EC430 PIC Embedded Control Project: "Smart Car"

Kyle Aubrey & Jonathan Coppock
Group #1
Box 1387 & Box 1435

Department of Electrical and Computer Engineering

Rose-Hulman Institute of Technology

EC430: Microcomputers
Dr. Jianjian Song

February 21, 2000

TABLE OF CONTENTS:

INTRODUCTION:

The objective of our PIC embedded control project for EC430 was to develop and implement a “Smart Car”, which would be able to follow a black line on the floor using an RC car from Radio Shack, a PIC microcontroller, two IR emitter/detectors, and other various hardware.

The motivation for doing this particular project was a request by Van Cottom of the ECE department. He requested that our group work on this project, as it is hoped that the “Smart Car” can be used as a project for Rose-Hulman’s Catapult Program. Van Cottom funded the cost of parts for this project.

This report will summarize the successful completion of this project. Topics to be covered include the internal workings, the user’s manual, documented code, flowcharts, and a bill of materials.

INTERNAL WORKINGS:

Strategy:

The strategy of this project was to use one PIC microcontroller to control operation of the “Smart Car”. The PIC microcontroller takes in two IR inputs, which allow for interrupting in the software. Also, the PIC has two outputs, which are used to drive the motors of the RC car, using variable duty cycle square waves. The outputs of the PIC cannot go directly to the motors, as they require high current. Consequently, the PIC outputs are used to switch two MOSFET transistors. Then, the MOSFET’s are connected to the RC car’s motors.

The car is designed to move forward. The normal forward speed of the car uses a 90% duty cycle. To turn the car, one of the tracks are slowed down to a 30% duty cycle. For example, to turn the car left while both tracks are at 90% duty cycle, lower the left track to a 30% duty cycle. Then, increasing the duty cycle back to 90% will make the car go straight again. The high and low duty cycles are parameters in the assembly code that can be easily changed.

The interrupt service routine is called by a change on one of the IR input sensors. The purpose of the interrupt service routine is to change the duty cycle as necessary. The duty cycle is implemented by dividing a cycle into 10 segments. Then, a delay can be created to represent one piece, or 1/10 of a complete cycle. Then, if a 30% duty cycle is desired, the line is asserted high while three of the delays occur, and returns to low for the remaining 7 delays, giving one complete cycle with a 30% duty cycle.

