Date: 8/8/02
Student Name: Victor Mitselmakher

TA: Tae Choi,
Uriel Rodriguez
Instructors: Dr. A. A Arroyo
Dr. Schwartz

University of Florida
Department of Electrical and Computer Engineering
Intelligent Machines Design Laboratory
EEL5666

Self-Centering Autonomous Bulldozer
(SCAB)

TABLE OF CONTENTS

ABSTRACT

The Self-Centering Autonomous Bulldozer (SCAB) is a tireless worker which is designed to navigate in an environment populated with movable objects inside of an arena or on a table top. The robot will move objects away from the perceived center of the arena and reposition itself in the middle of the arena until all objects have been moved to their furthest possible location.

EXECUTIVE SUMMARY

The SCAB Robot was designed to push objects and act as an "intelligent" auto-controlled bulldozer. The SCAB Robot uses the Rabbit 2000 processor integrated into a single board computer called the Jackrabbit BL1810. The SCAB robot was originally envisioned to use a sonar, locate and push objects clearing an area. However the sonar proved to be faulty and the robot was re-designed to work with long range IR sensors. The major challenge behind the design and implementation of the SCAB was the use of the Rabbit 2000 processor. To my knowledge this is the first robot based on the Rabbit 2000 with sophisticated PWM control and behavior. Only one other robot based on the Rabbit 2000 showed up after doing Google.com searches, and its source code was not published and it appeared to be incomplete. I believe that the lessons learned from the SCAB can be used to build a much more sophisticated Robot based on the Rabbit 2000, incorporating some of the more interesting features of the chip, such as internet access. I encourage others to consider using the Rabbit 2000 or its successor the Rabbit 3000 for their robots and building on the SCAB source code.

INTRODUCTION

The intention of this robot is to act as a bulldozer, detecting objects and pushing them away, clearing a circular or rectangular arena of all objects. Once the robot is powered, it uses ranging infra red sensors to find the closest object. The robot includes a behavior to help filter detection of objects and separate them from detecting a wall. The closest detectable object is then pushed away as far as possible. The robot then returns to the previously calculated center and repeats the behavior. A variety of supporting sensors and actuation methods are used to support and enhance this behavior.

INTEGRATED SYSTEM

The integrated system is composed of a round wooden platform based on the Talrik Junior design. The robot is controlled by a Rabbit 2000 processor which is integrated into a single board computer called Jackrabbit 1810.

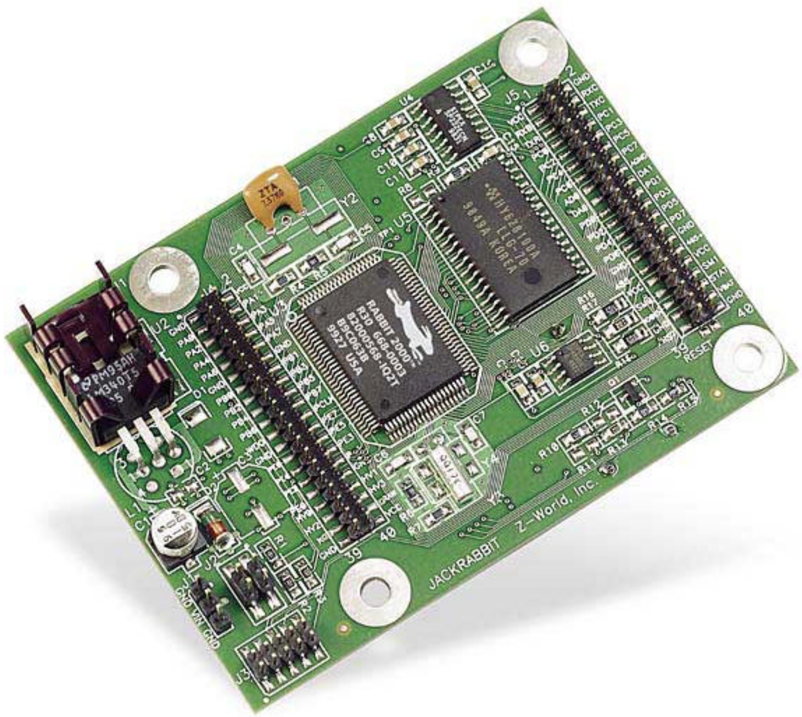

Figure 1: Jackrabbit BL1810 SBC using the Rabbit 2000 Processor

The mechanical part consists of two Futaba S3001 servo motors used for rotating the robot wheels. The wheels are salvaged from an R/C toy car. The robot is equipped with a main power switch, a push button, electronic compass CMPS003 made by Devantech, two types of IR sensors, the Sharp GP2D12s which are used for edge detection and the long range Sharp GP2Y0A0 for forward object detection. Furthermore, two Hamamatsu P5587 shaft encoders are mounted to count the number of wheel rotations. A polaroid 6500 series sonar was planned in the original design, however it proved to be faulty and was excluded from the final version. Two extra prototyping boards are used, one containing the MAXIM 127, an 8 channel A/D converter using the I2C interface and three voltage regulators, the second protoboard is mounted on the top of the robot and contains the push button and buzzer. An 18WH lithium-ion polymer battery is used to power the robot and provides over an hour of runtime.

