

Introduction To MD-1

The MD-1 Board uses an Allegro UCN5804B which has a translator and driver built in a single 16pin package. It provides complete control and drive for a four-phase unipolar stepper motor with continuous output current ratings to 1.25A per phase and 35V. The board has +5V regulator and the speed control potentiometer, which can be adjusted from 50 steps/sec to 200 steps/sec.
Please note:

Improper connection of the stepper motor to the board may damage it. See Fig 1 for correct connection, make sure to turn off the power before plug or unplug the motor from the board.

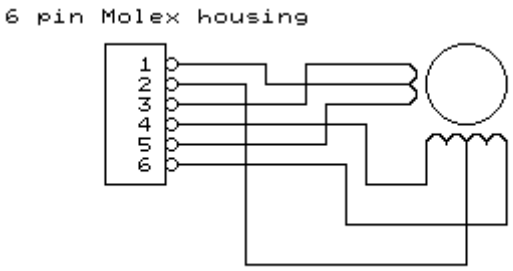
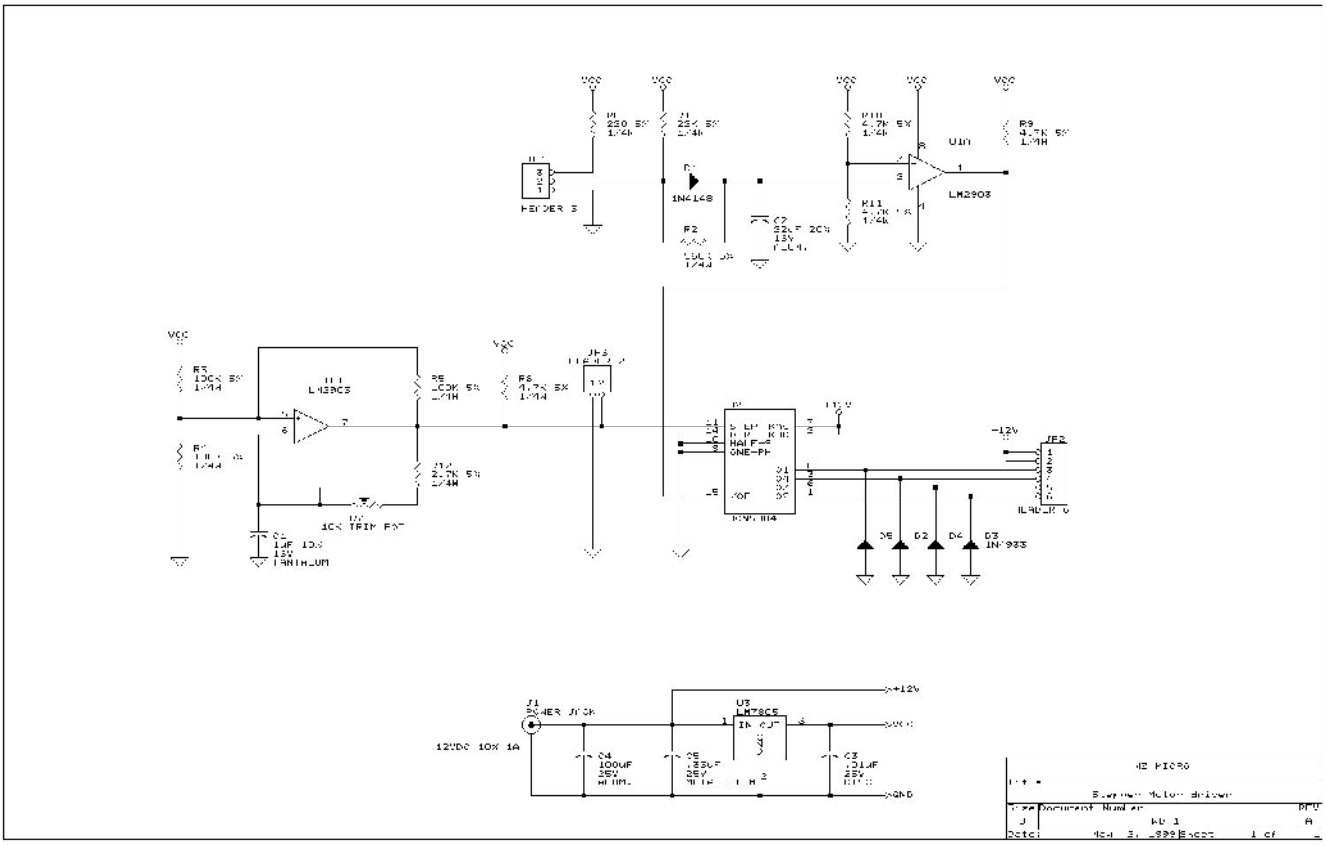
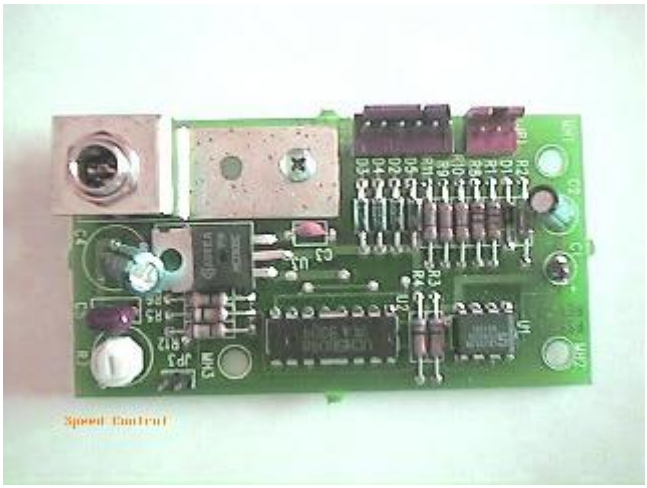


Fig.1 Stepper motor connection



.



