Technische Universiteit Delft
Faculteit Elektrotechniek Wiskunde en Informatica
Practicumgroep, Elektrotechnisch Practicum

Elektrotechnisch Practicum

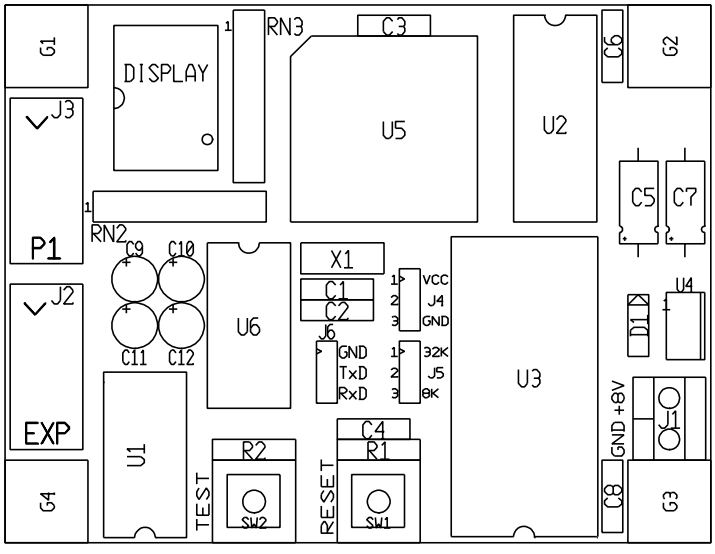
Practicumhandleiding

Computer Architectuur en Organisatie (CAO)

Code ET2043P

MIPS-practicum

Microcontroller-practicum



J.L.J.M. van Velzen
B.H.H. Juurlink
S.J. Prins

februari 2005

Technische Universiteit Delft
Faculteit Elektrotechniek Wiskunde en Informatica
Practicumgroep, Elektrotechnisch Practicum

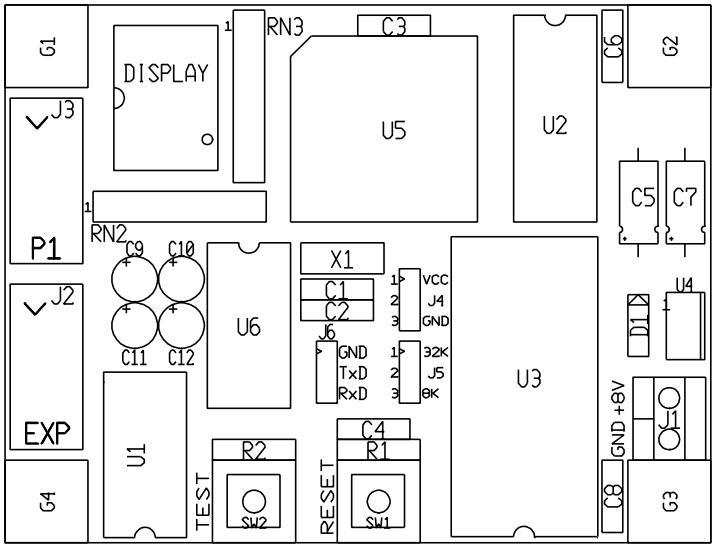
Practicumhandleiding

Computer Architectuur en Organisatie (CAO)

Code ET2043P

MIPS-practicum

Microcontroller-practicum



J.L.J.M. van Velzen
B.H.H. Juurlink
S.J. Prins

februari 2005

Inhoud

1 Algemene informatie over het practicum

1.1 Doel van dit practicum

Het doel van de cursus Computerarchitectuur en –organisatie is een inzicht te bieden in de basisstructuur van computersystemen. Daartoe bespreken we één concrete architectuur, de MIPS-machine. Aan de hand van de MIPS-assembler trachten we de relatie duidelijk te maken tussen software en hardware en de rol die de architectuur van de instructieset daarbij speelt.
Binnen het kader van het practicum krijg je de mogelijkheid wat ervaring op te doen met het programmeren in assembly-taal. Daarbij beperken we ons niet tot de MIPS-assembler, maar dien je ook een aantal opdrachten uit te voeren voor de 8051 microcontroller.
Het voornaamste doel van dit practicum is dus niet om je op te leiden tot assembler-programmeur, maar wel om je te wijzen op de gelijkenissen van de verschillende architecturen en dat een assembleertaal één van de abstractieniveaus is waarmee men computersystemen kan bekijken en analyseren.

1.2 Plaats en tijd

Tijd: ‘s morgens: 8.45 uur tot 12.30 uur
‘s middags: 13.30 uur tot 17.15 uur
Je moet je inschrijven voor het practicum; na je inschrijving weet je op welke tijd je op het practicum verwacht wordt.

Plaats: Elektrotechnisch Practicum
Kluyverweg 6
Zaal D

1.3 Inschrijving

Om deel te kunnen nemen aan het practicum moet je je online inschrijven via het inschrijvingssysteem van de Werkgroep Elektrotechnisch Practicum. Het adres hiervan is: arum.et.tudelft.nl. Zorg dat je je tijdig aanmeldt! Hoe eerder je dit doet, des te vrijer is de keuze van je practicumdag.

1.4 Practicumcoördinator

De coördinator van het practicum is J.L.J.M. van Velzen.
Adres: Kluyverweg 6, kamer B 0.47
Tel: 015-2785743
Email: j.vanvelzen@ewi.tudelft.nl

1.5 Benodigd studiemateriaal naast deze handleiding

- Hoofdstuk 3 van het boek *Computer Organization and Design* van Patterson en Hennessy (ISBN 1-55860-491-X).
- Appendix A van bovengenoemd boek. Deze kun je ook downloaden van http://www.cs.wisc.edu/~larus/SPIM/cod-appa.pdf.
- Het verdient aanbeveling de SPIM simulator ook op je eigen PC te installeren. Deze kun je downloaden van http://www.cs.wisc.edu/~larus/spim.html.
- De 8051 tutorial, die je kunt vinden op http://www.8052.com/tut8051.phtml
- De slides van de colleges Computerarchitectuur en –organisatie kunnen worden gedownload vanaf http://ce.et.tudelft.nl/~benj/Courses/CAO.

Het programma WinUptools dat op het practicum wordt gebruikt is freeware.
De benodigde documentatie en software is ook op Blackboard te vinden (code ET2043p).

1.6 Vereiste voorbereiding

Het is belangrijk om iedere practicumsessie van tevoren goed voor te bereiden. Bij elke opdracht staat precies welke voorbereiding vereist is. Lees ook de hele opdracht van tevoren goed door, zodat je weet wat je op het practicum moet gaan doen.

Bereid je je niet goed voor, dan kom je zeker weten in tijdnood. De voorbereiding is dan ook verplicht en zal door de practicumleiding worden gecontroleerd. Mocht blijken dat je je niet voldoende voorbereid hebt, dan kun je niet meedoen met het practicum!

1.7 Beoordeling

Als na het practicum is gebleken dat je over voldoende kennis en vaardigheden beschikt, wordt een voldoende (geen cijfer) toegekend. Zonder een voldoende voor het practicum is het cijfer voor het vak Computerarchitectuur en -organisatie niet geldig.

1.8 Inhoud van het practicum

1.8.1 MIPS Opdracht (eerste dagdeel):

- Een MIPS assembly programma schrijven en testen met behulp van de SPIM simulator.

1.8.2 Opdracht A (tweede dagdeel):

- Het bouwen van de 8051 practicumcomputer;
- Hardware testen met een oscilloscoop;
- Functioneel testen met een testprogramma in EPROM;
- Kennismaking met de practicumcomputer, de EPROM-programmer en de gebruikte software;
- Het vertalen van een eenvoudig programma van assemblytaal in hexadecimale machinecode;
- Het programmeren van een EPROM;
- Het gebruik van de editor en de assembler om automatisch machinecode te genereren;
- Het hergebruiken van een niet-lege EPROM.

1.8.3 Opdracht B (derde dagdeel):

- Het schrijven van een assembly-programma aan de hand van een gegeven Programma Structuur Diagram;
- Kennismaking met de ingebouwde timers van de 8051, in de eerste opdracht;
- Kennismaking met het interruptmechanisme van de 8051, in de tweede opdracht;
- Het zelf opstellen van een Programma Structuur Diagram voor de tweede opdracht.

1.8.4 Opdracht C (vierde dagdeel):

- Gebruik van de UART om invoer/uitvoer en seriële communicatie te realiseren;
- Het maken van een zender en een ontvanger die een boodschap kunnen uitwisselen;
- Het zelf opstellen van een Programma Structuur Diagrammen voor de programma's.

2 De MIPS Opdracht

Vereiste voorbereiding vóór de practicumsessie (zie ook paragraaf 1.5):
- Bestudeer Hoofdstuk 3, 4 en Appendix A van het boek *Computer Organisation and Design*
- Het verdient aanbeveling de SPIM simulator op uw PC te installeren en uit te proberen.
- Bestudeer de voorbeeldprogramma's van 2.5 en 2.6.
- Lees alle varianten behorende bij deze practicumsessie goed door.

2.1 Inhoud van deze opdracht

- Je moet voor de practicumsessie de vereiste voorbereiding hebben uitgevoerd, anders ben je uitgesloten van deelname;
- Bij deze opdracht ga je **één** van de hierna beschreven varianten uitwerken. De assistent zal je tijdens het practicum vertellen welke van de varianten je moet uitwerken;
- Je schrijft dus een MIPS assembly programma en test dit met de SPIM simulator.

Op de computers in de PC-zaal heb je de beschikking over het programma **PCSpim**. Dit programma is een simulator voor MIPS R2000/R3000 assembly programma's. Het levert een simpele debugger en een eenvoudige set operating system services. Voor het schrijven van assembly programma's voor de simulator kun je gebruik maken van **Editpad**.

Alvorens aan de opdracht te beginnen kun je het programma PCSpim eerst uitproberen. Open hiertoe het bestand 'kw.s', een MIPS assembly programma dat Fibonacci getallen uitrekent. Probeer vooral de volgende mogelijkheden eens uit:

- uitvoeren van het programma;
- invoegen van een breekpunt;
- single-steppen door het programma.

2.1.1 Registerconventies

In de opdrachten staat expliciet dat men de MIPS registerconventies moet volgen. Je dient je hier aan te houden.

2.1.2 Dynamische geheugenallocatie

De opdrachten maken gebruik van *dynamische geheugenallocatie*. In C kan men hiervoor de functie `malloc(n)` gebruiken die een pointer oplevert naar een blok geheugen van `n` bytes. In de MIPS simulator dient men hiervoor de system call `sbrk` te gebruiken, die wordt beschreven op bladzijde A-49 van het boek.

2.1.3 Pseudoinstructies

Op het college wordt voornamelijk aandacht besteed aan "echte" MIPS instructies (instructies die in de MIPS processor zijn geïmplementeerd). De assembler accepteert echter ook *pseudoinstructies*, die naar één of meer MIPS instructies worden vertaald, zoals `bgt` (branch if greater than), `move`, etc. Vaak wordt een programma leesbaarder als pseudoinstructies worden gebruikt en het verdient derhalve ook aanbeveling ze te gebruiken.

2.2 Variant I – Selection Sort

Sorteren van een rij getallen kan op vele manieren, bijvoorbeeld met behulp van het Bubble Sort algoritme (beschreven in par. 3.10 van het boek). Een ander algoritme voor het sorteren van een rij getallen is het Selection Sort algoritme. Dit werkt als volgt. Eerst zoekt dit algoritme het kleinste element in de rij en zet dit aan het begin van de rij (op plaats v[0]). Vervolgens zoekt het algoritme het kleinste element tussen de elementen v[1], v[2], ..., v[n-1] (n is de lengte van de rij) en zet dit op plaats v[1], enz. Hieronder is een implementatie van dit algoritme in C te vinden.

Vertaal de C code naar MIPS assembly code en test je programma met behulp van de SPIM simulator. Volg daarbij de MIPS register conventies (zie bijvoorbeeld Figuur A.10 op bladzijde A-23 van het boek). Denk na over de vraag welke registers door de caller gesaved moeten worden en welke door de callee. Vergeet niet om je programma van commentaar te voorzien, zodat het leesbaar is. Als je klaar bent met de opdracht, moet je aan één van de assistenten laten zien dat het werkt en een geprinte versie met naam en studienummer inleveren.

Probeer niet het hele programma in een keer te schrijven maar implementeer bijvoorbeeld eerst de `swap` functie en test deze.

```
#include <stdio.h>
#include <stdlib.h>

void swap(int v[], int i, int j)
// Actie: verwissel de elementen v[i] en v[j]
{
   int tmp;

   tmp = v[i];
   v[i] = v[j];
   v[j] = tmp;
}

int indexMinimum(int v[], int first, int last)
// Resultaat: de index van het kleinste element van de elementen
// v[first], v[first+1],…, v[last]
{
   int i, min, mini;

   mini = first;
   min = v[first];

   for (i = first+1; i <= last; i++)
      if (v[i] < min)
      {
         mini = i;
         min = v[i];
      }

   return mini;
}

void selectionSort(int a[], int length)
// Actie: sorteert de tabel a van klein naar groot
{
   int i, mini;

   for (i=0; i<length-1; i++)
   {
      mini = indexMinimum(a, i, length-1);
      swap(a, i, mini);
   }
}

void main(void)
{
   int length;
   int *a;

   printf("Hoeveel getallen wilt u sorteren? ");
   scanf("%d", &length);
   a = (int *)malloc(length * sizeof(int));
   printf("Voer %d getallen in\n", length);
   for (i=0; i<length; i++)
      scanf("%d", &a[i]);
   selectionSort(a, length);
   printf("De gesorteerde getallen zijn\n");
   for (i=0; i<length; i++)
      printf("%d ", a[i]);
}
```

2.3 Variant II – Sort by Counting

Een ander eenvoudig sorteeralgoritme is gebaseerd op de volgende pseudocode:

```
Voor elk element
      Zij h het aantal elementen kleiner dan dit element
      Zet dit element op positie h in de gesorteerde tabel
```

We noemen dit algoritme *Sort by Counting*.

Voor dit algoritme hebben we wel een extra array nodig, want anders raken we het element op positie h kwijt. Ook moeten we oppassen dat als elementen meer dan een keer voorkomen, we geen exemplaren kwijtraken. We zeggen derhalve dat a[i] kleiner is dan a[j] als a[i] < a[j] of als a[i]=a[j] en i<j. Hieronder is een implementatie van dit algoritme in C te vinden.

Vertaal de C code naar MIPS assembly code en test je programma met behulp van de SPIM simulator. Volg daarbij de MIPS register conventies (zie bijvoorbeeld Figuur A.10 op bladzijde A-23 van het boek). Denk na over de vraag welke registers door de caller gesaved moeten worden en welke door de callee. Vergeet niet om je programma van commentaar te voorzien, zodat het leesbaar is. Als je klaar bent met de opdracht, moet je aan één van de assistenten laten zien dat het werkt en een geprinte versie met naam en studienummer inleveren.

Probeer niet het hele programma in een keer te schrijven maar implementeer bijvoorbeeld eerst de `countLessThan` functie en test deze.

```
#include <stdio.h>
#include <stdlib.h>

int countLessThan(int a[], int length, int i)
// Resultaat: het aantal elementen kleiner dan a[i]
{
   int count = 0,
       j;

   for (j=0; j<length; j++)
   {
      if (a[j]<a[i] || (a[j]==a[i] && j<i))
         count++;
   }
   return count;
}

void sortByCounting(int a[], int length)
{
   int i, lessThan;
   int *b = (int *)malloc(length*sizeof(int));

   for (i=0; i<length; i++)
   {
      lessThan = countLessThan(a, length, i);
      b[lessThan] = a[i];
   }
   for (i=0; i<length; i++)
      a[i] = b[i];

   free(b);
}

void main(void)
{
   int length;
   int *a;
   printf("Hoeveel getallen wilt u sorteren? ");
   scanf("%d", &length);
   a = (int *)malloc(length * sizeof(int));
   printf("Voer %d getallen in\n", length);
   for (i=0; i<length; i++)
      scanf("%d", &a[i]);
   sortByCounting(a, length);
   printf("De gesorteerde getallen zijn\n");
   for (i=0; i<length; i++)
      printf("%d ", a[i]);
}
```

2.4 Variant III – Insertion Sort

Nog een ander sorteeralgoritme kan als volgt worden omschreven: We houden een extra tabel bij die de elementen bevat die tot nu toe zijn gesorteerd. In het begin is deze tabel leeg. Vervolgens bepalen we voor ieder element op welke positie deze dient te komen in de gesorteerde tabel en voegen hem daar tussen. We noemen dit algoritme *Insertion Sort.* Hieronder is een implementatie van dit algoritme in C te vinden.

Vertaal de C code naar MIPS assembly code en test je programma met behulp van de SPIM simulator. Volg daarbij de MIPS register conventies (zie bijvoorbeeld Figuur A.10 op bladzijde A-23 van het boek). Denk na over de vraag welke registers door de caller gesaved moeten worden en welke door de callee. Vergeet niet om je programma van commentaar te voorzien, zodat het leesbaar is. Als je klaar bent met de opdracht, moet je aan één van de assistenten laten zien dat het werkt en een geprinte versie met naam en studienummer inleveren.

Probeer niet het hele programma in een keer te schrijven maar implementeer bijvoorbeeld eerst de `insert` functie en test deze.

```
void insert(int a[], int length, int elem, int i)
// Actie: voegt element elem in op positie i van het array a
{
   int j;
   for (j=length-1; j>=i; j--)
      a[j+1] = a[j];
   a[i] = elem;
}

int binarySearch(int a[], int length, int elem)
// Resultaat: de kleinste i zodanig dat a[i] >= elem
{
   int low=-1, hi=length, mid;
   while (low < hi-1)
   {
      mid = (low + hi)/2;
      if (a[mid] >= elem)
         hi = mid;
      else if (a[mid] < elem)
         low = mid;
   }
   return hi;
}

void insertionSort(int a[], int length)
// Actie: sorteert de tabel m.b.v. het Insertion Sort algoritme
{
   int i, positie;
   int *b = (int *)malloc(length*sizeof(int));
   for (i=0; i<length; i++)
   {
      positie = binarySearch(b, i, a[i]);
      insert(b, i, a[i], positie);
   }
   for (i=0; i<length; i++)
      a[i] = b[i];
   free(b);
}

void main(void)
{
   int length;
   int *a;
   printf("Hoeveel getallen wilt u sorteren? ");
   scanf("%d", &length);
   a = (int *)malloc(length * sizeof(int));
   printf("Voer %d getallen in\n", length);
   for (i=0; i<length; i++)
      scanf("%d", &a[i]);
   insertionSort(a, length);
   printf("De gesorteerde getallen zijn\n");
   for (i=0; i<length; i++)
      printf("%d ", a[i]); }
```

2.5 MIPS VOORBEELD 1

```
# Programmer:  Mark Fienup
# Calculate Powers Example
# Algorithm:
# Main:
# maxNum = 3
# maxPower = 5
#
# CalculatePowers(maxNum, maxPower)
#
# end main
#
#
# CalculatePowers(integer numLimit, integer powerLimit)
# begin
#       integer i, j
#
#       for i := 1 to numLimit do
#            for j := 1 to powerLimit do
#                 result = Power(i, j)
#                 print i " raised to " j " power is " result
#            end for j
#       end for i
#
#
# integer Power(integer x, integer y)
# begin
#       integer result, counter
#
#       result = 1
#       for counter := 1 to y do
#            result := result * x
#       end for
#       return result
###############################################################################

                .data
maxNum:         .word 3
maxPower:       .word 5

        .text
        .globl main
main:
        lw      $a0, maxNum
        lw      $a1, maxPower
        jal     CalculatePowers # call CalculatePowers

        li      $v0, 10         # exit system call
        syscall
endMain:

        .data
str1:   .asciiz " raised to "
str2:   .asciiz " power is "
endLine: .asciiz "\n"

        .text
        .globl CalculatePowers
CalculatePowers:
#       Register Usage
#       $s0 contains i
#       $s1 contains j
#       $s2 contains numLimit
#       $s3 contains powerLimit
#       $t0 contains result

        addi    $sp, $sp, -20           # make room on stack for 5 registers
        sw      $s0, 4($sp)             # save $s0 on stack
        sw      $s1, 8($sp)             # save $s1 on stack
        sw      $s2, 12($sp)            # save $s2 on stack
        sw      $s3, 16($sp)            # save $s3 on stack
        sw      $ra, 20($sp)            # save $ra on stack

        move    $s2, $a0                # copy param. $a0 into $s2 (numLimit)
        move    $s3, $a1                # copy param. $a1 into $s3 (powerLimit)
```

```
for1:
        li      $s0, 1
forCompare1:
        ble     $s0, $s2, forBody1
        j       endFor1
forBody1:

for2:
        li      $s1, 1
forCompare2:
        ble     $s1, $s3, forBody2
        j       endFor2
forBody2:

        move    $a0, $s0                # call Power(i, j)
        move    $a1, $s1
        jal     Power                   # returns result value in $v0
        move    $t0, $v0

        li      $v0, 1                  # system call code for print_int
        move    $a0, $s0                # integer to print
        syscall

        li      $v0, 4                  # system call code for print_str
        la      $a0, str1               # addr. of string to print
        syscall

        li      $v0, 1                  # system call code for print_int
        move    $a0, $s1                # integer to print
        syscall

        li      $v0, 4                  # system call code for print_str
        la      $a0, str2               # addr. of string to print
        syscall

        li      $v0, 1                  # system call code for print_int
        move    $a0, $t0                # integer to print
        syscall

        li      $v0, 4                  # system call code for print_str
        la      $a0, endLine            # addr. of string to print
        syscall

        addi    $s1, $s1, 1
        j       forCompare2
endFor2:

        addi    $s0, $s0, 1
        j       forCompare1
endFor1:

        lw      $s0, 4($sp)             # restore $s0 from stack
        lw      $s1, 8($sp)             # restore $s1 from stack
        lw      $s2, 12($sp)            # restore $s2 from stack
        lw      $s3, 16($sp)            # restore $s3 from stack
        lw      $ra, 20($sp)            # restore $ra from stack
        addi    $sp, $sp, 20            # remove call frame from stack
        jr      $ra                     # return to calling routine
endCalculatePowers:


        .text
        .globl Power
Power:
#       Register Usage
#       $a0 contains x
#       $a1 contains y
#       $t0 contains counter
#       $v0 contains result

# Since no #s registers are used and no subprograms are called, we do not need
# to save any registers!!!

        li      $v0, 1
for3:
        li      $t0, 1
forCompare3:
```

```
        ble     $t0, $a1, forBody3
        j       endFor3
forBody3:

        mul     $v0, $v0, $a0

        addi    $t0, $t0, 1
        j       forCompare3
endFor3:

        jr      $ra
endPower:
```

2.6 MIPS VOORBEELD 2

Dit programma is te gebruiken als main voor de opdracht.

```
              .data
str1:         .asciiz "Insert the array size \n"
str2:         .asciiz "Insert the array elements,one per line  \n"
str3:         .asciiz "The sorted array is : \n"
str5:         .asciiz "\n"

              .text
main:
              la $a0,str1             # Print of str1
              li $v0,4                #
              syscall                 #

              li   $v0,5              # Get the array size(n)  and
              syscall                 # and put it in $v0
              move $s2,$v0            # $s2=n
              sll $s0,$v0,2           # $s0=n*4
              sub $sp,$sp,$s0         # This instruction creates a stack frame
                                      # large enough to contain the array
              la   $a0,str2           #
              li $v0,4                # Print of str2
              syscall                 #

              move $s1,$zero          # i=0
for_get:      bge  $s1,$s2,exit_get   # if i>=n go to exit_for_get
              sll  $t0,$s1,2          # $t0=i*4
              add  $t1,$t0,$sp        # $t1=$sp+i*4
              li   $v0,5              # Get one element of the array
              syscall                 #
              sw   $v0,0($t1)         # The element is stored
                                      # at the address $t1
              la   $a0,str5
              li   $v0,4
              syscall
              addi $s1,$s1,1          # i=i+1
              j    for_get
exit_get:     move $a0,$sp            # $a0=base address af the array
              move $a1,$s2            # $a1=size of the array
              jal  isort              # isort(a,n)
                                      # In this moment the array has been
                                      # sorted and is in the stack frame
              la   $a0,str3           # Print of str3
              li   $v0,4
              syscall

              move $s1,$zero          # i=0
for_print:    bge  $s1,$s2,exit_print # if i>=n go to exit_print
              sll  $t0,$s1,2          # $t0=i*4
              add  $t1,$sp,$t0        # $t1=address of a[i]
              lw   $a0,0($t1)         #
              li   $v0,1              # print of the element a[i]
              syscall                 #

              la   $a0,str5
              li   $v0,4
              syscall
              addi $s1,$s1,1          # i=i+1
              j    for_print
exit_print:   sub  $sp,$sp,$s0        # elimination of the stack frame

              li   $v0,10             # EXIT
              syscall                 #
```

3 De hardware van de computer

3.1 Technische specificaties

3.1.1 Belangrijke microcontrollers uit de 8051 reeks

8031 of 80C31 : 128 byte intern data RAM, geen intern ROM
8032 of 80C32 : 256 byte intern data RAM, geen intern ROM
8051 of 80C51 : 128 byte intern data RAM, 4 Kb intern ROM
8052 of 80C52 : 256 byte intern data RAM, 8 Kb intern ROM
87C51 : 128 byte intern data RAM, 4 Kb intern EPROM
87C52 : 256 byte intern data RAM, 8 Kb intern EPROM
89C51RB2 : 512 byte intern data RAM, 16 Kb intern Flash EPROM

3.1.2 Behuizing

40 pens DIL -Dual In Line- (printversie 1.1 en 1.2)
44 pens PLCC -Plastic Leadless Chip Carrier- (printversie 1.3)

3.1.3 Jumper J4

stand GND : extern programmageheugen (8031, 8032, 8051, 8052 en CMOS versies)
stand VCC : intern programmageheugen (alleen 87C51, 87C52 en 89C51RB2)

3.1.4 Jumper J5

stand 8k : extern programmageheugen van 8 Kilobyte (8192 byte) in 27C64 EPROM
stand 32k : extern programmageheugen van 32 Kilobyte (32768 byte) in 27C256 EPROM

3.1.5 Extern geheugen

Het externe programmageheugen en het externe datageheugen zijn in de practicumcomputer in dezelfde geheugenruimte gemapped, dwz. er is geen onderscheid in adressering. De standaard practicumcomputer bevat een 8 Kbyte EPROM (type 27C64). In printversie 1.2 en 1.3 kan ook een 32 Kbyte EPROM worden geplaatst (27C256).
In plaats van de 8 Kbyte EPROM kan ook een 8 Kbyte statische RAM geplaatst worden, bijv. type 6264C (8 Kbyte). In dat geval moet er een 87C51 gemonteerd worden waarin zich 4 Kbyte EPROM programmageheugen bevindt. Eventueel kan een 87C51 of een 87C52 samen met een 27C64 (of 27C256) gemonteerd worden. Er is dan slechts interne RAM beschikbaar, geen externe.