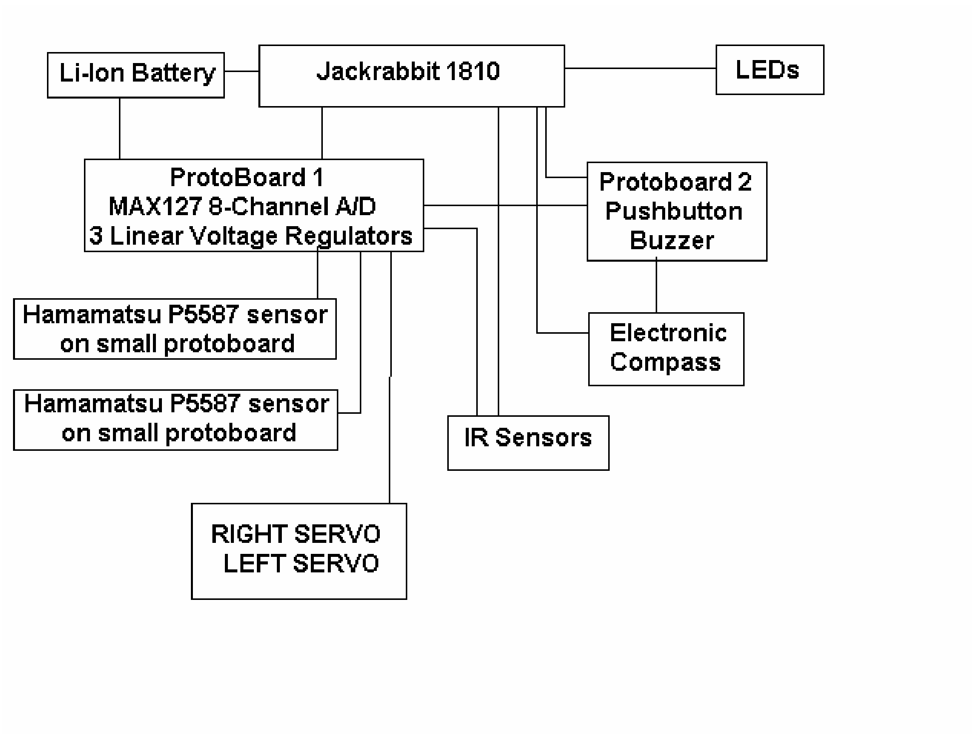

Figure 2: Integrated System

MOBILE PLATFORM

The mobile platform used is based on the Talrik Junior platform and was cut out on the T-tech machine. A round shape was chosen because of its symmetric nature which allows the robot to rotate with a reduced possibility of clipping something, as well as allowing the sensors to have optimal positioning. Extra modifications were added to the platform, such as special metal IR sensor mounts, the shaft encoder mounts, an electronic compass stand, and a hard yellow plastic bulldozer shovel in the front.

ACTUATION

The robot is propelled by two Futaba S3001 servos modified to allow free rotation. The wheels contain a shaft encoder to allow for travel distance measurement. A special challenge in the actuation of the robot was the implementation of the PWM signals used to control the servos, since the Rabbit 2000 does not contain a PWM driver. The Rabbit 2000 Timer system, serial port timer prescalers and a custom interrupt routine are used to generate PWM signals with a precision of 11.7μs.

SENSORS

The sensors play a crucial role in the design of the robot, since they act as the only means of telling the robot about its environment. Therefore a careful selection and calibration of sensors is necessary to accomplish the desired behaviors. The original design of the robot included a sonar which was to be used for object detection and ranging. However the sonar was later abandoned because it was not functioning. Two IR sensors are used instead of the sonar with an effective maximum distance of 250cm. The IR sensors proved a good choice because of their extremely narrow beam properties and fairly accurate distance measurements. Two Hamamatsu P5587 shaft encoders and a Devantech CMPS003 electronic compass are also used to help navigation. Additionally two more IR sensors are used for edge detection.

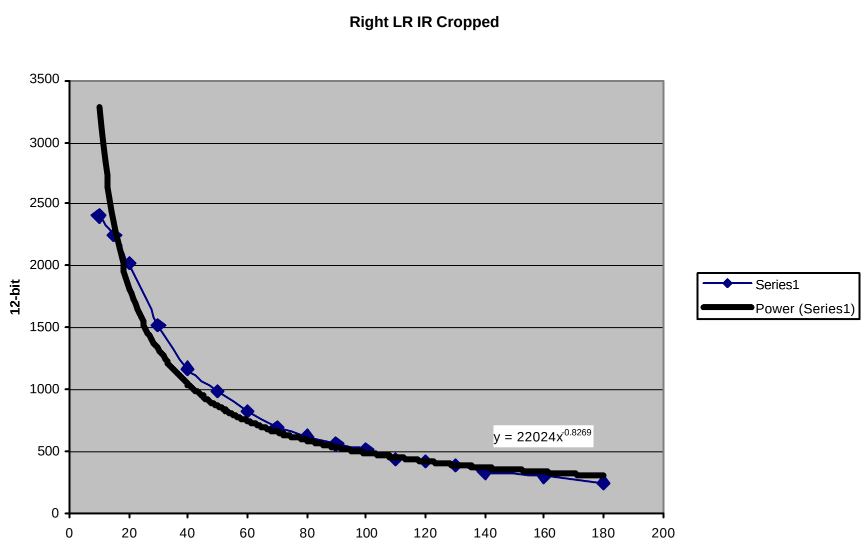

Figure 3: Long Range IR Output vs Distance

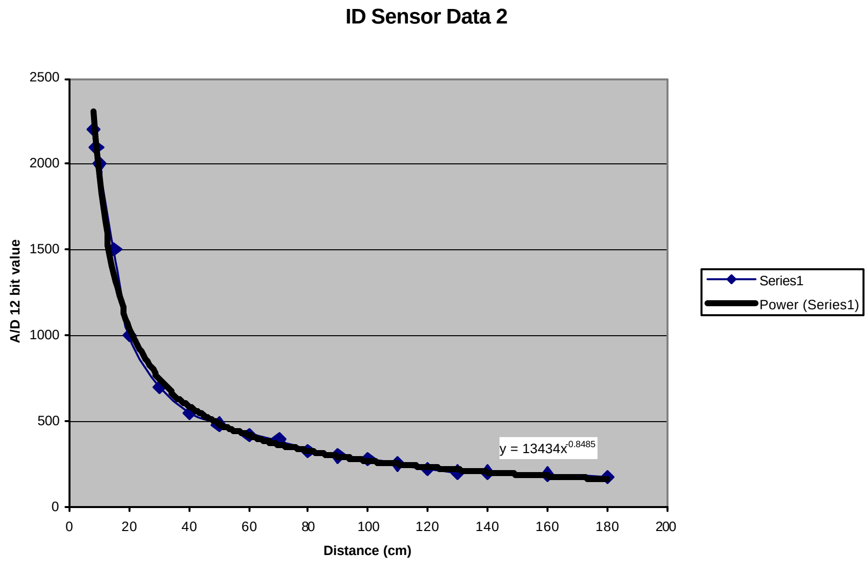

Figure 4: Edge Detector IR Output vs. Distance

BEHAVIORS

The robot includes several components to its behavior which interact to form its complete autonomous behavior. When started, the robot will perform a 360 degree rotation, employing its long range IR sensors to scan all objects in the vicinity of the IR range and determine the direction towards the closest object, as well as the location of the present geometrical center of clear space around the objects. Next the robot will move towards the closest object and use IR sensors to detect when it approaches the object and slow down. The robot will push the object forward until it runs into a wall, detects an edge, or when the object slips from the bulldozer shovel, at which point the robot will turn around and return to the center location calculated during its 360 degree scan. Once the robot returns to the location, it will perform another rotational scan and push the closest object. It will continue to behave in this manner until all objects have been pushed away to their maximum distance and the maximum amount of clear space is achieved.