The board can be powered with +24VDC. The JP1 is for the photo sensor switch. If nothing is connected to it then as soon as the power is applied to the board, the motor will start running, If pin1 and 2 of the JP1 is shorted then the motor will stop after a few seconds. To make the motor stop immediately you need to remove C2.

If you have any technical questions, please feel free to drop us an email. All additional or replacement parts can be ordered from us.

Introduction To MD-2

The MD-2 Board is a very basic design of 8031 microcontroller with dual motor drivers and five input ports (see schematic). The board requires +5V and +24V to power (or you may use any thing from +12V to +32V). The U2 and U3 are Allegro UDN2543 protected quad power drivers, which are use for driving the unipolar stepper motor. Each of the four outputs can sink up to 700mA in the ON state; the peak current is rated at 1A per channel. The J2 and J3 are Stepper motor connectors, improper motor connection may damage the board, make sure to turn off the power before plug or unplug the motor from the board.

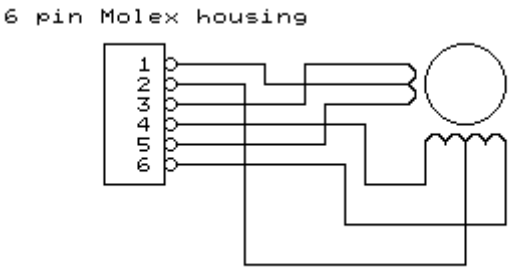
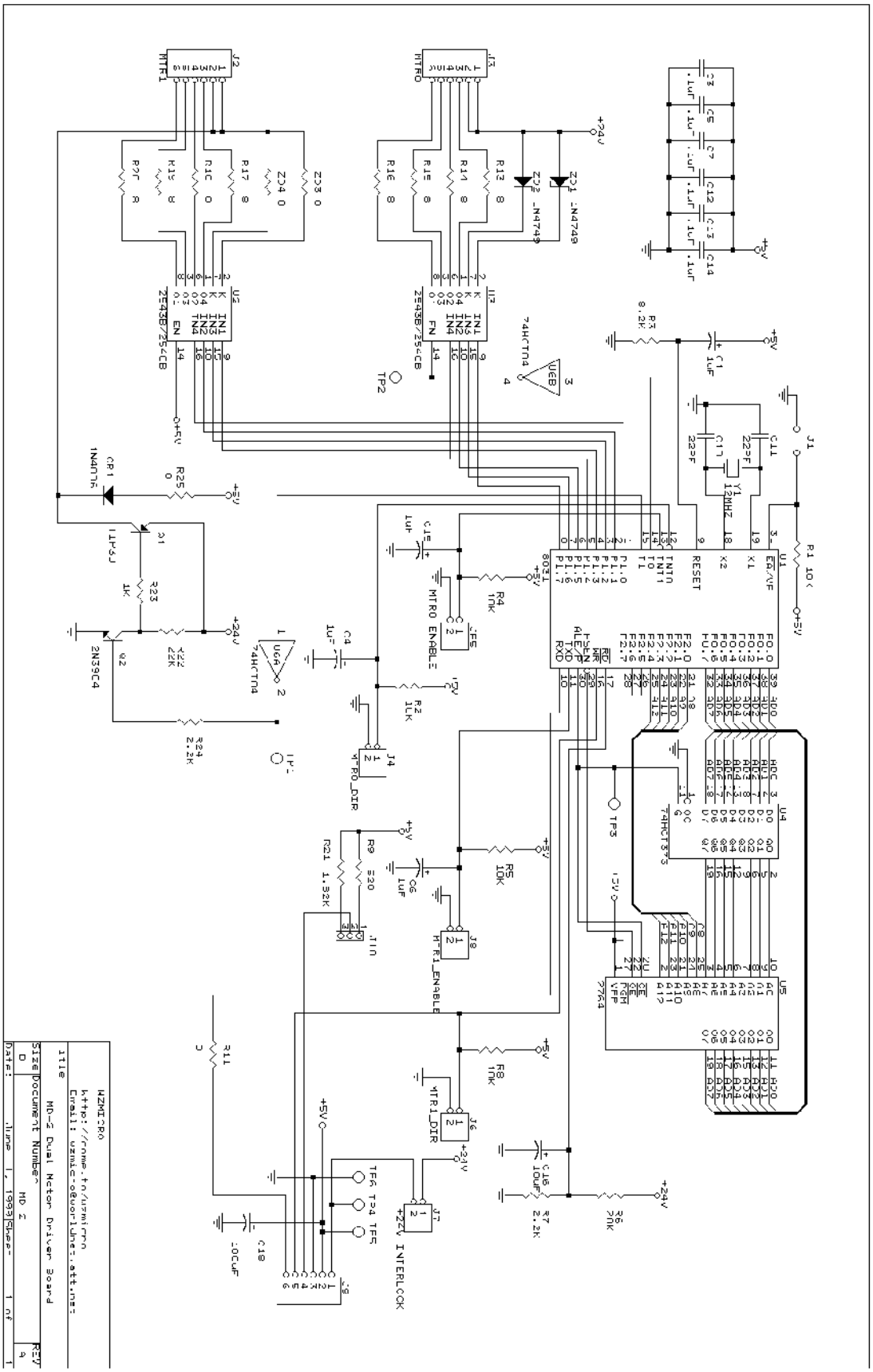
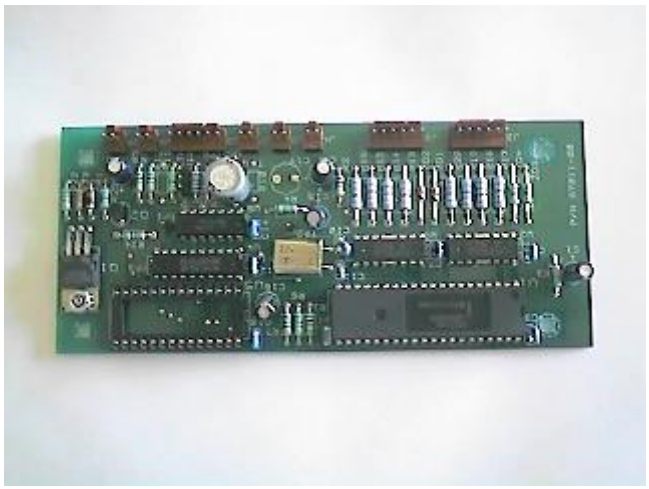


