

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 possibile traduzione in assembly e linguaggio macchina è:

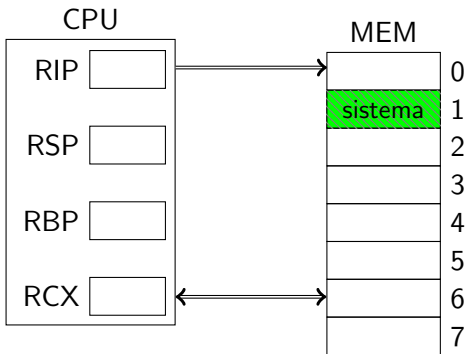
```
| .text |
| .global main |
2000| main: pushq %rbp |55
2001|     movq %rsp, %rbp |48 89 e5
2004|     subq $8, %rsp |48 83 ec 08
2008|     movl $0, -8(%rbp) |c7 45 f8 00 00 00 00
200f|     movl $0, -4(%rbp) |c7 45 fc 00 00 00 00
2016| for: cmpq $0x2000, -4(%rbp) |48 81 7d fc 00 20 00 00
2013|     jge fine |7d 14
2020|     movslq -4(%rbp), %rcx |48 63 4d fc
2024|     movsbl buf(%rcx), %eax |0f be 81 00 30 00 00
202b|     addl %eax, -8(%rbp) |01 45 f8
202e|     addl $1, -4(%rbp) |83 45 fc 01
2032|     jmp for |eb e2
2034| fine: movl -8(%rbp), %eax |8b 45 f8
2037|     popq %rbp |5d
2038|     ret |c3
| .data |
| ... |
3000| buf: .byte 2, 6, -1, 200 |02 06 ff c8
|     ... |
|     ... |
4ffc|     .byte 15, 3, -32, 1 |0f 03 e0 01
```

- ▶ Supponiamo per semplicità che gli indirizzi possibili vadano da 0000 a 7fff (32 KiB di memoria).

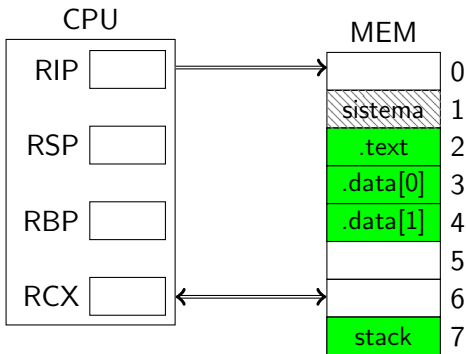
- ▶ Supponiamo per semplicità che gli indirizzi possibili vadano da 0000 a 7fff (32 KiB di memoria).
- ▶ Immaginiamo di suddividere la memoria in 8 *pagine*, ciascuna grande 4 KiB (1000 in esadecimale);

- ▶ Supponiamo per semplicità che gli indirizzi possibili vadano da 0000 a 7fff (32 KiB di memoria).
- ▶ Immaginiamo di suddividere la memoria in 8 *pagine*, ciascuna grande 4 KiB (1000 in esadecimale);
- ▶ Riserviamo per il sistema le pagine 0 e 1.

- ▶ Supponiamo per semplicità che gli indirizzi possibili vadano da 0000 a 7fff (32 KiB di memoria).
- ▶ Immaginiamo di suddividere la memoria in 8 *pagine*, ciascuna grande 4 KiB (1000 in esadecimale);
- ▶ Riserviamo per il sistema le pagine 0 e 1.
- ▶ Il nostro programma contiene il codice nella pagina che va da 2000 a 3fff (pagina 2) e la variabile `buf` nelle due pagine successive (3 e 4). Usiamo l'ultima pagina (7) come pila.

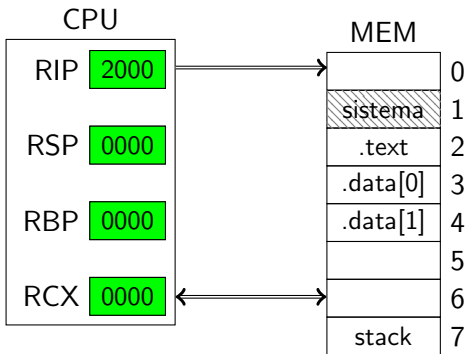


Vediamo prima l'esecuzione sul sistema che ha tutta la memoria. Mostriamo solo alcuni dei registri della CPU. Tutti i numeri saranno mostrati in esadecimale.

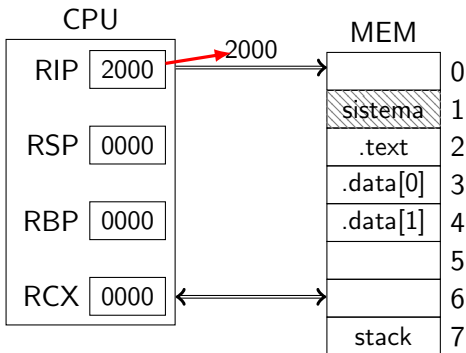


Carichiamo il programma in memoria.

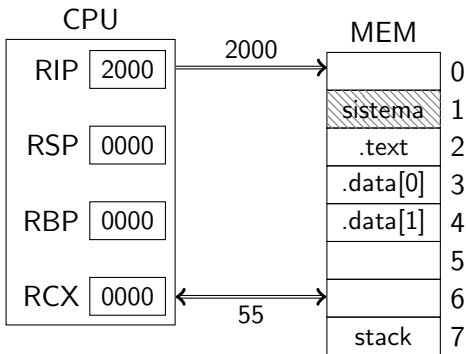




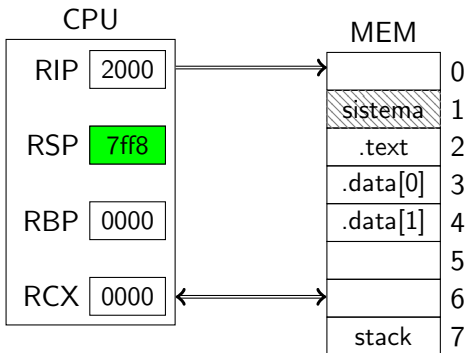
Inizializziamo tutti i registri



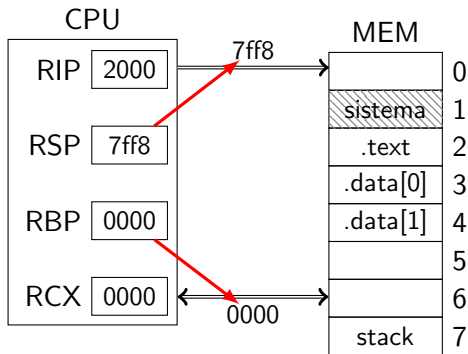
**Avviamo il sistema.** La CPU tenta di prelevare l'istruzione all'indirizzo contenuto in RIP. Inizia quindi una operazione di lettura in memoria all'indirizzo 2000.



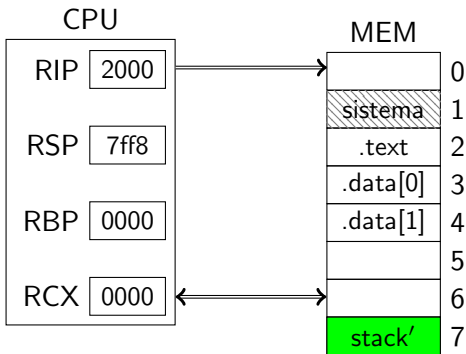
L'operazione di lettura restituisce l'istruzione `pushq %rbp`.



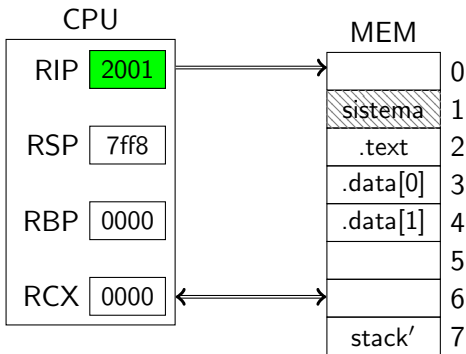
La CPU inizia ad eseguire l'istruzione. Il primo passo è decrementare RSP di 8.



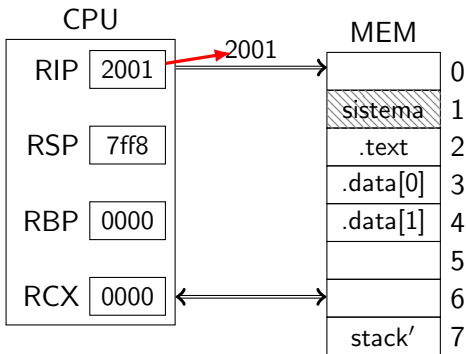
Il secondo passo è scrivere il contenuto di RBP all'indirizzo contenuto in RSP. La CPU inizia quindi una operazione di scrittura all'indirizzo 7ff8.



Dopo la scrittura il contenuto dello stack sarà cambiato.

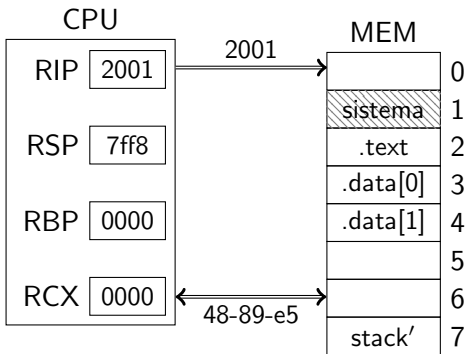


La CPU ha completato l'esecuzione dell'istruzione `pushq %rbp` e può passare alla successiva. Incrementa RIP di 1 (che è la dimensione dell'istruzione appena terminata).

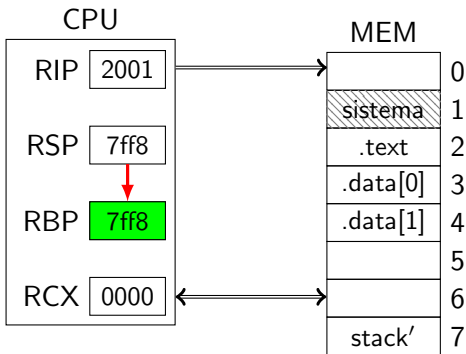


Nuovo ciclo: la CPU tenta di prelevare l'istruzione all'indirizzo contenuto in RIP. Inizia una operazione di lettura in memoria all'indirizzo 2001.

