ARCHITETTURA DEGLI ELABORATORI

A.A. 2020-2021

Università di Napoli Federico II
Corso di Laurea in Informatica

Docenti
Proff. Luigi Sauro gruppo 1 (A-G)
Silvia Rossi gruppo 2 (H-Z)

MEMORIE

La Memoria Virtuale

Agli albori dell'informatica la memoria centrale di un computer raramente superava le 4k parole ed i sistemi operativi erano, a dir poco, essenziali.

A quell'epoca, chiunque si disponesse a scrivere un programma doveva prima informarsi di quale fosse la disponibilità di spazio nella memoria centrale del sistema da adoperare, al netto delle necessità della CPU e della "parte residente" del sistema operativo.

Fatto il conteggio doveva scrivere un programma fatto di pezzi da mandare in esecuzione uno per volta (***overlay,* sovrapposizione**), registrando i risultati parziali in gruppi di memoria i cui indirizzi fossero noti anche ai successivi pezzi di programma.

Questi li prelevavano per usarli ed, a loro volta registravano i risultati in locazioni prefissate.

Senza questa procedura il programma non poteva essere eseguito.

La Memoria Virtuale

Tutto questo è oggi completamente ignoto agli utilizzatori di computer.

Il fatto che talvolta siano indicati dei requisiti minimi del sistema per usare un certo programma, è un semplice suggerimento per ottenere dal programma un comportamento ragionevole, senza lunghissimi tempi di attesa tra due operazioni successive.

Oggi è assolutamente normale che programmi che tra occupazione diretta ed indiretta avrebbero bisogno di 25-30 Mbyte girino su macchine con memoria centrale da 16 Mbyte, grazie alla tecnica della "***memoria virtuale***", inventata da un gruppo di ricercatori inglesi di Manchester nel 1961 ed adottata dall'IBM negli anni '70.

La tecnica si avvale di strutture hardware che è fondamentale che siano nel circuito integrato della CPU e ciò spiega perché essa è apparsa nei sistemi basati su microprocessori "single chip" relativamente di recente.

La gestione è affidata al sistema operativo

In estrema sintesi la tecnica della *memoria virtuale* prevede che il sistema operativo assegni una parte degli spazi liberi nella memoria centrale ai segmenti di un programma istallato su disco.

Il sistema operativo seguirà criteri prefissati, assegnando uno spazio compreso tra un minimo ed un massimo, in funzione della dimensione del programma.

Gli spazi potranno essere anche non continui.

Il risultato è che programmi piccoli saranno trasferiti interamente ma, naturalmente, in funzione della mappa di occupazione corrente potranno essere dislocati in maniera frammentata in qualsiasi parte della memoria.

La memoria è gestita dinamicamente

Programmi di grandi dimensioni avranno a disposizione spazi insufficienti a contenerli completamente e saranno soggetti a continue operazioni di "***swapping***", cioè di sostituzione dinamica dei loro segmenti nello spazio di memoria centrale assegnato.

L'operatore può imporre al sistema operativo di dedicare più spazio ad un programma, ma se esso usa una gran parte del campo d'indirizzamento del processore, non potrà in ogni caso essere completamente trasferito in memoria.

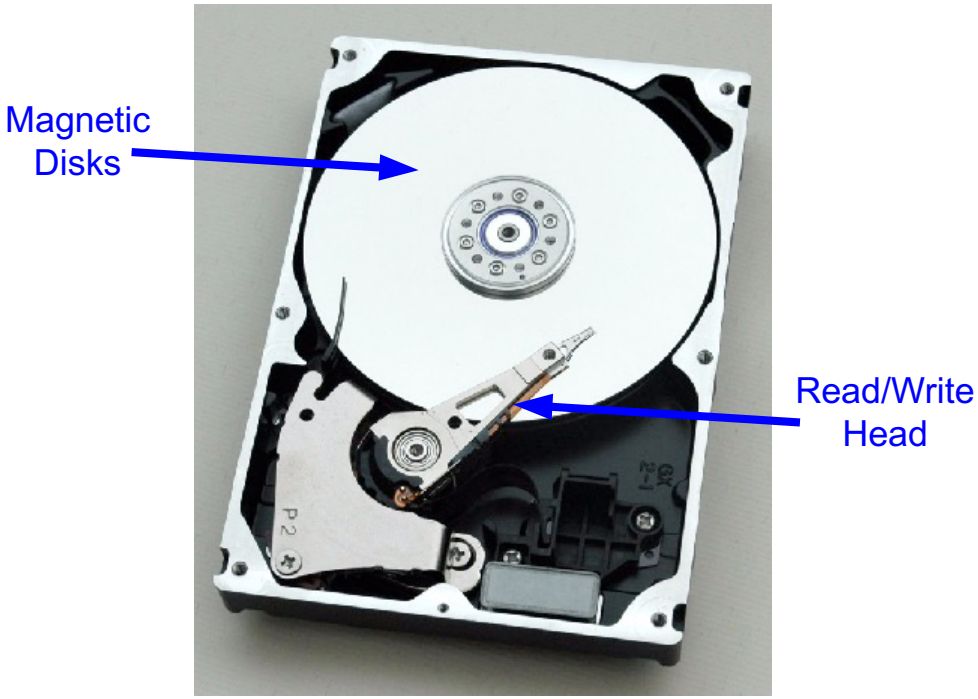
La tecnica di gestione degli spazi ha molti punti in comune con il funzionamento di una memoria cache.

Anche in questo caso, infatti, se gli spazi a disposizione nella memoria principale sono saturati, e il programma accede ad una locazione di memoria che non corrisponde ad un indirizzo esistente nella memoria centrale, si deve ricorrere ad un algoritmo di sostituzione.

Dimensioni della pagina di memoria

La parte di programma contenente l'indirizzo richiesto viene prelevato dal disco (trasferimento in DMA) e sostituita in memoria ad una delle parti che vi risiedevano.

