TITLE PAGE

INTERACTIVE TOY

(FURBY.ASM - Version 25)

INVENTOR: Dave Hampton

Attorney Docket No. 64799
FITCH, EVEN, TABIN & FLANNERY
Suite 900
135 South LaSalle Street
Chicago, Illinois 60603-4277
Telephone (312) 372-7842

```
;ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
ÍÍÍÍ»
;º
º
;º      SPC81A Source Code   (Version 25)
º
;º
º
;º      Written by: Dave Hampton  /  Wayne Schulz
º
;º      Date:         July 30, 1998
º
;º
º
;º      Copyright (C) 1996,1997,1998 by Sounds Amazing!
º
;º      All rights reserved.
º
;ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
ÍÍÍͼ
;
;
;*********************************************************************


;  remember   SBC    if there is a borrow carry is CLEARED
;  also SBC   if the two numbers are equal you still get a negative
result


;
;
;ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
ÍÍÍÍ»
;º   MODIFICATION LIST :
º
;


; Furby29/30/31/32
;      Final testing for shipment of code on 8/2/98.
;      Tables updated, motor speed updated, wake up/name fix
;      sequential tables never getting first entry,fixed.
;      New diag5.asm, Light3.asm (if light osc stalls it wont hang
system).
;
; Furby33
;      In motor brake routine, turn motors off before turning reverse
;      braking pulse on to save transistors.
;
; Furby34
;      Cleanup start code and wake routines.
;      Light sensor goes max dark and stays there to reff time, then
;      call sleep macro and shut down.
;
; Furby35
;      Adds four new easter eggs,BURP ATTACK, SAY NAME, TWINkLE SONG,
;      and ROOSTER LOVES YOU. Also add new names.
;
;
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
; Release 3

;; File "testR3a"

; 1. Light sensor has a hysteresis point of continually triggering
sensor.
; 2. Light sensor decrements two instead of one on hungry counter.
; 3. Diagnos 'r   de for light sensor wont trigger very easily.
; 4. When a furby -eceives the I.R. sleep command he sends the same
command
;     out before goi.- to sleep.
;
; 5. When hungry is  ow enough to trigger sick counter, each sensor
;     deducts two instead of one for each hit.
;
; 6. When diagnostics complete c'ear memory, reset hungry & sick to FF
;     randomly choose new name an  voice, then write EEPROM before
;     going to sleep. Also extend EEPROM diagnostic to test all locations
;     for pass/fail of device.
;
; 7. Add new light routine

; 8. Change hide and seek egg to light,light,light,tummy.

; 9. Change sick/hungry counter so that it can only get so sick and
;     not continue down to zero. (MAX_SICK)

;10. In diagnostics, motor position test ,,,, first goes forward
continuously
;     until the front switch is pressed, then goes reverse continuously
;     until the front switch is pressed again, and then does normal
position
;     calibration stopping at the calibration switch.

;11. On power up we still use tilt and invert to generate startup random
;     numbers, but if feed switch is pressed for cold boot, we use it to
;     generate random numbers, because it is controlled by the user where
;     the tilt and invert are more flaky.

;12. No matter what age, 25% of time he randomly pulls speech from age
;     to generate more Furbish at older ages.

;13. Twinkle song egg
;     When song is complete, if both front and back switches are pressed
;     we goto deep sleep. That means only the invert can wake us up, not
;     the tilt switch.


;ª
ª
;Èffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
ffff¼
;*****************************************************************
;*****************************************************************
;*****************************************************************
;*****************************************************************
;*****************************************************************
```

```
; Actual numeric value for TI pitch control

;  bit 7 set = subtract value from current course value
;        clr = add value to current course value
;  bit 6 set = select music pitch table
;        clr = select normal speech pitch table
;  bit 0-5 value to change course value (no change = 0)

; A math routine in 'say_0' converts the value for + or -
; if <80 then subtracts from 80 to get the minus version of 00
; ie, if number is 70 then TI gets sent 10 (which is -10)
; If number is 80 or > 80 then get sent literal as positive.

; NOTE: MAX POSITIVE IS 8F (+16 from normal voice of 00)
;       MAX NEGATIVE is 2F (-47 from normal voice of 00)

;This is a difference of 80h - 2Fh or 51h


; 8Fh is hi voice  (8f is very squeeeeeke)
; 2Fh lo voice ( very low)


; The math routine in 'Say_0' allows a +-decimal number in the speech
table.
; A value of 80 = no change or 00 sent to TI
; 81 = +1
; 8f = +16
;
;value of 7F = -1 from normal voice
;70 = -16

; The voice selection should take into consideration that the hi voice
; selection plus an aditional offset is never greater than 8f
; Or a low voice minus offset never less than 2f.

Voice1        EQU   83h   ;(+3) hi voice
Voice2        EQU   7Ah   ;(-6) mid voice
Voice3        EQU   71h   ;(-15) low voice

;;;; we converted to a random selection table, but since all voice
tables
;    use the equates plus some offset, we made the change in the SAY_0
;    routine. We always assign voice 3 which is the lowest, and based on
;    the random power up pitch selection, the ram location 'Rvoice'
holds
;    the number to add to the voice+offset received from the macro
table.

Voice EQU   Voice3          ;pitch (choose Voice1, Voice2,
Voice3)(voice2=norm)

; Select Voice3 since it is the lowest and then add the difference to
get
; Voice2 or Voice3. Here we assign that difference to an equate to be
; used in the voice table that is randomly selected on power up.

S_voice1  EQU  18 ;Voice3 + 18d = Voice1
S_voice2  EQU  09 ;Voice3 + 09d = Voice2
```

```
S_voice3  EQU  0  ;Voice3 + 00d = Voice3


;***********************************************************************

; Motor speed pulse width :
; Motor_on = power to motor, Motor_off is none.


Mpulse_on   EQU   16  ;
Mpulse_off  EQU   16  ;


Cal_pos_fwd EQU   134   ;calibration switch forward direction
Cal_pos_rev EQU   134   ;calibration switch forward direction

;***********************************************************************
;***********************************************************************
;***********************************************************************
;***********************************************************************
;***********************************************************************
;
;ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
;³                        PORTS                       ³
;³ SPC40A has : 16 I/O pins                           ³
;³ PORT_A 4 I/O pins  0-3                             ³
;³ PORT_C 4 I/O pins  0-3                             ³
;³ PORT_D 8 I/O pins  0-7                             ³
;³                                                    ³
;³                         RAM                        ³
;³                                                    ³
;³ SPC40A has : 128 bytes of RAM                      ³
;³ from $80 - $FF                                     ³
;³                                                    ³
;³                         ROM                        ³
;³ SPC40A has :                                       ³
;³ BANK0 user ROM from $0600 - $7FFF                  ³
;³ BA^K1 user ROM from $8000 - $FFF9                  ³
;³                                                    ³
;³                                                    ³
;³                       VECTORS                      ³
;³ NMI   vector  $7FFA / $7FFB                        ³
;³ RESET vector  $7FFC / $7FFD                        ³
;³ IRQ   vector  $7FFE / $7FFF                        ³
;ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
;ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
;³                        PORTS                       ³
;³ SPC120A has : 17 I/O pins                          ³
;³ PORT_A 4 I/O pins  0-3                             ³
;³ PORT_B 4 I/O pins  0,1,2,4,5                       ³
;³ PORT_C 4 I/O pins  0-3 input only                  ³
;³ PORT_D 8 I/O pins  0-7                             ³
;³                                                    ³
;³                         RAM                        ³
;³ SPC120A has : 128 bytes of RAM                     ³
