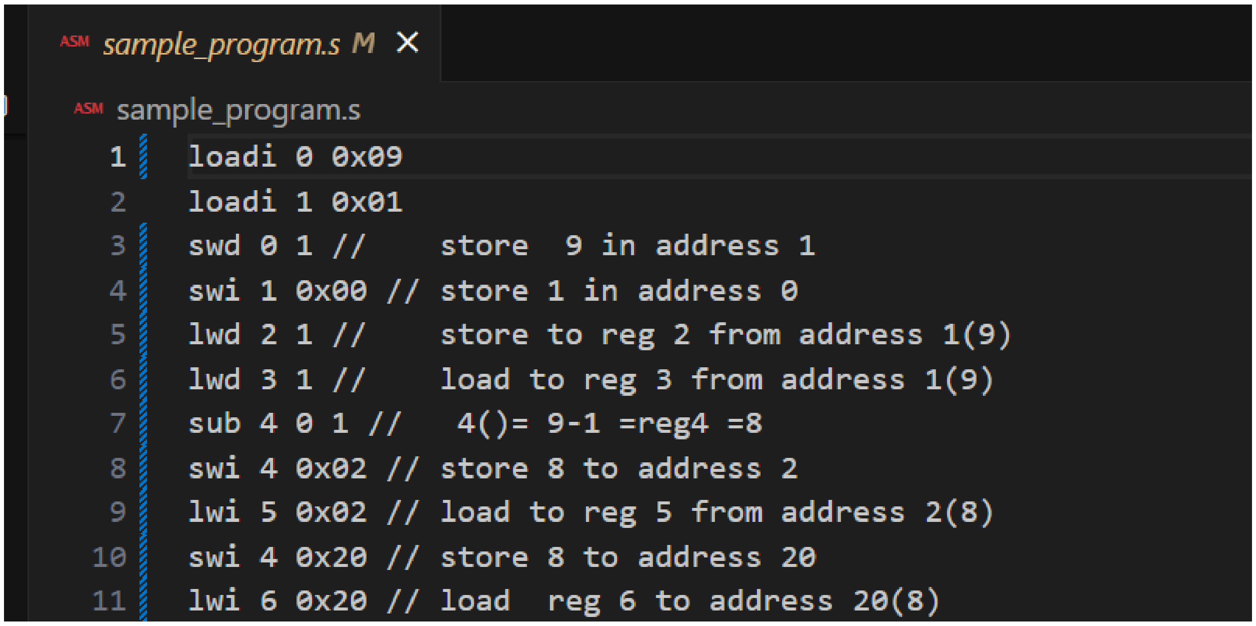
As we can see in the above figures the system with cache completes the program after the system without a cache completes it.Therefore, the system without cache demonstrates greater efficiency overall.This happens because the program requests data from memory locations that are not near each other. If they happened to be near we can see that the system with cache perform with higher speed than the cacheless one.

Without the cache module cpu only have to wait for the main memory to read or write only one block but with the cache it has to wait for four blocks thus four times time delayed.

However, in the system with cache, if a read miss or write miss occurs and the dirty bit is set to 1, the process becomes significantly more time-consuming. Specifically, it takes 42 clock cycles (42x8 = 336 time units) to handle the output of that instruction.

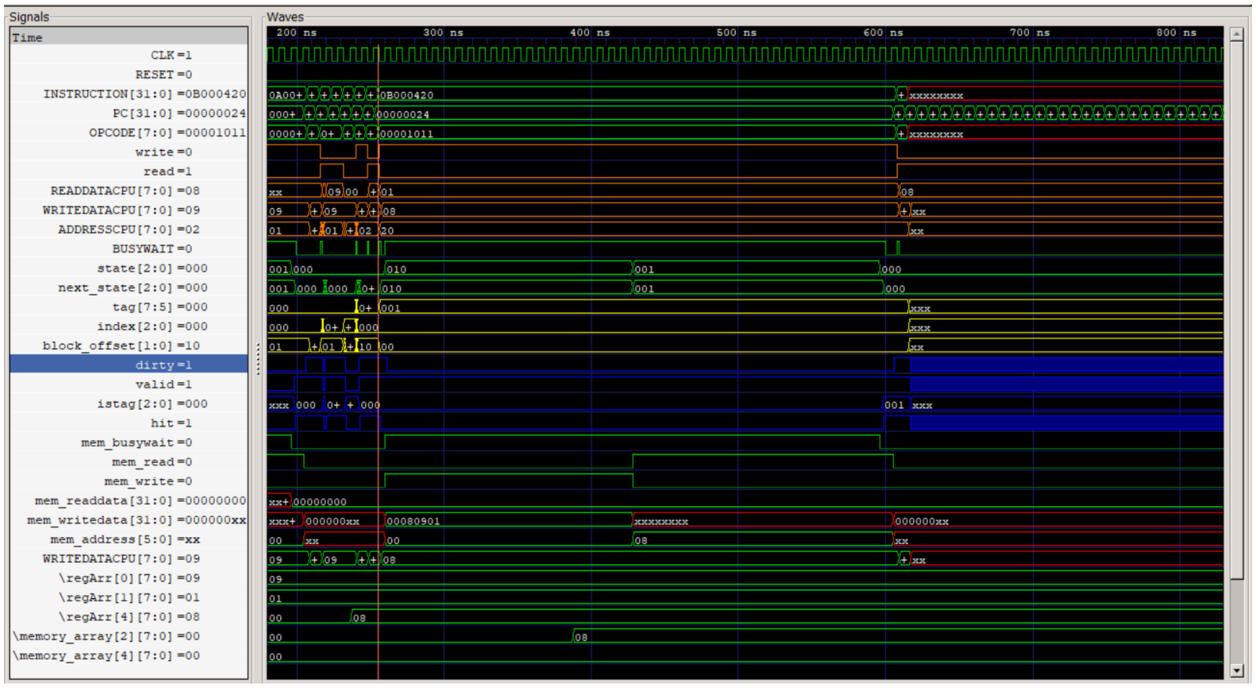
The timing diagram below illustrates the behavior of the system in the event of a miss with a dirty bit set to 1.

