LudoProgrammazione su 6502/6510

L’anellide solitario

Benvenuti al secondo appuntamento di LudoProgrammazione, in cui, con il pretesto di illustrare dei programmi tanto inutili quanto divertenti (almeno per me), si vogliono introdurre alcuni concetti base di programmazione in assembly 6502/6510, utilizzando il molto amato Commodore 64.
Il programma che andrò a descrivere visualizzerà sul nostro schermo un anellide (chiamarlo verme mi pareva brutto) che se ne viaggia tranquillo per lo schermo, incontrando ostacoli da evitare, consistenti nelle altre lettere presenti nello schermo su cui potrebbe sbattrere durante il suo peregrinare. Naturalmente il suo muoversi non ci dovrà ostacolare minimamente nello scrivere a video tramite la tastiera, se non altro per costruire attorno al nostro nuovo animaletto un percorso con cui farlo divertire.

Il movimento

Il movimento dell’anellide non sarà solamente del tipo strisciante, bensì risulterà un po' più complesso, sarà del tipo “propagatorio” (ammesso che mi venga passato il termine). Praticamente in nostro anellide sarà composto da un certo numero di settori (un settore, un carattere, una ‘o’ minuscola per la precisione), ma uno di questi settori risulterà speciale (una ‘O’ maiuscola). Speciale in quanto ci darà la sensazione (con un po' di fantasia) che il nostro verme, prima di compiere un passo in avanti, trasmetta ad ogni singolo settore, un’onda che, partendo dalla coda fino ad arrivare alla testa, gli permetta di compiere l’agognato passettino. Quindi i movimenti di cui tenere conto fin’ora sono due, la propagazione del settore in movimento e poi l’effettivo avanzamento del nostro vermiciattolo. Naturalmente dovremmo fornirlo dell’intelligenza necessaria a deviare, cercando una via d’uscita, nel caso in cui incontrasse un ostacolo o di ritornare sui propri passi nel caso non vedesse vie d’uscita, il tutto tenendo presente che i versi di marcia di cui doteremo l’animaletto sono i 4 principali nella rosa dei venti, Nord, Sud, Est ed Ovest.

Sulla carta

Il movimento che vogliamo realizzare sostanzialmente implica che da qualche parte dobbiamo tenere memorizzate le posizioni di ogni singolo settore del nostro anellide, per far ciò ci riserveremo un numero di words pari al numero di settori, ogni word ovviamente indicherà in quale indirizzo della memoria video andrà stampato quel determinato settore. Per tenere traccia di dove si trovi il settore in “movimento” (la O maiuscola per capirci) utilizzeremo un byte-indice, che chiameremo Sector, ed un altro byte-indice ci servirà per determinare dove si trovi la testa, battezzandolo Head. Ma perché utilizzare un indice anche per la testa, non sarebbe bastato dire che la testa è sempre l’ultima (o la prima) delle word che identifica il corpo e far scorrere l’indicatore Sector da 0 fino all’ultima word ? Sarebbe potuta essere una soluzione, ma non mi andava l’idea di dover spostare tutti gli indirizzi di ogni word di una posizione nel momento in cui Sector avrebbe raggiunto la testa.

Ciclo	Video	Sector	Head	Body
1	Ooooo	0	4	$0400,$0401,$0402,$0403,$0404
2	oOooo	1	4	$0400,$0401,$0402,$0403,$0404
...	...			
5	ooooO	4	4	$0400,$0401,$0402,$0403,$0404
6	Ooooo	1	0	$0405,$0401,$0402,$0403,$0404

Fig. 1: Movimento dell'anellide verso Est. Nella colonna all'estrema destra vengono evidenziate in rosso le locazioni di memoria video (su C64) puntate dal settore che si muove (Sector, O maiuscola) mentre le locazioni sottolineate indicano dove si trova la testa (Head)

L’idea è quella di spostare, alla bisogna, l’indicatore Head in avanti (o indietro) tenendo conto degli sforamenti all’inizio e alla fine rispetto alla lunghezza dell’anellide. In questo modo ci limiteremo (al massimo) a sovrascrivere una sola word nel momento in cui verrà fatto un passo oltre quella che a video risulta la capoccia del vermiciattolo. Nella figura 1 vengono mostrati i sei cicli (supponendo un numero di settori di cinque caratteri) necessari per l’avanzamento di un passo e come si comporteranno i vari indici e la memoria.