Fig.1 Stepper motor connection

The connector J4, J5, J6 and J8 are configured as input ports, you may use it as output but remember to remove the filter capacitor. The connector J7 is 24V interlock if you don't use it then put a jumper across it.
The R6, R7 and C16 form a voltage divider to allow the software to detect the present of the motor driver voltage (+24V).
The connector J9 pin 1, 2 and 3 are power supply inputs. Pin 6 is a RXD pin of 8031, with little modification by connecting the RXD and TXD pin to RS232 driver (MAX232); this board can then be programmed to accept commands from the computer.
If you have any technical questions or need help in programming this board, please feel free to drop us an email. All additional or replacement parts can be ordered from us. The Molex connector housing to use with this board can be ordered from Digikey or also from us.


WZMICRO
http://rome.ro/wzmicro
Email: wzmicro@worldnet.att.net
Title
MD-Z Dual Motor Driver Board
Size
D
Document Number
MD Z
Rev
A
Date: June 1, 1999
Sheet 1 of 1



Code Sample

To help the beginner with the 8051 programming, the following is a listing of the motor driving sample code You may use it as is or modify it to suit your need. The source file and TASM cross compiler are also included in the disk.
In this program we use J5 to enable the motor0 (J3) and J4 controls the direction. The J6 and J8 control the motor1 (J2) Two timers T0 and T1 are used so both motors can run simultaneously.

```
;*****************************************************************************
;       (C) Copyright 1999 By WZMICRO Inc.
;       ALL RIGHTS RESERVED
;
;       TITLE:    MD2 Dual Motor Driver Board software
;
;       FILE:     MD2.ASM
;
;       DESCRIPTION: Sample Motor driving code for MD2 board
;
;*****************************************************************************
;       SOFTWARE HISTORY
;       01/01/99        Initial release
;
;                                                 Check Sum:
;*****************************************************************************
; INCLUDE FILES
;*****************************************************************************
#include        "mnemonics.def"         ;8031 mnemonic definition file
.LIST

;*****************************************************************************
; PERFORMANCE CONSTANTS
;*****************************************************************************
MAX_MTR1_STPS   .equ    00500H          ;Motor1 max run steps
MAX_MTR0_STPS   .equ    00500H          ;Motor0 max run steps
MAX_MTR0_TIME   .equ    0EF00H
MAX_MTR1_TIME   .equ    0EF00H
```

```
;*************************************************************************
;    I/O ADDRESSES
;*************************************************************************
mtr1_en         .equ    0B5H            ;motor1 enable bit (P3.5)
mtr0_en         .equ    0B4H            ;motor0 enable bit (P3.4)
mtr1_dir        .equ    0B6H            ;motor1 direction bit (P3.6)
mtr0_dir        .equ    0B2H            ;motor0 direction bit (P3.2)
mtr1_on         .equ    0B1H            ;motor1 on/off bit (P3.1)
mtr0_on         .equ    0B3H            ;motor0 on/off bit (P3.3)

;*************************************************************************
;    MODE FLAGS - Internal RAM Bit Addresses
;*************************************************************************
timer0_flag     .equ    40H             ;timer 0 interrupt flag (28H.0)
timer1_flag     .equ    41H             ;timer 1 interrupt flag (28H.1)
mtr0_cnt0_flg   .equ    42H             ;0 step count occured flag (28H.2)
mtr1_cnt0_flg   .equ    43H             ;0 step count occured flag (28H.3)

;function cycle flags
mtr1_cycle      .equ    48H             ;motor1 cycle flag
mtr0_cycle      .equ    49H             ;motor0 cycle flag

;*************************************************************************
;    REGISTER SPACE
;*************************************************************************
timer0_cnts     .equ    08H             ;timer 0 counts (2 bytes)
timer1_cnts     .equ    0AH             ;timer 1 counts (2 bytes)

;motor 0 variables
mtr0_request    .equ    10H             ;motor request: FOR_REQ or STOP_REQ
mtr0_state      .equ    11H             ;motor state: FOR_RAMP_UP, FOR_MAX_SPD
mtr0_mtr_num    .equ    12H             ;motor number: MTR0, MTR1 etc.
mtr0_step_cnt   .equ    15H             ;step count: count down (2 bytes)

;motor 1 variables
mtr1_request    .equ    18H             ;motor request: FOR_REQ or STOP_REQ
mtr1_state      .equ    19H             ;motor state: FOR_RAMP_UP, FOR_MAX_SPD
mtr1_mtr_num    .equ    1AH             ;motor number: MTR0, MTR1 etc.
mtr1_step_cnt   .equ    1DH             ;step count: count down (2 bytes)

mtr1_pha        .equ    0CH             ;keeps track of motor phases
mtr0_pha        .equ    0DH             ;keeps track of motor phases

;*************************************************************************
;    CONSTANTS
;*************************************************************************
CLEAR           .equ    00
IEMASK          .equ    0AH             ;enable timer 0 and timer 1 ints
STACKRAM        .equ    30H             ;location of bottom of stack
RAMSIZE         .equ    128             ;128 bytes of internal RAM
TIMERMODE       .equ    11H             ;MODE MASK: both timers set 16 bit timer

;physical motors:
MOTOR1          .equ    0
MOTOR0          .equ    1

;motor requests used by motor control routines:
STOP_REQ        .equ    0
```

```
FOR_REQ         .equ    1
REV_REQ         .equ    2

;motor states used by motor control routines:
ZERO_SPD        .equ    0
STOPPING        .equ    1
RUNNING         .equ    2

;*****************************************************************************
;    INTERRUPT JUMP TABLE
;*****************************************************************************
