Non nego che potrebbe sembrare banale parlare estesamente anche del reset di un computer: alla quale operazione, sotto certi aspetti, è da sempre legato un sentimento di rassegnazione tipico di chi si ritrova il computer irrimediabilmente paralizzato - pensiero tipico: «Si è bloccato ... si, si è bloccato ... mannaggia, non ho salvato il programma ... beh, che ci vuoi fare ... (e il malcapitato diede il terribile Ctrl-Amiga-Amiga)». Questo fin dai tempi della SYS64738 del buon 64. Ma, nell'Amiga, tutto è interessante: vedremo in questa puntata come è possibile modificare il processo standard di inizializzazione (nonché altre cosucce carine riguardo ExecBase) a nostro vantaggio; mi raccomando, non fatevi venire in mente di scrivere un virus ...

ExecBase e Reset

di Maurizio Mangrella - Eboli (SA)

Il Reset

Anzitutto bisogna distinguere tra il reset iniziale (quello che avviene all'accensione della macchina) e quello causato dall'utente (con la pressione contemporanea dei tasti Control, Left Amiga e Right Amiga).

Durante il reset iniziale, il S.O. (o, almeno, quella parte che è residente: vedi dopo) controlla l'efficienza dei chip custom (proprio non so in base a quale principio): se uno dei chip non funziona a dovere lo schermo diventa blu (Denise permettendo: potrebbe essere proprio lì il guasto!) e l'Amiga va in stato di Halt indefinito. Poi (negli Amiga 1000) il BIOS minimale inserito su una piccola ROM carica il KickStart: d'ora in poi, riferendomi alla ROM, intenderò, per i 500 e i 2000, la ROM residente del KickStart, per i 1000 il banco RAM di 256K in cui viene caricato il S.O., e che viene protetto in scrittura dopo il caricamento.

Successivamente (qui comincia la procedura del reset da tastiera) il S.O. diagnostica l'efficienza della ROM (schermo grigio scuro: tutto OK) e della RAM (schermo grigio chiaro: tutto OK), quindi compare lo schermo bianco (e la petulante mano del WorkBench se non era stato ancora inserito un dischetto bootabile) e si procede al boot.

In un computer basato su 68000 e famiglia, al momento del Reset, bisogna fornire, nella locazione 4, una longwork contenente l'indirizzo iniziale della routine di Reset, e, nella locazione 0, il Supervisor Stack Pointer iniziale per la routine di Reset (che viene automaticamente eseguita in Modo Supervisore); tutto ciò è automaticamente gestito da Exec. Faccio notare che, se durante il processo di inizializzazione del Reset svolto autonomamente dal micro, si verifica per qualunque motivo una Exception, il 68000 va in stato di Halt indefinito: in pratica potrà risvegliarsi solo al prossimo Reset. Se, invece, (prerogativa del KickStart) si verifica un'Exception durante l'esecuzione della routine di Reset vera e propria, lo schermo diventa giallo e il led di Power On lampeggia come in una Guru Meditation: potete a questo punto scegliere tra l'ippica e il whisky...

Motivi di interesse

Vi chiederete a questo punto (oltre al codice di colori, che per fortuna non ho avuto modo di sperimentare di persona!) cos'altro ci sia di interessante nel Reset di Amiga. Beh, vi dirò che l'Amiga non dimentica tutto durante il Reset, e questo è utile sotto certi aspetti, pericoloso per altri. L'utilità sta (come vedremo nei prossimi paragrafi) nella possibilità di modificare il processo standard di inizializzazione e di poter conservare dati utili anche al di là di una fatale eventualità di un Reset o di una Guru Meditation: alcuni programmi (come l'asdg.device della ASDG software) portano all'eccesso queste possibilità, consentendo di installare cose come «recoverable RAM disks», cioè RAM disk resistenti al Reset; ad esempio il già citato VD0: della

```
/* AddResident(residentList,residentTag)
 *
 *  Aggiunge la struttura Resident puntata da residentTag nella
 * lista residentList. residentList e' un APTR a un array di pun-
 * tatori a strutture Resident.
 */

int AddResident(reslist,restag)
  struct Resident **reslist;
  struct Resident *restag;
{
  short int i,j;
  BYTE pri;

  pri = restag->rt_Pri;                    /* Ottimizza i confronti */
  /* Cerca la Resident con priorita' inferiore o uguale alla nostra */
  for(i = 0;((reslist[i])&&(reslist[i]->rt_Pri > pri));i++);
  for(j = 0;(reslist[j]);j++);             /* Cerca la fine della lista */
  for(;j >= i;j--) reslist[j+1] = reslist[j]; /* Sposta tutti i puntatori */
  reslist[i] = restag;                     /* Inserisce la nostra Resident */
  return((int)i);                          /* Ritorna la posizione della
Resident */
}
```

Listato 1 - Una routine per le strutture Resident.