Dovendo gestire la direzione, avanti o indietro e il verso (Est, Sud, Ovest, Nord, proprio in questo ordine) avremo bisogno di altri preziosi bytes. Fin’ora quindi abbiamo:

```
NumSec          = 6             ; Numero settori dell'anellide


Body    words $0000*NumSec      ; Serie di indirizzi ai vari settori dell'anellide
Sector  byte  $00                ; indice del settore "alzato"
Head    byte  $00                ; indice della testa
Dir     byte  $01                ; Verso del movimento $01 Avanti $FF indietro
Verse   byte  $00                ; Verso di marcia Est Sud Ovest Nord (0,1,2,3)
```

Nella riga di dichiarazione di Body è stato indicato che si vogliono riservare “Numsec” words inizializzandole a $0000, questo grazie all’indicazione assemply di moltiplicazione ‘*’.

Sommare o sottrarre ?

Da qualche parte nel ciclo principale dovremo far avanzare o indietreggiare il nostro Sector, dipendentemente dalla direzione (Dir) presa dal nostro lombrico. Verrebbe spontaneo quindi utilizzare una sorta di salto condizionato dal valore di dir che faccia una somma (ADC) o una sottrazione (SBC). c'è un'altra via, ed è quella di eseguire la somma di un numero negativo. Operazione che con carta e penna ci riucirebbe semplicissima, basterebbe mettere un segno '-' davanti al secondo operando, ma nel mondo informatico la cosa non è così intuitiva.
Per esempio se doessimo eseguire l'operazione 10-7, o nel nostro caso 10+(-7), dovremmo pensare ai byte (o word o a qualunque serie di bit) non più come ad un contenitore di numeri da $00 a $FF (0-255), ma bensì come se il bit più a sinistra (il più significativo) fosse un indicatore di segno, tale bit sarà a 1 per rappresentare i numeri negativi e 0 per quelli positivi. Naturalmente, essendoci mangiati un bit, la capacità rappresentativa del nostro byte sarà sempre di 256 valori ma rappresenteranno i valori da -128 a +127.

Bytes negativi

Per passare da un numero positivo (nel nostro esempio 7) al suo opposto dobbiamo imparare a fare il complemento a due di quel numero:

Come prima cosa scriviamo il nostro numero in forma binaria, quindi invertiamo tutti i bit, gli zeri diventano "uni" mentre i vari uno diventano zero, e fin qui abbiamo il complemento a uno, dopodiché prendiamo il risultato dell'inversione e sommiamo 1, quello che abbiamo ottenuto è il complemento a due. A questo punto non ci resta che sommare il complemento a due per ottenere come risultato la differenza desiderata. Nel nostro caso, dovendo anche sommare e sottrarre ad indirizzi di memoria (come vedremo più avanti), dobbiamo lavorare con le word, questo non cambia nulla a livello di operatività, semplicemente dovremo eseguire la complementazione fino a "riempire" tutti e 16 i bit a nostra disposizione. Per esempio la rappresentazione di -1 in una word è una fila di 16 uno ($FFFF).

```
Complemento a due al fine
di eseguire l'operazione:
10+(-7) = 3

 00000111   (7)

 11111000 + (Inversione)
 00000001 =
 --------
 11111001   (-7)

11111       (Riporti)
 00001010 + (10)
 11111001 = (-7)
 --------
 00000011   (3)
```