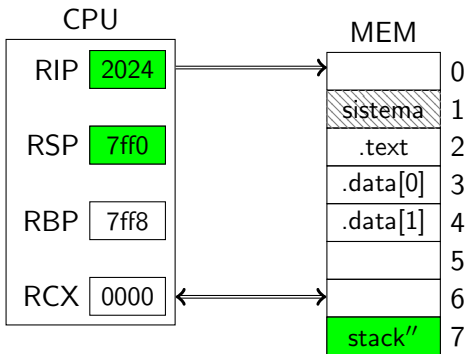




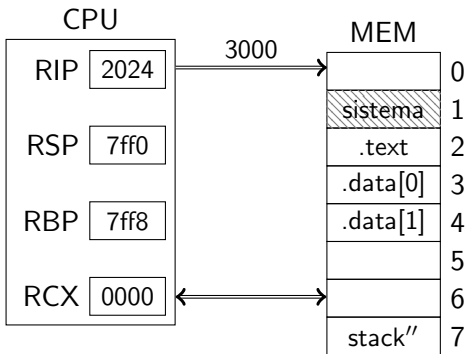
La CPU riceve l'istruzione `movq %rsp, %rbp`.



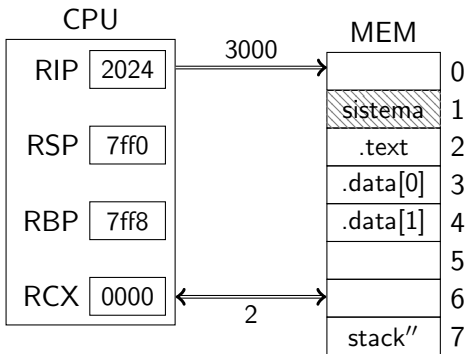
Per eseguirla copia il contenuto di RSP in RBP.  
L'istruzione non prevede accessi in memoria.



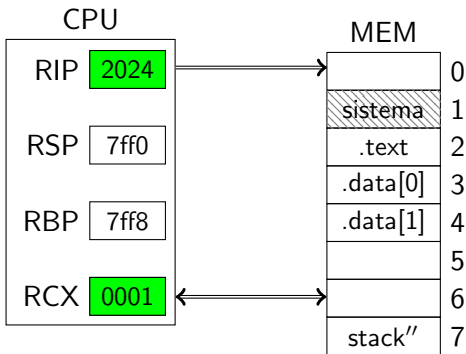
Dopo alcune istruzioni, l'esecuzione del programma arriva all'istruzione "movsbl buf(%rcx), %rax" che si trova all'indirizzo 2024. Questa è la prima istruzione che accede alla sezione .data del programma.



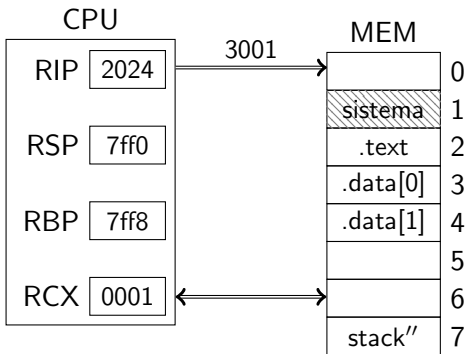
La CPU somma il contenuto di RCX e la costante 3000 (buf nel sorgente), ottenendo l'indirizzo 3000. Inizia dunque una operazione di lettura a questo indirizzo.



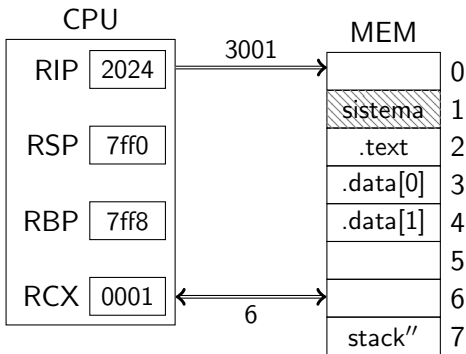
La CPU riceve il primo byte contenuto nel buffer (valore 2) e lo copia nel registro EAX (non mostrato).



L'esecuzione prosegue sommando EAX in  $-8(\%rbp)$ , incrementando RCX, etc., fino a quando si torna all'indirizzo 2024

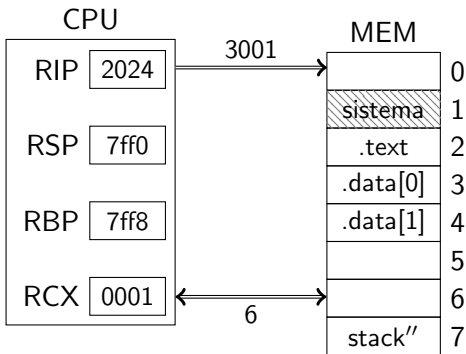


La CPU somma il contenuto di RCX e la costante 3000 (buf nel sorgente), ottenendo l'indirizzo 3001. Inizia dunque una operazione di lettura a questo indirizzo.

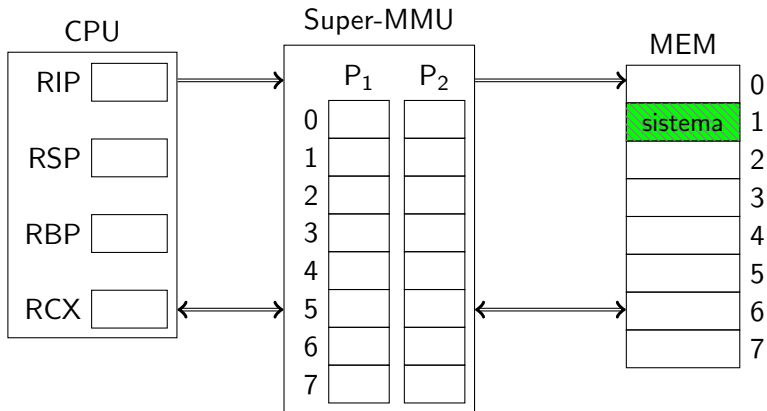


La cpu legge il valore 6 e lo somma al registro eax (non mostrato).

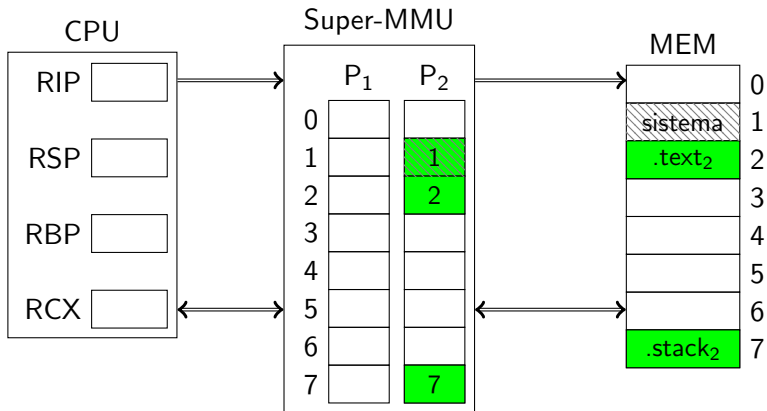




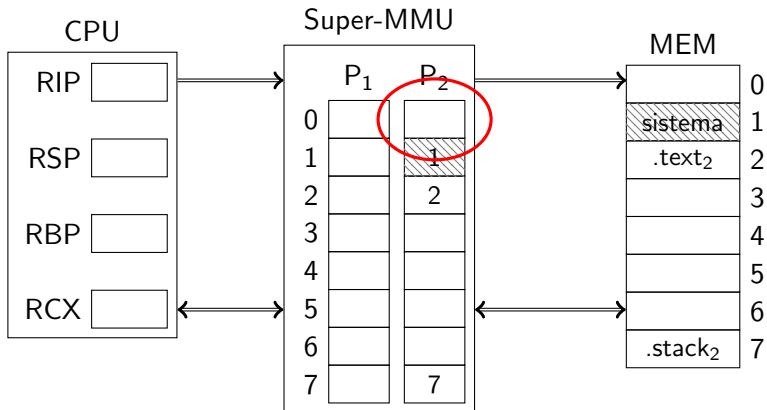
L'esecuzione prosegue in questo modo, sommando tutti i valori di buf.



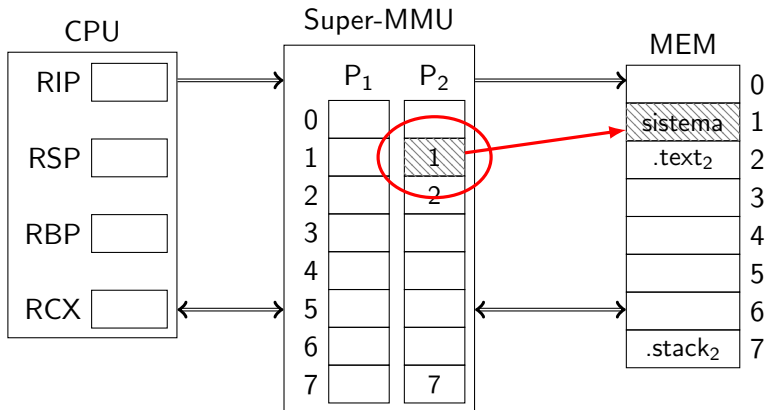
Vediamo ora un esempio di esecuzione con più processi e paginazione.



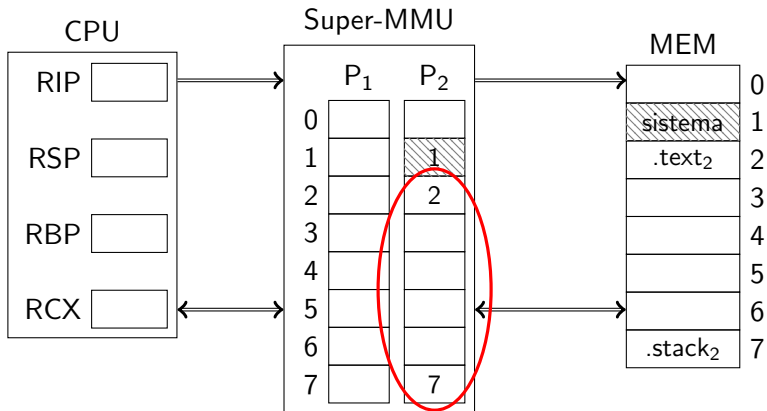
Supponiamo che ci sia già un processo P<sub>2</sub>, caricato come in figura.