ASDG o il CARD: nientemeno che della Commodore-Amiga.

Il male sta (come abbiamo detto) nella possibilità di infezione da parte di un malvagio virus: nel qual caso è possibile installare un antivirus che pure si dimostrerà anch'esso esente dal Reset.

AllocAbs()

Titolo migliore non mi è venuto in mente per questo paragrafo ...

Questa funzione di Exec consente di allocare memoria a indirizzi assoluti, passando in A1 l'indirizzo del blocco che si vuole ottenere e in D0 la sua lunghezza (in byte). Nel caso questo blocco sia disponibile la routine ritorna in D0 lo stesso valore di A1, altrimenti torna 0.

AllocAbs() sembrerebbe un surrogato di AllocMem(): ma la differenza è grande; in effetti, la memoria chiesta al sistema tramite AllocMem() è sempre vulnerabile, in quanto la lista che la controlla (la MemList della struttura ExecBase, fig. 1) viene reinizializzata al momento del Reset. Al contrario AllocAbs() mantiene i suoi dati in altre liste (avevo pensato a KickMemPtr, ma questo puntatore vale sempre NULL... quindi non si capisce a cosa possa servire), che sono del tutto invulnerabili a un Reset. Pertanto, se vi serve (e vi servirà!) memoria «intoccabile», usate pure tranquillamente la AllocAbs(). Se non sapete dove «pescare» i vostri segmenti, potete richiederli al sistema tramite la AllocMem() e poi passarne l'indirizzo alla AllocAbs(): al momento del Reset rimarrà valida solo la richiesta effettuata tramite allocAbs().

Resident e ExecBase

L'esame attento di alcuni programmi mi ha anche aiutato a scoprire alcune faccende interessanti. Il nostro punto di riferimento sarà sempre la struttura ExecBase.

Dei vari parametri contenuti (la funzione di molti di essi si può intuire dal semplice nome) mi sono parsi i più interessanti ColdCapture, CoolCapture, WarmCapture, CheckSum, IntVects[], Quantum, Elapsed, DispCount, IdleCount, ResModules, LastAlert, PowerSupply Frequency, VBlankFrequency e KickTagPtr.

I vettori Capture puntano a routine in LM che vengono lanciate con una JSR in particolari momenti dell'attivazione del sistema dopo un reset o all'accensione:

```
struct ExecBase {
 struct Library  LibNode;
 UWORD   SoftVer;
 WORD    LowMemChkSum;
 ULONG   ChkBase;
 APTR    ColdCapture;
 APTR    CoolCapture;
 APTR    WarmCapture;
 APTR    SysStkUpper;
 APTR    SysStkLower;
 ULONG   MaxLocMem;
 APTR    DebugEntry;
 APTR    DebugData;
 APTR    AlertData;
 APTR    MaxExtMem;
 UWORD   ChkSum;
 struct  IntVector IntVects[16];
 struct  Task *ThisTask;
 ULONG   IdleCount;
 ULONG   DispCount;
 UWORD   Quantum;
 UWORD   Elapsed;
 UWORD   SysFlags;
 BYTE    IDNestCnt;
 BYTE    TDNestCnt;
 UWORD   AttnFlags;
 UWORD   AttnResched;
 APTR    ResModules;
 APTR    TaskTrapCode;
 APTR    TaskExceptCode;
 APTR    TaskExitCode;
 ULONG   TaskSigAlloc;
 UWORD   TaskTrapAlloc;
 struct  List MemList;
 struct  List ResourceList;
 struct  List DeviceList;
 struct  List IntrList;
 struct  List LibList;
 struct  List PortList;
 struct  List TaskReady;
 struct  List TaskWait;
 struct  SoftIntList SoftInts[5];
 LONG    LastAlert[4];
 UBYTE   VBlankFrequency;
 UBYTE   PowerSupplyFrequency;
 struct  List SemaphoreList;
 APTR    KickMemPtr;
 APTR    KickTagPtr;
 APTR    KickCheckSum;
 UBYTE   ExecBaseReserved[10];
 UBYTE   ExecBaseNewReserved[20];
}
```

