PE1MFB MEMORY KEYER

BY

Drs. Remco den Besten, PE1MFB

SUMMARY:

The PE1MFB memory keyer is the easiest and cheapest solution to obtain a memory keyer which you can be proud of. It offers the following features :

1 - 255 words per minute

8 memories which can be
- programmed
- read
- repeated continuously or delayed
- put into 'beacon mode'

paddle activity can be iambic or MFB-squeeze

carrier mode

simple I/O to the user (two displays, 5 switches)

memory treatment can be
- normal i.e. same read speed and options for all memories
- unique i.e. read speed and options for each specific memory
- direct i.e. reading memories by pushbuttons directly

on-board side tone oscillator

MANUAL

After power-up the PE1MFB memory keyer introduces itself by flashing 'PE1MFB'. After this the CW speed in WPM will be shown in hexadecimal on the displays. If there is a RAM failure 'E0' (error# 0) will be displayed due to the RAM memory check after power-up.

DEFAULT MENU

The CW speed, in Words Per Minute, can be altered by pressing F3 (decrement) or F4 (increment).

Pressing F2 results in a carrier. It is stopped by any key or paddle press.

After power-up the keyer will be iambic (can be changed, see patches further ,, this means by simultaneously pressing the dot and dash, the keyer outputs alternating dots and dashes. When MFB-squeeze is desired (try it out!) press F5. The display will show 'Fb' for a while. Pressing F5 again will put the keyer in the iambic mode again indicated by 'dF' (default).

By pressing F1, the memory menu is entered.

MEMORY MENU

There are several memory modes which allow different memory access and treatment. The memory mode is specified by a letter preceding the memory number. A 'n' indicates normal memory mode, an 'u' unique mode. Changes in memory mode will be delt with later. With F2 a memory can be selected from 0 to 7, after which it will turn to 0 again with the next F2 press.

Assume we selected memory 0. To program it press F5. The display will show '-0'. Programming starts with a short dot press only (not too long otherwise the dot will be stored). Any other key or dash press quits. When programming starts the memory will be erased automatically. So incase we accidentally pressed F5 we can exit without losing the information which may be stored. Programming the memories is done with a fixed programming speed. It can be changed, see later.

Reading a memory can be done in two ways: by pressing F3 or a dot press. During memory Read the display will show 'r[memory#]'. Reading stops automatically after 10 counts silence or.

Pressing F4 results in Continuous repeating the memory and 'c[memory#]' will be displayed.

Reading (or repeating) a memory can be stopped with any key or paddle press.

Returning to the default menu is done with a dash press.

Pressing F1 again results in entering the ...

SPECIAL MENU

In the special menu certain options can be selected concerning memory access and treatment.

The special menu characterizes itself by flashing display output.

When repeating a memory a waittime can be specified. By pressing F3 the waittime in seconds is displayed. The waittime may vary between 0 and 255 (FFh) seconds. A dot press increases the time, a dash press decreases.

Any keypress (i.e. F1 - F5) returns back to the special menu.
When memoryrepeat is selected (F4 in the memory menu) and waittime is specified, a countdown in seconds will be displayed after the memory is read. With meteorscatter no waittime (0 secs.) of course must be specified. Waittime for example can be handy when giving automatic CQ. During countdown the keys and paddle are checked every second for potential stop.

With F4 memories are put into 'beacon mode'. If waittime is specified a carrier is outputted during countdown while repeating memories. If no waittime is specified pressing F4 will result in error1 ('E1').
Only when waittime is specified F4 toggles between 'br' (Beacon Repeat) and 'nr' (Normal Repeat).

One can choose if all the memories must have the same treatment or if every memory must have special options different from one another. In the Normal mode reading and repeating a memory will done with the speed specified in the default menu (the keyer speed). Also waittime and/or beacon mode affects all memories. Pressing F5 toggles between Unique and Normal memory mode. Both in the memory menu and in the special menu, the user is informed which mode is pending by a letter ('u' for Unique,'n' for Normal) followed by the memory#.

When Unique memory mode is specified, the read speed of every memory can be altered with F2 and the paddle. For example, memory# 3 is selected. The display shows (flashing) 'u3'. Pressing F2 displays the current read speed of this memory. It can be changed with the paddle similar as the waittime. After power-up of the keyer all the memories will have the same read speed, i.e. the default keyer speed after power-up.

When we are in the Normal memory mode (the displays shows e.g. 'n2') the program speed of the memories can be changed by pressing F2. The program speed is uniform. Regardless of which memory mode is pending, all memories will be programmed with the same speed. Note also that the program speed is independent of the keyer speed or memory read speed. When e.g. performing Meteor Scatter programming a memory can be done without reducing the keyer speed or memory read speed.

Subsequently waittime and/or beacon mode can be specified for every memory with F3 and/or F4 respectively.

You can leave the special menu by pressing a dash or F1. You will enter the default menu again.

When a dot is pressed in the special menu you will enter the ..

DIRECT MEMORY MENU

The direct memory mode allows direct access to four memories. The display will show 'd?'. When you programmed memory#0 - 3 pressing F2 to F5 will result in reading memory# 0 - 3 respectively. In the direct memory mode the keyer is active. When reading a memory the display will show 'd[memory#]'.

The speed of the keyer will be the same as in the default menu. The speed of the memories depends on which memory mode is pending. The direct mode is especially designed for contest purposes where direct memory access is wanted. Paddle activity in the direct memory mode overrules reading of memories.

