

# **CORSO INTRODUTTIVO DI RETRO PROGRAMMAZIONE IN BASIC**

## **Parte II**

Pierpaolo Basile, [pierpaolo.basile@gmail.com](mailto:pierpaolo.basile@gmail.com)

Apulia Retrocomputing

Università degli Studi di Bari Aldo Moro

## Parte II

- grafica bitmap
- sprite
- scrolling
- audio con il SID

# Modalità bitmap

- Nella modalità `bitmap` l'intero schermo `320x200` è indirizzabile pixel per pixel
- Ogni pixel può assumere solo 2 valori (0 o 1)
- Per ogni cella 8x8 è possibile definire due colori nell'area di memoria video (indirizzo 1024)
  - quest'area non contiene più i codici carattere, ma le informazioni colore della relativa cella 8x8. I 4 bit più significativi indicano il colore quando il pixel è 1, i 4 bit meno significativi al colore quando il pixel è 0
- Per indirizzare tutti i pixel sono necessari  $40 \times 25 \times 8 = 8000$  byte quest'area di memoria deve essere allocata nella RAM indirizzabile dal VIC-II

# Modalità bitmap

- Il modo più **efficace** per gestire la modalità bitmap è il **linguaggio macchina**, poiché richiede molti accessi alla memoria (per adesso utilizzeremo il BASIC)
  - la modalità bitmap richiede più RAM sottratta al BASIC
  - in linguaggio macchina il BASIC può non essere utilizzato e si libera maggiore memoria
- Per accedere alla modalità bitmap bisogna settare ad 1 il bit 5 di un registro VIC-II all'indirizzo 53265
  - attivare bitmap: `POKE 53265, PEEK(53265) OR 32`
  - disabilitare bitmap: `POKE 53265, PEEK(53265) AND 223`

# Modalità bitmap

- Se attiviamo la modalità bitmap senza **pulire la memoria video** visualizzeremo dei pixel casuali che dipendono dal contenuto della RAM e dalla memoria video dei caratteri
- **Come si può accedere al singolo pixel e impostare il colore della relativa area 8x8?**
- Per indicare che la memoria bitmap deve iniziare all'indirizzo 8192, dobbiamo settare ad 1 il bit 3 del registro 53272, le prime istruzioni per abilitare il bitmap sono:

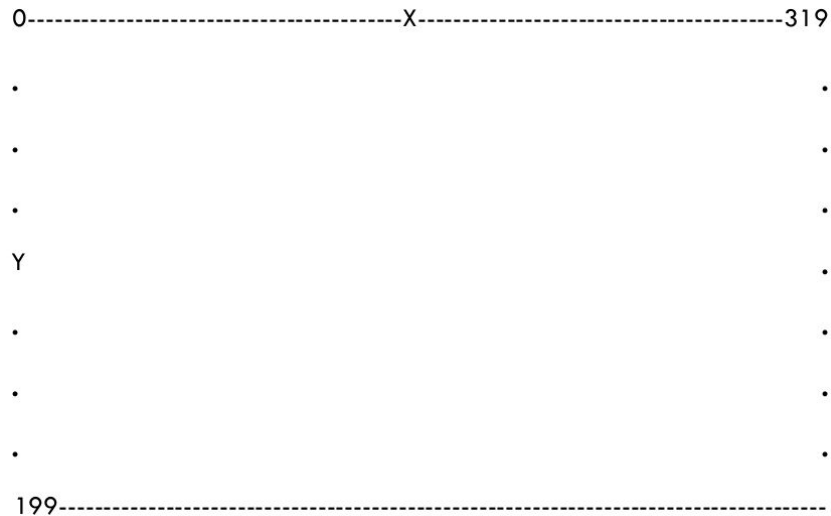
```
5 BASE=8192 : POKE 53272, PEEK(53272) OR 8  
10 POKE 53265, PEEK(53265) OR 32
```

# Modalità bitmap

```
5 BASE=8192 : POKE 53272, PEEK(53272) OR 8
10 POKE 53265, PEEK(53265) OR 32
20 FOR I = BASE TO BASE + 7999: POKE I, 0: NEXT
30 FOR I = 1024 TO 2023: POKE I, 16: NEXT
```

- la linea 20 pone a 0 tutti i pixel della bitmap
- la linea 30 imposta il colore di tutte le celle 8x8 a 16, quindi 0001 0000. Pixel acceso bianco (0001), pixel spento nero (0000). Nella memoria colore per ogni cella 8x8 sono contenute le info dei due colori (acceso/spento)

# Modalità bitmap



- Dobbiamo pensare allo schermo come ad una griglia di pixel 320x200, l'angolo in alto a sx ha coordinate (0,0)
- Però i pixel sono memorizzati in celle 8x8 che ricordano la composizione dei caratteri

## Modalità bitmap

- Prendiamo come riferimento il primo blocco 8x8 in alto a sinistra questo conterrà i primi 8 byte che descrivono i primi 8 pixel delle righe da 0 a 7

	0	1	2	3	4	5	6	7	← column
row	0	byte 0							
	1	byte 1							
	2	byte 2							
	3	byte 3							
	4	byte 4							
	5	byte 5							
	6	byte 6							
	7	byte 7							



# Modalità bitmap

TOP LINE ROW 0	-----	BYTE 0	BYTE 8	BYTE 16	BYTE 24	.....	BYTE 312
		BYTE 1	BYTE 9	.	.		BYTE 313
		BYTE 2	BYTE 10	.	.		BYTE 314
		BYTE 3	BYTE 11	.	.		BYTE 315
		BYTE 4	BYTE 12	.	.		BYTE 316
		BYTE 5	BYTE 13	.	.		BYTE 317
		BYTE 6	BYTE 14	.	.		BYTE 318
	-----	BYTE 7	BYTE 15	.	.		BYTE 319
SECOND LINE ROW 1	-----	BYTE 320	BYTE 328	BYTE 336	BYTE 344	.....	BYTE 632
		BYTE 321	BYTE 329	.	.		BYTE 633
		BYTE 322	BYTE 330	.	.		BYTE 634
		BYTE 323	BYTE 331	.	.		BYTE 635
		BYTE 324	BYTE 332	.	.		BYTE 636
		BYTE 325	BYTE 333	.	.		BYTE 637
		BYTE 326	BYTE 334	.	.		BYTE 638
	-----	BYTE 327	BYTE 335	.	.		BYTE 639

# Modalità bitmap

Per calcolare il byte della cella di memoria date le coordinate X,Y del pixel da modificare dobbiamo procedere in questo modo:

$ROW = INT(Y/8) \rightarrow BYTE = ROW * 320$  (ogni riga ha 320 byte)

Per sapere la colonna (0-39):

$COL = INT(X/8) \rightarrow BYTE = ROW * 320 + COL * 8$  (ogni colonna ha 8 byte)

Dentro la cella dobbiamo capire quale degli 8 byte scegliere

$LINE = Y \text{ AND } 7$  (gli ultimi 3 bit del valore di Y)

Infine abbiamo:

$BYTE = BASE + ROW * 320 + COL * 8 + LINE$

# Modalità bitmap

Ora sappiamo quale byte (gruppo di 8 bit modificare), ma dobbiamo capire quale è il pixel (BIT) di questi 8 da modificare:

$BIT = 7 - (X \text{ AND } 7)$  (a 7 dobbiamo sottrarre gli ultimi 3 bit di X)

Infine, per accendere un pixel:

$ROW = INT(Y/8) \rightarrow BYTE = ROW * 320$

$COL = INT(X/8) \rightarrow BYTE = ROW * 320 + COL * 8$

$LINE = Y \text{ AND } 7$

$BYTE = BASE + ROW * 320 + COL * 8 + LINE$

$BIT = 7 - (X \text{ AND } 7)$

$POKE \text{ BYTE}, PEEK(BYTE) \text{ OR } 2^{\uparrow}BIT$

# Bitmap: disegno funzione

```
5 BASE=8192 : POKE 53272, PEEK(53272) OR 8
10 POKE 53265, PEEK(53265) OR 32
20 FOR I = BASE TO BASE + 7999: POKE I, 0: NEXT
30 FOR I = 1024 TO 2023: POKE I, 16: NEXT
50 FOR X = 0 TO 319 STEP .5
60 Y = INT(90+80*SIN(X/10))
70 CO = INT(X/8)
80 RO = INT(Y/8)
85 LN = Y AND 7
90 BY = BASE + RO*320 + 8*CH + LN
100 BI = 7-(X AND 7)
110 POKE BY, PEEK(BY) OR (2↑BI)
120 NEXT X
125 POKE 1024,5
130 GET K$: IF K$="" THEN GOTO 130
```

# Bitmap Multicolor

- Questa modalità è identica alla bitmap, ma la risoluzione orizzontale si dimezza ed ogni pixel richiede 2 bit nella memoria video. Questi 2 bit identificano i 4 possibili colori

00	background color register (53281)
01	upper 4 bits of screen memory
10	lower 4 bits of screen memory
11	color memory

# Bitmap Multicolor

- Per attivare, settare ad 1 il bit 5 di 53265 e il bit 4 di 53270

```
POKE 53265,PEEK(53265) OR 32 : POKE 53270,PEEK(53270) OR 16
```

- Per disattivare

```
POKE 53265,PEEK(53265) AND 223 : POKE 53270,PEEK(53270)  
AND 239
```

# Bitmap Multicolor

```
5 BASE=8192 : POKE 53272, PEEK(53272) OR 8
10 POKE 53265,PEEK(53265) OR 32:POKE 53270,PEEK(53270) OR 16
20 L = 0 : BC = 55296 : BS = 1024
30 FOR C = 0 TO 999
40 POKE BC+C, L : POKE BS+C, L*16+15-L
50 L = L + 1
60 IF L>15 THEN L = 0
70 NEXT C
80 FOR I = BASE TO BASE + 7999: POKE I, 27: NEXT
90 GET K$ : IF K$="" THEN GOTO 90
```

**SPRITE**



# Cos'è uno SPRITE?

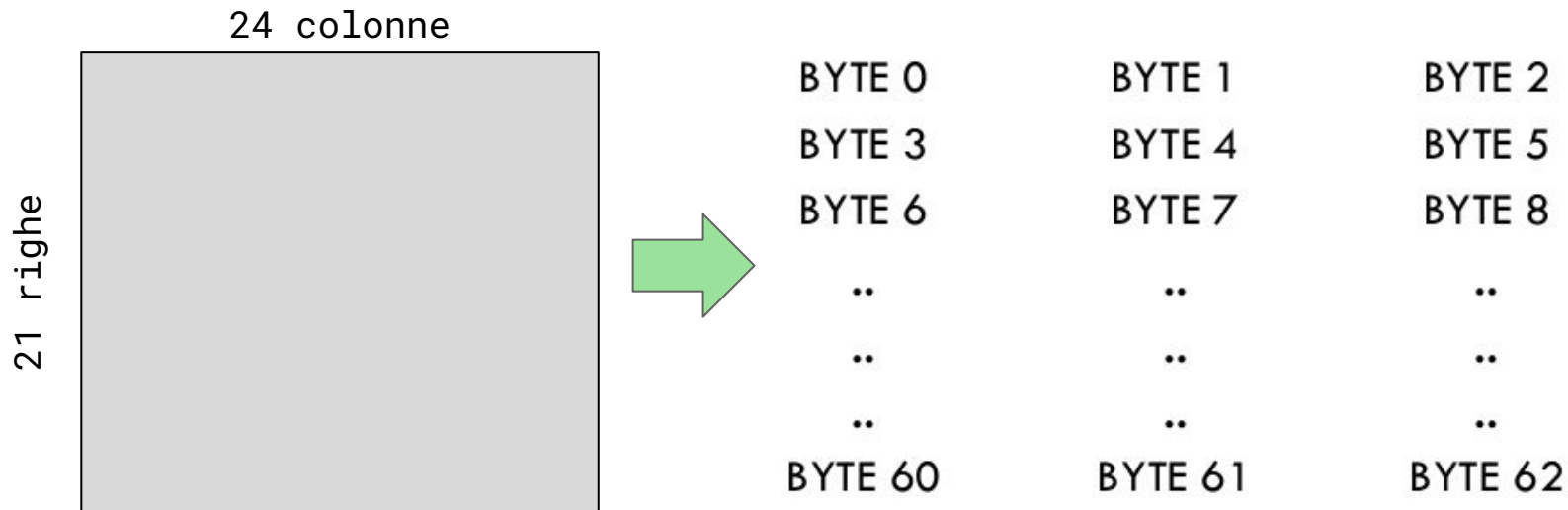
- Uno **sprite** è un **oggetto grafico** che può essere **visualizzato** e **spostato** sullo schermo in qualsiasi modalità grafica del VIC-II
- Ha dimensioni **24x21** pixel
- Ogni sprite può assumere uno dei 16 colori
  - esiste anche la possibilità di sprite multicolor
- Possono essere ingranditi (2X) sia orizzontalmente sia verticalmente
- Gestione delle collisioni tra sprite e sprite/background

# SPRITE

- Gli sprite sono gestiti in modo hardware dal VIC-II
- Il VIC-II può gestire 8 sprite numerati da 0 a 7
  - il numero stabilisce anche la priorità
  - con tecniche più sofisticate è possibile gestire più di 8 sprite, ma non è possibile avere più di 8 sprite sulla stessa riga
- Ogni sprite ha le sue coordinate (X,Y), il colore e la capacità di abilitare/disabilitare le collisioni

# Definire uno SPRITE

- La dimensione di uno sprite è 24x21 quindi sono necessari 3 byte per riga ( $3 \times 21 = 63 \text{ byte} = 504 \text{ bit}$ )
- I byte sono disposti in questo modo

