Technische beschrijving van de tiepstem

Instituut voor Perceptie Onderzoek
Postbus 513, 5600 MB Eindhoven

RD/rd 86/16
10.09.1986

Rapport no. 548

Technische beschrijving van de tiepstem

R.J.H. Deliege

INHOUD

1 Inleiding.

In het projekt "een eenvoudige spraaksynthetisator als hulpmiddel voor spraakgehandicapten" is het de bedoeling te komen tot een prototype van een communicatiehulpmiddel op basis van tekst-naar-spraak omzetting.
In eerste instantie is programmatuur ontwikkeld voor tekst-naar-spraak omzetting en spraakuitgifte met behulp van difonen op een groot computersysteem (VAX). Omdat de volledige tekst-naar-spraak omzetting een te gecompliceerd probleem is om binnen dit project op te lossen, is uitgegaan van een min of meer fonetische spelling als invoer. Ook is in deze programmatuur het automatisch genereren van intonatie patronen aan de hand van klemtoon markeerders opgenomen.
Voor de realisatie van een eerste prototype van een spraakhulpmiddel voor gehandicapten werd gedacht aan implementatie van deze software op een standaard microprocessor board (VME board met 68000 processor). Als in- en uitvoer orgaan zou dan een personal computer gebruikt worden. Deze ideen zijn vastgelegd in eisenblad nr. 106 [2].
Dit eisenblad was echter reeds voor de realisatie van een proefmodel achterhaald. Dit was gedeeltelijk het gevolg van technische ontwikkelingen en gedeeltelijk vanwege het wisselen van de projekt-uitvoerder.
Het in dit eisenblad reeds genoemde Elcoma diphone board was intussen namelijk gereed gekomen. Dit board bevat op één eurokaart alles nodig voor het spraaksynthese gedeelte van het hulpmiddel. Als invoer dienen ASCII karakters (via een seriele of parallelle ingang). De gebruikte (semi-fonetische) spelling is dezelfde als gebruikt bij de VAX simulatie [2]. De leestekens (?) zijn ook in dit ontwerp geimplementeerd. De nieuwe difoonverzameling en cijfergrammatica zijn niet geimplementeerd vanwege tijdgebrek. De uitgebreide functie's (functietoetsen) zijn niet geimplementeerd omdat deze alleen voor een specifieke toepassing (communicatiehulpmiddel) bedoeld zijn. Duurregels waren nog niet voldoende uitontwikkeld om toe te kunnen passen.
Al met al kunnen we dus stellen dat het beschikbare diphone board aan de gestelde eisen voor het spraaksynthese gedeelte van het hulpmiddel voldeed. Wat overbleef was het bedieningsgedeelte. Dit is als front-end voor het diphone board uitgevoerd.
Door deze splitsing was de noodzaak voor een 68000 microprocessor vervallen. Een draagbare, batterijgevoede versie werd daardoor haalbaar, zodat het idee van het modulair opgezette moederapparaat geheel verlaten is.
De in het eisenblad voorgestelde bediening (functietoetsen) was ook tamelijk complex en is nog eens geheel herzien. Getracht is de bediening zo gebruikersvriendelijk mogelijk te maken.
Om ervaringen van gebruikers in het ontwerp mee te nemen zijn eerst een tweetal (identieke) proefmodellen ontwikkeld en gebouwd. Hiermee kan dan een (beperkte) evaluatie uitgevoerd worden die een terugmelding geeft over de in dit proefmodel toegepaste ideeen.
Dit rapport geeft een technische beschrijving van dit proefmodel (genaamd "tiepstem").
Een uitgebreide beschrijving van een dergelijk microprocessor systeem is reeds gegeven in rapport nr. 525 [1], zodat ditmaal met een iets minder gedetaileerde beschrijving volstaan is.
Een gebruikershandleiding is afzonderlijk beschreven [7].

2 Ontwerpspecificaties.

De eisen die we (ontwerpers) in eerste instantie aan de tiepstem stelden zijn de volgende :

- zo compact en licht mogelijk. Voor de praktische bruikbaarheid van een hulpmiddel zijn deze eisen vanzelfsprekend erg belangrijk. Compactheid mag echter niet ten koste van het bedieningsgemak gaan.
- batterijgevoed. Samen met de vorige eisen zorgt dit ervoor dat het apparaat overal te gebruiken is.
- voorzien van een normaal (niet te klein) toetsenbord.
- voorzien van een diplay van zodanige afmetingen dat meerdere zinnen op dit display passen en edit operaties hierop mogelijk zijn.
- een volumeregelaar.
- een automatische aan/uit schakeling. Voor transport moet nog een extra aan/uit schakelaar aanwezig zijn.
- de volgende functies moeten verricht kunnen worden :
  - normale invoer van zinnen
  - verkorte invoer (via afkortingen)
  - herhalen van zinnen
  - wijzigen (editten) van zinnen
  - getalinvoer in cijfers

3 Systeemopzet.

Zoals reeds vermeld is het apparaat uit twee modules opgebouwd : het diphone board dat de spraaksynthese verzorgt en een bedieningsgedeelte dat de in- en uitvoer verzorgt.

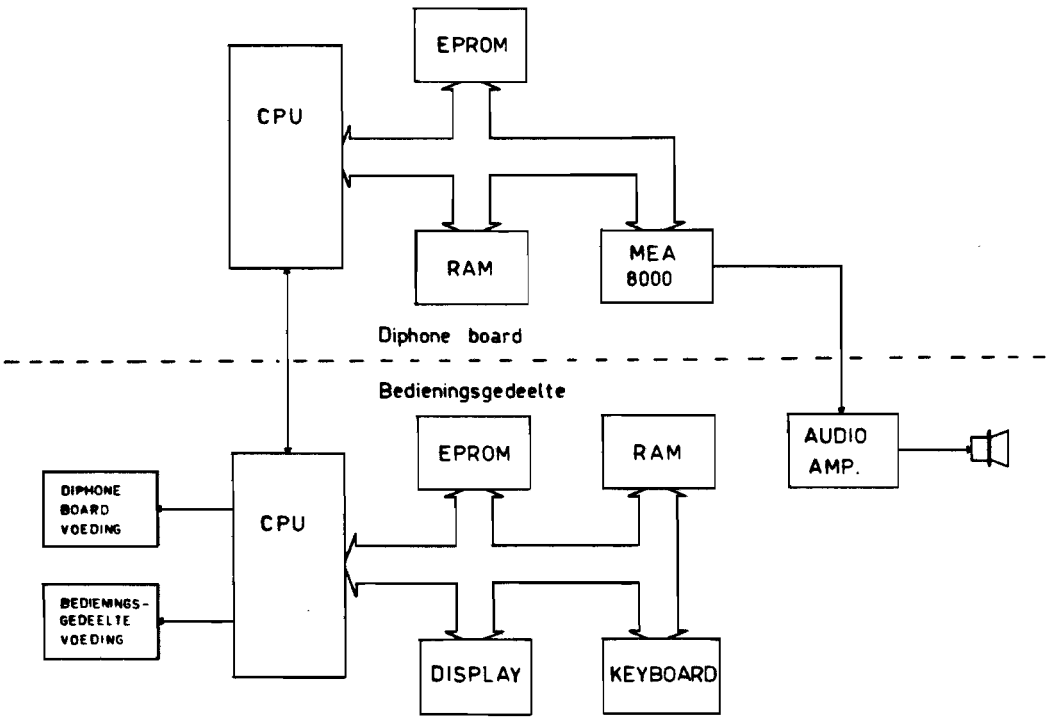


Fig. 3.1 Blokschema tiepstem

In dit samenspel is het bedieningsgedeelte de "master". dat wil zeggen deze schakelt het diphone board naar behoefte aan en uit. Het intypen van een boodschap, het editten etcetera zal allemaal door het bedieningsgedeelte verzorgd worden. De ingetypte tekst komt in het RAM geheugen van dit gedeelte te staan. Als op de spreektoets gedrukt wordt zullen het diphone board en het audio gedeelte ingeschakeld worden. De uit te spreken zin wordt dan naar het diphone board gezonden. Als dit board uitgesproken is (of niet kan spreken vanwege foutieve invoer) worden diphone board en audio gedeelte weer uitgeschakeld.
Het apparaat is van een automatische aan/uit schakeling voorzien die het apparaat na twee minuten niet gebruikt te zijn uitschakelt. Met behulp van het toetsenbord is het apparaat weer in te schakelen.
Het RAM geheugen van het bedieningsgedeelte blijft permanent onder spanning omdat dit tevens als permanent geheugen fungeert waarin zinnen onder een afkorting kunnen worden opgeslagen.

De behuizing en toetsenbord zijn van een standaard home computer (Acorn Atom). Display, luidspreker en volumeregelaar konden hier eenvoudig in gemonteerd worden.

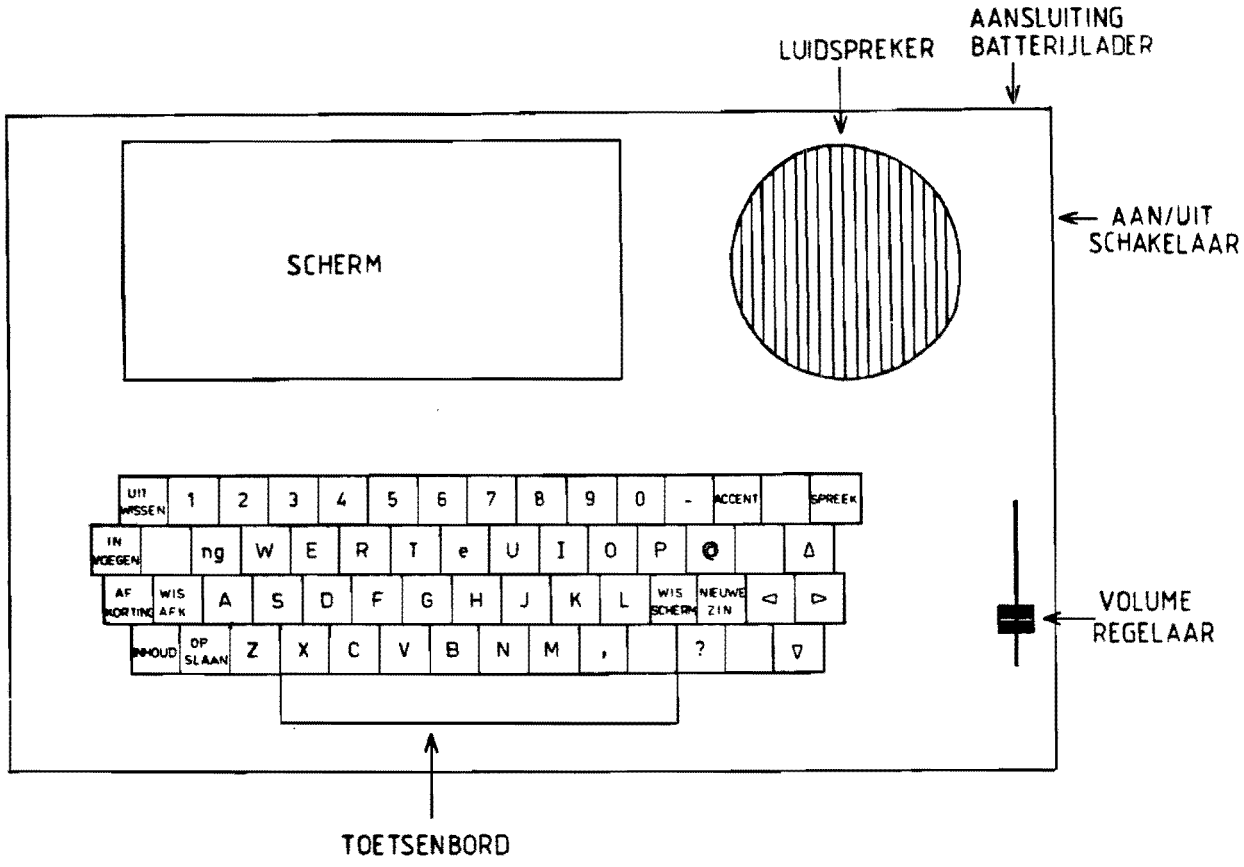


Fig. 3.2 Uitwendige tiepstem.

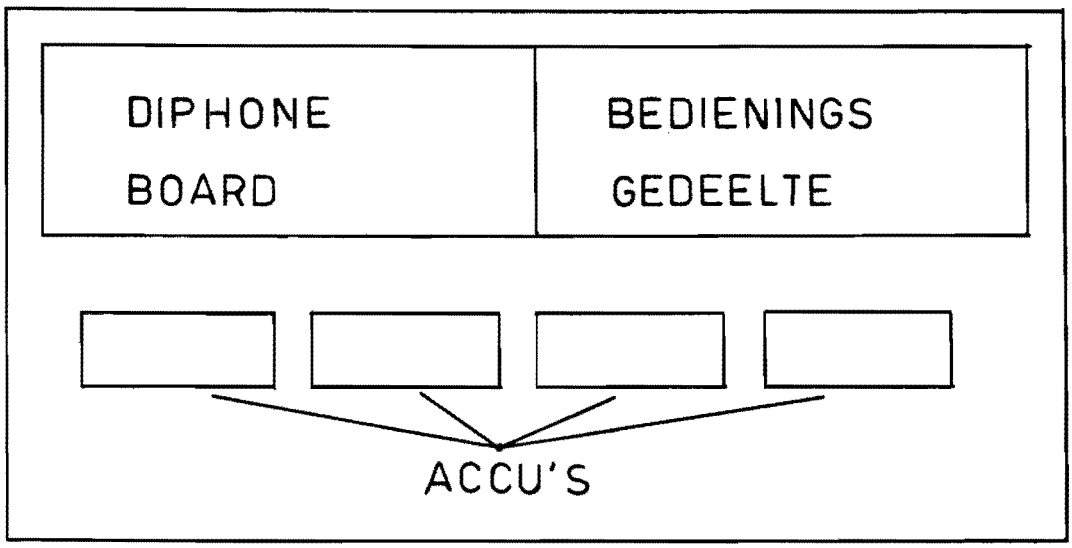


Fig. 3.3 Inwendige tiepstem.

4 Het bedieningsgedeelte.

Zoals in figuur 3.1 te zien is, is het bedieningsgedeelte een zelfstandig microprocessor systeem bestaande uit :

- microprocessor
- EPROM
- RAM
- toetsenbord
- display

Bovendien bevat dit gedeelte alle voedingsschakelingen en de audio versterker. Al deze onderdelen zullen achtereenvolgens toegelicht worden

4.1 De processor.

Omdat het diphone board voorzien is van een 8031 microprocessor en er van deze processor ook een CMOS uitvoering bestaat ligt het voor de hand ook in het bedieningsgedeelte deze processor toe te passen. Deze processor (80C31) is er een uit de MCS 51 reeks, zonder intern programmageheugen. Deze chip bevat behalve de processor nog een clock generator, twee timers, vier I/O poorten en 128 byte RAM. Deze processor is in CMOS uitgevoerd en kent om stroom te sparen twee speciale toestanden :

- idle clock grotendeels van de processor afgeschakeld, interrupt logica en timers werken nog.
- power down clock geheel stilgezet. Alleen het interne RAM behoudt zijn informatie.

Voor meer informatie wordt verwezen naar het Intel databook [3].

Deze processor wordt in deze schakeling als volgt gebruikt :

De gebruikte clock frequentie is 11,059 MHz (in verband met gebruik van de interne seriele poort). Uit deze clock wordt door deling een 1,3 MHz (voor het display) en een 337 Hz (als timerclock) signaal gemaakt.

Timer 0 dient als algemene timer Deze krijgt daartoe een externe clock aangeboden van 337 Hz om tijden in de orde van minuten/seconden mogelijk te maken. Bij toepassing als auto power off timer wordt de interrupt van deze timer gebruikt, in andere gevallen wordt gewoon gewacht (gepolled).
Timer 1 dient als baudrate generator voor de interne seriele poort (9600 baud).

Poort 0 en 2 worden voor de adres/databus gebruikt. Poort 3 bevat een aantal processor-controle signalen :

Poort 3 bit	Naam	Betekenis
0	RxD	seriele ingang
1	TxD	seriele uitgang
2	$\overline{\text{INT0}}$	interrupt 0 ingang
3	$\overline{\text{INT1}}$	interrupt 1 ingang
4	T0	timer 0 ingang
5	cs_en	chip select enable uitgang
6	$\overline{\text{WR}}$	write puls uitgang
7	$\overline{\text{RD}}$	read puls uitgang

Van poort 1 worden gebruikt :

Poort 1 bit	Naam	Betekenis
0	ram_sel	extra chip select voor RAM (uitgang)
1	diph_pwr_on	diphone board aan (uitgang)
2	pwr_off	apparaat uit (uitgang)
3	batt_low	batterijspanning < 8 V (ingang)
4	speak_rdy	MEA niet bezig (ingang)
5		
6		
7		

De seriele poort wordt gebruikt voor de communicatie met het diphone board

Externe interrupt 0 is verbonden met het toetsenbord interrupt 1 is vrij voor uitbreidingen

4.2 Geheugen.

Omdat de processor een gemultiplexte lower order address bus en data bus heeft, moet voor het lower order address een latch toegepast worden Met behulp van het signaal ALE (address latch enable) wordt het lower order address in de latch vastgehouden
Het programmageheugen bestaat uit één EPROM (27C64 : 8 Kbyte) Dit is ruim voldoende voor het toegepaste programma. Het signaal /PSEN (program store enable) wordt als enable signaal gebruikt. Het adresbereik wordt niet gedekodeerd omdat er geen behoefte is aan verder programmageheugen.
Voor het datageheugen wordt het adresbereik wel gedekodeerd met behulp van twee 3 -> 8 dekoders De eerste verdeelt het datageheugen gebied in acht 8K byte gebieden Dit sluit goed aan op de gebruikte RAM chip (6264 : 8 Kbyte) Het hoogste 8K veld wordt met behulp van de tweede dekoder in weer acht 1K byte gebieden onderverdeeld voor de peripheral chips. Dit is erg ruim maar kwa dekodering het eenvoudigste.
Deze dekoders worden alleen "ge-enabled" als /PSEN niet aktief is, zodat nooit programma- en datageheugen tegelijk aangesproken kunnen worden Om de uitgangen van de dekoders gegarandeerd inactief te houden bij in- en uitschakelen moeten de dekoders nog eens apart enabled worden via de processor (signaal /cs_en). Dit is nodig omdat het RAM informatie moet bewaren ook bij uitgeschakelde schake-

ling. Het RAM heeft ook nog een chip select die door de processor gestuurd wordt. Tussen de instructie fetches door wordt $\overline{\text{PSEN}}$ namelijk inactief. Omdat RAM en EPROM hetzelfde adresbereik gebruiken zou dan telkens het RAM geselecteerd worden. Dit geeft geen enkel probleem (er zijn dan namelijk toch geen actieve read of $\overline{\text{write}}$ signalen), maar het stroomverbruik neemt wel toe.
De uitgangen $\overline{\text{RAM1}}$ en $\overline{\text{SERIAL}}$ worden nog niet gebruikt maar zijn voor uitbreiding van de schakeling bedoeld.

