**Dalhousie University Faculty of Engineering**

**ECED 3403**

Assignment 3: Testing

Testing

Submitted by:

Ray Su B00634512

Test Explanation:

This test is to test each type of cache’s functionality, such as Read (hit), Read (miss), write (hit), write (miss).

* Direct mapping cache test:
* Read test:

Test file:

Asm file:

org #$100

Main

add #1,R0 ;increment R0

cmp #2,R0 ;test is R0 = 2?

beq Func ;go func if equal

bal Main

org #$200

Func

movlz #0,R0 ;clear R0

movlz #1,R1 ;set R1 to 1

movlz #2,R2 ;set R2 to 2

bal wait ;finish Func

org #$500

wait

bal wait ;finished

end Main

S-Record:

S00C0000ctest.asm61

S10B01008860906A7D20FC3F39

S10B02000098099812987C3D56

S1050500FF3FB7

S9030100FB

Test procedure:

There will be break points set at PC=100, PC=200 and PC=500. Program will run the first 4 lines of code twice in a loop, it will load instruction from memory to cache in the first time and will hit instructs in the second time. Then goes to Func, load instruction again from memory. And will finally stay in wait. There will be 4 lines of code to run before program hits a break point. Every time hit break point will display clock, so by comparing the time cost can assume if cache works or not.

Expect test result:

Time to run 4-line code for first time should be much longer than it runs second time. Time to run 4-line code for first time should be approximately equal to time to run Func.

Because of direct mapping cache, at the end Func will overwrite Main, so we cannot see main code in cache anymore.

Actual test:

Load S-record and set break points:

Run CPU and hit break point 100, display clock:



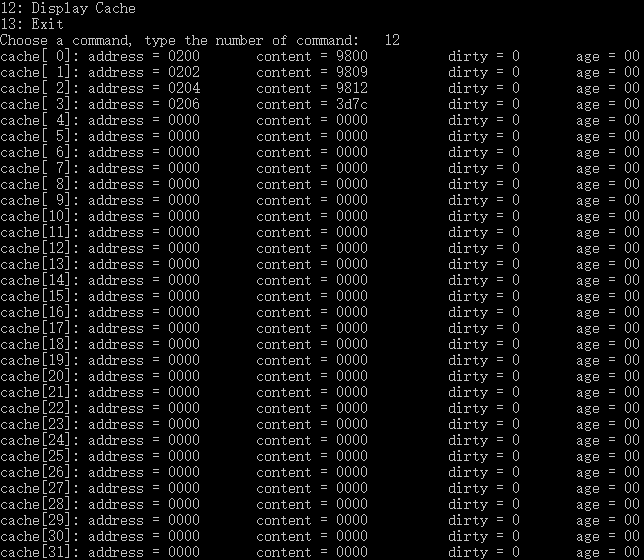
Continue CPU and hit break point 200, display clock:



Continue CPU and hit break point 500, display clock:



Display cache:



Test result:

Time to run 4-line code for first time is 24.

Time to run 4-line code for second time is 9.

Time to Func is 24.

So Read function is working.

* Write through test:

Test file:

Asm file:

addr\_1000 equ #$66

org #$100

Main

movlz addr\_1000,R1 ;set R1 to target memory address

ld R1,R0

add #1,R0 ;increment R0

st R0,R1 ;store R0 to memory

bal Main

org #$66

word #$1234

end Main

S-Record:

S00D0000cwrite.asmF5

S10D0100319B088088600188FB3FF2

S105006634124E

S9030100FB

Expect test result:

Memory and cache is consistent. Testing will keep incrementing the value of memory[0x66], which has initial value of 0x1234. A break point will be set at PC = 100, and value of cache and memory will be displayed. Dirty bit will not be used in this test.

Actual test:

Load S-Record and set break point at 100, run CPU:





Catch break point 1st time, display cache and memory value:







Continue CPU:





Test result:

Cache and memory are consistent, write through is working.

