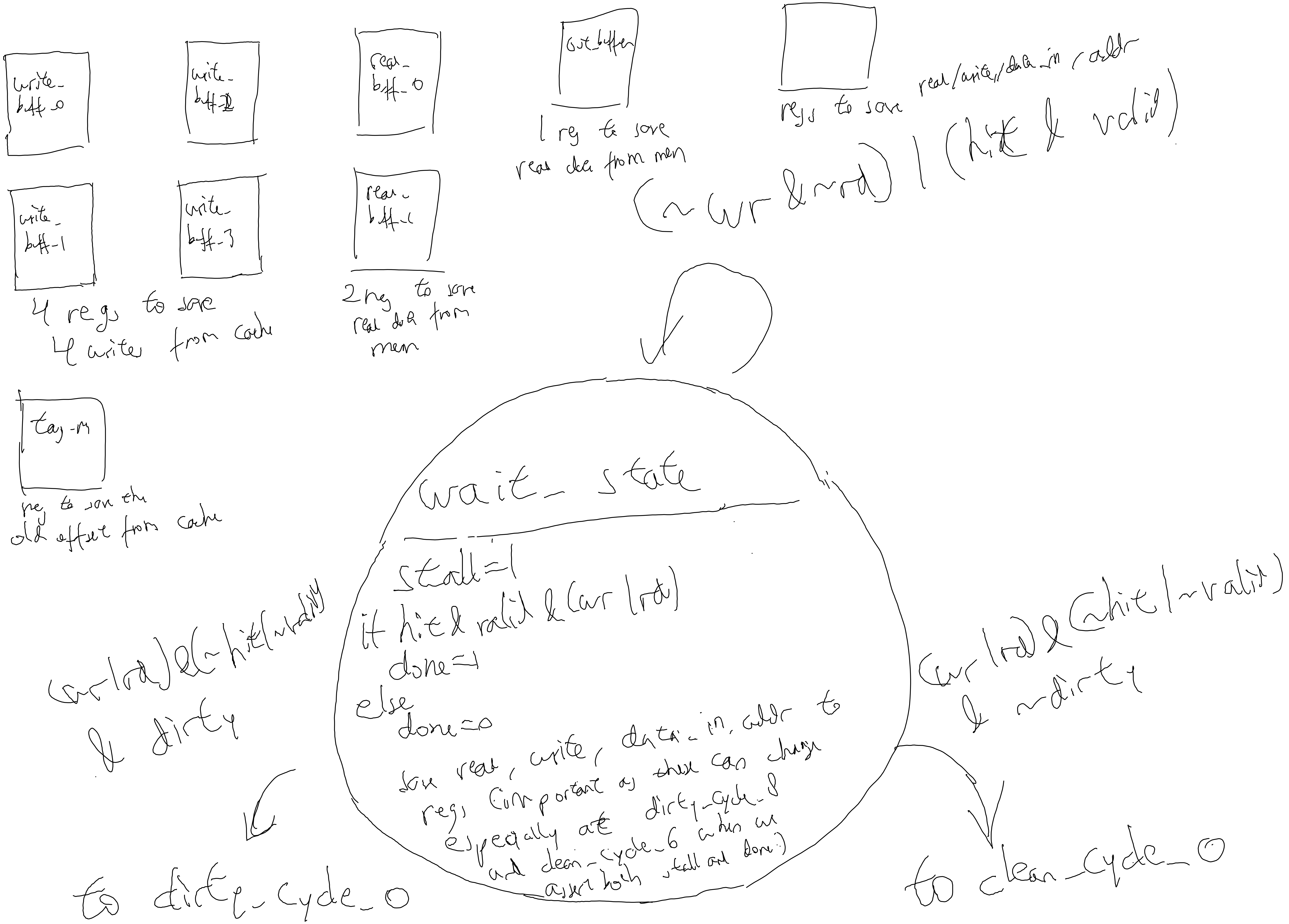
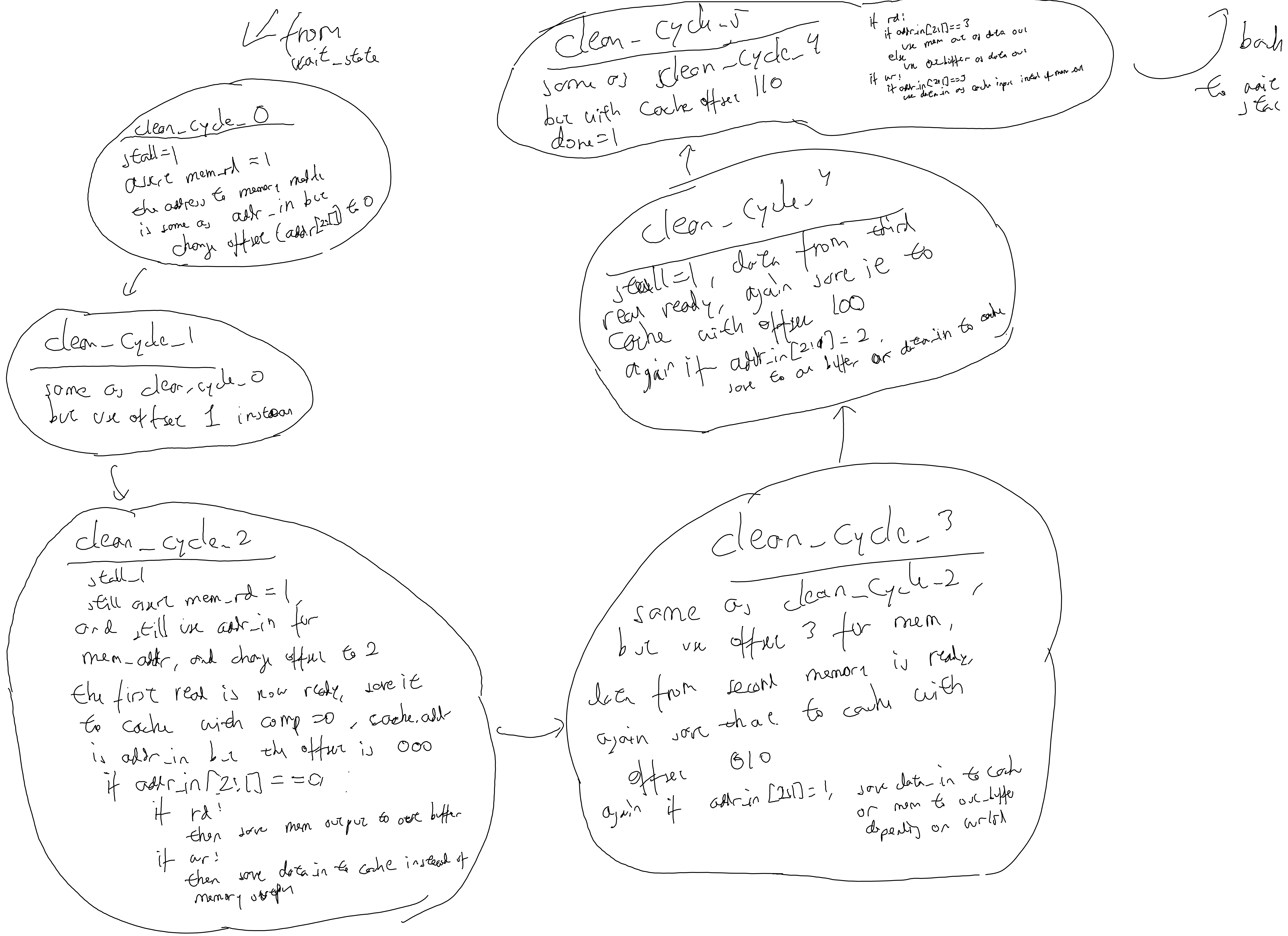
didn't change fsm for 2 way associative.