Poiché il tempo di accesso al disco costituisce una penalità di fallimento molto più pesante, in termini di tempo rispetto a quella di una cache, l'entità minima da trasferire è presa molto più grande del blocco della cache ed è una *pagina di memoria* di solito compresa tra 2k e 16k; normalmente in un accesso al disco vengono scambiate molte pagine.



Takes milliseconds to *seek* correct location on disk

Virtual Memory

- **Virtual addresses**
  - Programs use virtual addresses
  - Entire virtual address space stored on a hard drive
  - Subset of virtual address data in DRAM
  - CPU translates virtual addresses into ***physical addresses*** (DRAM addresses)
  - Data not in DRAM fetched from hard drive

Cache/Virtual Memory Analogues

Cache	Virtual Memory
Block	Page
Block Size	Page Size
Block Offset	Page Offset
Miss	Page Fault
Tag	Virtual Page Number

Physical memory acts as cache for virtual memory

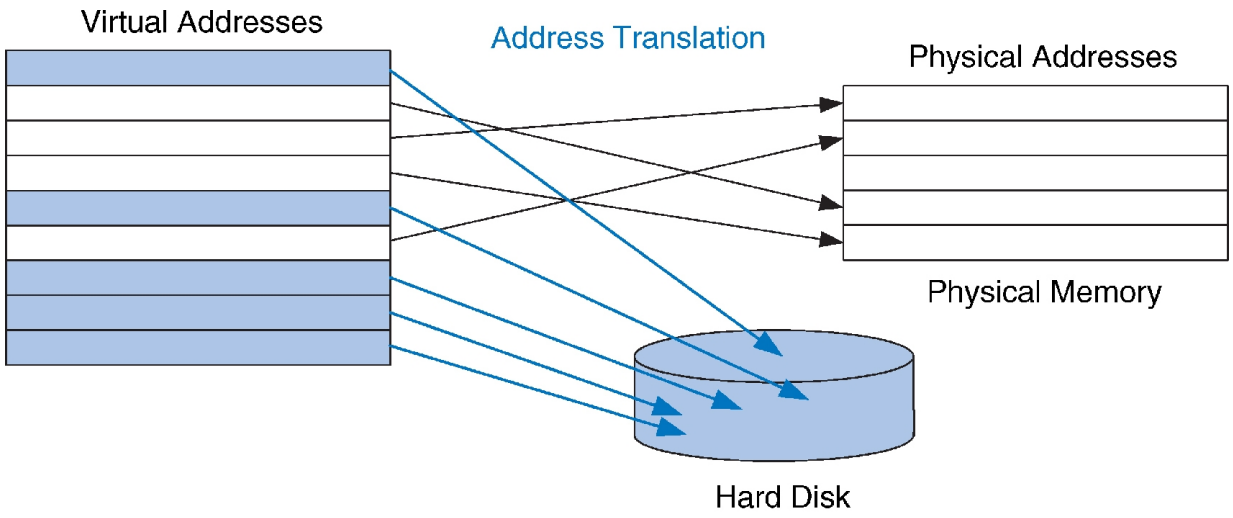
Definizioni per la memoria virtuale

- **Page size:** dimensione dei segmenti di memoria trasferiti dalla memoria di massa alla main memory (2K-16K)

- **Address translation:** determinare l'indirizzo fisico nella main memory a partire da un indirizzo virtuale

- **Page table:** lookup table usata per tradurre gli indirizzi virtuali in indirizzi fisici

Indirizzi fisici e virtuali



- **Data la grandezza delle pagine e la località spaziale e temporale durante l'esecuzione di un programma, la maggior parte degli accessi sono ritrovati nella main memory**
- **Ma i programmi sfruttano la maggior capacità della memoria virtuale**

Virtual Memory Example

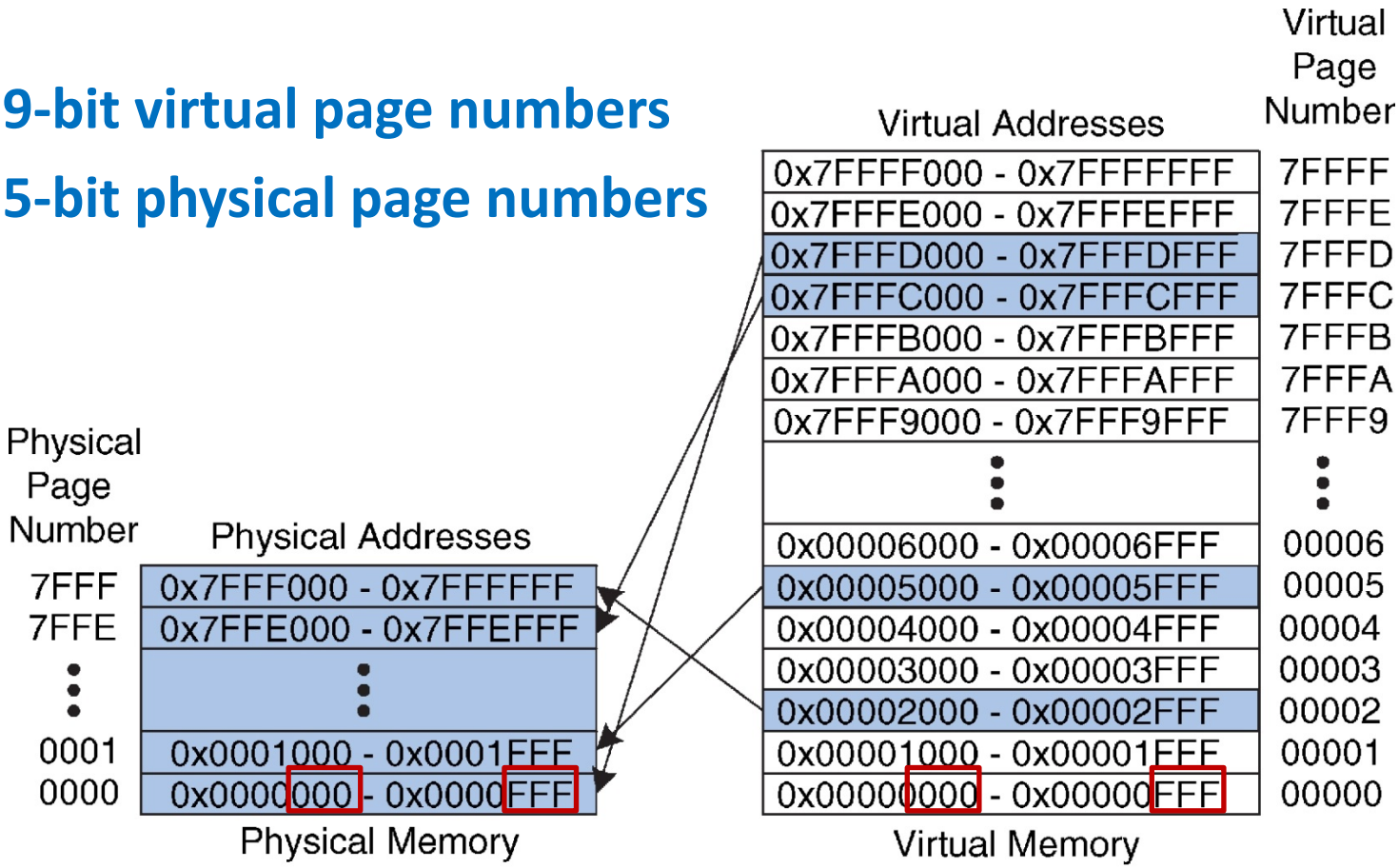
- **System:**
  - Virtual memory size: 2 GB = 2^{31} bytes
  - Physical memory size: 128 MB = 2^{27} bytes
  - Page size: 4 KB = 2^{12} bytes

