```
\ misc8051.fs

0 [if]
```

```
[then]

nowarn

: hello ." Talk to the target " ;
' hello is bootmessage

variable talks 0 talks !
: talking  true talks ! ;

\ ----- Virtual Machine ----- /
\ Subroutine threaded.
  0 constant S \ R0 = Stack pointer.
  1 constant A \ R1 = Internal address pointer.
$e0 constant T : .T T + ; \ Acc = Top of stack.
\ DPTR = Code memory address pointer, aka P.
\ B is used by um*, u/mod, and over, not preserved.

\ ----- 8051 Registers ----- /
$82 constant DPL $83 constant DPH
$98 constant SCON : .SCON SCON + ;
$99 constant SBUF
$80 constant P0 : .P0 P0 + ;
$90 constant P1 : .P1 P1 + ;
$a0 constant P2 : .P2 P2 + ;
$b0 constant P3 : .P3 P3 + ;
$81 constant SP
$d0 constant PSW : .PSW PSW + ;
$88 constant TCON : .TCON TCON + ;
$89 constant TMOD
$8a constant TL0 $8b constant TL1
$8c constant TH0 $8d constant TH1
$8f constant PCON
$a8 constant IE : .IE IE + ;
$b8 constant IP : .IP IP + ;
$f0 constant B  : .B  B + ;
\ $fd constant SP0 $80 constant RP0
$100 constant SP0 $80 constant RP0

\ ----- Subroutines ----- /
\ : clean  begin key?-s while key-s drop repeat ;
: listen  begin  key-s dup 7 - while  emit  repeat  drop ;
: (talk)  ( a - ) ( clean) 0 emit-s key-s
   drop dup $ff and emit-s 8 rshift $ff and emit-s ;
\ Enabling the '[char] | emit' tags results coming from target.
\ Words executed only for the host won't do that.  A debugging aid.
: talk  ( a - ) >red ( [char] | emit) (talk) listen >black ;
```

```
:m call  ( a - )
        hint
        [ dup $f800 and ] here [ 2 + $f800 and = if
                dup 8 rshift 32 * $11 + ] , , [ exit
        then $12 ] , [ dup 8 rshift ] , , m;

:m -:  (  - )
        [ >in @ label >in !
        create ] here [ , hide
        does> @ talks @ if  talk exit  then ] call m;

:m :  (  - ) -: header m;

:m ;a  (  - )
        edge c@-t $1f and $11 = if
                ] here [ 2 - dup c@-t $ef and swap c!-t exit
        then ] $22 , m;

:m ;l  (  - )
        edge c@-t $12 = if
                $02 ] here [ 3 - c!-t exit
        then ] $22 , m;

:m ;  (  - )
        edge here [ 2 - = if ;a exit then ]
        edge here [ 3 - = if ;l exit then ]
        $22 , m;

\ ----- Assembler ----- /
[ \ These are 'assembler', not 'target forth'.
: interrupt ( a - ) ] here swap org dup call ; org [ ;
: push $c0 ] , , [ ;   : pop $d0 ] , , [ ;
: set $d2 ] , , [ ;    : clr $c2 ] , , [ ;  \ bit
: setc $d3 ] , [ ;     : clrc $c3 ] , [ ;   \ carry
: toggle $b2 ] , , [ ; : reti $32 ] , [ ;
: nop 0 ] , [ ;
: inc  dup 8 < if $08 + ] , [ exit then $05 ] , , [ ;  \ Rn or direct
: dec  dup 8 < if $18 + ] , [ exit then $15 ] , , [ ;
: add  dup 8 < if $28 + ] , [ exit then $25 ] , , [ ;
: addc dup 8 < if $38 + ] , [ exit then $35 ] , , [ ;
: xch  dup 8 < if $c8 + ] , [ exit then $c5 ] , , [ ;
: ##p! $90 ] , [ dup 8 rshift ] , , [ ;
: mov   dup 8 < if $a8 + ] , , [ exit then
        over 8 < if swap $88 + ] , , [ exit then
        $85 ] , [ swap ] , , [ ;
: movbc $a2 ] , , [ ;  \ Move bit to carry.
: movcb $92 ] , , [ ;  \ Move carry to bit.
: [swap] $c4 ] , [ ;   \ Swap nibbles.

