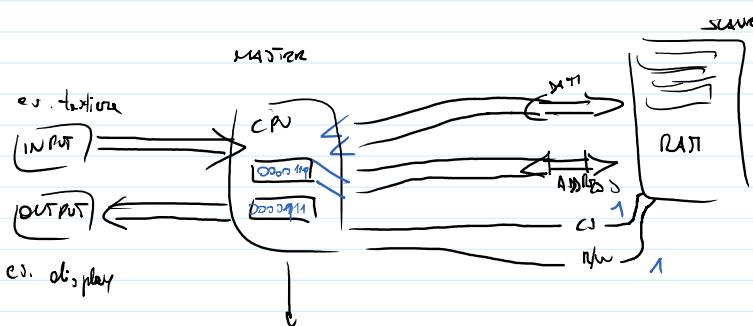


## Macchina di Von Neumann

giovedì 19 novembre 2020 16:33



I programmi (le istruzioni) sono gestiti in fasi:

- 1 - **FETCH**
- 2 - **DECODIFY**
- 3 - **EXECUTION**



Fetch: ciclo di lettura delle ram

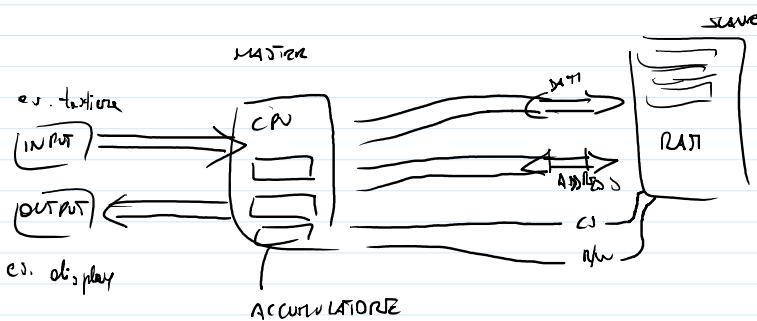
si usano anche dei registri, principalmente:

- PROGRAM COUNTER (PC)
- INSTRUCTION REGISTER (IR)

si manda come ADDRESS  
il registratore PROGRAM COUNTER

$$\begin{aligned} IR &\leftarrow \text{RAM}[PC] \\ \text{memoria} & \quad \quad \quad PC \leftarrow PC + 1 \\ \text{risultato} & \end{aligned}$$

dipende' disponibilita' le RAM con  $CS = 0$ , posso  
ritrovare l'indirizzo e posso cambiare il valore in PC



Decodifica: il valore nell'IR  
viene decodificato

Execution: In base al valore  
decodificato svolge i  
processi.

Inoltre se nello stesso indirizzo di un altro registro la somma del valore in PC venga sommata con quelle in ACC e registrata in IR.

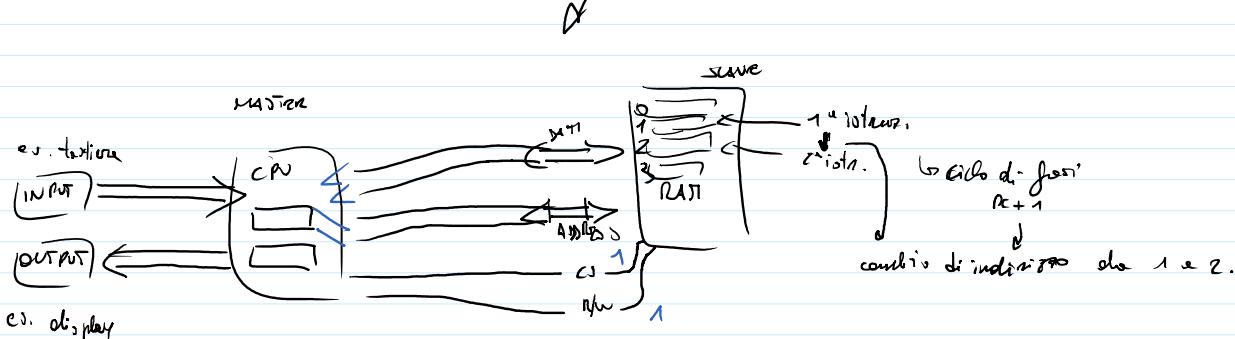
Infine si ritorna al  
FETCH con l'istruzione conservativa (indirizzo successivo) RAM non elaborante  
tornare

→ es. istruzione indirizzo 5 → ist. indirizzo 6

(e.g. se PC ha un overflow  
torna all'indirizzo 0)

Tale volta è stato fatta la macchina e' accesso.