De memory map ziet er als volgt uit :

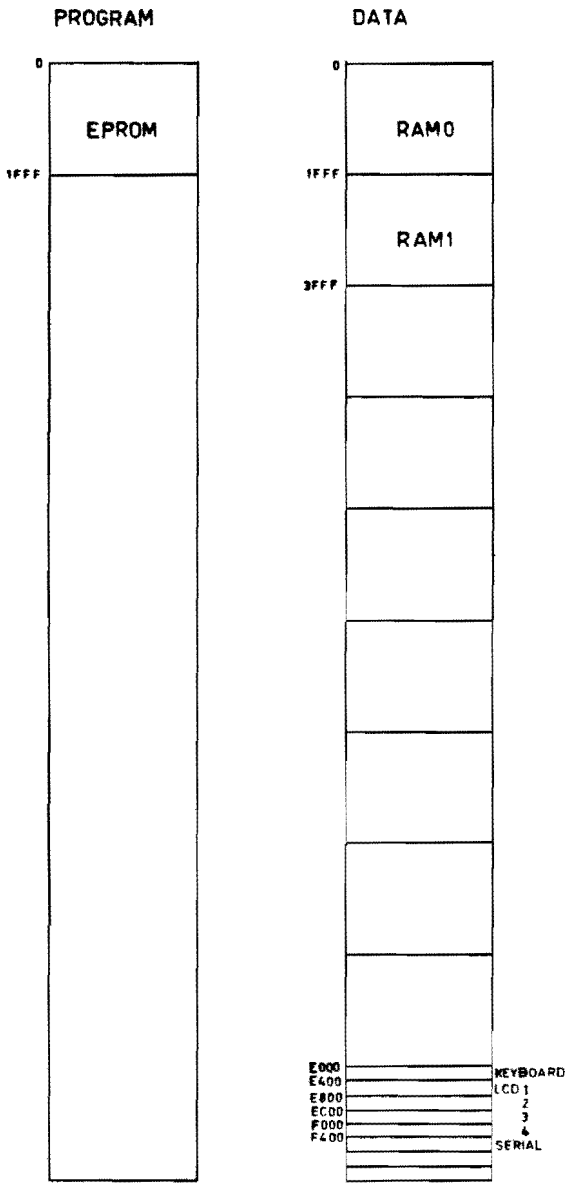


Fig. 4.1 Memory map.

4.3 Display.

Het display moet aan de volgende eisen voldoen :

- redelijke grootte (enige regels)
- laag stroomverbruik
- enkelvoudige voeding
- eenvoudige aansturing (ingebouwde karaktergenerator)

Het toegepaste LCD is een Epson EA-Y40087AT (8 x 40 karakters). Kwa aansturing bestaat dit display uit 4 diplays van elk 2 regels. Door middel van de $\overline{CS}$ ingangen kan het juiste display geselecteerd worden.
De $\overline{RDY}$ uitgangen van de diplays worden niet gebruikt, deze informatie wordt uit het statusregister gelezen.
Tussen het reset RC netwerk en het display bevindt zich een poort om de flanken van de resetpuls steiler te maken. Dit reset signaal is namelijk zeer kritisch.
Met een instelpotentiometer kan het kontrast ingesteld worden.

4.4 Toetsenbord.

De schakelaars van het toetsenbord zijn in matrix vorm met elkaar verbonden. Het scannen en dekoderen van deze matrix wordt door de processor gedaan. Een aparte keyboard controller is niet toegepast. Het scannen kan namelijk eenvoudiger en snel genoeg door de processor gebeuren. Dit beperkt het stroomverbruik van de schakeling en maakt het bovendien mogelijk de schakeling in te schakelen via het toetsenbord.
De benodigde chips voor de scanning zijn een latch en een buffer. In de latch kunnen we een (8 bit) patroon schrijven dat op de matrix gezet wordt. De uitgaande lijnen van de matrix kunnen dan via de buffer gelezen worden.
De acht diode's aan de uitgaande lijnen van de matrix vormen een OR functie. Als in de latch nu allemaal enen geschreven staan zal de uitgang van deze OR een indicatie geven of er een (willekeurige) toets ingedrukt is. Dit signaal wordt op een van de volgende twee manieren verwerkt :

1. De schakeling staat aan en wacht op invoer (in idle mode). Het signaal geeft een interrupt aan de processor die daardoor uit de idle mode komt en zal gaan testen welke toets is ingedrukt.
2. De schakeling staat standby (met de aan/uit schakelaar op aan). Het signaal zal de voedingsspanning van he bedieningsgedeelte inschakelen. Na enige tijd (ordegrootte 100 ms, voor reset en initialisaties) zal de processor het toetsenbord gaan scannen. Het is mogelijk dat de toets dan alweer losgelaten is, het enige effect is dan het inschakelen geweest.

4.5 Voeding.

De voeding van de schakeling bestaat uit 8 nikkel-cadmium accu's (Cap. 500 mAh).

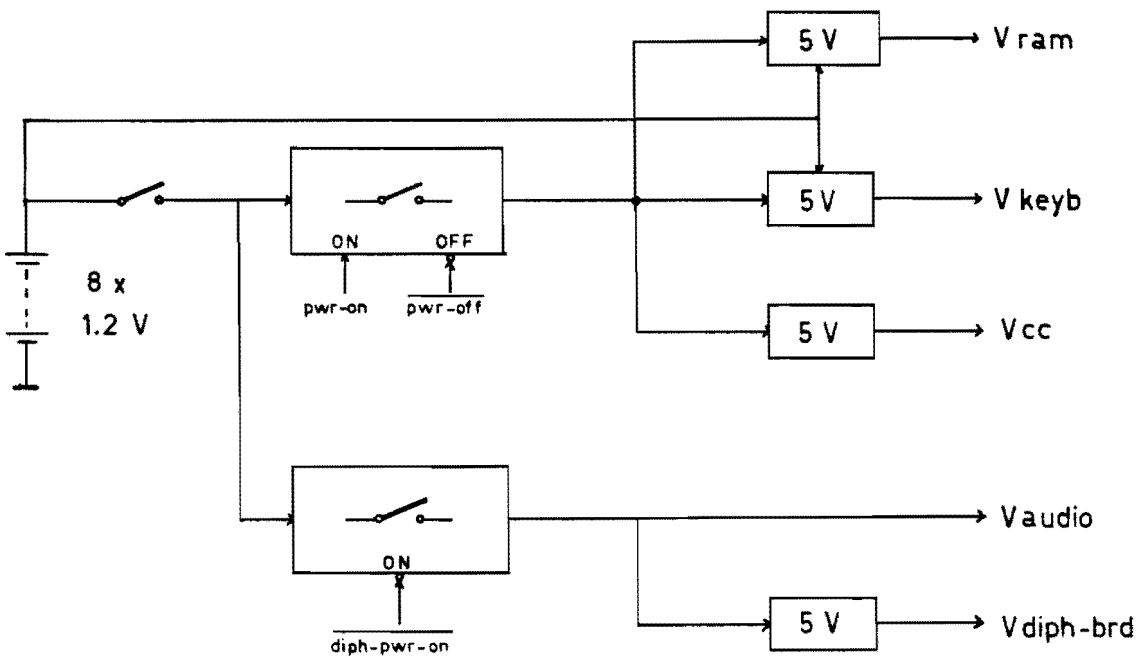


Fig. 4.2 Blokschema voeding.

De aan/uit schakelaar fungeert als hoofdschakelaar. Buiten deze schakelaar om krijgen RAM en toetsenbord (latch en buffer) altijd een spanning van ongeveer 5 V Omdat deze componenten standby bijna geen stroom verbruiken wordt deze 5 V via een spanningsdeler uit de batterijspanning verkregen. Na de mechanische aan/uit schakelaar volgt een elektronische. Door middel van het RC netwerkje bij deze schakelaar zal deze schakelaar bij het inschakelen van de spanning in geleiding komen. Door middel van twee stuuringangen kan de voeding uitgeschakeld (door de processor), en weer ingeschakeld worden (door het toetsenbord).
Na deze schakelaars volgen twee 5 V stabilisatoren voor toetsenbord en RAM. Deze zijn discreet uitgevoerd omdat de uitgangen van deze stabilisatoren altijd op spanning gehouden worden terwijl de ingangsspanning afgeschakeld wordt. Dit is met een geintegreerde regelaar niet mogelijk. Voor de rest van de schakeling is een geintegreerde 5 V regelaar toegepast (78L05).
De voedingsspanning gaat na de hoofdschakelaar ook nog naar een tweede elektronische schakelaar die door de processor bestuurd wordt. Deze schakelt diphone board en audio versterker in en uit. Voor het diphone board wordt de

spanning nog op 5 V gestabiliseerd met behulp van een geintegreerde regelaar (7805).

Met behulp van een zenerdiode en een transistor is nog een detectie voor een te lage batterijspanning gemaakt. Bij een batterijspanning lager dan 8 V wordt het signaal batt low hoog.

Het stroomverbruik van de schakeling is :

Als het apparaat uit staat	:	80 uA
Aan maar niet actief (idle)	:	12 mA
Aan, actief (toets ingedrukt)	:	30 mA
Sprekend (volumeregelaar dicht)	:	120 mA

4.6 Audio versterker.

De audio versterker bestaat uit twee geintegreerde audio versterkers (LM 388) in brugschakeling. Deze schakeling komt uit de LM 388 typical applications. De brugschakeling is toegepast om voldoende uitgangsvermogen te krijgen.
Met de instelpotentiometer wordt de gelijkstroominstelling van de brug ingesteld (0 V DC over de luidspreker).
Omdat de difoonspraak erg dof klinkt worden de hoge tonen nog wat opgehaald door middel van de twee condensatoren aan de versterker IC's. Om hoogfrequente storing van de MEA te verwijderen is aan de ingang van de versterker nog een RC filter geplaatst. Bovendien gaat dit filter oscillaties van de versterker tegen.

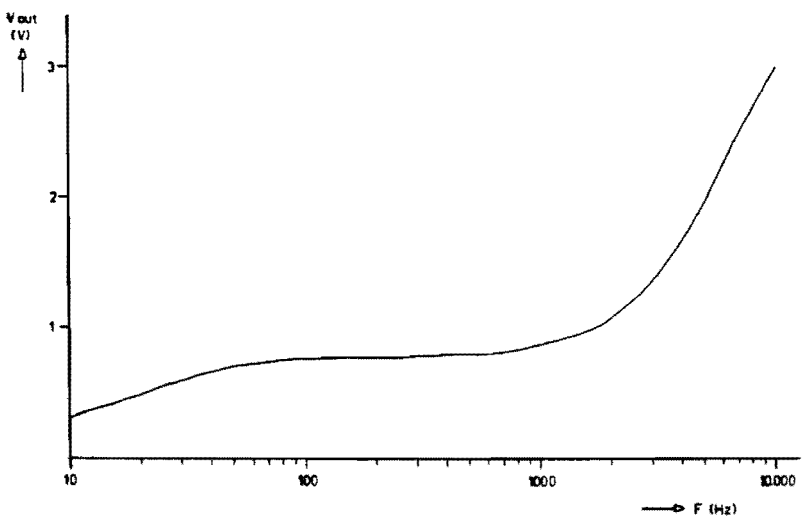


Fig. 4.3 Frequentiekarakteristiek versterker.

De koppelcondensator aan de ingang is nodig omdat anders de ingangsoffset spanning afhangt van de stand van de volumeregelaar.

De versterker wordt zoals reeds eerder vermeld naar behoefte in- en uitgeschakeld. Om te voorkomen dat dit hoorbare tikken via de luidspreker geeft, is in

serie met de luidspreker een relaiscontact geplaatst dat zowel bij in- als uitschakelen van de versterker de luidspreker 100 ms loskoppelt. Door deze korte tijd levert het relais nauwelijks een bijdrage tot het stroomverbruik.
Het relais wordt gestuurd door twee RC netwerken. Een dient voor het inschakelen en wordt gestuurd door de audio voedingsspanning. De ander dient voor het uitschakelen en wordt gestuurd door het speak ready signaal. Omdat de uitschakeltik meteen bij het uitschakelen optreed, wacht de processor na het detecteren van speak ready nog 20 ms alvorens de versterker uit te schakelen.

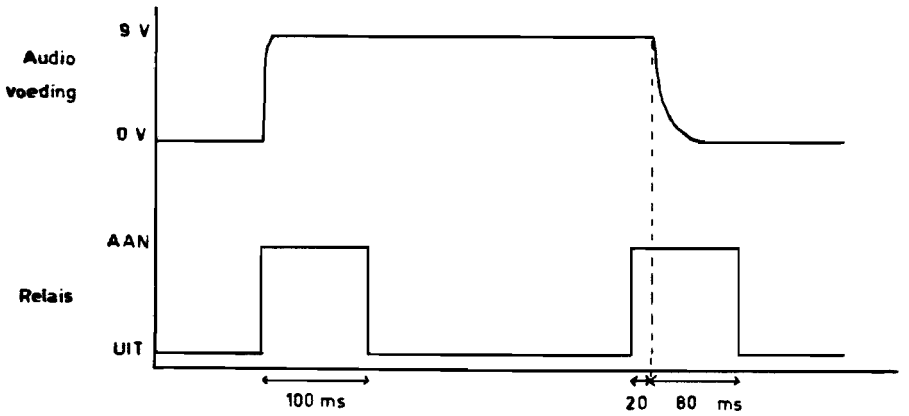


Fig. 4.4 Tijddiagram relaissturing

Om storing te voorkomen moet de massa van de volumeregelaar beslist op de massa van de versterker aangesloten worden. Een willekeurig massapunt voert teveel (digitale) storing.

Het maximale (gemeten) uitgangsvermogen bedraagt 2,6 W bij een batterijspanning van 9,6 V (8 x 1,2). De opgenomen stroom is dan 0,55 A.

4.7 Acculader.

De acculader is uiterst eenvoudig van opzet. De laadstroom is ongeveer 25 mA
De LED geeft een zichtbare indicatie dat er geladen wordt. De laadtijd bij geheel lege accu's is 32 uur. De batterijcontrole schakeling detecteert een te lage accuspanning echter voordat de accu's geheel leeg zijn, zodat een nacht opladen over het algemeen voldoende is.

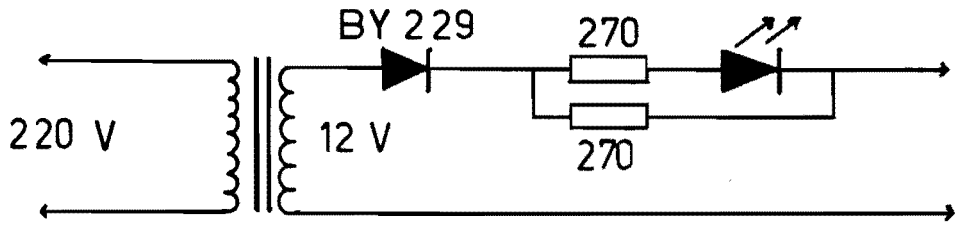


Fig. 4.5 Schema acculader.

Om ontlading (of kortsluiting) via de laadaansluiting te voorkomen is in de tiepstem meteen achter de connector een diode opgenomen.

5 Diphone board.

Het diphone board is opgebouwd uit de volgende componenten :

- een 8031 processor
- 6 EPROM's (27128 : 16 Kbyte) bevattend programma en difonen
- RAM (6116 : 2 Kbyte) voor buffers
- MEA 8000 speech synthesizer
- TDA 7050 audio versterker

Een beschrijving van dit board en de software is te vinden in het PII report van E. Falzoni.

Voor het gebruik in de tiepstem is dit board enigszins gemodificeerd.

Allereerst is het stroomverbruik zoveel mogelijk beperkt (nu 80 mA in plaats van 400 mA) door de volgende maatregelen :

- niet gebruikte componenten zijn verwijderd. Dit zijn de latch voor de parallelle ingang, de RS232 level converter (de communicatie is gewoon op 0 / 5 V) en de audio versterker. Deze audio versterker levert namelijk te weinig vermogen, ook al door de 5 V voeding.
- de processor is vervangen door zijn CMOS equivalent.
- de 6 27128 EPROM's zijn vervangen door 4 27C256 EPROM's.

De MEA $\overline{\text{REQ}}$ lijn is extern uitgevoerd. Na afvlakking geeft dit namelijk een indicatie of de MEA nog bezig is (signaal speak rdy). Dit is nodig omdat er verder geen indicatie is dat het board uitgesproken is. Door deze extra belasting en het gebruik van een CMOS processor is de interne pull-up weerstand van de processor poort niet meer voldoende en is een externe pull-up weerstand voor deze $\overline{\text{REQ}}$ lijn nodig.

