```
{
  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     -     -
  ----------------------------------------------------------



  Hardware resources
  ----------------------------------------------------------
  PORT 0:
  P0.0 - Input:  Not used
  P0.1 - Input:  Not used
  P0.2 - Input:  Not used
  P0.3 - Output: Not used
  P0.4 - Output: Not used
  P0.5 - Output: Not used
  P0.6 - Output: Not used
  P0.7 - Output: Not used


  PORT 1:
  P1.0 - Input: Not used
  P1.1 - Interrupt input - Depth Sensor 1
  P1.2 - Interrupt input - Depth Sensor 2
  P1.3 - Input:  Switch
  P1.4 - Output: Not used
  P1.5 - Output: Not used
  P1.6 - Output: Not used
  P1.7 - Output: LED

  PORT 2, LCD.COM uses these ports:
  P2.0 - D4
  P2.1 - D5
  P2.2 - D6
  P2.3 - D7
  P2.4 - Read/Write
  P2.5 - Reset
  P2.6 - Enable display 1
  P2.7 - Enable display 2
  ----------------------------------------------------------
}


{$N-} {Ingen numerisk co-prosessor}

Program Dybde;


Uses
  Dos; {Unit som inneholder de mest brukte rutinene}


Const

  versjon                = 'Version 1.10';
  dato                   = '08. aug. 2000 ';

  constTimerInterval     = 100;  {Timer interval i ms}

  constNumbOfSamples     = 2048; {Antall samplinger ved rate 500HZ}
  constSampleInterval    = 2;    {Samplingsinterval i ms ved rate 500Hz}
  constContSampleInterval = 1000; {Samplingsinterval i ms ved rate 1Hz}

  constADCh0             = 0;    {Konstant for † velge AD kanal 0}

  constAD_Uni5Volts      = 0;    {Konstant for † velge AD input range 0-5V}
  constAD_Uni10Volts     = 2;  {Konstant for † velge AD input range 0-10V}
  constAD_Bip5Volts      = 1;  {Konstant for † velge AD input range ñ5V}
  constAD_Bip10Volts     = 3;  {Konstant for † velge AD input range ñ10V}

  Seg   =   $F000;  {Segment adresse for CPU special registers}


  SRIC0 = $FF6D;  {Receive completion interrupt request register, port 0}
  RxB0  = $FF60;  {Receive buffer}

  MD1   = $FF8A;  {Modulo/Timer register for timer 1}
  TMC1  = $FF91;  {Timer control register for timer 1}
  TMIC2 = $FF9E;  {Timer unit interrupt controll register, timer 1}


  P0    = $FF00; {Port 0 register}
  PM0   = $FF01; {Mode register for port 0}
  PMC0  = $FF02; {Mode control register for port 0}

  P1    = $FF08; {Port 1 register}
  PM1   = $FF09; {Mode register for port 1}
```

```
  PMC1  = $FF0A; {Mode control register for port 1}


  INTM  = $FF40; {External interrupt mode register}
  EXIC0 = $FF4C; {External int. request register 0}
  EXIC1 = $FF4D; {External int. request register 1}
  EXIC2 = $FF4E; {External int. request register 2}

  AD_Base     = $2B0;   {Base adresse til AD-konverter kort}

  DisplayCtrl = Chr(160);       {Control command to display}
  Line1       = 128;          {Start of line 1}
  Line2       = 192;          {Start of line 2}
  Line3       = 148;          {Start of line 3}
  Line4       = 212;          {Start of line 4}


Var
  Frequency1      : Word;
  Frequency2      : Word;

  Frequency1Set   : Word;
  Frequency2Set   : Word;

  Depth1          : Real;
  Depth2          : Real;

  Depth1Str       : String;
  Depth2Str       : String;


  TimerCounter    : Word;       {Variabel timeren ›ker}


  Int13Vector     : Pointer;    {Peker til gammel serieport 0
                                   receive interrupt rutine}

  Int24Vector     : Pointer;    {External interrupt 0}
  Int25Vector     : Pointer;    {External interrupt 1}

  Int30Vector     : Pointer;    {Peker til gammel timer 1 interrupt rutine}

  EndProgram      : Boolean;    {Flagg som indikerer om
                                 programmet skal avsluttes}

  Serial0Interrupt  : Boolean;  {Flagg som indikerer at
                                 serielle data er mottatt}

  ContinuousSampling  : Boolean;{Flagg som indikerer 1Hz
                                 kontinuerlig sampling}

  Sampling      : Boolean;      {Flagg som indikerer at sampling skal utf›res}

  Led           : Boolean;      {Flagg som indikerer at Lysdioden lyser}

  Display       : Text;

  VenteTid      : Word;         {Stopptid i 0,1 sek (30 = 3 sek) }

  CReturn       : Boolean;      {Skriver ascii #13 i tilleg til line feed}

  SampleNr      : Integer;      {Nummer i SampleList arrayet, som skal ensres}

  MaxSampleNr   : Integer;      {Det h›yeste nunneret i 'SampleList', som
                                 er lagt inn}

  DepthAverage1 : Real;         {Gjennomsnitelig dybde for kanal 1}
  DepthAverage2 : Real;         {Gjennomsnitelig dybde for kanal 2}

  DepthAverage1Str :String;     {DepthAverage1 omgjort til string}
  DepthAverage2Str :String;     {DepthAverage2 omgjort til string}

  x             : Integer;      {for loop variabel}


  SampleList1   : array[1..10] of Real;
  SampleList2   : array[1..10] of Real;


(*************************************************
      I N I T I A L I S E R I N G
*************************************************)

Procedure VarInitialize;

Begin
  {initialiserer noen variabler}
  TimerCounter        := 0;

  Frequency1          := 0;
  Frequency2          := 0;

  Depth1              := 0;
  Depth2              := 0;

  EndProgram          := False;
```

```
193:   Serial0Interrupt    := False;
194:
195:   ContinuousSampling  := False;
196:   Sampling            := False;
197:
198:   MaxSampleNr         := 0;
199:   SampleNr            := 0;
200:
201: End;
202:
203:
204: Procedure Wait(tid: word);
205: Begin
206:   VenteTid := tid;
207:   while VenteTid > 0 do;
208: End;
209:
210:
211:
212: procedure InitPorts;
213: begin
214:   (* Initialize port 0. Pins 0 to 2 will be used for  *)
215:   (* input, the rest will be used for output.         *)
216:   mem[Seg:PMC0]:= $0;
217:   mem[Seg:PM0]:= $07;
218:   mem[Seg:P0]:= mem[Seg:P0] And ($FF-$F8); {Initialize ouputs to LOW}
219:
220:   (* Initialize port 1. Pins 1 to 3 will be used for  *)
221:   (* input, the rest will be used for output.         *)
222:   mem[SEG:PMC1]:= $0;
223:   mem[SEG:PM1]:= $0E;
224:   mem[Seg:P1]:= mem[Seg:P1] And ($FF-$F0); {Initialize outputs to LOW}
225:
226:   (* Set interrupt lines to trig on rising edges. *)
227:   mem[SEG:INTM]:= $55;
228: end;
229:
230:
231: Procedure InitDisplay;
232: Begin
233:   Assign(Display, 'LPT1');    {LCD.COM redirects data sendt to LPT1 to LCD}
234:   Rewrite(Display);       {Open for output}
235:
236:   {reset display}
237:   Write(Display, DisplayCtrl, Chr(1));      {Clear and home display}
238:   Wait(10);
239:   Write(Display, DisplayCtrl, Chr(1));      {Clear and home display}
240:   Wait(20);
241:   Write(Display, DisplayCtrl, Chr(1));      {Clear and home display}
242:
243:   {Init display}
244:   Write(Display, DisplayCtrl, Chr(40));   {4 bits data, 2 lines, 5*8 dots}
245:   Write(Display, DisplayCtrl, Chr(12)); {Display on, cursor and blink off}
246:   Write(Display, DisplayCtrl, Chr(6));  {Increment cursor, no display shift}
247:   Write(Display, DisplayCtrl, Chr(1));  {Clear and home display}
248: End;
249:
250:
251:
252: Procedure InitTimers;
253: Var
254:   MD1_Number: Word;
255:
256: Begin
257:   {Initialiserer Timer 1 til 1ms interval}
258:   MD1_Number:= Round(constTimerInterval * 78.125);
259:   memw[Seg:MD1]:= MD1_Number;
260:
261:   {Starter timer}
262:   mem[Seg:TMC1]:= $C0;
263: End;
264:
265:
266:
267: (*************************************************
268:           U T I L S
269: *************************************************)
270:
271:
272: Procedure ReadInput;
273: begin
274:   If (mem[Seg:P1] And $08) = $08 then
275:     CReturn := True
276:     else CReturn := False;
277: end;
278:
279:
280: procedure SetPortBit(PrtAdr:word; BitNr,BitValue:byte);
281: var
282:   Operand: byte;
283: begin
284:   if (BitNr=0) then Operand:=1;
285:   if (BitNr=1) then Operand:=2;
286:   if (BitNr=2) then Operand:=4;
287:   if (BitNr=3) then Operand:=8;
288:   if (BitNr=4) then Operand:=16;
```

```
289:   if (BitNr=5) then Operand:=32;
290:   if (BitNr=6) then Operand:=64;
291:   if (BitNr=7) then Operand:=128;
292:
293:   if (BitValue=0) then mem[SEG:PrtAdr]:=mem[SEG:PrtAdr] AND (255-Operand);
294:   if (BitValue=1) then mem[SEG:PrtAdr]:=mem[SEG:PrtAdr] OR Operand;
295: end;
296:
297:
298: Procedure WriteDisplayText(DispText: String; RowNr, ColNr: Byte);
299: Var
300:   Line: Byte;
301: Begin
302:   Case RowNr of
303:     1 : Line:= Line1;
304:     2 : Line:= Line2;
305:     3 : Line:= Line3;
306:     4 : Line:= Line4;
307:   End;
308:
309:   Write(Display, DisplayCtrl, Chr(Line + ColNr - 1));
310:   Write(Display, DispText);
311: End;
312:
313:
314: Procedure WriteDisplayStart;
315: Begin
316: { Display lay-out:  4 x 20 char display is used.
317:
318:
319:       Pos   12345678901234567890
320:             --------------------
321:       L1:       Depth Sensor
322:       L2:           Unit
323:       L3:       Version X.XX
324:       L4:
325: }
326:
327:   WriteDisplayText('Depth Sensor',1 ,5);
328:   WriteDisplayText('Unit', 2, 9);
329:   WriteDisplayText(versjon,3, 5);
330:   Wait(20);
331:
332:   Write(Display, DisplayCtrl, Chr(1));      {Clear and home display}
333:
334: {
335:   Display lay-out:  4 x 20 char display is used.
336:
337:
338:       Pos   12345678901234567890
339:             --------------------
340:       L1:   University of Bergen
341:       L2:       Institute of
342:       L3:   Solid Earth Physics
343:       L4:       00.mnd.0000
344: }
345:
346:   WriteDisplayText('University of Bergen',  1, 1);
347:   WriteDisplayText('Institute of',          2, 5);
348:   WriteDisplayText('Solid Earth Physics',   3, 1);
349:   WriteDisplayText( dato,                   4, 5);
350:   Wait(30);                   {Wait 3000ms}
351:
352:   Write(Display, DisplayCtrl, Chr(1));      {Clear and home display}
353:
354:
355: { Display lay-out:  4 x 20 char display is used.