CONCLUSION

The experience of designing and building the SCAB was a real challenge. Using the Rabbit 2000 processor for the control of the robot proved to be quite difficult because there is absolutely no example code available to help with this task. After extensive internet searches I was able to locate only one other robot based on the Rabbit 2000, and it was incomplete and its code was not published. Therefore I had to design and code the robot from scratch. A big challenge was realizing the PWM system, because the Rabbit 2000 does not have PWM drivers and I was forced to write PWM routines in assembly using the serial port timers. I believe that after this pioneering work, the Rabbit 2000 , and its successor the Rabbit 3000 can be used to build a more sophisticated robot. One of the interesting advantages of the Rabbit series of microcontrollers is their ability to be programmed and accessed through the internet, which would be an interesting addition.

DOCUMENTATION

After being turned on with the use of a switch located on top of the robot, SCAB's compass must be calibrated. This done with the aid of a real magnetic compass, by aligning the electronic compass to face any of the 4 directions (North, South, West, East), pressing the pushbutton and waiting for the robot to automatically rotate 90 degrees to the right. Then the electronic compass must be aligned once again and the procedure repeated until the compass has been parallel to all 4 directions, at which point the robot will start its main behavior (scanning objects) after a warning beep and a short delay interval.

APPENDICES

Source Code:

```
/****************************************************
 PROGRAM NAME:  SCAB10.c
  DESCRIPTION: This program integrates the behaviors of the SCAB robot, it is the main program
   Programmer: Victor Mitselmakher
Last modified: 8/07/02
*****************************************************/
// LIBRARIES
#use "I2C.LIB"

// FUNCTION PROTOTYPES
init_board();
read_ir();
delay(int j);
do_scan();
fforward();
read_compass();
servos_on();
servos_off();
servos_adj();
rotate_botr();
rotate_botl();
rotate_rr(int k);
rotate_lr(int k);
rotate90();
do_push();
do_return();
fall_check();
wander();
turnto(char k);
mcalibrate();
calibrate();
timera_isr();
wenc_on();
wenc_off();
void wheel_isr();

// DEFINITIONS
#define ADC_WR 0x50   // ADC I2C Address + Write bit
#define ADC_RD 0x51   // ADC I2C Address + Read bit
#define ADC_CTR2 0xa0  // Control byte to init Ch2, 0 to 5V
#define ADC_CTR3 0xb0  // Control byte to init Ch3, 0 to 5V
#define ADC_CTR5 0xd0  // Control byte to init Ch5, 0 to 5V
#define ADC_CTR6 0xe0  // Control byte to init Ch6, 0 to 5V

#define CMP_WR 0xc0   // Compass I2C Address + Write bit
#define CMP_RD 0xc1   // Compass I2C Address + Read bit
#define CMP_REG0 0x00  // Compass Register 0  (software #)
#define CMP_REG1 0x01  // Compass Register 1  (8-bit bearing)
#define CMP_REG2 0x02  // Compass Register 2
#define CMP_REG3 0x03  // Compass Register 3  (Reg2 and 3 bearing 0-3599)
#define CMP_REG15 0x0f  // Compass Register 15


#define FORWARD 0  // Constant to control servo direction
#define REVERSE 1  // Constant to control servo direction
#define OBJNUM      10         // Max number of supported objects

// Behavior Definitions:
#define SCAN       0         // SCAN Behavior, 360 degree sweep
#define DONE       1         // IDLE .. DO NOTHING
#define WANDER     2         // WANDER Behavior, roll around and avoid falling off the table
#define PUSH       3         // PUSH Behavior, push object while on top of table, avoid edge
#define RETURN     4         // RETURN Behavior, return from pushing object


// GLOBAL VARIABLES
// (BEHAVIOR VARIABLES)
int BEHAVIOR;                // Current BEHAVIOR of ROBOT
int BEH_STAGE;               // Current BEHAVIOR STAGE of ROBOT

// (SERVO VARIABLES)
int Rcount;                            // keep track of times ISR is called for right servo
int Lcount;                            // keep track of times ISR is called for left servo
int RCspeed;                 // CURRENT SPEED of right servo
int LCspeed;                 // CURRENT SPEED of left servo
int Rspeed;                  // DESIRED SPEED of right servo
int Lspeed;                  // DESIRED SPEED of left servo

// (IR VARIABLES)
unsigned int IRTR;                               // Conditioned data (12 bits) from IRTR (top right)
unsigned int IRTL;                               // Conditioned data (12 bits) from IRTL (top left)
unsigned int IRBR;                               // Conditioned data (12 bits) from IRBR (bottom right)
unsigned int IRBL;                               // Conditioned data (12 bits) from IRBL (bottom left)

int IRTRD;                                       // Distance (cm) from IRTR (top right)
int IRTLD;                                       // Distance (cm) from IRTL (top left)
```

```
int IRBRD;                                        // Distance (cm) from IRBR (bottom right)
int IRBLD;                                        // Distance (cm) from IRBL (bottom left)


// (COMPASS VARIABLES)
char CMPD;                                                                  // 8-bit compass reading

// (WHEEL ENCODER VARIABLES)
int WCOUNT;                            // Amount of ticks on wheel
int RETURND;                  // # of ticks to return to center
unsigned int LASTWHEEL;                // variable to help reject false clicks

// (OBJECT SCAN VARIABLES)
char objects[OBJNUM][2];                          // Objects array.. Stores [x][0]=distance, [x][1]=orientation


//////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: MAIN()
           INPUT: none
          OUTPUT: none
     DESCRIPTION:
*/
main()
{
int i;
int avgv, avgd;

// VARIABLE DECLARATIONS AND INITIALIZATIONS:


// MAIN PROGRAM:
   printf("Self Centering Autonomous Bulldozer (SCAB)\n");
   printf("Software Version 1.0\n");

   init_board();  // Initialize the Robot (Sets BEHAVIOR=SCAN)


// *TEST IRS*
   read_ir();
          while(IRTL == 0 || IRTR == 0 || IRBL == 0 || IRBR == 0 )
          {
          read_ir();
                              BitWrPortI(PEDR, &PEDRShadow, 1, 0);             // Horn on
                                        delay(1000);
                              BitWrPortI(PEDR, &PEDRShadow, 0, 0);             // Horn off
                                        delay(100);
          }

          mcalibrate();       // Calibrate compass

          servos_on();                          // Switch servos to IDLE
          delay(5000);

// *TEST IRS*
   read_ir();
          while(IRTL == 0 || IRTR == 0 || IRBL == 0 || IRBR == 0 )
          {
          read_ir();
                              BitWrPortI(PEDR, &PEDRShadow, 1, 0);             // Horn on
                                        delay(1000);
                              BitWrPortI(PEDR, &PEDRShadow, 0, 0);             // Horn off
                                        delay(100);
          }


//        BEHAVIOR = WANDER;                              // UNCOMMENT THIS TO GET WANDERING BEHAVIOR

while (1)
{
          costate
          {
   //////////////////////////////
                    if (BEHAVIOR == SCAN)
                    {
                    do_scan();