Het diphone board bevat zowel een seriele als parallelle aansluiting. In deze schakeling wordt alleen de seriele aansluiting gebruikt. Deze is bedoeld voor aansluiting van een (video) terminal (RS232, 9600 baud).
Na inschakelen van het diphone board wordt een logo naar de terminal gestuurd, gevolgd door een prompt. Nu kan tekst ingevoerd worden (alleen uppercase karakters) De invoerkarakters worden geechood. Na een punt of vraagteken wordt de zin uitgesproken. Flow control vindt plaats via het X-ON X-OFF protocol. Als de zin niet uitgesproken kan worden wordt deze hele zin geechood en de cursor (ongeveer) op de plaats van de fout gezet.
In de ingevoerde tekst kan ook ge-edit worden, dit wordt hier echter niet gebruikt, alle edit werk gebeurt in het bedieningsgedeelte.

6 Software.

De ontwerpeisen voor de software komen voort uit de functies die het bedieningsgedeelte moet kunnen verrichten.
Dit zijn onder andere :

- edit mogelijkheden
- opslag van zinnen onder afkortingen
- getalgrammatica
- communicatie met diphone board
- besturing voeding

Om het geheel overzichtelijk te houden is het programma verdeeld over een aantal modules. Deze zijn :

```
decl.h    : header file met algemene declaraties
screen.h  : header file met screen routines
main.s    : hoofdprogramma
init.s    : initialisaties
intvect.s : interrupt service routines
keyb.s    : keyboard scan routine
screen.s  : screen routines
speak.s   : communicatie met diphone board
```

6.1 Datastructuren.

De processor bevat vier banken van elk acht registers. Een aantal registers hebben in het programma een vaste functie. Dit zijn :

Registerbank 0 :

R7 : huidige toets
R6 : vorige toets
R5 : Y positie cursor (0-39)
R4 : X positie cursor (0-7)
R3 : vervolgregel byte (zie tekstbuffer)

R0, R1 en R2 worden algemeen gebruikt.

Registerbank 1 :

R7 : huidige toets
R6 : i3 (van Pascal programma, zie 6.9)
R4 : k "
R3 : j "
R2 : digits of i "

R0 en R1 worden algemeen gebruikt.

Het geheugengebied (RAM) is als volgt ingedeeld :

```
  0 - 319    : tekstbuffer
320 - 639    : tijdelijk tekstbuffer
640 - einde  : permanente geheugen (zinnen onder afkorting)
```

(Momenteel is één RAM chip aanwezig : einde = 8192)

De inhoud van het tekstbuffer is net zo georganiseerd als op het scherm : de eerste regel op adres 0 - 39, de tweede op 40 - 79 etcetera. Of een regel een nieuwe zin is of een vervolgregel op een vorige regel wordt aangegeven in R3. Bit x van R3 geset betekent regel x is een vervolgregel.
Tijdens het gebruik van afkortingen (oproepen. opslaan, wissen of tonen) worden scherm en buffer tijdelijk met andere informatie gevuld. De originele buffer-inhoud wordt dan zolang in het tijdelijk tekstbuffer geplaatst.

De opslag van zinnen (of zinsdelen) onder een afkorting geschiedt in het volgende formaat. Per opslag is er een veld als volgt :

<lengte veld> <lengte afkorting> afkorting <lengte zin> zin

De drie lengtes zijn ieder één byte groot. Een afkorting plus zin is dus maximaal 255 - 3 karakters, dit is meer dan voldoende, omdat een zin maximaal 128 karakters mag zijn voor het diphone board.
Alle velden zijn achter elkaar opgeslagen, het einde wordt aangegeven door een veld met <lengte veld> gelijk aan nul.

6.2 Decl.h

Deze file wordt in elke source-file "ge-included" en bevat algemene declaraties zoals :

- namen voor I/O poort bits
- namen voor geheugenadressen peripheral chips
- namen voor geheugenplaatsen in internal RAM
- diverse constanten
- een herdefinitie voor het PCON register omdat de gebruikte versie van de PMDS cross-assembler hiervoor een fout adres genereert

6.3 Screen.h

Deze file bevat de declaraties van alle screen routines.

6.4 Main.s

Het hoofdprogramma bestaat uit de volgende lus :

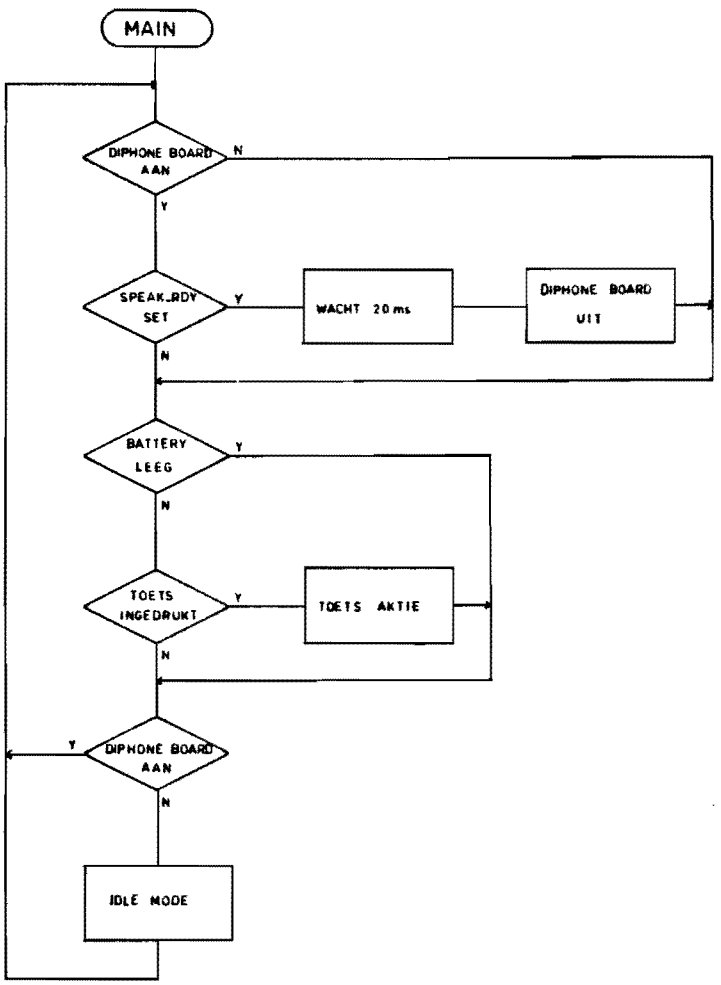


Fig 6.3 Hoofdprogramma.

Als de processor uit idle mode komt (doordat er een toets ingedrukt is) wordt de lus weer van voren af aan begonnen. In idle mode loopt de auto power-off timer die na 2 minuten een interrupt genereert (zie interrupt routines).

Omdat een spreekaktie (zie 6.9) het diphone board aan zal zetten maar niet wacht tot deze uitgesproken is wordt in het hoofdprogramma telkens getest of het diphone board nog aan staat en al uitgesproken is. In dit geval zal na 20 ms vertraging het diphone board uitgeschakeld worden.
Ook wordt in het hoofdprogramma telkens getest of de batterijspanning nog voldoende is. Is dit niet het geval dan wordt hiervan melding gemaakt op het display en wordt niet meer naar het toetsenbord gekeken.

Als een toets ingedrukt is wordt deze eerst vergeleken met de vorige ingedrukte

toets. Zijn dezen gelijk dan wordt getest of de bewuste toets repeterend is. Bij repeterende toetsen wordt na 0.5 seconde en vervolgens om de 0.1 seconde de betreffende actie herhaald.
Voor het verwerken van een toets wordt allereerst een onderscheid gemaakt tussen ASCII codes >= 20 (hex) en < 20 (hex). ASCII codes >= 20 (hex) zijn input karakters en worden in de tekstbuffer geplaatst.
Codes < 20 (hex) zijn functiecodes. De betreffende functie wordt dan uitgevoerd. Op het moment zijn de volgende functies gerealiseerd :

functiecode	functie
0	gereserveerd (geen aktie)
1	wis karakter
2	cursor omhoog
3	cursor omlaag
4	cursor links
5	cursor rechts
6	invoegen
7	spreek
8	wis scherm
9	nieuwe zin
10	afkorting oproepen
11	opslaan zin onder afkorting
12	inhoudsopgave afkortingen
13	wis afkorting

Omdat sommige functies afhangen van de voorgeschiedenis (zo zal een input karakter na een "spreek" actie automatisch een nieuwe zin openen) wordt de toestand vastgehouden in de variabele mode. Deze kent de toestanden inp mode, edit mode en speak mode.
De functies voor het permanente geheugen worden als volgt uitgevoerd :
opslaan : de nieuwe zin wordt achteraan de lijst geplaatst. Er is geen controle of de afkorting al eerder voorkomt. Wel wordt gecheckt of het geheugen vol is. Achter de nieuwe zin wordt een veld met <lengte veld> van nul geplaatst.
oproepen : de lijst wordt doorlopen tot de gegeven afkorting gevonden is. De bijbehorende zin wordt dan net als andere input karakters verwerkt.
wissen : de lijst wordt doorlopen tot de gegeven afkorting gevonden is. Alle zinnen die hier achter komen worden dan aangeschoven. Bij de afkorting "ALLES WEG" wordt (na bevestiging) <lengte veld> van het eerste veld nul gemaakt. Dit maakt de lijst leeg (ook nuttig om het opslag systeem te initialiseren).
tonen : de lijst wordt doorgelopen en op het display getoond. Van afkorting plus zin worden maximaal 40 karakters getoond (één regel op het display). Na zeven regels wordt op een spatie gewacht. Elke andere toets stopt het tonen van de zinnen.

6.5 Init.s

Deze module bevat de initialisatie van het systeem :

- de processor : seriele poort hoge prioriteit, interrupts level triggerd, timer

0 als counter in mode 1, timer 1 als timer in mode 2. De stackpointer wordt aangepast omdat twee registerbanken gebruikt worden.
- LCD : de commando's die gegeven worden zijn : reset, clear display en display on.
- diverse registers en variabelen (in RAM) krijgen een gedefinieerde beginwaarde.

De variabele overlap bepaalt hoe getallen uitgesproken worden :
set : 1234 = twaalfhonderd vier en dertig
reset : 1234 = duizend twee honderd vier en dertig

Na afloop van de initialisatie wordt naar het hoofdprogramma gesprongen.

6.6 Intvect.s

Deze module bevat vier interrupt service routines.

Reset :
bij een reset wordt naar de initialisatie gesprongen.

External interrupt 0 :
deze routine doet niets. Door de interrupt is de processor uit idle mode gekomen zodat het programma hervat wordt.

Timer 0 :
deze routine zal het apparaat uitschakelen.

Serial port :
de seriele poort verzorgt de communicatie met het diphone board.
Transmitter interrupts worden niet gebruikt, bij het zenden wordt "gepolled".
Bij een receiver interrupt worden drie speciale karakters onderscheiden : X-on en X-off voor handshaking, deze setten respectievelijk resetten de flag diph_rdy, en ESC. Een ontvangen ESC is namelijk een onderdeel van een cursor control sequence. Het aantal ontvangen escapes wordt in r0 bijgehouden zodat bij een foutieve invoer het bedieningsgedeelte ook de cursor ongeveer op de plaats van de fout kan zetten.

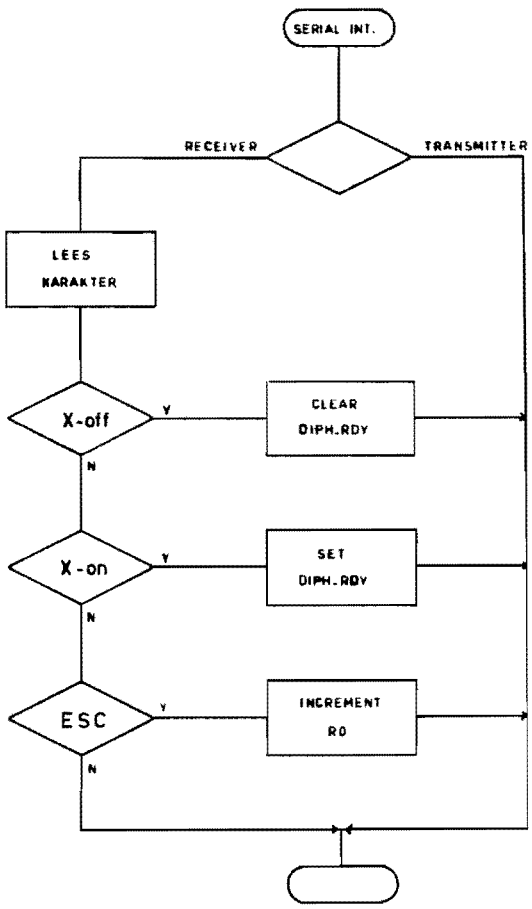


Fig. 6.1 Serial interrupt routine.

6.7 Keyb.s

Deze module bevat de keyboard scan routine. Deze test het toetsenbord of een toets is ingedrukt door FF (hex) naar de toetsenbord-latch te schrijven. Tegen dender wordt nu 250 keer (1.5 ms) de buffer gelezen. Wordt elke keer een waarde ongelijk aan 0 gelezen dan is er een toets ingedrukt.
Door nu achtereenvolgens telkens een rij hoog te maken wordt de plaats van de ingedrukte toets bepaald. Hieruit wordt het nummer van de ingedrukte toets bepaald (kolom*8 + rij).
Dit nummer wordt via een tabel in een ASCII code omgezet voor de alfa-numerieke toetsen en in een functiecode voor de functietoetsen.

6.8 Screen.s

Deze module is een verzameling van allerlei routines die een aantal basisfuncties voor het LCD verrichten.

- lcd rdy d en lcd rdy r : wacht tot het LCD ready is. (lcd_rdy_d verwacht het LCD adres in dptr, lcd_rdy_r in p2 & r0).
- clr screen : wis scherm (cursor verandert niet).
- refr screen : kopieert tekst buffer naar scherm.
- scroll : schuif tekst in buffer een regel omhoog. Als de (nieuwe) eerste regel een vervolgregel is wordt doorgeschoven, totdat dit niet meer het geval is. Hierna wordt de scherminhoud aangepast.
- goto xy : zet de cursor op de plaats aangegeven door r4 & r5.
- line feed : verricht de functie "nieuwe zin". De cursor gaat naar het begin van de volgende regel (eventueel scrollen als nodig).
- cont line : merkt huidige regel (r4) als vervolgregel.
- cursor up, cursor down, cursor right en cursor left : cursor bewegingen.

6.9 Speak.s

Als eerste schakelt deze routine het diphone board in.
De zin waarin de cursor zich bevindt wordt dan via de seriele poort verzonden. Handshaking vindt plaats via het X-on X-off protocol en de flag diph rdy (zie interrupt routines).
Bepaalde karakters (e, g en cijfers) worden voor verzenden geconverteerd. Het overzenden stopt na het zenden van een punt of vraagteken. Dan wordt maximaal 1 seconde gewacht of het diphone board gaat spreken (via het signaal speak rdy). Gedurende deze seconde wordt het aantal ontvangen escapes geteld om later de cursor op de plaats van de fout te kunnen zetten.
De cijfers naar woorden omzetting (getalgrammatica) is gebaseerd op het Algol programma beschreven door H.B. Corstius. Het programma is voor dit doel beperkt tot duizendtallen (maximaal 6 cijfers). Het is eerst herschreven in Pascal (VAX) en daarna omgezet in een assembler versie. Commentaar in het assembler programma verwijst dan ook naar statements in het Pascal programma (zie Appendix ?).

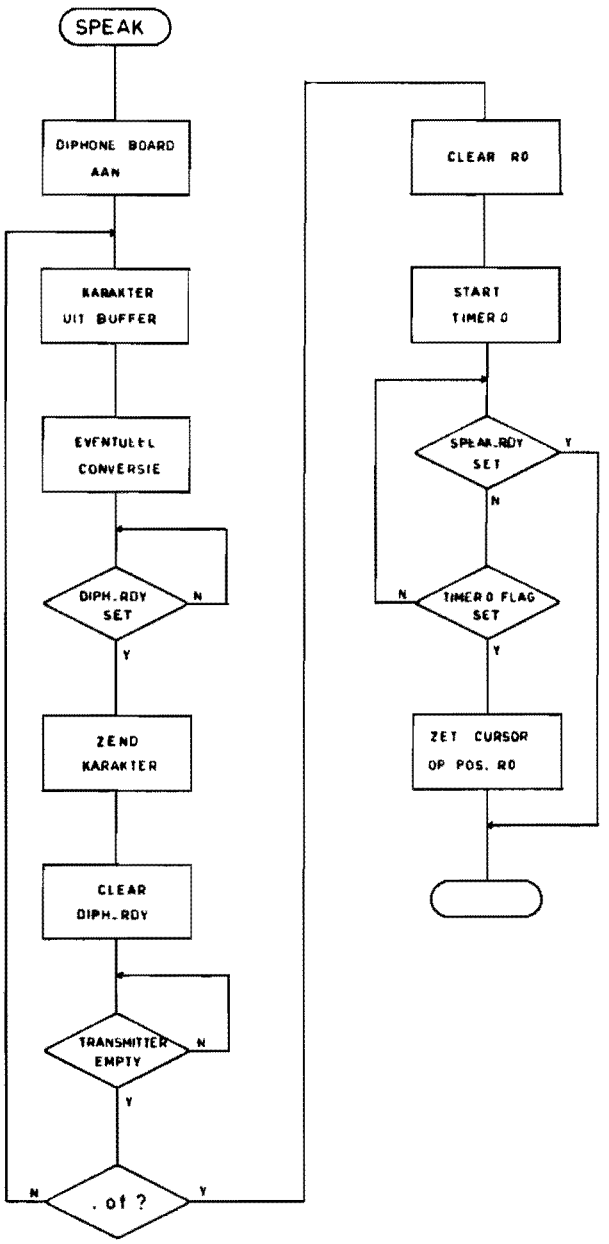


Fig. 6.2 Speak routine.