Nell'esempio di codice sotto riportato, che viene eseguito nel momento in cui si rende necessario un cambio di direzione (quando l'unica alternativa per il nostro amico terricolo è tornare sui propri passi), viene eseguita l'operazione complementazione su Dir in modo che da $01 passi a $FF (-1) e viceversa:

```
ChangeDir                   ; Nessuna cella libera nell'intorno della testa quindi torno indietro
       lda Dir              ; carico il valore della direzione in A ($01 Avanti $FF indietro)
                            ; mi appresto a fare il complemento a due di A
       eor #$FF             ; 1) exclusive Or su A con $FF
                            ; inverto il valore di tutti i bit (complemento a 1)
       clc                  ; pulisco il carry
       adc #$01             ; 2) sommo uno ad A (complemento a 2)
       sta Dir              ; copio A in Dir
```

In tal modo, quando dovremmo "avanzare", basterà semplicemente sommare Dir, come mostrato in questo spaccato di codice preso dalla routine WalkWorm:

```
WalkWom
       ...
       lda Sector           ; Riprendo l'indice al settore attuale
       clc                  ; mi preparo ad incrementarlo tramite ADC quindi pulisco il carry
       adc Dir              ; sommo Dir all'indice Sector A=A+Dir+carry
       ...
```

Tastare il terreno

Come abbiamo deciso inizialmente, ogni NumSec cicli il vermiciattolo dovrà esguire un reale spostamento sullo schermo, ma potrà occupare una casella che sia vuota. Quindi testerà nell'ordine, prendendo come riferimento la testa, la casella di fronte (rispetto al verso), quella alla sua sinistra e poi quella alla sua destra, la prima che troverà libera decreterà il suo verso di marcia, nel caso fossero tutte occupate l'unica alternativa sarà cambiare direzione e tornare sui propri passi.
Come sappiamo la memoria video (cioè quella che sfruttiamo in questa occasione) parte dall'indirizzo $0400 e termina in $07E7 e può essere immaginata come una sequenza, senza soluzione di continuità, di 40 bytes che si ripete per 25 volte, ogni sequenza rappresenta una riga dello schermo. Questo ci permette di mettere in "relazione" due celle video solamente conoscendone la distanza. Per esempio, sapendo l'indirizzo di memoria video M, il cui contenuto è visualizzato alla riga 15, colonna 10, come potremmo calcolare quale sarà la cella video esattamente sopra (riga 15 colonna 10) a M ? basterà eseguire l'operazione M-40, se si vuole puntare quella sottostante basterà sommare 40.
Penso che sia chiaro dove si voglia andare a parare, per testare se le celle intorno alla testa sono vuote (tranne quella dove sicuramente c'è il collo !?) basterà riservarsi 4 word in cui saranno memorizzati gli offset da usare per raggiungere l'"intorno" della testa, naturalmente, visto che abbiamo imparato ad usare il complemento a 2 lo sfrutteremo anche in questa occasione, infatti i valori delle words della cella che precede e di quella che si trova a sud della testa, sono già espressi con il complemento a due in modo che alla bisogna si debbano eseguire solamente delle somme:

```
                ;  EST    SUD  OVEST  NORD
                ;   +1    +40     -1    -40
Intorno word $0001, $0028, $FFFF, $FFD8
```