3.1.6 8 bit bidirectionele datapoort P1

Via connector P1 is poort P1 aangesloten.
Het 7-segmentdisplay met decimale punt is ook aangesloten op P1.

3.1.7 Expansieconnector EXP

Connector EXP heeft zes I/O lijnen.
P3.0 en P3.1 zijn bestemd voor seriële communicatie.
Druktoets TEST is aangesloten op P3.2.

3.1.8 Voedingsspanning

8 tot 10 Volt DC zonder koelplaat op spanningsstabilisator U4 met 8031, 8032 enz
8 tot 25 Volt DC zonder koelplaat op spanningsstabilisator U4 met 80C31, 80C32 enz.
8 tot 25 Volt DC met koelplaat op spanningsstabilisator U4 met alle typen.

3.1.9 Stroomverbruik

190 mA met een 8031, 8032, 8051 of 8052
35 mA met een 80C31, 80C32, 80C51, 80C52, 87C51, 87C52 of 89C51RB2

3.2 Memory map

3.2.1 versie 1.2 en 1.3 met 8K ROM of RAM

0000-0FFF interne ROM (87C51)
0000-1FFF externe ROM (samen met 8031 en 80C51)
externe RAM (samen met 87C51)
2000-7FFF mappings van 0000-1FFF
8000-9FFF externe ROM of RAM (niet op de print aanwezig)
A000-FFFF mappings van 8000-9FFF

3.2.2 versie 1.2 en 1.3 met 32K ROM

0000-0FFF interne ROM (87C51)
0000-7FFF externe ROM (samen met 8031 en 80C51)
8000-FFFF externe ROM (niet op de print aanwezig)

3.3 Special Function Registers

In het direct adresseerbare interne geheugen tussen 80H en FFH zijn de Special Function Registers (afkorting SFR) ondergebracht. Een deel van deze registers speelt een rol bij het uitvoeren van instructies. Dit zijn de registers A, B, DTPR, PSW en SP. De overige SFR's hebben een functie bij het gebruiken van de speciale hardware die op de processorchip is aangebracht, zoals timers, poorten en de seriële interface. Niet benutte adreslocaties bieden geen toegang tot de hardware. Het is goed nog eens te onderstrepen dat al deze Special Function Registers alleen met directe adressering benaderd kunnen worden. Met een korte beschrijving komen nu de Special Function Registers aan de orde.

3.3.1 Accumulator (register A)

De Accu (afkorting van Accumulator) is het meest gebruikte processorregister. In instructies wordt de Accu aangeduid met A.

3.3.2 Register B

Dit register wordt door de processor gebruikt bij de instructies MUL en DIV. Bij alle andere instructies is het als direct te adresseren locatie te gebruiken.

3.3.3 Program Status Word (PSW)

Het PSW bestaat uit een verzameling bits met de volgende functie:

7	6	5	4	3	2	1	0
CY	AC	F0	RS1	RS0	OV	-	P

Bit	Afkorting	Functie:
0	P	Pariteitsvlag
1	--	Door gebruiker te definiëren vlag
2	OV	Overflow flag
3	RS0	Registerbank Select control bit
4	RS1	Registerbank Select control bit
5	F0	Flag 0
6	AC	Auxiliary carry
7	CY	Carry

RS1	RS0	Bank	adressen
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

3.3.4 Data pointer (DPTR)

De DPTR is opgebouwd uit twee bytes, nl. DPL en DPH. Samen vormen zij een 16 bits register. DPTR kan als een 16 bits register benut worden in instructies die het externe geheugen adresseren, maar ook als twee 8 bits registers.

3.3.5 Port 0 t/m Port 3 (P0 t/m P3)

P0, P1, P2, en P3 zijn de 8 bits I/O-poorten van de microcontroller.

3.3.6 Serial Data Buffer (SBUF, SCON)

De seriële databuffer is opgebouwd uit twee onafhankelijke registers. Een byte dat in SBUF wordt geschreven zal worden overgebracht naar de transmit latch, van waaruit het via de seriële poort wordt verzonden. Bij het lezen van SBUF wordt een ontvangen byte uit de receive buffer gelezen. Register SCON is het besturingsregister van de seriële interface.

3.3.7 Timer T0 en T1 registers (TL0, TH0, TL1, TH1, TCON, TMOD)

De registers TL0, TH0, TL1 en TH1 vormen paarsgewijs de twee 16 bits registers van Timer T0 en T1, elk onderverdeeld in twee apart te gebruiken 8 bits registers. De registers TCON en TMOD besturen de timers.

3.3.8 Interrupt registers (IEN, IP)

Met de bits van het register IEN worden de verschillende interruptbronnen in de 8051 aan of uit gezet. Met de overeenkomstige bits van register IP kunnen de prioriteiten van de verschillende interrupts worden ingesteld.

3.3.9 Adressen van Special Function Registers

De volgende lijst bevat de adressen van de special function registers (SFR's) van de 8051. In de kolom 'Bit' staat een * als een register bit-adresseerbaar is.

Adres	Afkorting	Bit	Functie
80H	P0	*	Poort P0
81H	SP		Stackpointer
82H	DPL		Het lage byte (LSB) van DPTR
83H	DPH		Het hoge byte (MSB) van DPTR
87H	PCON		Power control
88H	TCON	*	Timer T0 en T1 control
89H	TMOD		Timer mode register
8AH	TL0		Timer T0 low (LSB)
8BH	TL1		Timer T1 low (LSB)
8CH	TH0		Timer T0 High (MSB)
8DH	TH1		Timer T1 High (MSB)
90H	P1	*	Poort P1
98H	SCON	*	Controle register van de seriële poort
99H	SBUF		Data register van de seriële poort
A0H	P2	*	Poort 2
A8H	IE	*	Interrupt enable register
B0H	P3	*	Poort 3
B8H	IP	*	Interrupt priority register
D0H	PSW	*	Program Status Word
E0H	ACC	*	Accumulator register
F0H	B	*	Hulpaccumulator voor MUL en DIV

Tabel 2.1. Geheugenadressen van de Special Function Registers in een 80C51

3.4 Input- en output-poorten

De 8051 heeft vier 8-bit parallelle poorten (P0, P1, P2 en P3). Poorten zijn verbindingen met de buitenwereld. De poorten kunnen door een programma worden benaderd via de SFR's (Special Function Registers). Door in een programma een poort te lezen wordt het niveau van de elektrische spanningen op de uitwendige pennen van een poort in een register van de microcontroller ingelezen. Daarbij wordt een spanning tussen 2,5 en 5 Volt gezien als een logisch "1" en een spanning tussen 0 en 1 Volt als een logische "0". Pen 0 van een poort komt overeen met bit 0 van het SFR van die poort, enzovoort. Via poorten kunnen gegevens opgehaald worden uit een aangesloten randapparaat.

Voorbeeld: Kopieer de 8 bits van poort P1 in de accumulator (lees P1)
machinecode : E5 90
assembly : MOV A, P1

Bij output wordt een byte in een poortadres geschreven. De spanning op een pen van de poort komt daarna overeen met het overeenkomstige bit in het poortgeheugen.

Voorbeeld: Stuur het bitpatroon in A naar poort P1 (schrijf P1)
machinecode: F5 90
assembly: MOV P1,A

Ook bit-instructies zijn toe te passen op een poort. Zo kan men een enkel bit wijzigen zonder de overige pennen te beïnvloeden.

Voorbeeld:

C2 97	CLR	P1.7	maak pen 7 van poort P1 laag
A2 B2	MOV	C,P3.2	pen 2 van poort P3 naar de carry-vlag met P3.2 is de druktoets TEST verbonden

In de practicumcomputer zijn P0 en P2 in gebruik als data- en adresbus voor de externe ROM of RAM. P1 is een zogenoemde 8 bits bidirectionele datapoort. Dit betekent dat via P1 zowel input als output mogelijk is. De inwendige outputschakeling van P1 bevat per bit een geschakelde transistor en een pull-up weerstand. Bij een 8051 is de weerstandswaarde van deze pull-up weerstand 9 Kohm, bij een 80C51 is dat 90 Kohm.
Een logisch "0" op de output is hard (een enigszins laagohmige spanningsbron). Een "1" daarentegen gedraagt zich als een hoogohmige bron (9 resp. 90 Kohm). Men dient rekening te houden met deze eigenschappen bij het aansluiten van een randapparaat. Bij input bepaalt het randapparaat het signaalniveau op de pennen van de poort. Dit is alleen mogelijk als output latch geen harde "0" bevat. Om de poortpen als input te kunnen gebruiken moet de pen hoogohmig zijn doordat er een "1" in de output latch staat.

Voorbeeld: MOV P1,#0FFH poort P1 is 8-bit inputpoort

Bij een reset wordt er automatisch FFH in deze SFR’s gezet.
In de practicumcomputer is P1 verbonden met een 7-segmentdisplay en tevens met een connector met de naam P1. Door een "1" op een pen te zetten gaat het bijbehorende segment uit. Een "0" daarentegen laat het segment branden. Het logische niveau op de pennen van de poort, en daarmee op de connector en het 7-segmentdisplay, kan dus zowel door het programma van de computer als door een randapparaat worden bepaald.

3.4.1 Connector P1

pen	bit	functie
1		GND (aarde = 0 Volt)
2		VCC (voedingsspanning = 5 Volt)
3	P1.0	i/o, segment A
4	P1.1	i/o, segment B
5	P1.2	i/o, segment C
6	P1.3	i/o, segment D
7	P1.4	i/o, segment E
8	P1.5	i/o, segment F
9	P1.6	i/o, segment G
10	P1.7	i/o, segment DP

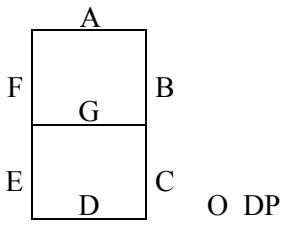


Fig. 3.1. Aansluitgegevens van de display.

Een aantal bits van P3 is toegankelijk via een connector genaamd EXP.

3.4.2 Connector EXP

pen	bit	functie
1		GND (aarde = 0 Volt)
2		VCC (voedingsspanning = 5 Volt)
3	P3.0	i/o, RxD (serial input)
4	P3.1	i/o, TxD (serial output)
5	P3.2	i/o, INT0* (interrupt input), drukschakelaar
6	P3.3	i/o, INT1* (interrupt input)
7	P3.4	i/o, T0 (timer 0 input)
8	P3.5	i/o, T1 (timer 1 input)
9		RESET output
10		ALE* output (2 MHz blokspanning)

De poortbits P3.0 t/m P3.5 kunnen ook als algemene input/output bits gebruikt worden.
Op P3.2 is een drukschakelaar met de naam TEST aangesloten. Door de schakelaar in te drukken wordt een "0" aangeboden. In rust geeft de schakelaar een "1".
Het voorgaande overzicht geeft de aansluitmogelijkheden van de computer weer. De pennummers van de twee connectors P1 en EXP zijn te vinden in de volgende figuur. Het is gebruikelijk dat pen 1 wordt gemarkeerd door een merkteken, bijvoorbeeld een pijltje of een stip.

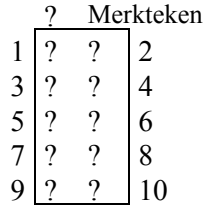


Fig. 3.2. Pennummers van de connectors.

3.5 Schemabeschrijving

3.5.1 De processor

Het hart van de schakeling is de 8051 microcontroller U5, met als klok een kristaloscillator waartoe het 12 MHz kristal X1 en de twee condensatoren C1 en C2 behoren.

3.5.2 Adres- en databus

Op poort P0 worden de laagstwaardige 8 bit van de adresbus en de databus gemultiplexed. Met behulp van het signaal ALE (address latch enable) en de 8 bit latch U2 wordt het low byte van de adresbus van de databus gescheiden. Het volledige 16 bit adres is beschikbaar aan de uitgang van U2 (low byte) en op poort P2 (high byte).

3.5.3 Programma- en datageheugen

Het signaal PSEN* (* betekent actief laag) wordt door de processor gegenereerd om instructies uit het externe programmageheugen te halen. Met het signaal RD* worden gegevens uit het externe datageheugen gehaald. Beide signalen worden gecombineerd door middel van de AND-poort configuratie bestaande uit U1A en U1B die het output-enable signaal OE* voor de geheugencomponent levert. Hierdoor bestaat geen verschil meer tussen het datageheugen en het programmageheugen.
Met jumper J4 kan worden gekozen tussen het interne programmageheugen (positie VCC) en het externe programmageheugen (positie GND). Een intern programmageheugen van 4 (8) Kbyte groot is alleen beschikbaar in een 8051 (8052) met interne ROM en een 87C51 (87C52) met interne EPROM.

3.5.4 Adresdecodering van versie 1.2 en 1.3 met J5 op positie 8K

Met de adreslijnen A0 t/m A12 worden de 8192 bytes van geheugenelement U3 (EPROM of RAM) geadresseerd. De adresdecodering is zeer eenvoudig. Het chip-enable signaal CE* van de geheugen-component wordt geleverd door adreslijn A15. Met A15 laag wordt geheugencomponent U3 geselecteerd, met A15 hoog wordt de RAM op een nog te ontwerpen uitbreidingsmodule geselecteerd. A13 en A14 zijn niet aangesloten. Op deze wijze wordt de geheugenruimte verdeeld in twee delen van 32 Kbyte waarbij alleen de eerste 8 Kbyte binnen een 32 Kbyte gedeelte kan worden benut.

3.5.5 Adresdecodering van versie 1.2 en 1.3 met J5 op positie 32K

Met de adreslijnen A0 t/m A14 worden de 32768 bytes van geheugenelement U3 (alleen EPROM) geadresseerd. De adresdecodering is zeer eenvoudig. Het chip-enable signaal CE* van de geheugen-component wordt geleverd door adreslijn A15.
Met A15 laag wordt geheugencomponent U3 geselecteerd, met A15 hoog wordt de ROM op een nog te ontwerpen uitbreidingsmodule geselecteerd. Op deze wijze wordt de geheugenruimte verdeeld in twee delen van 32 Kbyte. Externe RAM kan niet worden aangesloten.

3.5.6 Het 7-segmentdisplay

Datapoort P1 stuurt een 7-segmentdisplay. De zeven segmenten en de decimale punt zijn lichtgevende diodes die al goed zichtbaar zijn bij een stroom van 1,6 mA. De segmenten hebben een gemeenschappelijke anode-aansluiting die met VCC is verbonden. In serie met ieder segment is een weerstand opgenomen die de grootte van de doorlaatstroom bepaalt. Deze weerstanden zitten per vier in de SIL (single in line) netwerken Rn2 en Rn3. Door een 0 op een gegeven pen te zetten gaat het bijbehorende segment branden.

3.5.7 Datapoort P1 en connector P1

Op poort P1 is behalve het display ook connector P1 (J3) aangesloten. Op deze connector zijn tevens de 5 Volt voedingsspanning (VCC) en aarde (GND) beschikbaar. Een op connector P1 aangesloten schakeling kan de benodigde voeding uit de computer betrekken. De datapoort is bidirectioneel, dus voor zowel input als output geschikt.

3.5.8 Reset

De processor krijgt een reset-signaal uit een circuit bestaande uit drukschakelaar SW1 (RESET), weerstand R1, en condensator C4. NAND-poort U1D genereert een enkelvoudige resetpuls voor de processor. Ook bij het inschakelen van de voedingsspanning wordt een resetpuls gegenereerd (power on reset).

3.5.9 Connector EXP

Op connector EXP (J2) is het reset-signaal beschikbaar zodat een aangesloten schakeling tegelijk met de processor kan worden gereset. Het signaal ALE wordt gebufferd door U1C en is eveneens beschikbaar op connector EXP. De frequentie van dit signaal is 2 MHz. Voorts zijn nog 6 bits van poort P3 beschikbaar voor allerhande toepassingen. Ook VCC en GND zijn op deze connector beschikbaar om zo nodig een uitbreidingsmodule te voeden.

3.5.10 Drukschakelaar TEST

Bit 2 van poort P3 is aangesloten op drukschakelaar SW2 (TEST). Met de schakelaar in rust staat er een "1" op P3.2, ingedrukt staat er een "0" op P3.2. De schakelaar is niet ontdenderd. Men kan de schakelaar gebruiken als input-orgaan om een eenvoudige applicatie te bedienen.

3.5.11 De voeding

Het voedingsdeel van de computer bestaat uit de 5 Volt spanningsstabilisator U4. De uitgangsspanning van deze regelaar is 5 Volt (VCC) waarmee de gehele computer wordt gevoed. De ingangsspanning van U4 moet groter zijn dan 7,3 Volt. Diode D1 beschermt de computer tegen verkeerd aansluiten van de voedingsspanning. De elektrolytische condensatoren C5 en C7 zorgen voor de afvlakking van rimpels op de voeding en de stabiliteit van de voedingsspanning.

3.6 Soldeervoorschriften

Goed solderen is een kunst die moet worden geoefend. Voor het solderen gebruiken we soldeertin met een harskern. De hars laat het soldeer goed vloeien. Zorg dat de punt van de bout goed schoon is (gebruik het vochtige sponsje). Neem de soldeertin in de linkerhand en de soldeerbout in de rechter (of omgekeerd). Houd de punt van de bout gedurende niet meer dan twee seconden tegen de te solderen aansluiting. Duw gedurende niet langer dan een seconde de soldeertin tegen de aansluiting en de bout. De soldeertin moet goed vloeien. Haal eerst de soldeertin en kort daarna de bout van de verbinding. Even wachten en/of een beetje blazen.

Adem de rook van de verdampte hars niet in!

3.7 Montage

Bekijk eerst alle onderdelen en probeer ze te herkennen. Kijk goed naar de positie van ieder onderdeel alvorens het definitief vast te solderen. De bovenkant van de print (componentenzijde) bevat een witte opdruk die de positie van de componenten laat zien.
Begin de montage met de lage onderdelen en daarna de hogere. Dit maakt het mogelijk om de print plat op tafel te leggen bij het monteren van ICs en voeten. Gebruik zonodig de printklem om het wegglijden van het werkstuk te voorkomen.
Buig lange draden aan de onderkant van de print een beetje om zodat het onderdeel er niet uit kan vallen. Soldeer daarna het onderdeel vast en knip de draden af met een kniptang.

Monteer eerst de weerstanden R1 en R2 en diode D1. Raadpleeg de kleurcodetabel voor de juiste weerstandswaarden. Buig de aansluitdraden op de goede afstand 90 graden om met een platte tang of pincet en steek de component in de print. Let bij de diode op de polariteit. Het streepje is de kathode (dit een pijltje op de print, zit vast aan C5).

Neem nu de 28 pens DIL-voet voor de EPROM. Let op het merkteken van pin 1. Dit is een (zeer klein) schuin vlakje, een gaatje of een halfronde inkeping. Steek de voet in de print en soldeer pin 1

en pin 15 (diagonaal tegenover pin 1) vast. Bekijk of de voet vlak tegen de print ligt, eventueel verbeteren. Soldeer daarna alle pinnen vast. De EPROM wordt nu nog niet in de voet geplaatst.

Hierna zijn de voeten voor geïntegreerde circuits 74HCT132 (U1) en 74HCT573 (U2) aan de beurt. Let op pin 1. De ICs worden nu nog niet geplaatst.

Vervolgens de processorvoet (U5). Let op het schuine vlakje op een van de hoeken. Op de print is dit vlakje ook aangegeven. De processor wordt nog niet geplaatst.

Dan de weerstandsnetwerken Rn2 en Rn3 hebben geen oriëntatie. Rn1 bestaat niet in versie 2 en3.

Vervolgens het 7-segment display. Let op de positie van de decimale punt. Op de print staat een cirkeltje aan de kant van de decimale punt.

Hierna het kristal X1 met condensatoren C1 en C2. Het kristal wordt rechtopstaand gemonteerd. Deze componenten hebben geen oriëntatie.

Nu de elektrolytische condensatoren C5 en C7 met axiale draadeinden. Let op de polariteit. De pluskant wordt aangegeven met een cilindrische ril en de opdruk +.

Plaats nu de voedings-ontkoppelcondensatoren C3, C6, en C8 en condensator C4. Deze componenten hebben geen oriëntatie.

Nu kunnen de druktoets-schakelaars worden geplaatst. De vier pennen staan op de hoekpunten van een rechthoek zodat twee posities mogelijk zijn, het maakt niet uit welke. De schakelaar TEST heeft een lange toets en de schakelaar RESET heeft een korte toets. Knijp de pennen eerst plat met een tang voor de plaatsing in de print.

De connectors J2 en J3 worden vast gesoldeerd met de korte uiteinden in de print. Evenzo de connectors J4 en J5 waarop een jumper gezet moet worden.

Nu het aansluitblokje J1 (waarin twee schroefjes zitten) monteren. De schroefjes komen aan de buitenzijde van de print omdat hier gemakkelijk de draden van de voeding ingestoken moeten kunnen worden. Vooral bij deze component het soldeer goed laten vloeien en niet te weinig soldeer gebruiken.

De laatste component is de spanningsstabilisator U4 die met de vlakke kant aan de buitenzijde van de print wordt gesoldeerd.

Ten slotte worden de vier boutjes en moertjes gemonteerd. Deze dienen als pootjes, zodat de print niet vlak op tafel komt te liggen. Zo kan geen kortsluiting ontstaan door los liggende stukjes soldeer of afgeknipte draadeinden.