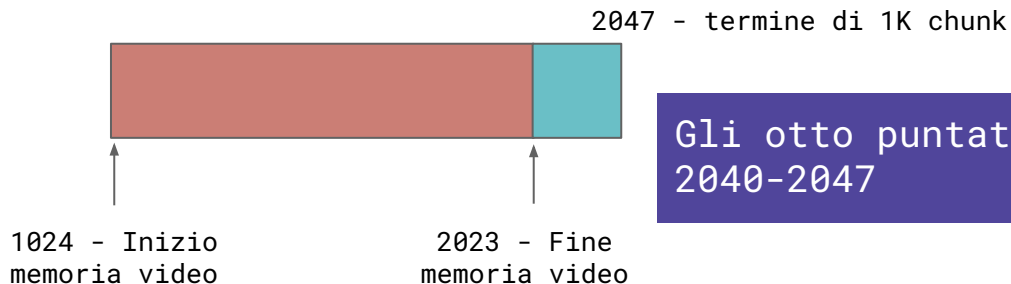


# SPRITE multicolor

- Come per i caratteri anche gli sprite hanno una modalità multicolor
- Nella modalità multicolor la risoluzione orizzontale si dimezza (12x21) ogni pixel è largo due ed è codificato da 2 bit (4 colori possibili)
  - 3 colori sono condivisi tra tutti gli sprite

# Puntatori agli sprite

- Ogni sprite occupa 63 byte più 1 byte che è utilizzato come segnaposto per indicare la fine dello sprite
  - in totale servono 64 byte per ogni sprite
- Ad ogni sprite è associato un **puntatore** che occupa 1 byte
  - servono 8 byte per puntare tutti gli sprite
  - questi 8 byte si trovano alla fine di un blocco da 1K della memoria video
  - all'avvio la memoria video inizia alla locazione 1024



Gli otto puntatori sono allocati nell'area 2040-2047

# Puntatori agli sprite

- La locazione dei puntatori varia se viene cambiata la posizione della memoria video
- Ogni puntatore occupa 1 byte quindi può assumere valori da 0 a 255
  - poiché ogni sprite occupa 64 byte è possibile posizionare uno sprite in qualsiasi area di memoria che rientra nei 16K indirizzati dal VIC
  - $(256 \times 64 = 16K)$  quindi il valore del puntatore va moltiplicato per 64 e sommato all'indirizzo di inizio dell'area indirizzata dal VIC per ottenere l'indirizzo assoluto di memoria dove si trova lo sprite

# Dove collocare gli sprite?

- Come abbiamo detto gli sprite possono essere collocati in tutta l'area di memoria indirizzabile dal VIC
  - dobbiamo considerare che il VIC indirizza la memoria video e i caratteri
  - se abbiamo il VIC nei banchi 1 e 3 questo non avrà accesso ai caratteri e avremo più spazio per gli sprite
  - se abbiamo il VIC nei banchi 0 e 2 il tutto deve convivere con la memoria caratteri e la memoria video
- Se stiamo utilizzando anche il BASIC possiamo allocare pochi sprite nel buffer cassetta (828-1019) oppure nella memoria riservata al BASIC come abbiamo fatto per la ridefinizione dei caratteri

# Come attivare gli sprite

- Il byte all'indirizzo 53269 indica per ogni bit quale sprite deve essere attivo
- Se volessimo attivare lo sprite 1  
`POKE 53269,PEEK(53269) OR 2`
- Per i colori vanno utilizzati i seguenti registri

ADDRESS		DESCRIPTION
53287	(\$D027)	SPRITE 0 COLOR REGISTER
53288	(\$D028)	SPRITE 1 COLOR REGISTER
53289	(\$D029)	SPRITE 2 COLOR REGISTER
53290	(\$D02A)	SPRITE 3 COLOR REGISTER
53291	(\$D02B)	SPRITE 4 COLOR REGISTER
53292	(\$D02C)	SPRITE 5 COLOR REGISTER
53293	(\$D02D)	SPRITE 6 COLOR REGISTER
53294	(\$D02E)	SPRITE 7 COLOR REGISTER



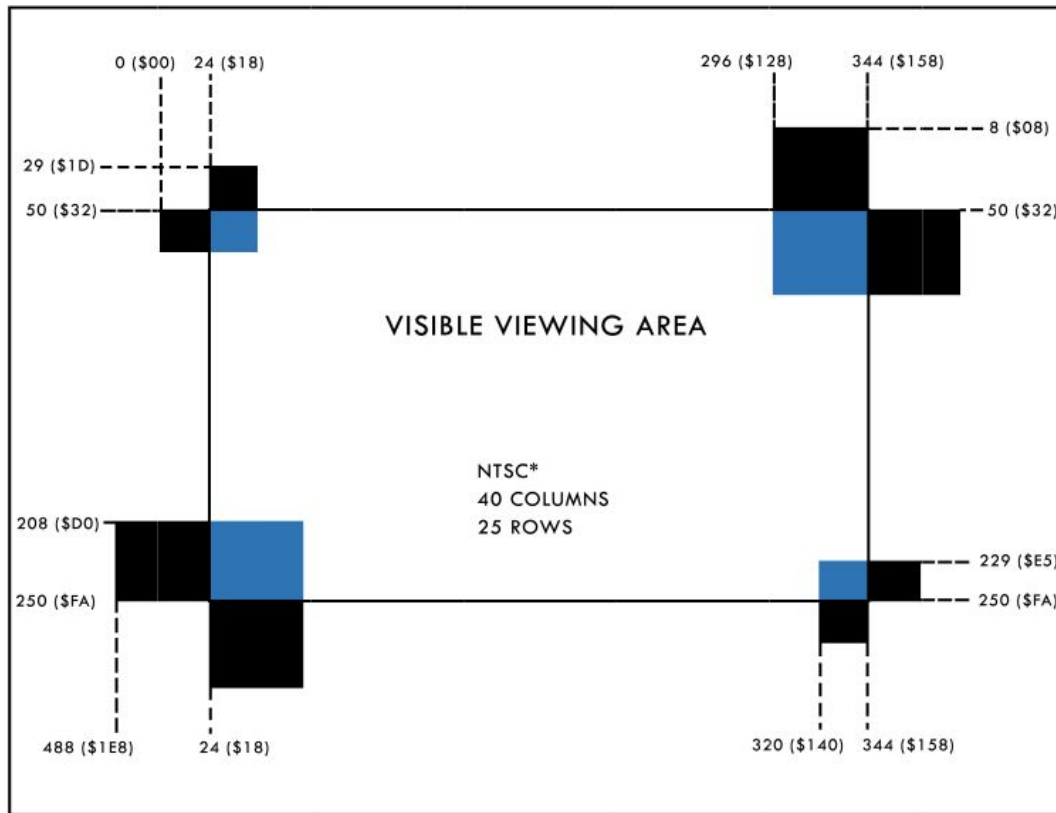
# Posizione degli sprite

- Ogni sprite ha due registri per il posizionamento orizzontale (X) e verticale (Y)
- La risoluzione verticale dello schermo è 200 pixel quindi un solo byte può essere utilizzato per lo spostamento Y
- La risoluzione orizzontale è 320 pixel quindi non può essere contenuta in un unico byte
  - esiste un unico byte per codificare il bit più significativo della posizione X, mettere ad 1 il bit corrispondente allo sprite con coordinata  $X > 255$
- La posizione (X,Y) di uno sprite coincide con l'angolo in alto a sinistra dello sprite

