

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

## **Parte I**

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

Apulia Retrocomputing

Università degli Studi di Bari Aldo Moro



- Socio di Apulia Retrocomputing
- Professore Associato presso il Dipartimento di Informatica dell'Università degli Studi di Bari
  - Metodi Avanzati di Programmazione
  - Metodi per il Ritrovamento dell'Informazione
  - Sviluppo di Videogiochi
  - Natural Language Processing
- Appassionato di retrogaming

**Pierpaolo Basile**  
**pierpaolo.basile@gmail.com**



@basilepp



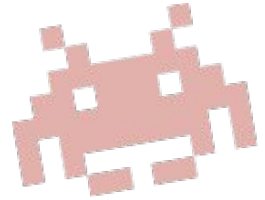
@pippokill81

# Di cosa parleremo?

- Breve storia del BASIC
- Introduzione ai concetti base del linguaggio
- Il BASIC del C64
- Primi programmi per il C64
- Introduzione alla grafica e al suono del C64
- Come sviluppare in BASIC usando i PC moderni
  - emulatori e tool moderni



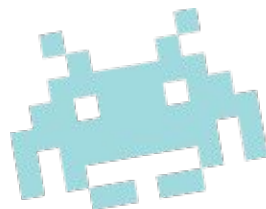
# IL BASIC



- Linguaggio di programmazione ad **alto livello**
- Sviluppato da **John G. Kemeny** e **Thomas E. Kurtz** al **Dartmouth College** nel **1964**
- Ideato come un linguaggio semplice e utilizzabile da studenti senza competenze informatiche o matematiche
- Il boom degli home computer negli anni 70 e 80 spinse la diffusione del BASIC
  - Furono sviluppati differenti dialetti
  - Versioni ridotte per poter risiedere in solo 4KB
  - Nel 1975 viene prodotto il Microsoft BASIC molto diffuso negli home computer

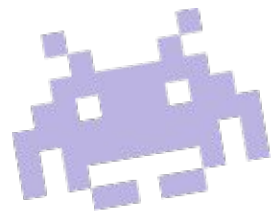
# II BASIC

- Iniziò a diffondersi con l'Altair BASIC sviluppato da Bill Gates e Paul Allen nel 1975
  - Occupava solo 4KB, venduto successivamente come Microsoft BASIC anche per altri computer
- Nel 1977 furono presentati 3 computer con interprete BASIC (basato sul Microsoft BASIC)
  - Apple II, Commodore PET, Radio Shack TRS-80
- Successivamente furono implementate altre versioni del BASIC
  - Atari BASIC, BBCmicro Basic (Acorn Computers)



# II BASIC

- Anche il primo IBM PC era fornito di un interprete BASIC sempre sviluppato da Microsoft
  - Successive evoluzioni: GW-BASIC, QuickBASIC, QBASIC
  - Altre versioni BASIC non Microsoft: TurboBasic (Borland), PowerBASIC
- Microsoft VisualBasic riporta alla ribalta il BASIC negli anni 90
- Versioni recenti:
  - True Basic, REALbasic
  - Gambas e FreeBasic (open-source)
  - QB64: <https://www.qb64.org/portal/>



# Caratteristiche

- Semplice da imparare
- **Poche istruzioni** non complesse
- Pochi costrutti strutturati (alcuni dialetti ne erano totalmente privi)
- Pensato come un linguaggio algoritmico
- Facilmente trasportabile
- Nasce come linguaggio compilato, ma molte sue implementazioni su home computer sono interpretate

# Hello world

```
10 PRINT "Hello World"  
20 END
```



# Ricorda che?

- In genere il BASIC degli home computer è **interpretato**
  - Questo significa che i comandi vengono eseguiti subito dopo aver premuto il tasto ENTER
- Per creare dei programmi è necessario mettere in sequenza le istruzioni
  - Nella maggior parte dei casi **la sequenza è determinata da un numero messo all'inizio della linea**
  - In questo caso l'istruzione non viene eseguita direttamente, ma memorizzata nella sequenza di istruzioni in memoria
- Il comando **RUN** permette l'esecuzione delle istruzioni nell'ordine determinato dai numeri di riga (dal più piccolo al più grande)

# Ricorda che?

- Nuove righe possono essere sempre inserite utilizzando nuovi numeri di riga non utilizzati
- Se si utilizza un numero di riga già esistente l'istruzione precedentemente assegnata a quel numero viene sovrascritta
- L'istruzione **END** termina l'esecuzione del programma
- L'istruzione **NEW** cancella il programma attualmente in memoria, l'istruzione **CLR** cancella solo le variabili
- L'istruzione **LIST** permette di stampare il programma presente in memoria

# Variabili

- Servono a **memorizzare un valore** in un'area di memoria
- **Ci riferiamo all'area di memoria utilizzando un nome**
- Essenzialmente in BASIC abbiamo due tipi di variabile
  - Numerica: 3, 4.2324, 4.3E+2
  - Stringa (sequenza di caratteri): "Pippo", "Hello World"
- Il nome delle variabili di tipo stringa termina con \$
- Possiamo utilizzare nome lunghi per le variabili, ma in genere **il BASIC degli home computer considera solo i primi caratteri**
  - Il BASIC del C64 considera solo i primi due caratteri, in questo caso AB e ABC individuano la stessa variabile, come CD\$ e CDE\$ individuano la stessa variabile stringa

# Variabili numeriche

- In genere le variabili numeriche memorizzano numeri reali con virgola
- Per specificare **una variabile intera si utilizza %** dopo il nome, ad esempio A%, NU%
  - nel C64 tutti i tipi di variabile occupano 5 byte anche se gli interi sono a 16bit (byte) quindi non ha senso utilizzare un intero per risparmiare memoria rispetto ad una variabile reale

# Matrici

- Per ogni tipo di variabile è possibile utilizzare delle matrici
- L'indice della matrice parte da 0
- Se le matrici hanno meno di 10 elementi possono essere utilizzate direttamente senza dichiarazione
  - $A(0)=2$ ,  $A(1)=5$ ,  $B$(0)="PIPP0"$ ,  $B$(1)="TOPOLINO"$
- Matrici di dimensioni maggiori di 10 vanno inizializzate con l'istruzione **DIM**
  - `DIM A(19)`, stiamo dichiarando un vettore di 20 elementi
  - `DIM A(4,4,4)`, stiamo dichiarando una matrice di 3 dimensioni per un totale di 125 ( $5*5*5$ ) elementi

# L'istruzione **INPUT**

- Serve a leggere un **INPUT** da tastiera e a memorizzarlo in una variabile (numerica o stringa)

```
10 INPUT A  
20 PRINT A
```

```
10 INPUT B$  
20 PRINT B$
```