        .org      00H                ;Hardware reset vector address
HRESET:
        AJMP     SysInit             ;Hardware reset vector
        .org      03H                ;External Interrupt 0 vector address
        AJMP     SysInit             ;Not used, vector to start of program
        .org      0BH                ;TIMER 0 overflow intr vector address
        AJMP     Timer0Int           ;TIMER 0 overflow vector
        .org      13H                ;External Interrupt 1 vector address
        .org      1BH                ;TIMER 1 overflow intr vector address
        AJMP     Timer1Int           ;TIMER 1 Interrupt Service Routine
        .org      23H                ;Serial Interrupt Vector Address
        AJMP     SysInit             ;Not used, vector to start of program

;*****************************************************************************
;    TIMER 0 INTERRUPT
;
;       MODE 1 OPERATION, 16 BIT TIMER
;       INTERRUPT PERIOD = 1usec * (0FFFFH - timer_counts)
;*****************************************************************************
Timer0Int:
      PUSH    ACC
      PUSH    PSW                   ;save registers

      SETB    timer0_flag           ;timer interrupt occured flag
      MOV     TL0,timer0_cnts       ;reload timer with timer counts
      MOV     TH0,timer0_cnts+1

      POP     PSW
      POP     ACC
      RETI

;*****************************************************************************
;    TIMER 1 INTERRUPT
;
;       MODE 1 OPERATION, 16 BIT TIMER
;       INTERRUPT PERIOD = 1usec * (0FFFFH - timer_counts)
;*****************************************************************************
Timer1Int:
      PUSH    ACC
      PUSH    PSW                   ;save registers

      MOV     TL1,timer1_cnts       ;reload timer with timer counts
      MOV     TH1,timer1_cnts+1
      SETB    timer1_flag

      POP     PSW
      POP     ACC
```

```
      RETI

;*****************************************************************************
; INITIALIZING ROUTINE
;*****************************************************************************
SysInit:
      MOV     IE,#CLEAR               ;Disable all interrupts
      MOV     PSW,#CLEAR              ;Init PSW
      MOV     SP,#STACKRAM-1          ;Init Stack Pointer
      CLR     A                       ;Clear Internal RAM
      MOV     R0,#RAMSIZE-1
SYS_RAMCLR:
      MOV     @R0,A
      DJNZ    R0,SYS_RAMCLR

      MOV     TCON,#CLEAR             ;Halt timers, clear overflow flags
      MOV     IE,#IEMASK              ;Setup Interrupt Enable Register
      MOV     TMOD,#TIMERMODE         ;Setup TIMER 0 & TIMER 1
      SETB    IE.7                    ;Enable Interrupts

      CLR     mtr0_cycle
      CLR     mtr1_cycle

      MOV     mtr0_request,#STOP_REQ
      MOV     mtr0_state,#ZERO_SPD
      MOV     mtr1_request,#STOP_REQ
      MOV     mtr1_state,#ZERO_SPD

      MOV     mtr1_pha,#033H
      MOV     mtr0_pha,#033H

      AJMP    MainLoop                ;jump to main loop

;*****************************************************************************
; MainLoop
;*****************************************************************************
MainLoop:
ML_MTR0_CYCLE:
        JNB     mtr0_cycle,ML_MTR1_CYCLE       ;check if time to run
        LCALL   Motor0Run

ML_MTR1_CYCLE:
        JNB     mtr1_cycle,ML_MTR0_CHK         ;check if time to run
        LCALL   Motor1Run

ML_MTR0_CHK:    ;check if time to run
        SETB    mtr0_on
        JB      mtr0_on,ML_MTR1_CHK             ;Check port
        JB      mtr0_cycle,ML_MTR1_CHK          ;if motor already run
        SETB    mtr0_cycle                      ;else start run cycle
        ;set up mtr0
        SETB    mtr0_dir
        JB      mtr0_dir,MTR0_CHK               ;check direction
        MOV     mtr0_request,#FOR_REQ           ;load motor 0 start request
        SJMP    MTR0_CHK1
MTR0_CHK:
        MOV     mtr0_request,#REV_REQ
MTR0_CHK1:
```

```
        MOV     mtr0_mtr_num,#MOTOR0            ;set motor
        CLR     mtr0_cnt0_flg                   ;clear zero step flag

ML_MTR1_CHK:
        SETB    mtr1_on
        JB      mtr1_on,ML_RET                  ;check port bit
        JB      mtr1_cycle,ML_RET               ;if motor already run
        SETB    mtr1_cycle                      ;else start run cycle
        SETB    mtr1_dir
        JB      mtr1_dir,MTR1_CHK               ;check direction
        MOV     mtr1_request,#FOR_REQ           ;motor 1 start request
        SJMP    MTR1_CHK1
MTR1_CHK:
        MOV     mtr1_request,#REV_REQ
MTR1_CHK1:
        MOV     mtr1_mtr_num,#MOTOR1            ;set motor
        CLR     mtr1_cnt0_flg                   ;clear zero step flag

;*****************************
; your code goes here
;
;
;

ML_RET:
        AJMP    MainLoop

;*************************************************************************
; Motor0Run:
;
;*************************************************************************
Motor0Run:
        SETB    mtr0_on
        JNB     mtr0_on,MR_MTR0_CNTRL          ;check port if continue run
MR_STOP_MTR0:
        MOV     mtr0_state,#STOPPING           ;else stop motor
MR_MTR0_CNTRL:
        LCALL   Motor0Control                  ;run motor
MR0_RET:
      RET

;*************************************************************************
; Motor1Run
;
;*************************************************************************
Motor1Run:
        SETB    mtr1_on
        JNB     mtr1_on,MR_MTR1_CNTRL           ;check port if continue run
MR_STOP_MTR1:
        MOV     mtr1_state,#STOPPING            ;else stop motor
MR_MTR1_CNTRL:
        LCALL   Motor1Control                   ;run motor
MR1_RET:
      RET

;*************************************************************************
; Motor0Control: Controls any motor by using timer 0 interrupt.
;       Motor0Control uses mtr0_request and mtr0_state to control motor:
```

```
