The atomic fetch-and-set x, y instruction unconditionally sets the memory location x to 1 and fetches the old value of x in y without allowing any intervening access to the memory location x. Consider the following implementation of P and V functions on a binary semaphore S.

```
void P (binary_semaphore *S)
{
unsigned y;
unsigned *x = & (S -> value);                    // Stores the value of s in x
do
{
fetch-and-set x, y;
} while (y);
}

void V (binary_semaphore *S) {
s - > value = 0;
}
```

Which one of the following is true?

A. The implementation may not work, if context switching is disabled in P

B. Instead of using fetch-and-set, a pair of normal load/ store can be used

C. The implementation of V is wrong

D. The code does not implement a binary semaphore

➔ **The implementation may not work, if context switching is disabled in P**

**Explanation :-**

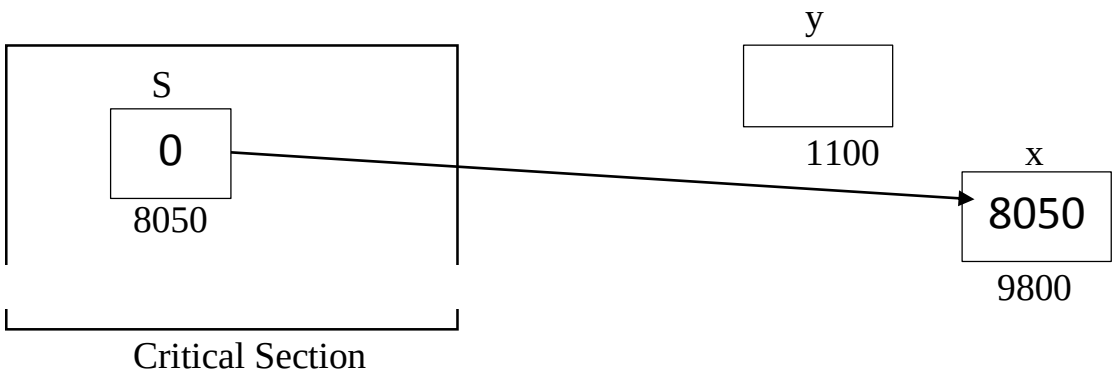
```
void P (binary_semaphore *S)
{
unsigned y;
unsigned *x = & (S -> value);                    // Stores the value of s in x
do
{
fetch-and-set x, y;                              //  fetches the old value of x, stores it in y
} while (y);
}

void V (binary_semaphore *S)
{
      s - > value = 0;                           // Sets the value of s as 0
}
```

In the above implementation, binary_semaphore S is used which may have value 0 or 1.