Figura 1 - La struttura ExecBase.

```
/* CheckExec.c          by Maurizio Mangrella  1989 */

#include <exec/types.h>
#include <exec/execbase.h>
#include <exec/resident.h>
#include <stdio.h>

void main()
{
  struct ExecBase *SysBase;
  struct Resident *resm;
  short int i;
  char *s;

  SysBase = *(struct ExecBase **)4L;

  printf("CheckExec          by Maurizio Mangrella  1989\n");
  printf(" Diagnostico completo di Exec\n\n");

  printf("ColdCapture: $%08X\n",SysBase->ColdCapture);
  printf("CoolCapture: $%08X\n",SysBase->CoolCapture);
  printf("WarmCapture: $%08X\n\n",SysBase->WarmCapture);

  printf("System Stack:  Lower $%08X    Upper $%08X\n\n",
    SysBase->SysStkLower,SysBase->SysStkUpper);

  for(i = 0;i < 16;i++)
    printf("IntVector[%02d]:  Data $%08X    Code $%08X\n",
      i,SysBase->IntVects[i].iv_Data,SysBase->IntVects[i].iv_Code);

  printf("\nModuli residenti:\n\n");

  resm = *((struct Resident **)SysBase->ResModules);
  for(i = 0;(resm);i++) {
    resm = ((struct Resident **)SysBase->ResModules)[i];
    if(resm) {
      printf("Indirizzo $%08X   Init $%08X   Pri %03d\n",
        resm,resm->rt_Init,resm->rt_Pri);
      printf("  Name: ");
      for(s = resm->rt_Name;(*s);s++) if((*s) != '\n') putchar(*s);
      printf("   IdString: ");
      for(s = resm->rt_IdString;(*s);s++) if((*s) != '\n') putchar(*s);
      printf("\n");
    }
  }

  printf("\nLastAlert    0: $%08X    1: $%08X\n",
    SysBase->LastAlert[0],SysBase->LastAlert[1]);
  printf("              2: $%08X    3: $%08X\n\n",
    SysBase->LastAlert[2],SysBase->LastAlert[3]);

}
```

Listato 2

ColdCapture viene lanciato prima del test della ROM (grigio scuro), CoolCapture prima del test della RAM (grigio chiaro) e WarmCapture subito dopo questo. Se uno di questi vettori contiene un indirizzo diverso da 0, Exec salta alla routine corrispondente. Avete a disposizione uno stack piuttosto capace (credo che vada oltre i 2000 byte): comunque non avete bisogno di salvarvi i registri, in quanto potete utilizzarli tutti senza pietà.

ChkSum è la somma delle prime 24 word (da SoftVer a MaxExtMem) in complemento a 1: serve a Exec per controllare la coerenza del sistema. In LM si può ricalcolare così:

▼

```
        MOVEQ    #$17,D1        ; numero di words da esaminare
        LEA      $22(A6),A0     ; punta alla prima word
        CLR.L    D0             ; azzera la somma
Loop    ADD.W    (A0)+,D0       ; effettua la somma
        DBRA     D1,Loop
        NOT.W    D0             ; nega la somma
        MOVE.W   D0,(A0)        ; e la sistema in ChkSum
```

Altro modo possibile è quello (a mio parere più elegante, ma è questione di gusti) che si può ottenere dal C (notare la coppia Forbid()-Permit() che consente di evitare problemi a tempo di esecuzione):