Si dice un programma  
SECURITÀ  
(ogni istruzione segue telle fasi e  
poi si passa alle successive)



→ Ciclo di fasi  
 $n+1$   
comincia da indirizzo che 1 e 2.

Se vogliamo gestire il corso di esecuzione le diverse istruzioni e' inutile uscire più celle di memoria.

Si vorrà quindi le istruzioni di controllo di flusso (**FLOW CONTROLS**), ovvero **DEVIAZIONI**.

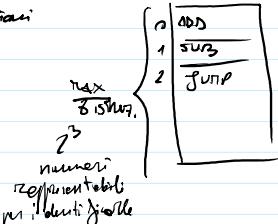
### • ISTRUZIONE DI SALTO (**JUMP**)

PC  $\leftarrow$  COSTANTE

$\rightarrow$  FETCH  $IR \leftarrow RAM[PC]; PC \leftarrow PC + 1$

$\rightarrow$  DECODE  $10101$  (16 bit)  $\rightarrow$  vede il contenuto delle istruzioni  
OVAI 3 13  
10000 codice operativo  
della istruzione da eseguire  
nella sequenza  
nell'esempio nro (2)  
 $\downarrow$  costante  
 $\downarrow$  informazioni  
 $\downarrow$  es. la costante

liste delle possibili istruzioni:



$\rightarrow$  EXEC

prendo i 13 bit meno significativi (o dx) di IR  $PC \leftarrow (IR)_{13\text{bit}}$   
e li inserisco in PC

dunque  $|PNT| \leq 2^{13}$  celle da 16 bit, ovvero  $2^{16}$  opere  
in 13 bit possono assumere valori da 0 a 1

### • ISTRUZIONE DI SALTO CONDIZIONALE (**CONDITIONAL JUMP**)

In base a una condizione - se verificata si esegue il jump, altrimenti si passa all'istruzione nelle celle successive.

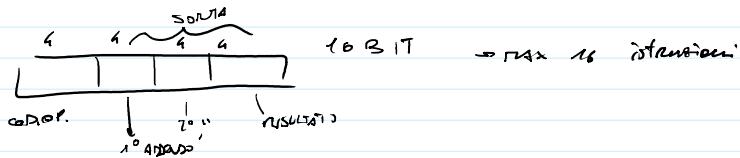
### • ISTRUZIONI DI TIPO LOGICO, ARITMETICO

(es. somma, divisione ecc..)

$\rightarrow$  norma obbliga un'espressione fra due operandi (che sono su 3)

SOMMA

$$C = A + B$$



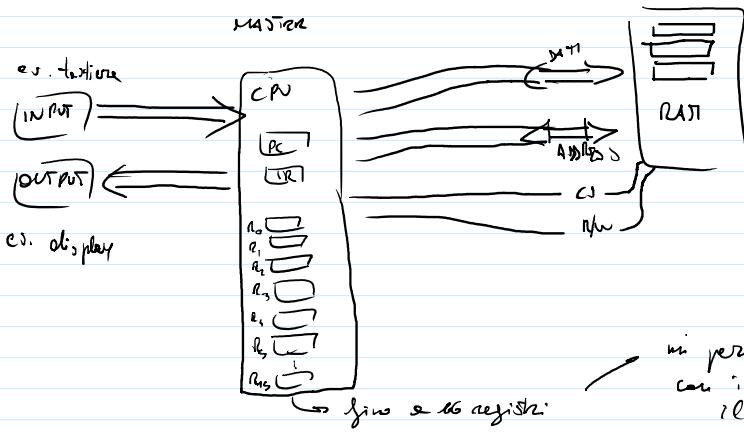
possiamo esprimere numeri solo con 4 bit, in tot.  $2^4$  numeri

Gli operandi vengono sempre implementati dai registri interni del processore

Come per A and B deve seguire dove finisce il risultato  
 $\xrightarrow{201}$

Si hanno quindi modi di indirizzamento di tipo registro, ovvero nel CPU vi sono diversi registri

identificati da un numero. Nell'esempio si può pensare a una macchina a 16 registri



mi permettono di identificare gli operandi e con i bit rimanenti sovrapposti indica dove indicatore il risultato

es.

