RoboSpeedy

EEL 5666 Final Technical Report

Author:
Steve Kerslake

Submitted in Partial Fulfillment of the Requirements
for EEL 5666 Robotics Course

Date
03 August 1999

Abstract

This paper concerns the development and testing of an autonomous robot. This robot is better known as RoboSpeedy. It was designed for high-speed navigation of large indoor spaces. The goal was a robot that could navigate a course in the same manner as a remotely controlled racecar. The central platform of RoboSpeedy is a Tamaya remote-controlled racecar chassis. This platform was expanded to allow the mounting of various sensors and electronics. RoboSpeedy has an Intel 8051GB micro-controller and 32K of SRAM. The micro-controller receives input from sonar, a flux-gate compass, and hacked Sharp infrared sensors. It uses these inputs to control the steering servo, the sonar turret servo, the motor control relays, and the motor itself.

Executive Summary

RoboSpeedy was designed for high-speed navigation of large indoor spaces. The goal was a robot that could navigate a course in the same manner as a remotely controlled racecar. The central platform of RoboSpeedy is a Tamaya remote-controlled racecar chassis. The bodywork was designed on AutoCAD and milled out in the IMDL using the T-Tech. I mounted the wooden body panels to the plastic chassis using #10 all thread. The wooden platform was completely immobile with respect to the plastic chassis. The entire wooden bodywork was reinforced with steel angle brackets. I wanted to make sure that RoboSpeedy was as crashproof as possible. This platform provided mounting space and protection for the various sensors and electronics that RoboSpeedy used.

RoboSpeedy has an Intel 8051GB micro-controller and 32K of SRAM. These are installed on a PCB manufactured by Tecny Electronics. This board also allows the mounting of an LM7805 voltage regulator and a MAX232. The micro-controller receives input from sonar, a flux-gate compass, and hacked Sharp infrared sensors. The Polaroid sonar was purchased from Mekatronics. This sonar has a range of fifteen feet and a simple continuous DC output. The IR sensors were also purchased from Mekatronics. These sensors are operated in the normal IMDL analog fashion. The compass is a Precision Navigation flux-gate magnetometer with a digital output.

The micro-controller uses these inputs to control the steering servo, the sonar turret servo, the motor control relays, and the motor itself. All of the motors and relays are optically isolated from the micro-controller. The opto-isolators will prevent the processor from being reset in the event of low battery power. The servos and relays operate from an eight-pack of AA batteries; the motor runs from a 7.2-volt DC battery pack with the same ground as the servos. Finally the micro-controller has its own circuit and battery pack.

Introduction

For those of you who were absent during the abstract and the executive summary this paper concerns the development and testing of an autonomous robot. This robot is better known as RoboSpeedy. It was designed for high-speed navigation of large indoor spaces. The goal was a robot that could navigate a course in the same manner as a remotely controlled racecar. This task is difficult only because the sensors that are available for our robots have a short range and a low resolution. I propose that a "steerable" sonar and a flux-gate compass could be used to give an autonomous agent the ability to operate at high speeds.

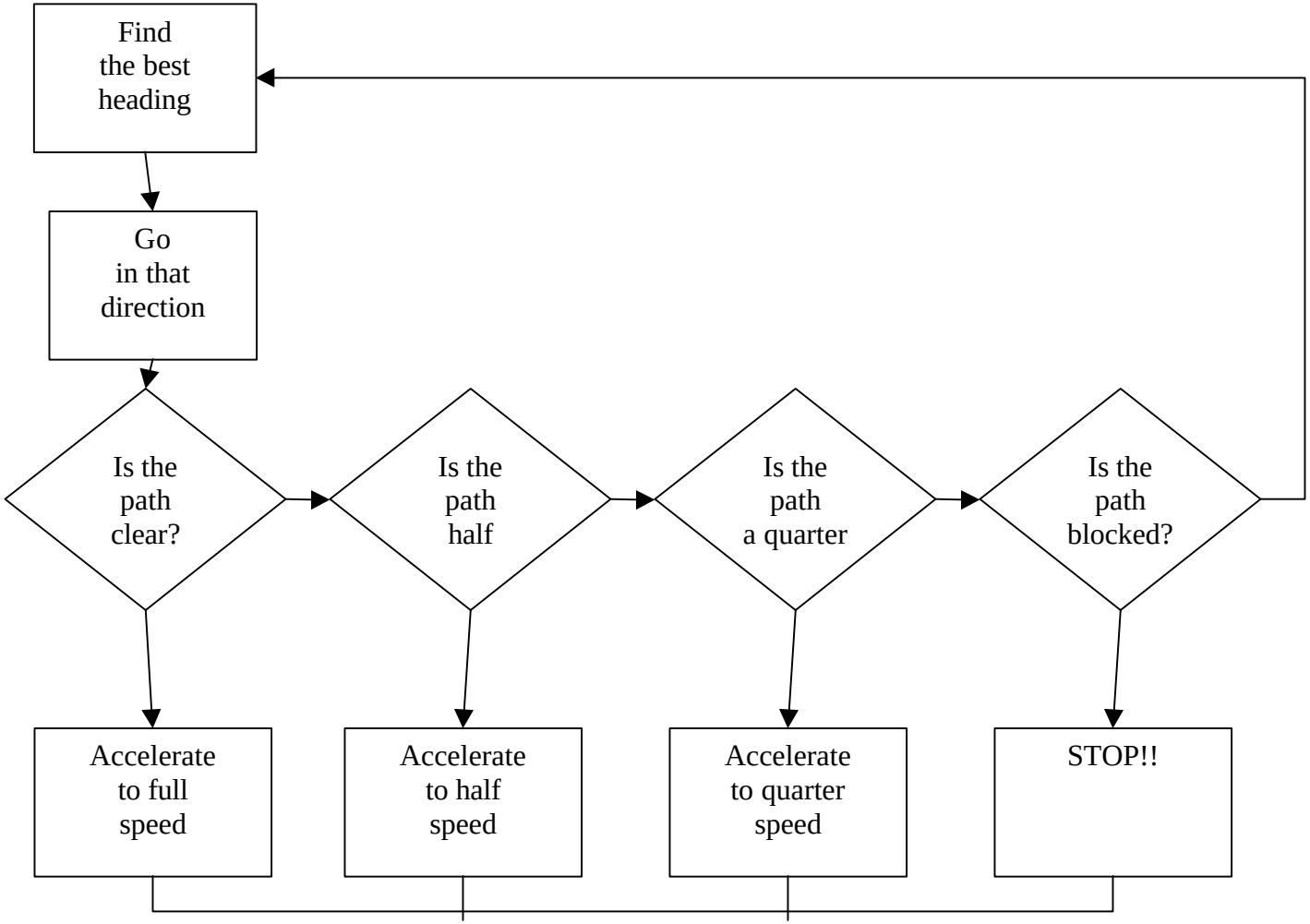
The Intel 8051GB provides RoboSpeedy's "reflexes." I initially decided to use the Intel processor because it has a 22MHz crystal. I didn't realize until after I received it that it divided the crystal speed by twelve internally. This, however, did not prove to be a handicap. This is because the 8051 instructions typically only require one or two machine cycles to execute. In the end, the 8051GB turned out to be an awesome robotics micro-controller. This is due to the large number of hardware controlled features that allow the programmer to set up the operation and then forget about it. These features include the Programmable Counter Array, the A/D converter, and the self-resetting interrupt system. The problems that I had with the 8051GB were that I had no software pre-written for me and I had no information resources available to me. This meant that I had to spend valuable time researching and learning how to program the 8051GB.

Integrated System

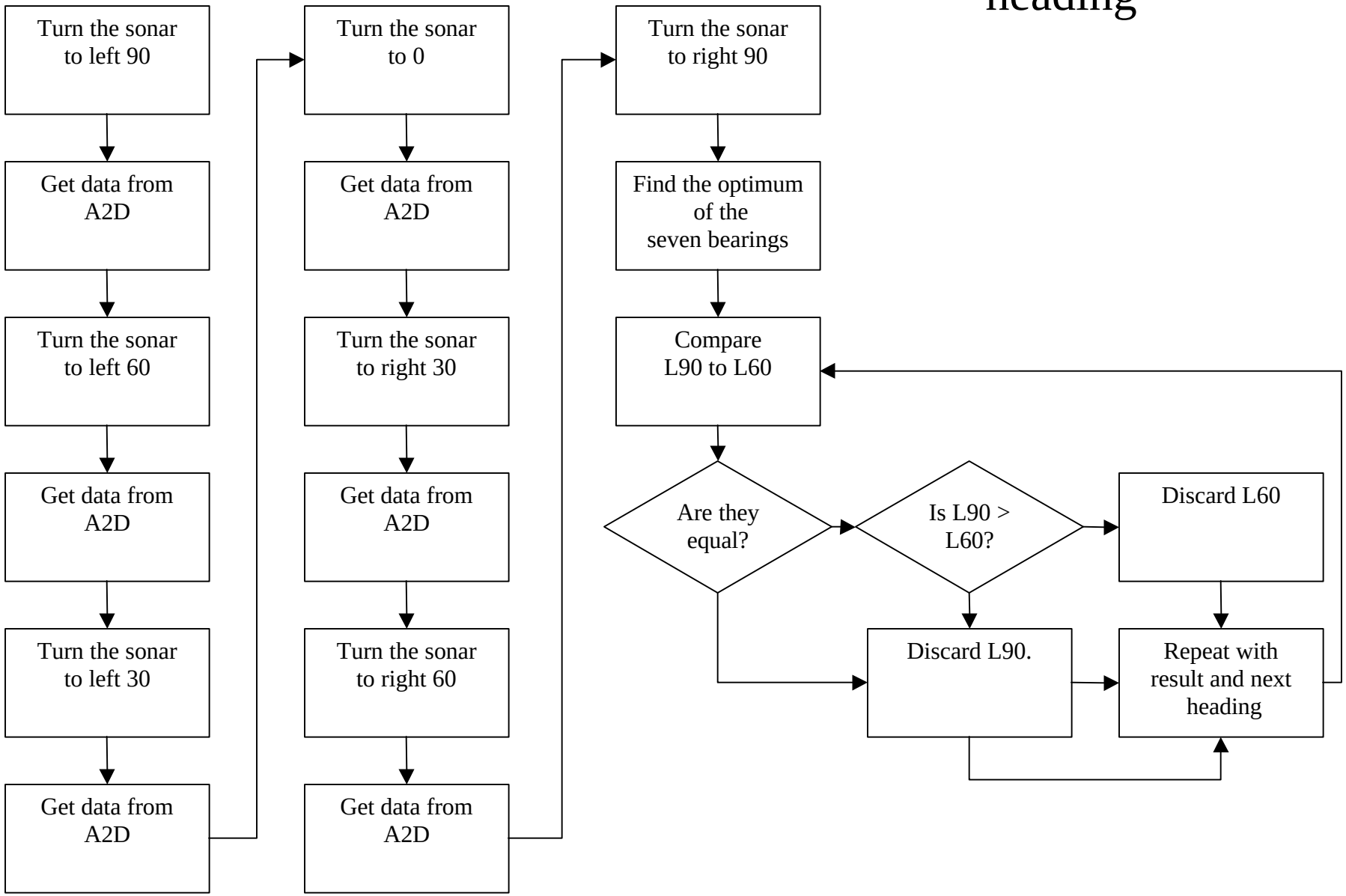
Note: RoboSpeedy is physically complete. Unfortunately, I have not had time to implement the programming in the way that I wanted to. The time factor has been further aggravated by the fact that I have had to do it all in assembler. I have written code to test all of the robot s systems. They are all powered up and work together. The only software that is incomplete is the final version of the navigation software. I have written a simple version of this software that uses the IR sensors and the sonar. This version will not pan the sonar and it does not accept input from the compass (although the compass is functional and the sonar does pan).

RoboSpeedy is designed to look for the longest available path and follow that path until a more optimal path becomes available. It does this by panning the sonar until it locates the best path. It then compares heading of the best path to the actual heading of the robot. If these paths do not match it will change direction until they do. If, for example, sonar detects an optimal path at left 90 and its current heading is 270. The robot will determine that left 90 is at 180 by subtracting 90 from 270. It will then turn to the left until it reaches 180. This heading matching also makes sure that the robot travels in a straight line. If it veers from the desired course the compass will detect this deviation and correct for it.

The distance available on the best path governs the robot's speed. Obviously, a longer path will dictate a higher speed than a short one. Placing the motor in reverse implements braking. This is touchy because this can cause the robot to swap ends (rapidly rotate 180 degrees). Due to the acceleration properties of the motor, optimal speed control will only be gained through extensive testing.


Find
the best
heading
Go
in that
direction
Is the
path
clear?
Is the
path
half
Is the
path
a quarter
Is the
path
blocked?
Accelerate
to full
speed
Accelerate
to half
speed
Accelerate
to quarter
speed
STOP!!

Find the best heading

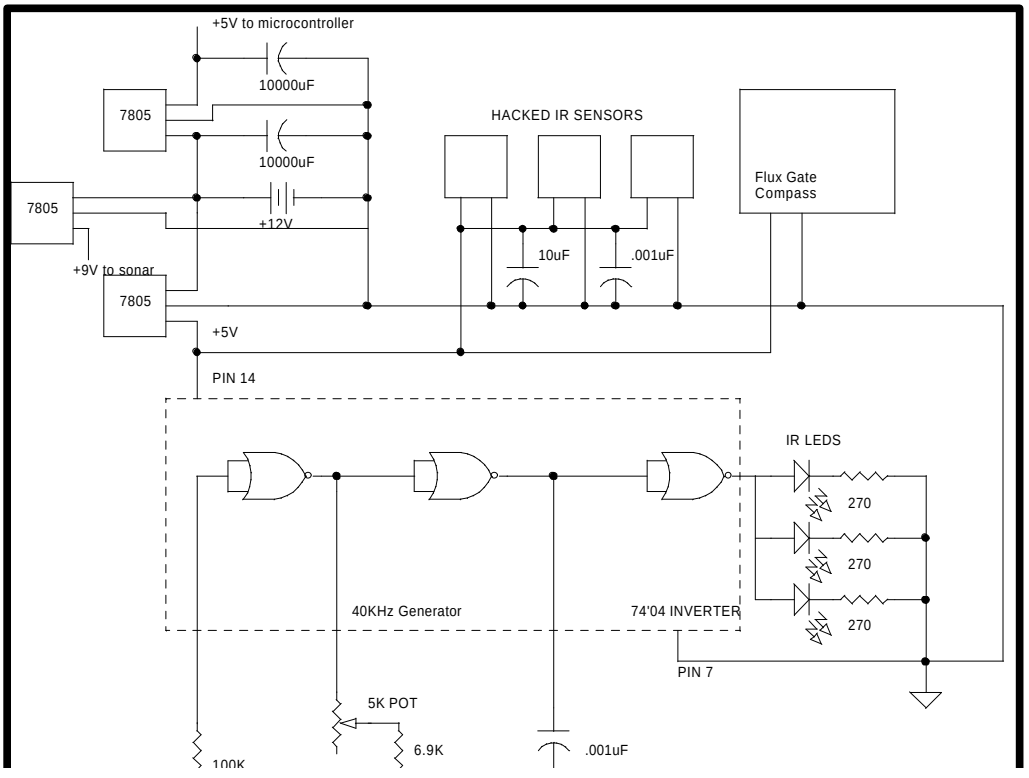


Mobile Platform

The mobile platform is based on a Tamaya remote-controlled racecar frame. The chassis has a full suspension that will help to dampen vibration. The original springs have been replaced with much stiffer ones to allow for the heavy battery load. The chassis holds the original 7.2-volt battery pack and two extra AA eight packs. The drive motor is original but the controller has been replaced with a digital controller that I assembled from parts in the lab.

The body of the platform is constructed of plywood and it is mounted to the chassis with #10 all-thread and it is reinforced with steel angle brackets. The body houses and protects all of the electronics. The electronics have been mounted completely within the body to prevent any damage in the event of a crash. This design includes a raised platform that places the sonar 11 inches from the ground. This was done to prevent ground interference and it may be possible to lower the platform in the future.

The entire system consists of two completely separate electric circuits. The motor control and servo circuits, and the sensor and micro-controller circuits. These circuits were



separated to prevent potential noise and power problems in the micro-controller circuit. The sensor and micro-controller circuit consisted of the 40-KHz generator for the IR LED's, the IR sensors, the compass, the sonar, and the micro-controller. The diagram for this circuit is shown in Figure 1.

Actuation

The entire system consists of two completely separate electric circuits. The motor control and servo circuits, and the sensor and micro-controller circuits. These circuits were separated to prevent potential noise and power problems in the micro-controller circuit. The sensor and micro-controller circuit consisted of the 40-KHz generator for the IR LED's, the IR sensors, the compass, the sonar, and the micro-controller. The diagram for this circuit is shown in Figure 1.

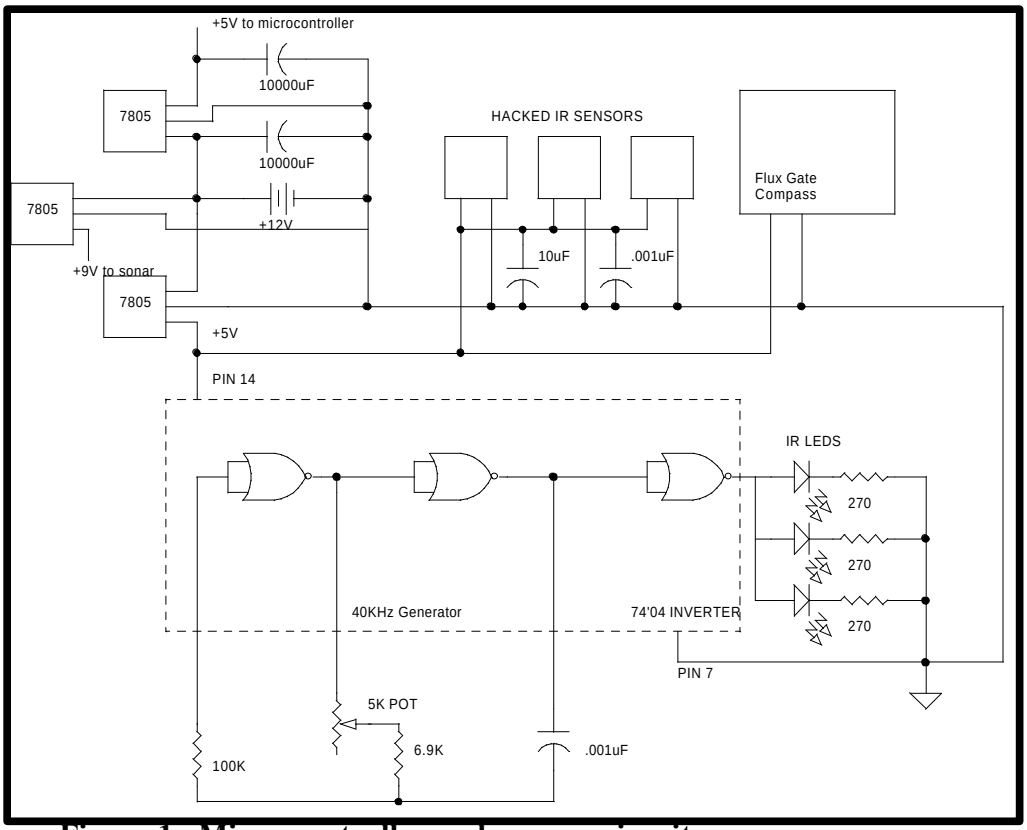


Figure 1. Micro-controller and sensors circuit.

The motor-controller and servo circuit consisted of the power mosfets for the motor and the motor relays and the power supply for the servos. The diagram for this circuit is shown in Figure 2.