▼

```
UWORD *p,sum,i;
  .
  .
p = (UWORD *)&SysBase->SoftVer;
for(i = 0,sum = 0;i < 24;i++) sum+=p[i];
Forbid(); SysBase->ChkSum = ~sum; Permit();
```

```
struct Resident {
 UWORD rt_MatchWord;            /* Vedi dopo */
 struct Resident *rt_MatchTag;  /* Punta all'inizio della struttura */
 APTR  rt_EndSkip;              /* Punta alla prima word successiva alla
struttura (EndSkip - MatchTag = 26) */
 UBYTE rt_Flags;                /* Vedi dopo */
 UBYTE rt_Version;              /* Versione del programma correlato */
 UBYTE rt_Type;                 /* Tipo (vedi dopo) */
 BYTE  rt_Pri;                  /* Priorita' */
 char  *rt_Name;                /* Nome */
 char  *rt_IdString;            /* Stringa di identificazione */
 APTR  rt_Init;                 /* Puntatore all'inizio del programma */
};
#define RTC_MATCHWORD   0x4AFC  /* Valore standard di rt_MatchWord */
#define RTF_COLDSTART   (1<<0)  /* Flag: indica che la struttura appartiene a
ResModules */
#define RTF_AUTOINIT    (1<<7)  /* ??? */

  I tipi delle strutture Resident sono uguali ai corrispondenti delle strutture
Node. Li riporto per completezza:

NT_UNKNOWN      0
NT_TASK         1
NT_INTERRUPT    2
NT_DEVICE       3
NT_MSGPORT      4
NT_MESSAGE      5
NT_FREEMSG      6
NT_REPLYMSG     7
NT_RESOURCE     8
NT_LIBRARY      9
NT_MEMORY       10
NT_SOFTINT      11
NT_FONT         12
NT_PROCESS      13
NT_SEMAPHORE    14
NT_SIGNALSEM    15
NT_BOOTNODE     16
```

Figura 2 - La struttura Resident.

```
struct IntVector {
 APTR    iv_Data;
 VOID    (*iv_Code)();
 struct  Node *iv_Node;
}
```

Figura 3 - La struttura IntVector.

IntVects[] è un array di 16 strutture IntVector (figura 3), ognuna delle quali gestisce un particolare evento legato all'hardware del computer; la corrispondenza, per quanto ho potuto capire, dovrebbe essere come rappresentato nella tabella della pagina accanto (non ne sono completamente sicuro, comunque...).

La struttura IntVector contiene tre item: iv_Data (un puntatore generico ad area dati che vi viene passato in A1 al momento della chiamata dell'interrupt), iv_Code (che punta alla routine di interrupt) e iv_Node (puntatore ad una struct Node). Val la pena di notare due cosette: 1) gli interrupt del 68000 sono stati «moltiplicati» (passando da sette a sedici!); 2) le routine di interrupt sono gestite via software, dunque NON devono terminare con una RTE (come si farebbe con un interrupt hardware) ma con una semplice RTS. A proposito di iv_Code, ricordo che, se questo vale NULL, al momento dell'esecuzione dell'interrupt Exec salta automaticamente a delle routine predisposte in ROM.

Dimenticavo che è possibile settare un interrupt con la funzione SetIntVector(), passando in D0 il numero dell'interrupt e in A1 la struct IntVector desiderata. Tramite la struttura puntata da iv_Node, si possono creare delle liste di interrupt server che vengono chiamati in sequenza (ognuno secondo la propria struct IntVector) in base alla loro priorità (definita nel campo In_Pri della struct Node): si può aggiungere in coda alla lista una IntVector con la funzione AddIntServer() e toglierla con la RemIntServer() (gli argomenti vengono passati come per la SetIntVector()).

Quantum è il numero di scambi di tick (normalmente 16) concessi ad ogni task: un tick equivale a un cinquantesimo di secondo. Elapsed è un contatore (che va da Quantum - 1 a 0) che segnala... il tempo scaduto. DispCount è il numero di scambi fra task effettuati dall'accensione (o dal reset), mentre IdleCount è il numero di tick (cinquantesimi di secondo) durante i quali, per un qualche motivo, Exec non ha potuto effettuare scambi di task. IdleCount viene continuamente incrementato durante tutto lo startup del KickStart, boot compreso.