;³ from $80 - $FF                                     ³
;³                                                    ³
;³                         ROM                        ³
;³ SPC120A has :
```

```
;³ BANK0 user RO         $0600 - $7FFA·                    ³
;³ BANK1 user RO         $8000 - $FFFF                     ³
;³ BANK2 user RO         $10000 - $17FFF                   ³
;³ BANK3 user RO         $1A000 - $1FFFF                   ³
;³                                                         ³
;³                                                         ³
;³                             VECTORS                     ³
;³ NMI   vector  $7FFA / $7FFB                             ³
;³ RESET vector  $7FFC / $7FFD                             ³
;³ IRQ   vector  $7FFE / $7FFF                             ³
;ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ


; unuseable areas in rom

;SPC40A:      8000H ÄÄ  DFFFH should be skiped (Dummy area)
;  bank 0 = 600 - 7FFA
;  bank 1 = 8000 - DFFF reserved , start @ E000 - FFFA

;SPC80A:      10000H ÄÄ 13FFFH should be skiped (Dummy area)
;  bank 0 = 600 - 7FFA
;  bank 1 = 8000 - FFFA
;  bank 2 = 10000-13FFF reserved ,  start at 14000 - 17FFF

;SPC120A: ;SPC120A: 18000H ÄÄ 19FFFH should be skiped (Dummy area)
;  bank 0 = 600 - 7FFA
;  bank 1 = 8000 - FFFA
;  bank 2 = 10000 - 17FFF
;  bank 3 = 18000 - 19FFF reserved ,  start at 1A000 - 1FFFA

;SPC256A: ;SPC256A: Non dummy area

;SPC512A: ;SPC512A: Non dummy area

;*****************************************************************

.CODE
.SYNTAX  6502
.LINKLIST
.SYMBOLS


;ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ PORT DIRECTION CONTROL REGISTER
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
Ports_dir        EQU    00     ; (write only)
;
; (4 I/O pins) controlled with each bit of this register
; you can't control each pin separately, only as a nibble
; 0 = input / 1 = output
;
; 7     6     5     4     3     2     1     0        (REGISTER
BITS)
; D     D     C     C     B     B     A     A        (PORT)
; 7654  3210  7654  3210  7654  3210  7654  3210     (PORT BITS)
;ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
ÄÄÄÄÄ

; ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ PORT CONFIGURATION CONTROL REGISTER
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ
```

```
;                  based on if the port pin is input or output
;
Ports_con        EQU     01     ; (write only)
;
; (4 I/O pins) controlled with each bit of this register
; 7      6      5      4      3      2      1      0          (REGISTER
BITS)
; D      D      C      C      B      B      A      A          (PORT)
; 7654   3210   7654   3210   7654   3210   7654   3210       (PORT BITS)

; port_a INPUTS can be either:
; 0 = float   1 = pulled high

; port_a OUTPUTS can be either:
; 0 = buffer   1 = upper (4) bits Open drain Pmos (source)
;                  lower (4) bits Open drain Nmos (sink)
;
; port_b INPUTS can be either:
; 0 = float   1 = pulled low

; port_b OUTPUTS can be either:
; 0 = buffer   1 = upper (4) bits Open drain Nmos (sink)
;                  lower (4) bits Open drain Nmos (sink)
;
; port_c INPUTS can be either:
; 0 = float   1 = pulled high
; port_c OUTPUTS can be either:
; 0 = buffer   1 = upper (4) bits Open drain Pmos (source)
;                  lower (4) bits Open drain Nmos (sink)
;
; port_d INPUTS can be either:
; 0 = float   1 = pulled low
; port_d OUTPUTS can be either:
; 0 = buffer   1 = Open drain Pmos (source)

;AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAA

;AAAAAAAAAAAAAAAAAAAAAAAAAAAAA I/O PORTS
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Port_A           EQU     02H     ; (read/write) for TI & speech recgn
CPU's
Data_D0            EQU   01H   ;bit 0 data nible port
Data_D1            EQU   02H   ;
Data_D2            EQU   04H   ;
Data_D3            EQU   08H   ;

Port_B             EQU   03H   ;b0/b1 = I/O  b4/b5 = inp only
TI_init            EQU   01H   ;B0 - TI reset control
TI_CTS             EQU   02H   ;B1 - hand shake to TI
IR_IN       EQU    10H   ;B4 - I.R. Recv data
TI_RTS             EQU   20H   ;B5 - TI wants data

Port_C           EQU     04H      ; (read/write)
Motor_cal   EQU    01H   ;C0 - lo when motor crosses switch
Pos_sen            EQU   02H   ;C1 - motor optical sensor (intt C1)
Touch_bck   EQU    04H   ;C2 - back touch
Touch_frnt  EQU    08H   ;C3 - front touch
```

```
Port_D         EQU     05H     ; (read/write)
Ball_side   EQU   01H   ;D0 - hi when on any side (TILT)
Ball_invert EQU   02H   ;D1 - hi when inverted
Light_in    EQU   04H   ;D2 - hi when bright light hits sensor
Mic_in          EQU   08H   ;D3 - hi pulse microphone input
Power_on    EQU   10H   ;D4 - power to rest of circuit
Motor_led   EQU   20H   ;D5 - motor I.R. led driver
Motor_lt    EQU   40H   ;D6 - motor drive left (forward)
Motor_rt    EQU   80H   ;D7 - motor drive right (reverse)

;AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAA

;AAAAAAAAAAAAAAAAAAAAAAAA DATA LATCH  PORT_D
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Latch_D        EQU     06H     ; (read)
; read to latch data from port_d, used for wake-up on pin change
;AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAA

;AAAAAAAAAAAAAAAAAAAAAAAA BANK SELECTION REGISTER
AAAAAAAAAAAAAAAAAAAAAAAAAA
Bank           EQU     07H     ; (read/write)  x x x x x x x b
; 0 = bank 0, 1 = bank 1       ;               7 6 5 4 3 2 1 0
; only two banks in SPC40a
;AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAA

;AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA WAKE UP
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Wake_up        EQU     08H     ; (read/write) x x x x x x x w
;                                             7 6 5 4 3 2 1.0

; w=(0=disable, 1=enable wake-up on port_d change)
; read to see if wake-up, or normal reset
; this is the only source for a wake-up
; Always reset stack on wake up.
;AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAA

;AAAAAAAAAAAAA  AAAAAAAAAAAAAA SLEEP
AAAAAAAAAAAAAAA. AAAAAAAAAA  AAAAAAAAAA
Sleep          E[illegible]      09H     ; (write)       x x x x x x x s
;                              ;               7 6 5 4 3 2 1 0
; s=(0=don't care, 1=s[illegible]
; writting 1 to bit0, f[illegible]es sleep
;AAAAAAAAAAAAAAAAAAAAA   AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAA

;AAAAAAAAAAAAAAAAAAAAAAAA TIMER A CONTROL REGISTER
AAAAAAAAAAAAAAAAAAAAAAAAAA
; this needs more work to understand DMH
TMA_CON            EQU     0BH     ; (write)
;
;
;                 7 6 5 4 3 2 1 0
;                 m x x x
;
;                 m= Timer one mode (0=Timer,1=Counter)
```

```
;
;           Bit3: IE1 Ā¿ IE1= 0: Counter clock= external clock from IOC2
;   .       Bit2: T1  Ā´     = 1, T1= 0: counter clock= CPUCLK/8192
;           Bit1: IE0 Ā´           T1= 1: counter clock= CPUCLK/65536
;           Bit0: T0  ĀÙ IE0= 0: Counter clock= external clock from IOC2
;                            = 1, T0= 0: counter clock= CPUCLK/4
;                                 T0= 1: counter clock= CPUCLK/64
;
;ĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀ
ĀĀĀĀĀ

;ĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀ INTERRUPTS
ĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀ
Interrupts      EQU     0DH     ; (read/write)
;
;       7 6 5 4 3 2 1 0
;       w m a b 3 2 1 e
;
;       w = (0=watch dog ON, power-on default) (1=watch dog OFF)
;       m = (0=Timer A generates NMI INT, 1=Timer A generates IRQ INT)
;       a = (0=Timer A interrupt off, 1=Timer A interrupt on)
;       b = (0=Timer B interrupt off, 1=Timer B interrupt on)
;       3 = (0=CPU CLK/1024 interrupt off,  1=CPU CLK/1024 interrupt
on)
;       2 = (0=CPU CLK/8192 interrupt off,  1=CPU CLK/8192 interrupt
on)
;       1 = (0=CPU CLK/65536 interrupt off, 1=CPU CLK/65536 interrupt
on)
;       e = (0=external interrupt off, 1=external interrupt on)
;           rising edge, from port_c bit1
;ĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀ
ĀĀĀĀĀ

;ĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀ TIMERS
ĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀ
; There are two 12bits timers.
; Timer A can be either a timer or a counter. (as set by TIMER_CON)
; Timer B can only be used as a timer.
;
; Timers count-up and on overflow from 0FFF to 0000, this carry bit will
, create an interrupt if the corresponding bit is set in INTERRUPTS
register.
