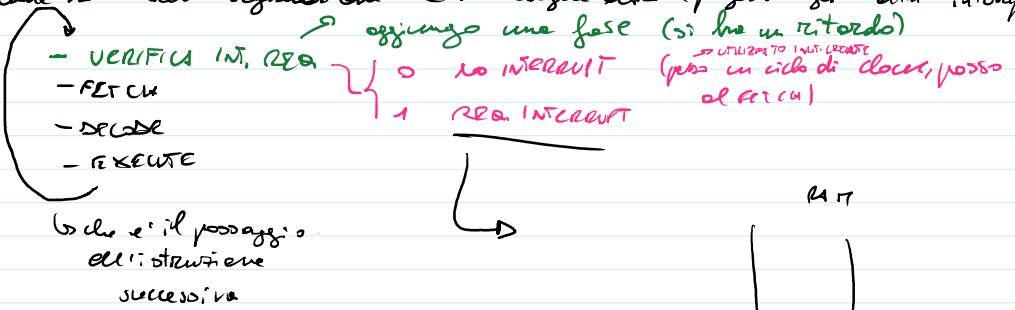


L'informazione del filo aggiunto è uno  $\Rightarrow$  che determina l'interruzione della CPU

Il meccanismo si ottiene in fasi:

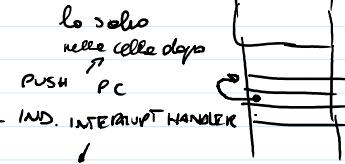
- **SEGNAZIONE HW**: implementato a livello hardware tramite il filo aggiuntivo, e giunge da I/O a CPU
- **RISPOSTA**: viene salvato lo stato dell'esecuzione corrente (es. salvare i dati sullo stack) e viene inviato un nuovo programma: gestore delle interruzioni (interrupt handler)
- **GESTIONE**
  - { - IMMEDIATA → uno o più interrotti solo da priorità più alta (gestione urgente e si utilizza poco tempo)
  - DIFFERITA → prende nota della segnalazione ma non esegue altro (preferisce gestire altri interrotti)
    - aggiunge una fase (si ha un ritardo)
    - VERIFICA INT, RIC
    - FET PC
    - DECODE
    - EXECUTE

\* da qui elabora le istruzioni nel seguente modo



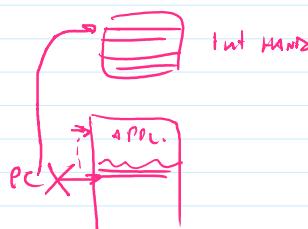
Il programma può essere interrotto alle fine dell'elaborazione dell'istruzione o prima da inizio all'elaborazione di un'altra istruzione

uno ciclo di clock in più



indirizzo del programma di gestione interruzioni

il passo successivo è l'esecuzione delle prime istruzioni del programma di gestione interruzioni e viene eseguito il return per le sue istruzioni

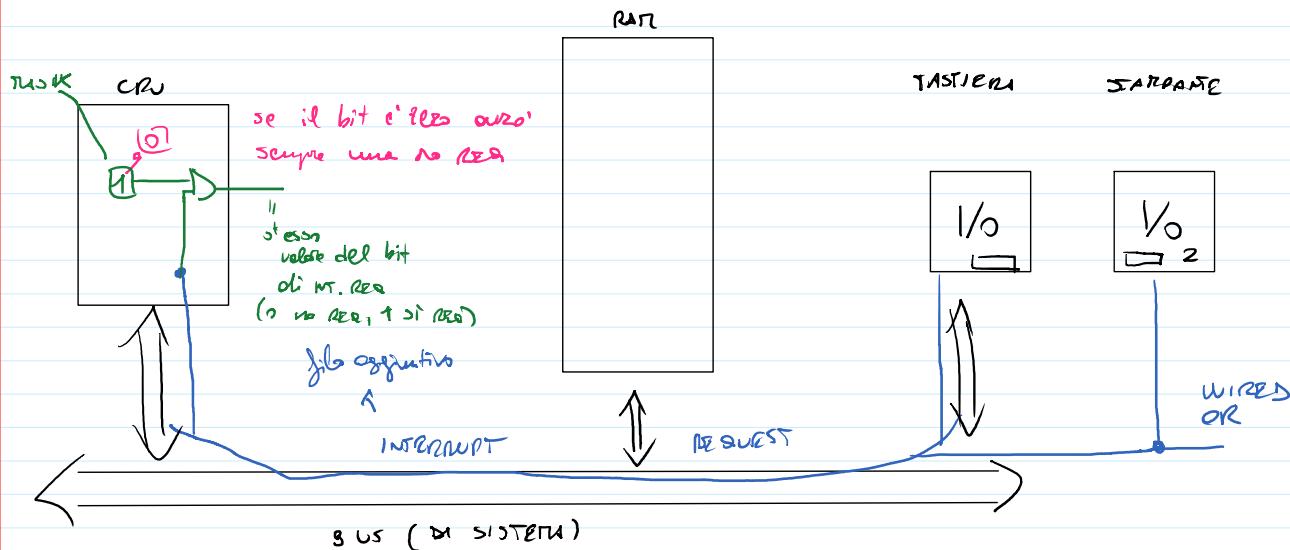


RISPOSTA DELL'CPU  
(la gestione vera e propria dell'interruzione avviene a livello software (intervaller) e non più hardware)

durante la gestione dell'interruzione non ho accesso il dispositivo I/O, quindi la richiesta di interruzione è ancora vigente: viene salvato PC (interrupt handler) e ritorno alla prima istruzione, senza dare il tempo di finire la gestione (non esegue neanche la 2^ istruzione)