Als de montage klaar is worden de ICs in de voeten geplaatst. Bij nieuwe ICs staan de pennen meestal niet recht. Buig ze eerst recht, door het IC met de zijkant van de pootjes op tafel te drukken. Zo worden alle pennen aan een kant gelijktijdig gebogen. De pennen niet met de vingers buigen!
Een IC moet zo in de voet geplaats worden dat de uitholling op een van de korte zijden past bij de witte opdruk op de print.

De componenten U6, C9, C10, C11, C12 en J6 bevinden zich niet in het standaard bouwpakket.

kleur	cijfer	vermenigvuldiger	tolerantie
	ring 1 en 2	ring 3	ring 4
zwart	0	1x	
bruin	1	10x	1%
rood	2	100x	2%
oranje	3	1.000x	
geel	4	10.000x	
groen	5	100.000x	
blauw	6	1.000.000x	
violet	7	-	
grijs	8	-	
wit	9	-	
zilver	-	0,01x	10%
goud	-	0,1x	5%

Tabel 3.2. Kleurcode-tabel voor weerstanden.

Voorbeeld: geel/violet/rood/goud = 4700 ohm 5%
Weerstanden worden gemaakt in een aantal standaardreeksen. De E12 reeks bevat 12 verschillende standaard weerstandswaarden per decade, aangegeven door de eerste twee kleurcoderingen. De waarden zijn: 10 12 15 18 22 27 33 39 47 56 68 82.
Er komen ook weerstanden voor met een kleurring extra. De eerste drie kleurringen geven dan de standaardwaarde aan. De vierde kleurring geeft de vermenigvuldigfactor en de vijfde de tolerantie.

3.8 Aansluitgegevens

Als de stabilisator niet extra wordt gekoeld mag de voedingsspanning niet hoger zijn dan 10 Volt (20 Volt bij gebruik van een CMOS-versie). Als op de stabilisator een koelplaatje wordt gemonteerd mag de voedingsspanning wat hoger zijn, maar nooit hoger dan 25 Volt. De temperatuur van de stabilisator mag een maximum van 50 graden Celcius niet overschrijden. De opgenomen voedingsstroom van de computer is maximaal 35 mA als de I/O-pennen niet zijn belast.

De computer kan op eenvoudige wijze worden gevoed uit een netspanningsadapter die een gelijkspanning produceert van 8 Volt of hoger en die minimaal twee keer de opgenomen stroom kan leveren. Zo wordt de adapter niet maximaal belast. De connector van zo'n adapter moet worden afgeknipt, de draden gestript en vervolgens vastgezet in het aansluitblokje van de computer. Let daarbij op de polariteit.
Voor kort durende experimenten kan een 9-Volt alkaline batterijtje worden gebruikt.

Attentie:
Een wisselspanningsadapter (AC-AC adapter) of een beltrafo zijn niet zonder meer geschikt als voedingsspanningsapparaat.
Voor kort durende experimenten met een CMOS-versie controller kan een 9-Volt alkaline batterij worden gebruikt.

3.9 Het testen van de computerhardware

Na het bouwen van de computer moet de goede werking worden vastgesteld.
Controleer of jumper J4 in de positie GND staat. Jumper J5 moet in de stand 8K staan.
Sluit de voeding aan, druk op de RESET knop en controleer met een oscilloscoop het signaal op pen 10 van poort EXP. Er moet op die pen een blokgolf te zien zijn met een frequentie van 2 MHz bij een controller met een /12 klok of 4 MHz bij een /6 klok . Het 7-segmentdisplay moet geheel gedoofd zijn.
Roep uw assistent erbij als er iets niet goed is.

3.10 Functionele test met een testprogramma

Er is een testprogramma voorhanden waarmee verschillende functies van de computer kunnen worden getest. Dit programma heet DT_IO en het zit een EPROM.
Zet de voeding uit, monteer deze EPROM en zet de voeding weer aan.

Start het programma door op de knop RESET te drukken.
Alle segmenten gaan nu een voor een aan en uit. Als er een of meerdere segmenten niet branden of als er segmenten gelijktijdig branden is er waarschijnlijk een soldeerfout gemaakt. Loop alle soldeerpunten nog even na en controleer of niet te weinig soldeer is gebruikt. Als geen enkel segment brandt roep dan uw assistent erbij.

Terwijl de segmenten een voor een aan en uit gaan kan met de oscilloscoop het signaal op de pennen van poort P1 worden bekeken. Merk op dat het signaal gedurende een achtste van de tijd laag is, en wel zolang het corresponderende segment aan is. Bij een logische 0 op een pen van P1 is een het aangesloten segment aan, bij een logische 1 dus uit.

Door op de knop TEST te drukken kunnen verschillende hardware functies van de computer worden getest. Het volgordecijfer van de test wordt op het 7-segment display getoond. Als de test slaagt gaat de punt aan. Als de punt knippert wordt de test herhaald tot de knop weer wordt ingedrukt. Na de laatste test komt het programma in een eindeloze lus. De volgende lijst geeft aan wat er wordt getest.

cijfer	**functie**
1	schakelaartest; de punt moet aan zijn
2	op connector EXP pennen 3 t/m 8 is een blokvormig signaal te zien met de oscilloscoop; de punt moet knipperen
3	timer 0 test; de punt moet knipperen
4	timer 1 test; de punt moet knipperen
5	geheugentest vanaf 100H-1FFFH; de punt moet aan zijn

Roep uw assistent erbij als er iets niet klopt.

Onderdelen en kostprijs van de 8051 computer

AANTAL	CODE	COMPONENT	WAARDE	AFMETING	STUKPRIJS	TOTAAL
2	Rn2,Rn3	WEERSTAND NETWERK	4 X 2K2	SIL8	0,20	0,40
1	U5	CONTROLLER 80C51/87C51		PLCC44	10,00	10,00
1	U4	SPANNINGSREGELAAR 7805	5 VOLT	TO220	0,34	0,34
1	D1	DIODE 1N4007	1A	SOD81	0,04	0,04
1	U3	EPROM 27C64	8K	DIL28	2,10	2,10
1	U1	DECODER 74HCT132		DIL14	0,30	0,30
1	U2	LATCH 74HCT573		DIL20	0,40	0,40
2	C1,C2	CONDENSATOR	22 pF	RM 5 mm	0,10	0,20
4	C3,C4,C6,C8	CONDENSATOR	100 nF	RM 5 mm	0,12	0,48
1	DISPLAY1	7 SEGM. DISPLAY HDN1131-0			1,20	1,20
1	C5	CONDENSATOR	100 uF	AXIAAL	0,20	0,20
1	C7	CONDENSATOR	10 uF	AXIAAL	0,13	0,13
1	R2	WEERSTAND 0,25 W	10K	SFR 25	0,02	0,02
1	R1	WEERSTAND 0,25 W	220K	SFR 25	0,02	0,02
1	SW2	DRUKTOETS		LANG	0,40	0,40
1	SW1	DRUKTOETS		KORT	0,40	0,40
1	X1	KRISTAL	12 MHz		0,50	0,50
1	(U1)	DIL-voet		DIL14	0,16	0,16
1	(U2)	DIL-voet		DIL20	0,22	0,22
1	(U3)	DIL-VOET		DIL28	0,35	0,35
1	(U5)	PLCC-voet		PLCC44	1,35	1,35
1	J1	AANSLUIT-BLOKJE		2 PENS	0,37	0,37
2	J4,J5	CONNECTORPENNEN		1X3	0,06	0,12
2	J2,J3	CONNECTORPENNEN		2X5	0,20	0,40
1		PRINT		86x66 mm	4,30	4,30
2		JUMPER			0,20	0,40
4		BOUT+MOER+RING		15x3 mm	0,05	0,20
1 *	U6	INTERFACE MAX232		DIL16	3,40	0,00
4 *	C9,C10,C11,C12	CONDENSATOR	10 uF	RM 5 mm	0,50	0,00
1 *	J6	CONNECTORPENNEN		1X3	0,12	0,00
		TOTAAL				€25,00

De componenten gemerkt met * behoren niet tot het bouwpakket. Ze kunnen eventueel later worden toegevoegd als er behoefte is aan een RS232 interface.

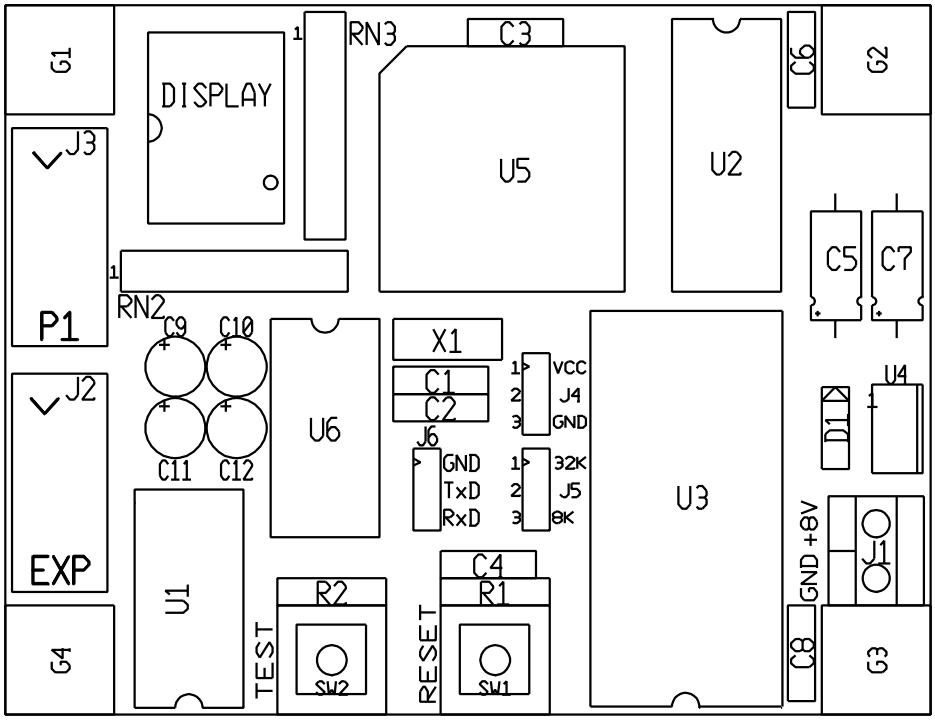


Fig 2.3. De componentenopstelling van de computer (versie 1.3)

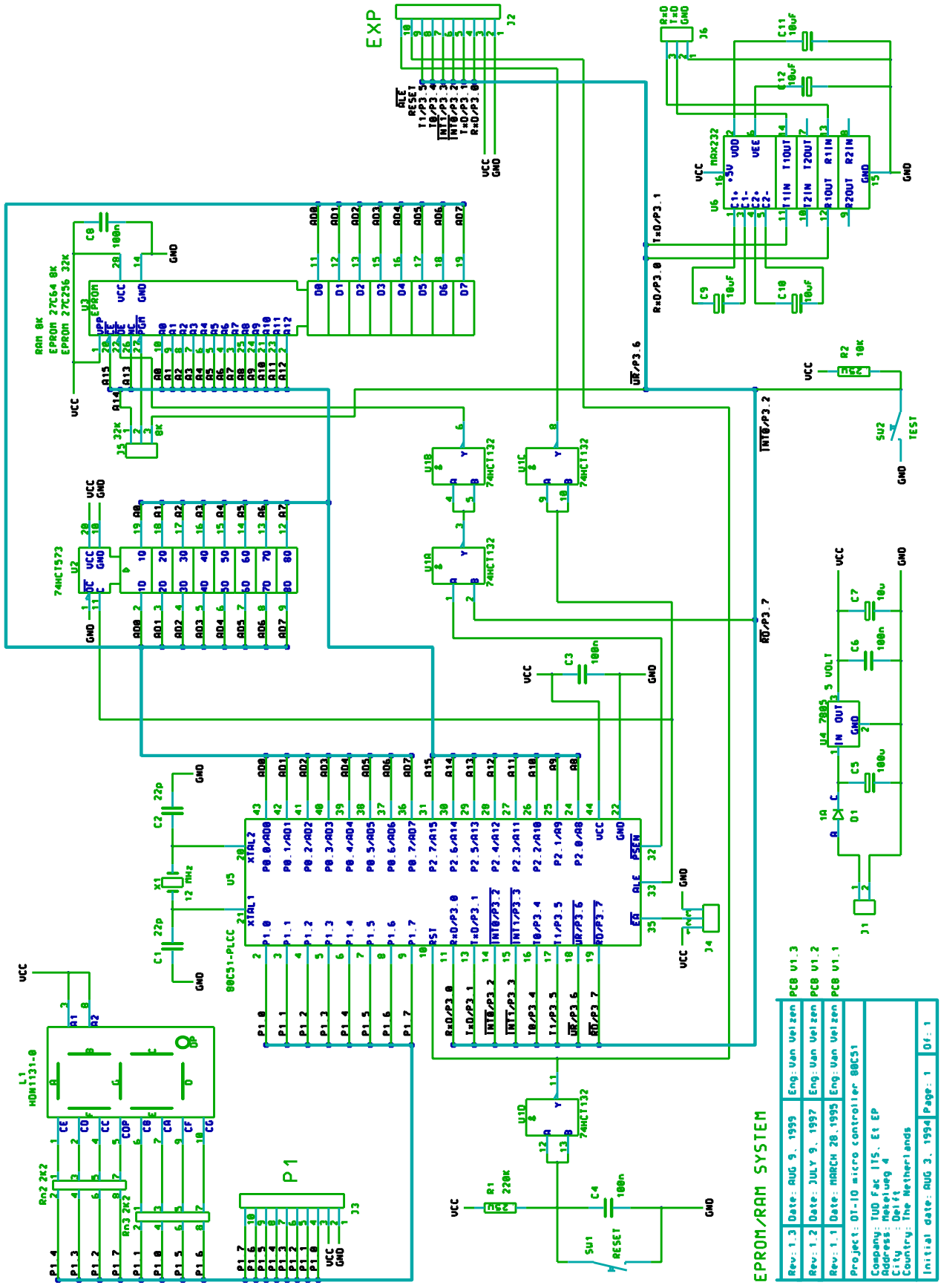


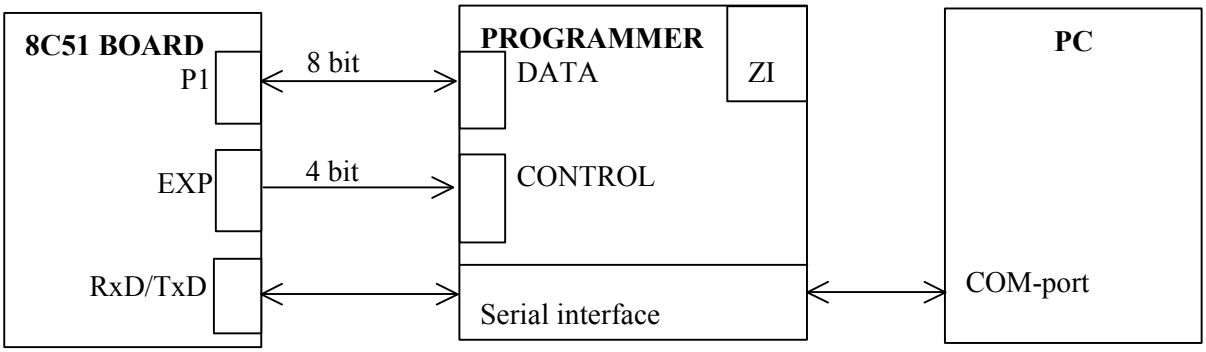
Fig. 2.4 Het schema van de computer (versie 1.3).

4 Het programmeren van een EPROM

De meeste EPROM programmeerapparaten verwerken binaire bestanden. Een dergelijk bestand bestaat uit de machinecode en dataconstanten die samen een programma vormen.

4.1 De apparatuur

Op het practicum is op iedere tafel een EPROM-programmeerapparaat aanwezig. Deze 'programmer' bestaat uit een programmeermodule waarmee het feitelijke programmeren van EPROMs wordt uitgevoerd. Op die module zit een 'ZIF-socket' (Zero Insertion Force). Een dergelijke IC-voet met kleminrichting vergemakkelijkt het wisselen van EPROMs. De programmeermodule wordt bestuurd door een microcontrollerboard. Het besturingsprogramma bevindt zich in een EPROM. Het wordt actief na een RESET van de computer. De computer krijgt commando's van de PC via een seriële verbinding. Ook de (te programmeren) inhoud van de EPROM wordt via de seriële verbinding getransporteerd. De hiervoor benodigde serie-interface tussen de PC en de besturingscomputer is op de programmeermodule aangebracht. De volgende figuur laat de opstelling zien.



Figuur 3.1. Opstelling van de PC en de EPROM

Fig. 4.1. De EPROM-programmer

4.2 Het programmeer-programma

De voorbereidingen voor het feitelijke programmeren gebeuren op de PC. Het programmeermenu van WinUptools is te bereiken met commando Programmer in het hoofdmenu. Het programmermenu
Een aantal commando's heeft betrekking op een gedeelte van het werkgeheugen dat Target Zone (TZ) wordt genoemd. De grootte van het werkgeheugen is gelijk aan die van een 8K byte EPROM (8192 bytes). Met de commando's kan onder meer de toekomstige inhoud van de EPROM worden samengesteld.

Het eerste en het laatste adres van de Target Zone kan worden ingesteld. De TZ van de EPROM is het gedeelte van de EPROM dat geprogrammeerd zal worden. De TZ in RAM is het feitelijke werkgebied. De inhoud hiervan kan in de TZ van de EPROM worden geprogrammeerd. Figuur 4.2 geeft de werkwijze schematisch weer.

Behalve een aantal commando's bevat de programmer een raster dat de inhoud van 256 bytes van het werkgeheugen laat zien. Meestal is het begin van de Target Zone zichtbaar maar met de schuifknop kunnen andere blokken van 256 bytes getoond worden.
De inhoud van het zichtbare werkgeheugen kan worden gemodificeerd door op een cel te klikken en een bytewaarde in te voeren.

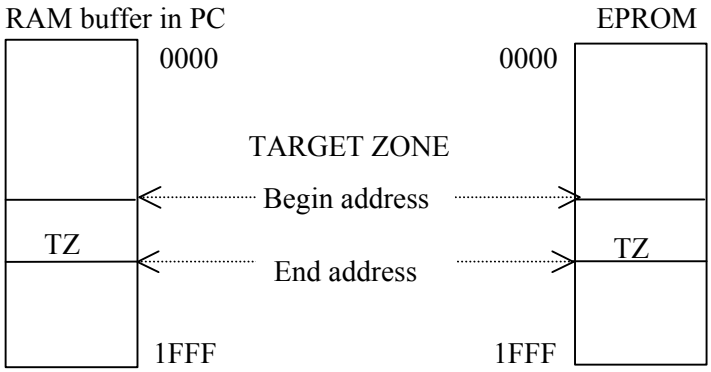


Fig. 4.2. Afbeelding van de TZ in het werkgeheugen en in de EPROM.

Hieronder staat een korte omschrijving van de functie van de verschillende commando's:

Target Zone	: Het begin- en eindadres van de Target Zone instellen.
Load TZ	: Een binaire file in de Target Zone in het werkgeheugen laden.
Save TZ	: De inhoud van de Target Zone in het werkgeheugen opslaan in een file.
Fill TZ	: De Target Zone in het werkgeheugen vullen met een waarde naar keuze.
Erase	: De inhoud van het gehele werkgeheugen met de waarde FF vullen.
Check TZ	: Controleren of de Target Zone van de EPROM leeg is (Blank Check).
Read TZ	: De inhoud van de Target Zone in EPROM kopiëren naar het werkgeheugen.
Program	: De inhoud van de Target Zone in het werkgeheugen kopiëren naar EPROM.
Verify TZ	: De inhoud van de Target Zones in EPROM en en het werkgeheugen vergelijken.
Checksum	: Alle bytes van EPROM en van de RAM modulo 10000H optellen.
Initialize	: De verbinding met de programmer testen en initialiseren.

4.3 Een niet lege EPROM hergebruiken

Het schrijven van een programma gaat bijna nooit in een keer goed. Na het vullen van de EPROM en testen van het programma blijkt meestal dat er wijzigingen aangebracht moeten worden. De EPROM moet worden gewist in een UV-wisser alvorens hij opnieuw gebruikt kan worden. Dit duurt ongeveer 30 minuten. Als men niet zolang wil wachten kan een nieuwe EPROM worden gepakt. Als iedere praktikant dat doet is de voorraad lege EPROMs spoedig uitgeput. Er is echter een eenvoudige methode om een EPROM ongewist opnieuw te gebruiken. Het verbeterde programma wordt daarbij achter het afgekeurde programma in de EPROM geplaatst. De inhoud van de EPROM vanaf adres 0000 tot het adres waar het gewijzigde programma komt te staan wordt met de NOP instructies gevuld. Nu zal na een RESET de processor een reeks NOP instructies tegenkomen voordat het programma uitgevoerd wordt. Deze methode is gebaseerd op het feit dat de opcode van de NOP instructie (No Operation) 00000000B is en dat in een EPROM een 1 wel in een 0 veranderd kan worden maar een 0 niet in een 1.

5 Timers van de 8051

De 8051 heeft twee timer/counters, T0 en T1. De functies van de timers T0 en T1 komen nagenoeg met elkaar overeen. Alle timer/counters kunnen slechts omhoog tellen: de inhoud van het telregister van een timer wordt bij iedere actieve flank van het ingangssignaal van de timer met 1 verhoogd. Als de maximale telstand is bereikt zal bij een volgende klokpuls de timer overlopen (overflow).
Om een tijdinterval te verkrijgen of om een bepaald aantal pulsen te tellen laat men een timer een waarde aftellen. Omdat de timers alleen omhoog kunnen tellen wordt het telregister van een timer geladen met 2^n - GETAL, waarbij n het aantal bits van de counter/timer is en GETAL de waarde die moet worden afgeteld. Daarna wordt de timer gestart. Men laat men een timer lopen tot de overflow komt die een bit zet en desgewenst ook een interrupt genereert.
Na een timer overflow gaat een timer door met tellen. Het telregister van een timer kan slechts programmatisch worden gewijzigd als de timer stil staat. Na elke overflow zal de timer dus doorgaans moeten worden gestopt en zal het af te tellen getal opnieuw in de timerregisters gezet moeten worden.

5.1 Instellen van timers T0 en T1

Er zijn twee Special Function Registers die de besturing van de timers T0 en T1 voor hun rekening nemen: TMOD en TCON. Register TMOD stelt de configuratie van de timers in, met register TCON kan de interactie van de timers met het programma worden bepaald.

5.1.1 Instellen van het TMOD register

TMOD (Timer/Counter Mode Control Register) (geheugenadres 89H)

7	6	5	4	3	2	1	0
GATE	C/T	M1	M0	GATE	C/T	M1	M0

De laagste 4 bits van TMOD configureren timer T0, de hoogste 4 bits zijn functioneel identiek maar hebben betrekking op timer T1. Beide timers beschikken ieder over vier modi, mode 0 t/m 3. De mode wordt binair ingevoerd in de bits M0 en M1 in register TMOD.

Door middel van bit C/T in register TMOD kan men kiezen voor de counter-functie (C/T=1) of de timer-functie (C/T=0). De counter-functie telt 1-0 (hoog-laag) overgangen van de externe timer-ingang T0 (P3.4) resp. T1 (P3.5). De maximale telfrequentie van het ingangssignaal is Fosc/24 (dit komt overeen met twee maal de machinecyclus). Aan de duty-cycle wordt geen beperking opgelegd, echter het signaal moet minimaal gedurende één machinecyclus hoog en gedurende minimaal één machinecyclus laag zijn.
De timer-functie telt iedere 12de puls van de oscillatorfrequentie. De intervaltijd is dus gelijk is aan één machinecyclus. In deze situatie zijn de pennen T0 resp. T1 vrij voor input of output.

De functie GATE maakt de externe besturing van de timers T0 en T1 mogelijk. Met een ‘1’ in bit GATE van register TMOD telt de timer alleen als de pen INT0 (P3.2) resp. INT1 (P3.3) hoog is en als de timer aan is gezet (zie register TCON). Een timer wordt gestopt met een laag signaal op een van de pennen INT0 resp. INT1. INT0 en INT1 zijn te beschouwen als externe start/stop-ingangen van de timers. Met een ‘0’ in bit GATE loopt de timer altijd als hij aan is gezet. In deze situatie zijn de pennen INT0 resp. INT1 vrij voor input of output.

Hier volgt een opsomming van de verschillende modi van de timers T0 en T1.

