```
{
*****************************************************************************
          Universitetet i Bergen
       Institutt for den Faste Jords Fysikk

        S Y N C  U N I T  P R O G R A M
          Skrevet av Anders Davidsen


Beskrivelse : Programmet styrer GUNCO, DFS, magnetometer og annet utstyr.
        Hendelser trigges enten eksternt (RS-232 linje, ASCII karakter)
        eller med intern timer.
CopyRight :   Universitetet i Bergen,
        Institutt for den Faste Jords Fysikk (1999, 2000)
Spraak    : Turbo Pascal 7.0
CPU       : NEC V25+ (NB: Register mapping forskjellig fra V25)
Tabs      : 4

Rev.      : Rev   Dato    Av  Beskrivelse
        ------------------------------------------------
        0.1   06. jan 99  AD  Basert paa Geo-Pinger
        0.20  13. jan 99  AD  Nytt LCD lagt inn
        0.30  02. feb 99  AD  DFS modus lagt inn
        0.50  09. mar 99  AD  Magnetometer lagt inn
        0.6   07. mai 99  OM  Magnetometer paa Mosby gaar
                    ikke mot kortet. Test/debug
                    versjon for aa finne feilen.
                    1. Storre time-out delay for
                       magnetometer.
                    2. Magnetometer default ON,
                       intervall 5s.
                    3. Serial char inn: Kast vekk
                       CR og LF.
        0.61  31. aug 99  OM  1. Starter i DFS modus (linje 342)
                    2. Magnetometer starter AV (l. 347).
                    3. Flere kommentarer (l. 159 .. 169).
                    4. Betinget sign-on message (1745 ..)
                    5. Endret ver. tekst (180, 181)
                    6. Erstattet EXIT med BREAK i
                       WriteOBSData rutinen.
                    7. Endret noe init tekst, i MAIN.
        0.7   07. sep 99  OM  1. Storre endringer i DFS trigge rutine.
        1.0   20. sep 99  OM  1. Lagt inn 'filter' paa trigge inngang,
                         slik at nytt trigge signal ikke blir
                         godtatt for det er gått en viss tid
                         (satt i konstant).
                    2. Endret magnetometerrutine slik at
                         time-out ikke trigger lesning.
        1.13  03. feb 00  OM  Linje 1098: Invertert polaritet på
                    externt triggesignal (utgang), fordi
                    det skal drive optocoupler.
        1.14  13. apr 00  OM  1. Endret GUNCO timing:

*****************************************************************************


}
{$N-} {No numeric co-processor}

{
  ***********************************************************
       10MHz Klokke: 12.8us timer resolution.
       0.1ms = 7.8125 tics ->     8 tics =   0.1024ms
       1.0ms = 78.125 tics ->    78 tics =   0.9984ms
      10.0ms = 781.25 tics ->   781 tics =   9.9968ms
       100.0ms = 7812.5 tics ->  7813 tics = 100.0064ms
  ***********************************************************


  Display functions
  ============================================================
  Instruction   DB5   DB4   DB3   DB2   DB1   DB0
  ------------------------------------------------------------
  Clear display 0   0   0   0   0   1
  Return home   0   0   0   0   1   -
  Entry mode set  0   0   0   1   I/D   S
  Display on/off  0   0   1   D   C   B
  Cursor/disp sh  0   1   S/C   R/L   -   -
  Function set  1   DL    N   F   -   -
  ------------------------------------------------------------

    I/D = 1: Increment
    I/D = 0: Decrement
    S = 1: Accompanies display shift
    S/C = 1: Display shift
    S/C = 0: Cursor move
    R/L = 1: Shift to right
    R/L = 0: Shift to left
    DL  = 1: 8 bits, DL = 0: 4 bits
    N = 1: 2 lines, N = 0: 1 line
    F = 1: 5*10 dots, F = 0: 5*8 dots
    D = 1: Display on
    D = 0: Display off
    C = 1: Cursor on
    B = 1: Cursor blinking
  ------------------------------------------------------------
```

```
  97:
  98:
  99:   Hardware resources
 100:   ------------------------------------------------------------
 101:   PORT 0:
 102:   P0.0 - Input: Wire blast fra DFS
 103:   P0.1 - Input: Time break fra GUNCO
 104:   P0.2 - Input: Fileno data fra DFS
 105:   P0.3 - Output: Not used
 106:   P0.4 - Output: Start DFS
 107:   P0.5 - Output: Time break til DFS
 108:   P0.6 - Output: Magnetometer ext trig
 109:   P0.7 - Output: Not used
 110:
 111:
 112:   PORT 1:
 113:   P1.0 - Interrupt input - Magnetometer print
 114:   P1.1 - Interrupt input - DFS clock
 115:   P1.2 - Interrupt input - Button "Mode"
 116:   P1.3 - Interrupt input - Button "Select"
 117:   P1.4 - Output: Closure to GUNCO
 118:   P1.5 - Output: Fire to GUNCO
 119:   P1.6 - Output: External trigger signal
 120:   P1.7 - Output: LED
 121:
 122:   PORT 2, LCD.COM uses these ports:
 123:   P2.0 - D4
 124:   P2.1 - D5
 125:   P2.2 - D6
 126:   P2.3 - D7
 127:   P2.4 - Read/Write
 128:   P2.5 - Reset
 129:   P2.6 - Enable display 1
 130:   P2.7 - Enable display 2
 131:   ------------------------------------------------------------
 132:
 133:
 134:   RS-232 data format
 135:   ------------------------------------------------------------
 136:   Magnetometer: $PUIBR,MAG,mmmmm*FF
 137:
 138:           mmmmm - Magnetometer data
 139:
 140:   ShotInfo DFS: $PUIBR,SHT,DFS,e,llll,iiiii,m,fffff*FF
 141:
 142:           e   - Status
 143:           llll  - Trig pulse length
 144:           iiiii - Trig interval
 145:           m     - Trig mode
 146:           ffff  - DFS file number
 147:
 148:   ShotInfo OBS: $PUIBR,SHT,OBS,e,xxxx,cccc,tttt,llll,iiiii,m*FF
 149:           e   - Status
 150:           xxxx  - External trig delay (may be negative)
 151:           cccc  - GUNCO Closure/Fire delay
 152:           tttt  - GUNCO Trig/Fire delay
 153:           llll  - Trig pulse length
 154:           iiiii - Trig interval
 155:           m   - Trig mode
 156:
 157:
 158:   Trig mode (m):  0 - No Trigging
 159:           1 - External Trigging
 160:           2 - Internal Trigging
 161:
 162:   Status (e):   0 - Fire OK
 163:           1 - No GUNCO Time Break
 164:           2 - No DFS Wire Blast
 165:           3 - No DFS Filenumber
 166:           8 - Info
 167:           9 - Trigger error
 168:
 169:
 170:   Note:   Delays, lengths and intervals are all written in
 171:       milliseconds
 172:
 173:
 174:   Host commands over RS-232:
 175:
 176:           'A' or 'a'  = Fire GUNCO and DFS (if DFS mode chosen)
 177:           'B' or 'b'  = Send info stuff to host
 178:           '?'     = System information
 179:           'Q' or 'q'  = Exit to DOS
 180:
 181:
 182:   Pin-out serial connector, DSUB 25 male:
 183:
 184:           Pin 2 = Data in to sync unit
 185:           Pin 3 = Data out from sync unit
 186:           Pin 7 = GND
 187:
 188:
 189:   Display lay-out:  4 x 20 char display is used.
 190:
 191:                1         2
 192:       Pos   12345678901234567890
```

```
193:          --------------------
194:       L1:   Mode: DFS
195:       L2:   GUN : -----
196:       L3:   DFS : -----
197:       L4:   MAG : -----
198:
199:       LCD_VAR_1_POS     = 7;
200:       LCD_VAR_2_POS     =  14;
201:       LCD_FFID_POS      = 2;
202:       LCD_GUNCO_POS     = 8;
203:       LCD_STATUS_POS      = 14;
204:
205: }
206: Program SyncUnit;
207:
208: Uses
209:   Dos;
210:
211: Const
212:
213:   VERSION         = 'Version 1.14';
214:   VERSION_DATE      = '13. apr. 2000 ';
215:
216:   MODE_OBS  = 0;
217:   MODE_DFS  = 1;
218:
219:   TRIG_CHAR = 'A';          {ASCII command via RS-232 that starts firing sequence}
220:
221:   TRIG_LOCKED_DURATION  =  15;    {Deny false triggers from Eiva in 15s period after last one}
222:   TIMER_INTERVAL = 2;         {Timer interval in ms}
223:
224:   MAG_INTVL_MAX = 20000;      {Max 20s}
225:   MAG_INTVL_MIN = 5000;       {Min 5s}
226:
227:   EXT_TRIG_DLY_STEP = 10;         {trig delay step 10ms}
228:   EXT_TRIG_DLY_MAX = 4000;      {Max 4000ms}
229:
230:   StepClosureFireDelay = 10;      {Delay colsure/fire step 10ms}
231:   MaxClosureFireDelay = 1000;         {Max 1000ms}
232:
233:   StepGUNCO_TrigFireDelay = 2;    {GUNCO trig fire delay step 2ms}
234:   MaxGUNCO_TrigFireDelay  = 254;       {Max 254ms}
235:   MinGUNCO_TrigFireDelay = 0;     {Min 0ms}
236:
237:   StepTrigLength = 2;         {Trig length step 2ms}
238:   MaxTrigLength = 100;        {Max 100ms}
239:   MinTrigLength = 2;          {Min 2ms}
240:
241:   TRIG_INTVL_STEP = 1000;       {Trig interval step 1s}
242:   TRIG_INTVL_MAX = 20000;       {Max 20s}
243:   TRIG_INTVL_MIN = 4000;        {Min 4s}
244:
245:   GUNCO_FTB_TIMEOUT = 200;    {Max 200ms between GUNCO fire and timebreak}
246:   DFS_WIREBLAST_TIMEOUT = 500;    {Max 500ms between DFS start and wire blast}
247:   DFS_FFID_TIMEOUT = 500;       {Max 500ms between end of fire routine and DFS filenumber received}
248:   MAG_READ_TIMEOUT = 4500;      {Max 4500ms between mag. trig and mag. reading}
249:
250:
251:
252:
253:
254:
255:   TrigMode_Values: Array[0..2] of String[3] = ('---','Ext','Int');
256:   SyncUnitMode_Values: Array[0..1] of String[3] = ('OBS','DFS');
257:
258:
259:   LCD_VAR_1_POS     = 7;
260:   LCD_VAR_2_POS     =  14;
261:   LCD_FFID_POS      = 2;
262:   LCD_GUNCO_POS     = 8;
263:   LCD_STATUS_POS      = 14;
264:   LCD_LINE_1        = 1;
265:   LCD_LINE_2        = 2;
266:   LCD_LINE_3        = 3;
267:   LCD_LINE_4        = 4;
268:
269:   DisplayCtrl = Chr(160);       {Control command to display}
270:   Line1     = 128;          {Start of line 1}
271:   Line2   = 192;          {Start of line 2}
272:   Line3   = 148;          {Start of line 3}
273:   Line4   = 212;          {Start of line 4}
274:
275:
276:   {***********   V 2 5 +   R E G I S T E R   M A P P I N G   *********}
277:
278:   Seg   =   $F000; {Segment address for CPU special registers}
279:
280:   PRC   = $FFEB; {Processor control register}
281:
282:   P0    = $FF00; {Port 0 register}
283:   PM0   = $FF01; {Mode register for port 0}
284:   PMC0  =   $FF02; {Mode control register for port 0}
285:
286:   P1    = $FF08; {Port 1 register}
287:   PM1   = $FF09; {Mode register for port 1}
288:   PMC1  = $FF0A; {Mode control register for port 1}
```

```
289:
290:   INTM  = $FF40; {External interrupt mode register}
291:   EXIC0   =   $FF4C; {External int. request register 0}
292:   EXIC1 = $FF4D; {External int. request register 1}
293:   EXIC2 = $FF4E; {External int. request register 2}
294:
295:
296:   SRIC0   = $FF6D; {Receive completion interrupt request register, port 0}
297:   RxB0  = $FF60; {Receive buffer}
298:   SCS0  = $FF6B; {Serial status register 0}
299:
300:   MD1   = $FF8A; {Modulo/Timer register for timer 1}
301:   TMC1  = $FF91; {Timer control register for timer 1}
302:   TMIC2 = $FF9E; {Timer unit interrupt controll register, timer 1}
303:
304:   Base8255=   $300;  {Base address to 8255 (magnetometer interface)}
305:
306:   CR    = #$0D;
307:   LF    = #$0A;
308:   ESC   = #$1B; {# will force char type}
309:
310: type
311:   typeMenuMode = (menuFirst, menuReady, menuSyncUnitMode, menuExtTrigDelay,
312:   menuTrigLength, menuTrigMode,
313:   menuTrigInterval, menuMagOn, menuMagInterval, menuLast);
314:
315:   typeOBS_Values = Record
316:     ExtTrigDelay: Integer;
317:     ClosureFireDelay: Word;
318:     GUNCO_TrigFireDelay: Byte;
319:     TrigLength: Byte;
320:     TrigInterval: Word;
321:     TrigMode: Byte;
322:     ClosureExtTrigDelay: Word;
323:     MinExtTrigDelay: Integer;
324:     MinClosureFireDelay: Byte;
325:   End;
326:
327:   typeDFS_Values = Record
328:     TrigLength: Byte;
329:     TrigInterval: Word;
330:     TrigMode: Byte;
331:     FileNumber: Word;
332:   End;
333:
334:   typeMag_Values = Record
335:     TrigInterval: Word;
336:     DataValue: LongInt;
337:   End;
338:
339: Var
340:   TextString: String;
341:   ValueToStr: String;
342:
343:   MenuMode: typeMenuMode;
344:
345:   OBS_Values: typeOBS_Values;
346:   DFS_Values: typeDFS_Values;
347:   Mag_Values: typeMag_Values;
348:
349:   Display: Text;
350:
351:   TimerLimit: Word;
352:
353:   TestChar: Char;
354:   TrigDelayCounter: Word;
355:   MagCounter: Word;
356:   SyncUnitMode: Byte;
357:   FileNumberCharCount: Word;
358:
359:   Int2Vector : Pointer; {Magnetometer interrupt}
360:   Int13Vector: Pointer; {Receiver interrupt}
361:   Int24Vector: Pointer; {External interrupt 0}
362:   Int25Vector: Pointer; {External interrupt 1}
363:   Int26Vector: Pointer; {External interrupt 2}
364:   Int28Vector: Pointer; {Timer 0 interrupt}
365:   Int30Vector: Pointer; {Timer 1 interrupt}
366:
367:   EndTestProgram: Boolean;
368:   Serial0Interrupt: Boolean;
369:   ModeButtonInterrupt: Boolean;
370:   SelectButtonInterrupt: Boolean;
371:   FastValueChange: Boolean;
372:   CheckSelectButtonState: Boolean;
373:   DFS_IntervalTrigOn: Boolean;
374:   OBS_IntervalTrigOn: Boolean;
375:   ShotCounter : Integer;
376:   DisplayFire: Boolean;
377:   FileNumberReceived: Boolean;
378:   GetMagData: Boolean;
379:   MagDataReady: Boolean;
380:   MagIntDidHappen : Boolean;
381:   MagnetometerOn: Boolean;
382:   WaitingForMagData : Boolean;
383:   MagTimeOut : Boolean;
384:   MagStatus  : byte;
```

```
385:   Mag_TimeOut_Counter : Integer;
386:   Mag_Counter : Integer;
387:   Seconds : Integer;
388:   MilliSeconds : Integer;
389:   TrigLockOut : Boolean;
390:   TrigMutePeriod : Integer;
391:   ErrorCode : Integer;
392:
393: (*****************************************************
394:         W A I T  R O U T I N E
395: *****************************************************)
396:
397: Procedure Wait(TimeDelay: Word);
398: Var
399:   OffsetTrigDelayCounter: Word;
400:
401: Begin
402:   OffsetTrigDelayCounter:= TrigDelayCounter;
403:
404:   If ((OffsetTrigDelayCounter + TimeDelay) >= TimerLimit) Then Begin
405:     While (TrigDelayCounter >= OffsetTrigDelayCounter) Do;
406:     OffsetTrigDelayCounter:= (TimeDelay - (TimerLimit - OffsetTrigDelayCounter));
407:     While (TrigDelayCounter < OffsetTrigDelayCounter) do;
408:   End Else Begin
409:     OffsetTrigDelayCounter:= (OffsetTrigDelayCounter + TimeDelay);
410:     While (TrigDelayCounter < OffsetTrigDelayCounter) Do;
411:   End;
412: End;
413:
414:
415: Procedure WriteDisplayText(DispText: String; RowNr, ColNr: Byte);
416: Var
417:   Line: Byte;
418: Begin
419:   Case RowNr of
420:     1 : Line:= Line1;
421:     2 : Line:= Line2;
422:     3 : Line:= Line3;
423:     4 : Line:= Line4;
424:   End;
425:
426:   Write(Display, DisplayCtrl, Chr(Line + ColNr - 1));
427:   Write(Display, DispText);
428: End;
429:
430:
431:
432:
433: (*****************************************************
434:   V A R I A B L E S   I N I T I A L I Z A T I O N
435: *****************************************************)
436:
437: procedure VarInitialize;
438: begin
439:   EndTestProgram      := false;
440:   Serial0Interrupt    := false;
441:   ModeButtonInterrupt   := false;
442:   SelectButtonInterrupt := false;
443:   FastValueChange     := false;
444:   CheckSelectButtonState  := false;
445:   DFS_IntervalTrigOn    := false;
446:   OBS_IntervalTrigOn    := false;
447:   DisplayFire       := false;
448:   FileNumberReceived    := false;
449:   GetMagData        := false;
450:   MagDataReady      := false;
451:   MagnetometerOn      := false;
452:   MagIntDidHappen     := false;
453:   WaitingForMagData   := false;
454:   MagTimeOut        := false;
455:   MagStatus       := 0;
456:   Mag_TimeOut_Counter   := 0;
457:   ShotCounter       := 0;
458:   Mag_Counter       := 0;
459:   Seconds         := 0;
460:   MilliSeconds      := 0;
461:   TrigLockOut       := false;
462:   TrigMutePeriod      := TRIG_LOCKED_DURATION;
463:
464:
465:   TimerLimit        := (65536 - TIMER_INTERVAL);
466:
467:   MenuMode        := menuSyncUnitMode;
468:   SyncUnitMode      := MODE_OBS;
469:
470:   With OBS_Values Do Begin
471:     ExtTrigDelay    := 0;         {0ms trig delay to EPC}
472:     ClosureFireDelay  := 600;         {600ms delay between closure and fire signal}
473:     GUNCO_TrigFireDelay := 128;         {128ms delay between GUNCO trigging and fire}
474:     TrigLength      := 10;          {10ms pulse length on closure and EPC_trig}
475:     TrigInterval    := 15000;       {15s trig interval}
476:     TrigMode      := 1;         {Trig when ASCII 'A' or 'a' received}
477:     ClosureExtTrigDelay := ClosureFireDelay+GUNCO_TrigFireDelay+ExtTrigDelay;
478:     MinExtTrigDelay   := (-10) * Trunc((GUNCO_TrigFireDelay)/EXT_TRIG_DLY_STEP);  {Delay between fire and
     EPC trig}
479:     MinClosureFireDelay := 10 * Round((TrigLength+3)/10);   {Delay between closure and fire to GUNCO}
```

```
480:   End;
481:
482:   With DFS_Values Do Begin
483:     TrigLength    := 100;   {100ms pulse length to Start DFS, Closure and Time break DFS}
484:     TrigInterval  := 15000;   {15s trig interval}
485:     TrigMode    := 1;   {External trigging, trigs on ASCII a or A}
486:     FileNumber    := 0;   {File number from DFS}
487:   End;
488:
489:   With Mag_Values Do Begin
490:     TrigInterval  := 5000;  {5s trig interval}
491:     DataValue   := 0;   {Value from magnetometer}
492:   End;
493:
494:   TrigDelayCounter  := 0;     {Timer interrupt counter}
495:   MagCounter      := 0;   {Timer interrupt counter}
496:   FileNumberCharCount := 0;   {Counts how many character received from DFS}
497: end;
498:
499:
500: (*************************************************
501:   D I S P L A Y   I N I T I A L I Z A T I O N
502: *************************************************)
503:
504: Procedure InitializeDisplay;
505: Var
506:   I: Byte;
507:   Str4: String[4];
508:   Str3: String[3];
509: Begin
510:   Assign(Display, 'LPT1');    {LCD.COM redirects data sendt to LPT1 to LCD}
511:   Rewrite(Display);       {Open for output}
512:
513:   {Init display}
514:   Write(Display, DisplayCtrl, Chr(40));   {4 bits data, 2 lines, 5*8 dots}
515:   Write(Display, DisplayCtrl, Chr(12)); {Display on, cursor and blink off}
516:   Write(Display, DisplayCtrl, Chr(6));  {Increment cursor, no display shift}
517:   Write(Display, DisplayCtrl, Chr(1));  {Clear and home display}
518:
519:   WriteDisplayText('Recording Sync', LCD_LINE_1, 4);
520:   WriteDisplayText('Unit',           LCD_LINE_2, 9);
521:   WriteDisplayText(VERSION,          LCD_LINE_3, 5);
522:   Wait(1000);
523:
524:   Write(Display, DisplayCtrl, Chr(1));      {Clear and home display}
525:
526:   WriteDisplayText('University of Bergen', LCD_LINE_1, 1);
527:   WriteDisplayText('Institute of',         LCD_LINE_2, 5);
528:   WriteDisplayText('Solid Earth Physics',  LCD_LINE_3, 1);
529:   WriteDisplayText( VERSION_DATE,          LCD_LINE_4, 5);
530:   Wait(1500);                   {Wait 1500ms}
531:
532:   Write(Display, DisplayCtrl, Chr(1));      {Clear and home display}
533:
534:   WriteDisplayText('Mode: ',      LCD_LINE_1, 1);
535:   WriteDisplayText('GUN : -----', LCD_LINE_2, 1);
536:   WriteDisplayText('DFS : -----', LCD_LINE_3, 1);
537:   WriteDisplayText('MAG : -----', LCD_LINE_4, 1);
538:   If (SyncUnitMode = MODE_DFS) then WriteDisplayText('DFS', LCD_LINE_1, LCD_VAR_1_POS)
539:   else WriteDisplayText('OBS', LCD_LINE_1, LCD_VAR_1_POS);
540:
541: End;
542:
543:
544: (*************************************************************
545:   P O R T / T I M E R / 8 2 5 5   I N I T I A L I Z A T I O N
546: *************************************************************)
547:
548: procedure InitPorts;
549: begin
550:   (* Initialize port 0. Pins 0 to 2 will be used for  *)
551:   (* input, the rest will be used for output.         *)
552:   mem[Seg:PMC0]:= $0;
553:   mem[Seg:PM0]:= $07;
554:   mem[Seg:P0]:= mem[Seg:P0] And ($FF-$F8); {Initialize ouputs to LOW}
555:
556:   (* Initialize port 1. Pins 1 to 3 will be used for  *)
557:   (* input, the rest will be used for output.         *)
558:   mem[SEG:PMC1]:= $0;
559:   mem[SEG:PM1]:= $0E;
560:   mem[Seg:P1]:= mem[Seg:P1] And ($FF-$F0); {Initialize outputs to LOW}
561:
562:   (* Set interrupt lines to trig on rising edges. *)
563:   mem[SEG:INTM]:= $55;
564: end;
565:
566:
567: Procedure InitTimers;
568: Var
569:   MD0_Number: Word;
570: Begin
571:   {Initialize Timer 1 to 2ms}
572:   MD0_Number:= Round(TIMER_INTERVAL*78.125);
573:   memw[Seg:MD1]:= MD0_Number;
574:
575:   {Start timer}
```

```
576:   mem[Seg:TMC1]:= $C0;
577: End;
578:
579:
580:
581:
582: Procedure Init8255;