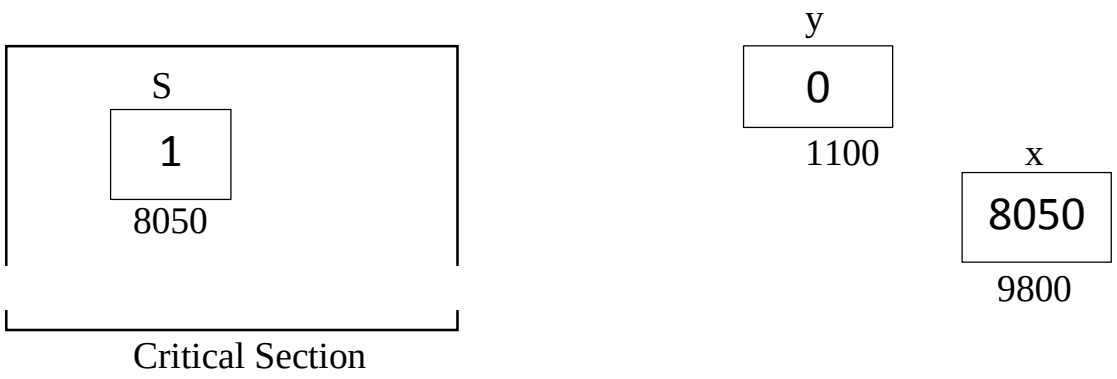
Consider the two cases -

**Case 1 : Where S=0**



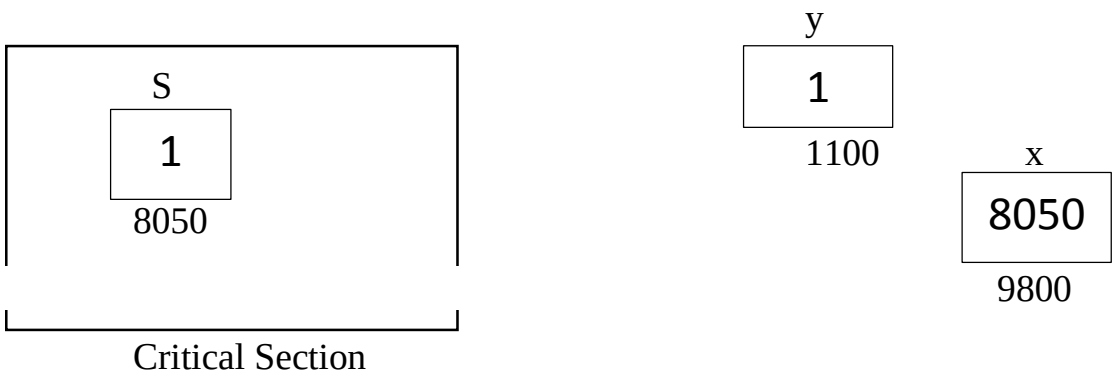
Then the atomic fetch-and-set x, y instruction unconditionally sets the memory location x to 1 and fetches the old value of x in y without allowing any intervening access to the memory location x.

It means value of x i.e. S=1 will be set and previous ( or old ) value of x will be store in y i.e. y=0.



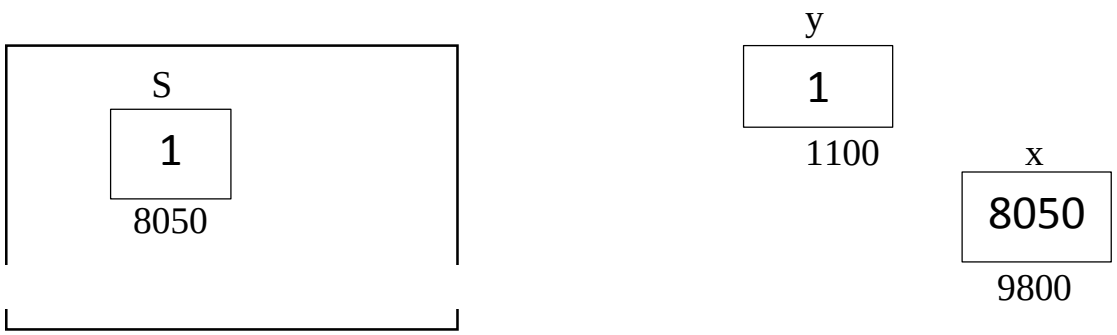

And when we execute while loop, it gets the value of y=0 and then it come out of while loop. After that we can execute V( ) and make S=0 and can enter in C.S. ( critical section). Hence, code is properly implementing.

**Case 2 : Where S=1**




Then the atomic fetch-and-set x, y instruction unconditionally sets the memory location x to 1 and fetches the old value of x in y without allowing any intervening access to the memory location x.

It means value of x i.e. S=1 will be set and previous ( or old ) value of x will be store in y i.e. y=1




And when we execute while loop, it gets the value of y=1 and then it continues the while loop. After that we can not execute V( ) and can not enter in C.S. ( critical section). Hence, the code will not work properly if context switching is disabled in P.

**Option A is correct** - It means, here, context switching is happened properly, if it execute V ( ).

If some other process doesn't execute V( ), then the while loop of a process will continue forever. Or we can say, if the context switching is disabled in P, the while loop will run forever as no other process will be able to execute V( ).

It means in that case, context switching is not working and implementation may not work properly because P( ) never give control to the other process and while loop perform repeatedly. And hence context switching is must.

**In option B -** If we use normal load & Store instead of Fetch & Set there is good chance that more than one Process sees S value as 0 & then mutual exclusion wont be satisfied. So this option is wrong.

**In option C -** Here we are setting S→ value to 0, which is correct. (As in fetch & Set we wait if value of S→ value is 1. So implementation is correct. This option is wrong.

**In option D** – Here, only one process can be in critical section here at a time. So this is binary semaphore & Option (D) is wrong.