SOMMA	R0000	R0001	R0011	R0110
	1	1		

numeri di registro

il valore in •  
dipendono  
da valori PRECEDENTEMENTE  
incorriti in R<sub>1</sub> e R<sub>2</sub> da  
altri istruzioni

IND

$$R_6 \leftarrow R_1 + R_3$$

nella fase di decode

es.

STORE

00001	00001	0010	0111

$$R_7 \leftarrow L_5 - R_2$$

#### • ISTRUZIONE DI TIRO LOAD/STORE

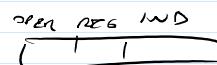
indirizzi

Load. REGISTRO C ← RAM [ IND10880 ]



d  
inserire l'indirizzo RAM  
da cui leggere i dati  
mette l'instruzione diretta

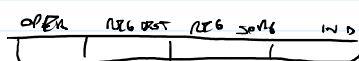
Se uso dei registri  
general, specifica  
quel è il registro



è possibile usare un meccanismo di indirizzamento indiretto → mediante registro

└ mediante celle di memoria KAR

es. mediante registro REG ← RAM [ REG<sub>10880</sub> ]



→ da specificare 2 registri

es. mediante RAM REG ← RAM [ RAM [IND] ]

d  
parzialmente  
... alla ...

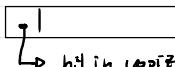
da solo l'indirizzo, sempre  
deve leggere una seconda  
cella di memoria

pondi: Ponte  
 nell'efficienza  
 (devo elaborare due  
 accessi in lettura alle RAM)

Lo devo solo l'inoltizzo, dunque  
 devo leggere una seconda  
 volta in lettura il valore che  
 esso riserva.

### • OPERAZIONE DI SCORRIMENTO (SHIFT)

es.



bit in posizione  $K$ , viene spostato nella posizione  $K+1$

Comeva si applica una moltiplicazione  $\times 2$ )

(in  $K-1$  e' divisione per 2)

} nel corso di rappresentazione  
di un numero serve  
segno

ATTENZIONE: nel caso di rappresentazione di un numero con segno, posso andare in overflow, lasciando di far camminare il segno.

(tale problema deve essere segnalato)

In questo caso posso eseguire un'istruzione di salto, non eseguendo  
di fatto l'operazione e non raggiungendo l'overflow.

Nel caso di scorrimento a destra:

n° pos.  $0\ 1\ 1\ 0\ 1$

SHIFT verso destra:



nella posizione della cifra più significativa  
riporto la stessa cifra "uscita" a destra,  
come se fosse in ciclo.

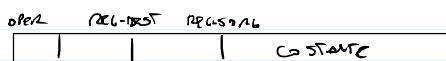
Stesso procedimento anche per i numeri negativi.

- Se vedo un scorrimento diverso posso far passare l'ultimo bit (meno significativo) alle prime posizioni, non perdendo alcun bit:



### Modi di INDIRIZZAMENTO

#### • modo indirizzamento INDICATO (INDEXED)



$$REG \leftarrow REG [REG_{SOUR} + COSTANTE]$$

Più modo preferito oggi perché  
più generale rispetto agli altri

Lo posso fare e meno dell'istruzione del modo indirizzamento indiretto  
e dell'indiretto tramite registro

Lo mi manda rispetto  
a un indice

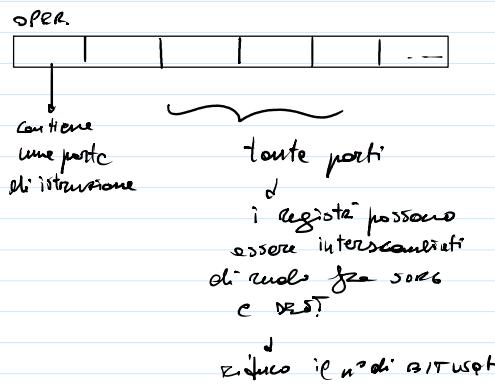
#### • modo indirizzamento DI TIPO AUTOINCREMENTO