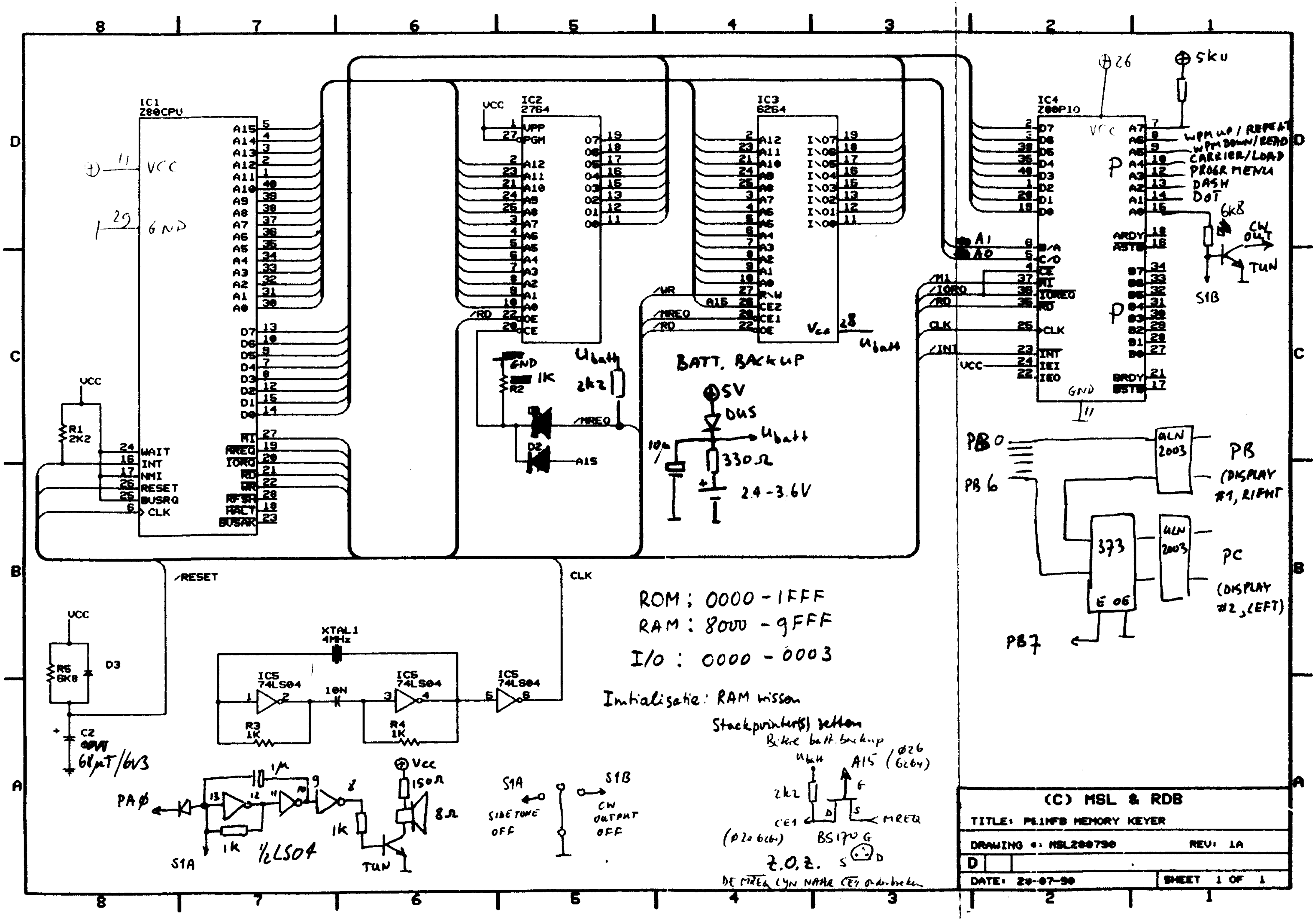
To exit the direct memory mode press F1. You will return to the default menu.

The access to different menus is sequential: default -> memory -> special -> default (or direct -> default).

TECHNICAL INFORMATION:

The PE1MFB memory keyer is a small single board computer (dedicated system). On board are a Z80A-CPU, Z80A-PIO, 8 Kbyte SRAM (e.g. 6264, 4364), 8 Kbyte (EP)ROM (2764) and minor additional hardware. The system clock must be around 4 MHz because of software delays (a standard 4.19 MHz crystal from an old digital clock suits perfectly). The PIO is addressed at I/O address 00h, RAM from 8000h and (EP)ROM from 0000h. Port B of the PIO with an additional latch (LS373) will be used to control the displays, port A for the paddle function keys and CW output.

PE1MFB MEMORY KEYER



ROM : 0000 - 1FFF
RAM : 8000 - 9FFF
I/O : 0000 - 0003

Initialisatie: RAM wissen
Stackpointer(s) zetten

Betere batt. backup

Z.O.Z.