La label Intorno identifica l'inizio dei otto bytes organizzati come quattro words per gestire il controllo del terreno, sebbene ogni word sia espressa come MSB/LSB nel codice assembly, va tenuto presente che in realtà a livello di memoria saranno scritte con i bytes invertiti cioè LSB/MSB, in virtù del fatto che abbiamo indicato in assembly che sono words.

```
ChkContinue
       sta VerseT                ; metto in VerseT (Verso da testare) il valore di A (0-3)
       asl a                     ; A=A*2 per puntare correttamente poi all'LSB relativo al verso
                                 ; della tabella Intorno
       tax                       ; X=A imposto il registro indice X col valore di A
       lda Sector                ; carico in A l'indice al settore attuale
                                 ; che arrivati qui è la testa dell'anellide
       asl a                     ; A=A*2 per raggiungere l'LSB puntato da sector
                                 ; nella tabella Body
       tay                       ; Y=A imposto il registro indice Y col valore di A
       lda Body,Y                ; Carico in A l'LSB dell'indirizzo video di Sector
       clc                       ; solita pulizia del carry
       adc Intorno,X             ; sommo l'offset (LSB) da utilizzare per testare la cella posizionata
                                 ; nella direzione di VerseT
       sta BodyPtr               ; metto A nell'LSB di BodyPtr
       iny                       ; incremento indice Y per prelevare l'MSB
                                 ; della cella video di Sector (la testa)
       inx                       ; Incremento X, indice per l'MSB di intorno
       lda Body,y                ; Carico in A l'MSB dell'indirizzo video di Sector
       adc Intorno,X             ; Sommo l'MSB dell'offset opportuno per puntare alla cella dell'intorno
       sta BodyPtr+1             ; metto A nell'MSB di BodyPtr (Bodyptr+1)
       jsr CheckOk               ; routine di controllo della cella memorizzata
                                 ; in BodyPtr LSB/MSB
                                 ; l'ultima istruzione prima di rts di CheckOk
                                 ; è una ldx $00 o ldx $FF
                                 ; nell'ordine vuol dire cella libero (quindi posso
                                 ; farci camminare l'anellide) cella occupata
       beq EndLFNP               ; ... se X==0 allora ho terminato la ricerca nell'intorno
       dec Count                 ; ... altrimenti provo con un altra cella dell'intorno
       bpl LoopCheck             ; Se Count > $FF abbiamo ancora celle nell'intorno non testate
```

Nel pezzo di codice estrapolato e riportato sopra, vi è l'esempio di come ricalcolare una cella nell'intorno della testa (in particolare in questo pezzo di codice l'indice della testa corrisponde a Sector).

In particolare si noti come viene raggiunto l'indice all'LSB della cella, contenuta in quello che si potrebbe assimilare ad un array di words. Avendo l'indice Sector che varia tra 0 e 3 non possiamo usarlo direttamente come offset di Body, in quanto la nostra esigenza è saltare 2 bytes alla volta, per fare ciò moltiplichiamo l'indice per due con l'istruzione ASL (Arithmetic Shift Left) la quale non fa altro che spostare tutti i bit (con 'ASL A' lo si fa nell'accumulatore) di una posizione a sinistra (il bit più a sinistra va a finire nel carry) mettendo uno 0 nel bit più a destra. Questo equivale a moltiplicare il numero rappresentato per due, questa è l'unica forma di moltiplicazione immediata che può essere effettuata senza utilizzare un'apposita routine, naturalmente l'istruzione può essere reiterata al fine di moltiplicare per le altre potenze di 2.

```
Moltiplicazione x 2 con ASL A

A= $14 00010100 (20)

C=0 ← 00101000 ← 0 $28 (40)
```

Tornando al codice notiamo che tramite i registri-indice X e Y riusciamo agevolmente a calcolare la cella dell'intorno sommando le words di Body (indirizzi di memoria video) con le words di Intorno (valori degli offset) senza preoccuparci di dover sommare o sottrarre, avendo nelle words di intorno la rappresentazione complemento a 2 dei valori negativi.