Virtual Memory Example

- **System:**
  - Virtual memory size: 2 GB = 2^{31} bytes
  - Physical memory size: 128 MB = 2^{27} bytes
  - Page size: 4 KB = 2^{12} bytes
- **Organization:**
  - Virtual address: **31** bits -> il 32 è a 0
  - Physical address: **27** bits -> gli altri 5 a 0
  - Page offset: **12** bits
  - # Virtual pages = $2^{31}/2^{12}$ = $\mathbf{2^{19}}$ (VPN = **19** bits)
  - # Physical pages = $2^{27}/2^{12}$ = $\mathbf{2^{15}}$ (PPN = **15** bits)

Virtual Memory Example

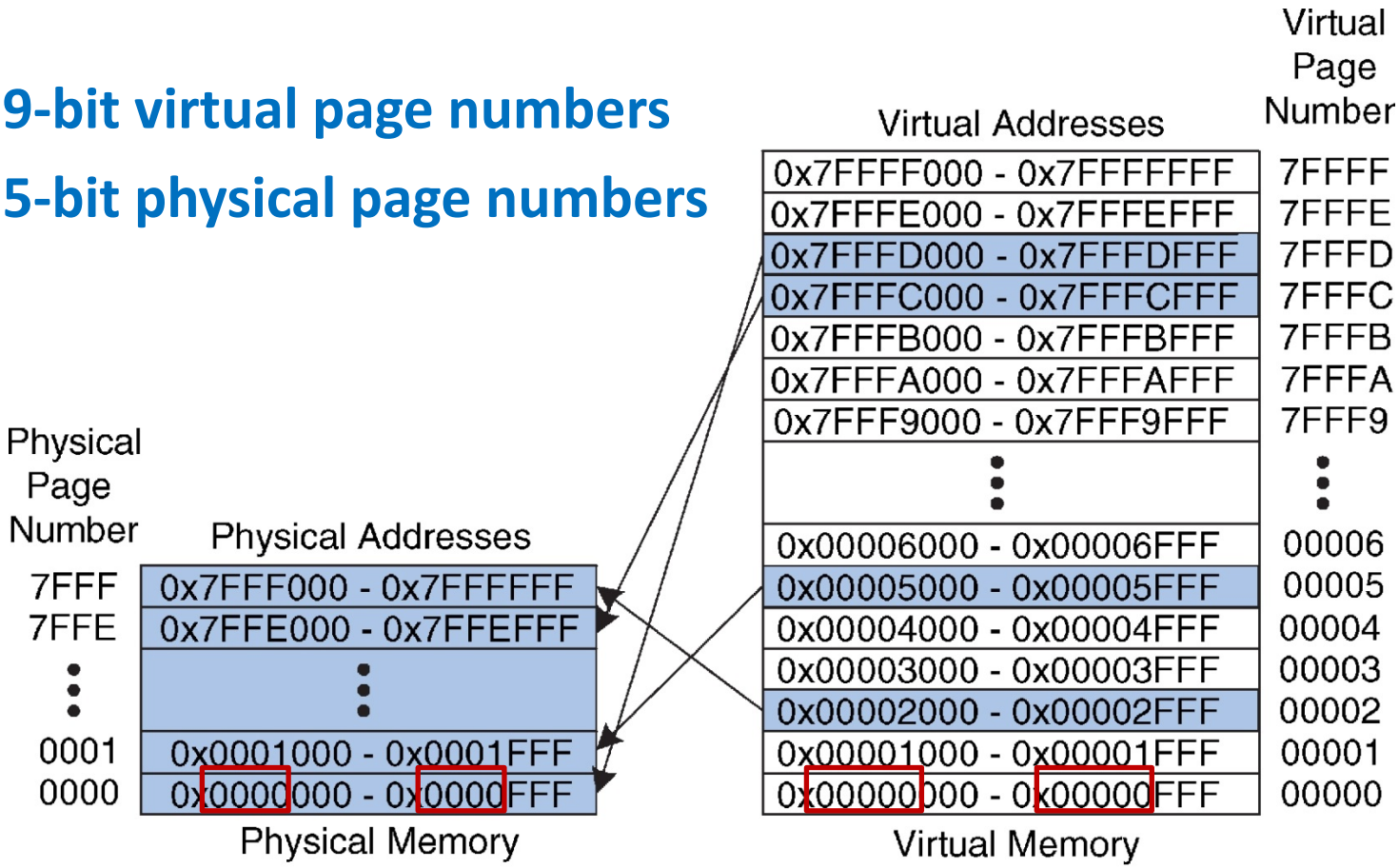
- **19-bit virtual page numbers**
- **15-bit physical page numbers**