583: Begin
584:   {Write control register, all ports are inputs, mode 0}
585:   Port[Base8255+3]:= $9B;
586: End;
587:
588:
589:
590: (*************************************************************
591:   G E T   C O M M A N D   L I N E   P A R A M E T E R S
592: *************************************************************)
593:
594: procedure GetParameters;
595: begin
596:
597:
598:  case ParamCount of
599:   0 : Begin
600:       end;
601:   1 : begin
602:       If (ParamStr(1) = 'OBS') then SyncUnitMode := MODE_OBS
603:       else SyncUnitMode := MODE_DFS;
604:       end;
605:   2 : begin
606:       If (ParamStr(1) = 'OBS') then SyncUnitMode := MODE_OBS
607:       else SyncUnitMode := MODE_DFS;
608:       Val(ParamStr(2), TrigMutePeriod, ErrorCode);
609:       If ErrorCode <> 0 then TrigMutePeriod := TRIG_LOCKED_DURATION;
610:       end;
611:
612:  end; {Case}
613:
614: end;  {GetParameters}
615:
616:
617:
618:
619: (*************************************************
620:          U T I L S
621: *************************************************)
622:
623: Function Hex2Str(Value: Byte): String;
624:   Function Digit(n: Byte): Char;
625:     Begin
626:       If (n < 10) Then Digit:= Chr(48+n)
627:       Else Digit:= Chr(55+n);
628:     End;
629: Begin
630:   Hex2Str:= Digit(Value Div 16) + Digit(Value Mod 16);
631: End;
632:
633:
634:
635: Function MakeCSum(Data: String): String;
636: Var
637:   T   : Word;
638:   ChkSum  : Byte;
639: Begin
640:   ChkSum:= Ord(Data[2]);
641:   For T:= 3 To Length(Data)-1 Do
642:     ChkSum:= ChkSum Xor Ord (Data[T]);
643:   MakeCSum:= Hex2Str(ChkSum);
644: End;
645:
646:
647: Procedure WriteDFSData(err: Byte);
648: Begin
649:   TextString:= '$PUIBR,SHT,DFS,';
650:
651:   Str(err:1, ValueToStr);
652:   TextString:= TextString + ValueToStr + ',';
653:
654:   With DFS_Values Do Begin
655:     Str(TrigLength:4, ValueToStr);
656:     TextString:= TextString + ValueToStr + ',';
657:
658:     Str(TrigInterval:5, ValueToStr);
659:     TextString:= TextString + ValueToStr + ',';
660:
661:     Str(TrigMode:1, ValueToStr);
662:     TextString:= TextString + ValueToStr + ',';
663:
664:     Str(FileNumber:4, ValueToStr);
665:     TextString:= TextString + ValueToStr + '*';
666:   End;
667:
668:   {'$PUIBR,SHT,DFS,e,llll,iiiii,m,ffff*FF'}
669:   Writeln(TextString + MakeCSum(TextString));
670: End;
671:
```

```
672:
673: Procedure WriteOBSData(err: Byte);
674: Begin
675:   TextString:= '$PUIBR,SHT,OBS,';
676:
677:   Str(err:1, ValueToStr);
678:   TextString:= TextString + ValueToStr + ',';
679:
680:   With OBS_Values Do Begin
681:     Str(ExtTrigDelay:4, ValueToStr);
682:     TextString:= TextString + ValueToStr + ',';
683:
684:     Str(ClosureFireDelay:4, ValueToStr);
685:     TextString:= TextString + ValueToStr + ',';
686:
687:     Str(GUNCO_TrigFireDelay:4, ValueToStr);
688:     TextString:= TextString + ValueToStr + ',';
689:
690:     Str(TrigLength:4, ValueToStr);
691:     TextString:= TextString + ValueToStr + ',';
692:
693:     Str(TrigInterval:5, ValueToStr);
694:     TextString:= TextString + ValueToStr + ',';
695:
696:     Str(TrigMode:1, ValueToStr);
697:     TextString:= TextString + ValueToStr + '*';
698:   End;
699:
700:   {'$PUIBR,SHT,OBS,e,xxxx,cccc,tttt,llll,iiiii,m*FF'}
701:   Writeln(TextString + MakeCSum(TextString));
702: End;
703:
704: Procedure CheckSelectButton;
705: Begin
706:   If ((mem[Seg:P1] And $08) = $08) Then FastValueChange:= True
707:   Else Begin
708:     FastValueChange:= False;
709:     CheckSelectButtonState:= False;
710:   End;
711: End;
712:
713: procedure SetPortBit(PrtAdr:word; BitNr,BitValue:byte);
714: var
715:   Operand:byte;
716: begin
717:   if (BitNr=0) then Operand:=1;
718:   if (BitNr=1) then Operand:=2;
719:   if (BitNr=2) then Operand:=4;
720:   if (BitNr=3) then Operand:=8;
721:   if (BitNr=4) then Operand:=16;
722:   if (BitNr=5) then Operand:=32;
723:   if (BitNr=6) then Operand:=64;
724:   if (BitNr=7) then Operand:=128;
725:
726:   if (BitValue=0) then mem[SEG:PrtAdr]:=mem[SEG:PrtAdr] AND (255-Operand);
727:   if (BitValue=1) then mem[SEG:PrtAdr]:=mem[SEG:PrtAdr] OR Operand;
728: end;
729:
730: Procedure ClearDisplayLine(RowNr: Byte);
731: Var
732:   Line: Byte;
733: Begin
734:   Case RowNr of
735:     1 : Line:= Line1;
736:     2 : Line:= Line2;
737:     3 : Line:= Line3;
738:     4 : Line:= Line4;
739:   End;
740:
741:   Write(Display, DisplayCtrl, Chr(Line));          {Move cursor}
742:   Write(Display, '                    ');          {Clear line}
743: End;
744:
745:
746:
747: Function LeadingZeros(s: string) : string;
748: var
749:   k : integer;
750: begin
751:   for k:=1 to (SizeOf(s)-1) do
752:     if (s[k] = ' ') then s[k] := '0';
753:
754:   LeadingZeros := s;
755: end;
756:
757:
758: (*************************************************
759:     M A G N E T O M E T E R   T R I G G I N G
760: *************************************************)
761:
762: Procedure TriggerMagnetometer;
763:
764: Begin
765:   GetMagData := False;
766:   MagStatus := 0;
767:   MagTimeOut := False;
```

```
768:   INC(Mag_Counter);
769:   If (Mag_Counter > 10000) then Mag_Counter := 1;
770:
771:   SetPortBit(P0, 6, 1);   {Trigger to magnetometer HIGH}
772:   Wait(10);
773:   SetPortBit(P0, 6 ,0);   {Trigger to magnetometer LOW}
774:
775:   mem[Seg:TMIC2]    := $47;         {Disable timer 2 interrupt while updating variable}
776:   Mag_TimeOut_Counter := MAG_READ_TIMEOUT;
777:   mem[Seg:TMIC2]    := $7;          {Enable this interrupt again}
778:
779: end;
780:
781:
782: (*************************************************
783:     M A G N E T O M E T E R   R E A D O U T
784: *************************************************)
785:
786: Procedure ReadMagData;
787:
788: Const
789:   Mag_NoPrintCmd  = $01;
790:   Mag_SoftError = $02;
791:
792: Var
793:   Reg8255_A, Reg8255_B, Reg8255_C: Byte;
794:   TimeCount: Word;
795:   Cast: LongInt;
796:   kk : Integer;
797:   k : integer;
798:   MagStatus : Byte;
799:   s : string;
800:   LCD_text : string;
801:   MagValueString : string;
802:   SerialString : string;
803:
804: Begin
805:   MagDataReady := False;
806:   MagStatus := 0;
807:
808:   If (MagIntDidHappen and not(MagTimeOut)) then
809:     begin                 {Everything ok, so .....}
810:     Reg8255_A:= Port[Base8255+0];   {read magnetometer data}
811:     Reg8255_B:= Port[Base8255+2];   {N O T E :  Address pins 0 and 1 are mixed}
812:     Reg8255_C:= Port[Base8255+1];
813:     With Mag_Values Do Begin
814:       DataValue:= (Reg8255_A And $0F);
815:       Reg8255_A:= Reg8255_A Shr 4;
816:       DataValue:= DataValue + (Reg8255_A And $0F) * 10;
817:       DataValue:= DataValue + (Reg8255_C And $0F) * 100;
818:       Reg8255_C:= Reg8255_C Shr 4;
819:       DataValue:= DataValue + (Reg8255_C And $0F) * 1000;
820:       Cast:= 10000;
821:       DataValue:= DataValue + (Reg8255_B And $0F) * Cast;
822:       Str(DataValue:5, MagValueString);
823:       MagValueString := LeadingZeros(MagValueString);
824:       LCD_text := MagValueString;
825:     end; {with}
826:     end
827:   else                    {Something is wrong, but what?}
828:     begin
829:     MagValueString := '00000';
830:     If MagTimeOut then
831:       begin                       {It was a time-out ...}
832:       LCD_text := 'PrintCmd?';
833:       MagStatus := MagStatus or Mag_NoPrintCmd;
834:       end
835:     else
836:       if not(MagIntDidHappen) then
837:       begin
838:         LCD_text := 'Soft Err ';
839:         MagStatus := MagStatus or Mag_SoftError;
840:       end;
841:     end;
842:
843:   WriteDisplayText(LCD_text, LCD_LINE_4, LCD_VAR_1_POS);
844:
845:   {Str(MagStatus:3, s);}          {'M:_xxx_mmmmm_ccccc*FF'  hvor xxx=status, mmmmm=mag data, ccccc=mag
      counter}
846:   s := Hex2Str(MagStatus);
847:   SerialString:= 'M:' + s[2] + '_' + MagValueString + '_';
848:   Str(Mag_Counter:5, s);
849:   SerialString := SerialString + LeadingZeros(s) + '*';
850:   Writeln(SerialString, MakeCSum(SerialString));
851:
852:   MagIntDidHappen := false;       {Reset flags ..}
853:   MagTimeOut := false;
854:
855: End;
856:
857:
858:
859:
860:
861: Procedure Interval_DisplayFireRoutine;
862: Var
```

```
  I: Byte;
Begin
  DisplayFire:= False;

  ClearDisplayLine(1);            {clear line}
  {WriteDisplayText('F  I  R  E', 1, 6);}
  WriteDisplayText('>', 1, 1);
  Wait(40);

  For I:= 1 To 9 Do Begin
    WriteDisplayText('  >', 1 ,(I*2));
    Wait(30);
  End;

  ClearDisplayLine(1);
  WriteDisplayText('R  E  A  D  Y', 1, 4);
  FileNumberCharCount:= 0;
  FileNumberReceived:= False;
End;


Procedure GetFileNumberFromRawData;
Var
  I,J: Byte;
  DummyData1, DummyData2, DummyFactor: Word;
Begin
  DummyData1:= 0;
  DummyData2:= 0;

  With DFS_Values Do Begin
    For I:= 0 To 3 Do Begin
      DummyData1:= (FileNumber And $000F);

      DummyFactor:= 1;
      If (I > 0) Then For J:= 1 To I Do DummyFactor:= (DummyFactor * 10);

      DummyData2:= DummyData2 + (DummyData1 * DummyFactor);

      FileNumber:= FileNumber Shr 4;
    End;
  FileNumber:= DummyData2;
  End;
End;


(*************************************************************
    D F S   F i r e   S e q u e n c e
    N o t e :   N o   e x t   t r i g g e r   o u t p u t
*************************************************************)

Procedure DFS_FireRoutine;

Var
  OffsetDelayCounter: Word;
  Str4: String[5];
  DFS_WireBlast_Ok  : Boolean;
  DFS_FFID_Ok     : Boolean;
  Gunco_TB_Ok     : Boolean;
  StatusInfo      : Byte;
  s         : String;
  Message       : String;
  Str_DFS_WB      : String;
  Str_GUNCO_TB    : String;

Begin
  TrigDelayCounter  := 0;   {This variable is incremented in timer interrupt routine.}
  DFS_WireBlast_Ok  := True;  {Assume no error conditions ... }
  DFS_FFID_Ok     := True;
  Gunco_TB_Ok     := True;
  StatusInfo      := 0;
  INC(ShotCounter);
  If (ShotCounter >= 10000) then ShotCounter := 1;


  {-----------------------
    Fire sequence
   -----------------------}

  With DFS_Values Do Begin
    SetPortBit(P0, 4, 1);               {Start DFS on}
    SetPortBit(P1, 4, 1);               {Closure to GUNCO on}