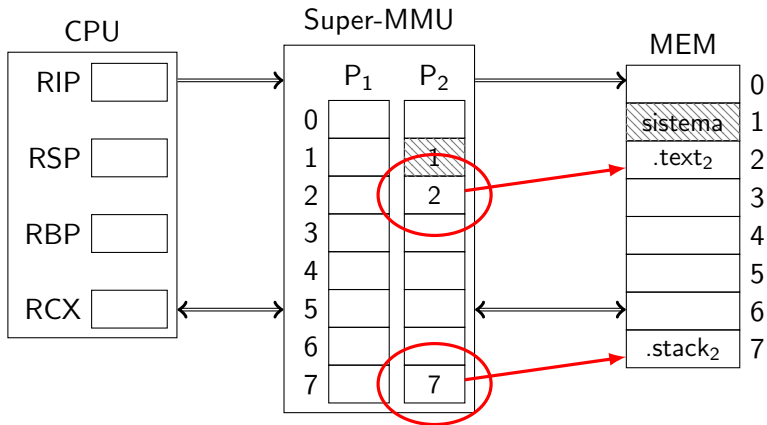
La pagina 0 è lasciata con P=0, per intercettare le dereferenziazioni di puntatori nulli.



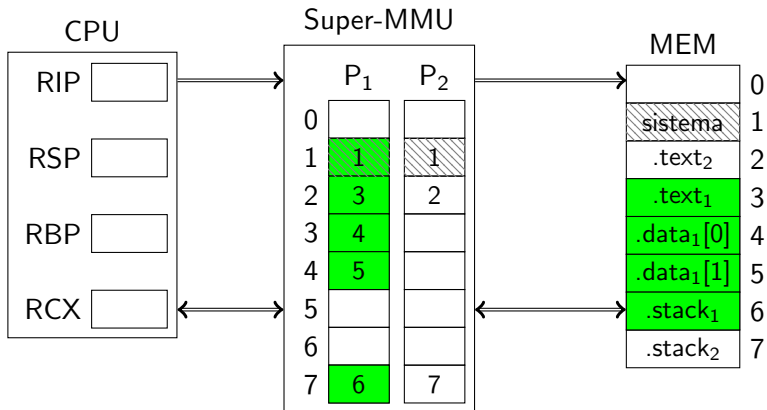
La pagina 1 corrisponde al frame 1. È marcata come inaccessibile da livello utente.



Le pagine da 3 a 6 non sono usate (P=0)

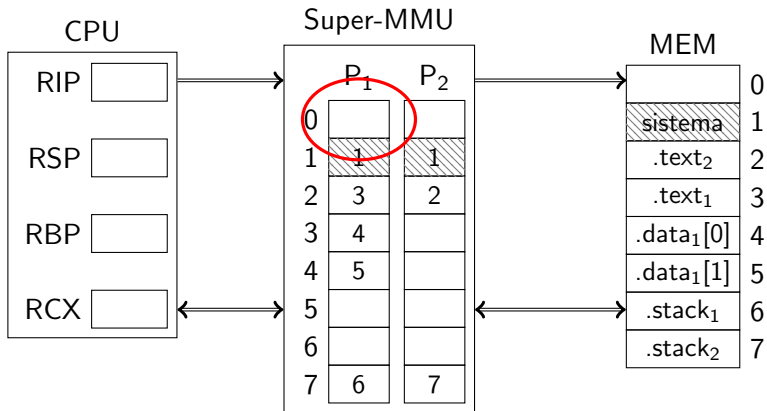


Le altre pagine corrispondono ai frame che le contengono

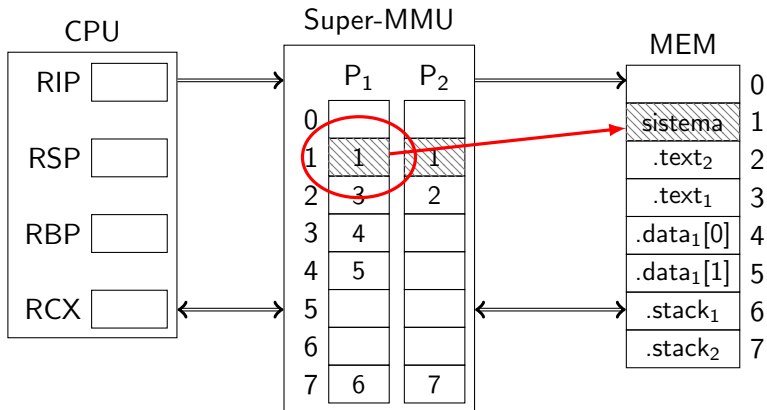


Carichiamo P<sub>1</sub> nello spazio che rimane.

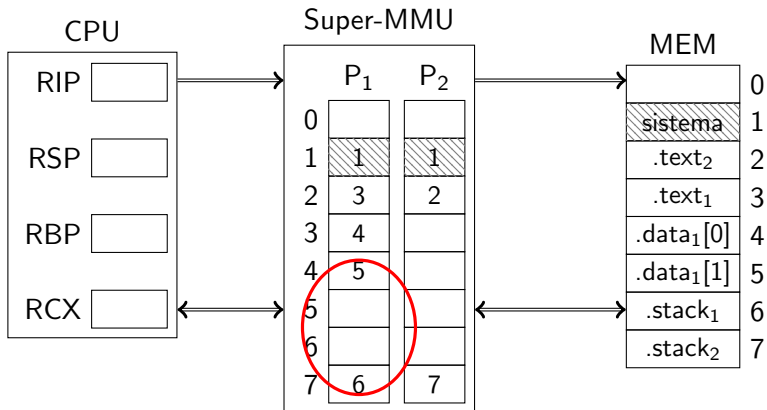




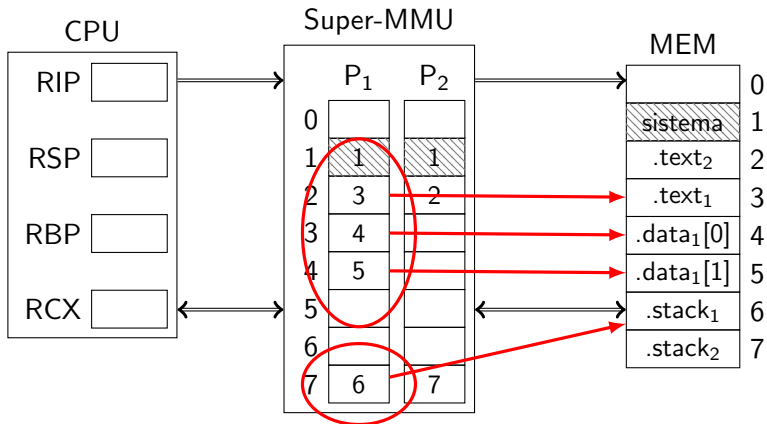
Anche per  $P_1$  La pagina 0 è lasciata con  $P=0 \dots$



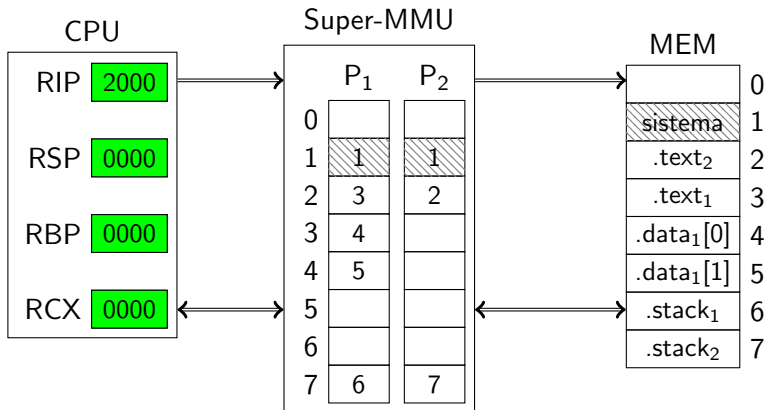
... e la pagina 1 corrisponde al frame 1, ma è inaccessibile da livello utente.



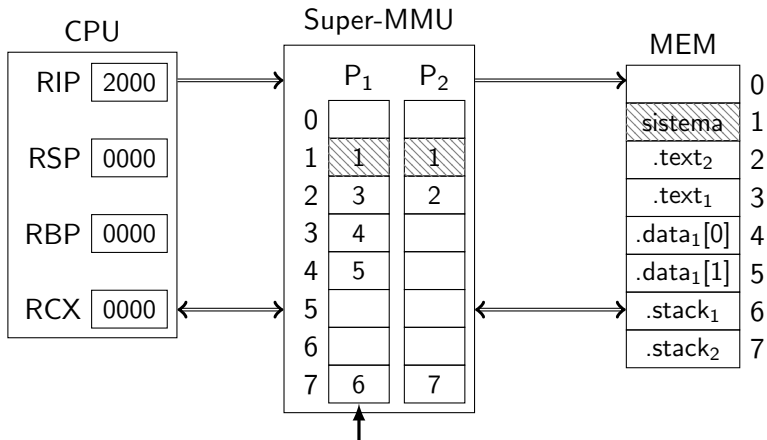
Le pagine 5 e 6 non sono usate (P=0) ...