Page 4K= 12 bit=3 esadecimali -> numero di parola (spiazzamento di pagina)

Virtual Memory Example

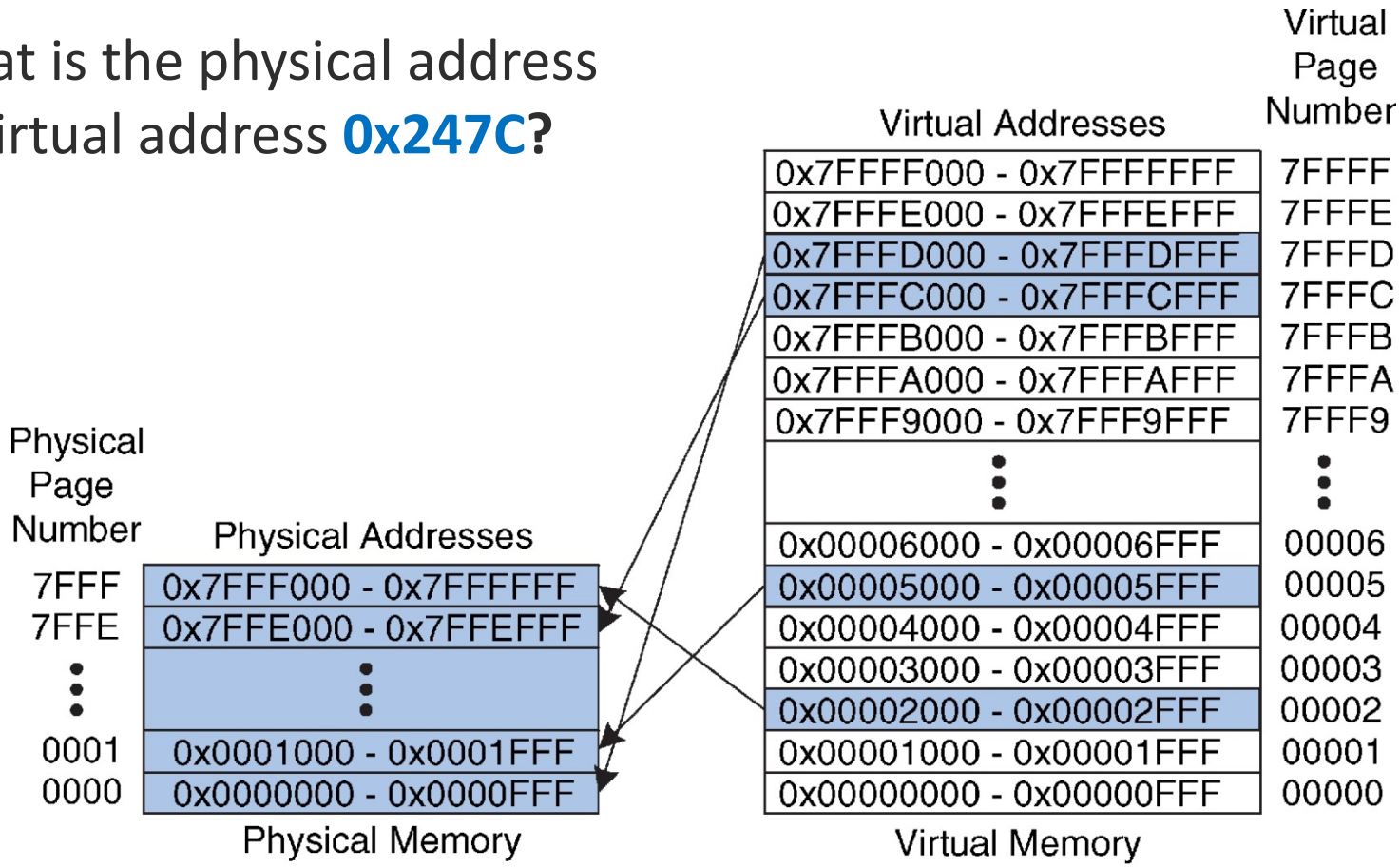
- **19-bit virtual page numbers**
- **15-bit physical page numbers**



Virtual Memory size 2G= 31 bit, quindi 8 esadecimali con il più significativo che arriva fino a 7