# L'istruzione **INPUT**

- L'istruzione **INPUT** può prevedere un messaggio

```
10 INPUT "INSERISCI UN NUMERO";A  
20 PRINT "IL NUMERO E' ";A
```

```
10 INPUT "COME TI CHIAMO";B$  
20 PRINT "CIAO ";B$
```

# L'istruzione PRINT

- L'istruzione **PRINT** come abbiamo visto permette di stampare messaggi o variabili

```
10 INPUT "INSERISCI UN NUMERO";A  
20 PRINT "IL NUMERO E' ";A
```

```
10 INPUT "COME TI CHIAMO";B$  
20 PRINT "CIAO ";B$
```



# L'istruzione IF

- L'istruzione **IF <condizione> THEN <istruzione>** serve a valutare una <condizione> e nel caso questa sia vera esegue l'<istruzione> dopo il THEN altrimenti esegue la prossima istruzione (*successiva linea*)
- Molti BASIC dell'epoca non permettevano l'istruzione ELSE e dopo il THEN poteva esserci una sola istruzione
  - questo rendeva difficile la programmazione strutturata, ritorneremo su questo in seguito...

```
10 INPUT "INSERISCI UN NUMERO";A
20 IF A>10 THEN PRINT "IL NUMERO E' MAGGIORE DI 10"
30 END
```

# Il ciclo FOR

- L'istruzione **FOR** permette di eseguire un blocco di istruzioni più volte
- L'istruzione è del tipo  
`FOR I=1 TO 10 <istruzioni> NEXT I`
- Può essere utilizzata la keyword **STEP** per modificare l'incremento della variabile del ciclo

```
10 FOR I=1 TO 10
20 PRINT "ITERAZIONE";I
30 NEXT I
40 END
```

# Il ciclo FOR

- STEP modifica l'incremento della variabile del ciclo

```
10 FOR P=0 TO 10 STEP 2
20 PRINT "SOLO PARI";P
30 NEXT P
40 END
```

```
10 FOR C=10 TO 0 STEP -1
20 PRINT "COUNTDOWN";C
30 NEXT C
40 END
```

# **Salto incondizionati**

- Molti BASIC non permettono la programmazione strutturata per questo sono necessarie delle istruzioni di salto incondizionato
- Essenzialmente sono previste due istruzioni di salto
  - **GOTO <N>**, dove <N> è un numero di linea di un'istruzione del programma, l'esecuzione salta alla linea <N>
  - **GOSUB <N>**, dove <N> è un numero di linea come nel GOTO, ma in questo caso il salto è ad una subroutine che compie una sequenza di azioni, questa sequenza termina con l'istruzione **RETURN** che fa riprendere l'esecuzione all'istruzione successiva all'esecuzione della GOSUB

# GOTO

```
10 PRINT "LOOP INFINITO"  
20 GOTO 10
```

```
10 I=0  
20 PRINT I  
30 I=I+1  
40 IF I<10 THEN GOTO 20  
50 END
```

# GOSUB

```
10 INPUT "NUMERO";N
20 GOSUB 1000
30 PRINT "IL QUADRATO E' ",N
40 END
1000 N=N*N
1010 RETURN
```

# DEF FN

- Alcuni BASIC permettono di definire delle funzioni con il comando **DEF FN**
  - spesso l'implementazione è molto limitata, il nome della funzione segue le stesse regole delle variabili (max 2 caratteri) e la funzione può prevedere un solo parametro

```
10 DEF FNQ(X)=X*X
20 INPUT "INSERISCI UN NUMERO";N
30 PRINT "IL QUADRATO E' ";FNQ(N)
40 END
```

# GET

- L'istruzione GET serve a leggere un carattere da tastiera e ad assegnarlo ad una variabile
  - se non viene premuto nulla viene restituito un carattere nullo

```
10 PRINT "VUOI TERMINARE (S/N)"
20 GET K$:IF K$ = "" THEN GOTO 20
30 IF K$ = "S" THEN END
40 GOTO 10
```

I ':' permettono di concatenare più istruzioni su un'unica riga.  
Sono molto potenti, ma vanno utilizzati con cura!!!



# IF...THEN e i ':'

- Volendo dopo l'istruzione THEN posso concatenare più istruzioni utilizzando il ':', queste verranno eseguite se la condizione dell'IF è vera, altrimenti verrà eseguita l'istruzione successiva nel programma

```
10 INPUT N
20 IF N<10 THEN PRINT "MINORE DI 10" : GOTO 10
30 IF N>10 THEN PRINT "MAGGIORE DI 10" : GOTO 10
```

# Guess the number

- Utilizzando le istruzioni che abbiamo visto implementiamo un semplice gioco dove il giocatore ha 10 tentativi per indovinare un numero da 0 a 100
- Per fare questo dobbiamo introdurre la funzione **RND(X)** che genera un numero casuale tra 0 e 1
  - $X=0$  genera lo stesso numero,  $X<0$  cambia il seme del generatore casuale,  $X>0$  genera un nuovo numero
  - la variabile TI è una variabile di sistema che viene incrementata ogni sessantesimo di secondo, TI\$ memorizza sotto forma di stringa hhmmss. TI e TI\$ sono inizializzate a 0 durante l'accensione, il valore di TI\$ può essere modificato quello di TI no
  - **RND(-TI)** può essere un modo per inizializzare il generatore

```
5 N=RND(-TI)
10 N=INT(100*RND(1))+1)
20 PRINT "INDOVINA IL NUMERO A CUI STO PENSANDO (0-100)"
30 A=1 : REM TENTATIVI
40 IF A>10 THEN GOTO 400
50 INPUT "NUMERO";G
60 IF G=N THEN PRINT "HAI VINTO! TENTATIVI";A:GOTO 500
70 IF G<N THEN PRINT "TROPPO PICCOLO"
80 IF G>N THEN PRINT "TROPPO GRANDE"
90 A=A+1
100 GOTO 40
400 PRINT "NON HAI INDOVINATO, MI DISPIACE"
500 PRINT "ALTRA PARTITA (S/N)"
510 GET K$:IF K$="" GOTO 510
520 IF K$="S" THEN GOTO 10
530 IF K$="N" THEN END
540 GOTO 500
```

# Salti incondizionati ON...GOTO

- Esiste un'altra istruzione di salto incondizionato che è la **ON**: **ON** <exp> **GOTO** <N1>, <N2>, <N3>, ...
- Al posto del GOTO ci può essere GOSUB
- Se il valore di <exp> è  $\leq 0$  o supera la dimensione della lista di numeri viene eseguita l'istruzione alla riga successiva altrimenti viene eseguita l'istruzione che si trova alla posizione corrispondente al risultato di <exp>

```
ON X GOTO 200, 300, 400
```