356:
357:
358:       Pos   12345678901234567890
359:             --------------------
360:       L1:    G U N  D E P T H S
361:       L2:   Sensor: #1      #2
362:       L3:   1s :   0,00m   0,00m
363:       L4:   10s:   0,00m   0,00m
364: }
365:
366:
367:   WriteDisplayText('G U N  D E P T H S',1, 2);
368:   WriteDisplayText('Sensor: #1      #2',2, 1);
369:   WriteDisplayText('1s :',             3, 1);
370:   WriteDisplayText('10s:',             4, 1);
371:   WriteDisplayText('m',                3, 12);
372:   WriteDisplayText('m',                3, 20);
373:   WriteDisplayText('m',                4, 12);
374:   WriteDisplayText('m',                4, 20);
375:
376:
377: end;
378:
379:
380: Procedure WriteSerial;
381: Begin
382:   if Depth1 > 0 then write((Depth1):3:2)
383:     else write('*');
384:
```

```
385:   write(' ');
386:
387:
388:   if Depth2 > 0 then write((Depth2):3:2, chr(10))
389:     else write('*', chr(10));
390:
391:   if CReturn then write(chr(13));
392:
393: End;
394:
395:
396: Procedure CheckChar; {Tester hvilken bokstav som mottas serielt}
397: Var
398:   RawData : Byte;
399:   TestChar: Char;
400:
401: Begin
402:   Serial0Interrupt:= False;
403:
404:   {Leser seriell data fra receive buffer}
405:   RawData:= mem[Seg:RxB0];
406:
407:   {Konverterer data til ASCII tekst, store bokstaver}
408:   TestChar:= UpCase(Chr(RawData));
409:
410:   {Sjekker bokstav}
411:   Case TestChar of
412:
413:     'C': ContinuousSampling := True;
414:
415:     'S': ContinuousSampling := False;
416:
417:         {Avslutter program}
418:     chr(27):  EndProgram := True;
419:
420:     'Q': WriteSerial;
421:
422:
423:   End;
424: End;
425:
426:
427: Procedure ChangeLed;
428: Begin
429:         if Led = false then
430:           begin
431:             Led := true;
432:             SetPortBit(P1, 7, 1);                {LED on}
433:           end
434:         else
435:           begin
436:             Led := false;
437:             SetPortBit(P1, 7, 0);                {LED off}
438:           end;
439: End;
440:
441:
442: (*********************************************
443:       I N T E R R U P T S
444: *********************************************)
445:
446: Procedure HandleSerial0; Interrupt;
447: Begin
448:   {Resetter interuppt flagg og disabler interrupt}
449:   mem[Seg:SRIC0]:= $47;
450:
451:   {Klarsignal for † lese data}
452:   Serial0Interrupt:= True;
453:
454:   {Enabler interrupt}
455:   mem[Seg:SRIC0]:= $7;
456:
457:   {Slutt på interrupt}
458:   Inline($0F/$92);
459: End;
460:
461:
462: Procedure HandleTimer1; Interrupt;
463: Begin
464:   {Resetter interuppt flagg og disabler interrupt}
465:   mem[Seg:TMIC2]:= $47;
466:
467:   { ker teller}
468:   Inc(TimerCounter, constTimerInterval);
469:
470:   {teller ned antall millisekunder til ventetiden
471:     (i wail prosedyren er ferdig)}
472:
473:   If VenteTid > 0 then VenteTid := VenteTid - 1;
474:
475:
476:   {Enabler interrupt}
477:   mem[Seg:TMIC2]:= $7;
478:
479:   {Slutt på interrupt}
480:   Inline($0F/$92);
```

```
481: End;
482:
483:
484: Procedure HandleCounter1; Interrupt;
485: Begin
486:   { Disable the interrupt to avoid other }
487:   { interrupt caused by switch bouncing  }
488:   mem[Seg:EXIC0]:= $47;
489:
490:   Inc(Frequency1);
491:
492:   {Enable this interrupt again}
493:   mem[Seg:EXIC0]:= $7;
494:
495:   {Signal end of interrupt}
496:   Inline($0F/$92);
497: End;
498:
499:
500: Procedure HandleCounter2; Interrupt;
501: Begin
502:   { Disable the interrupt to avoid other }
503:   { interrupt caused by switch bouncing  }