Virtual Memory Example

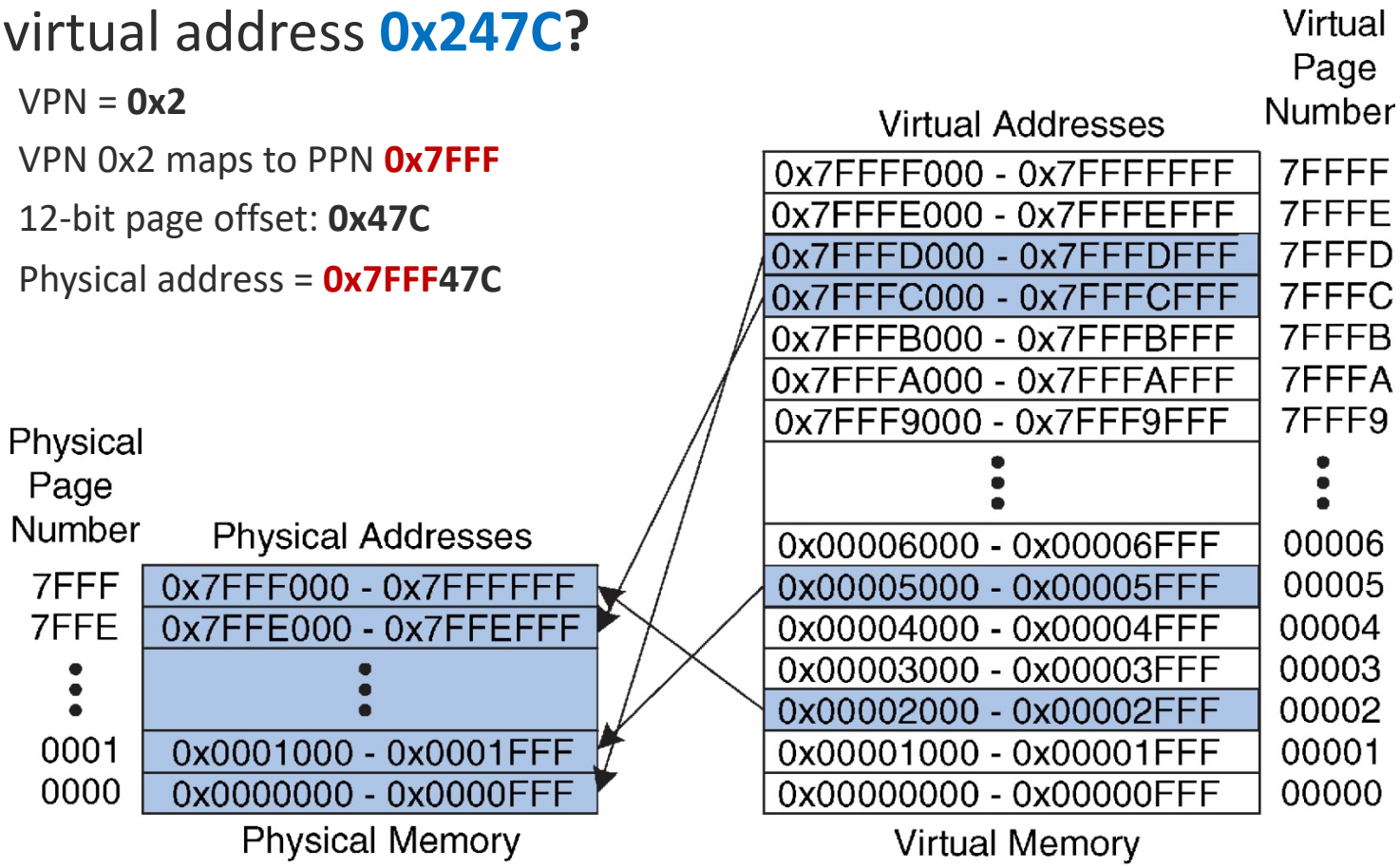
What is the physical address of virtual address **0x247C**?



Virtual Memory Example

What is the physical address of virtual address **0x247C**?

- VPN = **0x2**
- VPN 0x2 maps to PPN **0x7FFF**
- 12-bit page offset: **0x47C**
- Physical address = **0x7FFF47C**



La Memory Management Unit (MMU)

Quello a cui fa riferimento il processore nell'elaborazione è detto indirizzo "***virtuale***" o logico ed ha come unica limitazione di essere entro il campo di indirizzamento massimo del processore.

Il compito di tradurre questo indirizzo in un indirizzo che punti ad una locazione effettivamente esistente nel sistema (***indirizzo fisico***) è demandato ad una struttura logica contenuta nel chip del processore, che si chiama **Memory Management Unit (MMU)** e che opera sotto il controllo software del sistema operativo.

Traduzione degli indirizzi

La memoria virtuale si comporta come una sorta di cache completamente associativa per il disco.

Tuttavia, se si gestisse la memoria virtuale come le cache, ossia tramite un comparatore sui bit più significativi per verificare la corrispondenza delle pagine, servirebbe un numero improponibile di comparatori, dato l'elevato numero di pagine in cui è organizzata la RAM.

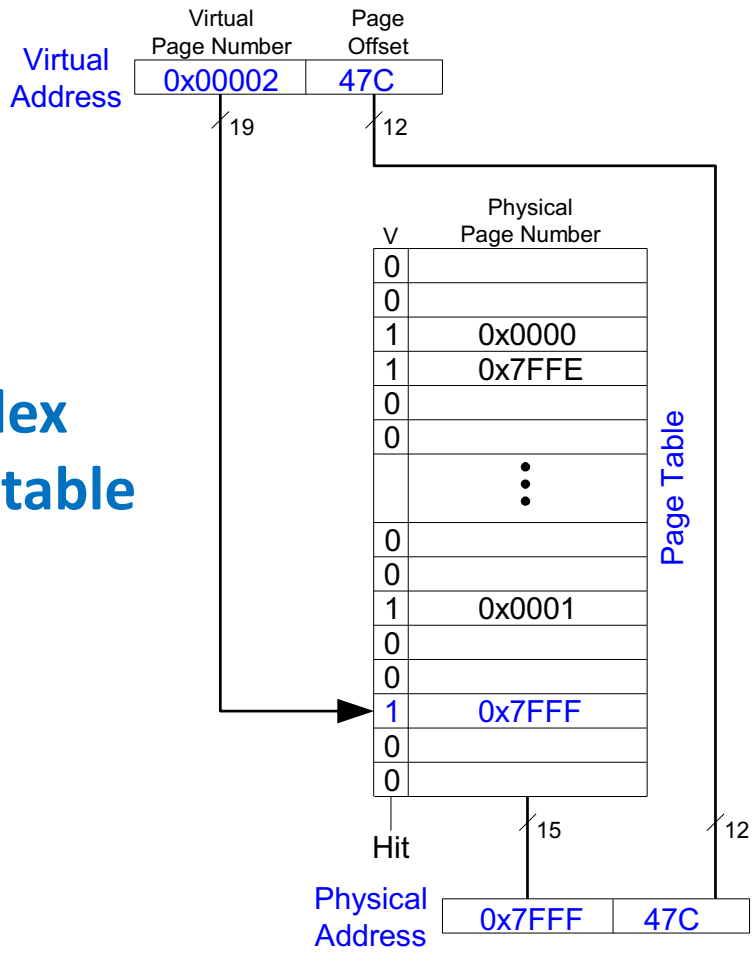
In alternativa, il processo di traduzione degli indirizzi è gestito mediante **tabelle di paginazione**.