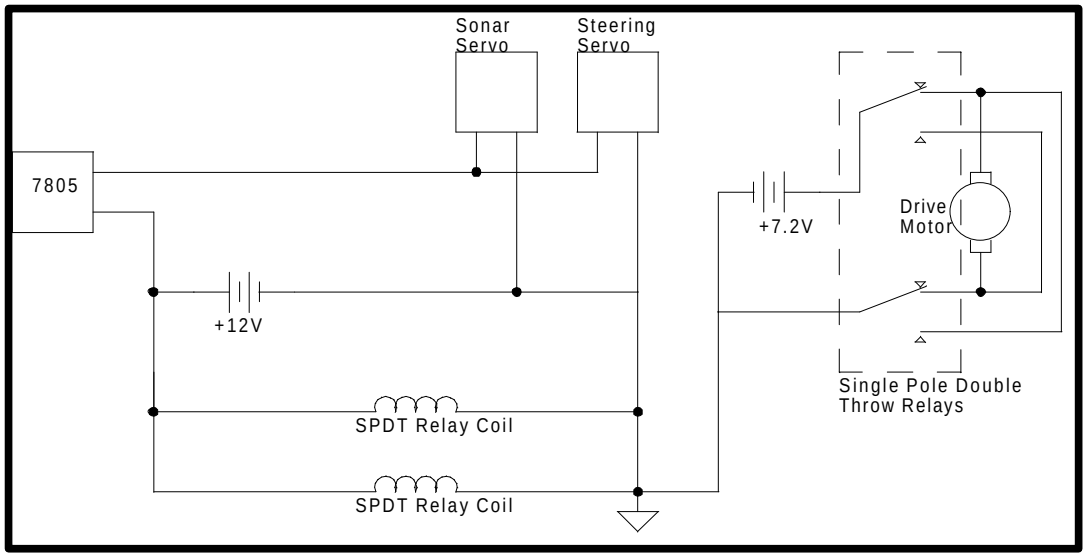


Figure 2. Motor controller circuit.

The micro-controller controlled the power motor and the servos with pulse width modulation. The signals were transmitted from the micro-controller to the mosfets, relays, and servos via opto-isolators. The non-inverting buffer was used because the micro-controller can not drive the opto-isolator LED's. This circuit is detailed in Figure 3.

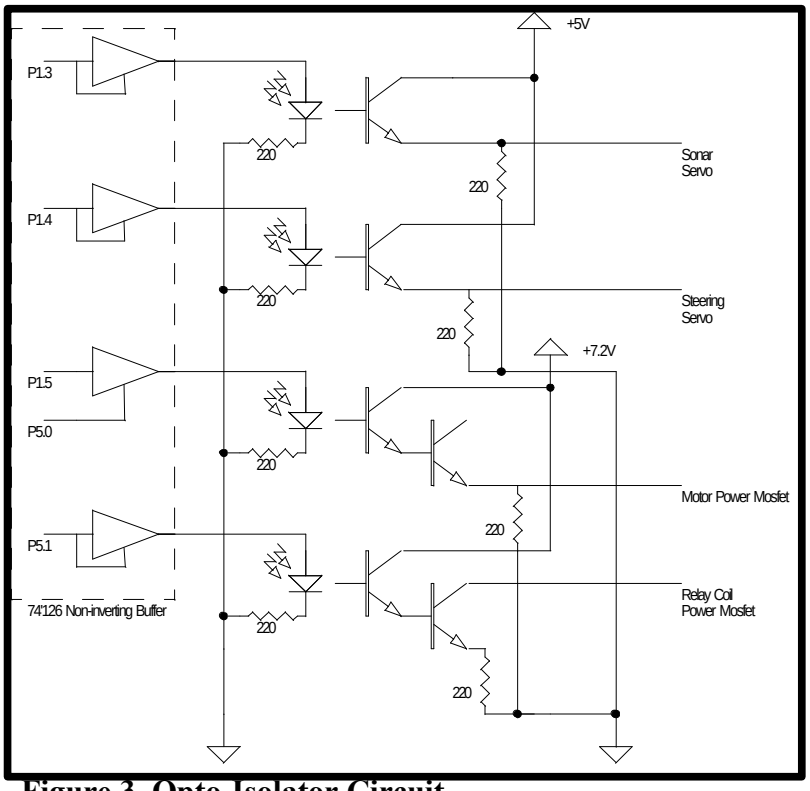


Figure 3. Opto-Isolator Circuit.

Separate test code was written for steering servo, the sonar servo, and the motor-controller. This test code is included in the appendix.

Sensors

Compass Configuration

Speedy's compass is mounted on the chassis, and it is positioned as far as possible from the motor. This is necessary because the Vector compass is very sensitive to magnetic fields. It is a compass after all. The V2G is a gimbaled compass. I chose a gimbaled compass because my robot may not always be level. The gimbals are designed to allow up to a 20-degree tilt. The digital compass communicates with the microcontroller with a serial interface. It can act as either the master or the slave in a serial peripheral role. I am using the compass as the slave because the 8051 will not act as a slave. Speedy will use the relative headings generated by the compass to help it steer and to avoid obstacles. Readings from the compass will provide feedback to Speedy's steering and allow it to maintain a straight course. Referencing the direction the sonar is pointing to the current bearing of the robot will help the sonar guide the robot around obstacles.

The Interface

The compass outputs a sixteen bit binary number to represent the heading in degrees. The numbers are from 0 the 359 decimal, which corresponds to 0 to 167 hex. The entire interface utilizes 8 of the microcontroller I/O pins. I am currently using 10 pins but since I don't intend to calibrate the compass I will not need /Cal and CI. The pins are used as follows:

P4.0 = /Slave Select
P4.1 = Serial Clock
P4.2 = Serial Data Out
P4.3 = /Poll – Continuous
P4.4 = /Calibrate
P4.5 = Calibrate Indicator
P4.6 = End of Conversion
P4.7 = Power

P3.3 = /Master - Slave
P3.2 = /Reset

In order to get the compass to operate a very meticulous order of operations must be employed. The following is an overview of the steps required to get an output:

1 /M-S must be pulled high.
2 /BCD-Bin must be high or not connected.
3 The /Res pin should be tied low for high resolution or high for low resolution.
4 To request a reading:
    - Pulse /P-C low for at least 10msec. There is no maximum time for holding /P-C low, but it must be high before /SS is taken low. Also /SS must be high before taking /P-C low.
    - The Vector calculates heading while EOC is low. The Vector will drop EOC low after /P-C is dropped low to indicate that it is calculating. When it is through calculating it will set EOC high.
    - After EOC goes high you have to wait 10msecs to take /SS low. /SS must be held low to clock data out.
    - SCLK must then be clocked by the microcontroller. It must be high when /SS is taken low. /SS must be low for at least 5msecs before taking SCLK low for the first time.
    - Take SCLK low.
    - Take SCLK high and read the first data bit from SDO.
    - Repeat this cycle seven more times, reading a bit each time SCLK goes high.
    - After the eighth SCLK you must wait another 5msecs.
    - Then SCLK is clocked eight more times.
    - When SCLK goes high for the sixteenth time it should be kept that way until /SS is returned high.

The timing for this operation is very important. If it is not followed exactly the compass will lock up until it is reset.

The total amount of time required for a compass reading is calculated as follows:

10msecs for /P-C.

80-100msecs for EOC.

10msecs after EOC to set /SS low.

5msecs after /SS goes low before SCLK goes low.

5msecs after 8th bit.

Maximum clock rate of 1Mhz for 16 clock cycles

(I ran the clock at 40Khz)

110msecs

I will probably use EOC as an interrupt. In other words I will poll the compass and then wait for the interrupt before servicing the rest of the compass operation. This will allow me to poll the compass and then go back to work until the calculations are completed. I have included my test code as an addendum to this report.

Testing

I tested the compass under various conditions. Initially I tested it by itself to see how repeatable its calculations were. I did this by taking several readings in a row while the compass was sitting still. These results are listed in Table 2. I then tested repeatability by rotating the compass 90 degrees between measurements. I did this in four different directions. These results are in table 1.

Run	Heading 1	Heading 2	Heading 3	Heading 4
1	279	190	21	118
2	281	190	23	113
3	278	189	20	118
4	283	191	20	113
5	278	188	20	115
Mean	279.8	189.6	20.80	115.4

Table 1

Run	Heading 1	Heading 2	Heading 3	Heading 4
1	16	113	276	184
2	16	113	275	184
3	16	113	276	184
4	16	112	275	184
5	16	112	277	184
6	16	113	275	184
7	16	112	276	184
8	16	113	276	184
9	16	113	275	184
10	16	113	275	184
Mean	16	112.70	275.60	184.00

Table 2

I repeated these tests after placing the compass in the chassis. There was no significant change. Finally, I repeated the first tests with the motor running at various speeds. Table 3 and 4 contain the results of these tests.

Run	Heading 1	Heading 2	Heading 3	Heading 4
Original w/o Motor running	280	190	21	118
1	285	193	24	121
2	286	193	27	116
3	283	192	25	121
4	288	194	23	116
5	282	191	25	118
Mean	284.8	192.6	24.80	118.4

Table 3

Run	Heading 1	Heading 2	Heading 3	Heading 4
Original w/o Motor running	16	113	276	184
1	19	113	276	184
2	20	113	275	184
3	19	113	276	184
4	19	112	275	184
5	19	112	277	184
6	20	113	275	184
7	18	112	276	184
8	19	113	276	184
9	19	113	275	184
10	20	113	275	184
Mean	19.2	112.7	275.70	184.00

Table 4

Sonar

The sonar was purchased from Mekatronics and has a range of 15 feet. I regulated the voltage to the sonar with a 7809 voltage regulator. This prevents the range from changing as the batteries voltage drops. I tested the sonar in the lab and ranged it using a voltmeter and the 9" tiles. The data I obtained is included in Table 5.

Range (inches)	Voltage 1st Run	Voltage 2nd Run	Difference
9	.3	.25	.05
18	.514	.509	.005
27	.761	.766	.005
36	1.01	1.00	.01
45	1.27	1.26	.01
54	1.52	1.52	0.0
63	1.77	1.76	.01
72	2.02	2.02	0.0
81	2.28	2.28	0.0
90	2.54	2.53	.01
99	2.80	2.78	.02
108	3.04	3.04	0.0
117	3.31	3.28	.03
126	3.56	3.55	.01
135	3.84	3.80	.04
144	4.08	4.06	.02
153	4.32	4.32	0.0
162	4.59	4.51	.08
171	4.84	4.80	.04
180	4.96	4.96	0.0

Table 5 Sonar test results

The IR sensors were Sharp sonar sensors with the analog hack. They perform the same as everyone else's IR sensors.

Behaviors

The simplified navigation program that I implemented has the following basic behaviors:

- Front left IR/Front Right IR – If sensor level > 100 then turn wheels to left and back up.
- Rear IR – if sensor level > 100 straighten wheels and go forward
- If left IR 10 > right IR and left IR > 100 then go right
- If right IR 10 > left IR and left IR > 100 then go left
- Sonar > 15 ft – Go fast
- Sonar > 9 ft and not going fast -> go medium; if going fast, go slow
- Sonar > 6 ft and not going fast -> go slow; if going fast, reverse
- Sonar < 1 ft -> reverse

This is really basic. The platform that I have constructed is capable of performing much more complex behaviors. I want to implement much more complex behaviors, but I don't have the skills yet. Hopefully, when I learn how to write for the Intel in C, I will be able to implement these behaviors.

Appendix

This is the simplified navigation code for RoboSpeedy

```
SON_SPD
PAGE 1

8000                     1            org     8000h
8000 028200              2            jmp     8200h
                         3
                         4            $MOD51GB
                         5
                         6    ;Pulse width signals
                         7    PWM_MOT         EQU     R0
                         8    PWM_STR         EQU     R1
                         9    DATA_BYTE       EQU     R4
                        10    Milkyway        EQU     R1
                        11    ;Motor Direction
  00F9                  12    STP_GO          EQU     P5.1
  00F8                  13    FOR_REV         EQU     P5.0
                        14
                        15    ;Speeds
  00B0                  16    FWD_LO          EQU     0B0H    ;C0
  00A0                  17    FWD_MED         EQU     0A0H    ;B0
  00A0                  18    FWD_HI          EQU     0A0H    ;B0
  0080                  19    REV_LO          EQU     080H    ;90
                        20
                        21    ;Steering Directions
  003F                  22    Straight        EQU     03fH
  0030                  23    Full_Left       EQU     030H
  0037                  24    Half_Left       EQU     037H
  003A                  25    Qtr_Left        EQU     03AH
  004D                  26    Full_Right      EQU     04DH
  0045                  27    Half_Right      EQU     045H
  0042                  28    Qtr_Right       EQU     042H
                        29
                        30
                        31
                        32    ;Sonar ranges
  00F5                  33    Sonar_15        EQU     245
  00D0                  34    Sonar_12        EQU     208
  009B                  35    Sonar_9         EQU     155
  0067                  36    Sonar_6         EQU     103
  0033                  37    Sonar_3         EQU     51
  001A                  38    Sonar_1         EQU     26
                        39
8200                    40            org     8200h
8200                    41    main:
                        42            ;enable A/D operation
8200 759710             43            mov     ACON,#010h
                        44
                        45            ;STP_GO is cleared to stop motor
                        46            ;FOR_REV is set to go in reverse
                        47
                        48            ;initalize the motor and steering PWM waveforms
8203 7590FF             49            mov     P1,#0FFH        ;must set all bits on port1
to 0
8206 78B0               50            mov     PWM_MOT,#FWD_LO   ;set motor for 1/8 speed
8208 88FC               51            mov     CCAP2h,PWM_MOT
820A 793C               52            mov     PWM_STR,#03Ch   ;set steering servo to center
820C 89FB               53            mov     CCAP1h,PWM_STR
820E C2F8               54            clr     FOR_REV           ;set relays for FORWARD
8210 C2F9               55            clr     STP_GO          ;block PWM signal to motor
8212 1282F3             56            call    PCA_INIT
                        57
8215 D2F9               58            setb    STP_GO
```

```
8217 1282FF           59    Scan: call    lngDelay
                      60
821A C3               61    L_Rear: clr     C             ; Clear carry bit
821B E594             62            mov     A,AD1         ; get the left rear IR
821D 9464             63            subb    A,#100        ; subtract threshold from value
821F 402E             64            jc      L_Front       ; jump if not too close
                      65
8221                  66    rev2for:
8221 30F929           67            jnb     STP_GO,stp_rev2  ;already stopped
8224 30F826           68            jnb     FOR_REV,stp_rev2 ;or going forward
8227 C2F9             69            clr     STP_GO           ;stop motor
8229 C2F8             70            clr     FOR_REV          ;place in forward
822B 793F             71            mov     PWM_STR,#Straight ;center wheels
822D 89FB             72            mov     CCAP1H,PWM_STR    ;
822F 1282FF           73            call    lngDelay
8232 78B0             74            mov     PWM_MOT,#FWD_LO  ;set motor for 1/8 speed
8234 88FC             75            mov     CCAP2h,PWM_MOT
8236 D2F9             76            setb    STP_GO           ;start motor
8238 128314           77            call    PRINT
823B 466F7277         78            db      'Forward from IR\n',0,
823F 61726420
8243 66726F6D
8247 2049525C
824B 6E00
824D                  79    stp_rev2:
824D 80C8             80            jmp     Scan
                      81
                      82
824F                  83    L_Front:
824F C3               84            clr     C             ; Clear carry bit
8250 E5A4             85            mov     A,AD2         ; get the left front IR
8252 9464             86            subb    A,#100        ; subtract threshold from value
8254 4003             87            jc      R_Front       ; jump if not too close
                      88
8256 028260           89            jmp     for2rev
                      90
8259                  91    R_Front:
8259 C3               92            clr     C             ; Clear carry bit
825A E5B4             93            mov     A,AD3         ; get the left front IR
825C 9464             94            subb    A,#100        ; subtract threshold from value
825E 402E             95            jc      Sonar         ; jump if not too close
                      96
8260                  97    for2Rev:
8260 30F929           98            jnb     STP_GO,stp_rev0         ;already stopped
8263 20F826           99            jb      FOR_REV,stp_rev0        ;or backing
8266 C2F9            100            clr     STP_GO                  ;stop motor
8268 D2F8            101            setb    FOR_REV                 ;reverse it
826A 7931            102            mov     PWM_STR,#031h           ;turn wheels to right
826C 89FB            103            mov     CCAP1H,PWM_STR          ;
826E 1282FF          104            call    lngDelay                ;Give it 1/2 a second
8271 7880            105            mov     PWM_MOT,#REV_LO         ;set motor for 1/8
speed
8273 88FC            106            mov     CCAP2h,PWM_MOT
8275 D2F9            107            setb    STP_GO                  ;start motor
8277 128314          108            call    PRINT
827A 4261636B        109            db      'Backing from IR\n',0,
827E 696E6720
8282 66726F6D
8286 2049525C
```

```
SON_SPD
PAGE 3

828A 6E00
828C                    110    stp_rev0:
828C 8089               111            jmp     Scan
                        112
828E C3                 113    Sonar:  clr     C
828F E584               114            mov     A,AD0           ; Get the sonar value
8291 94F5               115            subb    A,#Sonar_15     ; subtract threshold from
value
8293 4016               116            jc      Ahd_hlf         ; jump if distance is < 15 ft
                        117
8295 78A0               118            mov     PWM_MOT,#FWD_HI  ;set motor for High speed
8297 88FC               119            mov     CCAP2h,PWM_MOT
8299 128314             120            call    PRINT
829C 41686561           121            db      'Ahead Full\n',0,
82A0 64204675
82A4 6C6C5C6E
82A8 00
                        122
82A9 4117               123            jmp     Scan
                        124
82AB                    125    Ahd_hlf:
82AB C3                 126            clr     C
82AC E584               127            mov     A,AD0           ; Get the sonar value
82AE 949B               128            subb    A,#Sonar_9      ; subtract threshold from
value
82B0 401B               129            jc      Ahd_slow        ; jump if distance is < 9 ft
82B2 B8A0AB             130            cjne    PWM_MOT,#FWD_HI,for2Rev
                        131                                    ; If motor is in high gear
                        132                                    ; turn it off
                        133
82B5 78A0               134            mov     PWM_MOT,#FWD_MED  ;set motor for Medium speed
82B7 88FC               135            mov     CCAP2h,PWM_MOT
82B9 128314             136            call    PRINT
82BC 41686561           137            db      'Ahead Medium\n',0,
82C0 64204D65
82C4 6469756D
82C8 5C6E00
                        138
82CB 4117               139            jmp     Scan
                        140
82CD                    141    Ahd_slow:
82CD C3                 142            clr     C
82CE E584               143            mov     A,AD0           ; Get the sonar value
82D0 9433               144            subb    A,#Sonar_3      ; subtract threshold from
value
82D2 408C               145            jc      for2rev         ; jump if distance is < 3 ft
82D4 B8A089             146            cjne    PWM_MOT,#FWD_HI,for2rev
                        147                                    ; If motor is in high gear
                        148                                    ; turn it off
                        149
82D7 78B0               150            mov     PWM_MOT,#FWD_LO  ;set motor for Low speed
82D9 88FC               151            mov     CCAP2h,PWM_MOT
82DB 128314             152            call    PRINT
82DE 41686561           153            db      'Ahead Slow\n',0,
82E2 6420536C
82E6 6F775C6E
82EA 00
                        154
82EB 4117               155            jmp     Scan
                        156
82ED                    157    All_Stop:
```

```
82ED C2F9            158            clr     STP_GO
82EF 4117            159            jmp     Scan
                     160
                     161
                     162
                     163
                     164
82F1 80FE            165    wait:   jmp     wait
                     166
                     167
                     168
                     169            ;initializes the programmable counter array for the
motor
                     170            ;and the steering servo
82F3                 171    PCA_INIT:
82F3 75D906          172            mov     CMOD,#006h      ;set PCA frequency to
external
82F6 75DB42          173            mov     CCAPM1,#42h     ;put PCA module 1 into PWM
mode
82F9 75DC42          174            mov     CCAPM2,#42h     ;put PCA module 0 into PWM
mode
82FC D2DE            175            setb    CR              ;start the PCA clock
82FE 22              176            ret
                     177
82FF                 178    lngDelay:
                  =1 179            $INCLUDE(lngDelay.inc)
82FF 7D0A         =1 180            mov   R5,#00Ah
8301 7E35         =1 181    delay2: mov     R6,#035h
8303 7FFF         =1 182    delay3: mov     R7,#0FFh
8305 DFFE         =1 183    delay4: djnz    R7, delay4              ;wait >500msec
8307 DEFA         =1 184            djnz    R6, delay3
8309 DDF6         =1 185            djnz    R5, delay2
830B 22           =1 186            ret
                     187
830C                 188    cout:
                  =1 189            $INCLUDE(cout.inc)
830C 3099FD       =1 190            JNB     TI,$            ;Wait until transmission
completed.
830F C299         =1 191            CLR     TI              ;Clear interrupt flag.
8311 F599         =1 192            MOV     SBUF,A          ;Write out character.
8313 22           =1 193            RET
                     194
8314                 195    PRINT:
                  =1 196            $INCLUDE(PRINT.inc)
8314 C0E0         =1 197            PUSH    ACC      ; push the A and DPTR registers so
their contents
8316 C083         =1 198            PUSH    DPH      ; don't get lost in case they are
being used
8318 C082         =1 199            PUSH    DPL      ; before calling this function.
                  =1 200
                  =1 201    ; move the stack pointer down low in order to pop the first
string's byte
                  =1 202    ; address
831A E581         =1 203            MOV     A,SP
831C 24FD         =1 204            ADD     A,#0FDH
831E F581         =1 205            MOV     SP,A
8320 D083         =1 206            POP     DPH
8322 D082         =1 207            POP     DPL
                  =1 208
                  =1 209    ; send (PRINT) the string back to the PC computer
8324              =1 210    READ_TX:
8324 E4           =1 211            CLR     A
8325 93           =1 212            MOVC    A,@A+DPTR
8326 A3           =1 213            INC     DPTR
                  =1 214
8327 B45C0C       =1 215            CJNE    A,#'\',SND
```

```
                =1   216
832A E4         =1   217            CLR     A
832B 93         =1   218            MOVC    A,@A+DPTR
832C A3         =1   219            INC     DPTR
                =1   220
832D B46E06     =1   221            CJNE    A,#'n',SND
                =1   222
8330 740D       =1   223            MOV     A,#0DH
8332 710C       =1   224            ACALL   cout
8334 740A       =1   225            MOV     A,#0AH
                =1   226
8336 710C       =1   227    SND:    ACALL   cout
8338 B400E9     =1   228            CJNE    A,#0,READ_TX
                =1   229
                =1   230    ; place the new return addres in stack
833B C082       =1   231            PUSH    DPL
833D C083       =1   232            PUSH    DPH
                =1   233
                =1   234    ; move the stack pointer up high in order to pop the A and
DPTR registers
833F E581       =1   235            MOV     A,SP
8341 2403       =1   236            ADD     A,#3
8343 F581       =1   237            MOV     SP,A
8345 D082       =1   238            POP     DPL
8347 D083       =1   239            POP     DPH
8349 D0E0       =1   240            POP     ACC
834B 22         =1   241            RET
                     242
834C                 243    SEND_BYTE:
                =1   244            $INCLUDE(SEND_BYT.inc)
834C 3099FD     =1   245            JNB    TI, SEND_BYTE           ; WAIT FOR THE TI FLAG
TO SET
834F C299       =1   246            CLR             TI                              ;
CLEAR THE TI FLAG ONCE SENT
8351 F599       =1   247                    MOV     SBUF,A                  ; SEND THE
CONTENTS OF REGISTER A
8353 22         =1   248            RET                              ; RETURN
                =1   249
;*************************************************************************
                =1   250
                =1   251
                =1   252
                     253
                     254
                     255
                     256            end

VERSION 1.2h ASSEMBLY COMPLETE, 0 ERRORS FOUND
```