# ...salti incondizionati

```
10 PRINT"MENU"  
20 PRINT"1.PRIMI"  
30 PRINT"2.SECONDI"  
40 PRINT"3.DOLCI"  
50 PRINT"4.ESCI"  
60 INPUT S  
70 ON S GOSUB 100,200,300,400  
80 GOTO 10  
100 PRINT"PASTA,ZUPPE":GOTO 10  
200 PRINT"CARNE,PESCE,INSALATE":GOTO 10  
300 PRINT"TORTE,SEMIFREDDI,GELATI":GOTO 10  
400 END
```

# READ e DATA

- L'istruzione **READ** può essere utilizzata per leggere una sequenza di valori presenti nelle istruzioni **DATA**
- Le istruzioni **DATA** servono a memorizzare dei valori in sequenza in memoria

```
10 FOR I=1 TO 10
20 READ A
30 PRINT A
40 NEXT I
50 END
60 DATA 0,10,20,30,40,50,60,70,80,90
```

# Funzioni numeriche

- **ABS(X)**: valore assoluto
- **ATN(X)**, **COS(X)**, **SIN(X)**, **TAN(X)**: arcotangente, coseno, seno, tangente
- **EXP(X)**, **LOG(X)**:  $e^X$  e  $\log_e(X)$
- **INT(X)**: restituisce la parte intera di un numero reale
- **SGN(X)**: restituisce il segno, -1, 0, +1
- **SQR(X)**: radice quadrata





































# Funzioni sulle stringhe

































- **ASC(X\$)**: restituisce il codice ASCII del primo carattere di X\$
- **CHR\$(X)**: restituisce il carattere associato al codice ASCII X
- **LEFT\$(X\$,X)**: restituisce la stringa degli X caratteri più a sinistra in X\$, analogo **RIGHT\$(X\$,X)**
- **MID\$(X\$,S,X)**: restituisce la stringa di X caratteri a partire dalla posizione S in X\$
- **STR\$(X)**: restituisce il numero X convertito in stringa
- **VAL(X\$)**: converte la stringa X\$ nel numero che rappresenta



# I codici carattere

- Nel C64 c'è una ROM che contiene i caratteri che vengono visualizzati a video
  - esistono due set di caratteri MAIUSCOLO/minuscolo, ogni set è ripetuto in modalità reverse
- I codici ASCII differiscono dal codice del carattere nella memoria video
  - in base al set selezionato ad ogni codice è associato un carattere diverso
- Il set di caratteri del C64 oltre a lettere e numeri prevede una serie di caratteri grafici detti PETSCII che possono essere utilizzati per la grafica

SET1	SET2	POKE	SET1	SET2	POKE	SET1	SET2	POKE
@		0	!		33		B	66
A	a	1	"		34		C	67
B	b	2	#		35		D	68
C	c	3	\$		36		E	69
D	d	4	%		37		F	70
E	e	5	&		38		G	71
F	f	6	'		39		H	72
G	g	7	(		40		I	73
H	h	8	)		41		J	74
I	i	9	*		42		K	75
J	j	10	+		43		L	76
K	k	11	,		44		M	77
L	l	12	-		45		N	78
M	m	13	.		46		O	79
N	n	14	/		47		P	80
O	o	15	0		48		Q	81
P	p	16	1		49		R	82
Q	q	17	2		50		S	83
R	r	18	3		51		T	84
S	s	19	4		52		U	85
T	t	20	5		53		V	86
U	u	21	6		54		W	87
V	v	22	7		55		X	88
W	w	23	8		56		Y	89
X	x	24	9		57		Z	90
Y	y	25	:		58			91
Z	z	26	;		59			92
[		27	<		60			93
£		28	=		61			94
]		29	>		62			95
↑		30	?		63	SPACE		96
←		31			64			97
SPACE		32		A	65			98

SET1	SET2	POKE	SET1	SET2	POKE	SET1	SET2	POKE
		99			109			119
		100			110			120
		101			111			121
		102			112			122
		103			113			123
		104			114			124
		105			115			125
		106			116			126
		107			117			127
		108			118			

Codes from 128 – 255 are reversed images of codes 0 – 127.

- POKE 53272,21 o PRINT CHR\$(142)  
seleziona il set 1
- POKE 53272,23 o PRINT CHR\$(14)  
seleziona il set 2
- SHIFT+C= per passare da un set all'altro
- i caratteri in reverse si ottengono sommando 128 al codice del carattere

Codici carattere C64

# Codici ASCII

PRINT	CHR\$	PRINT	CHR\$	PRINT	CHR\$	PRINT	CHR\$
	0		22	,	44	B	66
	1		23	-	45	C	67
	2		24	.	46	D	68
	3		25	/	47	E	69
	4		26	0	48	F	70
WHT	5		27	1	49	G	71
	6	RED	28	2	50	H	72
	7	CRSR→	29	3	51	I	73
DISABLES SHIFT	8	GRN	30	4	52	J	74
ENABLES SHIFT	9	BLU	31	5	53	K	75
	10	SPACE	32	6	54	L	76
	11	!	33	7	55	M	77
	12	"	34	8	56	N	78
RETURN	13	#	35	9	57	O	79
SWITCH TO LOWER CASE	14	\$	36	:	58	P	80
	15	%	37	;	59	Q	81
	16	&	38	<	60	R	82
CRSR↓	17	'	39	=	61	S	83
RVS ON	18	(	40	>	62	T	84
CLR HOME	19	)	41	?	63	U	85
INST DEL	20	*	42	@	64	V	86
	21	+	43	A	65	W	87

PRINT	CHR\$	PRINT	CHR\$	PRINT	CHR\$	PRINT	CHR\$
X	88		114	f8	140		166
Y	89		115	SHIFT RETURN	141		167
Z	90		116	SWITH TO UPPER CASE	142		168
[	91		117		143		169
£	92		118	BLK	144		170
]	93		119	CRSR↑	145		171
↑	94		120	RVS OFF	146		172
←	95		121	CLR HOME	147		173
	96		122	INST DEL	148		174
	97		123	Brown	149		175
	98		124	Lt Red	150		176
	99		125	Gray 1	151		177
	100		126	Gray 2	152		178
	101		127	Lt Green	153		179
	102		128	Lt Blue	154		180
	103	Orange	129	Gray 3	155		181
	104		130	PUR	156		182
	105		131	←CRSR	157		183
	106		132	YEL	158		184
	107	f1	133	CYN	159		185
	108	f3	134	SPACE	160		186
	109	f5	135		161		187
	110	f7	136		162		188
	111	f2	137		163		189
	112	f4	138		164		190
	113	f6	139		165		191