                    } // END SCAN BEHAVIOR
   ///////////////////////////////
                    if (BEHAVIOR == WANDER)
                    {
                    wander();
                    } // END WANDER BEHAVIOR
          ////////////////////////////////
                    if (BEHAVIOR == PUSH)
                    {
                    do_push();
                    } // END PUSH BEHAVIOR
          ////////////////////////////////
                    if (BEHAVIOR == RETURN)
                    {
                    do_return();
                    } // END RETURN BEHAVIOR
          ////////////////////////////////
```

```
                    waitfor(DelayMs(1000));                           // Short rest period

          } // end costate
} // end while(1)
} // End main()
////////////////////////////////////////////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: INIT_BOARD()
           INPUT: none
          OUTPUT: none
     DESCRIPTION: * Initializes Port A as output
                                                  * Set up Timer A ISR Vector
                                                  * Set prescalers for Timer A
                                                  * Enable interrupts
*/
init_board()
{
// I2C
          i2c_init(); // Set up I2C Lines

// ***** PORT INITS *****
          WrPortI(SPCR, &SPCRShadow, 0x84);  // Init Port A as output
          WrPortI(PEDDR, &PEDDRShadow, 0x0f);  // Init Port E bit 0,1,2,3 as output, others as inputs

// ****** INTERRUPT VECTOR INITS *******
          SetVectIntern(0x0a, timera_isr);           // set up Timer A ISR
          SetVectExtern2000(1, wheel_isr);    // set up External ISR (Wheel encoder)

          // Enable timer A, also toggle timer A6 and A7 interrupts. (PWM Interrupts are still OFF)
          WrPortI (TACSR, &TACSRShadow, 0xc1);

// ***** REGISTER INITS ********
          WrPortI(TAT1R, &TAT1RShadow, 85);                           // Set Prescaler A1 = 85
          WrPortI(TAT4R, &TAT4RShadow, 3);                            // Set Prescaler A4 = 3 for 115200 bps

// ***** VARIABLE INITS ********
          BEHAVIOR = SCAN;              // Initial Behavior of Robot
          BEH_STAGE = 0;                          // Initial Stage of Behavior = 0

          WCOUNT = 0;                             // Amount of ticks on wheel
          LASTWHEEL = 1;                // variable to help reject false clicks

     BitWrPortI(PEDR, &PEDRShadow, 1, 1);                           // Turn on headlights
     BitWrPortI(PEDR, &PEDRShadow, 1, 3);
     BitWrPortI(PEDR, &PEDRShadow, 1, 2);                 // Orange lights on

} // End init_board()
////////////////////////////////////////////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: READ_IR()
           INPUT: none
          OUTPUT: none
     DESCRIPTION: * Reads the IR data from the ADC from all 4 IR sensors
                                                            * Converts raw data into distance (cm) and stores in global variables
*/
read_ir()
{
          int return_code;
          char IR_B0, IR_B1;                      // Raw data bytes from sensor

///////////// Init ADC for Ch2 conversions: ////////////////
          return_code=i2c_start_tx();   // Start condition
          return_code=i2c_write_char(ADC_WR);     // Address + Write bit

          return_code=i2c_write_char(ADC_CTR2);   // Control byte
   i2c_stop_tx();   // Stop condition

////////// Get IR Data
          return_code=i2c_start_tx();   // Start condition
          return_code=i2c_write_char(ADC_RD);     // Address + Read bit
   return_code=i2c_read_char(&IR_B0); // Read Data Byte 1
          return_code=i2c_send_ack();  // Send ACK
   return_code=i2c_read_char(&IR_B1); // Read Data Byte 2
          return_code=i2c_send_nak();  // Send NACK
   i2c_stop_tx();   // Stop condition

// CONDITIONING DATA BYTES:
#asm
          ld a, (IR_B1)                                               // Load A with LSB
          ld l, a                                                               // Put LSB into L

          ld a, (IR_B0)                                               // Load A with MSB
          ld h, a                                                               // Put MSB into H

          ld                  a, 00h
          cp                  a                                                                         // Reset carry flag

          rr HL                                                                           // Right shift (1)
          rr HL                                                                           // Right shift (2)
```

```
        rr HL                                                                    // Right shift (3)
        rr HL                                                                    // Right shift (4)

        ld       (IRTR), hl                                    // Store shifted data
#endasm

        if (IRTR != 0)
        IRTRD = (float) 73333 * (float) pow((float) 1/(float) IRTR,1.179);
////////(Done with Ch2)/////////


///////////// Init ADC for Ch3 conversions: /////////////////
        return_code=i2c_start_tx();  // Start condition
        return_code=i2c_write_char(ADC_WR);     // Address + Write bit

        return_code=i2c_write_char(ADC_CTR3);  // Control byte
   i2c_stop_tx();   // Stop condition

////////// Get IR Data
        return_code=i2c_start_tx();  // Start condition
        return_code=i2c_write_char(ADC_RD);     // Address + Read bit
   return_code=i2c_read_char(&IR_B0); // Read Data Byte 1
        return_code=i2c_send_ack();  // Send ACK
   return_code=i2c_read_char(&IR_B1); // Read Data Byte 2
        return_code=i2c_send_nak();  // Send NACK
   i2c_stop_tx();   // Stop condition

// CONDITIONING DATA BYTES:
#asm
        ld a, (IR_B1)                                          // Load A with LSB
        ld l, a                                                          // Put LSB into L

        ld a, (IR_B0)                                          // Load A with MSB
        ld h, a                                                          // Put MSB into H

        ld                 a, 00h
        cp                 a                                                                     // Reset carry flag

        rr HL                                                                    // Right shift (1)
        rr HL                                                                    // Right shift (2)
        rr HL                                                                    // Right shift (3)
        rr HL                                                                    // Right shift (4)

        ld       (IRTL), hl                                    // Store shifted data
#endasm

// PROCESSING DATA:
        if (IRTL != 0)
        IRTLD = (float) 73333 * (float) pow((float) 1/(float) IRTL,1.179);
////////(Done with Ch3)/////////


///////////// Init ADC for Ch5 conversions: /////////////////
        return_code=i2c_start_tx();  // Start condition
        return_code=i2c_write_char(ADC_WR);     // Address + Write bit

        return_code=i2c_write_char(ADC_CTR5);  // Control byte
   i2c_stop_tx();   // Stop condition

////////// Get IR Data