    SetPortBit(P1, 7, 1);               {LED on}
    While (TrigDelayCounter < 100) Do;          {100 ms delay}
    SetPortBit(P0, 4, 0);               {Start DFS off}
    SetPortBit(P1, 4, 0);               {Closure to GUNCO off}
{   SetPortBit(P1, 7, 0);}                {LED off}

    {---------------------------
         Check Wire blast
     ---------------------------}

    While (((mem[Seg:P0] And $01) <> $01) and DFS_WireBlast_Ok) Do
    begin                         {Wait for Wire blast from DFS}
      If (TrigDelayCounter > DFS_WIREBLAST_TIMEOUT) Then
      begin
      DFS_WireBlast_Ok := False;
```

```
959:       If DFS_IntervalTrigOn Then DisplayFire:= True;
960:       end; {if}
961:     end;   {while}
962:
963:     SetPortBit(P1, 5, 1);                {Fire to GUNCO on}
964:     OffsetDelayCounter:= TrigDelayCounter + 100;
965:     While (TrigDelayCounter < OffSetDelayCounter) Do;   {100 ms delay}
966:     SetPortBit(P1, 5, 0);                {Fire to GUNCO off}
967:
968:     {------------------------------
969:       Check Gunco Time Break (FTB)
970:      ------------------------------}
971:
972:     OffsetDelayCounter:= (TrigDelayCounter + 150);          {100+150=250 ms timeout for TB}
973:
974:     While (((mem[Seg:P0] And $02) <> $02) and Gunco_TB_Ok) Do {Wait for TimeBreak from GUNCO}
975:     begin
976:       If (TrigDelayCounter > OffsetDelayCounter) Then
977:       Begin
978:       Gunco_TB_Ok := False;
979:       If DFS_IntervalTrigOn Then DisplayFire:= True;
980:       end; {if}
981:     end;   {while}
982:
983:     SetPortBit(P0, 5, 1);                {Time break to DFS on}
984:     OffsetDelayCounter:= TrigDelayCounter + 100;      {100 ms delay}
985:     While (TrigDelayCounter < OffSetDelayCounter) Do;
986:     SetPortBit(P0, 5, 0);                {Time break to DFS off}
987:
988:     {-------------------------
989:        Check FFID from DFS
990:      -------------------------}
991:
992:     OffsetDelayCounter:= (TrigDelayCounter + DFS_FFID_TIMEOUT); { = 500ms }
993:
994:     While (Not(FileNumberReceived) and DFS_FFID_Ok) Do
995:     Begin
996:       If (TrigDelayCounter > OffsetDelayCounter) Then
997:       Begin
998:       DFS_FFID_Ok := False;
999:       If DFS_IntervalTrigOn Then DisplayFire:= True;
1000:      end;  {if}
1001:    end;    {while}
1002:
1003:    {SetPortBit(P1, 7, 0);}                {LED off}
1004:
1005:    {--------------------------------------------------
1006:       Prepare LCD and RS-232 serial output
1007:     --------------------------------------------------}
1008:
1009:    IF not(DFS_WireBlast_Ok) then       {Deal with DFS WireBlast information}
1010:    begin
1011:      StatusInfo := StatusInfo or $01;
1012:      Str_DFS_WB := 'WireB?';
1013:    end
1014:    else Str_DFS_WB :='      ';
1015:
1016:
1017:    IF not(Gunco_TB_Ok) then          {Deal with GUNCO Time Break information}
1018:    begin
1019:      StatusInfo   := StatusInfo or $02;
1020:      Str_GUNCO_TB := 'FTB? ';
1021:    end
1022:    else Str_GUNCO_TB :='Ok!  ';
1023:
1024:    IF not(DFS_FFID_Ok) then StatusInfo := StatusInfo or $04; {... and with DFS FFID info}
1025:
1026:    if FileNumberReceived then
1027:    begin
1028:      FileNumberReceived:= False;
1029:      FileNumberCharCount:= 0;
1030:      GetFileNumberFromRawData;
1031:    end
1032:    else FileNumber := 0;
1033:
1034:    Str(FileNumber:4, Str4);
1035:    Str4 := Str4 + ' ';
1036:    If not(DFS_FFID_Ok) then Str4 := 'FFID?';
1037:    WriteDisplayText(Str_GUNCO_TB, LCD_LINE_2, LCD_VAR_1_POS);
1038:    WriteDisplayText(Str4,         LCD_LINE_3, LCD_VAR_1_POS);
1039:    WriteDisplayText(Str_DFS_WB,   LCD_LINE_3, LCD_VAR_2_POS);
1040:
1041:
1042:    {Serial O/P:  'S:_x_ffff_ssss*CS'    hvor x=hex status, ffff=FFID, ssss=shot counter, CS=checksum}
1043:
1044:    s := Hex2Str(StatusInfo);
1045:    {Str(StatusInfo:3, s);}
1046:    Message := 'S:' + s[2] + '_';
1047:    Str(FileNumber:4, s);
1048:    Message := Message + LeadingZeros(s) + '_';
1049:    Str(ShotCounter:4, s);
1050:    Message := Message + LeadingZeros(s) + '*';
1051:    WriteLn(Message + MakeCSum(Message));
1052:
1053:
1054:  End; {with}
```

```
1055:
1056:    If DFS_IntervalTrigOn Then DisplayFire:= True;
1057:
1058:
1059: end;
1060:
1061:
1062:
1063: (*************************************************
1064:     O B S   F i r e   S e q u e n c e
1065: *************************************************)
1066:
1067: Procedure OBS_FireRoutine;
1068:
1069: Var
1070:   OffsetDelayCounter  : Word;
1071:   OffsetDelayCnt        : Word;
1072:   Str4          : String[5];
1073:   Gunco_TB_Ok     : Boolean;
1074:   StatusInfo      : Byte;
1075:   s           : String;
1076:   Message       : String;
1077:   Str_GUNCO_TB      : String;
1078:
1079: Begin
1080:   TrigDelayCounter    := 0;   {This variable is incremented in timer interrupt routine.}
1081:   Gunco_TB_Ok     := True;
1082:   StatusInfo      := 0;
1083:   INC(ShotCounter);
1084:   If (ShotCounter >= 10000) then ShotCounter := 1;
1085:
1086:   {Fire sequence}
1087:   With OBS_Values Do Begin
1088:
1089:   SetPortBit(P1, 4, 1);         {Closure to GUNCO on}
1090:   SetPortBit(P1, 7, 1);         {LED on}
1091:   While (TrigDelayCounter < 30) Do;   {Delay 30 (old 100) ms}
1092:   SetPortBit(P1, 4, 0);         {Closure to GUNCO off}
1093:
1094:   While (TrigDelayCounter < 60) Do;   {60 (old 600) ms delay between closure and fire}
1095:
1096:   SetPortBit(P1, 5, 1);         {Fire to GUNCO on}
1097:   While (TrigDelayCounter < 90) Do;   {30 (old 100) ms delay, then ... (new 90 old 700)}
1098:   SetPortBit(P1, 5, 0);         {.. fire to GUNCO off}
1099:
1100:   While (TrigDelayCounter < 168) Do;    {60+108=168....old... 98-  708-600=108 ms since fire command started
      ...}
1101:
1102:   SetPortBit(P1, 6, 0);         {Ext Trig on  !!! NB INVERTERT 3/2/2000}
1103:   While (TrigDelayCounter < 178) Do;    {.. then 10 ms delay}
1104:   SetPortBit(P1, 6, 1);         {Ext Trig off !!! NB INVERTERT 3/2/2000}
1105:                       {Ext trigg start 20 ms prior to FTB, which is 128 ms after FIRE start}
1106:
1107:
1108:   {------------------------------
1109:     Check Gunco Time Break (FTB)
1110:    ------------------------------}
1111:
1112:   OffsetDelayCounter:= (TrigDelayCounter + 60);         {-10 to +50 FTB search}
1113:
1114:   While (((mem[Seg:P0] And $02) <> $02) and Gunco_TB_Ok) Do {Wait for TimeBreak from GUNCO}
1115:   begin
1116:     If (TrigDelayCounter > OffsetDelayCounter) Then
1117:     Begin
1118:     Gunco_TB_Ok := False;
1119:     If OBS_IntervalTrigOn Then DisplayFire:= True;
1120:     end; {if}
1121:   end;   {while}
1122:
1123:
1124:   {-------------------------------------------------
1125:      Prepare LCD and RS-232 serial output
1126:    -------------------------------------------------}
1127:
1128:   IF not(Gunco_TB_Ok) then          {Deal with GUNCO Time Break information}
1129:   begin
1130:     StatusInfo   := StatusInfo or $02;
1131:     Str_GUNCO_TB := 'FTB? ';
1132:   end
1133:   else Str_GUNCO_TB :='Ok!  ';
1134:
1135:   Str(ShotCounter:5, Str4);
1136:   Str4 := Str4 + ' ';
1137:   WriteDisplayText(Str_GUNCO_TB, LCD_LINE_2, LCD_VAR_1_POS);
1138:   WriteDisplayText(Str4,         LCD_LINE_2, 14);
1139:
1140:   {Serial O/P:  'S:_x_ssss*CS'    hvor x=hex status, ssss=shot counter, CS=checksum}
1141:
1142:   s := Hex2Str(StatusInfo);
1143:   {Str(StatusInfo:3, s);}
1144:   Message := 'S:' + s[2] + '_';
1145:   Message := Message + LeadingZeros(Str4) + '*';
1146:   WriteLn(Message + MakeCSum(Message));
1147:
1148:   end; {with}
1149:
```

```
1150:   If NOT(TrigLockOut) then SetPortBit(P1, 7, 0);    {LED off}
1151:
1152: end;
1153:
1154:
1155:
1156:
1157: (*************************************************
1158:     O L D    O B S   F i r e   S e q u e n c e
1159: *************************************************)
1160: (*
1161: Procedure xx_OBS_FireRoutine;
1162: Var
1163:   OffsetDelayCounter  : Word;
1164:   OffsetDelayCnt      : Word;
1165: Begin
1166:   TrigDelayCounter:= 0;
1167:
1168:   {Fire sequence}
1169:   With OBS_Values Do Begin
1170:     SetPortBit(P1, 4, 1);                                  {Closure to GUNCO on}
1171:     SetPortBit(P1, 7, 1);                {LED on}
1172:     While (TrigDelayCounter < Triglength) Do;
1173:     SetPortBit(P1, 4, 0);                {Closure to GUNCO off}
1174:     While (TrigDelayCounter < ClosureFireDelay) Do;
1175:     SetPortBit(P1, 5, 1);                {Fire to GUNCO on}
1176:     SetPortBit(P1, 7, 0);                {LED off}
1177:     OffsetDelayCounter:= (TrigDelayCounter + GUNCO_FTB_TIMEOUT);
1178:
1179:     If (ExtTrigDelay < ((-1) * TrigLength)) Then Begin
1180:       If (ClosureExtTrigDelay <= (ClosureFireDelay+TrigLength)) Then Begin
1181:         While (TrigDelayCounter < ClosureExtTrigDelay) Do;
1182:         SetPortBit(P1, 6, 1);              {Trig to EPC on}
1183:         While (TrigDelayCounter < (ClosureFireDelay+TrigLength)) Do;
1184:         SetPortBit(P1, 5, 0);              {Fire to GUNCO off}
1185:         While (TrigDelayCounter < (ClosureExtTrigDelay+TrigLength)) Do;
1186:         SetPortBit(P1, 6, 0);              {Trig to EPC off}
1187:
1188:         While ((mem[Seg:P0] And $02) <> $02) Do         {Wait for TimeBreak from GUNCO}
1189:         If (TrigDelayCounter > OffsetDelayCounter) Then Begin
1190:           WriteDisplayText('No GUNCO', 3, 12);
1191:           WriteDisplayText('Time Break', 4, 11);
1192:           While (TrigDelayCounter < (OffsetDelayCounter + 500)) Do;
1193:           WriteDisplayText('        ', 3, 12);
1194:           WriteDisplayText('          ', 4, 11);
1195:           WriteOBSData(1);
1196:           If OBS_IntervalTrigOn Then DisplayFire:= True;
1197:           Exit;
1198:         End;
1199:
1200:       End Else Begin
1201:         While (TrigDelayCounter < (ClosureFireDelay+TrigLength)) Do;
1202:         SetPortBit(P1, 5, 0);              {Fire to GUNCO off}
1203:         While (TrigDelayCounter < ClosureExtTrigDelay) Do;
1204:         SetPortBit(P1, 6, 1);              {Trig to EPC on}
1205:         While (TrigDelayCounter < (ClosureExtTrigDelay+TrigLength)) Do;