7 Literatuur.

1. Waterham R.P.
   Technische beschrijving van de Compacte Spraakhulp (CSH I).
   IPO Rapport nr. 525

2. Menting P.G.
   Een eenvoudige spraaksynthetisator als hulpmiddel voor gehandicapten.
   IPO Eisenblad nr. 106

3. Intel Microcontroller Handbook 1985

4. Falzoni E.
   Development of hardware and software for the diphone speech board.
   PII report

5. Scheffers M.T.M
   User's guide to the OM8002 diphone speech synthesis board.
   CAB report EDP8502

6. Corstius H.B.
   Automatic translation of numbers into dutch.
   Foundations of language 1(1965), pp. 59-62

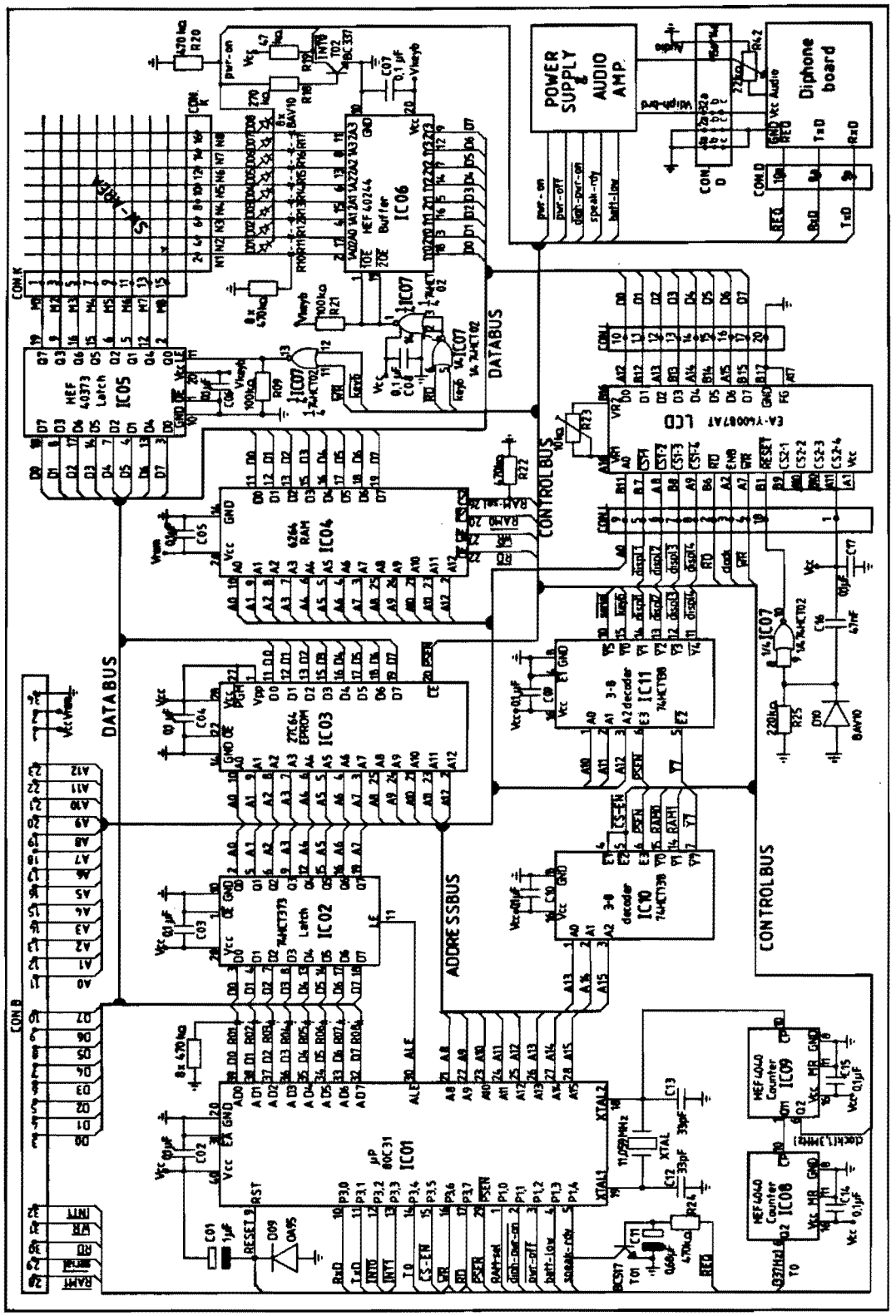
7. Deliege R.J.H.
   Gebruikershandleiding tiepstem.
   IPO handleiding nr. 73

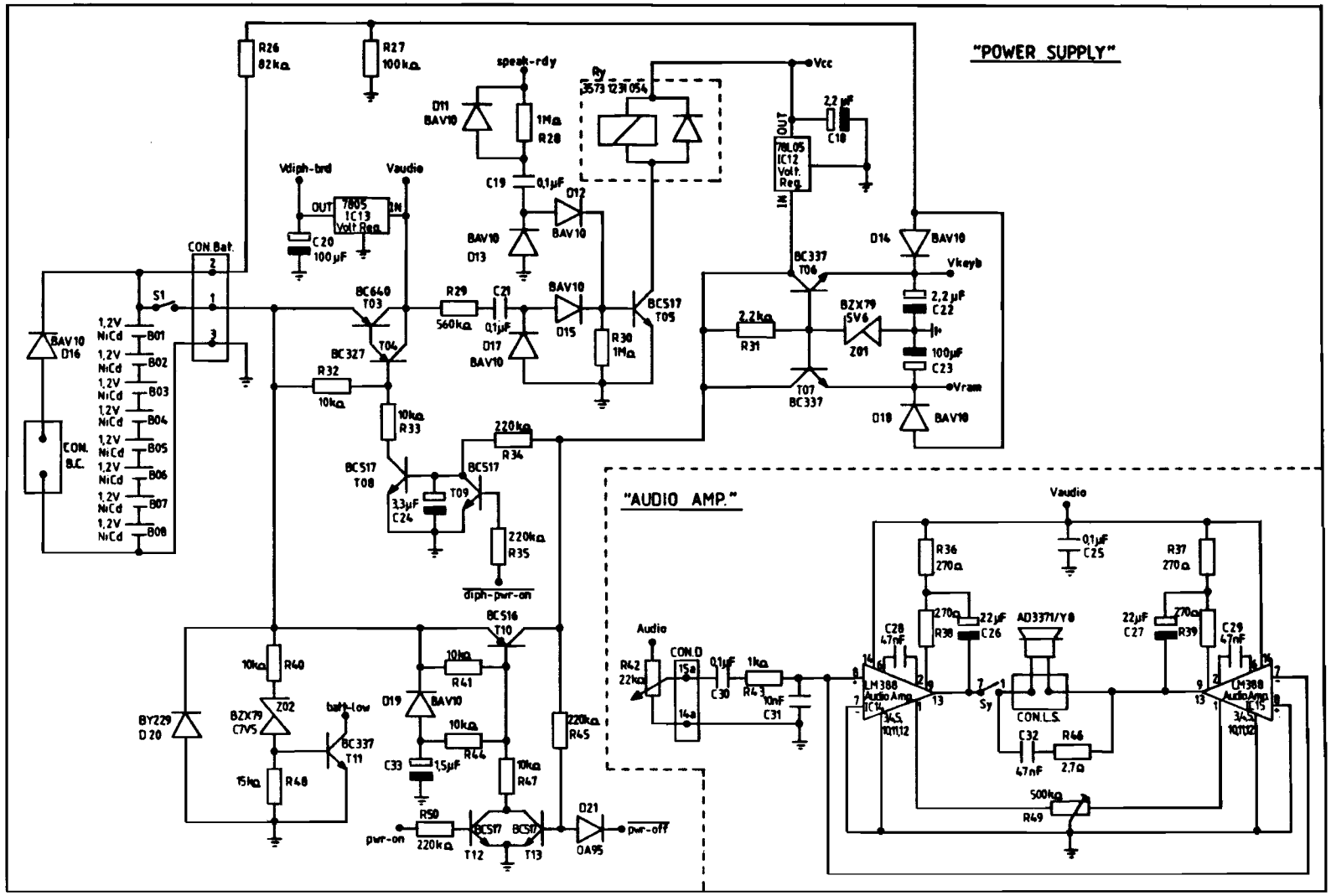
A Getalgrammatica programma.

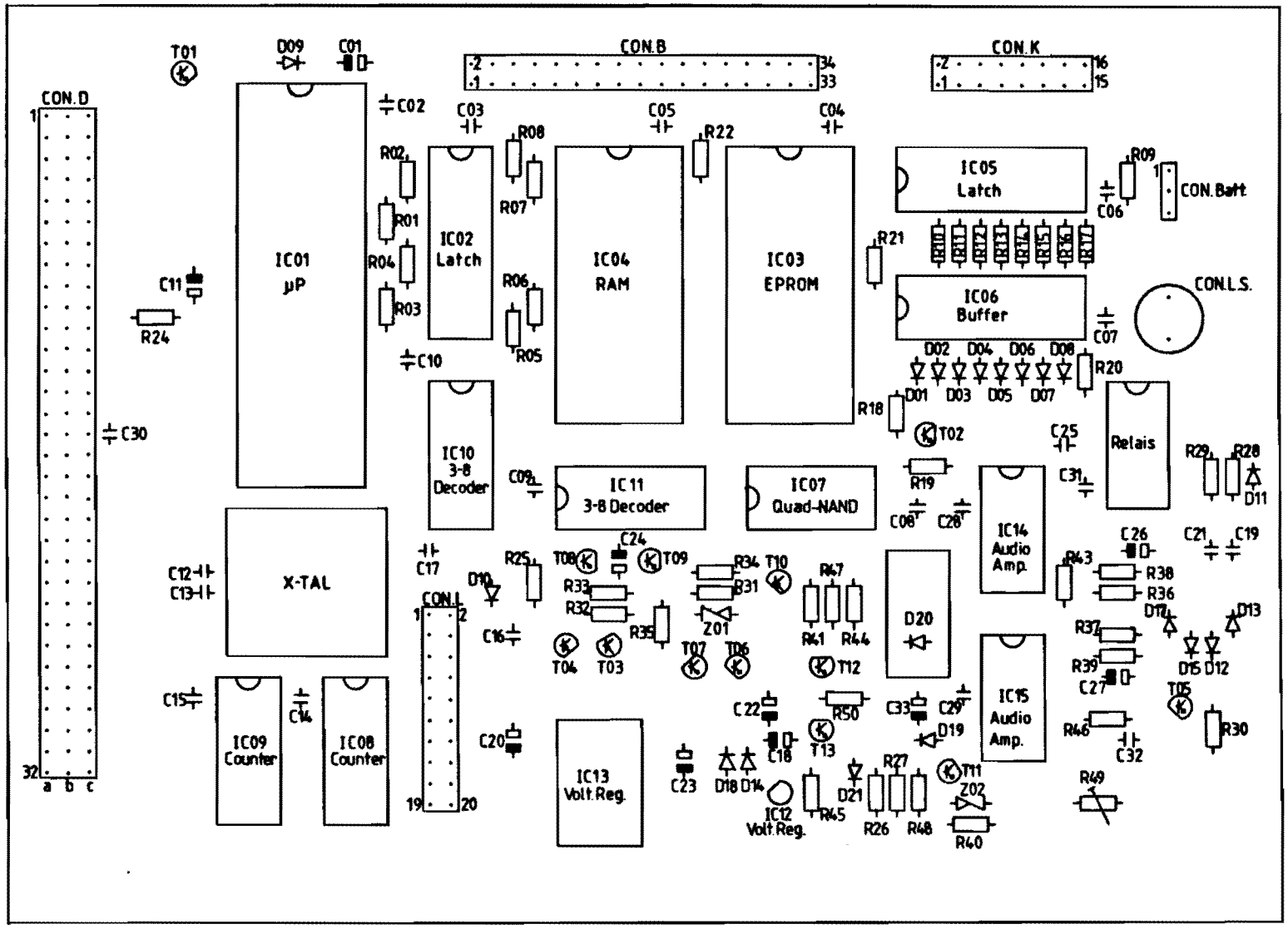
```
program getal(input,output);

var
   digits,number,i : integer;
   nr : array[1..6] of integer;
   words : array[1..27] of varying[10] of char;
   overlap : boolean;
   c : char:

value
   words := ('een','twee','drie','vier','vijf'
            'zes','zeven','acht','negen',
            'elf','twaalf' 'dertien','veertien','vijftien',
            'zestien','zeventien','achtien','negentien',
            'tien','twintig','dertig','veertig','vijftig',
            'zestig','zeventig','tachtig','negentig');
label 1,2

procedure split(number : integer);
var i : integer;
begin
   for i:=1 to 6 do nr[i] := 0;
   digits := 0;
   i := 7;
   repeat
       digits := digits + 1;
       i := i - 1;
       nr[i] := number mod 10;
       number := number div 10;
   until number = 0;
end; {split}

procedure choose(j,k : integer);
begin
   write(words[9*j + k])
end; {choose}

procedure tens(j,k : integer);          { write 10*j + k }
begin
   if k <> 0 then begin if j<2 then choose(j,k)
                              else begin choose(0,k);
                                         write(' en ');
                                         choose(2,j);
                                   end
                  end
             else if j <> 0 then choose(2,j);
end; {tens}

procedure hundreds(j,k : integer);      { write (10*j + k) * 100 }
begin
   if j + k <> 1 then tens(j,k);
   if k <> 0 then write('honderd ');
```

```
end; {hundreds}

procedure thousands(k : integer):        { write 1000**k }
begin
   if k = 0 then goto 1
            else write('duizend ');
end: {thousands}

procedure next3digits(i : integer):
var i3,k : integer;
begin
   i3 := 3 * i;
   if i = 0 then k := 0 else k = nr[7-i3];
   if overlap and (nr[6-i3] * k <> 0) and (nr[5-i3] + nr[4-i3] = 0)
   then begin
          hundreds(nr[6-i3],nr[7-i3]);
          tens(nr[8-i3],nr[9-i3]);
          thousands(i-1);
          next3digits(i-2);
       end {overlapping case}
   else begin
          hundreds(0,nr[4-i3]);
          tens(nr[5-i3],nr[6-i3]);
          if (nr[4-i3] + nr[5-i3] + nr[6-i3] <> 0) or (i = 0)
               then thousands(i)
          next3digits(i-1);
       end {non overlapping case}
end; {next3digits}

begin
2:      writeln;
       write('Overlap (y/n) : ');
       readln(c);
       if c = 'y' then overlap := true else overlap := false;
       write('Getal : ');
       readln(number);
       split(number);
       i := (digits-1) div 3;
       next3digits(i);
1:      if number <> 0 then goto 2:
end.
```

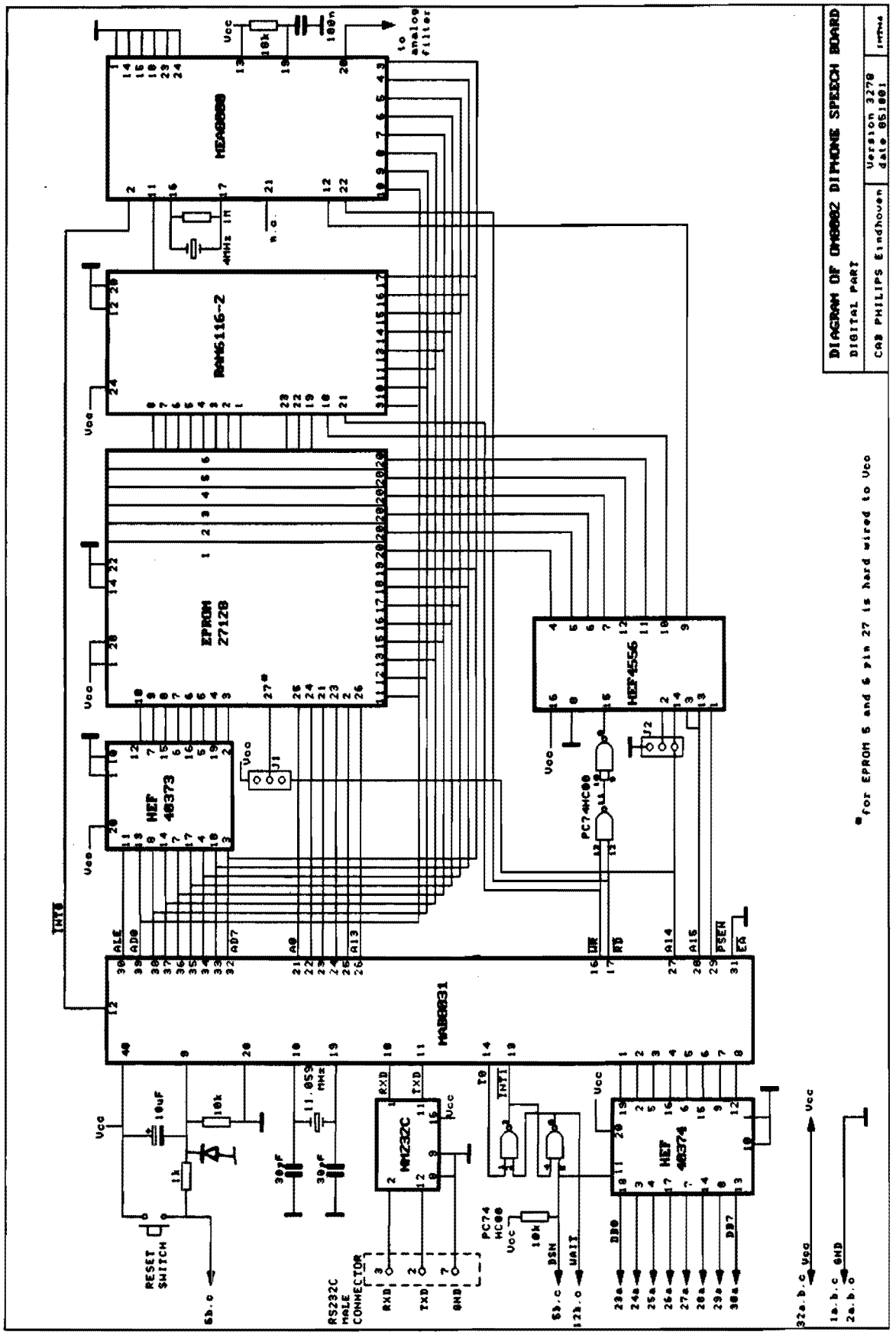
B Schema bedieningsgedeelte.