DE MREQ LYN NAAR CE1 onderbreken

(C) MSL & RDB

TITLE: PE1MFB MEMORY KEYER

DRAWING #: MSL280790 REV: 1A

D

DATE: 28-07-90 SHEET 1 OF 1

```
     page  63
     .list
     title     PA3FYM memory keyer software (c) R. den Besten
(PA3FYM)
     .z80      ; we will run Z80 code
     aseg      ; make absolute segments

;last edit date : 30-06-92, changed (PE1MFB to PA3FYM)

;PIO bit assignments
;port A bit (inputs active low)
;    0    CW output
;    1    dot input
;    2    dash                 / direct menu
;    3    program / special menu toggle
;    4    carrier  / change memory  /  read speed memories
;    5    wpm down / read memory /  waittime for repeat memory
;    6    wpm up     / repeat memory   /  beacon mode
;    7    iambic or mfb toggle / program memory  / memory mode
;port  B  bit  (all  outputs)  bit  0-6  for  displays,  bit7  for
switching 2nd display
;    0    e
;    1    d
;    2    c
;    3    b
;    4    a
;    5    g
;    6    f


;Nog implementeren:
;- verschillende geheugens achter elkaar lezen

true equ  0ffh
false     equ  0
bb2  equ  true      ;true for Bigboard ][ software development
wr_wpm    equ  18        ;words per minute when loading memory
def_wpm   equ  18        ;default words per minute after cold
start
silence equ    10        ;silence   counts   for   end   read   or
repeat memory
sel_spd   equ  150       ;memory selection speed with paddle
backfl    equ  0abh      ;backup identifier
ram  equ  8000h          ;start address of 8 kbyte static RAM

     if   bb2
mrom equ  0f000h         ;BigBoard ][ monitor rom entry table
inkey     equ  mrom+9         ;console input
?key equ  mrom+6         ;status  :  0  =  no  char,  ffh  =  char
available
conout    equ  mrom+12        ;console output
     endif

pio  equ  00h       ;base i/o address of PIO
pioad     equ  pio       ;pio/a data register
pioac     equ  pio+1          ;pio/a control register
piobd     equ  pio+2          ;pio/b data register
```

```
piobc     equ  pio+3          ;pio/b control register

     if   bb2
out3 equ  0dbh      ;BigBoard gen.purp. I/O port #3
i_o2 equ  0dah      ;BigBoard gen.purp. I/O port #2
     else
out3 equ  piobd
i_o2 equ  pioad
     endif

Fdot equ  11111100b ;dot press for option
Fdash     equ  11111010b
F1   equ  11110110b ;push buttons
F2   equ  11101110b
F3   equ  11011110b
F4   equ  10111110b
F5   equ  01111110b

;several 7 segment display codes
disp_P    equ  79h       ;'P'
disp_r    equ  21h       ;'r'
disp_E    equ  73h       ;'E'
disp_C    equ  53h       ;'C'
disp_L    equ  43h       ;'L'
disp_cc equ    23h       ;'c'
disp_H    equ  6dh       ;'H'
disp_t    equ  63h       ;'t'
disp_b    equ  67h       ;'b'
disp_nn   equ  25h       ;'n'
disp_y    equ  6eh       ;'y'
disp_U    equ  4fh       ;'U'
disp_N    equ  5dh       ;'N'
disp_uu   equ  07h       ;'u'
disp_?    equ  39h       ;'?'
mem_siz   equ  1536      ;PE1MFB  memory  keyer  memory  size  in
bytes

     if   bb2
     org  0100h
     else
     org  0000h
     endif

romstrt   equ  $
init:     defb 0,0,0          ;three NOPs to stabelise
     di             ;first mask out interrupts
     jr   initpio

mfbstrt   equ  $
     defb '(C) PA3FYM'   ;copyright message from me (pa3fym)
mfblen    equ  $-mfbstrt


;initialise pio
initpio:ld     a,00001111b    ;set  port  B  to  mode  0  (output
only)
     out  (piobc),a ;output to controlreg. B
     ld   hl,pioAtab     ;hl first address table
     ld   b,piolen  ;b=bytecount
     ld   c,pioac        ;c=control register port A
     otir           ;output data to control port A


     ld   a,(backup)     ;check for backupflag
     cp   backfl         ;if ab already in memory then skip
following
     jr   z,start0

;perform RAM test
     ld   hl,ram         ;get RAM address
     ld   c,0
     ld   a,1       ;test byte
rtstl:    ld   b,32      ;nr of 256 byte blocks (32 = 8 kbyte)
rst2:     ld   (hl),a         ;write test byte
     rlca           ;rotate to test all bits
     inc  hl
     inc  c
     jr   nz,rst2
     djnz rst2      ;finish after total 8 kbyte

     ld   c,32      ;nr of 256 byte blocks (32 = 8kbyte)
rtst3:    dec  hl
     rrca
     cp   (hl)      ;verify test pattern that is written
     jr   z,okido        ;if OK then proceed
     ld   a,disp_E+80h   ;else show 'E0' on displays
     out  (out3),a  ;which indicates RAM failure
     and  7fh
     out  (out3),a
     ld   a,5fh          ;'0' on right display
     out  (out3),a
     jr   $         ;stick forever
okido:    djnz rtst3
     dec  c
     jr   nz,rtst3

;now do some initialisation
     ld   hl,ram         ;fill RAM with 00h
     ld   de,ram+1
     ld   bc,8191        ;we have 8K RAM
     ld   (hl),0
     ldir

     ld   hl,ramcopy     ;move RAM resident data to proper
place
     ld   de,ram
     ld   bc,copylen
     ldir

     ld   a,(rdm_wpm)    ;get default words per minute after
coldstart
     ld   hl,wpm         ;fill in all the pointers
     ld   b,9
def: ld   (hl),a
     inc  hl
     djnz def
```

```
start0:   if   bb2
     ld   (stack),sp     ;save original  (BB][) stackpointer
     endif

     ld   sp,stack  ;define (new) stack

     ei             ;finally enable interrupts

     iff  bb2
     call hello          ;introduce the pa3fym memory keyer
     endif


;-------------------------------------------------------------
----------------
;DEFAULT menu entry
;-------------------------------------------------------------
----------------

startl:   call clc_wpm        ;calculate delay word
     ld   a,(wpm)
     call puthex         ;output wpm rate to the displays
start2:   call paddle         ;check paddle activity

     if   bb2
     call ?key      ;any keypress exits to operating system
     jr   nz,zcpr2
     endif

     call const          ;check if a button has been pressed
     jr   z,start2  ;if not then recheck
     or   6         ;set bit 1,2 (mask out dot and dash)
     call defdel         ;default delay
     ld   hl,deftab ;point to default menu table
     ld   bc,defsiz/3
     cpir           ;find match with accu
     jr   nz,startl
     call search         ;jump to right routine
     jr   startl

     if   bb2
zcpr2:    ld   sp,(stack)     ;restore old stack
     ret            ;and return to CPM
     endif

deftab:   defb fl,f2,f3,f4,f5
     defw keymod,incwpm,decwpm,carrier,mem_men
defsiz    equ  $-deftab

incwpm:   ld   hl,wpm
     inc  (hl)
     ret
decwpm:   ld   hl,wpm