ACC.	D	ADDR	00E0H	PREDEFINED
ACON	D	ADDR	0097H	PREDEFINED
AD0.	D	ADDR	0084H	PREDEFINED
AD1.	D	ADDR	0094H	PREDEFINED
AD2.	D	ADDR	00A4H	PREDEFINED
AD3.	D	ADDR	00B4H	PREDEFINED
AHD_HLF.	C	ADDR	82ABH	
AHD_SLOW	C	ADDR	82CDH	
ALL_STOP	C	ADDR	82EDH	NOT USED
CCAP1H	D	ADDR	00FBH	PREDEFINED
CCAP2H	D	ADDR	00FCH	PREDEFINED
CCAPM1	D	ADDR	00DBH	PREDEFINED
CCAPM2	D	ADDR	00DCH	PREDEFINED
CMOD	D	ADDR	00D9H	PREDEFINED
COUT	C	ADDR	830CH	
CR	B	ADDR	00DEH	PREDEFINED
DATA_BYTE.		REG4		NOT USED
DELAY2	C	ADDR	8301H	
DELAY3	C	ADDR	8303H	
DELAY4	C	ADDR	8305H	
DPH.	D	ADDR	0083H	PREDEFINED
DPL.	D	ADDR	0082H	PREDEFINED
FOR2REV.	C	ADDR	8260H	
FOR_REV.		NUMB	00F8H	
FULL_LEFT.		NUMB	0030H	NOT USED
FULL_RIGHT		NUMB	004DH	NOT USED
FWD_HI		NUMB	00A0H	
FWD_LO		NUMB	00B0H	
FWD_MED.		NUMB	00A0H	
HALF_LEFT.		NUMB	0037H	NOT USED
HALF_RIGHT		NUMB	0045H	NOT USED
LNGDELAY	C	ADDR	82FFH	
L_FRONT.	C	ADDR	824FH	
L_REAR	C	ADDR	821AH	NOT USED
MAIN	C	ADDR	8200H	NOT USED
MILKYWAY		REG1		NOT USED
P1	D	ADDR	0090H	PREDEFINED
P5	D	ADDR	00F8H	PREDEFINED
PCA_INIT	C	ADDR	82F3H	
PRINT.	C	ADDR	8314H	
PWM_MOT.		REG0		
PWM_STR.		REG1		
QTR_LEFT		NUMB	003AH	NOT USED
QTR_RIGHT.		NUMB	0042H	NOT USED
READ_TX.	C	ADDR	8324H	
REV2FOR.	C	ADDR	8221H	NOT USED
REV_LO		NUMB	0080H	
R_FRONT.	C	ADDR	8259H	
SBUF	D	ADDR	0099H	PREDEFINED
SCAN	C	ADDR	8217H	
SEND_BYTE.	C	ADDR	834CH	
SND.	C	ADDR	8336H	
SONAR.	C	ADDR	828EH	
SONAR_1.		NUMB	001AH	NOT USED
SONAR_12		NUMB	00D0H	NOT USED
SONAR_15		NUMB	00F5H	
SONAR_3.		NUMB	0033H	
SONAR_6.		NUMB	0067H	NOT USED

```
SONAR_9. . . . . .    NUMB  009BH
SP . . . . . .  D ADDR  0081H  PREDEFINED
STP_GO . . . . . .    NUMB  00F9H
STP_REV0 . . . . . .  C ADDR  828CH
STP_REV2 . . . . . .  C ADDR  824DH
STRAIGHT . . . . . .    NUMB  003FH
TI . . . . . .  B ADDR  0099H  PREDEFINED
WAIT . . . . . .  C ADDR  82F1H
```

This code is for the precision navigation compass. It utilizes the 8051 interrupt system.

```
CMPS_INT
PAGE 1

                        1           $MOD51GB
                        2
  00C0                  3           EOC     equ     P4.0
  00C1                  4           SS      equ     P4.1
  00C2                  5           MS      equ     P4.2
  00C3                  6           RESET   equ     P4.3
  00C4                  7           SCLK    equ     P4.4
  00C5                  8           SDO     equ     P4.5
  00C6                  9           POLL    equ     P4.6
  00C7                 10           POW     equ     P4.7
                       11
                       12           CompCount  equ  R7
                       13           DataByte   equ  R4
                       14
8000                   15           org     8000h
8000 028200            16           jmp     8200h
                       17
802B                   18           org     802bh
802B 0282AD            19           jmp     T2_ISR
                       20
8100                   21           org     8100h
8100 0000              22           db      0,0
8200                   23           org     8200h
8200 128285            24           call    Comp_Init       ;Initialize the compass
8203 128389            25           call    Print
8206 54686520          26           db      'The compass is initialized\n',0,
820A 636F6D70
820E 61737320
8212 69732069
8216 6E697469
821A 616C697A
821E 65645C6E
8222 00
8223 1282A2            27           call    T2_Init         ;Initialize T2
8226 128389            28           call    Print
8229 54696D65          29           db      'Timer 2 is initialized\n',0,
822D 72203220
8231 69732069
8235 6E697469
8239 616C697A
823D 65645C6E
8241 00
                       30
8242 D2C2              31           setb    MS
8244 128389            32    ComRd: call    Print
8247 5C6E2020          33           db      '\n     Press the g key to get a compass
heading\n',0,
824B 20202050
824F 72657373
8253 20746865
8257 2067206B
825B 65792074
825F 6F206765
8263 74206120
8267 636F6D70
826B 61737320
826F 68656164
8273 696E675C
8277 6E00
```

```
CMPS_INT
PAGE 2

8279 128379          34    start:  call    cin
827C B467FA          35            cjne    A,#'g',start
                     36
827F D2AD            37            setb    ET2             ;enable Timer 2 interrupt
8281 D2AF            38            setb    EA              ;enable interrupts
                     39
8283 80FE            40    wait:   jmp     wait
                     41
8285                 42    Comp_Init:
                     43            ;set all the bits on the compass
8285 D2C7            44            setb    POW             ;turn power off
8287 D2C6            45            setb    POLL            ;set the poll bit
8289 D2C1            46            setb    SS              ;set the slave select bit
828B D2C3            47            setb    RESET           ;set the NOT Reset line
828D D2C4            48            setb    SCLK            ;set the shift clock
828F D2C5            49            setb    SDO             ;setting SDO will set that
pin on the
                     50                                    ;8051 as an input
8291 D2C0            51            setb    EOC             ;same for EOC
                     52
8293 C2C7            53            clr     POW             ;turn power on
                     54
                     55            ;Initialize the counter for the compass process
controller
8295 7F00            56            mov     CompCount,#0
8297 C2C3            57            clr     RESET           ;reset compass
8299 128363          58            call    Delay10         ;wait 10msecs
829C D2C3            59            setb    RESET           ;clear reset
829E 12836C          60            call    lngDelay        ;wait .5secs
                     61
                     62
82A1 22              63            ret
                     64
82A2                 65    T2_Init:
                     66            ;load the timer to run the compass
82A2 C2CA            67            clr     TR2             ;turn Timer 2 off
82A4 75CDB7          68            mov     TH2,#0B7h       ;load the T2 high byte
82A7 75CCCB          69            mov     TL2,#0CBh       ;load the T2 low byte
82AA D2CA            70            setb    TR2             ;turn Timer 2 on
82AC 22              71            ret
                     72
82AD                 73    T2_ISR:
                =1   74            $INCLUDE(T2_ISR.INC)
82AD C2CF       =1   75            clr     TF2             ;Clear the T2 interrupt flag
82AF C2CA       =1   76            clr     TR2             ;Stop the timer
82B1 75CDB7     =1   77            mov     TH2,#0B7h       ;Load the timer with the
value
82B4 75CCCB     =1   78            mov     TL2,#0CBh       ;needed for a 10msec interval
82B7 D2CA       =1   79            setb    TR2             ;turn timer back on
82B9 0F         =1   80            inc     CompCount       ;Increment the process
counter
                =1   81
82BA BF0103     =1   82            cjne    CompCount,#1,temp0
82BD 0282FB     =1   83            jmp     CLR_POLL
82C0 BF0203     =1   84    temp0:  cjne    CompCount,#2,temp1
82C3 028300     =1   85            jmp     SET_POLL
82C6 BF0303     =1   86    temp1:  cjne    CompCount,#3,temp2
82C9 028305     =1   87            jmp     WAIT_EOC
82CC BF0403     =1   88    temp2:  cjne    CompCount,#4,temp3
82CF 02830D     =1   89            jmp     CLR_SS
82D2 BF0503     =1   90    temp3:  cjne    CompCount,#5,temp4
82D5 028312     =1   91            jmp     READ_SDO
```

```
CMPS_INT
PAGE 3

82D8 BF0603     =1    92    temp4:  cjne    CompCount,#6,temp5
82DB 028312     =1    93            jmp     READ_SDO
82DE 128389     =1    94    temp5:  call    Print
82E1 54686973   =1    95            db      'This didnt work!! \n\n',0,
82E5 20646964
82E9 6E742077
82ED 6F726B21
82F1 21205C6E
82F5 5C6E00
82F8 02835D     =1    96            jmp     exit_T2
                =1    97
82FB            =1    98    CLR_POLL:
                =1    99
82FB C2C6       =1   100            clr     POLL              ;clr POLL
82FD 02833E     =1   101            jmp     end_T2
                =1   102
8300             =1   103    SET_POLL:
8300 D2C6       =1   104            setb    POLL              ;set POLL
8302 02833E     =1   105            jmp     end_T2
                =1   106
8305            =1   107    WAIT_EOC:
8305 20C036     =1   108            jb      EOC,end_T2        ;wait for the EOC flag to
be
8308 7F02       =1   109            mov     CompCount,#2      ;set
830A 02833E     =1   110            jmp     end_T2
                =1   111
830D C2C1       =1   112    CLR_SS: clr     SS                ; set slave select for data
recovery
830F 02833E     =1   113            jmp     end_T2
                =1   114
                =1   115
8312 7E08       =1   116    READ_SDO:  mov     R6,#08
                =1   117
8314            =1   118    ReadSDO:
8314 C2C4       =1   119            clr     SCLK
8316 12835E     =1   120            call    clkDelay
8319 D2C4       =1   121            setb    SCLK
831B 7420       =1   122            mov     A, #020h        ;set A.5
831D 55C0       =1   123            anl     A,P4            ;compare A.5 with P4.5(SDO)
831F 6006       =1   124            jz      zero            ;jump if P4.5(EOC) is not set
                =1   125
8321 EC         =1   126            mov     A, DataByte     ;load R4 into A
8322 D3         =1   127            setb    C               ;set the carry bit
8323 33         =1   128            rlc     A               ;roll A left with carry
8324 FC         =1   129            mov     DataByte, A     ;put contents of A back into
R4
8325 612B       =1   130            ajmp    rdDone          ; done with read
                =1   131
8327 EC         =1   132    zero:   mov     A, DataByte     ;load R4 into A
8328 C3         =1   133            clr     C               ;clear the carry bit
8329 33         =1   134            rlc     A               ;roll A left with carry
832A FC         =1   135            mov     DataByte, A     ;put contents of A back into
R4
                =1   136
832B 12835E     =1   137    rdDone: call   clkdelay         ;delay for low clock
832E DEE4       =1   138            djnz   R6, ReadSDO      ;this loop will execute 8
times
                =1   139
8330 BF050B     =1   140    done8:  cjne   CompCount,#5,end_T2
                =1   141                                   ;SDO_READ will execute when
                =1   142                                   ;CompCount = 5 & 6
                =1   143                                   ;done8 will only run when
CompCount=5
8333 EC         =1   144            mov    A,DataByte       ;mov Data to A
```

```
8334 758381     =1   145            mov    DPH,#081h        ;first 8 bits will be saved
8337 758200     =1   146            mov    DPL,#000h        ;at 8100h
833A F0         =1   147            movx   @DPTR,A          ;
833B 02833E     =1   148            jmp    end_T2           ;end the routine
                =1   149
                =1   150
833E BF061C     =1   151    end_T2: cjne    CompCount,#6,exit_T2
8341 D2C1       =1   152            setb    SS              ;Slave select off
8343 EC         =1   153            mov     A,DataByte
8344 758381     =1   154            mov     DPH,#081h
8347 758201     =1   155            mov     DPL,#001h
834A F0         =1   156            movx    @DPTR,A
834B 7F00       =1   157            mov     CompCount,#00
834D 758200     =1   158            mov     DPL,#00h
8350 E0         =1   159            movx    A,@DPTR
8351 F583       =1   160            mov     DPH,A
8353 8C82       =1   161            mov     DPL,DATABYTE
8355 1283C1     =1   162            call    pint16u
8358 128389     =1   163            call    print
835B 5C00       =1   164            db      '\',0,
                =1   165
                =1   166
                =1   167
835D            =1   168    exit_T2:
835D 32         =1   169            reti
                     170
835E 7873            171    clkdelay: mov   R0,#073h
8360 D8FE            172    clk:    djnz    R0,clk
8362 22              173            ret
                     174
8363 7825            175    delay10: mov    R0,#025h
8365 79FF            176    delay:  mov     R1,#0FFh
8367 D9FE            177    delay1: djnz    R1, delay1              ;wait >10 msec
8369 D8FA            178            djnz    R0, delay
836B 22              179            ret
                     180
                     181
836C 780A            182    lngDelay: mov   R0,#00Ah
836E 7935            183    delay2: mov     R1,#035h
8370 7AFF            184    delay3: mov     R2,#0FFh
8372 DAFE            185    delay4: djnz    R2, delay4              ;wait >500msec
8374 D9FA            186            djnz    R1, delay3
8376 D8F6            187            djnz    R0, delay2
8378 22              188            ret
                     189
8379                 190    cin:
                =1   191            $INCLUDE(cin.inc)
8379 3098FD     =1   192            JNB     RI,$             ; WAIT FOR THE RI FLAG TO
SET
837C C298       =1   193            CLR     RI               ; CLEAR THE RECEIVE FLAG
ONCE RECIEVED
837E E599       =1   194            MOV     A,SBUF           ; PLACE RECEIVED BYTE IN
REGISTER A
8380 22         =1   195            RET
                     196
8381                 197    cout:
                =1   198            $INCLUDE(cout.inc)
8381 3099FD     =1   199            JNB     TI,$            ;Wait until transmission
completed.
8384 C299       =1   200            CLR     TI              ;Clear interrupt flag.
8386 F599       =1   201            MOV     SBUF,A          ;Write out character.
8388 22         =1   202            RET
```

```
                        203
8389                    204     PRINT:
                  =1    205             $INCLUDE(Print.inc)
8389 C0E0         =1    206             PUSH    ACC       ; push the A and DPTR registers so
their contents
838B C083         =1    207             PUSH    DPH       ; don't get lost in case they are
being used
838D C082         =1    208             PUSH    DPL       ; before calling this function.
                  =1    209
                  =1    210     ; move the stack pointer down low in order to pop the first
string's byte
                  =1    211     ; address
838F E581         =1    212             MOV     A,SP
8391 24FD         =1    213             ADD     A,#0FDH
8393 F581         =1    214             MOV     SP,A
8395 D083         =1    215             POP     DPH
8397 D082         =1    216             POP     DPL
                  =1    217
                  =1    218     ; send (PRINT) the string back to the PC computer
8399              =1    219     READ_TX:
8399 E4           =1    220             CLR     A
839A 93           =1    221             MOVC    A,@A+DPTR
839B A3           =1    222             INC     DPTR
                  =1    223
839C B45C0C       =1    224             CJNE    A,#'\',SND
                  =1    225
839F E4           =1    226             CLR     A
83A0 93           =1    227             MOVC    A,@A+DPTR
83A1 A3           =1    228             INC     DPTR
                  =1    229
83A2 B46E06       =1    230             CJNE    A,#'n',SND
                  =1    231
83A5 740D         =1    232             MOV     A,#0DH
83A7 7181         =1    233             ACALL   cout
83A9 740A         =1    234             MOV     A,#0AH
                  =1    235
83AB 7181         =1    236     SND:    ACALL   cout
83AD B400E9       =1    237             CJNE    A,#0,READ_TX
                  =1    238
                  =1    239     ; place the new return addres in stack
83B0 C082         =1    240             PUSH    DPL
83B2 C083         =1    241             PUSH    DPH
                  =1    242
                  =1    243     ; move the stack pointer up high in order to pop the A and
DPTR registers
83B4 E581         =1    244             MOV     A,SP
83B6 2403         =1    245             ADD     A,#3
83B8 F581         =1    246             MOV     SP,A
83BA D082         =1    247             POP     DPL
83BC D083         =1    248             POP     DPH
83BE D0E0         =1    249             POP     ACC
83C0 22           =1    250             RET
                        251
83C1                    252     pint16u:
                  =1    253             $INCLUDE(pint16u.inc)
                  =1    254             ;print 16 bit unsigned integer in DPTR, using base
10.
                  =1    255             ;warning, destroys r2, r3, r4, r5, psw.5
83C1 C0E0         =1    256             push    acc
83C3 E8           =1    257             mov     a, r0
83C4 C0E0         =1    258             push    acc
83C6 C2D5         =1    259             clr     psw.5
83C8 AA82         =1    260             mov     r2, dpl
```

```
83CA AB83       =1   261            mov     r3, dph
                =1   262
83CC 7C10       =1   263    pint16a:mov     r4, #16 ;ten-thousands digit
83CE 7D27       =1   264            mov     r5, #39
83D0 911F       =1   265            acall   pint16x
83D2 6007       =1   266            jz      pint16b
83D4 2430       =1   267            add     a, #'0'
83D6 128381     =1   268            lcall   cout
83D9 D2D5       =1   269            setb    psw.5
                =1   270
83DB 7CE8       =1   271    pint16b:mov     r4, #232        ;thousands digit
83DD 7D03       =1   272            mov     r5, #3
83DF 911F       =1   273            acall   pint16x
83E1 7003       =1   274            jnz     pint16c
83E3 30D507     =1   275            jnb     psw.5, pint16d
83E6 2430       =1   276    pint16c:add     a, #'0'
83E8 128381     =1   277            lcall   cout
83EB D2D5       =1   278            setb    psw.5
                =1   279
83ED 7C64       =1   280    pint16d:mov     r4, #100        ;hundreds digit
83EF 7D00       =1   281            mov     r5, #0
83F1 911F       =1   282            acall   pint16x
83F3 7003       =1   283            jnz     pint16e
83F5 30D507     =1   284            jnb     psw.5, pint16f
83F8 2430       =1   285    pint16e:add     a, #'0'
83FA 128381     =1   286            lcall   cout
83FD D2D5       =1   287            setb    psw.5
                =1   288
83FF EA         =1   289    pint16f:mov     a, r2           ;tens digit
8400 ABF0       =1   290            mov     r3, b
8402 75F00A     =1   291            mov     b, #10
8405 84         =1   292            div     ab
8406 7003       =1   293            jnz     pint16g
8408 30D505     =1   294            jnb     psw.5, pint16h
840B 2430       =1   295    pint16g:add     a, #'0'
840D 128381     =1   296            lcall   cout
                =1   297
8410 E5F0       =1   298    pint16h:mov     a, b            ;and finally the ones digit
8412 8BF0       =1   299            mov     b, r3
8414 2430       =1   300            add     a, #'0'
8416 128381     =1   301            lcall   cout
                =1   302
8419 D0E0       =1   303            pop     acc
841B F8         =1   304            mov     r0, a
841C D0E0       =1   305            pop     acc
841E 22         =1   306            ret
                =1   307
                =1   308    ;ok, it's a cpu hog and a nasty way to divide, but this code
                =1   309    ;requires only 21 bytes!  Divides r2-r3 by r4-r5 and leaves
                =1   310    ;quotient in r2-r3 and returns remainder in acc.  If Intel
                =1   311    ;had made a proper divide, then this would be much easier.
                =1   312
841F 7800       =1   313    pint16x:mov     r0, #0
8421 08         =1   314    pint16y:inc     r0
8422 C3         =1   315            clr     c
8423 EA         =1   316            mov     a, r2
8424 9C         =1   317            subb    a, r4
8425 FA         =1   318            mov     r2, a
```

```
8426 EB         =1   319            mov     a, r3
8427 9D         =1   320            subb    a, r5
8428 FB         =1   321            mov     r3, a
8429 50F6       =1   322            jnc     pint16y
842B 18         =1   323            dec     r0
842C EA         =1   324            mov     a, r2
842D 2C         =1   325            add     a, r4
842E FA         =1   326            mov     r2, a
842F EB         =1   327            mov     a, r3
8430 3D         =1   328            addc    a, r5
8431 FB         =1   329            mov     r3, a
8432 E8         =1   330            mov     a, r0
8433 22         =1   331            ret
                     332
                     333            END
```