\ ----- Conditionals ----- /
:m then hide here [ over - 1 - swap ] c!-t m;
:m cond hide , here 0 , m;
:m if   $60 cond m;  :m 0=if $70 cond m;
:m if' $50 cond m;   :m 0=if' $40 cond m;
:m if. $30 , cond m; :m 0=if. $20 , cond m;
:m -if 7 .T if. m;   :m +if 7 .T 0=if. m;
:m begin here hide m;
:m end [ dup >r 1 + - r> c!-t ] hide m;
:m until if end m;
:m 0=until 0=if end m;
:m until. if. end m;
:m 0=until. 0=if. end m;
:m -until  -if end m;
:m again call ; m;

\ ----- Stack operations ----- /
:m nip [ S inc ] m;
:m drop hint $e6 , nip m;
:m dup S dec $f6 , m;
```

```
:m swap $c6 , m;
:m (over) $86 , B , dup $e5 , B , m;
:m 2drop nip drop m;

\ ----- Optimizing ----- /
\ The hint helps #, doesn't hurt anything else?
:m ?dup  (  - ?)
        edge here [ 2 - - if ] hint dup [ exit then
        edge @-t $e608 = if
                -2 ] allot here [ there 2 erase exit
        then ] hint dup m;

:m ?lit  (  - ?)
        edge here [ 4 - - if 0 exit then
        edge @-t $18f6 = ] edge [ 2 + c@-t $74 = and if
                ] here [ 1 - c@-t -4 ] allot here
                [ there 4 erase -1 exit
        then 0 ] m;

:m =if ?lit [ 0= if abort then ] $b4 , cond m; \ Does literal = T?.
:m <if =if then if' m; \ Is T <= literal?.
:m =until  =if end m;
:m <until  <if end m;

\ ----- More stack operations ----- /
:m # ?dup $74 , , m;      :m ## [ dup ] # [ 8 rshift ] # m;
:m ~# [ invert ] # m;
:m push [ T push ] drop m; :m pop ?dup [ T pop ] m;
\ $75 = mov direct,#data
:m SP! $75 , S , , m;     :m RP! $75 , SP , , m;
:m stacks SP0 SP! RP0 RP! m;

\ ----- Arithmetic and logic ----- /
:m 1+ $04 , m;     :m 1- $14 , m;
:m u1+ $06 , m;    :m u1- $16 , m;
:m invert $f4 , m; :m negate invert 1+ m;

:m logic ( opcode) [ >r ] ?lit [ if r> ] , , exit [ then r> ] 2 + , nip m;
:m +   $24 logic m;
:m +'  $34 logic m;
:m ior $44 logic m;
:m and $54 logic m;
:m xor $64 logic m;

\ Don't use # after the SFR, a special case.
:m logic! ( opcode) [ >r ] ?lit [ if ]
   [ r> ] , [ swap ] , , [ exit then r> 1 - ] , , drop m;
:m ior! $43 logic! m;
:m and! $53 logic! m;
:m xor! $63 logic! m;

:m |u/mod swap $86 , B , $84 , $a6 , B , m;
:m |um* $86 , B , $a4 , swap $e5 , B , m;
:m |* ?lit [ if ] $75 , B , , $a4 , [ exit then ]
   $86 , B , nip $a4 , m;
:m 2*' $33 , m;  :m 2* clrc 2*' m;
:m 2/' $13 , m;  :m 2/ [ 7 .T movbc ] 2/' m;

\ ----- Memory access ----- /
:m (#!)  [ dup 8 < if $f8 + ] , [ exit then ] $f5 , , m; \ No drop.
:m #!  ?lit [ if
                over 8 < if swap $78 + ] , , [ exit then ]
                $75 , [ swap ] , , [ exit
        then ]  (#!) drop m;

:m (#@) [ dup 8 < if $e8 + ] , [ exit then ] $e5 , , m;  \ No dup.
:m #@ ?dup (#@) m;

:m a ?dup $e9 , m;
```

```
\ Use of A is not reentrant, push and pop where needed.
:m a! ?lit [ if ] $79 , , exit [ then ] $f9 , drop m;
:m @ ?dup $e7 , m;
:m @+ @ $09 , m;
:m ! $f7 , drop m;
:m !+ ! $09 , m;

:m #for  ( direct - ) #! begin m;
:m #next  ( direct - ) [ dup 8 < if ] $d8 or cond end exit [ then ]
        $d5 , cond end m;

:m |p ?dup $e5 , DPL , dup $e5 , DPH , m;
:m |@p dup $e4 , $93 , m;
:m p! $f5 , DPH , drop $f5 , DPL , drop m;
:m p+ $a3 , m;
:m |@p+ |@p p+ m;
:m (!x) $f0 , m;
:m !x (!x) drop m;
:m !x+ !x p+ m;
:m @x ?dup $e0 , m;
:m @x+ @x p+ m;

0 org : reset

:m see ' >body [ @ ] decode m;
```