# Posizione degli sprite

LOCATION		DESCRIPTION
DECIMAL	HEX	
53248	(\$D000)	SPRITE 0 X POSITION REGISTER
53249	(\$D001)	SPRITE 0 Y POSITION REGISTER
53250	(\$D002)	SPRITE 1 X POSITION REGISTER
53251	(\$D003)	SPRITE 1 Y POSITION REGISTER
53252	(\$D004)	SPRITE 2 X POSITION REGISTER
53253	(\$D005)	SPRITE 2 Y POSITION REGISTER
53254	(\$D006)	SPRITE 3 X POSITION REGISTER
53255	(\$D007)	SPRITE 3 Y POSITION REGISTER
53256	(\$D008)	SPRITE 4 X POSITION REGISTER
53257	(\$D009)	SPRITE 4 Y POSITION REGISTER
53258	(\$D00A)	SPRITE 5 X POSITION REGISTER
53259	(\$D00B)	SPRITE 5 Y POSITION REGISTER
53260	(\$D00C)	SPRITE 6 X POSITION REGISTER
53261	(\$D00D)	SPRITE 6 Y POSITION REGISTER
53262	(\$D00E)	SPRITE 7 X POSITION REGISTER
53263	(\$D00F)	SPRITE 7 Y POSITION REGISTER
53264	(\$D010)	SPRITE X MSB REGISTER

# Area visibile degli sprite



Maggiori dettagli nella Programmer's Reference Guide

# Esempio sprite



0,0,0

0,0,0

0,0,0

0,0,0

0,0,0

00000111,11100000,00000000 -> 7, 224, 0

00001111,11110000,00000000 -> 15, 240, 0

00001111,11000000,00000000 -> 15, 192, 0

00011111,11110000,00000000 -> 31, 240, 0

00011111,11111000,00000000 -> 31, 248, 0

00011111,11110000,00000000 -> 31, 240, 0

00000111,11100000,00000000 -> 7, 224, 0

00001111,11000000,00000000 -> 15, 192, 0

00011111,11110000,00000000 -> 31, 240, 0

00111111,11111000,00000000 -> 63, 248, 0

00111101,01111000,00000000 -> 61, 120, 0

00111111,11111000,00000000 -> 63, 248, 0

00111111,11111000,00000000 -> 63, 248, 0

00001110,11100000,00000000 -> 14, 224, 0

00011100,01110000,00000000 -> 28, 112, 0

00111100,01111000,00000000 -> 60, 120, 0

**V=53248**

[illegible]

# Esempio sprite

```
10 V=53248 : REM VIC ADR
20 PRINT CHR$(147)
30 POKE 2040,14
40 FOR I=0 TO 62 : READ A : POKE 896+I,A : NEXT
50 POKE 53280,0 : POKE 53281,0 : POKE V+39,1
60 POKE V+21,1
70 POKE V+1,100 : POKE V+16,0
80 FOR X=24 TO 255 : POKE V,X : NEXT
85 GET K$:IF K$="" THEN GOTO 85
90 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
100 DATA 7,224,0,15,240,0,15,192,0
110 DATA 31,240,0,31,248,0,31,240,0
120 DATA 7,224,0,15,192,0,31,240,0
130 DATA 63,248,0,61,120,0,63,248,0
140 DATA 63,248,0,14,224,0,28,112,0,60,120,0
```

# Sprite multicolor

- La risoluzione orizzontale è dimezzata ed ogni pixel è codificato da 2 bit che indicano il colore

00	(trasparente) colore di sfondo del video (53281)
01	sprite multicolor register #0 (53285)
10	sprite color register (dipende dallo sprite)
11	sprite multicolor register #1 (53286)

- Per abilitare uno sprite multicolor bisogna porre ad 1 il corrispondente bit nel registro 53276

`POKE 53276,PEEK(53276) OR 2` (per lo sprite 1)

# Espansione degli sprite

- Gli sprite possono essere espansi sia orizzontalmente che verticalmente
- Durante l'espansione non cambia la risoluzione dello sprite, ma semplicemente ogni pixel equivale a due pixel
- Per l'espansione orizzontale si utilizzano i bit nel registro 53277

`POKE 53277,PEEK(53277) OR 2` (espande sprite 1 orizz.)

- Per l'espansione verticale il bit nel registro 53271

`POKE 53271,PEEK(53271) OR 2` (espande sprite 1 vert.)



# Priorità degli sprite

- Il numero dello sprite indica la sua priorità, lo sprite 0 è quello con maggiore priorità
  - gli sprite con maggiore priorità sono visualizzati davanti a quelli con più bassa priorità
  - i pixel non accessi di uno sprite funzionano come una "finestra", ovvero permettono di visualizzare eventuali pixel di sprite con più bassa priorità che sono sovrapposti
- Può essere settata anche una priorità tra sprite e background agendo sui bit corrispondenti ad ogni sprite del registro 53275
  - 0: sprite ha priorità maggiore del background
  - 1: background ha priorità maggiore rispetto allo sprite