        return_code=i2c_start_tx();  // Start condition
        return_code=i2c_write_char(ADC_RD);     // Address + Read bit
   return_code=i2c_read_char(&IR_B0); // Read Data Byte 1
        return_code=i2c_send_ack();  // Send ACK
   return_code=i2c_read_char(&IR_B1); // Read Data Byte 2
        return_code=i2c_send_nak();  // Send NACK
   i2c_stop_tx();   // Stop condition

// CONDITIONING DATA BYTES:
#asm
        ld a, (IR_B1)                                          // Load A with LSB
        ld l, a                                                          // Put LSB into L

        ld a, (IR_B0)                                          // Load A with MSB
        ld h, a                                                          // Put MSB into H

        ld                 a, 00h
        cp                 a                                                                     // Reset carry flag

        rr HL                                                                    // Right shift (1)
        rr HL                                                                    // Right shift (2)
        rr HL                                                                    // Right shift (3)
        rr HL                                                                    // Right shift (4)

        ld       (IRBR), hl                                    // Store shifted data
#endasm

// PROCESSING DATA:
        if (IRBR != 0)
        IRBRD = (float) 178657 * (float) pow((float) 1/(float) IRBR,1.20934);
////////(Done with Ch5)/////////

///////////// Init ADC for Ch6 conversions: /////////////////
        return_code=i2c_start_tx();  // Start condition
        return_code=i2c_write_char(ADC_WR);     // Address + Write bit
```

```
        return_code=i2c_write_char(ADC_CTR6);   // Control byte
   i2c_stop_tx();   // Stop condition

////////// Get IR Data
        return_code=i2c_start_tx();   // Start condition
        return_code=i2c_write_char(ADC_RD);     // Address + Read bit
   return_code=i2c_read_char(&IR_B0); // Read Data Byte 1
        return_code=i2c_send_ack();  // Send ACK
   return_code=i2c_read_char(&IR_B1); // Read Data Byte 2
        return_code=i2c_send_nak();  // Send NACK
   i2c_stop_tx();   // Stop condition

// CONDITIONING DATA BYTES:
#asm
        ld a, (IR_B1)                                   // Load A with LSB
        ld l, a                                                 // Put LSB into L

        ld a, (IR_B0)                                   // Load A with MSB
        ld h, a                                                 // Put MSB into H

        ld              a, 00h
        cp              a                                                       // Reset carry flag

        rr HL                                                           // Right shift (1)
        rr HL                                                           // Right shift (2)
        rr HL                                                           // Right shift (3)
        rr HL                                                           // Right shift (4)

        ld      (IRBL), hl                              // Store shifted data
#endasm

// PROCESSING DATA:
        if (IRBL != 0)
        IRBLD = (float) 193500 * (float) pow((float) 1/(float) IRBL,1.22011);
////////(Done with Ch6)/////////


} // End read_ir()
////////////////////////////////////////////////////////////////////////////////////////////////////////



////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: DO_SCAN()
         INPUT: none
        OUTPUT: none
    DESCRIPTION: * Scans the surroundings and finds objects, records data
    */
do_scan()
{
int i, j,k;
char cstart, cobj;
int dobj,dobjl,dobjr, avgd, avgv;



        for(i=0;i<OBJNUM;i++)                           // Clear array
        {
        objects[i][0] = 0;
        objects[i][1] = 0;
        }

        read_compass();                                         // Get Compass data
        cstart = CMPD;                                                  // Starting orientation
   i=0;

        turnto(cstart+5);

while(1)
{
        read_compass();                                         // Get Compass data

        avgv = 0;
        avgd = 0;


        for(j=0;j<3;j++)                                        // Get average IR variance and distance to object in front of bot
        {
        read_ir();
        avgv += abs(IRBLD-IRBRD);
        avgd += (IRBLD+IRBRD)/2;
        }
        avgv = avgv/3;
        avgd = avgd/3;

        if((avgv < 10)                                                  // Found object?   must verify
                &&      avgd < 100)
                {
                cobj = CMPD;                                    // Save current compass heading
                dobj = avgd;                                    // Save distance to object
                turnto(cobj-15);                        // Turn 15 clicks left of obj
                read_ir();                                              // Get IR Data
                dobjl = IRBLD;                                  // Save distance left of object from left IR
                turnto(cobj+15);                        // Turn 15 clicks right of obj
```

```
                              read_ir();                                                          // Get IR Data
                              dobjr = IRBRD;                                            // Save distance right of object from right IR
                              turnto(cobj+2);                         // Return to face object

                              if((abs(dobjl - dobj) > 20)                       // Verify object data
                                                  &&        ((dobjr - dobj) > 20))
                              {                   // Verified
                              objects[0][0] = dobj;
                              objects[0][1] = cobj;

/*                            i++;
                              turnto(cobj+15);    */
                              //        sound buzzer
                              BitWrPortI(PEDR, &PEDRShadow, 1, 0);              // Horn on
                                        delay(1000);
                              BitWrPortI(PEDR, &PEDRShadow, 0, 0);              // Horn off
                                        delay(100);
                                        BEHAVIOR = PUSH;
                                        break;
                              } // End if -- verify
                              else
                              turnto(cobj+10);

                    } // End if -- Found object?
                    else
                       rotate_botr();

} // end while(1)

} // End do_scan()
//////////////////////////////////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: WENC_ON()
           INPUT: none
          OUTPUT: none
     DESCRIPTION: * Turns on the wheel encoders
     */
wenc_on()
{
          WCOUNT = 0;                             // Amount of ticks on wheel
          LASTWHEEL = 1;                // variable to help reject false clicks

// Turn on external interrupts
          WrPortI(I0CR, NULL, 0x31);              // enable external INT0 on PE4, both edges, priority 1