; The timer will be auto reloaded with the user setup value, and
start,,,
; count-up again.
;
; Counter will reset by user loading #00 into register TMA_LSB and
TMA_MSB.
; Counter registers can be read on-t e-fly, this will not affect
register,,,
; values, or reset them.
;
;ĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀ
ĀĀĀĀĀ

;ĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀ TIMER A (low byte
)ĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀĀ
TMA_LSB         EQU     10H     (read/write)
;
; all 8bits valid (lower 8bits of 12bit timer)
```

```
;AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAA

;AAAAAAAAAAAAAAAAAAAAAAAAA TIMER A (high byte)
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
TMA_MSB          EQU      11H      (read/write)
; read          x  x  x  x  11 10 9  8    timer upper 4bits
;               7  6  5  4  3  2  1  0
;
   rite         x  x  t  c  11 10 9  8    timer upper 4bits
                7  6  5  4  3  2  1  0    register bit

;                t=(0=speech mode, 1=Tone mode)
;                this connects the AUDA pin to either
;                the DAC., or Timer generated square wave
;
;                c=(0=CPU clock,   1=CPU clock/4:
;AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAA

;AAAAAAAAAAAAAAAAAAAAAAAAA TIMER B (low byte
)AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
TMB_LSB          EQU      12H
;
; all 8bits valid (lower 8bits of 12bit timer)
;AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAA

;AAAAAAAAAAAAAAAAAAAAAAAAA TIMER B (high byte)
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
TMB_MSB          EQU      13H
; read          x  x  x  x  11 10 9  8    timer upper 4bits
;               7  6  5  4  3  2  1  0
;
; write         x  x  t  c  11 10 9  8    timer upper 4bits
;               7  6  5  4  3  2  1  0    register bit
;
;                t=(0=speech mode, 1=Tone mode)
;                this connects the AUDB pin to either
;                the DAC2, or Timer generated square wave
;
;                c=(0=CPU clock,   1=CPU clock/4:
;AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAA

;AAAAAAAAAAAAAAAAAAAAAAAAAAAA  D/A converters
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
DAC1             EQU      14H      ; (write)
DAC2             EQU      15H      ; (write)
;AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAA

;AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAA
; this needs more work to understand DMH
;     16H    ADCoutputPort16H:

DAC_ctrl     EQU   16H

;
```

```
;           Bit7: I/O 0: Disable ADC; 1: Enable ADC
;           Bit6: I/O
;           Bit5: I/O
;           Bit4: I/O
;           Bit3: I/O
;           Bit2: I/O
;           Bit1: I/O
;           Bit0: I/O
;AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAA
;ÚAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA¿
;³ Operating equate definition
;ÀAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAÙ
;EQdef

; to calculate sampl
; CPU clk/sample rat         or
; Hi & Lo timer reg co        = FFF
; FFF - divisor = valu      oa  hi & lo reg.

;ex: 6mHZ clk = 166nSEC

;******** start Tracker


;/* here is som  definition chnge of time interrupt constant */Tracker

;SystemClock:     EQU     6000000          ;Select 6000000Hz it will be the
same
                                           ;as before
SystemClock:      EQU     3579545          ;Select 3579545Hz while we are
use that
                                           ;crystal

TimeA_low:        EQU     <(4096-(SystemClock/5859))       ;put constant
definition
TimeA_hi:         EQU     >(4096-(SystemClock/5859))

TimeB_low:        EQU     <(4096-(SystemClock/1465))
TimeB_hi:         EQU     >(4096-(SystemClock/1465))


;********* end Tracker


Port_def    EQU   A7h   ;D hi=out,D lo=inp / C hi=out,C lo=inp
                        ;B hi=inp,B lo=out / A hi=out,A lo=out

Con_def           EQU   50H   ;D hi=out buffer, D lo=in pull lo
;                       ;C hi=out buffer, C lo=in pull hi
;                       ;B hi=in  hi-Z  , B lo=out buffer
;                       ;A hi=out buffer, A lo=out buffer
;

Intt_dflt   EQU   D0h   ;sets interrupt reg = no watchdog,irq
                        ; timer B , and EXt port C bit 1 = off

;***** run EQU's

;*************************************************************************
```

```
; Send a braking pulse to stop motor drift, and this EQU is a decimal
number
; that determines how many times through the 2.9 mSec loop (how many
loops)
; the brake pulse is on. If attempting to make single count jumps, the
; brake pulse needs to be between 26 and 30. For any jump greater than
10
; braking between 22 and 80 is acceptable. ( Long jumps are not critical
; but short jump will begin to oscillate if braking is too great.)


; 60 long & 20 short work at 3.6v and no pulse width

Drift_long  EQU   60 ;number times thru intt before clearing pulse
Drift_short EQU   25 ;

;********************************************************************

; set this with a number from 0 - 255 to determine timeout of all
sensors
; for the sequential increments. If it times out the table pointer
; goes back to the start, else each trigger increments through the
table.

; NOTE: this time includes the motor/speech execution time !!!

Global_time EQU   16     ; 1= 742 mSEC ;; 255 = 189.3 seconds


;********************************************************************
; This determines how long Firby waits with no sensor activity, then
; calls the Bored_table for a random speech selection.
; Use a number between 1 & 255. Should probably not be less than 10.

; SHOULD BE > 10 SEC TO ALLOW TIME FOR TRAINING OF SENSORS

Bored_yeld  EQU   40     ; 1= 742 mSEC ;; 255 = 189.3 seconds

;********************************************************************
;
; Each sensor has a sequential random split which must equal 16.
; Each sensor has a different assignment.
; The tables are formatted with the first X assignments random
; and the remaining as sequential.

Seq_front   EQU   8
Ran_front   EQU   8

Seq_back    EQU   9
Ran_back    EQU   7

Seq_tilt    EQU   10
Ran_tilt    EQU   6

Seq_invert  EQU   8
Ran_invert  EQU   8

Seq_sound   EQU   0
Ran_sound   EQU   16
```

```
Seq_light    EQU   0
Ran_light    EQU   16

Seq_feed     EQU   8
Ran_feed     EQU   8

Seq_wake     EQU   0
Ran_wake     EQU   16

Seq_bored    EQU   7
Ran_bored    EQU   9

Seq_hunger   EQU   5
Ran_hunger   EQU   11

Seq_sick     EQU   4
Ran_sick     EQU   12


; rev  furb11ja

; Each sensor also determines how often it is random or sequential
; as in 50/50  or 60/40  etc.
; These entries are subtracted from the random number generated
; and determine the split. (the larger here, the more likely sequential
pick)

Tilt_split  EQU   80h    ;
Invert_split      EQU    80h   ;
Front_split EQU   80h    ;
Back_split  EQU   80h    ;
Feed_split  EQU   80h    ;
Sound_split EQU   80h    ;
Light_split EQU   80h    ;
Bored_split EQU   80h    ;
Hunger_split      EQU    80h   ;
Sick_split  EQU   80h    ;

;**********************************  **********************************

Random_age  EQU   30h    ;at any age, below this number when a
;                        random number is picked will cause him
;                        to pull from the age 1 table. More Furbish.

