Supponiamo di dover eseguire il seguente programma:

```
char buf[0x2000] = { 2, 6, -1, 200, ...,
  15, 3, -32, 1};
int main()
{
  int sum = 0;
  for (int i = 0; i < 0x2000; i++)
    sum += buf[i];
  return sum;
}
```

L'array buf contiene una serie di numeri di cui vogliamo conoscere la somma.

Una possible traduzione in assembly e linguaggio macchina è:

```
    | .text                                |
    | .global main                         |
0000| main: pushq %rbp                     |55
0001|       movq  %rsp, %rbp               |48 89 e5
0004|       subq $8, %rsp                  |48 83 ec 08
0008|       movl $0, -8(%rbp)              |c7 45 f8 00 00 00 00
000f|       movl $0, -4(%rbp)              |c7 45 fc 00 00 00 00
0016| for:  cmpq $0x2000, -4(%rbp)         |48 81 7d fc 00 20 00 00
0013|       jge fine                       |7d 14
0020|       movslq -4(%rbp), %rcx          |48 63 4d fc
0024|       movsbl buf(%rcx), %eax         |0f be 81 00 10 00 00
002b|       addl %eax, -8(%rbp)            |01 45 f8
002e|       addl $1, -4(%rbp)              |83 45 fc 01
0032|       jmp for                        |eb e2
0034| fine: movl -8(%rbp), %eax            |8b 45 f8
0037|       popq %rbp                      |5d
0038|       ret                            |c3
    |  .data                               | ...
1000| buf:  .byte 2, 6, -1, 200            |02 06 ff c8
    |         ...                          | ...
2ffc|       .byte 15, 3, -32, 1            |0f 03 e0 01
```

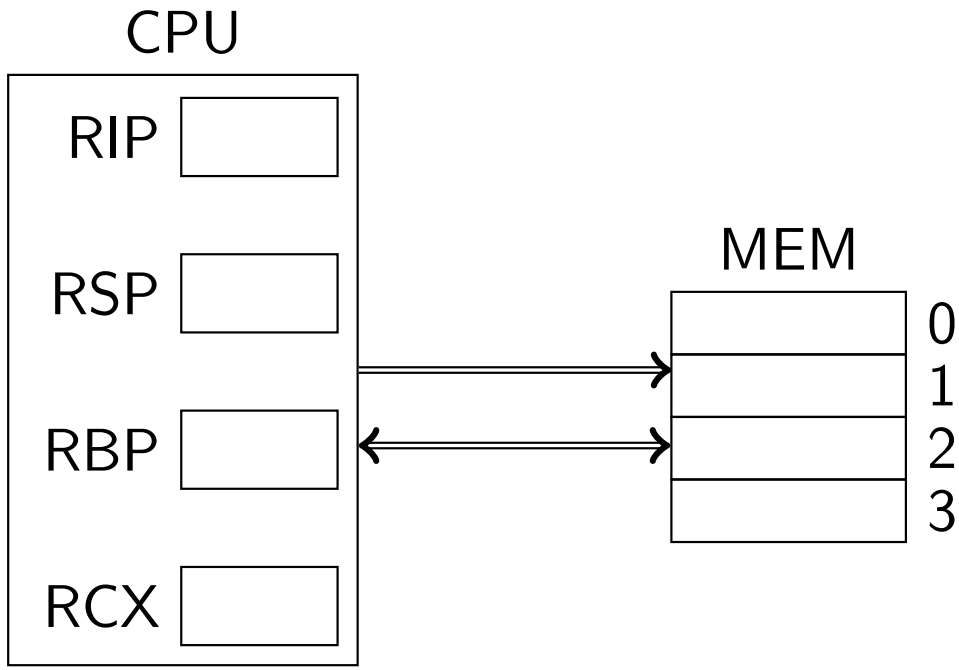
- Supponiamo per semplicità che gli indirizzi possibili vadano da 0000 a 3fff (16 KiB di memoria).

- Supponiamo per semplicità che gli indirizzi possibili vadano da 0000 a 3fff (16 KiB di memoria).
- Immaginiamo di suddividere la memoria in 4 *pagine*, ciascuna grande 4 KiB (1000 in esadecimale);

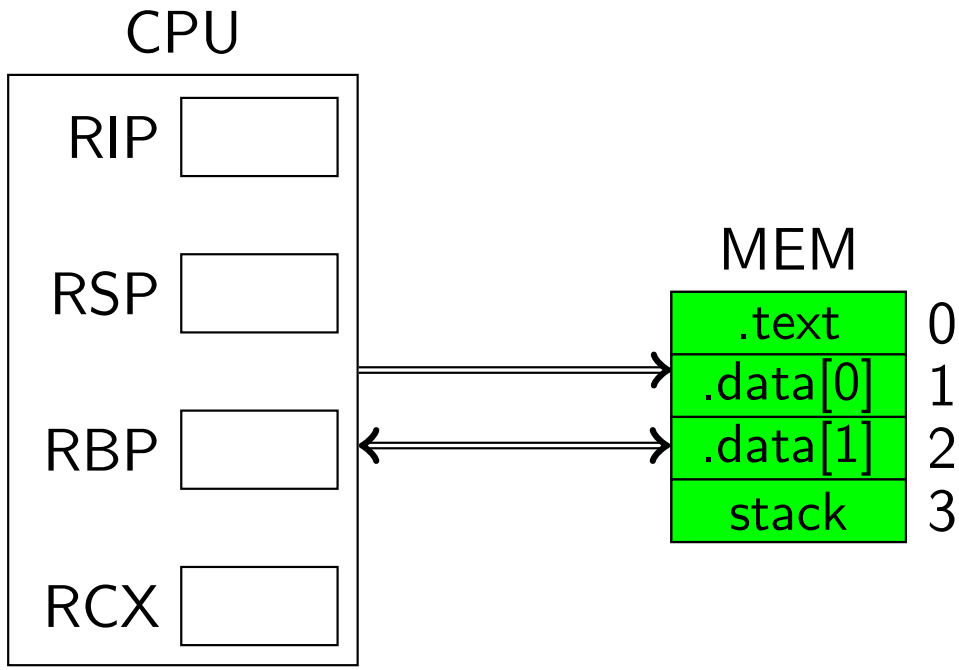
- Supponiamo per semplicità che gli indirizzi possibili vadano da 0000 a 3fff (16 KiB di memoria).
- Immaginiamo di suddividere la memoria in 4 *pagine*, ciascuna grande 4 KiB (1000 in esadecimale);
- Il nostro programma contiene il codice nella pagina che va da 0000 a 0fff (pagina 0) e la variabile `buf` nelle due pagine successive (1 e 2). Usiamo l'ultima pagina (3) come pila.

- Supponiamo per semplicità che gli indirizzi possibili vadano da 0000 a 3fff (16 KiB di memoria).
- Immaginiamo di suddividere la memoria in 4 *pagine*, ciascuna grande 4 KiB (1000 in esadecimale);
- Il nostro programma contiene il codice nella pagina che va da 0000 a 0fff (pagina 0) e la variabile `buf` nelle due pagine successive (1 e 2). Usiamo l'ultima pagina (3) come pila.
- Il programma ha dunque bisogno di tutto lo spazio di indirizzamento.

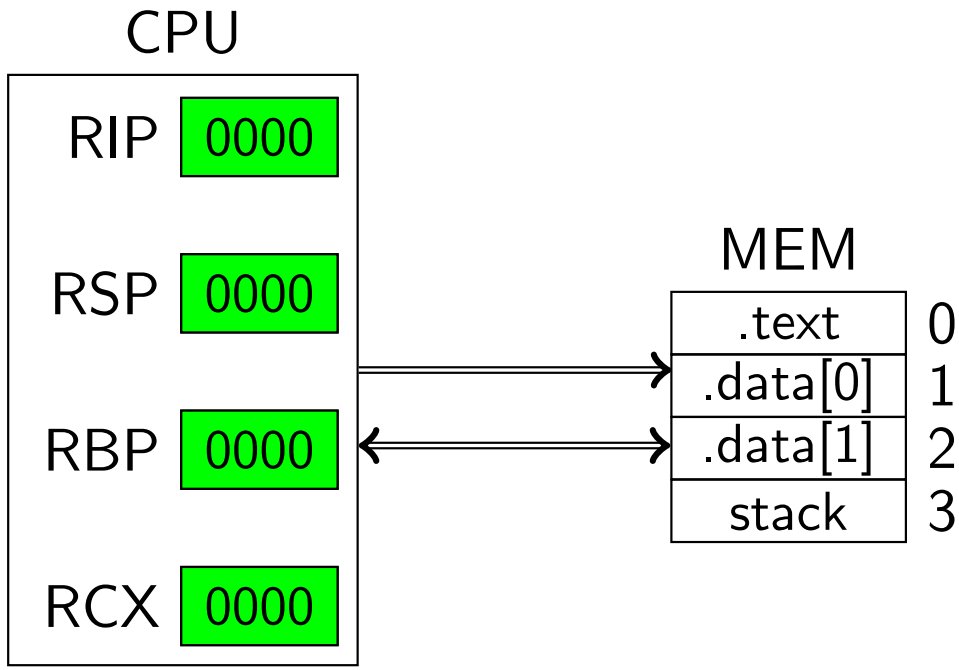
- Supponiamo per semplicità che gli indirizzi possibili vadano da 0000 a 3fff (16 KiB di memoria).
- Immaginiamo di suddividere la memoria in 4 *pagine*, ciascuna grande 4 KiB (1000 in esadecimale);
- Il nostro programma contiene il codice nella pagina che va da 0000 a 0fff (pagina 0) e la variabile `buf` nelle due pagine successive (1 e 2). Usiamo l'ultima pagina (3) come pila.
- Il programma ha dunque bisogno di tutto lo spazio di indirizzamento.
- Dobbiamo però eseguirlo su un sistema che ha soltanto 8 KiB (2 frame).



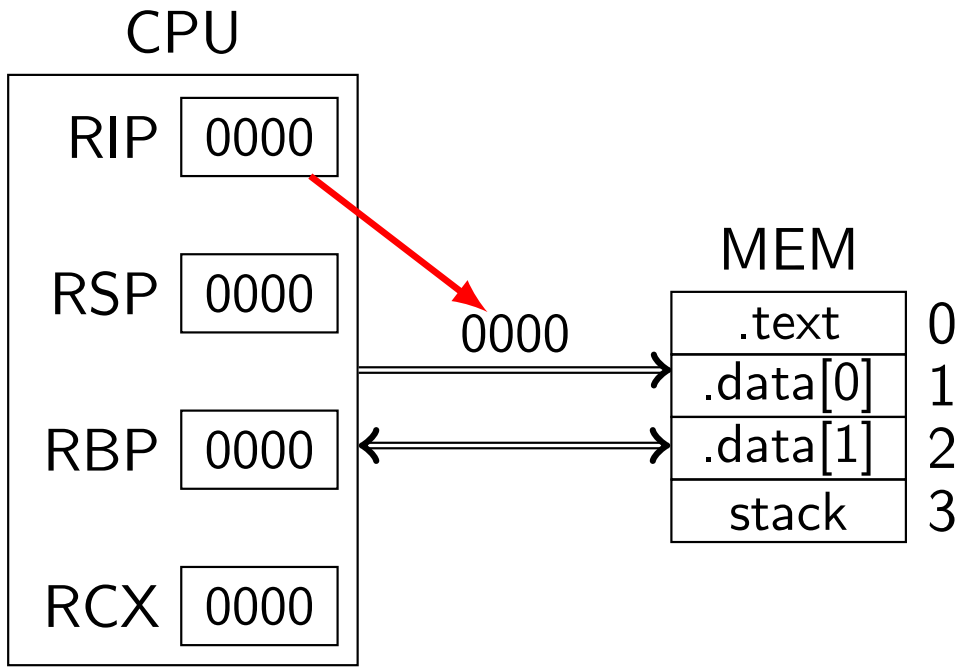
Vediamo prima l'esecuzione sul sistema che ha tutta la memoria. Mostriamo solo alcuni dei registri della CPU. Tutti i numeri saranno mostrati in esadecimale.



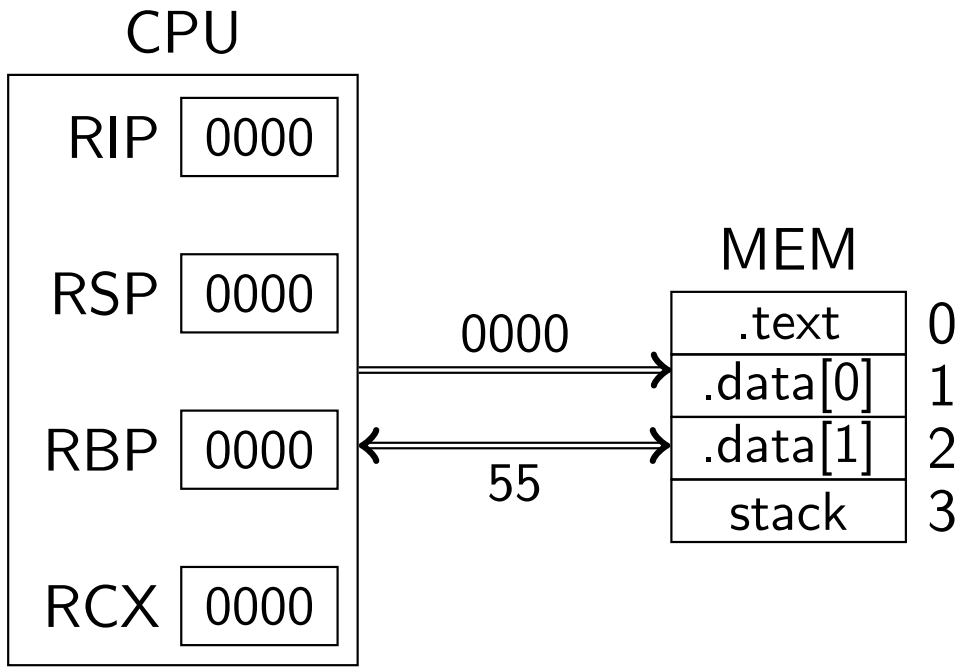
Carichiamo il programma in memoria.



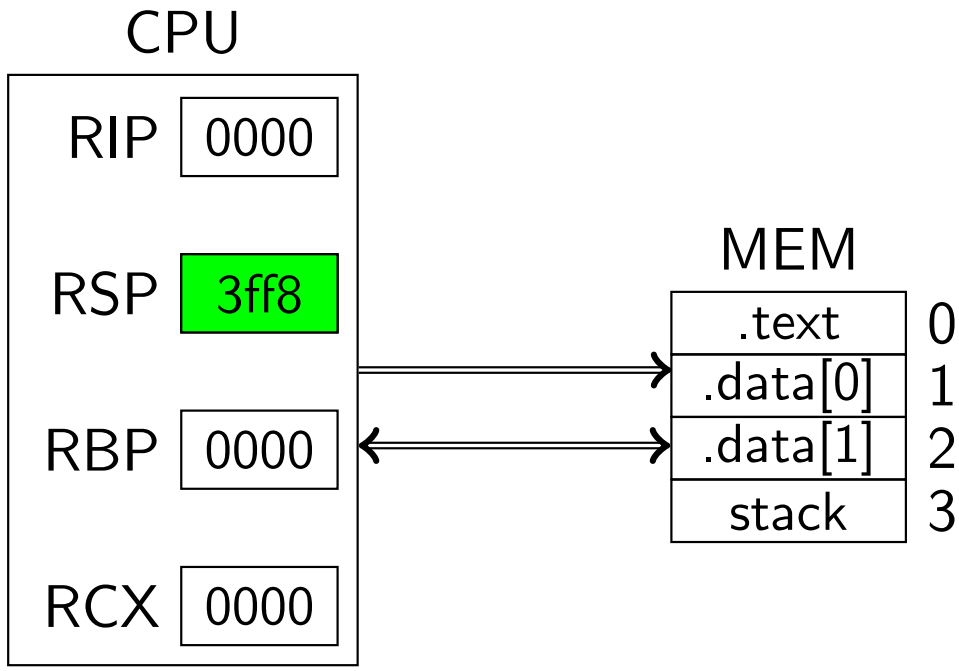
Inizializziamo tutti i registri con zero.



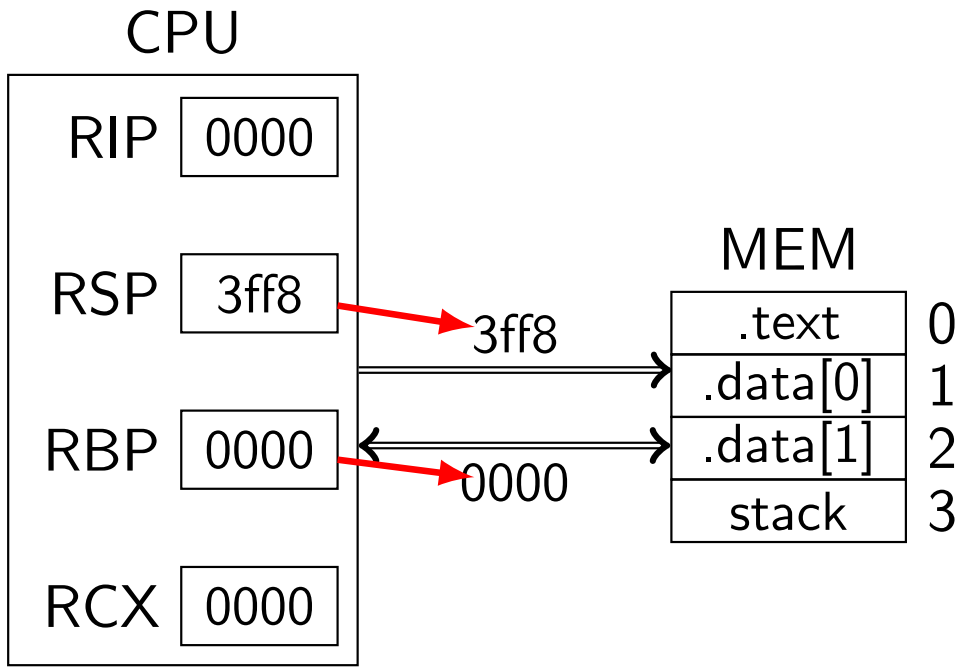
Avviamo il sistema. La CPU tenta di prelevare l'istruzione all'indirizzo contenuto in RIP. Inizia quindi una operazione di lettura in memoria all'indirizzo 0000.



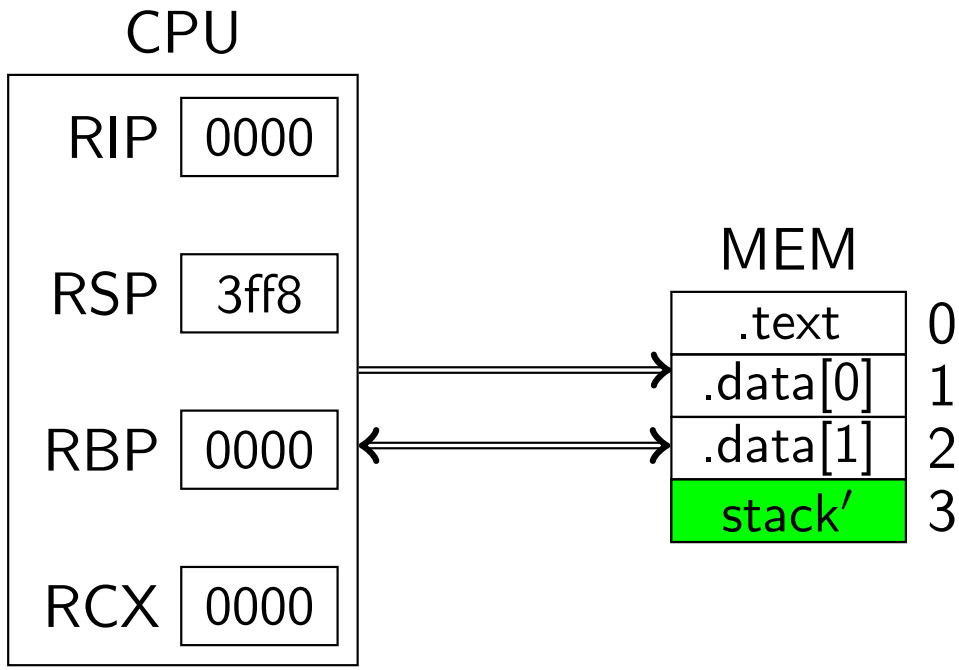
L'operazione di lettura restituisce l'istruzione `pushq %rbp`.



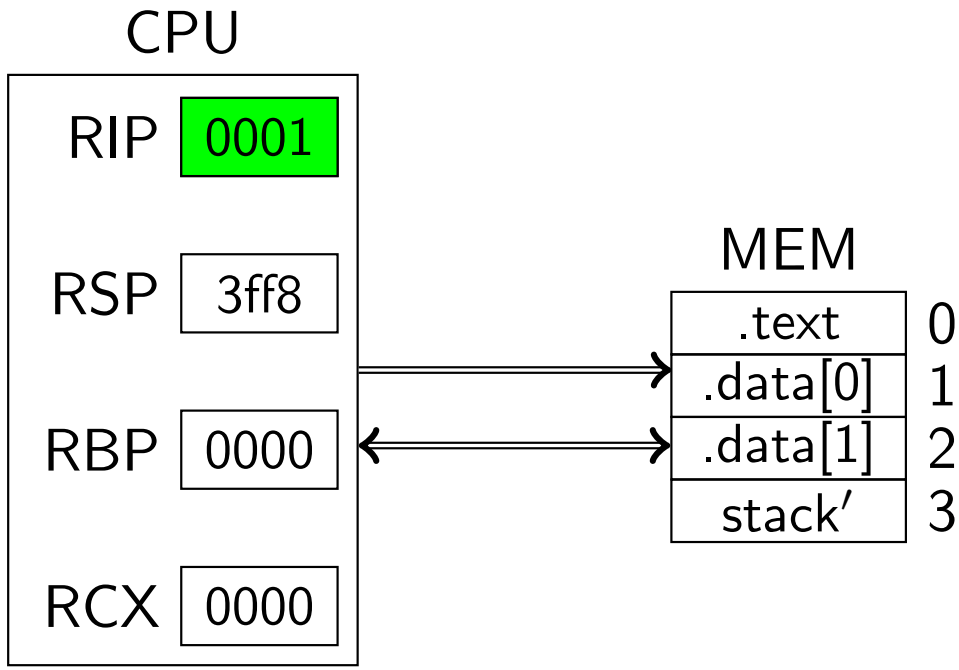
La CPU inizia ad eseguire l'istruzione. Il primo passo è decrementare RSP di 8.



