Studente: **Daniel Zotti** Data Esame: **01/01/2011**

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Corso di laurea triennale in Ingegneria informatica

PROGETTO:

SPACE - CAR

Esame di Calcolatori Elettronici II

(a.a. 2010/2011)

Descrizione generale

- Il programma è un applicazione di tipo videoludico, implementato con il linguaggio Assembly, che si ispira ad un gioco con cui giocavo da piccolo negli anni '80, che ha come protagonista una navicella spaziale.
- Lo scopo del gioco è fare più punti possibili, raccogliendo le monete che si presentano durante il tragitto, schivando le numerose asteroidi sul percorso.
- Si possono raccogliere un massimo di 5 vite (cuoricini) e si hanno a disposizione 8 velocità diverse (livelli)
- Una volta finite le vite a disposizione, il gioco termina e viene stampato a schermo il punteggio finale

Strumenti utilizzati

- Ho sviluppato il programma sul sistema operativo **MS-DOS 6.22**, fatto funzionare in macchina virtuale (VirtualBox) sotto Ubuntu **Linux**.
- Il codice è stato scritto inizialmente su editor di testo **Geany** ma, viste le innumerevoli prove, alla fine ho utilizzato **EDIT** del DOS.
- Per la creazione dell'eseguibile sono stati usati l'assemblatore **MASM5** della Microsoft (che ha il compito di creare il codice oggetto OBJ a partire dal codice sorgente ASM) e il programma di linker **LINK2** (che trasforma l'OBJ in file eseguibile EXE)
- Ho fatto alcuni test su **Windows XP** e il programma funziona correttamente, mentre su Vista e Seven dà qualche problema e non riesce a partire.

Implementazione: I Segmenti

- Per questo programma ho utilizzato 3 segmenti tra i 4 disponibili:
  - Pila: segmento per la pila di 512byte dove si salvano provvisoriamente dei dati. [SS]
  - Dati: segmento dove salvo le variabili utilizzate dal programma. [DS]
  - _prog ('CODE'): segmento dove scrivo il codice del programma (con Macro, ”EQU” e Procedure). [CS]

- NB: Con la direttiva ASSUME forzo l'uso dei segmenti corretti per tutti i simboli del segmento

Segmento PILA

- Il segmento Pila è di tipo **STACK** e salva all'inizio dell'esecuzione del programma una porzione di memoria RAM (in questo caso 512byte) che serve appunto al programma per salvare provvisoriamente dei dati.
- **PARA** indica che la pila sarà posizionata nel primo indirizzo libero della RAM multiplo di 2.
- (In altri casi: BYTE nel primo indirizzo libero, PARA nel primo indirizzo libero multiplo di 16, PAGE nel primo indirizzo libero multiplo di 256)

Segmento DATI

- Il segmento Dati contiene tutte le variabili utilizzate dal programma.
- In particolare avremo le stringhe con i messaggi da stampare a schermo e le variabili fondamentali come:
  - **life** - numero di vite (tipo DB – define byte)
  - **level** - velocità di gioco (DB)
  - **score** – punteggio (DW – define word)
  - **tLevel** - cicli totali (DB)
  - **maxScore** – punteggio massimo raggiungibile (DW)
- In più le variabili utilizzate dalla procedura random (spiegate in seguito)
- **PARA** indica anche in questo caso che il segmento sarà posizionato nel primo indirizzo libero della RAM multiplo di 2.