;*********************************************************************

Learn_chg   EQU   31     ;amount to inc or dec training of words
;----------------------------------
Food        EQU   20h    ;amount to increase 'Hungry' for each feeding
Need_food   EQU   80h    ;below this starts complaining about hunger
Sick_reff   EQU   60h    ;below this starts complaining about sickness
Really_sick EQU   C0h    ;below this only complains about sickness
Max_sick    EQU   80h    ;cant go below this when really sick

Hungry_dec  EQU   01     ;subtract X amount for each sensor trigger
Sick_dec    EQU   01     ;subtract X amount for each sensor trigger
;----------------------------------
Nt_word           EQU    FEH   ;turn speech word active off
Nt_last           EQU    FBH   ;bit 2 off - last word sent to TI
```

```
Nt_term           EQU   F7h   ;bit 3 off -terminator to speech TI
Clr_spch    EQU   FCH   ;clears spch_activ & word_activ
CTS_lo            EQU   FDH   ;makes TI_CTS go lo
;--------
Motor_rev   EQU   FDH   ;clears motor fwd bit
Motor_inactv      EQU   FEh   ;kill motor activ bit
Motor_ntseek      EQU   FBh   ;kill motor seek bit
Motor_off   EQU   C0h   ;turns both motor lines off (hi)
Motor_revs  EQU   7FH   ;bit 7 lo
Motor_fwds  EQU   BFh   ;bit 6 lo
Ntmot_on    EQU   DFh   ;clears motor pulse on req
Nt_IRQdn    EQU   F7h   ;clear IRQ stat
Nt_Motor_led      EQU   DFh   ;motor opto led off
Motor_led_rst     EQU   100   ;X * 2.9 millSec for shut off time

Nt_Init_motor     EQU   FBh   ;cks motor speed only on wake up
NT_Init_Mspeed    EQU   F7h   ;clears 2nd part of motor speed test

Opto_spd_reld     EQU   80    ;number of IRQ to count opto pulse speed
Speed_reff  EQU   30    ;value to adjust speed to

Nt_macro_actv     EQU   7Fh   ;clears request
;--------
Not_bside   EQU   F7h   ;clear ball side done flag
Not_binvrt  EQU   EFh   ;clear ball invert done flag
Not_tch_bk  EQU   BFh   ;clear touch back sense done flag
Not_tch_ft  EQU   DFh   ;clear touch back sense done flag
Not_feed    EQU   FDh   ;clear feed sense done flag
Sound_reload      EQU   05    ;X * 742 milisec time between trigger
Snd_cycle_rled    EQU   02    ;sound sense referrence cycle timer
;--------
Light_reload      EQU   07    ;X * 742 millsec until new reff level set
;--------
Nt_Slot_dn  EQU   FEh   ;clr IR slot low detected

Nt_lt_reff  EQU   EFh   ;turns reff off
Nt_lght_stat      EQU   FEh   ;clears light bright status to dim status

;;; Bright & Dim equates have been moved to the light include file.

;;;Bright    EQU   05    ;light sensor trigger > reff level
;;;Dim               EQU   05    ;Light sensor trigger < reff level


;--------
;Qik_snd_reload   EQU   01    ;
;Nt_snd_reff      EQU   DFh   ;kill sound reff level bit
Nt_do_snd   EQU   FEh   ;clears sound state change req
Nt_snd_stat EQU   FBh   ;clears Sound_stat
;--------
Nt_fortune  EQU   FEh   ;kills fortune teller mode
Nt_Rap            EQU   FDh   ;kills Rap mode
Nt_hideseek EQU   FBh   ;kills Hide & seek game mode
Nt_simon    EQU   F7h   ;kills simon say game mode
;--------
Nt_do_tummy EQU   F7h   ;clears sensor change req
Nt_do_back  EQU   EFh   ;clears sensor change req
Nt_do_feed  EQU   DFh   ;clears sensor change req
Nt_do_tilt  EQU   BFh   ;clears sensor change req
Nt_do_invert      EQU   7Fh   ;clears sensor change req
Nt_do_lt_brt      EQU   FDh   ;clears sensor change req
```

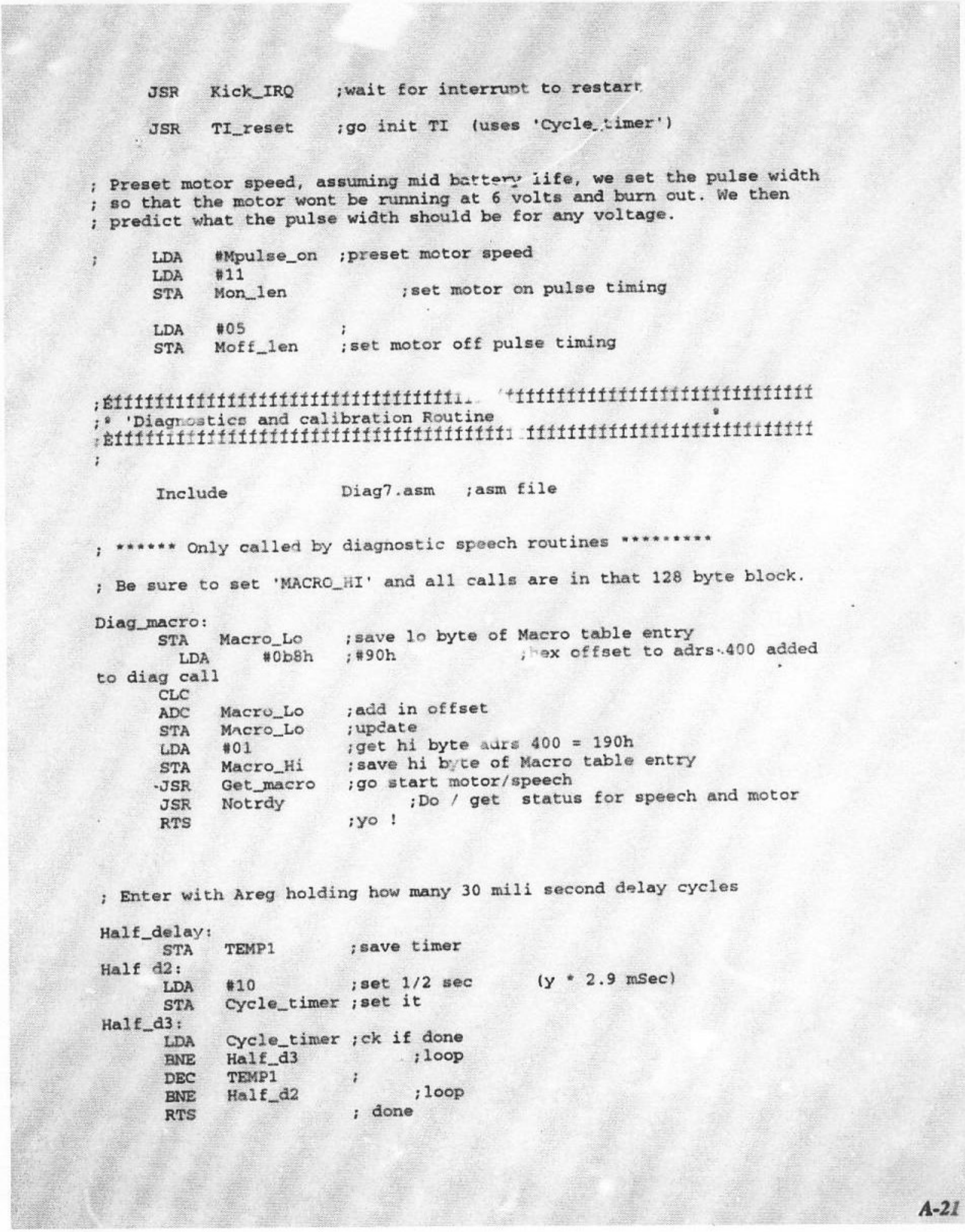
```
Nt_do_lt_dim      EQU   FBh   :clears sensor change req
;--------
Nt_temp_gaml      EQU   FEh   ;clears game mode bits
Nt_half_age EQU   BFh   ;clears req for 2 table instead of 4
Nt_randm    EQU   7Fh   ;clears random/sequential status

GameT_reload      EQU   24    ; 1= 742 mSEC ;; 255 = 189.3 seconds

;ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
;³ Variable definition      (Ram = $80 to $FF)
;ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ
;Rdef

;***** DO NOT CHANGE RAM ASSIGNMENTS (X pointer used as offsett)

;**************  The next group of RAM locations can be used by any
;            sensor routine but cannot be used to save data.
;            TEMP ONLY !