5.1.2 Mode 0: M1=0, M0=0

De timers T0 en T1 gedragen zich onafhankelijk van elkaar als een 8 bits timer met een vaste 5 bits voordeler (prescaler) in TL0 respectievelijk TL1. Dus de frequentie van het ingangssignaal (kloksignaal) wordt door 32 (2^5) gedeeld en het resultaat wordt aangeboden aan de 8 bits timer TH0 respectievelijk TH1. De effectieve telfrequentie met C/T=0 is dus Fosc/(12*32).
Bij een overflow (de telwaarde gaat van FFH naar 00H) wordt het timer overflow bit (TF0 respectievelijk TF1 in register TCON) op '1' gezet.

5.1.3 Mode 1: M1=0, M0=1

De timers T0 en T1 gedragen zich onafhankelijk van elkaar als een 16 bits timer. T0 is samengesteld uit TL0 en TH0 terwijl T1 is samengesteld uit TL1 en TH1. De effectieve telfrequentie met C/T=0 is Fosc/12.
Bij een overflow (timer telwaarde gaat van FFFFH naar 0000H) wordt het timer overflow bit (TF0 resp. TF1 in register TCON) op '1' gezet.

5.1.4 Mode 2: M1=1, M0=0

De timers T0 en T1 zijn twee onafhankelijk van elkaar werkende 8 bits timers zonder prescaler. Het hoge telregister TH0 resp. TH1 wordt gebruikt als 'reload value', in het lage telregister TL0 resp. TL1 wordt geteld. Bij een overflow (telwaarde gaat van FFH naar 00H) wordt de waarde uit het hoge telregister automatisch gekopieerd in het lage telregister. Het tellen gaat vanaf deze waarde verder. De effectieve telfrequentie met C/T=0 is Fosc/12.

5.1.5 Mode 3: M1=1, M0=1

In deze mode bestaat timer T0 uit twee verschillende 8 bits tellers, TL0 en TH0. Het lage telregister TL0 wordt bestuurd door de bits GATE en C/T van timer T0 in register TMOD. De externe ingangen hierbij zijn T0 op pen P3.4 en INT0 op P3.2. De bits TR0 en TF0 in register TCON hebben betrekking op timer TL0.
Het hoge telregister TH0 loopt alleen op 1/12de van de oscillatorfrequentie. Met het timer run bit TR1 van timer T1 wordt timer TH0 gestart. Bij overflow van TH0 wordt het overflowbit TF1 geset. De timers TL0 en TH0 kunnen allebei interrupts genereren.
Timer T1 staat stil in mode 3. De telregisters behouden hun inhoud als timer T1 in mode 3 komt. Als timer T0 in mode 3 staat kan timer T1 gestart worden door van mode 3 naar een willekeurige andere mode te schakelen. TR1 en TF1 zijn niet beschikbaar voor timer T1. Met timer T0 in mode 3 kan timer T1 nog worden benut als baudrategenerator of in een applicatie die geen interrupt vereist.

5.1.6 Instellen van het TCON register

TCON (Timer/Counter Control Register) (geheugenadres 88H)

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Timer T0	Timer T1	Functie
IT0	IT1	**Interrupt type controlebits** IT0 voor INT0 en IT1 voor INT1. Deze bits moeten worden geset en gereset door de software. Met IT0 resp. IT1 op '1' gezet is de externe interruptingang INT0 resp. INT1 gevoelig voor een de dalende flank. Met een '0' in een van deze bits is de interrupt actief tijdens een laag niveau van de externe overeenkomstige interrupt lijn.

IE0	IE1	**Interrupt statusbits**. Deze bits worden door de processor geset bij detectie van een interrupt op de overeenkomstige externe interruptingang INT0 resp. INT1. Als een interruptingang flankgevoelig is wordt het statusbit automatisch gereset bij het afhandelen van de interrupt. Bij een niveau-interrupt moet het statusbit door de software worden gereset.
TR0	TR1	**Timer runbits** TR0 voor timer T0 en TR1 voor timer T1. Deze bits moeten worden geset en gereset door de software. Met een '1' in een runbit loopt de timer/counter. In de runmode wordt de inhoud van het telregister bij iedere ingangsklokpuls met 1 verhoogd. Met een '0' staat de timer/counter stil.
TF0	TF1	**Timer overflowbits** TF0 voor timer T0 en TF1 voor timer T1. Deze bits worden op '1' gezet door de hardware bij het overlopen van de bijbehorende timer. Ze worden automatisch op '0' gezet tijdens het afhandelen van de bijbehorende interrupt of door de software indien niet van interrupts gebruik wordt gemaakt. Zie voor meer details de paragraaf over interrupts.

Het telregister van een timer kan slechts programmatisch worden gewijzigd als de timer stil staat. In alle gevallen geldt dat na een timer overflow een timer door gaat met tellen.

LET OP: Timer T1 wordt vaak gebruikt als baudrate-generator voor de seriële poort. In dat geval moet men voorzichtig zijn met het programmeren van de registers TMOD en TCON.

6 Interruptmechanisme van de 8051

De CPU moet de uitvoering van een programma kunnen onderbreken om aandacht te kunnen geven aan een externe gebeurtenis (een signaal van een randapparaat) of een interne gebeurtenis (bijvoorbeeld het aflopen van een timer). Stel bijvoorbeeld dat een timer overloopt. Dan genereert die timer een *interrupt*, waarna de CPU een interrupt serviceroutine uitvoert en daarna het onderbroken programma vervolgt.
Bij een interrupt wordt de huidige waarde van de programcounter op de stack gezet, waarna de PC wordt geladen met een bepaald geheugenadres, dat het bij de interruptbron behorende vectoradres wordt genoemd. Op dat adres moet de interrupt serviceroutine beginnen, bijvoorbeeld met een spronginstructie naar de plaats waar de interrupt serviceroutine in het programmageheugen staat. De instructie RETI haalt het terugkeeradres van de stack en reset de prioriteitsniveau flip-flop.

In de 8051 bestaan 6 interruptbronnen. De UART heeft twee bronnen die dezelfde interrupt delen: RI en TI.

Nr	Naam Interruptbron	Prioriteit	Vector-adres
1	INT0 externe interrupt 0	1	0003H
2	TF0 timer T0 overflow	2	000BH
3	INT1 externe interrupt 1	3	0013H
4	TF1 timer T1 overflow	4	001BH
5	RI/TI (UART)	5	0023H

Tabel 6.1. De interruptbronnen van de 8051.

6.1 Instellen van de interrupts

Elke interruptbron kan individueel actief of passief gemaakt worden door het corresponderende bit in het IE register op een ‘1’ of een ‘0’ te zetten.
De reactietijd na een interrupt (*interrupt latency*) bedraagt tussen de 3 en 9 machinecycli. Dit is de tijd die verloopt tussen de activering van de interrupt en het begin van de executie van de eerste instructie van de interruptroutine. De interrupt latency is niet constant omdat de CPU een interrupt pas herkent aan het eind van de executie van een instructie.

IE (Interrupt Enable Register) (geheugenadres A8H)

7	6	5	4	3	2	1	0
EA	---	---	ES	ET1	EX1	ET0	EX0

Een ‘1’ zet de specifieke interrupt aan.

Bit	Afkorting	Functie
0	EX0	Geef externe interrupt 0 door.
1	ET0	Zet interrupt timer T0 aan.
2	EX1	Geef externe interrupt 1 door.
3	ET1	Zet interrupt timer T1 aan.
4	ES	Zet interrupt seriële poort aan.
5	---	Niet gebruikt
6	---	Niet gebruikt
7		EA Algemene interrupt enable/disable. Een ‘0’ in dit bit blokkeert alle interrupts.

6.2 Interruptprioriteitenstructuur

Aan elke interruptbron kan de prioriteit ‘hoog’ of ‘laag’ worden toegekend. De interruptprioriteiten worden gedefinieerd in register IP. Een interrupt met prioriteit ‘laag’ kan worden geïnterrumpeerd door een interrupt met prioriteit ‘hoog’. Als een interrupt met prioriteit ‘hoog’ en een interrupt met prioriteit ‘laag’ gelijktijdig optreden gaat de interrupt met prioriteit ‘hoog’ voor.

Binnen de prioriteitsniveaus ‘hoog’ en ‘laag’ wordt een tweede prioriteitsschema gebruikt. Als meerdere interrupts binnen een prioriteitsniveau gelijktijdig optreden wordt de interrupt met de hoogste prioriteit als eerste afgehandeld. De hoogste prioriteit heeft in het volgende overzicht de waarde 1. De prioriteitsvolgorde van de vijf 8051 interrupts wordt vastgelegd in register IP (bit 0 t/m 4).

IP (Interrupt Priority Register) (geheugenadres B8H)

7	6	5	4	3	2	1	0
---	---	---	PS	PT1	PX1	PT0	PX0

Een ‘1’ zet de interrupt prioriteit op hoog en een ‘0’ op laag.

Bit	Afkorting	Prioriteit	Functie
0	PX0	1	Externe interrupt INT0.
1	PT0	2	Timer T0.
2	PX1	3	Externe interrupt INT1.
3	PT1	4	Timer T1.
4	PS0	5	Seriële poort.
5	---		Niet gebruikt.
6	---		Niet gebruikt.
7	---		Niet gebruikt.

6.3 Memory-map van de EPROM bij gebruik van interrupts

Aangezien iedere interruptbron een vast vectoradres gebruikt, zullen we bij het afbeelden van het programma op de EPROM extra voorzichtig moeten zijn. We moeten het adres van de interruptvector natuurlijk reserveren voor de interruptroutine of tenminste een spronginstructie naar de interruptroutine. De onderstaande tabel toont een mogelijke manier om het hoofdprogramma en de interruptroutine goed in de EPROM te krijgen. In dit programma wordt gebruik gemaakt van timer T0, dus 000BH is het vectoradres.

Adres	**Instructie**		**Uitleg**
0000H		LJMP BEGIN	Spronginstructie naar begin hoofdprogramma
000BH		LJMP INTROUT	Spronginstructie naar interruptroutine
0100H	BEGIN	----	Start hoofdprogramma bij label ‘BEGIN’
	LUS	----	Hoofdprogrammalus
		SJMP LUS	
INTROUT		----	Start interruptroutine
		RETI	Einde interruptroutine

Tabel 6.2. Mogelijke memory map van de EPROM bij gebruik van de interrupt van timer T0.

Mocht er na het programmeren van de EPROM een fout in het programma blijken te zitten, dan kan de EPROM met de juiste code als volgt opnieuw geprogrammeerd worden:

1. Vervang de LJMP INTROUT instructie door een NOP en plaats de nieuwe spronginstructie erachter;
2. Vervang het hoofdprogramma en de interruptroutine vanaf hun oude startadres door NOPs;
3. Plaats het nieuwe hoofdprogramma en de interruptroutine achter de oude interruptroutine.

7 Programmaontwikkeling voor de 8051

Er zijn verschillen methoden om de machinecode waaruit een programma bestaat te genereren, te weten: handmatig coderen, assembleren en compileren. Op het practicum wordt de eerstgenoemde methode gebruikt bij het uitvoeren van de eerste programmeeropdracht. Daarna wordt de assembler gebruikt.

7.1 Handmatig coderen

Kleine programma's kunnen worden gemaakt zonder een assembler te gebruiken. Hierbij moeten we zelf de machinecode in de vorm van hexadecimale getallen bepalen en opschrijven. Vervolgens wordt een EPROM met deze gegevens geprogrammeerd.

Allereerst worden de instructies in assembly-taal opgeschreven waarbij van labels gebruik gemaakt mag worden. Daarna wordt van iedere instructie de machine-codebytes bepaald en genoteerd. De lijst met instructies (de instructieset) in deze handleiding is daarbij onmisbaar.

Om het coderen overzichtelijk te houden wordt een geschikt formulier gebruikt. Achter in deze handleiding bevindt zich een aantal van deze formulieren. Het eerste formulier is voorzien van een voorbeeldprogramma. Het coderen verloopt als volgt:

- Noteer van iedere benodigde instructie de kolommen LABEL, OPCODE en OPERAND.
- Bepaal daarna van boven naar beneden het machine-codebyte dat de opcode representeert.
- Noteer dit byte in kolom BYTE 1 in hexadecimale vorm.
- Stel vast uit hoeveel bytes een instructie bestaat en gebruik dit gegeven om de kolom ADRES in te vullen, te beginnen met het adres waar het programma moet beginnen.
- Als de instructie een operand bevat moet die ook vertaald worden. De operand komt in BYTE 2 en eventueel in BYTE 3 te staan.

Achterwaartse sprongen kunnen worden opgelost als de kolom ADRES netjes is ingevuld, immers bij ieder label van een vertaalde instructie is nu het adres bekend. Omdat een voorwaartse adresreferentie nog niet kan worden opgelost laten we de operand-bytes nog even open. Nadat het gehele programma is vertaald kunnen de opengelaten sprongadressen worden ingevuld.

De kolommen BYTE 1, BYTE 2 en BYTE 3 bevatten de machine-code van het programma. Deze bytes worden op de PC ingetypt en met een EPROM-programmer in een EPROM gezet.

Het volgende recept is typisch voor het handmatig programmeren van een EPROM. De
Kies in het hoofdmenu van WinUptools het EPROM-programmer menu met commando Programmer.

7.1.1 Recept 1

Erase	De buffer leeg maken om een lege EPROM te simuleren.
Initialize	De EPROM-programmer initialiseren; de programmer meldt zich.
	De EPROM in programmer plaatsen. Pen 1 komt aan de kant van het klempookje.
Target Zone	De TZ instellen.
	De machine-code invoeren in het werkgeheugen.
Check	Testen of de TZ in de EPROM leeg is.
Program	De TZ programmeren.

7.2 Assembleren

Eerst wordt met behulp van een editor een tekstfile gemaakt waarin de instructies in assembly-taal worden genoteerd. Met een vertaalprogramma, genaamd assembler, wordt deze tekstfile omgezet in een binaire file die daarna in de buffer van de programmer moet worden geladen. Vervolgens wordt hiermee een EPROM geprogrammeerd.

De assembler wordt in het volgende hoofdstuk behandeld. Een efficiënt gebruik van de assembler wordt bevorderd door dat hoofdstuk grondig te bestuderen. Er is niet overwogen om de oorspronkelijk documentatie, die in het Engels is gesteld, te vertalen omdat in de computerwereld deze taal veel wordt gebruikt. Het is leerzaam daar eens kennis van te nemen.

Het volgende voorbeeld kan dienen om een indruk te krijgen van de diverse stappen die nodig zijn om een programma te ontwikkelen en in een EPROM te zetten.

7.2.1 Recept 2

1. Voer een projectnaam in met het commando New in het hoofdmenu van WinUptools, bijvoorbeeld OPDR2.A51.
2. Start de editor met het commando Edit. De default editor is Notepad. Creëer een nieuwe file.
3. Tik het programma in, gebruik alleen hoofdletters en gebruik de TAB-toets om kolommen te maken. Neem boven in dit tekstbestand een ORG directief op dat verwijst naar een vrije locatie in de EPROM. Dit adres is 0000H voor een lege EPROM.
   Voorbeeld: ORG 0200H
4. Bewaar de file en sluit de editor af. De file OPDR2.A51 wordt op de harde schijf bewaard.
5. Assembleer de file OPDR2.A51 door het commando Assemble te geven. De assembler genereert een listfile (OPDR2.LST), een binaire file (OPDR2.BIN en een hexfile (OPDR2.HEX).
6. Als de assembler fouten meldt moeten die eerst worden opgelost alvorens verder te gaan.
7. Bekijk de listfile OPDR2.LST met het commando View file.
   Lijkt alles te kloppen? Ook het beginadres?
8. Kies het EPROM-programmer menu met commando Programmer.
9. Programmeer een EPROM met recept 3.

Als de EPROM al een programma bevat moet dit worden “gewist” door de machinecode te vervangen door een reeks NOP instructies. Ook moet het startadres van het nieuwe te programmeren programma worden bepaald. Ga als volgt te werk:

7.2.2 Recept 3

De EPROM-programmer selecteren (indien nog niet gedaan).

Erase	De buffer leeg maken.
Initialize	De programmer testen.
	De EPROM in de programmer plaatsen. Pen 1 aan de kant van het klempookje.

Zonodig nu eerst het oude programma wissen. Als de EPROM leeg is hoeft dit niet te worden gedaan.

Target Zone	De TZ instellen vanaf het eerste adres van het te wissen stuk tot/met 1 minder dan het ORG adres (meestal is dit de huidige TZ).
Fill TZ	De TZ vullen met bytes met waarde 0.
Program	De TZ in de EPROM programmeren met de NOP instructies.

Het nieuwe programma in de EPROM zetten.

Target Zone	De TZ instellen vanaf het ORG adres tot de gewenste waarde.
Load TZ	De binaire file laden in de TZ.
Check TZ	Testen of de TZ van de EPROM leeg is.
Program	De TZ in de EPROM programmeren.

8 Assembler Documentation

8.1 Introduction

The ASM51EP assembler is a program development tool used to translate a 8051 assembly language source file into an object code file containing the corresponding machine code. Two different types of object code files can be generated. The first type is binary coded and suitable to be programmed into an EPROM. This EPROM has to be placed in the 8051 'target' computer. The second type is a so-called 'hex' file containing machine code information in readable ASCII text. This file can be transported (downloaded) to a target computer, where the machine code portion is converted to the actual binary machine code and placed in RAM by a running program (absolute loader).

In short: the assembler inputs an assembly source file and outputs a list file, a hex file, and a binary file. Option parameters are available in order to choose from two types of hex formats and two types of binary formats. This will be explained later.

The source file is translated during two passes. While reading the file for the first time, the assembler stores encountered labels and their corresponding addresses in a *symbol table* and translates instructions into machine code. During the second pass, references to labels are resolved and output files are generated.

The assembler keeps track of its most important task - where to put what in program memory - by means of a pointer, called the *location counter*. After each translation of an instruction the location counter marks the location (memory address) of the next byte to be stored in program memory. Note that the assembler's location counter and the processor's program counter have completely different functions.

Apart from storing machine code in memory, the assembler can be instructed to allocate and fill memory with data. This data is also placed in program memory. If the computer system has separate program and data memory areas, data placed in program memory can be read only, not written. However, if the memory areas are overlapping there is no physical difference in the actual storage location. Consequently, data stored in program memory can be accessed (read and written) as if it had been stored in data memory.

8.2 Assembly line format

An assembly source-line generally consists of three parts:

```
Label          Instruction     Comment

example:
Lab_1:         MOV             A,#12          ;load accumulator with number
```

If the first character of a line is one of the letters A to Z, or a to z , the first symbol of the line is regarded as a label. Labels may be up to 9 characters long and may contain the underscore symbol (_). If the first character is a space or a tab character, the assembler decides that there is no label. Labels must be delimited by a space, a tab or a colon (:). Note that a delimiting colon is not a part of the label and is not counted either.

The assembler is case sensitive with respect to labels (i.e. upper case and lower case characters are recognized as different characters). The -U option is used when the assembler has to treat upper case and lower case characters in labels equally. See the paragraph on command line options for more details. The instruction part of the line has to be in upper case. It is interpreted as an assembler pseudo-instruction or machine-language instruction.
When a semicolon (;) appears anywhere in the line, the text following it is interpreted as a comment.

8.3 Arithmetic expressions

Numerical constants and character constants are arithmetic expressions. Also labels are arithmetic expressions. $ is a special arithmetic expression, its value is the value of the location counter.

The assembler does not recognize parentheses '( )'.
The integer range is 0 to 65535.

When expr1 and expr2 are arithmetic expressions, then also:

```
-expr1          ; a negative value
expr1 + expr2   ; Sum
expr1 - expr2   ; Difference
expr1 * expr2   ; Product
expr1 / expr2   ; Integer Division (quotient part of divide result)
expr1 % expr2   ; Modulo Division (remainder part of divide result)
```

Examples:
3+7*2 equals 3+(7*2) and gives 17.
17/3 gives 5
17%3 gives 2

8.4 Notation of numerical and alphabetic constants

Numerical constants	:			
Decimal numbers	:	normal notation.		
Hexadecimal numbers	:	marked by suffix H, must start with a number 0 to 9		
Binary numbers	:	marked by suffix B.		
Character constants	:	a character enclosed by single quotes.		
String constants	:	a string of characters enclosed by single quotes.		
Examples	:			
Decimal numbers	:	0	00001	12324
Hexadecimal numbers	:	0CH	0A3H	8ABCH
Binary numbers	:	11B	011B	00000011B
Character constants	:	'A'	'0'	'!'
String constants	:	'XYZ'	'1234H'	'this is text',0

8.5 Pseudo-Instructions

The following pseudo-instructions are assembler directives. They control the assembler but do not generate (executable) machine code.

```
            ORG     expr
            DB      expr,expr,expr ...
            DW      expr,expr,expr ...
            DS      expr
Label       EQU     expr
            END
```

8.5.1 Overview

The expression 'expr' is evaluated to a number.
ORG initializes (or sets) the location counter the value of 'expr'.
DB inserts bytes in a program and advances the location counter accordingly.
DW inserts double bytes in a program and advances the location counter accordingly.
DS only advances the location counter with the value of 'expr'. No data is stored.
EQU assigns the value of 'expr' to a symbol.

8.5.2 Short description

ORG (Origin)
The instruction

```
        ORG     expr
```

assigns the value of the arithmetic expression to the location counter. This instruction is used to make the program start or continue at a specific location in the memory. In the next example ORG is used to define the start address of a program at location 100H.

Example:

```
        ORG     100H
START   LJMP    SOMEWHERE
```

DB (Define Byte)
The instruction

```
        DB      expr,expr,....
```

saves the values of the arithmetic expressions as bytes in the program memory, and increments the location counter as required. String constants are also allowed. The next example shows how bytes and string values may be combined.

Example:

```
COMMAND DB      13,10,'COMMAND',0  ;CR,LF,String,Delimiter
```

This line places the (hex) byte values 0D,0A,43,4F,4D,4D,41,4E,44,00 in program memory in a consecutive order. The zero-byte marks the end of the string.

DW (Define Word)
The instruction

```
        DW      expr,expr,....
```

saves the values of the arithmetic expressions as two byte values in the program memory, and increments the location counter as required. The byte order is high byte at the current location counter followed by the low byte. String constants are also allowed. The following example shows the use of this directive.

Example:

```
        DW      1234H,ABCH,'ABC'
```

resulting in the consecutive byte values 12,34,0A,BC,41,42,43 (hex) in program memory.

DS (Define Storage)
The instruction

```
        DS      expr
```

increments the location counter with the value of the arithmetic expression. If the program is to be placed in external RAM and program memory and data memory are physically identical, the defined storage area may be used to store variables. In the next example DS is used to define a data storage area in external RAM at location 1000H.

Example:

```
        ORG     1000H
NUMBER  DS      2       ; reserve 2 bytes for variable NUMBER
RESULT  DS      4       ; reserve 4 bytes for variable RESULT
KEY     DS      1       ; reserve 1 byte for variable KEY
```

To read or write a variable in external data memory use the MOVX instruction:

```
        MOV     DPTR,#RESULT    ; initialize Data Pointer first
        MOVX    A,@DPTR         ; read first byte of RESULT
        MOVX    @DPTR,A         ; write first byte of RESULT
                                ; (DPTR)=1002H
```

Directive DS may also be used, combined with ORG instructions, to manage the internal RAM memory. This only works because DS does not leave any data in the binary and hex output files. The assembler assigns values to the labels which can be used in the program. In the next example DS is used to define the same data storage area in internal RAM at location 50H.

Example:

```
        ORG     50H
NUMBER  DS      2       ; reserve 2 bytes for variable NUMBER
RESULT  DS      4       ; reserve 4 bytes for variable RESULT
KEY     DS      1       ; reserve 1 byte for variable KEY
```

To read or write a variable in internal RAM use the MOV instruction with direct addressing.

```
        MOV     A,RESULT    ; read first byte of RESULT
                            ; address of RESULT is 52H
```

Note that the same method is used in both cases. However, variables have to be accessed differently.

EQU (Equate)
The instruction

```
LABEL   EQU     expr
```

assigns the value of the arithmetic expression to identifier LABEL. The next example shows how to use it to define constants.