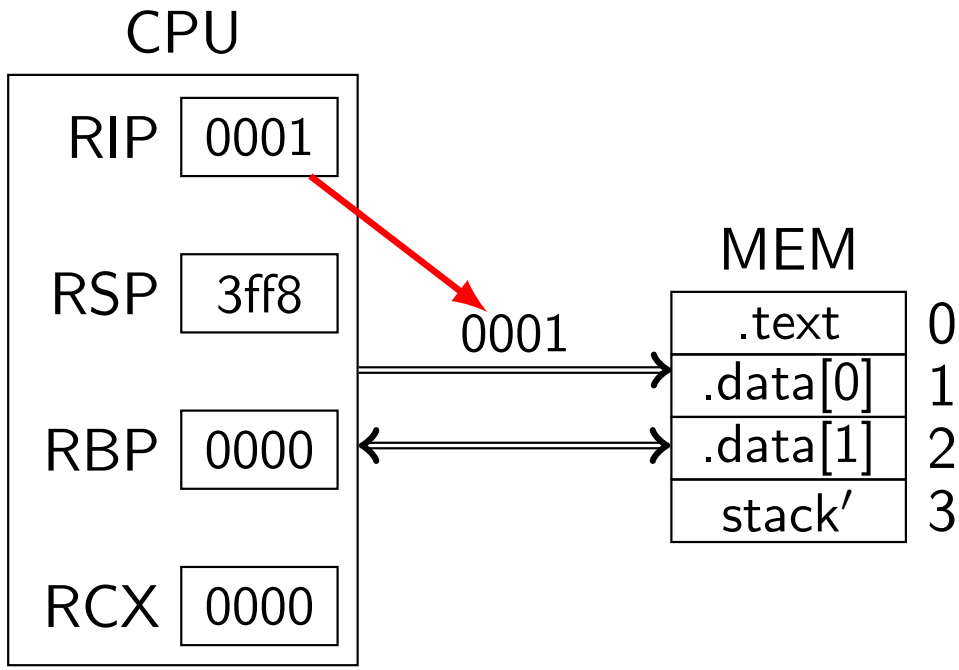
Il secondo passo è scrivere il contenuto di RBP all'indirizzo contenuto in RSP. La CPU inizia quindi una operazione di scrittura all'indirizzo 3ff8.



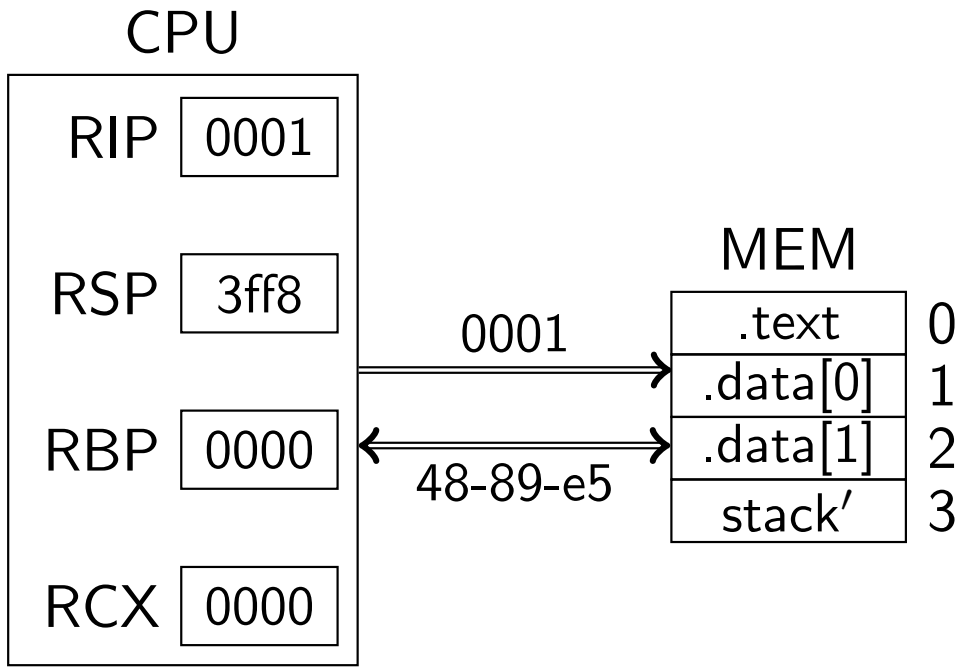
Si noti che dopo la scrittura il contenuto dello stack sarà cambiato.



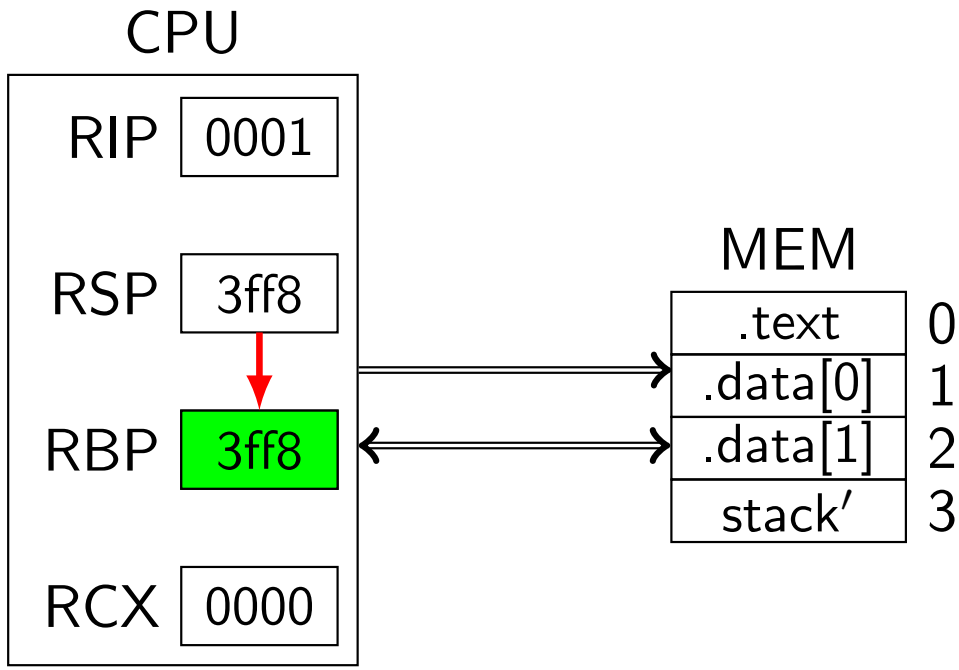
La CPU ha completato l'esecuzione dell'istruzione `pushq %rbp` e può passare alla successiva. Incrementa RIP di 1 (che è la dimensione dell'istruzione appena terminata).



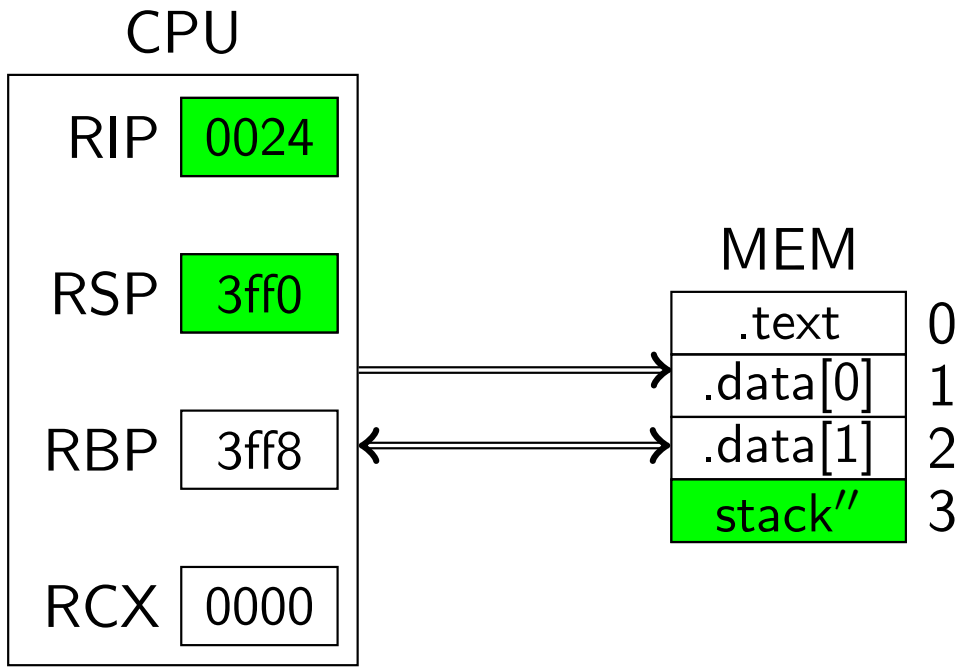
Nuovo ciclo: la CPU tenta di prelevare l'istruzione all'indirizzo contenuto in RIP. Inizia una operazione di lettura in memoria all'indirizzo 0001.



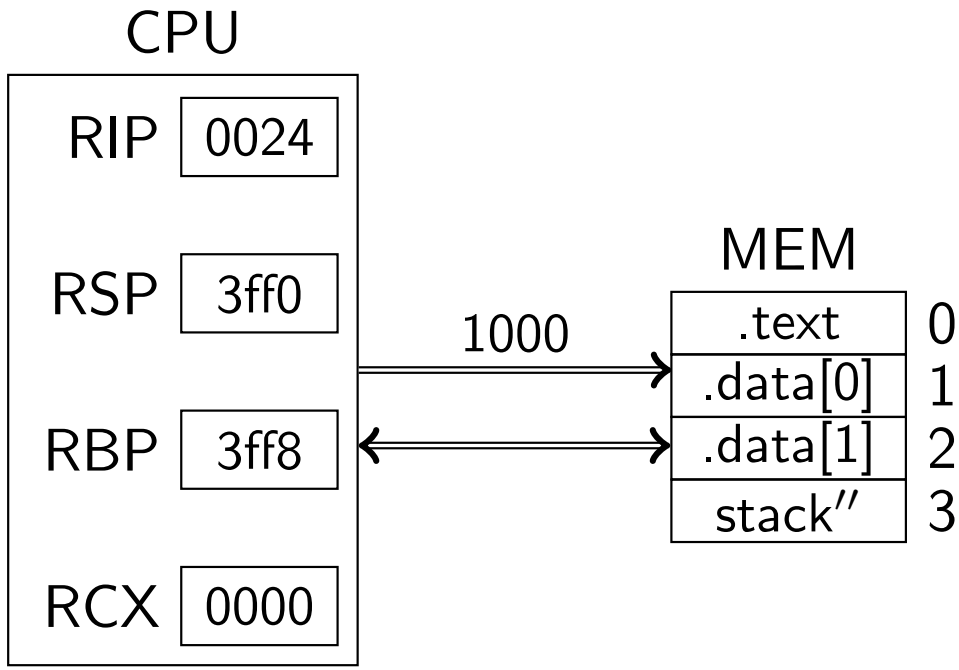
La CPU riceve l'istruzione `movq %rsp, %rbp`.



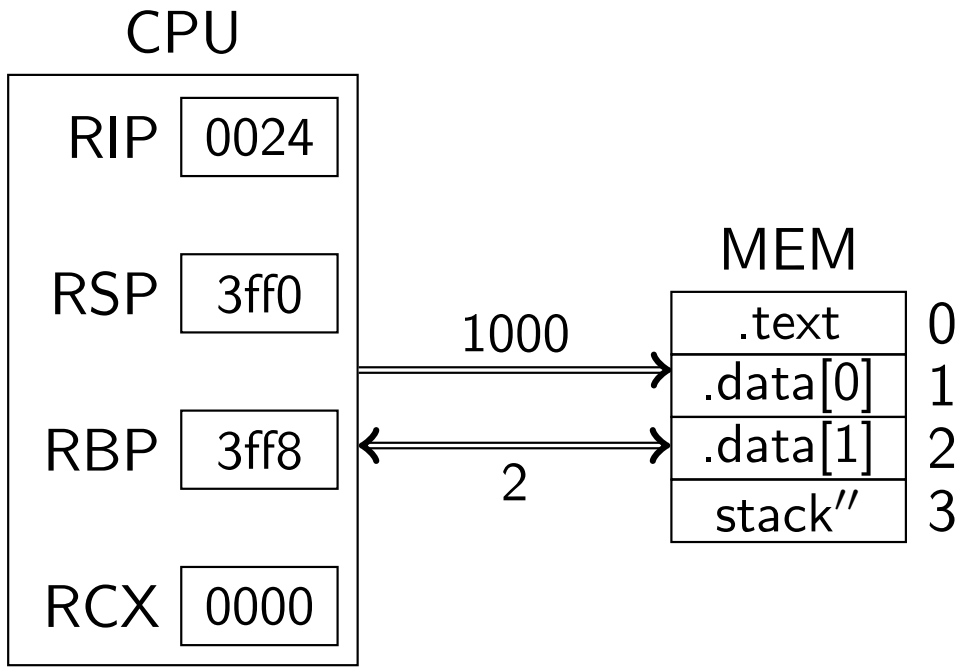
Per eseguirla copia il contenuto di RSP in RBP.
L'istruzione non prevede accessi in memoria.



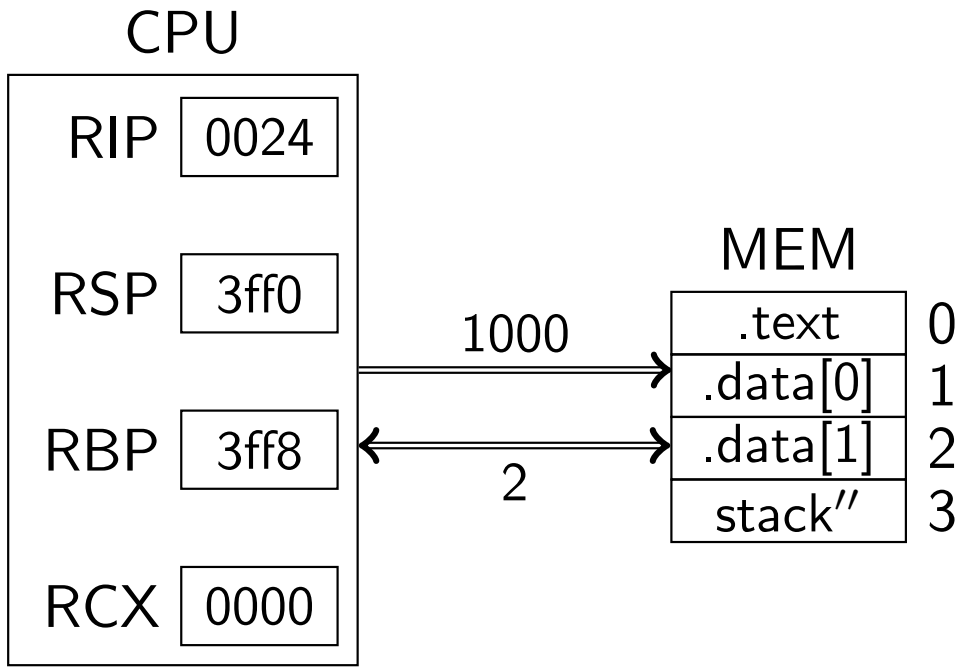
Dopo alcune istruzioni, l'esecuzione del programma arriva all'istruzione "`movsbl buf(%rcx), %rax`" che si trova all'indirizzo 0024. Questa è la prima istruzione che accede alla sezione `.data` del programma.



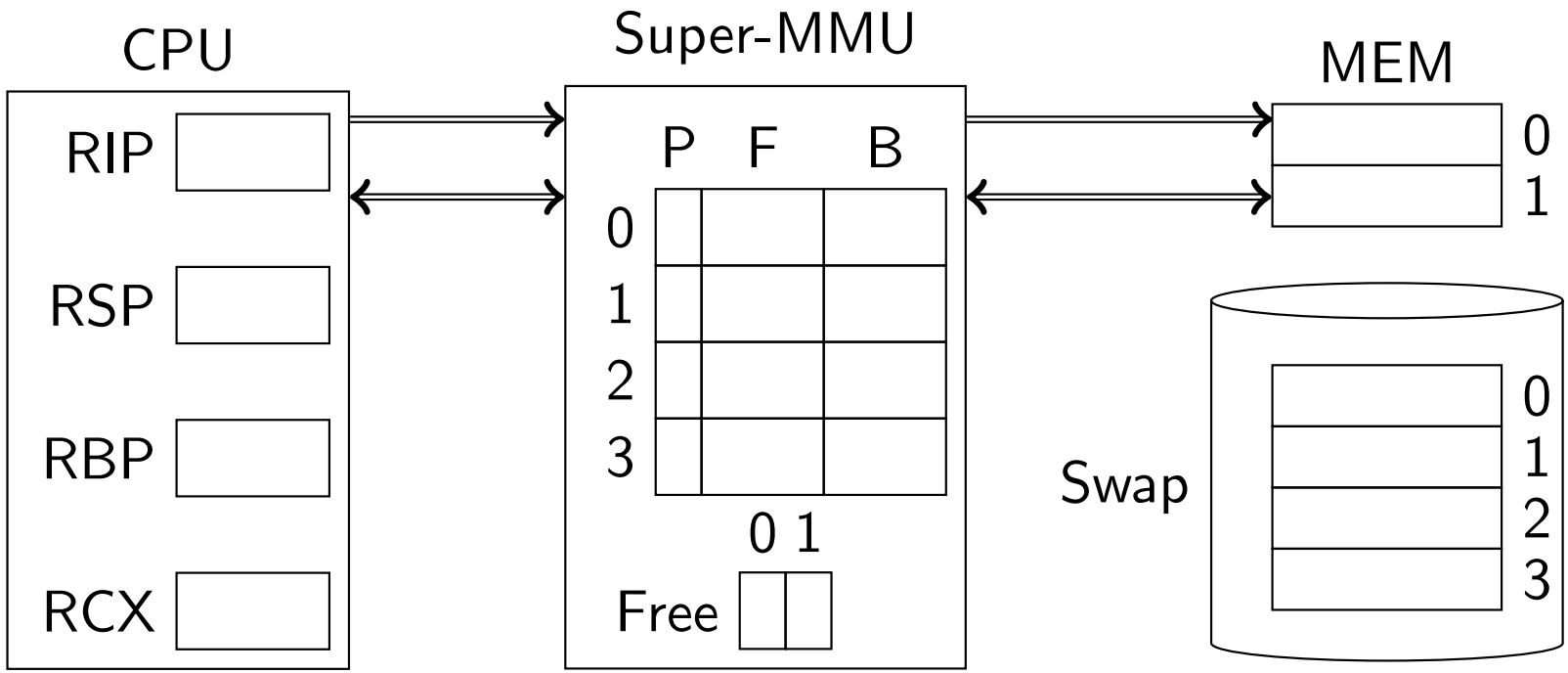
La CPU somma il contenuto di RCX e la costante 1000 (buf nel sorgente), ottenendo l'indirizzo 1000. Inizia dunque una operazione di lettura a questo indirizzo.



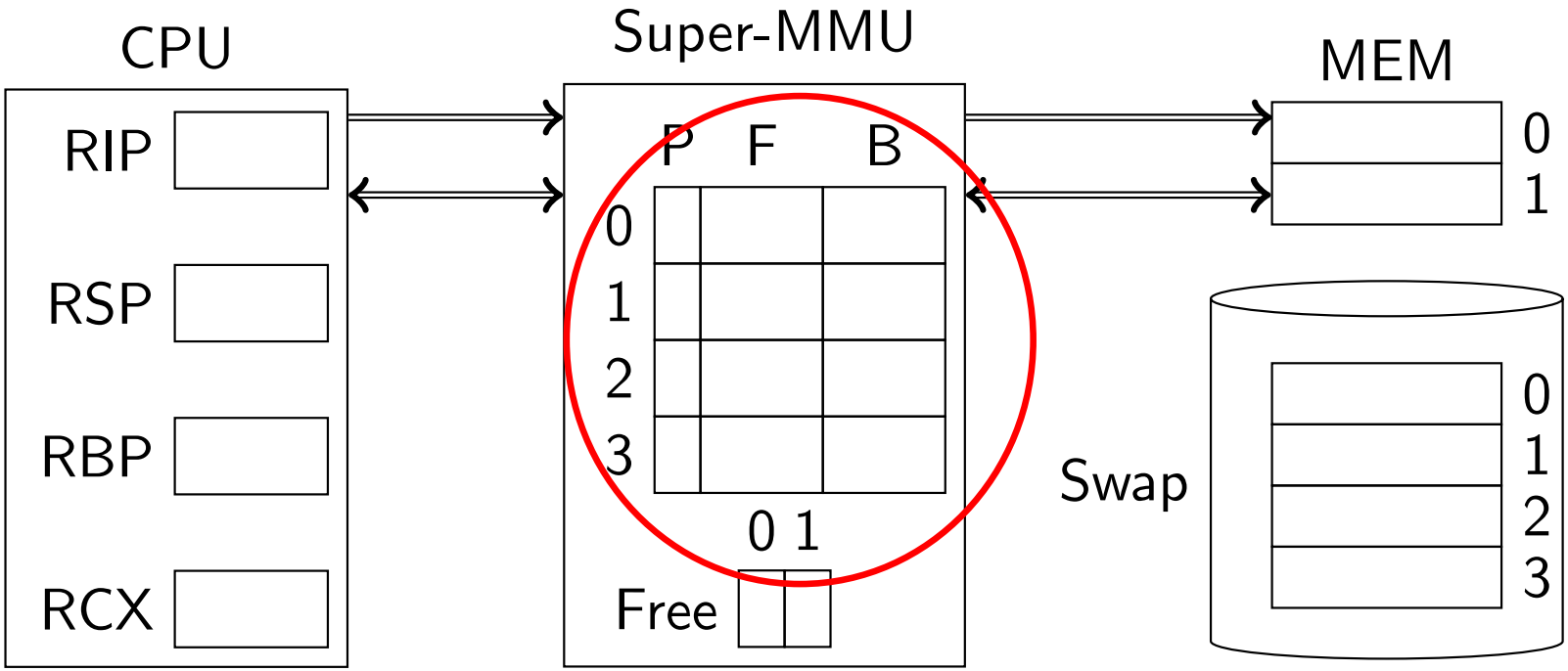
La CPU riceve il primo byte contenuto nel buffer (valore 2) e lo copia nel registro EAX (non mostrato).