Possò intervenire aggiungendo un bit detto al registro di stato, detto "bit di modifica"

dell'interruzione)



dunque modifica: inizialmente  $\text{MASK} = 1$  quando arriva la richiesta

PUSH PC  
 $PC \leftarrow \text{IND. INT. HANDLER}$   
 $\underline{\text{MASK}} \leftarrow 0$

così facendo posso passare all'esecuzione  
 dopo dell'interrupt handler  
 (non deve infatti essere interrotto da una richiesta  
 di interruzione)

Sarà il gestore di interruzione a che interroghi i dispositivi I/O  
 per vedere chi ha mandato la richiesta di interruzione (ve ed osservare  
 il registro mappato in memoria)

↳ N.B. Posso egualmente gestire efficientemente l'interruzione con I/O :

ed esempio c'è un programma in corso, viene interrotto dalla stampante  
 che richiede lo stampo del carotter, finisca lo stampo, mentre si  
 esegue la stampante, riprende il programma poi, prende la stampante  
 finisce subito una nuova interruzione per stampare un altro carotter e così via

Nel caso di 2 richieste di interruzione (de entrambi i dispositivi), come  
 gestisco le priorità?

Il criterio seguito è: in base ai tempi di funzionamento dei dispositivi  
 e cosa succede se un dispositivo non gestisce per tempo la richiesta.

T.SISTEMA

Se non leggo per troppo rischio  
 di sovraccarico, perdere  
 dei caratteri  
 (es. scritti in sequenza,  
 non riesce a stampare  
 il primo, allora si prende  
 il secondo che non  
 riesce a stampare etc.)

STAPPANTE

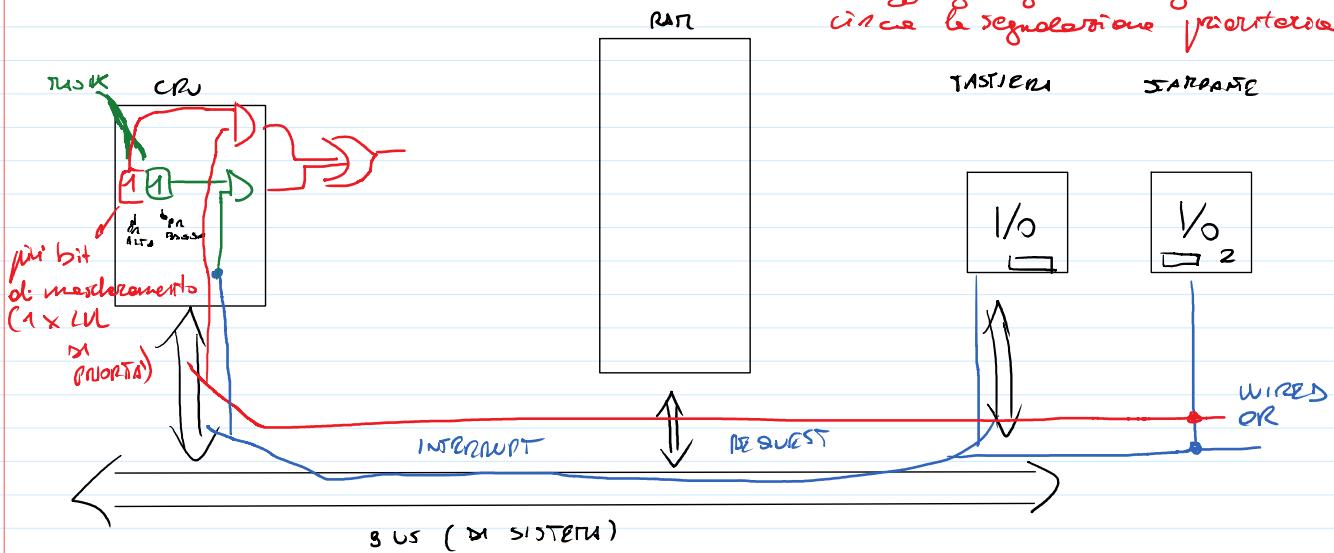
Se stampate iniziate una INTERRUPT REA  
 e attendete la risposta per poter  
 stampare altro

Si inserisce il concetto di PRIORITÀ:

il primo, allora si pensa  
il secondo che non  
riesce a superare esso,  
ella fine basta l'ultimo  
carattere)

si inserisce il canale di PRIORITA':  
avendo priorità il dispositivo con  
più bassa istanza