Segmento _prog

- Il segmento _prog contiene le macro (**MACRO**), le procedure (**PROC**), le costanti (**EQU**) e tutto il codice del programma.
- 'CODE' è il nome del segmento nel caso in cui si dovessero collegare segmento diversi (non è questo il caso)
- **PARA** indica che la pila sarà posizionata nel primo indirizzo libero della RAM multiplo di 2.
- Un esempio di costante: ”kESC EQU 1BH” rende più leggibile il codice per quanto riguarda il tasto Escape

Funzionamento generale (I)

- Il gioco si svolge in una cornice di 22 righe per 30 colonne, create con i caratteri ASCII estesi per avere una cornice continua. Fuori dalla cornice sono segnate le vite, il livello, lo score e vengono stampati i vari messaggi ”interattivi”.
- Il gioco inizia e nella prima riga in alto viene stampato un carattere in una colonna random. Possono venir stampati 3 caratteri diversi, con diverse probabilità:
  - **Moneta** (25%)
  - **Cuoricino/Vita** (5%)
  - **Ostacolo/Asterodie** (70%)
- Ad ogni ”ciclo”, tutta l'area giocabile viene spostata di una riga in basso e viene stampato un nuovo carattere con lo stesso metodo spiegato in precedenza.
  Si crea così un effetto che riproduce il movimento verso l'alto della navicella che viene posizionata nell'ultima riga in basso (e viene ristampata ad ogni ciclo nella stessa riga)

Funzionamento generale (II)

- Dopo qualche ciclo gli asteroidi/monete/vite riempiono tutte le righe dello schermo e il gioco vero e proprio ha inizio.
- La navicella con l'utilizzo dei **tasti freccia** si può muovere a **destra e a sinistra** (nei limiti della cornice) per schivare le asteroidi e prendere gli oggetti utili
- Se la navicella:
  - Prende una **moneta** → al punteggio viene sommato un numero pari al valore del livello
  - Prende una **vita** → aumentano di uno i cuoricini, fino ad un massimo di 5 vite
  - Colpisce un **asteroide** → il round finisce, e se ha accumulato altre vite, può continuare con il round successivo finchè non finisce tutti i cuoricini
- Se la navicella non ha più vite quando colpisce l'asteroide, il gioco finisce e viene stampato a schermo il punteggio finale

Funzionamento generale (III)

- Per aumentare/diminuire la velocità del gioco si possono utilizzare i **tasti freccia su/giù**. Più il gioco è veloce e più valgono le monete accumulate (es: livello 8 → la moneta incrementa lo score di 8 punti, livello 1 → la moneta incrementa la score di 1 punto)
- NB: La scelta di aumentare/diminuire la velocità del gioco a piacimento non è sicuramente un'ottima scelta per un gioco dove bisogna battere un record (in quando si rischia di imbrogliare), ma è più utile per la dimostrazione pratica del progetto d'esame. Per un gioco ”vero” sarebbe stato meglio (e più facile) poter aumentare di livello ogni tot cicli (o ogni volta che si supera un certo punteggio), ma per provarlo durante l'esame sarebbe stato troppo lungo e noioso. (i primi livelli sono molto lenti e quindi noiosi ;-))

Struttura del codice

- Per comodità e chiarezza si sono usate le Macro e le Procedure durante la stesura del codice.
- Nello specifico, dato che si è scritto l'intero programma in un unico file (car.asm), per rendere la lettura e la ricerca di eventuali errori più semplice, si è organizzato il tutto nel modo seguente:
  - **Costanti**
  - **Macro**
  - **Corpo** del programma
  - **Procedure**

Struttura del codice (II)

NB: Ora verrà spiegato brevemente e in parole semplici ***COSA*** fanno il **programma**, le **macro** e le **procedure**

Per capire bene ***COME*** vengono fatte tutte le operazioni, daremo un'occhiata direttamente al **codice**

Corpo del programma (I)

- Per prima cosa è utile sapere che ho dedicato certi registri a uno specifico utilizzo:
  - DX → Posizione della navicella
  - BX → Tipo di carattere colpito (in BL → x=asteroide, v=vita, m=moneta)
  - CX → Contatore dei cicli (CH=cicli massimo, CL=ciclo istantaneo)
    in pratica quando CH=CL è finito il ciclo e si va in giù di una riga
- Il programma inizia con la procedura cls (Clear Screeen) che pulisce lo schermo e lo preparara alla stampa delle istruzioni
- Posiziona il cursore a 1,0 (riga, colonna) con la macro setCur riga,colonna
- Stampa le istruzioni con la macro stpMex mex
- Nasconde il cursore con la procedura outCur
- Aspetta un tasto qualsiasi (waitKey) per iniziare finalmente il gioco
- Start: è l'etichetta in cui tornerà il programma ogni volta in cui la navicella colpirà un asteroide (e avrà ancora vite a disposizione)

