

# Report: studio sincronizzazione variabili condivise e istruzioni atomiche su architettura RISC-V

Esame di Embedded systems - 056899 A.A. 2023/2024

Francesco Maria Tranquillo  
Gioele Peltrera

# Introduzione

In questo report analizzeremo i metodi software per la sincronizzazione multi-thread e la loro implementazione hardware, tra cui mutex, semafori, spinlock e compare exchange. Inoltre affronteremo anche il tema delle istruzioni atomiche e illustreremo una soluzione architetturale per una cpu RISC-V che possa supportare tali soluzioni.

I test eseguiti sono frutto di simulazioni attraverso la piattaforma gem5, utilizzando una cpu timing o O3 a due core, con un protocollo di coerenza MESI a due livelli implementato attraverso Ruby, e sistema operativo Linux.

```
41 import m5
42 from m5.objects import Root
43
44 from gem5.components.boards.riscv_board import RiscvBoard
45 from gem5.components.cachehierarchies.ruby.mesi_two_level_cache_hierarchy import (
46     MESITwoLevelCacheHierarchy,
47 )
48 from gem5.coherence_protocol import CoherenceProtocol
49 from gem5.components.memory import SingleChannelDDR3_1600
50 from gem5.components.processors.cpu_types import CPUTypes
51 from gem5.components.processors.simple_switchable_processor import (
52     SimpleSwitchableProcessor,
53 )
54 from gem5.isas import ISA
55 from gem5.resources.resource import DiskImageResource, obtain_resource, CheckpointResource
56 from gem5.simulate.simulator import Simulator
57 from gem5.utils.requires import requires
58
59 # Run a check to ensure the right version of gem5 is being used.
60 requires(
61     isa_required=ISA.RISCV,
62     coherence_protocol_required=CoherenceProtocol.MESI_TWO_LEVEL,
63 )
64
65 # Setup the cache hierarchy.
66 # For classic, PrivateL1PrivateL2 and NoCache have been tested.
67 # For Ruby, MESI_Two_Level and MI_example have been tested.
68 cache_hierarchy = MESITwoLevelCacheHierarchy(
69     l1d_size="32kB",
70     l1d_assoc=8,
71     l1i_size="32kB",
72     l1i_assoc=8,
73     l2_size="256kB",
74     l2_assoc=16,
75     num_l2_banks=2,
76 )
77 # Setup the system memory.
78 memory = SingleChannelDDR3_1600()
79
80 # Setup a single core Processor.
81 processor = SimpleSwitchableProcessor(
82     starting_core_type = CPUTypes.ATOMIC,
83     switch_core_type=CPUTypes.O3,#CPUTypes.TIMING,
84     isa=ISA.RISCV,
85     num_cores=2
86 )
87
88 # Setup the board.
89 board = RiscvBoard(
90     clk_freq="1GHz",
91     processor=processor,
92     memory=memory,
93     cache_hierarchy=cache_hierarchy,
94 )
95
96 # Set the Full System workload.
97 board.set_kernel_disk_workload(
98
99     kernel=obtain_resource("riscv-bootloader-vmlinux-5.10", "/home/user/Documents/POLIMI/HD/binaries/"
100 ),
101     disk_image=DiskImageResource("/home/user/Documents/POLIMI/HD/ubuntu_ARM/riscv-disk-img"),
102     readfile="/home/user/Documents/POLIMI/HD/sources/acquire_release",
103     checkpoint = CheckpointResource("/home/user/Documents/POLIMI/HD/gem5_installation/gem5/m5out/cpt.281422923000")
104 )
```

```

104 simulator = Simulator(board=board)
105 print("Beginning simulation!")
106 # Note: This simulation will never stop. You can access the terminal upon boot
107 # using m5term ('./util/term'): `./m5term localhost <port>`. Note the `<port>`
108 # value is obtained from the gem5 terminal stdout. Look out for
109 # "system.platform.terminal: Listening for connections on port <port>".
110 simulator.run()
111
112

```

Per ottenere dei risultati più significativi, all'inizio del codice di ogni thread verrà messo un ciclo che continuerà a ripetersi finché entrambi i thread non si troveranno in quello stesso ciclo, così che il primo thread creato non inizi subito l'esecuzione della zona critica prima che il secondo thread venga creato dal main.

```

pthread_spinlock_t sp_lock;

void *func(void *par)
{
    int increm = *((int *)par);
    unsigned int old_val = 0;
    int local_start = 0;

    while(!local_start)
    {
        pthread_spin_lock(&sp_lock);
        if(start!=0 && start!=increm)
            local_start=1;
        start = increm;
        pthread_spin_unlock(&sp_lock);
    }
    ...
    ...
}

```

# SpinLock

```
1#include <stdio.h>
2#include <pthread.h>
3#include <stdlib.h>
4// #include <time.h>
5#include <gen5/m5ops.h>
6#define N_TIMES 10
7#define N_INCREM 3
8
9pthread_spinlock_t sp_lock;
10unsigned int shared_value = 0;
11int start = 0;
12int change = 0;
13
14void *func(void *par)
15{
16    int increment = *((int *)par);
17    unsigned int old_val = 0;
18    int local_start = 0;
19
20    while(!local_start)
21    {
22        pthread_spin_lock(&sp_lock);
23        if(start!=0 && start!=increment)
24            local_start=1;
25        start = increment;
26        pthread_spin_unlock(&sp_lock);
27    }
28
29    for(int i=0; i<N_TIMES; i++)
30    {
31        pthread_spin_lock(&sp_lock);
32        if(shared_value!=old_val)
33            change++;
34        for(int j=0; j<N_INCREM; j++)
35        {
36            shared_value = shared_value + increment;
37        }
38        old_val=shared_value;
39        pthread_spin_unlock(&sp_lock);
40    }
41    return NULL;
42}
43
44int main()
45{
46    pthread_t thread_1, thread_2;
47    int param_1= 1;
48    int param_2= 2;
49
50    if(pthread_spin_init(&sp_lock, PTHREAD_PROCESS_PRIVATE) != 0)
51    {
52        printf("ERROR in initializing spin lock!\n");
53        return -1;
54    }
55    m5_switch_cpu();
56    pthread_create(&thread_1, NULL, func, (void *)&param_1);
57    pthread_create(&thread_2, NULL, func, (void *)&param_2);
58
59    pthread_join(thread_1, NULL);
60    pthread_join(thread_2, NULL);
61    if(pthread_spin_destroy(&sp_lock) != 0)
62    {
63        printf("ERROR in initializing spin lock!\n");
64        return -1;
65    }
66    printf("Number of changes in thread lock owning: %d\n", change);
67    fflush(stdout);
68    m5_exit(0);
69    return 0;
70}
```

Lo spin lock esegue un'operazione di atomic-swap (marchiata come acquire, ergo tutte le operazioni successive alla swap non potranno essere riordinate ed eseguite prima di questa operazione atomica).

Successivamente se l'atomic swap dovesse aver fallito, quindi nel caso in cui il valore contenuto all'indirizzo di memoria della variabile globale di lock fosse già 1, grazie alla "bnez" invece che concludere la funzione con una ret, passiamo ad un loop, che precede la logica di load reserve-store conditional.

```
00000000001a940 <__pthread_spin_lock>:
1a940: 4785          li a5,1
1a942: 0cf527af      amoswap.w.aq a5,a5,(a0)
1a946: 2781          sext.w a5,a5
1a948: e399          bnez a5,1a94e <__pthread_spin_lock+0xe>
1a94a: 4501          li a0,0 PRESO LOCK
1a94c: 8082          ret
1a94e: 4705          li a4,1 NON PRESO LOCK
1a950: 411c          lw a5,0(a0)
1a952: fffd          bnez a5,1a950 <__pthread_spin_lock+0x10>
1a954: 100527af      lr.w a5,(a0)
1a958: e781          bnez a5,1a960 <__pthread_spin_lock+0x20>
1a95a: 1ce526af      sc.w.aq a3,a4,(a0)
1a95e: fafd          bnez a3,1a954 <__pthread_spin_lock+0x14>
1a960: 2781          sext.w a5,a5
1a962: f7fd          bnez a5,1a950 <__pthread_spin_lock+0x10>
1a964: 4501          li a0,0 PRESO LOCK
1a966: 8082          ret

00000000001a968 <__pthread_spin_unlock>:
1a968: 0f50000f      fence iorw,ow
1a96c: 0805202f      amoswap.w zero,zero,(a0)
1a970: 4501          li a0,0
1a972: 8082          ret
```

Il core switch1, a riga 618, è il primo a concludere correttamente l'atomic swap, quindi tutte le successive atomic swap non ritorneranno più 0, ma 1 (l'indirizzo 0x8a8b0 si vede dal file di object dump che rappresenta l'indirizzo della variabile globale sp\_lock).

```

612 290489370000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: No tag match for address: 0x10107d880
613 290489371000: board.processor.switch1.core.mmu.dtb: lookup(vpn=0x8a8b0, asid=0): hit ppn 0x10107d
614 290489371000: board.processor.switch1.core.mmu.dtb: translate(vpn=0x8a8b0, asid=0): 0x10107d8b0
615 290489372000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: address: 0x10107d880 found
616 290489372000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Icache: No tag match for address: 0x10107d880
617 290489372000: board.cache_hierarchy.ruby_system.l1_controllers1.MESI_Two_Level-L1cache.sm:860: [ 0x1 0x0 0x0 0x0 0x0 0x0
618 290489372000: global: Testing Lock for addr: 0x10107d880 cur -1 con 1
619 290489372000: board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: Cache hit at [0x10107d8b0, line 0x10107d880]
620 290489372000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: address: 0x10107d880 found
621 290489372000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Icache: No tag match for address: 0x10107d880
622 290489373000: board.processor.switch1.core.mmu.itb: lookup(vpn=0x1a944, asid=0): hit ppn 0x101268
623 290489373000: board.processor.switch1.core.mmu.itb: translate(vpn=0x1a944, asid=0): 0x101268944
624 290489373000: board.cache_hierarchy.ruby_system.l2_controllers0.MESI_Two_Level-L2cache.sm:373: Addr: 0x10107d880 State: M
625 290489374000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: No tag match for address: 0x101268940
626 290489374000: board.cache_hierarchy.ruby_system.l1_controllers1.MESI_Two_Level-L1cache.sm:843: [ 0x85 0x47 0xaf 0x27 0xf5
627 290489374000: board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: Cache hit at [0x101268944, line 0x101268940]
628 290489374000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: No tag match for address: 0x101268940
629 290489375000: board.processor.switch1.core.mmu.itb: lookup(vpn=0x1a948, asid=0): hit ppn 0x101268 bnez
630 290489375000: board.processor.switch1.core.mmu.itb: translate(vpn=0x1a948, asid=0): 0x101268948
631 290489376000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: No tag match for address: 0x101268940
632 290489376000: board.cache_hierarchy.ruby_system.l1_controllers1.MESI_Two_Level-L1cache.sm:843: [ 0x85 0x47 0xaf 0x27 0xf5
633 290489376000: board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: Cache hit at [0x101268948, line 0x101268940]
634 290489376000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: No tag match for address: 0x101268940
635 290489377000: board.processor.switch1.core.mmu.itb: lookup(vpn=0x1a948, asid=0): hit ppn 0x101268
636 290489377000: board.processor.switch1.core.mmu.itb: translate(vpn=0x1a948, asid=0): 0x101268948
637 290489378000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: address: 0x10107d880 found
638 290489378000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Icache: No tag match for address: 0x10107d880
639 290489378000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: address: 0x10107d880 found