;******************** koball
TEMP0       equu  80h
TEMP1       equ   81h
TEMP2       equ   82h
TEMP3       equ   83h
TEMP4       equ   84h
IN_DAT            equ   85h
;******************** end koball
;* END TEMP RAM   **************

Task_ptr    EQU   86h   ;what function is in process
Port_A_image      EQU   87h   ;
Port_B_Image      EQU   88H   ;output port image
Port_D_Image      EQU   89H   ;output port image
;-------
Word_lo           EQU   8Ah   ;speech word lo adrs
Word_hi           EQU   8Bh   ; "        "   hi "
Saysent_lo  EQU   8CH   ;saysent word pointer
Saysent_hi  EQU   8DH   ; "        "       "
Bank_ptr    EQU   8EH   ;which bank words are in
Which_word  EQU   8FH   ;which word or saysent to call
Sgroup            EQU   90H   ;which saysent group table
Tx_data           EQU   91H   ;
;-------
Which_motor EQU   92h   ;holds table number of motor positon
Mgroup            EQU   93H   ;which motor group table
Motor_lo    EQU   94H   ;
Motptr_lo   EQU   95h   ;table pointer to get motor position
Motptr_hi   EQU   96H   ;
Which_delay EQU   97H   ;how much time between motor calls
Intt_Temp   EQU   98H   ;
Drift_fwd   EQU   99h   ;time motor reverses to stop drift
Drift_rev   EQU   9Ah   ;
Pot_timeL   EQU   9Bh   ;motor uses to compare against current positon

; moved to hi ram that is not cleared on power up
;Pot_timeL2

Moff_len    EQU   9Ch   ;holds motor power off pulse time
Mon_len           EQU   9Dh   ;holds motor power on pulse time
Motor_pulse1      EQU   9Eh   ;motor pulse timer
Slot_vote   EQU   9Fh   ;need majority cnt to declare a valid slot
```

```
Motor_led_timer     EQU   A0h   ;how long after motion done led on for IR
Mot_speed_cnt       EQU   A1h   ;motor speed test
Mot_opto_cnt        EQU   A2h   ; "
Cal_switch_cnt      EQU   A3h   ;used to eliminate noisy reads
motorstoped equ     A4h   ;times wheel count when stopping
Drift_counter       EQU   A5h   ;decides how much braking pulse to apply
;-------
Mili_sec     EQU    A6h   ;used in calc pot position by timer
Cycle_timer EQU     A7h   ;bypasses intt port c updates to motor
Sensor_timer        EQU   A8h   ;times between sensor trigger
Bored_timer EQU     A9h   ;time with no activity to random speech
;-------
Invrt_count EQU     AAh   ;which speech/motor call is next
Tilt_count  EQU     ABh   ;which speech/motor call is next
Tchfrnt_count       EQU   ACh   ;which speech/motor call is next
Tchbck_count        EQU   ADh   ;which speech/motor call is next
Feed_count  EQU     AEh   ;which speech/motor call is next
;-------
Last_IR             EQU   AFh   ;last IR sample data to compare to next
Wait_time   EQU     B0h   ;used in IRQ to create 2.8mSec timers
;-------
Light_timer EQU     B1h   ;Light sensor routines
Lght_count  EQU     B2h   ;which speech/motor call is next
Light_reff  EQU     B3h   ;holds previous sample
;---------------
Sound_timer EQU     B4h   ;time to set new reff level
Sound_count EQU     B5h   ;which speech/motor call is next
;---------------
Milisec_flag        EQU   B6h   ;set every 742 miliseconds
Macro_Lo    EQU     B7h   ;table pointer
Macro_Hi    EQU     B8h   ; "   "
Egg_cnt             EQU   B9h   ;easter egg table count pointer

;******************** Koball code rev B

HCEL_LO             EQU   BAh   ;
HCEL_HI             EQU   BBh   ;
BIT_CT              EQU   BCh   ;

;******************** end koball

Light_shift EQU     BDh   ;( was TMA_INT ) used for threshold change

;********************

Prev_random EQU     BEh   ;prevents random number twice in a row
Bored_count EQU     BFh   ;sequential selection for bored table
TEMP5       EQU     C0h   ;general use also used for wake up

Temp_ID2    EQU     C1h   ;use in sensor training routines
Temp_ID             EQU   C2h   ;use in sensor training routines
Learn_temp  EQU     C3h   ;use in sensor training routines


Req_macro_lo        EQU   C4h   ;holds last call to see if sleep or IR req
Req_macro_hi        EQU   C5h   ;

Sickr_count EQU     C6h   ;sequential counter for sick speech table
Hungr_count EQU     C7h   ;sequential counter for hunger speech table
```

```
Motor_pulse2        EQU   C8h   ;motor pulse timer


;***** DO NOT CHANGE BIT ORDER ******


Stat_0              Equ   C9h   ;System status
Want_name     EQU   01H   ;bit 0 =set forces system to say Furby's name
Lt_prev_dn    EQU   02H   ;bit 0 = done flag for quick light changes
Init_motor    EQU   04H   ;bit 1 = on wakeup do motor speed/batt test
Init_Mspeed EQU     08H   ;bit 3 = 2nd part of motor speed test
Train_Bk_prev       EQU   10H    ;bit 4 = set when 2 back sw hit in a row
Say_new_name        EQU   20H    ;bit 5 = only happens on cold boot
REQ_dark_sleep      EQU   40H    ;bit 6 = set -dark level sends to sleep
Dark_sleep_prev     EQU   80H    ;bit 7 = if set on wake up thendont
gotosleep
;
Stat_1              EQU   CAH    ;system status
Word_activ    EQU   01H   ;bit 0 = set during any speech
Say_activ     EQU   02H   ;bit 1 = when saysent is in process
Word_end      EQU   04H   ;bit 2 = set when sending FF word end to TI
Word_term     EQU   08H   ;bit 3 = set to send 3 #ffh to end speech
Up_light      EQU   10H   ;bit 4 =set when shift is incrmntg
Snd_reff      EQU   20H   ;bit 5 = set for new referrenc cycle
Half_age      EQU   40H   ;bit 6 = set for 2 tables of age instead of 4.
Randm_sel     EQU   80H   ;bit 7 =decides random/sequential for tables

Stat_2              EQU   CBH    ;system status more
Motor_actv    EQU   01H   ;bit 0 = set = motor in motion
Motor_fwd     EQU   02H   ;bit 1 = set=fwd clr=rev
Motor_seek    EQU   04H   ;bit 2 = seeking to next position
Bside_dn      EQU   08H   ;bit 3 = set = previously flaged
Binvrt_dn     EQU   10H   ;bit 4 = set- prev done
Tchft_dn      EQU   20H   ;bit 5 =  "     "
Tchbk_dn      EQU   40H   ;bit 6 =  "     "
Macro_actv    EQU   80H   ;bit 7 =set when macro in process
;
Stat_3              EQU   CCh    ;system status
Lght_stat     EQU   01H   ;bit 0 = set=bright clr = dim
Feed_dn             EQU   02H    ;bit 1 = set- prev done
Sound_stat    EQU   04H   ;bit 2 =  "     "
IRQ_dn              EQU   08H    ;bit 3 = set when IRQ occurs by IRQ
Lt_reff             EQU   10H    ;bit 4 =set for light sense reff cycle
Motor_on      EQU   20H   ;bit 5 = set=motor pulse power on
M_forward     EQU   40H   ;bit 6 = lr = move motor forward
M_reverse     EQU   80H   ;bit 7 =clr = move motor reverse
;

;*************************************************************

; Following bit maps are reserved for easter egg / games


Stat_4              EQU   CDh    ;system task request state
Do_snd              EQU   01H    ;bit 0 = set when sound > prev reff level
Do_lght_brt EQU     02H   ;bit 1 = set when light > prev reff level
Do_lght_dim EQU     04H   ;bit 2 = set when light < prev reff level
Do_tummy      EQU   08H   ;bit 3 = set when front touch triggered
Do_back             EQU   10H    ;bit 4 = set when back touch triggered
```

```
Do_feed                 EQU   20H   ;bit 5 = set when feed sensor triggered
Do_tilt                 EQU   40H   ;bit 6 = set when tilt sensor triggered
Do_invert     EQU    80H   ;bit 7 = set when inverted sensor triggered
;
Stat_5                  Equ   CEh   ;game status
temp_gam1     EQU    01H   ;bit 0 =used in game play
temp_gam2     EQU    02H   ;bit 0 = "     "        "    "
temp_gam3     EQU    04H   ;bit 1 =
temp_gam4     EQU    08H   ;bit 3 =
temp_gam5     EQU    10H   ;bit 4 =
temp_gam6     EQU    20H   ;bit 5 =
temp_gam7     EQU    40H   ;bit 6 =
temp_gam8     EQU    80H   ;bit 7 =
;
Game_1                  EQU   CFh   ;system game status
Fortune_mode            EQU   01H   ;bit 0 =set = furby in fortune teller mode
Rap_mode      EQU    02H   ;bit 0 =set = furby in RAP SONG mode
Hideseek_mode           EQU   04H   ;bit 1 =set = furby in hide & seek game
mode
Simonsay_mode           EQU   08H   ;bit 3 =set = furby in simon says game
mode
Burp_mode     EQU    10H   ;bit 4 =set = mode
Name_mode     EQU    20H   ;bit 5 =
Twinkle_mode            EQU   40H   ;bit 6 =
Rooster_mode            EQU   80H   ;bit 7 =
;
Qualify1:     EQU    D0h   ;easter egg disqualified when clear
DQ_fortune    EQU    01h   ;bit 0 = fortune teller
DQ_rap                  EQU   02h   ;bit 1 = rap song
DQ_hide                 EQU   04h   ;bit 2 = hide and seek
DQ_simon      EQU    08h   ;bit 3 = simon says
DQ_burp                 EQU   10h   ;bit 4 = burp attack
DQ_name                 EQU   20h   ;bit 5 = says his name
DQ_twinkle    EQU    40h   ;bit 6 = sings song
DQ_rooster    EQU    80h   ;bit 7 = rooster loves you
;

; ********** THIS GROUP OF RAM IS SAVED IN EEPROM


; Need to read these from EEPROM and do test for false data

; "age" uses bit 7 to extend the "age_counter" to 9 bits, and this
; is saved in EERPOm also.