enable both cache for wait_state.
if one hit & valid, use the data_out / data_in for that.
if not, we go to dirty_cycle_0 or clean_cycle_0, and we only enable 1 cache until we get back to wait_state. We pick the cache according to the specs in the cache design page


write_buff_0
write_buff_2
read_buff_0
out_buffer
write_buff_1
write_buff_3
read_buff_1
4 regs to save 4 writes from cache
2 reg to save read data from mem
1 reg to save read data from mem
regs to save read/write/data_in/addr
tag_m
reg to save the old offset from cache
(~wr &~rd) | (hit & valid)
wait_state
stall=1
if hit & valid & (wr | rd)
done=1
else
done=0
save read, write, data_in, addr to regs (important as these can change especially at dirty_cycle_8 and clean_cycle_6 when we assert both stall and done)
(wr|rd)&(~hit|~valid) & dirty
to dirty_cycle_0
(wr|rd)&(~hit|~valid) & ~dirty
to clean_cycle_0



from wait_state

clean_cycle_0

stall=1
assert mem_rd = 1
the address to memory module
is same as addr_in but
change offset (addr[2:1]) to 0

clean_cycle_1

same as clean_cycle_0
but use offset 1 instead

clean_cycle_2

stall_1
still assert mem_rd = 1,
and still use addr_in for
mem_addr, and change offset to 2
the first read is now ready, save it
to cache with comp =0, cache.addr
is addr_in but the offset is 000

```
if addr_in[2:1] == 0 :
    if rd!
        then save mem output to out buffer
    if wr!
        then save data in to cache instead of
        memory output
```

clean_cycle_3

same as clean_cycle_2,
but use offset 3 for mem,
data from second memory is ready,
again save ~~that~~ to cache with
offset 010
again if addr_in [2:1] = 1, save data_in to cache
or mem to out_buffer
depending on wr/rd

clean_cycle_4

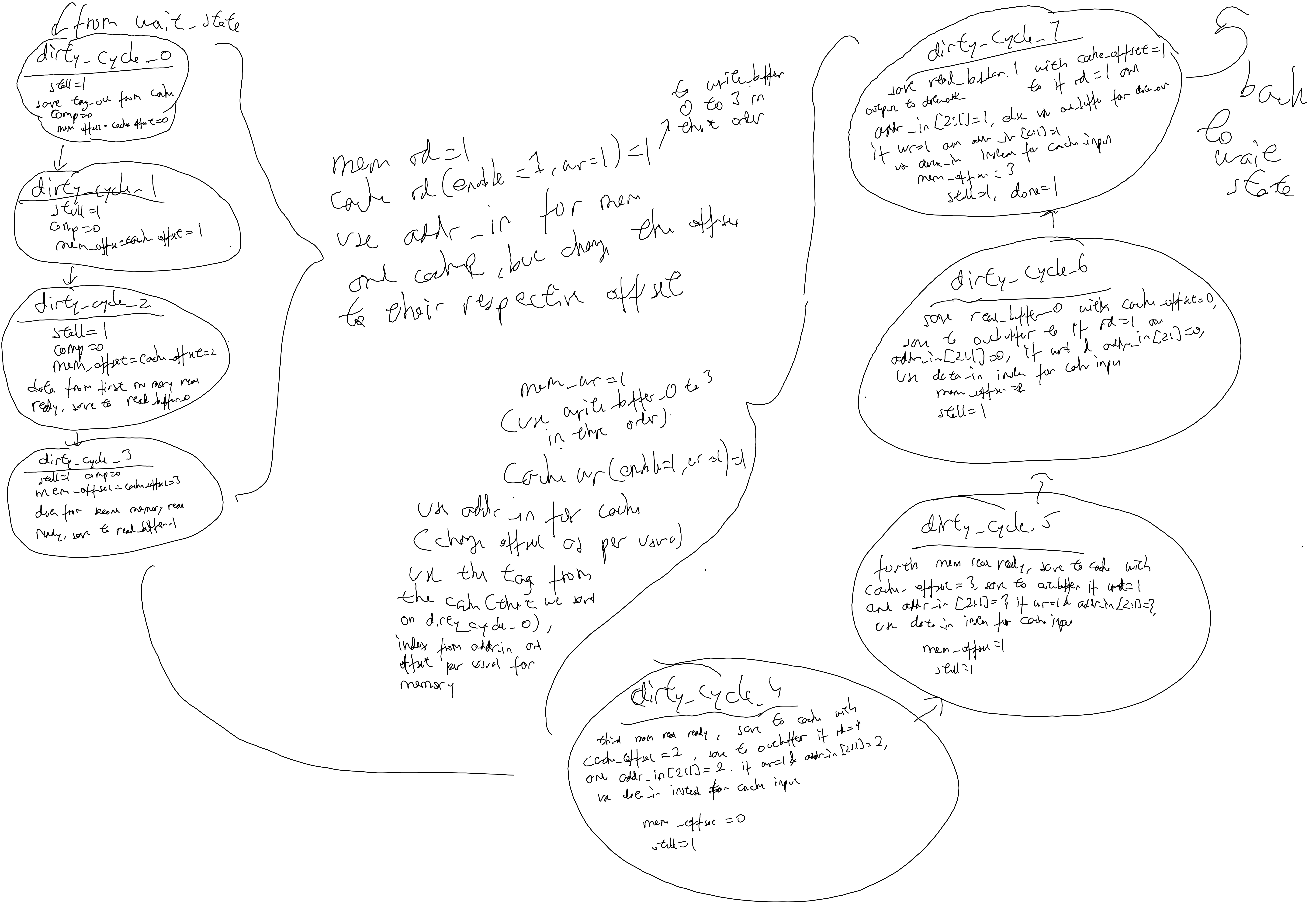
stall=1, data from ~~third~~
read ready, again save it to
cache with offset 100
again if addr_in[2:1] = 2, save to out buffer or data_in to cache