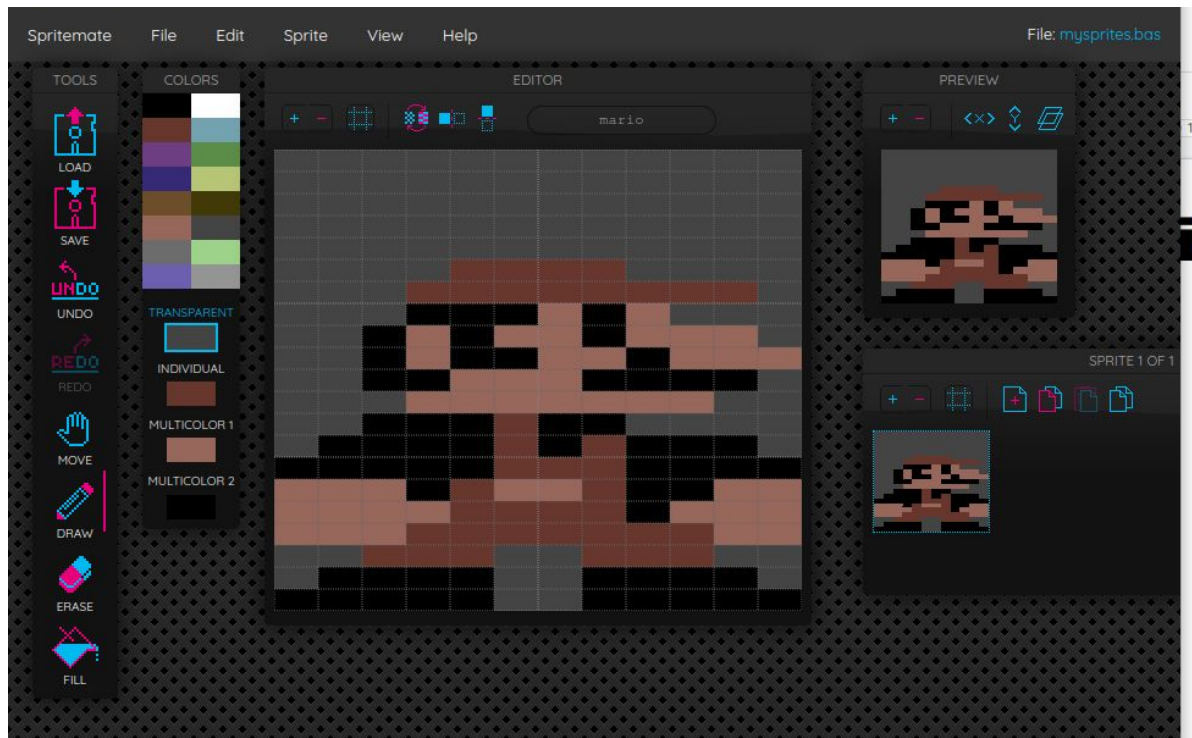
# Collisioni sprite-sprite

- Il VIC-II può identificare le collisioni tra sprite in modo automatico quando qualsiasi parte accesa dello sprite entra in contatto con una parte accesa di un altro sprite
- In questo caso il VIC-II imposta a 1 i bit corrispondenti agli sprite coinvolti nella collisione nel registro 53278
- Il valore in questo registro rimane fino a quando non viene letto da una PEEK
- Le collisioni vengono identificate anche quando gli sprite sono fuori dall'area visibile

# Collisioni sprite-background

- Il VIC-II può identificare le collisioni tra sprite e background
- In questo caso il VIC-II imposta a 1 i bit corrispondenti agli sprite coinvolti nella collisione nel registro 53279
- Il valore in questo registro rimane fino a quando non viene letto da una PEEK
- Nel caso di sprite multicolor, il colore 01 è considerato come trasparente e quindi non influisce con le collisioni con il background

# Esempio sprite multicolor



<https://www.spritemate.com/>

# Esempio sprite multicolor

```
10 V=53248 : REM VIC ADR
20 PRINT CHR$(147)
30 POKE 2040,14
40 FOR I=0 TO 62 : READ A : POKE 896+I,A : NEXT
50 POKE 53280,14 : POKE 53281,14
55 POKE V+39,2 : POKE 53285,10 : POKE 53286,0
60 POKE V+21,1 : POKE 53276,1
70 POKE V+1,100 : POKE V+16,0
80 FOR X=24 TO 255 : POKE V,X : NEXT
85 GET K$:IF K$="" THEN GOTO 85
100 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
110 DATA 170,0,2,170,168,3,247,64,13,215,84,13,245,213,15,87
120 DATA 252,1,85,80,15,239,0,63,238,252,255,234,255,87,150,245
130 DATA 85,170,213,86,170,165,10,130,160,63,195,252,255,195,255,130
```

# Ancora sulla grafica...

- Il bit 4 del registro 53265 attiva (1) o disattiva (0) lo schermo

```
POKE 53265, PEEK(53265) AND 239 : REM DISATTIVA SCHERMO
```

```
POKE 53265, PEEK(53265) OR 16 : REM ATTIVA SCHERMO
```

- con lo schermo disattivato la CPU è leggermente più veloce
- Il VIC-II permette la gestione del raster
  - è possibile leggere la posizione del raster: parte meno significativa in 53266 e parte più significativa bit 7 di 53265
  - scrivendo in questi registri è possibile abilitare un interrupt che si verifica quando il raster raggiunge la posizione indicata

# Registro di stato interrupt

- Alla locazione 53273 si trova il registro di stato degli interrupt dove il corrispettivo bit viene settato ad 1 quando l'interrupt si verifica
  - dopo aver gestito l'interrupt bisogna settare a 0 il corrispettivo bit
- Alla locazione 53274 si trova il registro per abilitare gli interrupt

LATCH	BIT #	DESCRIPTION
IRST	0	Set when the current raster count = stored raster count
IMDC	1	Set by SRITE-DATA collision (1 <sup>st</sup> one only, until reset)
IMMC	2	Set by SPRITE-SPRITE collision (1 <sup>st</sup> one only, until reset)
ILP	3	Set by negative transition of light pen (1 per frame)
IRQ	7	Set by latch set and enabled

# VIC-II e interrupt

- L'indirizzo della routine di gestione dell'interrupt va impostato nelle locazioni 788-789
  - va comunque eseguita la routine standard del KERNEL
- La gestione del raster interrupt permette di poter eseguire diversi trick sul VIC-II
  - usare diverse modalità grafiche switchando quando il raster raggiunge una determinata linea
  - visualizzare più di 8 sprite switchando i puntatori degli sprite su determinate righe (o cambiare i colori degli sprite)
- Il BASIC è troppo lento per la gestione degli interrupt e quindi va fatta in linguaggio macchina



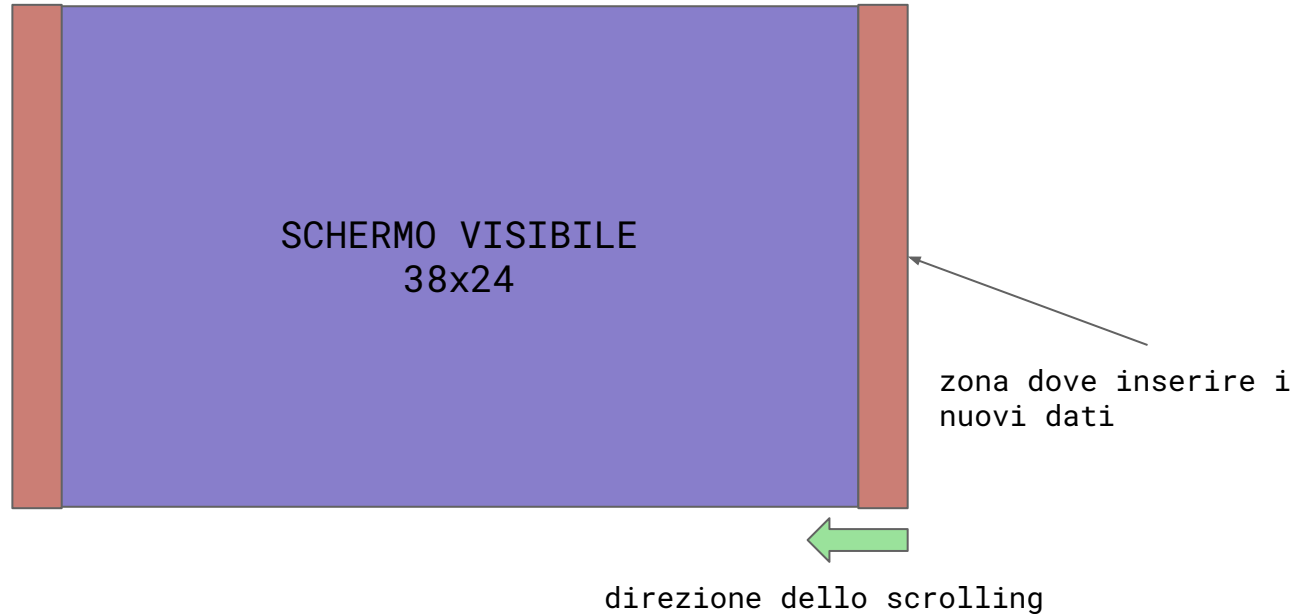
SCROLLING

# Scrolling

- Il VIC-II supporta lo `scrolling pixel per pixel` sia orizzontale che verticale
- Lo scrolling può avvenire verso sinistra-destra, verso su-giù, lo scrolling verticale e orizzontale possono essere combinati
- Molta parte dello scrolling è gestita in modo hardware dal VIC-II, ma è lasciato al programmatore il compito di modificare il contenuto dello schermo
  - `questo può essere fatto solo in linguaggio macchina`, poichè la copia di porzioni dello schermo deve essere veloce per evitare sfarfallii