La page table

- Contiene un entry per ogni pagina virtuale
  - Memorizzata in memoria fisica
  - Ex: array con indice il numero di pagina virtuale
- Campi di una entry:
  - **Valid bit:** 1 se la pagina è presente nella main memory
  - **Physical page number:** dove la pagina è locata nella main memory

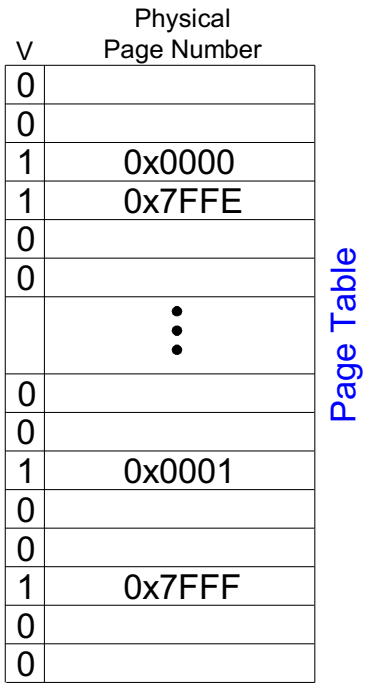
Page Table Example

VPN is index into page table



Page Table Example 1

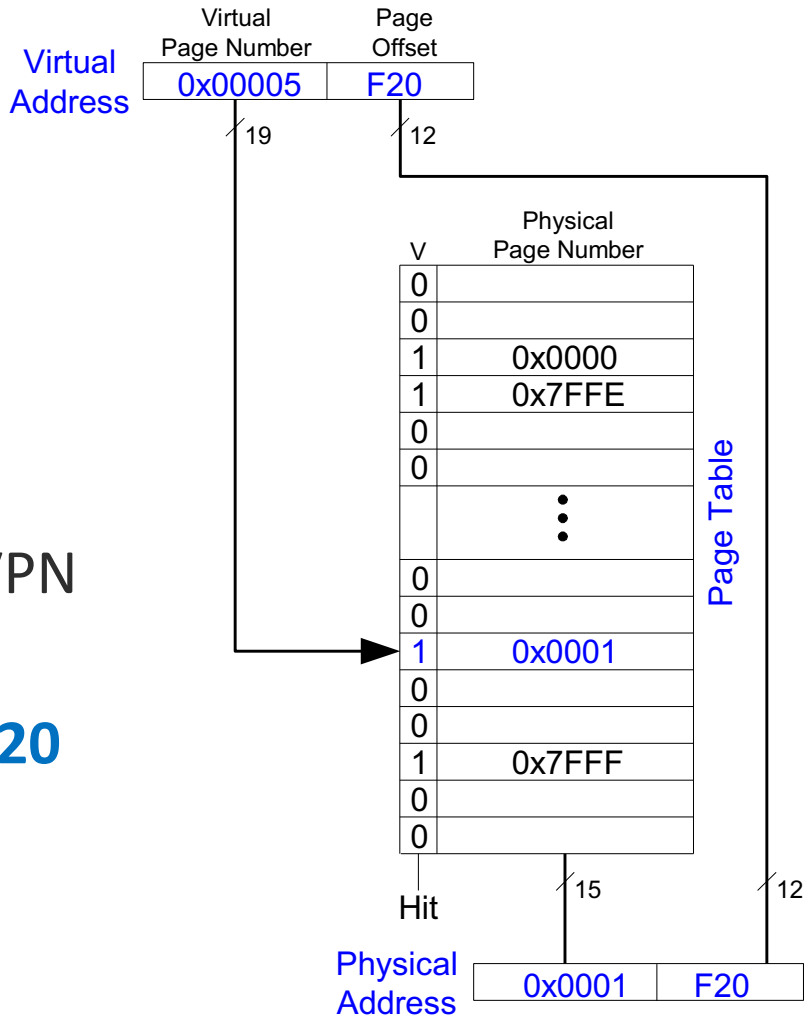
What is the physical address of virtual address **0x5F20**?