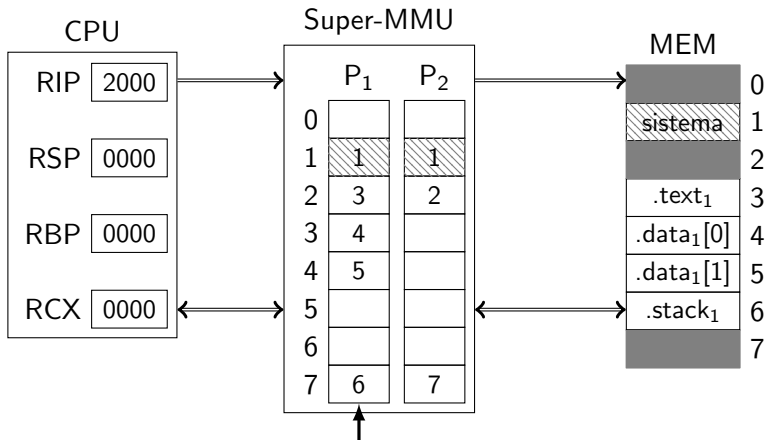
... e le altre corrispondono ai frame che le contengono



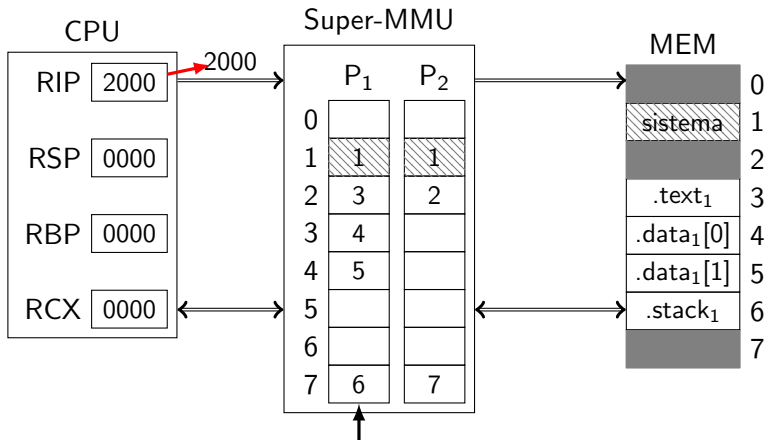
Supponiamo che il sistema metta in esecuzione P<sub>1</sub>, caricando lo stato dei registri...



... e rendendo attiva la tabella P<sub>1</sub>.

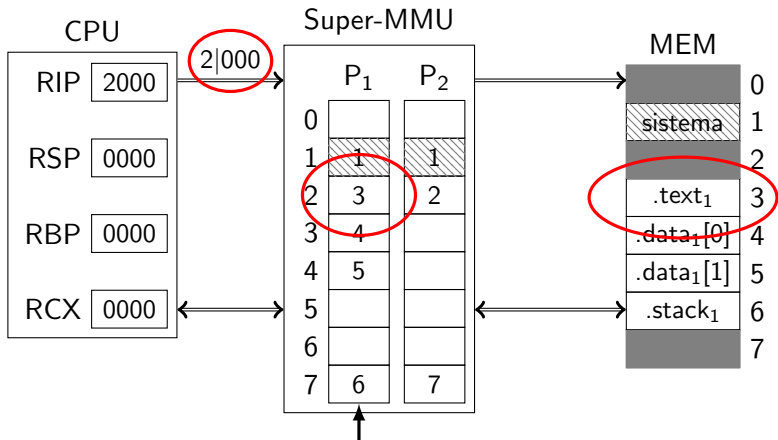


Tutte le pagine che non sono nel codominio di P<sub>1</sub> diventano inaccessibili.

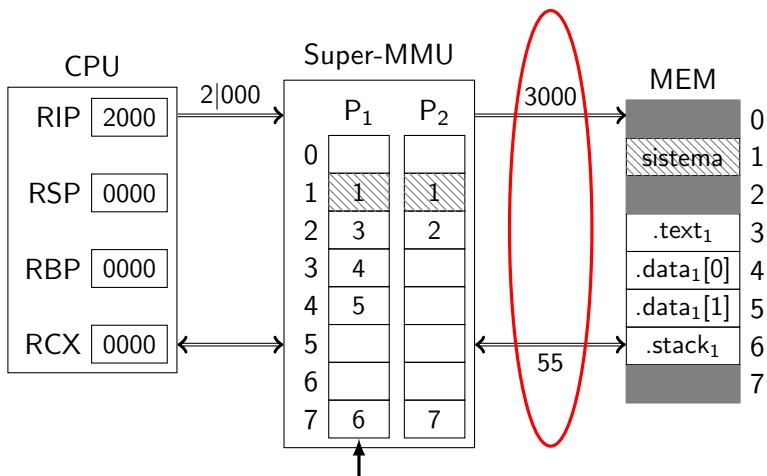


P<sub>1</sub> comincia la sua esecuzione. La CPU esegue una lettura all'indirizzo 2000.

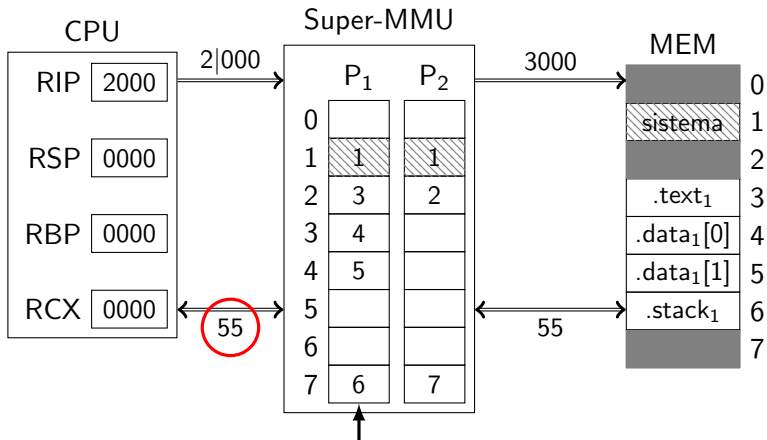




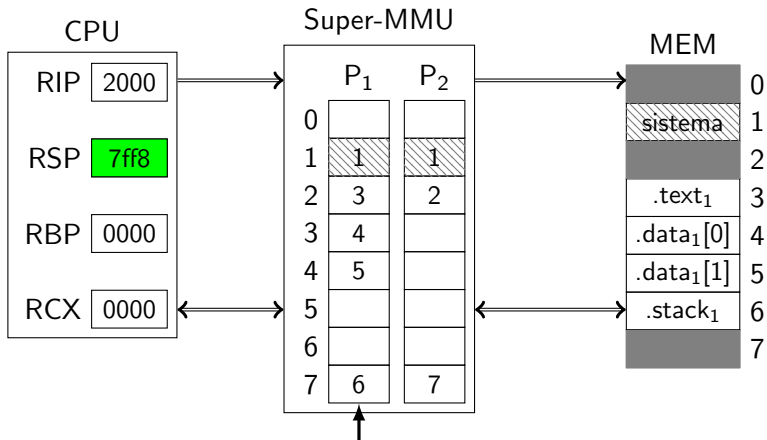
La MMU intercetta l'operazione e scompone l'indirizzo in numero di pagina (2) e offset (000). Consulta quindi l'entrata numero 2 della tabella di corrispondenza e trova il corrispondente numero di frame (3)



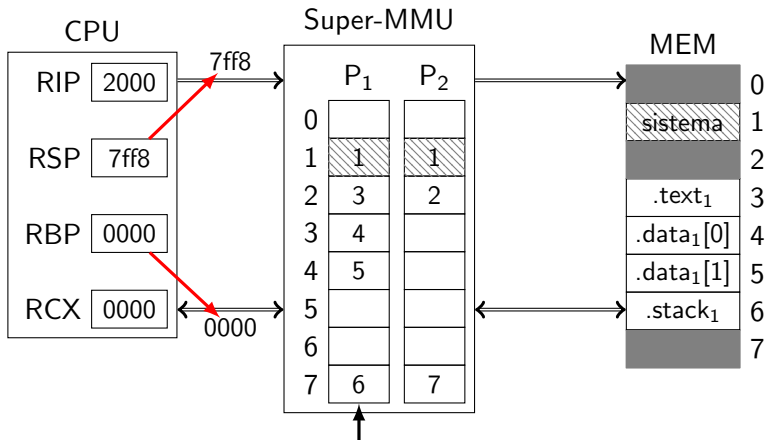
Completa l'accesso dopo aver tradotto l'indirizzo.



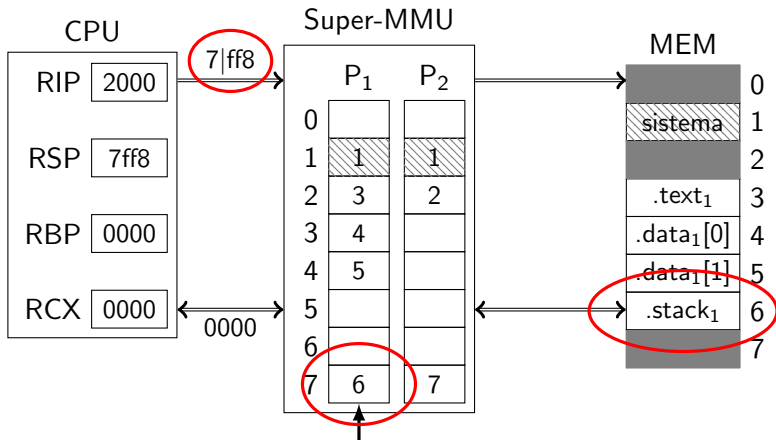
L'operazione di lettura restituisce l'istruzione `pushq %rbp`, che la CPU inizia ad eseguire.



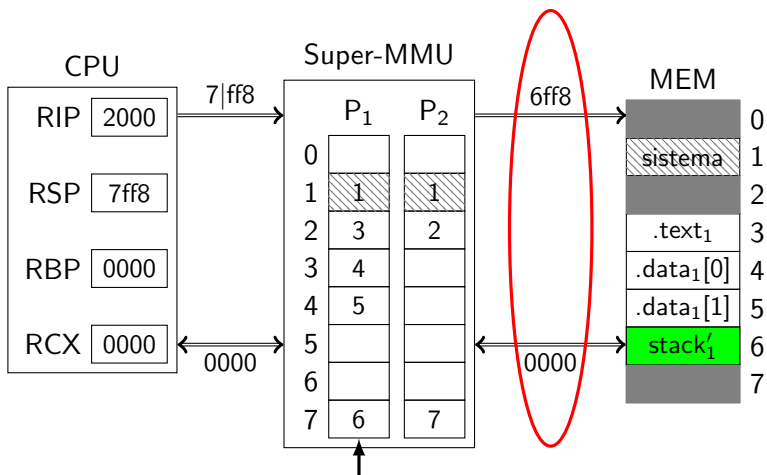
Il primo passo è decrementare RSP di 8.