     dec  (hl)
     ret

keymod:   ld   a,(iamflg)     ;get iambic flag
```

```
     cpl             ;toggle
     ld   (iamflg),a
     or   a
     ld   de,256*71h+67h ;when pa3fym mode show 'Fb'
     jr   z,iam0
     ld   de,256*2fh+71h ;if iambic then show 'dF' (default)
iam0:     ld   (display),de
     call dis_out        ;show selected mode for a while
     ld   b,3
     ld   c,255
iam1:     call delay
     djnz iam1
     ret


;----------------------------------------------------------------
----------------
;MEMORY menu entry
;----------------------------------------------------------------
----------------
mem_men:call   defdel        ;smooth selection
mem3:     ld   a,(modflg)     ;determine memory mode
     or   a
     ld   a,disp_nn ;normal memory mode show 'n[m#]'
     jr   z,mem5          ;when normal skip following
     ld   a,disp_uu ;unique memory mode show 'u[m#]'
mem5:     call sho_mem        ;display [mode],[m#]
     call clc_bit        ;set proper memory bit mask
mem0:     in   a,(i_o2)  ;anti dender
     ld   b,a
     in   a,(i_o2)
     cp   b
     jr   nz,mem0
     and  0feh      ;mask out CW outp.
     call defdel         ;with delay
     ld   hl,memtab ;point to memory menu table
     ld   bc,sizmem/3
     cpir           ;find match with accu
     jr   nz,mem0
     call search
     cp   0abh      ;AB = stopcode to return to default menu
     jr   nz,mem3
     ret

memtab:   defb fdot,fdash,f1,f2,f3,f4,f5
     defw wrt_mem,rpt_,rd_,chmem,special,ex_ab,rd_
sizmem    equ  $-memtab

rd_: call rd_mem
     jr   del_ex
rpt_:     ld   (reptfl),a     ;set reptfl to not zero
     call rpt_mem
     xor  a
     ld   (reptfl),a     ;clear reptfl
del_ex:   jp   defdel         ;return follows in exit routine

chmem:    ld   hl,memory ;else change m#
     inc  (hl)
     ld   a,(hl)
     cp   8         ;check if memory >= 7
     ret  c         ;if not then OK
     ld   (hl),0         ;else force selection memory 0
     ret

;--------------------------------------------------------------
----------------
;SPECIAL menu entry, all options flashing, indicating special
menu
;--------------------------------------------------------------
----------------
special:ld     a,(modflg)
     or   a
     ld   a,disp_nn ;show flashing 'n[m#]' or 'u[m#]'
     jr   z,sp3
     ld   a,disp_uu
sp3: call sho_mem
     call con_fl
     jr   nz,sp0         ;if keypress then check option
     call dis_off        ;shut down displays
     call con_fl         ;also when displays are off check
options
     jr   z,special
sp0: ld   c,150          ;smooth selection
     call delay
     ld   hl,spectab     ;point to special menu table
     ld   bc,speclen/3
     cpir           ;find match with accu
     jr   nz,special     ;if no match then poll again
     call search         ;execute option
     cp   0abh      ;AB = stopcode to..
     jr   nz,special
     ret            ;return to default menu

spectab:defb   fdot,fdash,f1,f2,f3,f4,f5
     defw memmod,beacon,wait,memwpm,ex_ab,ex_ab,dir_mem
speclen   equ  $-spectab

memmod:   ld   a,(modflg)
     cpl
     ld   (modflg),a
     ret

beacon:   ld   iy,waitfl ;point first to waitfl
     call modchk         ;memory mode
     ld   a,(iy)         ;check if we have wait time else
beaconmode..
     or   a         ;is useless, because we have normal repeat
then
     ld   a,1
     jp   z,error        ;if we have no waittime then error
(return fol)
     ld   iy,beacfl ;point to beaconflags
     call modchk         ;point to right flag depending on
memory
     ld   a,(iy)         ;get beaconflag
     cpl            ;toggle
     ld   (iy),a          ;store it
     or   a
     ld   a,disp_b  ;show 'br' for Beacon Repeat
     jr   nz,mode         ;or 'nr' for Normal Repeat
     ld   a,disp_nn
mode:     ld   (dis2),a
     ld   a,disp_r
     ld   (disl),a
     call dis_out
     ld   b,3
spl: ld   c,255           ;show message for a while
     call delay
     djnz spl
     ret            ;return to special menu

wait:     ld   iy,waitfl ;point to waitflags
     call modchk          ;determine mode
     jr   flupdwn         ;change it
memwpm:   ld   iy,wpm          ;point to wpm array
     call modchk          ;check mode
     ld   de,256*76h+disp_r;if  unique  mode  then  show  'Sr'
(speed read)
     jr   nz,gochg
     ld   iy,wrt_wpm      ;else change pointer to wrt_wpm
     ld   de,256*76h+disp_P;when  normal  mode  show  'SP'  (speed
progr)
gochg:    ld   (display),de   ;show info a while
     call dis_out
     ld   b,4
hoi: call defdel
     djnz hoi

;routine for changing (iy) with flashing displays
flupdwn:ld     a,(iy)          ;get contents pointer
     call puthex          ;output it on the displays
     call con_fl
     jr   nz,flup         ;if keypress then check option
     call dis_off
     call con_fl          ;check port while flashing
     jr   z,flupdwn
flup:     ld   c,150           ;smooth selection
     call delay
     bit  1,a        ;check upcount (dotpress)
     jr   nz,fldwn
     inc  (iy)       ;increase contents ptr
     jr   flupdwn
fldwn:    bit  2,a        ;check downcount (dash press)
     ret  nz         ;anything else exits
     dec  (iy)       ;decrease contents ptr
     jr   flupdwn         ;poll again


;routine in which during c=200 delay 20 times port A is chec-
ked
con_fl:   ld   b,20       ;total delay approx. c=200
     ld   c,10
cfl0:     call delay           ;to make selection more smooth
     call constl          ;check port a much as possible
     ret  nz         ;if key press exit
     djnz cfl0
     xor  a          ;set zeroflag
     ret             ;our time is over, exit

;-------------------------------------------------------------
----------------
;DIRECT memory entry = keyer and direct access to 4 memories
;-------------------------------------------------------------
----------------
dir_mem:ld     (dirflg),a     ;make dirflg not zero
     ld   de,256*2fh+disp_?
     ld   (display),de   ;show 'd?'
     call dis_out

dir0:     call paddle          ;we want to CW and directly have
access to
     call const          ;4 memories
     jr   z,dir0
     or   6          ;set bit 1,2
     call defdel

     ld   hl,dirtab
     ld   bc,dirsiz/3
     cpir
     jr   nz,dir0
     call search
     cp   0abh       ;ABh is stop code to exit to default menu
     jr   nz,dir_mem     ;else poll again
     ret

dirtab:   defb f1,f2,f3,f4,f5