Page Table Example 1

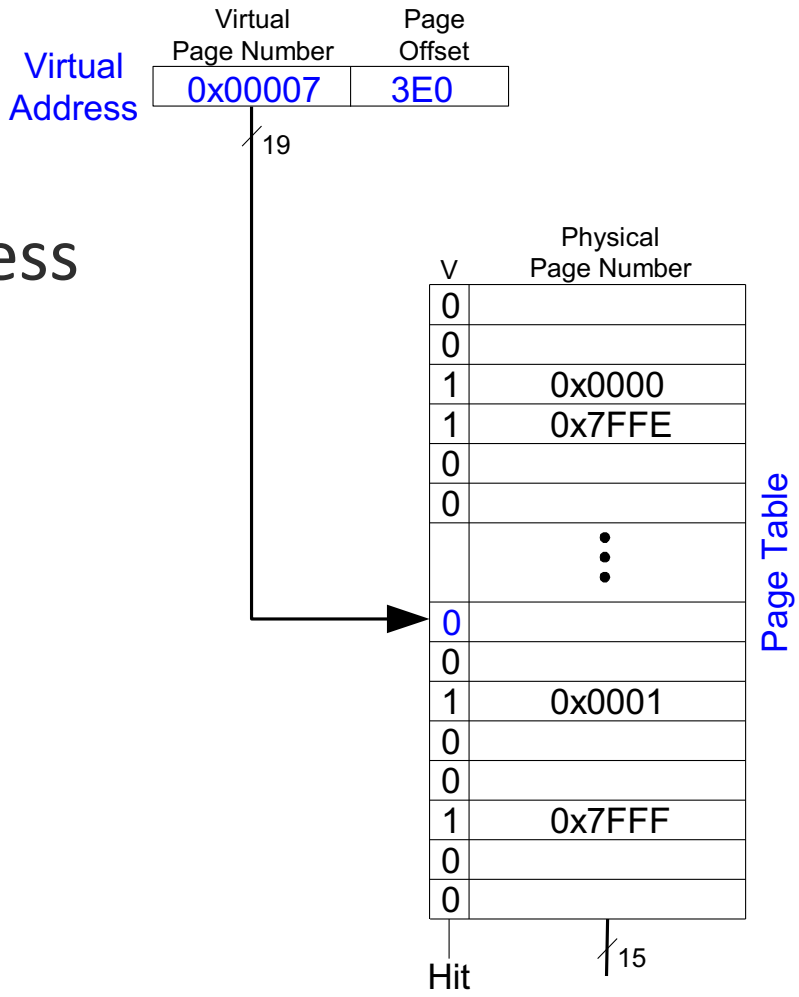
What is the physical address of virtual address **0x5F20**?

- VPN = **5**
- Entry 5 in page table VPN 5 => physical page **1**
- Physical address: **0x1F20**



Page Table Example 2

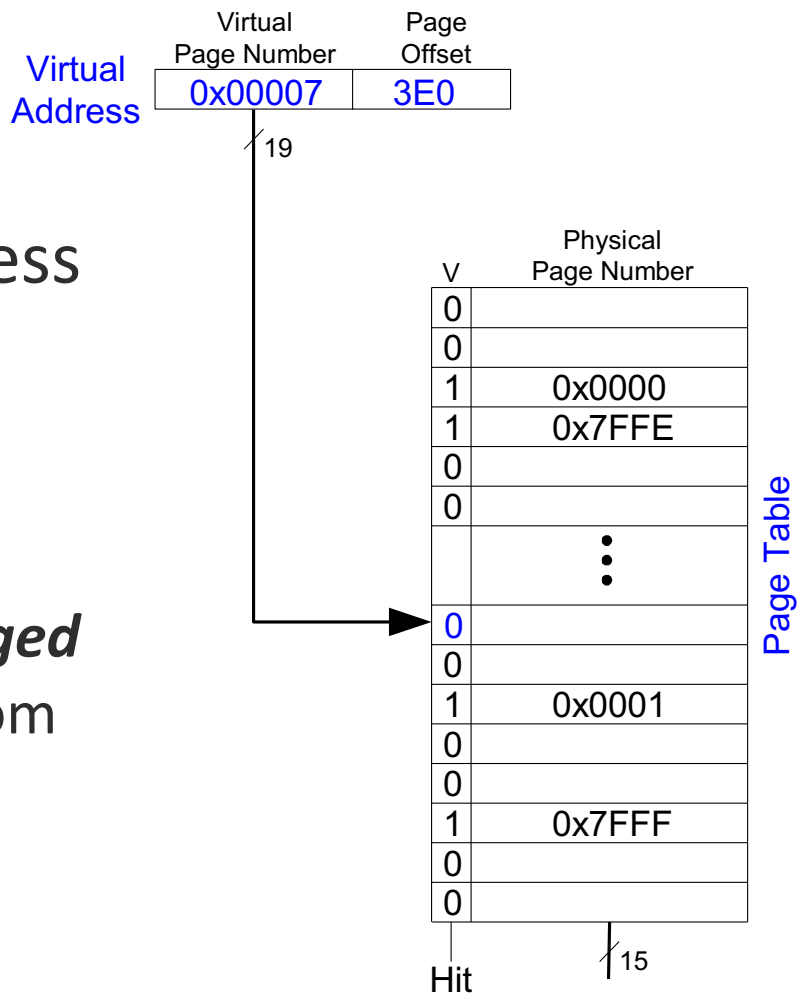
What is the physical address of virtual address **0x73E0**?



Page Table Example 2

What is the physical address of virtual address **0x73E0**?

- VPN = **7**
- Entry 7 is invalid
- Virtual page must be ***paged*** into physical memory from disk



Page Table Challenges

- **Page table is large**
  - usually located in physical memory
- Load/store requires 2 main memory accesses:
  - one for translation (page table read)
  - one to access data (after translation)
- Cuts memory performance in half
  - ***Unless we get clever…***