(si aggiungono fili con info  
circa la segnalazione prioritaria)



Se gestore d'interruzione va a vedere l'info del file di priorita e pone i valori  
dei bit di mask  $\rightarrow$  (no resp.)

es. PRIORITA'

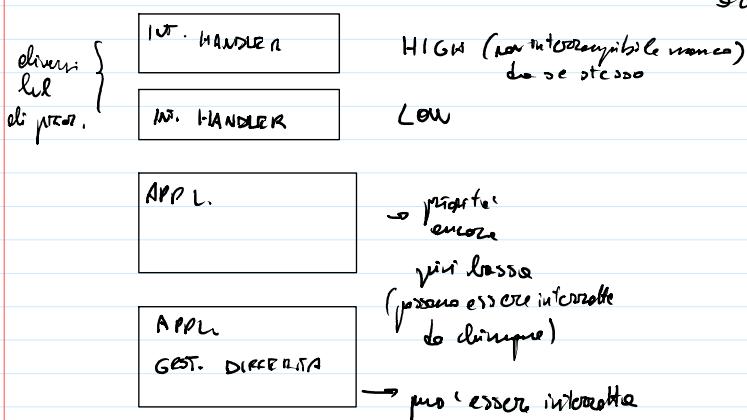
- ALTO  
- BASSO

$\rightarrow$  2 bit di  
masking

Inizia l'esec. del gestore d'interr.  
poi task possono es. CTA

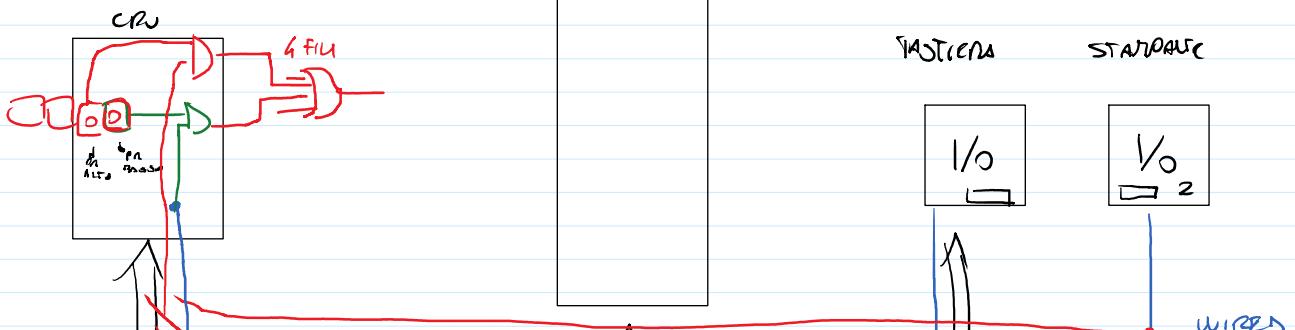
po. task a  
completato  
in no era  
completato  
le risposte

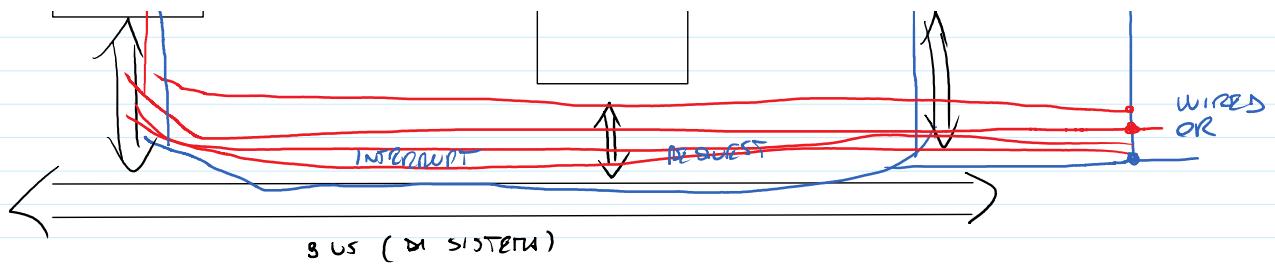
Nel caso di un'interruzione di gr. bassa  
e può essere interrotto solo da quelle  
superiori entrambi VDT