     defw m3,m2,m1,m0,dir_ex
dirsiz    equ  $-dirtab


dir_ex:   xor  a
     ld   (dirflg),a     ;make dirflg zero
     jp   ex_ab          ;return to default menu

m0:  ld   a,0        ;memory0
     jr   read_it
m1:  ld   a,1        ;..1
     jr   read_it
m2:  ld   a,2        ;..2
     jr   read_it
m3:  ld   a,3        ;..3
read_it:ld     (memory),a     ;store m#
     call clc_bit
     ld   ix,(wpmw) ;get original wpm because read speed may
differ
     call dirmem
     ld   (wpmw),ix ;restore org wpmw
     ret


;Determination of relative speed necessary for wpm delay
routine.
```

```
;A translation table is used because the delay time is reci-
procal related to
;the words per minute.

clc_wpm:ld     a,(wpm)         ;get Words Per Minute
clc_wp0:ld     e,a
     ld   d,0
     ld   hl,wpm_tab      ;for the conversion table
     add  hl,de
     add  hl,de           ;multiply by 2 because we have 16 bit
words
     ld   a,(hl)          ;get lo byte
     inc  hl
     ld   h,(hl)          ;get hi byte
     ld   l,a        ;hl now contains converted number
     ld   (wpmw),hl ;store it
     ret

;Calculation of the memory mask byte.
;memory 8 to F not implemented yet
clc_bit:push   af
     push hl
     ld   hl,memory
     ld   a,(hl)
     and  00000111b ;remove offset for 8 to F
     or   a          ;check memory0/8 else we will shift 255
times!
     ld   a,00000001b    ;mask byte for memory 0 or 8
     jr   z,clcl          ;if memory=0,8 then skip calculation
     ld   a,(hl)          ;get m# again
     and  07h        ;bit 3 not necessary, mask it
     ld   b,a        ;b times shift left
     ld   a,00000001b    ;set bitpattern
clc0:     sla  a          ;shift left 00000010 = memory1,9 etc.
     djnz clc0       ;how many times we have to shift?
clcl:     ld   (bit_mem),a    ;store result e.g. memory7,F =
10000000
     pop  hl
     pop  af
     ret



;asynchronous memory write routine
;------------------------------------------------------------
----------------
wrt_mem:exx               ;save regs
     ld   ix,(wpmw) ;save original wpmw
     ld   a,20h          ;'-' indicates we entered programming
mode
     call sho_mem
     call defdel          ;smooth selection
tjek:     call constl          ;check if we're not mistaking
     jr   z,tjek          ;wait for dot press indicating start
program
     cp   Fdot       ;check for dot press
     jr   nz,wrt_ex ;any other key exits
```

```
;arrive here when a dotpress occured
     call defdel         ;smooth otherwise dot will be stored
     call era_mem        ;first clear proper memory
     ld   a,disp_P  ;show P[m#] indicating programming memory
     call sho_mem
     ld   a,(wrt_wpm)    ;get default programming speed
     call clc_wp0        ;calculate delay word
     ld   de,messge ;point to start of memories

;arrive here when programming the memories with the paddle
pad_wrt:ld     hl,messge+mem_siz-4  ;(-4  safety  for  a  dash
1110!)
     ld   a,e       ;check for memory overflow
     cp   l         ;check if e < l
     jr   c,roger        ;if so then proceed
     ld   a,d       ;else get hibyte
     cp   h         ;check if d < h
     jr   nc,wrt_ex ;if  memory  overflow  (i.e.  d  >=  h)  then
exit
     sbc  hl,de          ;amount of used memory
     ld   a,h
     cp   l         ;check if 1/2 to 2/3 of memory is filled
     jr   nz,roger
     ld   a,0       ;if so warn user by deletion of the P
     call sho_mem
roger:    in   a,(i_o2)
     bit  1,a       ;dot entry
     jr   z,dot_         ;it  was  a  dot  so  jump  direct  to  dot
handling
roger0: in     a,(i_o2)  ;recheck
     bit  2,a       ;dash entry
     jr   z,dotb
     bit  l,a       ;if no dash check dot
     jr   nz,no_act ;when no dot/dash fill in blanks

dot_:     call wrdot
     ld   a,(iamflg)     ;get keyer mode
     or   a
     jr   nz,roger0 ;iambic (we had a dot so first check dash)
     jr   pad_wrt        ;MFB-squeeze

;arrive here when a dash is pressed
dotb:     call wrdash         ;if so perform it
     bit  1,c       ;check dot press during dash
     jr   nz,pad_wrt
     call wrdot          ;if so store dot
     ld   a,(iamflg)     ;check for pa3fym keyer mode
     or   a
     jr   z,pad_wrt
     jr   roger0

no_act:   xor  a         ;when nothing happens fill in a zero
     call stor_it        ;store it in the appropriate memory
     call wpm_del        ;wait one count subsequently
     call const          ;check  for  keypress  indicating  end
wrt mode
     jr   z,pad_wrt
```

```
wrt_ex:   call defdel        ;smooth with delay
     ld   (wpmw),ix ;restore original wpmw
     exx            ;restore regs
     ret            ;return to memory menu

wrdot:    ld   b,1       ;count# for dot
     jr   wrdotl

wrdash:   ld   b,3       ;count# for dash
wrdotl:   ld   l,0feh
wrdot0:   in   a,(i_o2)  ;check for dot/dash while dash/dot is
pending
     ld   a,01h          ;set bit0 which is CW output
     out  (i_o2),a
     call stor_it
     call wpm_del        ;delay
     and  1
     cp       1
     jr   z,wrdot2
     ld   c,a
wrdot2:   djnz wrdot0
     ld   a,0
     out  (i_o2),a  ;reset bit 0 to finish token with
     call stor_it
     call wpm_del        ;one count delay
     ret

;store 0 or 1 in (DE) in the right memory, automatic counter
increment follows
stor_it:push   hl
     ld   hl,bit_mem     ;get memory mask byte
     ld   L,(hl)         ;store in L
     or   a         ;check memorydata for 0
     jr   z,stor_0  ;if so store 0 else store 1
     ld   a,(de)         ;read memories contents
     or   L         ;set proper memory bit to 1
stor_ex:ld     (de),a         ;store result
     inc  de        ;increase memory addr
     pop  hl
     ret
stor_0:   ld   a,L       ;get mask byte
     cpl            ;make complement
     ld   L,a       ;restore it
     ld   a,(de)         ;read memories contents
     and  L         ;set proper bit to 0
     jr   stor_ex

;memory erase routine
;Originally programmed as a special option 'erase'. Now prior
to program a
;memory an auto erase is performed. This eliminates a special
push button
era_mem:call   dis_off        ;shut down displays while era-
sing
     ld   de,messge
     ld   hl,messge+mem_siz-4
     ld   a,(bit_mem)
     cpl
     ld   c,a
era0:     ld   a,e       ;check memory overflow
     cp   l
     jr   c,eral
     ld   a,d
     cp   h
     jr   nc,era_ex ;if so then exit

eral:     ld   a,(de)         ;get contents of memories