Verso la fine del pezzo di codice incontriamo l'istruzione DEC (DECrement memory) la quale decrementa di uno la memoria puntata dall'argomento, l'istruzione successiva BPL (Brach if Plus) non fa altro che controllare il flag S (Segno) nello status register, il quale è stato valorizzato dall'istruzione precedente con il valore del bit più a sinistra del valore di Count dopo che è stato decrementato. Quindi, nello specifico Count è un contatore con valori maggiori o uguali a zero e quindi la BPL farà eseguire un salto a LoopCheck, ma nel momento in cui Count ha valore 0 e viene decrementato ulteruormente, il suo valore diviene $FF (che in complemento a due significa -1), settando il flag di segno e facendo in modo che la BPL "fallisca", uscendo così dal Loop.
L'istruzione BPL viene più spesso usata in seguito ad una istruzione di comparazione (CMP, CPX e CPY) per confrontare due valori:

```
LDA NumeroA   ; Carica l'accumulatore con il valore di NumeroA
CMP NumeroB   ; Confronta NumeroA con NumeroB
BPL PiuGrande ; NumeroA >= NumeroB ? Si! Vai alla  PiuGrande
...           ; NO!? (NumeroA < NumeroB) continua l'esecuzione
```

Come abbiamo già visto le comparazioni eseguono in realtà una differenza interna (NumeroA-NumeroB) per settare opportunamente i flag del registro di stato, in effetti è logico che il flag N si comporti di conseguenza. L'istruzione che ragiona esattamente al contrario di BPL è BMI (Brach if Minus) la quale esegue il salto d'esecuzione se il flag N risulta settato. Quindi sostituendo BMI a BPL nell'esempio precedente, il "branch" verrebbe percorso se NumeroA < NumeroB.

Controllare i limiti

Tornando al programma, dobbiamo sì evitare di sovrapporci agli ostacoli (i vari caratteri sullo schermo), ma dobbiamo evitare anche di finire in una zona di memoria al di fuori di quella video ($0400-$07E7). Come avrete già intuito per leggere e scrivere sulla cella di memoria video utilizziamo l'indirizzamento indiretto indicizzato sfruttando pesantemente le locazioni $FB,$FC della pagina zero. Per comodità, all'inizio del listato è stata definita una "costante" BodyPtr da usare al posto di $FB in modo da avere migliore leggibilità del codice:

```
BodyPtr         = $FB           ; indirizzo ZP (Zero Page)
```

Supponendo di aver scelto uno dei tre versi da controllare per eseguire il prossimo passo, verrà caricata a partire da BodyPtr l'indirizzo della cella video scelta da testare, quindi verrà richiamata la routine che sotto riporto:

```
CheckOk                         ; arrivati a questo punto mi aspetto che in BodyPtr e BodyPtr+1
                                ; ci siano LSB e MSB della cella video da controllare
        lda BodyPtr+1           ; Controllo che l'indirizzo non sia inferiore a $0400
        cmp #$04                ; qundi è sufficiente controllare l'MSB
        bcc NotOk               ; MSB < $04, Si posizione al di fuori dello schermo
        beq CheckFree           ; MSB = $04, Si posizione sicuramente
                                ; all'interno dello schermo ($0400 - $07E7)
CheckMaxBound                   ; MSB > $04
        cmp #$07                ; Compara MSB con $07
        beq CheckMaxBoundLSB    ; MSB==$07 ? Controlla LSB (deve essere < $E8)
        bcs NotOk               ; MSB >= $07 ? (MSB!=$07 test precedente,
                                ; il test è in realtà MSB > $07) sono fuori dallo schermo
        jmp CheckFree           ; MSB < $07 => controlla se libera
CheckMaxBoundLSB
        lda BodyPtr             ; A=LSB(Bodyptr)
        cmp #$E8                ; test per minore
        bcc CheckFree           ; A < $E8 ? controlla se cella libera

NotOk                           ; La cella è al di fuori dello schermo
        ldx #$FF                ; lo segnalo al chiamante ponendo X=$FF
        rts

CheckFree
        ldy #$00                ; x= flag for ok 0 =0k FF=!ok
        lda (BodyPtr),y         ; carico in A tramite Indiretto indicizzato ZP
        cmp #$20                ; Compara A con valore spazio
        bne NotOk               ; se non è uno spazio segnala che non va bene
EndOfCheck                      ; altrimenti ... cella video "camminabile"
        ldx #$00                ; lo segnalo al chiamante ponendo X=$00
        rts
```