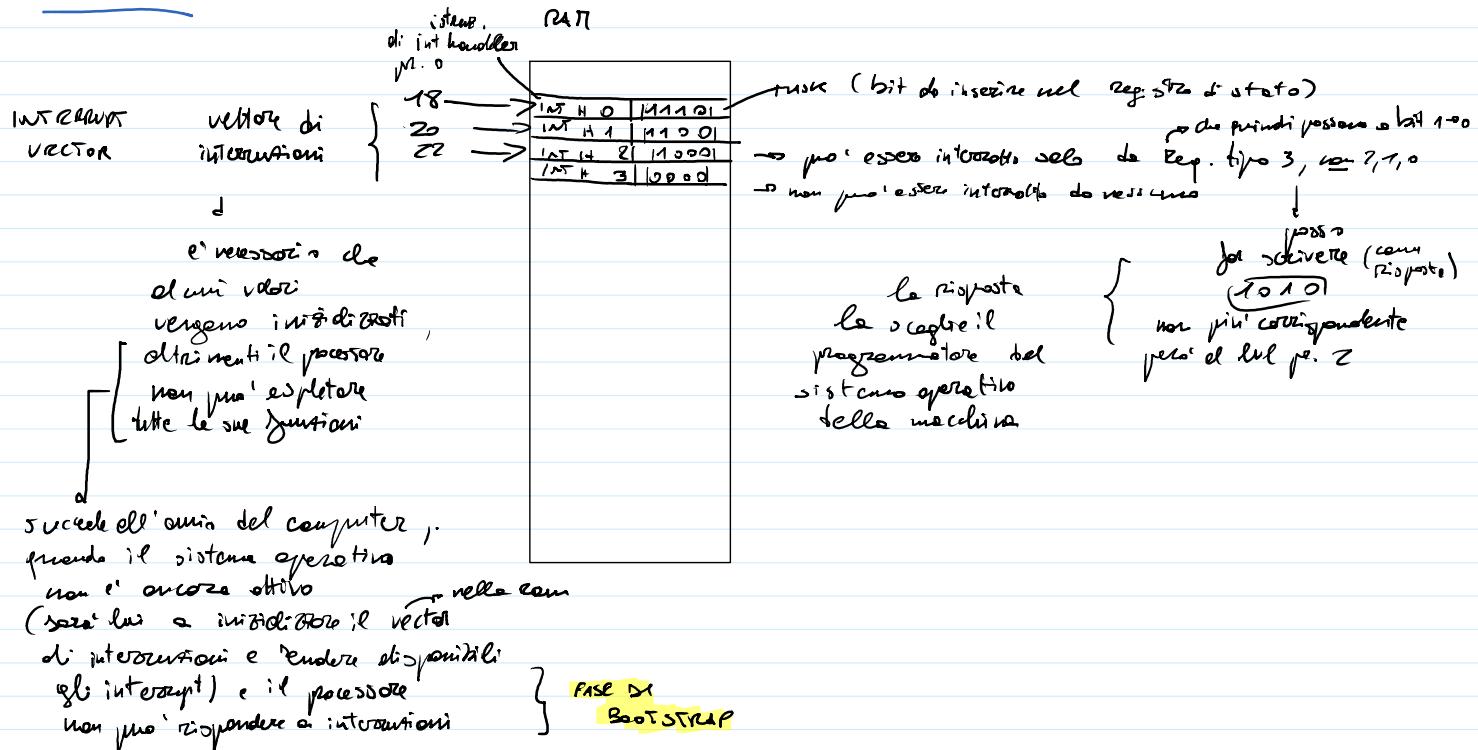
Con 4 livelli di priorita':

### • livello HW

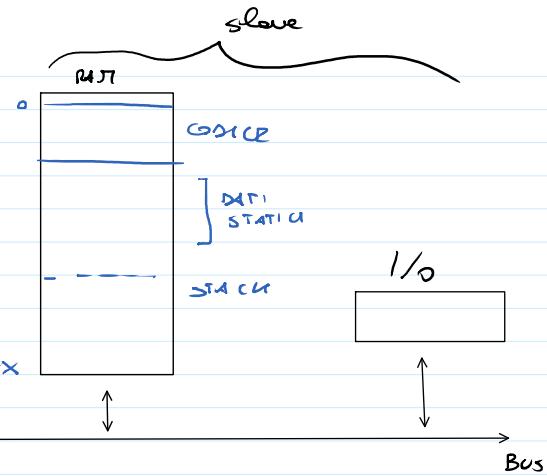
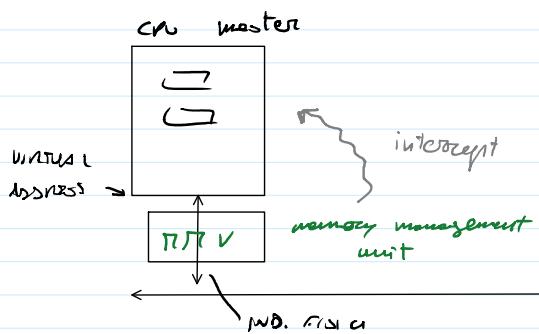




### livello ca



Per ottimizzare meglio l'utilizzo della memoria



La RAM non ha celle specificate a contenere solo certi tipi di istruzioni, proprio per una ragione di efficienza (granti programmi = più spazio)

La razionalizzazione dell'uso delle RAM  
attraverso gestione alla virtual memory,  
tramite un dispositivo fisico, MMU.

Lo stesso offre una "traduzione": CPU vede indirizzi virtuali,  
tradotti da MMU in indirizzi fisici (utilizzati x  
accedere alle RAM, ai dispositivi I/O ecc...)