     and  c          ;erase proper bit in memory
     ld   (de),a          ;store result
     inc  de
     jr   era0
era_ex:   ld   a,(modflg)     ;determine memory mode
     or   a
     ld   a,disp_nn ;show that we're still in the memory menu
     jr   z,eraex0
     ld   a,disp_uu
eraex0:   call sho_mem
     ret

;direct memory mode entry
dirmem:   exx
     ld   a,2fh          ;show 'd [m#]'
     jr   rdm0

;repeat memory entry
rpt_mem:exx
     ld   iy,beacfl
     call modchk          ;point to correct flag
     ld   a,(iy)          ;get beaconflag
     or   a
     ld   a,disp_cc ;when zero then we have normal repeat
     jr   z,rdm0
     ld   a,disp_b  ;if beacon mode show 'b[m#]'
     jr   rdm0      ;else 'c[m#]'

;memory read routine
rd_mem:   exx            ;save regs by switching to alternati-
ve set
     ld   a,disp_r  ;during read display 'r[m#]' (read m#)
rdm0:     call sho_mem
     ld   iy,wpm          ;point to read speed array
     call modchk          ;check memory mode
     jr   z,repeat  ;when normal skip following
     ld   a,(iy)          ;get wpm for m#
     call clc_wp0         ;calculate new delay word


repeat:   ld   hl,messge+mem_siz-4
     ld   a,(bit_mem)    ;get memory byte mask
     ld   c,a       ;save it
     ld   b,silence ;get silence counts before auto-quit
     ld   de,messge
rd0: call constl          ;any keypress exits read mode
     jp   nz,rd_ex
     ld   a,e       ;check for end of memory
     cp   l
```

```
        jr   c,rdl
        ld   a,d
        cp   h
        jp   nc,rd_ex  ;if end of memory then exit
rdl:    ld   a,(de)
        and  c         ;check for 0 or 1
        jr   z,rd_nul
        ld   a,01h          ;output 1 on bit0 output port
        out  (i_o2),a
        call wpm_del
        inc  de
        ld   b,silence ;reset silence counts
        jr   rd0

rd_nul:      ld   a,0
        out  (i_o2),a
        call wpm_del
        inc  de
        djnz rd0        ;after a silent period exit automatically

;arrive here when finished reading memory
        ld   a,(dirflg)     ;if we are in direct mode exit imme-
diately
        or   a         ;no repeat or wait allowed
        jr   nz,rd_ex

        ld   a,(reptfl)     ;check for repeat mode
        or   a         ;if flag is zero
        jr   z,rd_ex        ;exit..
        ld   iy,waitfl ;point to proper waitflag
        call modchk         ;check memory mode
        ld   a,(iy)         ;OK we want to repeat, do we have to
wait?
        or   a         ;when 0 direct repeat
        jr   z,repeat

        ld   iy,beacfl ;point to beacon mode flags
        call modchk         ;check memory mode
        ld   a,(iy)         ;check for beacon mode during wait
        or   a
        jr   z,silwait
        ld   a,1       ;output carrier during wait time
        out  (i_o2),a

silwait:ld     hl,(wpmw) ;get words per minute delay word
        ex   de,hl          ;save it
        ld   hl,52400  ;approx. 1 second count for wpm_del routi-
ne
        ld   (wpmw),hl ;misuse the wpm_del routine
        ld   iy,waitfl
        call modchk         ;point to correct waitfl again
        ld   b,(iy)         ;get waittime
bsec:        ld   a,b       ;output countdown to displays
        call puthex
        call wpm_del        ;1 sec delay
        call constl         ;check  paddle  and  buttons  every
second
        jr   nz,rep_ex ;if keypress then exit
```

```
     djnz bsec
     ex   de,hl          ;restore original wpmw delay word
     ld   (wpmw),hl
     ld   iy,beacfl
     call modchk         ;point to correct flag
     ld   a,(iy)         ;get beaconflag again
     or   a
     ld   a,disp_cc ;when zero then we have normal repeat
     jr   z,nobeac
     ld   a,disp_b  ;if beacon mode show 'b[m#]'
nobeac:   call sho_mem
     jp   repeat         ;and repeat

rep_ex:   ex   de,hl          ;restore wpmw delay word
     ld   (wpmw),hl
rd_ex:    ld   a,0       ;force CW output to zero
     out  (i_o2),a
     exx            ;restore registers
     ret            ;return without smooth (due to direct mem
mode)
                    ;smoothing is done in mem_men

error:    push bc
     call geteq          ;get display code
     ld   (disl),a  ;error code must be in A
     ld   a,disp_E  ;display E(rror)
     ld   (dis2),a
     ld   b,4       ;allow 4 flashes to emphasize ERROR
err0:     call dis_out
     ld   c,150          ;show for a while
     call delay
     call dis_off        ;shutdown displays
     ld   c,150          ;wait some time
     call delay
     djnz err0      ;determine how many flashes left
     pop  bc        ;exit
     ret


dis_off:ex     af,af          ;clear displays
     ld   a,80h
     out  (out3),a
     ex   af,af
     ret

;Words Per Minute delay routine
wpm_del:push   bc
     ld   bc,(wpmw) ;get delay word
del_0:    defb 0,0,0,0,0,0,0,0
     defb 0,0,0,0,0,0
     dec  c
     jr   nz,del_0
     in   a,(i_o2)
     djnz del_0
     pop  bc
     ret

ex_ab:    ld   a,0abh         ;abh is exit code to default
```

```
menu
defdel:   ld   c,sel_spd
;general purpose delay routine. Enter amount of delay in C
delay:    push bc
del0:     ld   b,0ffh
dell:     djnz dell
     dec  c
     jr   nz,del0
     pop  bc
     ret

;hello shows 'HI' flashing pa3fym in CW
hello:    ld   a,0ch          ;'I' is simulated by 'l'
     ld   (disl),a
     ld   a,disp_H  ;'H'
     ld   (dis2),a
     ld   hl,pa3fym ;get start of message
hel0:     ld   a,(hl)
     cp   5fh       ;5fh stop code (1st entry display table)
     jr   z,hel_ex
     ld   b,8       ;a byte has 8 bits
hell:     rla            ;rotate   left
     call c,dis_out ;if it was a 'l' then show 'HI'
     call nc,dis_off     ;else clear displays
     ld   c,105
     call delay
     djnz hell      ;rotate 8 times
     inc  hl        ;increase pointer
     jr   hel0      ;and get next address

hel_ex:   ld   c,255          ;wait a moment
     call delay
     ret            ;and return

     if   bb2
hexout: push   af        ;prints contents of A in hex
     rra
     rra
     rra
     rra
     call nibble
     pop  af
nibble: and    0fh
     add  a,90h
     daa
     adc  a,40h
     daa
     jp   conout
     endif

;puthex outputs contents of register C in hex to the displays
puthex:   push af        ;we need lo-nibble later on so push
it
     rra
     rra
     rra
     rra            ;high bits automatically masked out in
'geteq'
```