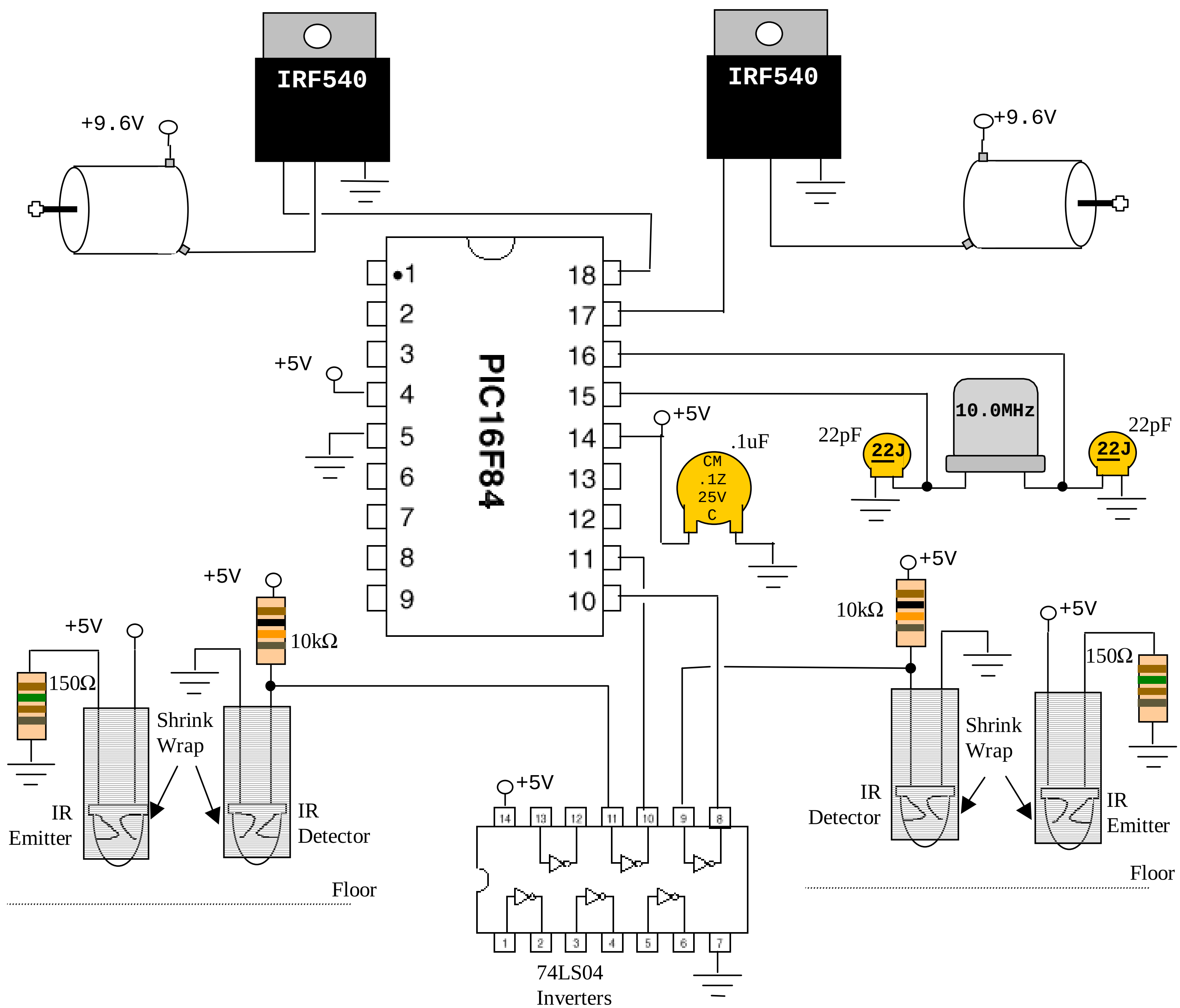
Specification:

1. The “Smart Car” will follow a line of black tape.
2. The car will have dual motors, one for both left and right tracks.
3. The motors will be speed controlled by high power MOSFET’s using variable duty cycles.
4. The battery for the car will be a 9.6V NiCad rechargeable battery.
5. A 5V regulator will be used to reduce the 9.6V to the 5V used by the PIC.
6. Two IR emitter/detectors will be used to sense the black line.
7. The IR emitter/detectors will be in the front of the car and approximately 4 inches apart.
8. If the right IR senses the black tape, the right motor will be issued a lower duty cycle to slow the right side down until the IR does not sense the tape any more. It will then be issued the normal higher duty cycle. This also applies to the left side.

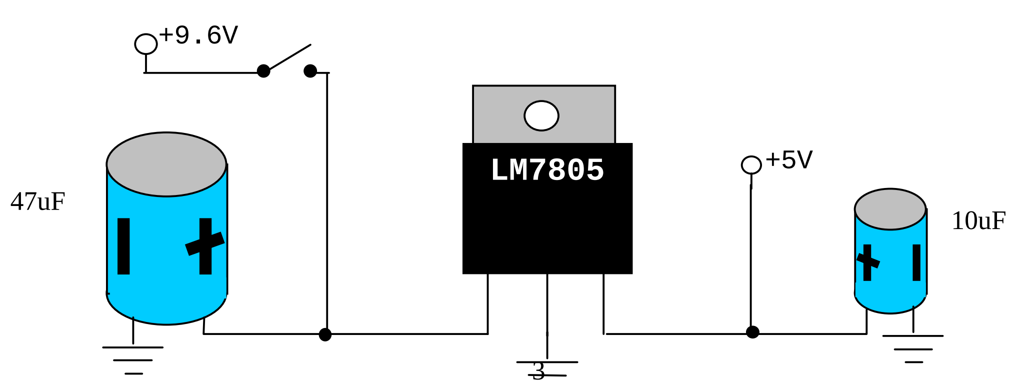
Hardware Schematic:

The hardware schematic can be found on the next page. There are a total of five capacitors, including bypass capacitors for power sources and the capacitors for the 10MHz crystal oscillator. There are also two IR emitters and two IR detectors. The IR emitters include 150Ω resistors to allow the proper voltage level for operation. The IR detectors include 10kΩ pull-up resistors. The output of the IR detectors pass through inverters, before connecting to the PIC. The dual motors of the RC car are powered by 9.6V, and pass through switching MOSFET’s before connecting to the PIC. The hardware design also includes a voltage regulator to step 9.6V down to 5V.

LEFT SIDE **RIGHT SIDE**



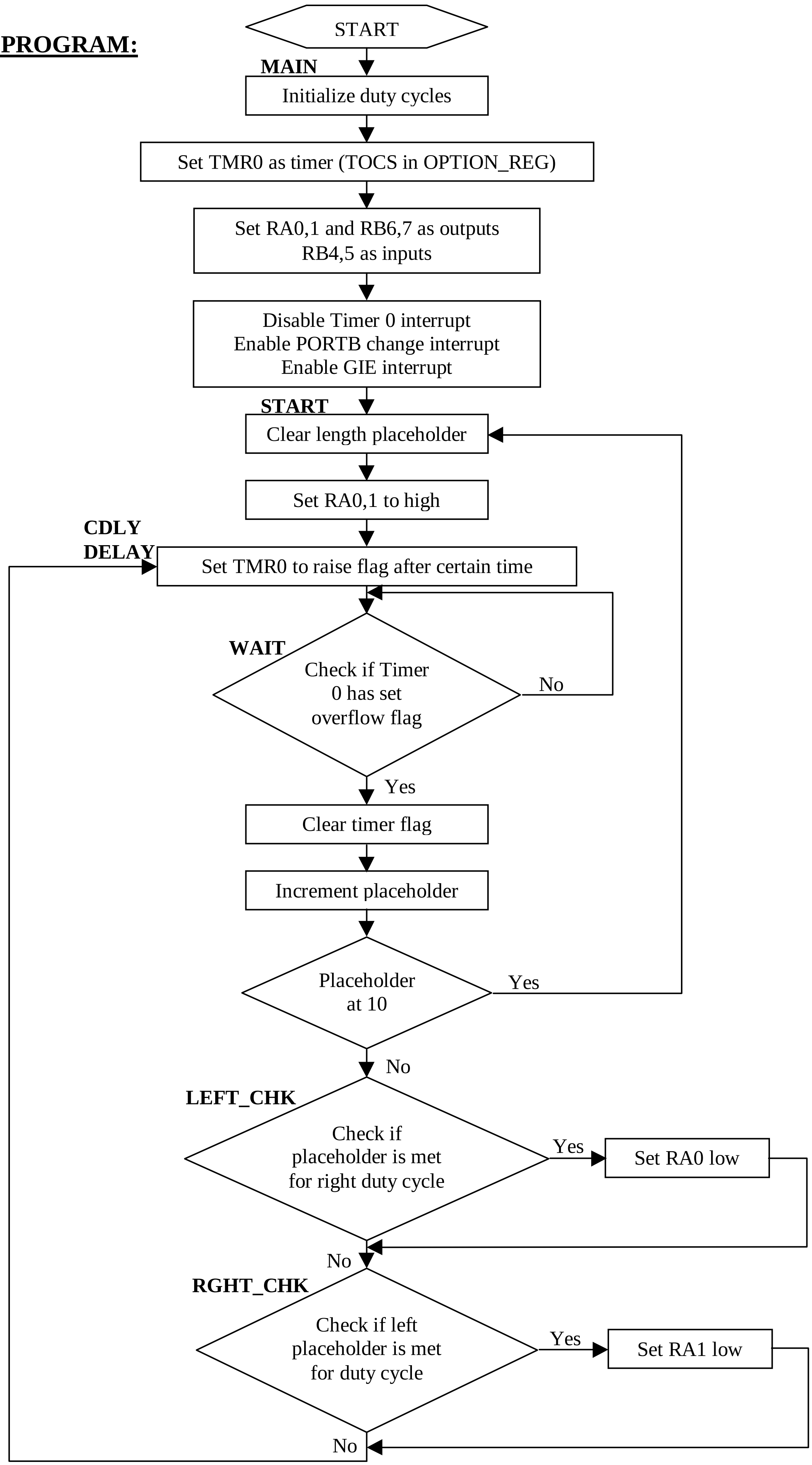
9.6V to 5.0V Regulation



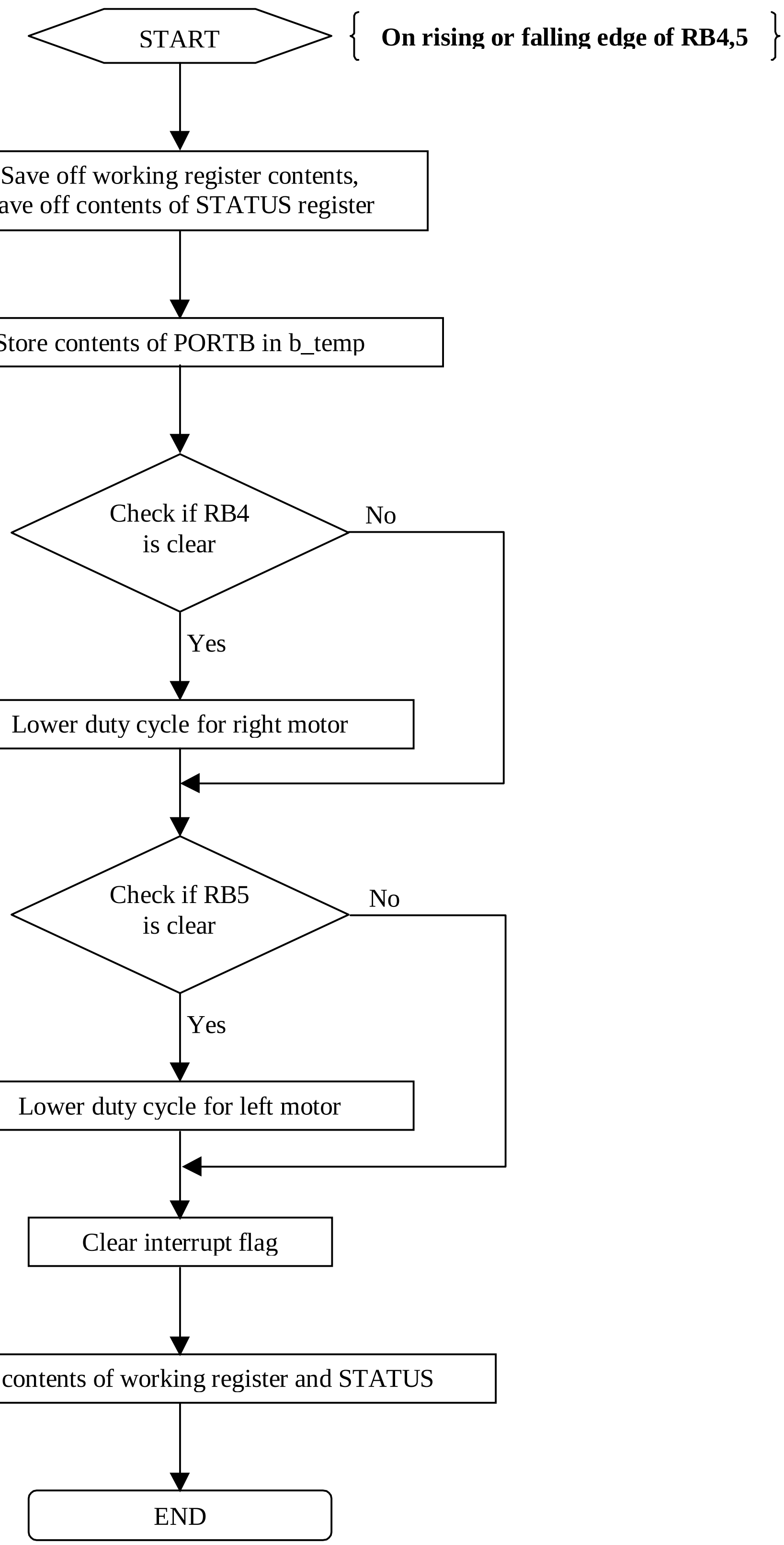
Software Flowcharts:

The detailed software flowcharts can be found on the next two pages. The first page includes the software flowchart for the MAIN program. The second page includes the software flowchart for the CHANGE_DUTY interrupt service routine, which changes the duty cycle of either the left or right track's motor when a change is detected on one of the IR detector inputs of the PIC.

MAIN PROGRAM:



CHANGE_DUTY:


START
On rising or falling edge of RB4,5
Save off working register contents,
Save off contents of STATUS register
Store contents of PORTB in b_temp
Check if RB4
is clear
No
Yes
Lower duty cycle for right motor
Check if RB5
is clear
No
Yes
Lower duty cycle for left motor
Clear interrupt flag
Return contents of working register and STATUS
END

USER’S MANUAL:

The “Smart Car” is very simple to use. To start the car, simply switch the power to ON. The switch can be found under the car, at the back of the car.

Start the car so that it is positioned in the direction of the part of the course that it is going to begin on. This will give the car the best chance of finding the course and getting off to a good start.

For best performance, the course should be made of electrical tape that is double-wide. Also, the floor color should be considerably lighter than the tape, preferably white or a very light reflective color.

Starting and stopping blocks can be made on the course simply by creating large black squares that the sensors can both fit inside of. Whenever both of the car’s sensors are over black, both of the motors are stopped. If the motors are stopped for long enough by a large enough black square, the car can be completely stopped. Smaller areas of black can be used as a "speed bump” of sorts. For example wide strips of black can be placed perpendicular to the course before turns to slow the car down. Also, if a turn is too sharp for the car, black tape can be concentrated on the inside of the curve to help the car corner. This works because it slows down one side of the car longer, helping it to corner more sharply.

BILL OF MATERIALS:

Part	**Quantity**	**Price/Part X Quantity**
RC Car	1	$40.00
9.6V NiCad Battery	1	$25.00
Battery Recharger	1	$7.00
PIC16F84	1	$5.00
10MHz Crystal	1	$2.00
IRF540 MOSFET	2	$3.00
74LS04 Hex Inverter	1	$0.25
150Ω Resistor	2	$0.20
10kΩ Resistor	2	$0.20
0.1μF Capacitor	1	$0.25
22pF Capacitor	2	$0.50
Electrolytic Capacitor	2	$0.50
LM7805 5V Regulator	1	$1.00
IR Emitter/Detector	2	$1.00
	TOTAL COST:	**$85.90**

RECOMMENDATIONS:

Our group recommends that a similar RC car be used by the Catapult Program. However, we would not recommend the type of RC car with two rubber tracks. Also, the gear ratio of our car was geared for high speed, rather than greater torque, but the car we used like to overshoot curves.