Corpo del programma (II)

- Pulisce nuovamente lo schermo con cls e con la procedura wBordo disegna la cornice di 22x30
- Posiziona il cursore con setCur 4,40 a destra della cornice, stampa la scritta ”LIFE: ” con stpMex e disegna il numero di cuoricini rossi che trova nella variabile life con la macro stpChrC char,tipo,colore
- Posiziona il cursore con setCur 6,40 a destra della cornice, stampa la scritta ”LEVEL: ” con stpMex e disegna il numero di pallini blu che trova nella variabile level con la macro stpChrC char,tipo,colore
- Posiziona il cursore con setCur 8,40 e stampa la scritta ”SCORE: ” con stpMex e mette in AX il punteggio che trova nella variabile score per poi stamparlo in decimale grazie alla procedura word2ascii.
- Metto in DX la posizione della navicella (in basso in centro) e chiamo la procedura setCar per posizionarla.
- Attende il tasto INVIO per iniziare il gioco.

Corpo del programma (III)

- Una volta premuto INVIO inizia il gioco
- Inizializzo prima del ciclo il registro BX a 0000H per dire che la navicella non ha ancora sbattuto contro nessuna asteroide o raccolto alcuna moneta/vita
- Inizializzo CX con il contatore:
  - MOV CH,tLevel → ciclo massimo (da raggiungere)
  - MOV CL,0 ciclo → intermedio iniziale
- Controllo se CL è maggiore di CH, questo può accadere solo se durante il ciclo ho diminuito il livello mentre CL era quasi alla fine del ciclo:
  ES: CH=10, CL=9, diminuisco il livello di 2 → ora CH=8,CL=9
  Se non faccio un controllo il programma si blocca!
  (Ovviamente al primo ciclo non servirà)

Corpo del programma (IV)

- Se CL è ancora minore di CH continuo col ciclo
  - Controllo se nel ciclo precedente ho preso una moneta (BL='m'), allora vado all'etichetta addMon, aumento il valore della variabile score e aggiorno la stampa del risultato. Vado all'etichetta aspKey
  - Controllo se nel ciclo precedente ho preso una vita (BL='v'), allora vado all'etichetta addVita, aumento il valore della variabile life e aggiorno la stampa dei cuoricini (controllando anche se sono già 5). Vado all'etichetta aspKey
- AspKey: controllo se ho preso un asteroide (BL='x'), se è così vado a Dead (in realtà vado a Dead2 che rimanda a Dead in quanto il salto è troppo lungo), altrimenti posiziono la navicella e controllo con setCar se in questo ciclo ho sbattuto da qualche parte (NB: setCar in uscita riempie il registro BL con dei caratteri in base a cosa ha colpito e nel ciclo successivo si effettua il controllo)
- Applico il ritardo di 1 tick (1/18 di secondo), incremento il contatore intermedio e controllo se sono a fine ciclo (se sono a fine ciclo vado a Continue)

Corpo del programma (V)

- Se non sono a fine ciclo chiamo la procedura pressKey che controlla se è viene premuto un pulsante. Se non viene premuto vado a Continue, altrimenti con la procedura waitKey capisco che tasto è stato premuto e agisco di conseguenza:
- ESC → Vado all'etichetta Esci:
  P o p → Vado all'etichetta I_Pause: (Inizio del ciclo di pausa)
  freccia DX → Vado all'etichetta Destra:
  freccia SX → Vado all'etichetta Sinistra:
  freccia SU → Vado all'etichetta Su:
  freccia GIU → Vado all'etichetta Giu:
  (in realtà vado a Destra2 che rimanda a Destra, come spiegato in precedenza per i salti CONDIZIONATI troppo lunghi!)
- I_Pause: salvo i registri usati nella pausa, scrivo nella zona messaggi ”PAUSE” e inizio il ciclo di attesa della pausa con la procedura Pause: waitKey.
- Se premo ESC (vado a Esci) e esco dal gioco, altrimenti attendo un tasto P o p per andare a F_Pause cancellare la scritta ”PAUSE”, ripristinare i registri, uscire dalla pausa, e andare all'etichetta AspKey