;               mtr0_state =  ZERO_SPD, STOPPING, RUNNING
;
;*****************************************************************************
Motor0Control:

        MOV     R1,#mtr0_state                  ;get mtr0_state address

        CJNE    @R1,#ZERO_SPD,M0C_STEP_FLAG     ;check if state = ZERO_SPD
        AJMP    M0C_STEP_TIME                   ;step motor

M0C_STEP_FLAG:  ;state <> ZERO_SPD check if time to step
        JB      timer0_flag,M0C_STOP            ;return if not time to step
        AJMP    M0C_RET

M0C_STOP:       ;time to step, check if state = STOPPING
        CLR     TCON.4
        CJNE    @R1,#STOPPING,M0C_STEP_MOTR     ;check if state = STOPPING
        MOV     mtr0_state,#ZERO_SPD            ;save new motor state
        SETB    mtr0_en
        CLR     TCON.4                          ;disable timer 0
        CLR     timer0_flag                     ;clear timer flag
        CLR     mtr0_cycle
        AJMP    M0C_RET                         ;return

M0C_STEP_MOTR:
        MOV     R0,mtr0_mtr_num                 ;set up next call
        MOV     A,mtr0_request                  ;set up next call
        LCALL   StepMotorNum                    ;step motor
        CLR     timer0_flag
M0C_STEP_TIME:
        MOV     timer0_cnts,#MAX_MTR0_TIME
        MOV     timer0_cnts+1,#MAX_MTR0_TIME>>8
        MOV     TL0,timer0_cnts                 ;reload timer with timer counts
        MOV     TH0,timer0_cnts+1
        SETB    TCON.4
        MOV     mtr0_state,#RUNNING             ;set motor state
        CLR     mtr0_en                         ;enable motor
M0C_RET:
      RET

;*****************************************************************************
; Motor1Control: Controls any motor by using timer 1 interrupt.
;       Motor1Control uses mtr1_request and mtr1_state to control motor:
;               mtr1_state = ZERO_SPD, STOPPING and RUNNING
;
;*****************************************************************************
Motor1Control:
        MOV     R1,#mtr1_state                  ;get mtr1_state address

        CJNE    @R1,#ZERO_SPD,M1C_STEP_FLAG     ;check if state = ZERO_SPD
        AJMP    M1C_STEP_TIME                   ;step motor

M1C_STEP_FLAG:  ;state <> ZERO_SPD check if time to step
        JB      timer1_flag,M1C_STOP            ;return if not time to step
        AJMP    M1C_RET

M1C_STOP:       ;time to step, check if state = STOPPING
        CLR     TCON.6
```

```
        CJNE    @R1,#STOPPING,M1C_STEP_MOTR     ;check if state = STOPPING
        MOV     mtr1_state,#ZERO_SPD            ;save new motor state
        SETB    mtr1_en
        CLR     TCON.6                          ;disable timer 1
        CLR     timer1_flag                     ;clear timer flag
        CLR     mtr1_cycle
        AJMP    M1C_RET                         ;return

M1C_STEP_MOTR:
        MOV     R0,mtr1_mtr_num                 ;set up next call
        MOV     A,mtr1_request                    ;set up next call
        LCALL   StepMotorNum                    ;step motor
        CLR     timer1_flag                     ;clear step flag

M1C_STEP_TIME:
        MOV     timer1_cnts,#MAX_MTR1_TIME
        MOV     timer1_cnts+1,#MAX_MTR1_TIME>>8
        MOV     TL1,timer1_cnts                 ;reload timer with timer counts
        MOV     TH1,timer1_cnts+1
        SETB    TCON.6                          ;enable interrupt
        MOV     mtr1_state,#RUNNING             ;set motor state
        CLR     mtr1_en                         ;enable motor
M1C_RET:
      RET


;*************************************************************************
; StepMotorNum: Step motor one step.
;       Pass value of mtr0_mtr_num or mtr1_mtr_num in R0.
;       Pass value of mtr0_request or mtr1_request in A.
;*************************************************************************
StepMotorNum:
        CJNE    R0,#MOTOR1,SMN_MTR0     ;check if not motor1
        CLR     C                       ;else
        CJNE    A,#FOR_REQ,SMN_MTR1_REV
        MOV     A,mtr1_pha              ;load motor phase
        RR      A                       ;shift phase for FORWARD
        AJMP    SMN_SAVE_MTR1
SMN_MTR1_REV:
        MOV     A,mtr1_pha              ;load motor phase
        RL      A                       ;shift phase for REVERSE
SMN_SAVE_MTR1:
        MOV     mtr1_pha,A              ;save new phase
        ANL     A,#0FH                  ;clear all but bits 2 & 3
        MOV     B,A                     ;store new bits 2 & 3 in B
        CLR     IE.7                    ;disable interrupts
        MOV     A,P1                    ;get current motor phases
        ANL     A,#0F0H                 ;clear bits 2 & 3
        ORL     A,B                     ;OR in new bits 2 & 3
        MOV     P1,A                    ;set new motor phases
        SETB    IE.7                    ;re-enable interrupts
        AJMP    SMN_RET

SMN_MTR0:
        CLR     C
        CJNE    A,#FOR_REQ,SMN_MTR0_REV
        MOV     A,mtr0_pha              ;load motor phase
        RR      A                       ;shift phase for FORWARD
```

```
        AJMP    SMN_SAVE_MTR0
SMN_MTR0_REV:
        MOV     A,mtr0_pha              ;load motor phase
        RL      A                       ;shift phase for REVERSE
SMN_SAVE_MTR0:
        MOV     mtr0_pha,A              ;save new phase
        ANL     A,#0F0H                 ;clear all but bits 6 & 7
        MOV     B,A                     ;store new bits 6 & 7 in B
        CLR     IE.7                    ;disable interrupts
        MOV     A,P1                    ;get current motor phases
        ANL     A,#0FH                  ;clear bits 6 & 7
        ORL     A,B                     ;OR in new bits 6 & 7
        MOV     P1,A                    ;set new motor phases
        SETB    IE.7                    ;re-enable interrupts

SMN_RET:
      RET


;*******************************************************************************
.END
```