The next car should probably have wheels, rather than tracks. We found that for this application of the RC car, the tracks provide too much friction. This makes it difficult for the car to change speeds quickly and to turn accurately.

Also, the next car should also have a gear ratio that provides greater torque and less speed. Again, this will allow the car to change speed more quickly and smoothly, as well as allow the car to turn more accurately.

ACKNOWLEDGMENTS:

We would like to thank Van Cottom for allowing us the opportunity to work on this exciting project. It has been an excellent learning experience in embedded design. We are also grateful to Van Cottom for providing full financial funding of the parts used in this design.

Also, we would like to thank Dr. Jianjian Song for his guidance on this project. His advice and consultation was extremely helpful for successful completion of this project.

APPENDIX (Documented Assembly Code):

```
;*********************************************************************
; SMART CAR --- This program runs on the PICmicro PIC16F84.  The PIC
;               is used as a double duty cycle controller for a dual
;               motor RC car.  It begins with a high duty cycle and
;               when an interrupt is received from one of the two IR
;               sensors on the front of the car, it issues a lower
;               duty cycle to that side of the car to cause the car
;               to turn. When the interrupt returns to normal the
;               duty cycle is returned to high again to allow the
;               car to go straight.
;
;*********************************************************************
;
;    Filename:      SMARTCAR.ASM
;    Date:          January 27 - February 14, 2000
;    File Version:  REV. A
;
;    Authors:       Kyle Aubrey & Jonathan Coppock
;    Class:         EC430 - MICROCONTROLLERS
;
;
;*********************************************************************
;
;
;    Files required:  p16F84.inc
;
;*********************************************************************
; Test   Test     Meas.     Meas.Rgt     Meas.Lft     Meas.     Error
;        Duty     Freq.     HighTime     HighTime    Duty R/L
;  (1)   90%     8.91kHz     98.80uS     102.00uS   88.5/91.4   1.6/ 1.6%
;  (2)   60%     8.49kHz     63.20uS      66.40uS   53.0/55.7  11.6/ 7.2%
;  (3)   30%     7.35kHz     32.00uS      35.20uS   23.5/25.9  13.6/21.6%
;  (4)   10%     7.00kHz      9.20uS      12.40uS    6.4/ 8.6  36.0/14.0%
;
;
;
;
;*********************************************************************


      list      p=16F84             ; list directive to define processor
      #include <p16F84.inc>         ; processor specific variable definitions

      __CONFIG   _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

; '__CONFIG' directive is used to embed configuration data within .asm file.
; The labels following the directive are located in the respective .inc file.

;***** DUTY CYLE CONSTANTS********************************************
;Change from .1 to .10 to change duty cycle from 10% to 100%          *
;*********************************************************************

CONSTANT            high_duty_cycle=.9
CONSTANT            low_duty_cycle=.3

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

;***** VARIABLE DEFINITIONS******************************************
w_temp            EQU     0x0C      ; variable used for context saving
status_temp       EQU     0x0D      ; variable used for context saving
b_temp            EQU     0x0E      ; variable used for saving PORTB
length            EQU     0x0F      ; variable used as a place holder
duty_             EQU     0x10      ; variable for saving right duty cycle
```

```
duty_l          EQU     0x11     ; variable for saving left duty cycle
;*************************************************************************

            ORG     0x000
            goto    main            ;Skip over ISR
            ORG     0x004           ;Interrupt vector location
            goto    change_duty

;***** INITIALIZATION OF PIC*******************************************

main        movlw   high_duty_cycle
            movwf   duty_r          ;Initialize duty cycles
            movwf   duty_l

;***** PORT A & B pin and Timer0 setup*********************************
            BSF     STATUS,RP0
            BCF     OPTION_REG,T0CS ;Timer transition on Int. clock