clean_cycle_5

same as clean_cycle_4
but with cache offset 110
done=1

```
if rd:
    if addr_in[2:1]==3
        use mem out as data out
    else
        use outbuffer as data out
if wr!
    if addr_in[2:1]==3
        use data_in as cache input instead of mem out
```

back
to wait
state


from wait_state
dirty_cycle_0
stall=1
save tag_out from cache
comp=0
mem offset = cache offset=0
dirty_cycle_1
stall=1
comp=0
mem_offset=cache offset=1
dirty_cycle_2
stall=1
comp=0
mem_offset=cache_offset=2
data from first memory read ready, save to read_buffer_0
dirty_cycle_3
stall=1 comp=0
mem_offset=cache offset=3
data from second memory read ready, save to read_buffer_1
mem rd=1
cache rd(enable =1, wr=1)=1
use addr_in for mem and cache, but change the offset to their respective offset
to write_buffer 0 to 3 in that order
mem_wr=1
(use write_buffer_0 to 3 in that order).
Cache wr(enable=1, wr=1)=1
use addr_in for cache (change offset as per usual)
use the tag from the cache (that we saved on dirty_cycle_0), index from addr_in and offset per usual for memory
dirty_cycle_4
third mem read ready, save to cache with cache_offset=2, save to outbuffer if rd=1 and addr_in[2:1]=2. if wr=1 & addr_in[2:1]=2, use data_in instead for cache input
mem_offset=0
stall=1
dirty_cycle_5
forth mem read ready, save to cache with cache_offset=3, save to outbuffer if wr=1 and addr_in[2:1]=3 if wr=1 & addr_in[2:1]=3, use data_in instead for cache input
mem_offset=1
stall=1
dirty_cycle_6
save read_buffer_0 with cache_offset=0, save to outbuffer to if rd=1 and addr_in[2:1]=0, if wr=1 & addr_in[2:1]=0, use data_in instead for cache input
mem_offset=2
stall=1
dirty_cycle_7
save read_buffer_1 with cache_offset=1 to if rd=1 and addr_in[2:1]=1, else use outbuffer for data_out output to data_out
if wr=1 and addr_in[2:1]=1 use data_in instead for cache input
mem_offset=3
stall=1, done=1
back to wait state