192-223=96-127  
224-254=160-190  
255=126

# Le istruzioni POKE e PEEK

- L'istruzione **POKE A,V** serve a **scrivere in una locazione di memoria A un valore V**
  - il **C64** può indirizzare **64KB** quindi il valore di A è nell'intervallo 0-65535
  - il **C64** è un computer ad **8-bit** quindi V sarà nell'intervallo 0-255
  - l'istruzione POKE permette di scrivere in qualsiasi area della memoria del C64 (*alcuni indirizzi fanno riferimento alla ROM quindi in realtà non verrà scritto nulla*)
- L'istruzione **PEEK(A)** restituisce il valore memorizzato all'indirizzo A

# Le istruzioni POKE e PEEK

- Queste istruzioni sono importantissime perché permettono di **accedere ai registri dei coprocessori del C64**
  - **SID**: chip audio
  - **VIC-II**: chip video
- Poiché i coprocessori condividono lo stesso spazio di memoria indirizzabile dal C64 (64KB) **alcune locazioni di memoria corrispondono ai registri di questi processori**
- Nello stesso spazio di memoria troveremo RAM, ROM, co-processor, dispositivi I/O
  - questo significa che non possiamo avere un accesso diretto a tutta la RAM disponibile, perché lo spazio di indirizzamento non è sufficiente

# Grafica con i caratteri

- In modalità testo il C64 ha una risoluzione di **40x25 caratteri**
- In modalità testo il VIC-II utilizza **due aree di memoria**
  - l'**area per memorizzare i caratteri** a video, dall'indirizzo 1024 all'indirizzo 2023<sup>1</sup>
  - l'**area colore**, da 55296 a 56295
- Questo significa che con l'istruzione POKE possiamo accedere a queste aree di memoria
  - per visualizzare (leggere) caratteri a video in una posizione specifica
  - per cambiare (leggere) il colore di un carattere in una determinata posizione

<sup>1</sup> questi indirizzi possono cambiare e dipendono dall'area di memoria scelta per il VIC-II

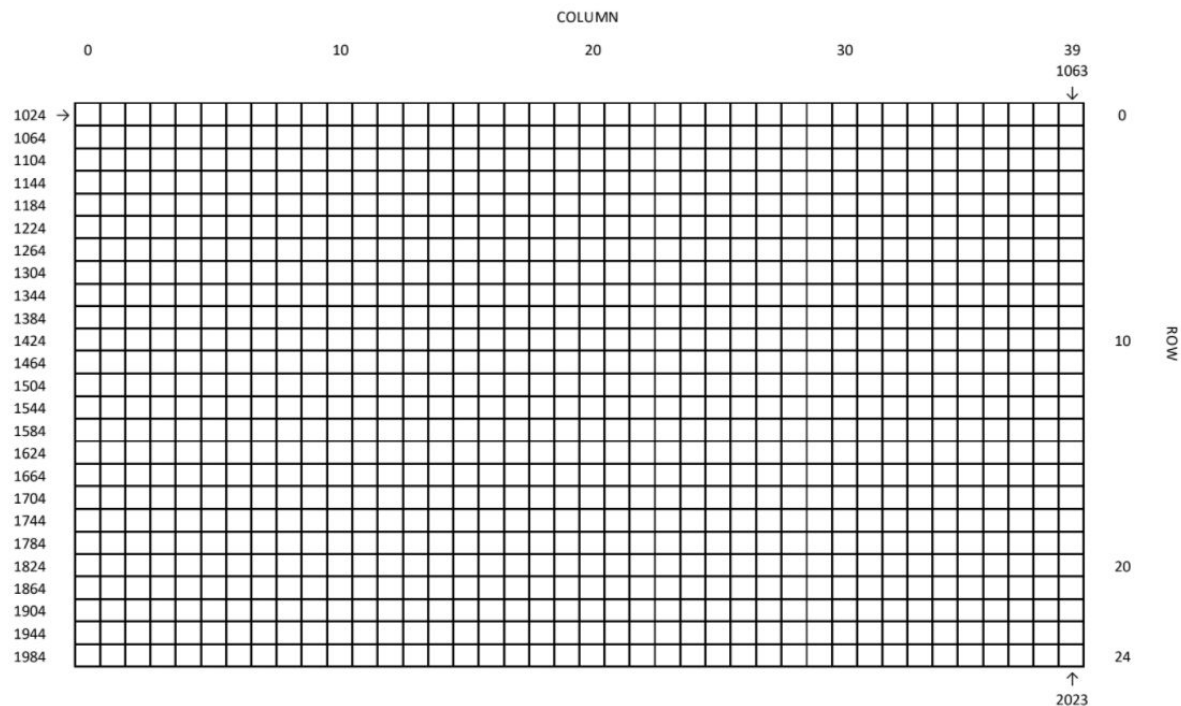
# I colori

- Il VIC-II è capace di visualizzare **16 colori**

0	1	2	3
4	5	6	7
8	9	A	B
C	D	E	F

- è possibile cambiare il **colore del bordo**, inserendo un valore da 0 a 15 nella locazione **53280**
- il **colore di sfondo** si può cambiare modificando la locazione **53281**
- questi sono anche i codici colore da utilizzare nella memoria colore