;"AGE" MUST BE IN D1h BECAUSE EEPROM READ & WRITE USE THE EQU FOR START
RAM.

Age           EQU    D1h   ;age = 0-3 (4 total)
Age_counter EQU      D2h   ;inc on motor action,rolls over & inc age

Name          EQU    D3h   ;holds 1-6 pointer to firby's name
Rvoice                  EQU   D4h   ;which one of three voices
Pot_timeL2  EQU      D5h   ;counter from wheel I.R. sensor
Hungry_counter          EQU   D6h   ;holds hungry/full counter
Sick_counter            EQU   D7h   ;healthy/sick counter
Seed_1                  EQU   D8h   ;only seed 1 & seed 2 are saved
Seed_2                  EQU   D9h   ; "       "        "

; These are used for training each sensor. There is a word number which
; is 1-16 for the sesnor table macro list and a ram for count which
; determines how often to call the learned word.

; *** DO NOT CHANGE ORDER----- RAM adrs by Xreg offset

Tilt_learned      EQU   DAh   ;which word trained                  1
Tilt_lrn_cnt      EQU   DBh   ;count determines how often called   2

Feed_learned      EQU   DCh   ;which word trained                  3
Feed_lrn_cnt      EQU   DDh   ;count determines how often called   4

Light_learned     EQU   DEh   ;which word trained                  5
Light_lrn_cnt     EQU   DFh   ;count determines how often called   6

Dark_learned      EQU   E0h   ;which word trained                  7
Dark_lrn_cnt      EQU   E1h   ;count determines how often called   8

Front_learned     EQU   E2h   ;which word trained                  9
Front_lrn_cnt     EQU   E3h   ;count determines how often called   10

Sound_learned     EQU   E4h   ;which word trained                  11
Sound_lrn_cnt     EQU   E5h   ;count determines how often called   12

Wake_learned      EQU   E6h   ;which word trained                  13
Wake_lrn_cnt      EQU   E7h   ;count determines how often called   14

Invert_learned    EQU   E8h   ;which word trained                  15
Invert_lrn_cnt    EQU   E9h   ;count determines how often called   16

; next is equates defining which ram to use for each sensor
; according to the sensor ram defined above. (compare to numbers above)

Tilt_ID               EQU   00    ;defines what offset for above ram
definitions
Feed_ID               EQU   02    ; "
Light_ID     EQU      04    ; "
Dark_ID               EQU   06    ; "
Front_ID     EQU      08    ; "
Sound_ID     EQU      10    ; "
Wake_ID               EQU   12    ; "
Invert_ID    EQU      14    ; "
Back_ID               EQU   EEh   ;special value triggers learn mode

;**********************************************************************
*
; For power on test, WE only clear ram to E9h and use EAh for a
; messenger to the warm boot routine. We always clear ram and initialize
; registers on power up, but if it is a warm boot then read EEPROM
; and setup ram locations. Location EAH is set or cleared during power
up
; and then the stack can use it during normal run.


Warm_cold    EQU   EDh   ;
Spcl_seed1   EQU   EEh   ;
Spcl_seed2   EQU   EFh   ;
Deep_sleep   EQU   F0h   ;0=no deep sleep 11h is. (tilt wont wakeup)


;********** Need to allow stack growth down  ( EAh- FFH ) *********
```

```
Stacktop    EQU   FFH   ;Stack Top


;*******************************************************************
****
;*******************************************************************
****
;*******************************************************************
****
;*******************************************************************
****

        ORG      00H
        BLKW     300H,00H          ;Fill 0000 ÄÄÄ 05FFH= 00

;ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿
;³                                                    ³
;³      P R O G R A M    S T A R T S   H E R E       ³
;³                                                    ³
;ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ


        ORG     0600H

RESET:


      Include            Wake2.asm   ;asm file



;********* end Tracker



; For power on test, WE only clear ram to E9h and use EAh for a
; messenger to the warm boot routine. We always clear ram and initialize
; registers on power up, but if it is a warm boot then read E²PROM
; and setup ram locations. Location EAH is set or cleared during power
up
; and then the stack can use it during normal run.


; Clear RAM to 00H
; -------------------------------------------------------------------
----
        LDA      #00H              ; data for fill
        LDX      #E9H             ; start at ram location

RAMClear:
        STA      00,X              ; base 00, offset x
        DEX                        ; next ram location
        CPX      #7FH             ; check for end
        BNE      RAMClear          ; branch, not finished
                                   ; fill done
; ----------------------------------  ------------------------------
----


Main:

InitIO:
        LDA    #01          ;turn DAC on
        STA    DAC_ctrl     ;DAC control

        LDA    #Port_def    ;set direction control
        STA    Ports_dir    ;load reg

        LDA    #Con_def     ;set configuration
        STA    Ports_con    ;load reg

        LDA    #00          ;set for bank 0
        STA    Bank         ;set it
        LDA    #00H         ;disable wakeup control
        STA    Wake_up             ;
        LDA    #00h         ;disable sleep control
        STA    Sleep        ;set dont care

          LDA       #Intt_dflt     ;Initialize timers, etc.
          STA       Interrupts     ;load reg

          LDA       #00H           ;set timer mode
          STA       TMA_CON        ;set reg
        LDA    #TimeA_low   ;get preset timer for interrupts
        STA    TMA_LSB             ;load

        LDA    #TimeA_hi    ;get hi byte for preset
        STA    TMA_MSB             ;load it

        LDA    #TimeB_low   ;get preset timer for interrupts
        STA    TMB_LSB             ;load
        LDA    #TimeB_hi    ;get hi byte for preset
        STA    TMB_MSB             ;load it

        LDA    #C0h         ;preset status for motors off
        STA    Stat_3              ;

        LDA    #00H         ;init ports
        STA    Port_A              ;output

        LDA    #33H         ;init ports
        STA    Port_B_Image        ;ram image
        STA    Port_B              ;output

        LDA    #0FH         ;init ports
        STA    Port_C              ;output

        LDA    #D0H         ;init ports
        STA    Port_D_Image        ;ram image
        STA    Port_D              ;output

        LDA    #FFh         ;milisec timer reload value
        STA    Mili_sec     ;also preset IRQ timer

          CLI               ;Enable IRQ
```



```
     JSR   Kick_IRQ    ;wait for interrupt to restart

     JSR   TI_reset    ;go init TI  (uses 'Cycle_timer')

; Preset motor speed, assuming mid battery life, we set the pulse width
; so that the motor wont be running at 6 volts and burn out. We then
; predict what the pulse width should be for any voltage.

;     LDA   #Mpulse_on  ;preset motor speed
      LDA   #11
      STA   Mon_len              ;set motor on pulse timing

      LDA   #05         ;
      STA   Moff_len    ;set motor off pulse timing

;Éfffffffffffffffffffffffffffffffffi.   '*ffffffffffffffffffffffffffffff
;* 'Diagnostics and calibration Routine                    *
;Èffffffffffffffffffffffffffffffffffffi ffffffffffffffffffffffffffffff
;

      Include            Diag7.asm   ;asm file

; ****** Only called by diagnostic speech routines *********

; Be sure to set 'MACRO_HI' and all calls are in that 128 byte block.