VERSION 1.2h ASSEMBLY COMPLETE, 0 ERRORS FOUND

ACC.	D	ADDR	00E0H	PREDEFINED
B.	D	ADDR	00F0H	PREDEFINED
CIN.	C	ADDR	8379H	
CLK.	C	ADDR	8360H	
CLKDELAY	C	ADDR	835EH	
CLR_POLL	C	ADDR	82FBH	
CLR_SS	C	ADDR	830DH	
COMPCOUNT.		REG7		
COMP_INIT.	C	ADDR	8285H	
COMRD.	C	ADDR	8244H	NOT USED
COUT	C	ADDR	8381H	
DATABYTE		REG4		
DELAY.	C	ADDR	8365H	
DELAY1	C	ADDR	8367H	
DELAY10.	C	ADDR	8363H	
DELAY2	C	ADDR	836EH	
DELAY3	C	ADDR	8370H	
DELAY4	C	ADDR	8372H	
DONE8.	C	ADDR	8330H	NOT USED
DPH.	D	ADDR	0083H	PREDEFINED
DPL.	D	ADDR	0082H	PREDEFINED
EA	B	ADDR	00AFH	PREDEFINED
END_T2	C	ADDR	833EH	
EOC.		NUMB	00C0H	
ET2.	B	ADDR	00ADH	PREDEFINED
EXIT_T2.	C	ADDR	835DH	
LNGDELAY	C	ADDR	836CH	
MS		NUMB	00C2H	
P4	D	ADDR	00C0H	PREDEFINED
PINT16A.	C	ADDR	83CCH	NOT USED
PINT16B.	C	ADDR	83DBH	
PINT16C.	C	ADDR	83E6H	
PINT16D.	C	ADDR	83EDH	
PINT16E.	C	ADDR	83F8H	
PINT16F.	C	ADDR	83FFH	
PINT16G.	C	ADDR	840BH	
PINT16H.	C	ADDR	8410H	
PINT16U.	C	ADDR	83C1H	
PINT16X.	C	ADDR	841FH	
PINT16Y.	C	ADDR	8421H	
POLL		NUMB	00C6H	
POW.		NUMB	00C7H	
PRINT.	C	ADDR	8389H	
PSW.	D	ADDR	00D0H	PREDEFINED
RDDONE	C	ADDR	832BH	
READSDO.	C	ADDR	8314H	
READ_SDO	C	ADDR	8312H	
READ_TX.	C	ADDR	8399H	
RESET.		NUMB	00C3H	
RI	B	ADDR	0098H	PREDEFINED
SBUF	D	ADDR	0099H	PREDEFINED
SCLK		NUMB	00C4H	
SDO.		NUMB	00C5H	
SET_POLL	C	ADDR	8300H	
SND.	C	ADDR	83ABH	
SP	D	ADDR	0081H	PREDEFINED
SS		NUMB	00C1H	
START.	C	ADDR	8279H	

```
T2_INIT. . . . . .  C ADDR  82A2H
T2_ISR . . . . . .  C ADDR  82ADH
TEMP0. . . . . . .  C ADDR  82C0H
TEMP1. . . . . . .  C ADDR  82C6H
TEMP2. . . . . . .  C ADDR  82CCH
TEMP3. . . . . . .  C ADDR  82D2H
TEMP4. . . . . . .  C ADDR  82D8H
TEMP5. . . . . . .  C ADDR  82DEH
TF2. . . . . . . .  B ADDR  00CFH  PREDEFINED
TH2. . . . . . . .  D ADDR  00CDH  PREDEFINED
TI . . . . . . . .  B ADDR  0099H  PREDEFINED
TL2. . . . . . . .  D ADDR  00CCH  PREDEFINED
TR2. . . . . . . .  B ADDR  00CAH  PREDEFINED
WAIT . . . . . . .  C ADDR  8283H
WAIT_EOC . . . . .  C ADDR  8305H
ZERO . . . . . . .  C ADDR  8327H
```

This is code for the compass that does not utilize the interrupt system.

```
CMPS_TST
PAGE 1

                      1
                      2
                      3           $MOD51GB
                      4
  00C0                5           EOC     equ     P4.0
  00C1                6           SS      equ     P4.1
  00C2                7           MS      equ     P4.2
  00C3                8           RESET   equ     P4.3
  00C4                9           SCLK    equ     P4.4
  00C5               10           SDO     equ     P4.5
  00C6               11           POLL    equ     P4.6
  00C7               12           POW     equ     P4.7
                     13
                     14
8000                 15           org     8000h
                     16
8000 D2C7            17           setb    POW
8002 D2C6            18           setb    POLL
8004 D2C1            19           setb    SS
8006 D2C3            20           setb    RESET
8008 D2C4            21           setb    SCLK
800A D2C5            22           setb    SDO
800C D2C0            23           setb    EOC
                     24
800E C2C7            25           clr     POW             ;turn power on
                     26
8010 1280A7          27           call   delay10         ;wait for it to stabalize
                     28
8013 C2C3            29           clr     RESET           ;reset compass
                     30
8015 1280A7          31           call   delay10        ;wait 10msec
                     32
8018 D2C3            33           setb    RESET           ;clear reset
                     34
801A 1280B9          35           call   lngDelay       ;wait .5secs
                     36
801D D2C2            37           setb    MS
                     38
801F 1280D6          39    ComRd:  call    Print
8022 5C6E2020        40           db      '\n     Press the g key to get a compass
heading\n',0,
8026 20202050
802A 72657373
802E 20746865
8032 2067206B
8036 65792074
803A 6F206765
803E 74206120
8042 636F6D70
8046 61737320
804A 68656164
804E 696E675C
8052 6E00
                     41
8054 1280C6          42    start:  call    cin
8057 B467FA          43           cjne    A,#'g',start
                     44
805A C2C6            45           clr     POLL              ;clr POLL
                     46
```

```
805C 1280A7           47            call   delay10
                      48
805F D2C6             49            setb    POLL              ;set POLL
                      50
8061 30C0FD           51    waitEOC: jnb    EOC, waitEOC    ;wait for EOC to reset
                      52
8064 1280A7           53            call   delay10         ;wait >10msecs
                      54
8067 C2C1             55            clr     SS              ; set slave select for data
recovery
                      56
8069 1280B0           57            call   delay5         ;wait >5msecs
                      58
806C 7D02             59            mov     R5,#02
806E 7E08             60    SDOrd:  mov     R6,#08
                      61
8070 C2C4             62    ReadSDO: clr    SCLK
8072 1280A2           63            call   clkdelay
8075 D2C4             64            setb    SCLK
8077 7420             65            mov     A, #020h          ;set A.5
8079 55C0             66            anl     A,P4            ;compare A.3 with P4.3
807B 6006             67            jz      zero            ;jump if P4.3 is not set
                      68
807D EC               69            mov     A, R4           ;load R4 into A
807E D3               70            setb    C               ;set the carry bit
807F 33               71            rlc     A               ;roll A left with carry
8080 FC               72            mov     R4, A           ;put contents of A back into
R4
8081 0187             73            ajmp    rdDone          ; done with read
                      74
8083 EC               75    zero:   mov     A, R4           ;load R4 into A
8084 C3               76            clr     C               ;clear the carry bit
8085 33               77            rlc     A               ;roll A left with carry
8086 FC               78            mov     R4, A           ;put contents of A back into
R4
                      79
8087 1280A2           80    rdDone: call   clkdelay                ;delay for low clock
808A DEE4             81            djnz    R6, ReadSDO
                      82
808C DD0C             83    done8:  djnz    R5, hlfdelay    ; Read second half of data
                      84
808E D2C1             85            setb    SS              ; Slave select off
                      86
8090 8F83             87            mov     DPH, R7
8092 8C82             88            mov     DPL, R4
8094 128117           89            call    pint16u
                      90
8097 02801F           91            ljmp    ComRd
                      92
                      93
809A 1280B0           94    hlfdelay: call delay5         ;Wait >5msecs
809D EC               95            mov     A, R4           ;transfer MSB to R7
809E FF               96            mov     R7, A           ;      "    "
809F 02806E           97            ljmp    SDOrd          ;Start next 8 bits
                      98
80A2 7873             99    clkdelay: mov   R0,#073h
80A4 D8FE            100    clk:    djnz    R0,clk
80A6 22              101            ret
                     102
80A7 7825            103    delay10: mov    R0,#025h
80A9 79FF            104    delay:  mov     R1,#0FFh
```

```
80AB D9FE           105    delay1: djnz    R1, delay1              ;wait >10 msec
80AD D8FA           106            djnz    R0, delay
80AF 22             107            ret
                    108
80B0 7812           109    delay5: mov    R0,#012h
80B2 79FF           110    del:  mov     R1,#0FFh
80B4 D9FE           111    del1: djnz    R1, del1             ;wait >5 msec
80B6 D8FA           112            djnz    R0, del
80B8 22             113            ret
                    114
                    115
                    116
80B9 780A           117    lngDelay: mov   R0,#00Ah
80BB 7935           118    delay2: mov     R1,#035h
80BD 7AFF           119    delay3: mov     R2,#0FFh
80BF DAFE           120    delay4: djnz    R2, delay4              ;wait >500msec
80C1 D9FA           121            djnz    R1, delay3
80C3 D8F6           122            djnz    R0, delay2
80C5 22             123            ret
                    124
                    125
80C6 3098FD         126    cin:    JNB     RI,$             ; WAIT FOR THE RI FLAG TO
SET
80C9 C298           127            CLR     RI               ; CLEAR THE RECEIVE FLAG
ONCE RECIEVED
80CB E599           128            MOV     A,SBUF           ; PLACE RECEIVED BYTE IN
REGISTER A
80CD 22             129            RET
                    130
                    131
                    132
80CE 3099FD         133    cout:   JNB     TI,$            ;Wait until transmission
completed.
80D1 C299           134            CLR     TI              ;Clear interrupt flag.
80D3 F599           135            MOV     SBUF,A          ;Write out character.
80D5 22             136            RET
                    137
80D6 C0E0           138    PRINT:  PUSH    ACC      ; push the A and DPTR registers so
their contents
80D8 C083           139            PUSH    DPH      ; don't get lost in case they are
being used
80DA C082           140            PUSH    DPL      ; before calling this function.
                    141
                    142    ; move the stack pointer down low in order to pop the first
string's byte
                    143    ; address
80DC E581           144            MOV     A,SP
80DE 24FD           145            ADD     A,#0FDH
80E0 F581           146            MOV     SP,A
80E2 D083           147            POP     DPH
80E4 D082           148            POP     DPL
                    149
                    150    ; send (PRINT) the string back to the PC computer
80E6                151    READ_TX:
80E6 E4             152            CLR     A
80E7 93             153            MOVC    A,@A+DPTR
80E8 A3             154            INC     DPTR
                    155
80E9 B45C0C         156            CJNE    A,#'\',SND
                    157
80EC E4             158            CLR     A
80ED 93             159            MOVC    A,@A+DPTR
80EE A3             160            INC     DPTR
                    161
80EF B46E06         162            CJNE    A,#'n',SND
```

```
                    163
80F2 740D           164             MOV     A,#0DH
80F4 11CE           165             ACALL   cout
80F6 740A           166             MOV     A,#0AH
                    167
80F8 11CE           168     SND:    ACALL   cout
80FA B400E9         169             CJNE    A,#0,READ_TX
                    170
                    171     ; place the new return addres in stack
80FD C082           172             PUSH    DPL
80FF C083           173             PUSH    DPH
                    174
                    175     ; move the stack pointer up high in order to pop the A and
DPTR registers
8101 E581           176             MOV     A,SP
8103 2403           177             ADD     A,#3
8105 F581           178             MOV     SP,A
8107 D082           179             POP     DPL
8109 D083           180             POP     DPH
810B D0E0           181             POP     ACC
810D 22             182             RET
                    183
810E 540F           184     UTIL_BINTOASC:  anl     a,#00fh                 ; Keep Only
Low Bits
8110 2490           185                     add     a,#090h                 ; Add 144
8112 D4             186                     da      a                       ; Decimal
Adjust
8113 3440           187                     addc    a,#040h                 ; Add 64
8115 D4             188                     da      a                       ; Decimal
Adjust
8116 22             189                     ret                             ; Return To
Caller
                    190
                    191             ;print 16 bit unsigned integer in DPTR, using base
10.
8117                192     pint16u:        ;warning, destroys r2, r3, r4, r5, psw.5
8117 C0E0           193             push    acc
8119 E8             194             mov     a, r0
811A C0E0           195             push    acc
811C C2D5           196             clr     psw.5
811E AA82           197             mov     r2, dpl
8120 AB83           198             mov     r3, dph
                    199
8122 7C10           200     pint16a:mov     r4, #16 ;ten-thousands digit
8124 7D27           201             mov     r5, #39
8126 3175           202             acall   pint16x
8128 6007           203             jz      pint16b
812A 2430           204             add     a, #'0'
812C 1280CE         205             lcall   cout
812F D2D5           206             setb    psw.5
                    207
8131 7CE8           208     pint16b:mov     r4, #232        ;thousands digit
8133 7D03           209             mov     r5, #3
8135 3175           210             acall   pint16x
8137 7003           211             jnz     pint16c
8139 30D507         212             jnb     psw.5, pint16d
813C 2430           213     pint16c:add     a, #'0'
813E 1280CE         214             lcall   cout
8141 D2D5           215             setb    psw.5
                    216
8143 7C64           217     pint16d:mov     r4, #100        ;hundreds digit
8145 7D00           218             mov     r5, #0
8147 3175           219             acall   pint16x
8149 7003           220             jnz     pint16e
```

```
814B 30D507          221            jnb     psw.5, pint16f
814E 2430            222    pint16e:add     a, #'0'
8150 1280CE          223            lcall   cout
8153 D2D5            224            setb    psw.5
                     225
8155 EA              226    pint16f:mov     a, r2           ;tens digit
8156 ABF0            227            mov     r3, b
8158 75F00A          228            mov     b, #10
815B 84              229            div     ab
815C 7003            230            jnz     pint16g
815E 30D505          231            jnb     psw.5, pint16h
8161 2430            232    pint16g:add     a, #'0'
8163 1280CE          233            lcall   cout
                     234
8166 E5F0            235    pint16h:mov     a, b            ;and finally the ones digit
8168 8BF0            236            mov     b, r3
816A 2430            237            add     a, #'0'
816C 1280CE          238            lcall   cout
                     239
816F D0E0            240            pop     acc
8171 F8              241            mov     r0, a
8172 D0E0            242            pop     acc
8174 22              243            ret
                     244
                     245    ;ok, it's a cpu hog and a nasty way to divide, but this code
                     246    ;requires only 21 bytes!  Divides r2-r3 by r4-r5 and leaves
                     247    ;quotient in r2-r3 and returns remainder in acc.  If Intel
                     248    ;had made a proper divide, then this would be much easier.
                     249
8175 7800            250    pint16x:mov     r0, #0
8177 08              251    pint16y:inc     r0
8178 C3              252            clr     c
8179 EA              253            mov     a, r2
817A 9C              254            subb    a, r4
817B FA              255            mov     r2, a
817C EB              256            mov     a, r3
817D 9D              257            subb    a, r5
817E FB              258            mov     r3, a
817F 50F6            259            jnc     pint16y
8181 18              260            dec     r0
8182 EA              261            mov     a, r2
8183 2C              262            add     a, r4
8184 FA              263            mov     r2, a
8185 EB              264            mov     a, r3
8186 3D              265            addc    a, r5
8187 FB              266            mov     r3, a
8188 E8              267            mov     a, r0
8189 22              268            ret
                     269
                     270            END
```