Corpo del programma (VI)

- Destra: Controllo se ho raggiunto il limite destro della cornice (limDX) se è così non faccio niente e torno a AspKey, altrimenti incremento di uno la posizione della navicella (INC DX) e nella zona a sinistra in cui non c'è più la navicella disegno uno spazio. Torno ad AspKey
- Sinistra: Procedura simmetrica a Destra:
- Su: Controllo se ho ragiunto il livello massimo, se è così non faccio niente e torno a AspKey, altrimenti incremento il livello (level), abbasso di due la durata del ciclo (tLevel), stampo il numero dei pallini del livello, scrivo ”+1 LEVEL” nella zona messaggi e vado a AspKey
- Giu: Procedura simmetrica a Su:
- Dead: il programma decrementa life, se è 0 allora vado a Lose, altrimenti stampo dei messaggi a schermo e, dopo aver premuto INVIO, ricomincio il round da Start:

Corpo del programma (VII)

- Prima o poi arrivo all'etichetta Continue: dove chiamo la procedura goGIU (che abbassa l'area incorniciata di una riga), scelgo un numero random da 0 a 99 e in base al numero stampo, con diverse probabilità, una moneta, una vita o un asteroide.
  (Per la dimostrazione del progetto ho scelto una probabilià del 5% per la vita, ma secondo me, per rendere più complicato il gioco, sarebbe meglio mettere una probabilità dell'1% - in pratica 1 ogni 100 righe)
- Lose: stampa nella zona messaggi ”GAME OVER”, stampa il punteggio finale, aspetta un tasto e va a Esci:
- Win: stampa nella zona messaggi ”Hai raggiunto il puteggio finale”, stampa il punteggio finale, aspetta un tasto e va a Esci:
  NB: Prima di arrivare a questa etichetta si confronta lo score con il valore maxScore, se è maggiore o uguale si va a Win:
- Esci: stampa il messaggio di uscita e aspetta un invio per fare il cls e tornare a DOS

MACRO (I)

- setCur posiziona il cursore in DH=riga, DL=colonna chiamando la procedura posCur
- stpChrT stampa un carattere in modalità TTY (aggiornamento automatico cursore) settando in entrata AL=carattere e chiamando la procedura writeTTY
- stpChrC stampa tanti caratteri a colori senza aggiornare il cursore. Setta in entrata AL=carattere, CX=numero di volte da stampare, BL=colore e chiama la procedura writeCOL
- stpChrBN stampa un carattere in Bianco e Nero senza aggiornare il cursore. Setta in entrata AL=carattere e chiama la procedura writeBN
- Random setta in AX=numero massimo del random, e chiama la procedura rand che in AX metterà in uscita il numero random compreso tra 0 e AX in entrata

MACRO (II)

- stpMex stampa un messaggio salvato in memoria (Segmento Dati) a schermo. Setta in entrata DS=Segmento Dati (passando per AX), DX=Offset della stringa che si trova nel segmento Dati
- Ritardo imposta un multiplo del ritardo che crea la procedura delay. Setta in entrata CX=tick (numero per cui moltiplicare il ritardo di delay)
- NB: E' comodo utilizzare le **MACRO** al posto della **PROCEDURE** in quanto non serve scrivere CALL per richiamarle e si possono passare dei parametri in modo semplice e intuitivo.
  Non bisogna però abusare di questo **MACRO** in quanto se vengono chiamate n volte, verranno copiate per n volte nel codice, mentre le procedure ogni volta in cui vengono richiamate, il programma punta direttamente a loro senza doverle riscrivere.