Introduction to MD-3

The MD3 stepper motor controller board is implemented around the Atmel AT89C2051 microcontroller and SGS TEA3718 bipolar motor chopper driver. Two TEA3718 U5, U6 and some external components provide the full control function of a two-phase bipolar stepper motor. The system is commanded according to the desired mode of operation by the Microcontroller U1.

Connections

The J11 is the power-input connector; you may use any DC power source from +9V-32V. The MD3 already has the onboard +5V voltage regulator.
The motor is connected to J1 see Figure1 for the proper connection of the motor. Incorrect motor connection may damage the board.

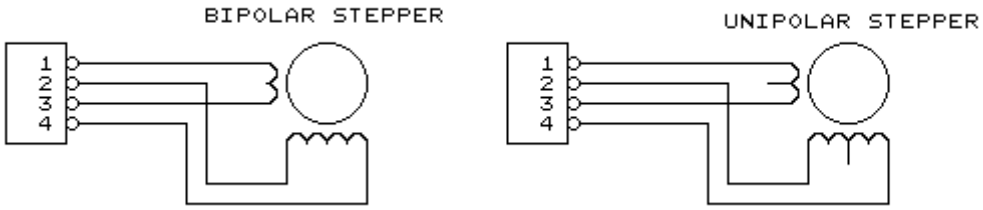
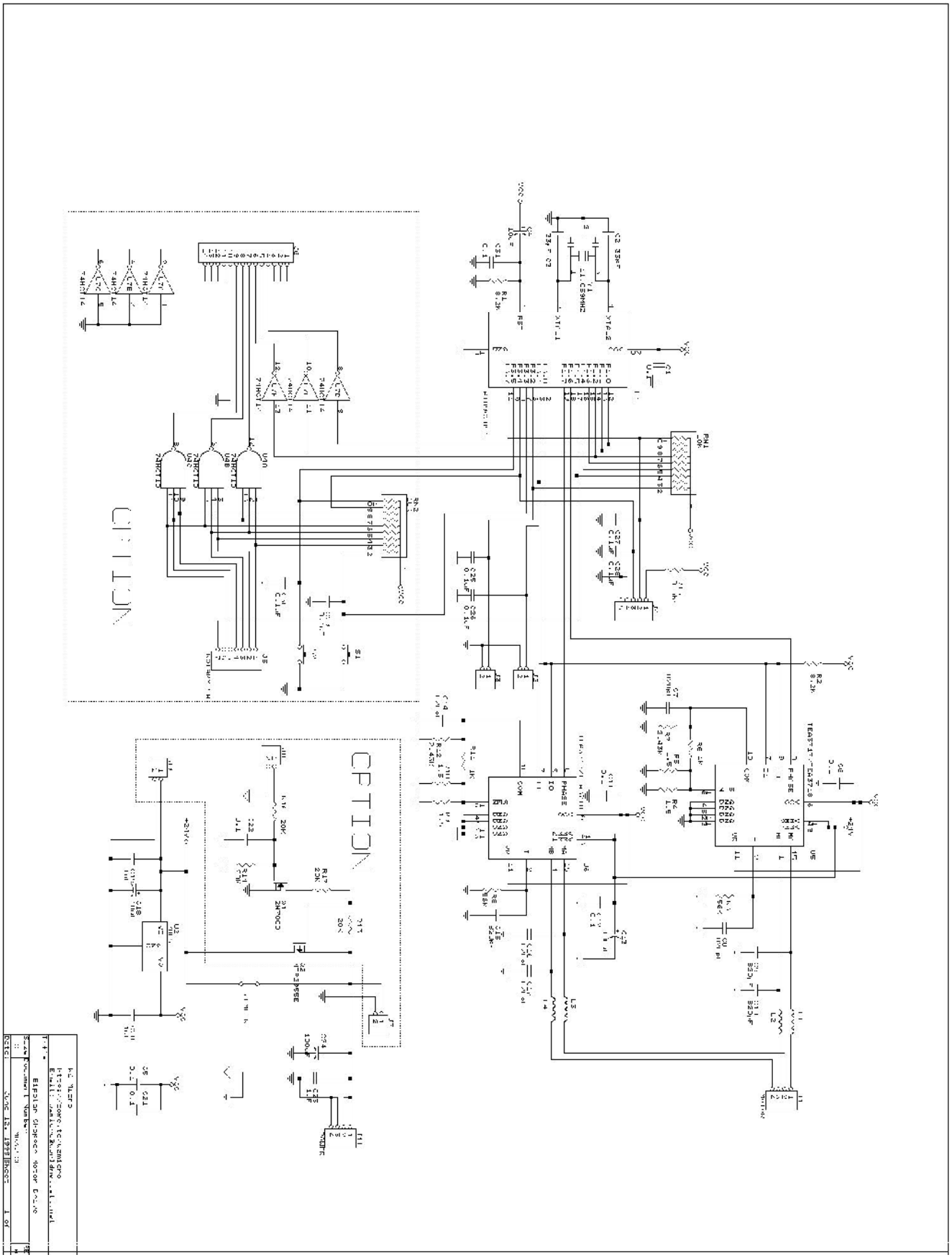


Figure 1 Motor connection

The J3, J4, and J5 are configured as input ports. All the components that are marked as “OPTION” on the schematic; those are not loaded on the MD3 board. You may use those if they are applicable to your application.