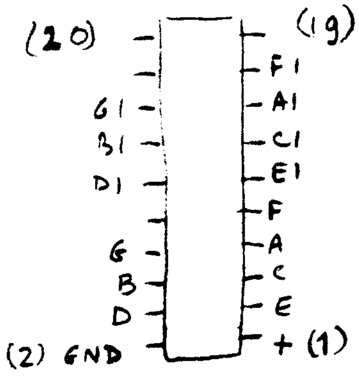
```
     call geteq          ;first translate hi-nibble
     ld   (dis2),a  ;store it at the right place
     pop  af        ;restore information
     call geteq          ;translate lo-nibble
     ld   (dis1),a  ;store it, printing on displays follows


;dis_out outputs contents of addr. 'dis1' and 'dis2' to the
two displays
;display 2 is left display, display 1 is right display

dis_out:ex     af,af'          ;output routine for pio
     ld   a,(dis2)  ;get data to be displayed on display 2;
     or   80h       ;set bit 7 to select left display
     out  (out3),a
     and  7fh       ;reset bit 7
     out  (out3),a
     ld   a,(dis1)  ;get data for display 1 (bit7 always 0)
     out  (out3),a  ;and output data to it
     ex   af,af'
     ret

sho_mem:ld     (dis2),a
     ld   a,(memory)     ;get m#
     call geteq          ;translate to display code
     ld   (dis1),a  ;store it
     jr   dis_out        ;RET follows in dis_out

;translation binary -> 7 segment format
geteq:    exx
     and  0fh       ;make in 0-f range (table has only 16
entries)
     ld   e,a
     ld   d,0
     ld   hl,dis_tab     ;get translation table
     add  hl,de          ;translate to display codes
     ld   a,(hl)         ;equivalent now in accu
     exx
     ret            ;tranlation completed and exit

;general keyer routine
paddle:   in   a,(i_o2)
     bit  1,a       ;dot entry
     jr   z,roger26 ;it was a dot so jump direct to dot hand-
ling
roger27:in     a,(i_o2)  ;recheck
     bit  2,a       ;dash entry
     jr   z,dash_
     bit  1,a       ;when no dash check dot
     ret  nz        ;no dot/dash so exit all this

;arrive here when dot has to be performed
roger26:call   dot
     ld   a,(iamflg)
     or   a
     jr   nz,roger27     ;iambic-squeeze (we had a dot, 1st
check dash)
     jr   paddle         ;MFB-squeeze
```

```
;arrive here when a dash has to be performed
dash_:      call dash      ;perform a dash
     bit    1,c        ;check dot-press during dash
     jr     nz,paddle ;immediate recheck when no dot during dash
     call dot          ;if there was a dot then now perform it
     ld     a,(iamflg)
     or     a
     jr     z,paddle
     jr     roger27           ;we had a dot so now check first for
a dash

dot: ld     b,1         ;one count for a dot
     jr     dot7
dash:       ld   b,3          ;3 counts for a dash
dot7:       ld   l,0feh
dot0:       ld   a,01h            ;set bit0 which is CW output
     out    (i_o2),a
     call wpm_del           ;1 count delay
     and    1           ;in wpm_del port is read mask out bit0
     cp     1           ;check if something happened
     jr     z,dot3            ;if not then skip
     ld     c,a         ;else save info in c
dot3:       djnz dot0        ;if we perform a dash do this three
times
     ld     a,0
     out    (i_o2),a   ;reset bit 0 to finish token with
     call wpm_del           ;one count delay
     ret

;routine for continuous carrier e.g. for adjusting a linear
amplifier
carrier:ld     a,01h           ;generate CW output
     out    (i_o2),a
     ld     c,200            ;smooth selection
     call delay
car0:       call constl
     jr     z,car0
     ld     a,0
     out    (i_o2),a   ;exit with no CW output
     ld     c,200            ;delay a bit
     call delay
     ret                ;return to main menu

;calculation of right memory pointer hl to all or specific
memory
modchk:     ld   a,(modflg)      ;check unique memory treatment
     or     a
     ret    z           ;if not then return direct
     inc    iy          ;else increase pointer
     ld     bc,(memory)
     ld     b,0         ;clear b
     add    iy,bc             ;add m# to pointer
     ret                ;hl points to specific memory pointer

search:     add  hl,bc           ;search table in hl for match
with accu
     add    hl,bc           ;add residue from cpir bytecount to
hl ..
```