# Scrolling

- Durante lo scrolling il numero di colonne e righe viene ridotto per riservare lo spazio ai nuovi dati che diventeranno visibili durante lo scrolling



# Procedura per lo scrolling

1. ridurre lo schermo (espandere i bordi)
2. impostare il registro di scrolling al massimo (o minimo in base alla direzione)
3. inserire i nuovi dati nella porzione nascosta
4. decrementare (o incrementare) il registro di scrolling fino a raggiungere il minimo (o massimo)
5. eseguire una routine che copia l'intero schermo spostandolo di un carattere rispetto alla direzione dello scrolling
6. ripetere dal punto 2

# Procedura per lo scrolling

- Per ridurre lo schermo orizzontalmente bisogna impostare a 0 il bit 3 del registro 53270  
`POKE 53270, PEEK(53270) AND 247`
- Per ridurre lo schermo verticalmente bisogna impostare a 0 il bit 3 del registro 53265  
`POKE 53265, PEEK(53265) AND 247`
- Durante lo scrolling verso sinistra dobbiamo inserire i nuovi dati nella colonna 40 (destra, colonna 0)
- Durante lo scrolling verso l'alto dobbiamo inserire i nuovi dati nella riga 25 (basso, riga 0)

# Procedura per lo scrolling

- Ricorda: le dimensioni della memoria video non cambiano, cambia solo l'area visibile
- I valori da 0 a 7 per effettuare lo scrolling vanno inseriti nei bit da 0 a 3 dei registri 53265 per lo scrolling verticale e 53270 per quello orizzontale

```
POKE 53265, (PEEK(53265) AND 248) + Y
```

```
POKE 53270, (PEEK(53270) AND 248) + X
```

- da sinistra verso destra la X deve andare da 0 a 7
- dal basso verso l'alto la Y deve andare da 0 a 7
- quando la X e/o la Y raggiungono il massimo (minimo) bisogna reimpostare il loro valore per continuare con lo scrolling

# Esempio scrolling

```
1 REM COPIA CARATTERI
5 PRINT CHR$(142)
10 POKE 52, 56: POKE 56, 56: CLR
20 POKE 56334, PEEK (56334) AND 254
30 POKE 1, PEEK(1) AND 251
35 PRINT "COPIA CARATTERI..."
40 FOR I = 0 TO 2047: POKE I + 14336, PEEK (I + 53248) : NEXT
50 POKE 1, PEEK(1) OR 4
60 POKE 56334, PEEK(56334) OR 1
70 POKE 53272, PEEK (53272) OR 14
90 REM REDEF @
100 FOR I=14336 TO 14336+7:READ V:POKE I,V:NEXT I
110 PRINT CHR$(147)
120 REM RIEMPI SCHERMO
130 POKE 53280,0:POKE 53281,0
140 MC=55296 : MS=1024
150 FOR I=0 TO 999 : POKE MC+I,1 : POKE MS+I,0 : NEXT
160 POKE 53270, PEEK(53270) AND 247
170 FOR X=7 TO 0 STEP -1: POKE 53270, (PEEK(53270) AND 248)+X : NEXT
180 GOTO 170
1000 DATA 36,24,126,231,189,189,165,36
```

Ridefinizione  
dei caratteri

Riempimento  
schermo

Scrolling  
orizzontale  
verso sinistra

# Note scrolling

- Riusciamo ad ottenere uno scrolling fluido in BASIC perché la memoria video è riempita sempre con lo stesso carattere e quindi non è necessario effettuare spostamenti di caratteri
- Se la copia dei caratteri avviene nel momento in cui il raster sta disegnando la riga coinvolta nella copia avremo uno sfarfallio
- È necessario realizzare routine in linguaggio macchina che copiano lo schermo in poco tempo in modo da non essere interrotte dal raster mentre spostano i caratteri in memoria



SID

# Il SID

- Il SID è il **coprocessore audio** del C64 ed è praticamente un sintetizzatore che mette a disposizione tre voci con controllo completo **ATTACK/DECAY/SUSTAIN/RELEASE (ADSR)**
- Dispone di filtri analogici, generazione del rumore e modulazione
- Come per il VIC-II sarà necessario agire su una serie di registri del SID che sono mappati nella memoria nell'intervallo di indirizzi da 54272 a 54296

# Esempio

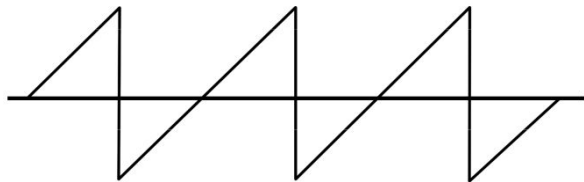
```
5 S=54272
10 FOR L=S TO S+24:POKE L,0:NEXT
20 POKES+5,9:POKES+6, 0
30 POKES+24,15
40 READ HF, LF, DR
50 IF HF<0 THEN END
60 POKE S+1,HF:POKE S,LF
70 POKE S+4,33
80 FOR T=1 TO DR:NEXT
90 POKE S+4,32:FOR T=1 TO 50:NEXT
100 GOTO 40
110 DATA 25,177,250,28,214,250
120 DATA 25,177,250,25,177,250
130 DATA 25,177,125,28,214,125
140 DATA 32,94,750,25,177,250
150 DATA 28,214,250,19,63,250
160 DATA 19,63,250,19,63,250
170 DATA 21,154,63,24,63,63
180 DATA 25,177,250,24,63,125
190 DATA 19,63,250,-1,-1,-1
```

registro di partenza del SID  
resettiamo tutti i registri del SID  
impostiamo i valori ADSR della voce #1  
impostiamo il volume al massimo  
leggiamo la parte alta e bassa della freq. e la durata  
terminiamo se HF<0  
impostiamo la frequenza della voce #1  
attiviamo l'onda sonora (dente di sega) per la voce #1  
durata  
disattiviamo l'onda sonora voce #1  
suoniamo la prossima nota  
istruzioni DATA per le frequenza e la durata di ogni  
nota

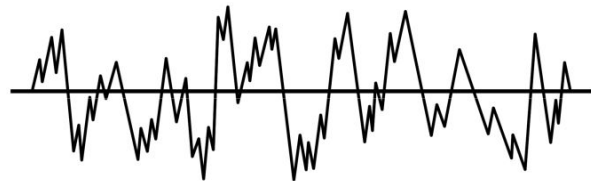
# Registri del SID

- Il registro S+24 controlla il volume da 0 a 15
- Per impostare la frequenza dell'onda sonora dobbiamo agire su due registri (HF e LF) per ogni voce
- Sia  $F_{out}$  la frequenza che vogliamo riprodurre, il valore di frequenza da inserire nei registri è
$$F_n = F_{out} * 17,03 \text{ (PAL)} \mid F_n = F_{out} * 16,40 \text{ (NTSC)}$$
- La parte alta di  $F_n$  va messa in HF la parte bassa in LF
$$HF = \text{INT}(F_n / 256)$$
$$LF = F_n - HF * 256$$