1206:         SetPortBit(P1, 6, 0);              {Trig to EPC off}
1207:
1208:         While ((mem[Seg:P0] And $02) <> $02) Do       {Wait for TimeBreak from GUNCO}
1209:         If (TrigDelayCounter > OffsetDelayCounter) Then Begin
1210:           WriteDisplayText('No GUNCO', 3, 12);
1211:           WriteDisplayText('Time Break', 4, 11);
1212:           While (TrigDelayCounter < (OffsetDelayCounter + 500)) Do;
1213:           WriteDisplayText('        ', 3, 12);
1214:           WriteDisplayText('          ', 4, 11);
1215:           WriteOBSData(1);
1216:           If OBS_IntervalTrigOn Then DisplayFire:= True;
1217:           Exit;
1218:         End;
1219:
1220:       End;
1221:     End Else If (ExtTrigDelay < 0) Then Begin
1222:       If (ClosureExtTrigDelay <= (ClosureFireDelay+TrigLength)) Then Begin
1223:         If (GUNCO_TrigFireDelay <= TrigLength) Then Begin
1224:           While (TrigDelayCounter < ClosureExtTrigDelay) Do;
1225:           SetPortBit(P1, 6, 1);              {Trig to EPC on}
1226:
1227:           While ((mem[Seg:P0] And $02) <> $02) Do         {Wait for TimeBreak from GUNCO}
1228:           If (TrigDelayCounter > OffsetDelayCounter) Then Begin
1229:             SetPortBit(P1, 6, 0);            {Trig to EPC off}
1230:             SetPortBit(P1, 5, 0);            {Fire to GUNCO off}
1231:             WriteDisplayText('No GUNCO', 3, 12);
1232:             WriteDisplayText('Time Break', 4, 11);
1233:             While (TrigDelayCounter < (OffsetDelayCounter + 500)) Do;
1234:             WriteDisplayText('        ', 3, 12);
1235:             WriteDisplayText('          ', 4, 11);
1236:             WriteOBSData(1);
1237:             If OBS_IntervalTrigOn Then DisplayFire:= True;
1238:             Exit;
1239:           End;
1240:
1241:           While (TrigDelayCounter < (ClosureFireDelay+TrigLength)) Do;
1242:           SetPortBit(P1, 5, 0);              {Fire to GUNCO off}
1243:           While (TrigDelayCounter < (ClosureExtTrigDelay+TrigLength)) Do;
1244:           SetPortBit(P1, 6, 0);              {Trig to EPC off}
1245:
```

```
1246:         End Else Begin
1247:           While (TrigDelayCounter < ClosureExtTrigDelay) Do;
1248:           SetPortBit(P1, 6, 1);              {Trig to EPC on}
1249:           While (TrigDelayCounter < (ClosureFireDelay+TrigLength)) Do;
1250:           SetPortBit(P1, 5, 0);              {Fire to GUNCO off}
1251:
1252:           While ((mem[Seg:P0] And $02) <> $02) Do         {Wait for TimeBreak from GUNCO}
1253:           If (TrigDelayCounter > OffsetDelayCounter) Then Begin
1254:             SetPortBit(P1, 6, 0);            {Trig to EPC off}
1255:             WriteDisplayText('No GUNCO', 3, 12);
1256:             WriteDisplayText('Time Break', 4, 11);
1257:             While (TrigDelayCounter < (OffsetDelayCounter + 500)) Do;
1258:             WriteDisplayText('        ', 3, 12);
1259:             WriteDisplayText('          ', 4, 11);
1260:             WriteOBSData(1);
1261:             If OBS_IntervalTrigOn Then DisplayFire:= True;
1262:             Exit;
1263:           End;
1264:
1265:           While (TrigDelayCounter < (ClosureExtTrigDelay+TrigLength)) Do;
1266:           SetPortBit(P1, 6, 0);              {Trig to EPC off}
1267:
1268:         End;
1269:       End Else Begin
1270:         While (TrigDelayCounter < (ClosureFireDelay+TrigLength)) Do;
1271:         SetPortBit(P1, 5, 0);              {Fire to GUNCO off}
1272:         While (TrigDelayCounter < ClosureExtTrigDelay) Do;
1273:         SetPortBit(P1, 6, 1);              {Trig to EPC on}
1274:
1275:         While ((mem[Seg:P0] And $02) <> $02) Do         {Wait for TimeBreak from GUNCO}
1276:         If (TrigDelayCounter > OffsetDelayCounter) Then Begin
1277:           SetPortBit(P1, 6, 0);            {Trig to EPC off}
1278:           WriteDisplayText('No GUNCO', 3, 12);
1279:           WriteDisplayText('Time Break', 4, 11);
1280:           While (TrigDelayCounter < (OffsetDelayCounter + 500)) Do;
1281:           WriteDisplayText('        ', 3, 12);
1282:           WriteDisplayText('          ', 4, 11);
1283:           WriteOBSData(1);
1284:           If OBS_IntervalTrigOn Then DisplayFire:= True;
1285:           Exit;
1286:         End;
1287:
1288:         While (TrigDelayCounter < (ClosureExtTrigDelay+TrigLength)) Do;
1289:         SetPortBit(P1, 6, 0);              {Trig to EPC off}
1290:
1291:       End;
1292:     End Else Begin
1293:       If (GUNCO_TrigFireDelay <= TrigLength) Then Begin
1294:         If (ClosureExtTrigDelay <= (ClosureFireDelay+TrigLength)) Then Begin
1295:
1296:           While ((mem[Seg:P0] And $02) <> $02) Do         {Wait for TimeBreak from GUNCO}
1297:           If (TrigDelayCounter > OffsetDelayCounter) Then Begin
1298:             SetPortBit(P1, 5, 0);            {Fire to GUNCO off}
1299:             WriteDisplayText('No GUNCO', 3, 12);
1300:             WriteDisplayText('Time Break', 4, 11);
1301:             While (TrigDelayCounter < (OffsetDelayCounter + 500)) Do;
1302:             WriteDisplayText('        ', 3, 12);
1303:             WriteDisplayText('          ', 4, 11);
1304:             WriteOBSData(1);
1305:             If OBS_IntervalTrigOn Then DisplayFire:= True;
1306:             Exit;
1307:           End;
1308:
1309:           While (TrigDelayCounter < ClosureExtTrigDelay) Do;
1310:           SetPortBit(P1, 6, 1);              {Trig to EPC on}
1311:           While (TrigDelayCounter < (ClosureFireDelay+TrigLength)) Do;
1312:           SetPortBit(P1, 5, 0);              {Fire to GUNCO off}
1313:
1314:           While (TrigDelayCounter < (ClosureExtTrigDelay+TrigLength)) Do;
1315:           SetPortBit(P1, 6, 0);              {Trig to EPC off}
1316:         End Else Begin
1317:
1318:           While ((mem[Seg:P0] And $02) <> $02) Do         {Wait for TimeBreak from GUNCO}
1319:           If (TrigDelayCounter > OffsetDelayCounter) Then Begin
1320:             SetPortBit(P1, 5, 0);            {Trig to EPC off}
1321:             WriteDisplayText('No GUNCO', 3, 12);
1322:             WriteDisplayText('Time Break', 4, 11);
1323:             While (TrigDelayCounter < (OffsetDelayCounter + 500)) Do;
1324:             WriteDisplayText('        ', 3, 12);
1325:             WriteDisplayText('          ', 4, 11);
1326:             WriteOBSData(1);
1327:             If OBS_IntervalTrigOn Then DisplayFire:= True;
1328:             Exit;
1329:           End;
1330:
1331:           While (TrigDelayCounter < (ClosureFireDelay+TrigLength)) Do;
1332:           SetPortBit(P1, 5, 0);              {Fire to GUNCO off}
1333:
1334:           While (TrigDelayCounter < ClosureExtTrigDelay) Do;
1335:           SetPortBit(P1, 6, 1);              {Trig to EPC on}
1336:
1337:           While (TrigDelayCounter < (ClosureExtTrigDelay+TrigLength)) Do;
1338:           SetPortBit(P1, 6, 0);              {Trig to EPC off}
1339:
1340:         End;
1341:       End Else Begin
```

```
1342:         While (TrigDelayCounter < (ClosureFireDelay+TrigLength)) Do;
1343:         SetPortBit(P1, 5, 0);               {Fire to GUNCO off}
1344:
1345:       (*  While ((mem[Seg:P0] And $02) <> $02) Do         {Wait for TimeBreak from GUNCO}
1346:         If (TrigDelayCounter > OffsetDelayCounter) Then Begin
1347:           WriteDisplayText('No GUNCO', 3, 12);
1348:           WriteDisplayText('Time Break', 4, 11);
1349:           While (TrigDelayCounter < (OffsetDelayCounter + 500)) Do;
1350:           WriteDisplayText('        ', 3, 12);
1351:           WriteDisplayText('          ', 4, 11);
1352:           WriteOBSData(1); *)
1353: (*          If OBS_IntervalTrigOn Then DisplayFire:= True;
1354:           (*Exit;
1355:         End;*)
1356:
1357: (*        While (TrigDelayCounter < ClosureExtTrigDelay) Do;
1358:         SetPortBit(P1, 6, 1);               {Trig to EPC on}
1359:
1360:         While (TrigDelayCounter < (ClosureExtTrigDelay+TrigLength)) Do;
1361:         SetPortBit(P1, 6, 0);               {Trig to EPC off}
1362:
1363:       End;
1364:     End;
1365:
1366:     WriteOBSData(0);
1367:   End;
1368:
1369:   If OBS_IntervalTrigOn Then DisplayFire:= True;
1370: End;
1371:
1372: *)
1373:
1374:
1375: (*************************************************
1376:     C O M M A N D   P A R S E R
1377: *************************************************)
1378:
1379: Procedure CheckChar;
1380: Var
1381:   RawData: Byte;
1382:   s : string;
1383: Begin
1384:   Serial0Interrupt:= False;
1385:
1386:   {Read serialdata from receive buffer}
1387:   RawData:= mem[Seg:RxB0];
1388:
1389:   TestChar:= UpCase(Chr(RawData));
1390:
1391:   Case TestChar of
1392:
1393:     TRIG_CHAR: Begin
1394:           If (SyncUnitMode = MODE_OBS) Then
1395:           Begin
1396:             With OBS_Values Do
1397:             Begin
1398:             If (TrigMode = 1) Then
1399:             Begin
1400:               If not(TrigLockOut) then
1401:               Begin
1402:                 WriteDisplayText('Ext Fire',LCD_LINE_1,13);
1403:                 TrigLockOut := True;
1404:                 Seconds := 0;
1405:                 OBS_FireRoutine;
1406:                 WriteDisplayText('        ',LCD_LINE_1,13);
1407:               End
1408:               else
1409:                 WriteDisplayText('Locked! ',LCD_LINE_1,13);
1410:             End; {if}
1411:             End; {with}
1412:           End {if}
1413:           Else
1414:           Begin
1415:             With DFS_Values Do
1416:             Begin
1417:             If (TrigMode = 1) Then
1418:             Begin
1419:               If not(TrigLockOut) then
1420:               begin
1421:                 WriteDisplayText('Ext Fire',LCD_LINE_1,13);
1422:                 FileNumberCharCount:= 0;
1423:                 FileNumberReceived:= False;
1424:                 TrigLockOut := True;
1425:                 Seconds := 0;
1426:                 DFS_FireRoutine;
1427:                 WriteDisplayText('        ',LCD_LINE_1,13);
1428:               end
1429:               else
1430:                 WriteDisplayText('Locked! ',LCD_LINE_1,13);
1431:             End;
1432:             End;
1433:           End;
1434:         End;
1435:
1436:     'B':  If (SyncUnitMode = MODE_OBS) Then WriteOBSData(8)
1437:         Else WriteDFSData(8);
```

```
1438:
1439:     '?':  Begin
1440:           WriteLn('Program name: Sync Unit');
1441:           WriteLn('Version: '   + VERSION);
1442:           WriteLn('Date: '      + VERSION_DATE);
1443:           WriteLn('Parameters: [OBS | DFS] [Muteperiod (s)]');
1444:           If (SyncUnitMode = MODE_OBS) Then s := 'OBS'
1445:             else s := 'DFS';
1446:           WriteLn('Mode: ' + s);
1447:           WriteLn('Trig char: ' + TRIG_CHAR);
1448:           WriteLn('Trig mute period: ', TrigMutePeriod, 's');
1449:         End;
1450:
1451:     'Q',ESC:  EndTestProgram:= True;
1452:
1453:      CR,LF:   ;   {Discard}
1454:
1455:   End; {Case}
1456: End;
1457:
1458:
1459:
1460: (*********************************************************
1461:     ' M O D E   B U T T O N '   L C D   A C T I O N
1462: *********************************************************)
1463:
1464: Procedure MenuModeRoutine;
1465: var
1466:   Str4:string[4];
1467:
1468: Begin
1469:   ModeButtonInterrupt:= False;
1470:   If Not(OBS_IntervalTrigOn Or DFS_IntervalTrigOn) Then Begin
1471:     MenuMode := SUCC(MenuMode);
1472:     IF MenuMode = menuLast THEN MenuMode := SUCC(menuFirst);
1473:
1474:     If (SyncUnitMode = MODE_OBS) Then Begin
1475:       CASE menuMode OF
1476:
1477:         menuReady:          BEGIN
1478:                           With OBS_Values Do Begin
1479:                             If (TrigMode = 2) Then Begin
1480:                               ClearDisplayLine(1);
1481:                               WriteDisplayText('Interval trigg: OFF',1,1);
1482:                             End Else Begin
1483:                               MenuMode:= Succ(MenuMode);
1484:                               ClearDisplayLine(1);
1485:                               WriteDisplayText('Mode:',1,1);
1486:                               WriteDisplayText(SyncUnitMode_Values[SyncUnitMode],1,7);
1487:                             End;
1488:                           End;
1489:                         END;
1490:         menuSyncUnitMode:     Begin
1491:                           ClearDisplayLine(1);
1492:                           WriteDisplayText('Mode:',1,1);
1493:                           WriteDisplayText(SyncUnitMode_Values[SyncUnitMode],1,7);
1494:                         End;
1495:         menuExtTrigDelay:     BEGIN
1496:                           ClearDisplayLine(1);
1497:                           WriteDisplayText('Ext trig dly:',1,1);
1498:                           str(OBS_Values.ExtTrigDelay:4, Str4);
1499:                           WriteDisplayText(Str4 + 'ms',1,15);
1500:                         END;
1501: (*        menuGUNCO_TrigFireDelay:  BEGIN
1502:                           ClearDisplayLine(1);
1503:                           WriteDisplayText('GUNCO fire dly:',1,1);
1504:                           str(OBS_Values.GUNCO_TrigFireDelay:4, Str4);
1505:                           WriteDisplayText(Str4 + 'ms',1,15);
1506:                         END;
1507:         menuClosureFireDelay:   BEGIN
1508:                           ClearDisplayLine(1);
1509:                           WriteDisplayText('Clsr/fire dly:',1,1);
1510:                           str(OBS_Values.ClosureFireDelay:4, Str4);
1511:                           WriteDisplayText(Str4 + 'ms',1,15);
1512:                         END;
1513: *)
1514:         menuTrigLength:       Begin
1515:                           ClearDisplayLine(1);
1516:                           WriteDisplayText('Trig pulse len:',1,1);
1517:                           str(OBS_Values.TrigLength:4, Str4);
1518:                           WriteDisplayText(Str4 + 'ms',1,15);
1519:                         End;
1520:         menuTrigMode:         Begin
1521:                           ClearDisplayLine(1);
1522:                           WriteDisplayText('Trigger mode:',1,1);
1523:                           WriteDisplayText(TrigMode_Values[OBS_Values.TrigMode],1,15);
1524:                         End;
1525:         menuTrigInterval:     Begin
1526:                           With OBS_Values Do Begin
1527:                             If (TrigMode = 2) Then Begin
1528:                               ClearDisplayLine(1);
1529:                               WriteDisplayText('Trig intrval:',1,1);
1530:                               str((TrigInterval/1000):4:0, Str4);
1531:                               WriteDisplayText(Str4 + 's',1,15);
1532:                             End Else Begin
1533:                               MenuMode:= Succ(MenuMode);
```

```
1534:                             ClearDisplayLine(1);
1535:                             WriteDisplayText('Magnetometer:',1,1);
1536:                             If MagnetometerOn Then
1537:                               WriteDisplayText('ON',1,15)
1538:                             Else WriteDisplayText('OFF',1,15);
1539:                           End;
1540:                         End;
1541:                       End;
1542:         menuMagOn:          Begin
1543:                         ClearDisplayLine(1);
1544:                         WriteDisplayText('Magnetometer',1,1);
1545:                         If MagnetometerOn Then
1546:                           WriteDisplayText('ON',1,15)
1547:                         Else WriteDisplayText('OFF',1,15);
1548:                       End;
1549:         menuMagInterval:      Begin
1550:                         If MagnetometerOn Then Begin
1551:                           ClearDisplayLine(1);
1552:                           WriteDisplayText('Mag interval:',1,1);
1553:                           str((Mag_Values.TrigInterval/1000):4:0, Str4);
1554:                           WriteDisplayText(Str4 + 's',1,15);
1555:                         End Else Begin
1556:                           menuMode:= Succ(menuFirst);
1557:
1558:                           With OBS_Values Do Begin
1559:                             If (TrigMode = 2) Then Begin
1560:                               ClearDisplayLine(1);
1561:                               WriteDisplayText('Interval trigg: OFF',1,1);
1562:                             End Else Begin
1563:                               MenuMode:= Succ(MenuMode);
1564:                               ClearDisplayLine(1);
1565:                               WriteDisplayText('Mode:',1,1);
1566:                               WriteDisplayText(SyncUnitMode_Values[SyncUnitMode],1,7);
1567:                             End;
1568:                           End;
1569:                         End;
1570:                       End;
1571:       End;
1572:
1573:     End Else Begin
1574:       CASE menuMode OF
1575:
1576:         menuReady:          BEGIN
1577:                         With DFS_Values Do Begin
1578:                           If (TrigMode = 2) Then Begin
1579:                             ClearDisplayLine(1);
1580:                             WriteDisplayText('Interval trig: OFF',1,1);
1581:                           End Else Begin
1582:                             MenuMode:= Succ(MenuMode);
1583:                             ClearDisplayLine(1);
1584:                             WriteDisplayText('Mode: ',1,1);
1585:                             WriteDisplayText(SyncUnitMode_Values[SyncUnitMode],1,7);
1586:                           End;
1587:                         End;
1588:                       END;
1589:         menuSyncUnitMode:     Begin
1590:                         ClearDisplayLine(1);
1591:                         WriteDisplayText('Mode: ',1,1);
1592:                         WriteDisplayText(SyncUnitMode_Values[SyncUnitMode],1,7);
1593:                       End;
1594:         menuExtTrigDelay:     BEGIN
1595:                         MenuMode:=menuTrigLength;
1596:                         ClearDisplayLine(1);
1597:                         WriteDisplayText('Trig pulse: ',1,1);
1598:                         str(DFS_Values.TrigLength:4, Str4);
1599:                         WriteDisplayText(Str4 + 'ms',1,13);
1600:                       END;
1601:         menuTrigLength:       Begin
1602:                         ClearDisplayLine(1);
1603:                         WriteDisplayText('Trig pulse: ',1,1);
1604:                         str(DFS_Values.TrigLength:4, Str4);
1605:                         WriteDisplayText(Str4 + 'ms',1,13);
1606:                       End;
1607:         menuTrigMode:       Begin
1608:                         ClearDisplayLine(1);
1609:                         WriteDisplayText('Trigger mode: ',1,1);
1610:                         WriteDisplayText(TrigMode_Values[DFS_Values.TrigMode],1,15);
1611:                       End;
1612:         menuTrigInterval:     Begin
1613:                         With DFS_Values Do Begin
1614:                           If (TrigMode = 2) Then Begin
1615:                             ClearDisplayLine(1);
1616:                             WriteDisplayText('Trig interval: ',1,1);
1617:                             str((TrigInterval/1000):4:0, Str4);
1618:                             WriteDisplayText(Str4 + 's',1,15);
1619:                           End Else Begin
1620:                             MenuMode:= Succ(MenuMode);
1621:                             ClearDisplayLine(1);
1622:                             WriteDisplayText('Magnetometer: ',1,1);
1623:                             If MagnetometerOn Then
1624:                               WriteDisplayText('ON',1,15)
1625:                             Else WriteDisplayText('OFF',1,15);
1626:                           End;
1627:                         End;
1628:                       End;
1629:         menuMagOn:          Begin
```

```
1630:                         ClearDisplayLine(1);
1631:                         WriteDisplayText('Magnetometer: ',1,1);
1632:                         If MagnetometerOn Then
1633:                           WriteDisplayText('ON ',1,15)
1634:                         Else WriteDisplayText('OFF',1,15);
1635:                       End;
1636:         menuMagInterval:      Begin