"POWER SUPPLY"
"AUDIO AMP."
R26 82kΩ
R27 100kΩ
speak-rdy
Ry 3573 1231 054
D11 BAV10
1MΩ R28
C19 0,1µF
D12 BAV10
BAV10 D13
Vdiph-brd
Vaudio
OUT 7805 IC13 Volt Reg. IN
C20 100µF
CON.Bat.
S1
1,2V NiCd B01
1,2V NiCd B02
1,2V NiCd B03
1,2V NiCd B04
1,2V NiCd B05
1,2V NiCd B06
1,2V NiCd B07
1,2V NiCd B08
BAV10 D16
CON. B.C.
BC640 T03
BC327 T04
R29 560kΩ
C21 0,1µF
D17 BAV10
BAV10 D15
BC517 T05
R30 1MΩ
R32 10kΩ
10kΩ R33
220kΩ R34
BC517 T08
BC517 T09
3,3µF C24
220kΩ R35
diph-pwr-on
Vcc
78L05 IC12 Volt. Reg. OUT IN
2,2µF C18
D14 BAV10
Vkeyb
BC337 T06
2,2kΩ R31
BZX79 5V6 Z01
2,2µF C22
100µF C23
Vram
T07 BC337
D18 BAV10
BC516 T10
10kΩ R40
BZX79 C7V5 Z02
BY229 D20
batt-low
BC337 T11
15kΩ R48
10kΩ R41
D19 BAV10
10kΩ R44
C33 1,5µF
10kΩ R47
220kΩ R45
R50 220kΩ
pwr-on
BC517 T12
BC517 T13
D21 OA95
pwr-off
Audio
R42 22kΩ
CON.D 15a 14a
0,1µF C30
1kΩ R43
10nF C31
Vaudio
0,1µF C25
R36 270Ω
R37 270Ω
270Ω R38
22µF C26
AD3371/Y8
22µF C27
270Ω R39
C28 47nF
C29 47nF
LM388 Audio Amp IC14
LM388 Audio Amp IC15
Sy
CON.L.S.
C32 47nF
R46 2,7Ω
500kΩ R49


CON.D
CON.B
CON.K
CON.Batt
CON.L.S.
CON.L
IC01 µP
IC02 Latch
IC03 EPROM
IC04 RAM
IC05 Latch
IC06 Buffer
IC07 Quad-NAND
IC08 Counter
IC09 Counter
IC10 3-8 Decoder
IC11 3-8 Decoder
IC12 Volt.Reg.
IC13 Volt.Reg.
IC14 Audio Amp.
IC15 Audio Amp.
X-TAL
Relais
D20
T01
T02
T03
T04
T05
T06
T07
T08
T09
T10
T11
T12
T13
Z01
Z02

C Schema diphone board.



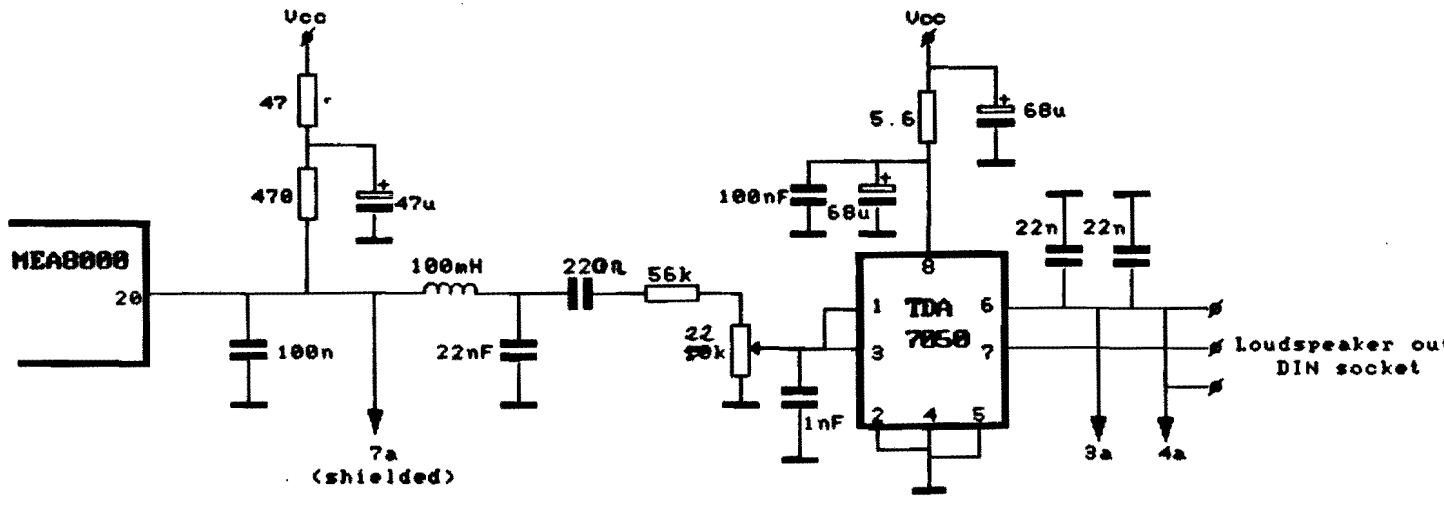
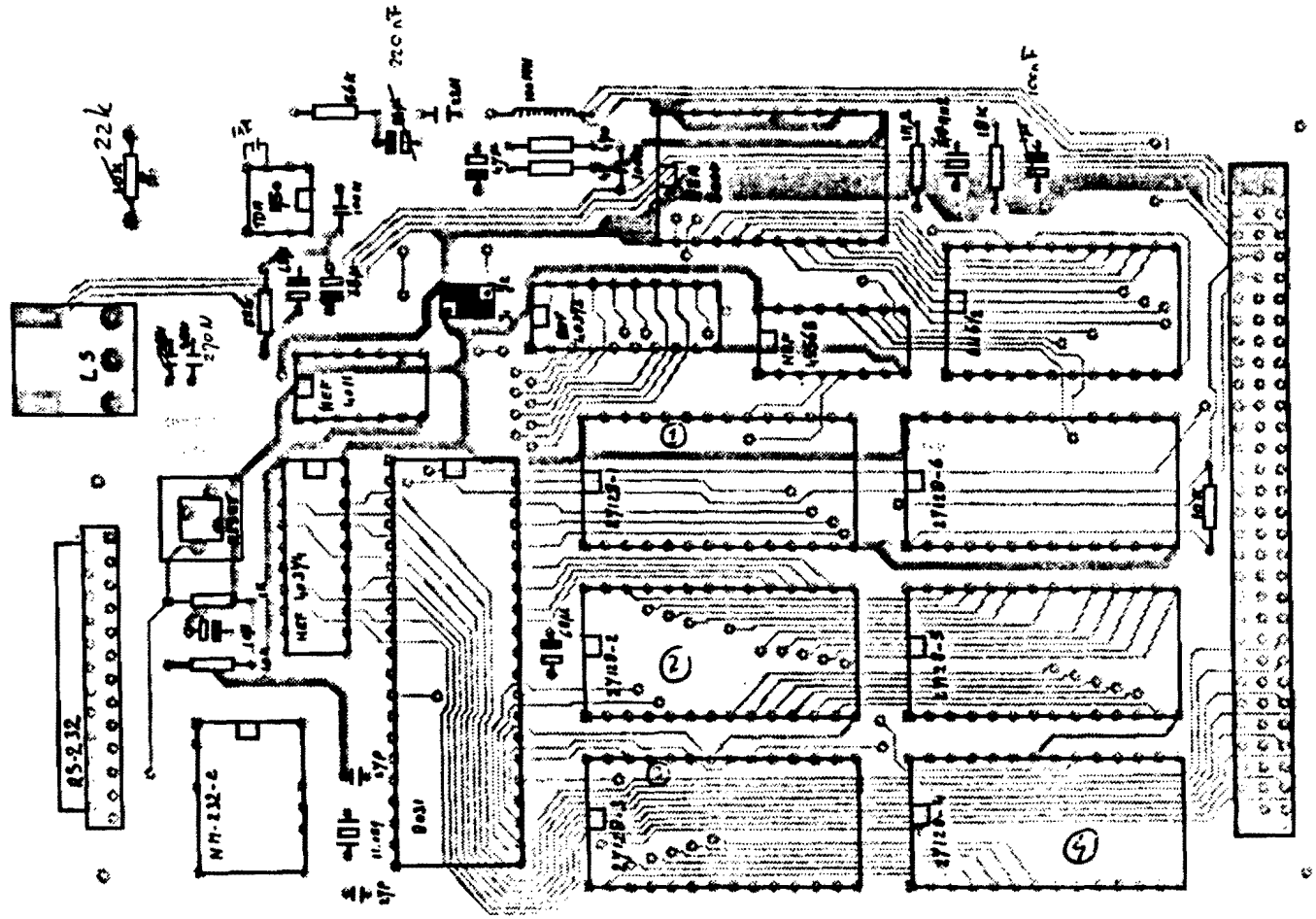