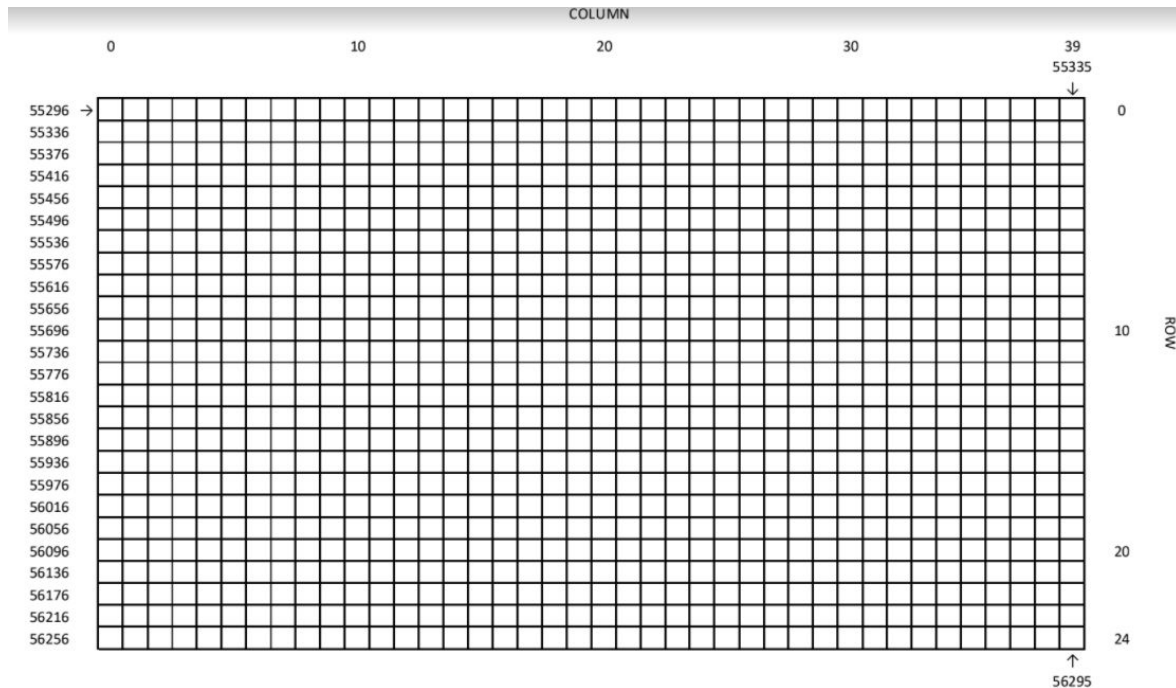
# Mappa memoria caratteri



Dati X la colonna e Y la riga, la locazione di memoria si ottiene come:  
$$1024 + Y \cdot 40 + X$$




# Mappa memoria colori



Dati X la colonna e Y la riga, la locazione di memoria si ottiene come:  
$$55296 + Y \cdot 40 + X$$

# Pallina rimbalzante

```
10 PRINT CHR$(5):PRINT CHR$(147)
20 POKE 53280,2 : POKE 53281,0
30 X = 1 : Y = 1
40 DX = 1 : DY = 1
50 O = X + 40 * Y
60 POKE 1024 + O, 81
80 FOR T = 1 TO 10 : NEXT T
90 POKE 1024 + O, 32
100 X = X + DX
110 IF X <= 0 OR X >= 39 THEN DX = -DX
120 Y = Y + DY
130 IF Y <= 0 OR Y >= 24 THEN DY = -DY
140 GOTO 50
```

colore bianco e cancella schermo  
colore cornice e sfondo  
coordinate  
incremento coordinate  
calcolo offset memoria  
stampa pallina   
ciclo di attesa  
stampa spazio  
incrementa X  
controlla incremento X  
incrementa Y  
controlla incremento Y  
ripeti

# Sistemi numerici

- Quando parliamo di calcolatori è importante avere dimestichezza con due sistemi numerici
  - **sistema binario:** è un sistema posizionale con 2 cifre (0, 1)
  - **sistema esadecimale:** è un sistema posizionale con 16 cifre (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)
- Nei sistemi posizionali il valore di una cifra  $c$  è dato dalla sua posizione, se il sistema prevede  $n$  cifre il valore di una cifra ad una determinata posizione  $p$  è  $c \cdot n^p$ 
  - le posizioni partono da 0 da destra verso sinistra

# Sistemi numerici (conversione in decimale)

- Decimale: 384
  - $3 \cdot 10^2 + 8 \cdot 10^1 + 4 \cdot 10^0 = 384$
- Binario: 10011011
  - $1 \cdot 2^7 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^1 + 1 \cdot 2^0 = 155$
- Esadecimale: FA43
  - $15 \cdot 16^3 + 10 \cdot 16^2 + 4 \cdot 16^1 + 3 \cdot 16^0 = 64067$
- Ogni cifra esadecimale si codifica con 4 bit  $2^4 = 16$ 
  - $0 = 0000, 1=0001, 2=0010, \dots F=1111$
  - la conversione esadecimale-binario e viceversa è immediata
    - AF = 10101111, 11000011 = C3

# Operatori booleani

A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1

A	NOT
0	1
1	0

# Maschere di bit

- Molto spesso è necessario modificare solo alcuni valori di un numero binario, ovvero **modificare lo stato di alcuni bit**
- Supponiamo di avere il valore 10010001 e di voler **impostare a 1** i bit 6 e 1 senza alterare gli altri bit, in questo caso è sufficiente fare un **OR** con il valore 01000010
- Se volessimo **impostare a 0** i bit 7 e 0 dovremmo fare un **AND** con il valore 01111110

# Maschere di bit

Stato  
iniziale →

1	0	0	1	0	0	0	1	OR
0	1	0	0	0	0	1	0	MASK
1	1	0	1	0	0	1	1	

Stato  
iniziale →

1	0	0	1	0	0	0	1	AND
0	1	1	1	1	1	1	0	MASK
0	0	0	1	0	0	0	0	

# I modi carattere

Il VIC-II prevede tre modi carattere:

1. caratteri **standard**

- a. ogni carattere ha dimensione 8x8 pixel e 2 colori (sfondo e primo piano)

2. caratteri **multicolor**

- a. ogni carattere ha dimensione 4x8 pixel e ogni pixel è codificato da 2 bit (4 colori)

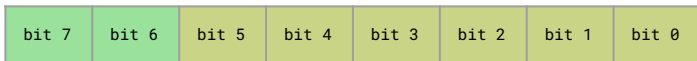
00	colore di sfondo (condiviso con tutti i caratteri)
01	colore nella locazione 53282 (condiviso con tutti i caratteri)
10	colore nella locazione 53283 (condiviso con tutti i caratteri)
11	colore specificato nella memoria colore



# I modi carattere

## 3. extended background color

a. in questo modo il set di caratteri è limitato a 64 e i primi 2 bit di ogni codice carattere identificano il colore dello sfondo



b. il colore di primo piano è specificato nella memoria colore

c. questa modalità non può essere utilizzata insieme al multicolor

00	colore di background in 53281
01	colore di background in 53282
10	colore di background in 53283
11	colore di background in 53284

# I set di caratteri

- Abbiamo visto che il C64 ha 2 set di caratteri ognuno dei quali ha 128 caratteri
  - set 1: maiuscole/grafici
  - set 2: minuscole/maiuscole
- Ogni set ha la modalità reverse in cui i pixel sono invertiti
- In totale abbiamo  $128 \times 4 = 512$  caratteri, poiché ogni carattere è composto da  $8 \times 8$  pixel per descrivere un carattere abbiamo bisogno di 8 byte
- In totale la memoria caratteri è  $512 \times 8 = 4096$  (4KB)
- Questi caratteri sono contenuti in una memoria ROM che però deve essere indirizzata sempre nello stesso spazio di indirizzi di 64KB