PROCEDURE (I)

- wBordo si occupa di stampare la cornice dell'area ”giocabile”
  In pratica posiziona il cursore in 0,0 e stampa la il carattere ”angolo alto sinistro” (0DAH), tramite un ciclo ripetuto 28 volte stampa il resto della riga con caratteri linea (0C4H) e stampa il carattere ”angolo alto destro” (0BFH)
  Si sposta nella seconda riga, stampa il carattere | (OB3H) su tutte le 20 righe della colonna del bordo sinistro da (1,0) → (20,0)
  Fa lo stesso sul bordo destro da (1,29) → (20,29)
  Si sposta alla 22esima riga (l'utlima) e agisce in modo speculare alla prima riga
- wLife si occupa di stampare un cuoricino nella prima riga, in una posizione random (tra 1 e 28) della colonna.
- wMon e wOst sono simili a wLife
- goGIU grazie alla funzione 07H dell'INT10H sposta in giù un area rettangolare dello schermo che va dall'angolo in alto a sinistra (CH,CL) all'angolo in basso a destra (DH,DL). In BH si mette il colore delle righe lasciate vuote (in questo caso bianco su nero - 07H)

PROCEDURE (II)

- writeTTY utilizza la funzione 0EH dell'INT10H che stampa un carattere passato in AL di colore BL nella pagina RAM BH, aggiornando il cursore.
- writeCOL e writeBN sono procedure simili a writeTTY che utilizzano rispettivamente la funzione 09H e 0AH dell'INT10H, ma NON aggiornano il cursore
- scrivi schiama direttamente la funzione 09H dell'INT10H
- clock chiama la procedura 00H dell'INT1AH che restituisce in DX=parte bassa del clock (ore e minuti) e in CX=parte alta del clock (secondi e centesimi)
- waitKey chiama la procedura 00H dell'INT16H che aspetta la pressione di un tasto (blocca il programma finchè non premi un tasto) restituisce in AL=codice carattere ascii, AH=codice di scansione
- pressKey controlla se viene premuto un tasto. Chiama la procedura 01H dell'INT16H che modifica lo ZERO Flag a 1 se viene premuto un tasto

PROCEDURE (III)

- posCur utilizza la funzione 02H dell'INT10H che posiziona il cursore in DH,DL (riga,colonna) nella pagina video BH.
- STAasci aggiunge 30 al registro AL che contiene un numero da 0 a 9 e stampa il carattere ascii/numerico chiamando la macro stpChrT
- readCur chiama la funzione 08H dell'INT10H che in uscita mette in AH e AL rispettivamente il colore e il codice del carattere ascii puntato dal cursore
- outCur nasconde il cursore chiamando la funzione 01H dell'INT10H che ridimensiona il cursore. In CL si mette il valore della linea iniziale e in CH di quella finale se il bit5 di CX è 1, il cursore sparisce
- cls pulisce lo schermo chiamando la sottofunzione 03H della funzione 00H dell'INT10H che imposta la modalità schermo a 80x24 (e che pulisce anche lo schermo)
- tornaDOS chiama la funziona 4CH dell'INT21H per tornare al DOS

PROCEDURE (IV)

- setCar disegna la navicella sullo schermo e chiama la procedura checkCar per controllare se ha preso una moneta/vita o se ha sbattuto contro un asteroide.
  Inizializza CX a 0000H (variabile in cui verrà segnalato se e che cosa è stato colpito/preso). NB: In DH c'è la posizione della navicella e in BH salverà il valore dell'oggetto colpito.
- checkCar chiama readCur e controlla (in base al colore) se in quel punto c'è una moneta, una vita o un asteroide
- rand procedura che genera un numero Random tra 0 e AX (parametro passato alla chiamata)
- delay procedura che chiama la procedura clock e in base al time del sistema crea un ritardo.
- word2dec procedura che trasforma una word in caratteri ASCII per stampare il valore in decimale.

Il CODICE

E' arrivato il momento di veder in dettaglio il codice...