DIAGRAM OF OM8002 DIPHONE SPEECH BOARD
ANALOG PART

CAB PHILIPS Eindhoven	Version 3278 date:851001	IMTMS



jumper setting for 27256 in socket 1 and 3 (2 and 4 empty)
EPROM

5 and 6 must be 27128

```
.nolist
!************************************************
!*                                              *
!*      General declarations                    *
!*                                              *
!************************************************
!
!
! P1.0 = RAM select
#define ramsel          p1(0)
!
! P1.1 = diphone board power on
#define diph_pwr_on     p1(1)
!
! P1.2 = power off
#define pwr_off         p1(2)
!
! P1.3 = battery low
#define batt_low        p1(3)
!
! P1.4 = diphone board ready with speaking
#define speak_rdy       p1(4)
!
! P3.5 = enable addres decoders (chip selects)
#define cs_en           p3(5)
!
! F0 = diphone board ready for receive
#define diph_rdy        f0
!
! keyboard address
#define keyb_addr       0xe000
!
! LCD addresses
#define lcd1    0xe400
#define lcd2    0xe800
#define lcd3    0xec00
#define lcd4    0xf000
#define lcd1_up 0xe4
!
! Permanent storage start address (in RAM)
#define buffer  640
!
! Variables in internal RAM
#define x_save  0x7f
#define y_save  0x7e
#define lcd_up  0x7d
#define bufh    0x7c
#define bufl    0x7b
#define flags   0x7a
#define mode    0x79
#define number  0x78
!
! Flag bits
#define repeat          acc(0)
#define speak_pressed   acc(1)
#define overlap         acc(2)
```

```
!
! Constants
#define inp_mode         0
#define edit_mode        1
#define speak_mode       2
!
! RAM size : 8K (0000-1fff)
#define maxmem  0x20
!
!
! tijdelijk PCON register
#define pcon    0x87
!
.list
!
```

```
.nolist
!*************************************************
!*                                               *
!*       Screen routines declarations            *
!*                                               *
!*************************************************
!
.extern lcd_rdy_d,lcd_rdy_r,clr_screen,refr_screen,goto_xy,line_feed,cont_l
.extern cursor_up,cursor_down,cursor_right,cursor_left
!
.list
!
```

```
#
.list
!************************************************
!*                                              *
!*      M O D U L E    I N I T                  *
!*                                              *
!*      Initialisation                          *
!*                                              *
!*                                              *
!*                                              *
!*      Copyright  december 1985                *
!*      Institute for perception research, IPO  *
!*      Eindhoven, the Netherlands.             *
!*                                              *
!*      Writer : Ir. R.J.H. Deliege             *
!*                                              *
!************************************************
!
#include "decl.h"
!
.sect   _init
.define init
.extern main,lcd_rdy_d,empty_buffer
!
!       8031 initialisation
!
init:   clr     ramsel          ! disable RAM
        mov     ip,#0x10        ! serial port high prio., remainder low
        setb    ea              ! global enable interrupts
        mov     tmod,#0x25      ! timer 0 : counter, mode 1
!                                 timer 1 : timer, mode 2 (baudr gen)
        mov     th1,#0xfd       ! reload value timer 1 (9600 bd)
        mov     scon,#0x40      ! serial port mode 1 (8 bit UART)
        mov     tcon,#0x40      ! run timer 1
!                                 interrupts level triggered
        setb    ren             ! enable receiver
        setb    es              ! enable serial int
        mov     sp,#0x0f        ! register bank 0 & 1 in use
        clr     cs_en           ! enable chip select decoder
!
!       L C D   initialisation
!
        mov     dptr,#lcd1
        mov     r0,#4           ! 4 LCD's
lcd_ini:
        acall   lcd_rdy_d
        mov     a,#0x10         ! reset
        movx    @dptr,a
        acall   lcd_rdy_d
        mov     a,#1            ! clear display
        movx    @dptr,a
        acall   lcd_rdy_d
        mov     a,#0x0d         ! display on
        movx    @dptr,a
        mov     a,dph           ! next LCD
        add     a,#4
```

```
        mov     dph,a
        djnz    r0,lcd_ini
!
!       Variables initialisation
!
        clr     a
        mov     r3,a            ! continuation lines byte
        mov     r4,a            ! cursor X position
        mov     r5,a            ! cursor Y position
        mov     bufh,a          ! buffer pointer
        mov     bufl,a
        mov     flags,a         ! flags
        mov     a,flags
        setb    overlap
        mov     flags,a
        mov     mode,#inp_mode  ! start in input mode
        lcall   empty_buffer
        acall   goto_xy         ! place cursor
        ajmp    main
!
```

```
#
.list
!*************************************************
!*                                               *
!*      M O D U L E    I N T V E C T             *
!*                                               *
!*      Interrupt vectors                        *
!*                                               *
!*                                               *
!*                                               *
!*      Copyright  december 1985                 *
!*      Institute for perception research, IPO   *
!*      Eindhoven, the Netherlands.              *
!*                                               *
!*      Writer : Ir. R.J.H. Deliege              *
!*                                               *
!*************************************************
!
#include "decl.h"
!
!       Reset
!
.sect   _reset
.extern init
.base   0
        ajmp    init
!
!       External int. 0 : keyboard
!
.sect   _keyb_int
.base   3
        clr     ie0
        reti
!
!       Timer 0 int.
!
.sect   _timer_0_int
.base   0x0b
        setb    cs_en           ! disable address decoders
        clr     pwr_off         ! power off whole system
        sjmp    .
!
!       Serial port int.
!
.sect   _serial_int
.base   0x23
        jb      ri,0f           ! do nothing with transmitter int.
        reti
0:      push    acc             ! save accumulator
        clr     ri              ! clear receiver int.
        mov     a,sbuf          ! read char
        cjne    a,#0x13,1f      ! X-off
        clr     diph_rdy        ! diphone board not ready
        sjmp    3f
1:      cjne    a,#0x11,2f      ! X-on
        setb    diph_rdy        ! diphone board ready
```

```
        sjmp    3f
2:      cjne    a,#0x1b,3f      ! ESC (from cursor control seq)
        inc     r0              ! error
3:      pop     acc
        reti
!
```

```
#
.list
!*************************************************
!*                                               *
!*      M O D U L E    K E Y B                   *
!*                                               *
!*      Keyboard scan routine                    *
!*                                               *
!*                                               *
!*                                               *
!*      Copyright  december 1985                 *
!*      Institute for perception research, IPO   *
!*      Eindhoven, the Netherlands.              *
!*                                               *
!*      Writer : Ir. R.J.H. Deliege              *
!*                                               *
!*************************************************
!
!       Register usage :
!
!       R7 : current key
!       R6 : previous key
!       R2 : key column
!
#include "decl.h"
!
.sect   _keyb
.define keyb
!
keyb:   mov     a,r7            ! save previous key
        mov     r6,a
        mov     r7,#0           ! reset new key
        mov     r0,#250         ! debounce counter
        mov     dptr,#keyb_addr ! keyboard address
        mov     a,#0xff         ! all rows
        movx    @dptr,a
debounce:
        movx    a,@dptr         ! scan
        jz      no_key          ! no key was pressed
        djnz    r0,debounce     ! scan 250 times to eliminate debounce
!
!       scan keyboard
!
        mov     r0,#1           ! start with row 1
        mov     r1,#8           ! 8 rows to scan
        mov     a,r0
scan:   movx    @dptr,a
        movx    a,@dptr         ! scan row
        jz      next_row        ! no key pressed in this row
        mov     r2,a            ! save pressed key column
        sjmp    transl_key      ! one key is enough
next_row:
        mov     a,r0            ! next row
        rl      a
        mov     r0,a
        djnz    r1,scan         ! 8 rows scanned
```

```
        sjmp    no_key          ! after all no key was pressed
!
!       translate key address into number (0..63)
!
transl_key:
        mov     a,r2
0:      rrc     a
        jc      1f
        inc     r7
        sjmp    0b
1:      mov     a,r7
        rl      a
        rl      a
        rl      a
        mov     r7,a            ! R7.3..R7.5 is key column number
!
        mov     a,r0
2:      rrc     a
        jc      3f
        inc     r7
        sjmp    2b
                                ! R7.0..R7.2 is key row number
!
!       translate number into ASCII code
!
3:      mov     a,r7
        add     a,#2            ! start of table
        movc    a,@a+pc         ! look up in table
        mov     r7,a
no_key: ret
!
!       keyboard table
!
!       ASCII value     key number      key function
!-------------------------------------------------------
.data1  0               ! reserved      no key
.data1  'H'             !  1
.data1  'e'             !  2
.data1  '7'             !  3
.data1  ' '             !  4
.data1  0               !  5
.data1  0               !  6
.data1  0               !  7
.data1  12              !  8            display stored sentences
.data1  10              !  9            use pre stored sentence
.data1  6               ! 10            insert character
.data1  1               ! 11            delete
.data1  'N'             ! 12
.data1  'J'             ! 13
.data1  'U'             ! 14
.data1  '8'             ! 15
.data1  11              ! 16            store sentence
.data1  13              ! 17            remove stored sentence
.data1  0               ! 18
.data1  '1'             ! 19
.data1  'M'             ! 20
```

```
.data1  'K'             ! 21
.data1  'I'             ! 22
.data1  '9'             ! 23
.data1  'Z'             ! 24
.data1  'A'             ! 25
.data1  'g'             ! 26
.data1  '2'             ! 27
.data1  ','             ! 28
.data1  'L'             ! 29
.data1  'O'             ! 30
.data1  '0'             ! 31
.data1  'X'             ! 32
.data1  'S'             ! 33
.data1  'W'             ! 34
.data1  '3'             ! 35
.data1  0               ! 36
.data1  8               ! 37        clear screen
.data1  'P'             ! 38
.data1  '-'             ! 39
.data1  'C'             ! 40
.data1  'D'             ! 41
.data1  'E'             ! 42
.data1  '4'             ! 43
.data1  '?'             ! 44
.data1  9               ! 45        line feed
.data1  'Q'             ! 46
.data1  "'"             ! 47
.data1  'V'             ! 48
.data1  'F'             ! 49
.data1  'R'             ! 50
.data1  '5'             ! 51
.data1  0               ! 52
.data1  4               ! 53                cursor left
.data1  0               ! 54
.data1  0               ! 55
.data1  'B'             ! 56
.data1  'G'             ! 57
.data1  'T'             ! 58
.data1  '6'             ! 59
.data1  3               ! 60                cursor down
.data1  5               ! 61                cursor right
.data1  2               ! 62                cursor up
.data1  7               ! 63                speak
```

```
#
.list
!*************************************************
!*                                               *
!*      M O D U L E   M A I N                    *
!*                                               *
!*      Main program.                            *
!*                                               *
!*                                               *
!*                                               *
!*      Copyright  december 1985                 *
!*      Institute for perception research, IPO   *
!*      Eindhoven, the Netherlands.              *
!*                                               *
!*      Writer : Ir. R.J.H. Deliege              *
!*                                               *
!*************************************************
!
#include "decl.h"
!
.sect   _main
.define main, line_begin, line_end, empty_buffer, bank_0, bank_1, dec_dptr
.extern keyb, speak, init
#include "screen.h"
!
!       main loop :
!       check diphone board, battery and keyboard
!
main:   jb      diph_pwr_on,0f  ! if diphone board is off
        jnb     speak_rdy,0f    ! test if diphone board is still speaking
        mov     th0,#0xff
        mov     tl0,#0xfa       ! 20 msec
        clr     tf0             ! clear possible timer 0 int.
        setb    tr0             ! run timer 0
        jnb     tf0,.           ! wait for timer 0 int.
        clr     tr0             ! stop timer 0
        setb    diph_pwr_on     ! power off diphone board
        clr     diph_rdy        ! and it is not ready now
!
0:      lcall   batt_check      ! test if battery is (nearly) empty
        jnc     1f
        lcall   idle            ! if so do nothing except idle
        sjmp    main
!
1:      lcall   keyb            ! scan keyboard
        mov     a,r7
        jnz     2f
        lcall   idle            ! no key pressed
        sjmp    main
2:      clr     c
        subb    a,r6            ! compare with previous key
        jz      3f              ! same key
        lcall   key_action
        sjmp    main
3:      lcall   auto_repeat
        sjmp    main            ! loop until key is released
```

```
!
!       Idle mode is entered when there is nothing to do.
!       This mode will be terminated by one of the following interrupts :
!       - timer 0  : auto power off time expired
!       - keyboard : back to work again
!
idle:   mov     a,flags
        clr     repeat          ! clear auto-repeat
        mov     flags,a
        jnb     diph_pwr_on,1f  ! first wait for diphone board power off
        mov     dptr,#keyb_addr ! set all keyboard rows to 1
        mov     a,#0xff
        movx    @dptr,a
        mov     th0,#0x61       ! load timer 0 (2 min)
        mov     tl0,#0
        clr     tf0             ! clear possible pending timer 0 int.
        setb    tr0             ! run timer 0
        setb    et0             ! enable timer 0 int.
        setb    ex0             ! enable keyboard int.
        inc     pcon            ! go in idle mode
!
!       idle
!
        clr     tr0             ! stop timer 0
        clr     et0             ! disable timer 0 int.
        clr     ex0             ! disable keyboard int.
1:      ret
!
!       Check battery
!
batt_check:
        jnb     batt_low,batt_ok
        lcall   clr_screen
        mov     r0,#6           ! write message to screen
        mov     dptr,#lcd2+1
0:      lcall   lcd_rdy_d
        mov     a,r0
        movc    a,@a+pc
        jz      1f              ! last char ? (\0)
        movx    @dptr,a
        inc     r0
        sjmp    0b
.asciz  "            Battery opladen !"
1:      setb    c
        ret
batt_ok:
        clr     c
        ret
!
!
!       KEY - ACTIONS
!
!       Test if key is a input character ( >= 0x20 )
!       or a special function key ( < 0x20 ).
!
key_action:
```

```
        mov     a,r7
        jb      acc(7),char
        jb      acc(6),char
        jb      acc(5),char
        sjmp    func
!
!       INPUT CHARACTER.
!
!       Place it in buffer and on screen.
!
char:   mov     a,mode
        cjne    a,#speak_mode,1f! we are already in input mode
        cjne    r5,#0,0f        ! we are not at the beginning of a line
        sjmp    1f
0:      lcall   line_feed       ! start a new line
1:      mov     mode,#inp_mode  ! we are in input mode now
        mov     a,r7
        lcall   char_in
        ret
!
char_in:
        push    dph             ! save used registers
        push    dpl
        mov     dpl,r0
        push    dpl
        mov     b,a             ! save for screen
        mov     dph,bufh
        mov     dpl,bufl
        setb    ramsel
        movx    @dptr,a         ! store char in buffer
        clr     ramsel
        inc     dptr
        mov     bufh,dph        ! update buffer pointer
        mov     bufl,dpl
        mov     dph,lcd_up
        mov     dpl,#1
        lcall   lcd_rdy_d
        mov     a,b
        movx    @dptr,a         ! char to screen
        inc     r5              ! update cursor position
        cjne    r5,#40,2f       ! new line ?
        inc     r4
        cjne    r4,#8,0f        ! scroll needed ?
        sjmp    3f
0:      mov     a,r4
        inc     a
        mov     r0,a
        mov     a,r3
1:      rrc     a               ! continuation line bit
        djnz    r0,1b
        jnc     3f              ! is new line a continuation ?
        mov     r5,#0           ! if so, only update cursor
        lcall   goto_xy
3:      dec     r4              ! r4 will be incremented in line_feed
        lcall   line_feed       ! otherwise make a clear new line
        lcall   cont_line       ! make it a continuation line
```

```
2:      pop     acc
        mov     r0,a
        pop     dpl
        pop     dph
        ret
!
!       FUNCTION - KEYS
!
!
!       EDIT ROUTINES
!
func:   cjne    a,#1,func1      ! 1 : delete
        mov     a,mode
        cjne    a,#edit_mode,func0      ! different delete in edit mode
!                                 delete char under cursor
        mov     x_save,r4       ! save position
        mov     y_save,r5
        lcall   line_end
        lcall   dec_dptr        ! dptr is end of this sentence
        mov     p2,bufh         ! copy to pointer
        mov     r0,bufl
        mov     r5,dph          ! end value
        mov     r4,dpl
        mov     dph,bufh
        mov     dpl,bufl
        inc     dptr            ! copy from pointer
0:      setb    ramsel
        movx    a,@dptr         ! move in buffer
        movx    @r0,a
        clr     ramsel
        inc     dptr
        mov     a,r5
        cjne    a,dph,1f        ! all copied ?
        mov     a,r4
        cjne    a,dpl,1f
        sjmp    2f
1:      inc     r0
        cjne    r0,#0,0b
        inc     p2
        sjmp    0b
2:      mov     r4,x_save       ! restore position
        mov     r5,y_save
        lcall   refr_screen
        ret
!
func0:  mov     mode,#inp_mode  ! delete char left from cursor
        cjne    r5,#0,1f        ! no problem, we are not at the beginning
        mov     a,r4            !   of a line
        inc     a
        mov     r0,a
        mov     a,r3
0:      rrc     a
        djnz    r0,0b
        jc      1f              ! this is a continuation line
        ret                     ! nothing more to delete
1:      lcall   cursor_left
```

```
        mov     a,#' '
        lcall   char_in         ! wipe character
        lcall   cursor_left
        cjne    r5,#39,3f
        mov     a,r4            ! continuation line is empty now
        inc     a
        inc     a
        mov     r0,a
        clr     c               ! so this is no continuation line anymore
        mov     a,#0xff         ! prepare mask
2:      rlc     a
        djnz    r0,2b
        anl     a,r3
        mov     r3,a
3:      ret
!
func1:  cjne    a,#2,func2      ! 2 : cursor up
        lcall   cursor_up
        mov     mode,#edit_mode
        ret
!
func2:  cjne    a,#3,func3      ! 3 : cursor down
        lcall   cursor_down
        mov     mode,#edit_mode
        ret
!
func3:  cjne    a,#4,func4      ! 4 : cursor left
        lcall   cursor_left
        mov     mode,#edit_mode
        ret
!
func4:  cjne    a,#5,func5      ! 5 : cursor right
        lcall   cursor_right
        mov     mode,#edit_mode
        ret
!
func5:  cjne    a,#6,func6      ! 6 : insert character
        mov     a,mode
        cjne    a,#speak_mode,0f! insert only in input or edit mode
        sjmp    6f
0:      mov     x_save,r4       ! save position
        mov     y_save,r5
        lcall   line_end        ! seek last line
1:      lcall   dec_dptr
        setb    ramsel
        movx    a,@dptr         ! check last character of this line
        clr     ramsel
        cjne    a,#' ',2f       ! line is full
        sjmp    3f              ! nothing wrong
2:      dec     r4
        push    x_save          ! we stay at same position
        push    y_save
        push    bufh
        push    bufl
        lcall   line_feed       ! open a new line
        lcall   cont_line       ! and make it a continuation
```

```
        pop     bufl            ! back to previous position
        pop     bufh
        pop     y_save
        pop     x_save
        sjmp    1b              ! and try again
3:      lcall   dec_dptr
        mov     p2,dph          ! copy from pointer
        mov     r0,dpl
        inc     dptr            ! copy to pointer
4:      setb    ramsel
        movx    a,@r0           ! move
        movx    @dptr,a
        clr     ramsel
        lcall   dec_dptr        ! update pointers
        dec     r0
        cjne    r0,#0xff,5f
        dec     p2
5:      mov     a,dph
        cjne    a,bufh,4b       ! all copied ?
        mov     a,dpl
        cjne    a,bufl,4b
        mov     a,#' '          ! fill with space
        mov     dph,bufh
        mov     dpl,bufl
        setb    ramsel
        movx    @dptr,a
        clr     ramsel
        mov     r4,x_save       ! restore position
        mov     r5,y_save
        lcall   refr_screen
6:      ret
!
!
!       GENERAL FUNCTIONS
!
!
func6:  cjne    a,#7,func7      ! 7 : speak
        mov     a,mode
        cjne    a,#speak_mode,0f! we are already speaking
        sjmp    5f
0:      lcall   line_end
1:      lcall   dec_dptr
        setb    ramsel