          WrPortI(I1CR, NULL, 0x31);    // enable external INT1 on PE5, both edges, priority 1
} // End wenc_on()
//////////////////////////////////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: WENC_OFF()
           INPUT: none
          OUTPUT: none
     DESCRIPTION: * Turns off the wheel encoders
     */
wenc_off()
{
          // Turn off External interrupts
          WrPortI(I0CR, NULL, 0x00);                                  // disable external interrupt 0
          WrPortI(I1CR, NULL, 0x00);                                  // disable external interrupt 1
} // End wenc_off()
//////////////////////////////////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Interrupt name: WHEEL_ISR
      DESCRIPTION: External interrupt routine
                                        Increment wheel counter as the wheel turns
*/
nodebug root interrupt void wheel_isr()
{
//                  delay(5);
          if (LASTWHEEL == !BitRdPortI(PEDR, 5))                      // Lastwheel != Current?
                    {
                              WCOUNT++;
                              LASTWHEEL = !LASTWHEEL;
                    }

} // End WHEEL_ISR
//////////////////////////////////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: FALL_CHECK()
           INPUT: none
          OUTPUT: none
     DESCRIPTION: * Checks top IR's to see if theres an edge nearby
     */
```

```
fall_check()
{
                    read_ir();

                    if(IRTLD > 35)
                    {
                    servos_on();                  // Stop and IDLE
                    rotate_rr(50);                // Randomly rotate right 1-50 clicks
                    fforward();                             // Fast forward
                    } // End TL
                    else
                    if (IRTRD > 35)
                    {
                    servos_on();                  // Stop and IDLE
                    rotate_lr(50);                // Randomly rotate left 1-50 clicks
                    fforward();                             // Fast forward
                    } // End TR

} // End fall_check()
//////////////////////////////////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: WANDER()
           INPUT: none
          OUTPUT: none
     DESCRIPTION: * Rolls forward and checks for falls, avoiding them
*/
wander()
{
int i,j;
// Motors ON

          servos_on();                                        // Turn servos ON at idle speed

 while(1)
 {
          Lspeed = 113;                           // Left servo SLOW FORWARD
          Rspeed = 105;                           // Right servo SLOW FORWARD


          for (i=1;i<10;i++)            // # of speed transitions
          {
                    servos_adj();                           // Smooth servo speed transition
                    fall_check();                           // Check for falls
                    delay(2);
          }         // End for(i) loop

 } // End while(1)


} // End WANDER()
//////////////////////////////////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: DO_PUSH()
           INPUT: none
          OUTPUT: none
     DESCRIPTION: * Moves towards located object, avoids edge, stop at edge and sets behavior=RETURN
*/
do_push()
{
int i;

          servos_on();                                        // Turn servos ON at idle speed

          turnto(objects[0][1]+2);  // Face object

          fforward();                                         // Full forward

          wenc_on();                                                    // Wheel encoders on (start counting)
          while(1)
          {
                    read_ir();

                    if(   (IRTLD > 35)
                              || (IRTRD > 35)
                              ||  ((IRBLD > 100) && (IRBRD > 100)))
                    {
                    wenc_off();                             // Stop counting
                    servos_on();                  // Stop and IDLE
                    RETURND=WCOUNT;               // Remember distance travelled
                    read_compass();

                    turnto(CMPD-128);             // turn around
                    BEHAVIOR = RETURN;
                    break;
                    } // End if
          }

} // End DO_PUSHE()
//////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: FFORWARD()
           INPUT: none
          OUTPUT: none
     DESCRIPTION: * Fast forward
*/
fforward()
{
int i;

          servos_on();                                    // Turn servos ON at idle speed

// Roll forward a distance
          Lspeed = 120;                         // Left servo FORWARD
          Rspeed = 100;                         // Right servo FORWARD


          for (i=1;i<17;i++)            // # of speed transitions
          {
                    servos_adj();                         // Smooth servo speed transition
                    delay(2);
          }         // End for(i) loop

} // End FFORWARD()
//////////////////////////////////////////////////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: DO_RETURN()
           INPUT: none
          OUTPUT: none
     DESCRIPTION: * Returns to center of objects
*/
do_return()
{

          fforward();                                               // Fast forward
          wenc_on();
          while(WCOUNT < RETURND+3)
                    fall_check();

 wenc_off();
 RETURND=0;
 BEHAVIOR = SCAN;

 read_compass();
 turnto(objects[0][1]);
} // End DO_RETURN()
//////////////////////////////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: READ_COMPASS()
           INPUT: none
          OUTPUT: none
     DESCRIPTION: * Reads the compass data and displays it
     */
read_compass()
{
          int return_code;


          return_code=i2c_start_tx();   // Start condition
          return_code=i2c_write_char(CMP_WR);     // Address + Write bit  (0)
          return_code=i2c_write_char(CMP_REG1);   // Request Register #1

          return_code=i2c_start_tx();   // Start condition
          return_code=i2c_write_char(CMP_RD);     // Address + Read bit  (1)
// read high
   return_code=i2c_read_char(&CMPD); // Read Data
   i2c_stop_tx();   // Stop condition

//   printf("Compass: %d \n", CMPD);

} // End read_compass()
//////////////////////////////////////////////////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: MCALIBRATE()
           INPUT: none
          OUTPUT: none
     DESCRIPTION: * Manually Calibrates the compass with pushbutton
     */
mcalibrate()
{
          int return_code, i, j, pb1;
```

```
 for(i=0;i<4;i++)            // Do this 4 times
 {

         pb1=0;              // Clear button status

         // Signal start of calibrate
         for(j=1;j<3;j++)
         {
   BitWrPortI(PEDR, &PEDRShadow, 1, 0);           // Set port E bit 0
   BitWrPortI(PEDR, &PEDRShadow, 0, 2);           // Orange lights off
         delay(100);
   BitWrPortI(PEDR, &PEDRShadow, 0, 0);           // Clear port E bit 0
   BitWrPortI(PEDR, &PEDRShadow, 1, 2);           // Orange lights on
         delay(100);
         }         // End for(j)

         ////////// Wait for proper PB
         while (1)
         {
         if (!BitRdPortI(PBDR, 2))              // Button pushed?
         {