(differisce su sistemi diversi)

- modo indirizzamento di tipo **AUTOCREMENTO** (differenza su sistemi diversi)

Si distingue sull'operazione di **REG**

**dest:**



azionata **INC o DECREMENTO**  
 $R \leftarrow R \oplus 1$  **BISOGNA SPECIFICARE**  
 IC REGISTRATO IN CUI  
 DI LAVORO

↳ il valore per incrementare la memoria RAM

→ posso usare il valore precedente o successivo rispetto all'incremento

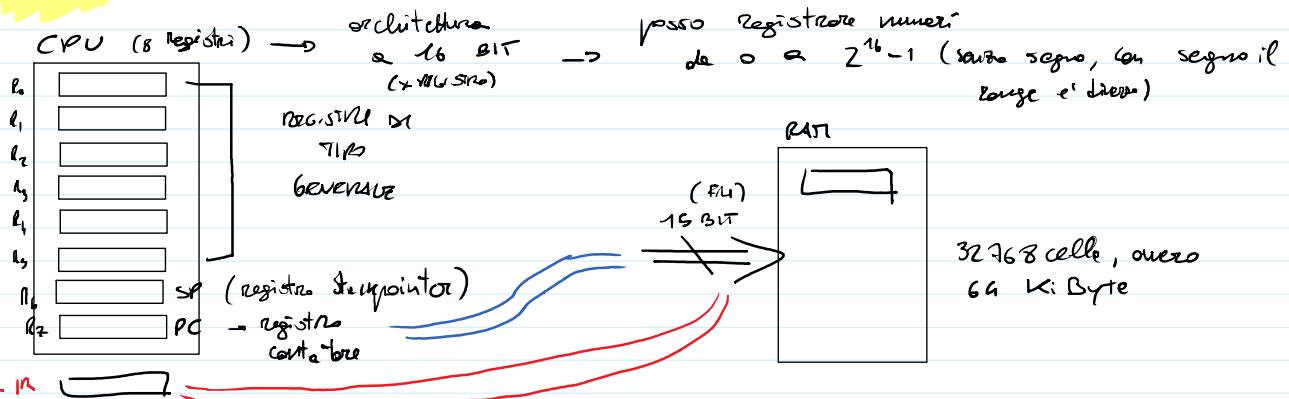
es.  $R \leftarrow R+1$ ,  $REG \leftarrow RAM[DEST]$

→ possono scambiarsi  
 (e' un altro modo di indirizzamento  
 di autocremento)

Nel caso in cui l'operazione viene effettuata solo su un registro, il codice di questo viene incluso nel codice operativo dell'istruzione, non c'è bisogno di riservare uno speciale codice di identità

Architettura processore del 1980 (in uso oggi per missioni aerospaziali)

### PDP-11



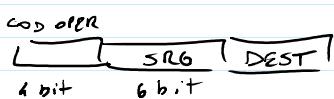
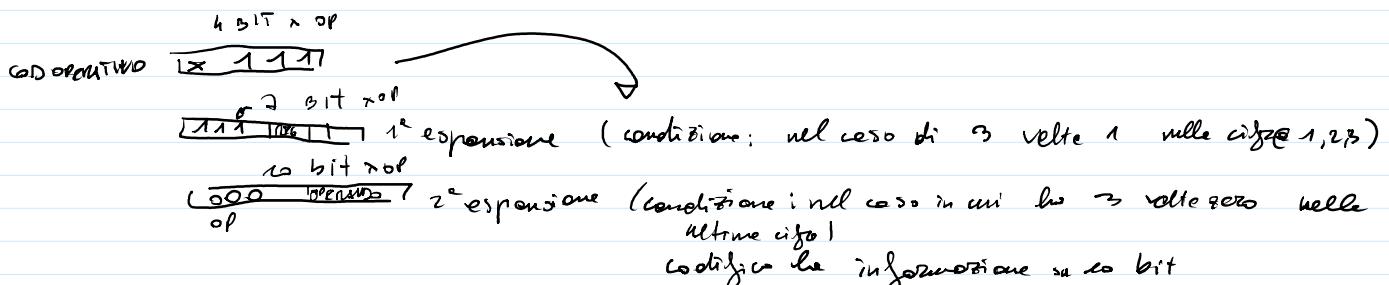
→ default si lavora su dati a 16 bit (single parole, word)