1637:                         If MagnetometerOn Then Begin
1638:                           ClearDisplayLine(1);
1639:                           WriteDisplayText('Mag interval: ',1,1);
1640:                           str((Mag_Values.TrigInterval/1000):4:0, Str4);
1641:                           WriteDisplayText(Str4 + 's',1,15);
1642:                         End Else Begin
1643:                           menuMode:= Succ(menuFirst);
1644:
1645:                           With DFS_Values Do Begin
1646:                             If (TrigMode = 2) Then Begin
1647:                               ClearDisplayLine(1);
1648:                               WriteDisplayText('Interval trigg: OFF',1,1);
1649:                             End Else Begin
1650:                               MenuMode:= Succ(MenuMode);
1651:                               ClearDisplayLine(1);
1652:                               WriteDisplayText('Mode: ',1,1);
1653:                               WriteDisplayText(SyncUnitMode_Values[SyncUnitMode],1,7);
1654:                             End;
1655:                           End;
1656:                         End;
1657:                       End;
1658:
1659:       End;
1660:     End;
1661:   End;
1662:
1663:   {Enable mode button interrupt again}
1664:   mem[Seg:EXIC1]:= $7;
1665: End;
1666:
1667:
1668:
1669: (*********************************************************
1670:   ' S E L E C T   B U T T O N '   L C D   A C T I O N
1671: *********************************************************)
1672:
1673:
1674: Procedure MenuSelectRoutine;
1675: var
1676:   Line2:string[10]; Str4:string[4];
1677:   SpeedFactor: Byte;
1678: Begin
1679:   SelectButtonInterrupt:= False;
1680:   CheckSelectButtonState:= True;
1681:   If Not(FastValueChange) Then Begin
1682:     TrigDelayCounter:= 0;
1683:     SpeedFactor:= 1;
1684:   End Else SpeedFactor:= 10;
1685:
1686:   If (SyncUnitMode = MODE_OBS) Then Begin
1687:     CASE menuMode OF
1688:
1689:       menuReady:          BEGIN
1690:                         If Not(FastValueChange) Then Begin
1691:                           If OBS_IntervalTrigOn Then Begin
1692:                             OBS_IntervalTrigOn:= False;
1693:                             DisplayFire:= False;
1694:                             ClearDisplayLine(1);
1695:                             WriteDisplayText('Interval trig: OFF',1,1);
1696:                           End Else Begin
1697:                             OBS_IntervalTrigOn:= True;
1698:                             DisplayFire:= True;
1699:                             ClearDisplayLine(1);
1700:                             WriteDisplayText('R  E  A  D  Y', 1, 4);
1701:                             ShotCounter := 0;
1702:                           End;
1703:                         End;
1704:                       END;
1705:       menuSyncUnitMode:     Begin
1706:                         SyncUnitMode:= MODE_DFS;
1707:                         ClearDisplayLine(1);
1708:                         WriteDisplayText('Mode:',1,1);
1709:                         WriteDisplayText(SyncUnitMode_Values[SyncUnitMode],1,7);
1710:                       End;
1711:       menuExtTrigDelay  :     BEGIN
1712:                         With OBS_Values Do Begin
1713:                           IF ExtTrigDelay >= EXT_TRIG_DLY_MAX THEN ExtTrigDelay:= minExtTrigDelay
1714:                           Else ExtTrigDelay:= ExtTrigDelay + EXT_TRIG_DLY_STEP * SpeedFactor;
1715:                           str(ExtTrigDelay:4, Str4);
1716:                           WriteDisplayText(Str4 + 'ms',1,15);
1717:                         END;
1718:                       End;
1719: (*      menuGUNCO_TrigFireDelay:  BEGIN
1720:                         With OBS_Values Do Begin
1721:                           IF GUNCO_TrigFireDelay >= maxGUNCO_TrigFireDelay
1722:                           THEN GUNCO_TrigFireDelay:= minGUNCO_TrigFireDelay
1723:                           Else GUNCO_TrigFireDelay:= GUNCO_TrigFireDelay + stepGUNCO_TrigFireDelay;
1724:                           minExtTrigDelay:= (-10) * Trunc(GUNCO_TrigFireDelay/EXT_TRIG_DLY_STEP);
1725:                           If ExtTrigDelay < minExtTrigDelay Then ExtTrigDelay:= minExtTrigDelay;
```

```
1726:                         str(GUNCO_TrigFireDelay:4, Str4);
1727:                         WriteDisplayText(Str4 + 'ms',1,15);
1728:                       END;
1729:                     End;
1730:       menuClosureFireDelay:   BEGIN
1731:                       With OBS_Values Do Begin
1732:                         IF ClosureFireDelay >= maxClosureFireDelay THEN ClosureFireDelay:=
     minClosureFireDelay
1733:                         Else ClosureFireDelay:= ClosureFireDelay + stepClosureFireDelay * SpeedFactor;
1734:                         str(ClosureFireDelay:4, Str4);
1735:                         WriteDisplayText(Str4 + 'ms',1,15);
1736:                       END;
1737:
1738:                     End;
1739: *)
1740:       menuTrigLength:       Begin
1741:                       With OBS_Values Do Begin
1742:                         IF TrigLength >= maxTrigLength THEN TrigLength:= minTrigLength
1743:                         Else TrigLength:= TrigLength + stepTrigLength;
1744:                         MinClosureFireDelay:= 10 * Round((TrigLength+3)/10);
1745:                         If ClosureFireDelay < minClosureFireDelay
1746:                           Then ClosureFireDelay:= minClosureFireDelay;
1747:                         str(TrigLength:4, Str4);
1748:                         WriteDisplayText(Str4 + 'ms',1,15);
1749:                       End;
1750:                     End;
1751:       menuTrigMode:         Begin
1752:                       With OBS_Values Do Begin
1753:                         IF TrigMode >= 2 THEN TrigMode:= 0
1754:                         Else Inc(TrigMode);
1755:                         WriteDisplayText(TrigMode_Values[TrigMode],1,15);
1756:                       End;
1757:                     End;
1758:       menuTrigInterval:     Begin
1759:                       With OBS_Values Do Begin
1760:                         IF TrigInterval >= TRIG_INTVL_MAX THEN TrigInterval:= TRIG_INTVL_MIN
1761:                         Else TrigInterval:= TrigInterval + TRIG_INTVL_STEP;
1762:                         str((TrigInterval/1000):4:0, Str4);
1763:                         WriteDisplayText(Str4 + 's',1,15);
1764:                       End;
1765:                     End;
1766:       menuMagOn:            Begin
1767:                       MagnetometerOn:= Not MagnetometerOn;
1768:
1769:                       If MagnetoMeterON Then Begin
1770:                         WriteDisplayText('ON ',1,15);
1771:                       End Else Begin
1772:                         WriteDisplayText('OFF',1,15);
1773:                       End;
1774:                     End;
1775:       menuMagInterval:      Begin
1776:                       With Mag_Values Do Begin
1777:                         IF (TrigInterval >= MAG_INTVL_MAX) THEN TrigInterval:= MAG_INTVL_MIN
1778:                         Else TrigInterval:= TrigInterval + 1000;
1779:                         str((TrigInterval/1000):4:0, Str4);
1780:                         WriteDisplayText(Str4 + 's',1,15);
1781:                       End;
1782:                     End;
1783:     END;
1784:
1785:     With OBS_Values Do ClosureExtTrigDelay:= ClosureFireDelay+GUNCO_TrigFireDelay+ExtTrigDelay;
1786:
1787:   End Else Begin
1788:     CASE menuMode OF
1789:
1790:       menuReady:            BEGIN
1791:                       If Not(FastValueChange) Then Begin
1792:                         If DFS_IntervalTrigOn Then Begin
1793:                           DFS_IntervalTrigOn:= False;
1794:                           DisplayFire:= False;
1795:                           ClearDisplayLine(1);
1796:                           WriteDisplayText('Interval trig: OFF',1,1);
1797:                         End Else Begin
1798:                           DFS_IntervalTrigOn:= True;
1799:                           DisplayFire:= True;
1800:                           ClearDisplayLine(1);
1801:                           WriteDisplayText('R  E  A  D  Y', 1, 4);
1802:                           ShotCounter := 0;
1803:                         End;
1804:                       End;
1805:                     END;
1806:       menuSyncUnitMode:     Begin
1807:                       SyncUnitMode := MODE_OBS;
1808:                       ClearDisplayLine(1);
1809:                       WriteDisplayText('Mode: ',1,1);
1810:                       WriteDisplayText(SyncUnitMode_Values[SyncUnitMode],1,7);
1811:                     End;
1812:       menuTrigLength:       Begin
1813:                       With DFS_Values Do Begin
1814:                         IF TrigLength >= maxTrigLength THEN TrigLength:= minTrigLength
1815:                         Else TrigLength:= TrigLength + stepTrigLength;
1816:                         str(TrigLength:4, Str4);
1817:                         WriteDisplayText(Str4 + 'ms',1,13);
1818:                       End;
1819:                     End;
1820:       menuTrigMode:         Begin
```

```
1821:                       With DFS_Values Do Begin
1822:                         IF TrigMode >= 2 THEN TrigMode:= 0
1823:                         Else Inc(TrigMode);
1824:                         WriteDisplayText(TrigMode_Values[TrigMode],1,15);
1825:                       End;
1826:                     End;
1827:       menuTrigInterval:     Begin
1828:                       With DFS_Values Do Begin
1829:                         IF TrigInterval >= TRIG_INTVL_MAX THEN TrigInterval:= TRIG_INTVL_MIN
1830:                         Else TrigInterval:= TrigInterval + TRIG_INTVL_STEP;
1831:                         str((TrigInterval/1000):4:0, Str4);
1832:                         WriteDisplayText(Str4 + 's',1,15);
1833:                       End;
1834:                     End;
1835:       menuMagOn:          Begin
1836:                       MagnetometerOn:= Not MagnetometerOn;
1837:
1838:                       If MagnetoMeterON Then Begin
1839:                         WriteDisplayText('ON ',1,15);