Figure 2. MD3 bipolar stepper motor driver board



Code Sample

In this sample program, The J2 is used to enable the motor. The J3 changes the direction of rotation, Only one timer is needed.

```
;*****************************************************************************
;       (C) Copyright 1999 By WZMICRO Inc.
;       ALL RIGHTS RESERVED
;
;       TITLE:    MD3 Stepping Motor Driver Board software
;
;       FILE:     MD3.ASM
;
;       DESCRIPTION: Sample Motor driving code for MD3 board
;
;*****************************************************************************
;       SOFTWARE HISTORY
;       06/01/99        Initial release
;
;                                                 Check Sum:
;*****************************************************************************
; INCLUDE FILES
;*****************************************************************************
#include        "mnemonics.def"         ;8031 mnemonic definition file
.LIST

;*****************************************************************************
; PERFORMANCE CONSTANTS
;*****************************************************************************
MAX_MTR0_STPS   .equ    00500H          ;Motor0 max run steps
MAX_MTR0_TIME   .equ    0EF00H

;*****************************************************************************
;    I/O ADDRESSES
;*****************************************************************************
mtr0_en         .equ    0B0H             ;motor0 enable bit (P3.0)
mtr0_on         .equ    0B1H             ;motor0 on/off bit (P3.1)
mtr0_dir        .equ    0B2H             ;motor0 direction bit (P3.2)

;*****************************************************************************
;    MODE FLAGS - Internal RAM Bit Addresses
;*****************************************************************************
timer0_flag     .equ    40H             ;timer 0 interrupt flag (28H.0)
mtr0_cnt0_flg   .equ    42H             ;0 step count occured flag (28H.2)

;function cycle flags
mtr0_cycle      .equ    49H             ;motor0 cycle flag

;*****************************************************************************
;    REGISTER SPACE
;*****************************************************************************
timer0_cnts     .equ    08H             ;timer 0 counts (2 bytes)

;motor 0 variables
mtr0_request    .equ    10H             ;motor request: FOR_REQ or STOP_REQ
```

```
mtr0_state      .equ    11H             ;motor state: FOR_RAMP_UP, FOR_MAX_SPD
mtr0_mtr_num    .equ    12H             ;motor number: MTR0, MTR1 etc.
mtr0_step_cnt   .equ    15H             ;step count: count down (2 bytes)

mtr0_pha        .equ    0DH             ;keeps track of motor phases

;*****************************************************************************
;    CONSTANTS
;*****************************************************************************
CLEAR           .equ    00
IEMASK          .equ    0AH             ;enable timer 0 and timer 1 ints
STACKRAM        .equ    30H             ;location of bottom of stack
RAMSIZE         .equ    128             ;128 bytes of internal RAM
TIMERMODE       .equ    11H             ;MODE MASK: both timers set 16 bit timer

;physical motors:
MOTOR0          .equ    1

;motor requests used by motor control routines:
STOP_REQ        .equ    0
FOR_REQ         .equ    1
REV_REQ         .equ    2

;motor states used by motor control routines:
ZERO_SPD        .equ    0
STOPPING        .equ    1
RUNNING         .equ    2

;*****************************************************************************
;    INTERRUPT JUMP TABLE
;*****************************************************************************
        .org      00H                 ;Hardware reset vector address
HRESET:
          AJMP     SysInit             ;Hardware reset vector
        .org      03H                 ;External Interrupt 0 vector address
        AJMP     SysInit              ;Not used, vector to start of program
        .org      0BH                 ;TIMER 0 overflow intr vector address
        AJMP     Timer0Int            ;TIMER 0 overflow vector
        .org      13H                 ;External Interrupt 1 vector address
        .org      1BH                 ;TIMER 1 overflow intr vector address
        AJMP     Timer1Int            ;TIMER 1 Interrupt Service Routine
        .org      23H                 ;Serial Interrupt Vector Address
        AJMP     SysInit              ;Not used, vector to start of program

;*****************************************************************************
;    TIMER 0 INTERRUPT
;
;       MODE 1 OPERATION, 16 BIT TIMER
;       INTERRUPT PERIOD = 1usec * (0FFFFH - timer_counts)
;*****************************************************************************
Timer0Int:
        PUSH    ACC
        PUSH    PSW                     ;save registers

        SETB    timer0_flag             ;timer interrupt occured flag
        MOV     TL0,timer0_cnts         ;reload timer with timer counts
      MOV     TH0,timer0_cnts+1
```

```
      POP     PSW
        POP     ACC
      RETI

;*****************************************************************************
;    TIMER 1 INTERRUPT
;
;       MODE 1 OPERATION, 16 BIT TIMER
;       INTERRUPT PERIOD = 1usec * (0FFFFH - timer_counts)
;*****************************************************************************
Timer1Int:
      PUSH    ACC
      PUSH    PSW                     ;save registers


      POP     PSW
      POP     ACC
      RETI

;*****************************************************************************