VERSION 1.2h ASSEMBLY COMPLETE, 0 ERRORS FOUND

ACC.	D	ADDR	00E0H	PREDEFINED
B.	D	ADDR	00F0H	PREDEFINED
CIN.	C	ADDR	80C6H	
CLK.	C	ADDR	80A4H	
CLKDELAY	C	ADDR	80A2H	
COMRD.	C	ADDR	801FH	
COUT	C	ADDR	80CEH	
DEL.	C	ADDR	80B2H	
DEL1	C	ADDR	80B4H	
DELAY.	C	ADDR	80A9H	
DELAY1	C	ADDR	80ABH	
DELAY10.	C	ADDR	80A7H	
DELAY2	C	ADDR	80BBH	
DELAY3	C	ADDR	80BDH	
DELAY4	C	ADDR	80BFH	
DELAY5	C	ADDR	80B0H	
DONE8.	C	ADDR	808CH	NOT USED
DPH.	D	ADDR	0083H	PREDEFINED
DPL.	D	ADDR	0082H	PREDEFINED
EOC.		NUMB	00C0H	
HLFDELAY	C	ADDR	809AH	
LNGDELAY	C	ADDR	80B9H	
MS		NUMB	00C2H	
P4	D	ADDR	00C0H	PREDEFINED
PINT16A.	C	ADDR	8122H	NOT USED
PINT16B.	C	ADDR	8131H	
PINT16C.	C	ADDR	813CH	
PINT16D.	C	ADDR	8143H	
PINT16E.	C	ADDR	814EH	
PINT16F.	C	ADDR	8155H	
PINT16G.	C	ADDR	8161H	
PINT16H.	C	ADDR	8166H	
PINT16U.	C	ADDR	8117H	
PINT16X.	C	ADDR	8175H	
PINT16Y.	C	ADDR	8177H	
POLL		NUMB	00C6H	
POW.		NUMB	00C7H	
PRINT.	C	ADDR	80D6H	
PSW.	D	ADDR	00D0H	PREDEFINED
RDDONE	C	ADDR	8087H	
READSDO.	C	ADDR	8070H	
READ_TX.	C	ADDR	80E6H	
RESET.		NUMB	00C3H	
RI	B	ADDR	0098H	PREDEFINED
SBUF	D	ADDR	0099H	PREDEFINED
SCLK		NUMB	00C4H	
SDO.		NUMB	00C5H	
SDORD.	C	ADDR	806EH	
SND.	C	ADDR	80F8H	
SP	D	ADDR	0081H	PREDEFINED
SS		NUMB	00C1H	
START.	C	ADDR	8054H	
TI	B	ADDR	0099H	PREDEFINED
UTIL_BINTOASC.	C	ADDR	810EH	NOT USED
WAITEOC.	C	ADDR	8061H	
ZERO	C	ADDR	8083H	

This program will test the A/D inputs. In this case that includes all three IR's and the sonar.

```
AD0123
PAGE 1

                      1
                      2    ; --- WARNING:  DO NOT APPLY MORE THAN 5 VOLTS TO ANY OF THE
A/D INPUTS
                      3
                      4            $MOD51GB
                      5
8000                  6            ORG     8000H
                      7
8000 128033           8    START:  CALL   CIN
                      9    ;        MOV     P1,#000h
8003 B473FA          10            CJNE    A,#'s',START
                     11
                     12
                     13    ;                       AIF            A/D Interrupt Flag
                     14    ;                       |ACE           A/D Conversion Enable
                     15    ;                       ||ACS1         A/D Channel Select 1
                     16    ;                       |||ACS0        A/D Channel Select 0
                     17    ;                       ||||AIM        A/D Input Mode
                     18    ;                       |||||ATM       A/D Triger Mode
                     19    ;                       ||||||
                     20    ;                       ||||||
8006 759710          21            MOV     ACON,#00010000B   ; ENABLE A/D OPERATION
                     22
8009                 23    WAIT:
                     24
8009 7401            25            MOV     A,#1
800B 113B            26            ACALL   COUT          ; THROUGH THE SERIAL PORT
800D E584            27            MOV     A,AD0         ; ALSO SEND THE READING
800F 113B            28            ACALL   COUT          ; THROUGH THE SERIAL PORT
                     29
8011 7402            30            MOV     A,#2
8013 113B            31            ACALL   COUT          ; THROUGH THE SERIAL PORT
8015 E594            32            MOV     A,AD1         ; ALSO SEND THE READING
8017 113B            33            ACALL   COUT          ; THROUGH THE SERIAL PORT
                     34
8019 7403            35            MOV     A,#3
801B 113B            36            ACALL   COUT          ; THROUGH THE SERIAL PORT
801D E5A4            37            MOV     A,AD2         ; ALSO SEND THE READING
801F 113B            38            ACALL   COUT          ; THROUGH THE SERIAL PORT
                     39
8021 7404            40            MOV     A,#4
8023 113B            41            ACALL   COUT          ; THROUGH THE SERIAL PORT
8025 E5B4            42            MOV     A,AD3         ; ALSO SEND THE READING
8027 113B            43            ACALL   COUT          ; THROUGH THE SERIAL PORT
                     44
8029 12802E          45            CALL    DELAY
802C 80DB            46            SJMP    WAIT
                     47
802E DAFE            48    DELAY:  DJNZ    R2,$
8030 DBFC            49            DJNZ    R3,DELAY
8032 22              50            RET
                     51
8033                 52    CIN:
8033 3098FD          53            JNB     RI,$            ; WAIT FOR THE TI FLAG TO SET
8036 C298            54            CLR     RI               ; CLEAR THE TI FLAG ONCE
SENT
8038 E599            55            MOV     A,SBUF           ; SEND THE CONTENTS OF
REGISTER A
803A 22              56            RET                      ; RETURN
803B                 57    COUT:
803B 3099FD          58            JNB     TI,$            ; WAIT FOR THE TI FLAG TO SET
```

```
803E C299             59            CLR     TI              ; CLEAR THE TI FLAG ONCE
SENT
8040 F599             60            MOV     SBUF,A          ; SEND THE CONTENTS OF
REGISTER A
8042 22               61            RET                     ; RETURN
                      62
                      63            END

VERSION 1.2h ASSEMBLY COMPLETE, 0 ERRORS FOUND
```

ACON	D	ADDR	0097H	PREDEFINED
AD0	D	ADDR	0084H	PREDEFINED
AD1	D	ADDR	0094H	PREDEFINED
AD2	D	ADDR	00A4H	PREDEFINED
AD3	D	ADDR	00B4H	PREDEFINED
CIN	C	ADDR	8033H	
COUT	C	ADDR	803BH	
DELAY	C	ADDR	802EH	
RI	B	ADDR	0098H	PREDEFINED
SBUF	D	ADDR	0099H	PREDEFINED
START	C	ADDR	8000H	
TI	B	ADDR	0099H	PREDEFINED
WAIT	C	ADDR	8009H	

This program tests the motor controller. It will allow you to run the motor up and down, and it will allow you to shift back and forth into reverse and forward.

```
PWM_MOT
PAGE 1

                        1    ; NOTE:  THIS IS A NON-BOOTUP PROGRAM, IT DOESN'T HAVE LINES
TO CLEAR THE WDT
                        2    ; SINCE THAT'S BEEN TAKEN CARE OF BY THE SMALL-C MONITOR.
                        3
                        4            $MOD51GB
                        5
                        6    PWMCNT  EQU     R2                  ; use R2 as the PMW duty
cycle counter
                        7
8000                    8            ORG     8000H
                        9
8000 7590FF            10            MOV     P1,#00FFH           ; IN ORDER FOR THE PWM
SIGNAL TO TAKE EFFECT,
                       11                                        ; ALL OUTPUT BITS ON PORT
1 MUST BE SET HIGH
                       12
8003 7A7F              13            MOV     PWMCNT,#07FH         ; START THE PW VALUE
WITH 01
8005 8AFC              14            MOV     CCAP2H, PWMCNT      ; LOAD THE PWM REGISTER
WITH THE PW VALUE
8007 128039            15            CALL    PCA_INIT            ; INITIALIZE THE PWM
                       16
800A                   17    MAIN_LOOP:
                       18
                       19
800A 128044            20            CALL    CIN                 ; WAIT FOR A KEY TO BE
PRESSED
800D B43105            21            CJNE    A,#'1',K2           ; IF KEY='1'
8010 0A                22            INC     PWMCNT              ; INCREMENT THE PULSE
WIDTH TIMER VALUE
8011 8AFC              23            MOV     CCAP2H, PWMCNT      ; LOAD THE PWM REGISTER
WITH THE PW VALUE
8013 80F5              24            SJMP    MAIN_LOOP
                       25
8015 B43205            26    K2:     CJNE    A,#'2',K3           ; IF KEY='2'
8018 1A                27            DEC     PWMCNT              ; DECREMENT THE PULSE
WITH TIMER VALUE
8019 8AFC              28            MOV     CCAP2H, PWMCNT      ; LOAD THE PWM REGISTER
WITH THE PW VALUE
801B 80ED              29            SJMP    MAIN_LOOP
                       30
801D B43304            31    K3:     CJNE    A,#'3',K4           ; IF KEY='3'
8020 C2F9              32            CLR     P5.1                ; TURN OFF MOTOR
8022 80E6              33            SJMP    MAIN_LOOP
                       34
                       35
8024 B43404            36    K4:     CJNE    A,#'4',KQ           ; IF KEY='4'
8027 D2F9              37            SETB    P5.1                ; TURN ON MOTOR
8029 80DF              38            SJMP    MAIN_LOOP
                       39
                       40
802B B47104            41    KQ:     CJNE    A,#'q',KW              ; IF KEY='q'
802E C2F8              42            CLR     P5.0                   ; GO FORWARD
8030 80D8              43            SJMP    MAIN_LOOP
                       44
8032 B477D5            45    KW:     CJNE    A,#'w',MAIN_LOOP       ; IF KEY='w'
8035 D2F8              46            SETB    P5.0                   ; GO BACKWARD
8037 80D1              47            SJMP    MAIN_LOOP
                       48
8039                   49    PCA_INIT:
8039 75D906            50            MOV     CMOD,#006H          ; PCA FREQUENCY = 1/Timer
0 overflow
803C 75DC42            51            MOV     CCAPM2,#42H         ; PUT PCA MODULE 2 INTO
PWM MODE
803F 8AFC              52            MOV     CCAP2H, PWMCNT      ; INITIALIZE DUTY CYCLE
```

```
8041 D2DE            53           SETB    CR                 ; START THE PCA CLOCK
8043 22              54           RET
                     55
8044                 56    CIN:
8044 3098FD          57           JNB     RI,$               ; WAIT FOR A BYTE
8047 C298            58           CLR     RI                 ; CLEAR THE RECEIVE FLAG
ONCE RECIEVED

PWM_MOT
PAGE 2

8049 E599            59           MOV     A,SBUF             ; GET THE RECEIVED BYTE INTO
THE A REGISTER
804B 22              60           RET
                     61
                     62
                     63           END
                     64

VERSION 1.2h ASSEMBLY COMPLETE, 0 ERRORS FOUND

PWM_MOT
PAGE 3

CCAP2H . . . . . .  D ADDR  00FCH  PREDEFINED
CCAPM2 . . . . . .  D ADDR  00DCH  PREDEFINED
CIN. . . . . .  C ADDR  8044H
CMOD . . . . . .  D ADDR  00D9H  PREDEFINED
CR . . . . . .  B ADDR  00DEH  PREDEFINED
K2 . . . . . .  C ADDR  8015H
K3 . . . . . .  C ADDR  801DH
K4 . . . . . .  C ADDR  8024H
KQ . . . . . .  C ADDR  802BH
KW . . . . . .  C ADDR  8032H
MAIN_LOOP. . . . . .  C ADDR  800AH
P1 . . . . . .  D ADDR  0090H  PREDEFINED
P5 . . . . . .  D ADDR  00F8H  PREDEFINED
PCA_INIT . . . . . .  C ADDR  8039H
PWMCNT . . . . . .    REG2
RI . . . . . .  B ADDR  0098H  PREDEFINED
SBUF . . . . . .  D ADDR  0099H  PREDEFINED
```

This program tests the steering servo. It will allow you center the wheels, turn them full left or right, and to step them left or right.

```
PWM_STR
PAGE 1

                      1
                      2    ; NOTE:  THIS IS A NON-BOOTUP PROGRAM, IT DOESN'T HAVE LINES
TO CLEAR THE WDT
                      3    ; SINCE THAT'S BEEN TAKEN CARE OF BY THE SMALL-C MONITOR.
                      4
                      5            $MOD51GB
                      6
                      7    PWMCNT  EQU     R2                  ; use R2 as the PMW duty
cycle counter
                      8
8000                  9            ORG     8000H
                     10
8000 7590FF          11            MOV     P1,#00FFH           ; IN ORDER FOR THE PWM
SIGNAL TO TAKE EFFECT,
                     12                                        ; ALL OUTPUT BITS ON PORT
1 MUST BE SET HIGH
                     13
8003 7A3B            14            MOV     PWMCNT,#03BH         ; START THE PW VALUE
WITH 01
8005 8AFB            15            MOV     CCAP1H, PWMCNT      ; LOAD THE PWM REGISTER
WITH THE PW VALUE
8007 128038          16            CALL    PCA_INIT            ; INITIALIZE THE PWM
                     17
800A                 18    MAIN_LOOP:
                     19
800A 128043          20            CALL    CIN                 ; WAIT FOR A KEY TO BE
PRESSED
800D B43106          21            CJNE    A,#'1',K2           ; IF KEY='1'
8010 7A31            22            mov     PWMCNT,#031h        ; Turn wheels full left
8012 8AFB            23            MOV     CCAP1H, PWMCNT      ; LOAD THE PWM REGISTER
WITH THE PW VALUE
8014 80F4            24            SJMP    MAIN_LOOP
                     25
8016 B43206          26    K2:     CJNE    A,#'2',K3           ; IF KEY='2'
8019 7A3B            27            mov     PWMCNT,#03bh        ; center wheels
801B 8AFB            28            MOV     CCAP1H, PWMCNT      ; LOAD THE PWM REGISTER
WITH THE PW VALUE
801D 80EB            29            SJMP    MAIN_LOOP
                     30
801F B43306          31    K3:     CJNE    A,#'3',KQ           ; IF KEY='3'
8022 7A4B            32            mov     PWMCNT,#04bh        ; Turn the wheels full
right
8024 8AFB            33            MOV     CCAP1H, PWMCNT      ; LOAD THE PWM REGISTER
WITH THE PW VALUE
8026 80E2            34            SJMP    MAIN_LOOP
                     35
8028 B47105          36    KQ:     CJNE    A,#'q',KW           ; IF KEY='q'
802B 0A              37            inc     PWMCNT              ; tick wheels to left
802C 8AFB            38            MOV     CCAP1H, PWMCNT      ; LOAD THE PWM R
802E 80DA            39            SJMP    MAIN_LOOP
                     40
8030 B477D7          41    KW:     CJNE    A,#'w',MAIN_LOOP
8033 1A              42            dec     PWMCNT
8034 8AFB            43            MOV     CCAP1H,PWMCNT
8036 80D2            44            SJMP    MAIN_LOOP
                     45
8038                 46    PCA_INIT:
8038 75D906          47            MOV     CMOD,#006h
803B 75DB42          48            MOV     CCAPM1,#42h
803E 8AFB            49            MOV     CCAP1H,PWMCNT
8040 D2DE            50            SETB    CR
8042 22              51            RET
                     52
8043                 53    CIN:
8043 3098FD          54            JNB     RI,$             ; WAIT FOR A BYTE
```

```
8046 C298            55           CLR     RI              ; CLEAR THE RECEIVE FLAG
ONCE RECIEVED
8048 E599            56           MOV     A,SBUF          ; GET THE RECEIVED BYTE INTO
THE A REGISTER
804A 22              57           RET
                     58
PWM_STR
PAGE 2

                     59
                     60           END
                     61

VERSION 1.2h ASSEMBLY COMPLETE, 0 ERRORS FOUND

PWM_STR
PAGE 3

CCAP1H . . . . . .  D ADDR  00FBH  PREDEFINED
CCAPM1 . . . . . .  D ADDR  00DBH  PREDEFINED
CIN. . . . . . . .  C ADDR  8043H
CMOD . . . . . . .  D ADDR  00D9H  PREDEFINED
CR . . . . . . . .  B ADDR  00DEH  PREDEFINED
K2 . . . . . . . .  C ADDR  8016H
K3 . . . . . . . .  C ADDR  801FH
KQ . . . . . . . .  C ADDR  8028H
KW . . . . . . . .  C ADDR  8030H
MAIN_LOOP. . . . .  C ADDR  800AH
P1 . . . . . . . .  D ADDR  0090H  PREDEFINED
PCA_INIT . . . . .  C ADDR  8038H
PWMCNT . . . . . .    REG2
RI . . . . . . . .  B ADDR  0098H  PREDEFINED
SBUF . . . . . . .  D ADDR  0099H  PREDEFINED
```