Prima di continuare, vi scarico un altro barile di routinette Exec predispo-

```
Vettore    Funzione

   0       Porta seriale: buffer vuoto (scrittura completata)
   1       DMA del disco: buffer vuoto (scrittura completata)
   2       ???
   3       IRQ delle porte I/O: tastiera e orologio in tempo reale
   4       ???
   5       Video Blank
   6       Blitter: operazione terminata
   7       Canale audio 0: tavola terminata
   8       Canale audio 1: tavola terminata
   9       Canale audio 2: tavola terminata
  10       Canale audio 3: tavola terminata
  11       Porta seriale: buffer pieno (lettura completata)
  12       DMA del disco: buffer pieno (lettura completata)
  13       ???
  14       ??? (quanti sono?)
  15       NMI del 68000
```

ste alla gestione dei task: Forbid() blocca temporaneamente lo scambio di contesto, Permit() lo riabilita. Disable() blocca la gestione degli interrupt software, Enable() la riabilita in gran forma. Infine Dispatch() forza uno scambio di contesto prematuro (prima del game over...), ExitIntr() ci consente di uscire da una routine di interrupt in modo ordinato (come S.O. comanda), Schedule() e Reschedule() (a giudicare dal nome) dovrebbero riorganizzare l'ordine predisposto dei task (lo scheduling, appunto): di più non vi so dire...

ResModules punta a un array di longword ognuna delle quali punta a una particolare struttura Resident: studiandone le relative stringhe di identificazione ho constatato che, normalmente, ad ogni libreria di sistema residente in ROM corrisponde una struct Resident. Dopo il completamento della procedura di inizializzazione il KickStart (seguendo l'ordine delle priorità stabilito dall'item rt_Prl) salta, con una JSR, ai vettori rt_Init di ogni struttura: ecco un altro modo per modificare lo startup standard. Da notare che, per preservare le sciagurate strutture dal reset, ognuna di queste (insieme con il relativo segmento di codice puntato da rt_Init) deve essere allocata con la AllocAbs(). Faccio inoltre notare che le routine vengono lanciate, nell'ordine di priorità stabilito dall'item rt_Pri, subito prima del boot (che è proprio rappresentato dall'ultima Resident, che ha nome «strap») e che bisogna salvare i registri sullo stack al momento del lancio (e riprenderli al ritorno, obviously...). Le routine Exec FindResident() e InitResident() gestiscono (in maniera un po' primordiale, a dir la verità) le strutture Resident: FindResident cerca, nella lista ResModules, una struttura in base all'item rt_Name (bisogna passare l'indirizzo di una stringa contenente il nome in A1). Per quanto riguarda initResident(), il sottoscritto sta ancora duramente combattendo per cavarne qualcosa di utile...

LastAlert è un array di 4 ULONG in cui troverete i dati relativi all'ultima Guru Meditation scatenata nel sistema a partire dalla sua accensione: può essere utile per il debugging, ad esempio.

PowerSupplyFrequency è la frequenza di rete (in Hertz), VBlankFrequency è la frequenza (sempre in Hz) di refresh del video; in genere sono uguali... ma provate a installare un Amiga NTSC (con relativo monitor) qui in Italia: otterrete 50 in PowerSupplyFrequency e 60 in VBlankfrequency.

Aggiungere altri elementi alla lista

```
* CoolDemo.asm        - by Maurizio Mangrella  1989  *

_SysBase        EQU     4

                XREF    _LVOAllocAbs
                XREF    _LVOForbid
                XREF    _LVOPermit

Main            MOVE.L  _SysBase,A6
                MOVE.L  #200,D0
                MOVE.L  #$0007FC00,A1
                JSR     _LVOAllocAbs(A6)
                TST.L   D0
                BEQ.S   PocaMemoria
                LEA     Demo(PC),A0
                MOVE.L  #$0007FC00,A1
                MOVEQ   #50,D0
Loop1           MOVE.L  (A0)+,(A1)+
                DBRA    D0,Loop1
                JSR     _LVOForbid(A6)
                MOVE.L  #$0007FC00,$2E(A6)
                MOVEQ   #$17,D1
                CLR.L   D0
                LEA     $22(A6),A0
Loop2           ADD.W   (A0)+,D0
                DBRA    D1,Loop2
                NOT.W   D0
                MOVE.W  D0,(A0)
                JSR     _LVOPermit(A6)
                MOVEQ   #0,D0
                RTS
PocaMemoria     MOVEQ   #1,D0
                RTS


Demo            CLR.L   D0
                CLR.L   D1
DLoop           MOVE.B  D1,D0
                ANDI.B  #$0F,D0
                BTST    #4,D1
                BEQ.S   DSkip
                EORI.B  #$0F,D0
DSkip           CMP.B   $DFF006,D1
                BNE.S   DSkip
                MOVE.W  D0,$DFF180
                ADDQ.B  #1,D1
                BTST    #6,$BFE001
                BNE.S   DLoop
                RTS
EndDemo

DemoSize        EQU     EndDemo-Demo

                CNOP    0,4
                END
```