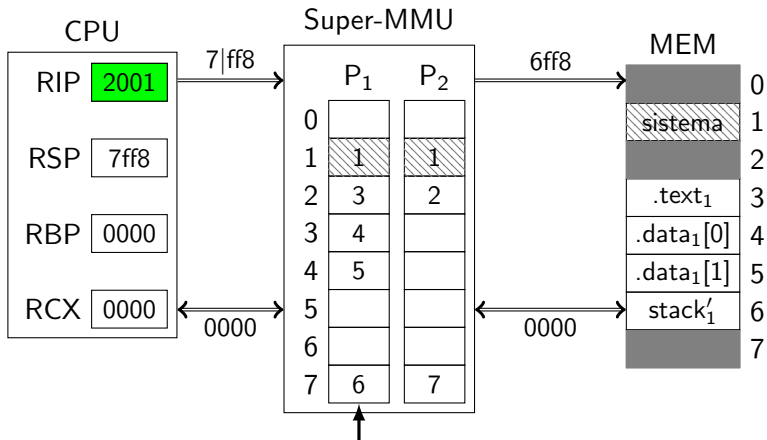
Il secondo passo è scrivere il contenuto di RBP all'indirizzo contenuto in RSP. La CPU inizia quindi una operazione di scrittura all'indirizzo 7ff8.



La MMU intercetta l'operazione e scompone l'indirizzo in numero di pagina (7) e offset (ff8). Consulta quindi l'entrata numero 7 della tabella di corrispondenza e trova il numero di frame (6)

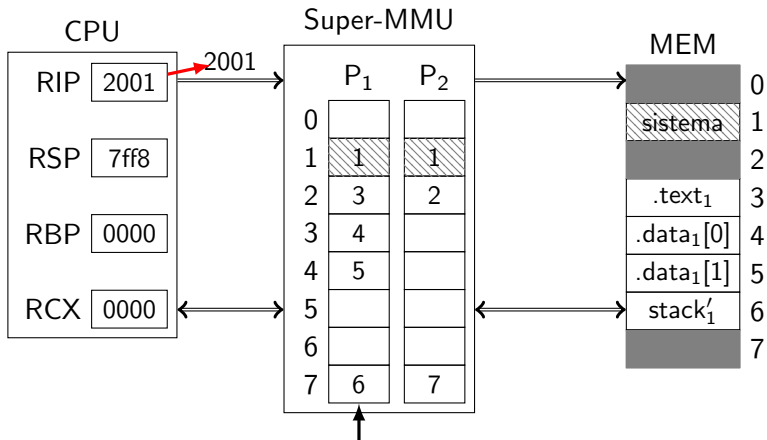


Completa l'operazione di scrittura, dopo aver trasformato l'indirizzo.

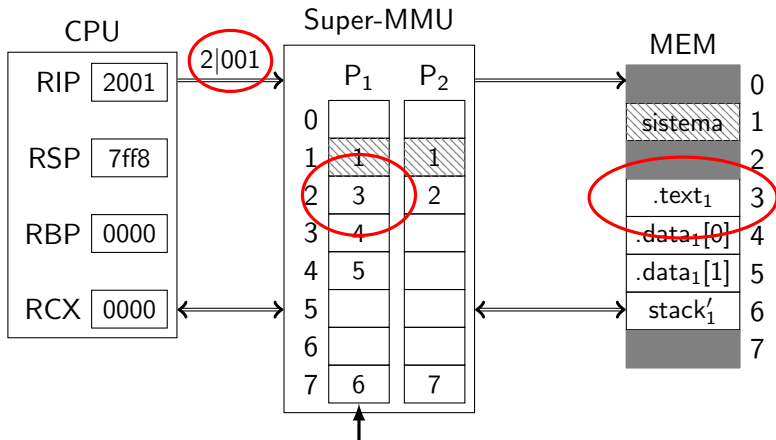


La CPU ha completato l'esecuzione dell'istruzione `pushq %rbp` e può passare alla successiva. Incrementa RIP di 1 (che è la dimensione dell'istruzione appena terminata).

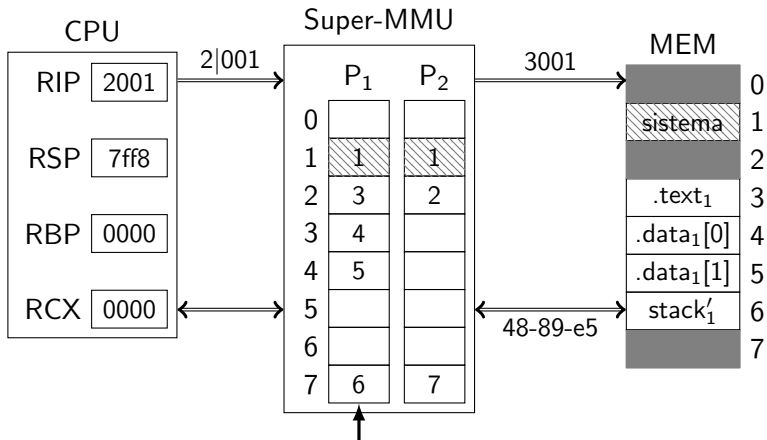




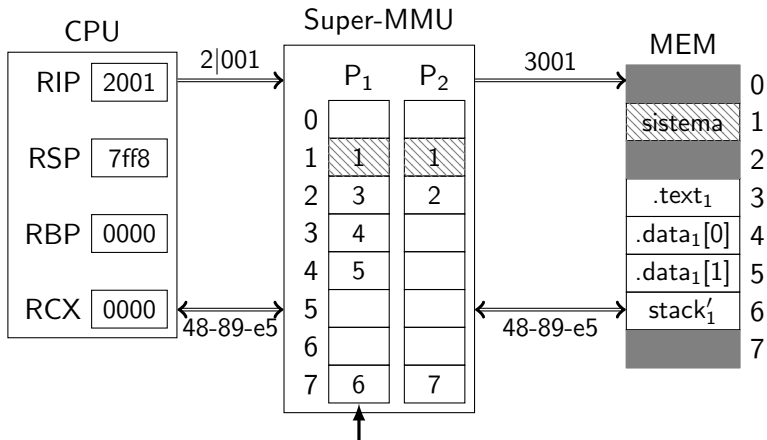
Nuovo ciclo: la CPU tenta di prelevare l'istruzione all'indirizzo contenuto in RIP. Inizia una operazione di lettura in memoria all'indirizzo 2001.



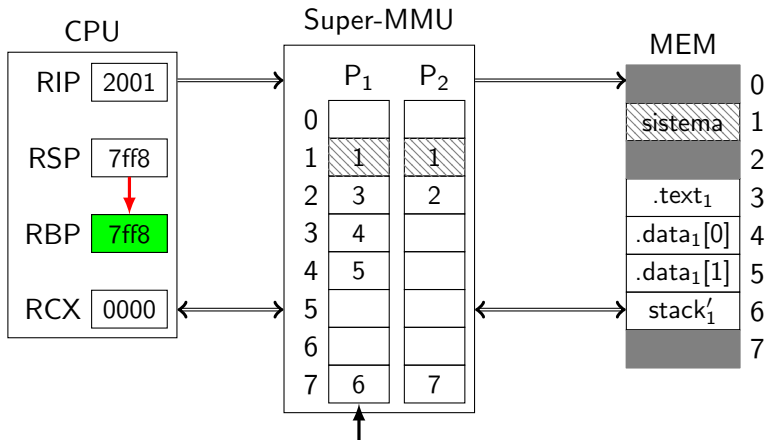
Come al solito, la MMU intercetta l'operazione e scompone l'indirizzo in numero di pagina virtuale (2) e offset (001).



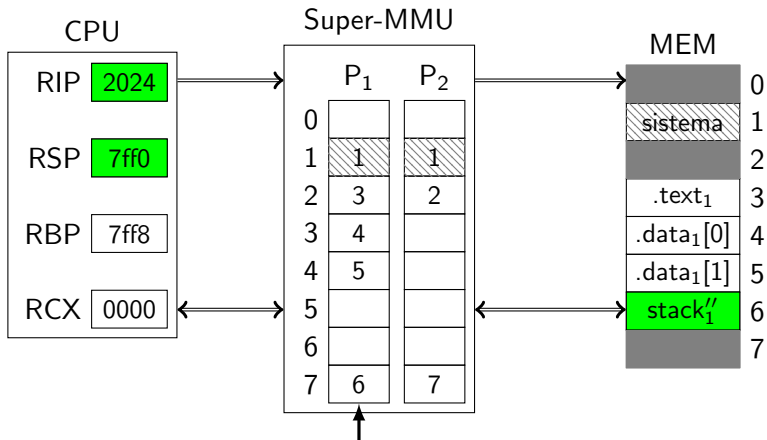
Completa poi l'accesso, ovviamente dopo aver trasformato l'indirizzo.



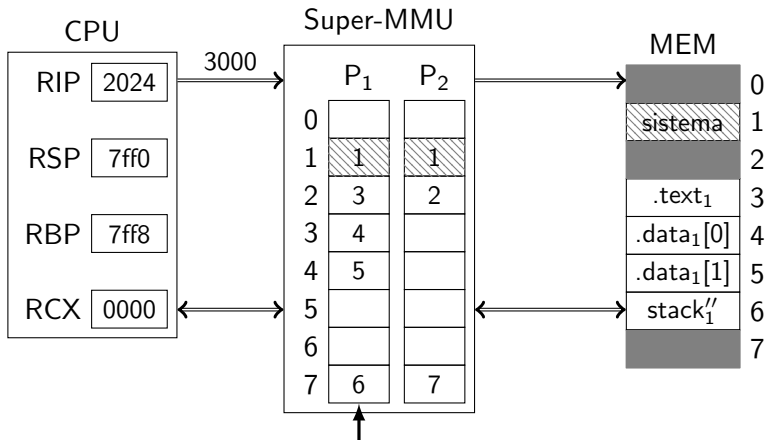
La CPU riceve l'istruzione `movq %rsp, %rbp`.