This program tests the sonar servo. It will allow you to place the sonar in any one of seven different positions from any other position.

```
PWM_SON
PAGE 1

                      1
                      2    ; NOTE:  THIS IS A NON-BOOTUP PROGRAM, IT DOESN'T HAVE LINES
TO CLEAR THE WDT
                      3    ; SINCE THAT'S BEEN TAKEN CARE OF BY THE SMALL-C MONITOR.
                      4
                      5            $MOD51GB
                      6
                      7    PWMCNT  EQU     R2                  ; use R2 as the PMW duty
cycle counter
                      8
8000                  9            ORG     8000H
                     10
8000 7590FF          11            MOV     P1,#00FFH           ; IN ORDER FOR THE PWM
SIGNAL TO TAKE EFFECT,
                     12                                        ; ALL OUTPUT BITS ON PORT
1 MUST BE SET HIGH
                     13
8003 7A3B            14            MOV     PWMCNT,#03BH         ; START THE PW VALUE
WITH 01
8005 8AFA            15            MOV     CCAP0H, PWMCNT      ; LOAD THE PWM REGISTER
WITH THE PW VALUE
8007 12804C          16            CALL    PCA_INIT            ; INITIALIZE THE PWM
                     17
800A                 18    MAIN_LOOP:
                     19
800A 128057          20            CALL    CIN                 ; WAIT FOR A KEY TO BE
PRESSED
800D B43106          21            CJNE    A,#'1',K2           ; IF KEY='1'
8010 7A62            22            MOV     PWMCNT,#062h        ; Place the sonar at 90
left
8012 8AFA            23            MOV     CCAP0H, PWMCNT      ; LOAD THE PWM REGISTER
WITH THE PW VALUE
8014 80F4            24            SJMP    MAIN_LOOP
                     25
8016 B43206          26    K2:     CJNE    A,#'2',K3           ; IF KEY='2'
8019 7A56            27            MOV     PWMCNT,#056h        ; Place the sonar at 60
left
801B 8AFA            28            MOV     CCAP0H, PWMCNT      ; LOAD THE PWM REGISTER
WITH THE PW VALUE
801D 80EB            29            SJMP    MAIN_LOOP
                     30
801F B43306          31    K3:     CJNE    A,#'3',K4           ; IF KEY='3'
8022 7A48            32            MOV     PWMCNT,#048h        ; Place the sonar at 30
left
8024 8AFA            33            MOV     CCAP0H, PWMCNT      ; LOAD THE PWM REGISTER
WITH THE PW VALUE
8026 80E2            34            SJMP    MAIN_LOOP
                     35
8028 B43406          36    K4:     CJNE    A,#'4',K5           ; IF KEY='4'
802B 7A3B            37            MOV     PWMCNT,#03bh        ; Place the sonar at 0
802D 8AFA            38            MOV     CCAP0H, PWMCNT      ; LOAD THE PWM REGISTER
WITH THE PW VALUE
802F 80D9            39            SJMP    MAIN_LOOP
                     40
8031 B43506          41    K5:     CJNE    A,#'5',K6           ; IF KEY='5'
8034 7A2D            42            MOV     PWMCNT,#02dh        ; Place the sonar at 30
right
8036 8AFA            43            MOV     CCAP0H, PWMCNT      ; LOAD THE PWM REGISTER
WITH THE PW VALUE
8038 80D0            44            SJMP    MAIN_LOOP
                     45
803A B43606          46    K6:     CJNE    A,#'6',K7           ; IF KEY='6'
803D 7A21            47            MOV     PWMCNT,#021h        ; Place the sonar at 60
right
```

```
803F 8AFA            48           MOV     CCAP0H, PWMCNT      ; LOAD THE PWM REGISTER
WITH THE PW VALUE
8041 80C7            49           SJMP    MAIN_LOOP
                     50
8043 B437C4          51    K7:    CJNE    A,#'7',MAIN_LOOP    ; IF KEY='7'
8046 7A15            52           MOV     PWMCNT,#015h        ; Place the sonar at 90
right
8048 8AFA            53           MOV     CCAP0H, PWMCNT      ; LOAD THE PWM REGISTER
WITH THE PW VALUE
804A 80BE            54           SJMP    MAIN_LOOP
                     55
804C                 56    PCA_INIT:
804C 75D906          57           MOV     CMOD,#006H          ; PCA FREQUENCY = 1/Timer
0 overflow
804F 75DA42          58           MOV     CCAPM0,#42H         ; PUT PCA MODULE 0 INTO
PWM MODE

PWM_SON
PAGE 2

8052 8AFA            59           MOV     CCAP0H, PWMCNT      ; INITIALIZE DUTY CYCLE
8054 D2DE            60           SETB    CR                  ; START THE PCA CLOCK
8056 22              61           RET
                     62
8057                 63    CIN:
8057 3098FD          64           JNB     RI,$              ; WAIT FOR A BYTE
805A C298            65           CLR     RI                ; CLEAR THE RECEIVE FLAG
ONCE RECIEVED
805C E599            66           MOV     A,SBUF            ; GET THE RECEIVED BYTE INTO
THE A REGISTER
805E 22              67           RET
                     68
                     69
                     70           END
                     71

VERSION 1.2h ASSEMBLY COMPLETE, 0 ERRORS FOUND

PWM_SON
PAGE 3

CCAP0H . . . . . .  D ADDR  00FAH  PREDEFINED
CCAPM0 . . . . . .  D ADDR  00DAH  PREDEFINED
CIN. . . . . .  C ADDR  8057H
CMOD . . . . . .  D ADDR  00D9H  PREDEFINED
CR . . . . . .  B ADDR  00DEH  PREDEFINED
K2 . . . . . .  C ADDR  8016H
K3 . . . . . .  C ADDR  801FH
K4 . . . . . .  C ADDR  8028H
K5 . . . . . .  C ADDR  8031H
K6 . . . . . .  C ADDR  803AH
K7 . . . . . .  C ADDR  8043H
MAIN_LOOP. . . . . .  C ADDR  800AH
P1 . . . . . .  D ADDR  0090H  PREDEFINED
PCA_INIT . . . . . .  C ADDR  804CH
PWMCNT . . . . . .    REG2
RI . . . . . .  B ADDR  0098H  PREDEFINED
SBUF . . . . . .  D ADDR  0099H  PREDEFINED
```

This program was written to test the IR sensor collision avoidance. It is the direct ancestor of the sonar navigation program.

```
LO_SPD2
PAGE 1

8000                     1           org     8000h
8000 028200              2           jmp     8200h
                         3
                         4           $MOD51GB
                         5
                         6    PWM_MOT EQU     R0
                         7    PWM_STR EQU     R1
  00F9                   8    STP_GO  EQU     P5.1
  00F8                   9    FOR_REV EQU     P5.0
                        10
                        11
8200                    12           org     8200h
8200                    13    main:
                        14           ;enable A/D operation
8200 759710             15           mov     ACON,#010h
                        16
                        17           ;STP_GO is cleared to stop motor
                        18           ;FOR_REV is set to go in reverse
                        19
                        20           ;initalize the motor and steering PWM waveforms
8203 7590FF             21           mov     P1,#0FFH        ;must set all bits on port1
to 0
8206 78C0               22           mov     PWM_MOT,#0c0h   ;set motor for 1/8 speed
8208 88FC               23           mov     CCAP2h,PWM_MOT
820A 793B               24           mov     PWM_STR,#03Bh   ;set steering servo to center
820C 89FB               25           mov     CCAP1h,PWM_STR
820E C2F8               26           clr     FOR_REV           ;set relays for FORWARD
8210 C2F9               27           clr     STP_GO          ;block PWM signal to motor
8212 128280             28           call    PCA_INIT
                        29
8215 D2F9               30           setb    STP_GO
8217 12828C             31    IRScan:   call    lngDelay
                        32
821A C3                 33    L_Rear: clr     C             ; Clear carry bit
821B E594               34           mov     A,AD1         ; get the left rear IR
821D 9464               35           subb    A,#100        ; subtract threshold from value
821F 4026               36           jc      L_Front       ; jump if value is less than
threshold
                        37
8221 30F921             38           jnb     STP_GO,stp_rev2  ;already stopped
8224 30F81E             39           jnb     FOR_REV,stp_rev2 ;or going forward
8227 C2F9               40           clr     STP_GO           ;stop motor
8229 C2F8               41           clr     FOR_REV          ;place in forward
822B 793B               42           mov     PWM_STR,#03bh    ;center wheels
822D 89FB               43           mov     CCAP1H,PWM_STR   ;
822F 12828C             44           call    lngDelay
8232 78C0               45           mov     PWM_MOT,#0C0h           ;set motor for 1/8
speed
8234 88FC               46           mov     CCAP2h,PWM_MOT
8236 D2F9               47           setb    STP_GO           ;start motor
8238 1282A1             48           call    PRINT
823B 466F7277           49           db      'Forward\n',0,
823F 6172645C
8243 6E00
8245                    50    stp_rev2:
8245 80D0               51           jmp     IRScan
                        52
                        53
8247                    54    L_Front:
8247 C3                 55           clr     C             ; Clear carry bit
8248 E5A4               56           mov     A,AD2         ; get the left front IR
```

```
LO_SPD2
PAGE 2

824A 946E              57            subb    A,#110        ; subtract threshold from value
824C 4003              58            jc      R_Front       ; jump if value is less than
threshold
                       59
824E 028258            60            jmp     forward
                       61
8251                   62    R_Front:
8251 C3                63            clr     C             ; Clear carry bit
8252 E5B4              64            mov     A,AD3         ; get the left front IR
8254 946E              65            subb    A,#110        ; subtract threshold from value
8256 40BF              66            jc      IRScan        ; jump if value is less than
threshold
                       67
8258                   68    forward:
8258 30F921            69            jnb     STP_GO,stp_rev0         ;already stopped
825B 20F81E            70            jb      FOR_REV,stp_rev0        ;or backing
825E C2F9              71            clr     STP_GO                  ;stop motor
8260 D2F8              72            setb    FOR_REV                 ;reverse it
8262 7931              73            mov     PWM_STR,#031h           ;turn wheels to right
8264 89FB              74            mov     CCAP1H,PWM_STR          ;
8266 12828C            75            call    lngDelay                ;Give it 1/2 a second
8269 7899              76            mov     PWM_MOT,#099h           ;set motor for 1/8
speed
826B 88FC              77            mov     CCAP2h,PWM_MOT
826D D2F9              78            setb    STP_GO                  ;start motor
826F 1282A1            79            call    PRINT
8272 4261636B          80            db      'Backing\n',0,
8276 696E675C
827A 6E00
827C                   81    stp_rev0:
827C 8099              82            jmp     IRScan
                       83
827E 80FE              84    wait:   jmp     wait
                       85
                       86
                       87
                       88            ;initializes the programmable counter array for the
motor
                       89            ;and the steering servo
8280                   90    PCA_INIT:
8280 75D906            91            mov     CMOD,#006h      ;set PCA frequency to
external
8283 75DB42            92            mov     CCAPM1,#42h     ;put PCA module 1 into PWM
mode
8286 75DC42            93            mov     CCAPM2,#42h     ;put PCA module 0 into PWM
mode
8289 D2DE              94            setb    CR              ;start the PCA clock
828B 22                95            ret
                       96
828C                   97    lngDelay:
                 =1    98            $INCLUDE(lngDelay.inc)
828C 7D0A        =1    99            mov   R5,#00Ah
828E 7E35        =1   100    delay2: mov     R6,#035h
8290 7FFF        =1   101    delay3: mov     R7,#0FFh
8292 DFFE        =1   102    delay4: djnz    R7, delay4              ;wait >500msec
8294 DEFA        =1   103            djnz    R6, delay3
8296 DDF6        =1   104            djnz    R5, delay2
8298 22          =1   105            ret
                      106
8299                  107    cout:
                 =1   108            $INCLUDE(cout.inc)
8299 3099FD      =1   109            JNB     TI,$            ;Wait until transmission
completed.
829C C299        =1   110            CLR     TI              ;Clear interrupt flag.
829E F599        =1   111            MOV     SBUF,A          ;Write out character.
82A0 22          =1   112            RET
```

```
                        113
82A1                    114     PRINT:
                  =1    115             $INCLUDE(PRINT.inc)
82A1 C0E0         =1    116             PUSH    ACC       ; push the A and DPTR registers so
their contents
82A3 C083         =1    117             PUSH    DPH       ; don't get lost in case they are
being used
82A5 C082         =1    118             PUSH    DPL       ; before calling this function.
                  =1    119
                  =1    120     ; move the stack pointer down low in order to pop the first
string's byte
                  =1    121     ; address
82A7 E581         =1    122             MOV     A,SP
82A9 24FD         =1    123             ADD     A,#0FDH
82AB F581         =1    124             MOV     SP,A
82AD D083         =1    125             POP     DPH
82AF D082         =1    126             POP     DPL
                  =1    127
                  =1    128     ; send (PRINT) the string back to the PC computer
82B1              =1    129     READ_TX:
82B1 E4           =1    130             CLR     A
82B2 93           =1    131             MOVC    A,@A+DPTR
82B3 A3           =1    132             INC     DPTR
                  =1    133
82B4 B45C0C       =1    134             CJNE    A,#'\',SND
                  =1    135
82B7 E4           =1    136             CLR     A
82B8 93           =1    137             MOVC    A,@A+DPTR
82B9 A3           =1    138             INC     DPTR
                  =1    139
82BA B46E06       =1    140             CJNE    A,#'n',SND
                  =1    141
82BD 740D         =1    142             MOV     A,#0DH
82BF 5199         =1    143             ACALL   cout
82C1 740A         =1    144             MOV     A,#0AH
                  =1    145
82C3 5199         =1    146     SND:    ACALL   cout
82C5 B400E9       =1    147             CJNE    A,#0,READ_TX
                  =1    148
                  =1    149     ; place the new return addres in stack
82C8 C082         =1    150             PUSH    DPL
82CA C083         =1    151             PUSH    DPH
                  =1    152
                  =1    153     ; move the stack pointer up high in order to pop the A and
DPTR registers
82CC E581         =1    154             MOV     A,SP
82CE 2403         =1    155             ADD     A,#3
82D0 F581         =1    156             MOV     SP,A
82D2 D082         =1    157             POP     DPL
82D4 D083         =1    158             POP     DPH
82D6 D0E0         =1    159             POP     ACC
82D8 22           =1    160             RET
                        161
82D9                    162     SEND_BYTE:
                  =1    163             $INCLUDE(SEND_BYT.inc)
82D9 3099FD       =1    164             JNB     TI, SEND_BYTE           ; WAIT FOR THE TI FLAG
TO SET
82DC C299         =1    165             CLR             TI                              ;
CLEAR THE TI FLAG ONCE SENT
82DE F599         =1    166                     MOV     SBUF,A                  ; SEND THE
CONTENTS OF REGISTER A
82E0 22           =1    167             RET                              ; RETURN
                  =1    168
;------------------------------------------------------------------------------
                  =1    169
                  =1    170
```

```
               =1   171
                    172
                    173
                    174
                    175           end
```

VERSION 1.2h ASSEMBLY COMPLETE, 0 ERRORS FOUND

ACC.	D	ADDR	00E0H	PREDEFINED
ACON	D	ADDR	0097H	PREDEFINED
AD1.	D	ADDR	0094H	PREDEFINED
AD2.	D	ADDR	00A4H	PREDEFINED
AD3.	D	ADDR	00B4H	PREDEFINED
CCAP1H	D	ADDR	00FBH	PREDEFINED
CCAP2H	D	ADDR	00FCH	PREDEFINED
CCAPM1	D	ADDR	00DBH	PREDEFINED
CCAPM2	D	ADDR	00DCH	PREDEFINED
CMOD	D	ADDR	00D9H	PREDEFINED
COUT	C	ADDR	8299H	
CR	B	ADDR	00DEH	PREDEFINED
DELAY2	C	ADDR	828EH	
DELAY3	C	ADDR	8290H	
DELAY4	C	ADDR	8292H	
DPH.	D	ADDR	0083H	PREDEFINED
DPL.	D	ADDR	0082H	PREDEFINED
FORWARD.	C	ADDR	8258H	
FOR_REV.		NUMB	00F8H	
IRSCAN	C	ADDR	8217H	
LNGDELAY	C	ADDR	828CH	
L_FRONT.	C	ADDR	8247H	
L_REAR	C	ADDR	821AH	NOT USED
MAIN	C	ADDR	8200H	NOT USED
P1	D	ADDR	0090H	PREDEFINED
P5	D	ADDR	00F8H	PREDEFINED
PCA_INIT	C	ADDR	8280H	
PRINT.	C	ADDR	82A1H	
PWM_MOT.		REG0		
PWM_STR.		REG1		
READ_TX.	C	ADDR	82B1H	
R_FRONT.	C	ADDR	8251H	
SBUF	D	ADDR	0099H	PREDEFINED
SEND_BYTE.	C	ADDR	82D9H	
SND.	C	ADDR	82C3H	
SP	D	ADDR	0081H	PREDEFINED
STP_GO		NUMB	00F9H	
STP_REV0	C	ADDR	827CH	
STP_REV2	C	ADDR	8245H	
TI	B	ADDR	0099H	PREDEFINED
WAIT	C	ADDR	827EH	