        movx    a,@dptr
        clr     ramsel
        cjne    a,#' ',2f       ! skip trailing spaces
        sjmp    3f
2:      cjne    a,#'.',4f       ! or full stop
3:      dec     r5
        cjne    r5,#0xff,1b
        mov     r5,#39
        dec     r4
        sjmp    1b
4:      inc     dptr
        mov     bufh,dph
        mov     bufl,dpl
```

```
        lcall   goto_xy
        mov     a,#'.'          ! put a full stop at the end
        lcall   char_in
        mov     mode,#speak_mode! we are in speak mode now
5:      lcall   speak           ! speak this sentence
        ret
!
!
func7:  cjne    a,#8,func8      ! 8 : clear screen
        ljmp    init            ! re-initialise system
!
!
func8:  cjne    a,#9,func9      ! 9 : line feed
        cjne    r5,#0,0f        ! already at the beginning of a line ?
        ret                     ! do nothing then
0:      lcall   line_feed
        ret
!
!
!       PERMANENT STORE ROUTINES
!
!       Storage format :
!
!       field : <total length>
!               <index length>
!               index
!               <sentence length>
!               sentence
!
!       all fields are consecutive stored, end of the list is
!       indicated by a total length of zero.
!
func9:  cjne    a,#10,func10    ! 10 : use pre-stored sentence
        lcall   store_use
        ret
!
func10: cjne    a,#11,func11    ! 11 : add a new sentence to the list
        lcall   store_add
        ret
!
func11: cjne    a,#12,func12    ! 12 : display stored sentences
        lcall   store_display
        ret
!
func12: cjne    a,#13,func13    ! 13 : remove stored sentence
        lcall   store_remove
        ret
!
func13: ret
!
store_use:                      ! use pre-stored sentence
        mov     b,#0            ! flag for search_index
        mov     a,flags
        clr     speak_pressed   ! clear speak key flag
        mov     flags,a
        lcall   search_index
```

```
        jc      0f              ! index found
        ljmp    index_error
0:      push    dph             ! save dptr (address sentence length)
        push    dpl
        lcall   restore_buffer  ! back to original
        lcall   bank_0
        mov     bufl,r0
        mov     bufh,r1
        mov     mode,r2
        lcall   refr_screen
        pop     dpl
        pop     dph
        setb    ramsel
        movx    a,@dptr         ! sentence length
        clr     ramsel
        mov     r1,a
1:      inc     dptr
        setb    ramsel
        movx    a,@dptr         ! character from stored sentence
        clr     ramsel
        mov     r7,a
        lcall   char            ! use it as input
        djnz    r1,1b
        mov     a,flags
        jb      speak_pressed,3f
        mov     r7,#10          ! to prevent from second action when
        ret                     ! key is still pressed
3:      mov     a,#7            ! simulate speak key pressed
        ljmp    func6           ! go to speak action
!
!
store_add:                      ! add a new sentence to the stored list
!       seek end of current list
        mov     dptr,#buffer    ! start of list
0:      setb    ramsel
        movx    a,@dptr         ! length of this field
        clr     ramsel
        jz      free            ! this is the end
        add     a,dpl           ! go to next field
        mov     dpl,a
        jnc     0b
        inc     dph
        sjmp    0b
free:   mov     r0,bufl         ! save it
        mov     r1,bufh
        mov     r2,mode
        lcall   bank_1          ! use registerbank 1
        push    dph             ! save dptr = start of new field
        push    dpl
        lcall   save_buffer
        mov     b,#1            ! flag for read_index
        lcall   read_index
0:      pop     dpl             ! restore dptr
        pop     dph
        push    dph             ! and store it for later use (start of fie
        push    dpl
```

```
        inc     dptr            ! length of index
        mov     p2,#0           ! start address of index
        mov     r0,#80
        mov     a,bufl
        clr     c
        subb    a,#80           ! calculate length of index
        mov     r1,a            ! counter for copy
        push    acc             ! index length
        setb    ramsel
        movx    @dptr,a         ! store it
        clr     ramsel
        inc     dptr
1:      setb    ramsel
        movx    a,@r0           ! copy index string
        movx    @dptr,a
        clr     ramsel
        inc     r0              ! increment pointers
        inc     dptr
        mov     a,#maxmem
        cjne    a,dph,2f        ! dph = maxmem : memory overflow
        lcall   restore_buffer  ! everything back to normal
        lcall   bank_0
        mov     bufl,r0
        mov     bufh,r1
        mov     mode,r2
        lcall   full            ! and call error routine
2:      djnz    r1,1b
        push    dph             ! and store it for later use (sentence leng
        push    dpl
        lcall   restore_buffer
        lcall   bank_0          ! back to original registers
        mov     bufl,r0
        mov     bufh,r1
        lcall   refr_screen
        lcall   line_begin      ! seek start of sentence
        mov     p2,dph          ! save it
        mov     r0,dpl
        lcall   line_end        ! seek last line of sentence
3:      lcall   dec_dptr
        setb    ramsel
        movx    a,@dptr
        clr     ramsel
        cjne    a,#' ',4f       ! skip trailing spaces
        dec     r5
        cjne    r5,#0xff,3b
        mov     r5,#39
        dec     r4
        sjmp    3b
4:      inc     dptr
        mov     bufh,dph        ! update cursor
        mov     bufl,dpl
        lcall   goto_xy
        mov     a,dpl           ! calculate length of sentence
        clr     c
        subb    a,r0
        mov     r1,a
```

```
        pop     dpl             ! start of sentence
        pop     dph
        push    acc             ! sentence length
5:      setb    ramsel
        movx    @dptr,a         ! sentence length
        clr     ramsel
        inc     dptr
        jnz     6f
        ajmp    9f              ! empty sentence
6:      setb    ramsel
        movx    a,@r0           ! copy sentence
        movx    @dptr,a
        clr     ramsel
        inc     r0              ! increment pointers
        cjne    r0,#0,7f
        inc     p2
7:      inc     dptr
        mov     a,#maxmem
        cjne    a,dph,8f        ! memory overflow ?
        lcall   full            ! error routine
8:      djnz    r1,6b
        clr     a
        setb    ramsel
        movx    @dptr,a         ! zero length for next field
        clr     ramsel
!       update total length of this field
        pop     acc             ! sentence length
        mov     r0,a
        pop     acc             ! index length
        add     a,r0
        add     a,#3            ! 3 length bytes
        pop     dpl             ! start of field (= total length)
        pop     dph
        setb    ramsel
        movx    @dptr,a
        clr     ramsel
        ret
9:      pop     acc             ! empty sentence
        pop     acc             ! restore stack (dummy pop's)
        pop     acc
        pop     acc
        ret                     ! do nothing with empty sentence
!
!
store_display:                  ! display all stored sentences and indexes
        mov     r0,bufl         ! save original registers and buffer
        mov     r1,bufh
        lcall   bank_1
        lcall   save_buffer
        mov     dptr,#buffer    ! start of list
0:      mov     r0,#7           ! line counter
        clr     a
        mov     r3,a            ! init buffer variables
        mov     r4,a
        mov     r5,a
        mov     bufl,a
```

```
        mov     bufh,a
        lcall   empty_buffer
        lcall   refr_screen
1:      setb    ramsel
        movx    a,@dptr         ! length of this field
        clr     ramsel
        jnz     2f
        ajmp    8f              ! end of list
2:      add     a,dpl           ! calculate start of next field
        push    acc             ! and save it
        mov     a,dph
        jnc     3f
        inc     a
3:      push    acc
        inc     dptr
        setb    ramsel
        movx    a,@dptr         ! index length
        clr     ramsel
        mov     r1,a            ! index counter
4:      inc     dptr
        setb    ramsel
        movx    a,@dptr         ! index
        clr     ramsel
        lcall   char_in         ! to screen
        djnz    r1,4b           ! whole index had ?
        mov     a,#' '
        lcall   char_in
        mov     a,#':'
        lcall   char_in
        mov     a,#' '
        lcall   char_in
        inc     dptr
        setb    ramsel
        movx    a,@dptr         ! sentence length
        clr     ramsel
        mov     r1,a            ! sentence counter
5:      inc     dptr
        setb    ramsel
        movx    a,@dptr         ! sentence
        clr     ramsel
        lcall   char_in
        cjne    r5,#0,6f        ! line full ?
        dec     r4
        lcall   goto_xy
        sjmp    7f
6:      djnz    r1,5b           ! whole sentence had ?
7:      lcall   line_feed       ! next line
        pop     dph             ! start of next field
        pop     dpl
        djnz    r0,1b           ! screen full ?
        push    dpl
        push    dph
        acall   wait            ! wait for user action
        pop     dph
        pop     dpl
        cjne    r7,#' ',9f      ! exit display routine
```

```
        ajmp    0b              ! and back for the next 7 sentences
8:      acall   wait
9:      lcall   restore_buffer  ! everything back to normal
        lcall   bank_0
        mov     bufl,r0
        mov     bufh,r1
        lcall   refr_screen
        ret
!
wait:   mov     r4,#7           ! put prompt on last line
        mov     r5,#0
        lcall   goto_xy
        mov     r0,#8
0:      mov     a,r0
        movc    a,@a+pc
        jz      1f              ! end of prompt
        lcall   char_in
        inc     r0
        sjmp    0b
.asciz  "Druk op spatiebalk voor volgende pagina"
1:      lcall   keyb
        mov     a,r7
        jnz     2f
        lcall   idle
        sjmp    1b
2:      clr     c
        subb    a,r6            ! compare with previous key
        jz      1b
        ret
!
!
store_remove:                   ! clear permanent store
        mov     b,#2            ! flag for search_index
        lcall   search_index
        jnc     inf             ! index not found
        ljmp    single          ! index found : remove single sentence
inf:    mov     dptr,#79        ! user index
        mov     r0,#18
0:      inc     dptr
        mov     a,r0
        movc    a,@a+pc
        jz      2f              ! end of string reached
        mov     r1,a
        setb    ramsel
        movx    a,@dptr
        clr     ramsel
        clr     c
        subb    a,r1            ! compare index to string
        jz      1f
        ljmp    index_error     ! not the same
1:      inc     r0
        sjmp    0b              ! next character
.asciz  "ALLES WEG"             ! string
2:      mov     r4,#4
        mov     r5,#0
        lcall   goto_xy
```

```
        mov     r0,#6           ! ask confirmation
        mov     dptr,#lcd3+1
3:      lcall   lcd_rdy_d
        mov     a,r0
        movc    a,@a+pc
        jz      4f
        movx    @dptr,a
        inc     r0
        sjmp    3b
!       next string must be 64 characters
.ascii  "Alle afkortingen wissen ?
.asciz  "Toets J of N ( Ja - Nee )"
4:      lcall   keyb
        cjne    r7,#'J',5f
        mov     dptr,#buffer    ! start of list
        clr     a               ! clear the list
        setb    ramsel
        movx    @dptr,a
        clr     ramsel
        sjmp    removed
5:      cjne    r7,#'N',4b
        sjmp    removed
single:
        mov     dpl,r4          ! start of this field
        mov     dph,r5
        mov     p2,r5
        setb    ramsel
        movx    a,@dptr         ! length of this field
        clr     ramsel
        add     a,dpl           ! calculate start of next field
        mov     r0,a
        jnc     1f
        inc     p2
1:      setb    ramsel
        movx    a,@r0           ! move remainder of list
        movx    @dptr,a
        clr     ramsel
        jz      removed         ! end of list
        inc     dptr            ! increment pointers
        inc     r0
        cjne    r0,#0,1b
        inc     p2
        sjmp    1b
removed:
        lcall   restore_buffer
        lcall   bank_0
        mov     bufl,r0
        mov     bufh,r1
        mov     mode,r2
        lcall   refr_screen
        ret
!
!
save_buffer:
!       save original buffer
        mov     p2,#0           ! copy from pointer
```

```
        mov     r0,#0
        mov     r1,#0
        mov     dptr,#320       ! copy to pointer
0:      setb    ramsel
        movx    a,@r0
        movx    @dptr,a
        clr     ramsel
        inc     dptr            ! increment pointers
        inc     r0
        cjne    r0,#0,1f
        inc     p2
        inc     r1
1:      cjne    r1,#1,0b        ! all 320 bytes copied ?
        cjne    r0,#64,0b
        ret
!
read_index:
        lcall   empty_buffer
!       put prompt in buffer and on screen
        clr     a
        mov     bufh,a          ! init buffer variables
        mov     bufl,a
        mov     r3,a
        mov     r4,a
        mov     r5,a
        mov     mode,#inp_mode  ! and go to input mode
        mov     a,#41
        mul     ab              ! calculate offset to correct message
        add     a,#10
        mov     r0,a
0:      mov     a,r0
        movc    a,@a+pc
        jnz     1f              ! last character ( \0 )
        ajmp    2f
1:      lcall   char_in
        inc     r0
        sjmp    0b
.asciz  "       ***  AFKORTING  ***                "
.asciz  "       ***  OPSLAAN  ***                   "
.asciz  "        ***  WISSEN  ***                   "
2:      mov     r0,#8
3:      mov     a,r0
        movc    a,@a+pc
        jz      index
        lcall   char_in
        inc     r0
        sjmp    3b
.asciz  "Afkorting (einde door toets afkorting):"
!       read index
index:  lcall   line_feed       ! new line
0:      lcall   keyb            ! scan keyboard
        mov     a,r7
        jnz     1f
        lcall   idle            ! no key pressed
        sjmp    0b
1:      clr     c
```

```
        subb    a,r6
        jz      6f              ! same key
        cjne    r7,#10,2f       ! end of index
        sjmp    7f
2:      cjne    r7,#0x20,3f     ! character or function ?
        sjmp    5f
3:      jnc     5f              ! character
        cjne    r7,#7,4f        ! edit or store action ?
        mov     a,flags         ! speak action
        setb    speak_pressed   ! remember it
        mov     flags,a
        sjmp    7f
4:      jnc     0b
5:      lcall   key_action      ! perform required action
        sjmp    0b
6:      lcall   auto_repeat
        sjmp    0b
7:      cjne    r5,#0,8f        ! empty index ?
        ajmp    index_error     ! do nothing then
8:      ret
!
restore_buffer:
!       restore buffer
        mov     p2,#0           ! copy to pointer
        mov     r0,#0
        mov     r1,#0
        mov     dptr,#320       ! copy from pointer
0:      setb    ramsel
        movx    a,@dptr
        movx    @r0,a
        clr     ramsel
        inc     dptr            ! increment pointers
        inc     r0
        cjne    r0,#0,1f
        inc     p2
        inc     r1
1:      cjne    r1,#1,0b        ! all 320 bytes copied ?
        cjne    r0,#64,0b
        ret
!
!
search_index:                   ! ask index and look for it in the list
        mov     r0,bufl         ! save it
        mov     r1,bufh
        mov     r2,mode
        lcall   bank_1          ! use register-bank 1
        lcall   save_buffer
        lcall   read_index
        mov     a,bufl
        clr     c
        subb    a,#80           ! calculate index length
        mov     r1,a
        mov     dptr,#buffer    ! start of list
search: setb    ramsel
        movx    a,@dptr         ! length of this field
        clr     ramsel
```

```
        jz      not_found       ! end of list
        mov     r5,dph          ! save start of field
        mov     r4,dpl
        inc     dptr
        setb    ramsel
        movx    a,@dptr         ! index length
        clr     ramsel
        mov     r2,a            ! save index length
        clr     c
        subb    a,r1
        jnz     incorrect       ! wrong index length
!       compare indexes
        mov     p2,#0
        mov     r0,#80          ! address of user supplied index
        inc     dptr
0:      setb    ramsel
        movx    a,@dptr         ! stored index
        mov     r3,a
        movx    a,@r0           ! user suplied index
        clr     ramsel
        clr     c
        subb    a,r3            ! compare characters in index
        jnz     incorrect       ! mismatch
        inc     dptr
        inc     r0
        djnz    r2,0b           ! whole index compared ?
correct:                        ! whole index is the same
        setb    c
        ret
incorrect:                      ! index is not the same
        mov     dph,r5          ! start of current field
        mov     dpl,r4
        setb    ramsel
        movx    a,@dptr         ! length of current field
        clr     ramsel
        add     a,dpl           ! calculate start of next field
        mov     dpl,a
        jnc     1f
        inc     dph
1:      sjmp    search          ! test next field
not_found:                      ! index string is not present
        clr     c
        ret
!
!
index_error:
        lcall   clr_screen
        mov     r0,#6
        mov     dptr,#lcd2+1
0:      lcall   lcd_rdy_d
        mov     a,r0
        movc    a,@a+pc
        jz      1f              ! end of message
        movx    @dptr,a         ! character to display
        inc     r0
        sjmp    0b
```

```
!       next string must be 64 characters
.ascii  "Afkorting niet in geheugen aanwezig !                  "
.asciz  "Druk op de spatiebalk om verder te gaan."
1:      lcall   keyb            ! wait for space
        mov     a,r7
        jnz     2f
        lcall   idle
        sjmp    1b
2:      cjne    a,#' ',1b
        clr     c
        subb    a,r6
        jz      1b
        lcall   bank_0          ! back to normal
        mov     bufl,r0
        mov     bufh,r1
        mov     mode,r2
        lcall   restore_buffer
        lcall   refr_screen
        mov     sp,#0x0f        ! re-initialise stack pointer
        ljmp    main
!
!       memory full :
!       give error message and go back to main program.
!
full:   lcall   clr_screen
        mov     r0,#6
        mov     dptr,#lcd2+1
0:      lcall   lcd_rdy_d
        mov     a,r0
        movc    a,@a+pc
        jz      1f              ! end of message
        movx    @dptr,a         ! character to display
        inc     r0
        sjmp    0b
!       next string must be 64 characters
.ascii  "Geheugen is vol !                                      "
.asciz  "Druk op de spatiebalk om verder te gaan."
1:      lcall   keyb            ! wait for keyboard input
        cjne    r7,#' ',1b
        lcall   refr_screen
        mov     sp,#0x0f        ! re-initialise stack pointer
        ljmp    main

!
!
!       AUXILARY ROUTINES
!
!       Check if key is an auto-repeating one and perform auto-repeat
!
auto_repeat:
        cjne    r7,#1,0f        ! delete key
        sjmp    repeat_key
0:      cjne    r7,#2,1f        ! cursor up
        sjmp    repeat_key
1:      cjne    r7,#3,2f        ! cursor down
        sjmp    repeat_key
```

```
2:      cjne    r7,#4,3f        ! cursor left
        sjmp    repeat_key
3:      cjne    r7,#5,4f        ! cursor right
        sjmp    repeat_key
4:      cjne    r7,#6,no_repeat ! insert character
repeat_key:
        mov     a,flags
        jb      repeat,0f       ! repeat already active ?
        mov     th0,#0xff       ! initial delay (0.5 sec)
        mov     tl0,#0x56
        clr     tf0
        setb    tr0             ! run timer
        mov     a,flags
        setb    repeat          ! set auto-repeat flag
        mov     flags,a
        ret
0:      jnb     tf0,no_repeat
        mov     th0,#0xff       ! repeat time (0.1 sec)
        mov     tl0,#0xdd
        clr     tf0
        lcall   key_action      ! same key acts again
no_repeat:
        ret
!
!
!       Seek first line of this sentence.
!       Input : r4 & r5
!       Output : dptr = startaddress of this sentence.
!
line_begin:
        mov     a,r4            ! X position
        mov     r1,a
        mov     a,#8
        clr     c
        subb    a,r4
        mov     r0,a            ! X position relative to last line
        mov     a,r3            ! continuation lines byte
0:      djnz    r0,1f
        sjmp    2f
1:      rl      a               ! shift until current line is in bit 7
        sjmp    0b
2:      rlc     a               ! continuation line bit in C
        jnc     3f              ! this is no continuation line
        dec     r1              ! otherwise examine line before
        sjmp    2b
3:      mov     a,r1            ! r1 is the first line of this sentence
        mov     b,#40
        mul     ab
        mov     dph,b           ! dptr is startaddress of sentence in buff
        mov     dpl,a
        ret
!
!       Seek last line of this sentence
!       Input : r4 & r5
!       Output: r5 = 0
!               r4 = next sentence
```

```
!               dptr according to r4 & r5
!
line_end:
        mov     a,r4            ! seek for cont. line bit for
        inc     a               !   line following current line
        inc     a
        mov     r1,a
        mov     a,r3
        clr     c
0:      rrc     a
        djnz    r1,0b
1:      jnc     2f              ! seek last line of this sentence
        inc     r4
        rrc     a
        sjmp    1b