## ROM caratteri

- La ROM caratteri per ogni codice carattere contiene 8 byte che descrivono il corrispondente carattere

[illegible][illegible]

# Grafica caratteri

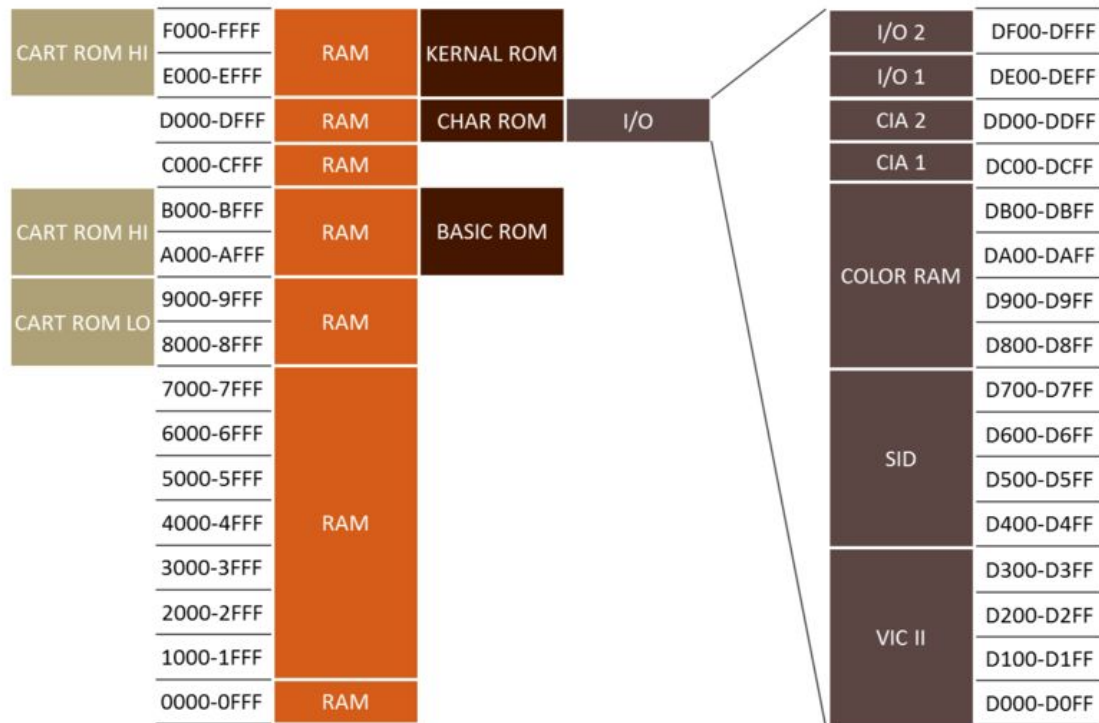
I caratteri oltre che per il testo possono essere utilizzati per la grafica. Per fare questo occorre:

- copiare la ROM caratteri nella RAM
- dire al VIC-II che la memoria caratteri si trova in RAM e non in ROM
- modificare i caratteri nella RAM in modo da personalizzarli
- nel modo standard avremo un solo colore di primo piano e uno di sfondo
- per avere più colori dobbiamo utilizzare il multicolor o l'extended background facendo attenzione a come codificare i pixel nei caratteri
- considerato che possiamo utilizzare un solo set per volta, al massimo possiamo utilizzare 256 caratteri

# Grafica caratteri

- Tutte le operazioni precedenti non sono banali
  - il VIC-II può vedere solo 16KB di memoria all'interno dei 64KB indirizzabili dal C64, quindi abbiamo 4 blocchi da 16, dobbiamo decidere in quale blocco copiare i caratteri
  - quando accediamo alla ROM per leggere i caratteri è necessario disattivare I/O, interrupt e tastiera
  - quando copiamo dalla ROM alla RAM stiamo riducendo la memoria disponibile per i programmi in RAM e dobbiamo avvisare di questo l'interprete BASIC
  - terminata la copia dobbiamo abilitare I/O, interrupt, tastiera
  - a questo punto possiamo modificare i caratteri nella RAM

# Memory map



[https://www.c64-wiki.com/wiki/Memory\\_Map](https://www.c64-wiki.com/wiki/Memory_Map)

<https://sta.c64.org/cbm64mem.html>

# VIC-II e memoria

L'indirizzo 53272 dice al VIC-II in quale area di memoria deve andare a cercare il set di caratteri. I bit interessati sono dal 3 a 1, bisogna fare attenzione a non cambiare gli altri bit

**POKE 53272, (PEEK(53272) AND 240) OR A**

A	BITS	LOCATION*	
		DECIMAL	HEX
0	XXXX000X	0	\$0000-\$07FF
2	XXXX001X	2048	\$0800-\$0FFF
4	XXXX010X	4096	\$1000-\$17FF ROM IMAGE in BANK 0 & 2 (default)
6	XXXX011X	6144	\$1800-\$1FFF ROM IMAGE in BANK 0 & 2
8	XXXX100X	8192	\$2000-\$27FF
10	XXXX101X	10240	\$2800-\$2FFF
12	XXXX110X	12288	\$3000-\$37FF
14	XXXX111X	14336	\$3800-\$3FFF

L'indirizzo va aggiunto all'indirizzo di partenza del blocco da 16K indirizzato dal VIC-II!!! All'avvio questo blocco parte da 0.

# VIC-II Memory bank

- Il VIC all'interno dei 64K può indirizzare 4 blocchi diversi da 16K di RAM
- Il blocco è selezionato dai bit 1 e 0 del registro 56576 del CIA-2

Bank no.	Bit pattern in 56576/\$DD00	Address		ROM chars available?
		Dec	Hex	
0	xxxxxx11	0-16383	\$0000-\$3FFF	Yes, at 4096-8191 \$1000-\$1FFF
1	xxxxxx10	16384-32767	\$4000-\$7FFF	No
2	xxxxxx01	32768-49151	\$8000-\$BFFF	Yes, at 36864-40959 \$9000-\$9FFF
3	xxxxxx00	49152-65535	\$C000-\$FFFF	No

