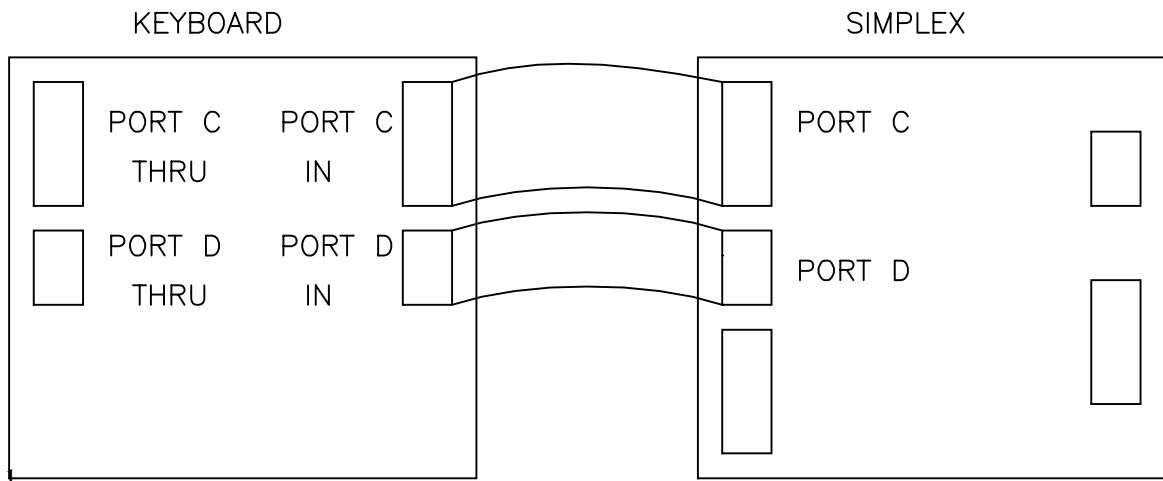


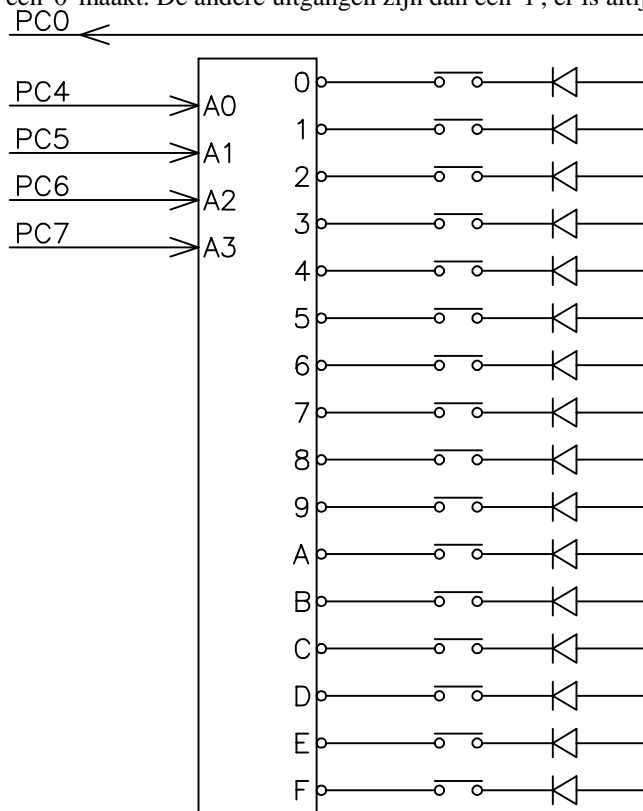
Het toetsenbord wordt op de SIMPLEX aangesloten via poort C en poort D.



Poort D wordt alleen gebruikt om het toetsenbord van voeding te voorzien. Voor de besturing en het uitlezen van het toetsenbord is poort C gebruikt.

Beide poorten zijn aan de linkerkant van het toetsenbord opnieuw op connectors beschikbaar. Hierdoor kunnen andere uitbreidingen eenvoudig worden aangesloten.

Het toetsenbord is opgebouwd rond een circuit dat, afhankelijk van 4 ingangssignalen, één van de 16 uitgangen een '0' maakt. De andere uitgangen zijn dan een '1', er is altijd maar één uitgang tegelijkertijd een '0'.



Door met behulp van 4 bits van poort C een van de uitgangen laag te maken, zal één van de toetsen aan één kant met een '0' verbonden zijn. Wanneer de toets wordt ingedrukt, wordt de andere kant van de toets ook met een '0'

verbonden, waardoor het uitgangssignaal een '0' zal zijn.

Door achtereenvolgens elk van de 16 toetsen op deze wijze af te vragen, kan voor elk van de toetsen worden gelezen of de toets is ingedrukt dan wel losgelaten.

De relatie tussen de bitcombinatie op poort C en de toets die wordt afgevraagd is:

Toets:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
PC7	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
PC6	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
PC5	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
PC4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Een voorbeeld programma is hieronder gegeven. Het programma leest het toetsenbord, en verstuurt vervolgens een 16-bits waarde naar de PC via de seriële poort. Elk bit in dit getal geeft de toestand van één van de toetsen aan, een '1' betekent dat de toets niet ingedrukt is, en een '0' betekent dat de toets is ingedrukt.

LET OP: De bits PC7, PC6, PC5 en PC4 worden ook gebruikt voor het LCD display.

```

*****
* definieren van de geheugen map
*****
        incl "map512.asm"

*****
* start van het programma
*****
PROGRAM          space          |kies het programma-gebied

reset            equ $           |na reset begint de micro op deze plaats
                lds #stackend    |begin met de stackpointer te laden

                ldx #databeg     |en zet dan het volledige datagebied op 00
clearram        clr 0,x
                inx
                cpx #dataend
                bls clearram

                ldx #regsbeg     |laat IX op de bank met I/O registers wijzen

*****
* modules
*****

*****
* seriële poort
*****
PROGRAM          space
clistart        equ $           |het beginadres van deze module

* initialisatie
PROGRAM          space

* initialiseer de seriële poort
                ldab #(scpl+scp0)
                stab baud        |kies 9600 BAUD ( 8MHz kristal)
                ldab #(te or re)
                stab sccr2       |en zet zender- en ontvanger aan
                bra cliend       |einde van de initialisatie

***** subroutines voor de seriële poort
PROGRAM          space

* kijk of er een karakter ontvangen is
* zoja, zet dan de zero-vlag op '0'
serincheck      equ $
                pshb
                ldab scsr        |lees de status vlaggen
                andb #rdrf       |en test op het 'receiver full' bit
                pulb
                rts

* kijk of de seriële poort gereed is om een teken te versturen
* zoja, zet dan de zero-vlag op '0'
seroutcheck     equ $
                pshb
                ldab scsr        |lees de status vlaggen

```

```

        andb #tc          |en test op het 'transmitter empty' bit
        pulb
        rts

* haal een teken op uit de ontvanger, en zet het in accumulator B
getchar          equ $
        bsr serincheck   |wacht totdat er een teken ontvangen is
        beq getchar
        ldab scdr        |en lees het teken uit het ontvanger-register
        rts

* verstuur een teken via de seriële poort
putchar          equ $
        bsr seroutcheck  |wacht totdat de zender gereed is
        beq putchar
        stab scdr        |en zet dan het teken in het verzend-register
        rts

cliend           equ $          |einde van de cli module

* het aantal bytes dat deze module nodig heeft in het programma gebied
clisize          equ cliend-clistart

```

```

*****
* real-time interrupt
*****
PROGRAM          space
rtistart         equ $          |het beginadres van deze module
rtirate          equ 5          |het aantal interrupts dat geteld wordt

DATA             space
rticount         rmb 1          |deze teller wordt bij elke interrupt verhoogd

* initialisatie van de real-time interrupt
PROGRAM          space

                ldx #regsbeg    |IX wijst naar de I/O registers
                ldab #$7E       |vul de entry in de interrupt-tabel in
                stab rtiint      |met een sprong-instructie naar de routine
                ldd #rtiintentry
                std rtiint+1     |die de interrupt afhandelt

                bset pactl-regsbeg,x,(rtrl or rtr0)
                                |zet de real-time interrupt tijd op 32.768ms
                                |met 8MHz Kristal (E / 2^16)
                bset tmsk2-regsbeg,x,rtii
                                |laat de interrupts door
                bra rtiend       |einde van de initialisatie

* subroutine voor de afhandeling van de real-time interrupt
PROGRAM          space

rtiintentry equ $
                ldab #rtif
                stab tflg2       |reset de interrupt-vlag
                inc rticount     |en tel de interrupt
                rti

***** real-time interrupt interface routines
PROGRAM          space

* test op time-out. zet de zero-vlag indien er een time-out is.
checktimeout     equ $
                pshb
                sei              |houd interrupts tijdelijk tegen
                ldab rticount    |kijk hoe vaak er een interrupt was
                subb #rtirate    |indien er minimaal 'rtirate' interrupts
                blo checkrti9     |zijn geweest, dan is er een time-out en moet
                stab rticount    |'rtirate' van de teller worden afgetrokken
                clrb             |daarnaast moet de zero-vlag gezet worden
checkrti9        cli            |daarna kunnen de interrupts weer worden
                pulb             |doorgelaten
                rts

rtiend           equ $          |einde van de real-time interrupt module

* het aantal bytes dat deze module nodig heeft in het programma gebied
rtisize          equ rtiend-rtistart