Example:

```
MIN     EQU     100
MAX     EQU     1000+MIN-1  ; evaluates a simple expression
```

An alternative method to organize variables space in internal RAM uses the EQU directive instead of the DS directive. In the next example, EQU is used to define the addresses in internal RAM of several variables.

Example:

```
POINTER EQU     50H     ; reserve 2 bytes for variable POINTER
RESULT  EQU     52H     ; reserve 4 bytes for variable RESULT
KEY     EQU     56H     ; reserve 1 byte for variable KEY
```

The same method can be used to define space for variables in external data memory. Again, the MOVX instruction has to be used to access variables in external RAM.

END
The instruction

```
        END
```

marks the end of the source file. The assembler discards all lines following the END directive.

8.6 Bit addressing

The bit addressable locations may be specified with their actual bit addresses.
Example:

```
        CLR     21H     ; clear bit 1 of location 20H
        CPL     0E7H    ; complement bit 7 of the accumulator
        JB      95H     ; jump if port P1 bit 5 is logical '1'
```

An alternative way uses the assemblers bit specifier '.' following the byte address in which the bit is located.

Example:

```
ACC     EQU     0E0H    ; address of accumulator
P1      EQU     090H    ; address of port P1
SW5     EQU     P1.5    ; bit address of port P1 bit 5
        CLR     20H.1   ; clear bit 1 of location 20H
        CPL     ACC.7   ; complement bit 7 of the accumulator
        JB      P1.5    ; jump if port P1 bit 5 is logical '1'
        JB      SW5     ; jump if port P1 bit 5 is logical '1'
```

The last line in this example shows the use of a fully symbolic bit address to specify a bit operand. This method clearly has many advantages over the blunt way of bit addressing used in the previous example. Be careful not to use instructions for bit addressing when accessing byte locations. The results will be unpredictable.

Example:

```
ACC     EQU     0E0H    ; address of accumulator
        CLR     ACC     ; does not clear the whole accumulator
        CLR     A       ; this clears the accumulator
```

8.7 Invoking the ASM51EP assembler from the DOS command line.

The DOS command to start the assembler is:

ASM51EP [-options] filename[.A51]<Enter>
or ASM51EP [-option] .. [-option] filename[.A51]<Enter>

filename : Name of the file to be assembled, with extension .A51 or without extension. The assembler uses the file filename.A51 as the source file, the file filename.LST as the list file, the file filename.HEX as the hexadecimal file, an the file filename.BIN as the binary file.

options : action if option specified:

-L	no list file generated
-H	no hex file generated
-B	no binary file generated
-S	no symbol table in listing included
-E	ASM51EP type hex file generated (only if -H not specified)
-I	8K byte memory image file for EPROM generated (only if -B not specified)
-U	labels converted to upper case
-Tn	tab-spacing in list file is n characters

defaults	:	action if option not specified: list file generated hex file generated binary file generated symbol table added in listing (only if -L not specified) hex file generated in INTEL Hex format binary dump file generated case sensitive labels tab-spacing is 10 characters
exit code	:	The assembler leaves a DOS exit code <> 0 if an error has occurred.
Examples	:	ASM51EP -L -H -B -U TEST<Enter> Generates no files. Errors are listed on the screen. Labels in lower case are converted to upper case so LABEL, Label, and label are identical. ASM51EP -SEI TEST.A51<Enter> Generates a list file not including a symbol table, a hex file in ASM51EP hex format, and a binary memory image file. ASM51EP -HT8B TEST<Enter> Generates a list file with horizontal tabs expanded to multiples of 8 characters including a symbol table, no hex file, and no binary file.

8.8 List file

The listing produced by the assembler has the following structure:

```
****** LISTING of ASM51EP (C:ADDN) ******
LINE   LOC    OBJ      T       SOURCE
  1    0000                            ; SUBROUTINE ADDN
  2    0000                            ; FUNCTIE:          Optellen van getallen met lengte N
  3    0000                            ;                   a + b -> a
  4    0000                            ; INPUT             R0 = pointer naar low byte van a
  5    0000                            ;                   R1 = pointer naar low byte van b
  6    0000                            ;                   R7 = aantal bytes per getal (is N)
  7    0000                            ; OUTPUT:           C = Carry
  8    0000                            ;                   R0 = pointer naar low byte van som
  9    0000                            ; VERANDERT:        A,C,R1,R7
 10    0000                            ;
 11    0000   C0 00   [2]              ADDN    PUSH  0     ; bewaar R0 op de stack
 12    0002   C3      [1]                      CLR   C     ; zet de carry op 0
 13    0003   E6      [1]              ADD1    MOV   A,@R0 ; haal byte van getal a op
 14    0004   37      [1]                      ADDC  A,@R1 ; tel byte van b op + Carry
 15    0005   F6      [1]                      MOV   @R0,A ; zet som terug
 16    0006   08      [1]                      INC   R0    ; verhoog pointer naar a
 17    0007   09      [1]                      INC   R1    ; verhoog pointer naar b
 18    0008   DF F9   [2]                      DJNZ  R7,ADD1 ; verlaag aantal, klaar?
 19    000A   D0 00   [2]                      POP   0     ; herstel R0
 20    000C   22      [2]                      RET         ; keer terug naar aanroeper
 21    000D
 22    000D                                    END
```

The column “LINE” contains the line number of the source text, in decimal notation.
The column “LOC” (Location) indicates the value of the location (address) counter (4 digit hexadecimal).
The column “OBJ” indicates the generated object code in hexadecimal notation.
The column “T” shows the execution time of instructions in cycles. At a clock of 12 MHz, one cycle lasts exactly one microsecond.
The column “SOURCE” shows the assembly-language source code.

8.9 HEX file

The assembler generates a readable object-code file that forms a “HEX DUMP”. Each line of this file has a specific format according to the selected type of hex file. You may select the well known universal INTEL hex format or the assembler specific ASM51EP hex format. How this is done will be explained later.

8.9.1 INTEL hex format (default)

Each line in an INTEL hex file consists of the following items:

:NATBB.....BC<CR><LF>

Where:

:	Start of line character.
N	Number of object-code bytes to follow (as a 2-digit hexadecimal number).
A	Address of first data byte to follow (as a 4-digit hexadecimal number).
T	Type of line (as a 2-digit hexadecimal number). 00 is a line containing object-code bytes 01 is the last line (no object-code bytes) 03 is a line holding the programs start address
B	These are the object-code bytes, each as a two-digit hexadecimal number.
C	Check-sum. 2’s complement value of the modulo 256 sum of all preceding bytes with the exception of ‘:’ (as a 2-digit hexadecimal number).

The CR (Carriage Return) and LF (Line Feed) characters are not required. However they give the receiver time to process the line before the next line comes in.

Example (spaces added here to show the field boundaries):
:10 0800 00 310029010000CD0100010200CD030001 EB
:00 0000 01 FF

8.9.2 ASM51EP type hex format (with -E option)

Each line in a hex file consists of the following items, separated by spaces and ending with a CR and a LF character:

N A:B B . . . . . B C<CR><LF>

Where:

N	Number of object-code bytes to follow (as a 2-digit hexadecimal number).
A	Address of bytes to follow (as a 4-digit hexadecimal number).
:	Separator character.
B	These are object-code bytes, each as a two-digit hexadecimal number, separated by spaces.
C	Sum of object-code bytes as a 4-digit hexadecimal number (check-sum).

Example:
07 1000:E8 22 31 32 33 34 00 01D4<CR><LF>

8.10 Binary file

The assembler generates (writes) a binary file that forms an “Object Code Dump”. The file structure and length is specific for the selected type of binary file.

8.10.1 Binary dump file (default):

Only one ORG directive at the beginning of the source file is allowed using this type of binary output. Object-code bytes are written to the binary file in a consecutive order. Absolute load addresses are not included. The DS directive does not put any data in this type of binary file.

8.10.2 Binary memory image file (with -I option):

More than one ORG directive may be used in the source file. Object-code bytes are written to the binary file as if they were programmed in an 8 Kbyte (EP)ROM. Fill bytes (with value FF) are written from address 0 to the first code byte, between program blocks and after the last code byte to the end of the virtual 8 Kbyte memory space. The DS directive inserts fill bytes too. The relative location of each byte in the file is equal to its absolute address modulo 2000H.

8.11 Error reports of ASM51EP

1	string length	Indicated text string is too long.
2	missing or invalid term	Missing valid arithmetic expression.
3	unknown expression	Arithmetic expression can not be computed.
4	missing expression	An arithmetic expression must follow.
5	missing term	Missing arithmetic term.
6	to many bytes	Line generates too many opcode bytes.
7	range error	Arithmetic expression does not fit in BYTE
8	syntax	Syntax error.
8	unknown address	Indicated address (or label) is not known.
10	relative jump range	Jump out of range.
11	far jump range	Jump out of range.
12	base address not bit addressable	as is.
13	unknown opcode	as is.
14	invalid expression	as is.
15	missing label	as is.
16	double defined label	as is.
17	invalid line	Line ending not allowed.
18	division by zero	Zero divide in expressions.
19	invalid bit number	Wrong bit number in bit addressing.
20	invalid register	Wrong register in register indirect addressing.
21	end of file encountered	No END directive found.

8.12 Voorbeeldprogramma

De volgende listing toont het programma DT_IO dat wordt gebruikt om de computer te testen. Het is zeer aan te bevelen om dit programma goed te bestuderen.

```
********* LISTING of ASM51EP version 1.07 (DT_IO) *********
LINE LOC  OBJ       T  SOURCE
   1 0000
   2 0000
   3 0000              ;Testprogramma voor de DT_IO computer.
   4 0000
   5 0000              ;versie 1.1
   6 0000              ;12 september 1995
   7 0000
   8 0000 00 90        P1      EQU     090H
   9 0000 00 B0        P3      EQU     0B0H
  10 0000
  11 0000 00 82        DPL     EQU     082H
  12 0000 00 83        DPH     EQU     083H
  13 0000
  14 0000 00 89        TMOD    EQU     089H
  15 0000 00 88        TCON    EQU     088H
  16 0000 00 8C        TH0     EQU     08CH
  17 0000 00 8A        TL0     EQU     08AH
  18 0000 00 8D        TH1     EQU     08DH
  19 0000 00 8B        TL1     EQU     08BH
  20 0000
  21 0000                      ORG     0
  22 0000
  23 0000              ;*******************************
  24 0000              ; TEST 7-SEGMENT DISPLAY
  25 0000              ;*******************************
  26 0000 74 FE    [1]         MOV     A,#11111110B    ;rollend segment
  27 0002 23       [1] START1  RL      A               ;roteer 0 naar links
  28 0003 F5 90    [1]         MOV     P1,A            ;stuur naar display
  29 0005 12 00 C4 [2]         LCALL   WACHT           ;even wachten
  30 0008 20 B2 F7 [2]         JB      P3.2,START1     ;herhaal tot TEST in
  31 000B
  32 000B              ;*******************************
  33 000B              ; TEST SCHAKELAAR
  34 000B              ;*******************************
  35 000B 75 90 F9 [2] TEST1   MOV     P1,#11111001B   ;cijfer 1
  36 000E 12 00 C4 [2]         LCALL   WACHT
  37 0011 30 B2 FD [2] TEST1A  JNB     P3.2,TEST1A     ;wacht tot TEST los
  38 0014 12 00 C4 [2]         LCALL   WACHT
  39 0017 75 90 79 [2]         MOV     P1,#01111001B   ;1 met puntje
  40 001A 20 B2 FD [2] TEST1B  JB      P3.2,TEST1B     ;wacht tot TEST in
  41 001D
  42 001D              ;*******************************
  43 001D              ; TEST POORT 3
  44 001D              ;*******************************
  45 001D 7A FF    [1] TEST2   MOV     R2,#0FFH        ;low byte van telwaarde
  46 001F 7B 2F    [1]         MOV     R3,#02FH        ;high byte van telwaarde
  47 0021 85 02 04 [2]         MOV     4,2             ;init telregister low
  48 0024 85 03 05 [2]         MOV     5,3             ;init telregister high
  49 0027 75 90 A4 [2]         MOV     P1,#10100100B   ;cijfer 2
  50 002A 30 B2 FD [2] START2A JNB     P3.2,START2A    ;herhaal tot TEST los
  51 002D 11 C4    [2]         ACALL   WACHT           ;ontdender wachtlus
  52 002F 30 B2 F8 [2]         JNB     P3.2,START2A    ;wachten als TEST in
  53 0032 C2 B0    [1] START2B CLR     P3.0            ;clear bits 0,1,2,3,4,5
  54 0034 C2 B1    [1]         CLR     P3.1
  55 0036 C2 B2    [1]         CLR     P3.2
  56 0038 C2 B3    [1]         CLR     P3.3
  57 003A C2 B4    [1]         CLR     P3.4
  58 003C C2 B5    [1]         CLR     P3.5
  59 003E D2 B0    [1]         SETB    P3.0            ;set bits 0,1,2,3,4,5
  60 0040 D2 B1    [1]         SETB    P3.1
  61 0042 D2 B2    [1]         SETB    P3.2
  62 0044 D2 B3    [1]         SETB    P3.3
  63 0046 D2 B4    [1]         SETB    P3.4
  64 0048 D2 B5    [1]         SETB    P3.5
  65 004A 12 00 D1 [2]         LCALL   TELLER          ;verlaag teller
  66 004D 70 02    [2]         JNZ     START2C         ;spring als teller <> 0
```

```
 67 004F B2 97    [1]          CPL     P1.7             ;inverteer puntje
 68 0051 20 B2 DE [2] START2C  JB      P3.2,START2B     ;herhaal tot TEST in
 69 0054
 70 0054                       ;*******************************
 71 0054                       ; TEST TIMER 0
 72 0054                       ;*******************************
 73 0054 75 90 B0 [2] TEST3    MOV     P1,#10110000B    ;cijfer 3
 74 0057 75 89 11 [2]          MOV     TMOD,#00010001B  ;init timer mode
 75 005A D2 8C    [1] START3   SETB    TCON.4           ;start bit timer 0
 76 005C 30 8D FD [2] RUN3     JNB     TCON.5,RUN3      ;check overflow
 77 005F C2 8C    [1]          CLR     TCON.4           ;stop bit
 78 0061 C2 8D    [1]          CLR     TCON.5           ;clear overflow
 79 0063 B2 97    [1]          CPL     P1.7             ;puntje
 80 0065 12 00 C4 [2]          LCALL   WACHT
 81 0068 30 B2 EF [2]          JNB     P3.2,START3
 82 006B D2 8C    [1] START3A  SETB    TCON.4           ;start bit timer 0
 83 006D 30 8D FD [2] RUN3A    JNB     TCON.5,RUN3A     ;check overflow
 84 0070 C2 8C    [1]          CLR     TCON.4           ;stop bit
 85 0072 C2 8D    [1]          CLR     TCON.5           ;clear overflow
 86 0074 B2 97    [1]          CPL     P1.7             ;puntje
 87 0076 12 00 C4 [2]          LCALL   WACHT
 88 0079 20 B2 EF [2]          JB      P3.2,START3A     ;herhaal tot TEST in
 89 007C
 90 007C                       ;*******************************
 91 007C                       ; TEST TIMER 1
 92 007C                       ;*******************************
 93 007C 75 90 99 [2] TEST4    MOV     P1,#10011001B    ;cijfer 4
 94 007F D2 8E    [1] START4   SETB    TCON.6           ;start bit timer 1
 95 0081 30 8F FD [2] RUN4     JNB     TCON.7,RUN4      ;check overflow
 96 0084 C2 8E    [1]          CLR     TCON.6           ;stop bit
 97 0086 C2 8F    [1]          CLR     TCON.7           ;clear overflow
 98 0088 B2 97    [1]          CPL     P1.7             ;puntje
 99 008A 12 00 C4 [2]          LCALL   WACHT
100 008D 30 B2 EF [2]          JNB     P3.2,START4
101 0090 D2 8E    [1] START4A  SETB    TCON.6           ;start bit timer 1
102 0092 30 8F FD [2] RUN4A    JNB     TCON.7,RUN4A     ;check overflow
103 0095 C2 8E    [1]          CLR     TCON.6           ;stop bit
104 0097 C2 8F    [1]          CLR     TCON.7           ;clear overflow
105 0099 B2 97    [1]          CPL     P1.7             ;puntje
106 009B 12 00 C4 [2]          LCALL   WACHT
107 009E 20 B2 EF [2]          JB      P3.2,START4A     ;herhaal tot TEST in
108 00A1
109 00A1                       ;*******************************
110 00A1                       ; TEST EXTERN GEHEUGEN
111 00A1                       ;*******************************
112 00A1 75 90 92 [2] TEST5    MOV     P1,#10010010B    ;cijfer 5
113 00A4 90 01 00 [2]          MOV     DPTR,#0100H      ;begin bij adres 100
114 00A7 E0       [2] DPOINT   MOVX    A,@DPTR
115 00A8 B4 00 14 [2]          CJNE    A,#0,ERROR       ;er moet 00 in staan
116 00AB A3       [2]          INC     DPTR             ;volgende lokatie
117 00AC E5 83    [1]          MOV     A,DPH
118 00AE B4 20 F6 [2]          CJNE    A,#020H,DPOINT   ;stop bij 2000
119 00B1 12 00 C4 [2]          LCALL   WACHT
120 00B4 30 B2 FD [2] DPO1     JNB     P3.2,DPO1        ;wacht tot TEST los
121 00B7 12 00 C4 [2]          LCALL   WACHT
122 00BA 75 90 12 [2]          MOV     P1,#00010010B    ;5 met puntje
123 00BD 80 03    [2]          SJMP    EINDE
124 00BF 75 90 86 [2] ERROR    MOV     P1,#10000110B    ;E zonder puntje (Error)
125 00C2 80 FE    [2] EINDE    SJMP    EINDE
126 00C4
127 00C4                       ;*******************************
128 00C4                       ; SUBROUTINE WACHT T = R0*R1
129 00C4                       ;*******************************
130 00C4 78 FF    [1] WACHT    MOV     R0,#0FFH
131 00C6 79 FF    [1] LUS      MOV     R1,#0FFH
132 00C8 19       [1] STAP     DEC     R1
133 00C9 B9 00 FC [2]          CJNE    R1,#00H,STAP
134 00CC 18       [1]          DEC     R0
135 00CD B8 00 F6 [2]          CJNE    R0,#00H,LUS
136 00D0 22       [2]          RET
137 00D1
```

```
138 00D1                ;************************************************
139 00D1                ; SUBROUTINE TELLER
140 00D1                ; FUNCTIE Verlaagt 16 bit tellerstand in R4/R5 met 1
141 00D1                ;         Reset de teller met de waarde in R2/R3
142 00D1                ;         als de tellerstand 0 wordt
143 00D1                ; INPUT   R4/R5   is teller
144 00D1                ;         R2/R3   is telwaarde
145 00D1                ; OUTPUT  A=0     afgeteld
146 00D1                ;         A=1     nog niet afgeteld
147 00D1                ;************************************************
148 00D1 1C         [1] TELLER  DEC     R4              ;verlaag low byte teller
149 00D2 EC         [1]         MOV     A,R4            ;zet in A om te testen
150 00D3 60 06      [2]         JZ      TEL0            ;test low byte
151 00D5 BC FF 0E   [2]         CJNE    R4,#0FFH,TEL1   ;test voor borrow
152 00D8 1D         [1]         DEC     R5              ;verlaag high byte
153 00D9 80 0B      [2]         SJMP    TEL1            ;naar uitgang
154 00DB ED         [1] TEL0    MOV     A,R5            ;zet in A om te testen
155 00DC 70 08      [2]         JNZ     TEL1            ;test high byte
156 00DE E4         [1]         CLR     A               ;output waarde is 0
157 00DF 85 02 04   [2]         MOV     4,2             ;reset tellerstand low
158 00E2 85 03 05   [2]         MOV     5,3             ;reset tellerstand high
159 00E5 22         [2]         RET
160 00E6 74 01      [1] TEL1    MOV     A,#1H           ;output waarde is 1
161 00E8 22         [2]         RET
162 00E9
163 00E9                        END
```

9 Notatie van getallen

De schrijfwijze van een cijfer of een getal hangt af van het grondtal. Een decimaal getal heeft grondtal 10, daaraan zijn we gewend. Een hexadecimaal getal heeft grondtal 16 (decimaal geschreven) en een binair getal heeft grondtal 2. De binaire en de hexadecimale schrijfwijze komen in de computerwereld veel voor. De binaire komt overeen met de manier waarop getallen in een computer worden opgeslagen en verwerkt, namelijk als reeksen bits die de waarde 0 of 1 kunnen hebben. De hexadecimale schrijfwijze is een compacte notatiewijze voor deze onoverzichtelijke bitpatronen.

In de tabel hieronder zijn voor getallen van 0 tot 15 de verschillende notaties naast elkaar gezet.

decimaal	hexadecimaal	Binair
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

De volgende tabel laat een beperkt aantal grotere getallen zien.

decimaal	hexadecimaal	Binair
16	10	10000
32	20	100000
64	40	1000000
128	80	10000000
255	FF	11111111
256	100	100000000
512	200	1000000000
1024	400	10000000000
2048	800	100000000000
4096	1000	1000000000000
8192	2000	10000000000000
16384	4000	100000000000000
32768	8000	1000000000000000
65535	FFFF	1111111111111111
65536	10000	10000000000000000

9.1 Getallen met of zonder teken (signed en unsigned)

Bij getallen zonder teken maakt ieder cijfer van dat getal deel uit van de getalswaarde.

Bijvoorbeeld:

234 dec	=	$2 \times 10^2 + 3 \times 10^1 + 4$
A3C hex	=	$10 \times 16^2 + 3 \times 16^1 + 12 = 2620$ dec
101 bin	=	$1 \times 2^2 + 0 \times 2^1 + 1 = 5$

In getallen met een teken wordt het hoogstwaardige bit gebruikt om het teken aan te geven. Er bestaan verschillende binaire coderingen, onder meer de 2's complement codering en de excess codering (ook wel offset binary genoemd). De 2's complement code wordt vaak gebruikt als er gerekend moet worden.
De volgende tabel laat de verschillende coderingen zien voor 8 bit getallen. Bit 7 is het tekenbit bij 2's complement getallen.

Decimaal	Hex	2's compl	excess 128
127	7F	01111111	11111111
126	7E	01111110	11111110
//			
3	03	00000011	10000011
2	02	00000010	10000010
1	01	00000001	10000001
0	00	00000000	10000000
-1	FF	11111111	01111111
-2	FE	11111110	01111110
-3	FD	11111101	01111101
//			
-127	81	10000001	00000001
-128	80	10000000	00000000

De excess code lijkt op de unsigned binary code maar de getalswaarde is 2^{n-1} lager (n is het aantal bits, inclusief het tekenbit). Hier komt de naam 'offset binary' vandaan. Bij een 8 bit getal wordt de code excess 128 genoemd. De excess code komt onder meer voor bij Digitaal-Analoog convertor. De laagste waarde komt overeen met de meest negatieve analoge spanning die de convertor kan genereren (bijv. -5 Volt), de hoogste waarde komt overeen met de meest positieve spanning (bijv. +5 Volt). De excess code kan eenvoudig in de 2's complement code worden omgezet (en vice versa) door het tekenbit te inverteren.

9.2 Vlaggen in get register PSW

De 8051 processor is zo ontworpen dat de vlaggen in het Processor Status Word betrekking hebben op bewerkingen van zowel unsigned als 2's complement getallen. Afhankelijk van het gebruikte getalformaat moet de bijbehorende vlag getest worden. Voor unsigned getallen is dat de C-vlag en voor signed getallen de OV-vlag.

Voorbeeld:
na 80+7F is C=0 en OV=0 (geen unsigned en geen signed overflow)
na FF+01 is C=1 en OV=0 (unsigned overflow maar geen signed overflow)
na 7F+01 is C=0 en OV=1 (geen unsigned overflow maar signed overflow)
na 80+80 is C=1 en OV=1 (unsigned overflow en signed overflow)

10 Opdracht A (A1 en A2)

Vereiste voorbereiding vóór de practicumsessie:
- Lees van deze handleiding hoofdstukken 1 t/m 9, behalve 5 en 6
- Beantwoord de vragen van 10.2.