Diag_macro:
      STA   Macro_Lo    ;save lo byte of Macro table entry
        LDA     #0b8h   ;#90h              ;hex offset to adrs 400 added
to diag call
      CLC
      ADC   Macro_Lo    ;add in offset
      STA   Macro_Lo    ;update
      LDA   #01         ;get hi byte adrs 400 = 190h
      STA   Macro_Hi    ;save hi byte of Macro table entry
     -JSR   Get_macro   ;go start motor/speech
      JSR   Notrdy            ;Do / get  status for speech and motor
      RTS               ;yo !


; Enter with Areg holding how many 30 mili second delay cycles

Half_delay:
      STA   TEMP1       ;save timer
Half d2:
      LDA   #10         ;set 1/2 sec       (y * 2.9 mSec)
      STA   Cycle_timer ;set it
Half_d3:
      LDA   Cycle_timer ;ck if done
      BNE   Half_d3           ;loop
      DEC   TEMP1       ;
      BNE   Half_d2           ;loop
      RTS               ; done
```

```
Test_byp:   ;We assume diagnostic only runs on coldboot

;*************************************************************

      LDA   #FFh           ;initialize word training variable
      STA   Temp_ID                ;

      LDA   #FFh           ;
      STA   Hungry_counter    ;preset furby's health
      STA   Sick_counter

;*********************************************************

; We sit here and wait for tilt to go away, and just keep incrementing
; counter until it does. This becomes the new random generator seed.

Init_rnd:
      INC   TEMP1          ;random counter
      LDA   Port_D                ;get switches
      AND   #03            ;check tilt & invert sw
      BNE   Init_rnd       ;loop til gone
      LDA   TEMP1          ;get new seed
      STA   Spcl_seed1     ;stuff it
      STA   Seed_1                ;also load for cold boot

;***********************************************************

; Use feed sw to generate a better random number

      JSR   Get_feed       ;go test sensor
      LDA   Stat_4                ;get system
      AND   #Do_feed       ;ck sw
      BNE   Feed_rnd       ;if feed sw then cold boot
      JMP   End_coldinit          ;else do warm boot
Feed_rnd:
      INC   TEMP1          ;random counter
      LDA   Stat_4                ;system
      AND   #DFh           ;clear any prev feed sw senses
      STA   Stat_4                ;update
      JSR   Get_feed       ;go test sensor
      LDA   Stat_4                ;get system
      AND   #Do_feed       ;ck sw
      BNE   Feed_rnd       ;wait for feed to go away
      LDA   TEMP1          ;get new seed
      STA   Spcl_seed1     ;stuff it
      STA   Seed_1                ;also load for cold boot


;*********************************************************

;; IF this is a cold boot , reset command then clear EEPROM and
;  chose a new name and voice.

Do_cold_boot:

      LDA   #00
      STA   Warm_cold      ;flag cold boot
      LDA   Stat_0            ;system
      ORA   #Say_new_name     ;make system say new name
      STA   Stat_0            ;

;*******  NOTE :::::
;
;   VOICE AND NAME SLECTION MUST HAPPEN BEFORE EEPROM WRITE OR
;   THEY WILL ALWAYS COME UP  00    because ram just got cleared!!!!!!


; Random voice selection here

      LDA   #80h          ;get random/sequential split
      STA   IN_DAT                ;save for random routine

      LDX   #00           ;make sure only gives random
      LDA   #10h          ;get number of random selections
      JSR   Ran_seq               ;go get random selection

      TAX
      LDA   Voice_table,X     ;get new voice
      STA   Rvoice            ;set new voice pitch


;**********************************************************************
*


; On power up or reset, Furby must go select a new name ,,, ahw how
cute.

      JSR   Random            ;
      AND   #1Fh          ;get 32 possible
      STA   Name          ;set new name pointer
      JSR   Do_EE_write ;write the EEPROM

End_coldinit:



;ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
;º 'Special initialization prior to normal run mode                  º
;º  Jump to Warm_boot when portD wakes us up
;ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
;

Warm_boot:  ;normal start when Port_D wakes us up.

      JSR   S_EEPROM_READ     ;read data to ram

;Eprom_read_byp:

;****************************************************************
; If light osc fails, or too dark and that sends us to sleep, we
; set 'Dark_sleep_prev' and save it in EEPROM in 'Seed_2'.
; when the sleep routine executes,(00 01 based on this bit)
; When we wake up we recover this bit and it becomes the previous done
; flag back in 'Stat_0', so that if the osc is
; still dark or failed, Furby wont go back to sleep.

      LDA   Seed_2            ;from EEPROM
      BEQ   No_prevsleep      ;jump if none
      LDA   Stat_0            ;system
      ORA   #Dark_sleep_prev ;prev done
      STA   Stat_0            ;update

No_prevsleep:

;*****************************************************************


      LDA   Spcl_seed1  ;recover start up random number
      STA   Seed_1            ;set generator

;*****************************************************************


; Pot_timeL2 is save in ram through sleep mode and then reloaded to
; Pot_timeL which is the working register for the motor position.
; This allows startup routines to clear ram without forgetting the
; last motor position.

      LDA   Pot_timeL2  ;get current count
      STA   Pot_timeL   ;save in motor routine counter

;*************************

; Get age and make sure it is not greater than 3 (age4)

      LDA   Age         ;get current age
      AND   #83h     ;preserve bit 7 which is 9th age counter bit
;;;;;            and insure age not >3

      STA   Age         ;set system

;*************************

      LDA   #Bored_reld ;reset timer
      STA   Bored_timer ;


      LDA   #03         ;set timer
      STA   Last_IR           ;timer stops IR from hearing own IR xmit

      JSR   Get_light   ;go get light level sample
      LDA   TEMP1       ;get new count
      STA   Light_reff  ;update system


;*****************************************************************************
*

      LDA   Warm_cold   ;decide if warm or cold boot
      CMP   #11h        ;ck for warm boot
      BEQ   No_zero           ;jump if is
      LDA    #00         ;point to macro 0 (SENDS TO SLEEP POSITION)
      STA    Macro_Lo
      STA    Macro_Hi
      JSR    Get_macro   ;go start motor/speech
      JSR    Notrdy            ;Do / get  status for speech and motor

No_zero:


      LDA    #11         ;preset motor speed
      STA    Mon_len           ;set motor on pulse timing

      LDA    #05         ;set motor to 3/4 speed for speed test
      STA    Moff_len    ;set motor off pulse timing
;
;
      LDA    #00         ;clear all system sensor requests
      STA    Stat_4            ;update


;  Currently uses 4 tables, one for each age.

      LDA    Stat_0            ;system
      ORA    #Init_motor ;flag motor to do speed test
      ORA    #Init_Mspeed      ;2nd part of test
      STA    Stat_0            ;update



;************************************************************

; Do wake up routine :

      lda    #Global_time      ;reset timer to trigger sensor learning
      STA    Sensor_timer      ;

      LDA    #80h        ;get random/sequential split
      STA    IN_DAT            ;save for random routine

      LDX    #00h        ;make sure only gives random
      LDA    #10h        ;get number of random selections
      JSR    Ran_seq           ;go get random selection
      LDA    TEMP1       ;get decision

      STA    IN_DAT            ;save decision
      LDA    #Wake_ID    ;which ram location for learned word count
(offset)
      JSR    Start_learn ;go record training info
      LDA    IN_DAT            ;get back word to speak

      JSR    Decid_age   ;do age calculation for table entry
      LDX    TEMP0       ;age offset
      LDA    Wakeup_S1,X ;get new sound/word
      STA    Macro_Lo    ;save lo byte of Macro table entry
      INX                ;
      LDA    Wakeup_S1,X ;get new sound/word
      STA    Macro_Hi    ;save hi byte of Macro table entry
      JMP    Start_macro ;go start speech

;************************************************************
```

```
;ÉÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
;º 'IDLE Routine                                      º
;ÈÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍÍ
;

Idle:

; Idle routine is the time slice task master  (TSTM) ugh!