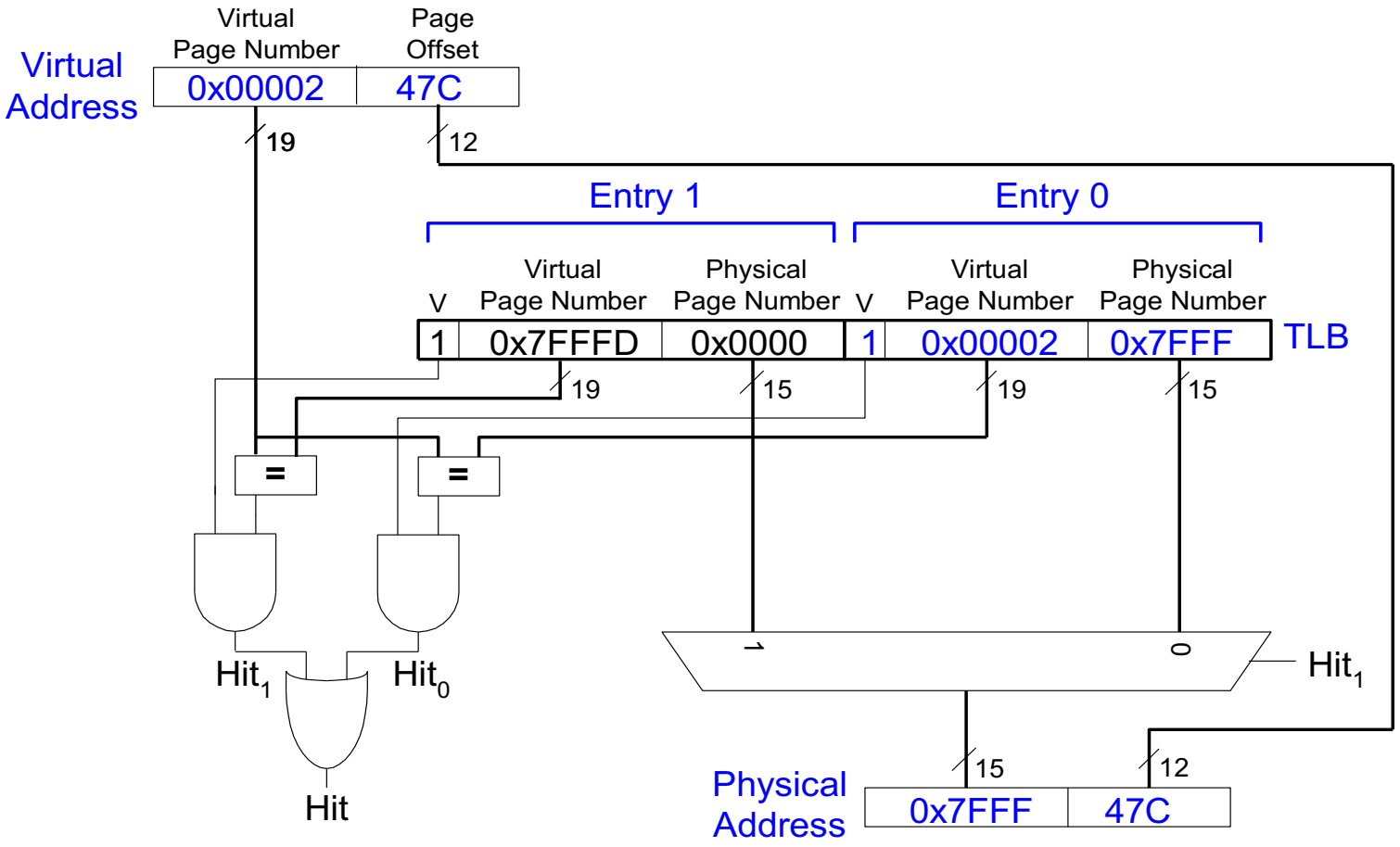
Translation Lookaside Buffer (TLB)

- Small cache of most recent translations
- Reduces # of memory accesses for *most* loads/stores from 2 to 1

TLB

- Page table accesses: high temporal locality
  - Large page size, so consecutive loads/stores likely to access same page
- TLB
  - Small: accessed in < 1 cycle
  - Typically 16 - 512 entries
  - Fully associative
  - > 99 % hit rates typical
  - Reduces # of memory accesses for most loads/stores from 2 to 1

Example 2-Entry TLB



Memory Protection

- Multiple processes (programs) run at once
- Each process has its own page table
- Each process can use entire virtual address space
- A process can only access physical pages mapped in its own page table

Virtual Memory Summary

- Virtual memory increases **capacity**
- A subset of virtual pages in physical memory
- **Page table** maps virtual pages to physical pages – address translation
- A **TLB** speeds up address translation
- Different page tables for different programs provides **memory protection**

Esercizi sulle cache

Domanda: Si ipotizzi che 4/5 delle istruzioni in un programma effettui un'operazione di scrittura o di lettura in RAM, e che la frequenza di successo di lettura in cache per le istruzioni sia 2/3 di quella per i dati. Si supponga, inoltre, che la penalità di fallimento sia la stessa per operazioni di scrittura e operazioni di lettura. Si supponga che, in cicli di clock, una lettura diretta in memoria costi 10, una in cache costi 1 e che una in entrambe (sia in memoria, che in cache) costi 16. Quale deve essere il tasso di successo dei dati per avere un guadagno maggiore di 1?

Date n istruzioni, costo senza cache è: (1 + 4/5)* n *10 = 18 n
Costo con cache: n (2/3 * x * 1 + (1- 2/3 * x) 16) + 4/5 n (x* 1+ (1-x) * 16)

.

$$G=\frac{18}{[16-10x+0{,}8(16-15x)]}=\frac{18}{28.8-22x}>1$$

$$22x>10.8$$
$$x>0.49$$

Esercizi cache

Domanda 2 (Punti 5): Si consideri una memoria cache set-associativa da 2048 parole, suddivisa in 128 blocchi da 16 locazioni ciascuna, in cui si sceglie di avere 4 blocchi per insieme, ossia 32 insiemi. Sia data l'indirizzo 0x000244F4, in quale insieme della cache essa sarà inserita e con quale etichetta? Analogamente, quali sono insieme ed etichetta se i blocchi per insieme fossero 2?

0x000244F4 = 0000 0000 0000 0010 0100 0100 1111 0100

0000 0000 0000 0010 0100 0100 1111 0100

11 01 → Parola nel set = 13

10011 → Set = 19

0000 0000 0000 0010 0100 0 → Etichetta = 72

Esercizi cache

Domanda 2 (Punti 5): Si consideri una memoria cache set-associativa da 2048 parole, suddivisa in 64 blocchi da 32 locazioni ciascuna, in cui si sceglie di avere 2 blocchi per insieme, ossia 32 insiemi. Sia data la parola 0x0000D33C , in quale insieme della cache essa sarà inserita e con quale etichetta?

0x0000D33F = 0000 0000 0000 0000 1101 0011 0011 1100

011 11 → Parola nel set = 15

0011 0 → Set = 6

0000 0000 0000 0000 1101 → Etichetta = 13