Se vale per la CPU, ciò non vale per I/O, infatti  
questo meccanismo risulta pesante

[Il PDP-11 fornisce quindi una struttura,  
lasciando le "ultime" celle allo stesso,  
le intermedie ci dati statici ecc...]

Vi sono 2 principali tecniche di  
traduzione

→ segmentazione

→ impaginazione

### • SEGMENTATION

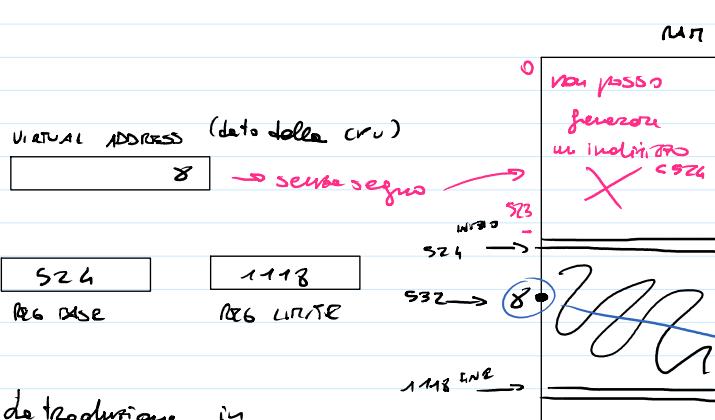
La RAM viene virtualizzata in modo tale che la CPU ne veda più d'una (piccolo numero)

La parola di memoria della RAM viene usata quindi solo per un certo tipo di dati, es.

il segmento STACK, il segmento CODE (dove si effettua il RET), il segmento dei DATI STATICI ecc... Posso forzare la CPU a vedere una certa parte della RAM.

Per farlo si introducono dei controlli: nella MMU si verifica il vincolo al segmento di memoria; es. nel seg. CODE, CPU può accedere in lettura solo durante la fase del RET.

(Altrimenti posso lanciare delle intercept come VIOLAZIONE DI REGOLE, entendo eventuali danni provocati da programmi che non devono operare su quel segmento)



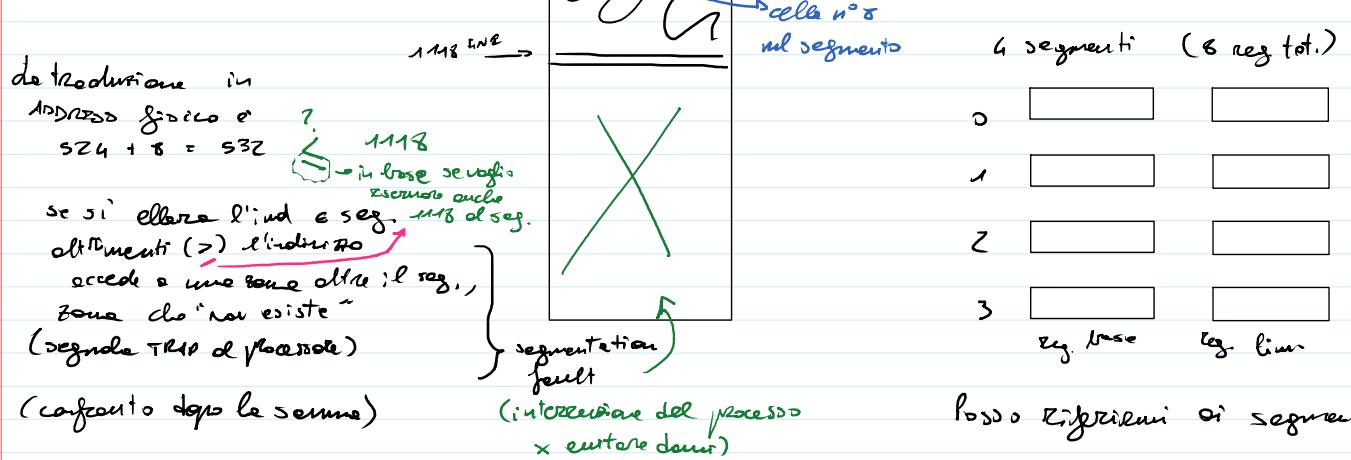
Una variante sarebbe

DIREZIONE SELGATO 1118 - 524

SEGMENTO  
celle n° 8  
nel segmento

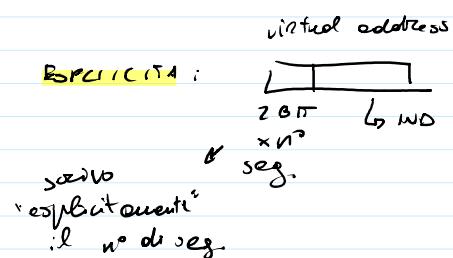
6 segmenti (6 reg tot.)

e l'operazione  
di somma viene anticipata  
dal compilatore  
(indirizzo è DIR  
n'utile)



Possò riferirsi ai segmenti con:

segmentos. / esplicita  
 \ implicita



Il costo di implementazione varia in base  
 al n° di segmenti e il tot. di memoria.