- All'accensione è selezionato il blocco 0 con ROM caratteri visibile

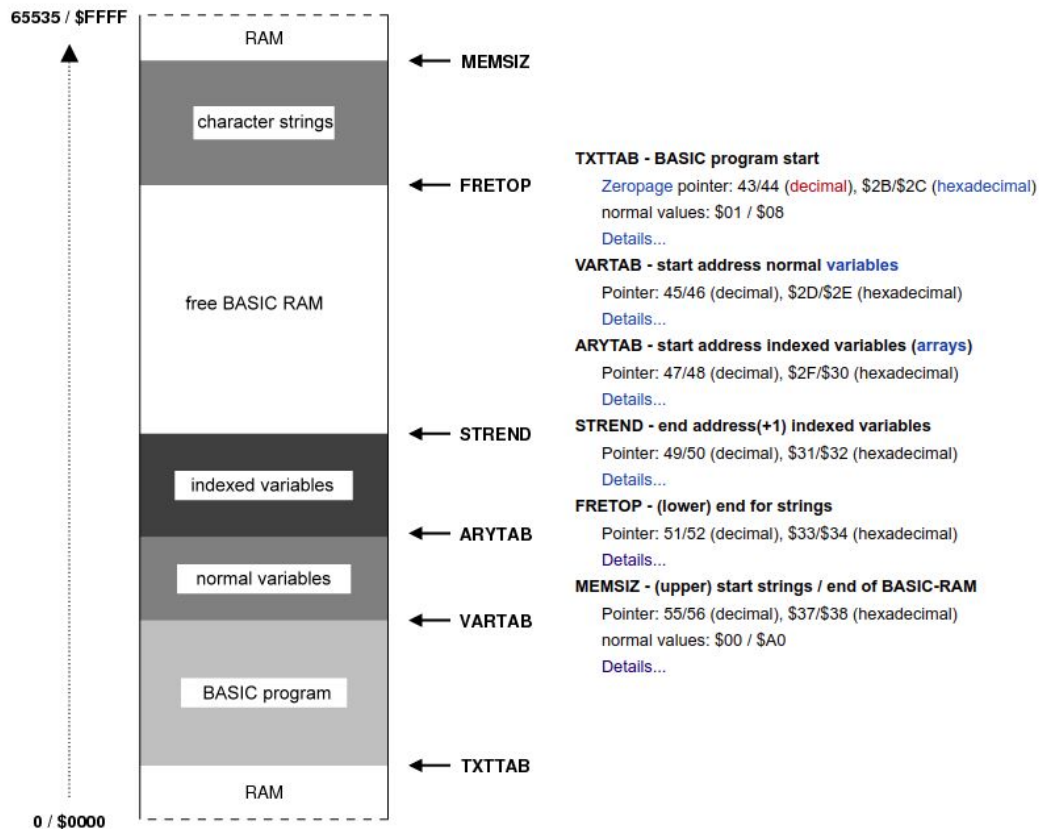


# Grafica caratteri

```
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
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
80 END
```

- seleziona il set 1 di caratt.
- riserva memoria in RAM
- disabilita interrupt
- fuori I/O dentro ROM
- copia i primi 256 caratteri dalla ROM alla RAM
- dentro I/O fuori ROM
- abilita interrupt
- seleziona il banco RAM
- termina

# BASIC memory



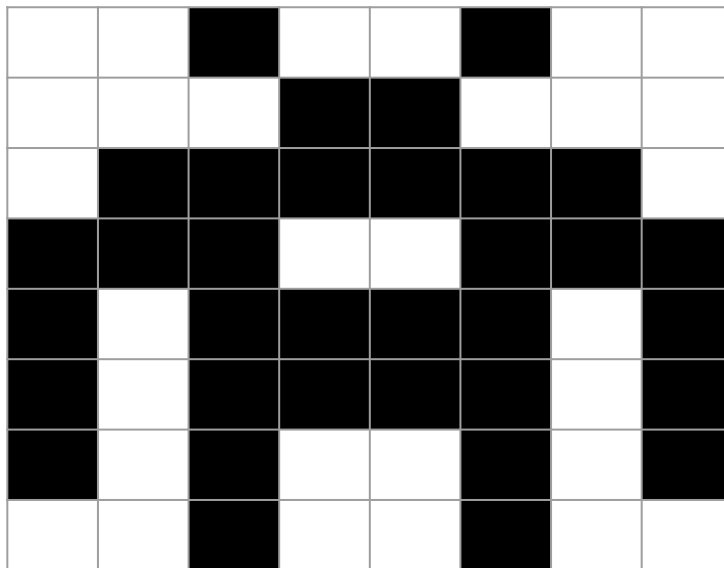
# Grafica caratteri

- Adesso a partire dalla locazione in RAM 14336 abbiamo il set 1 di caratteri completo
- Il codice 0 è la '@' se andiamo a leggere i primi 8 byte a partire da 14336 avremo i valori che descrivono il carattere

```
FOR I=14336 TO 14336+7:PRINT PEEK(I):NEXT I
```

- Adesso possiamo ridefinire il carattere '@' con un nostro carattere

# Ridefinizione carattere



00100100	36
00011000	24
01111110	126
11100111	231
10111101	189
10111101	189
10100101	165
00100100	36

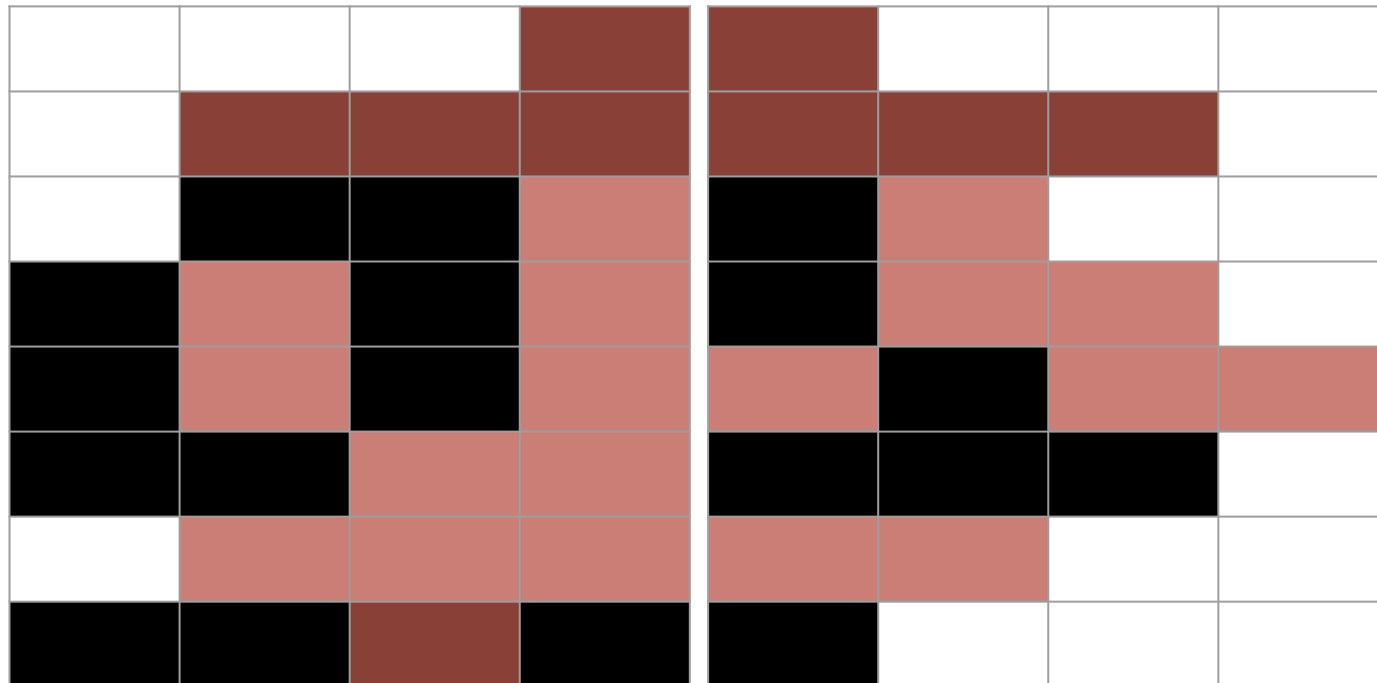
```
NEW : REM CANCELLA PRG PRECEDENTE
10 FOR I=14336 TO 14336+7:READ V:POKE I,V:NEXT I
20 DATA 36,24,126,231,189,189,165,36
```