Last, but not least, is this attempt at getting the compass integrated. My data is getting corrupted somewhere. I didn't have time to digest my errors

```
ROBOSPDY
PAGE 1

8000                     1            org     8000h
8000 028200              2            jmp     8200h
                         3
                         4            $MOD51GB
                         5
                         6    ;Pulse width signals
                         7    PWM_MOT         EQU     R0
                         8    PWM_STR         EQU     R1
                         9
                        10    ;Compass data register
                        11    CompCount       EQU     R2
                        12    DataByte        EQU     R4
                        13
                        14    ;Compass result direction flag
  00FF                  15    Left_Flag       EQU     P5.7
  00FE                  16    Right_Flag      EQU     P5.6
                        17
                        18    ;Motor Direction
  00F9                  19    STP_GO          EQU     P5.1
  00F8                  20    FOR_REV         EQU     P5.0
                        21
                        22    ;Speeds
  00C0                  23    FWD_LO          EQU     0C0H
  00B0                  24    FWD_MED         EQU     0B0H
  00B0                  25    FWD_HI          EQU     0B0H
  0090                  26    REV_LO          EQU     090H
                        27
                        28    ;Steering Directions
  003F                  29    Straight        EQU     03fH
  0030                  30    Full_Left       EQU     030H
  0037                  31    Half_Left       EQU     037H
  003A                  32    Qtr_Left        EQU     03AH
  004D                  33    Full_Right      EQU     04DH
  0045                  34    Half_Right      EQU     045H
  0042                  35    Qtr_Right       EQU     042H
                        36
                        37    ;Sonar ranges
  00F5                  38    Sonar_15        EQU     245
  00D0                  39    Sonar_12        EQU     208
  009B                  40    Sonar_9         EQU     155
  0067                  41    Sonar_6         EQU     103
  0033                  42    Sonar_3         EQU     51
  001A                  43    Sonar_1         EQU     26
                        44
                        45    ;Compass Pins
  00C0                  46    EOC     equ     P4.0
  00C1                  47    SS      equ     P4.1
  00C2                  48    MS      equ     P4.2
  00C3                  49    RESET   equ     P4.3
  00C4                  50    SCLK    equ     P4.4
  00C5                  51    SDO     equ     P4.5
  00C6                  52    POLL    equ     P4.6
  00C7                  53    POW     equ     P4.7
                        54
                        55    ;Compass data
  0081                  56    Base      equ     081h
  0000                  57    Actual_HI equ     000h
  0001                  58    Actual_LO equ     001h
```

```
                     59
  0002               60    Desired_HI  equ     002h
  0003               61    Desired_LO  equ     003h
                     62
802B                 63            org     802bh
802B 028495          64            jmp     T2_ISR
                     65
8100                 66            org     8100h
8100 00000000        67            db      0,0,0,0
                     68
                     69
8200                 70            org     8200h
8200                 71    main:
                     72            ;enable A/D operation
8200 759710          73            mov     ACON,#010h
                     74
                     75            ;STP_GO is cleared to stop motor
                     76            ;FOR_REV is set to go in reverse
                     77
                     78            ;initalize the motor and steering PWM waveforms
8203 7590FF          79            mov     P1,#0FFH        ;must set all bits on port1
to 0
8206 78C0            80            mov     PWM_MOT,#FWD_LO   ;set motor for 1/8 speed
8208 88FC            81            mov     CCAP2h,PWM_MOT
820A 793C            82            mov     PWM_STR,#03Ch   ;set steering servo to center
820C 89FB            83            mov     CCAP1h,PWM_STR
820E C2F8            84            clr     FOR_REV           ;set relays for FORWARD
8210 C2F9            85            clr     STP_GO          ;block PWM signal to motor
8212 12855A          86            call    PCA_INIT
                     87
                     88            ;Set up the compass Inits
8215 12846B          89            call    Comp_Init       ;Initialize the compass
8218 128589          90            call    Print
821B 54686520        91            db      'The compass is initialized\n',0,
821F 636F6D70
8223 61737320
8227 69732069
822B 6E697469
822F 616C697A
8233 65645C6E
8237 00
8238 12848A          92            call    T2_Init         ;Initialize T2
823B 128589          93            call    Print
823E 54696D65        94            db      'Timer 2 is initialized\n',0,
8242 72203220
8246 69732069
824A 6E697469
824E 616C697A
8252 65645C6E
8256 00
                     95
                     96
;*************************************************************************
8257 858183          97            mov     DPH,Base
825A 850282          98            mov     DPL,Desired_HI
825D 7400            99            mov     A,#0
825F F0             100            movx    @DPTR,A
8260 A3             101            inc     DPTR
8261 745A           102            mov     A,#05Ah
8263 F0             103            movx    @DPTR,A
```

```
                     104
;*******************************************************************************
                     105
                     106    ;        setb    ET2             ;enable Timer 2 interrupt
                     107    ;        setb    EA              ;enable interrupts
                     108
8264 D2F9            109            setb    STP_GO
                     110
8266 128574          111    Scan: call    lngDelay
                     112
                     113
8269 128589          114    L_Rear: call    Print
826C 4C5200          115            db      'LR',0,
826F C3              116            clr     C             ; Clear carry bit
8270 E594            117            mov     A,AD1         ; get the left rear IR
8272 9464            118            subb    A,#100        ; subtract threshold from value
8274 402E            119            jc      L_Front       ; jump if not too close
                     120
8276                 121    rev2for:
8276 30F929          122            jnb     STP_GO,stp_rev2  ;already stopped
8279 30F826          123            jnb     FOR_REV,stp_rev2 ;or going forward
827C C2F9            124            clr     STP_GO           ;stop motor
827E C2F8            125            clr     FOR_REV          ;place in forward
8280 793F            126            mov     PWM_STR,#Straight ;center wheels
8282 89FB            127            mov     CCAP1H,PWM_STR    ;
8284 128574          128            call    lngDelay
8287 78C0            129            mov     PWM_MOT,#FWD_LO  ;set motor for 1/8 speed
8289 88FC            130            mov     CCAP2h,PWM_MOT
828B D2F9            131            setb    STP_GO           ;start motor
828D 128589          132            call    PRINT
8290 466F7277        133            db      'Forward from IR\n',0,
8294 61726420
8298 66726F6D
829C 2049525C
82A0 6E00
82A2                 134    stp_rev2:
82A2 80C2            135            jmp     Scan
                     136
                     137
82A4                 138    L_Front:
82A4 128589          139            call    Print
82A7 4C4600          140            db      'LF',0,
82AA C3              141            clr     C             ; Clear carry bit
82AB E5A4            142            mov     A,AD2         ; get the left front IR
82AD 9464            143            subb    A,#100        ; subtract threshold from value
82AF 4003            144            jc      R_Front       ; jump if not too close
                     145
82B1 0282C1          146            jmp     for2rev
                     147
82B4                 148    R_Front:
82B4 128589          149            call    Print
82B7 524600          150            db      'RF',0,
82BA C3              151            clr     C             ; Clear carry bit
82BB E5B4            152            mov     A,AD3         ; get the left front IR
82BD 9464            153            subb    A,#100        ; subtract threshold from value
82BF 402E            154            jc      Compass_Calc  ; jump if not too close
                     155
                     156
82C1                 157    for2Rev:
```

```
82C1 30F929          158           jnb     STP_GO,stp_rev0         ;already stopped
82C4 20F826          159           jb      FOR_REV,stp_rev0        ;or backing
82C7 C2F9            160           clr     STP_GO                  ;stop motor
82C9 D2F8            161           setb    FOR_REV                 ;reverse it
82CB 7931            162           mov     PWM_STR,#031h           ;turn wheels to right
82CD 89FB            163           mov     CCAP1H,PWM_STR          ;
82CF 128574          164           call    lngDelay                ;Give it 1/2 a second
82D2 7890            165           mov     PWM_MOT,#REV_LO         ;set motor for 1/8
speed
82D4 88FC            166           mov     CCAP2h,PWM_MOT
82D6 D2F9            167           setb    STP_GO                  ;start motor
82D8 128589          168           call    PRINT
82DB 4261636B        169           db      'Backing from IR\n',0,
82DF 696E6720
82E3 66726F6D
82E7 2049525C
82EB 6E00
82ED                 170    stp_rev0:
82ED 4166            171           jmp     Scan
                     172
82EF                 173    Compass_Calc:
82EF 128392          174           call    Compass         ;get differential heading
82F2 E582            175           mov     A,DPL           ;from DPL  ( < B4 )
82F4 7002            176           jnz     TurnGo          ;if heading matches goto
Scan(sonarsoon???)
82F6 4166            177           jmp     Scan
                     178
82F8 C3              179    TurnGo: clr     C               ;clear carry
82F9 7F0A            180           mov     R7,#10          ;load R7 with 10
82FB 9F              181           subb    A,R7            ;is difference >10?
82FC 4022            182           jc      undr_10
82FE E582            183           mov     A,DPL
8300 C3              184           clr     C
8301 7F19            185           mov     R7,#25
8303 9F              186           subb    A,R7            ;is difference >25?
8304 400D            187           jc      undr_25
8306 20FE05          188           jb      Right_Flag,Right_Full
8309                 189    Left_Full:
8309 7930            190           mov     PWM_STR,#Full_Left
830B 02832D          191           jmp     comp_str
830E                 192    Right_Full:
830E 794D            193           mov     PWM_STR,#Full_Right
8310 02832D          194           jmp     comp_str
                     195
8313                 196    undr_25:
8313 20FE05          197           jb      Right_Flag,Right_Half
8316                 198    Left_Half:
8316 7937            199           mov     PWM_STR,#Half_Left
8318 02832D          200           jmp     comp_str
                     201
831B                 202    Right_Half:
831B 7945            203           mov     PWM_STR,#Half_Right
831D 02832D          204           jmp     comp_str
                     205
8320                 206    undr_10:
8320 20FE05          207           jb      Right_Flag,Right_Qtr
8323                 208    Left_Qtr:
8323 7937            209           mov     PWM_STR,#Half_Left
8325 02832D          210           jmp     comp_str
8328                 211    Right_Qtr:
```

```
8328 7937            212            mov     PWM_STR,#Half_Left
832A 02832D          213            jmp     comp_str
                     214
832D                 215    comp_str:
832D 89FB            216            mov     CCAP1H,PWM_STR
832F 4166            217            jmp     Scan
                     218
                     219
                     220
                     221
                     222
8331 C3              223    Sonar:  clr     C
8332 E584            224            mov     A,AD0           ; Get the sonar value
8334 94F5            225            subb    A,#Sonar_15     ; subtract threshold from
value
8336 4016            226            jc      Ahd_hlf         ; jump if distance is < 15 ft
                     227
8338 78B0            228            mov     PWM_MOT,#FWD_HI  ;set motor for High speed
833A 88FC            229            mov     CCAP2h,PWM_MOT
833C 128589          230            call    PRINT
833F 41686561        231            db      'Ahead Full\n',0,
8343 64204675
8347 6C6C5C6E
834B 00
                     232
834C 4166            233            jmp     Scan
                     234
834E                 235    Ahd_hlf:
834E C3              236            clr     C
834F E584            237            mov     A,AD0           ; Get the sonar value
8351 949B            238            subb    A,#Sonar_9      ; subtract threshold from
value
8353 401B            239            jc      Ahd_slow        ; jump if distance is < 9 ft
8355 B8B036          240            cjne    PWM_MOT,#FWD_HI,All_Stop
                     241                                    ; If motor is in high gear
                     242                                    ; turn it off
                     243
8358 78B0            244            mov     PWM_MOT,#FWD_MED  ;set motor for Medium speed
835A 88FC            245            mov     CCAP2h,PWM_MOT
835C 128589          246            call    PRINT
835F 41686561        247            db      'Ahead Medium\n',0,
8363 64204D65
8367 6469756D
836B 5C6E00
                     248
836E 4166            249            jmp     Scan
                     250
8370                 251    Ahd_slow:
8370 C3              252            clr     C
8371 E584            253            mov     A,AD0           ; Get the sonar value
8373 9433            254            subb    A,#Sonar_3      ; subtract threshold from
value
                     255
;*************************************************************************
                     256    ;        jc      for2rev         ; jump if distance is < 3 ft
                     257
;*************************************************************************
8375 B8B016          258            cjne    PWM_MOT,#FWD_HI,All_Stop
                     259                                    ; If motor is in high gear
                     260                                    ; turn it off
                     261
8378 78C0            262            mov     PWM_MOT,#FWD_LO  ;set motor for Low speed
837A 88FC            263            mov     CCAP2h,PWM_MOT
```

```
837C 128589          264            call    PRINT
837F 41686561        265            db      'Ahead Slow\n',0,
8383 6420536C
8387 6F775C6E
838B 00
838C 4166            266            jmp     Scan
                     267
838E                 268    All_Stop:
838E C2F9            269            clr     STP_GO
8390 4166            270            jmp     Scan
                     271
8392                 272    Compass:
8392 C004            273            push    04
8394 128410          274            call    load_actual     ;get actual from memory
8397 12841D          275            call    load_desired    ;get desired from memory
839A 128428          276            call    get_actual      ;put actual into DPTR
839D 128444          277            call    get_desired     ;put desired into B/Acc
83A0 128457          278            call    util_subbad     ;Actual-Desired
83A3 4021            279            jc      Neg             ;if actual is less than
desired jump to Neg
83A5 75F000          280            mov     B,#0            ;load B/Acc with 180
83A8 74B4            281            mov     A,#0B4h
83AA 128457          282            call    util_subbad     ;subtract 180 from DPTR
83AD 4042            283            jc      Left_Pos        ;if Actual-Desired is <180
jump Left_Pos
83AF                 284    Right_Pos:
83AF D2FE            285            setb    Right_Flag      ;set flag for right turn
83B1 128444          286            call    get_desired     ;put desired into B/Acc
83B4 758301          287            mov     DPH,#1          ;load DPTR with 360
83B7 758267          288            mov     DPL,#067h
83BA 128448          289            call    util_adcbad     ;Add 360 to desired->(DPTR)
83BD 8EF0            290            mov     B,R6            ;Load B/Acc with actual
83BF EF              291            mov     A,R7            ;
83C0 128457          292            call    util_subbad     ;get final difference
                     293                                    ;put it in DPTR
83C3 02840D          294            jmp     end_sub
                     295
83C6 8B83            296    Neg:    mov     DPH,R3          ;put desired in DPTR
83C8 8C82            297            mov     DPL,R4
83CA 8EF0            298            mov     B,R6            ;put actual in B/Acc
83CC EF              299            mov     A,R7
83CD 128457          300            call    util_subbad     ;DPTR = DPTR - B/Acc
83D0 75F000          301            mov     B,#0
83D3 74B4            302            mov     A,#0B4h
83D5 128457          303            call    util_subbad     ;if Desired-Actual < 180 jump
83D8 4027            304            jc      Right_Neg
83DA                 305    Left_Neg:
83DA D2FF            306            setb    Left_Flag
83DC 8EF0            307            mov     B,R6            ;put actual in B/Acc
83DE EF              308            mov     A,R7
83DF 758301          309            mov     DPH,#1          ;add 360 to actual
83E2 758267          310            mov     DPL,#067h
83E5 128448          311            call    util_adcbad     ;Actual+360->DPTR
83E8 8BF0            312            mov     B,R3            ;Put desired into
83EA EC              313            mov     A,R4            ;B/Acc
83EB 128457          314            call    util_subbad     ;get final difference
                     315                                    ;place it in DPTR
83EE 02840D          316            jmp     end_sub
                     317
83F1                 318    Left_Pos:
```

```
83F1 D2FF            319            setb    Left_Flag       ;set bit for left turn
83F3 8E83            320            mov     DPH,R6          ;put actual in DPTR
83F5 8F82            321            mov     DPL,R7
83F7 8BF0            322            mov     B,R3            ;put desired in B/Acc
83F9 8CF0            323            mov     B,R4
83FB 128457          324            call    util_subbad     ;get final difference
                     325                                    ;place it in DPTR
83FE 02840D          326            jmp     end_sub
                     327
8401                 328    Right_Neg:
8401 D2FE            329            setb    Right_Flag      ;set bit for right turn
8403 8B83            330            mov     DPH,R3          ;put desired in DPTR
8405 8C82            331            mov     DPL,R4
8407 8EF0            332            mov     B,R6            ;put actual in B/Acc
8409 EF              333            mov     A,R7
840A 128457          334            call    util_subbad     ;get final difference
                     335                                    ;place it in DPTR
840D                 336    end_sub:
840D D004            337            pop     04
840F 22              338            ret
                     339
8410                 340    load_actual:
8410 758381          341            mov     DPH,#Base
8413 758200          342            mov     DPL,#Actual_HI
8416 E0              343            movx    A,@DPTR         ;get high byte of compass
data
8417 FE              344            mov     R6,A            ;store it in R6
8418 A3              345            inc     DPTR            ;get low byte of data
8419 E0              346            movx    A,@DPTR         ;
841A FF              347            mov     R7,A            ;store low byte in R7
841B A3              348            inc     DPTR            ;
841C 22              349            ret
                     350
841D                 351    load_desired:
841D 758381          352            mov     DPH,#Base
8420 758202          353            mov     DPL,#Desired_HI
8423 E0              354            movx    A,@DPTR         ;get desired heading
8424 FB              355            mov     R3,A            ;high byte
8425 A3              356            inc     DPTR            ;get desired heading
8426 FC              357            mov     R4,A            ;low byte
8427 22              358            ret
                     359
8428                 360    get_actual:
8428 8E83            361            mov     DPH,R6          ;move actual into DPTR
842A 8F82            362            mov     DPL,R7
842C C002            363            push    02
842E C003            364            push    03
8430 C004            365            push    04
8432 C005            366            push    05
8434 C0D0            367            push    PSW
8436 1285C9          368            call    pint16u
8439 D0D0            369            pop     PSW
843B D005            370            pop     05
843D D004            371            pop     04
843F D003            372            pop     03
8441 D002            373            pop     02
8443 22              374            ret
                     375
8444                 376    get_desired:
```

```
8444 8BF0           377           mov     B,R3            ;move desired into B/Acc
8446 EC             378           mov     A,R4
8447 22             379           ret
                    380
8448                381   util_adcbad:
8448 C0E0           382           push    acc                     ; Make Sure Acc Gets
Saved
844A 2582           383           add     a,dpl                   ; Add 'A' To DPL
844C F582           384           mov     dpl,a                   ; Move Result To DPL
844E E583           385           mov     a,dph                   ; Get DPH
8450 35F0           386           addc    a,b                     ; Add 'B' To DPH + CY
8452 F583           387           mov     dph,a                   ; Move Bck To DPH
8454 D0E0           388           pop     acc                     ; Recover Original
'A'
8456 22             389           ret                             ; Return To Caller
                    390
8457                391   util_subbad:
8457 C0E0           392           push    acc                     ; Make Sure Acc Gets
Saved
8459 C3             393           clr     c                       ; Clear For SUBB
845A C582           394           xch     a,dpl                   ; Swap
845C 9582           395           subb    a,dpl                   ; Subtract
845E F582           396           mov     dpl,a                   ; Move Back To DPL
8460 E583           397           mov     a,dph                   ; Get DPH
8462 95F0           398           subb    a,b                     ; Subtract + CY
8464 F583           399           mov     dph,a                   ; Move Back To DPH
8466 D0E0           400           pop     acc                     ; Recover Original
'A'
8468 22             401           ret                             ; Return To Caller
                    402
                    403
                    404
8469 80FE           405   wait:   jmp     wait
                    406
                    407
846B                408   Comp_Init:
                    409           ;set all the bits on the compass
846B D2C7           410           setb    POW             ;turn power off
846D D2C6           411           setb    POLL            ;set the poll bit
846F D2C1           412           setb    SS              ;set the slave select bit
8471 D2C3           413           setb    RESET           ;set the NOT Reset line
8473 D2C4           414           setb    SCLK            ;set the shift clock
8475 D2C5           415           setb    SDO             ;setting SDO will set that
pin on the
                    416                                   ;8051 as an input
8477 D2C0           417           setb    EOC             ;same for EOC
                    418
8479 C2C7           419           clr     POW             ;turn power on
                    420
                    421           ;Initialize the counter for the compass process
controller
847B 7A00           422           mov     CompCount,#0
847D C2C3           423           clr     RESET           ;reset compass
847F 12856B         424           call    Delay10         ;wait 10msecs
8482 D2C3           425           setb    RESET           ;clear reset
8484 128574         426           call    lngDelay        ;wait .5secs
8487 D2C2           427           setb    MS              ;set compass as slave
8489 22             428           ret
                    429
848A                430   T2_Init:
                    431           ;load the timer to run the compass
848A C2CA           432           clr     TR2             ;turn Timer 2 off
848C 75CDB7         433           mov     TH2,#0B7h       ;load the T2 high byte
848F 75CCCB         434           mov     TL2,#0CBh       ;load the T2 low byte
```

```
8492 D2CA            435            setb    TR2             ;turn Timer 2 on
8494 22              436            ret
                     437
8495                 438    T2_ISR:
                =1   439            $INCLUDE(T2_ISR.INC)
8495 C0E0       =1   440            push    Acc
8497 C0D0       =1   441            push    PSW
8499 C007       =1   442            push    07
849B C006       =1   443            push    06
849D C004       =1   444            push    04
849F C003       =1   445            push    03
84A1 C082       =1   446            push    082h
84A3 C083       =1   447            push    083h
                =1   448
84A5 C2CF       =1   449            clr     TF2             ;Clear the T2 interrupt flag
84A7 C2CA       =1   450            clr     TR2             ;Stop the timer
84A9 75CDB7     =1   451            mov     TH2,#0B7h       ;Load the timer with the
value
84AC 75CCCB     =1   452            mov     TL2,#0CBh       ;needed for a 10msec interval
84AF D2CA       =1   453            setb    TR2             ;turn timer back on
84B1 0A         =1   454            inc     CompCount       ;Increment the process
counter
                =1   455
84B2 BA0103     =1   456            cjne    CompCount,#1,temp0
84B5 0284F3     =1   457            jmp     CLR_POLL
84B8 BA0203     =1   458    temp0:  cjne    CompCount,#2,temp1
84BB 0284F8     =1   459            jmp     SET_POLL
84BE BA0303     =1   460    temp1:  cjne    CompCount,#3,temp2
84C1 0284FD     =1   461            jmp     WAIT_EOC
84C4 BA0403     =1   462    temp2:  cjne    CompCount,#4,temp3
84C7 028505     =1   463            jmp     CLR_SS
84CA BA0503     =1   464    temp3:  cjne    CompCount,#5,temp4
84CD 02850A     =1   465            jmp     READ_SDO
84D0 BA0603     =1   466    temp4:  cjne    CompCount,#6,temp5
84D3 02850A     =1   467            jmp     READ_SDO
84D6 128589     =1   468    temp5:  call    Print
84D9 54686973   =1   469            db      'This didnt work!! \n\n',0,
84DD 20646964
84E1 6E742077
84E5 6F726B21
84E9 21205C6E
84ED 5C6E00
84F0 028549     =1   470            jmp     exit_T2
                =1   471
84F3            =1   472    CLR_POLL:
                =1   473
84F3 C2C6       =1   474            clr     POLL              ;clr POLL
84F5 028536     =1   475            jmp     end_T2
                =1   476
84F8            =1   477    SET_POLL:
84F8 D2C6       =1   478            setb    POLL              ;set POLL
84FA 028536     =1   479            jmp     end_T2
                =1   480
84FD            =1   481    WAIT_EOC:
84FD 20C036     =1   482            jb      EOC,end_T2        ;wait for the EOC flag to
be
8500 7A02       =1   483            mov     CompCount,#2      ;set
8502 028536     =1   484            jmp     end_T2
                =1   485
8505 C2C1       =1   486    CLR_SS: clr     SS                ; set slave select for data
recovery
8507 028536     =1   487            jmp     end_T2
```

```
                =1   488
                =1   489
850A 7E08       =1   490    READ_SDO:  mov     R6,#08
                =1   491
850C            =1   492    ReadSDO:
850C C2C4       =1   493            clr     SCLK
850E 128566     =1   494            call    clkDelay
8511 D2C4       =1   495            setb    SCLK
8513 7420       =1   496            mov     A, #020h        ;set A.5
8515 55C0       =1   497            anl     A,P4            ;compare A.5 with P4.5(SDO)
8517 6006       =1   498            jz      zero            ;jump if P4.5(EOC) is not set
                =1   499
8519 EC         =1   500            mov     A, DataByte     ;load R4 into A
851A D3         =1   501            setb    C               ;set the carry bit
851B 33         =1   502            rlc     A               ;roll A left with carry
851C FC         =1   503            mov     DataByte, A     ;put contents of A back into
R4
851D A123       =1   504            ajmp    rdDone          ; done with read
                =1   505
851F EC         =1   506    zero:   mov     A, DataByte     ;load R4 into A
8520 C3         =1   507            clr     C               ;clear the carry bit
8521 33         =1   508            rlc     A               ;roll A left with carry
8522 FC         =1   509            mov     DataByte, A     ;put contents of A back into
R4
                =1   510
8523 128566     =1   511    rdDone: call   clkdelay         ;delay for low clock
8526 DEE4       =1   512            djnz   R6, ReadSDO      ;this loop will execute 8
times
                =1   513
8528 BA050B     =1   514    done8:  cjne   CompCount,#5,end_T2
                =1   515                                    ;SDO_READ will execute when
                =1   516                                    ;CompCount = 5 & 6
                =1   517                                    ;done8 will only run when
CompCount=5
852B EC         =1   518            mov    A,DataByte       ;mov Data to A
852C 758381     =1   519            mov    DPH,#081h        ;first 8 bits will be saved
852F 758200     =1   520            mov    DPL,#000h        ;at 8100h
8532 F0         =1   521            movx   @DPTR,A          ;
8533 028536     =1   522            jmp    end_T2           ;end the routine
                =1   523
                =1   524
8536 BA0610     =1   525    end_T2: cjne    CompCount,#6,exit_T2
8539 D2C1       =1   526            setb    SS              ;Slave select off
853B EC         =1   527            mov     A,DataByte
853C 758381     =1   528            mov     DPH,#081h
853F 758201     =1   529            mov     DPL,#001h
8542 F0         =1   530            movx    @DPTR,A
8543 7A00       =1   531            mov     CompCount,#00
8545 758200     =1   532            mov     DPL,#00h
8548 E0         =1   533            movx    A,@DPTR
                =1   534    ;        mov     DPH,A
                =1   535    ;        mov     DPL,DATABYTE
                =1   536    ;        call    pint16u
                =1   537    ;        call    print
                =1   538    ;        db      '\',0,
                =1   539
                =1   540
                =1   541
8549            =1   542    exit_T2:
                =1   543    ;        call    Print
                =1   544    ;        db      'I',0,
8549 D083       =1   545            pop     083h
```

```
854B D082       =1   546            pop     082h
854D D003       =1   547            pop     03
854F D004       =1   548            pop     04
8551 D006       =1   549            pop     06
8553 D007       =1   550            pop     07
8555 D0D0       =1   551            pop     PSW
8557 D0E0       =1   552            pop     Acc
8559 32         =1   553            reti
                     554
                     555            ;initializes the programmable counter array for the
motor
                     556            ;and the steering servo
855A                 557    PCA_INIT:
855A 75D906          558            mov     CMOD,#006h      ;set PCA frequency to
external
855D 75DB42          559            mov     CCAPM1,#42h     ;put PCA module 1 into PWM
mode
8560 75DC42          560            mov     CCAPM2,#42h     ;put PCA module 0 into PWM
mode
8563 D2DE            561            setb    CR              ;start the PCA clock
8565 22              562            ret
                     563
8566 7F73            564    clkdelay: mov     R7,#073h
8568 DFFE            565    clk:      djnz    R7,clk
856A 22              566              ret
                     567
856B 7E25            568    delay10:  mov     R6,#025h
856D 7FFF            569    delay:    mov     R7,#0FFh
856F DFFE            570    delay1:   djnz    R7, delay1    ;wait >10 msec
8571 DEFA            571              djnz    R6, delay
8573 22              572              ret
                     573
8574                 574    lngDelay:
                =1   575            $INCLUDE(lngDelay.inc)
8574 7D0A       =1   576            mov   R5,#00Ah
8576 7E35       =1   577    delay2: mov     R6,#035h
8578 7FFF       =1   578    delay3: mov     R7,#0FFh
857A DFFE       =1   579    delay4: djnz    R7, delay4              ;wait >500msec
857C DEFA       =1   580            djnz    R6, delay3
857E DDF6       =1   581            djnz    R5, delay2
8580 22         =1   582            ret
                     583
8581                 584    cout:
                =1   585            $INCLUDE(cout.inc)
8581 3099FD     =1   586            JNB     TI,$            ;Wait until transmission
completed.
8584 C299       =1   587            CLR     TI              ;Clear interrupt flag.
8586 F599       =1   588            MOV     SBUF,A          ;Write out character.
8588 22         =1   589            RET
                     590
8589                 591    PRINT:
                =1   592            $INCLUDE(PRINT.inc)
8589 C0E0       =1   593            PUSH    ACC      ; push the A and DPTR registers so
their contents
858B C083       =1   594            PUSH    DPH      ; don't get lost in case they are
being used
858D C082       =1   595            PUSH    DPL      ; before calling this function.
                =1   596
                =1   597    ; move the stack pointer down low in order to pop the first
string's byte
                =1   598    ; address
858F E581       =1   599            MOV     A,SP
8591 24FD       =1   600            ADD     A,#0FDH
8593 F581       =1   601            MOV     SP,A
8595 D083       =1   602            POP     DPH
8597 D082       =1   603            POP     DPL
```

```
                =1   604
                =1   605    ; send (PRINT) the string back to the PC computer
8599            =1   606    READ_TX:
8599 E4         =1   607            CLR     A
859A 93         =1   608            MOVC    A,@A+DPTR
859B A3         =1   609            INC     DPTR
                =1   610
859C B45C0C     =1   611            CJNE    A,#'\',SND
                =1   612
859F E4         =1   613            CLR     A
85A0 93         =1   614            MOVC    A,@A+DPTR
85A1 A3         =1   615            INC     DPTR
                =1   616
85A2 B46E06     =1   617            CJNE    A,#'n',SND
                =1   618
85A5 740D       =1   619            MOV     A,#0DH
85A7 B181       =1   620            ACALL   cout
85A9 740A       =1   621            MOV     A,#0AH
                =1   622
85AB B181       =1   623    SND:    ACALL   cout
85AD B400E9     =1   624            CJNE    A,#0,READ_TX
                =1   625
                =1   626    ; place the new return addres in stack
85B0 C082       =1   627            PUSH    DPL
85B2 C083       =1   628            PUSH    DPH
                =1   629
                =1   630    ; move the stack pointer up high in order to pop the A and
DPTR registers
85B4 E581       =1   631            MOV     A,SP
85B6 2403       =1   632            ADD     A,#3
85B8 F581       =1   633            MOV     SP,A
85BA D082       =1   634            POP     DPL
85BC D083       =1   635            POP     DPH
85BE D0E0       =1   636            POP     ACC
85C0 22         =1   637            RET
                     638
85C1                 639    SEND_BYTE:
                =1   640            $INCLUDE(SEND_BYT.inc)
85C1 3099FD     =1   641            JNB    TI, SEND_BYTE           ; WAIT FOR THE TI FLAG
TO SET
85C4 C299       =1   642            CLR             TI                                  ;
CLEAR THE TI FLAG ONCE SENT
85C6 F599       =1   643                    MOV     SBUF,A                  ; SEND THE
CONTENTS OF REGISTER A
85C8 22         =1   644            RET                              ; RETURN
                =1   645
;*************************************************************************
                =1   646
                =1   647
                =1   648
                     649
85C9                 650    pint16u:
                =1   651            $INCLUDE(PINT16U.INC)
                =1   652            ;print 16 bit unsigned integer in DPTR, using base
10.
                =1   653            ;warning, destroys r2, r3, r4, r5, psw.5
85C9 C0E0       =1   654            push    acc
85CB E8         =1   655            mov     a, r0
85CC C0E0       =1   656            push    acc
85CE C2D5       =1   657            clr     psw.5
85D0 AA82       =1   658            mov     r2, dpl
85D2 AB83       =1   659            mov     r3, dph
                =1   660
85D4 7C10       =1   661    pint16a:mov     r4, #16 ;ten-thousands digit
```

```
85D6 7D27       =1   662            mov     r5, #39
85D8 D127       =1   663            acall   pint16x
85DA 6007       =1   664            jz      pint16b
85DC 2430       =1   665            add     a, #'0'
85DE 128581     =1   666            lcall   cout
85E1 D2D5       =1   667            setb    psw.5
                =1   668
85E3 7CE8       =1   669    pint16b:mov     r4, #232        ;thousands digit
85E5 7D03       =1   670            mov     r5, #3
85E7 D127       =1   671            acall   pint16x
85E9 7003       =1   672            jnz     pint16c
85EB 30D507     =1   673            jnb     psw.5, pint16d
85EE 2430       =1   674    pint16c:add     a, #'0'
85F0 128581     =1   675            lcall   cout
85F3 D2D5       =1   676            setb    psw.5
                =1   677
85F5 7C64       =1   678    pint16d:mov     r4, #100        ;hundreds digit
85F7 7D00       =1   679            mov     r5, #0
85F9 D127       =1   680            acall   pint16x
85FB 7003       =1   681            jnz     pint16e
85FD 30D507     =1   682            jnb     psw.5, pint16f
8600 2430       =1   683    pint16e:add     a, #'0'
8602 128581     =1   684            lcall   cout
8605 D2D5       =1   685            setb    psw.5
                =1   686
8607 EA         =1   687    pint16f:mov     a, r2           ;tens digit
8608 ABF0       =1   688            mov     r3, b
860A 75F00A     =1   689            mov     b, #10
860D 84         =1   690            div     ab
860E 7003       =1   691            jnz     pint16g
8610 30D505     =1   692            jnb     psw.5, pint16h
8613 2430       =1   693    pint16g:add     a, #'0'
8615 128581     =1   694            lcall   cout
                =1   695
8618 E5F0       =1   696    pint16h:mov     a, b            ;and finally the ones digit
861A 8BF0       =1   697            mov     b, r3
861C 2430       =1   698            add     a, #'0'
861E 128581     =1   699            lcall   cout
                =1   700
8621 D0E0       =1   701            pop     acc
8623 F8         =1   702            mov     r0, a
8624 D0E0       =1   703            pop     acc
8626 22         =1   704            ret
                =1   705
                =1   706    ;ok, it's a cpu hog and a nasty way to divide, but this code
                =1   707    ;requires only 21 bytes!  Divides r2-r3 by r4-r5 and leaves
                =1   708    ;quotient in r2-r3 and returns remainder in acc.  If Intel
                =1   709    ;had made a proper divide, then this would be much easier.
                =1   710
8627 7800       =1   711    pint16x:mov     r0, #0
8629 08         =1   712    pint16y:inc     r0
862A C3         =1   713            clr     c
862B EA         =1   714            mov     a, r2
862C 9C         =1   715            subb    a, r4
862D FA         =1   716            mov     r2, a
862E EB         =1   717            mov     a, r3
862F 9D         =1   718            subb    a, r5
8630 FB         =1   719            mov     r3, a
```

```
8631 50F6       =1   720            jnc     pint16y
8633 18         =1   721            dec     r0
8634 EA         =1   722            mov     a, r2
8635 2C         =1   723            add     a, r4
8636 FA         =1   724            mov     r2, a
8637 EB         =1   725            mov     a, r3
8638 3D         =1   726            addc    a, r5
8639 FB         =1   727            mov     r3, a
863A E8         =1   728            mov     a, r0
863B 22         =1   729            ret
                     730
                     731
                     732
                     733            end
```