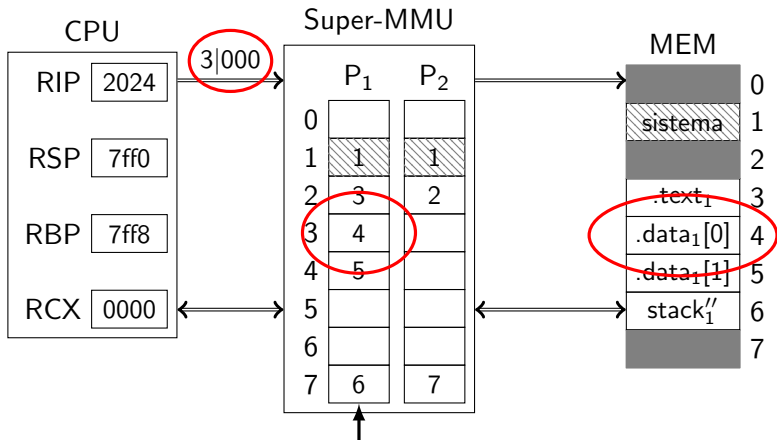
Per eseguirla copia il contenuto di RSP in RBP.  
L'istruzione non prevede accessi in memoria.



Dopo alcune istruzioni, l'esecuzione del programma arriva all'istruzione `movsb1 buf(%rcx), %rax` che si trova all'indirizzo (virtuale) 2024.

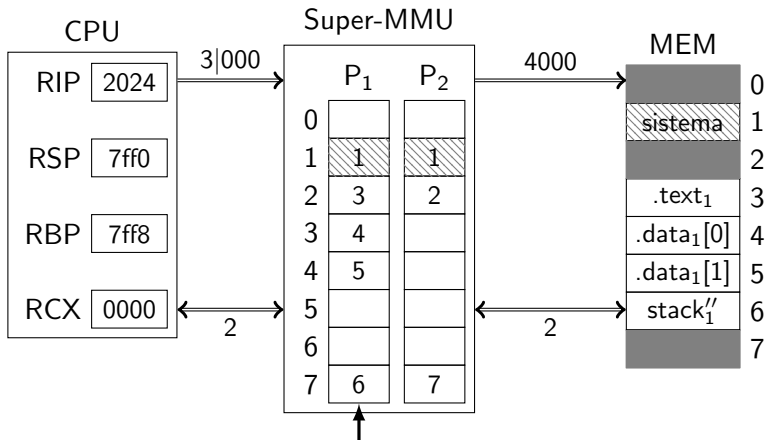


La CPU somma il contenuto di RCX e la costante 3000 (buf nel sorgente), ottenendo l'indirizzo 3000. Inizia dunque una operazione di lettura a questo indirizzo.

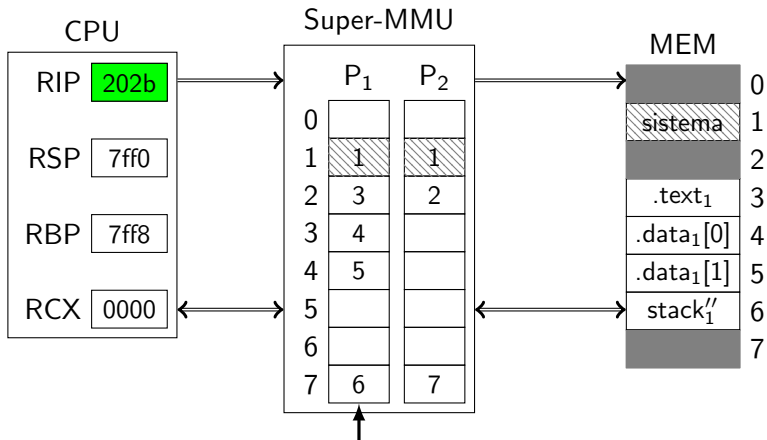


Ancora una volta la MMU scompone l'indirizzo in numero di pagina (3) e offset (000). L'entrata numero 3 della tabella di corrispondenza dice che la pagina 3 si trova nel frame 4.

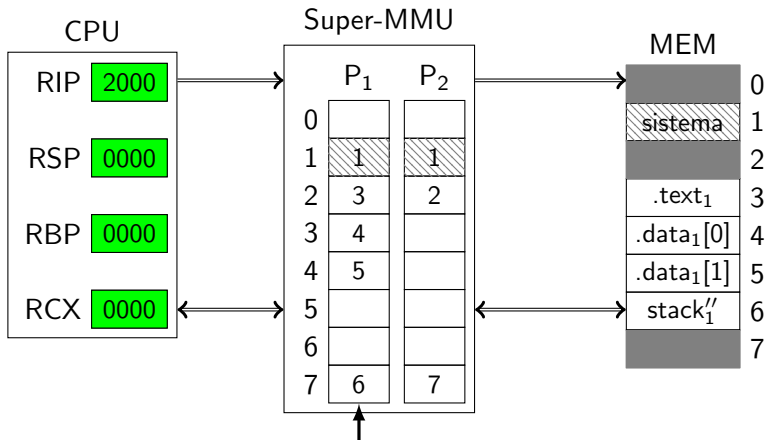




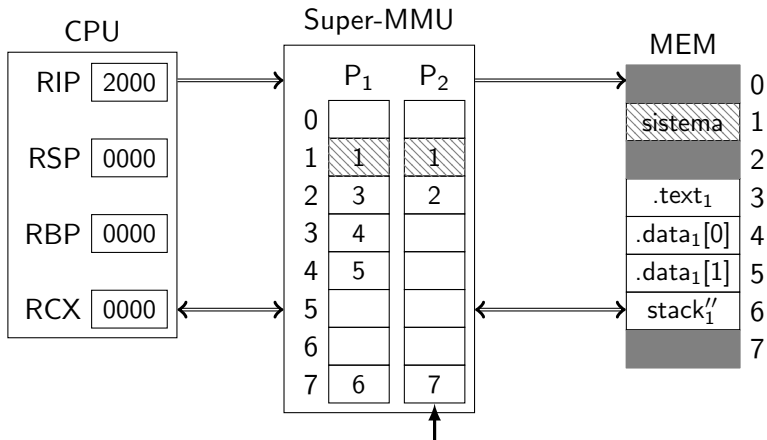
Completa dunque l'accesso, sempre dopo aver trasformato l'indirizzo. La cpu completa l'esecuzione dell'istruzione sommando il valore appena ricevuto al registro eax (non mostrato).



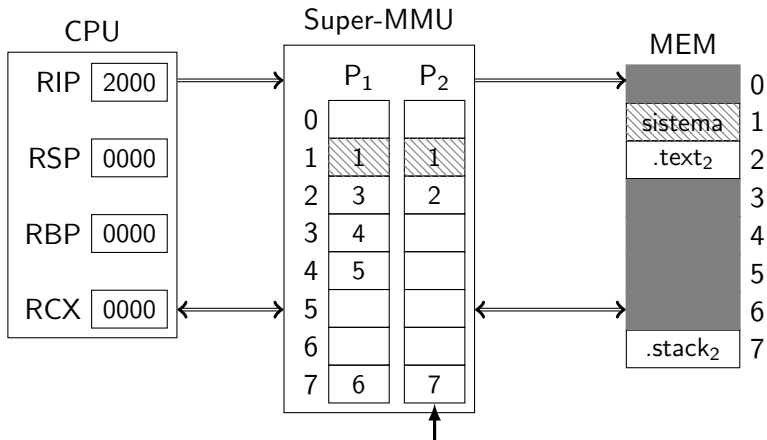
Supponiamo che a questo punto ci sia un'interruzione con cambio di processo e vada in esecuzione P<sub>2</sub>



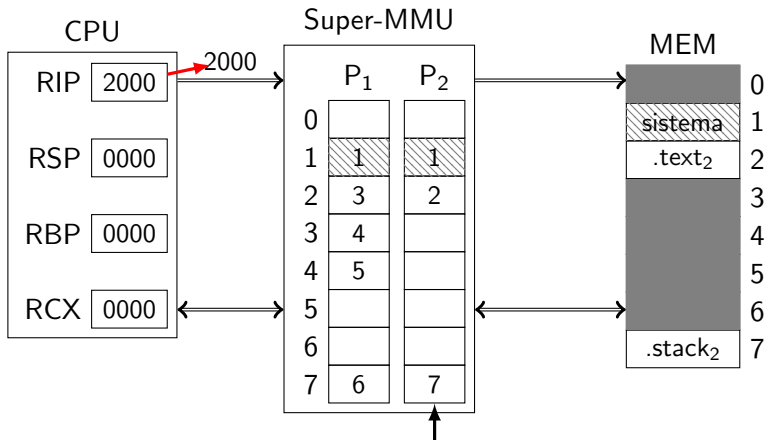
Il sistema carica i registri di P<sub>2</sub>...



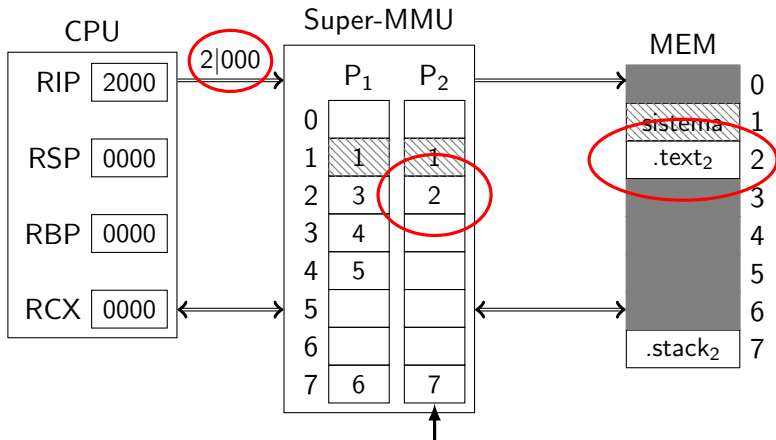
... e attiva la tabella P<sub>2</sub>



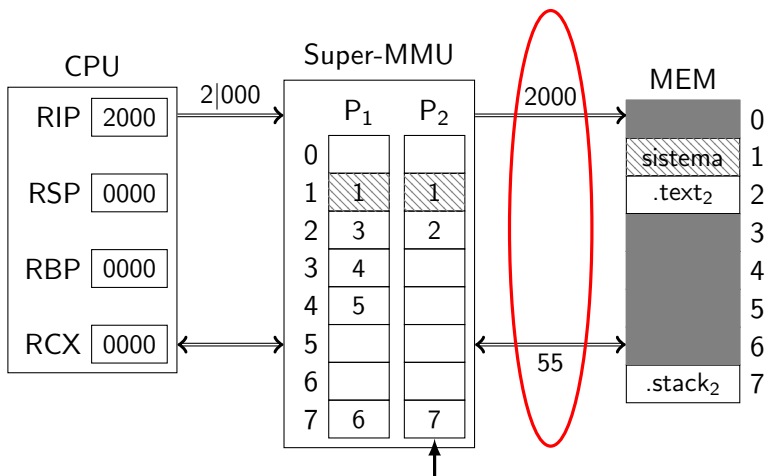
Diventano accessibili le pagine nel codominio di P<sub>2</sub> e non accessibili tutte le altre.