Listato 3

ResModules (che, poi, non è propriamente una lista...) non è difficile: ci viene in aiuto un altro parametro, KickTagPtr. Questo, come ResModules, è un puntatore a puntatore a Resident, cioè punta a un array di puntatori a strutture Resident terminante con una longword a 0. Normalmente, questo parametro vale 0, o punta a un array vuoto: ma possiamo modificarlo. Facendolo puntare a un array di puntatori da noi predefiniti il KickStart, al momento del Reset, reinizializzerà ResModules inserendovi le struct Resident da noi specificate con KickTagPtr, e nell'ordine di priorità. Thank you! Qui a fianco, un esempio in C (mi sto accorgendo che sto un po' esagerando con l'Assembler).

Come commento alla routine, ho presunto che, ai puntini, vengano sostituite righe nelle quali la struttura restag venga opportunamente inizializzata.

```
struct ExecBase *SysBase;
SysBase = *(struct ExecBase **)4L;    /* ricava ExecBase */
struct Resident restag;               /* la nostra Resident */
struct Resident *tagcopy;             /* copia "sicura" di restag */
struct Resident **taglist;            /* la nostra lista KickTagPtr */
  .
  .
CopyMem(&restag,0x7FC00,26);              /* copia restag */
taglist = (struct Resident **)0x7FC1A;    /* indirizzo della lista */
taglist[0] = 0x7FC00; taglist[1] = 0L;    /* la inizializza */
SysBase->KickTagPtr = taglist;            /* inizializza KickTagPtr */
SysBase->KickCkeckSum = SumKickData();    /* risistema il checksum */
```

```
* ResidentDemo.asm        © by Maurizio Mangrella  1989  *

_SysBase        EQU     4

NT_UNKNOWN      EQU     0
RTW_COLDSTART   EQU     1

                XREF    _LVOSumKickData

Main            MOVE.L  _SysBase,A6          ;riferimento a Exec
                MOVE.L  #$0007FC00,A1        ;indirizzo della nostra zona RAM
                LEA     Resident,A0          ;indirizzo dei nostri dati
                MOVEQ   #33,D0               ;34 bytes (2 long + 26 x Resident)
Loop1           MOVE.B  (A0)+,(A1)+          ;copia i dati ...
                DBRA    D0,Loop1             ;... fino alla fine
                MOVE.L  #$0007FC22,A1        ;indirizzo del programma Demo
                LEA     Demo(PC),A0          ;ne forma l'indirizzo
                MOVEQ   #64,D0               ;64 longwords (256 bytes)
Loop2           MOVE.L  (A0)+,(A1)+          ;copia il programma
                DBRA    D0,Loop2
                MOVE.L  #$0007FC00,$226(A6)  ;indirizzo lista in KickTagPtr
                JSR     _LVOSumKickData(A6)  ;ricalcola il checksum
                MOVE.L  D0,$22A(A6)          ;lo sistema in KickCheckSum
                CLR.L   D0                   ;ritorna 0 (tutto OK)
                RTS

Demo            MOVEM.L D0/D1/A0,-(SP)       ;salva i registri
                LEA     $00DFF000,A0         ;base dei registri hardware
                MOVE.W  #$0200,$100(A0)      ;disabilita il raster di Denise
                MOVE.W  #$0020,$09A(A0)
                CLR.L   D1                   ;D1 = 0 (contatore righe)
                CLR.L   D0                   ;D0 = 0 (colore)
DLoop           MOVE.B  D1,D0                ;prende gli 8 bits bassi di D1
                ANDI.B  #$0F,D0              ;isola il nibble basso
                BTST    #4,D1                ;se il bit 4 di D1 ♦ settato ...
                BEQ.S   DSkip
                EORI.B  #$0F,D0              ;... inverte il nibble basso
DSkip           CMP.B   $006(A0),D1          ;aspetta la riga
                BNE.S   DSkip
                MOVE.W  D0,$180(A0)          ;setta il colore
                ADDQ.B  #1,D1                ;prossima riga
                BTST    #6,$BFE001           ;tasto sinistro del mouse premuto?
                BNE.S   DLoop                ;no: continua
                MOVEM.L (SP)+,D0/D1/A0       ;recupera i registri
                RTS                          ;ritorna

Resident        DC.L    $0007FC08            ;Indirizzo della Resident
                DC.L    0                    ;Fine lista

                DC.W    $4AFC                ;MatchWord
                DC.L    $0007FC08            ;MatchTag: punta a inizio Resident
                DC.L    $0007FC22            ;EndSkip: punta a fine Resident
                DC.B    RTW_COLDSTART        ;Flags: segmento per ColdStart
                DC.B    33                   ;Version: 33
                DC.B    NT_UNKNOWN           ;Type: UNKNOWN
                DC.B    -5                   ;Pri: -5
                DC.L    $0007FC1A            ;Name: punta a "\0" (stringa vuota)
                DC.L    0                    ;IdString: puntatore nullo
                DC.L    $0007FC22            ;Init: punta al programma

                CNOP    0,4
                END
```