            BCF     TRISA,0         ;set RA0,1 as outputs for two motors
            BCF     TRISA,1         ;(0->R) (1->L)
            BCF     TRISB,7         ;set RB7,6 as outputs so they
            BCF     TRISB,6         ;will not interrupt
            BSF     TRISB,5
            BSF     TRISB,4         ;set RB5,4 as inputs for interrupts
            BCF     STATUS,RP0

;***** set up interrupts***********************************************

            BCF     INTCON,T0IE     ;disable Timer0 interrupt
            BCF     INTCON,RBIF     ;clear PORTB change interrupt flag
            BSF     INTCON,RBIE     ;enable PORTB change interrupt
            BSF     INTCON,GIE      ;enable global interrupt

;***** START OF MAIN PROGRAM*******************************************

start       clrf    length          ;set length place holder to zero
            movlw   .3              ;set RA0,1 to high
            movwf   PORTA

cdly        call    delay           ;call a delay dependant on timer
            incf    length,f        ;after delay, increment place holder one
            movlw   .10
            subwf   length,w        ;check to see if place holder is at end
            btfsc   STATUS,Z        ;of length of 10
            goto    start           ;if so start over

rght_chk    movf    duty_r,w        ;If not at the end, check to see if duty
            subwf   length,w        ;cycle of the right side has been met, if
            btfsc   STATUS,C        ;so then lower output, else keep high and
            call    downr           ;check left side
left_chk    movf    duty_l,w        ;Check other duty cycle on the left side,
            subwf   length,w        ;if met lower duty cycle on that side, lower
            btfsc   STATUS,C        ;output, else keep high and keep calling
            call    downl           ;delays and checking to see if duty cycles
            goto    cdly            ;have been met.

downr       bcf     PORTA,0         ;Lower right signal line
            return
downl       bcf     PORTA,1         ;Lower left signal line
            return
;***** DELAY LOOP******************************************************
delay       movlw   .254            ;To setup an overall 10uS delay
            MOVWF   TMR0            ;causing an approx. 10kHz signal
wait        BTFSS   INTCON,T0IF     ;check if Timer0 overflow
            GOTO    wait
```

```
            BCF     INTCON,T0IF     ;Clear Timer Flag
            return

;*******************************************************************
;    Interrupt service routine                                       *
;*******************************************************************
change_duty
            movwf   w_temp          ; save off current W register contents
            movf    STATUS,w        ; move status register into W register
            movwf   status_temp     ; save off contents of STATUS register

            MOVF    PORTB,w         ; Store contents of PORTB in b_temp
            MOVWF   b_temp
check_1     btfss   b_temp,4        ; Check Right Interrupt, if clear (black)
            goto    du_lowr         ; lower duty cycle to slow on right motor,
            movlw   high_duty_cycle ; else keep normal
            movwf   duty_r
check_2     btfss   b_temp,5        ; Check Left Interrupt, if clear (black)
            goto    du_lowl         ; lower duty cycle to slow on left motor,
            movlw   high_duty_cycle ; else keep normal
            movwf   duty_l

            BCF     INTCON,RBIF     ; clear interrupt flag
            movf    status_temp,w   ; retrieve copy of STATUS register
            movwf   STATUS          ; restore pre-isr STATUS register contents
            swapf   w_temp,f
            swapf   w_temp,w        ; restore pre-isr W register contents
            retfie

du_lowr     movlw   low_duty_cycle  ; Lower duty cycle for right motor
            movwf   duty_r
            goto    check_2         ; Check other interrupt
du_lowl     movlw   low_duty_cycle  ; Lower duty cycle for left motor
            movwf   duty_l

            BCF     INTCON,RBIF     ; clear interrupt flag
            movf    status_temp,w   ; retrieve copy of STATUS register
            movwf   STATUS          ; restore pre-isr STATUS register contents
            swapf   w_temp,f
            swapf   w_temp,w        ; restore pre-isr W register contents
            retfie                  ; return from interrupt

            END
```