                   pb1=1;                                                        // Yes
                            delay(500);
                   }

         if (!BitRdPortI(PBDR, 2) && pb1==1)              // Button still pushed?
                   break;                                                                                                // Button is now
debounced
         } // end while(1)

         // Sound that button was pushed
   BitWrPortI(PEDR, &PEDRShadow, 1, 0);           // Set port E bit 0
   BitWrPortI(PEDR, &PEDRShadow, 0, 2);           // Orange lights off
         delay(100);
   BitWrPortI(PEDR, &PEDRShadow, 0, 0);           // Clear port E bit 0
   BitWrPortI(PEDR, &PEDRShadow, 1, 2);           // Orange lights on


         ////// Calibrate in one direction
         delay(1000);                                                                      // Pause
         return_code=i2c_start_tx();   // Start condition
         return_code=i2c_write_char(CMP_WR);     // Address + Write bit  (0)
         return_code=i2c_write_char(CMP_REG15);  // Request Register #15

         return_code=i2c_start_tx();   // Start condition
         return_code=i2c_write_char(CMP_WR);     // Address + Write bit  (0)
         return_code=i2c_write_char(0xff);      // Write 255


         delay(100);                                                                       // Pause
         servos_on();
         delay(2000);
         rotate90();                                                                       // Auto-rotate ~90 degrees
         servos_off();
         delay(500);
 } // End For(i)


         // Sound end of calibrate
         for(j=0;j<5;j++)
         {
   BitWrPortI(PEDR, &PEDRShadow, 1, 0);           // Buzz on
   BitWrPortI(PEDR, &PEDRShadow, 0, 2);           // Orange lights off
         delay(200);
   BitWrPortI(PEDR, &PEDRShadow, 0, 0);           // Buzz off
   BitWrPortI(PEDR, &PEDRShadow, 1, 2);           // Orange lights on
         delay(100);
         }         // End for(j)

   BitWrPortI(PEDR, &PEDRShadow, 1, 2);           // Orange lights on

} // End mcalibrate()
//////////////////////////////////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: CALIBRATE()
           INPUT: none
          OUTPUT: none
     DESCRIPTION: * Auto-Calibrates the compass (rough)
     */
calibrate()
{
         int return_code, i, j;

         servos_on();                                // Turn servos ON at idle speed
         delay(50000);

         for(i=0;i<4;i++)
         {
         return_code=i2c_start_tx();   // Start condition
         return_code=i2c_write_char(CMP_WR);     // Address + Write bit  (0)
```

```
        return_code=i2c_write_char(CMP_REG15);  // Request Register #15

        return_code=i2c_start_tx();   // Start condition
        return_code=i2c_write_char(CMP_WR);     // Address + Write bit  (0)
        return_code=i2c_write_char(0xff);       // Write 255
        delay(5000);
        rotate90();                             // Rotate 90 degrees
        delay(10000);
        } // End for(i)


} // End calibrate()
//////////////////////////////////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: DELAY()
           INPUT: j
          OUTPUT: none
     DESCRIPTION: * Delays the bot... experimental values
     */
delay(int j)
{
        int i;

                for (i=0;i<j;i++)
                {
                #asm
                                ld       b, 0xff            // Load B=255
                d_loop1:
                                nop
                                nop
                                nop
                                nop
                                nop                                    // 10 clocks worth of NOPs
                                djnz     d_loop1   // Decrement B and jump if not zero (5 clocks)
                #endasm
                }  // End for(i) loop

} // End delay()
//////////////////////////////////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: ROTATE_BOTR()
           INPUT: none
          OUTPUT: none
     DESCRIPTION: * Rotates the robot clockwise a few degrees and stops
*/
rotate_botr()
{
int i,j;
// Motors ON


        servos_on();                                    // Turn servos ON at idle speed


        Lspeed = 115;                           // Left servo SLOW FORWARD
        Rspeed = 117;                           // Right servo SLOW REVERSE


        for (i=1;i<16;i++)            // # of speed transitions
        {
                servos_adj();                           // Smooth servo speed transition
                delay(2);
        }       // End for(i) loop

        delay(50);

        servos_on();                            // Switch servos to IDLE

} // End ROTATE_BOTR()
//////////////////////////////////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: ROTATE_BOTL()
           INPUT: none
          OUTPUT: none
     DESCRIPTION: * Rotates the robot counterclockwise a few degrees and stops
*/
rotate_botl()
{
int i,j;
// Motors ON


        servos_on();                                    // Turn servos ON at idle speed

        Lspeed = 101;                           // Left servo SLOW REVERSE
```

```
        Rspeed = 104;                           // Right servo SLOW FORWARD


        for (i=1;i<7;i++)             // # of speed transitions
        {
                servos_adj();                           // Smooth servo speed transition
                delay(2);
        }       // End for(i) loop

        delay(50);
        servos_on();                            // Switch servos to IDLE

} // End ROTATE_BOTL()
//////////////////////////////////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: ROTATE_LR()
           INPUT: int k
          OUTPUT: none
     DESCRIPTION: * Rotates the robot counterclockwise a random amount
*/
rotate_lr(int k)
{
int i,j;

i = (float) k * rand();                 // I = random (1..k)
i++;

        for(j=0;j<i;j++)
        rotate_botl();


} // End ROTATE_LR()
//////////////////////////////////////////////////////////////////////////////////////////////////////////



//////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: ROTATE_RR()
           INPUT: int k
          OUTPUT: none
     DESCRIPTION: * Rotates the robot clockwise a random amount
*/
rotate_rr(int k)
{
int i,j;

i = (float) k * rand();                 // I = random (1..k)
i++;

        for(j=0;j<i;j++)
        rotate_botr();


} // End ROTATE_RR()
//////////////////////////////////////////////////////////////////////////////////////////////////////////




//////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: ROTATE90()
           INPUT: none
          OUTPUT: none
     DESCRIPTION: * Rotates the robot clockwise 90 degrees and stops (experimental)
*/
rotate90()
{
int i,j;
// Motors ON


        servos_on();                                    // Turn servos ON at idle speed


        Lspeed = 115;                           // Left servo SLOW FORWARD
        Rspeed = 117;                           // Right servo SLOW REVERSE


        for (i=1;i<16;i++)            // # of speed transitions
        {
                servos_adj();                           // Smooth servo speed transition
                delay(2);