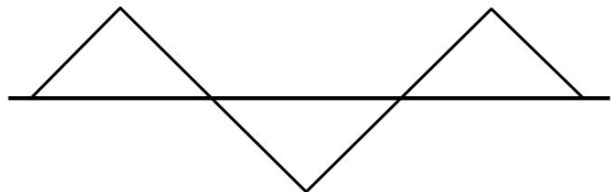
# Onda sonora



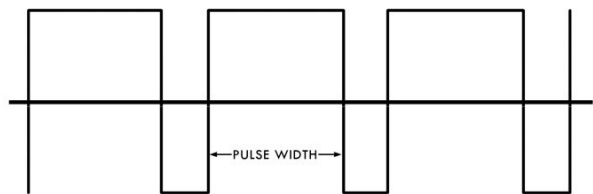
Dente di sega



Rumore



Triangolare

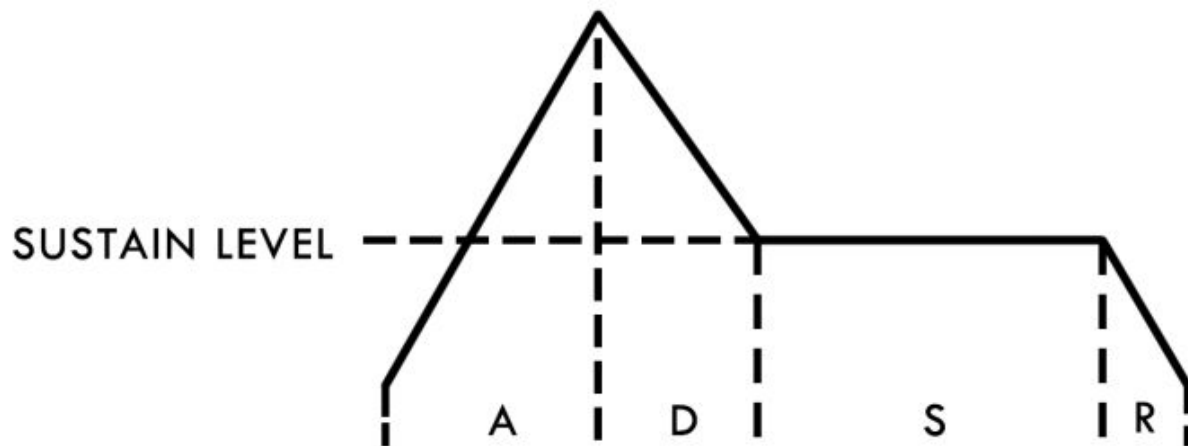


**Rettangolare** - è possibile impostare la lunghezza dell'impulso, in particolare definendo la porzione dell'onda che sarà alta usando 12bit. La parte bassa (L) in un registro dedicato, i restanti 4 bit della parte alta (H) sono in un altro registro.

$$P = 256 * H + L$$

$$P_{width} = (P / 40.95) \% \leftarrow \% \text{ parte alta dell'impulso}$$

# Inviluppo ADSR















- A** (ATTACK): la velocità con la quale l'onda raggiunge il volume massimo
- D** (DECAY): la velocità con la quale l'onda scende al volume medio
- S** (SUSTAIN): il tempo in cui l'onda si mantiene al livello medio
- R** (RELEASE): la velocità con la quale l'onda giunge al volume minimo

# Valori ADSR

È possibile definire l'inviluppo ADSR per ogni voce utilizzando dei valori di riferimento prestabiliti.

VALUE	ATTACK RATE (TIME/CYCLE)	DECAY/RELEASE RATE (TIME/CYCLE)
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

# Tabella registri SID

Address						Reg #	Data								Reg Name	Reg Type
A4	A3	A2	A1	A0	(Hex)		D7	D6	D5	D4	D3	D2	D1	D0		
<b>VOICE 1</b>																
0	0	0	0	0	00		F7	F6	F5	F4	F3	F2	F1	F0	Freq Lo	Write-only
1	0	0	0	0	01		F15	F14	F13	F12	F11	F10	F9	F8	Freq Hi	Write-only
2	0	0	0	1	02		PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0	PW LO	Write-only
3	0	0	0	1	03		—	—	—	—	PW11	PW10	PW9	PW8	PW HI	Write-only
4	0	0	1	0	04	NOISE					TEST	RING MOD	SYNC	GATE	Control Reg	Write-only
5	0	0	1	0	05	ATK3	ATK2	ATK1	ATK0	DCY3	DCY2	DCY1	DCY0		Attack/Decay	Write-only
6	0	0	1	1	06	STN3	STN2	STN1	STN0	RIS3	RIS2	RIS1	RIS0		Sustain/Release	Write-only
<b>VOICE 2</b>																
7	0	0	1	1	07		F7	F6	F5	F4	F3	F2	F1	F0	Freq Lo	Write-only
8	0	1	0	0	08		F15	F14	F13	F12	F11	F10	F9	F8	Freq Hi	Write-only
9	0	1	0	0	09		PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0	PW LO	Write-only
10	0	1	0	1	0A		—	—	—	—	PW11	PW10	PW9	PW8	PW HI	Write-only
11	0	1	0	1	0B	NOISE					TEST	RING MOD	SYNC	GATE	Control Reg	Write-only
12	0	1	1	0	0C	ATK3	ATK2	ATK1	ATK0	DCY3	DCY2	DCY1	DCY0		Attack/Decay	Write-only
13	0	1	1	0	0D	STN3	STN2	STN1	STN0	RIS3	RIS2	RIS1	RIS0		Sustain/Release	Write-only
<b>VOICE 3</b>																
14	0	1	1	1	0E		F7	F6	F5	F4	F3	F2	F1	F0	Freq Lo	Write-only
15	0	1	1	1	0F		F15	F14	F13	F12	F11	F10	F9	F8	Freq Hi	Write-only
16	1	0	0	0	10		PW7	PW6	PW5	PW4	PW3	PW2	PW1	PW0	PW LO	Write-only
17	1	0	0	0	11		—	—	—	—	PW11	PW10	PW9	PW8	PW HI	Write-only
18	1	0	0	1	12	NOISE					TEST	RING MOD	SYNC	GATE	Control Reg	Write-only
19	1	0	0	1	13	ATK3	ATK2	ATK1	ATK0	DCY3	DCY2	DCY1	DCY0		Attack/Decay	Write-only
20	1	0	1	0	14	STN3	STN2	STN1	STN0	RIS3	RIS2	RIS1	RIS0		Sustain/Release	Write-only
<b>Filter</b>																
21	1	0	1	0	15		—	—	—	—	—	FC2	FC1	FC0	FC LO	Write-only
22	1	0	1	1	16		FC10	FC9	FC8	FC7	FC6	FC5	FC4	FC3	FC HI	Write-only
23	1	0	1	1	17		RES3	RES2	RES1	RES0	Filt EX	Filt 3	Filt 2	Filt 1	RES/Filt	Write-only
24	1	1	0	0	18	3 OFF	HP	BP	LP	VOL3	VOL2	VOL1	VOL0		Mode/Vol	Write-only
<b>Misc</b>																
25	1	1	0	0	19		PX7	PX6	PX5	PX4	PX3	PX2	PX1	PX0	POTX	Read-only
26	1	1	0	1	1A		PY7	PY6	PY5	PY4	PY3	PY2	PY1	PY0	POTY	Read-only
27	1	1	0	1	1B		07	06	05	04	03	02	01	00	OSC3/Random	Read-only
28	1	1	1	0	1C		E7	E6	E5	E4	E3	E2	E1	E0	ENV3	Read-only