1840:                       End Else Begin
1841:                         WriteDisplayText('OFF',1,15);
1842:                       End;
1843:                     End;
1844:       menuMagInterval:      Begin
1845:                       With Mag_Values Do Begin
1846:                         IF TrigInterval >= MAG_INTVL_MAX THEN TrigInterval:= MAG_INTVL_MIN
1847:                         Else TrigInterval:= TrigInterval + 1000;
1848:                         str((TrigInterval/1000):4:0, Str4);
1849:                         WriteDisplayText(Str4 + 's',1,15);
1850:                       End;
1851:                     End;
1852:     END;
1853:
1854:   End;
1855:
1856:   { Enable this interrupt again. }
1857:   mem[SEG:EXIC2]:=7;
1858: End;
1859:
1860:
1861:
1862:
1863: Procedure StopTimers;
1864: Begin
1865:   {Stop timer 1}
1866:   mem[Seg:TMC1]:= $0;
1867: End;
1868:
1869:
1870:
1871: (*********************************************
1872:       I N T E R R U P T S
1873: *********************************************)
1874:
1875:
1876:
1877:
1878: {=======  M a g n e t o m e t e r   P r t C m d   i n t e r r u p t =========}
1879:
1880: Procedure HandleMag; Interrupt;
1881: Begin
1882:   MagDataReady  := true;    {Read magnetometer data}
1883:   MagIntDidHappen := true;
1884:   Mag_TimeOut_Counter := 0;   {Kill time-out that's ticking ....}
1885:   Inline($0F/$92);        {Signal end of interrupt}
1886: End;
1887:
1888:
1889:
1890:
1891: {=======  S e r i a l   0   i n t e r r u p t =========}
1892:
1893: Procedure HandleSerial0; Interrupt;
1894: Begin
1895:   { Disable the interrupt to avoid other }
1896:   { interrupt caused by switch bouncing  }
1897:   mem[Seg:SRIC0]:= $47;
1898:
1899:   {Serial0Interrupt:= True;}
1900:
1901:   {Enable this interrupt again}
1902:   mem[Seg:SRIC0]:= $7;
1903:
1904:   {Signal end of interrupt}
1905:   Inline($0F/$92);
1906: End;
1907:
1908:
1909:
1910:
1911: {=======  D F S   c l o c k   i n t e r r u p t =========}
1912:
1913: Procedure HandleDFSFileNumber; Interrupt;
1914: Begin
1915:   { Disable the interrupt to avoid other }
1916:   { interrupt caused by switch bouncing  }
```

```
1917:   mem[Seg:EXIC0]:= $47;
1918:
1919:   Inc(FileNumberCharCount);
1920:
1921:   With DFS_Values Do begin
1922:     If ((mem[Seg:P0] And $04) = $04) Then FileNumber:= (FileNumber Or $8000)
1923:     Else FileNumber:= (FileNumber And ($FFFF - $8000));
1924:     If (FileNumberCharCount < 20) Then FileNumber:= FileNumber Shr 1
1925:     Else FileNumberReceived:= True;
1926:   End;
1927:
1928:   {Enable this interrupt again}
1929:   mem[Seg:EXIC0]:= $7;
1930:
1931:   {Signal end of interrupt}
1932:   Inline($0F/$92);
1933: End;
1934:
1935:
1936:
1937:
1938:
1939: {=======  P u s h   b u t t o n   i n t e r r u p t =========}
1940:
1941:
1942: Procedure HandleModeButton; Interrupt;
1943: Begin
1944:   { Disable the interrupt to avoid other }
1945:   { interrupt caused by switch bouncing  }
1946:   mem[Seg:EXIC1]:= $47;
1947:
1948:   ModeButtonInterrupt:= True;
1949:
1950:   {Signal end of interrupt}
1951:   Inline($0F/$92);
1952: End;
1953:
1954: Procedure HandleSelectButton; Interrupt;
1955: Begin
1956:   { Disable the interrupt to avoid other }
1957:   { interrupt caused by switch bouncing  }
1958:   mem[Seg:EXIC2]:= $47;
1959:
1960:   SelectButtonInterrupt:= True;
1961:
1962:   {Signal end of interrupt}
1963:   Inline($0F/$92);
1964: End;
1965:
1966:
1967:
1968: {=======  T i m e r   1   i n t e r r u p t =========}
1969:
1970:
1971: Procedure HandleTimer1; Interrupt;
1972: Begin
1973:   { Disable the interrupt to avoid other }
1974:   { interrupt caused by switch bouncing  }
1975:   mem[Seg:TMIC2]:= $47;
1976:
1977:   If TrigDelayCounter>= TimerLimit Then TrigDelayCounter:= 0;
1978:   Inc(TrigDelayCounter, TIMER_INTERVAL);
1979:
1980:   Inc(MagCounter, TIMER_INTERVAL);
1981:
1982:   If (MagCounter >= Mag_Values.TrigInterval) Then
1983:   Begin
1984:     MagCounter:= 0;
1985:     GetMagData:= True;
1986:   End;
1987:
1988:   if (Mag_TimeOut_Counter > 0) then
1989:   begin
1990:     Dec(Mag_TimeOut_Counter, TIMER_INTERVAL);;
1991:     if (Mag_TimeOut_Counter <= 0) then
1992:     begin
1993:       MagTimeOut := true;
1994:       MagDataReady:= true;
1995:     end;
1996:   end;
1997:   Inc(MilliSeconds, TIMER_INTERVAL);
1998:   If (Milliseconds >= 1000) then
1999:     begin
2000:       MilliSeconds := 0;
2001:       Inc(Seconds);
2002:       If (Seconds >= 60) then Seconds := 0;
2003:     end;
2004:
2005:   If (Seconds >= TrigMutePeriod) then
2006:   begin
2007:    TrigLockOut := False;
2008:    SetPortBit(P1, 7, 0);                  {LED off}
2009:   end;
2010:
2011:
2012:   {Enable this interrupt again}
```

```
2013:   mem[Seg:TMIC2]:= $7;
2014:
2015:   {Signal end of interrupt}
2016:   Inline($0F/$92);
2017: End;
2018:
2019:
2020:
2021:
2022:
2023: Procedure EnableMagInterrupt;
2024: Begin
2025:   {Save the current interrupt vector}
2026:   GetIntVec(2, Int2Vector);
2027:
2028:   {Set interrupt vector to point to the 'HandleMag' routine}
2029:   SetIntVec(2, @HandleMag);
2030: End;
2031:
2032: Procedure EnableSerialsInterrupt;
2033: Begin
2034:   {Save the current interrupt vector}
2035:   GetIntVec(13, Int13Vector);
2036:
2037:   {Set interrupt vector to point to the 'ReceiveInterrupt' routine}
2038:   SetIntVec(13, @HandleSerial0);
2039:
2040:   {Enable serial receive 0 interrupt}
2041:   mem[Seg:SRIC0]:= $7;
2042: End;
2043:
2044: Procedure EnableDFSFileInterrupt;
2045: Begin
2046:   {Save the current interrupt vector}
2047:   GetIntVec(24, Int24Vector);
2048:
2049:   {Set interrupt vector to point to the 'HandleDFSFileNumber' routine}
2050:   SetIntVec(24, @HandleDFSFileNumber);
2051:
2052:   {Enable interrupt from INTP0}
2053:   mem[Seg:EXIC0]:= $7;
2054: End;
2055:
2056: Procedure EnableButtonsInterrupt;
2057: begin
2058:   { Save the current interrupt vectors. }
2059:   GetIntVec(25, Int25Vector);
2060:   GetIntVec(26, Int26Vector);
2061:
2062:   { Set interrupt vectors to point to the 'HandleButtons' routines }
2063:   SetIntVec(25, @HandleModeButton);
2064:   SetIntVec(26, @HandleSelectButton);
2065:
2066:   { Enable interrupt from INTP1/INTP2. }
2067:   mem[SEG:EXIC1]:=$7;
2068:   mem[SEG:EXIC2]:=$7;
2069: end;
2070:
2071: Procedure EnableTimersInterrupt;
2072: Begin
2073:   {Save the current interrupt vector}
2074: {    GetIntVec(28, Int28Vector);}
2075:   GetIntVec(30, Int30Vector);
2076:
2077:   {Set interrupt vector to point to the 'TimerInterrupt' routine}
2078: {    SetIntVec(28, @HandleTimer0);}
2079:   SetIntVec(30, @HandleTimer1);
2080:
2081:   {Enable timer 0 interrupt}
2082: {    mem[Seg:TMIC0]:= $6;}
2083:   mem[Seg:TMIC2]:= $7;
2084: End;
2085:
2086:
2087:
2088: (*******************************************************
2089:     E X I T  P R O G R A M  R O U T I N E S
2090: *******************************************************)
2091:
2092: Procedure DisableMagInterrupt;
2093: Begin
2094:   SetIntVec(2, Int2Vector);
2095: End;
2096:
2097:
2098: Procedure DisableSerialsInterrupt;
2099: Begin
2100:   mem[Seg:SRIC0]:= $47;
2101:   SetIntVec(13, Int13Vector);
2102: End;
2103:
2104: Procedure DisableDFSFileInterrupt;
2105: Begin
2106:   mem[Seg:EXIC0]:= $47;
2107:   SetIntVec(24, Int24Vector);
2108: End;
```

```
2109:
2110: Procedure DisableButtonsInterrupt;
2111: Begin
2112:   mem[Seg:EXIC1]:= $47;
2113:   mem[Seg:EXIC2]:= $47;
2114:   SetIntVec(25, Int25Vector);
2115:   SetIntVec(26, Int26Vector);
2116: End;
2117:
2118: Procedure DisableTimersInterrupt;
2119: Begin
2120:   mem[Seg:TMIC2]:= $47;
2121:
2122:   SetIntVec(30, Int30Vector);
2123: End;
2124:
2125: Procedure ClearDisplay;
2126: Begin
2127:
2128:   Write(Display, DisplayCtrl, Chr(1));  {Clear display}
2129:   WriteDisplayText('Exit to DOS',1,2);
2130:
2131: End;
2132:
2133:
2134:
2135: (*********************************************
2136:     M A I N  P R O C E D U R E
2137: *********************************************)
2138:
2139: Var
2140:   s : string;
2141:
2142: Begin
2143:   InitPorts;
2144:
2145:   VarInitialize;
2146:   GetParameters;
2147:   EnableSerialsInterrupt;
2148:   EnableTimersInterrupt;
2149:   EnableDFSFileInterrupt;
2150:   EnableMagInterrupt;
2151:
2152:   Init8255;
2153:
2154:   InitTimers;
2155:
2156:   InitializeDisplay;
2157:
2158:   EnableButtonsInterrupt;
2159:
2160:
2161: (*  If (SyncUnitMode = MODE_DFS) Then
2162:   begin
2163:     WriteDisplayText('FFID', LCD_LINE_3, LCD_FFID_POS);
2164:     WriteDisplayText('----', LCD_LINE_4, LCD_FFID_POS);
2165:   end;
2166:   WriteDisplayText('GUNCO', LCD_LINE_3, LCD_GUNCO_POS);
2167:   WriteDisplayText('-----', LCD_LINE_4, LCD_GUNCO_POS);
2168: *)
2169:
2170:   Repeat
2171:     If Serial0Interrupt Then CheckChar;
2172:     If ModeButtonInterrupt Then MenuModeRoutine;
2173:     If SelectButtonInterrupt Then MenuSelectRoutine;
2174:     If CheckSelectButtonState Then If (TrigDelayCounter >= 1000) Then CheckSelectButton;
2175:     If FastValueChange Then MenuSelectRoutine;
2176:     If DisplayFire Then Begin
2177:       If (SyncUnitMode = MODE_OBS) And (TrigDelayCounter >= (OBS_Values.TrigInterval-1500))
2178:         Then Interval_DisplayFireRoutine;
2179:       If (SyncUnitMode = MODE_DFS) And (TrigDelayCounter >= (DFS_Values.TrigInterval-1500))
2180:         Then Interval_DisplayFireRoutine;
2181:     End;
2182:     If OBS_IntervalTrigOn Then If (TrigDelayCounter >= OBS_Values.TrigInterval) Then OBS_FireRoutine;
2183:     If DFS_IntervalTrigOn Then If (TrigDelayCounter >= DFS_Values.TrigInterval) Then DFS_FireRoutine;
2184:     If MagnetometerOn Then If GetMagData Then TriggerMagnetometer;
2185:     If MagDataReady then ReadMagData;
2186:     If ((mem[Seg:SCS0] and $10) <> 0 ) then Serial0Interrupt:= True;  {Any incoming characters?}
2187: {   Str(Seconds:2, s);
2188:     WriteDisplayText(s, LCD_LINE_4, 19);
2189: }
2190:   Until EndTestProgram;
2191:
2192:   DisableButtonsInterrupt;
2193:   DisableSerialsInterrupt;
2194:   DisableDFSFileInterrupt;
2195:   DisableTimersInterrupt;
2196:   DisableMagInterrupt;
2197:
2198:   ClearDisplay;
2199:   StopTimers;
2200: End.
2201:
```