; We must call each routine and interleave with a call to speech
; to insure we never miss a TI request for data.

      JSR   Notrdy             ;Do / get  status for speech and motor

;***********************************************************
; THis bit is set when light sensor is darker than 'Dark_sleep'

      LDA   Stat_0             ;system
      AND   #REQ_dark_sleep    ;ck for req
      BEQ   No_dark_req ;jump if not

      LDA   Stat_0             ;system
      AND   #BFh         ;kill req
      STA   Stat_0             ;update

      LDA   #A6h         ;sleep macro
      STA   Macro_Lo
      LDA   #00h         ;sleep macro
      STA   Macro_Hi     ;
      JMP   Start_macro ;go say it


No_dark_req:

;***********************************************************

; When any sensor or timer calls the "start_macro" routine, the
; Macro_Lo & Macro_Hi are saved. Everyone jumps back to Idle and when
; speech/motor routines are finished, this routine will look at the
; macros that were used and execute another function if a match is
found.
;
; Checks for his name first, then any IR to send, and finally, the sleep
; commands. THe temp macro buffers are cleared before


;
Spcl_Name1:
      LDX   #00          ;offset
Spcl_Name2:
      LDA   Ck_Name_table,X   ;ck lo byte
      CMP   #FFh         ;ck for end of table (note 255 cant execute)
      BEQ   Spcl_IR1    ;done if is
      CMP   Req_macro_lo        ;ck against last speech request
      BNE   Not_Name2    ;jump if not
      INX                ;to hi byte
      LDA   Ck_Name_table,X   ;ck hi byte
      CMP   Req_macro_hi        ;ck against last speech request
```

```
        BNE     Not_Name3    ;jump if not
        JMP     Say_Sname    ;speak it
Not_Name2:
        INX                  ;
Not_Name3:
        INX                  ;
        JMP     Spcl_Name2   ;loop til done

Say_Sname:
        LDA     Stat_0
        AND     #DFh         ;kill req for startup new name
        STA     Stat_0              ;update

        LDA     Name         ;current setting for table offset
        CLC
        ROL     A            ;2's comp
        TAX
        LDA     Name_table,X        ;get lo byte
        STA     Macro_Lo     ;save lo byte of Macro table entry
        INX                  ;
        LDA     Name_table,X        ;get hi byte
        STA     Macro_Hi     ;save hi byte of Macro table entry
        JSR     Get_macro    ;go start motor/speech
        JSR     Notrdy              ;Do / get  status for speech and motor
;
Spcl_IR1:
        LDX     #00          ;offset
Spcl_IR2:
        LDA     IRxmit_table,X    ;ck lo byte
        CMP     #FFh         ;ck for end of table (note 255 cant execute)
        BEQ     Spcl_IR_dn  ;done if is
        CMP     Req_macro_lo      ;ck against last speech request
        BNE     Not_IRxmit2 ;jump if not
        INX                  ;to hi byte
        LDA     IRxmit_table,X    ;ck hi byte
        CMP     Req_macro_hi      ;ck against last speech request
        BNE     Not_IRxmit3 ;jump if not
        INX                  ;point to IR table
        LDA     IRxmit_table,X    ;
        STA     TEMP2        ;xmit temp ram
        LDA     #FDh         ;TI command for IR xmit
        STA     TEMP1        ;
        JSR     Xmit_TI             ;go send it

        LDA     #Bored_reld ;reset bored timer
        STA     Bored_timer ;

        LDA     #03          ;set timer
        STA     Last_IR             ;timer stops IR from hearing its own IR
xmit

        JMP     Spcl_IR_dn  ;done - ola ......
Not_IRxmit2:
        INX                  ;lo byte
Not_IRxmit3:
        INX                  ;hi byte
        INX                  ;xmit pointer
        JMP     Spcl_IR2     ;loop til done
Spcl_IR_dn:
;
```

```
;
Spcl_macro1:
      LDX    #00           ;offset
Spcl_sleep1:
      LDA    Sleepy_table,X    ;ck lo byte
      CMP    #FFh          ;ck for end of table (note 255 cant execute)
      BEQ    Ck_macro_dn ;done if is
      CMP    Req_macro_lo      ;ck against last speech request
      BNE    Not_sleepy2 ;jump if not
      INX                  ;to hi byte
      LDA    Sleepy_table,X    ;ck hi byte
      CMP    Req_macro_hi      ;ck against last speech request
      BNE    Not_sleepy3 ;jump if not
      LDA    #00           ;clear macro pointers for wake up
      STA    Req_macro_lo
      STA    Req_macro_hi

;mod F-rels2 ;
;   Before going to sleep send sleep cmnd to all others.

      LDA    #15           ;
      STA    TEMP2         ;xmit temp ram
      LDA    #FDh          ;TI command for IR xmit
      STA    TEMP1         ;
      JSR    Xmit_TI             ;go send it

;need to wait >600 milisec before going to sleep because we arent using
;busy flags from TI and need to make sure it is done transmitting the
;I.R. code, the sleep routine kills the TI and it would never send the
cmnd.

      LDA    #25           ;how many 30 milisec cycles to call
      JSR    Half_delay  ;do 30milisec delay cycles

;end mod

      JMP    GoToSleep     ;nity-night

Not_sleepy2:
      INX                  ;
Not_sleepy3:
      INX                  ;
      JMP    Spcl_sleep1 ;loop til done
;
Ck_macro_dn:
      LDA    #00           ;clear macro pointers for wake up
      STA    Req_macro_lo
      STA    Req_macro_hi
      JMP    Test_new_name       ;on to task master
;

;;;;;;; SLEEP TABLE & IR table ..... MOVE TO INCLUDE FILE LATER

Sleepy_table:
      DW     91     ;hangout

      DW     166    ;wake up
      DW     167    ;wake up
      DW     168    ;wake up
      DW     169    ;wake up
```

```
        DW      258     ;Back sw
        DW      259     ;Back sw
        DW      260     ;Back sw

        DW      403     ;IR
        DW      413     ;IR
        DW      429     ;IR

        DB      FFh,FFh      ;FF FF   is table terminator

IRxmit_table:
        DW      [illegible]     ;trigger macro
        DB      00      ;which IR command to call ( 0 - 0f )
        DW      13      ;trigger macro
        DB      00      ;which IR command to call ( 0 - 0f )
        DW      17      ;trigger macro
        DB      00      ;which IR command to call ( 0 - 0f )
        DW      19      ;trigger macro
        DB      00      ;which IR command to call ( 0 - 0f )
        DW      26      ;trigger macro
        DB      00      ;which IR command to call ( 0 - 0f )
        DW      29      ;trigger macro
        DB      00      ;which IR command to call ( 0 - 0f )
        DW      33      ;trigger macro
        DB      00      ;which IR command to call ( 0 - 0f )
        DW      34      ;trigger macro
        DB      00      ;which IR command to call ( 0 - 0f )
        DW      44      ;trigger macro
        DB      00      ;which IR command to call ( 0 - 0f )
        DW      45      ;trigger macro
        DB      00      ;which IR command to call ( 0 - 0f )
        DW      48      ;trigger macro
        DB      00      ;which IR command to call ( 0 - 0f )
        DW      50      ;trigger macro
        DB      00      ;which IR command to call ( 0 - 0f )
        DW      55      ;trigger macro
        DB      00      ;which IR command to call ( 0 - 0f )
        DW      60      ;trigger macro
        DB      00      ;which IR command to call ( 0 - 0f )
        DW      149     ;from rooster wake up
        DB      00      ;

        DW      352     ;trigger macro
        DB      01      ;which IR command to call ( 0 - 0f )
        DW      363     ;trigger macro
        DB      01      ;which IR command to call ( 0 - 0f )
        DW      393     ;trigger macro
        DB      01      ;which IR command to call ( 0 - 0f )

        DW      248     ;trigger macro
        DB      02      ;which IR command to call ( 0 - 0f )
        DW      313     ;trigger macro
        DB      02      ;which IR command to call ( 0 - 0f )

        DW      86      ;trigger macro
        DB      03      ;which IR command to call ( 0 - 0f )
        DW      93      ;trigger macro
        DB      03      ;which IR command to call ( 0 - 0f )
        DW      339     ;trigger macro
```