Nelle prime 2 righe di codice viene controllato che l'MSB dell'indirizzo di memoria non sia inferiore a $04. Questo controllo viene eseguito tramite l'istruzione BCC (Branch on Carry Clear), anchessa può essere utilizzata similmente a BPL per confrontare due numeri, con la differenza che BCC controlla il flag di Carry. BCC nel ostro caso specifico eseguirà Iil salto a NotOk se l'MSB della cella video risultasse strettamente inferiore a $04, ciò implicherebbe che la cella si trova prima della prima riga nella memoria video, al di fuori della nostra vista. Oltre a BCC c'è l'istruzione BCS (Branch on Carry Set) che eseguirà il salto se il carry risulta settato, e quindi si comporterà all'opposto rispetto a BCC.
Nella riga successiva con l'istruzione BEQ viene richiesto un altro salto condizionato nel caso in cui l'MSB sia esattamente $04, se così fosse infatti siamo sicuramente all'interno del video, ora ci basterà controllare se il valore della cella è uno spazio (CheckFree). Le richieste di salto condizionato possono sempre essere eseguite in cascata, in quanto non vanno a modificare i flag di status e quindi non interferiranno tra di loro.
Potrete tranquillamente analizzare da soli la restante parte di codice, essendo abbastanza simile come logica a quella utilizzata nelle prime righe. L'unica cosa da specificare è il fatto che volutamente questa routine ha due punti di uscita, NotOk e EndOfCheck, entrambi caricano in X un valore ($00 e $FF), questo caricamento prima di uscire va a modificare i flag di stato in modo che il chiamante li possa interpretare a seconda dell'esito del controllo, il fatto di usare X non ha nulla di particolare, solo una scelta durante lo sviluppo.

Interazione con l'ambiente

Dopo aver sviluppato le varie parti di avanzamento, controllo e deviazione del vermiciattolo, anche stavolta incastreremo in tutto in mezzo alle chiamate che vengono effettuate 60 volte al secondo dal Commodore 64, in modo tale da poter interagire con l'anellide. La tecnica è semplice ed è illustrata sia nell'articolo precedente che commentata nel listato completo, ma ci sono un paio di cosette che vanno specificate.

Quando illustrai (nel precedente articolo) come interporre la nostra routine tra le chiamate base dell'interrupt, indicai, di salvare nello stack lo stato dei registri, per poi ripristinarli prima della chiamata per la continuazione dell'interrupt. Il mio è stato un eccesso di zelo, nel senso che il salvataggio e il ripristino dello stato dei registri male non fa (se non rallentare di un pochettino il tutto), la cosa comunque risulta inutile in quanto tutto il necessario per non fare danni, viene già fatto dal sistema stesso. Questo è il motivo per cui troverete la routine "incastonata" più asciutta, avendola privata dei vari push e pop sullo stack.
Inoltre quando utilizziamo questa tecnica, dovremmo fare attenzione affinché la nostra routine stia al di sotto del sassantesimo di secondo, altrimenti ci ritroveremmo con una routine di interrupt in esecuzione ed un'altra che si sovrappone, causando comportamenti imprevedibili.

Miglioramenti

Il programma è lungi dall'essere perfetto, infatti non gestisce lo scroll del video e a volte il lombrico, quando torna nei suoi passi tende ad eseguire una deviazione anche senza aver incontrato effettivamente un ostacolo, inoltre il nostro animaletto soffre sicuramente nel poter formare solamente angoli retti, secondo me sarebbe felicissimo di potersi diagonalizzare e non spezzarsi a novanta gradi il più delle volte. Inoltre la colonna sonora (un ping ad ogni ostacolo) non è affatto delle migliori. Lascio agli uomini di buona volontà l'implementazione di miglioramenti e correzioni, la ricompensa sarà alta, perché interiore, ma nulla vieta la sua pubblicazione seguita sicuramente da pubblico elogio.

Emanuele Bonin

Emanuele.Bonin71@Gmail.com