* Write back test:

Test file:

Asm file:

addr\_66 equ #$66

addr\_e6 equ #$e6

org #$100

Main

movlz addr\_66,R1 ;set R1 to target memory[66] address

movlz addr\_e6,R2

ld R1,R0

add #1,R0 ;increment R0

st R0,R1 ;store R0 to memory[66]

st R0,R2 ;store R0 to memory[e6]

bal Main

org #$66

word #$1234

end Main

S-Record:

S00D0000cwrite.asmF5

S10D0100319B088088600188FB3FF2

S105006634124E

S9030100FB

Expect test result:

Memory and cache is consistent. Testing will keep incrementing the value of memory[0x66], which has initial value of 0x1234. A break point will be set at PC = 100, and value of cache and memory will be displayed. Because of write back, data will be write back to cache when dirty bit has been set and will be overwrited. In this test, address 66 and e6 will use same cache, so one will be overwrited by another.

Actual test:

Load S-record and set break points:

Run CPU and hit break point 100, display clock:



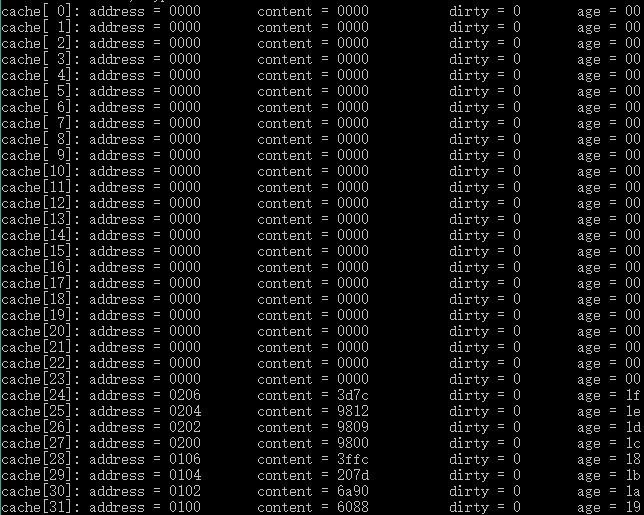
Continue CPU and hit break point 200, display clock:



Continue CPU and hit break point 500, display clock:



Display cache:



Test result:

The old data has been write back to memory, and then overwrited by new data. So write back is working.

* Associative cache test:
* Read test:

Test file:

Asm file:

org #$100

Main

add #1,R0 ;increment R0

cmp #2,R0 ;test is R0 = 2?

beq Func ;go func if equal

bal Main

org #$200

Func

movlz #0,R0 ;clear R0

movlz #1,R1 ;set R1 to 1

movlz #2,R2 ;set R2 to 2

bal wait ;finish Func

org #$500

wait

bal wait ;finished

end Main

S-Record:

S00C0000ctest.asm61

S10B01008860906A7D20FC3F39

S10B02000098099812987C3D56

S1050500FF3FB7

S9030100FB

Test procedure:

There will be break points set at PC=100, PC=200 and PC=500. Program will run the first 4 lines of code twice in a loop, it will load instruction from memory to cache in the first time and will hit instructs in the second time. Then goes to Func, load instruction again from memory. And will finally stay in wait. There will be 4 lines of code to run before program hits a break point. Every time hit break point will display clock, so by comparing the time cost can assume if cache works or not.

Expect test result:

Time to run 4-line code for first time should be much longer than it runs second time. Time to run 4-line code for first time should be approximately equal to time to run Func.

Because of associative mapping cache, Func will have higher age than 4 lines code in Main.