La velocità varia in base alla tecnologia  
 (se è la stessa della CPU allora tempi ridotti)

(risparmio 2 bit)

**IMPLICATIVA:** in base all'istruzione

eseguita si fa riferimento a un certo seg. Es. Istruzione di ADD e ROR operano  
 sulla stack, il segmento non è  
 altro che il seg. di stack.

il segmento è realizzato  
 per funzionare in un certo  
 modo

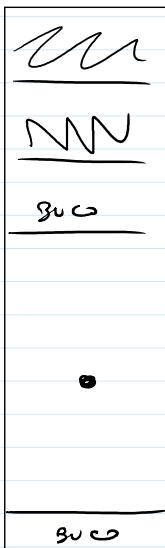
{ Es. CDR è in fase di FETCH, allora  
 il seg. su cui lavora è quello di codice

- oppure viene associato un segmento  
 a un certo registro, quando si lavora con tale  
 registro, si lavora su un certo segmento.

Se in esecuzione vengono mandati molti programmi, la memoria (non condivisa fra essi)  
 ha un limite e dovrà attendere il termine di esecuzione per mandarne altri

RAN

the solution desirable;



## Segmentation

Il problema è che i blocchi  
più o meno grossi ovviamente  
memoria non utilizzate  
(o i programmi sono troppo  
grossi per essere inseriti)  
(i), si dovrebbe quindi  
spostare la posizione dei  
segmenti (aggiustando  
gli indirizzi in reg. base e reg. lim.)

• se vivamente user  
e' difficile, lo c'  
invece lo RIALLOCRAZIE  
e tempo di esecuzione  
(valta del passo), ogni contenuto  
di esse viene spostato in pellegr  
super)

imposte una certa quantità di memoria libera  
individuabile

In base alle dimensione  
di richiede più  
tempo } } no efficace

Una tecnica per implementare il seg. STACK e' di riservare una quantita' di memoria prefissata, e aggiungere memoria (se ne ha libera) in caso di necessita'

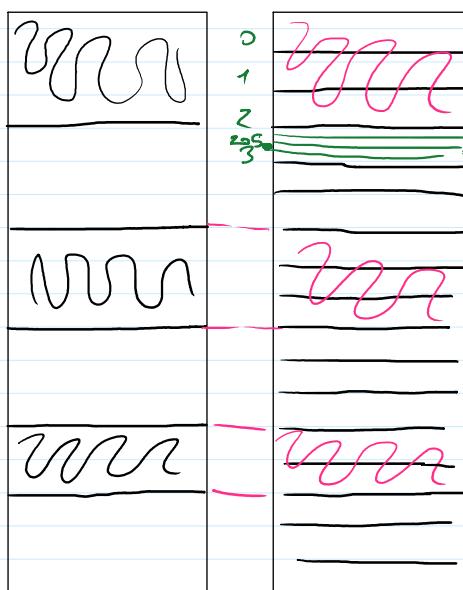
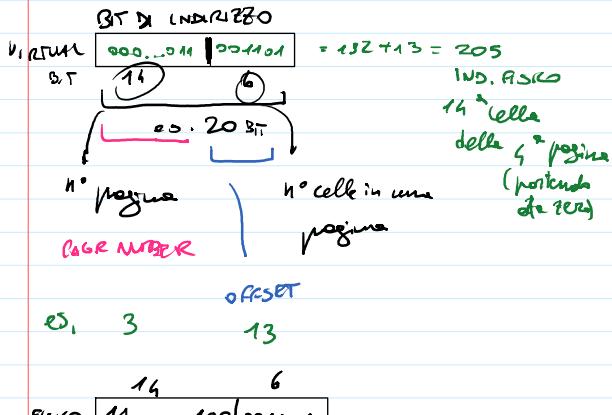
• tramite cui si TRASFERISCONO le interazioni  
(creando dinamica in base alle esigenze del programma)

La stessa tecnica puo' essere impiegata anche per altri seg. (a questo non concorre riscrivere grandi quantita' di memoria e altri seg.)

Per entrore la collocazione  
sia l'interpretazione.

## • PAGING

possò unire i 3 segmenti come  
tellocore? si con il PAGING



47 egrave (suddendo la

L di 2<sup>o</sup> celle  
L 1/4<sup>e</sup> celle di membrana  
L comprendendo le celle n. 0 )

(: celle libere

Per tradurre il n° di pagina  
da virtuale a fisico  
bisogna di un TELCO  
che mi associa le corrispondenze

Crea una tabella con il n° di pagina  
corrispondente all'indirizzo  
di memoria

  
 Posso "comportare" le pagine  
libere  
 ➔ posso usare in seguito  
anche la segmentazione

n° pagina virtuale	0	37
1	S1	
2	S	
3	722	
4	1	
$2^{14}$ pagine con 16 bit ( $2^{14}$ elementi nella tabella)	1	1

ho una tabella con il n° di pagina  
corrispondente all'indirizzo  
di memoria

- interpreta i bit come numero (dx)
- associa il n° (dx) al valore corrispondente (sx)  
(Sostituzione)

