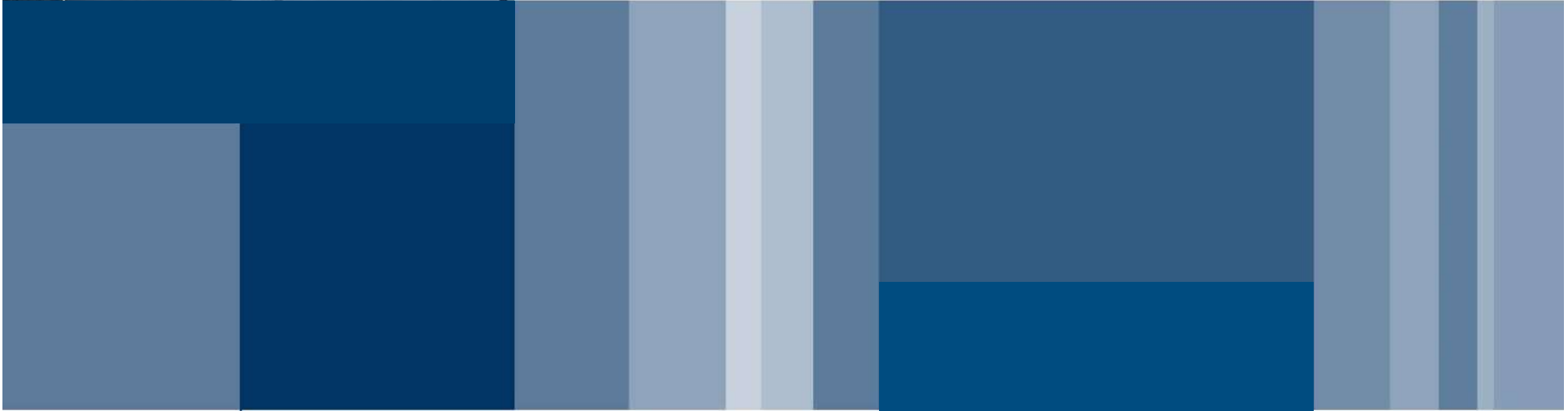




**POLITECNICO**  
**MILANO 1863**



# **MICROCONTROLLERS**

## **LAB – Atomic Operation**



## Main

```
while(1) {  
    // test (16-bit) oscilla tra 255 ed 256  
  
    if(test == 255)  
        test = 256;  
    else if(test == 256)  
        test = 255;  
}
```

Matematicamente parlando test  
può assumere solo 2 valori:

255 ed 256

```
// -----PORTB-----  
ANSELB.RB7 = 0;           // PORTB.RB7 Input Digital Buffer On  
TRISB.RB7 = 1;           // PORTB.RB7 Configured as Input  
IOCB = 0b10000000;       // RB7 IOCB Active  
// -----  
  
// ----Interrupt-----  
INTCON = 0b10001000;  
// -----
```

Interrupt-on-Change  
sul PORTB attivato

Interrupt Service Routine

```
void interrupt() { //ISR
```

```
if(INTCON.RBIF) { // Controllo la variabile test  
                    ad ogni interrupt
```

```
    if(PORTB.RB7)  
        if(test != 255 && test != 256)
```

```
            LATD++;
```

```
    INTCON.RBIF = 0; // Se test esce dal range segnalalo  
                      sul PORTD
```

```
}
```

```
}
```



```
while(1) {  
    if(test == 255)  
        test = 256;  
    else if(test == 256)  
        test = 255;  
}
```

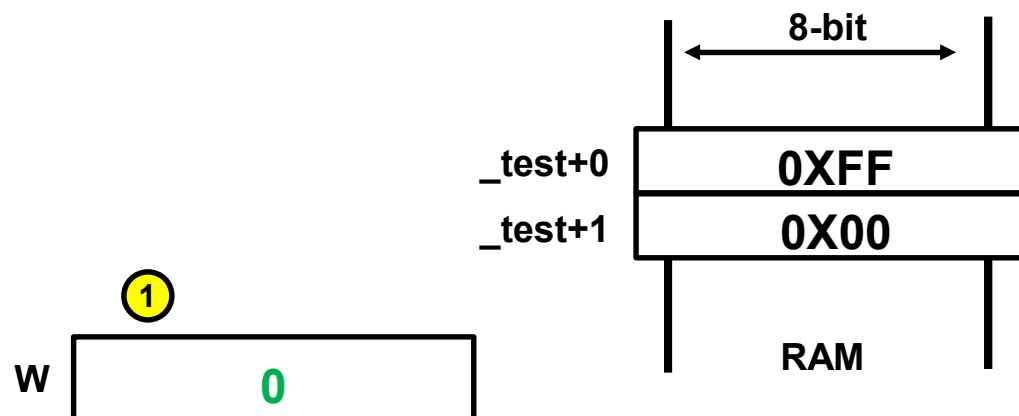
;TestAtom.c,25 :: test = 256;  
MOVLW 0  
MOVWF \_test+0  
MOVLW 1  
MOVWF \_test+1  
GOTO L\_main3