10.1 Inhoud van deze opdracht

- Je moet vóór de practicumsessie de vereiste voorbereiding hebben uitgevoerd;
- De thuis beantwoorde vragen worden eerst besproken;
- Daarna kun je beginnen aan het bouwen en testen van het computerbordje;
- Opdracht A1 bestaat uit het met de hand coderen van een programmaatje in assemblytaal. Vervolgens ga je dat programmaatje in de EPROM zetten om het door de 8051 computer te laten uitvoeren;
- Bij opdracht A2 moet je het programma van A1 uitbreiden. Ditmaal schrijf je het in de teksteditor en laat je het programma door de assembler omzetten in binaire code. Daarna programmeer je het in de EPROM om het door de 8051 te laten uitvoeren;
- Je moet de list en hex files bekijken om daar enkele vragen over te beantwoorden.

10.2 Theorievragen

Bekijk eerst het voorbeeldprogramma "`ROLLEND SEGMENT`" in Appendix C. De inhoud van de kolommen `BYTE 1`, `BYTE 2` en `BYTE 3` bestaat uit machinecodes die het resultaat zijn van de vertaling van de assemblytaal in de kolommen `LABEL`, `OPCODE` en `OPERAND`. In de kolom `ADRES` staat het adres van de opcode van een instructie.

Vraag 1:
In het programma "`ROLLEND SEGMENT`" wordt de stand van de toets TEST ingelezen en getest met de instructie `JB P3.2,START1`. Het is belangrijk te begrijpen hoe de relatieve sprong-instructies `JB` (Jump if Bit set) en `JNB` (Jump if Not Bit set) werken. De machinecode van de bedoelde instructie is `20 B2 F7`. Wat is de betekenis van deze bytes? Kijk goed in de instructieset!

Toelichting:
- De druktoets TEST op het controllerbordje is aangesloten op pen 2 van poort P3 in de processor (de notatie is P3.2). Poort P3 heeft adres B0H. Als de toets niet is ingedrukt staat er een spanning van 5 Volt op P3.2, wat door de controller gezien wordt al een logische "1". **Een logische "0" (0 Volt) geeft dus een ingedrukte toets aan.**
- Het display is aangesloten op poort P1 van de microcontroller. Poort P1 heeft adres 90H. Figuur 3.1 toont welk segment wordt aangestuurd door welke pen van poort P1. **Bij een logische "1" gaat het aangesloten segment uit, bij een "0" gaat het aan!** De listing van paragraaf 8.12 laat hiervan voorbeelden zien.

Antwoord:

`20`	
`B2`	
`F7`	

Vraag 2:
Op regel 130 van het voorbeeldprogramma van paragraaf 8.12 begint een subroutine genaamd `WACHT`. De functie van deze subroutine is enige tijd te wachten alvorens naar de aanroeper terug te keren (met de instructie `RET`). De wachttijd is evenredig met (dus niet gelijk aan) het produkt van de inhoud van de registers `R0` en `R1`. In het voorbeeldprogramma worden beide registers gevuld met `FFH` (255). Kolom T in de instructieset geeft aan uit hoeveel machinecycli een instructie bestaat. De executietijd van een machinecyclus is 12 x de periodetijd van de oscillatorfrequentie (12 MHz). Met deze gegevens kan de executietijd van een instructie worden bepaald.

A: Wat is de executietijd van de instructie `DEC R1` in machinecycli.
B: Wat is de executietijd van de instructie `CJNE R1,#00H,STAP` in machinecycli.
C: Wat is de executietijd van de binnenste lus (met label `STAP`) in machinecycli.
D: Wat is de executietijd van de buitenste lus (met label `LUS`) in machinecycli.
E: Geef een formule voor de executietijd van de gehele subroutine uitgedrukt in machinecycli.
F: Wat is de totale executietijd in milliseconden?
G: Hoe kun je de formule van E vereenvoudigen tot een benadering van de executietijd van de gehele lus? Schrijf een sterk vereenvoudigde formule voor de executietijd op. Onder welke voorwaarden geldt deze benadering?

Antwoorden:

A	
B	
C	
D	
E	
F	
G	

Vraag 3:
- Hoe kan de executietijd van de subroutine WACHT worden verkleind?
- Hoe kan de executietijd van de subroutine WACHT worden vergroot?

Antwoorden:

KLEINER	
GROTER	

10.3 Bouwen en testen van het 8051 bordje

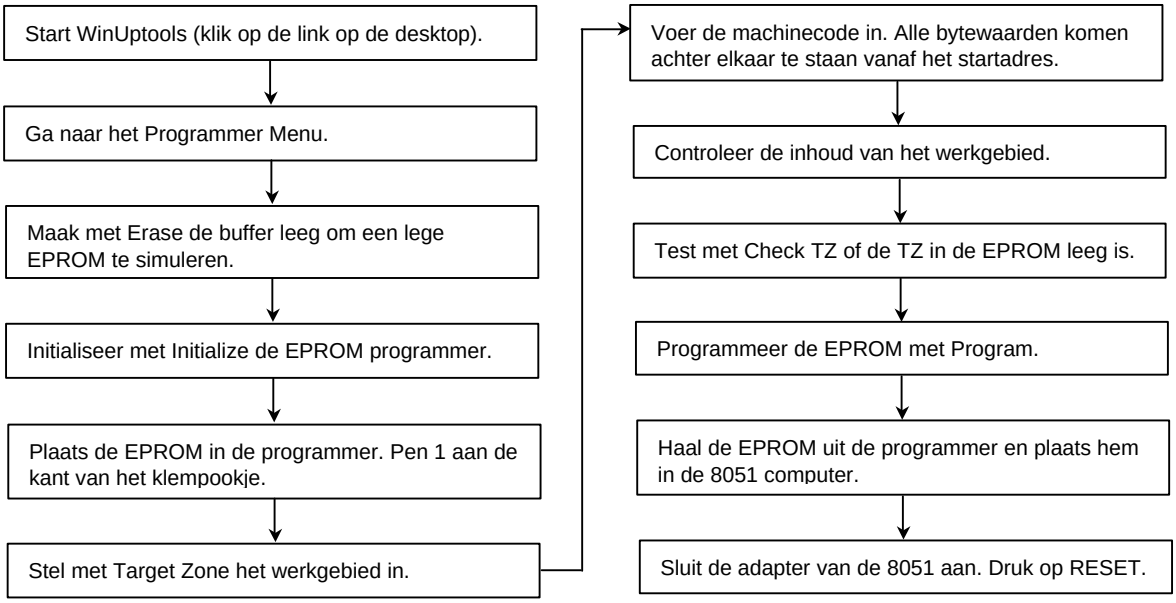
Elke student bouwt zijn eigen 8051 bordje. Verzamel daartoe alle componenten en volg de instructies uit paragrafen 3.5 e.v. nauwkeurig op. Na het bouwen volgt de testfase, paragrafen 3.9 en 3.10 geven hierover nadere uitleg.

10.4 Opdracht A1

Bekijk het programma “Opdracht A1” op het tweede formulier van Appendix D. Dit programma is geschreven in assemblytaal; aan jou de taak om dit met de hand om te zetten naar hexadecimale codes zoals dit bij Rollend Segment is gedaan. Vertaal de instructies met behulp van de instructietabellen die in de practicumhandleiding staan en noteer vervolgens de machinecode in hexadecimale vorm in de kolommen BYTE 1, BYTE 2 en BYTE 3. Bepaal de adressen van de instructies (kolom ADRES) en reken de relatieve sprongen uit. Het programma doet het volgende:

a: In register A wordt de waarde B6 gezet.
b: De inhoud van register A wordt in poort P1 gezet.
c: Wacht op het indrukken van de druktoets TEST.
d: Zodra de toets is ingedrukt worden de 8 bits van register A geïnverteerd. Gebruik hiervoor de instructie CPL A.
e: De inhoud van register A wordt in poort P1 gezet.
f: Wacht op het loslaten van de druktoets.
g: Het programma wordt voortgezet bij stap c.

Als het programma in hexidecimale machinecode is vertaald, dan kan het in de EPROM worden gezet om het door de 8051 computer uit te laten voeren. We nemen aan dat de EPROM nog leeg is. Volg dan de procedure zoals weergegeven in Figuur 10.1.



Figuur 10.1. Procedure voor het programmeren van een handmatig gecodeerd programma in de EPROM.

Controleer na het programmeren van de EPROM op een goede werking van het programma. Doet de computer wat je had verwacht? Demonstreer de werking het programma aan de practicumassistent.

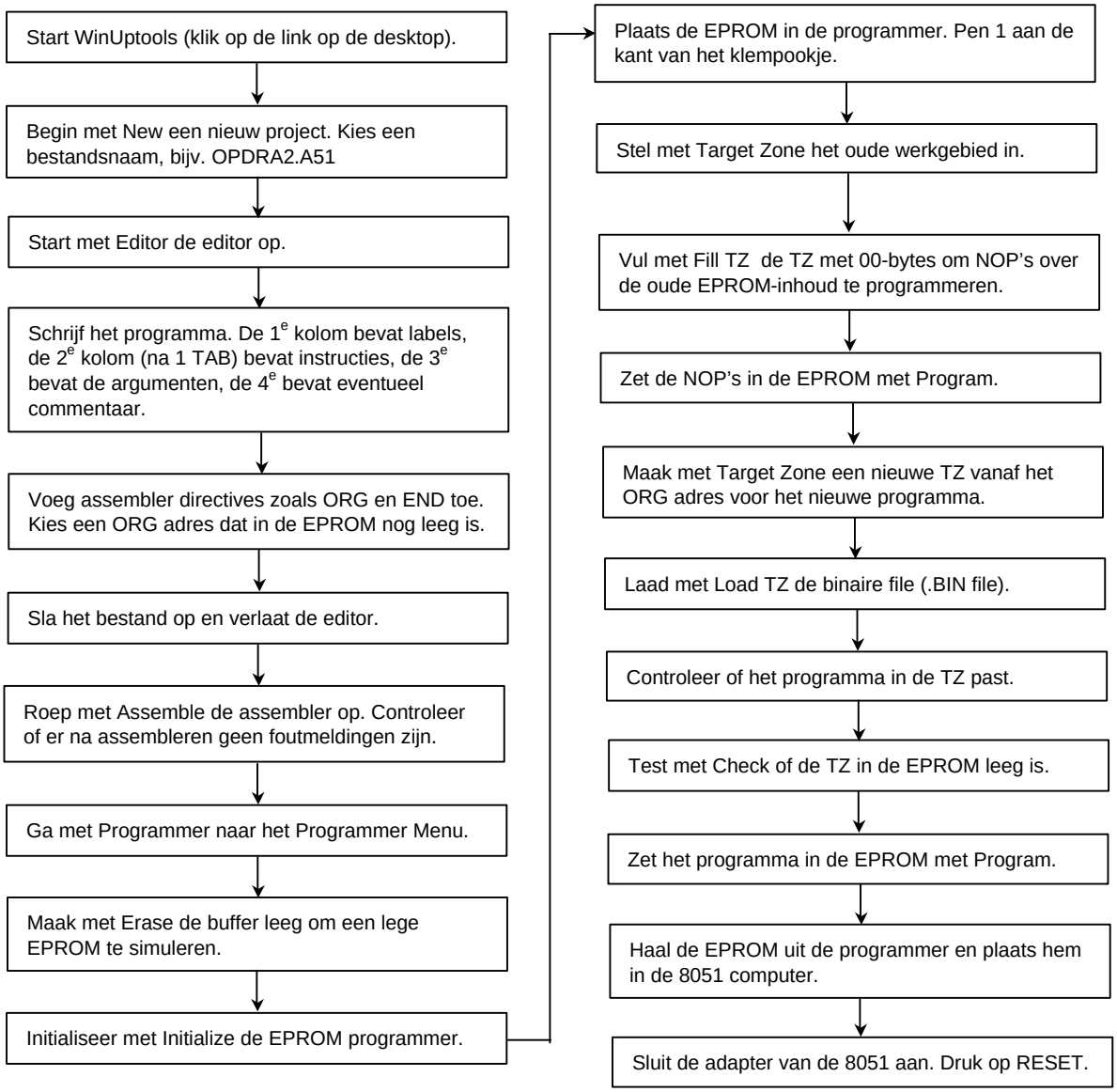
10.5 Opdracht A2

Een mechanische schakelaar heeft last van "schakeldender" waardoor de spanning op P3.2 er als volgt uit kan zien. In plaats van één enkelvoudige puls kan het indrukken van de druktoets een reeks pulsjes tot gevolg hebben. Het aantal wisselingen is niet voorspelbaar. Het gehele verschijnsel is voorbij na maximaal 40 milliseconden. Er worden dus fouten gemaakt als een programma snel genoeg is om alle wisselingen te zien.

In deze opdracht wordt het effect van schakeldender bestreden. Om ervoor te zorgen dat het denderen van de druktoets geen effect heeft op de goede werking van het programma van de eerste opdracht wordt op geschikte plaatsen een wachtlus van 50 milliseconden toegevoegd. Het doel hiervan is te wachten totdat de schakeldender (zowel na indrukken als loslaten) is afgelopen voordat het programma verder gaat. Deze wachtlus moet in een subroutine worden ondergebracht. Neem hiervoor de subroutine WACHT uit het voorbeeldprogramma in de handleiding en pas deze routine zo aan dat de gewenste wachttijd verkregen wordt. Het programma doet nu het volgende:

a: In register A wordt de waarde B6H gezet.
b: De inhoud van register A wordt in poort P1 gezet.
c: Wacht op het indrukken van de druktoets TEST.
d: Zodra de toets is ingedrukt wordt eerst 50 milliseconden gewacht.
e: De 8 bits van register A worden geïnverteerd.
f: De inhoud van register A wordt in poort P1 gezet.
g: Wacht op het loslaten van de druktoets.
h: Zodra de toets is losgelaten wordt 50 milliseconden gewacht.
i: Het programma wordt voortgezet bij stap c.

Ga te werk volgens de procedure die uiteengezet is in Figuur 10.2.



Figuur 10.2. Procedure om een programma te schrijven in de editor, te assembleren en in de EPROM te zetten. De EPROM is niet leeg, dus moeten er eerst 00-en over het oude programma heen worden geschreven.

Het programma moet dus in de editor worden geschreven, waarbij de nodige *assembler directives* moeten worden toegevoegd. Assembler directives zijn instructies die niet worden omgezet in executeerbare code, maar die de assembler nodig heeft om een goede vertaling van assembly naar bytecodes te maken. Zie het hoofdstuk over de assemblytaal voor meer informatie hierover.
In de EPROM staat op dit moment natuurlijk al het programma van opdracht A1. We moeten ervoor zorgen dat dit programma wordt gewist en dat alleen het nieuwe programma wordt uitgevoerd. Hiertoe programmeren we 00-en (dit is de opcode van de NOP, No Operation instructie) over het oude programma heen. Daarna plaatsen we het nieuwe programma erachter.

10.6 Vragen bij opdracht A2

Vraag 1:
Als je programma foutloos is geassembleerd, kijk dan naar de *list file*. Dit is een bestand met de extensie .LST dat verschijnt in je werkdirectory nadat je de assembler hebt opgestart. Verklaar de betekenis van de verschillende kolommen die je in de listingfile ziet:

Antwoord:

LINE	
LOC	
OBJ	
T	
SOURCE	

Vraag 2:
Kijk nu naar de .HEX file die de assembler ook in je werkdirectory heeft aangemaakt. Hoe zijn de records in de .HEX file opgebouwd?

Antwoord:

Vraag 3:
Zoek nu de CJNE instructie die in jouw programma is gebruikt op in de instructieset in Appendix B. Er staat in drie regels met symbolen uitgelegd wat de instructie precies doet. Wat betekent elk van deze regels?

Antwoord:

1	
2	
3	

Wat is de betekenis van de code in de laatste kolom bij de CJNE instructie?

Antwoord:

Na het beantwoorden van deze vragen ben je klaar met opdracht A. Vraag aan de assistent welke opdracht je volgende keer moet uitwerken en noteer dat hier:

B-opdracht:

11 Opdracht B1 – Kookwekker (B1a en B1b)

Vereiste voorbereiding vóór de practicumsessie:
- Leeswerk: hoofdstukken 5 en 6 resp. over timers en interrupts
- Bestudeer het Programma Structuur Diagram van de eerste B-opdracht grondig.

11.1 Inhoud van deze opdracht

- Je moet vóór de practicumsessie de vereiste voorbereiding hebben uitgevoerd;
- De eerste opdracht (B1a) bestaat uit het omzetten van een algoritme, weergegeven in een PSD, naar 8051 assemblytaal;
- Het te schrijven programma voor de eerste opdracht zal gebruik maken van de timers die in de 8051 microcontroller zijn ingebouwd.
- De tweede opdracht (B1b) bestaat uit het aanpassen van de eerste opdracht, zodanig dat het programma werkt op basis van interrupts.

11.2 De kookwekker met timer polling (B1a)

In het algemeen werkt een kookwekker als volgt:

1. Met een toetsenbord kan een bepaalde kooktijd worden ingesteld;
2. Na een druk op de knop begint de wekker met terugtellen;
3. Zodra de kooktijd is verstreken stopt de wekker met tellen;
4. De wekker genereert een alarmsignaal, bijvoorbeeld een geluidssignaal.

Met het 8051 bordje zullen we de wekker als volgt implementeren:

1. Als “toetsenbord”gebruiken we een schakelaarkastje (met 8 schakelaars), die we aansluiten op poort P1 van het 8051 bordje;
2. De TEST toets zal als startknop van de wekker fungeren;
3. De kooktijd zal worden afgeteld door middel van **timer polling**: een timer wordt ingesteld op een bepaalde beginwaarde, gestart en het programma blijft wachten tot de timer een overflow genereert.
4. Het alarmsignaal zal klinken uit een speaker die via een transistor op poort P3 zal zijn aangesloten. De frequentie van dit geluidssignaal is 400 Hz. Het geluidssignaal zal 2 sec. lang klinken;
5. Na het klinken van het alarm wordt de wekker automatisch gereset, zodat deze weer zal beginnen met tellen zodra de TEST toets wordt ingedrukt;
6. De maximale kooktijd begrenzen we op 10 seconden.

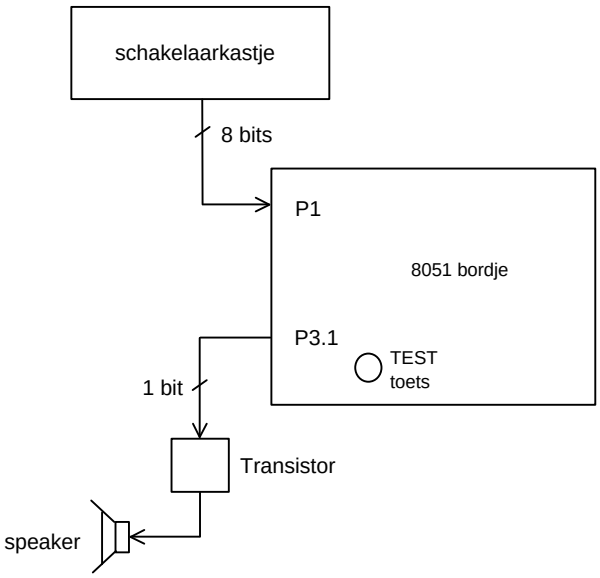


Fig. 11.1. Schema van de kookwekker.

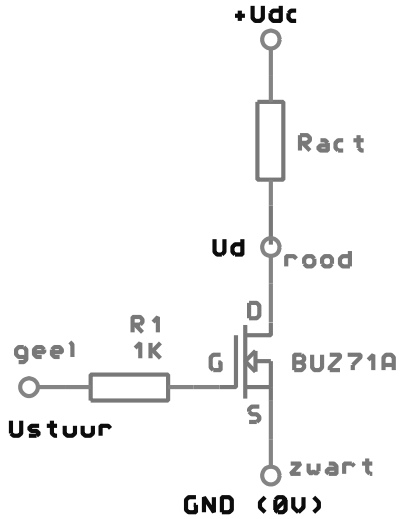


Fig. 11.2. Aansluitschema van de MOSFET.

Zie Figuur 11.1 voor een schematische weergave van de opstelling. Het geluidssignaal wordt gemaakt met een luidspreker. De stroom door de luidspreker wordt met behulp van een MOSFET in- en uitgeschakeld. De sturing van de MOSFET wordt geleverd door de 8051 via P3.1. Een hoog signaal zet de transistor open, zodat een stroom door de luidspreker gaat lopen. Er ontstaat een geluidssignaal met een frequentie van 400 Hz door op P3.1 een bloksignaal met die frequentie te zetten. In Figuur 11.2 is weergegeven hoe de transistor en de luidspreker moeten worden aangesloten. De spanning Udc is 5 Volt en R_{act} is de luidspreker.

11.2.1 Structuur van het programma

Fig. 11.3. Programma Structuur Diagram (PSD) voor de kookwekker.

In Figuur 11.3 is de structuur van het programma voor de kookwekker weergegeven door middel van een Programma Structuur Diagram (PSD). Het hoofdprogramma bestaat uit een eindeloze lus met daarin twee onderdelen: het aftellen van de kooktijd met behulp van een timer (KOOKTIJD) en het genereren van het geluidssignaal met behulp van programmatische wachtlussen (ZOEMER).

De module KOOKTIJD gebruikt twee tellervariabelen om de gewenste tijdintervallen voor de kooktijd te realiseren.Variabele *insteltijd* telt seconden en variabele *deler* telt timer overflows. De timer is zo ingesteld dat deze iedere 50 ms een overflow genereert. In deze module wordt de timer overflow bit “ge-polled”, d.w.z. het programma wacht net zo lang totdat er een overflow volgt. Bij elke overflow wordt de deler met 1 verminderd, zodat als de deler nul bereikt, er één seconde verstreken is. Na elke overflow moet de timer worden gestopt, opnieuw geladen met de juiste instelwaarde en weer gestart. Na elke seconde wordt de variabele insteltijd verlaagd en de deler weer opnieuw op 20 gezet, zodat bij het nul worden van insteltijd de gehele kooktijd is verstreken.

De module ZOEMER genereert het geluidssignaal. De frequentie van het geluidssignaal is 400 Hz, de periodeduur is dus 2.5 ms. Tijdens de eerste helft van de periode is het signaal hoog, daarna is het gedurende een halve periode laag. Het geluid moet 2 seconden duren, dus er moeten 800 perioden worden gegenereerd. Daarvoor worden twee tellers gebruikt (teller1 en teller2). Merk op dat voor het wachten niet een timer maar een software-wachtlus wordt gebruikt.

Bestudeer nu het hoofdstuk over timers elders in deze handleiding. Beantwoord daarbij voor jezelf de volgende vragen:
1. Welke timermodus kies je voor het aftellen van de kooktijd?
2. Welke instelwaarde voor de timer gebruik je om een interval van 50 ms te realiseren?
3. Wat is de executietijd van een machinecyclus?
4. Hoe kun je een wachtlus maken die een executietijd heeft van 1,25 ms?

11.2.2 Nuttige tips bij het instellen van de timer

De assembler kent de adressen van de registers niet, die moet je zelf dus nog opgeven. Gebruik de adressen uit de onderstaande tabel:

Registernaam	Geheugenadres
TCON	88H
TMOD	89H
TL0	8AH
TL1	8BH
TH0	8CH
TH1	8DH

Tabel 11.1. Geheugenadressen van de timerregisters in de 8051.

11.3 Kookwekker met interrupts (opdracht B1b)

De specificaties van de kookwekker in deze opdracht zijn gelijk aan die in de eerste opdracht. Het programma van de eerste opdracht moet zodanig worden gewijzigd dat de timer overflow een interrupt genereert. De interruptroutine vervult de functie van de module KOOKTIJD. Behalve de toevoegingen om interrupts mogelijk te maken zijn er waarschijnlijk nog een paar kleine aanpassingen nodig zowel in het hoofdprogramma als in de interruptroutine.

Bestudeer nu de paragrafen over interrupts elders in deze handleiding. Beantwoord daarbij voor jezelf de volgende vragen:
1. Wat is het interrupt enable bit van de gebruikte timer?
2. Wanneer moeten interrupts worden aangezet en weer uitgezet?
3. Heeft het gebruik van interrupts in deze opdracht een voordeel t.o.v. polling? Waarom niet/wel?

Maak nu het programma op basis van interrupts.

12 Opdracht B2 – Self-timer (B2a en B2b)

Vereiste voorbereiding vóór de practicumsessie:
- Leeswerk: hoofdstukken 5 en 6 resp. over timers en interrupts
- Bestudeer het Programma Structuur Diagram van de eerste B-opdracht grondig.