```

Il core switch0, a riga 670, ritorna dall'atomic swap con un risultato diverso da 0 dopo aver fatto una miss sulla variabile sp\_lock, poichè la sua entry (presente nella cache L1 del core) è stata invalidata dalla prima atomic swap del core 1.

```

668 290489384000: board.cache_hierarchy.ruby_system.l1_controllers0.L1Icache: No tag match for address: 0x10107d880
669 290489384000: board.cache_hierarchy.ruby_system.l1_controllers0.MESI_Two_Level-L1cache.sm:869: [ 0x1 0x0 0x0 0x0 0x0 0x0
670 290489384000: global: Testing Lock for addr: 0x10107d880 cur -1 con 0
671 290489384000: board.cache_hierarchy.ruby_system.l1_controllers0.sequencer: Cache miss at [0x10107d8b0, line 0x10107d880]
672 290489384000: board.cache_hierarchy.ruby_system.l1_controllers0.MESI_Two_Level-L1cache.sm:826: 0x10107d880
673 290489384000: board.cache_hierarchy.ruby_system.l1_controllers0.L1Dcache: address: 0x10107d880 found
674 290489384000: board.cache_hierarchy.ruby_system.l1_controllers0.L1Icache: No tag match for address: 0x10107d880
675 290489384000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: address: 0x10107d880 found
676 290489384000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Icache: No tag match for address: 0x10107d880
677 290489384000: board.cache_hierarchy.ruby_system.l1_controllers1.MESI_Two_Level-L1cache.sm:552: address: 0x10107d880, dest
678 290489384000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: address: 0x10107d880 found
679 290489384000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Icache: No tag match for address: 0x10107d880

```

```

687 290489387000: board.processor.switch0.core.mmu.itb: lookup(vpn=0x1a948, asid=0): hit ppn 0x101268      bnez
688 290489387000: board.processor.switch0.core.mmu.itb: translate(vpn=0x1a948, asid=0): 0x101268948
689 290489388000: board.cache_hierarchy.ruby.system.l1_controllers0.L1Dcache: No tag match for address: 0x101268940
690 290489388000: board.cache_hierarchy.ruby.system.l1_controllers0.MESI_Two_Level-L1Cache.sm:843: [ 0x85 0x47 0xaf 0x27 0xf5 0xa0 0x00 0x00 ]
691 290489388000: board.cache_hierarchy.ruby.system.l1_controllers0.L1Dcache: No tag match for address: 0x101268940
692 290489389000: board.processor.switch0.core.mmu.itb: lookup(vpn=0x1a94c, asid=0): hit ppn 0x101268      li
693 290489389000: board.processor.switch0.core.mmu.itb: translate(vpn=0x1a94c, asid=0): 0x10126894c
694 290489389000: board.cache_hierarchy.ruby.system.l2_controllers0.MESI_Two_Level-L2Cache.sm:309: Addr: 0x10107d880 State: MT_M
695 290489389000: board.cache_hierarchy.ruby.system.l2_controllers0.MESI_Two_Level-L2Cache.sm:190: machineID: L2Cache-0, requestorID: 0
696 290489390000: board.cache_hierarchy.ruby.system.l2_controllers0.MESI_Two_Level-L2Cache.sm:373: Addr: 0x10107d880 State: MT_R
697 290489390000: board.cache_hierarchy.ruby.system.l1_controllers0.L1Dcache: No tag match for address: 0x101268940
698 290489390000: board.cache_hierarchy.ruby.system.l1_controllers0.MESI_Two_Level-L1Cache.sm:843: [ 0x85 0x47 0xaf 0x27 0xf5 0xa0 0x00 0x00 ]
699 290489390000: board.cache_hierarchy.ruby.system.l1_controllers0.sequencer: Cache hit at [0x10126894c, line 0x101268940]
700 290489390000: board.cache_hierarchy.ruby.system.l1_controllers0.L1Dcache: No tag match for address: 0x101268940
701 290489391000: board.processor.switch0.core.mmu.itb: lookup(vpn=0x1a950, asid=0): hit ppn 0x101268      lw
702 290489391000: board.processor.switch0.core.mmu.itb: translate(vpn=0x1a950, asid=0): 0x101268950
703 290489392000: board.cache_hierarchy.ruby.system.l1_controllers0.L1Dcache: No tag match for address: 0x101268940
704 290489392000: board.cache_hierarchy.ruby.system.l1_controllers0.MESI_Two_Level-L1Cache.sm:843: [ 0x85 0x47 0xaf 0x27 0xf5 0xa0 0x00 0x00 ]
705 290489392000: board.cache_hierarchy.ruby.system.l1_controllers0.sequencer: Cache hit at [0x101268950, line 0x101268940]
706 290489392000: board.cache_hierarchy.ruby.system.l1_controllers0.L1Dcache: No tag match for address: 0x101268940
707 290489393000: board.processor.switch0.core.mmu.dtb: lookup(vpn=0x8a8b0, asid=0): hit ppn 0x10107d      translate(vpn=0x8a8b0, asid=0): 0x10107d8b0
708 290489393000: board.processor.switch0.core.mmu.dtb:
709 290489394000: board.cache_hierarchy.ruby.system.l1_controllers0.L1Dcache: address: 0x10107d880 found
710 290489394000: board.cache_hierarchy.ruby.system.l1_controllers0.L1Cache: no tag match for address: 0x10107d880
711 290489394000: board.cache_hierarchy.ruby.system.l1_controllers0.MESI_Two_Level-L1Cache.sm:835: f 0x1 0x0 0x0 0x0 0x0 0x0 0x0 0x0
712 290489394000: board.cache_hierarchy.ruby.system.l1_controllers0.sequencer: Cache hit at [0x10107d8b0, line 0x10107d880]
713 290489394000: board.cache_hierarchy.ruby.system.l1_controllers0.L1Dcache: address: 0x10107d880 found
714 290489394000: board.cache_hierarchy.ruby.system.l1_controllers0.L1Cache: no tag match for address: 0x10107d880
715 290489395000: board.processor.switch0.core.mmu.itb: lookup(vpn=0x1a950, asid=0): hit ppn 0x101268      bnez
716 290489395000: board.processor.switch0.core.mmu.itb: translate(vpn=0x1a950, asid=0): 0x101268950
717 290489395000: board.cache_hierarchy.ruby.system.l1_controllers0.L1Dcache: address: 0x10107d880 found

```

Entrambi i core riescono a fare hit poiché la variabile `sp_lock` è usata da entrambi in sola lettura.

```

2015 29048963000: board.cache_hierarchy.ruby_system.ll_controllers0.L1Dcache: No tag match for address: 0x101268940
2016 290489633000: board.processor.switch0.core.mmu.itb: lookup(vpn=0x105e4, asid=0): hit ppn 0x10125e4
2017 290489633000: board.processor.switch0.core.mmu.itb: translate(vpn=0x105e4, asid=0): 0x10125e5e4 jal>spin_unlock
2018 290489633000: board.processor.switch0.core.mmu.dtb: lookup(vpn=0x8a8b0, asid=0): hit ppn 0x10107d
2019 290489634000: board.processor.switch0.core.mmu.dtb: translate(vpn=0x8a8b0, asid=0): 0x10107d8b0
2020 290489634000: board.cache_hierarchy.ruby_system.ll_controllers0.L1Dcache: address: 0x10107d8b0 found
2021 290489634000: board.cache_hierarchy.ruby_system.ll_controllers0.L1Dcache: No tag match for address: 0x10107d8b0
2022 290489634000: board.cache_hierarchy.ruby_system.ll_controllers0.MESI_Two_Level_L1Dcache.sm:835: [ 0x1 0x0 0x0 0x0 0x0 0x0
2023 290489634000: board.cache_hierarchy.ruby_system.ll_controllers0.sequencer: Cache hit at [0x10107d8b0, line 0x10107d8b0]
2024 290489634000: board.cache_hierarchy.ruby_system.ll_controllers0.L1Dcache: address: 0x10107d8b0 found
2025 290489634000: board.cache_hierarchy.ruby_system.ll_controllers0.L1Dcache: No tag match for address: 0x10107d8b0
2026 290489634000: board.cache_hierarchy.ruby_system.ll_controllers0.L1Dcache: No tag match for address: 0x10125e5c0
2027 290489634000: board.cache_hierarchy.ruby_system.ll_controllers0.MESI_Two_Level_L1Dcache.sm:843: [ 0x85 0x27 0x23 0x20 0xf4
2028 290489634000: board.cache_hierarchy.ruby_system.ll_controllers0.sequencer: Cache hit at [0x10125e5e4, line 0x10125e5c0]
2029 290489635000: board.processor.switch0.core.mmu.itb: lookup(vpn=0x1a950, asid=0): hit ppn 0x101268
2030 290489635000: board.processor.switch0.core.mmu.itb: translate(vpn=0x1a950, asid=0): 0x101268950
2031 290489635000: board.processor.switch1.core.mmu.itb: lookup(vpn=0x1a968, asid=0): hit ppn 0x101268 fence
2032 290489635000: board.processor.switch1.core.mmu.itb: translate(vpn=0x1a968, asid=0): 0x101268968
2033 290489636000: board.cache_hierarchy.ruby_system.ll_controllers0.L1Dcache: No tag match for address: 0x101268940
2034 290489636000: board.cache_hierarchy.ruby_system.ll_controllers0.MESI_Two_Level_L1Dcache.sm:843: [ 0x85 0x47 0xaf 0x27 0xf5
2035 290489636000: board.cache_hierarchy.ruby_system.ll_controllers0.sequencer: Cache hit at [0x101268968, line 0x101268940]
2036 290489636000: board.cache_hierarchy.ruby_system.ll_controllers0.L1Dcache: No tag match for address: 0x101268940
2037 290489636000: board.cache_hierarchy.ruby_system.ll_controllers0.L1Dcache: No tag match for address: 0x101268940
2038 290489636000: board.cache_hierarchy.ruby_system.ll_controllers0.MESI_Two_Level_L1Dcache.sm:843: [ 0x85 0x47 0xaf 0x27 0xf5
2039 290489636000: board.cache_hierarchy.ruby_system.ll_controllers0.sequencer: Cache hit at [0x101268950, line 0x101268940]
2040 290489636000: board.cache_hierarchy.ruby_system.ll_controllers0.L1Dcache: No tag match for address: 0x101268940
2041 290489637000: board.processor.switch1.core.mmu.itb: lookup(vpn=0x1a96c, asid=0): hit ppn 0x101268 amoswap
2042 290489637000: board.processor.switch1.core.mmu.itb: translate(vpn=0x1a96c, asid=0): 0x10126896c
2043 290489637000: board.processor.switch0.core.mmu.itb: lookup(vpn=0x1a950, asid=0): hit ppn 0x101268
2044 290489637000: board.processor.switch0.core.mmu.itb: translate(vpn=0x1a950, asid=0): 0x101268950
2045 290489638000: board.cache_hierarchy.ruby_system.ll_controllers0.L1Dcache: No tag match for address: 0x101268940
2046 290489638000: board.cache_hierarchy.ruby_system.ll_controllers0.MESI_Two_Level_L1Dcache.sm:843: [ 0x85 0x47 0xaf 0x27 0xf5
2047 290489638000: board.cache_hierarchy.ruby_system.ll_controllers0.sequencer: Cache hit at [0x101268950, line 0x101268940]
2048 290489638000: board.cache_hierarchy.ruby_system.ll_controllers0.L1Dcache: No tag match for address: 0x101268940
2049 290489638000: board.cache_hierarchy.ruby_system.ll_controllers0.L1Dcache: No tag match for address: 0x101268940
2050 290489638000: board.cache_hierarchy.ruby_system.ll_controllers0.MESI_Two_Level_L1Dcache.sm:843: [ 0x85 0x47 0xaf 0x27 0xf5
2051 290489638000: board.cache_hierarchy.ruby_system.ll_controllers0.sequencer: Cache hit at [0x10126896c, line 0x101268940]
2052 290489638000: board.cache_hierarchy.ruby_system.ll_controllers0.L1Dcache: No tag match for address: 0x101268940
2053 290489639000: board.processor.switch0.core.mmu.dtb: lookup(vpn=0x8a8b0, asid=0): hit ppn 0x10107d
2054 290489639000: board.processor.switch0.core.mmu.dtb: translate(vpn=0x8a8b0, asid=0): 0x10107d8b0
2055 290489639000: board.processor.switch1.core.mmu.dtb: lookup(vpn=0x8a8b0, asid=0): hit ppn 0x10107d
2056 290489639000: board.processor.switch1.core.mmu.dtb: translate(vpn=0x8a8b0, asid=0): 0x10107d8b0