→ se anche lavorare solo con 8 bit (un byte)

Ma anche con 32 bit (double words, accappono due celle o registri consecutivi) da R<sub>7</sub> da CPU

**N.B.** Il registro delle istruzioni è presente, ma usato solo per il PC, e funziona come già visto. Non considerato come registro interno, ma non manipolabile da alcun insieme di istruzioni.

### CODIFICA DELLE ISTRUZIONI



3+3      3+3 → 3 bit mi dicono il modo di indirizzamento

→ 3 bit specificano il registro

Ese. 111 → R7 = PC

000	registro
001	indiretto tramite registro
010	autoincremento
011	autoincremento indiretto *
100	auto decremento
101	auto decremento indiretto *
110	indexed
111	indexed indiretto *

\* non visto

- dire significa quindi prendere il contenuto in R3

• 001 010 → RAR[RS]

• 010 101 → RAR[RS], RS ← RS +

- 011 101 autoincr. indiretto

RAR[RAR[RS]], RS ← RS + 1

occede due volte alla memoria,  
poi incrementa il registro  
(passo al successivo)

- 101 101 autoincr. indir.  
(è l'inverso dell'autoincr.)

RS ← RS - 1, RAR[RAR[RS]]

nel tipo indiretto si accede una seconda volta alla RAM (alla prima si ottiene un indirizzo da usare nella seconda lettura)

↳ RAR[RS] lo uso come indirizzo da LEGGERE (indirizzo la RAM alla posizione in cui mettere l'elemento in posizione RAR[RS])

il bit più significativo ( $\leq 5x$ )  
ha valore 1 → byte  
o word

[ se ha il tipo byte, lo incrementa: RS ← RS + 1  
se l'operando è di tipo parola e 16 bit

0101 RS ← RS + 2 (l'ultimo bit rappresenta dove è posizionata l'informazione, nonché il bit che determina il byte dell'informazione su cui scegliere le parole (l'ultima o successiva))

- 110 000 indexed

RAR [R0 + C05]

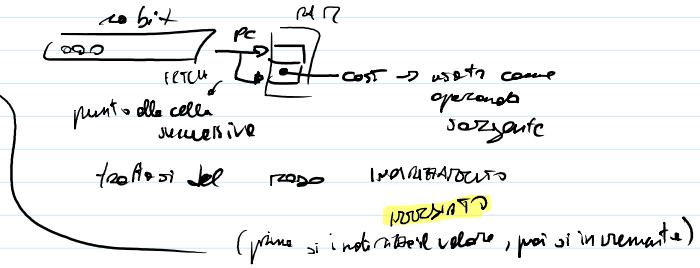
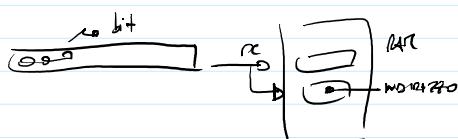
Indirizzo la RAM sulla posizione

del registro + una costante  
e 16 bit

(o dunque ha  
2 word  
worte)

se entrambi i registri sono  
in modo indexed ha  
3 words (una x registro  
più una delle costante)

- 010 111 autoincremento
- 011 111 autoincremento, indiretto



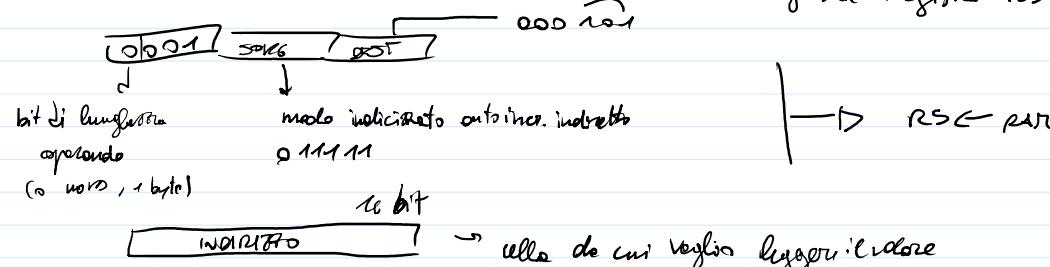
RAR[memoria]  
definito

modo indiretto

Cio' succede quando si applica  
l'autoincremento (indiretto) sul program counter