Listato 4

Curiosità: provando ad allocare con AllocAbs() la memoria che mi serviva, ho notato che il tutto non funzionava; eliminando le chiamate alla funzione tutto andava liscio. Hmmm, strano ... Qualcuno sa spiegarmene il perché?

Infine, date un'occhiata a SumKickData() (routine del KickStart 1.2 e segg.) che risistema per noi il parametro KickChecksum di ExecBase (con cui Exec controlla l'integrità dei suoi dati). Durante i cambiamenti non è necessario invocare la Forbid() in quanto questi parametri vengono adoperati da Exec solo durante il reset.

Software support

Soliti programmetti di esempio: cominciamo con una routinetta «accunciatella accunciatella» (come si dice dalle mie parti) per inserire una struttura residente in una lista organizzata alla maniera di ResModules in base alla priorità della struttura stessa: la routine presume che la lista sia in RAM e liberamente modificabile (ad ogni inserimento diventa di 4 byte più lunga).

CheckExec è un programma corto che però fornisce un output chilometrico: stampa su _stdout le più importanti informazioni concernenti ExecBase; spero che vi torni in qualche modo utile...

CoolDemo, infine, vi mostrerà come usare i vettori Capture (nel caso specifico CoolCapture): a me, in particolare, è piaciuto l'effetto cromatico del tutto; ai posteri — come sempre — l'ardua sentenza. ResidentDemo ripropone la stessa routine, con la differenza che, per l'inizializzazione, ci si serve, questa volta, di KickTagPtr.

È tutto (sto cominciando a dare i numeri...): ci rivedremo dopo un Reset... o dopo aver gettato l'Amiga dalla finestra in un momento di drammatica disperazione. MC

Qui Romaufficio a voi Managers.

FIERA DI ROMA
ORARIO 9,30-19,00

PROMOSSA
DALL'ISTITUTO MIDES

SERVIZIO INFORMAZIONI A CURA
IBM ITALIA

REGISTRAZIONE VISITATORI SU COMPUTERS

Buffetti

FEDERLEASING
IN FIERA A CANONI AGEVOLATI

ROMAUFFICIO'90

12ª MOSTRA DELLE NUOVE TECNOLOGIE
PER L'AZIENDA
LO STUDIO PROFESSIONALE
LA PUBBLICA AMMINISTRAZIONE

16-20 MARZO
5 giorni da non perdere

MASSIMO CURCIOTTI

Microforum

MITO

PREFORMATTED & VERIFIED • 100% ERROR FREE •

DOS 3.0 OR HIGHER

Nuovo!

Nuovo da Microforum!
I dischi Mito oggi li trovi anche preformattati e verificati:
con poche lire in più ti assicuri un risparmio di tempo,
la certezza della qualità e una velocità impagabile nelle situazioni in cui devi salvare i dati senza l'obbligo di uscire dal programma.

Mito Microforum, *ancora una volta più avanti.*

Media Disk Antonelli
12, Via Ciociaria - 00162 Roma
Telefono 06/4240379

Floppy's Market
5, P.za del Popolo
56029 S.Croce sull'Arno (PI)
Telefono 0571/35124

Simple Soft
Via Cesana 6 - 20132 Milano
Telefono 02/2841141

Microforum
MANUFACTURING INC.
TORONTO - CANADA