504:   mem[Seg:EXIC1]:= $47;
505:
506:   Inc(Frequency2);
507:
508:   {Enable this interrupt again}
509:   mem[Seg:EXIC1]:= $7;
510:
511:   {Signal end of interrupt}
512:   Inline($0F/$92);
513: End;
514:
515:
516: Procedure EnableSerialsInterrupt;
517: Begin
518:   {Lagrer DOS interrupt vector}
519:   GetIntVec(13, Int13Vector);
520:
521:   {Setter interrupt vector til 'HandleSerial0'}
522:   SetIntVec(13, @HandleSerial0);
523:
524:   {Enabler seriell 0 receive interrupt}
525:   mem[Seg:SRIC0]:= $7;
526: End;
527:
528:
529: Procedure EnableTimersInterrupt;
530: Begin
531:   {Lagrer DOS interrupt vector}
532:   GetIntVec(30, Int30Vector);
533:
534:   {Setter interrupt vector til 'HandleTimer0'}
535:   SetIntVec(30, @HandleTimer1);
536:
537:   {Enabler timer 1 interrupt}
538:   mem[Seg:TMIC2]:= $7;
539: End;
540:
541:
542: Procedure EnableCounter1Interrupt;
543: Begin
544:   {Save the current interrupt vector}
545:   GetIntVec(24, Int24Vector);
546:
547:   {Set interrupt vector to point to the 'HandleDFSFileNumber' routine}
548:   SetIntVec(24, @HandleCounter1);
549:
550:   {Enable interrupt from INTP0}
551:   mem[Seg:EXIC0]:= $7;
552: End;
553:
554:
555: Procedure EnableCounter2Interrupt;
556: Begin
557:   {Save the current interrupt vector}
558:   GetIntVec(25, Int25Vector);
559:
560:   {Set interrupt vector to point to the 'HandleDFSFileNumber' routine}
561:   SetIntVec(25, @HandleCounter2);
562:
563:   {Enable interrupt from INTP0}
564:   mem[Seg:EXIC1]:= $7;
565: End;
566:
567:
568:
569: (*******************************************************
570:    A V S L U T N I N G S P R O S E D Y R E R
571: *******************************************************)
572:
573: Procedure DisableSerialsInterrupt;
574: Begin
575:   {Disabler seriell 0 receive interrupt}
576:   mem[Seg:SRIC0]:= $47;
```

```
577:
578:   {Setter interruptvektor tilbake til DOS rutine}
579:   SetIntVec(13, Int13Vector);
580: End;
581:
582:
583: Procedure DisableTimersInterrupt;
584: Begin
585:   {Disabler timer 1 interrupt}
586:   mem[Seg:TMIC2]:= $47;
587:
588:   {Setter interruptvektor tilbake til DOS rutine}
589:   SetIntVec(30, Int30Vector);
590: End;
591:
592:
593: Procedure DisableCounter1Interrupt;
594: Begin
595:   mem[Seg:EXIC0]:= $47;
596:   SetIntVec(24, Int24Vector);
597: End;
598:
599:
600: Procedure DisableCounter2Interrupt;
601: Begin
602:   mem[Seg:EXIC1]:= $47;
603:   SetIntVec(25, Int25Vector);
604: End;
605:
606:
607: Procedure StopTimers;
608: Begin
609:   {Stopper timer 1}
610:   mem[Seg:TMC1]:= $0;
611: End;
612:
613:
614: (*********************************************
615:    M A I N  P R O C E D U R E
616: *********************************************)
617:
618: Begin
619:   VarInitialize;              {Initialiserer variabler}
620:
621:   InitPorts;                  {Initialiserer portene}
622:
623:   EnableSerialsInterrupt;     {Oppretter handle for interrupt
624:                                    fra seriell 0 receiver}
625:
626:   EnableTimersInterrupt;      {Oppretter handle for interrupt fra timer 1}
627:
628:   ReadInput;                  {Sjekker hvilken stilling bryteren st†r i}
629:
630:   InitTimers;                 {Setter opp timer interval}
631:
632:   InitDisplay;                {Initialiserer displayet}
633:
634:   WriteDisplayStart;          {Skriver info og fast displaytekst, ved opstart}
635:
636:   EnableCounter1Interrupt;
637:
638:   EnableCounter2Interrupt;
639:
640:
641:
642:   Write('! !', chr(10));      {viser at programmet har startet}
643:   if CReturn then write(chr(13));
644:
645:
646:   {"Synkroniserer"}
647:   mem[Seg:TMC1]:= $0;       {Stopper timer 1}
648:   mem[Seg:TMC1]:= $C0;      {Starter timer 1}
649:   Frequency1 := 0;
650:   Frequency2 := 0;
651:   TimerCounter  := 0;
652:
653:
654:   {Program loop, g†r helt til programmet avsluttes}
655:   Repeat
656:     {Sjekker mottatt bokstav ved seriell 0 receive interrupt}
657:     If Serial0Interrupt Then CheckChar;
658:
659:     {Rekner ut dybden hvert sekund}
660:     If (TimerCounter >= 1000) Then
661:       begin
662:         Frequency1Set := Frequency1;
663:         Frequency2Set := Frequency2;
664:
665:
666:         Frequency1 := 0;
667:         Frequency2 := 0;
668:
669:         TimerCounter  := 0;
670:
671:
672:         if Frequency1Set > 2000 then
```

```
673:         begin
674:           Depth1 := (Frequency1Set-2000)/100;
675:           str(Depth1:6:2, Depth1Str);
676:           WriteDisplayText(Depth1Str, 3, 6);
677:         end
678:         else
679:         begin
680:           depth1 := 0;
681:           WriteDisplayText('  ?.??', 3, 6);
682:         end;
683:
684:
685:         if Frequency2Set > 2000 then
686:         begin
687:           Depth2 := (Frequency2Set-2000)/100;
688:           str(Depth2:6:2, Depth2Str);
689:           WriteDisplayText(Depth2Str, 3, 14);
690:         end
691:         else
692:         begin
693:           Depth2 := 0;
694:           WriteDisplayText('  ?.??', 3, 14);
695:         end;
696:
697:
698:         {Rekner ut gjennomsnittsverdiene for de siste (1 -) 10 m†lingene}
699:
700:         if MaxSampleNr < 10 then MaxSampleNr := MaxSampleNr + 1;
701:
702:         if SampleNr = 10 then SampleNr := 0;
703:
704:         SampleNr := SampleNr + 1;
705:
706:
707:         SampleList1[SampleNr] := Depth1;
708:         SampleList2[SampleNr] := Depth2;
709:
710:
711:         DepthAverage1 := 0;
712:         DepthAverage2 := 0;
713:
714:         for x := 1 to MaxSampleNr do
715:         begin
716:           DepthAverage1 := DepthAverage1 + SampleList1[x];
717:           DepthAverage2 := DepthAverage2 + SampleList2[x];
718:         end;
719:
720:         DepthAverage1 := Depthaverage1 / MaxSampleNr;
721:         DepthAverage2 := Depthaverage2 / MaxSampleNr;
722:
723:
724:         if DepthAverage1 > 0 then
725:         begin
726:           str(DepthAverage1:6:2, DepthAverage1Str);
727:           WriteDisplayText(DepthAverage1Str, 4, 6);
728:         end
729:         else
730:         begin
731:           WriteDisplayText('  ?.??', 4, 6);
732:         end;
733:
734:         if DepthAverage2 > 0 then
735:         begin
736:           str(DepthAverage2:6:2, DepthAverage2Str);
737:           WriteDisplayText(DepthAverage2Str, 4, 14);
738:         end
739:         else
740:         begin
741:           WriteDisplayText('  ?.??', 4, 14);
742:         end;
743:
744:
745:
746:         if ContinuousSampling = true then WriteSerial;
747:
748:         ChangeLed;           {Sl†r LEDen Av/P†}
749:
750:       end;
751:
752:   Until EndProgram;
753:
754:   Write(Display, DisplayCtrl, Chr(1));      {Clear and home display}
755:   WriteDisplayText('Program Terminated',2,2);
756:
757:   DisableSerialsInterrupt;  {Setter handle for seriell 0 receive
758:                 interrupt tilbake til DOS}
759:
760:   DisableTimersInterrupt;     {Setter handle for timer 1 interrupt
761:                 tilbake til DOS}
762:
763:   DisableCounter1Interrupt;
764:   DisableCounter2Interrupt;
765:
766:   StopTimers;                {Stopper timer 1}
767: End.
```