L'implementazione farà avere anche i "tag": più devo in legge a dare solo registrate  
le pagine e quanti sono

LA SUODIVISIONE CONTA:

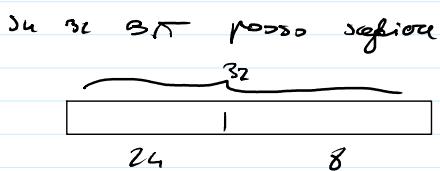
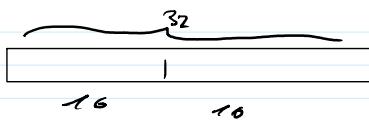


Tabelle di  $2^{24}$  (16 milioni di pagine)

La memoria da riservare non può essere statica  
dentro l'MRU



$2^{16}$

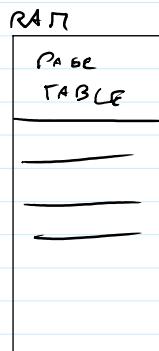
Pagine possono sfiori dentro l'MRU, ma  
ciò comporta che ogni pagina contiene  
minimamente  $2^{16}$  celle  
(se volessi aggredire memoria devo obbligatoriamente  
aggiungere tutte le celle delle pagine, è chiaro  
di tenere solo una minima parte!)  
es. un nuovo segmento mi costa 64 kB  
(cioè significa che l'occupazione di 1 celle  
mi spiega 63 kB)

riservando una porzione non esagerata  
di spazio

Di più quindi di inserire le PAGE TABLE  
nella RAM (tecnologia di memoria  
dinamica, elegante ma più lenta)

Il funzionamento può essere  
comunque migliorato tramite  
una tecnica di CASHING

➔ è diversa di una memoria fisica nella RAM

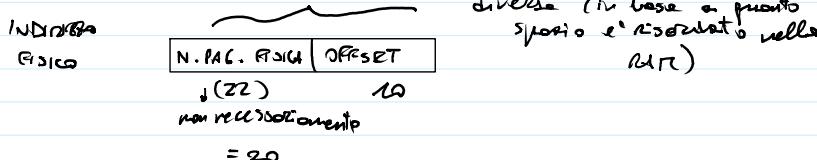
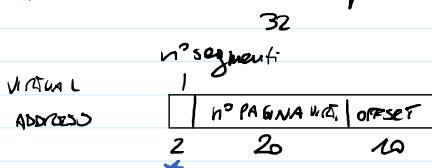


RAM  
programma diversamente

non  
programma diversamente

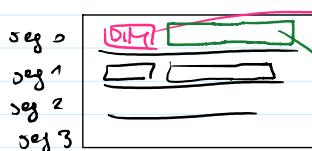
Nel caso di implicito, non avrò il n°  
di seg. e spazio ma ho comunque le seg. table  
e il n° di seg. viene comunicato dalla CPU  
alle mme da file che non sono quelli di  
indirizzamento ↗ (in base alle operazioni  
individua i segmenti)

### PAGING + SEGMENTATION (esplicito)



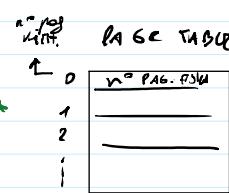
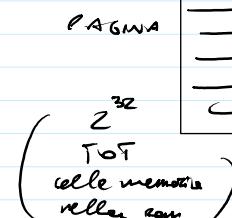
$2^{22}$  celle di  
memoria  
(pagine)

seg. table (dipende da n° seg. \*)  
(+ olt.  $2^2$  segmenti)

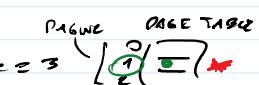


size mi identifica  
la dimensione del  
segmento  
(avendo il n° di pagina che  
costituiscono quel seg.)

indirizzo delle PAGE TABLE  
(avendo elenco delle pagine  
proprie al segmento)



o. di 16 pagine  
espresso come un  
 $2^2$  bit  
(n° pag. fisica)



La CPU tiene l'indir. virtuale al quale dovrà dare il input: 32 bit,

so riferimenti al n° seg. (es. TOT = 1 avendo seg 1)

E' indirizzabile all'ind del seg. Poi osserva la size e l'ind della sua PAGE TABLE

Si fa un confronto fra size e l'ind. della pag. virtuale

\* se size  $\geq$  n° PAG. VIRT. vi e' un errore (manca la pag.)