## • MOVE

equividente  
di una LOAD



$R3 \leftarrow PAR[INDIRETTO]$

alle da cui leggo il valore

→ posso muovere (copiare) il contenuto di un registro in un altro

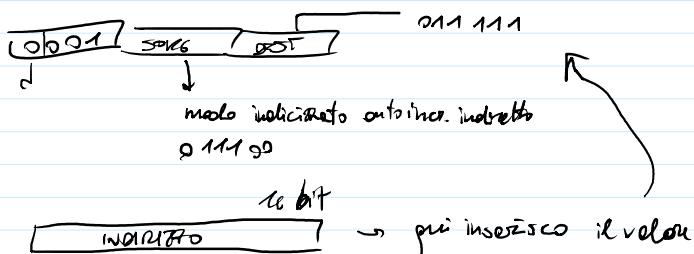


stessa cosa posso farla fra celle  
(anche fra registro e celle di memoria)

} il comportamento viene distinto dal  
bit \* più significativo

## • STORE

(memorizzo nelle  
celle indirette)



## • COMPARARE (confronto)

↳ sottraggo all'operando  $\rightarrow RC$ , l'operando TEST :  $\rightarrow RC - TEST$

ma il risultato non viene memorizzato

Nel registro di stato (state register) vengono contenuti come serie di bit che riuniscono

il valore risultato delle ALU relativamente all'operazione di confronto sopra

(primo registro se il risultato è positivo, negativo, overflow, zero)

$TEST \rightarrow RC$

$RC = TEST$

in base a 10 paia di istruzioni posso eseguire un confronto e quindi scegliere di fare un jumpt condizionato (es. se risultato el zero  $\rightarrow$  esce oppure si va avanti)

• BIT SET :  $RC \wedge TEST$  nessun risultato memorizzato

esiste anche BIT TEST  $\rightarrow OR$

viene registrato temporaneamente  
il risultato nello STATE

esiste anche BIT TEST  $\rightarrow$  OR  
SET CLEAR  
 $\dots$

vengono registrati temporaneamente  
il risultato nella STATE  
REGISTER

$\rightarrow$  Rove e' codificati con  $X \ 0\ 0\ 1$   
byte word  
" " 16 bit

$\rightarrow$  corrente  $X \ 0\ 1\ 0$   
si lavora solo su byte  
byte word  
" " 8 bit

BASE BASE NO:  
"3 or 11"  
 $\rightarrow$  BIT SET  $X \ 0\ 1\ 1$   
word byte

- ADD      OP CODE      DEST  $\leftarrow$  DEST + SRC  
0110      summa
- SUB      1111      DEST  $\leftarrow$  DEST - SRC  
0      sottrazione

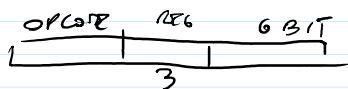
} entrambe effettuate  
su 16 bit

il bit più significativo  
serve solo per  
identificare ADD e SUB  
(non per distinguere  
fra word e byte)

Si riserva da in ADD e DEST, DEST viene sorcebitto (in questo viene usato come sorgente  
del risultato somma e SRC)

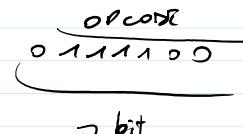
N.B. IRLC vengono usati con segno invertito (altrimenti otterrei un - - - +, somma)  
e DEST con segno positivo

[Espansione a 7 bit]



i 3 bit rimanenti vengono usati per individuare il Registro  
e gli altri 6 bit prima riduce l'operando

es. XOR ; DEST  $\leftarrow$  DEST XOR REGISTRO  
registro a celle  
dest viene sorcebitto