VERSION 1.2h ASSEMBLY COMPLETE, 0 ERRORS FOUND

```
ACC. . . . . .  D ADDR  00E0H  PREDEFINED
ACON . . . . . .  D ADDR  0097H  PREDEFINED
ACTUAL_HI. . . . . .    NUMB  0000H
ACTUAL_LO. . . . . .    NUMB  0001H  NOT USED
AD0. . . . . .  D ADDR  0084H  PREDEFINED
AD1. . . . . .  D ADDR  0094H  PREDEFINED
AD2. . . . . .  D ADDR  00A4H  PREDEFINED
AD3. . . . . .  D ADDR  00B4H  PREDEFINED
AHD_HLF. . . . . .  C ADDR  834EH
AHD_SLOW . . . . . .  C ADDR  8370H
ALL_STOP . . . . . .  C ADDR  838EH
B. . . . . .  D ADDR  00F0H  PREDEFINED
BASE . . . . . .    NUMB  0081H
CCAP1H . . . . . .  D ADDR  00FBH  PREDEFINED
CCAP2H . . . . . .  D ADDR  00FCH  PREDEFINED
CCAPM1 . . . . . .  D ADDR  00DBH  PREDEFINED
CCAPM2 . . . . . .  D ADDR  00DCH  PREDEFINED
CLK. . . . . .  C ADDR  8568H
CLKDELAY . . . . . .  C ADDR  8566H
CLR_POLL . . . . . .  C ADDR  84F3H
CLR_SS . . . . . .  C ADDR  8505H
CMOD . . . . . .  D ADDR  00D9H  PREDEFINED
COMPASS. . . . . .  C ADDR  8392H
COMPASS_CALC . . . . . .  C ADDR  82EFH
COMPCOUNT. . . . . .    REG2
COMP_INIT. . . . . .  C ADDR  846BH
COMP_STR . . . . . .  C ADDR  832DH
COUT . . . . . .  C ADDR  8581H
CR . . . . . .  B ADDR  00DEH  PREDEFINED
DATABYTE . . . . . .    REG4
DELAY. . . . . .  C ADDR  856DH
DELAY1 . . . . . .  C ADDR  856FH
DELAY10. . . . . .  C ADDR  856BH
DELAY2 . . . . . .  C ADDR  8576H
DELAY3 . . . . . .  C ADDR  8578H
DELAY4 . . . . . .  C ADDR  857AH
DESIRED_HI . . . . . .    NUMB  0002H
DESIRED_LO . . . . . .    NUMB  0003H  NOT USED
DONE8. . . . . .  C ADDR  8528H  NOT USED
DPH. . . . . .  D ADDR  0083H  PREDEFINED
DPL. . . . . .  D ADDR  0082H  PREDEFINED
END_SUB. . . . . .  C ADDR  840DH
END_T2 . . . . . .  C ADDR  8536H
EOC. . . . . .    NUMB  00C0H
EXIT_T2. . . . . .  C ADDR  8549H
FOR2REV. . . . . .  C ADDR  82C1H
FOR_REV. . . . . .    NUMB  00F8H
FULL_LEFT. . . . . .    NUMB  0030H
FULL_RIGHT . . . . . .    NUMB  004DH
FWD_HI . . . . . .    NUMB  00B0H
FWD_LO . . . . . .    NUMB  00C0H
FWD_MED. . . . . .    NUMB  00B0H
GET_ACTUAL . . . . . .  C ADDR  8428H
GET_DESIRED. . . . . .  C ADDR  8444H
HALF_LEFT. . . . . .    NUMB  0037H
HALF_RIGHT . . . . . .    NUMB  0045H
LEFT_FLAG. . . . . .    NUMB  00FFH
LEFT_FULL. . . . . .  C ADDR  8309H  NOT USED
```

LEFT_HALF.	C	ADDR	8316H	NOT USED
LEFT_NEG	C	ADDR	83DAH	NOT USED
LEFT_POS	C	ADDR	83F1H	
LEFT_QTR	C	ADDR	8323H	NOT USED
LNGDELAY	C	ADDR	8574H	
LOAD_ACTUAL.	C	ADDR	8410H	
LOAD_DESIRED	C	ADDR	841DH	
L_FRONT.	C	ADDR	82A4H	
L_REAR	C	ADDR	8269H	NOT USED
MAIN	C	ADDR	8200H	NOT USED
MS		NUMB	00C2H	
NEG.	C	ADDR	83C6H	
P1	D	ADDR	0090H	PREDEFINED
P4	D	ADDR	00C0H	PREDEFINED
P5	D	ADDR	00F8H	PREDEFINED
PCA_INIT	C	ADDR	855AH	
PINT16A.	C	ADDR	85D4H	NOT USED
PINT16B.	C	ADDR	85E3H	
PINT16C.	C	ADDR	85EEH	
PINT16D.	C	ADDR	85F5H	
PINT16E.	C	ADDR	8600H	
PINT16F.	C	ADDR	8607H	
PINT16G.	C	ADDR	8613H	
PINT16H.	C	ADDR	8618H	
PINT16U.	C	ADDR	85C9H	
PINT16X.	C	ADDR	8627H	
PINT16Y.	C	ADDR	8629H	
POLL		NUMB	00C6H	
POW.		NUMB	00C7H	
PRINT.	C	ADDR	8589H	
PSW.	D	ADDR	00D0H	PREDEFINED
PWM_MOT.		REG0		
PWM_STR.		REG1		
QTR_LEFT		NUMB	003AH	NOT USED
QTR_RIGHT.		NUMB	0042H	NOT USED
RDDONE	C	ADDR	8523H	
READSDO.	C	ADDR	850CH	
READ_SDO	C	ADDR	850AH	
READ_TX.	C	ADDR	8599H	
RESET.		NUMB	00C3H	
REV2FOR.	C	ADDR	8276H	NOT USED
REV_LO		NUMB	0090H	
RIGHT_FLAG		NUMB	00FEH	
RIGHT_FULL	C	ADDR	830EH	
RIGHT_HALF	C	ADDR	831BH	
RIGHT_NEG.	C	ADDR	8401H	
RIGHT_POS.	C	ADDR	83AFH	NOT USED
RIGHT_QTR.	C	ADDR	8328H	
R_FRONT.	C	ADDR	82B4H	
SBUF	D	ADDR	0099H	PREDEFINED
SCAN	C	ADDR	8266H	
SCLK		NUMB	00C4H	
SDO.		NUMB	00C5H	
SEND_BYTE.	C	ADDR	85C1H	
SET_POLL	C	ADDR	84F8H	
SND.	C	ADDR	85ABH	
SONAR.	C	ADDR	8331H	NOT USED
SONAR_1.		NUMB	001AH	NOT USED

```
SONAR_12 . . . . . .    NUMB  00D0H  NOT USED
SONAR_15 . . . . . .    NUMB  00F5H
SONAR_3. . . . . .      NUMB  0033H
SONAR_6. . . . . .      NUMB  0067H  NOT USED
SONAR_9. . . . . .      NUMB  009BH
SP . . . . . .    D ADDR  0081H  PREDEFINED
SS . . . . . .          NUMB  00C1H
STP_GO . . . . . .      NUMB  00F9H
STP_REV0 . . . . . .  C ADDR  82EDH
STP_REV2 . . . . . .  C ADDR  82A2H
STRAIGHT . . . . . .    NUMB  003FH
T2_INIT. . . . . .    C ADDR  848AH
T2_ISR . . . . . .    C ADDR  8495H
TEMP0. . . . . .      C ADDR  84B8H
TEMP1. . . . . .      C ADDR  84BEH
TEMP2. . . . . .      C ADDR  84C4H
TEMP3. . . . . .      C ADDR  84CAH
TEMP4. . . . . .      C ADDR  84D0H
TEMP5. . . . . .      C ADDR  84D6H
TF2. . . . . .    B ADDR  00CFH  PREDEFINED
TH2. . . . . .    D ADDR  00CDH  PREDEFINED
TI . . . . . .    B ADDR  0099H  PREDEFINED
TL2. . . . . .    D ADDR  00CCH  PREDEFINED
TR2. . . . . .    B ADDR  00CAH  PREDEFINED
TURNGO . . . . . .    C ADDR  82F8H
UNDR_10. . . . . .    C ADDR  8320H
UNDR_25. . . . . .    C ADDR  8313H
UTIL_ADCBAD. . . . . .  C ADDR  8448H
UTIL_SUBBAD. . . . . .  C ADDR  8457H
WAIT . . . . . .      C ADDR  8469H
WAIT_EOC . . . . . .  C ADDR  84FDH
ZERO . . . . . .      C ADDR  851FH
```