(ho sfornato il segmento), non esiste la pagina virtuale nel segmento (ha indici 0, 1, 2)

la pagina  
3 non  
esiste

oltremodo (es. n° pag. virt. = 1 < 3) effetta la traduzione,

accedo alla pagina di indice 1, \*

avendo trovo i 22 bit da inserire

come n° PAGINA FISICA, e spazio

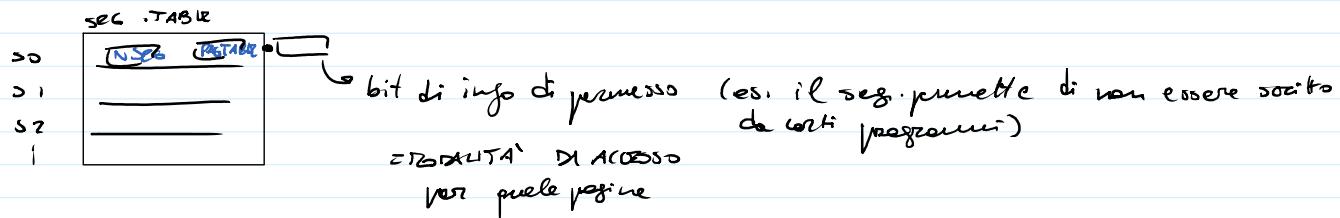
lo stesso offset in entrata.

Cioè confronto fra

n° pagine allocato nel seg.

e il n° delle pagine

dove la segmentazione può essere resa più efficiente con l'aggiunta di indirizzi assoluti:



Possiamo raggruppare i permessi in:

- READ : un solo di lettura

- WRITE : un solo di scrittura

- EXEC : permette di far il fetch e distinguere le operazioni di READ e WRITE e prendere delle decisioni.

Possiamo aggiungere altri bit per ottimizzare il meccanismo di ADDRESS GENERAZIONE

PAGE TABLE



anche la singola pagina può avere bit di ingresso in più, ad esempio per sapere come viene impiegata, ma soprattutto se è già stata visitata:

- ✓ la CPU ha generato un indirizzo relativo a quella pagina per poi scriverla o leggerla (ha letto la pagina)
- ✗ non è stato letto

e ancora

accesso solo in scrittura

✓ .. " lettura

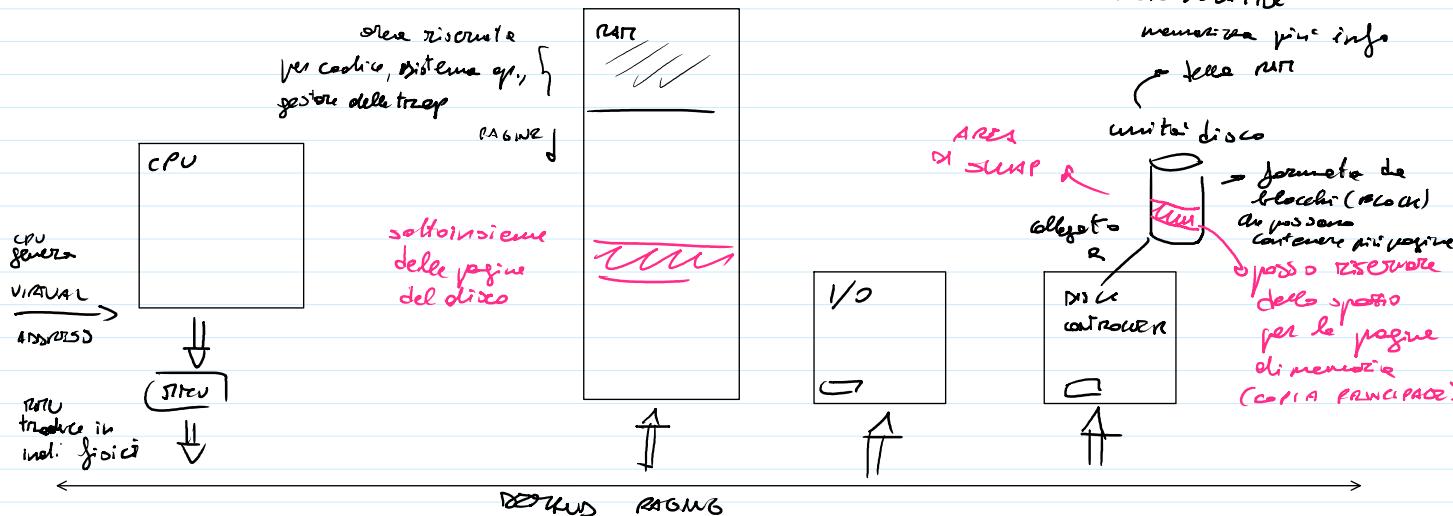
se spengo la macchina i detti non sono veri

NON VOLATILE

memoria più info  
telle PTE

unità di disco

→ formate da blocchi (blocks) che possono contenere più pagine  
oppo a riservare dello spazio per le pagine di memoria (CALLA PRINCIPALE)



- vedo così a utilizzare solo le pagine che mi interessano (utilizzate dai processi)
- se lo PTE c'è software, vedo se si puo' fare a meno, cancellandole dalla memoria RAM, caricando la pagina virtuale nell'unità disco (sostitivo, aggiorno i dati di una versione di pagina nel disco obsoleta)