Instruction set:


sample_program.s M ×

sample_program.s

```
loadi 0 0x09
loadi 1 0x01
swd 0 1 //     store  9 in address 1
swi 1 0x00 // store 1 in address 0
lwd 2 1 //     store to reg 2 from address 1(9)
lwd 3 1 //     load to reg 3 from address 1(9)
sub 4 0 1 //    4()= 9-1 =reg4 =8
swi 4 0x02 // store 8 to address 2
lwi 5 0x02 // load to reg 5 from address 2(8)
swi 4 0x20 // store 8 to address 20
lwi 6 0x20 // load  reg 6 to address 20(8)
```



```
Time
CLK=1
RESET=0
INSTRUCTION[31:0]=0B000420
PC[31:0]=00000024
OPCODE[7:0]=00001011
write=0
read=1
READDATACPU[7:0]=08
WRITEDATACPU[7:0]=09
ADDRESSCPU[7:0]=02
BUSYWAIT=0
state[2:0]=000
next_state[2:0]=000
tag[7:5]=000
index[2:0]=000
block_offset[1:0]=10
dirty=1
valid=1
istag[2:0]=000
hit=1
mem_busywait=0
mem_read=0
mem_write=0
mem_readdata[31:0]=00000000
mem_writedata[31:0]=000000xx
mem_address[5:0]=xx
WRITEDATACPU[7:0]=09
\regArr[0][7:0]=09
\regArr[1][7:0]=01
\regArr[4][7:0]=08
\memory_array[2][7:0]=00
\memory_array[4][7:0]=00
```

Figure :System With cache

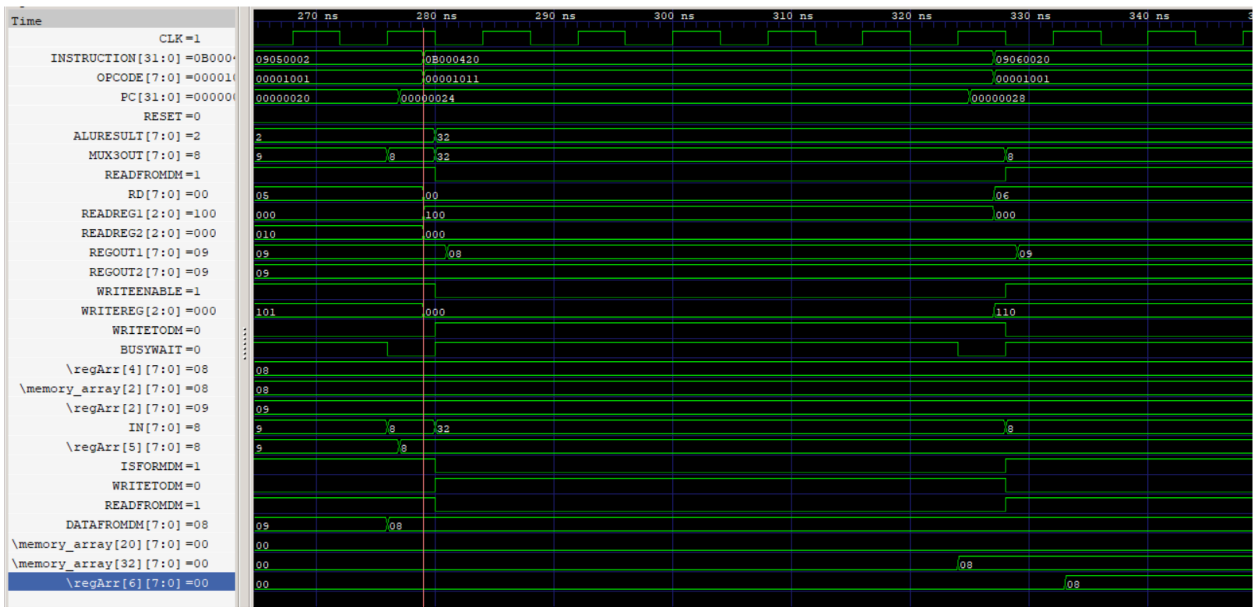


Figure : System without cache

The figures above show the time consumption differences between the two systems for a given instruction set when there is a read miss or write miss with the dirty bit set to 1. These figures clearly demonstrate that the system without cache performs better in this scenario. It is more efficient when there are misses.

In conclusion, if the program achieves a high hit rate, the system with cache is highly efficient due to its ability to quickly access frequently used data. However, if there are numerous misses, the system without cache is more time-efficient in completing instructions compared to the system with cache. This highlights the importance of considering the hit rate when evaluating the performance of a cache-based system versus a cacheless system.