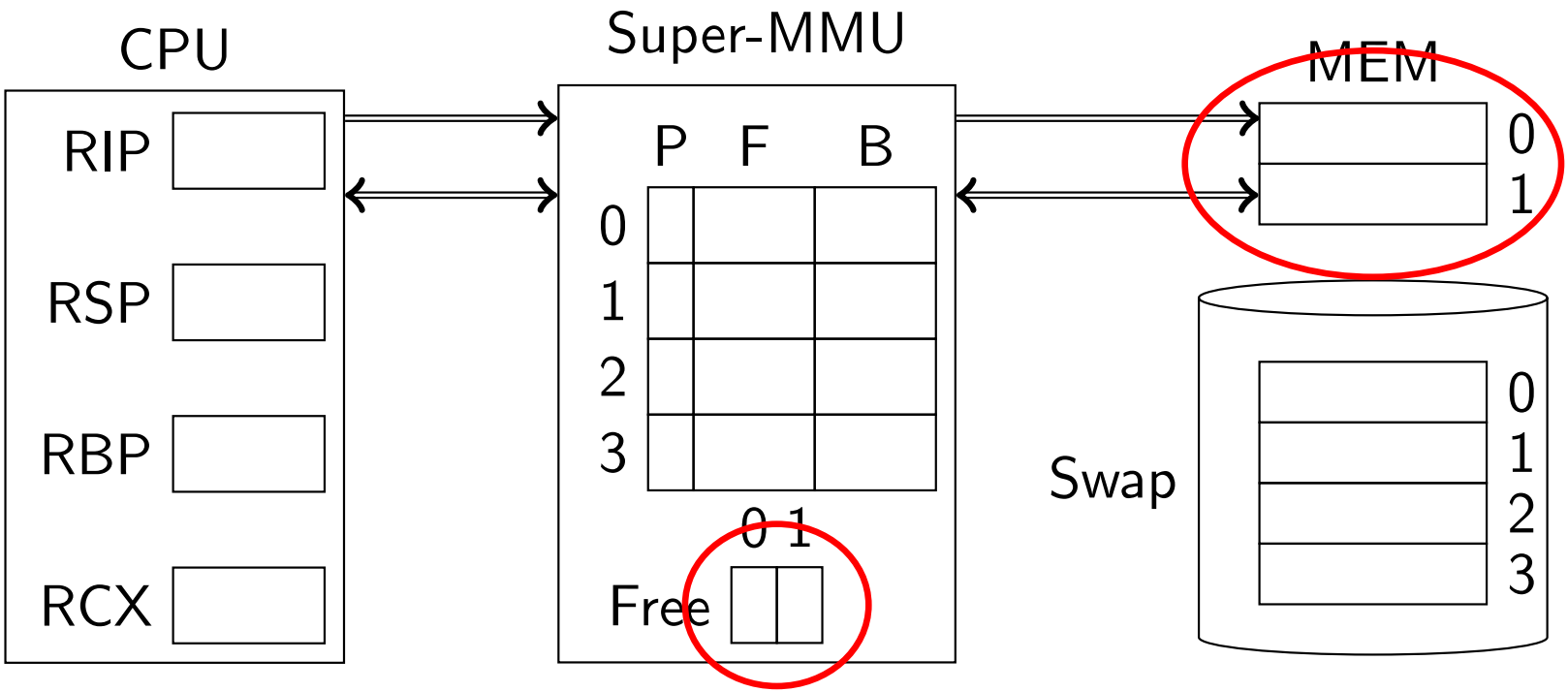
L'esecuzione prosegue sommando EAX in `-8(%rbp)`, incrementando RCX, etc., fino a sommare tutti i valori.



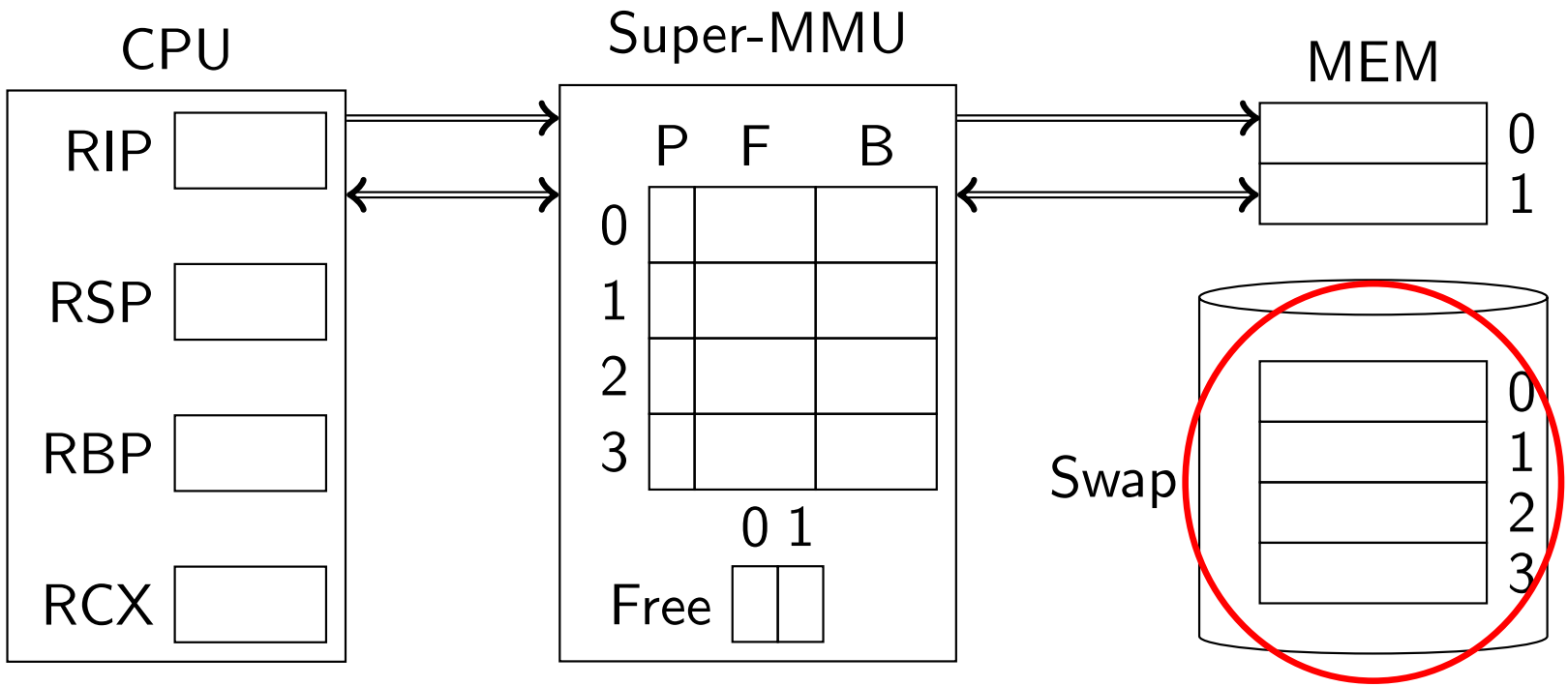
Vediamo ora l'esecuzione con la memoria virtuale.



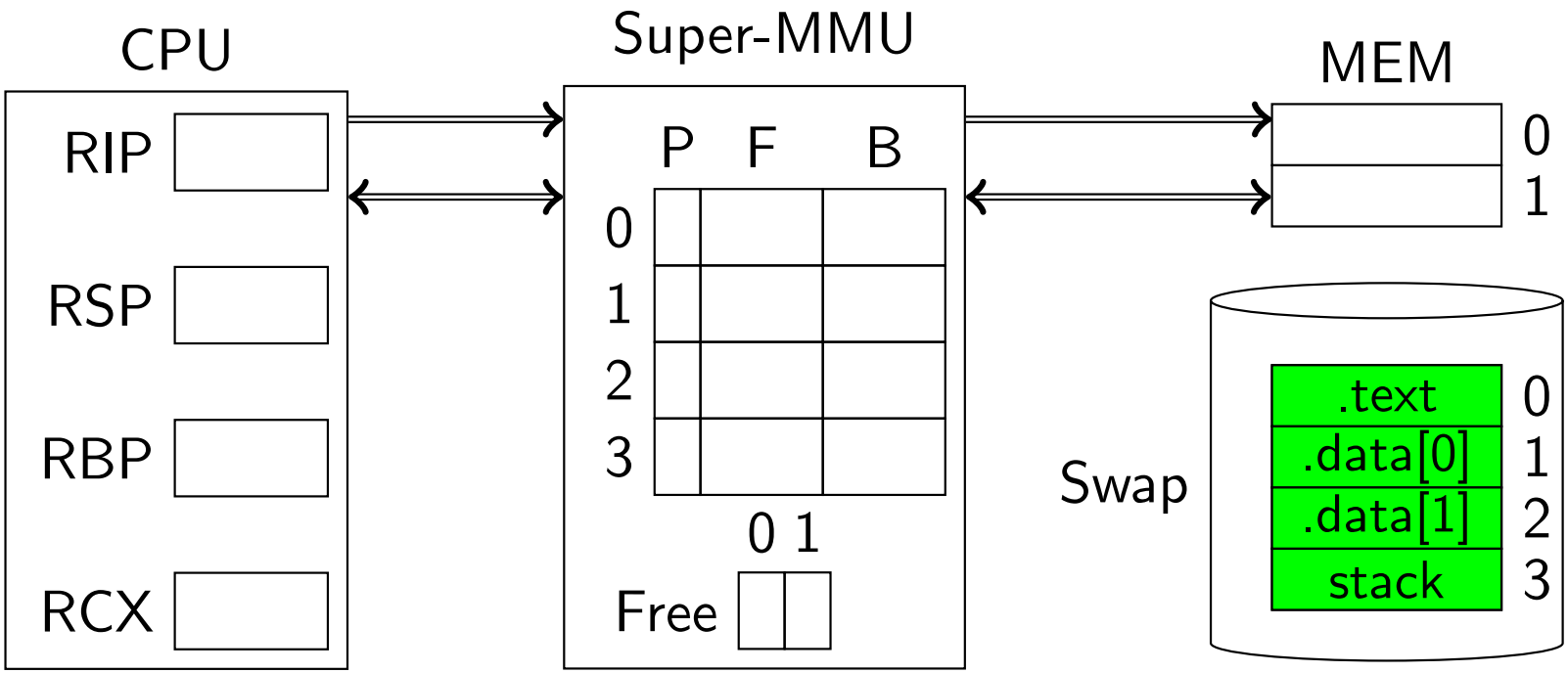
La MMU contiene una tabella di corrispondenza con una riga per ogni possibile pagina virtuale



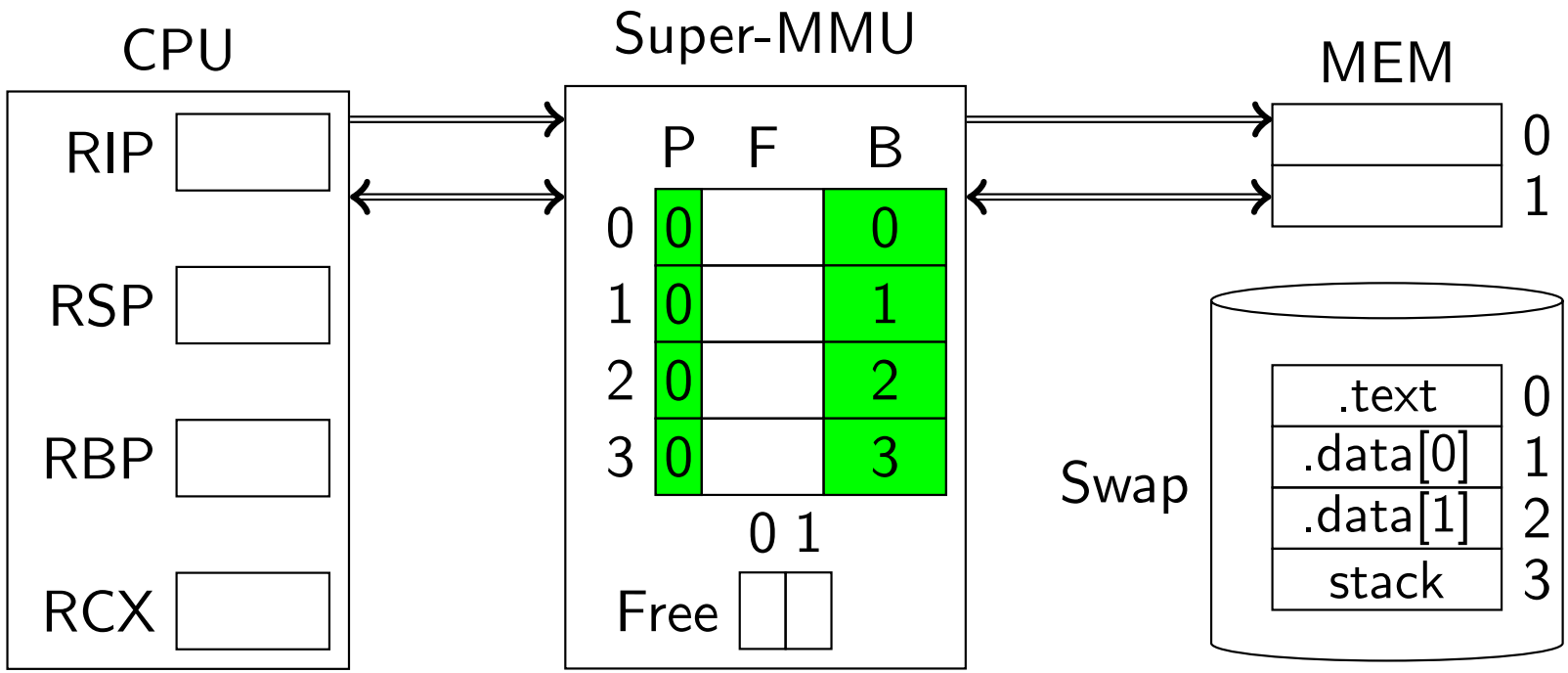
. . . e una tabellina Free con un bit per ogni frame.



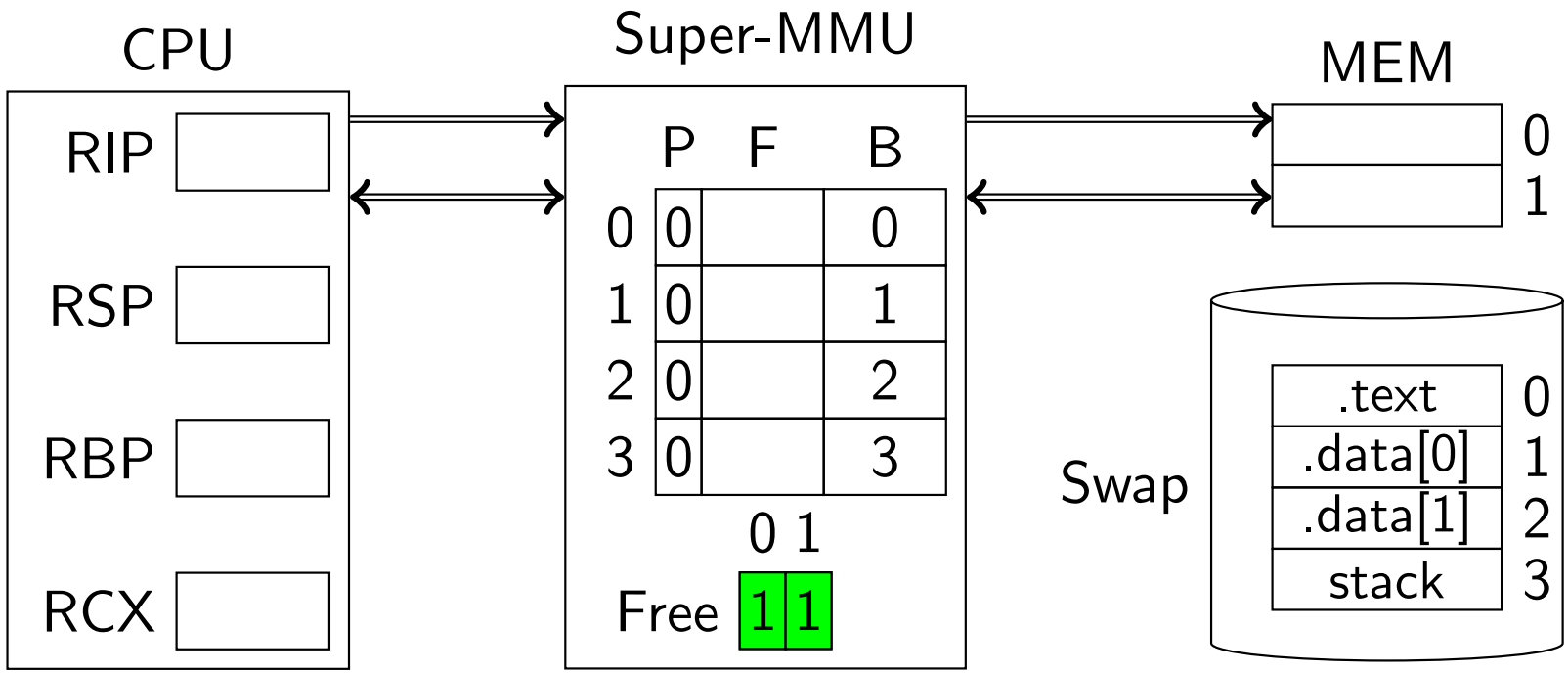
L'area di swap è una parte dell'hard disk.



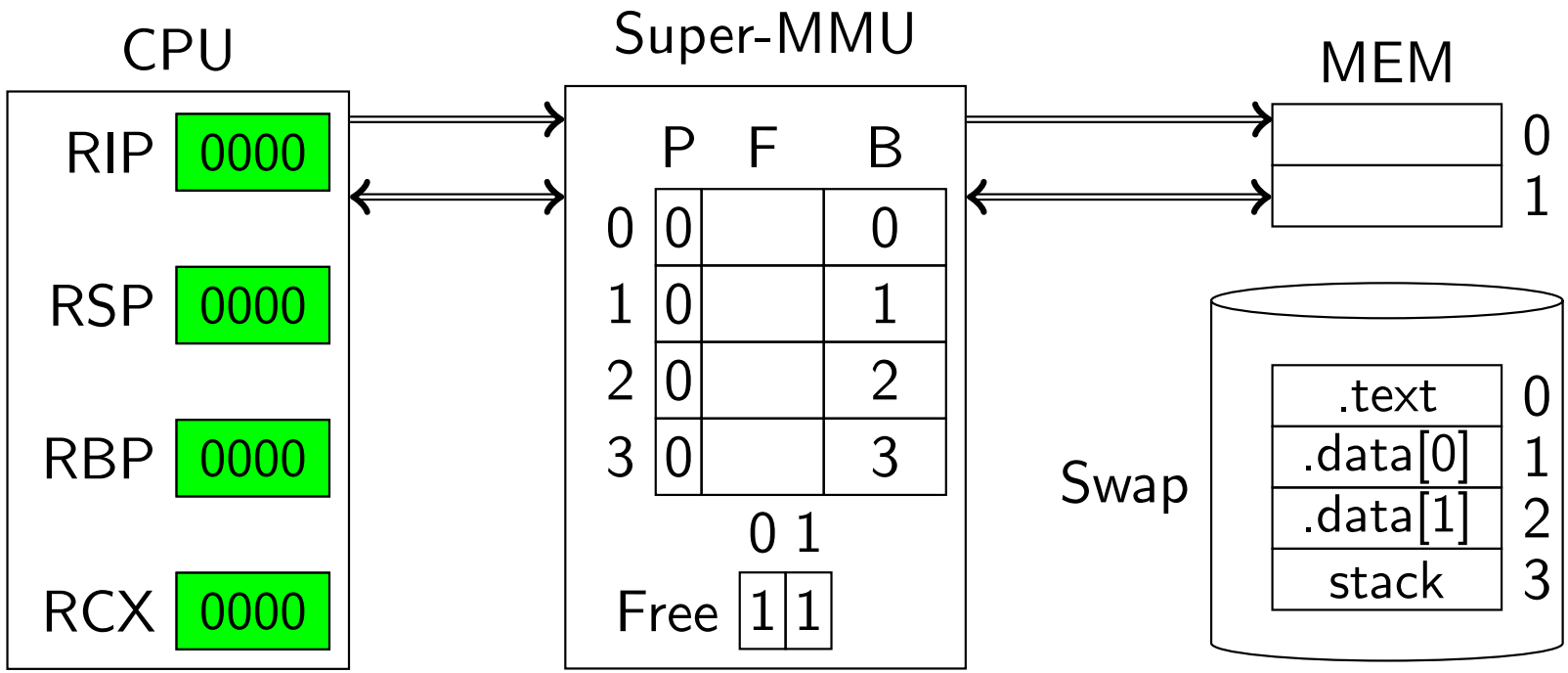
Inizializziamo l'area di swap sull'hard disk con il contenuto del nostro programma. L'ordine con cui copiamo le pagine non è importante.



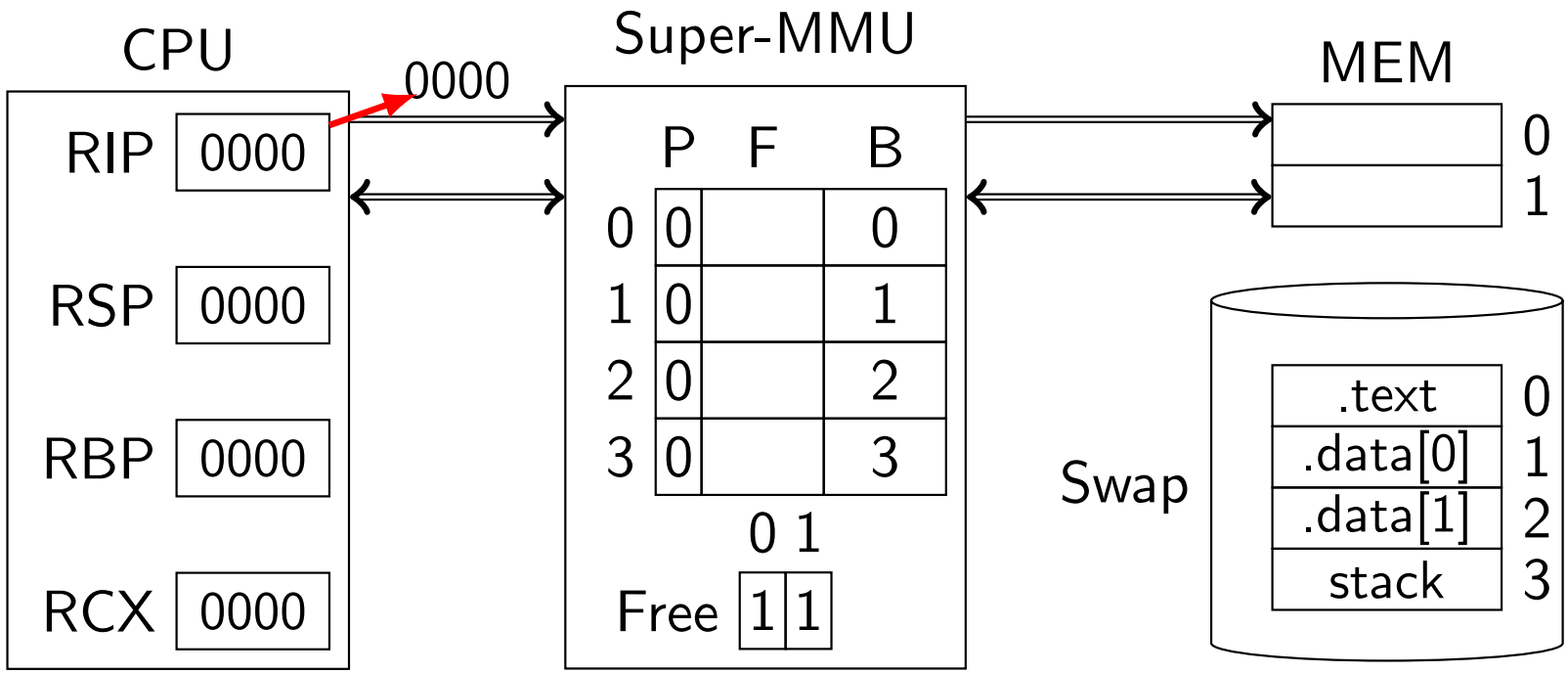
Inizializziamo la tabella di corrispondenza: tutte le pagine sono assenti ($P = 0$) e i campi B puntano ai corrispondenti blocchi nell'area di swap.



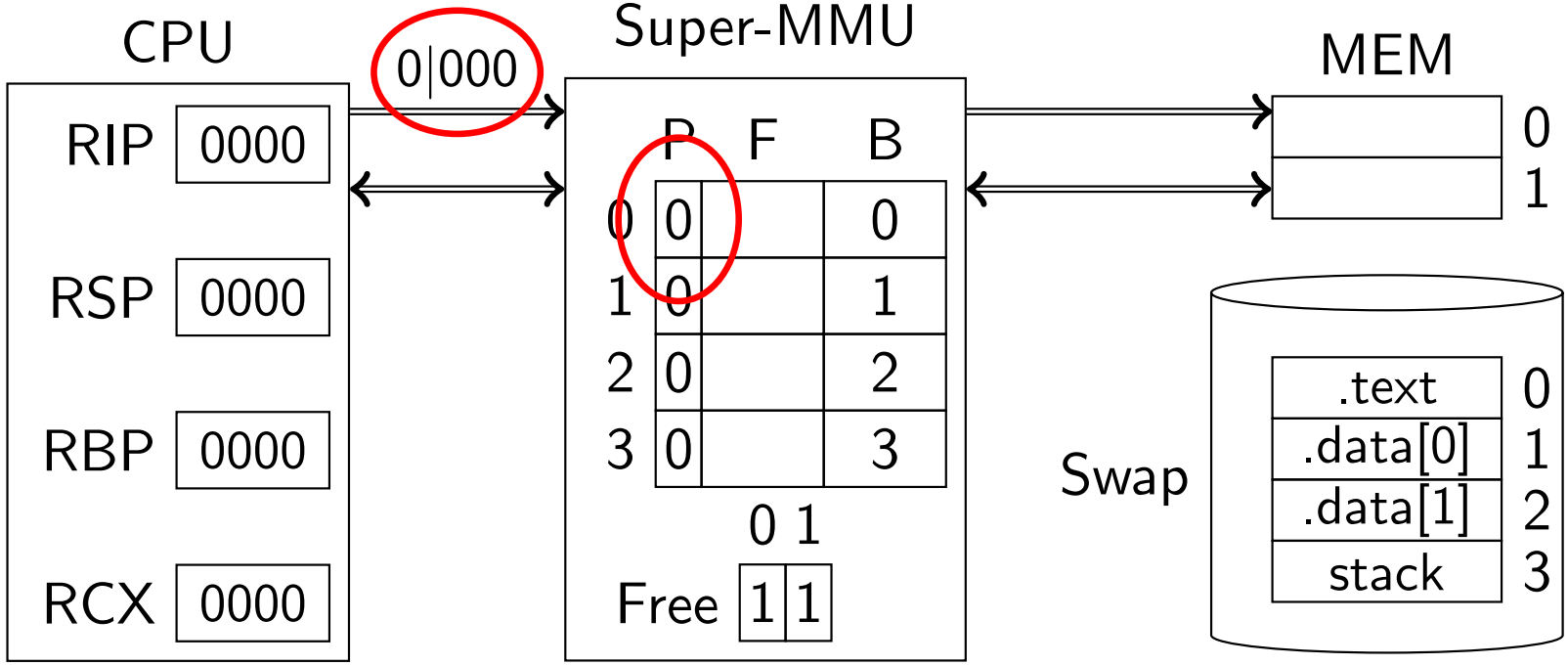
Inizialmente tutti frame sono liberi.



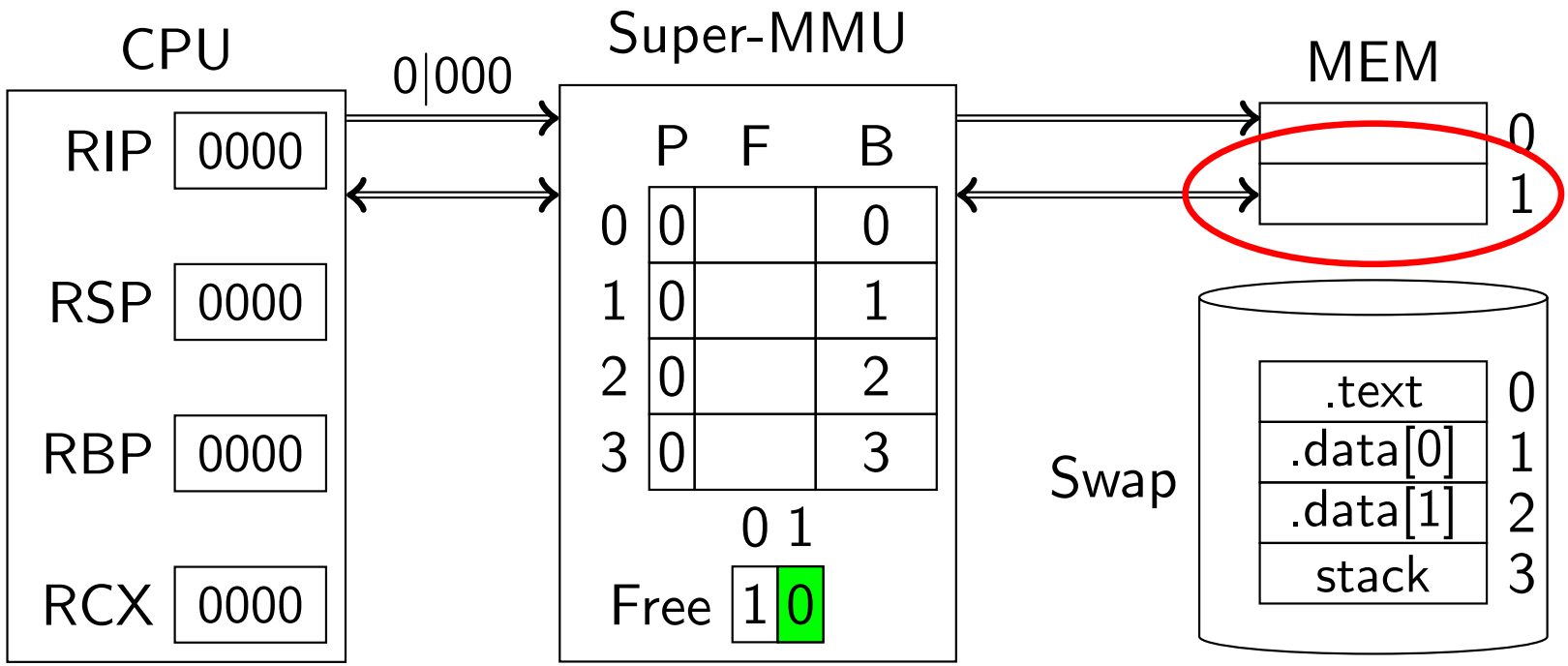
Infine, inizializziamo tutti i registri con zero.



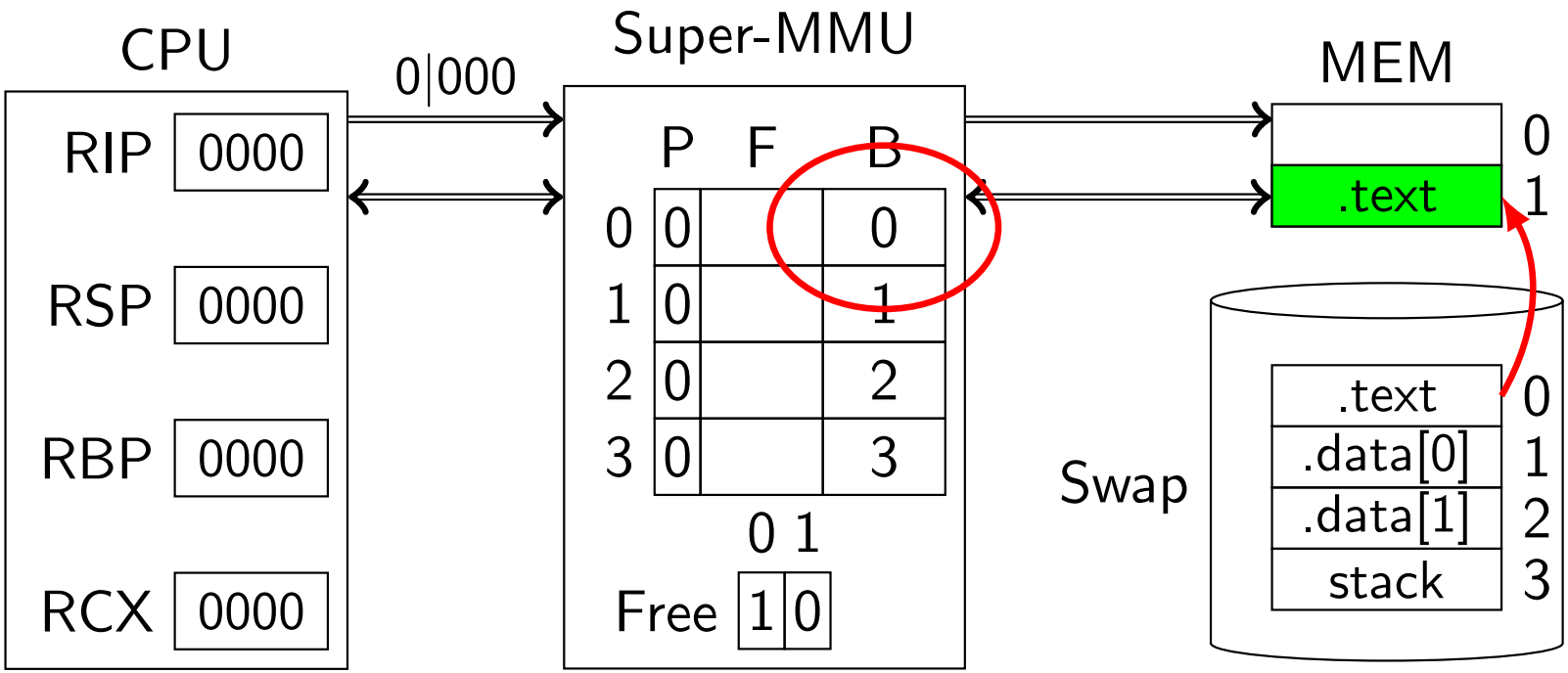
La CPU tenta di prelevare l'istruzione all'indirizzo contenuto in RIP. Inizia quindi una operazione di lettura in memoria all'indirizzo 0000.



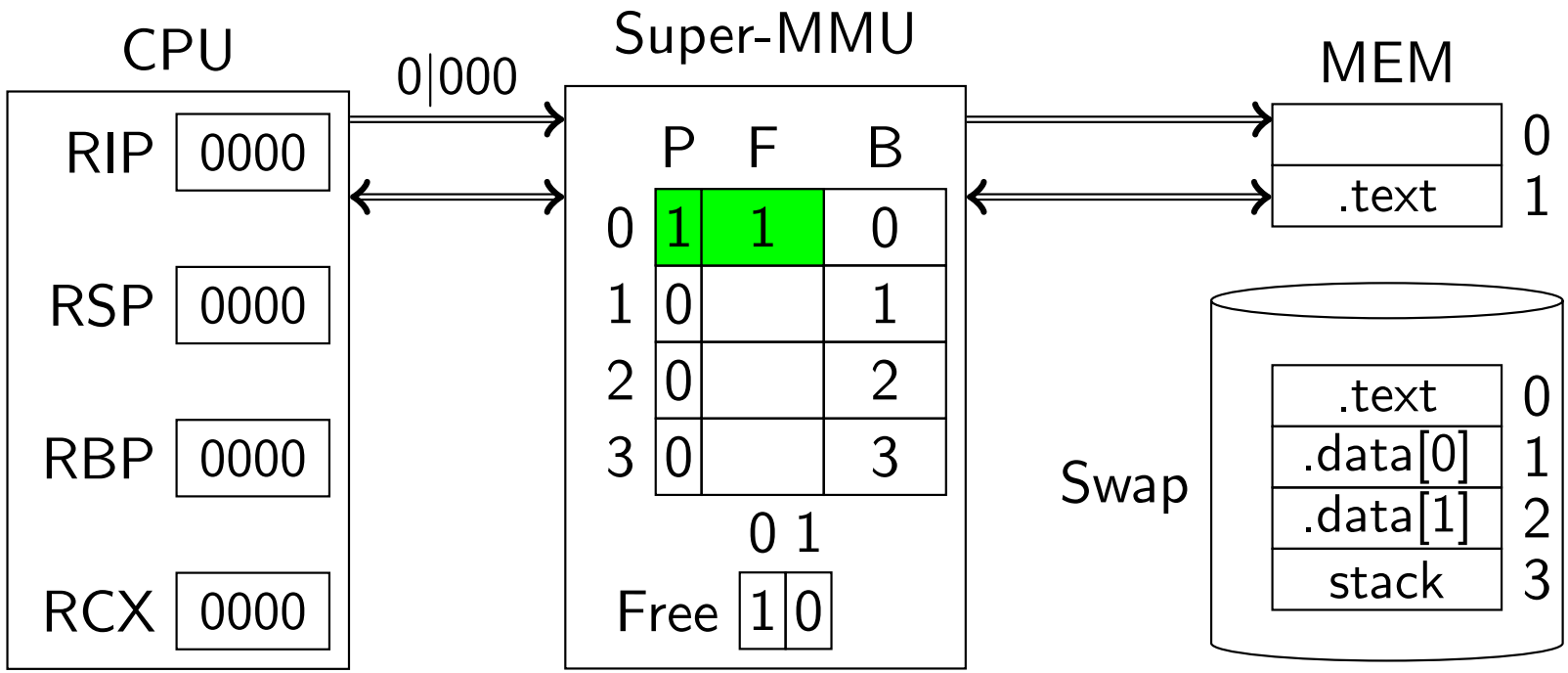
La MMU intercetta l'operazione e scompone l'indirizzo in numero di pagina (0) e offset (000). Consulta quindi l'entrata numero 0 della tabella di corrispondenza e scopre che la pagina 0 non è presente.



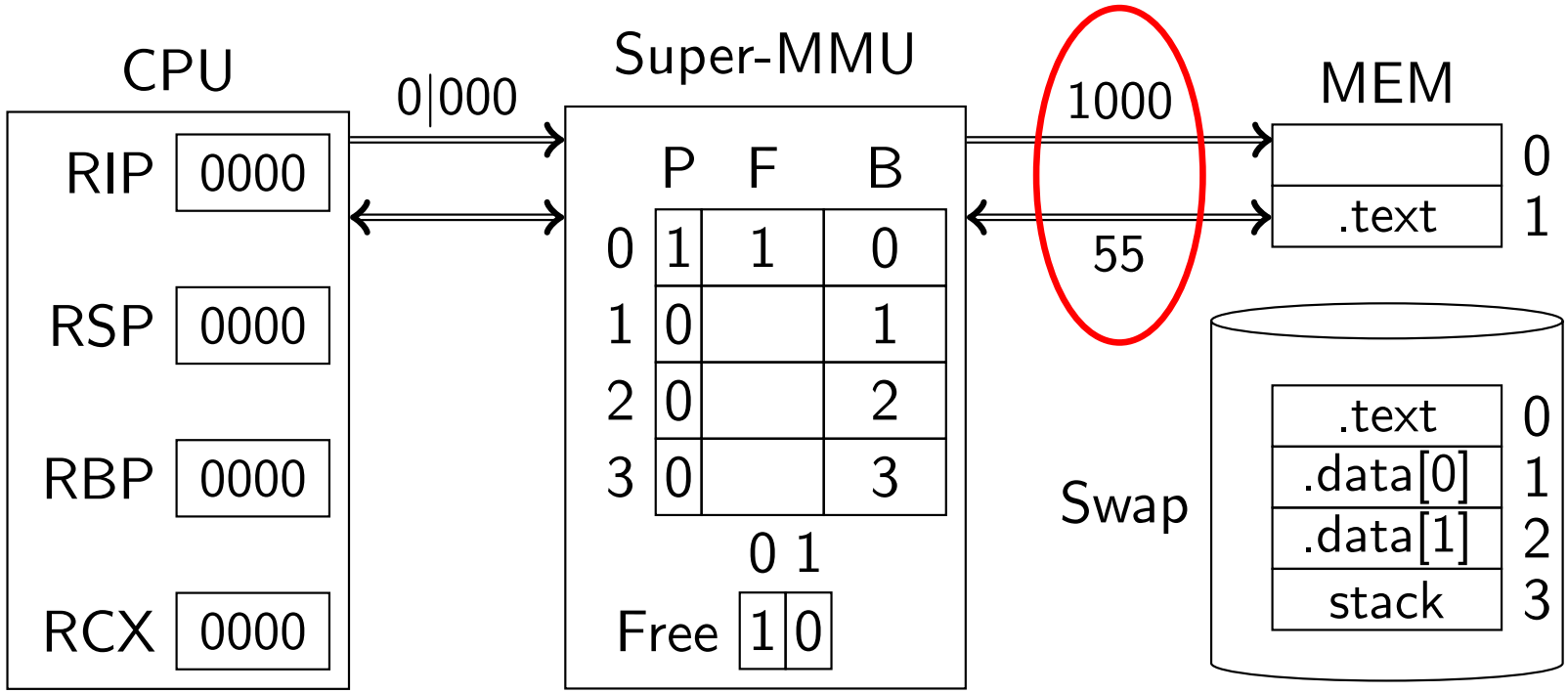
Cerca un frame non utilizzato in cui caricare la pagina 0. Supponiamo che scelga il frame numero 1. Marca il frame 1 come occupato.



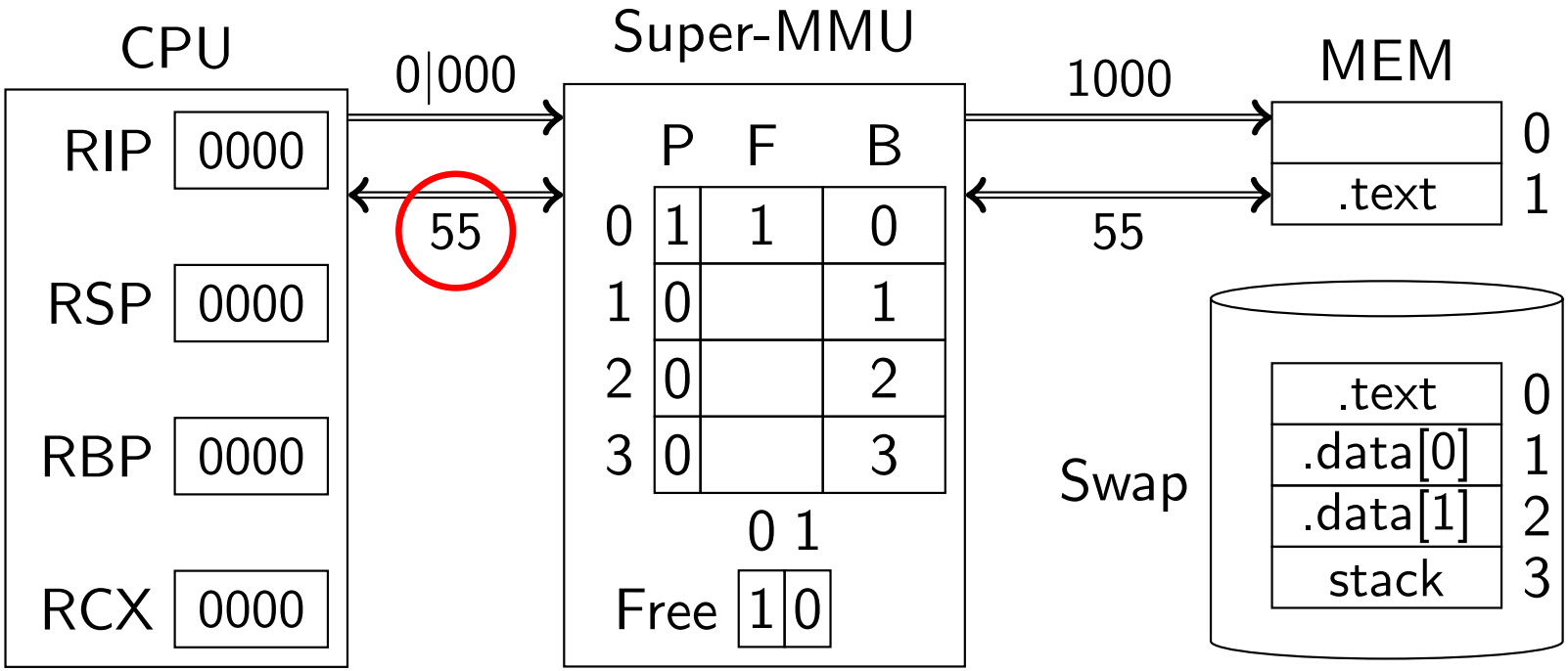
Ordina quindi un trasferimento dal blocco 0 (come specificato dal campo B della tabella) alla frame 1.



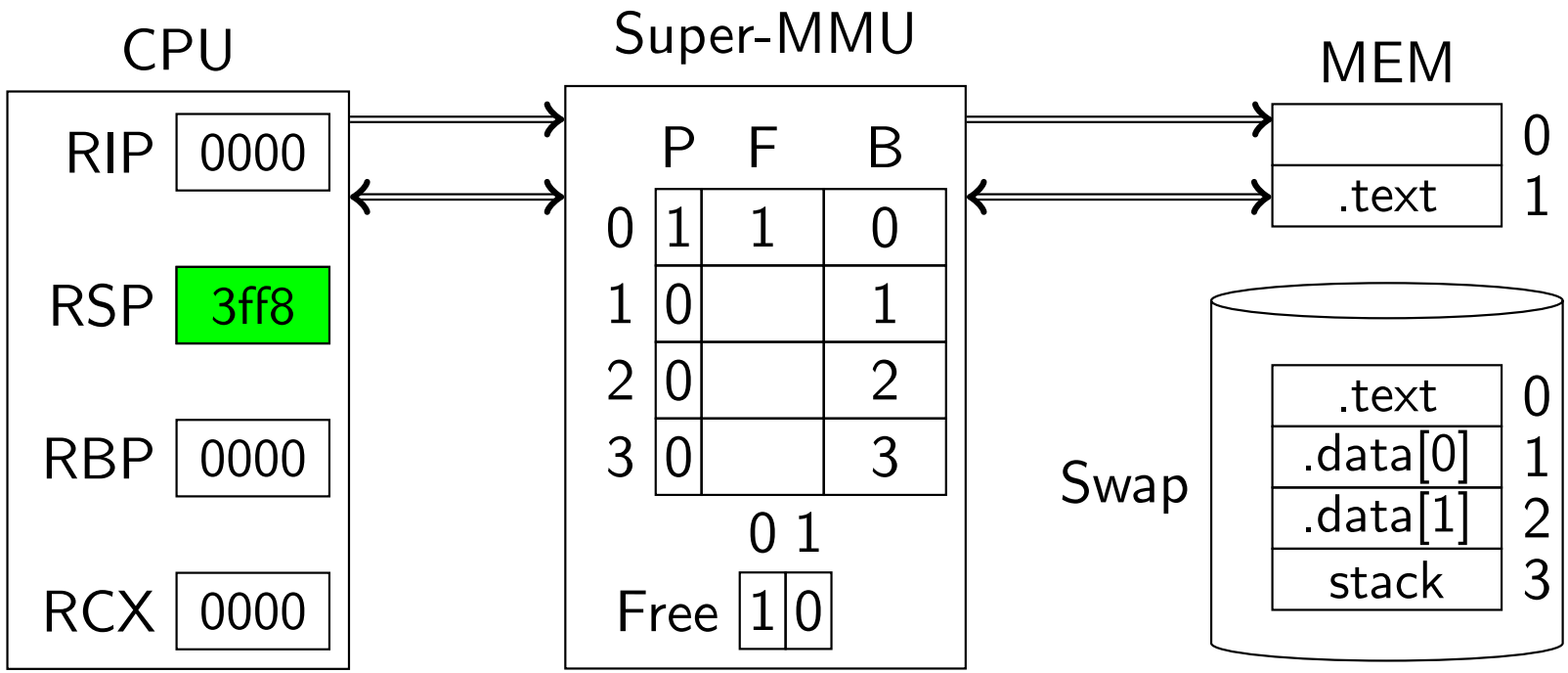
Aggiorna la tabella di corrispondenza: ora la pagina virtuale 0 è presente ($P = 1$) e si trova nel frame 1 ($F = 1$).



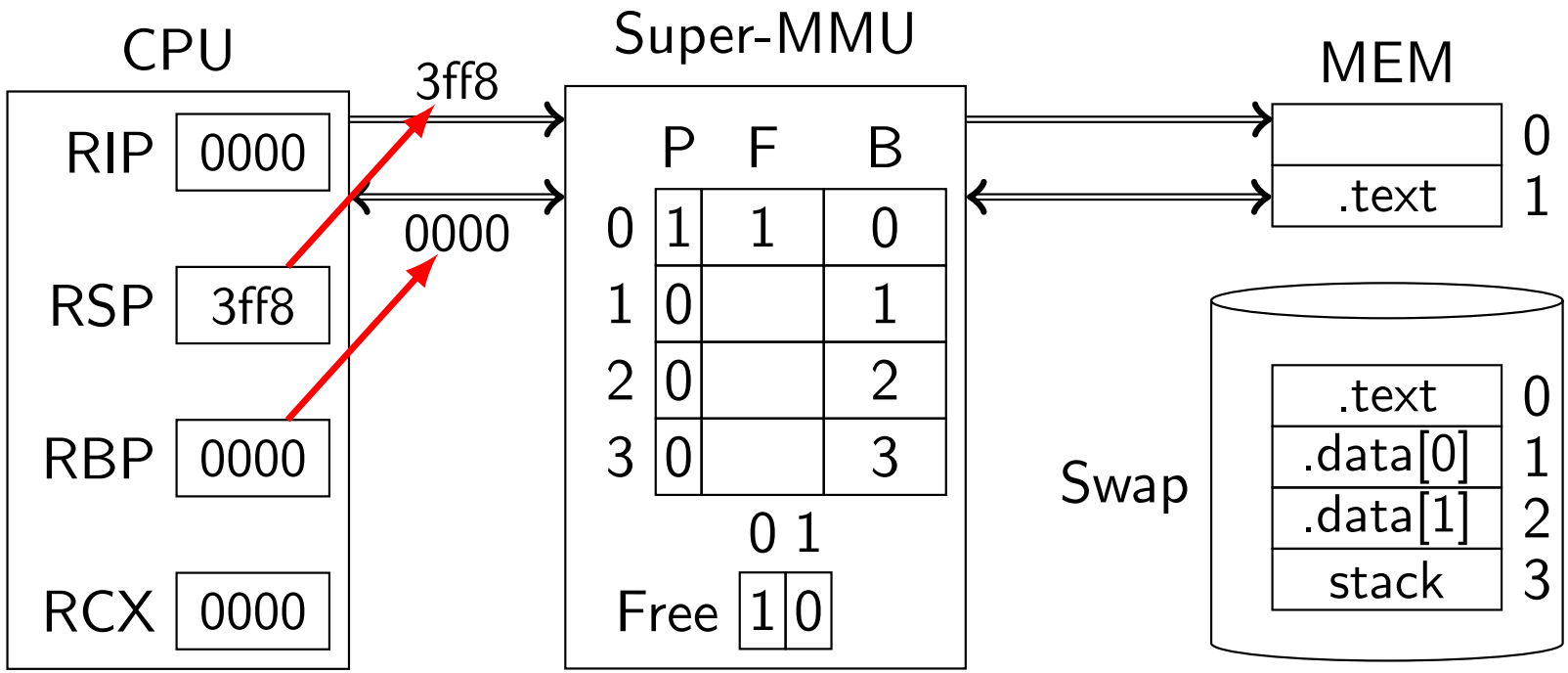
Completa infine l'accesso, dopo aver trasformato l'indirizzo. Per tutto questo tempo la CPU è stata ferma in attesa del completamento dell'operazione.



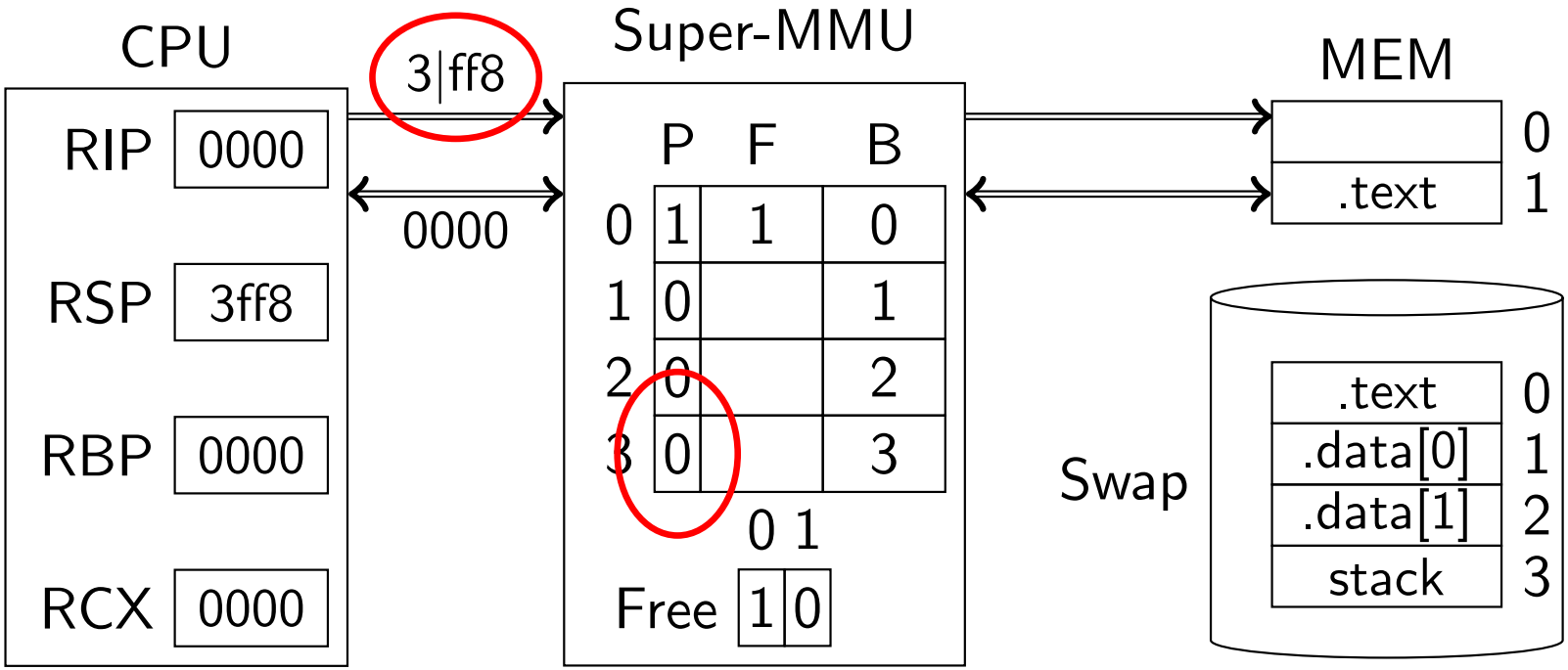
L'operazione di lettura restituisce l'istruzione `pushq %rbp`, che la CPU inizia ad eseguire.



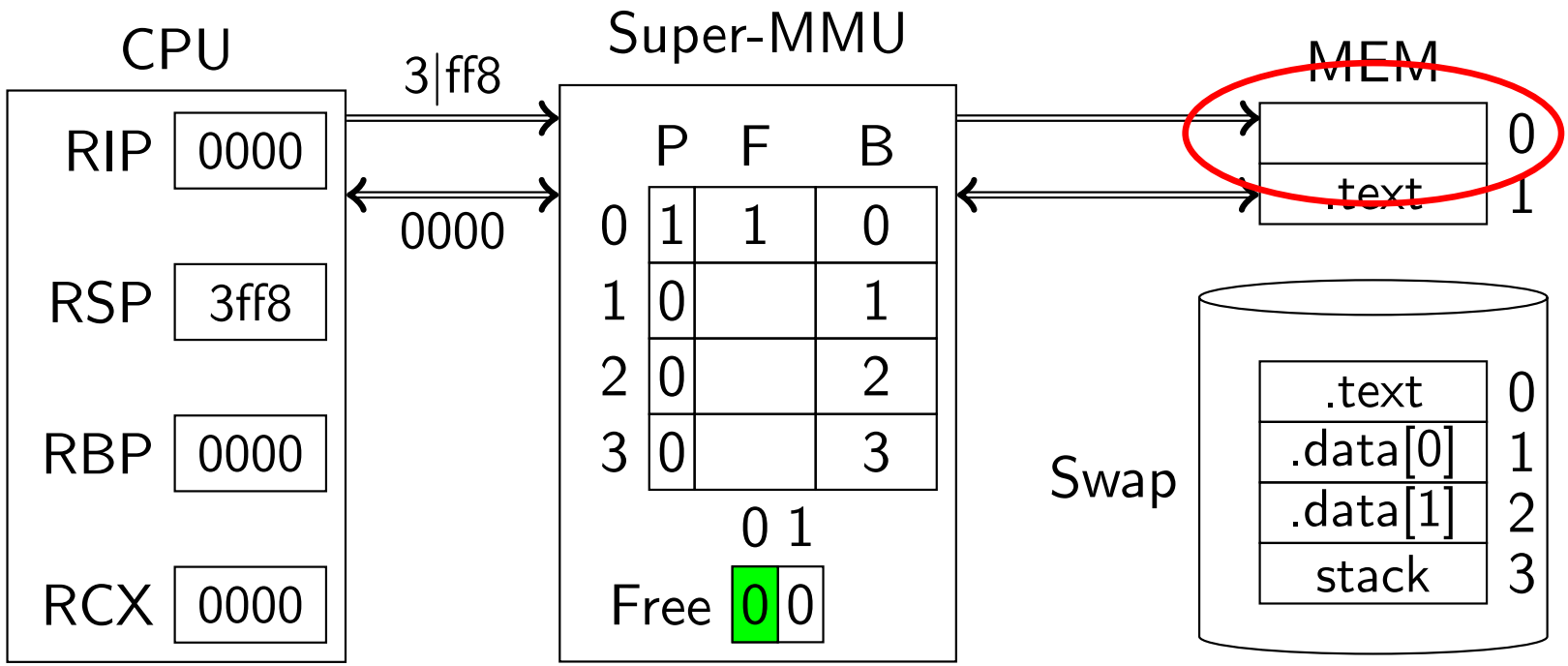
Il primo passo è decrementare RSP di 8.



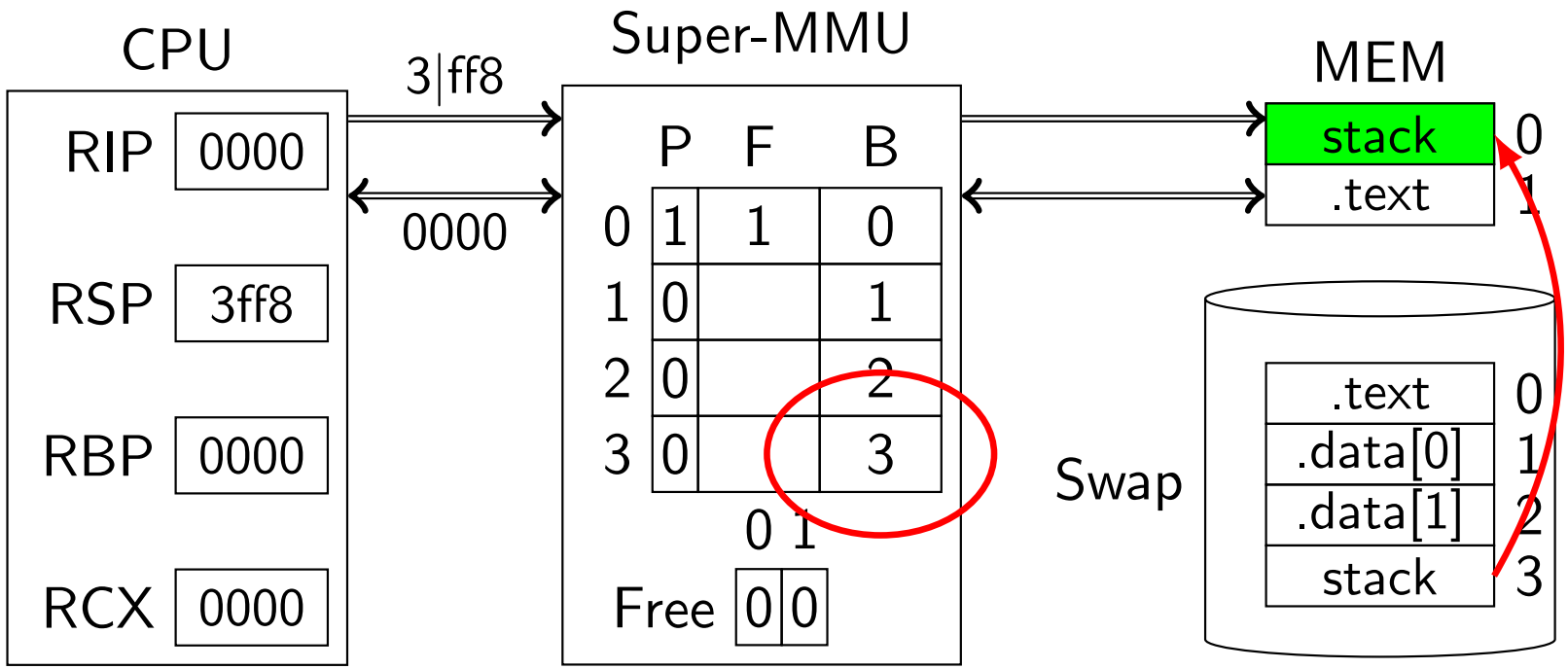
Il secondo passo è scrivere il contenuto di RBP all'indirizzo contenuto in RSP. La CPU inizia quindi una operazione di scrittura all'indirizzo 3ff8.



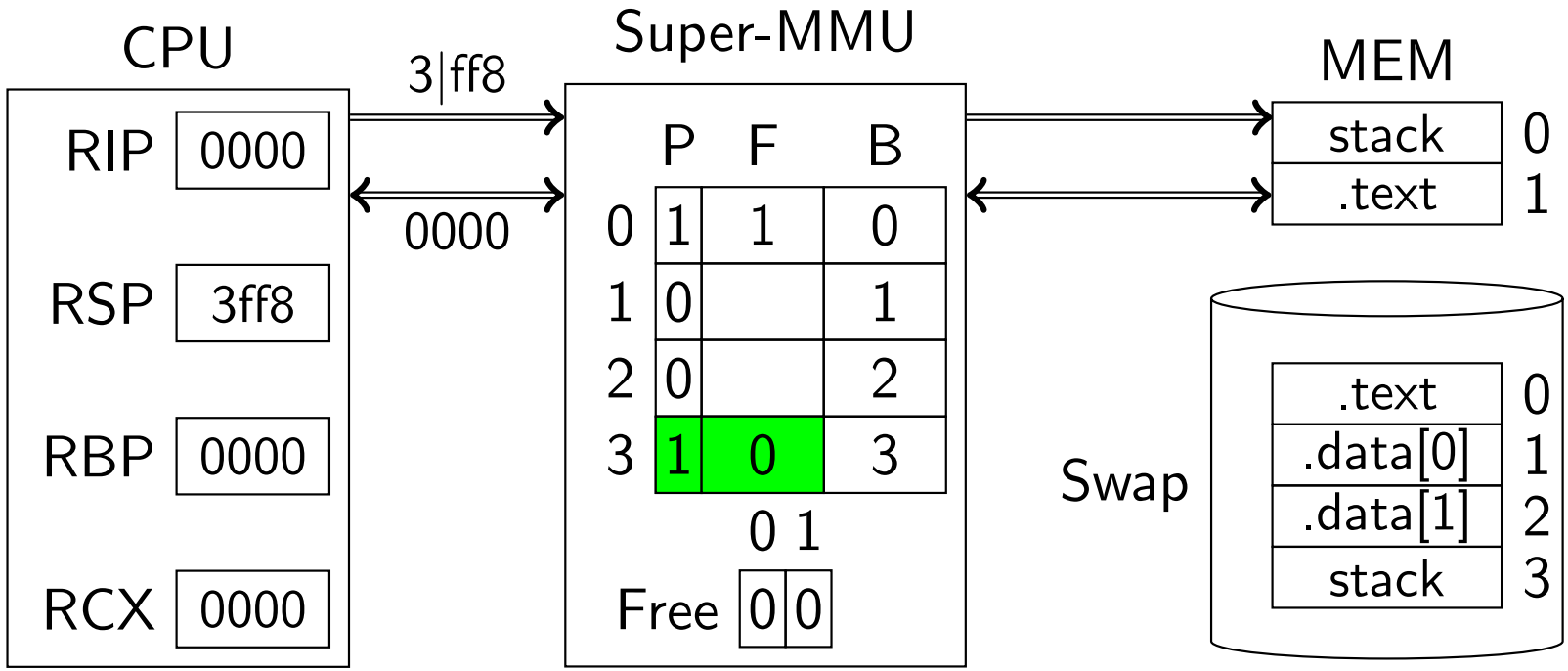
La MMU intercetta l'operazione e scompone l'indirizzo in numero di pagina (3) e offset (ff8). Consulta quindi l'entrata numero 3 della tabella di corrispondenza e scopre che anche la pagina 3 non è presente.



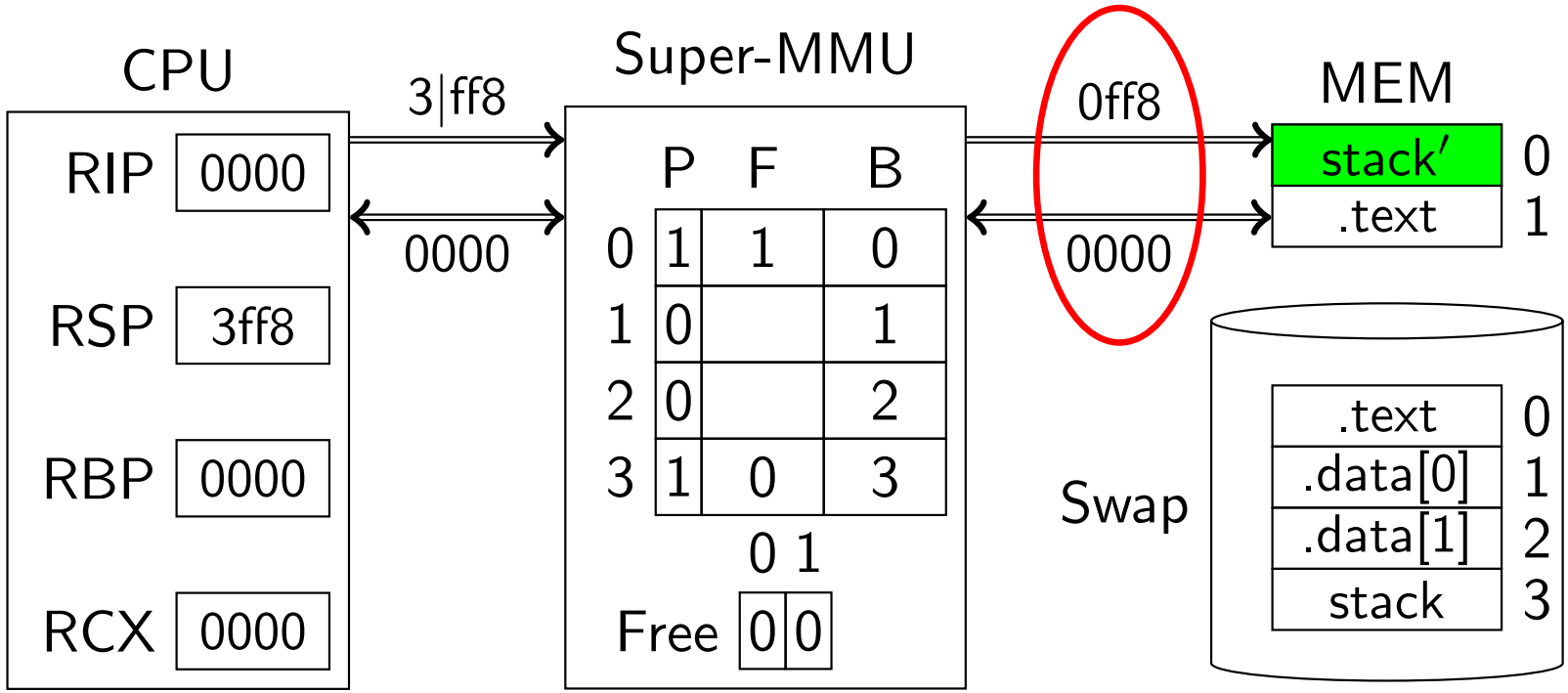
Cerca allora un frame non utilizzato in cui caricare la pagina 3. È rimasto solo il frame 0, quindi sceglie quello.



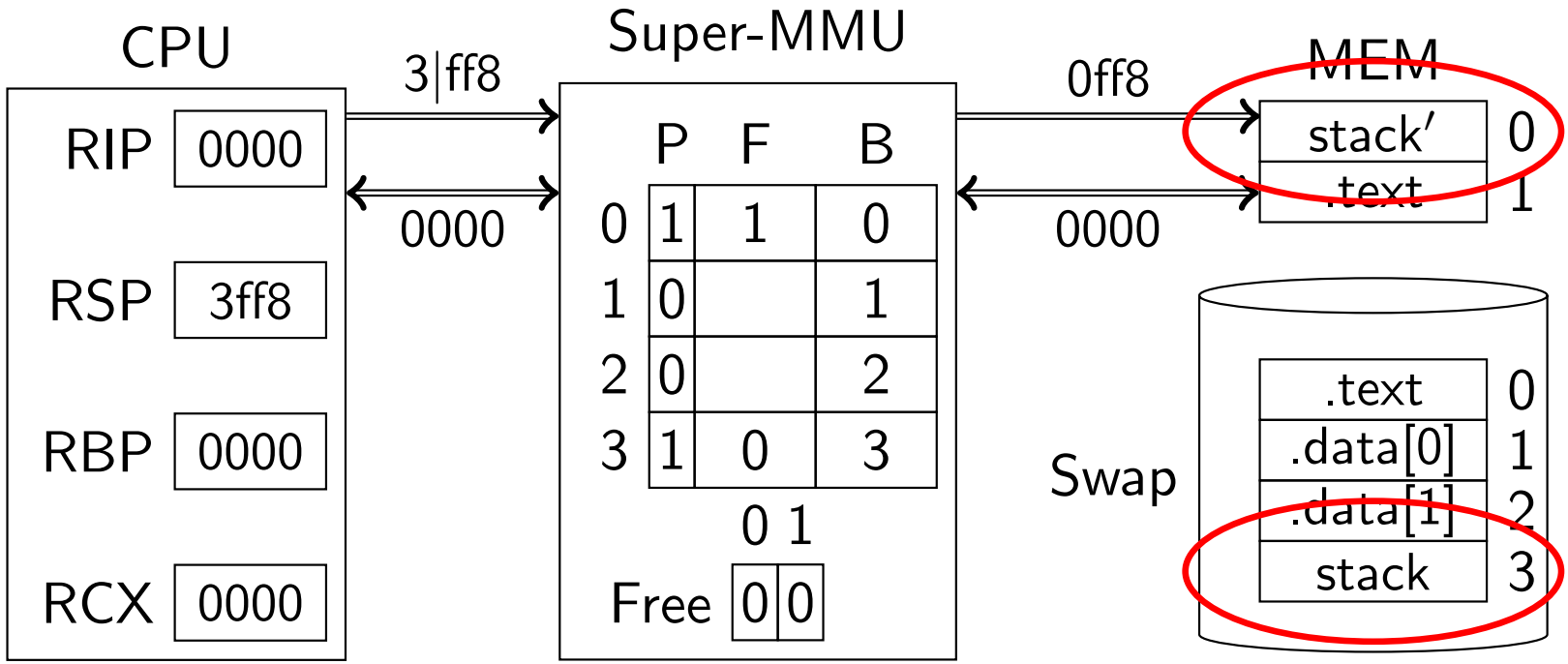
Ordina un trasferimento dal blocco 3 (campo B della riga 3) al frame 0.



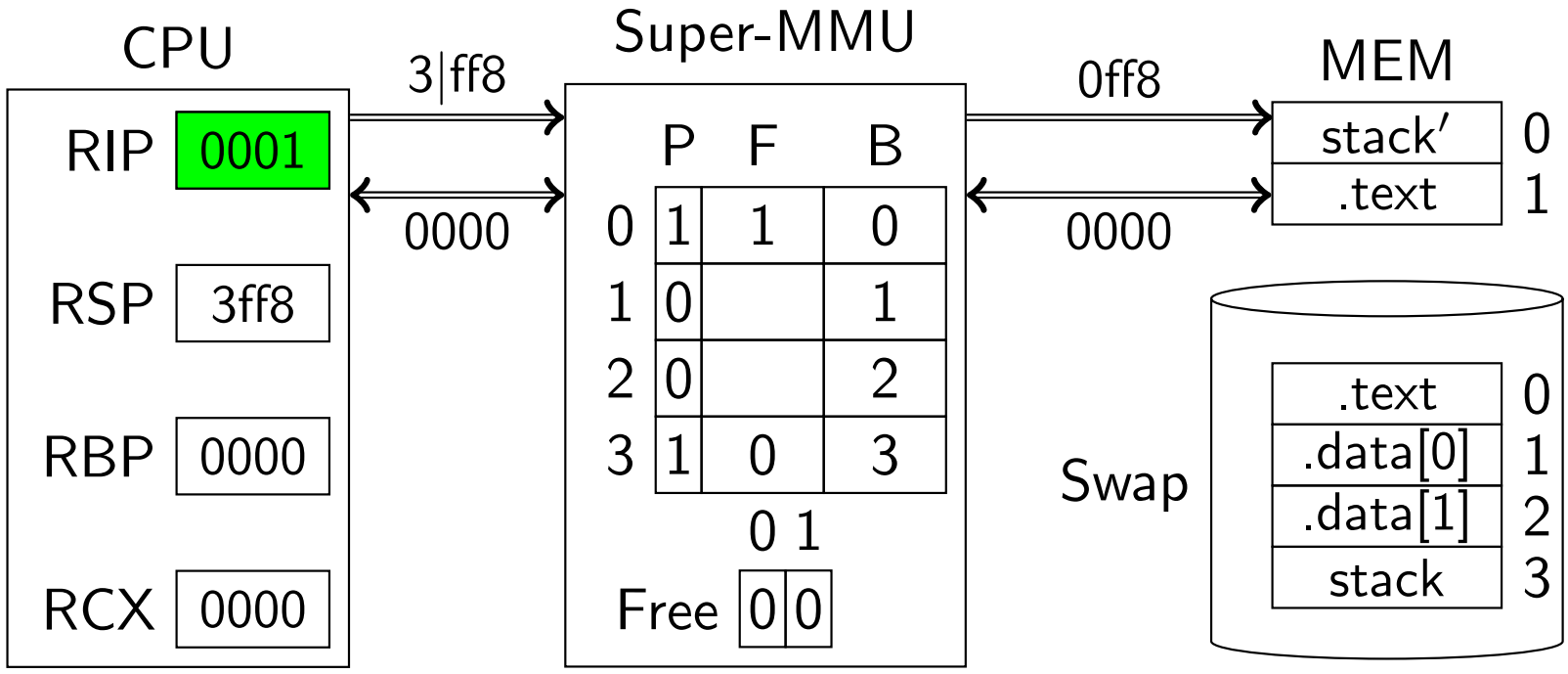
Alla fine del trasferimento aggiorna la tabella di corrispondenza: ora la pagina virtuale 3 è presente (P = 1) e si trova nel frame 0 (F = 0).



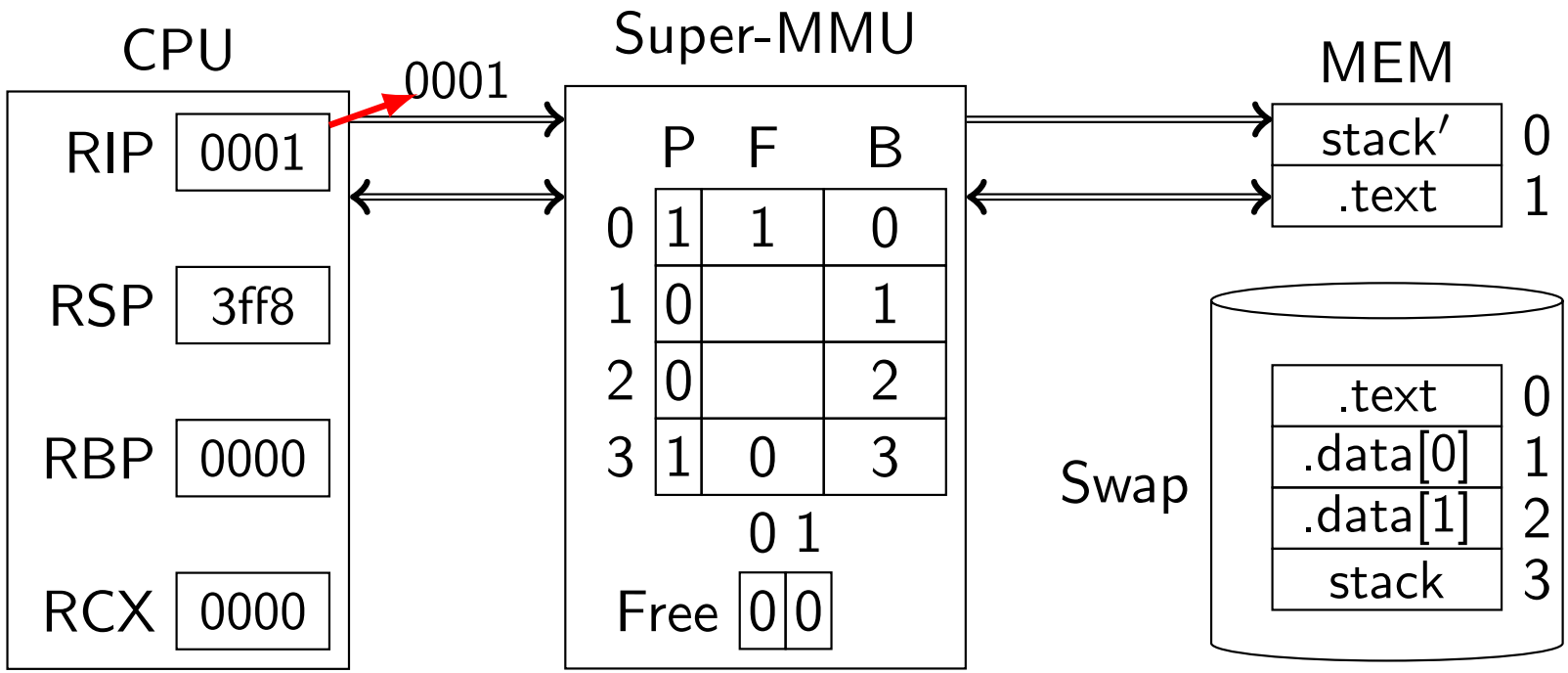
Infine completa l'operazione di scrittura, dopo aver trasformato l'indirizzo.



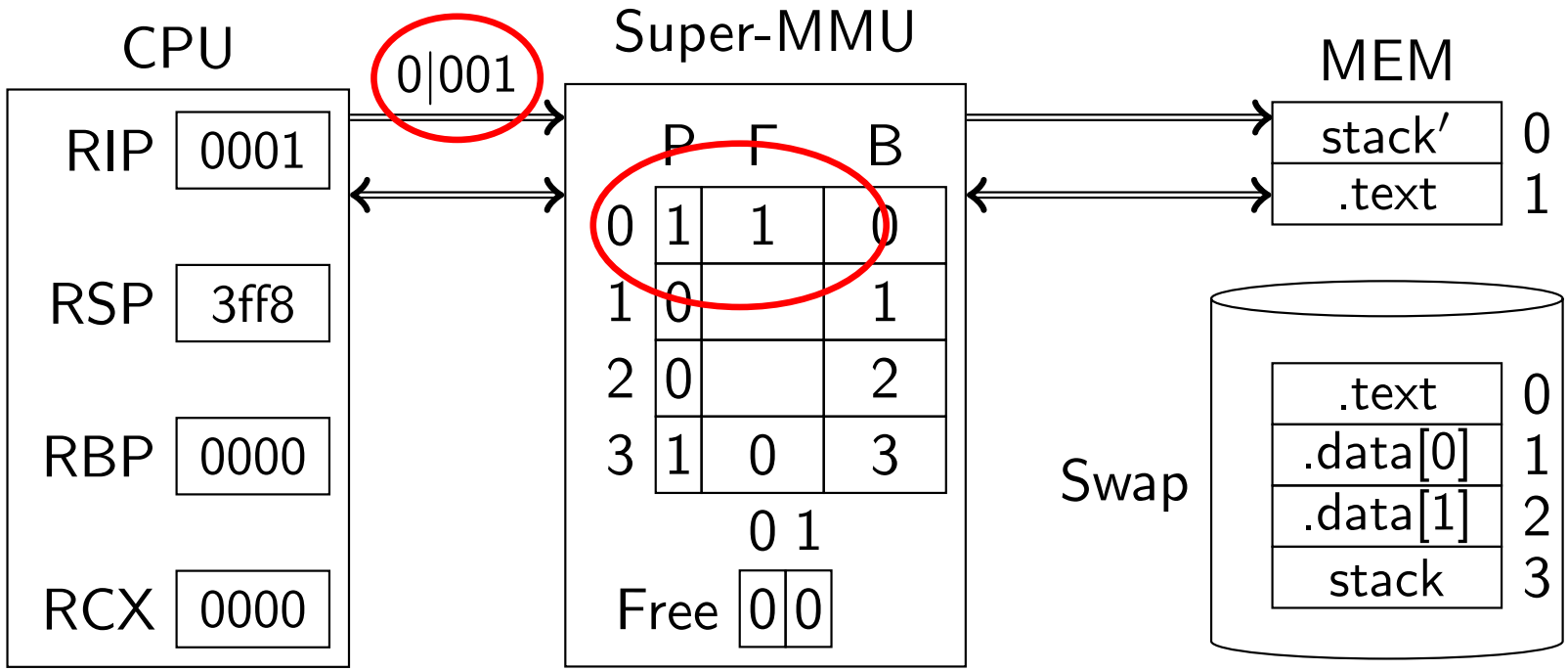
Si noti che dopo la scrittura il contenuto dello stack sarà cambiato rispetto a quello che è rimasto memorizzato nell'area di swap.



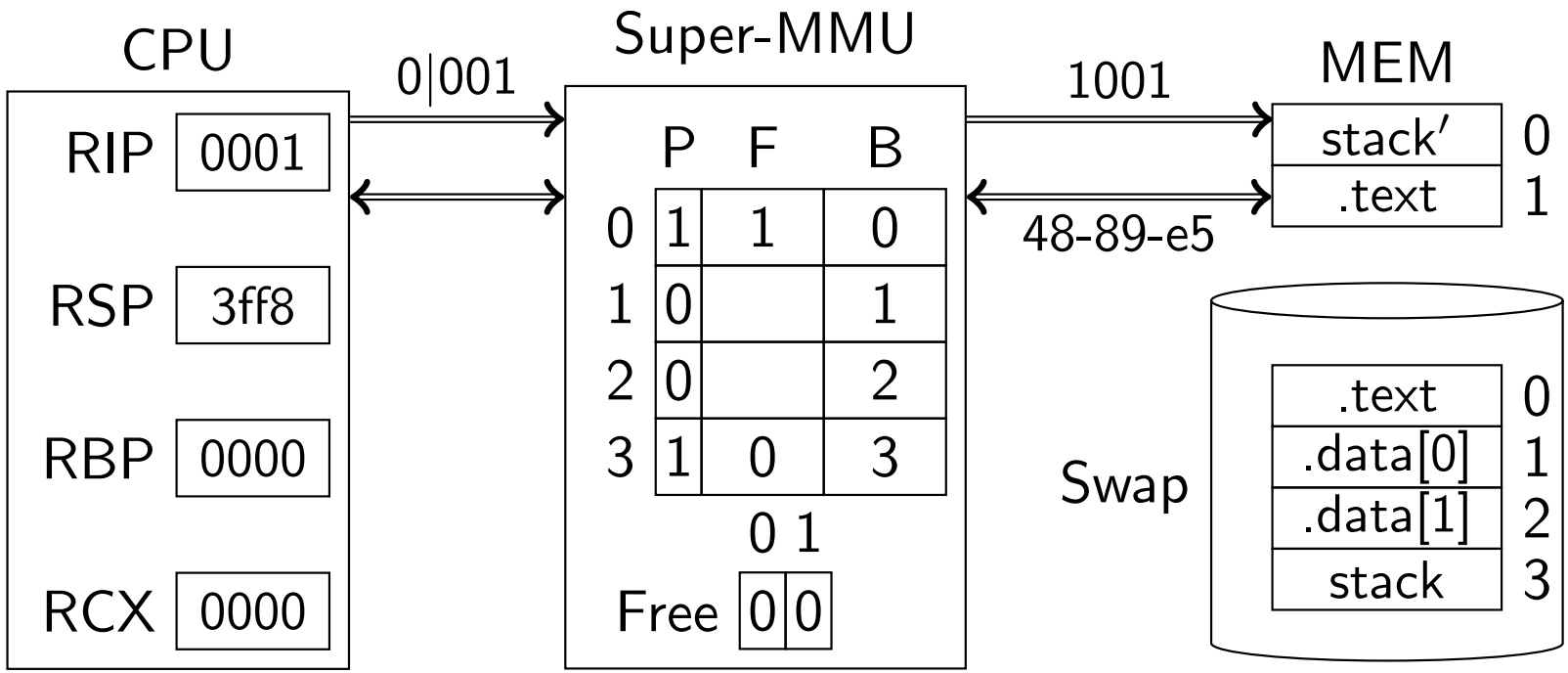
La CPU ha completato l'esecuzione dell'istruzione `pushq %rbp` e può passare alla successiva. Incrementa RIP di 1 (che è la dimensione dell'istruzione appena terminata).



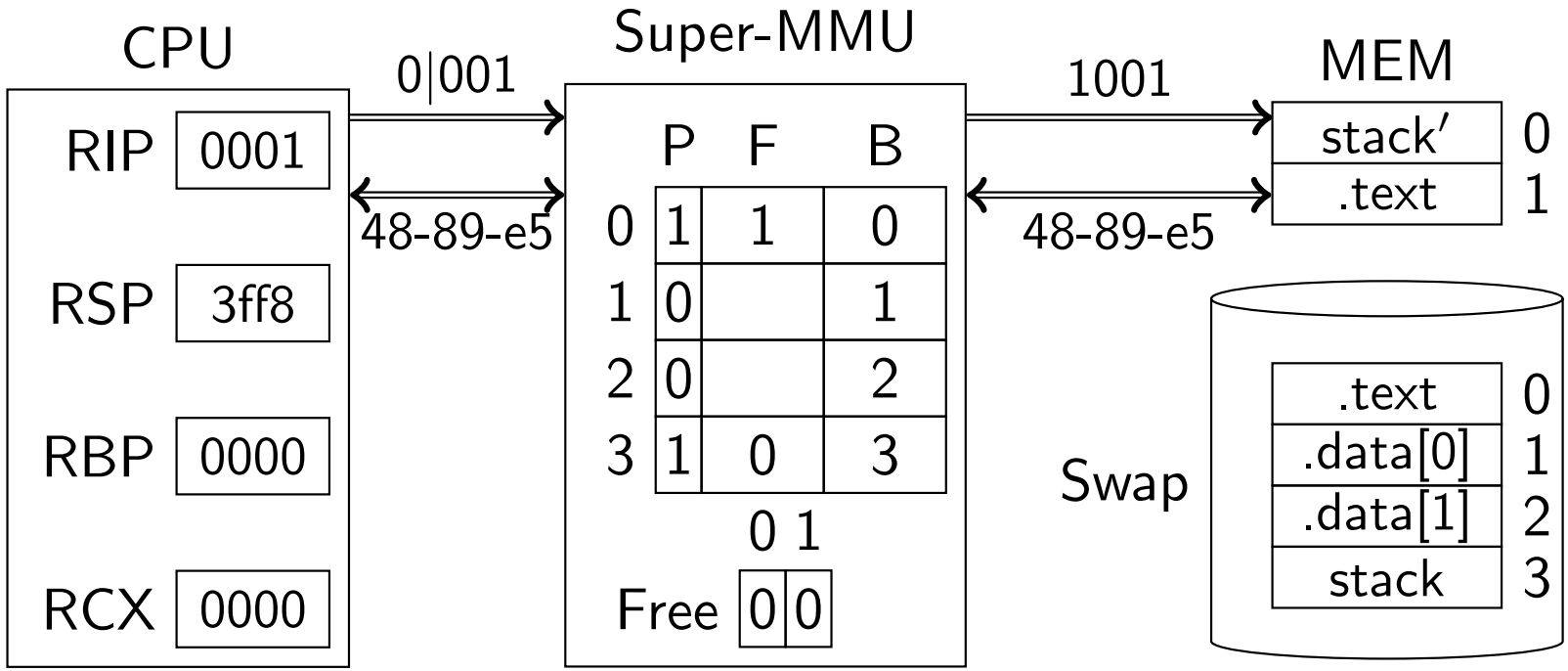
Nuovo ciclo: la CPU tenta di prelevare l'istruzione all'indirizzo contenuto in RIP. Inizia una operazione di lettura in memoria all'indirizzo 0001.



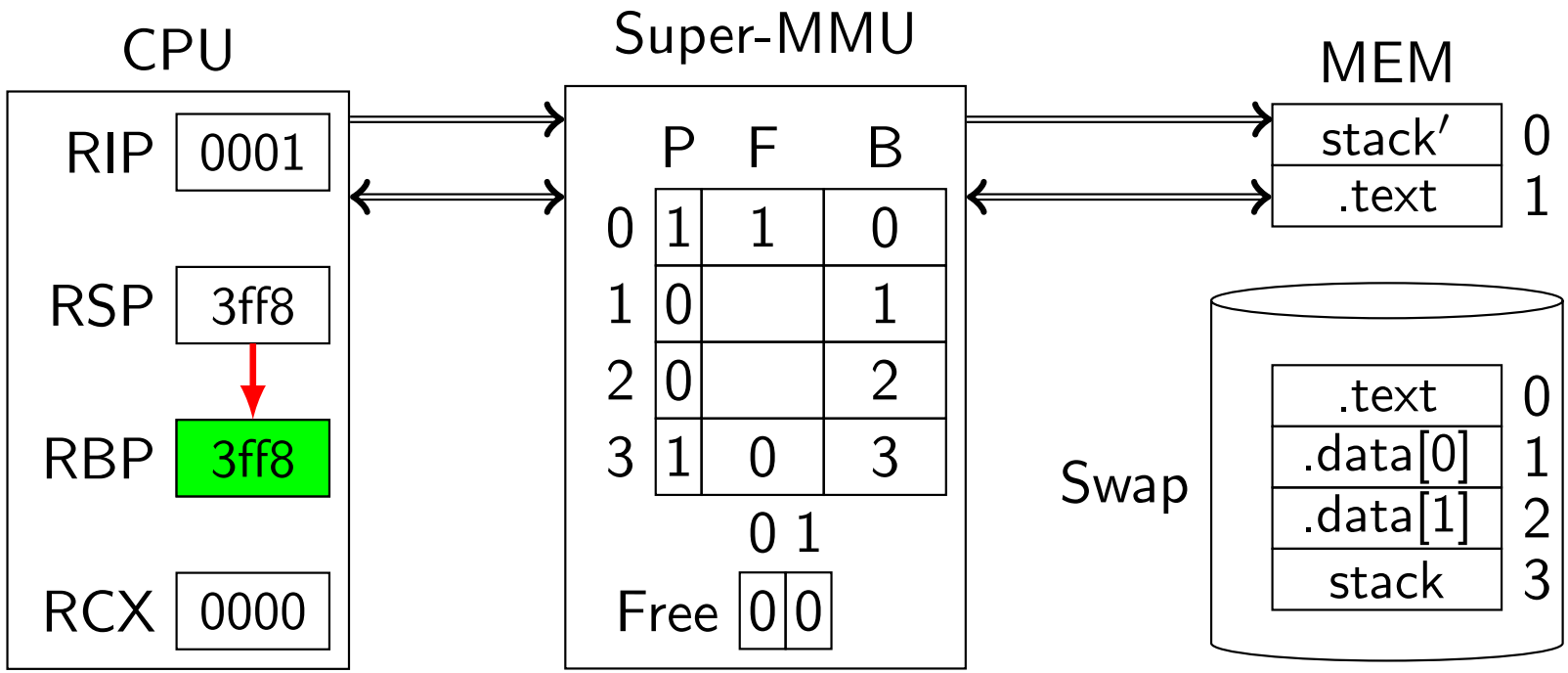
Come al solito, la MMU intercetta l'operazione e scompone l'indirizzo in numero di pagina virtuale (0) e offset (001). Questa volta, però, scopre che la pagina virtuale 0 è già presente.



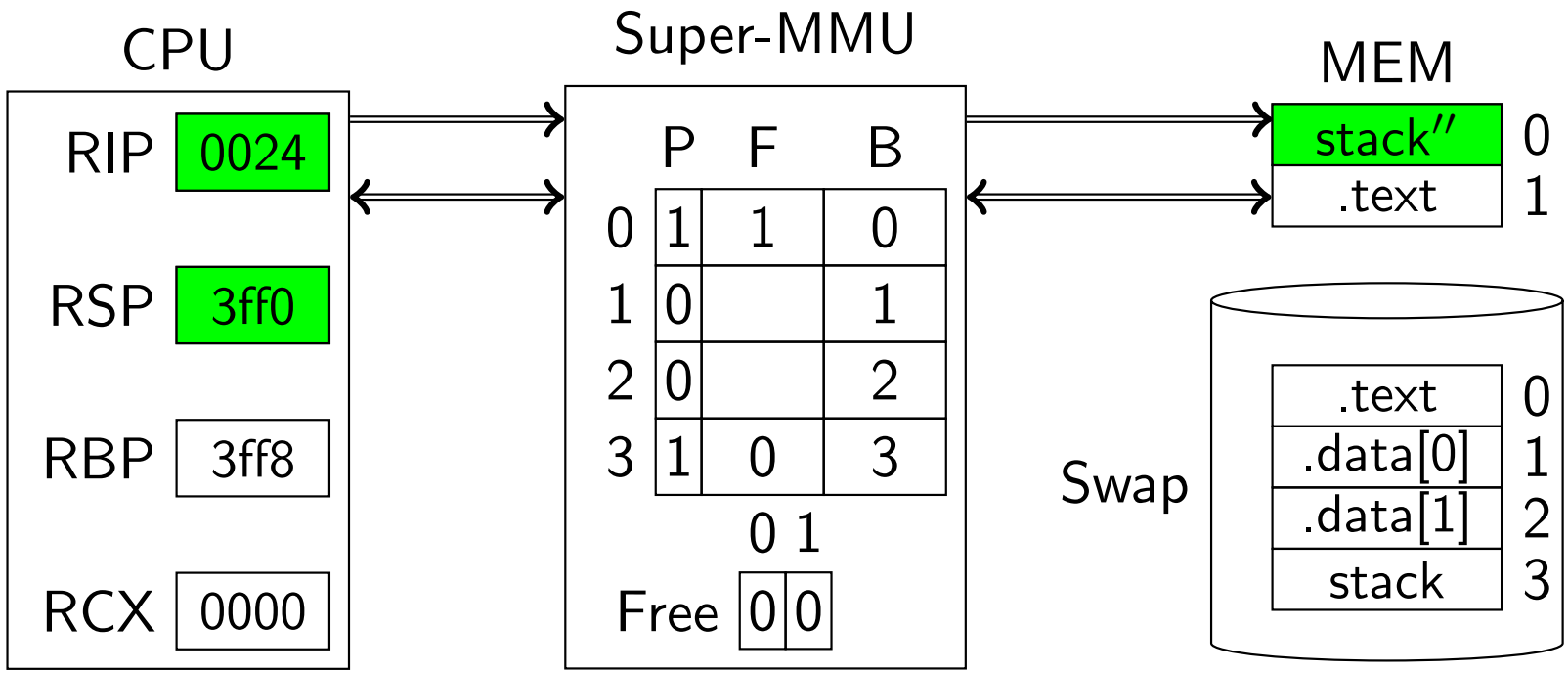
Può dunque completare subito l'accesso, ovviamente dopo aver trasformato l'indirizzo.



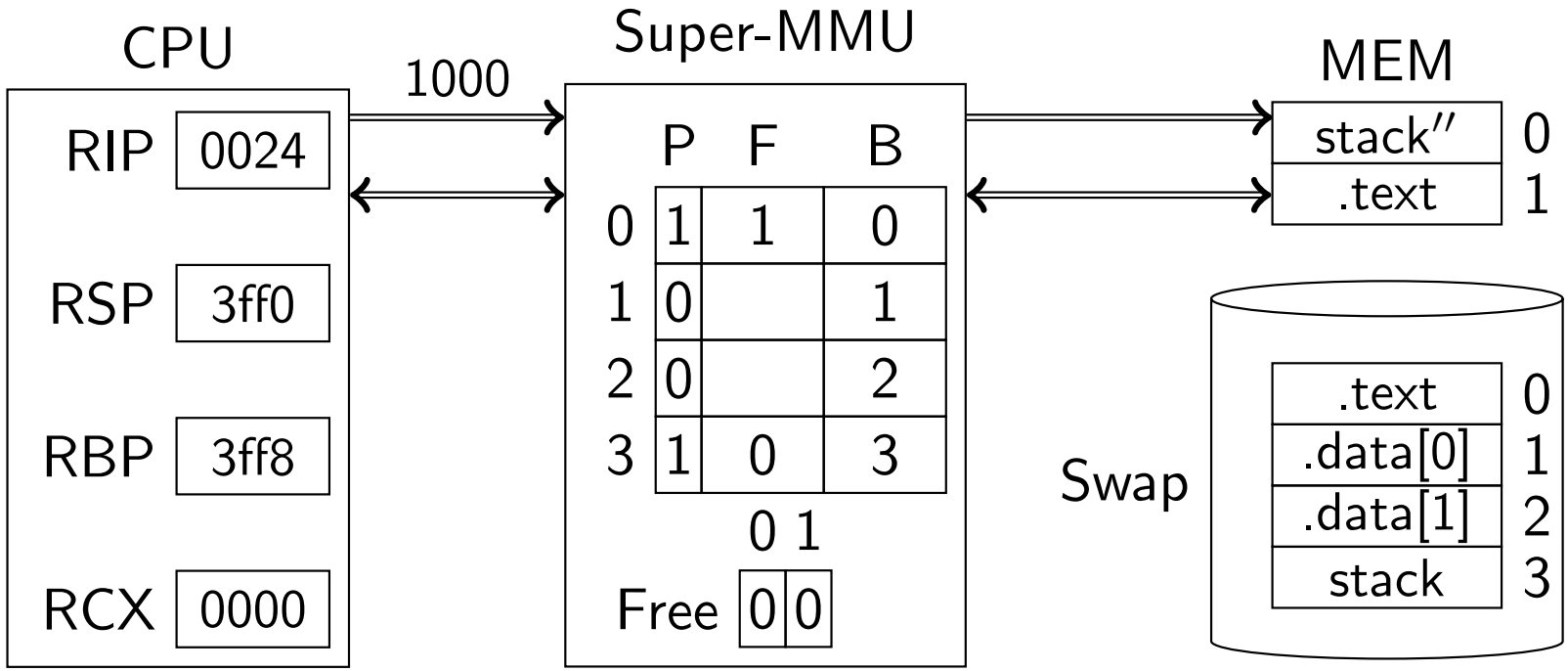
La CPU riceve l'istruzione `movq %rsp, %rbp`.



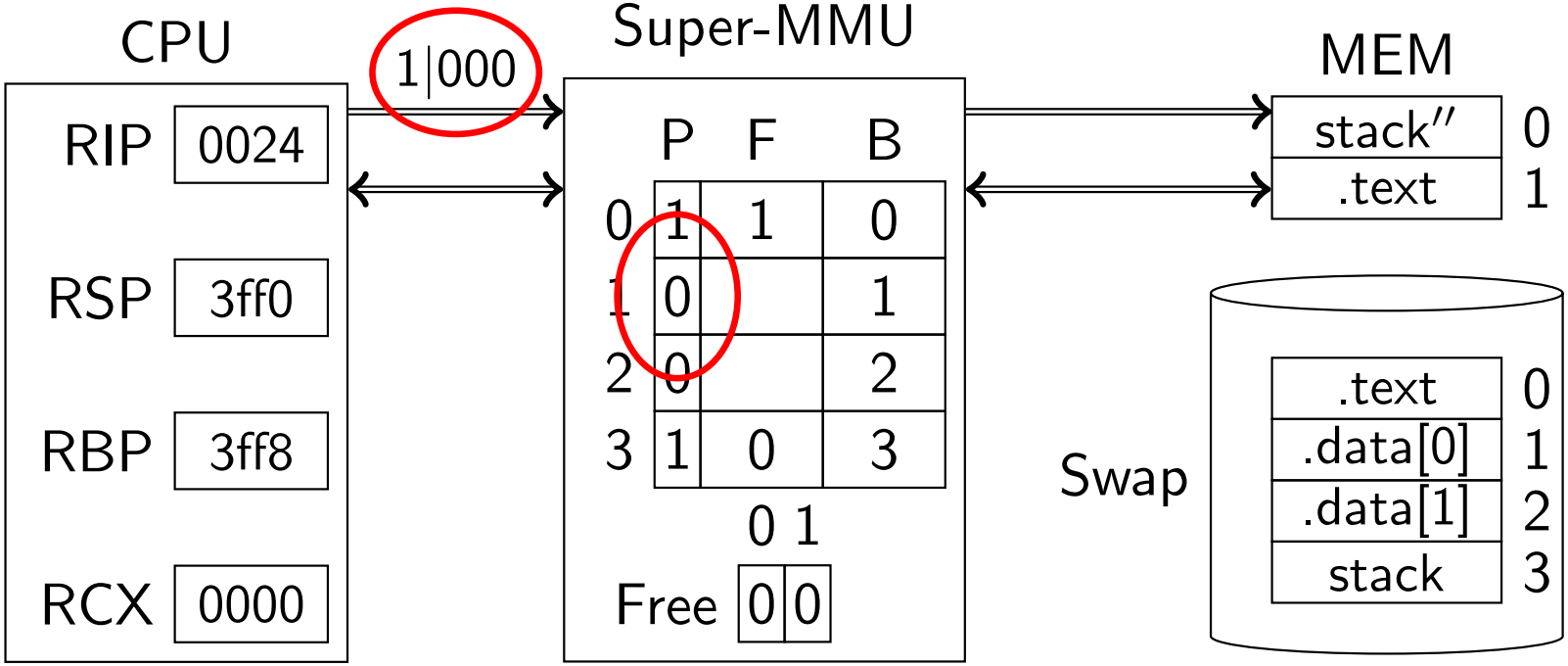
Per eseguirla copia il contenuto di RSP in RBP.
L'istruzione non prevede accessi in memoria.



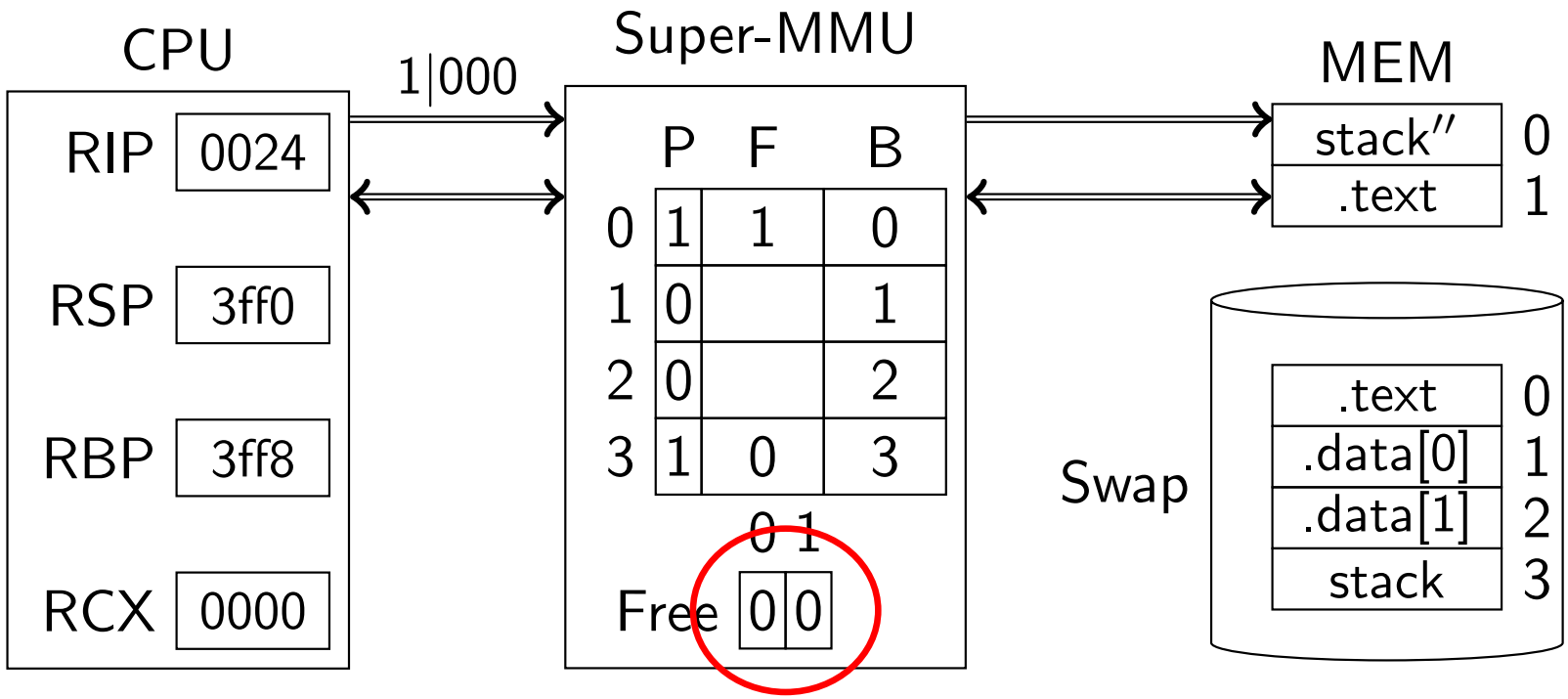
Dopo alcune istruzioni, l'esecuzione del programma arriva all'istruzione `movsbl buf(%rcx), %rax` che si trova all'indirizzo (virtuale) 0024. Questa è la prima istruzione che accede alla sezione `.data` del programma.



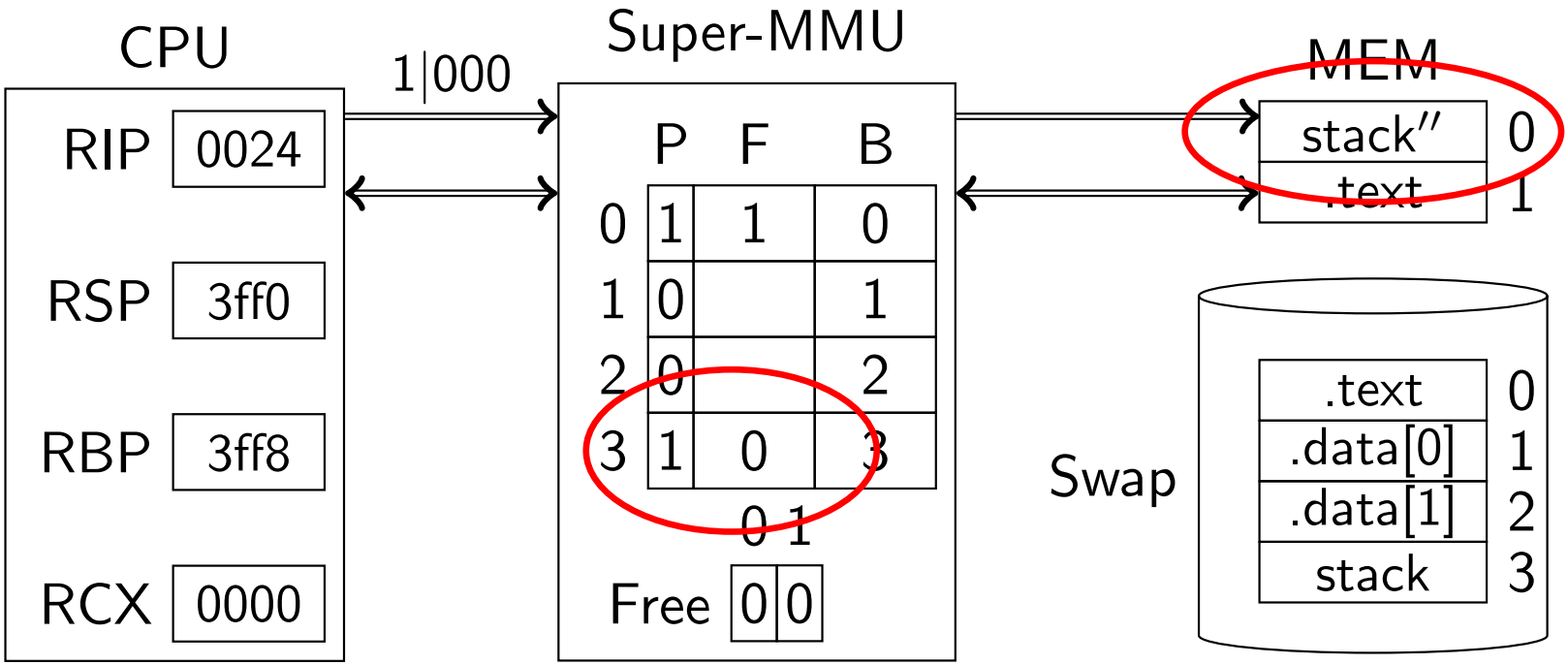
La CPU somma il contenuto di RCX e la costante 1000 (buf nel sorgente), ottenendo l'indirizzo 1000. Inizia dunque una operazione di lettura a questo indirizzo.



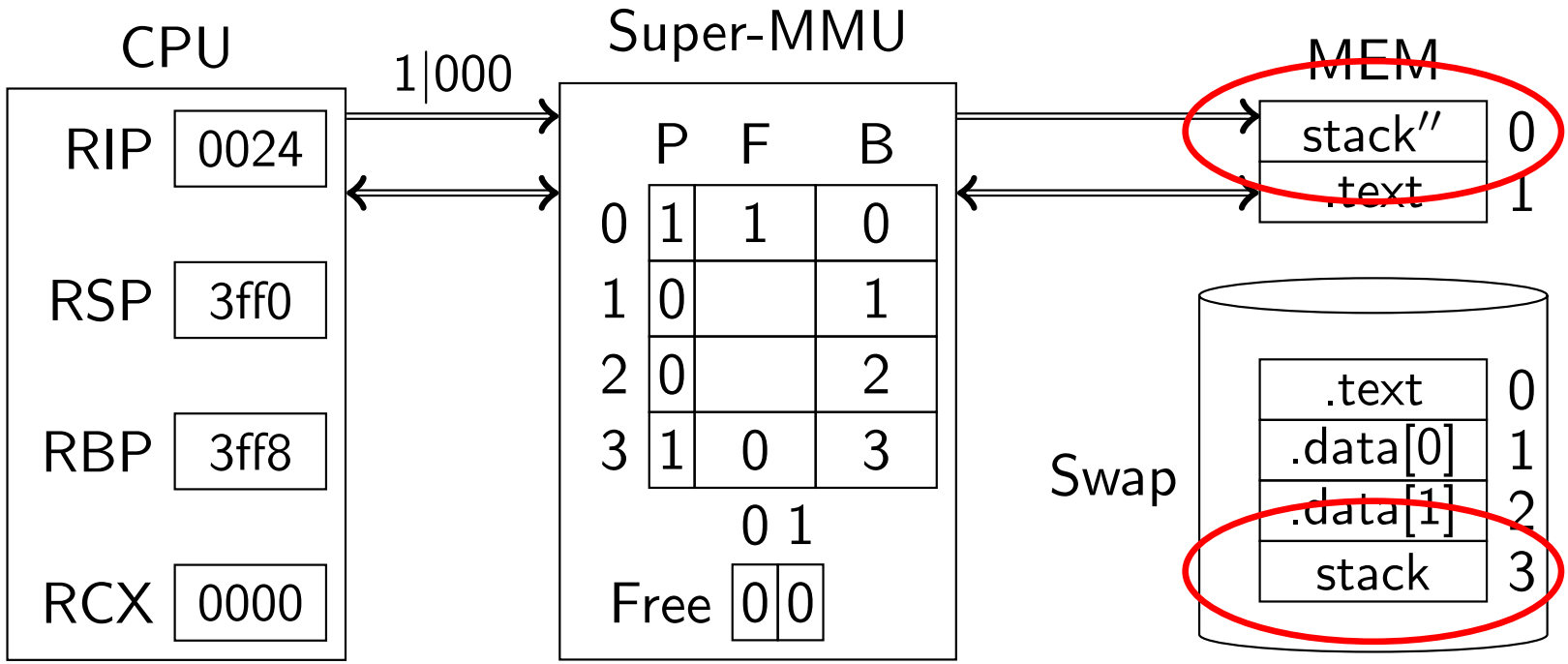
Ancora una volta la MMU intercetta l'operazione e scompone l'indirizzo in numero di pagina (1) e offset (000). L'entrata numero 1 della tabella di corrispondenza dice che la pagina 1 non è presente.



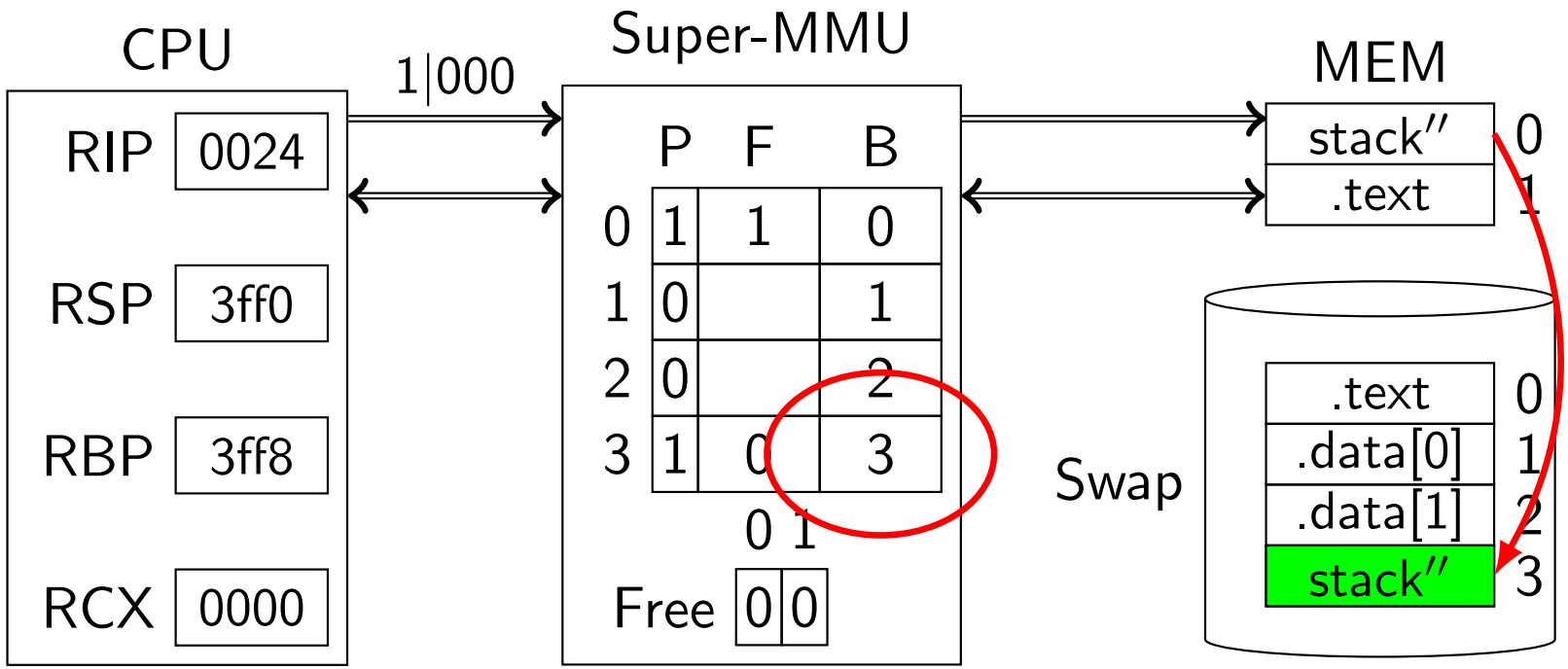
Questa volta, però, tutti i frame sono occupati. Per caricare la pagina richiesta dal processore è necessario eliminare una di quelle già presenti.



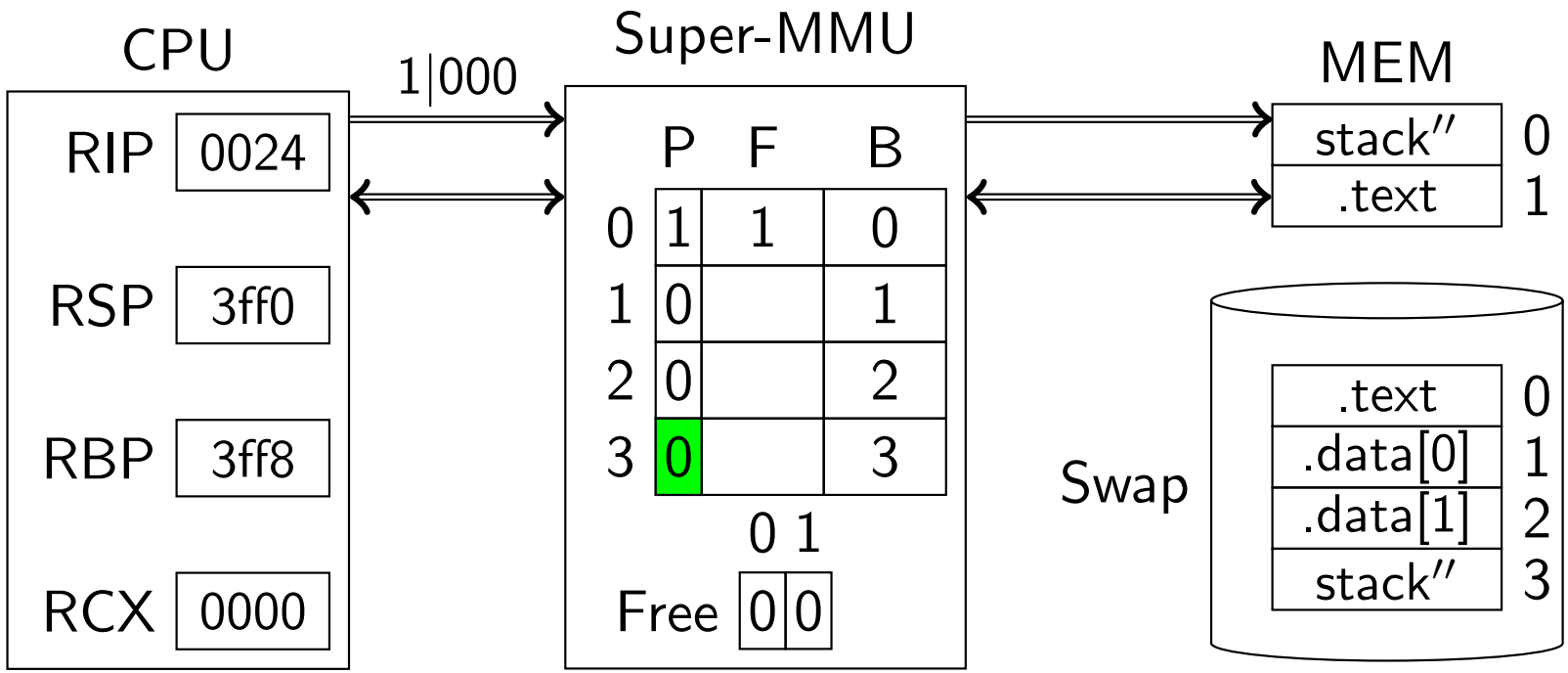
Supponiamo che la MMU scelga di riutilizzare il frame 0, in questo momento occupato dalla pagina virtuale numero 3.



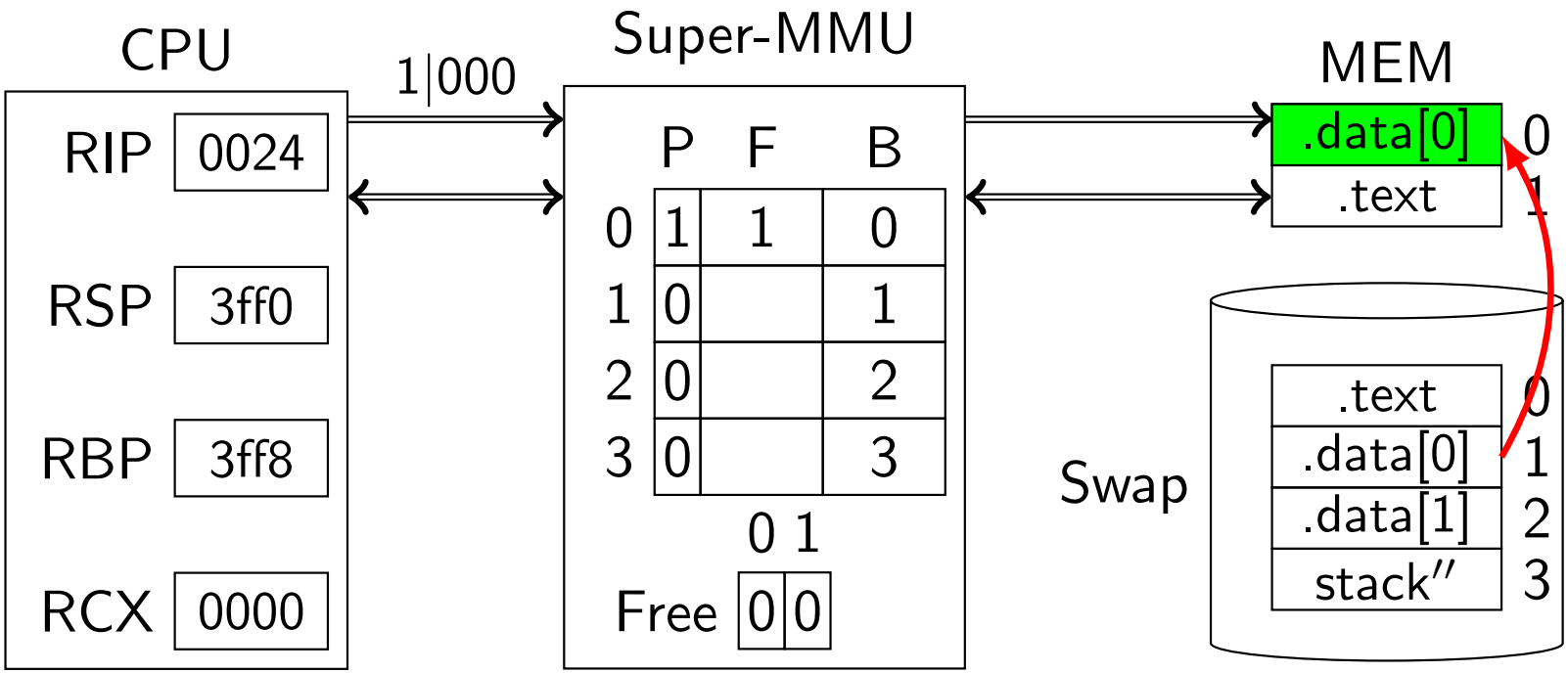
Prima di sovrascrivere il frame 0 è necessario aggiornare il blocco della pagina 3 sull'hard disk. In questo caso, se non lo facessimo, perderemmo il valore corrente della variable `sum`.



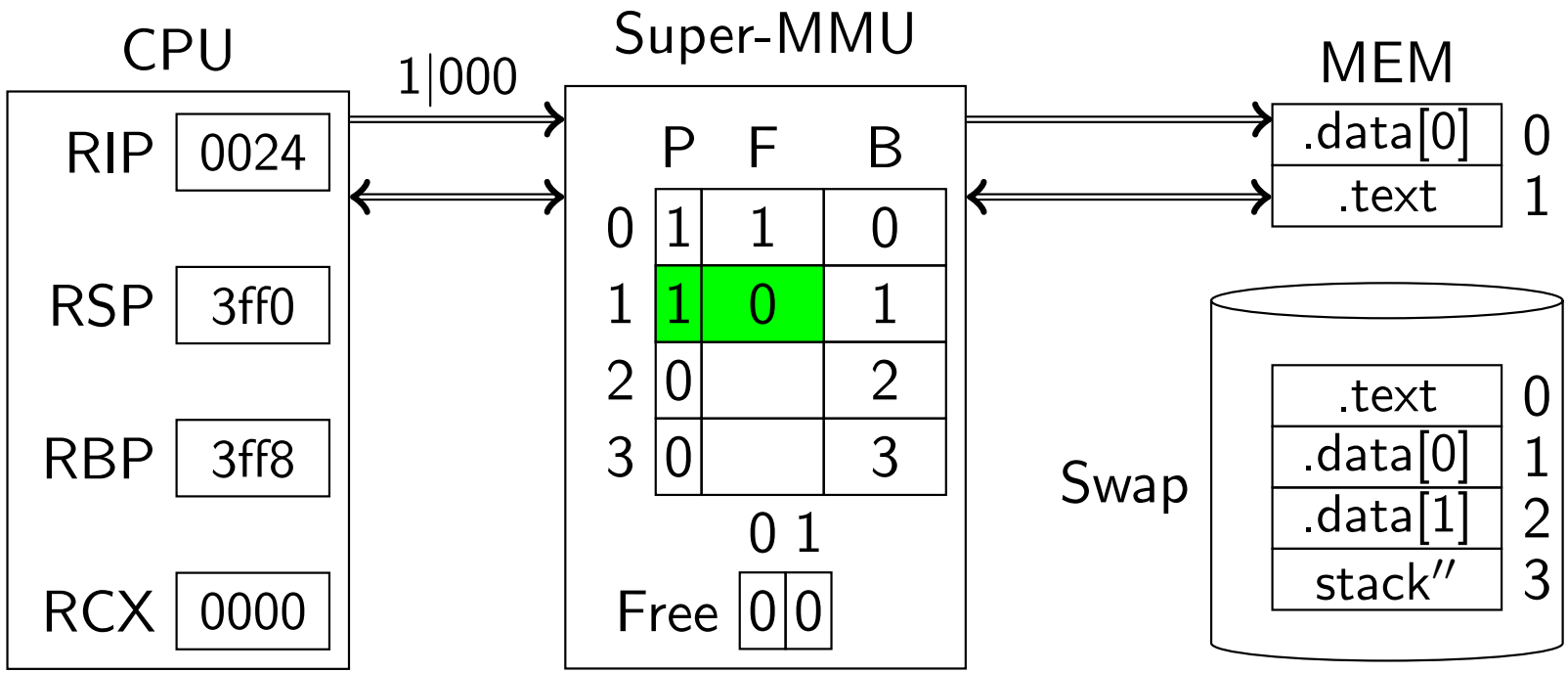
La MMU ordina quindi un trasferimento dal frame 0 al blocco 3 (campo B della riga 3).



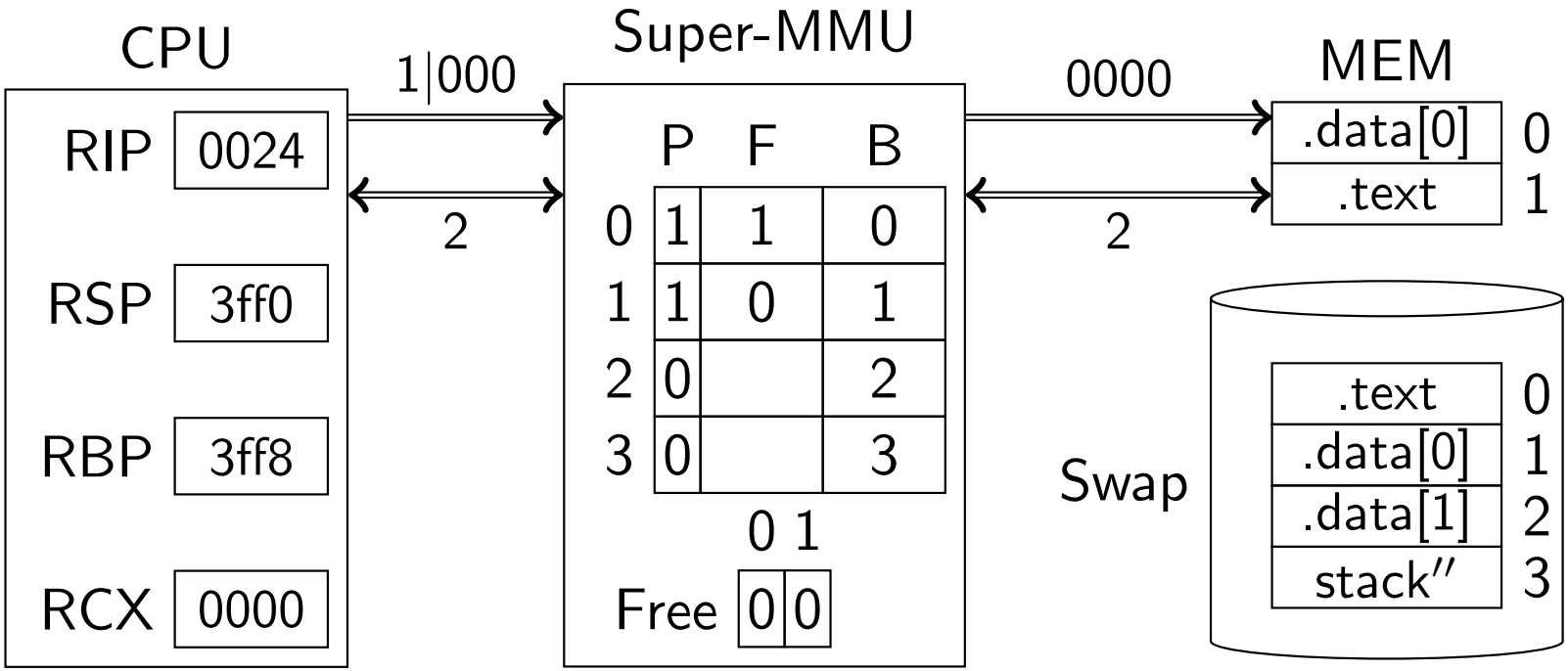
Una volta ricopiata la pagina 3 sul suo blocco, la MMU può marcarla come non presente e riutilizzare il frame 0 che la conteneva.



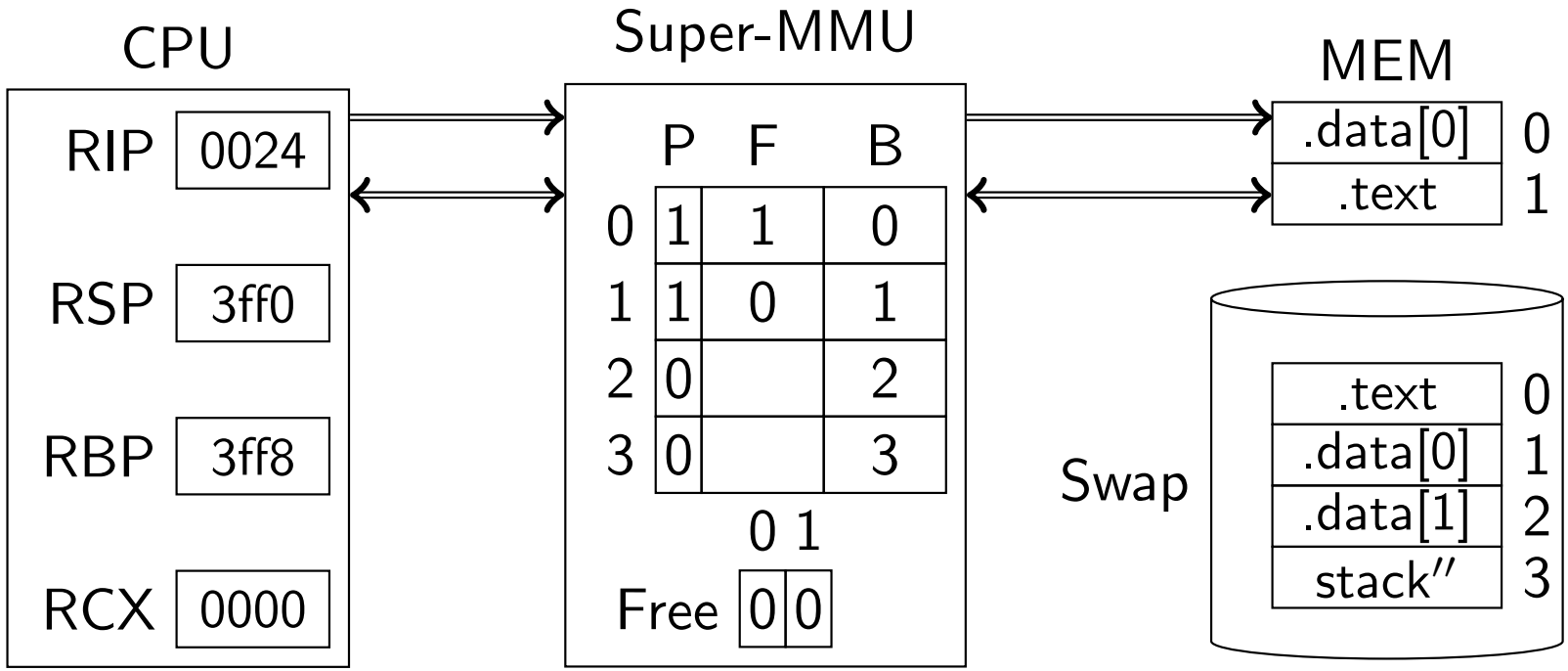
Ordina dunque un trasferimento dal blocco 1 (campo B della riga 1) al frame 0.



Aggiorna poi la tabella di corrispondenza per ricordare che la pagina virtuale 1 è ora presente (P $=$ 1) e si trova nel frame 0 (F $=$ 0).



Infine, completa l'accesso, sempre dopo aver trasformato l'indirizzo. La cpu può ora completare l'esecuzione dell'istruzione sommando il valore appena ricevuto al registro eax (non mostrato).



L'istruzione successiva (`addl %eax, -8(%rbp)`) vorrà scrivere nello stack, che ora non è più presente. La MMU se ne accorgerà e lo ricaricherà al posto di qualche altra pagina, e così via.