```



A riga 2121, quando si conclude l'unlock, il core 1 fa miss sull'indirizzo della variabile `sp_lock`, poiché la modifica fatta dall'atomic-swap invalida tutti i blocchi della cache.

```
2118 29048967000: board.cache_hierarchy.ruby_system.l1_controllers1: MESI_Two_Level-L1cache.sm:826: 0x10107d880
2119 290489657000: board.cache_hierarchy.ruby_system.l1_controllers1: MESI_Two_Level-L1cache.sm:869: [ 0x1 0x0 0x0 0x0 0x0 0x0 0x0 0
2120 290489657000: global: Testing Lock for addr: 0x10107d880 cur -1 con 1
2121 290489657000: board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: Cache miss at [0x10107d8b0, line 0x10107d880]
2122 290489657000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: address: 0x10107d880 found
2123 290489657000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Icache: No tag match for address: 0x10107d880
2124 290489658000: board.processor.switch1.core.mmu.itb: lookup(vpn=0x1a970, asid=0): hit ppn 0x101268
2125 290489658000: board.processor.switch1.core.mmu.itb: translate(vpn=0x1a970, asid=0): 0x101268970
2126 290489658000: board.cache_hierarchy.ruby_system.l2_controllers0: MESI_Two_Level-L2cache.sm:373: Addr: 0x10107d880 State: SS_MB
2127 290489659000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: No tag match for address: 0x101268940
2128 290489659000: board.cache_hierarchy.ruby_system.l1_controllers1: MESI_Two_Level-L1cache.sm:843: [ 0x85 0x47 0xaf 0x27 0xf5 0xc
2129 290489659000: board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: Cache hit at [0x101268970, line 0x101268940]
2130 290489659000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: No tag match for address: 0x101268940
```

A riga 2208 il core 0 esegue la load reserve

```
2206 290489676000: board.processor.switch1.core.mmu.itb: lookup(vpn=0x105f4, asid=0): hit ppn 0x10125e
2207 290489676000: board.processor.switch1.core.mmu.itb: translate(vpn=0x105f4, asid=0): 0x10125e5f4
2208 290489676000: board.processor.switch0.core.mmu.itb: lookup(vpn=0x1a954, asid=0): hit ppn 0x101268
2209 290489676000: board.processor.switch0.core.mmu.itb: translate(vpn=0x1a954, asid=0): 0x101268954 lr
2210 290489677000: board.cache_hierarchy.ruby_system.l1_controllers0.L1Dcache: No tag match for address: 0x101268940
2211 290489677000: board.cache_hierarchy.ruby_system.l1_controllers0: MESI_Two_Level-L1cache.sm:843: [ 0x85 0x47 0xaf 0x27 0xf5 0xc
2212 290489677000: board.cache_hierarchy.ruby_system.l1_controllers0.sequencer: Cache hit at [0x101268954, line 0x101268940]
2213 290489677000: board.cache_hierarchy.ruby_system.l1_controllers0.L1Dcache: No tag match for address: 0x101268940
2214 290489677000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: No tag match for address: 0x10125e5c0
2215 290489677000: board.cache_hierarchy.ruby_system.l1_controllers1: MESI_Two_Level-L1cache.sm:843: [ 0x85 0x27 0x23 0x20 0xf4 0xfe
2216 290489677000: board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: Cache hit at [0x10125e5f4, line 0x10125e5c0]
2217 290489677000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: No tag match for address: 0x10125e5c0
2218 290489678000: board.processor.switch0.core.mmu.dtb: lookup(vpn=0x8a8b0, asid=0): hit ppn 0x10107d
2219 290489678000: board.processor.switch0.core.mmu.dtb: translate(vpn=0x8a8b0, asid=0): 0x10107d8b0
2220 290489678000: board.processor.switch1.core.mmu.dtb: lookup(vpn=0x3fd4660c14, asid=0): hit ppn 0x2781c4
2221 290489678000: board.processor.switch1.core.mmu.dtb: translate(vpn=0x3fd4660c14, asid=0): 0x2781c4c14
2222 290489678000: board.cache_hierarchy.ruby_system.l2_controllers0: MESI_Two_Level-L2cache.sm:309: Addr: 0x10107d880 State: MT_SB
2223 290489678000: board.cache_hierarchy.ruby_system.l2_controllers0: MESI_Two_Level-L2cache.sm:190: machineID: L2Cache-0, requestor
```

e a riga 2233 grazie al messaggio "global: Setting Lock":

```
2232 290489679000: board.cache_hierarchy.ruby_system.l1_controllers0: MESI_Two_Level-L1cache.sm:835: [ 0x1 0x0 0x0 0x0 0x0 0x0 0x0 0
2233 290489679000: global: Setting Lock for addr: 0x10107d880 to 0
2234 290489679000: board.cache_hierarchy.ruby_system.l1_controllers0.sequencer: Cache hit at [0x10107d8b0, line 0x10107d880]
2235 290489679000: board.cache_hierarchy.ruby_system.l1_controllers0.L1Dcache: address: 0x10107d880 found
```

la load reserve scrive sul lock globale il valore univoco corrispondente al suo contesto e, a riga 2343-44, con i messaggi:

```
2341 290489704000: board.cache_hierarchy.ruby_system.l1_controllers0: MESI_Two_Level-L1cache.sm:826: 0x10107d880
2342 290489704000: board.cache_hierarchy.ruby_system.l1_controllers0: MESI_Two_Level-L1cache.sm:869: [ 0x1 0x0 0x0 0x0 0x0 0x0 0x0 0
2343 290489704000: global: Testing Lock for addr: 0x10107d880 cur 0 con 0
2344 290489704000: global: Clear Lock for addr: 0x10107d880
2345 290489704000: board.cache_hierarchy.ruby_system.l1_controllers0.sequencer: Cache miss at [0x10107d8b0, line 0x10107d880]
2346 290489704000: board.cache_hierarchy.ruby_system.l1_controllers0.L1Dcache: address: 0x10107d880 found
2347 290489704000: board.cache_hierarchy.ruby_system.l1_controllers0.L1Icache: No tag match for address: 0x10107d880
2348 290489705000: board.processor.switch0.core.mmu.itb: lookup(vpn=0x1a95c, asid=0): hit ppn 0x101268
2349 290489705000: board.processor.switch0.core.mmu.itb: translate(vpn=0x1a95c, asid=0): 0x10126895c
```

la store conditional si conclude correttamente, seguita da una miss, sempre a causa dell'invalidazione del blocco di memoria data da una modifica.

# Compare exchange

```

1#include <stdio.h>
2#include <pthread.h>
3#include <stdlib.h>
4#include <stdatomic.h>
5#include <gem5/mSops.h>
6#define N 50
7
8pthread_spinlock_t sp_lock;
9int shared_value = 0;
10int start = 0;
11int change = 0;
12
13void *func(void *par)
14{
15    int increm = *((int *)par);
16    unsigned int old_val = 0;
17    int local_start = 0;
18    int curr = 0;
19    int expected = 0;
20    int n_retry = 0;
21
22    while(!local_start)
23    {
24        pthread_spin_lock(&sp_lock);
25        if(start!=0 && start!=increm)
26            local_start=1;
27        start = increm;
28        pthread_spin_unlock(&sp_lock);
29    }
30
31    for(int i=0; i<N; i++)
32    {
33        curr = expected;
34
35        while(!atomic_compare_exchange_weak_explicit(&shared_value, &expected,
36            curr+increm, memory_order_relaxed, memory_order_relaxed))
37            curr = expected;
38    }
39    return NULL;
40}
41
42int main()
43{
44    pthread_t thread_1, thread_2;
45    int param_1 = 1;
46    int param_2 = -1;
47
48    if(pthread_spin_init(&sp_lock, PTHREAD_PROCESS_PRIVATE) != 0)
49    {
50        printf("ERROR in initializing spin lock!\n");
51        return -1;
52    }
53
54    m5_switch_cpu();
55    pthread_create(&thread_1, NULL, func, (void *)&param_1);
56    pthread_create(&thread_2, NULL, func, (void *)&param_2);
57
58    pthread_join(thread_1, NULL);
59    pthread_join(thread_2, NULL);
60    if(pthread_spin_destroy(&sp_lock) != 0)
61    {
62        printf("ERROR in initializing spin lock!\n");
63        return -1;
64    }
65    printf("Final value: %d\n", shared_value);
66    fflush(stdout);
67    m5_exit(0);
68    return 0;
69}

```

Per quanto riguarda l'`atomic_compare_exchange()`, quello che si cerca di fare è modificare il valore contenuto nella variabile condivisa dai thread, gli si passa il valore atteso che dev'essere conenuto al momento della chiamata a funzione e: in caso il valore fosse corretto si modifica la variabile condivisa con il valore desiderato, mentre in caso contrario si modifica semplicemente il valore atteso con il valore contenuto nella variabile condivisa in questo momento.

La funzione `atomic_compare_exchange()`, viene tradotta dal compilatore direttamente come una load reserve, store conditional.

A riga 2624: entrambi i thread arrivano alla prima load reserve ed entrambi hanno come next state S (poiché quella pagina contiene un'istruzione, quindi non da modificare)

2617	290489694000	board.cache_hierarchy.ruby_system.ll_controllers0.sequencer.response_ports0: Sending packet back over port	318	105ac: 9fb9	addw a5,a5,a4
2618	290489694000	board.cache_hierarchy.ruby_system.ll_controllers0.sequencer.response_ports0: Hit callback done!	319	105ae: 2781	sext.w a5,a5
2619	290489694000	board.cache_hierarchy.ruby_system.ll_controllers0: executing uo_profileInstHit	322	105b0: fcf42423	sw a5,-56(s0)
2620	290489694000	board.cache_hierarchy.ruby_system.ll_controllers0: executing po_obsrvEHit	333	105b4: fcf42783	lw a5,-56(s0)
2621	290489694000	board.cache_hierarchy.ruby_system.ll_controllers0: executing k_ppoMandatoryQueue	334	105b8: 00078590	sext.w a1,a5
2622	290489694000	board.cache_hierarchy.ruby_system.ll_controllers0: next_state: S	335	105bc: fd043603	ld a2,-48(s0)
2623	290489694000	board.cache_hierarchy.ruby_system.ll_controllers0.l1bcache: No tag match for address: 0x10125e5c0	336	105c0: fcc48793	addi a5,s0,-52
2624	290489695000	board.processor.switch0.core.mmu.l1b: [lookup(vpn=0x185c8, asid=0): hit ppn 0x10125e	337	105c4: 4500	lw a4,0(a5)
2625	290489695000	board.processor.switch0.core.mmu.l1b: [translate(vpn=0x185c8, asid=0): 0x10125e5c8	338	105c8: 880a	mv a3,a4
2626	290489695000	board.cache_hierarchy.ruby_system.ll_controllers1.sequencer.response_ports0: Timing request for address 0x10	339	105cc: 1006272f	lr.w a4,(a2)
2627	290489695000	board.cache_hierarchy.ruby_system.ll_controllers1.sequencer.response_ports0: Request ReadReq 0x10125e5c8 iss	340	105cc: 00d71563	bne a4,a3,105d6 <func+0xc2>
2628	290489695000	board.processor.switch0.core.mmu.l1b: [lookup(vpn=0x185c8, asid=0): hit ppn 0x10125e	341	105d0: 18b6252f	sc.w a0,a1,(a2)
2629	290489695000	board.processor.switch0.core.mmu.l1b: [translate(vpn=0x185c8, asid=0): 0x10125e5c8	342	105d4: f975	bnez a0,105d8 <func+0xb4>
2630	290489695000	board.cache_hierarchy.ruby_system.ll_controllers0.sequencer.response_ports0: Timing request for address 0x10	343	105d6: 40d706bb	subw a3,a4,a3
2631	290489695000	board.cache_hierarchy.ruby_system.ll_controllers0.sequencer.response_ports0: Request ReadReq 0x10125e5c8 iss	344	105da: 2681	sext.w a3,a3
2632	290489696000	board.cache_hierarchy.ruby_system.ll_controllers0.l1bcache: No tag match for address: 0x10125e5c0	345	105dc: 00160613	seqz a2,a3
2633	290489696000	board.cache_hierarchy.ruby_system.ll_controllers0: [l1bcache.Controller 0], Time: 2904896960, state: S, event:	346	105de: 0006069b	sext.w a3,a2
2634	290489696000	board.cache_hierarchy.ruby_system.ll_controllers0: executing h_fetchHit	347	105e4: c391	bnez a3,105e6 <func+0xd4>
2635	290489696000	board.cache_hierarchy.ruby_system.ll_controllers0: MESI_Two_Level-L1bcache.sm:843: [ 0x93 0x7 0xc4 0xfc 0x98	348	105e6: c398	sw a4,0(a5)
2636	290489696000	board.cache_hierarchy.ruby_system.ll_controllers0.sequencer: Cache hit at [0x10125e5c8, line 0x10125e5c0]	349	105e8: 02061793	slli a5,a2,0x20
2637	290489696000	board.cache_hierarchy.ruby_system.ll_controllers0.sequencer: read data [ 0x93 0x7 0xc4 0xfc 0x08 0x43 0xba 0	350	105ec: 9381	slli a5,a5,0x20
2638	290489696000	board.cache_hierarchy.ruby_system.ll_controllers0.sequencer: hit callback for ReadReq 0x10125e5c0	351	105ee: 0017c793	xori a5,a5,1
2639	290489696000	board.cache_hierarchy.ruby_system.ll_controllers0.sequencer.response_ports0: Hit callback needs response 1	352	105f2: 0fff7793	zext.b a5,a5
2640	290489696000	board.cache_hierarchy.ruby_system.ll_controllers0.sequencer.response_ports0: Sending packet back over port	353	105f6: ffd1	bnez a5,105f2 <func+0x7e>
2641	290489696000	board.cache_hierarchy.ruby_system.ll_controllers0.sequencer.response_ports0: Hit callback done!	354	105f8: fe442783	lw a5,-28(s0)
2642	290489696000	board.cache_hierarchy.ruby_system.ll_controllers0: executing uo_profileInstHit	355	105fc: 2783	addiw a5,a5,1
2643	290489696000	board.cache_hierarchy.ruby_system.ll_controllers0: executing po_obsrvEHit	356	105fe: fe442223	sw a5,-28(s0)
2644	290489696000	board.cache_hierarchy.ruby_system.ll_controllers0: executing k_ppoMandatoryQueue	357	10600: fe442783	lw a5,-28(s0)



A riga 2672: il core 1 è il primo ad acquisire il lock con:

- Issuing LL
- LLSC Monitor - inserting load linked - addr=0x10107d880 - cpu=1
- Cache hit at [0x10107d8b4, line 0x10107d880]
- next\_state: M

qui next state è M, nonostante basti lo stato S, poiché già si trovava in M prima della load (e non è ancora stata eseguita la write back).

```
2661 290489696000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: No tag match for address: 0x10125e5c0
2662 290489697000: board.processor.switch0.core.mmu.dtb: lookup(vpn=0x8a8b4, asid=0): hit ppn 0x10107d
2663 290489697000: board.processor.switch0.core.mmu.dtb: translate(vpn=0x8a8b4, asid=0): 0x10107d8b4
2664 290489697000: board.processor.switch0.core.isa: [cid:0]: Reserved address 10107d8b4.
2665 290489697000: board.cache_hierarchy.ruby_system.l1_controllers0.sequencer.response_ports1: Timing request for address 0x10107d8b4 on port 1
2666 290489697000: board.cache_hierarchy.ruby_system.l1_controllers0.sequencer: Issuing LL
2667 290489697000: board.cache_hierarchy.ruby_system.l1_controllers0.sequencer.response_ports1: Request LoadLockedReq 0x10107d8b4 issued
2668 290489697000: board.processor.switch1.core.mmu.dtb: lookup(vpn=0x8a8b4, asid=0): hit ppn 0x10107d
2669 290489697000: board.processor.switch1.core.mmu.dtb: translate(vpn=0x8a8b4, asid=0): 0x10107d8b4
2670 290489697000: board.processor.switch1.core.isa: [cid:1]: Reserved address 10107d8b4.
2671 290489697000: board.cache_hierarchy.ruby_system.l1_controllers1.sequencer.response_ports1: Timing request for address 0x10107d8b4 on port 1
2672 290489697000: board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: Issuing LL
2673 290489697000: board.cache_hierarchy.ruby_system.l1_controllers1.sequencer.response_ports1: Request LoadLockedReq 0x10107d8b4 issued
2674 290489698000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: address: 0x10107d880 found
2675 290489698000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: No tag match for address: 0x10107d880
2676 290489698000: board.cache_hierarchy.ruby_system.l1_controllers1: [L1Cache_Controller 1], Time: 290489698, state: M, event: Load, addr: 0x10107d880
2677 290489698000: board.cache_hierarchy.ruby_system.l1_controllers1: executing h_load_hit
2678 290489698000: board.cache_hierarchy.ruby_system.l1_controllers1: MESI_Two_Level-L1cache.sm:835: [ 0x1 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x64 0x0 0x0 0x0 0x0 0x0 0x0
2679 290489698000: global: Setting Lock for addr: 0x10107d880 to 1
2680 290489698000: board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: LLSC Monitor - inserting load linked - addr=0x10107d880 - cpu=1
2681 290489698000: board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: Cache hit at [0x10107d8b4, line 0x10107d880]
2682 290489698000: board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: read data [ 0x1 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x64 0x0 0x0 0x0 0x0 0x0 0x0 0x75 0x
2683 290489698000: board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: Hit callback for LoadLockedReq 0x10107d8b4
2684 290489698000: board.cache_hierarchy.ruby_system.l1_controllers1.sequencer.response_ports1: Hit callback needs response 1
2685 290489698000: board.cache_hierarchy.ruby_system.l1_controllers1.sequencer.response_ports1: Sending packet back over port
2686 290489698000: board.cache_hierarchy.ruby_system.l1_controllers1.sequencer.response_ports1: Hit callback done!
2687 290489698000: board.cache_hierarchy.ruby_system.l1_controllers1: executing uu_profileDataHit
2688 290489698000: board.cache_hierarchy.ruby_system.l1_controllers1: executing po_observeHit
2689 290489698000: board.cache_hierarchy.ruby_system.l1_controllers1: executing k_popMandatoryQueue
2690 290489698000: board.cache_hierarchy.ruby_system.l1_controllers1: next_state: M
2691 290489698000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: address: 0x10107d880 found
2692 290489698000: board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: No tag match for address: 0x10107d880
```

Il core 1 ottiene il lock a seguito della found (a riga 2674) per la pagina della shared variable.

Successivamente a riga 2676 c'è la risposta (come detto prima in stato M), mentre a riga 2693 anche il core 0 fa address found, ma si trova in stato I (a riga 2695).

```
2672 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: Issuing LL
2673 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer.response_ports1: Request LoadLockedReq 0x10107d8b4 issued
2674 : board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: address: 0x10107d880 found
2675 : board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: No tag match for address: 0x10107d880
2676 : board.cache_hierarchy.ruby_system.l1_controllers1: [L1Cache_Controller 1], Time: 290489698, state: M, event: Load, addr: 0x10107d880
2677 : board.cache_hierarchy.ruby_system.l1_controllers1: executing h_load_hit
2678 : board.cache_hierarchy.ruby_system.l1_controllers1: MESI_Two_Level-L1cache.sm:835: [ 0x1 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x64 0x0 0x0 0x0 0x0 0x0 0x0 0x
2679 : global: Setting Lock for addr: 0x10107d880 to 1
2680 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: LLSC Monitor - inserting load linked - addr=0x10107d880 - cpu=1
2681 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: Cache hit at [0x10107d8b4, line 0x10107d880]
2682 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: read data [ 0x1 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x64 0x0 0x0 0x0 0x0 0x0 0x0 0x0
2683 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: Hit callback for LoadLockedReq 0x10107d8b4
2684 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer.response_ports1: Hit callback needs response 1
2685 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer.response_ports1: Sending packet back over port
2686 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer.response_ports1: Hit callback done!
2687 : board.cache_hierarchy.ruby_system.l1_controllers1: executing uu_profileDataHit
2688 : board.cache_hierarchy.ruby_system.l1_controllers1: executing po_observeHit
2689 : board.cache_hierarchy.ruby_system.l1_controllers1: executing k_popMandatoryQueue
2690 : board.cache_hierarchy.ruby_system.l1_controllers1: next_state: M
2691 : board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: address: 0x10107d880 found
2692 : board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: No tag match for address: 0x10107d880
2693 : board.cache_hierarchy.ruby_system.l1_controllers0.L1Dcache: address: 0x10107d880 found
2694 : board.cache_hierarchy.ruby_system.l1_controllers0.L1Dcache: No tag match for address: 0x10107d880
2695 : board.cache_hierarchy.ruby_system.l1_controllers0: [L1Cache_Controller 0], Time: 290489698, state: I, event: Load, addr: 0x10107d880
2696 : board.cache_hierarchy.ruby_system.l1_controllers0: executing oo_allocateL1DcacheBlock
2697 : board.cache_hierarchy.ruby_system.l1_controllers0: executing i_allocateTBE
```

Da riga 2744 a riga 2749: il core 1 finisce con la store conditional e verifica che la reservation fosse ancora valida (riga 2766, 2767), concludendo con lo stato M.

```

2743 : board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: No tag match for address: 0x10125e5c0
2744 : board.processor.switch1.core.mmu.dtb: lookup(vpn=0x8a8b4, asid=0): hit ppn 0x10107d
2745 : board.processor.switch1.core.mmu.dtb: translate(vpn=0x8a8b4, asid=0): 0x10107d8b4
2746 : board.processor.switch1.core.isa: [cid:1]: load_reservation_addr empty? no.
2747 : board.processor.switch1.core.isa: [cid:1]: addr = 10107d880.
2748 : board.processor.switch1.core.isa: [cid:1]: last locked addr = 10107d880.
2749 : board.processor.switch1.core.isa: [cid:1]: SC success! Current locked addr = ffffffff0.
2750 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer.response_ports1: timing request for address 0x1010
2751 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: Issuing SC
2752 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer.response_ports1: Request StoreCondReq 0x10107d8b4
2753 : board.cache_hierarchy.ruby_system.l2_controllers0: MESI_Two_Level-L2cache.sm:373: Addr: 0x10107d880 State: MT
2754 : board.cache_hierarchy.ruby_system.l2_controllers0: [L2Cache_Controller 0], Time: 290489704, state: MT, event:
2755 : board.cache_hierarchy.ruby_system.l2_controllers0: executing b_forwardRequestToExclusive
2756 : board.cache_hierarchy.ruby_system.l2_controllers0: executing uu_profileMiss
2757 : board.cache_hierarchy.ruby_system.l2_controllers0: executing set_setMRU
2758 : board.cache_hierarchy.ruby_system.l2_controllers0: executing jj_popLIRequestQueue
2759 : board.cache_hierarchy.ruby_system.l2_controllers0: next_state: MT_IIB
2760 : board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: address: 0x10107d880 found
2761 : board.cache_hierarchy.ruby_system.l1_controllers1.L1Icache: No tag match for address: 0x10107d880
2762 : board.cache_hierarchy.ruby_system.l1_controllers1: [L1Cache_Controller 1], Time: 290489704, state: M, event: S
2763 : board.cache_hierarchy.ruby_system.l1_controllers1: executing hh_store_hit
2764 : board.cache_hierarchy.ruby_system.l1_controllers1: MESI_Two_Level-L1cache.sm:860: [ 0x1 0x0 0x0 0x0 0x0 0x0 0x
2765 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: LLSC Monitor - clearing due to store conditional
2766 : global: Testing Lock for addr: 0x10107d880 cur 1 con 1
2767 : global: Clear Lock for addr: 0x10107d880
2768 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: Cache hit at [0x10107d8b4, line 0x10107d880]
2769 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: set data [ 0x1 0x0 0x0 0x0 0x0 0x0 0x0 0x64 0
2770 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: Hit callback for StoreCondReq 0x10107d8b4
2771 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer.response_ports1: Hit callback needs response 1
2772 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer.response_ports1: Sending packet back over port
2773 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer.response_ports1: Hit callback done!
2774 : board.cache_hierarchy.ruby_system.l1_controllers1: executing uu_profileDataHit
2775 : board.cache_hierarchy.ruby_system.l1_controllers1: executing po_observeHit
2776 : board.cache_hierarchy.ruby_system.l1_controllers1: executing k_popMandatoryQueue
2777 : board.cache_hierarchy.ruby_system.l1_controllers1: next_state: M
2778 : board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: address: 0x10107d880 found
2779 : board.cache_hierarchy.ruby_system.l1_controllers1.L1Icache: No tag match for address: 0x10107d880
2780 : board.processor.switch1.core.mmu.itb: lookup(vpn=0x105d4, asid=0): hit ppn 0x10125e

```

A riga 2899: il core 0 fa miss sulla entry di cache della shared variable, quindi recupera la entry che adesso può tornare in stato S. Infatti a riga 2909 viene fatta fare la write back dal core 1.

```

2895 : board.cache_hierarchy.ruby_system.l1_controllers0: MESI_Two_Level-L1cache.sm:814: 0x10107d880
2896 : board.cache_hierarchy.ruby_system.l1_controllers0: executing hx_load_hit
2897 : board.cache_hierarchy.ruby_system.l1_controllers0: MESI_Two_Level-L1cache.sm:851: [ 0x1 0x0 0x0 0x0 0x0 0x0 0x0 0x64 0x0 0x0 0x0 0x0 0x0 0x0
2898 : global: Setting Lock for addr: 0x10107d880 to 0
2899 : board.cache_hierarchy.ruby_system.l1_controllers0.sequencer: LLSC Monitor - inserting load linked - addr=0x10107d880 - cpu=0
2900 : board.cache_hierarchy.ruby_system.l1_controllers0.sequencer: Cache miss at [0x10107d8b4, line 0x10107d880]
2901 : board.cache_hierarchy.ruby_system.l1_controllers0.sequencer: read data [ 0x1 0x0 0x0 0x0 0x0 0x0 0x0 0x64 0x0 0x0 0x0 0x0 0x0 0x0 0x75 0x7f
2902 : board.cache_hierarchy.ruby_system.l1_controllers0.sequencer: Hit callback for LoadLockedReq 0x10107d8b4
2903 : board.cache_hierarchy.ruby_system.l1_controllers0.sequencer.response_ports1: Hit callback needs response 1
2904 : board.cache_hierarchy.ruby_system.l1_controllers0.sequencer.response_ports1: Sending packet back over port
2905 : board.cache_hierarchy.ruby_system.l1_controllers0.sequencer.response_ports1: Hit callback done!
2906 : board.cache_hierarchy.ruby_system.l1_controllers0: executing s_deallocateTBE
2907 : board.cache_hierarchy.ruby_system.l1_controllers0: executing o_popIncomingResponseQueue
2908 : board.cache_hierarchy.ruby_system.l1_controllers0: executing kd_wakeUpDependents
2909 : board.cache_hierarchy.ruby_system.l1_controllers0: next_state: S
2910 : board.cache_hierarchy.ruby_system.l1_controllers0.L1Dcache: address: 0x10107d880 found
2911 : board.cache_hierarchy.ruby_system.l1_controllers0.L1Icache: No tag match for address: 0x10107d880
2912 : board.cache_hierarchy.ruby_system.l2_controllers0: [L2Cache_Controller 0], Time: 290489715, state: MT_IIB, event: WB_Data, addr: 0x10107d880
2913 : board.cache_hierarchy.ruby_system.l2_controllers0: executing m_writeDataToCache

```

Da qui in poi quando prendo la load reserve sulla shared variable lo stato sarà S e passerà in stato M solo dopo la store conditional (riga 5122).

[illegible]

```

5031 : board.cache_hierarchy.ruby_system.l1_controllers0: [L1Cache_Controller 0], Time: 290489828, state: SM, event: Ack, addr: 0x10107d880
5032 : board.cache_hierarchy.ruby_system.l1_controllers0: executing q_updateAckCount
5033 : board.cache_hierarchy.ruby_system.l1_controllers0: executing o_popIncomingResponseQueue
5034 : board.cache_hierarchy.ruby_system.l1_controllers0: next state: SM
5035 : board.cache_hierarchy.ruby_system.l1_controllers0.L1Dcache: address: 0x10107d880 found
5036 : board.cache_hierarchy.ruby_system.l1_controllers0.L1Icache: No tag match for address: 0x10107d880
5037 : board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: address: 0x10107d880 found
5038 : board.cache_hierarchy.ruby_system.l1_controllers1.L1Icache: No tag match for address: 0x10107d880
5039 : board.cache_hierarchy.ruby_system.l1_controllers1: [L1Cache_Controller 1], Time: 290489828, state: S, event: Inv, addr: 0x10107d880
5040 : board.cache_hierarchy.ruby_system.l1_controllers1: executing forward_eviction_to_cpu
5041 : board.cache_hierarchy.ruby_system.l1_controllers1: executing fi_sendInvAck
5042 : board.cache_hierarchy.ruby_system.l1_controllers1: executing l_popRequestQueue
5043 : board.cache_hierarchy.ruby_system.l1_controllers1: next state: I
5044 : board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: address: 0x10107d880 found
5045 : board.cache_hierarchy.ruby_system.l1_controllers1.L1Icache: No tag match for address: 0x10107d880
5046 : board.cache_hierarchy.ruby_system.l1_controllers1.L1Dcache: No tag match for address: 0x10125e5c0
5047 : board.cache_hierarchy.ruby_system.l1_controllers1: [L1Cache_Controller 1], Time: 290489828, state: S, event: Ifetch, addr: 0x10125e5c0
5048 : board.cache_hierarchy.ruby_system.l1_controllers1: executing h_ifetch_hit
5049 : board.cache_hierarchy.ruby_system.l1_controllers1: MESI_Two_Level-L1icache: sm:843: [ 0x93 0x7 0xc4 0xfc 0x98 0x43 0xba 0x86 0x2f 0x27 0x6 0x10 0x7d 0x88 0x0 ]
5050 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: Cache hit at [0x10125e5d4, line 0x10125e5c0]
5051 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: read data [ 0x93 0x7 0xc4 0xfc 0x98 0x43 0xba 0x86 0x2f 0x27 0x6 0x10 0x7d 0x88 0x0 ]
5052 : board.cache_hierarchy.ruby_system.l1_controllers1.sequencer: Hit callback for ReadReq 0x10125e5d4

```

veniva comunque fatta una hit. Mentre da quel punto in poi, dopo la transizione da S a M, abbiamo sempre una miss.

# Semaphores e Mutex

```

1 #include <stdio.h>
2 #include <pthread.h>
3 #include <stdlib.h>
4 #include <semaphore.h>
5 #include <gens/nops.h>
6 #define N_TIMES 100
7 #define N_INCREM 3
8
9 sem_t my_sem;
10 pthread_spinlock_t sp_lock;
11 unsigned int shared_value = 0;
12 int start = 0;
13 int change = 0;
14
15 void *func(void *par)
16 {
17     int increm = *((int *)par);
18     unsigned int old_val = 0;
19     int local_start = 0;
20
21     while(!local_start)
22     {
23         pthread_spin_lock(&sp_lock);
24         if(start!=0 && start!=increm)
25             local_start=1;
26         start = increm;
27         pthread_spin_unlock(&sp_lock);
28     }
29
30     for(int i=0; i<N_TIMES; i++)
31     {
32         sem_wait(&my_sem);
33         if(shared_value!=old_val)
34             change++;
35         for(int j=0; j<N_INCREM; j++)
36         {
37             shared_value = shared_value + increm;
38         }
39         old_val=shared_value;
40         shared_value += increm;
41     }
42     return NULL;
43 }
44
45 int main()
46 {
47     pthread_t thread_1, thread_2;
48     int param_1= 1;
49     int param_2= -1;
50
51     printf("Testing semaphore...\n");
52     if(sem_init(&my_sem, 0, 1)!=0 || pthread_spin_init(&sp_lock, PTHREAD_PROCESS_PRIVATE)!=0)
53     {
54         printf("ERROR in initializing!\n");
55         return -1;
56     }
57     m5_switch_cpu();
58     pthread_create(&thread_1, NULL, func, (void *)&param_1);
59     pthread_create(&thread_2, NULL, func, (void *)&param_2);
60
61     pthread_join(thread_1, NULL);
62     pthread_join(thread_2, NULL);
63     if(sem_destroy(&my_sem)!=0 || pthread_spin_destroy(&sp_lock)!=0)
64     {
65         printf("ERROR in cleaning!\n");
66         return -1;
67     }
68     printf("Number of changes in thread lock owning: %d\n", change);
69     printf("Final value: %d\n", shared_value);
70     fflush(stdout);
71     m5_exit(0);
72     return 0;
73 }

```

```

1 #include <stdio.h>
2 #include <pthread.h>
3 #include <stdlib.h>
4 #include <gens/nops.h>
5 #define N_TIMES 10
6 #define N_INCREM 3
7
8 pthread_mutex_t my_mutex;
9 pthread_spinlock_t sp_lock;
10 unsigned int shared_value = 0;
11 int start = 0;
12 int change = 0;
13
14 void *func(void *par)
15 {
16     int increm = *((int *)par);
17     unsigned int old_val = 0;
18     int local_start = 0;
19
20     while(!local_start)
21     {
22         pthread_spin_lock(&sp_lock);
23         if(start!=0 && start!=increm)
24             local_start=1;
25         start = increm;
26         pthread_spin_unlock(&sp_lock);
27     }
28
29     //printf("Entro %d\n", increm);
30
31     for(int i=0; i<N_TIMES; i++)
32     {
33         if(pthread_mutex_lock(&my_mutex)!=0)
34             return (void *) -1;
35         if(shared_value!=old_val)
36             change++;
37         for(int j=0; j<N_INCREM; j++)
38         {
39             shared_value = shared_value + increm;
40         }
41         old_val=shared_value;
42         shared_value += increm;
43     }
44     if(pthread_mutex_unlock(&my_mutex)!=0)
45         return (void *) -1;
46
47     // printf("Esco %d\n", increm);
48     return 0;
49 }
50
51 int main()
52 {
53     pthread_t thread_1, thread_2;
54     int param_1= 1;
55     int param_2= -1;
56     void *ret;
57
58     printf("Testing mutex...\n");
59     if(pthread_mutex_init(&my_mutex, NULL)!=0 || pthread_spin_init(&sp_lock, PTHREAD_PROCESS_PRIVATE)!=0)
60     {
61         printf("ERROR in initializing!\n");
62         return -1;
63     }
64     m5_switch_cpu();
65     pthread_create(&thread_1, NULL, func, (void *)&param_1);
66     pthread_create(&thread_2, NULL, func, (void *)&param_2);
67
68     pthread_join(thread_1, &ret);
69     if(!ret)
70         printf("IL THREAD 1 HA FALLITO!\n");
71     pthread_join(thread_2, &ret);
72     if(!ret)
73         printf("IL THREAD 2 HA FALLITO!\n");
74     if(pthread_mutex_destroy(&my_mutex)!=0 || pthread_spin_destroy(&sp_lock)!=0)
75     {
76         printf("ERROR in cleaning!\n");
77         return -1;
78     }
79     printf("Number of changes in thread lock owning: %d\n", change);
80     printf("Final value: %d\n", shared_value);
81     fflush(stdout);
82     m5_exit(0);
83     return 0;
84 }

```

Dopo aver realizzato un implementazione per la sincronizzazione fra thread con i semafori e i mutex, ci siamo resi conto che venivano tradotti dal compilatore di nuovo come delle load reserve store conditional. Quindi il loro funzionamento era analogo alle implementazioni, più semplici e facilmente analizzabili dal simulatore, svolte precedentemente.

Ad esempio per il semaforo:

```

15010 000000000001ab16 <__new_sem_wait>:
15017 1ab16: 1141      addi    sp,sp,-16
15018 1ab18: e022      sd      s0,0(sp)
15019 1ab1a: e406      sd      ra,8(sp)
15020 1ab1c: 842a      mv      s0,a0
15021 1ab1e: 2e0200ef jal     3adfe <__pthread_testcancel>
15022 1ab22: 601c      ld      a5,0(s0)
15023 1ab24: 02079713 slli    a4,a5,0x20
15024 1ab28: 9301      srli    a4,a4,0x20
15025 1ab2a: c30d      beqz    a4,1ab4c <__new_sem_wait+0x36>
15026 1ab2c: fff78693 addi    a3,a5,-1
15027 1ab30: 1004372f lr.d    a4,(s0)
15028 1ab34: 00f71563 bne     a4,a5,1ab3e <__new_sem_wait+0x28>
15029 1ab38: 1cd4362f sc.d.aq a2,a3,(s0)
15030 1ab3c: fa75      bnez    a2,1ab30 <__new_sem_wait+0x1a>
15031 1ab3e: 00f71763 bne     a4,a5,1ab4c <__new_sem_wait+0x36>
15032 1ab42: 60a2      ld      ra,8(sp)
15033 1ab44: 6402      ld      s0,0(sp)
15034 1ab46: 4501      li      a0,0
15035 1ab48: 0141      addi    sp,sp,16
15036 1ab4a: 8082      ret
15037 1ab4c: 8522      mv      a0,s0
15038 1ab4e: 6402      ld      s0,0(sp)
15039 1ab50: 60a2      ld      ra,8(sp)
15040 1ab52: 0141      addi    sp,sp,16
15041 1ab54: bf39      j       1aa72 <__new_sem_wait_slow64.constprop.0>
15042

```

# Atomic operations

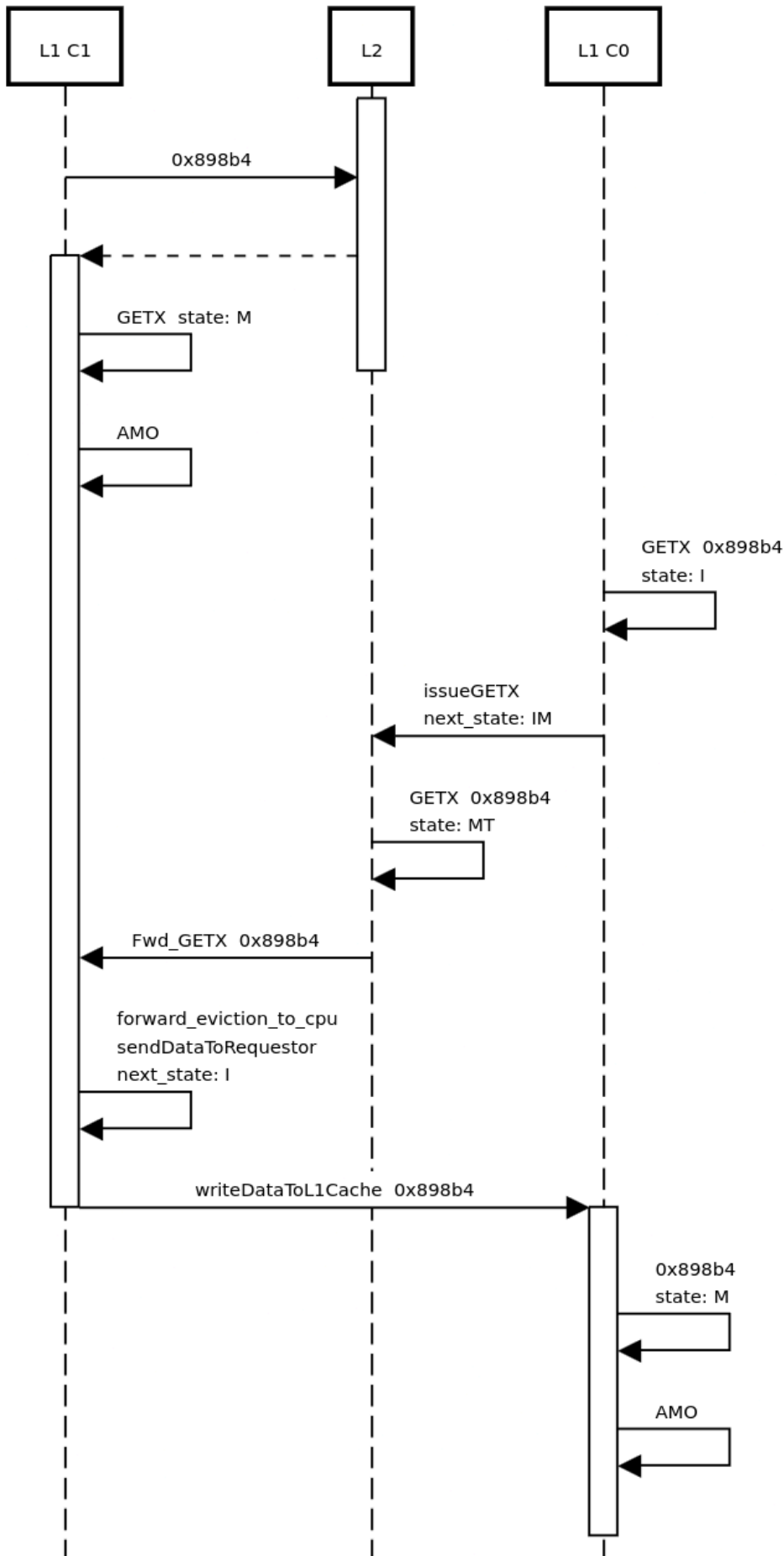
```
1#include <stdio.h>
2#include <stdlib.h>
3#include <stdatomic.h>
4#include <gem5/m5ops.h>
5#include <pthread.h>
6#define N_TIMES 10
7
8pthread_spinlock_t sp_lock;
9atomic_int my_index = ATOMIC_VAR_INIT(0);
10//int my_index=0;
11int start = 0;
12
13void *func(void *par)
14{
15    int increm = *((int *)par);
16    int n_times = (((int *)par)+1);
17    int local_start = 0;
18
19    while(!local_start)
20    {
21        pthread_spin_lock(&sp_lock);
22        if(start!=0 && start!=increm)
23            local_start=1;
24        start = increm;
25        pthread_spin_unlock(&sp_lock);
26    }
27
28    for(int i=0; i<n_times; i++)
29    {
30        atomic_fetch_add(&my_index, increm);
31        atomic_fetch_add(&my_index, increm);
32        atomic_fetch_add(&my_index, increm);
33        atomic_fetch_add(&my_index, increm);
34        atomic_fetch_add(&my_index, increm);
35
36        atomic_fetch_add(&my_index, increm);
37        atomic_fetch_add(&my_index, increm);
38        atomic_fetch_add(&my_index, increm);
39        atomic_fetch_add(&my_index, increm);
40        //my_index = my_index+increm;
41    }
42    return NULL;
43}
44
45int main()
46{
47    pthread_t thread_1, thread_2;
48    int param_1[]={1, N_TIMES};
49    int param_2[]={-1, N_TIMES};
50
51    if(pthread_spin_init(&sp_lock, PTHREAD_PROCESS_PRIVATE) != 0)
52    {
53        printf("ERROR in initializing spin lock!\n");
54        return -1;
55    }
56    m5_switch_cpu();
57    pthread_create(&thread_1, NULL, func, (void *)&param_1);
58    pthread_create(&thread_2, NULL, func, (void *)&param_2);
59    pthread_join(thread_1, NULL);
60    pthread_join(thread_2, NULL);
61    if(pthread_spin_destroy(&sp_lock) != 0)
62    {
63        printf("ERROR in initializing spin lock!\n");
64        return -1;
65    }
66    printf("%d\n", my_index);
67    return 0;
68}
```

Per le operazioni atomiche abbiamo creato 2 threads che eseguono in un ciclo 10 incrementi ad una variabile condivisa, in modo che la zona critica fosse più lunga e permettesse ai 2 thread di sovrapporsi.

Dopo aver analizzato la simulazione di gem5, ci siamo resi conto che il loro funzionamento era strettamente legato al modello di memoria delle cache, basandosi sul fatto che un entry nello stato M invalidasse quelle di tutti gli altri e la tenesse nello stato M fino al completamento dell'operazione atomica, seguendo una logica riportata nel sequence diagram seguente.



## Atomic operations



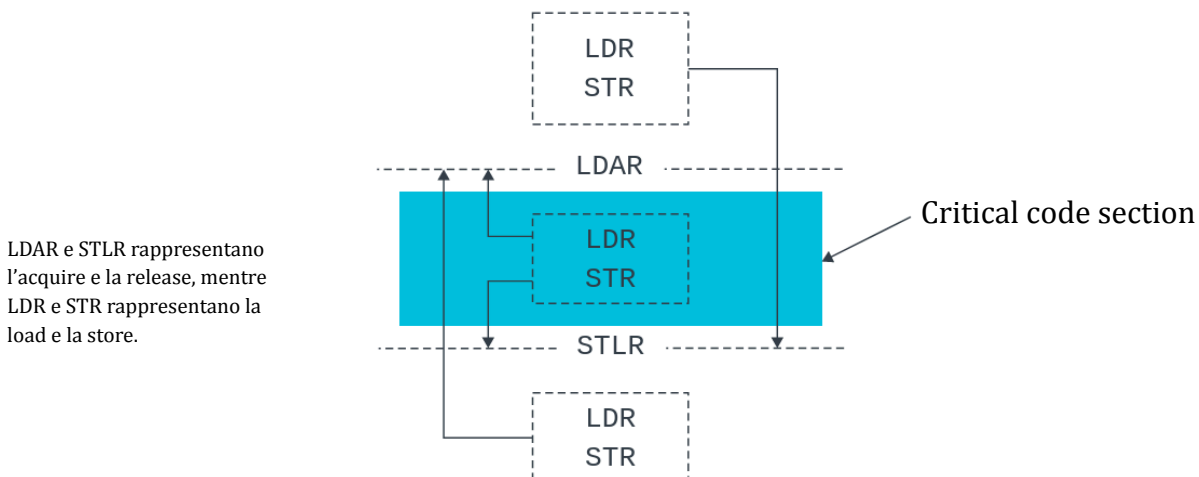
# Fences

```

1 #include <stdio.h>
2 #include <atomic>
3 #include <pthread.h>
4 #include <gen5/m5ops.h>
5 #define DIM 500
6
7 using namespace std;
8 pthread_spinlock_t sp_lock;
9 int start = 0;
10 long long prec_elements[DIM] = {0};
11 //atomic<long long> My_values[DIM] = {0};
12 long long my_values[DIM] = {0};
13 unsigned int i = 0;
14
15 void do_stuff()
16 {
17     int a=1;
18     int b=2;
19     for(int j=0; j<100; j++)
20     {
21         b = (a-b)*(a+b);
22     }
23 }
24
25 void *producer(void *par)
26 {
27     int local_start = -1;
28     while(local_start != 1)
29     {
30         pthread_spin_lock(&sp_lock);
31         if(local_start==1)
32         {
33             local_start=0;
34             ++start;
35         }
36         if(start>=2)
37             if(start>=2)
38                 local_start=1;
39                 pthread_spin_unlock(&sp_lock);
40             }
41             printf("Inizio\n");
42             for(i=0; i<DIM; i++)
43             {
44                 do_stuff();
45                 prec_elements[i] = i/1.321;
46                 //My_values[i].store(i/1.321, memory_order_release);
47                 my_values[i] = i/1.321;
48             }
49             return NULL;
50 }
51
52 void *consumer(void *par)
53 {
54     unsigned int counter=0;
55     long long prec = -1;
56     long long val = -1;
57     unsigned int index = 0;
58     int local_start = -1;
59     while(local_start != 1)
60     {
61         pthread_spin_lock(&sp_lock);
62         if(local_start==1)
63         {
64             local_start=0;
65             ++start;
66         }
67         if(start>=2)
68             local_start=1;
69             pthread_spin_unlock(&sp_lock);
70         }
71         index = i;
72         while(index < DIM)
73         {
74             val = my_values[index];
75             //val = My_values[index].load(memory_order_acquire);
76             prec = prec_elements[index];
77             if(val!=0 && prec==0)
78             {
79                 ++counter;
80                 printf("prec_element | new_value: %lld | %lld\n", prec, val);
81             }
82             index = i;
83         }
84         printf("Number of memory order inconsistency: %lld\n", counter);
85         return NULL;
86     }
87 }
88
89 int main()
90 {
91     pthread_t thread_1, thread_2;
92     if(pthread_spin_init(&sp_lock, PTHREAD_PROCESS_PRIVATE) != 0)
93     {
94         printf("ERROR in initializing spin lock!\n");
95         return -1;
96     }
97
98     m5_switch_cpu();
99     pthread_create(&thread_1, NULL, producer, NULL);
100    pthread_create(&thread_2, NULL, consumer, NULL);
101    pthread_join(thread_1, NULL);
102    pthread_join(thread_2, NULL);
103    fflush(stdout);
104    if(pthread_spin_destroy(&sp_lock) != 0)
105    {
106        printf("ERROR in initializing spin lock!\n");
107        return -1;
108    }
109    }
110    //
111    return 0;
112 }

```

Per testare le fences abbiamo sfruttato la logica di acquire release così che si potessero testare anche delle fence più deboli oltre alla classica istruzione “fence” in grado di impedire il reordering in entrambi i versi.



In assembly la store acquire e load release vengono tradotte tramite in fence e decoratori di istruzione (.aq e .rl):

- acquire viene tradotto come una fence seguita da un'operazione atomica marchiata come acquire, la quale impedisce qualsiasi operazione successiva di venire riordinata e completata prima dell'operazione atomica.