P<sub>2</sub> comincia la sua esecuzione. La CPU esegue una lettura all'indirizzo 2000.

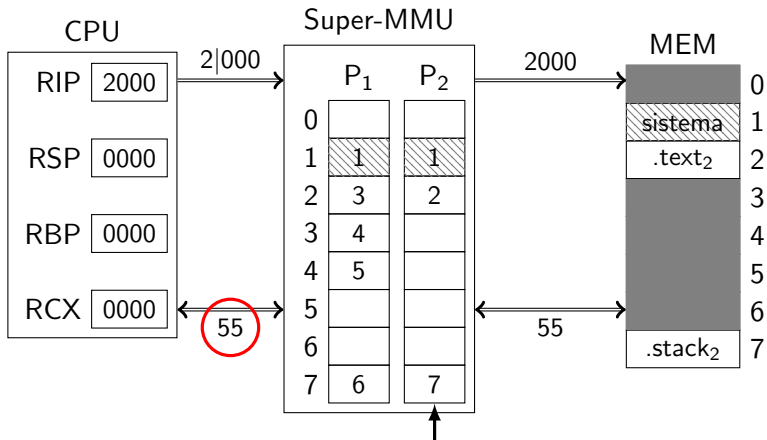


La MMU intercetta l'operazione e scompone l'indirizzo in numero di pagina (2) e offset (000). Consulta quindi l'entrata numero 2 della tabella di corrispondenza e trova il corrispondente numero di frame (2)

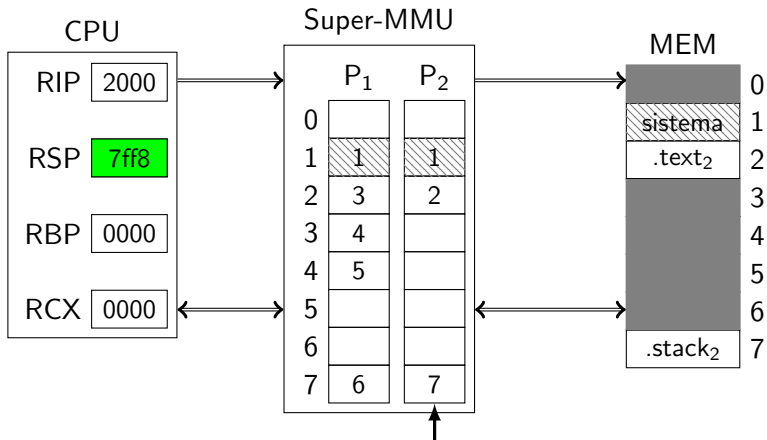


Completa l'accesso dopo aver tradotto l'indirizzo. Supponiamo che la prima istruzione di P<sub>2</sub> sia una pushq %rbp.

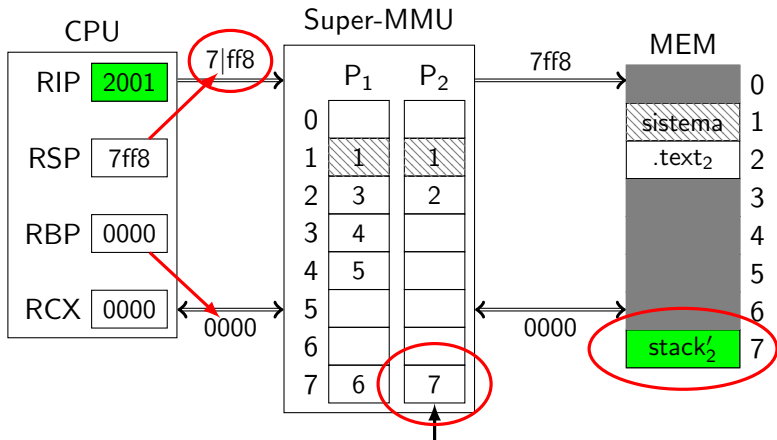




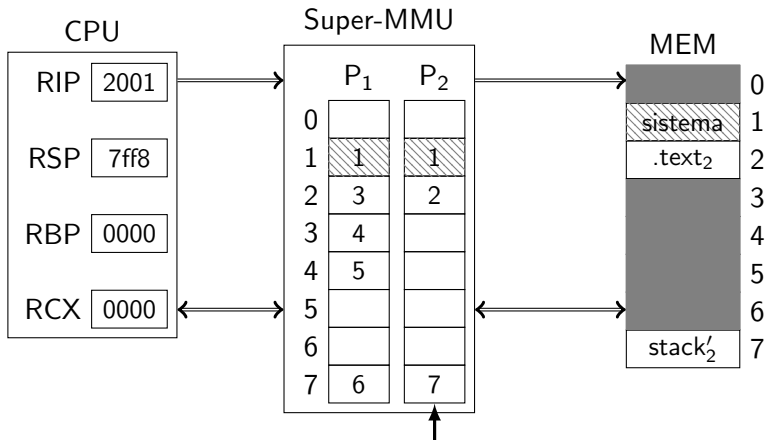
L'operazione di lettura restituisce l'istruzione, che la CPU inizia ad eseguire.



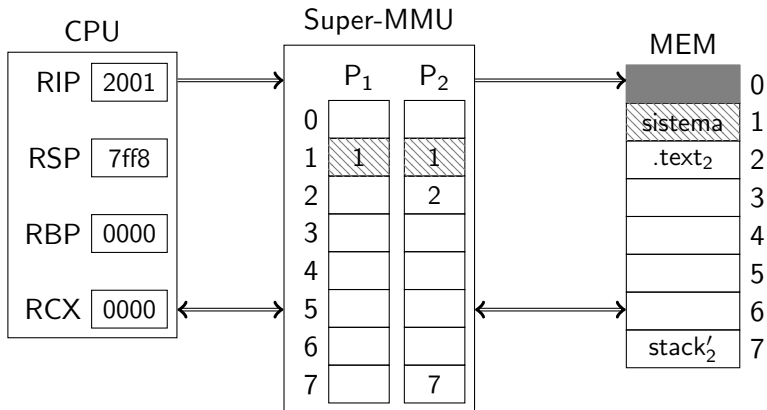
Decrementare RSP di 8 ...



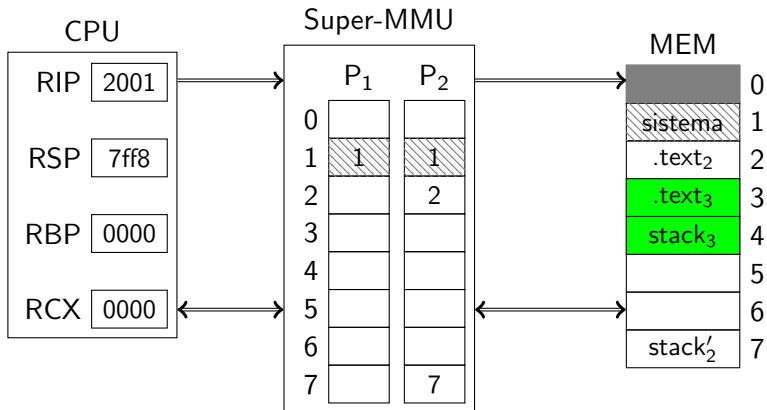
Scrive il contenuto di RBP all'indirizzo contenuto in RSP. La MMU traduce opportunamente l'indirizzo (in questo caso resta invariato)



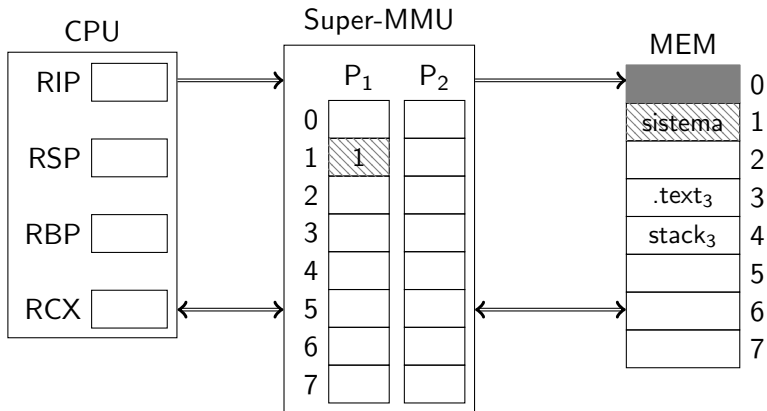
Supponiamo che ora P<sub>2</sub> venga interrotto e il sistema decida di caricare un altro processo, P<sub>3</sub>.



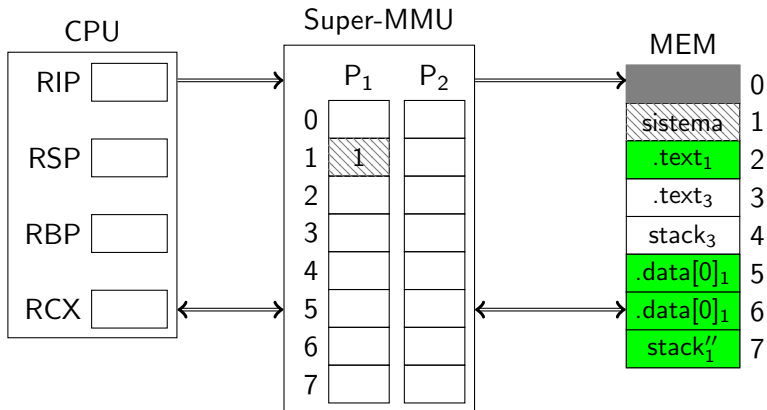
Per fare spazio, il sistema rimuove le pagine di P<sub>1</sub> dopo averle copiate nello swap (operazione di *swap-out*).