Actual test:

Load S-record and set break points:

Run CPU and hit break point 100, display clock:



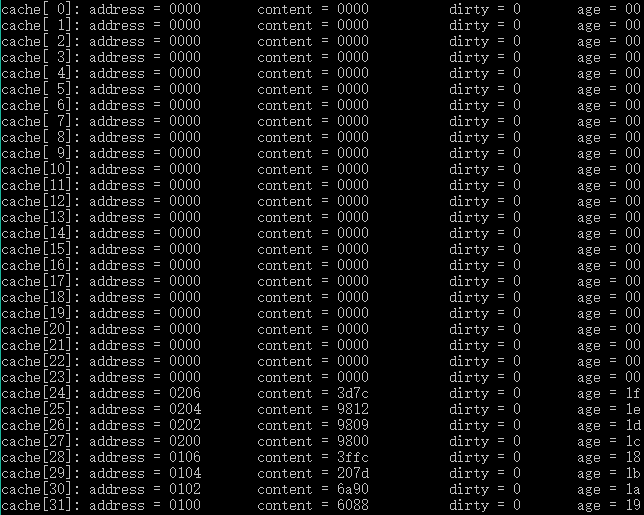
Continue CPU and hit break point 200, display clock:



Continue CPU and hit break point 500, display clock:



Display cache:



Test result:

Time to run 4-line code for first time is 24.

Time to run 4-line code for second time is 9.

Time to Func is 24.

So Read function is working.

* Write back test:

Test file:

Asm file:

addr\_66 equ #$66

addr\_e6 equ #$e6

org #$100

Main

movlz addr\_66,R1 ;set R1 to target memory[66] address

movlz addr\_e6,R2

ld R1,R0

add #1,R0 ;increment R0

st R0,R1 ;store R0 to memory[66]

st R0,R2 ;store R0 to memory[e6]

org #$14c ;fill up cache with content of 0x0000

bal Main

org #$66

word #$1234

end Main

S-Record:

S00D0000cwrite.asmF5

S10F0100319B329F0880886001880288CF

S105014CD93F95

S105006634124E

S9030100FB

Test procedure:

There will be break points set at PC=100, PC=144, and display the cache and memory before overwrite and after overwrite.

Expect test result:

Data in cache with dirty bit set will be stored to memory after it is overwrited.

Actual test:

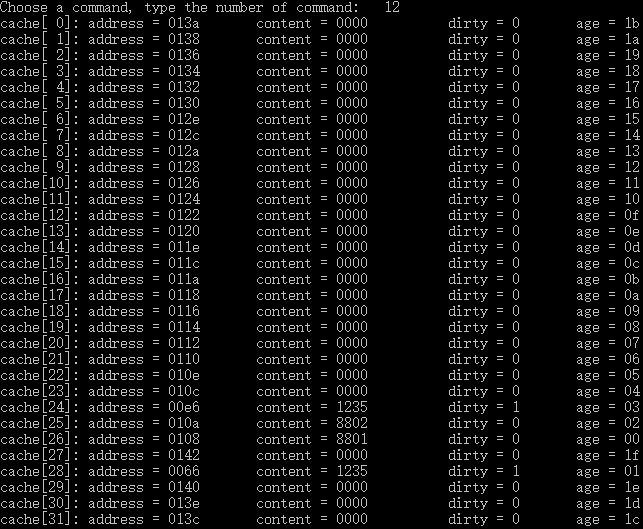
Load S-Record and set break point at 100, run CPU:







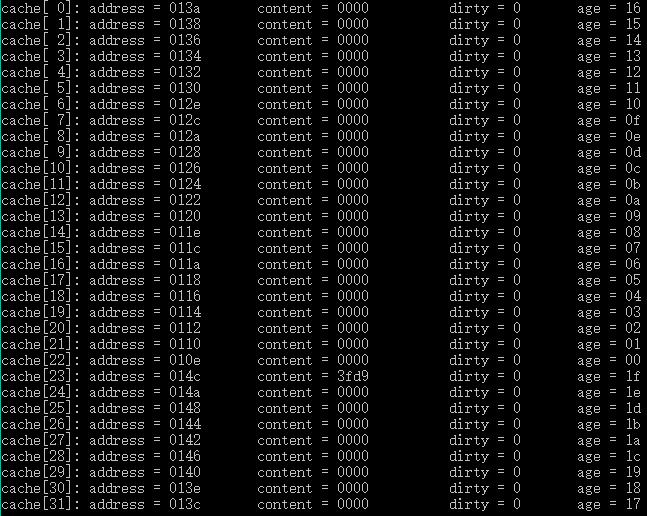
Catch break point 1st time, display cache and memory value (before overwrite):







Catch break point 2nd time, display cache and memory value (after overwrite):







Test Result:

Data with dirty bit set is write back to memory after overwrite, so write back is working.