; INITIALIZING ROUTINE
;*****************************************************************************
SysInit:
      MOV     IE,#CLEAR               ;Disable all interrupts
      MOV     PSW,#CLEAR              ;Init PSW
      MOV     SP,#STACKRAM-1          ;Init Stack Pointer
      CLR     A                       ;Clear Internal RAM
      MOV     R0,#RAMSIZE-1
SYS_RAMCLR:
      MOV     @R0,A
      DJNZ    R0,SYS_RAMCLR

      MOV     TCON,#CLEAR             ;Halt timers, clear overflow flags
      MOV     IE,#IEMASK              ;Setup Interrupt Enable Register
      MOV     TMOD,#TIMERMODE         ;Setup TIMER 0 & TIMER 1
      SETB    IE.7                    ;Enable Interrupts

      CLR     mtr0_cycle

      MOV     mtr0_request,#STOP_REQ
      MOV     mtr0_state,#ZERO_SPD
      MOV     mtr0_pha,#033H

      AJMP    MainLoop                ;jump to main loop

;*****************************************************************************
; MainLoop
;*****************************************************************************
MainLoop:
ML_MTR0_CYCLE:
        JNB     mtr0_cycle,ML_MTR0_CHK       ;check if time to run
        LCALL   Motor0Run

ML_MTR0_CHK:    ;check if time to run
        SETB    mtr0_on
        JB      mtr0_on,ML_RET                    ;Check port
        JB      mtr0_cycle,ML_RET          ;if motor already run
        SETB    mtr0_cycle                      ;else start run cycle
```

```
        ;set up mtr0
        SETB    mtr0_dir
        JB      mtr0_dir,MTR0_CHK               ;check direction
        MOV     mtr0_request,#FOR_REQ           ;load motor 0 start request
        SJMP    MTR0_CHK1
MTR0_CHK:
        MOV     mtr0_request,#REV_REQ
MTR0_CHK1:
        MOV     mtr0_mtr_num,#MOTOR0            ;set motor
        CLR     mtr0_cnt0_flg                   ;clear zero step flag

;*****************************
; your code goes here
;
;
;

ML_RET:
        AJMP    MainLoop

;*************************************************************************
; Motor0Run:
;
;*************************************************************************
Motor0Run:
        SETB    mtr0_on
        JNB     mtr0_on,MR_MTR0_CNTRL          ;check port if continue run
MR_STOP_MTR0:
        MOV     mtr0_state,#STOPPING           ;else stop motor
MR_MTR0_CNTRL:
        LCALL   Motor0Control                  ;run motor
MR0_RET:
        RET


;*************************************************************************
; Motor0Control: Controls any motor by using timer 0 interrupt.
;       Motor0Control uses mtr0_request and mtr0_state to control motor:
;               mtr0_state =  ZERO_SPD, STOPPING, RUNNING
;
;*************************************************************************
Motor0Control:

        MOV     R1,#mtr0_state                 ;get mtr0_state address

        CJNE    @R1,#ZERO_SPD,M0C_STEP_FLAG    ;check if state = ZERO_SPD
        AJMP    M0C_STEP_TIME                  ;step motor

M0C_STEP_FLAG:  ;state <> ZERO_SPD check if time to step
        JB      timer0_flag,M0C_STOP               ;return if not time to step
        AJMP    M0C_RET

M0C_STOP:       ;time to step, check if state = STOPPING
        CLR     TCON.4
        CJNE    @R1,#STOPPING,M0C_STEP_MOTR    ;check if state = STOPPING
        MOV     mtr0_state,#ZERO_SPD           ;save new motor state
        SETB    mtr0_en
        CLR     TCON.4                         ;disable timer 0
```

```
        CLR     timer0_flag                     ;clear timer flag
        CLR     mtr0_cycle
        AJMP    M0C_RET                         ;return

M0C_STEP_MOTR:
        MOV     R0,mtr0_mtr_num                 ;set up next call
        MOV     A,mtr0_request                  ;set up next call
        LCALL   StepMotorNum                    ;step motor
        CLR     timer0_flag
M0C_STEP_TIME:
        MOV     timer0_cnts,#MAX_MTR0_TIME
        MOV     timer0_cnts+1,#MAX_MTR0_TIME>>8
        MOV     TL0,timer0_cnts                 ;reload timer with timer counts
        MOV     TH0,timer0_cnts+1
        SETB    TCON.4
        MOV     mtr0_state,#RUNNING             ;set motor state
        CLR     mtr0_en                         ;enable motor
M0C_RET:
        RET

;*************************************************************************
; StepMotorNum: Step motor one step.
;       Pass value of mtr0_mtr_num or mtr1_mtr_num in R0.
;       Pass value of mtr0_request or mtr1_request in A.
;*************************************************************************
StepMotorNum:
        CLR     C
        CJNE    A,#FOR_REQ,SMN_MTR0_REV
        MOV     A,mtr0_pha              ;load motor phase
        RR      A                       ;shift phase for FORWARD
        AJMP    SMN_SAVE_MTR0
SMN_MTR0_REV:
        MOV     A,mtr0_pha              ;load motor phase
        RL      A                       ;shift phase for REVERSE
SMN_SAVE_MTR0:
        MOV     mtr0_pha,A              ;save new phase
        ANL     A,#0C0H                 ;clear all but bits 6 & 7
        MOV     B,A                     ;store new bits 6 & 7 in B
        CLR     IE.7                    ;disable interrupts
        MOV     A,P1                    ;get current motor phases
        ANL     A,#03FH                 ;clear bits 6 & 7
        ORL     A,B                     ;OR in new bits 6 & 7
        MOV     P1,A                    ;set new motor phases
        SETB    IE.7                    ;re-enable interrupts

SMN_RET:
        RET


;*************************************************************************
.END
```