Quindi carica P<sub>3</sub> e inizializza opportunamente la sua tabella di corrispondenza (non mostrata).

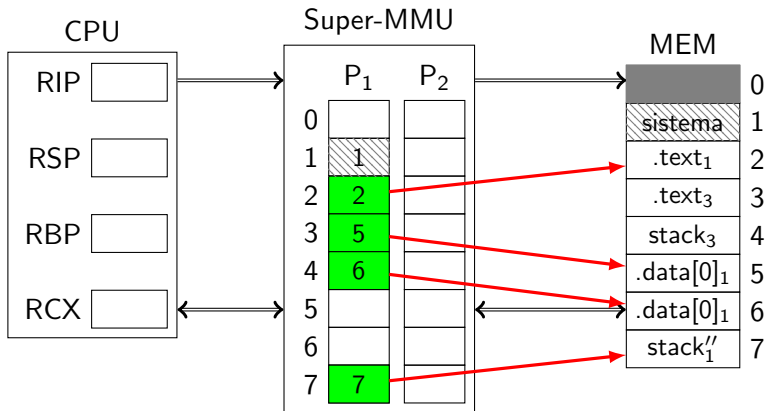


Successivamente, P<sub>2</sub> termina. Il sistema libera tutte le sue pagine.

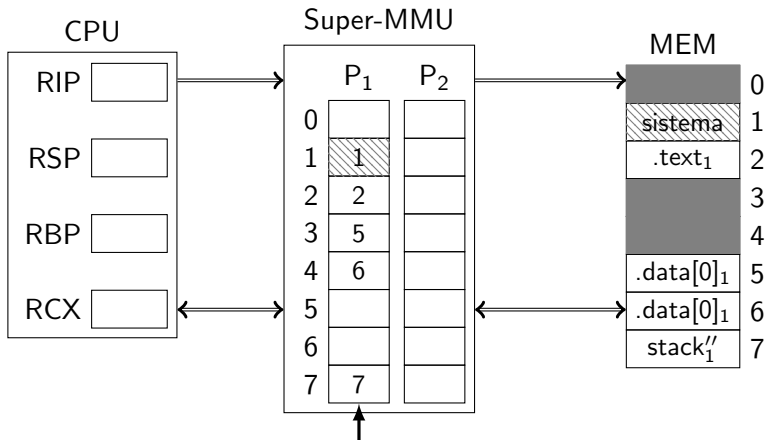


Ancora dopo, il sistema decide di ricaricare P<sub>1</sub> per rimetterlo in esecuzione.

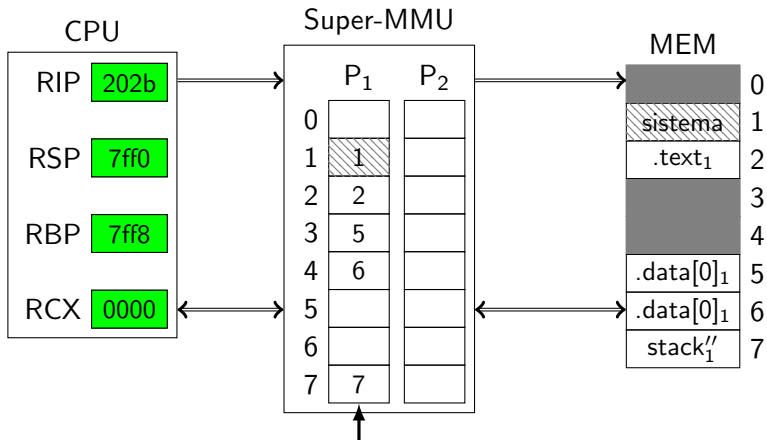




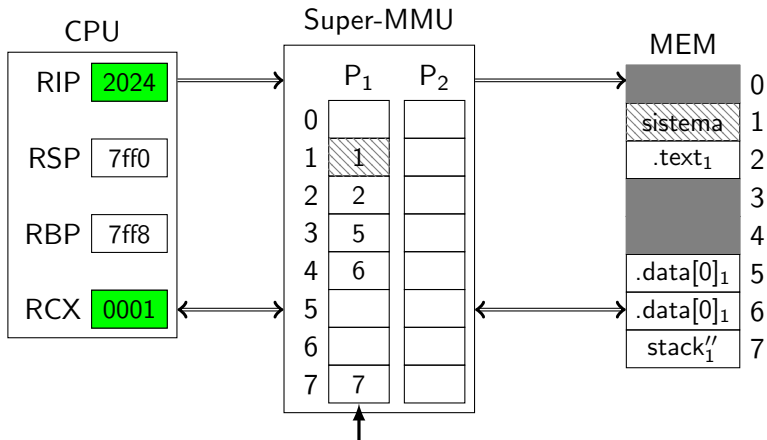
Il sistema inizializza la tabella di corrispondenza di P<sub>1</sub> con i nuovi numeri di frame.



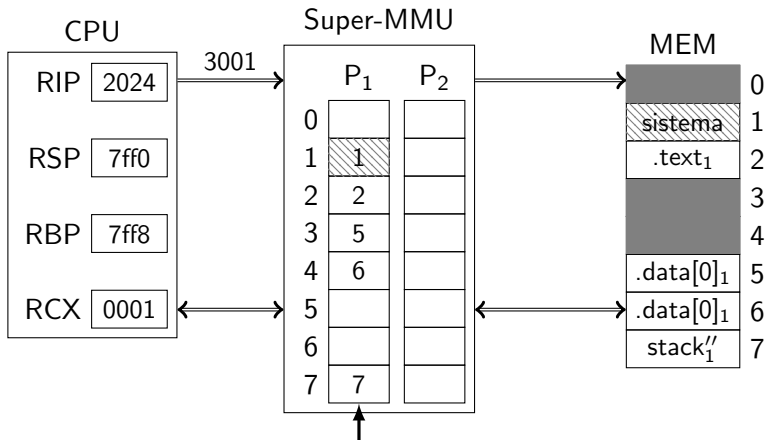
Attiva la tabella di P<sub>1</sub>. Le pagine di P<sub>3</sub> diventano inaccessibili.



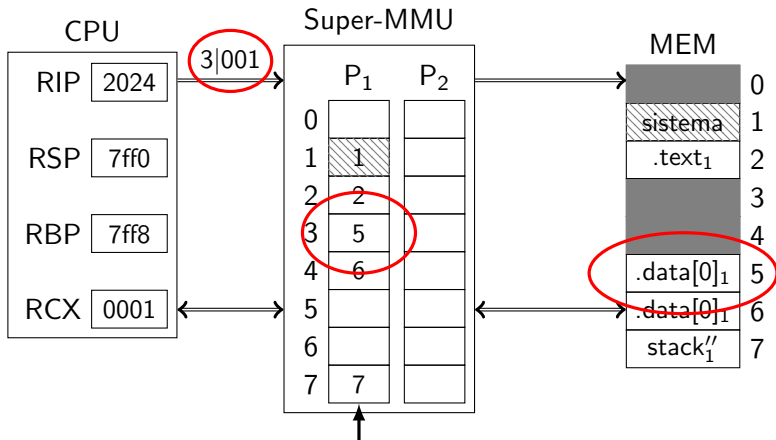
Infine, ricarica i registri con l'ultimo stato salvato di P<sub>1</sub> e gli cede il controllo.



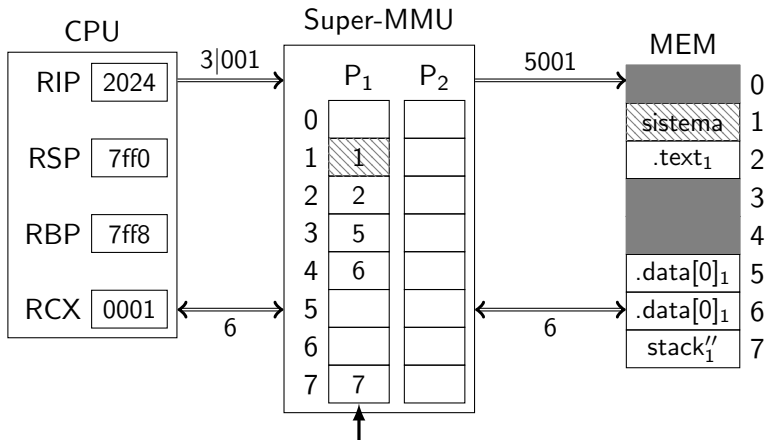
L'esecuzione prosegue aggiungendo al totale il valore precedentemente letto e incrementando %rcx, fino a quando si ritorna all'indirizzo 2024.



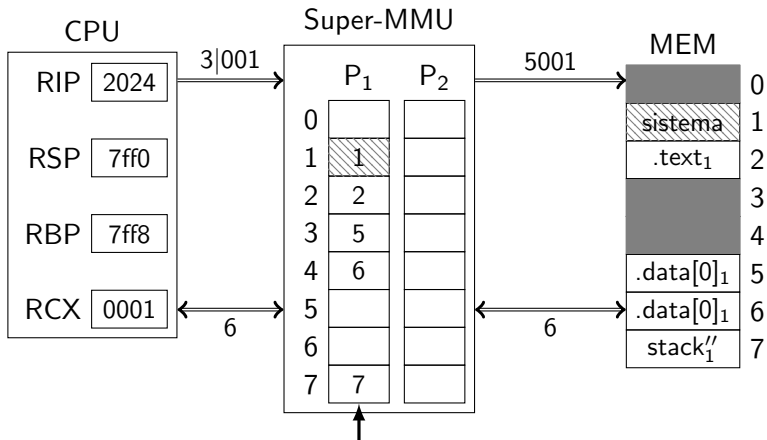
La CPU somma il contenuto di RCX e la costante 3000 (buf nel sorgente), ottenendo l'indirizzo 3001. Inizia dunque una operazione di lettura a questo indirizzo.



Ancora una volta la MMU scompone l'indirizzo in numero di pagina (3) e offset (001). L'entrata numero 3 della tabella di corrispondenza dice che la pagina 3 si trova nel frame 5.



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



Si noti come i dati di P<sub>1</sub> si trovavano nel frame 4 prima dello *swap-out*. Ora si trovano nel frame 5, ma il programma continua a funzionare come se niente fosse.