* Write through test:

Test file:

Asm file:

addr\_66 equ #$66

addr\_e6 equ #$e6

org #$100

Main

movlz addr\_66,R1 ;set R1 to target memory[66] address

movlz addr\_e6,R2

ld R1,R0

add #1,R0 ;increment R0

st R0,R1 ;store R0 to memory[66]

st R0,R2 ;store R0 to memory[e6]

org #$14c

bal Main

org #$66

word #$1234

end Main

S-Record:

S00D0000cwrite.asmF5

S10F0100319B329F0880886001880288CF

S105014CD93F95

S105006634124E

S9030100FB

Test procedure:

There will be break points set at PC=144, and display the cache and memory before overwrite and after overwrite.

Expect test result:

Data will keep consistent with memory no matter it is overwrite or not. Dirty bit will not be used in this test.

Actual test:

Load S-Record and set break point at 100, run CPU:





Catch break point 1st time, display cache and memory value (before overwrite):







Test result:

Cache and memory keep consistent, so write through is working.

* Hybrid cache test:
* Read test:

Test file:

Asm file:

org #$100

Main

add #1,R0 ;increment R0

cmp #2,R0 ;test is R0 = 2?

beq Func ;go func if equal

bal Main

org #$200

Func

movlz #0,R0 ;clear R0

movlz #1,R1 ;set R1 to 1

movlz #2,R2 ;set R2 to 2

bal wait ;finish Func

org #$500

wait

bal wait ;finished

end Main

S-Record:

S00C0000ctest.asm61

S10B01008860906A7D20FC3F39

S10B02000098099812987C3D56

S1050500FF3FB7

S9030100FB

Test procedure:

There will be break points set at PC=100, PC=200 and PC=500. Program will run the first 4 lines of code twice in a loop, it will load instruction from memory to cache in the first time and will hit instructs in the second time. Then goes to Func, load instruction again from memory. And will finally stay in wait. There will be 4 lines of code to run before program hits a break point. Every time hit break point will display clock, so by comparing the time cost can assume if cache works or not.

Expect test result:

Time to run 4-line code for first time should be much longer than it runs second time. Time to run 4-line code for first time should be approximately equal to time to run Func.

Because of hybrid cache, Func will overwrite Main in the first page at the end of program.

Actual test:

Load S-record and set break points:

Run CPU and hit break point 100, display clock:



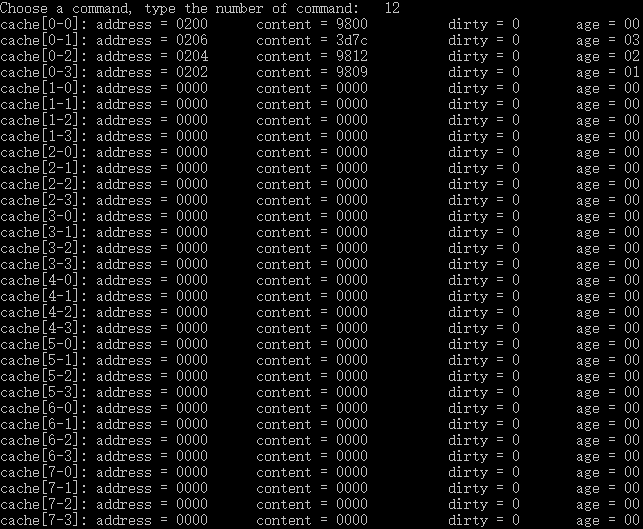
Continue CPU and hit break point 200, display clock:



Continue CPU and hit break point 500, display clock:



Display cache:



* Write through test:

Test file:

Asm file:

addr\_66 equ #$66

addr\_e6 equ #$e6

addr\_1e6 equ #$1e6

addr\_2e6 equ #$2e6

addr\_3e6 equ #$3e6

org #$100

Main

movlz addr\_66,R1 ;set R1 to target memory[66] address

ld R1,R0

add #1,R0 ;increment R0

st R0,R1 ;store R0 to memory[66]

movlz addr\_e6,R1

st R0,R1 ;store R0 to memory[e6]

movlz addr\_1e6,R1

movh addr\_1e6,R1

st R0,R1 ;store R0 to memory[e6]

movlz addr\_2e6,R1

movh addr\_2e6,R1

st R0,R1 ;store R0 to memory[e6]

movlz addr\_3e6,R1

movh addr\_3e6,R1

st R0,R1 ;store R0 to memory[e6]

bal Main

org #$66

word #$1234

end Main

S-Record:

S00D0000hwrite.asmF0

S1210100319B088088600188319F0188319F09A00188319F11A00188319F19A00188A1

S105011EF03FAC

S105006634124E

S9030100FB

Test procedure:

There will be break points set at PC=11c and PC=11e, and display the cache and memory before overwrite and after overwrite.

Expect test result:

Data will keep consistent with memory no matter it is overwrite or not. Dirty bit will not be used in this test.

Actual test:

Load S-record, set break point:







Run CPU. Catch break point 1st time, display cache and memory value (before overwrite):



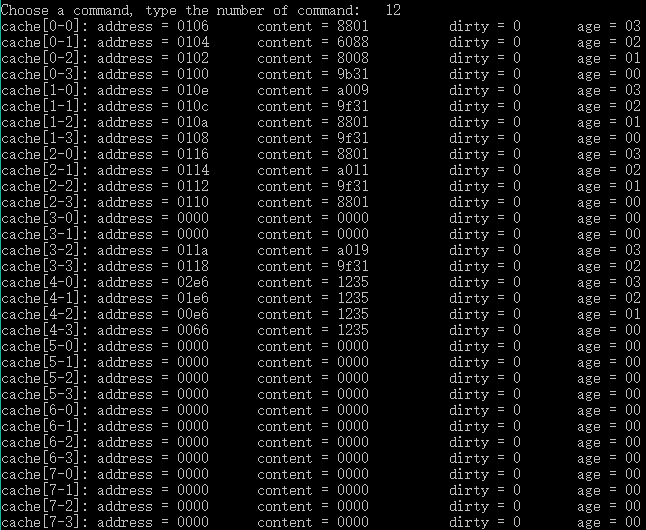












Continue CPU. Catch break point 2nd time, display cache and memory value (after overwrite):



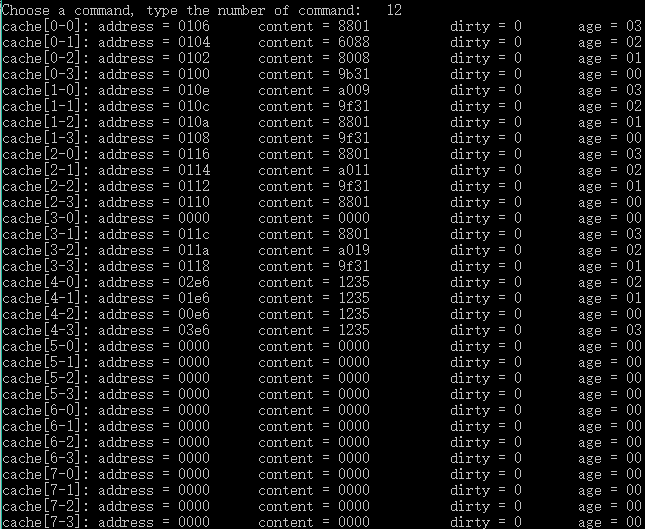












* Write through test:

Test file:

Asm file:

addr\_66 equ #$66

addr\_e6 equ #$e6

addr\_1e6 equ #$1e6

addr\_2e6 equ #$2e6

addr\_3e6 equ #$3e6

org #$100

Main

movlz addr\_66,R1 ;set R1 to target memory[66] address

ld R1,R0

add #1,R0 ;increment R0

st R0,R1 ;store R0 to memory[66]

movlz addr\_e6,R1

st R0,R1 ;store R0 to memory[e6]

movlz addr\_1e6,R1

movh addr\_1e6,R1

st R0,R1 ;store R0 to memory[e6]

movlz addr\_2e6,R1

movh addr\_2e6,R1

st R0,R1 ;store R0 to memory[e6]

movlz addr\_3e6,R1

movh addr\_3e6,R1

st R0,R1 ;store R0 to memory[e6]

bal Main

org #$66

word #$1234

end Main

S-Record:

S00D0000hwrite.asmF0

S1210100319B088088600188319F0188319F09A00188319F11A00188319F19A00188A1

S105011EF03FAC

S105006634124E

S9030100FB

Test procedure:

There will be break points set at PC=11c and PC=11e, and display the cache and memory before overwrite and after overwrite.

Expect test result:

Data with dirty bit set will be write back to memory when overwrite.

Actual test:

Load S-record, set break point:







Run CPU. Catch break point 1st time, display cache and memory value (before overwrite):



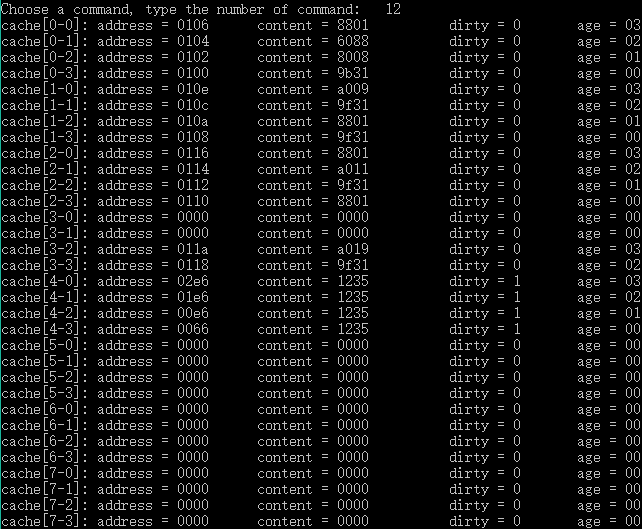












Continue CPU. Catch break point 2nd time, display cache and memory value (after overwrite):



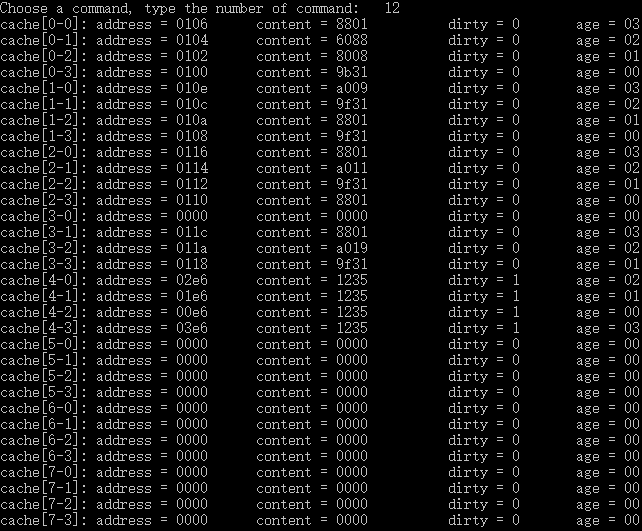












Test result:

Data with dirty bit set has been wrote back to memory when overwrite, so write back is working.

**Dalhousie University Faculty of Engineering**

**ECED 3403**

Assignment 3: Cache

Code

Submitted by:

Ray Su B00634512

This code only contains code of cache. There are only three changes in the other files. The first is instead of calling bus(), CPU will call cache(). The second change is there is new function in debugger to display caches, which is a tool to help us to see the value in the cache. The last change is in debugger, which is added the constructor of cache and creates a cache object.