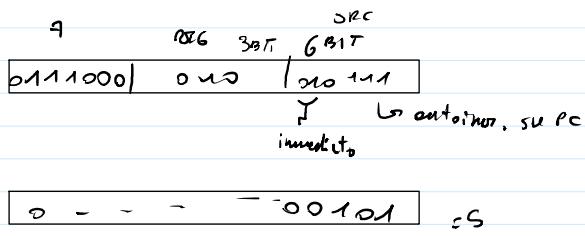


MUL

o la PDP-11 ha anche l'intenzione di moltiplicazione: prende i 1° bit del 1° oper, 16 bit del 2°,  
e il risultato viene rappresentato su 32 bit

9      MUL      ...      DRC

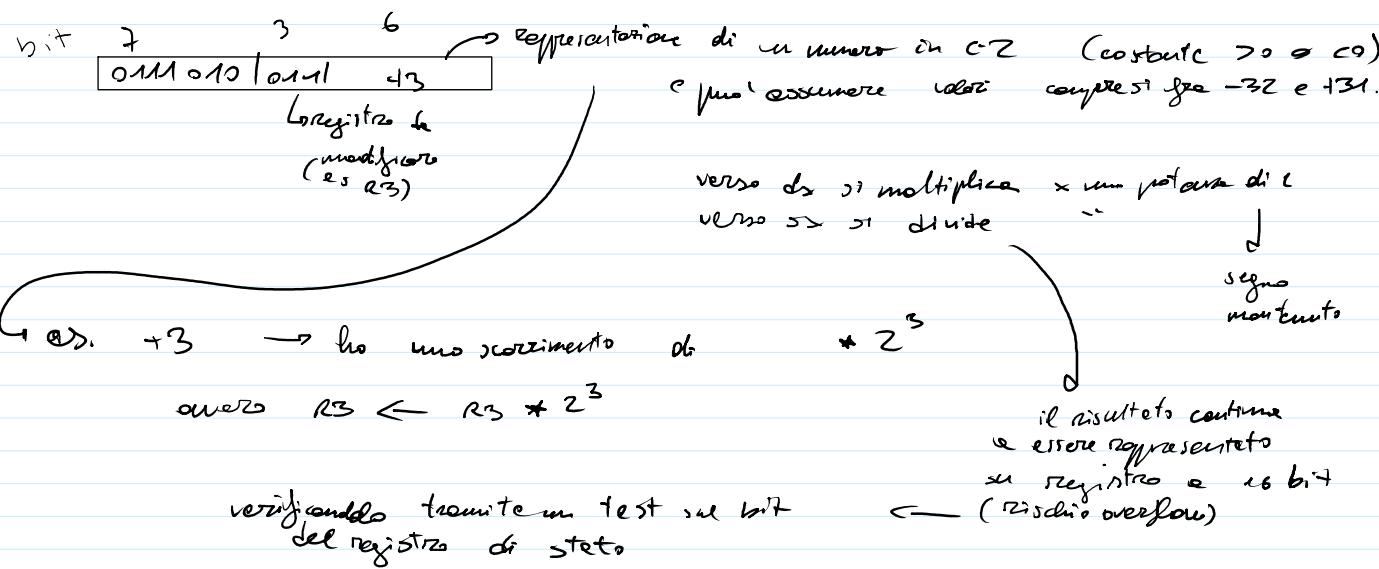
- Per PDP-11 ha anche l'istruzione di moltiplicazione: prende 16 bit del 1° oper, 16 bit del 2°, e il risultato viene rappresentato su 32 bit



$$\rightarrow R_2, R_3 \leftarrow R_2 * S$$

i primi 16 bit li metto in  $R_2$   
e gli altri 16 più significativi  
li inserisco in  $R_3$  (registro successivo)

- **ARITHMETIC SHIFT** (scorrimento aritmetico) (eseguito  $\pm 2^n$  base allo shift a dx o sra)



Esiste anche la versione estesa su 2 registri (motivo del range di scorrimento)  
 (SWFT su una coppia)

$$6 \text{ tot.} > 16 \text{ bit}$$

$-32, +31$

di infatti  
ne altre i 16 bit