```

```

*****
* toetsenbord
*****
PROGRAM          space
keybostart equ $          |het beginadres van deze module

* initialisatie van de toetsenbord routines
PROGRAM          space
                bra keyboend      |einde van de initialisatie

* Subroutine voor het afvragen van het toetsenbord
* Laat een 16-bits waarde in (D) achter. Elk bit in (D) komt overeen met
* een toets: bit0 met toets0, etc.
* Een '0' in een bit betekent dat de toets ingedrukt is.
PROGRAM          space
leestoetsen equ $
DATA             space
toetsteller rmb 1          |16 toetsen af te vragen
toetswaarden  rmb 2          |16 bits voor 16 toetsen
PROGRAM          space
                tpa
                psha
                sei              |zet interrupts tijdelijk af
                ldab ddrc        |bewaars huidige waarde van ddrc op de stack
                pshb
                orab #(bit4 or bit5 or bit6 or bit7)
                andb #not( bit0)
                stab ddrc        |maak bit4,5,6,7 outputs en bit0 een input
                ldab portc       |bewaars oude waarde van portc op de stack
                pshb
                clr toetsteller  |nog geen toetsen gelezen
leestoetsen0    ldab portc      |kies een van de toetsen
                andb #low(not( bit4 or bit5 or bit6 or bit7))
                orab toetsteller
                stab portc       |een van de toetsen is nu geactiveerd
                ldab portc       |lees de toets terug: een '0' op bit0
                rorb              |betekent een ingedrukte toets.
                ror toetswaarden
                ror toetswaarden+1
                ldab toetsteller
                addb #$10         |naar de volgende toets
                stab toetsteller
                bne leestoetsen0
                pulb
                stab portc       |zet oude waarde terug in portc
                pulb
                stab ddrc        |zet oude waarde terug in ddrc
                pula
                tap              |zet interrupts terug in de oude toestand
                ldd toetswaarden
                rts

keyboend equ $          |einde van de toetsenbord module

* het aantal bytes dat deze module nodig heeft in het programma gebied
keybosize equ keyboend-keybostart

```

```

*****
* het hoofdprogramma
*****
PROGRAM          space
                 bra main0

* subroutine om een byte (in B) te verzenden in 8 enen of nullen
sendbyte        equ $
                psha
                tba
                pshx
                pshb
                ldx #8           |er zijn 8 bits te versturen
sendbyte0       ldab #'0'       |neem aan dat er een '0' verstuurd moet
                tsta           |verstuur het meest linkse bit van A
                bpl sendbyte1    |dit is een '1' als A negatief is, en anders
                ldab #'1'       |een '0'
sendbyte1       jsr putchar
                lsla           |schuif het volgende bit naar links
                dex            |er is een bit gedaan
                bne sendbyte0    |ga door met zenden totdat alle 8 bits gedaan
                pulb
                pulx
                pula
                rts

main0           jsr leestoetsen |lees het toetsenbord
                pshb
                tab
                jsr sendbyte     |verstuur de eerste 8 bits
                pulb
                jsr sendbyte     |en dan de volgende 8 bits
                ldab # $0D       |carriage-return
                jsr putchar
                ldab # $0A       |line-feed
                jsr putchar
main1           jsr checktimeout
                bne main1       |wacht op de volgende time-out
                bra main0       |blijf dit herhalen
                end

```