# Caratteri multicolor

Ogni carattere ha dimensione 4x8 pixel e ogni pixel è codificato da 2 bit (4 colori)

00	colore di sfondo (condiviso con tutti i caratteri)
01	colore nella locazione 53282 (condiviso con tutti i caratteri)
10	colore nella locazione 53283 (condiviso con tutti i caratteri)
11	colore specificato nella memoria colore

# Caratteri multicolor



00 sfondo (53281)

01 (53282)

10 (53283)

11 colore car.

Ridefiniamo i caratteri 64 e 65 in multicolor

# Caratteri multicolor

00	00	00	10	10	00	00	00
00	10	10	10	10	10	10	00
00	01	01	11	01	11	00	00
01	11	01	11	01	11	11	00
01	11	01	11	11	01	11	11
01	01	11	11	01	01	01	00
00	11	11	11	11	11	00	00
01	01	10	01	01	00	00	00

00 sfondo (53281)

01 (53282)

10 (53283)

11 colore car.

Ridefiniamo i caratteri 64 e 65 in multicolor

# Caratteri multicolor

00	00	00	10	00000010	2
00	10	10	10	00101010	42
00	01	01	11	00010111	23
01	11	01	11	01110111	119
01	11	01	11	01110111	119
01	01	11	11	01011111	95
00	11	11	11	00111111	63
01	01	10	01	01011001	89

Carattere 64 -> 2,42,23,119,119,95,63,89



# Caratteri multicolor

10	00	00	00	10000000	128
10	10	10	00	10101000	168
01	11	00	00	01110000	112
01	11	11	00	01111100	124
11	01	11	11	11011111	223
01	01	01	00	01010100	84
11	11	00	00	11110000	240
01	00	00	00	01000000	64

Carattere 65 -> 128, 168, 112, 124, 223, 84, 240, 64

# Caratteri multicolor

```
5 PRINT CHR$(142)
10 POKE 52, 56: POKE 56, 56: CLR
20 POKE 56334, PEEK (56334) AND 254
30 POKE 1, PEEK(1) AND 251
40 FOR I = 0 TO 2047: POKE I + 14336, PEEK (I + 53248) : NEXT
50 POKE 1, PEEK(1) OR 4
60 POKE 56334, PEEK(56334) OR 1
70 POKE 53272, PEEK (53272) OR 14
80 FOR I=14336+512 TO 14336+512+15:READ V:POKE I,V:NEXT I
90 POKE 53281,1 : POKE 53282,0 : POKE 53283,10
100 PRINT CHR$(147) : POKE 55296+492,10 : POKE 55296+493,10
110 POKE 1024+492,64 : POKE 1024+493,65
115 POKE 53270, PEEK(53270) OR 16
120 GET K$:IF K$="" THEN GOTO 120
125 POKE 53270, PEEK(53270) AND 239
130 DATA 2,42,23,119,119,95,63,89
140 DATA 128, 168, 112, 124, 223, 84, 240, 64
```

# Caratteri multicolor

```
80 FOR I=14336+512 TO 14336+512+15:READ V:POKE I,V:NEXT I
90 POKE 53281,1 : POKE 53282,0 : POKE 53283,10
```

- La **linea 80** ridefinisce i caratteri 64 e 65.
- Per calcolare l'area di memoria RAM da modificare dobbiamo sommare all'indirizzo di partenza dei caratteri (14336) il valore  $64 \times 8$  (codice 64).
- Poi copiamo 16 valori ( $8\text{byte} \times 2 = 2$  caratteri)
- La **linea 90** imposta i 3 colori condivisi tra tutti i caratteri

# Caratteri multicolor

```
100 PRINT CHR$(147) : POKE 55296+492,10 : POKE 55296+493,10
110 POKE 1024+492,64 : POKE 1024+493,65
```

- La linea 100 pulisce lo schermo e alla riga 12 e alla colonna 12 ( $40 \times 12 + 12 = 492$ ) e 13 (493) imposta il codice colore 10
  - la memoria colore accetta codici a 4bit (0-15), nel nostro caso il colore ha codice (0010 = 2), **ma dobbiamo mettere il bit 4 ad 1 per indicare che il carattere è in multicolor (1010 = 10)**
- La linea 110 nella memoria video alla riga 12 colonna 12 scrive il carattere 64 alla colonna 13 il carattere 65

# Extended background color

```
10 POKE 53281,1
20 POKE 53282,2
30 POKE 53283,5
40 POKE 53284,6
50 POKE 53265,PEEK(53265) OR 64
60 P=1024+40*12+18 : REM MEM SCHERMO
70 POKE P,1:POKE P+1,65:POKE P+2,129: POKE P+3,193
80 GET K$:IF K$="" THEN GOTO 80
90 END
```

- In questo caso possiamo scegliere 4 colori di background (linee 10-40)
- I primi 64 caratteri di ogni set possono essere utilizzati con un colore di primo piano (mem. colori) e 4 colori di sfondo
- I 4 colori di sfondo sono selezionati dai primi due bit di ogni codice carattere
- Nell'esempio utilizziamo la 'A' che ha codice 1 (00000001) e cambiamo i primi due bit per selezionare i vari colori di sfondo (01000001 = 65, 10000001 = 129, 11000001 = 193)

# GAME OVER

PARTE I