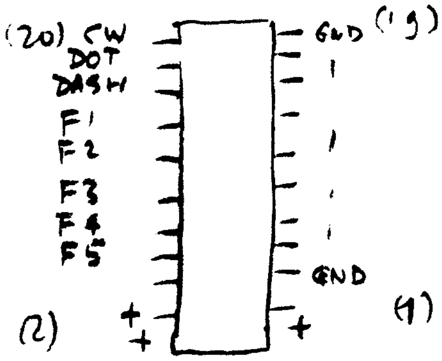
```
     add  hl,bc          ;3 times to get pointer
     ld   a,(hl)
     inc  hl
     ld   h,(hl)
     ld   l,a       ;hl now contains routine
     jp   (hl)      ;"call (hl)"

;$* callsign check routine, werkt nog niet (20-02-91)
check:    ld   b,6
     ld   hl,mfbstrt+4
     ld   de,chktab+5
check0:   ld   a,(de)
     cp   (hl)
     ret  nz
     inc  hl
     dec  de
     djnz check0
     ld   a,0abh
     call hexout
     ret

chktab:   defb 'BFM1EP'
chksiz    equ  $-chktab

const:    in   a,(i_o2)  ;read key and button port
     and  11111000b ;mask out dot, dash (and CW output)
     cp   11111000b ;make  compare:  f8  equivalent  with  no
keypress
     ret
const1:   in   a,(i_o2)  ;read key and button port
     and  11111110b ;mask out CW output only
     cp   11111110b
     ret

pioAtab:defb   11001111b ;select mode 3
     defb 11111110b ;only bit 0 output, bits 1-7 are inputs
     defb 00000000b ;00h is interrupt vector
     defb 00000111b ;disable.int.,OR,input active low,no mask
piolen    equ  $-pioAtab

;flash message pa3fym copyright!!
pa3fym:   defb 00101110b,11101000b,10111000b,10101011b,101110-
00b,10101110b
     defb 10001110b,10111011b,10001110b,11100000b
;nothing between here!! First entry of table is stopcode for
pa3fym message
;translation table for conversion BCD -> segment
dis_tab:defb   5fh,0ch,3bh,3eh,6ch,76h,77h,1ch    ;0,1,2,3,4,-
5,6,7
     defb 7fh,7eh,7dh,67h,53h,2fh,73h,71h    ;8,9,A,b,C,d,E,F


;Words per minute conversion table. This table supplied by a
little PASCAL
;program (I'm not that crazy!) has the equivalents for wpm.
The table numbers
;are obtained by the following equation : result = 76500/wpm.
The factor
```

```
;76500 results 'rounded' in one unit difference between 255
and 254 wpm.
;Because of the fact that the equation is assymptotic I didn't
wrote a routine
;which calculates 'result'. By the way, this is the fastest
solution
;and we have memory space enough!

wpm_tab:defw   637,65535,38250,25500,19125,15300,12750,10929,-
9563,8500
     defw 7650,6955,6375,5885,5464,5100,4781,4500,4250,4026
     defw 3825,3643,3477,3326,3187,3060,2942,2833,2732,2638
     defw 2550,2468,2391,2318,2250,2186,2125,2068,2013,1962
     defw 1912,1866,1821,1779,1739,1700,1663,1628,1594,1561
     defw 1530,1500,1471,1443,1417,1391,1366,1342,1319,1297
     defw 1275,1254,1234,1214,1195,1177,1159,1142,1125,1109
     defw 1093,1077,1062,1048,1034,1020,1007,994,981,968
     defw 956,944,933,922,911,900,890,879,869,860
     defw 850,841,832,823,814,805,797,789,781,773
     defw 765,757,750,743,736,729,722,715,708,702
     defw 695,689,683,677,671,665,659,654,648,643
     defw 637,632,627,622,617,612,607,602,598,593
     defw 588,584,580,575,571,567,562,558,554,550
     defw 546,543,539,535,531,528,524,520,517,513
     defw 510,507,503,500,497,494,490,487,484,481
     defw 478,475,472,469,466,464,461,458,455,453
     defw 450,447,445,442,440,437,435,432,430,427
     defw 425,423,420,418,416,414,411,409,407,405
     defw 403,401,398,396,394,392,390,388,386,384
     defw 382,381,379,377,375,373,371,370,368,366
     defw 364,363,361,359,357,356,354,353,351,349
     defw 348,346,345,343,342,340,338,337,336,334
     defw 333,331,330,328,327,326,324,323,321,320
     defw 319,317,316,315,314,312,311,310,308,307
     defw 306,305,304,302,301,300,299,298,297,295

ramcopy   equ   $
     .phase    ram
ramstrt   equ   $

;RAM scratch area
display:             ;16 bit entry for display (disl = lo byte)

dis1:     defb 0         ;data for display 1 (right display)
dis2:     defb 0         ;data for display 2 (left  display)

scrats:   defb 0,0,0,0,0,0,0,0,0

;flags for memory read, first entry is general, next are
memory specific
;if specified with modflg (must be true)
;  memory#:   all,0,1,2,3,4,5,6,7
reptfl:   defb 0,0,0,0,0,0,0,0,0;false= no repeat memory
waitfl:   defb 0,0,0,0,0,0,0,0,0;waittime  (secs.)  for  repeat
memory
beacfl:   defb 0,0,0,0,0,0,0,0,0;beacon mode flag
wpm: defb 0,0,0,0,0,0,0,0,0;memory read speed
memory:   defb 0         ;current memory
```

```
bit_mem:defb   0          ;AND bitmask for proper memory
wpmw:     defw 0          ;delay word for proper CW speed
dirflg:   defb false           ;true = direct memory mode
modflg:   defb false           ;0 = all memories same options
backup:   defb backfl          ;chicken/egg  flag  for  backup
option
patch:    defb 'PATCH'
iamflg:   defb false           ;keyer mode true = iambic, false
= pa3fym mode
rdm_wpm:defb   def_wpm         ;Words Per Minute storage
wrt_wpm:defb   wr_wpm          ;default  words  per  minute  pro-
gramming memory
copylen   equ  $-ramstrt
messge    equ  $          ;space for memories 0-7
stack     equ  messge+mem_siz+64

     .dephase
     end
```



DISPLAYS



GENCONN.

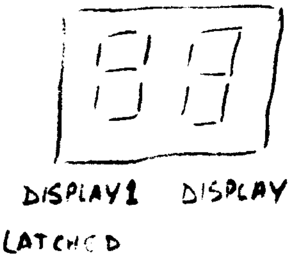
F1 = MEMORY ENTRY

F2 = CARRIER

F3 = WPM DWN

F4 = WPM UP

F5 = IAMBIC / MFB SQUEEZE



SEGMENTEN: