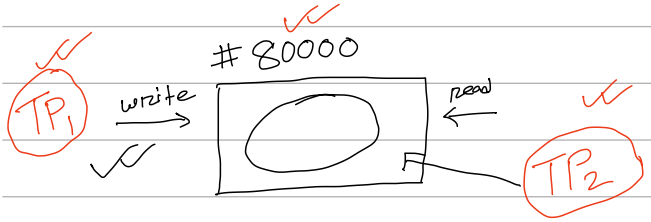
Synchronization

T_1 T_2



Data race

data race

Two memory accesses form a data race if they are from different threads to the same location, at least one is a write, and they occur one after another.

lr.d = load reserved double-word

Syntax : lr.d rd, (rs1)

load data from location stored in rs1 reg. → these instructions are used in sequence.
to rd register.

places a reservation on the memory add. specified by (rs1)

this reservation is held until a store-conditional instruction successfully writes to the same memory location. [(rs1)]

sc.d = store conditional double-word

Syntax : sc.d rd, (rs1), rs2

rs2	rs1	#80008
50	#80008	50

storing data from rs2 to address specified by (rs1)

success in storing ; saves 0 in rd.

fails " " ; " a non-zero value in rd.

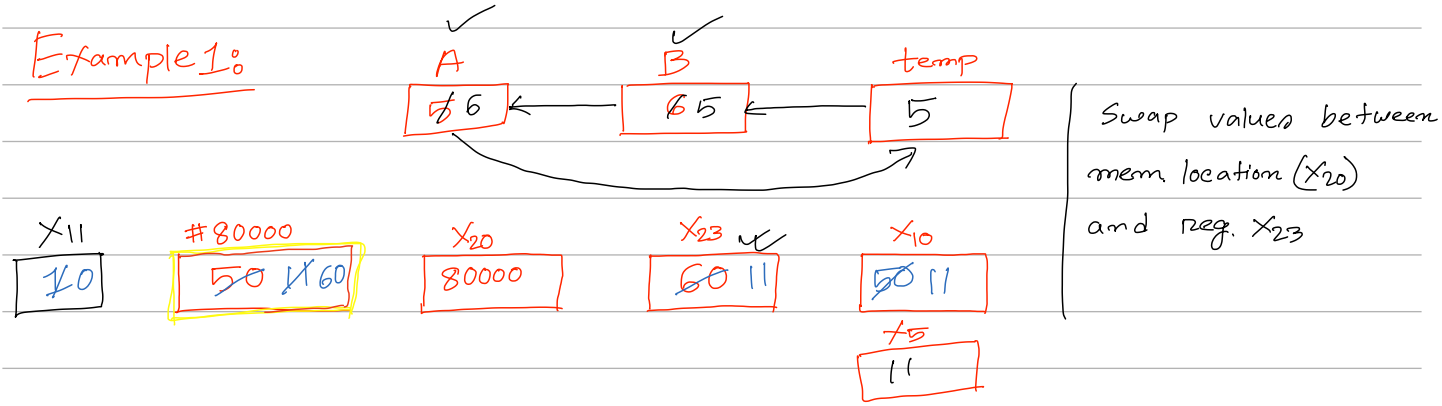
If the contents of the memory location specified by the load-reserved are changed before the store-conditional to the same address occurs, then the store conditional fails and does not write the value to memory.

Analogy

=> You are trying to load information stored in location #80000. At the same time some other process might be trying to change what's in that #80000 location.

=> You are trying to read what's in there but before you get the chance to write something new, someone else comes along and changes what's in that spot. Now, your attempt to write the new thing won't work.

Example 1:

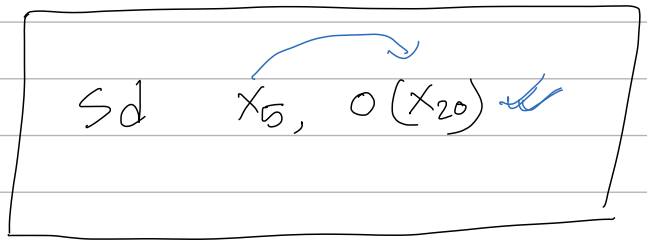


- Example 1: atomic swap (to test/set lock variable)

```
again:  lr.d x10,(x20)
        sc.d x11,(x20),x23 // X11 = status
        bne  x11,x0,again- // branch if store failed
        addi x23,x10,0     // X23 = loaded value
```

P1:

P2:



Example-2

```
addi x12, x0, 1   // copy locked value
again: lr.d x10, (x20)   // load-reserved to read lock
   bnex10, x0, again   // check if it is 0 yet
   sc.d x11, x12, (x20)   // attempt to store new value
   bnex11, x0, again   // branch if store fails
```

Unlock:

```
sd x0, 0(x20)   // free lock by writing 0
```