In questo modo si ottiene un ordine completo, al pari di un'operazione circondata da 2 fences

```
10696:    0f50000f          fence   iorw,ow
1069a:    0ce7b02f          amoswap.d,aq    zero,a4,(a5)
```

- release viene tradotta come una semplice load preceduta e seguita da 2 fences, così da forzare un ordine completo per quest'operazione

```
10794:    0ff0000f          fence
10798:    639c             ld      a5,0(a5)
1079a:    0ff0000f          fence
```

Dopo aver eseguito vari test si può notare come senza e fences create dalla logica di acquire-release il thread consumer è rileva numerose inconsistenze di memoria

```
==== m5 terminal: Terminal 0 ====
Inizio
prec_element | new_value: 0 | 182
prec_element | new_value: 0 | 191
prec_element | new_value: 0 | 196
prec_element | new_value: 0 | 200
prec_element | new_value: 0 | 204
prec_element | new_value: 0 | 208
prec_element | new_value: 0 | 215
prec_element | new_value: 0 | 219
prec_element | new_value: 0 | 227
prec_element | new_value: 0 | 230
prec_element | new_value: 0 | 239
prec_element | new_value: 0 | 246
prec_element | new_value: 0 | 252
prec_element | new_value: 0 | 254
prec_element | new_value: 0 | 256
prec_element | new_value: 0 | 258
prec_element | new_value: 0 | 288
prec_element | new_value: 0 | 290
prec_element | new_value: 0 | 300
prec_element | new_value: 0 | 302
prec_element | new_value: 0 | 305
prec_element | new_value: 0 | 307
prec_element | new_value: 0 | 312
prec_element | new_value: 0 | 314
prec_element | new_value: 0 | 318
prec_element | new_value: 0 | 367
prec_element | new_value: 0 | 369
Number of memory order inconsistency: 27
```

# Implementazioni

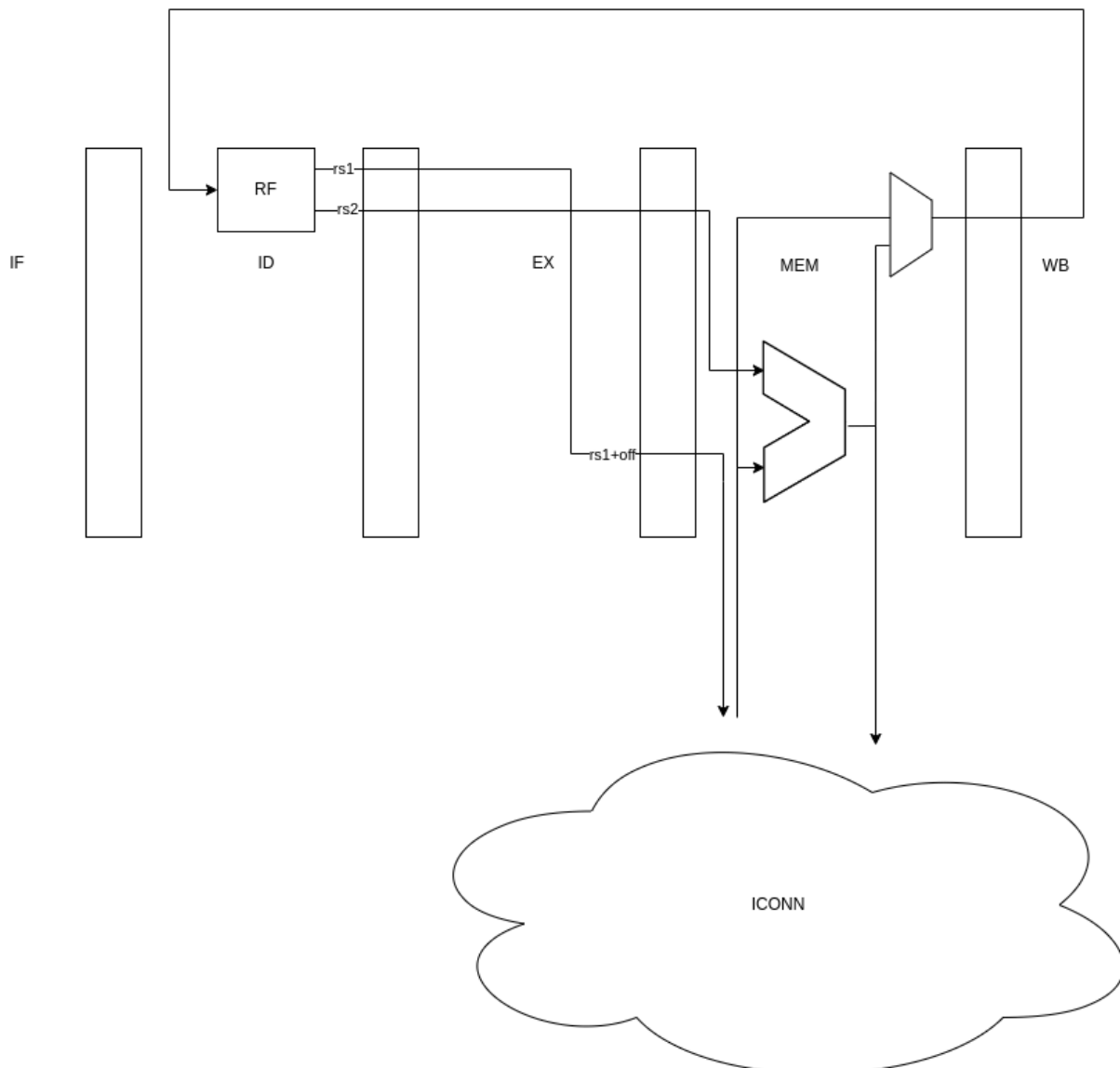
## Atomic

La difficoltà nell'implementare istruzioni atomiche in una CPU sta nel riuscire a eseguire multiple operazioni senza stravolgere la pipeline delle istruzioni ordinarie.

Una soluzione è inserire una piccola ALU, capace di eseguire solamente un subset di istruzioni, che rappresenta il numero di operazioni atomiche concesse da RISC-V, ovvero: SWAP, ADD, AND, OR, XOR, MAX e MIN.

Per garantire coerenza all'interno del processore, la pipeline modificata sarà in grado di eseguire nello stesso ciclo di pipeline un accesso a memoria per recuperare il dato che verrà successivamente elaborato dalla ALU e salvato in memoria tramite una store.

Come si può notare dall'esempio riportato, la soluzione riesce a calcolare  $rs1 + \text{offset}$  nello stadio di EX, accedere alla memoria tramite l'interconnect, nello stadio di MEM, e utilizzare il risultato come input della nuova ALU, per poi andare a salvare finalmente il risultato in memoria ancora tramite l'interconnect.



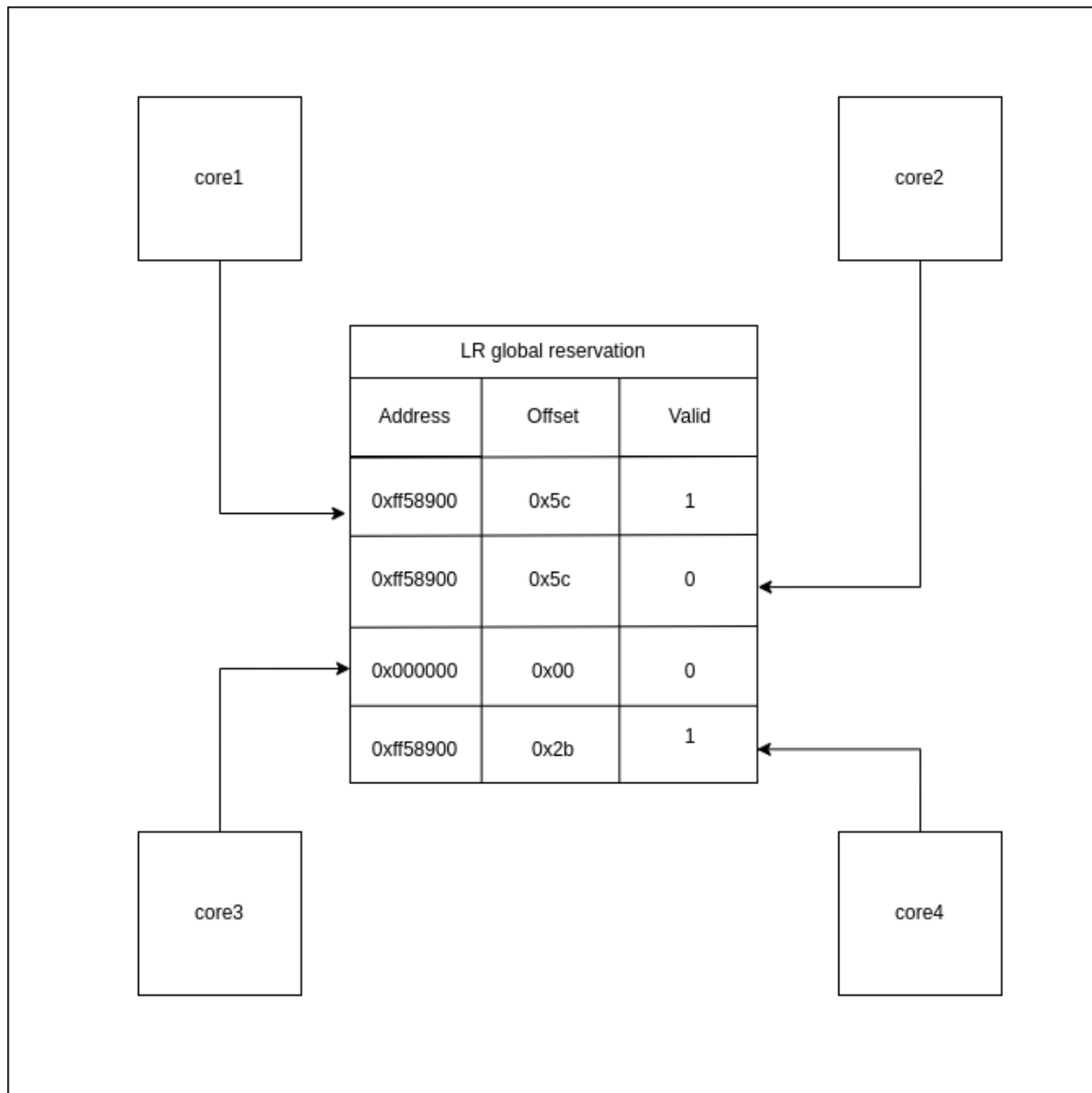
## LL SC

Per poter supportare una architettura che faccia uso di istruzioni Load-link/store-conditional, abbiamo pensato ad una soluzione che prevede una nuova struttura di memoria, condivisa tra i vari core del processore.

Questa struttura prevede una sola entry per ogni core, e ognuna contiene l'indirizzo di una pagina di memoria, un offset che si riferisce all'indirizzo su cui la LL ha preso una reservation, e un bit di validità per indicare se il core in questione ha ancora la reservation su quella regione di memoria.

Ogni volta che un core esegue una LL, questo andrà ad aggiornare la propria entry nella tabella, e provvede ad invalidare tutte le righe degli altri core che avevano privilegi sullo stesso indirizzo.

A questo punto, prima di eseguire la store conditional, il core dovrà controllare di avere il bit di validità settato ad 1, altrimenti la store conditional fallirà e la branch che la segue riporterà l'esecuzione del codice alla load link.





## Fences

Per implementare le fence instructions abbiamo pensato di aggiungere un flip-flop il quale viene settato a 1 non appena viene fatta la fetch di un'istruzione di tipo fence, e non rimane settato a 1 finché la pipeline e tutte il buffer contenente tutte le operazioni di memoria, non siano vuoti.

In questo modo riusciamo ad assicurare che tutte le istruzioni antecedenti alla fence vengano completate prima dell'esecuzione di una qualsiasi istruzione successiva, impedendo così qualsiasi tipo di reordering a livello di memoria e di istruzioni.

```
if(!(fence_instr ^ fetcher_out))
begin
    fence_n =1;
end
else
begin
    fence_n = fence_r && !(if_id_free && id_ex_free && ex_mem_free &&
        mem_wb_free && mem_op_buf_free);
end
```