Indirizzo SID: 54272

## Note

- il bit GATE serve ad accendere la relativa voce
- i valori ADSR da 0 a 15 (tabella precedente) vanno inseriti nei relativi bit (parte alta e bassa)



# ADSR - Esempi

- Proviamo a modificare l'ADSR dell'esempio precedente

```
20 POKE S+5,88:POKE S+6,89:REM A=5;D=8;S=5;R=9
```

- Proviamo la forma d'onda triangolare

```
20 POKE S+5,9:POKE S+6,9:REM A=0;D=9;S=0;R=9
```

```
70 POKE S+4,17
```

```
90 POKE S+4,16:FOR T=1 TO 50:NEXT
```

- Forma d'onda rettangolare simile al piano

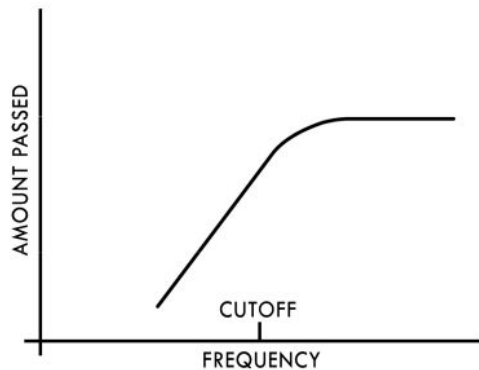
```
15 POKE S+3,8:POKE S+2,0
```

```
20 POKE S+5,9:POKE S+6,0: REM A=0;D=9;S=0;R=0
```

```
70 POKE S+4,65
```

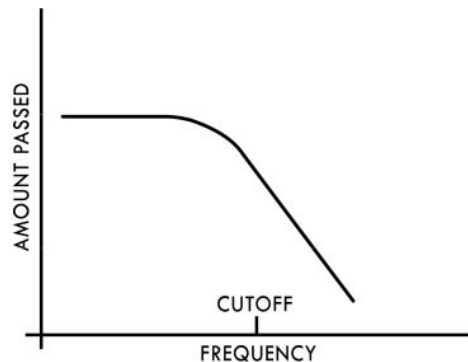
```
90POKE S+4,64:FOR T=1 TO 50:NEXT
```

# Filtri



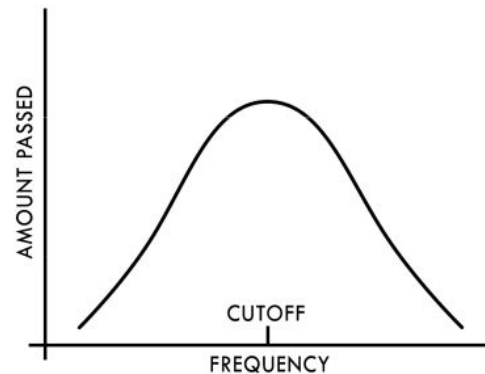
## Passa alto

Tutte le frequenze inferiori al cutoff vengono attutate



## Passa basso

Tutte le frequenze superiori al cutoff vengono attutate



## Passa banda

Tutte le frequenze intorno al cutoff superano il filtro le restanti frequenze vengono attutate

# Filtri

- La frequenza del filtro è indicata negli 11 bit nei registri 21 (3 bit meno significativi) e 22 (8 bit più significativi), il range del cutoff è 30 Hz-12 KHz
- I 4 bit più significativi del registro 24 attivano i filtri, più filtri possono essere attivati contemporaneamente
  - bit 6 - passa alto
  - bit 5 - passa banda
  - bit 4 - passa basso
- Variare i filtri durante l'involuppo ADSR può creare effetti interessanti

# Altri effetti

- Il registro 27 mette a disposizione un numero da 0 a 255 che dipende dall'onda sonora attiva sulla voce 3
- Il registro 28 è analogo al 27, ma il suo valore dipende dall'inviluppo ADSR della voce 3
- Il bit 7 del registro 24 rende udibile o meno l'output della voce 3
  - quando la voce 3 è utilizzata per modificare un'altra voce possiamo silenziare il suo output

# Altri effetti - esempio

```
10 S=54272
20 FOR L=0 TO 24:POKE S+L,0:NEXT
30 POKE S+14,5
40 POKE S+18,16
50 POKE S+3,1
60 POKE S+24,143
70 POKE S+6,240
80 POKE S+4,65
90 FR=5389
100 FOR T=1 TO 200
110 FQ=FR+PEEK(S+27)*3.5
120 HF=INT(FQ/256):LF=FQ-HF*256
130 POKE S+0,LF:POKE S+1,HF
140 NEXT
150 POKE S+24,0
```

la frequenza della voce 1 viene modificata dall'output della voce 3

# Sincronizzazione

- Il bit 1 del registro di controllo di ogni voce abilita la **sincronizzazione con un'altra voce**
  - la sincronizzazione è simile all'AND tra due onde sonore
  - voce 1 sync con voce 3
  - voce 2 sync con voce 1
  - voce 3 sync con voce 2

```
10 S=54272
20 FOR L=0 TO 24:POKE S+L,0:NEXT
30 POKE S+1,100
40 POKE S+5,219
50 POKE S+15,28
60 POKE S+24,15
70 POKE S+4,19
80 FOR T=1 TO 5000:NEXT
90 POKE S+4,18
100 FOR T=1 TO 1000:NEXT:POKE S+24,0
```

In questo esempio la voce 1 è sincronizzata con la voce 3 per ottenere un ronzio

# Modulazione

- Il bit 2 del registro di controllo di ogni voce abilita la modulazione dell'onda triangolare con un'altra voce
  - voce 1 è modulata con voce 3
  - voce 2 è modulata con voce 1
  - voce 3 è modulata con voce 2

```
10 S=54272
20 FOR L=0 TO 24:POKE S+L,0:NEXT
30 POKE S+1,130
40 POKE S+5,9
50 POKE S+15,30
60 POKE S+24,15
70 FOR L=1 TO 12:POKE S+4,21
80 FORT=1 TO 1000:NEXT:POKE S+4,20
90 FORT=1 TO 1000:NEXT:NEXT
```

In questo esempio la voce 1 è modulata con la voce 3 per ottenere un effetto simile al gong

# GAME OVER

PARTE II