Vere istruzioni  
eseguite dalla  
CPU



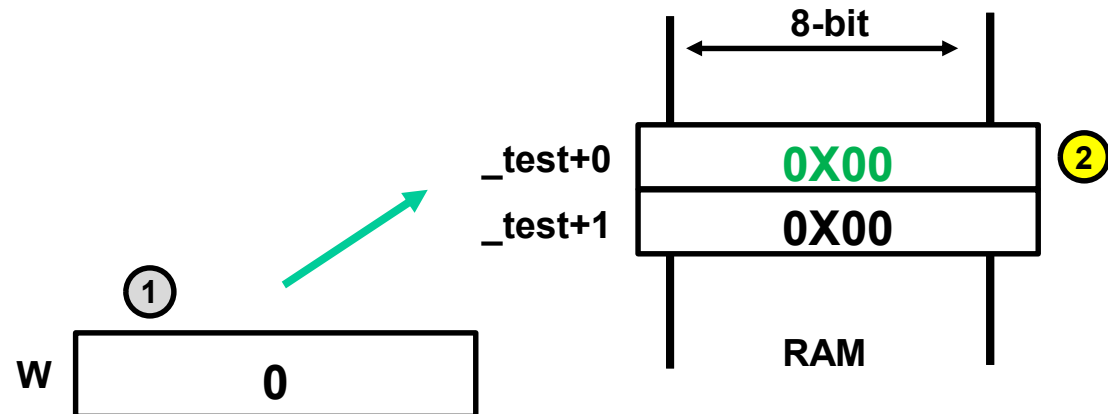
```
;TestAtom.c,25 ::          test = 256;
```

```
① MOVLW      0  
  MOVWF      _test+0  
  MOVLW      1  
  MOVWF      _test+1  
  GOTO       L_main3
```



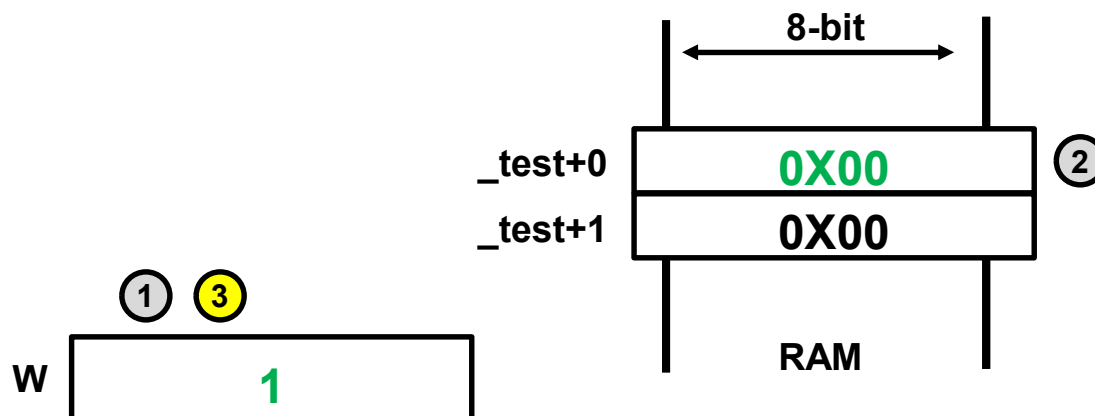


```
;TestAtom.c,25 ::          test = 256;  
    MOVLW          0  
    ② MOVWF        _test+0  
    MOVLW          1  
    MOVWF        _test+1  
    GOTO          L_main3
```



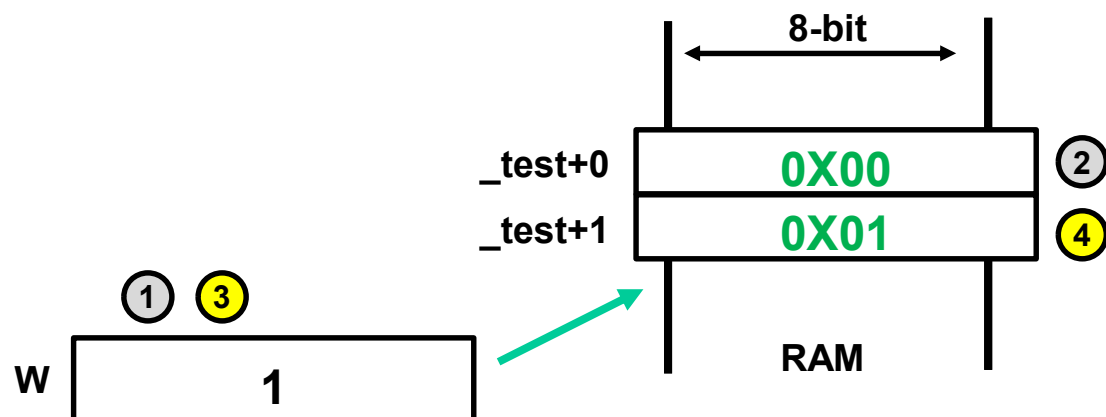


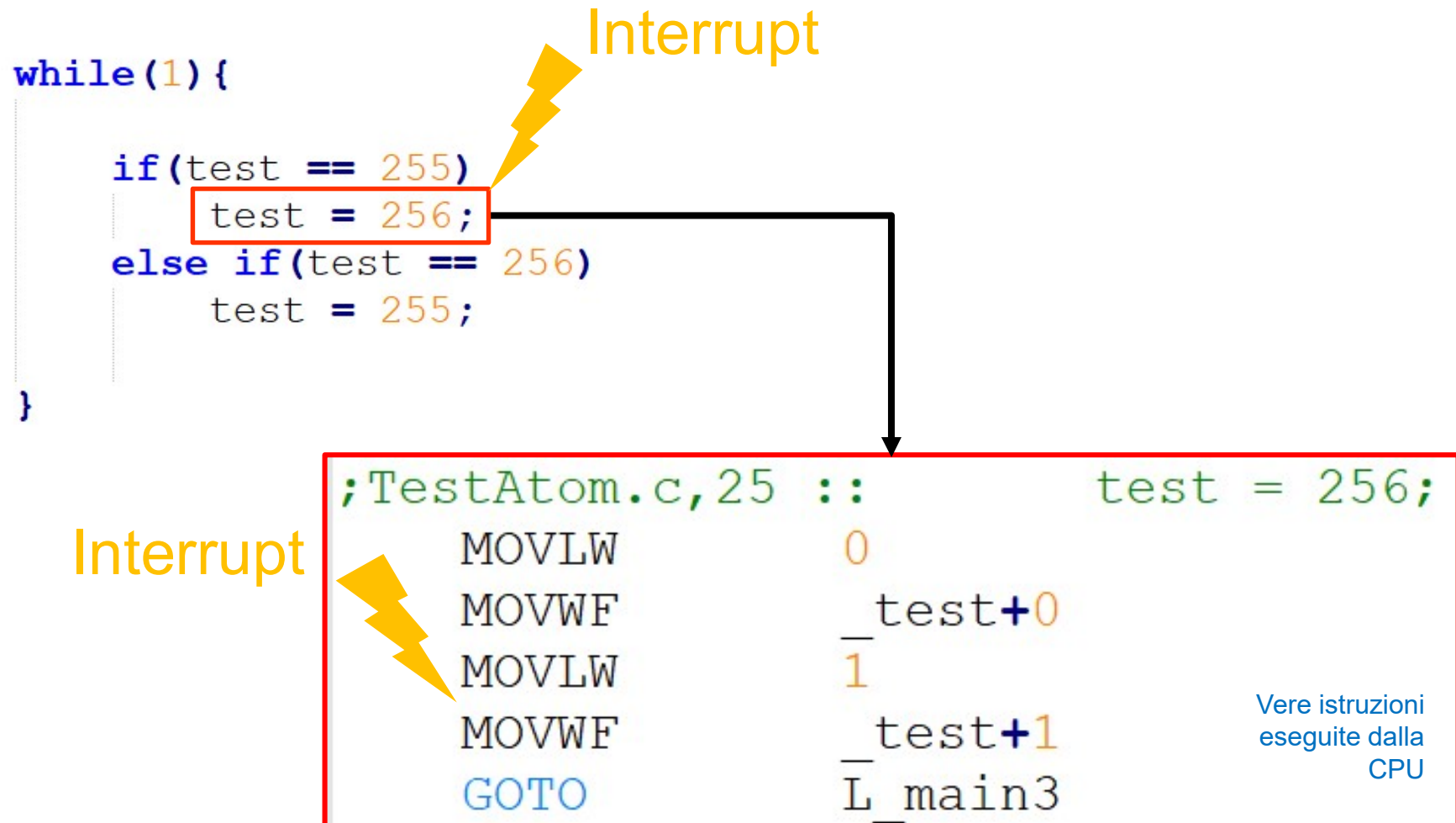
```
;TestAtom.c,25 ::          test = 256;  
    MOVLW          0  
    MOVWF          _test+0  
    ③ MOVLW          1  
    MOVWF          _test+1  
    GOTO           L_main3
```





```
;TestAtom.c,25 ::          test = 256;  
    MOVLW          0  
    MOVWF          _test+0  
    MOVLW          1  
    MOVWF          _test+1  
    GOTO           L_main3
```







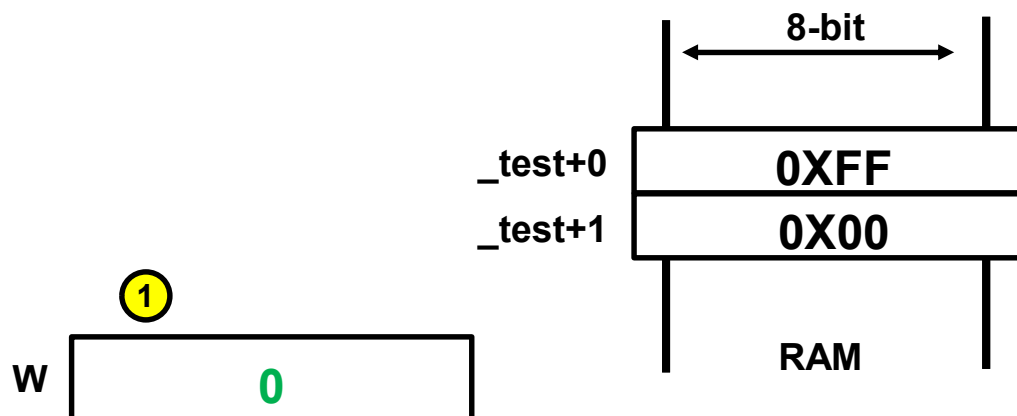


# Esecuzione Assembly con Interrupt

POLITECNICO  
MILANO 1863

```
;TestAtom.c,25 ::          test = 256;
```

```
① MOVLW      0  
  MOVWF      _test+0  
  MOVLW      1  
  MOVWF      _test+1  
  GOTO       L_main3
```

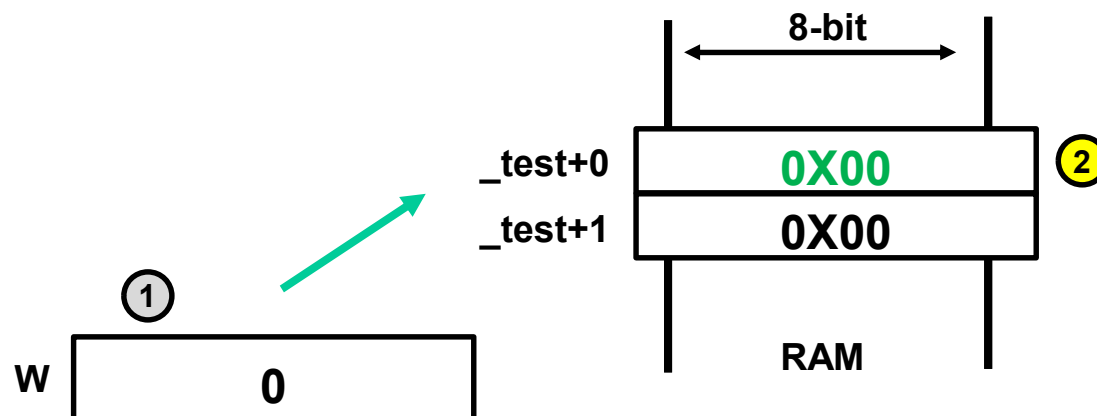




# Esecuzione Assembly con Interrupt

POLITECNICO  
MILANO 1863

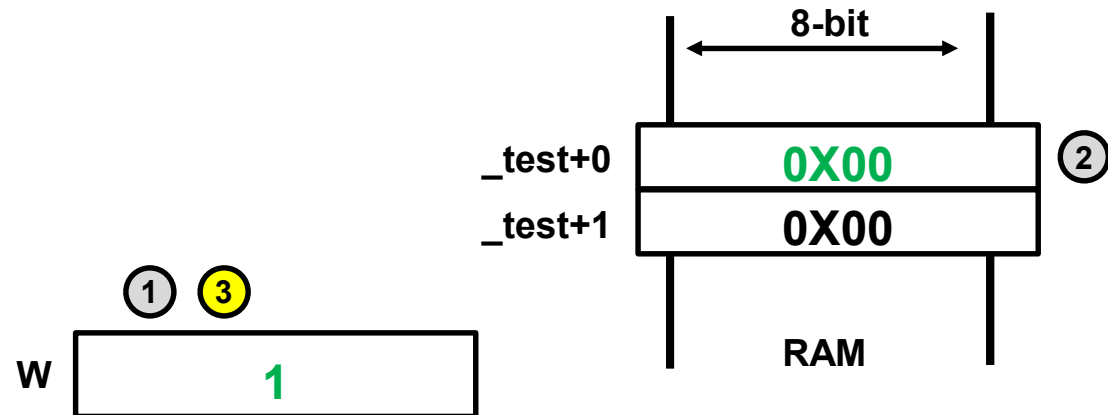
```
;TestAtom.c,25 ::          test = 256;  
    MOVLW          0  
    ② MOVWF         _test+0  
    MOVLW          1  
    MOVWF         _test+1  
    GOTO          L_main3
```





```
;TestAtom.c,25 ::      test = 256;  
    MOVLW      0  
    MOVWF      _test+0  
    ③ MOVLW      1  
    MOVWF      _test+1  
    GOTO       L_main3
```

Interrupt



**STOP EXE**

**CALL void interrupt() ISR**



```
;TestAtom.c,25 ::          test = 256;
    MOVLW          0
    MOVWF          _test+0
    ③ MOVLW          1
    MOVWF          _test+1
    GOTO           L_main3
```

```
void interrupt() { //ISR
```

```
    if (INTCON.RBIF) {
```

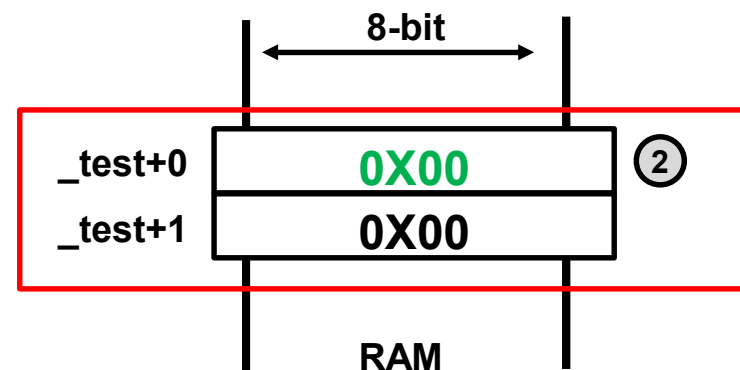
```
        if (PORTB.RB7)
```

```
            if (test != 255 && test != 256)
                LATD++;
```

```
        INTCON.RBIF = 0;
```

```
    }
```

```
}
```




Comportamento non voluto ed imprevedibile



**SOLUZIONE:** Rendere l'operazione atomica, i.e. garantire che la copia avvenga senza interruzioni

Rimandare  
l'interrupt dopo  
l'operazione

```
;TestAtom.c, 25 ::          test = 256;  
    MOVLW          0  
    MOVWF          _test+0  
    MOVLW          1  
    MOVWF          _test+1  
    GOTO           L_main3
```





- Disabilitare gli interrupt momentaneamente
- Scrivere codice che intrinsecamente non commette errori

```
while(1) {  
  
    INTCON.GIE = 0;  
  
    if(test == 255)  
        test = 256;  
    else if(test == 256)  
        test = 255;  
  
    INTCON.GIE = 1;  
}
```



Ogni variabile, definita globale e modificata nella ISR, può cambiare il suo valore ad ogni punto del programma.

A volte, i compilatori non tenendo in considerazione questo possibile ed imprevedibile cambiamento, eseguono delle ottimizzazioni sul codice che cambiano in maniera radicale il comportamento del programma portandoci lontano da quello voluto.

Per evitare qualsiasi problema, generalmente, a tutte le variabili di cui sopra si mette l'attributo «volatile».

```
volatile int foo;
```

«Volatile» evita qualsiasi ottimizzazione del codice legato a quella variabile