12.1 Inhoud van deze opdracht

- Je moet vóór de practicumsessie de vereiste voorbereiding hebben uitgevoerd;
- De eerste opdracht (B2a) bestaat uit het omzetten van een algoritme, weergegeven in een PSD, naar 8051 assemblytaal;
- Het te schrijven programma voor de eerste opdracht zal gebruik maken van de timers die in de 8051 microcontroller zijn ingebouwd.
- De tweede opdracht (B2b) bestaat uit het aanpassen van de eerste opdracht, zodanig dat het programma werkt op basis van interrupts.

12.2 De self-timer met timer polling (B2a)

De self-timer, zoals deze in de meeste fototoestellen aanwezig is, werkt als volgt:
1. Na een druk op de knop begint de self-timer met het aftellen;
2. Daarbij verschijnen de cijfers 9 t/m 0 op de display en klinkt elke seconde een piepje uit een luidspreker;
3. Zodra de self-timer de stand 0 heeft bereikt geeft de self-timer een puls af op een uitgangspin;
4. Wordt daarna weer op de knop gedrukt, dan begint het aftellen opnieuw.

Met het 8051 bordje zullen we de self-timer als volgt implementeren:
1. De TEST toets zal als startknop van de self-timer fungeren;
2. Zolang de TEST toets niet wordt ingedrukt wordt de punt op het display getoond;
3. Het geluidssignaal klinkt uit een luidspreker die via een transistor op poort P3.1 moet worden aangesloten. De frequentie van dit geluidssignaal is 500 Hz. Het geluidssignaal moet 100 milliseconden lang klinken;
4. Als display gebruiken we het 7-segmentsdisplay op het 8051 bordje dat is aangesloten op poort P1;
5. De puls moet worden gegenereerd op pin P3.5. De puls bestaat uit een uitgangssignaal dat gedurende 0.5 s hoog blijft en daarna weer laag wordt.

Figuur 12.1 laat zien hoe ingangs- en uitgangssignalen moeten worden aangesloten.
Het geluidssignaal wordt gemaakt met een luidspreker. De stroom door de luidspreker wordt met behulp van een MOSFET in- en uitgeschakeld. De sturing van de MOSFET wordt geleverd door de 8051 via P3.1. Een hoog signaal zet de transistor open, zodat een stroom door de luidspreker gaat

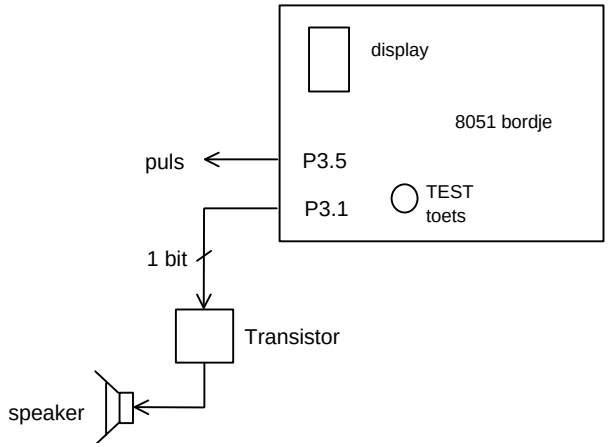


Fig. 12.1. Schema van de self-timer.

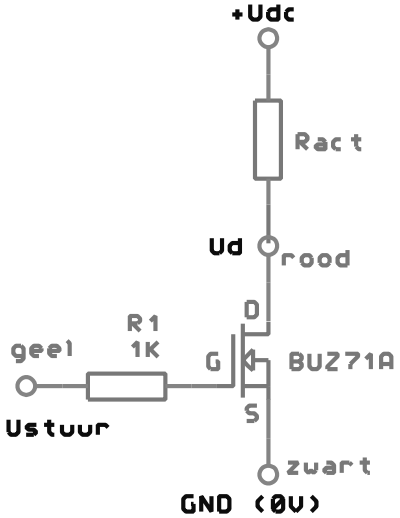
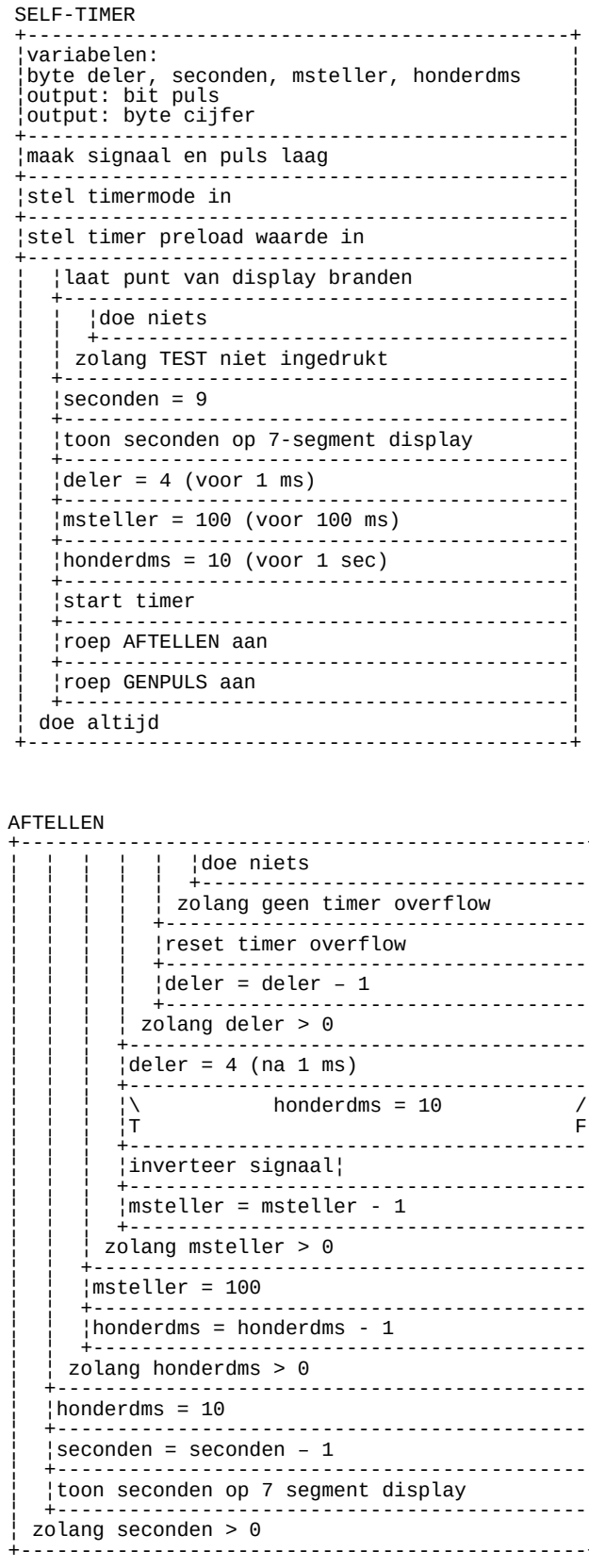


Fig. 12.2. Aansluitschema van de MOSFET.

lopen. Er ontstaat een geluidssignaal met een frequentie van 500 Hz door op P3.1 een bloksignaal met die frequentie te zetten. In Figuur 12.2 is weergegeven hoe de transistor en de luidspreker moeten worden aangesloten. De spanning Udc is 5 V en R_{act} is de luidspreker.

12.2.1 Structuur van het programma



Figuur 12.3. toont het Programma Structuur Diagram van het programma. Het bestaat uit een eindeloze lus met daarin twee onderdelen: het aftellen van de aflooptijd met behulp van een timer en het genereren van een puls met behulp van programmatische wachtlussen.
De tellervariabelen *deler*, *msteller*, *honderdms* en *seconden* zijn nodig om de gewenste tijdintervallen voor de self-timer te realiseren. Deler telt timer-overflows van 0.25 ms, msteller telt intervallen van 1 ms, honderdms telt intervallen van 100 ms en variabele seconden telt seconden.

In module AFTELLEN wordt de aflooptijd van de self-timer afgeteld. Gedurende de eerste 100 ms van ieder interval van een seconde wordt het uitgangssignaal voor de luidspreker elke milliseconde geïnverteerd. De frequentie van het geluidssignaal wordt dan 500 Hz. De tijdsduur van het geluidssignaal is 100 ms. In deze module wordt de timer overflow bit “ge-polled”, d.w.z. het programma wacht net zo lang totdat er een overflow volgt.

De module GENPULS genereert een puls met een pulsduur van 500 ms. Het wachten geschiedt met een wachtlus zoals die uit opdracht A.

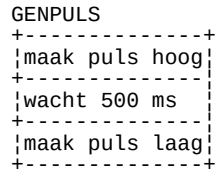


Fig. 12.3. Programma Structuur Diagram voor de self-timer.

Bestudeer nu het hoofdstuk over timers elders in deze handleiding. Beantwoord daarbij voor jezelf de volgende vragen:
1. Welke timermode kies je voor het aftellen van de ontsteektijd?
2. Wat is de executietijd van een machinecyclus?
3. Hoe kun je een wachttijd maken die een executietijd heeft van 500 ms zonder gebruik te maken van een timer?

12.2.2 Nuttige tips bij het instellen van de timer

De assembler kent de adressen van de registers niet, die moet je zelf dus nog opgeven. Gebruik de adressen uit de onderstaande tabel:

Registernaam	Geheugenadres
TCON	88H
TMOD	89H
TL0	8AH
TL1	8BH
TH0	8CH
TH1	8DH

Tabel 13.1. Geheugenadressen van de timerregisters in de 8051.

Bij gebruik van de display is het niet voldoende om het te tonen cijfer simpelweg naar poort P1 te kopiëren. Dit cijfer zal moeten worden geconverteerd naar 7-segmentscode, zodat bij elk cijfer de juiste segmentjes gaan branden. Met behulp van Figuur 3.1 zijn de 7-segmentscodes voor de cijfers 0 t/m 9 eenvoudig te bepalen als je in acht neemt dat een 0 op een segmentje het segmentje laat branden. Een handige conversiemethode is de " look-up" methode. De binaire cijfercode wordt gebruikt als index van een tabel in het externe programmageheugen waarin de 10 benodigde 7-segmentcodes zijn geplaatst. De instructie `MOVC A,@A+DPTR` is heel geschikt om een element van deze tabel uit te lezen. Bekijk het volgende programmasegment:

```
            MOV       A,CIJFER       ;zet cijfer klaar in A
            MOV       DPTR,#TABEL    ;zet pointer naar tabel
            MOVC      A,@A+DPTR      ;haal 7-segmentcode op
            ...
TABEL       DB        ???            ;7-segmentcode voor 0
            //
            DB        ???            ;7-segmentcode voor 9
```

De tabel wordt achter het programma geplaatst.

12.3 Self-timer met interrupts (B2b)

De specificaties van de self-timer in deze opdracht zijn gelijk aan die in de eerste opdracht. Het programma van de eerste opdracht moet zodanig worden gewijzigd dat de timer overflow een interrupt genereert. De interruptroutine vervult de functie van de module AFTELLEN. Behalve de toevoegingen om interrupts mogelijk te maken zijn er waarschijnlijk nog een paar kleine aanpassingen nodig in het hoofdprogramma en in de interruptroutine.

Bestudeer nu de paragrafen over interrupts elders in deze handleiding. Beantwoord daarbij voor jezelf de volgende vragen:
1. Wat is het interrupt enable bit van de gebruikte timer?
2. Wanneer moeten interrupts worden aangezet en weer uitgezet?
3. Heeft het gebruik van interrupts in deze opdracht een voordeel t.o.v. polling? Waarom niet/wel?

Maak nu het programma op basis van interrupts.

13 Opdracht B3 – Sirene (B3a en B3b)

Vereiste voorbereiding vóór de practicumsessie:
- Leeswerk: hoofdstukken 5 en 6 resp. over timers en interrupts
- Bestudeer het Programma Structuur Diagram van de eerste B-opdracht grondig.

13.1 Inhoud van deze opdracht

- Je moet vóór de practicumsessie de vereiste voorbereiding hebben uitgevoerd;
- De eerste opdracht (B3a) bestaat uit het omzetten van een algoritme, weergegeven in een PSD, naar 8051 assemblytaal;
- Het te schrijven programma voor de eerste opdracht zal gebruik maken van de timers die in de 8051 microcontroller zijn ingebouwd.
- De tweede opdracht (B3b) bestaat uit het aanpassen van de eerste opdracht, zodanig dat het programma werkt op basis van interrupts.

13.2 De sirene met timer polling (B3a)

De sirene werkt als volgt:
1. Met een schakelaar kan men kiezen tussen twee sirenemelodieën;
2. De sirene is drietonig en de melodieën ontstaan uit verschillende combinaties van deze tonen;
3. De sirene klinkt tijdens het indrukken van de druktoets TEST, maar de sirenemelodie kan op elk moment gewijzigd worden.

Met het 8051 bordje zullen we de sirene als volgt implementeren:
1. Het schakelaarkastje fungeert als keuzeschakelaar voor de sirenemelodie. Als de schakelaarstand 0 is volgt melodie 2; in alle andere gevallen volgt melodie 1;
2. Melodie 1 bestaat uit een toonserie van 400-600-800 Hz. Elke toon houdt 10 perioden aan;
3. Melodie 2 bestaat uit een toonserie van 600-400-800 Hz. Elke toon houdt 10 perioden aan;
4. Het geluidssignaal klinkt uit een luidspreker die via een transistor op poort P3.1 moet worden aangesloten.

Figuur 13.1 laat zien hoe de ingangs- en uitgangssignalen moeten worden aangesloten.
Het geluidssignaal wordt gemaakt met een luidspreker. De stroom door de luidspreker wordt met behulp van een MOSFET in- en uitgeschakeld. De sturing van de MOSFET wordt geleverd door de 8051 via P3.1. Een hoog signaal zet de transistor open, zodat een stroom door de luidspreker gaat lopen. Er ontstaat een geluidssignaal met een frequentie van bijvoorbeeld 400 Hz door op P3.1 een bloksignaal met die frequentie te zetten. Figuur 13.2 geeft aan hoe de MOSFET moet worden aangesloten. De spanning Udc is 5 V en R_{act} is de luidspreker.

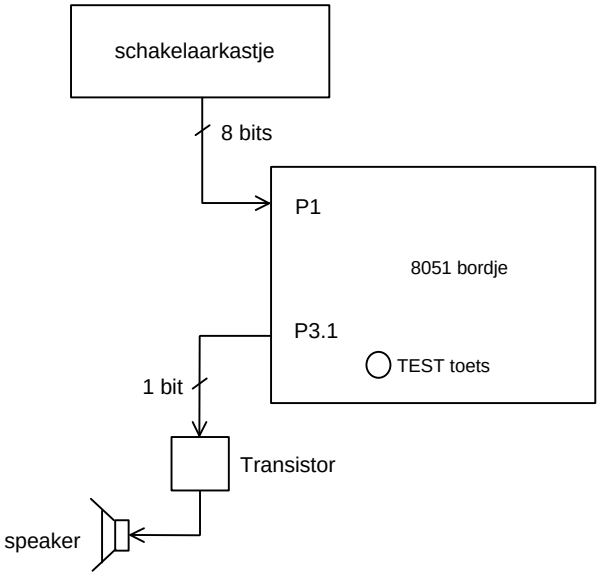


Fig. 13.1. Schema van de sirene.

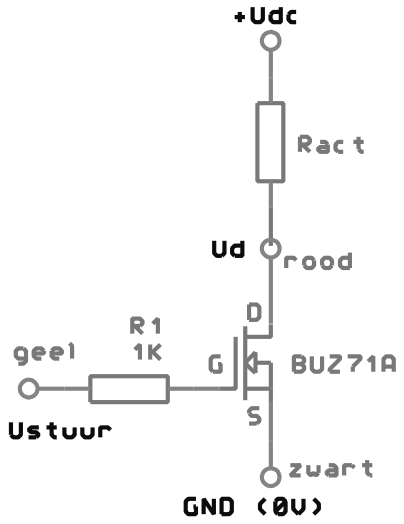


Fig. 13.2. Aansluitschema van de MOSFET.

13.2.1 Structuur van het programma

Figuur 13.3 toont het Programma Structuur Diagram van de sirene. Het hoofdprogramma bestaat uit een oneindige lus waarin de standen van de schakelaars en van de TEST toets worden gelezen. De schakelaarstand bepaalt welke van de twee melodieën gespeeld zal worden. In beide gevallen wordt driemaal een frequentiewaarde ingesteld (door een bepaalde tellerwaarde klaar te zetten) en wordt de TOON subroutine aangeroepen.

De subroutine TOON produceert tien frames van een signaal met één frequentie, die door het hoofdprogramma is ingesteld. We gebruiken het woord *frames* in plaats van perioden omdat het aantal perioden natuurlijk afhankelijk is van de ingestelde frequentie, maar tegelijkertijd willen we alledrie de tonen even lang laten duren. We maken daartoe tien frames van een vaste tijdsduur, zodanig dat de toon totaal 0.5 s duurt. Timer T0 wordt gebruikt om de juiste frequentie van het signaal te verkrijgen doordat deze tijdsduur tussen de signaalwisselingen bepaalt. Timer T1 bepaalt de tijdsduur van één frame (en daarmee het aantal perioden van het signaal dat binnen dat frame past). *Teller* is de variabele die de frames aftelt.

Fig. 13.3. Programma Structuur Diagram van de sirene.

Bestudeer nu de paragrafen over timers elders in deze handleiding. Beantwoord daarbij voor jezelf de volgende vragen:

1. Welke timermodus kies je voor de timers?
2. Wat is de executietijd van een machinecyclus?
3. Welke instelwaarde voor timer T1 gebruik je om een interval van 50 ms te realiseren?
4. Welke instelwaarde voor timer T0 gebruik je bij een gegeven frequentie?
5. Hoe ga je de frequentie-instelling van het hoofdprogramma doorgeven aan de subroutine TOON?

13.2.2 Nuttige tips bij het instellen van de timers

De assembler kent de adressen van de registers niet, die moet je zelf dus nog opgeven. Gebruik de adressen uit de onderstaande tabel:

Registernaam	Geheugenadres
TCON	88H
TMOD	89H
TL0	8AH
TL1	8BH
TH0	8CH
TH1	8DH

Tabel 13.1. Geheugenadressen van de timerregisters in de 8051.

Bij het vullen van de telregisters met de beginwaarde kun je handig gebruiken maken van het 16 bits DPTR Special Function Register. Je kunt de twee bytes waaruit dit register bestaat namelijk afzonderlijk adresseren:

Registernaam	Geheugenadres	Functie
DPL	82H	low byte DPTR
DPH	83H	high byte DPTR

Tabel 13.2. Geheugenadressen van de DPTR bytes in de 8051.

Iets dergelijks kun je ook doen door een 16 bits waarde met het assembler directive DW op te slaan in het programmageheugen. Via het label kun je deze waarde dan weer gebruiken bij een instructie.

13.3 Sirene met interrupts (B3b)

De specificaties van de sirene in deze opdracht zijn gelijk aan die in de eerste opdracht. Het programma van de eerste opdracht moet zodanig worden gewijzigd dat de timer overflows van timers T0 en T1 interrupts genereren. Er komen dus twee interruptroutines die samen de functie van de module TOON vervullen. Behalve de toevoegingen om interrupts mogelijk te maken zijn er waarschijnlijk nog een paar kleine aanpassingen nodig o.a. in het hoofdprogramma.

Bestudeer nu het hoofdstuk over interrupts elders in deze handleiding. Beantwoord daarbij voor jezelf de volgende vragen:

4. Wat zijn de interrupt enable bits van de gebruikte timers?
5. Wanneer moeten interrupts worden aangezet en weer uitgezet?
6. Heeft het gebruik van interrupts in deze opdracht een voordeel t.o.v. polling? Waarom niet/wel?

Maak nu het programma op basis van interrupts.

14 Opdracht C – Seriële communicatie

Vereiste voorbereiding vóór de practicumsessie:
- Leeswerk: De stof in dit hoofdstuk.
- Maak thuis zelf twee Programma Structuur Diagrammen voor de zender en de ontvanger.

14.1 Inhoud van deze opdracht

- Je moet voor de practicumsessie de hierboven beschreven voorbereiding hebben uitgevoerd;
- Voordat je begint met programmeren moet je de thuis voorbereide PSD's door een assistent laten goedkeuren.
- De opdracht bestaat uit vier delen: eerst maak je een zender, vervolgens maak je een ontvanger, daarna voeg je deze samen. Tenslotte laat je het programma werken op basis van interrupts in plaats van op basis van polling.

14.2 Inleiding over I/O

Bij **parallelle** I/O worden de acht bits van een databyte gelijktijdig getransporteerd via een datapoort die met een randapparaat is verbonden. Hiervoor zijn meeraderige kabels met bijbehorende connectors nodig. De transportsnelheid kan hoog zijn. Een transmissiesnelheid van 10 tot 100 kilobytes per seconde is heel normaal.
Bij **seriële** I/O worden databits na elkaar over één signaallijn getransporteerd. Seriële kabels bevatten vaak slechts twee signaallijnen (voor ontvangen en zenden) en een aarddraad. De transportsnelheid is veel lager dan bij parallelle I/O. Ieder bit wordt verzonden in een tijd T. De waarde 1/T wordt de *bitrate* genoemd (de term *baudrate* komt ook veel voor, maar heeft eigenlijk een andere betekenis). De zender en ontvanger moeten vanzelfsprekend met dezelfde bitrate zenden en ontvangen.

14.3 Seriële I/O met de 8051

De practicumcomputer is uitgerust met een seriële poort bestaande uit een inputlijn en een outputlijn. Deze poort is beschikbaar op connector EXP. De signalen op deze pennen hebben CMOS-niveaus, d.w.z. 5 Volt is een logische 1 en 0 Volt is een logische 0.

seriële input	RxD (Receive Data)	poort P3.0	EXP pen 3
seriële output	TxD (Transmit Data)	poort P3.1	EXP pen 4

Om de seriële poort te kunnen instellen en gebruiken is een aantal Special Function Registers aanwezig. SCON (Serial Control) is het register waarmee de poort wordt ingesteld op het gewenste gedrag. Register SBUF (Serial Buffer) bestaat intern uit twee verschillende registers, een voor ontvangen en een voor verzenden. Ontvangen bytes kunnen worden gelezen uit SBUF. Te verzenden bytes worden geschreven in dit register.
De seriële poort kan in vier modi (functies) werken. De modi zijn instelbaar met de bits SM0 en SM1 (Serial Mode) in register SCON. Alleen mode 1 zal hier worden besproken.

SCON (Serial Control Register) (geheugenadres 98H)

7	6	5	4	3	2	1	0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

14.3.1 Serial Mode 1

In mode 1 werkt de UART als een 8-bit UART met variabel bitrate. De modebits zijn: SM0=0, SM1=1 en SM2=0). Er wordt gezonden zodra er een instructie wordt uitgevoerd die SBUF als bestemming heeft. De hardware voegt automatisch een startbit en een stopbit toe. Het eerste bit dat wordt verzonden via pen TxD is een startbit (logisch 0). Dan volgen de acht databits, bit nummer 0 als

eerste. Het laatste bit is een stopbit (logisch 1).
De ontvanger moet worden aangezet door het bit REN (Receiver Enable) in register SCON op 1 te zetten. De ontvanger start bij een via pen RxD binnenkomend startbit. De 8 databits kunnen worden gelezen uit register SBUF. Het ontvangen stopbit wordt in bit RB8 van SCON gezet.
Figuur 14.1 laat zien hoe een "serial frame" er uit ziet. De databits kunnen 0 of 1 zijn.

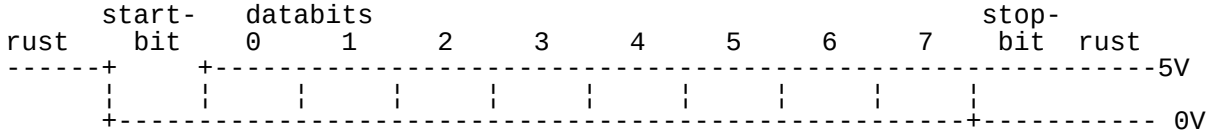


Fig. 14.1. Serial frame met start- en stopbits en databits.