        }       // End for(i) loop

        delay(1900);                  // TURNING Delay

        servos_on();                            // Switch servos to IDLE

} // End ROTATE90()
//////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: TURNTO()
           INPUT: char k
          OUTPUT: none
     DESCRIPTION: * Rotates the robot facing k compass value, k's range is (0..255)
*/
turnto(char k)
{
int i, dir;
char aa,bb;

          read_compass();

          aa = CMPD-k;
          bb = k-CMPD;
          if (aa > bb)                                        // Determine which way is the shortest to rotate
          dir=1;                        // Rotate right
          else
          dir=0;                        // Rotate left

          i=0;

          while(i<255)
          {
          i++;
          read_compass();

                    if(CMPD == k ||
                              (CMPD+1) == k ||
                              (CMPD+2) == k ||
                              (CMPD+3) == k ||
                              (CMPD-1) == k ||
                              (CMPD-2) == k ||
                              (CMPD-3) == k)
                    break;

          if (dir==1)
          rotate_botr();
          else
          rotate_botl();

          } // End while(i<255)

} // End TURNTO()
////////////////////////////////////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: SERVOS_ADJ()
           INPUT: none
          OUTPUT: none
     DESCRIPTION: * Adjusts servo speed to smooth it
*/
servos_adj()
{
          if (RCspeed < Rspeed)
          RCspeed++;
          else
          if (RCspeed > Rspeed)
          RCspeed--;

          if (LCspeed < Lspeed)
          LCspeed++;
          else
          if (LCspeed > Lspeed)
          LCspeed--;

          WrPortI(TAT6R, &TAT6RShadow, RCspeed);                      // Set Prescaler A6 = RCspeed
          WrPortI(TAT7R, &TAT7RShadow, LCspeed);                      // Set Prescaler A7 = LCspeed
} // End SERVOS_ADJ()
////////////////////////////////////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: SERVOS_ON()
           INPUT: none
          OUTPUT: none
     DESCRIPTION: * Turns on PWM Interrupts that control servos, sets motor speed to idle
*/
servos_on()
{
Rspeed=110;                             // Set idle desired speed
Lspeed=104;

RCspeed=110;                            // Set idle current speed
LCspeed=104;

          WrPortI(TAT6R, &TAT6RShadow, RCspeed);                      // Set Prescaler A6 = Rspeed
          WrPortI(TAT7R, &TAT7RShadow, LCspeed);                      // Set Prescaler A7 = Lspeed

          // Timers A6, A7 clocked by A1, turn ON PWM interrupt priority 1.
          WrPortI (TACR, &TACRShadow, 0xc1);
```

```
} // End SERVOS_ON()
////////////////////////////////////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Function name: SERVOS_OFF()
           INPUT: none
          OUTPUT: none
     DESCRIPTION: * Turns off PWM Interrupts that control servos
*/
servos_off()
{
          // Timers A6, A7 clocked by A1, turn OFF PWM interrupts
          WrPortI (TACR, &TACRShadow, 0xc0);

} // End SERVOS_OFF()
////////////////////////////////////////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////////////////////////////////////////
/* Interrupt name: TIMERA_ISR
      DESCRIPTION: Timer A Interrupt Service Routine, controls the PWM of servos
                   through A6 (right servo), A7 (left servo)
*/
#asm
timera_isr::
          push      af                                                                      ; save registers
          push      hl

          ioi       ld hl, (TACSR)                ; load TimerA interrupt flags into RegL (clears flag)

// Branch figure out who interrupted
          ld                  a, 40h
          and       l                                                                       ; mask off all but bit 6 of TACSR
          jr                  z, int_a7                           ; if zero then its NOT right servo's interrupt, branch to left servo

//(int6) A6 (Right servo) interrupted
int_a6:
// Increment Rcount
          ld                  hl, (Rcount)
          inc       hl                                                                      ; increment Rcount through RegHL
          ld                  (Rcount), hl

// Mask counter, Branch to SET6 or CLEAR6
          ld                  a, 01h
          and       l                                                                       ; mask off all but lowest bit of Rcount
          jr                  z, clear_a6

// (set6) Set bit PA3
set_a6:
   ioi    ld                  a, (PADR)                           ; Load a with PortA bits
          set       3, a                                                                              ; set bit 3 of Port A (PA3) to 1
          ioi       ld                  (PADR), a                           ; Store Accumulator to PortA
          jr                  done

// (clear6) Clear bit PA3
clear_a6:
          ioi       ld                  a, (PADR)                           ; Load a with PortA bits
          and       0xf7                                                                    ; set bit 3 of Port A (PA3) to 0
          ioi       ld                  (PADR), a                           ; Store Accumulator to PortA
          jr                  done


//////////////////////////////////////
//(int7) A7 (Left servo) Interrupted
int_a7:
// Increment Lcount
          ld                  hl, (Lcount)
          inc       hl                                                                      ; increment Lcount through RegHL
          ld                  (Lcount), hl

// Mask counter, Branch to SET7 or CLEAR7
          ld                  a, 01h
          and       l                                                                       ; mask off all but lowest bit of Lcount
          jr                  z, clear_a7

// (set7) Set bit PA0
set_a7:
          ioi       ld                  a, (PADR)                           ; Load l with PortA bits
          set       0, a                                                                              ; set bit 0 of Port A (PA0) to 1
          ioi       ld                  (PADR), a                           ; Store Accumulator to PortA
          jr                  done

// (clear7) Clear bit PA0
clear_a7:
          ioi       ld                  a, (PADR)                           ; Load l with PortA bits
          and       0xfe                                                                    ; set bit 0 of Port A (PA0) to 0
          ioi       ld                  (PADR), a                           ; Store Accumulator to PortA

done:
          pop       hl                                                                      ; restore registers
          pop       af
```

```
        ipres                                                              ; restore interrupts
        ret                                                                ; return
#endasm

// End TIMERA_ISR
//////////////////////////////////////////////////////////////////////////////////////////////////////


///////////////////////// END OF PROGRAM ///////////////////////////////////
```