2:      mov     r5,#0           ! go to end of this sentence
        inc     r4
        mov     a,r4
        mov     b,#40
        mul     ab
        mov     dpl,a
        mov     dph,b
        ret
!
!
!       fill buffer with spaces
!
empty_buffer:
        push    dph
        push    dpl
        mov     dptr,#0
0:      mov     a,#' '
        setb    ramsel
        movx    @dptr,a
        clr     ramsel
        inc     dptr
        mov     a,dph
        cjne    a,#1,0b         ! all 320 bytes had ?
        mov     a,dpl
        cjne    a,#64,0b
        pop     dpl
        pop     dph
        ret
!
!
!       Decrement datapointer
!
dec_dptr:
        clr     c
        mov     a,dpl
        subb    a,#1
        mov     dpl,a
        mov     a,dph
        subb    a,#0
        mov     dph,a
        ret
```

```
!
!
!
!       select working registerbank
!       save r7 : current key
!
bank_0: mov     a,r7
        clr     psw(3)
        mov     r7,a
        ret
!
bank_1: mov     a,r7
        setb    psw(3)
        mov     r7,a
        ret
```

```
#
.list
!*************************************************
!*                                               *
!*      M O D U L E    S C R E E N               *
!*                                               *
!*      Screen routines.                         *
!*                                               *
!*                                               *
!*                                               *
!*      Copyright  december 1985                 *
!*      Institute for perception research, IPO   *
!*      Eindhoven, the Netherlands.              *
!*                                               *
!*      Writer : Ir. R.J.H. Deliege              *
!*                                               *
!*************************************************
!
!
!       Register usage :
!
!       R3 : bit x = 1 : line x is a continuation line
!       R4 : X position of cursor
!       R5 : Y position of cursor
!
#include "decl.h"
!
.sect   _screen
.define lcd_rdy_d,lcd_rdy_r,clr_screen,refr_screen,goto_xy,line_feed,cont_
.define cursor_up,cursor_down,cursor_right,cursor_left
!
!       LCD ready routines.
!       Read LCD status and wait until ready.
!       Uses accumulator !
!
!       lcd_rdy_d : dptr contains LCD address
!
lcd_rdy_d:
        movx    a,@dptr         ! read LCD status
        jb      acc(7),lcd_rdy_d
        ret
!
!       lcd_rdy_r : p2 contains upper byte LCD address, r0 lower byte
!
lcd_rdy_r:
        movx    a,@r0
        jb      acc(7),lcd_rdy_r
        ret
!
!
!       Clear screen
!
clr_screen:
        push    dpl             ! save dptr
        push    dph
        mov     a,r0            ! and r0
```

```
        push    acc
        mov     dptr,#lcd1
        mov     r0,#4           ! 4 LCD's
0:      acall   lcd_rdy_d
        mov     a,#1            ! clear screen command
        movx    @dptr,a         ! to LCD
        mov     a,dph
        add     a,#4            ! next LCD
        mov     dph,a
        djnz    r0,0b
        pop     acc
        mov     r0,a
        pop     dph
        pop     dpl
        ret
!
!
!       Refresh screen
!
refr_screen:
        push    dpl             ! save dptr and r0
        push    dph
        mov     a,r0
        push    acc
        acall   clr_screen
        mov     x_save,r4       ! save cursor position
        mov     y_save,r5
        clr     a
        mov     r4,a            ! start from 0,0
        mov     r5,a
        mov     r0,#1           ! lower byte LCD address
        mov     p2,#lcd1_up     ! upper
        mov     dptr,#0         ! buffer address
0:      acall   lcd_rdy_r       ! LCD ready ?
        setb    ramsel          ! enable RAM
        movx    a,@dptr         ! read buffer
        clr     ramsel
        movx    @r0,a           ! write to LCD
        inc     dptr
        mov     a,dph           ! all 320 bytes had ?
        cjne    a,#1,1f
        mov     a,dpl
        cjne    a,#64,1f
        sjmp    2f
1:      inc     r5              ! increment y position
        cjne    r5,#40,0b       ! new line ?
        mov     r5,#0           ! y position
        inc     r4              ! increment x position
        acall   goto_xy
        mov     a,lcd_up
        mov     p2,a            ! new upper byte LCD address
        sjmp    0b
2:      mov     r4,x_save       ! restore cursor position
        mov     r5,y_save
        acall   goto_xy
        pop     acc
```

```
        mov     r0,a
        pop     dph
        pop     dpl
        ret
!
!
!       Scroll
!
scroll: push    dpl             ! save dptr and r0
        push    dph
        mov     a,r0
        push    acc
        mov     r4,#7           ! new position is 7,0
        mov     r5,#0
        mov     dptr,#280       ! new buffer pointer
        mov     bufh,dph
        mov     bufl,dpl
0:      mov     dptr,#40        ! copy from pointer
        clr     a
        mov     p2,a            ! copy to pointer
        mov     r0,a
1:      setb    ramsel          ! enable RAM
        movx    a,@dptr         ! move char
        movx    @r0,a
        clr     ramsel
        inc     dptr            ! increment copy from pointer
        mov     a,dph           ! all 320 bytes had ?
        cjne    a,#1,2f
        mov     a,dpl
        cjne    a,#64,2f
        sjmp    3f
2:      inc     r0              ! increment copy to pointer (lower)
        cjne    r0,#0,1b
        inc     p2              !                           (upper)
        sjmp    1b
3:      mov     a,r3            ! continuation lines
        clr     c               ! new line is no continuation
        rrc     a               ! scroll it also
        mov     r3,a            ! and save it
        jnb     acc(0),4f       ! first line should be no continuation
        dec     r4              ! if so, scroll another line
        mov     a,bufl          ! and decrement buffer pointer by 40
        clr     c
        subb    a,#40
        mov     bufl,a
        mov     a,bufh
        subb    a,#0
        mov     bufh,a
        sjmp    0b
4:      mov     dph,bufh        ! fill empty bufferspace with spaces
        mov     dpl,bufl
5:      mov     a,#' '
        setb    ramsel
        movx    @dptr,a
        clr     ramsel
        inc     dptr
```

```
        mov     a,dph
        cjne    a,#1,5b         ! end of buffer ?
        mov     a,dpl
        cjne    a,#64,5b
        acall   refr_screen
        pop     acc
        mov     r0,a
        pop     dph
        pop     dpl
        ret
!
!
!       Goto X,Y : positions the cursor according to r4 and r5 (X & Y)
!
goto_xy:
        push    dph             ! save used registers
        push    dpl
        mov     a,r0
        push    acc
        mov     dptr,#lcd1      ! LCD 1 address
        mov     r0,#4           ! 4 LCD's
0:      acall   lcd_rdy_d
        mov     a,#0x0e         ! cursor off command
        movx    @dptr,a
        mov     a,dph
        add     a,#4            ! next LCD
        mov     dph,a
        djnz    r0,0b
        mov     a,r4            ! X position (0..7)
        clr     acc(0)
        rl      a
        mov     dptr,#lcd1      ! LCD 1 address
        add     a,dph
        mov     dph,a           ! dptr points to correct LCD
        mov     lcd_up,a        ! and store it for later use
        acall   lcd_rdy_d
        mov     a,r4            ! X position
        rrc     a               ! bit 0 in carry
        mov     a,r5            ! Y position (0..39)
        jnc     1f              ! cursor on first line of this display
        add     a,#64           ! move to second line
1:      setb    acc(7)          ! to make it a set crsr adr command
        movx    @dptr,a
        acall   lcd_rdy_d
        mov     a,#0x0f         ! cursor on command
        movx    @dptr,a
        pop     acc
        mov     r0,a
        pop     dpl
        pop     dph
        ret
!
!
!       Line feed
!
line_feed:
```

```
        push    dph
        push    dpl
        mov     a,r0
        push    acc
        inc     r4              ! X position
        mov     r5,#0           ! Y position
        cjne    r4,#8,0f        ! buffer exceeded ?
        acall   scroll          ! scroll buffer (and screen)
        sjmp    3f
0:      acall   goto_xy         ! position cursor
        mov     a,r4            ! calculate buffer pointer
        mov     b,#40
        mul     ab
        mov     dph,b
        mov     bufh,b
        mov     dpl,a
        mov     bufl,a
        mov     r0,#40          ! fill new line with spaces
1:      mov     a,#' '
        setb    ramsel
        movx    @dptr,a
        clr     ramsel
        inc     dptr
        djnz    r0,1b
        mov     a,r4            ! new and next line are no continuation
        inc     a
        mov     r0,a
        clr     c
        mov     a,#0xfe         ! prepare mask
2:      rlc     a               ! shift zero to bit for current line
        djnz    r0,2b
        anl     a,r3            ! mask continuation line byte
        mov     r3,a
        acall   refr_screen     ! update screen
3:      pop     acc
        mov     r0,a
        pop     dpl
        pop     dph
        ret
!
!
!       Continuation line
!
cont_line:
        mov     a,r4            ! X pos
        inc     a
        mov     r0,a
        setb    c               ! this is a continuation line
        mov     a,#0
0:      rlc     a               ! prepare mask
        djnz    r0,0b
        orl     a,r3            ! mask continuation line byte
        mov     r3,a
        ret
!
!
```

```
!       Cursor movements
!
cursor_up:
        dec     r4              ! X position
        cjne    r4,#0xff,upd
        mov     r4,#0           ! stay on first line
        sjmp    upd
cursor_down:
        inc     r4              ! X position
        cjne    r4,#8,upd
        mov     r4,#7           ! stay on last line
        sjmp    upd
cursor_right:
        inc     r5              ! Y position
        cjne    r5,#40,upd
        mov     r5,#0
        inc     r4              ! X position
        cjne    r4,#8,upd
        mov     r4,#7           ! no more right
        mov     r5,#39
        sjmp    upd
cursor_left:
        dec     r5              ! Y position
        cjne    r5,#0xff,upd
        mov     r5,#39
        dec     r4              ! X position
        cjne    r4,#0xff,upd
        mov     r4,#0           ! no more left
        mov     r5,#0
upd:    mov     a,r4            ! update buffer pointer
        mov     b,#40
        mul     ab
        add     a,r5
        jnc     0f
        inc     b
0:      mov     bufh,b
        mov     bufl,a
        acall   goto_xy         ! update cursor
        ret
!
```

```
#
.list
!*************************************************
!*                                               *
!*      M O D U L E    S P E A K                 *
!*                                               *
!*      Send buffer to diphone board             *
!*      to make it audible.                      *
!*                                               *
!*                                               *
!*                                               *
!*      Copyright  december 1985                 *
!*      Institute for perception research, IPO   *
!*      Eindhoven, the Netherlands.              *
!*                                               *
!*      Writer : Ir. R.J.H. Deliege              *
!*                                               *
!*************************************************
!
!
#include "decl.h"
!
.sect   _speak
.define speak
.extern line_begin, bank_0, bank_1, dec_dptr
!
speak:  clr     diph_pwr_on     ! power on diphone board
        lcall   line_begin      ! seek start of sentence
!
!       Send buffer to diphone board
!
        clr     ti              ! clear transmitter int.
send_loop:
        jnb     diph_rdy,.      ! wait until diphone board is ready
        setb    ramsel
        movx    a,@dptr
        clr     ramsel
        acall   transl          ! check char before transmission
        acall   send
        inc     dptr
        cjne    a,#'.',1f       ! stop with full stop
        sjmp    2f
1:      cjne    a,#'?',send_loop! or question mark
2:      mov     r0,#0           ! error flag
        mov     th0,#0xfe       ! 1 sec
        mov     tl0,#0xa0
        clr     tf0
        setb    tr0             ! run timer
3:      jnb     speak_rdy,4f    ! wait until diphone board is speaking
        jb      tf0,error       ! or time out
        sjmp    3b
4:      clr     tr0             ! stop timer
        ret
!
!
!       Diphone board didn't understand it
```

```
!
error:  cjne    r0,#0,0f
        ret                     ! diphone board didn't speak at all
0:      mov     a,r0            ! ESC count
        clr     c
        subb    a,#8            ! calculate error position
        push    acc
        lcall   line_begin
        mov     a,r1            ! first line of this sentence
        mov     r4,a            ! cursor on start of sentence
        mov     r5,#0
        pop     acc
        mov     r0,a
1:      lcall   cursor_right    ! update cursor to error position
        djnz    r0,1b
        mov     mode,#edit_mode ! go in edit mode
        ret
!
!
!       Translate some special characters and numbers
!
transl: cjne    a,#'e',0f
        mov     a,#'Y'          ! e -> Y
        ret
0:      cjne    a,#'g',1f
        mov     a,#'Q'          ! g -> Q
1:      cjne    a,#'9',2f
        acall   num             ! a = 9
        ret
2:      jc      3f
        ret                     ! a > 9
3:      cjne    a,#'0',4f
        acall   num             ! a = 0
        ret
4:      jc      5f
        acall   num             ! a > 0
5:      ret                     ! a < 0
!
!
!       Translate numbers
!       register usage : r0 & r1 : general purpose
!                        r2      : digits or i
!                        r3      : j
!                        r4      : k
!                        r6      : i3
!                        r7      : reserved (last key)
!
num:    push    acc
        lcall   bank_1          ! use register bank 1
        pop     acc
        mov     r2,#0           ! nr of digits in number
0:      inc     r2
        inc     dptr
        setb    ramsel
        movx    a,@dptr         ! read next character
        clr     ramsel
```

```
        cjne    a,#'9',1f       ! is it a digit ?
        sjmp    0b              ! a = 9
1:      jc      2f
        sjmp    4f              ! a > 9
2:      cjne    a,#'0',3f
        sjmp    0b              ! a = 0
3:      jc      4f
        sjmp    0b              ! a > 0
4:      mov     r0,#number      ! clear buffer
        mov     r1,#6
        clr     a
5:      mov     @r0,a
        dec     r0
        djnz    r1,5b
        mov     r0,#number
        mov     a,r2            ! store digits
        mov     r1,a
6:      lcall   dec_dptr
        setb    ramsel
        movx    a,@dptr
        clr     ramsel
        clr     c
        subb    a,#'0'          ! convert ASCII to digit
        mov     @r0,a
        dec     r0
        djnz    r1,6b
        mov     a,r2            ! update dptr
        mov     r1,a
7:      inc     dptr
        djnz    r1,7b
        mov     a,r2
        dec     a
        mov     b,#3
        div     ab
        mov     r2,a            ! i := (digits-1) div 3
8:      acall   next3digits
        jnc     8b
        lcall   bank_0          ! back to original registers
        setb    ramsel
        movx    a,@dptr         ! a := next character
        clr     ramsel
        ret
!
next3digits:
        mov     a,r2
        mov     b,#3
        mul     ab
        mov     r6,a            ! i3 := 3 * i
        mov     a,flags
        jb      overlap,overlapping
        ajmp    non_overlapping
overlapping:                    ! overlapping case
        cjne    r2,#0,0f
        ajmp    non_overlapping ! i = 0
0:      mov     b,#7
        acall   nr
```

```
        mov     r1,a
        mov     b,#6
        acall   nr
        mov     b,r1
        mul     ab
        add     a,b
        cjne    a,#0,2f         ! nr[7-i3] * nr[6-i3] <> 0
1:      ajmp    non_overlapping
2:      mov     b,#5
        acall   nr
        mov     r1,a
        mov     b,#4
        acall   nr
        add     a,r1
        cjne    a,#0,1b         ! nr[5-i3] + nr[4-i3] <> 0
        mov     b,#6
        acall   nr
        mov     r3,a            ! j := nr[6-i3]
        mov     b,#7
        acall   nr
        mov     r4,a            ! k := nr[7-i3]
        acall   hundreds        ! hundreds(j,k)
        mov     b,#8
        acall   nr
        mov     r3,a            ! j := nr[8-i3]
        mov     b,#9
        acall   nr
        mov     r4,a            ! k := nr[9-i3]
        acall   tens            ! tens(j,k)
        mov     a,r2
        dec     a
        mov     r4,a            ! k := i-1
        acall   thousands       ! thousands(k)
        dec     r2
        dec     r2              ! i := i-2
        ret
non_overlapping:                ! non overlapping case
        mov     r3,#0           ! j := 0
        mov     b,#4
        acall   nr
        mov     r4,a            ! k := nr[4-i3]
        acall   hundreds        ! hundreds(j,k)
        mov     b,#5
        acall   nr
        mov     r3,a            ! j := nr[5-i3]
        mov     b,#6
        acall   nr
        mov     r4,a            ! k := nr[6-i3]
        acall   tens            ! tens(j,k)
        cjne    r2,#0,0f
        sjmp    1f              ! i = 0
0:      mov     b,#4
        acall   nr
        mov     r1,a
        mov     b,#5
        acall   nr
```

```
        add     a,r1
        mov     r1,a
        mov     b,#6
        acall   nr
        add     a,r1
        jnz     1f              ! nr[4-i3] + nr[5-i3] + nr[6-i3] <> 0
        clr     c               ! reset exit flag
        sjmp    2f
1:      mov     a,r2
        mov     r4,a            ! k := i
        acall   thousands       ! thousands(k)
2:      dec     r2              ! i := i-1
        ret
!
tens:                           ! write 10*j + k
        cjne    r4,#0,1f
        cjne    r3,#0,0f        ! k = 0
        ret                     ! j = 0
0:      mov     a,r3
        mov     r4,a
        mov     r3,#2
        acall   choose          ! choose(2,j)
        ret
1:      cjne    r3,#0,2f        ! k <> 0
        sjmp    3f              ! j = 0
2:      cjne    r3,#1,4f
3:      acall   choose          ! j = 1 : choose(j,k)
        ret
4:      mov     a,r3
        push    acc             ! save j
        mov     r3,#0
        acall   choose          ! choose(0,k)
        mov     r0,#8           ! write 'en'
5:      mov     a,r0
        movc    a,@a+pc
        jz      6f
        lcall   send
        inc     r0
        sjmp    5b
.asciz  ' EN '
6:      mov     r3,#2
        pop     acc             ! j
        mov     r4,a
        acall   choose          ! choose(2,j)
        ret
!
hundreds:                       ! write (10*j + k)*100
        mov     a,r3
        add     a,r4            ! a := j + k
        cjne    a,#1,0f
        sjmp    1f              ! j+k = 1
0:      acall   tens            ! tens(j,k)
1:      cjne    r4,#0,3f
2:      ret                     ! k = 0
3:      mov     r0,#8           ! write 'honderd '
4:      mov     a,r0
```

```
        movc    a,@a+pc
        jz      2b
        lcall   send
        inc     r0
        sjmp    4b
.asciz  "HONDYRD "
!
thousands:                      ! write 1000**k
        cjne    r4,#0,0f
        setb    c               ! k = 0 : set exit flag
        ret
0:      mov     r0,#8           ! write 'duizend, '
1:      mov     a,r0
        movc    a,@a+pc
        jz      2f
        lcall   send
        inc     r0
        sjmp    1b
.asciz  "DUIZYND, "
2:      clr     c               ! reset exit flag
        ret
!
choose: mov     a,r4
        dec     a
        cjne    r3,#0,ch0
        mov     b,#8            ! j = 0
        mul     ab              ! calculate index
        add     a,#9            ! add offset
        mov     r0,a
0:      mov     a,r0
        movc    a,@a+pc
        jnz     1f
        ret
1:      lcall   send
        inc     r0
        sjmp    0b
.asciz  "'-EEN  "
.asciz  "'TWEE  "
.asciz  "'DRIE  "
.asciz  "'VIER  "
.asciz  "'VIJF  "
.asciz  "'ZES   "
.asciz  "'ZEUVYN"
.asciz  "'-ACHT "
.asciz  "'NEEGYN"
ch0:    cjne    r3,#1,ch1
        mov     b,#12           ! j = 1
        mul     ab              ! calculate index
        add     a,#9            ! add offset
        mov     r0,a
0:      mov     a,r0
        movc    a,@a+pc
        jnz     1f
        ret
1:      lcall   send
        inc     r0
```

```
        sjmp    0b
.asciz  "'-ELF      "
.asciz  "'TWAALF    "
.asciz  "'DERTIEN   "
.asciz  "'VEERTIEN  "
.asciz  "'VIJFTIEN  "
.asciz  "'ZESTIEN   "
.asciz  "'ZEUVYNTIEN"
.asciz  "'-ACHT-TIEN"
.asciz  "'NEEGYNTIEN"
ch1:    mov     b,#11           ! j = 2
        mul     ab              ! calculate index
        add     a,#9            ! add offset
        mov     r0,a
0:      mov     a,r0
        movc    a,@a+pc
        jnz     1f
        ret
1:      lcall   send
        inc     r0
        sjmp    0b
.asciz  "'TIEN      "
.asciz  "'TWINTIG   "
.asciz  "'DERTIG    "
.asciz  "'VEERTIG   "
.asciz  "'VIJFTIG   "
.asciz  "'ZESTIG    "
.asciz  "'ZEUVYNTIG"
.asciz  "'TACHTIG   "
.asciz  "'NEEGYNTIG"
!
nr:                             ! a := nr[b-i3]
        mov     a,#number-6     ! a points to nr[0]
        add     a,b
        clr     c
        subb    a,r6            ! subtract i3
        mov     r0,a
        mov     a,@r0           ! a := nr[b-i3]
        ret
!
!
!       Send character in accumulator to serial port
!
send:   jnb     diph_rdy,.      ! wait until diphone board is ready
        mov     sbuf,a          ! send char
        clr     diph_rdy        ! diphone board is not ready anymore
        jnb     ti,.            ! wait for transmitter empty
        clr     ti
        ret
```