14.3.2 De bitrate in mode 1

Bij mode 1 wordt Timer 1 in de 8051 gebruikt om de gewenste bitrate te verkrijgen. De bitrate is afhankelijk van de Overflow Rate van Timer 1 (dit is het aantal keren per seconde dat de timer overloopt). De bitrate is ook afhankelijk van bit SMOD in register PCON.

PCON (Power Control Register) (geheugenadres 87H)

7	6	5	4	3	2	1	0
SMOD	--	--	--	--	--	--	--

Als Timer 1 wordt ingesteld als 8-bit "timer" in de auto-reload mode dan geldt de formule:

$$256-(TH1) = (2^{SMOD}/32) \times \text{kristal-frequentie} / (12 \times \text{bitrate}))$$

Voorbeeld:
TL1 en TH1 worden gevuld met de waarde F0H. De timer wordt gestart. TL1 zal na 16 cycli van de ingangsklok overlopen. TL1 wordt dan automatisch gevuld met de inhoud van TH1. De timer telt door en zal weer na 16 klokpulsen aflopen, enz.

14.3.3 Het instellen van de seriële poort

Als we met Timer 1 in de "8-bit auto-reload mode" de bitrate willen instellen op 1200, dan moet het volgende gebeuren:

- SMOD instellen op 0 of 1 (PCON.7, niet bit adresseerbaar!).
- Timer 1 instellen op 8-bit auto reload (TMOD bits 4-7).
- TH1 en TL1 laden met een berekende waarde. 256-(TH1)=52,083 afgerond 52, fout 0,16%, TH1=TL1=204. SMOD is hier 1.
  De afrondingsfout mag niet groter zijn dan 5%.
- De seriële mode op 1 instellen (SCON.7 en SCON.6).
- Timer 1 starten (TCON.6).
- Receiver aanzetten (SCON.4).

14.3.4 Zenden en ontvangen

Als het programma een byte wil **zenden** moet het volgende gebeuren:

- Wacht totdat bit TI in register SCON geset is (logisch 1 geworden). Een 1 in TI betekent dat SBUF leeg is geworden. Er kan dan een volgend byte verzonden worden. Polling en interrupt zijn twee methoden om op het 1 zetten van TI te reageren.
  Let op: na het resetten van de computer is TI=0. Voor het verzenden van het allereerste byte moet TI op 1 gezet worden.
- Reset dan TI (0 maken).
- Schrijf het te verzenden byte naar SBUF. De hardware doet de rest.

Als het programma een byte wil **ontvangen** moet het volgende gebeuren:

- Wacht totdat bit RI in register SCON geset is (logisch 1 geworden). Een 1 in RI betekent dat er een byte is ontvangen. Polling en interrupt zijn twee methoden om op het 1 zetten van RI te reageren.
- Reset dan RI (0 maken).
- Lees de ontvangen byte uit SBUF.

14.4 De opdracht

De seriële poorten van twee computers worden met elkaar verbonden. Figuur 14.2 geeft dit weer. Let op de kruisverbinding tussen RxD en TxD en vergeet de GND-draad niet. De computers kunnen berichten naar elkaar verzenden. Een bericht bestaat in deze opdracht uit een reeks tekens die op het zeven-segmentdisplay kunnen worden getoond.
De opdracht bestaat uit het bouwen van een programma met een zender en een ontvanger die onafhankelijk van elkaar kunnen werken. Eerst wordt de zender gebouwd, daarna de ontvanger en vervolgens worden de twee onderdelen samengevoegd. Dit alles zonder gebruik te maken van interrupts.

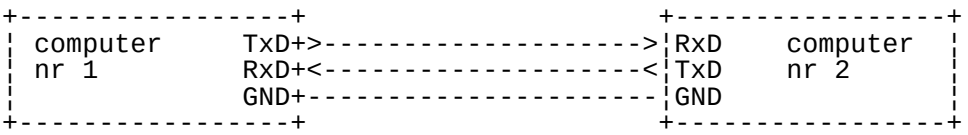


Fig. 14.2. Seriële verbinding tussen twee practicumcomputers.

14.4.1 Opdracht 1: De zender

- Maak eerst een subroutine die alle initialisaties van Timer 1 en de seriële poort uitvoert. De bitrate moet 300 worden.
- Maak dan een subroutine die het byte dat in A staat verzendt.
- Maak een subroutine die 50 milliseconden wacht.
- Maak een programma dat 20 maal per seconde de 7-segmentcode van de letter H verzendt. Zet het programma in EPROM en bekijk met de oscilloscoop het signaal op pen TxD.
- Maak tenslotte een programma dat het bericht H A L L O verstuurt (met spaties tussen de letters) zodra er op de knop TEST is gedrukt. Het einde van de boodschap wordt gemarkeerd door een extra byte met inhoud 00H.

14.4.2 Opdracht 2: De ontvanger

- Maak eerst een subroutine die alle initialisaties van Timer 1 en de seriële poort uitvoert. De bitrate moet 300 worden.
- Maak dan een subroutine die wacht op het binnenkomen van één byte. Het ontvangen byte wordt in A gezet. Daarna keert de subroutine terug.
- Maak een subroutine die een bericht kan ontvangen. Het laatste teken van een bericht heeft de waarde 00H. Het bericht wordt in het interne geheugen gezet met inbegrip van het laatste teken. Daarna keert de subroutine terug.
- Maak een subroutine die 500 milliseconden wacht.
- Maak een subroutine die een bericht dat in het geheugen staat één keer op het display toont. De display-tijd is een teken per halve seconde.
- Maak tenslotte een programma dat een bericht kan ontvangen en op het display tonen. Het programma herhaalt zichzelf.

14.4.3 Opdracht 3: Zend/ontvanger

Voeg de zender en de ontvanger samen tot een enkel programma dat zowel kan zenden als ontvangen.

14.4.4 Opdracht 4: Zend/ontvanger met interrupts

In deze opdracht wordt het programma van opdracht 3 zodanig gewijzigd dat gebruik gemaakt wordt van RI/TI interrupts. Er is slechts 1 systeeminterrupt beschikbaar voor de UART. De interruptroutine moet dus uitzoeken of de interrupt afkomstig is van de receiver (RI=1) of van de transmitter (TI=1).

MAAK EERST EEN PSD VOOR HET HOOFDPROGRAMMA EN DE INTERRUPTROUTINE ALVORENS HET PROGRAMMA TE SCHRIJVEN.

Appendix A: ASCII character codes

HEX	DEC	CHAR	CNTL	HEX	DEC	CHAR	HEX	DEC	CHAR	HEX	DEC	CHAR
00	0	NUL		20	32	SPACE	40	64	@	60	96	`
01	1	SOH	^A	21	33	!	41	65	A	61	97	a
02	2	STX	^B	22	34	"	42	66	B	62	98	b
03	3	ETX	^C	23	35	#	43	67	C	63	99	c
04	4	EOT	^D	24	36	$	44	68	D	64	100	d
05	5	ENQ	^E	25	37	%	45	69	E	65	101	e
06	6	ACK	^F	26	38	&	46	70	F	66	102	f
07	7	BEL	^G	27	39	'	47	71	G	67	103	g
08	8	BS	^H	28	40	(	48	72	H	68	104	h
09	9	HT	^I	29	41	)	49	73	I	69	105	I
0A	10	LF	^J	2A	42	*	4A	74	J	6A	106	j
0B	11	VT	^K	2B	43	+	4B	75	K	6B	107	k
0C	12	FF	^L	2C	44	,	4C	76	L	6C	108	l
0D	13	CR	^M	2D	45	-	4D	77	M	6D	109	m
0E	14	SO	^N	2E	46	.	4E	78	N	6E	110	n
0F	15	SI	^O	2F	47	/	4F	79	O	6F	111	o
10	16	DLE	^P	30	48	0	50	80	P	70	112	p
11	17	DC1	^Q	31	49	1	51	81	Q	71	113	q
12	18	DC2	^R	32	50	2	52	82	R	72	114	r
13	19	DC3	^S	33	51	3	53	83	S	73	115	s
14	20	DC4	^T	34	52	4	54	84	T	74	116	t
15	21	NAK	^U	35	53	5	55	85	U	75	117	u
16	22	SYN	^V	36	54	6	56	86	V	76	118	v
17	23	ETB	^W	37	55	7	57	87	W	77	119	w
18	24	CAN	^X	38	56	8	58	88	X	78	120	x
19	25	EM	^Y	39	57	9	59	89	Y	79	121	y
1A	26	SUB	^Z	3A	58	:	5A	90	Z	7A	122	z
1B	27	ESC		3B	59	;	5B	91	[	7B	123	{
1C	28	FS		3C	60	<	5C	92	\	7C	124	|
1D	29	GS		3D	61	=	5D	93	]	7D	125	}
1E	30	RS		3E	62	>	5E	94	^	7E	126	~
1F	31	US		3F	63	?	5F	95	_	7F	127	DEL

ACK=Acknowledge
BEL=Bell
BS =Backspace
CAN=Cancel
CR =Carriage Return
DC1=Direct Control 1
DC2=Direct Control 2
DC3=Direct Control 3
DC4=Direct Control 4
DLE=Data Link Escape
EM =End of Medium
ENQ=Enquiry
EOT=End Of Transmission
ESC=Escape
ETB=End Transmission Block
ETX=End Text
FF =Form Feed
FS =Form Separator
GS =Group Separator
HT =Horizontal Tab
LF =Line Feed
NAK=Negative Acknowledge
NUL=Null
RS =Record Separator
SI =Shift In
SO =Shift Out
SOH=Start Of Heading
STX=Start Text
SUB=Substitute
SYN=Synchronous Idle
US =Unit Separator
VT =Vertical Tab

Appendix B: Machine codes

In de volgende tabellen wordt een overzicht gegeven van de 8051 instructies. In de kolom INSTRUCTION staat de assembly-notatie. In de kolom OPERATION staat wat er gebeurt als de instructie wordt uitgevoerd. In kolom PSW worden de (mogelijke) veranderingen in het Processor Status Word vermeld. Kolom T geeft het aantal machinecycli en kolom L het aantal bytes waaruit de instructie bestaat. Tenslotte geeft kolom CODE de opcode in binaire vorm (dit is het eerste byte van de instructie). In de tabellen worden de volgende symbolen gebruikt:

Symbool	Betekenis
#	immediate adressering
data	8 bit waarde
data16	16 bit waarde
Rn	R0,R1,R2,R3,R4,R5,R6 of R7 van de huidige geselecteerde registerbank in bank 0 heeft R0 adres 0, R1 adres 1 etc. in bank 1 heeft R0 adres 8, R1 adres 9 etc. etc.
rrr	3 bit waarde 000,001,010,011,100,101,110 of 111
Ri	R0 of R1 van de huidige geselecteerde registerbank
@Ri	indirecte adressering van een interne geheugenlocatie via R0 of R1 0-127 bij een 128 byte intern datageheugen 0-255 bij een 256 byte intern datageheugen
direct	8 bit intern adres geheugenlocatie (0-127) of SFR (128-255)
bit	direct geadresseerd bit in intern datageheugen of in een SFR in datageheugen: bitadressen 0-127 bit=0 betekent geheugenadres 20H bit 0 bit=127 betekent geheugenadres 2FH bit 7 bit=25H.1 betekent geheugenadres 25H bit 1 in een SFR: binaire adressen eindigend op 000 (80H, 88H, 90H etc.) bit=87H betekent SFR adres 80H bit 7 bit=80H.7 betekent SFR adres 80H bit 7
rel	2’s complement 8 bit offset byte relatief t.o.v. het eerste byte van de volgende instructie rel=FEH betekent 2 bytes lager rel=2 betekent 2 bytes hoger
addr11	11 bit adres aaa = A10,A9,A8 A7-A0 in tweede byte

addr16	16 bit adres in tweede (high byte) en derde byte (low byte)
A_n	bit n van A, n = 0-7
A_{0-3}	bit 0 t/m 3 van A
not	logische NOT (inversie of 1's complement) bit: not 0 = 1, not 1 = 0 byte: not 11110000 = 00001111
and	logische AND bit: 0 and X = 0, 1 and 1 = 1 byte: 11110000 and 00110011 = 00110000
or	logische OR bit: 1 or X = 1, 0 or 0 = 0 byte: 11110000 or 11001100 = 11111100
exor	logische EXCLUSIVE OR bit: 0 exor 1 = 1, 0 exor 0 = 0, 1 exor 1 = 0 carry from A6 exor carry from A7 levert "1" op bij ongelijkheid borrow to A6 exor borrow to A7 levert "1" op bij ongelijkheid byte: 11110000 exor 11001100 = 00111100
(direct)	de inhoud van de geheugenplaats met het adres 'direct'
(Ri)	de inhoud van de geheugenplaats met het adres dat in Ri staat.
?	toekenning A? Rn betekent: de inhoud van Rn wordt gekopieerd in A (SP) ? (direct) betekent: de inhoud van adres 'direct' wordt gekopieerd in de geheugenplaats waar de stackpointer naar wijst. C? 0 betekent: C-vlag wordt altijd 0
?	verwisseling A? Rn betekent: de inhouden van Rn en A worden verwisseld

DATA TRANSFER					
INSTRUCTION	**OPERATION**	**PSW**	**T**	**L**	**CODE**
MOV A,Rn	A?Rn	-	1	1	11101rrr
MOV A,direct	A?(direct)	-	1	2	11100101
MOV A,@Ri	A?(Ri)	-	1	1	1110011i
MOV A,#data	A?data	-	1	2	01110100
MOV Rn,A	Rn?A	-	1	1	11111rrr
MOV Rn,direct	Rn?(direct)	-	1	2	10101rrr
MOV Rn,#data	Rn?data	-	1	2	01111rrr
MOV direct,A	(direct)?A	-	1	2	11110101
MOV direct,Rn	(direct)?Rn	-	2	2	10001rrr
MOV direct,direct	(direct)?(direct)	-	2	3	10000101
MOV direct,@Ri	(direct)?Ri	-	2	2	1000011i
MOV direct,#data	(direct)?data	-	2	3	01110101
MOV @Ri,A	(Ri)?A	-	1	1	1111011i
MOV @Ri,direct	(Ri)?(direct)	-	2	2	1010011i
MOV @Ri,#data	(Ri)?data	-	1	2	0111011i
MOV DPTR,#data16	DPTR?data16	-	2	3	10010000
MOVC A,@A+DPTR	A?(A+DPTR)	-	2	1	10010011
MOVC A,@A+PC	A?(A+PC)	-	2	1	10000011
MOVX A,@Ri	A?(Ri)	-	2	1	1110001i
MOVX A,@DPTR	A?(DPTR)	-	2	1	11100000
MOVX @Ri,A	(Ri)?A	-	2	1	1111001i
MOVX @DPTR,A	(DPTR)?A	-	2	1	11110000
PUSH direct	SP?SP+1 (SP)?(direct)	-	2	2	11000000
POP direct	(direct)?(SP) SP?SP-1	-	2	2	11010000
XCH A,Rn	A?Rn	-	1	1	11001rrr
XCH A,direct	A?(direct)	-	1	2	11000101
XCH A,@Ri	A?(Ri)	-	1	1	1100011i
XCHD A,@Ri	A_{0-3}?(Ri_{0-3})	-	1	1	1101011i

ARITHMETIC OPERATIONS					
INSTRUCTION	**OPERATION**	**PSW**	**T**	**L**	**CODE**
ADD A,Rn	A?A+Rn	C?carry from A7 AC?carry from A3 OV?carry from A6 XOR carry from A7	1	1	00101rrr
ADD A,direct	A?A+(direct)	C AC OV	1	2	00100101
ADD A,@Ri	A?A+(Ri)	C AC OV	1	1	0010011i
ADD A,#data	A?A+data	C AC OV	1	2	00100100
ADDC A,Rn	A?A+Rn+C	C AC OV	1	1	00111rrr
ADDC A,direct	A?A+(direct)+C	C AC OV	1	2	00110101
ADDC A,@Ri	A?A+(Ri)+C	C AC OV	1	1	0011011i
ADDC A,#data	A?A+data+C	C AC OV	1	2	00110100
SUBB A,Rn	A?A-Rn-C	C?borrow for A7 AC?borrow for A3 OV?borrow for A6 XOR borrow for A7	1	1	10011rrr
SUBB A,direct	A?A-(direct)-C	C AC OV	1	2	10010101
SUBB A,@Ri	A?A-(Ri)-C	C AC OV	1	1	1001011i
SUBB A,#data	A?A-data-C	C AC OV	1	2	10010100
INC A	A?A+1	-	1	1	00000100
INC Rn	Rn?Rn+1	-	1	1	00001rrr
INC direct	(direct)?(direct)+1	-	1	2	00000101
INC @Ri	(Ri)?(Ri)+1	-	1	1	0000011i
INC DPTR	DPTR?DPTR+1	-	2	1	10100011
DEC A	A?A-1	-	1	1	00010100
DEC Rn	Rn?Rn-1	-	1	1	00011rrr
DEC direct	(direct)?(direct)-1	-	1	2	00010101
DEC @Ri	(Ri)?(Ri)-1	-	1	1	0001011i
MUL AB	unsigned multiply A?low byte of A*B B?high byte of A*B	C?0 OV?1 if A*B>255 OV?0 if A*B?255	4	1	10100100
DIV AB	unsigned divide A?integer part of A/B B?remainder of A/B	C?0 OV?1 if divide by 0 OV?0 if not	4	1	10000100
DA A	A?decimal adjust A following ADD or ADDC in packed BCD format	C?1 if result>99 C?0 if result?99 OV?0	1	1	11010100

BOOLEAN VARIABLE MANIPULATION					
INSTRUCTION	**OPERATION**	**PSW**	**T**	**L**	**CODE**
CLR C	C?0	C?0	1	1	11000011
CLR bit	(bit)?0	-	1	2	11000010
SETB C	C?1	C?1	1	1	11010011
SETB bit	(bit)?1	-	1	2	11010010
CPL C	C?not C	C?not C	1	1	10110011
CPL bit	(bit)?not(bit)	-	1	2	10110010
ANL C,bit	C?C and (bit)	C	2	2	10000010
ANL C,/bit	C?C and not(bit)	C	2	2	10110000
ORL C,bit	C?C or (bit)	C	2	2	01110010
ORL C,/bit	C?C or not(bit)	C	2	2	10100000
MOV C,bit	C?(bit)	C	1	2	10100010
MOV bit,C	(bit)?C	-	2	2	10010010
JC rel	PC?PC+2 if C=1 then PC?PC+rel	-	2	2	01000000
JNC rel	PC?PC+2 if C=0 then PC?PC+rel	-	2	2	01010000
JB bit,rel	PC?PC+3 if (bit)=1 then PC?PC+rel	-	2	3	00100000
JNB bit,rel	PC?PC+3 if (bit)=0 then PC?PC+rel	-	2	3	00110000
JBC bit,rel	PC?PC+3 if (bit)=1 then PC?PC+rel and (bit)?0	-	2	3	00010000

PROGRAM BRANCHING					
INSTRUCTION	**OPERATION**	**PSW**	**T**	**L**	**CODE**
ACALL addr11	PC?PC+2 SP?SP+1 (SP)?PC_{7-0} SP?SP+1 (SP)?PC_{15-8} PC_{10-0}?addr11	-	2	2	aaa10001
LCALL addr16	PC?PC+3 SP?SP+1 (SP)?PC_{7-0} SP?SP+1 (SP)?PC_{15-8} PC_{15-0}?addr16	-	2	3	00010010
RET	PC_{15-8}?(SP) SP?SP-1 PC_{7-0}?(SP) SP?SP-1	-	2	1	00100010
RETI	PC_{15-8}?(SP) SP?SP-1 PC_{7-0}?(SP) SP?SP-1	-	2	1	00110010
AJMP addr11	PC?PC+2 PC_{10-0}?addr11	-	2	2	aaa00001
LJMP addr16	PC_{15-0}?addr16	-	2	3	00000010
SJMP rel	PC?PC+2 PC?PC+rel	-	2	2	10000000
JMP @A+DPTR	PC?A+DPTR	-	2	1	01110011
JZ rel	PC?PC+2 if A=0 then PC?PC+rel	-	2	2	01100000
JNZ rel	PC?PC+2 if A<>0 then PC?PC+rel	-	2	2	01110000
CJNE A,direct,rel	PC?PC+3 if A<>(direct) then PC?PC+rel if A<(direct) then C?1 else C?0	C	2	3	10110101
CJNE A,#data,rel	PC?PC+3 if A<>data then PC?PC+rel if A<data then C?1 else C?0	C	2	3	10110100
CJNE Rn,#data,rel	PC?PC+3 if Rn<>data then PC?PC+rel if Rn<data then C?1 else C?0	C	2	3	10111rrr

CJNE @Ri,#data,rel	PC?PC+3
	if (Ri)<>data then PC?PC+rel
	if (Ri)<data then C?1 else C?0
		C	2	3	1011011i

DJNZ Rn,rel	PC?PC+2 Rn?Rn-1 if Rn<>0 then PC?PC+rel	-	2	2	11011rrr
DJNZ direct,rel	PC?PC+3 (direct)?(direct)-1 if (direct)<>0 then PC?PC+rel	-	2	3	11010101

LOGICAL OPERATIONS FOR BYTE VARIABLES					
INSTRUCTION	**OPERATION**	**PSW**	**T**	**L**	**CODE**
ANL A,Rn	A?A and Rn	-	1	1	01011rrr
ANL A,direct	A?A and (direct)	-	1	2	01010101
ANL A,@Ri	A?A and (Ri)	-	1	1	0101011i
ANL A,#data	A?A and data	-	1	2	01010100
ANL direct,A	(direct)?(direct) and A	-	1	2	01010010
ANL direct,#data	(direct)?(direct) and data	-	2	3	01010011
ORL A,Rn	A?A or Rn	-	1	1	01001rrr
ORL A,direct	A?A or (direct)	-	1	2	01000101
ORL A,@Ri	A?A or (Ri)	-	1	1	0100011i
ORL A,#data	A?A or data	-	1	2	01000100
ORL direct,A	(direct)?(direct) or A	-	1	2	01000010
ORL direct,#data	(direct)?(direct) or data	-	2	3	01000011
XRL A,Rn	A?A exor Rn	-	1	1	01101rrr
XRL A,direct	A?A exor (direct)	-	1	2	01100101
XRL A,@Ri	A?A exor (Ri)	-	1	1	0110011i
XRL A,#data	A?A exor data	-	1	2	01100100
XRL direct,A	(direct)?(direct) exor A	-	1	2	01100010
XRL direct,#data	(direct)?(direct) exor data	-	2	3	01100011
CLR A	A?0	-	1	1	11100100
CPL A	A_n?not A_n n=0-7	-	1	1	11110100
RL A	A_{n+1}?A_n n=0-6 A_0?A_7	-	1	1	00100011
RLC A	A_{n+1}?A_n n=0-6 A_0?C C?A_7	-	1	1	00110011
RR A	A_n?A_{n+1} n=0-6 A_7?A_0	-	1	1	00000011
RRC A	A_n?A_{n+1} n=0-6 A_7?C C?A_0	-	1	1	00010011
SWAP A	A_{3-0}?A_{7-4}	-	1	1	11000100
NOP	No operation	-	1	1	00000000

Appendix C: Voorbeeld coderingsformulier

Programma: Rollend Segment

ADRES	BYTE 1	BYTE 2	BYTE 3	LABEL	OPCODE	OPERAND
0000	74	FE			MOV	A,#11111110B
0002	23			START1	RL	A
0003	F5	90			MOV	P1,A
0005	12	00	0D		LCALL	WACHT
0008	20	B2	F7		JB	P3.2,START1
000B	80	FE		EINDE	SJMP	EINDE
000D	78	FF		WACHT	MOV	R0,#0FFH
000F	79	FF		BUITLUS	MOV	R1,#0FFH
0011	19			BINLUS	DEC	R1
0012	B9	00	FC		CJNE	R1,#00H,BINLUS
0015	18				DEC	R0
0016	B8	00	F6		CJNE	R0,#00H,BUITLUS
0019	22				RET	

Appendix D: Coderingsformulier voor opdracht A

Programma: Opdracht A1

ADRES	BYTE 1	BYTE 2	BYTE 3	LABEL	OPCODE	OPERAND
				START	MOV	A,#0B6H
					MOV	P1,A
				LUS1	JB	P3.2,LUS1
					CPL	A
					MOV	P1,A
				LUS2	JNB	P3.2,LUS2
					SJMP	LUS1

Programma:
