EEL 5666
Intelligent Machine Design Lab
Final Report

Project “Transformer”
An Autonomous Robot

University of Florida
Department of Computer and Electrical Engineering

Su Va (Andy) Fong
8/03/98

Content

Abstract:

Before taking this class, I always wanted to build a robot. I wanted to build something with legs. When I saw the Robobug, I was so fascinated and wished to build one too. Finally, I got the chance to take this class and to build a robot. I want to build something from scratch instead of buying some already made model and assembling it. However, I found out that building a Robobug from scratch is not something I can do in one semester, especially for someone like me who knows nothing about mechanics. However, I still want to build something with legs. I read through some mechanics books and finally found the 'kneeless' leg mechanism. It is relativlely easy to build and to control. After more thinking and struggling, I feel that a walking robot sometimes is not very practical and it consumes a lot of power. Also, I cannot do much with just a walking robot. So, I decided to combine a tank robot and a walking robot into one. The tradeoff is that I have to make the legs shorter in order to make the transformation possible. I also need two sets of sensors for the walking mode and the tank mode because they are walking or moving in different direction.

Executive Summary:

This is the newest member in my family. Sometimes, it is a tank robot. Sometimes, it is a six-leg crawling creature. It’s characteristic is it unpredictable (random) behavior. I cannot predict when it will transform from one mode to another. It will walk around and avoid obstacles. It also likes human. It tends to look for human to play with. When it sees a human right in front of it. It will jiggle its tail. This is my newest pet.

Introduction

Transformer is an autonomous agent which can transform between a tank robot and a walking robot. It acts like a pet and tends to look for a human to play with. Whenever it sees a human in front of it, it will jiggle its tail. I intended to make this robot act by itself, more like a real pet that I cannot predict what it will do.

In this report, I will talk about the integrated system, mobile platform, actuation, sensors, behavior and some programming technique I used on this robot.

Integrated System

The robot is controlled by 68HC11EVBU with ME11 which has 32k memory expansion, two motor driver using output compare 2 and 3, and a 40 kHz modulated output port for IR LEDs. The ME11 also makes the 68HC11 ready for I/O port expansion. I added three input ports and output ports for reading the sensors and controlled the legs.

In order to drive the six legs, I also built a small motor driver board which contains three motor driver chips and two 74’04 chips. The motor driver is a replicate of the DC motor driver in the ME11. I made three of them and put them together in a small board. For circuit diagram, please refer to the ME11 Assembly Manual fig 8 (http://www.mil.ufl.edu/novasoft).

Mobile Platform

This robot will be able to transform between a walking robot and a tank robot. The body is made of aluminum angle and pivoted together. I also use aluminum channel for the legs. I choose to use aluminum because it is strong and relatively light.

Walking Mechanism

The leg mechanism I use is very straight forward and is relatively easy to build and control. Fig 1 shows the mechanism of a single leg.

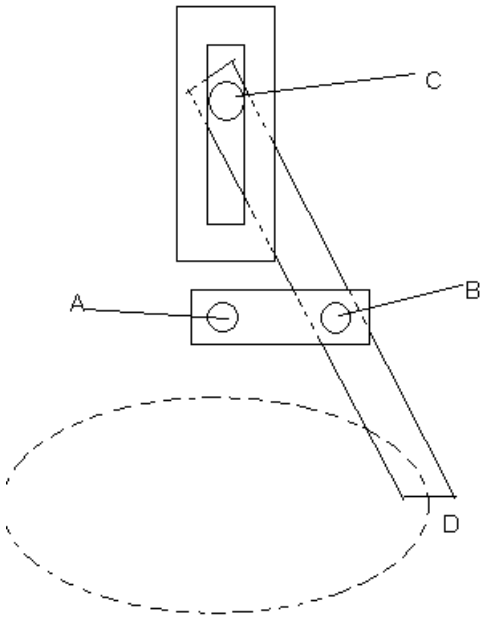

fig 1 The 'kneeless' leg mechanism

Point A and C are pivots. Point A is attached to a motor so that point B circles around point A. As point B turns, point C can move up and down. Therefore, point D can trace out an oval orbit which enable the robot to move back and forth. Fig 2 shows one side of the robot with the legs attached.

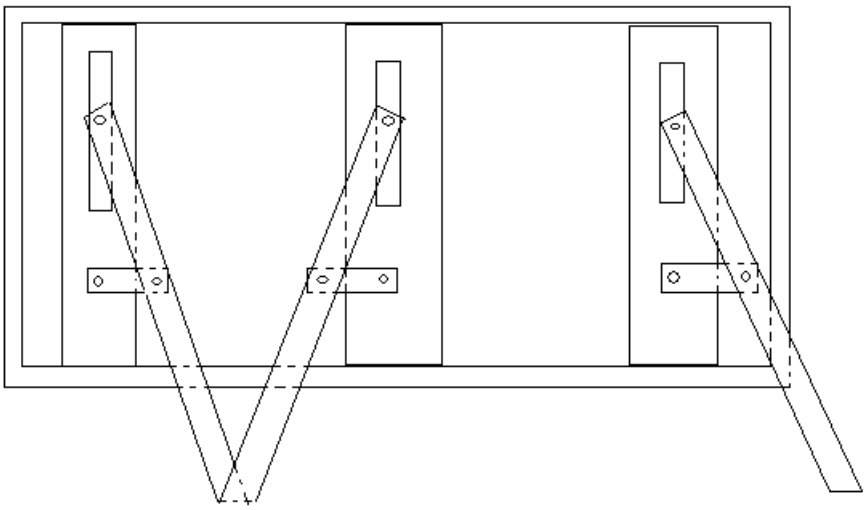

fig 2 Side view of the walking robot

The middle leg must be 180 degree out of phase of the other two legs. The legs on one side must be 180 degree out of phase with the legs on the other side.

Walking pattern

The walking patterns are shown in fig 3. This walking pattern is very essential and I try to maintain this walking pattern after each move when walking forward or turning left or right. To turn left, just reserve the motor direction on the left right side. To turn right, just reserve the motor direction on the right side. Some programming is need to keep the legs synchronized which will be cover in the programming technique section.

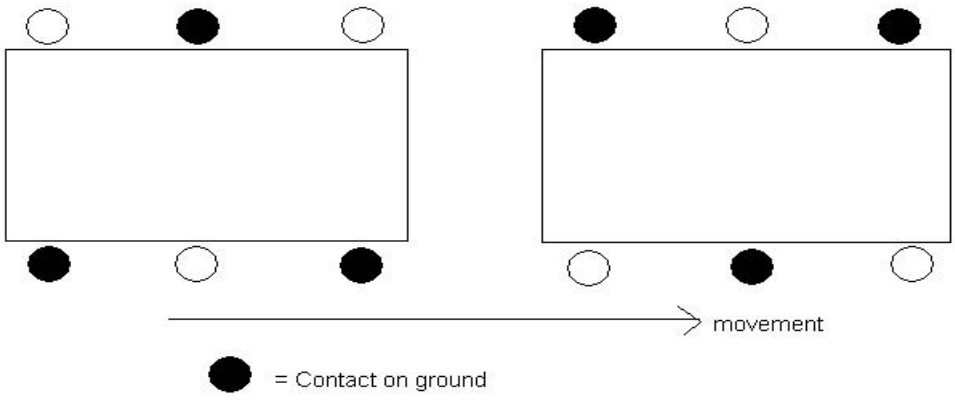

fig 3 Walking pattern (top view)

The tracks

The tracks are cut out from a toy and mounted on the side of the robot. When it transforms to a tank robot, it retracts all the legs to its highest position and let the tracks touch the ground as shown in fig 4. Fig 5 shows the top view of the robot with the position of the tracks and the legs.

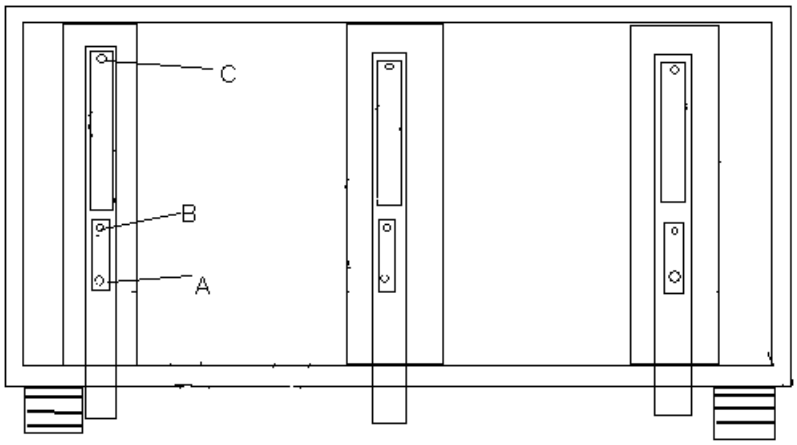

fig 4 Tank robot and the leg position

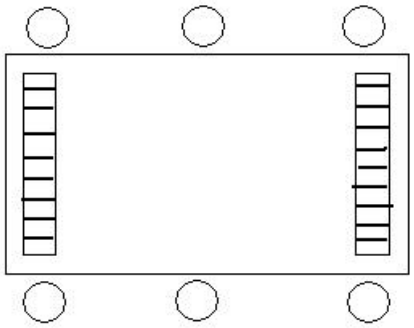

fig 5 Top view of teh tank robot

Actuation

This robot contains eight motors. Two for the tracks and Six for the legs. Because I need relatively high torque to move the legs and the motor must be able to turn 360 degree to move the leg forward, the motors are fully hacked from some used servo. All the electronics parts are taken out from the servo. The main gear is also modified so that 360 degree turn is possible. To get full power from the motors, the power is supplied directed from the 8-pack battery pack.

The motors for the legs are driven by the motor drivers board I build using three SN754410 and two 74’04 chips. Twelve output pins are connected to the motor drivers board to control the direction of the motors and enable the motors. Zero in the direction pins will move the legs backward and one will move the legs forward. Zero in the enable pins will stop the legs and one will move the legs.

The motors for the tracks are controlled by the ME11. The signals are pulse width modulated using output compare two and three.

Sensors

Five types of sensors are used in this robot: Motion sensor, IR sensors, Bump sensors, Touch sensors and Leg Position sensors.

Motion Sensor

The motion sensor is used to detect the present of a human. It is a hack from a backyard motion activated light. The signal is tapped out from pin 7 of the op-amp on the circuit board. The signal from the op-amp is very small. The voltage sway is from -0.5 volt to +0.5 volt with a DC offset of 12v. The signal is in very low frequency. It actually looks like a DC signal in the oscilloscope. The signal will move up and down if a heat source in front of it is moving to left or right respectively. Since the signal can be negative, it is not suitable for the A/D converter of the 68HC11. I need to filter off the 12v dc offset and give it a 2.5v offset and amplify it so that the range of the signal will be limited to about 0 to 5 volt. Then, I found out that after the amplifier, the signal does not move up and down as smooth as the original signal. It suddenly jumps up for a very short time and come back down. That might not be very useful if I connect it directly to the A/D converter of the 68HC11. So, I add a comparator and a D-Flip Flop to latch the result so that I can reset the D-FF and read the result anytime I want. The final circuit is shown it fig. 6.

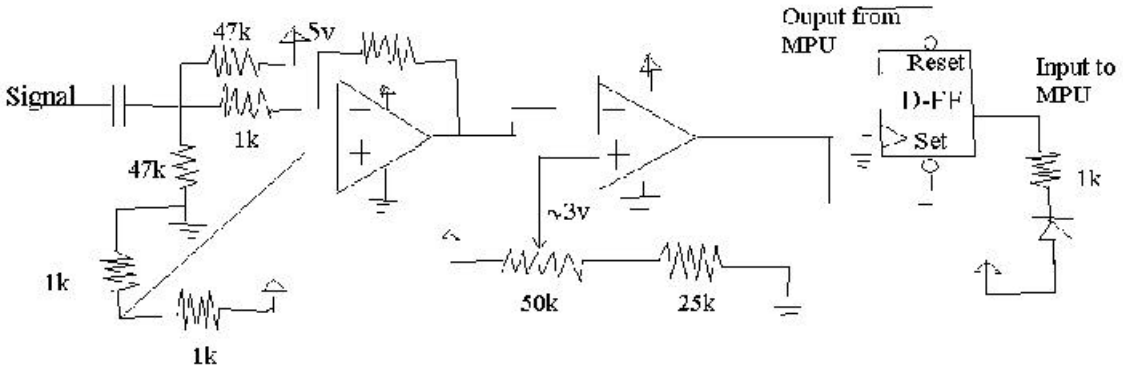

fig 6 Motion Detector Amplifier

Because of the noise from the motors, whenever the motors move, the D-FF will be set and give a false result. I have to power up the motion sensor unit with a separate battery. When using battery to power up the motion sensor unit, it takes about five minutes to charge up the capacitors. So, I used a ten battery pack to reduce the charging time. The sensitivity will also benefit from the ten battery pack instead of eight battery pack.

IR sensors

The IR sensors are hacked from the Sharp digital IR sensors. After the hacking, the sensors can give out analog signal from 1.5v to 2.5v depending on the distance of the object away from the sensors. So, that is used for collision avoidance. I used 330 ohm resistors for the IR LED and it can see while objects from 7” to 36”. The position of the IR sensors is shown in fig. 7.

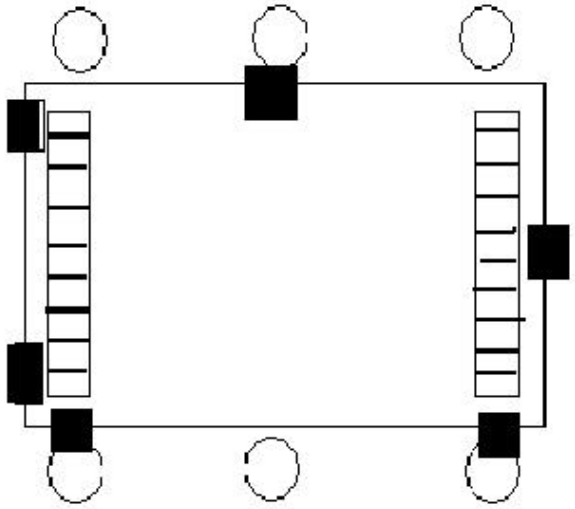

fig 7 IR Sensors

Bump Sensors

Since the IR sensors cannot see black object, I also added four bump sensors at the four corners for collision avoidance. The bump sensors have higher priority over the IR sensors. Fig. 8 shows the position and the structure of the bump sensors. The design is borrow from "Critter" study in class.

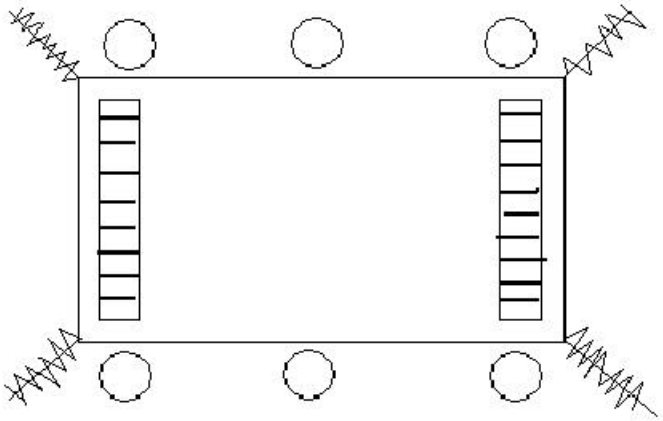

fig. 8 Bump Sensors

The input port for the bump sensors are pull up to Vcc when the sensors are not bumped. The metal wire in the middle of the spring is grounded. Therefore, whenever it hit something, the input of the bump sensor will be grounded.

Touch/Position Sensors

The Touch sensors are momentary push buttons and the Position sensors are mini-level switches. When the switches are open, they are pulled up to Vcc; otherwise, they are pull to ground. Fig. 9 shows the position of the touch sensors and the position sensors.

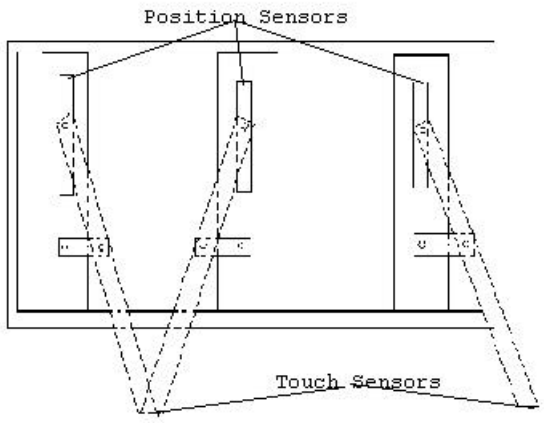

fig. 9 Touch Sensors and Position Sensors

The touch sensors are used to synchronize the legs' movement. It enables the program to know when to stop the legs and when to move the legs. More about controlling the legs will be discussed in the programming technique section.

The position sensors are use for transformation from walking mode to tank mode. It enables the legs to retract to their highest position and stop.

Behaviors

One of the main behavior of this robot is its unpredictable behavior. I intended to make it more like a real animal, so I put a lot of random decision in the program. However, I do make some choices happen more likely over the others by comparing several bits of the TCOUNT register. Therefore, I cannot predict when it will transform and when it will look for a human to play with.

The following is a summary of the robot's behavior :

- Dynamic Collision Avoidance
- Transform between Tank mode and Walking mode
- Tend to look for and walk to human
- When something is right in front of it, it will either turn around or transform
- When the power is low or it is tired (turn too many times) or if the legs slip or stuck, it will transform back to tank mode
- When it sees a human right in front of him, it will jiggle its tail

Programming Technique

In this section, I will talk about some of the programming technique that I used for the brain of transformer. I am using assembly for my program and sometimes, it takes a whole day to figure out a very small bug like missing a # sign, jump to the wrong label…etc. But I enjoy doing assembly because I have the control and I know what's going on in terms of hardware.

Dynamic collision avoidance

I use the IR reading to directly calculate the speed of the track. The range of the IR reading is about 87 to 130. To control the speed of the right track, I take the reading from the front left IR sensor and subtract it by 120. If the result is positive, the object is very closed to the front left corner. So, I reverse the direction of the right track. The speed of the track is the result multiplied by 25. (The full speed of the track is 255) If the result is negative, I will move the right track forward and the speed of the track will be the result multiplied by 7. So, when the IR sees nothing, the track speed will be 231. The control of the left track is the same.

This technique is very easy to implement. It does not require a lot of if … else decision. The turn is very smooth and nature. However, there is one drop back of this technique. When an object is big and right in front of both sensors (Same reading from both the left and right IR sensors), it will slow down and finally stop. In this state, the robot is trapped and is going nowhere. Something, it will finally turn away but it takes a long time. So, I programmed to robot to either transform or turn away when the speed of both tracks is lower than certain level. That way, the robot will never be trapped again. It will "hesitate" in front of a big object and than do something to get away.

Walking Algorithm

The walking algorithm depends heavily on the walking pattern. The walking algorithm is very simple. By changing the pattern and the direction of the motors, I can use the same walking algorithm to move forward or backward and turn left or right. When it first transform to a walking robot, the pattern is 101010. Bit 0 of the pattern it the front left leg of the robot. Bit 1 is the middle left leg. Bit 3 is the front right leg … and so forth. A zero in the pattern means the leg is pointing backward (opposite direction of the motor movement). A one in the pattern means the leg is point forward (same direction of the motor movement). When the robot first transforms to walking mode the legs' direction is shown in fig. 10a. For the walking algorithm, I first load in the walking pattern and store it to LEG_EN which will enable leg 1, 3 and 5. Then I read the touch sensors of the three legs, negate it and put it into LEG_EN. So leg 1, 3 and 5 will stop whenever it leaves the ground. The legs' direction is shown in fig. 10b. Then I negate the walking pattern and store it into LEG_EN. Now, the pattern becomes 010101. So, leg 0, 2 and 4 will move. Then I read in the touch sensor and put it into LEG_EN. So, leg 0, 2 and 4 will stop when they touch the ground. Now the direction of the legs is as shown in fig 10c. That finished one moving step and I store the walking pattern back to W_PATTERN and return. If the walking algorithm is call again, it will move another step forward and the walking pattern will go back to the original 101010 as shown in fig. 10d – fig. 10e. If I want to walk backward, I just need to reverse the direction of the motors and call the walk algorithm again.

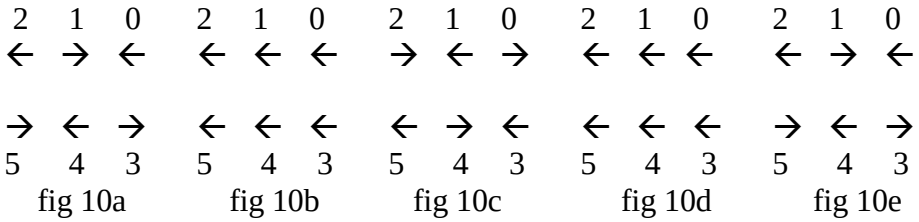

fig 10a fig 10b fig 10c fig 10d fig 10e

If I want to turn left, I just need to reverse the direction of the motors on the left hand side. Then, negate bit 0-2 of the walking pattern and call the walk algorithm. After the turn is done, I negate bit 0-2 of the walking pattern and store it back to W_PATTERN and I am really to make another left turn, right turn or move forward … etc. The sequence is shown in fig 11a –fig. 11d.(assuming the starting pattern is 101010)

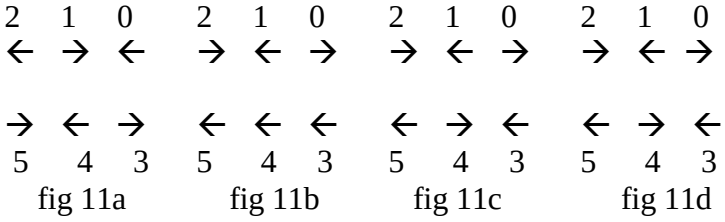

fig 11a fig 11b fig 11c fig 11d

(101010) (101101) (010010) (010101)

To make a right turn, just reverse bit 3-5 of the walking pattern and reverse the direction of the motors on the right hand side. Then, call the walk algorithm. After its done, reverse bit 3-5 of the walking pattern and store it back to W_PATTERN.

Dynamic Sensors Position

In my program, I have a function call "M_SET". It will determine which sensors is the Front Left sensor, which one is the Front Right sensor, which one is the Rear sensors, etc… depending on what mode the robot is in. So, whenever it transforms, it just

call that function and determine the position of the sensors. That way, I can use the same collision avoidance algorithm without changing anything.

Watchdog Program

I also use the RTI interrupt to write a small watchdog program to monitor the walking subroutine. If it stuck in the loop for more than five seconds, it will transform back to tank mode. Since the walking algorithm depends heavily on the touch sensors and the walking surface will affect the sensors' reading dramatically, this is a safety feature for preventing any problems happen when in the walking mode.

Conclusion

The project was successful. I could make it transform as its own will. Compare to other robots, my robot is not as useful because mine is just walking around. However, that is what I intended to do. I wanted to make it act like a real pet. My future development will be adding more complex behavior, programming it to look for food and learn how to walk. In this project, I learned a lot of practical knowledge which I can never learn in other classes. Things look very easy on paper is not that easy when I actually built it. Nothing is easy when it comes to the real world. Using the right tools and finding the right parts are also critical. With the right tools, it will take just minutes to finish the job. With the wrong tools, it takes days to do the same job. Cutting aluminum with a Dremel is not a very good idea, but that's the only tool I have. I also found that Epoxy Strip was extremely useful when I mount my bump sensor onto the body. It can mold like clay but after it's dry, it is very strong and can stick almost anything together. It can be found in Radio Shack. I enjoy this class and doing this project. Especially working with other people in the lab overnight. Sometimes, I think I put too much pressure on myself and that could be very stressful.

Appendix – source code

```
* Title         : Machine Intelligence Design Lab (Summer 98)
* Filename      : T-Former.ASM
* Programmer    : (Andy) Su Va Fong
* Date          : 07/31/97
* Version       : 1.0
* Description   : This program is the brain of my robot - Transformer

*******************************************************************************
* Define the address locations of the various registers and user-defined
* constants used in the program
*******************************************************************************
*
BAUD    EQU     $102B  ; BAUD rate control register to set the BAUD rate
SCCR1   EQU     $102C  ; Serial Communication Control Register-1
SCCR2   EQU     $102D  ; Serial Communication Control Register-2
SCSR    EQU     $102E  ; Serial Communication Status Register
SCDR    EQU     $102F  ; Serial Communication Data Register

EOS     EQU     $04    ; User-defined End Of String (EOS) character
CR      EQU     $0D    ; Carriage Return Character
LF      EQU     $0A    ; Line Feed Character
ESC     EQU     $1B    ; Escape Charracter

BASE    EQU     $1000  ; Beginning of Registers
PORTD   EQU     $08    ; Port D
DDRD    EQU     $09    ; Data Direction Register of Port D
ADCTL   EQU     $30
OPTION  EQU     $39
ADR1    EQU     $31    ; A/D Register 1
ADR2    EQU     $32    ; A/D Register 2
ADR3    EQU     $33    ; A/D Register 3
ADR4    EQU     $34    ; A/D Register 4
TOC2    EQU     $18    ; Output Compare 2 register
TOC3    EQU     $1A    ; Output Compare 3 register
TOC4    EQU     $1C    ; Output Compare 4 register
PACTL   EQU     $26    ;
TCTL1   EQU     $20    ; Timer Control register
TMSK1   EQU     $22    ; Timer Mask1 Register
TFLG1   EQU     $23    ; Timer Flag1 Register
TFLG2   EQU     $25    ; Timer Flag2 Register
TMSK2   EQU     $24    ; Timer Mask2 Register
TCNT    EQU     $0F

BUMP1   EQU     %00000001     ; Bump Sensor 1 postion
BUMP2   EQU     %00000010     ; Bump Sensor 2 postion
BUMP3   EQU     %00000100     ; Bump Sensor 3 postion
BUMP4   EQU     %00001000     ; Bump Sensor 4 postion

* Masks
BIT0    EQU     %00000001
BIT1    EQU     %00000010
BIT2    EQU     %00000100
BIT3    EQU     %00001000
BIT4    EQU     %00010000
BIT5    EQU     %00100000
BIT6    EQU     %01000000
BIT7    EQU     %10000000

out1byt equ     $e4fF   ; out1byt subroutine address
outcrlf equ     $e508   ; outcrlf subroutine address
Period  EQU     $FFFF   ; Period of the PWM for both tracks
LEG_EN  EQU     $4000   ; Leg enable address
LEG_DIR EQU     $5000   ; Leg direction address
CDS_SEL EQU     $6000   ; CDS cell selection address
MOTION  EQU     $6000   ; Motion detection address
LED     EQU     $6000   ; LED output address
IR_ADR  EQU     $7000   ; IR LED output address
```

```
T_ADR   EQU     $5000   ; Touch sensor input adress
B_ADR   EQU     $6000   ; Bump sensor input address
POS_ADR EQU     $4000   ; Leg position sensor input address
IR_TH   EQU     120     ; IR threshold value
IR_MAX  EQU     130     ; IR range (maximum)
IR_MIN  EQU     80      ; IR range (minimum)
BACKWARD EQU    %00000000       ; Direction of legs for walking backward
FORWARD EQU     %00111111       ; Direction of legs for walking forward
RIGHT   EQU     %00000111       ; Direction of legs for turning right
LEFT    EQU     %00111000       ; Direction of legs for turning left


*
*******************************************************************************
* Initialize Interrupt Jump Vectors
*******************************************************************************
* Jump Vector for TOC2_ISR
        ORG     $00DC
        JMP     TOC2ISR
* Jump Vector for TOC3_ISR
        ORG     $00D9
        JMP     TOC3ISR
* Jump Vector for RTI_ISR
        ORG     $00EB
        JMP     RTI_ISR
*******************************************************************************
*               Define Strings for displaying messages
*******************************************************************************
        ORG     $8000
        JMP     Main
ClrScr  FCB     ESC,$5B,$32,$4A         ; ANSI sequence to clear screen
        FCB     ESC,$5B,$3B,$48         ;   and move cursor to home
        FCB     EOS                     ; EOS character

*******************************************************************************
*                       Data Section
*******************************************************************************
*Tank Mode Data

Track0  FCB     0       ; Left Track Speed
Track1  FCB     0       ; Right Track Speed
Dir0    FCB     0       ; Left Track Direction
Dir1    FCB     0       ; Right Track Direction
High0   FDB     $0000   ; High time of the PWM for the left track
High1   FDB     $0000   ; High time of the PWM for the right track

*Sensors Data
Touch   FCB     0       ; Reading from touch sensors
Positn  FCB     0       ; Reading from leg position sensors
FL_BUMP FCB     0       ; Reading from Front left bump sensor
FR_BUMP FCB     0       ; Reading from Front Right bump sensor
RL_BUMP FCB     0       ; Reading from Rear Left bump sensor
RR_BUMP FCB     0       ; Reading from Rear Right bump sensor
Bump    FCB     0
Motion  FCB     0       ; Reading from the motion detection
IR0     FCB     0       ; Reading from IR sensor 0
IR1     FCB     0       ; Reading from IR sensor 1
IR2     FCB     0       ; Reading from IR sensor 2
IR3     FCB     0       ; Reading from IR sensor 3
IR4     FCB     0       ; Reading from IR sensor 4
IR5     FCB     0       ; Reading from IR sensor 5
CDS0    FCB     0       ; Reading from IR CDS 0
CDS1    FCB     0       ; Reading from IR CDS 1
CDS2    FCB     0       ; Reading from IR CDS 2
CDS3    FCB     0       ; Reading from IR CDS 3
CDS4    FCB     0       ; Reading from IR CDS 4
BATTERY FCB     0       ; Reading from Battery potential meter

MODE    FCB     0       ; mode of the robot (0 - tank, 1 - walk)
FL_IR   RMB     2       ; front left IR sensor address
FR_IR   RMB     2       ; front right IR sensor address
```

```
R_IR    RMB     2       ; rear IR sensor address

CA_L    FCB     0       ; Collision avoidance (left motor control)
CA_R    FCB     0       ; Collision avoidance (right motor control)

WALK_P  FCB     %00101010       ; Walking pattern

COUNT   FDB     0       ; Count from RTI (Testing only)
W_COUNT FDB     0       ; Count for walking timeout

Timeout FCB     0       ; Timeout - set if W_count > 1250

OUTPUT3 FCB     0       ; Buffer for output port 3

HUMAN   FCB     0       ; Set when human is detected

T_C     FCB     0       ; Count how many turns are made when walking
*******************************************************************
*                       MAIN PROGRAM
*******************************************************************
*  Initialization
Main    LDS     #$0041  ; Initial the stack pointer
        LDX     #BASE   ; Base address for system register
        LDAA    #0      ; Disable legs when first start
        STAA    LEG_EN
        JSR     InitOC  ; Initialize Output Compare
        JSR     InitRTI ; Initialize RTI
        CLI                     ; Enable the interrupt system
        BSET    OPTION,X BIT7   ; Turn on A/D converter
        LDAA    #%00110000      ; Setting for A/D converter
        STAA    ADCTL,X         ; Scan continuously and multiple channel

        LDAA    #40             ; Wait until the power of A/D converter
WAIT    DECA                    ; to stablize. (Charge up the Caps)
        BNE     WAIT


        LDAA    #%00110000      ; Direction of PortD 5:4 for L & R tracks
        STAA    DDRD,X          ; Set to output to control the tracks
        JSR     Sensors         ; Get the sensors reading
        JSR     Sensors         ; Usually the first reading is junky

* Always start in tank mode
        LDAA    POS_ADR         ; Retract all the legs to their highest pos
        ANDA    #%00111111      ; and start in tank mode
        EORA    #%00111111      ; If not all of the position sensors are closed
        BEQ     START           ; call the subroutine to transform to tank mode
        LDAA    #1
        STAA    MODE
        JSR     M1_M0

* Reset the motion sensor
START   LDY     #OUTPUT3        ; Reset the D-FF in the motion detection
        BCLR    0,Y BIT7        ; amplifying circuit
        LDAA    OUTPUT3
        STAA    MOTION
        BSET    0,Y BIT7
        LDAA    OUTPUT3
        STAA    MOTION

REPEAT  JSR     M_SET           ; Set the position of sensors according to
                               ; the mode
        JSR     Sensors         ; Get readings from all the sensors
        JSR     C_AVOID         ; Collision avoidance

* Check if Battery is ok
        LDAA    BATTERY         ; Check battery
        LSRA                    ; if lower than 9v, turn on a LED
        CMPA    #$68            ; and transform to tank mode
        BGE     B_OK
        LDY     #OUTPUT3
```

```
        BSET    0,Y BIT3
        LDAA    OUTPUT3
        STAA    LED

        JSR     M1_M0

B_OK    LDAA    TCNT,X          ; Randomly check (spin around) if
        ANDA    #%11111100      ; any human around
        BNE     CONT
        JSR     D_HUMAN         ; Check if anyone in front of me

CONT    LDAA    MODE            ; Check the mode, and call the corresponding
        BEQ     A0              ; arbitrator

* Check if the legs are stuck
        LDAA    Timeout         ; If stuck more than 5 secs, transform back
        BEQ     A1              ; to tank mode
        JSR     M1_M0
        LDAA    #0              ; Reset the timeout variable
        STAA    Timeout
        BRA     A0              ;
A1      JSR     M0_M1           ; else transform to walking mode
        JSR     MODE1           ; Call walking mode arbitrator
REPEAT_ BRA     REPEAT

A0      JSR     M1_M0           ; otherwise, transform to tank mode
        JSR     MODE0           ; Call tank mode arbitrator


* Output sensors reading to terminal (Testing only)
        PSHX
        LDX     #IR0
        JSR     out1byt
        JSR     out1byt
        JSR     out1byt

        JSR     out1byt
        JSR     out1byt
        JSR     out1byt

        JSR     out1byt
        JSR     out1byt
        JSR     out1byt
        JSR     out1byt
        JSR     out1byt

        JSR     out1byt
        jsr     outcrlf
        LDX     #$FFFF
TEST    DEX
        BNE     TEST
        PULX

        BRA     REPEAT_
*****************************************************************
*                       SUBROUTINE - InitOC
*****************************************************************
InitOC  PSHA
        PSHX
        LDX     #BASE
        BCLR    TMSK2,X BIT1    ; Set the pre-scaler frequency for TCNT
        BCLR    TMSK2,X BIT2    ; PR1:PR0=00 for 2MHz

* Initialize Output compare OC2 and OC3
        LDAA    #%10100000      ; OM2:Ol2 = 10 for setting to low
        STAA    TCTL1,X         ; OM3:OL3 = 10 for setting to low

        LDAA    #%01100000      ; Enable interrupt from OC2 and OC3
        STAA    TMSK1,X

        PULX
```

```
        PULA

        RTS
*****************************************************************************
*           Interupt Service Routine - TOC2ISR
*       Create the PWM according to High1
*****************************************************************************
TOC2ISR LDX     #BASE
        BRCLR   TFLG1,X BIT6 TOC2RTI   ; Check for correct interrupt

        LDAA    #BIT6           ; Clear the OC2 flag
        STAA    TFLG1,X

        LDD     High1
        BNE     MOVE1
STOP1   BCLR    TCTL1,X BIT6
        BRA     TOC2RTI

MOVE1   BRSET   TCTL1,X BIT6 LASTHI1
        BSET    TCTL1,X BIT6
        LDD     #Period
        SUBD    High1
        ADDD    TOC2,X
        STD     TOC2,X
        BRA     TOC2RTI

LASTHI1 BCLR    TCTL1,X BIT6
        LDD     High1
        ADDD    TOC2,X
        STD     TOC2,X

TOC2RTI RTI

*****************************************************************************
*           Interupt Service Routine - TOC3ISR
*       Create the PWM according to High0
*****************************************************************************
TOC3ISR LDX     #BASE
        BRCLR   TFLG1,X BIT5 TOC3RTI   ; Check for correct interrupt

        LDAA    #BIT5           ; Clear the OC3 flag
        STAA    TFLG1,X

        LDD     High0
        BNE     MOVE0
STOP0   BCLR    TCTL1,X BIT4
        BRA     TOC3RTI

MOVE0   BRSET   TCTL1,X BIT4 LASTHI0
        BSET    TCTL1,X BIT4
        LDD     #Period
        SUBD    High0

        ADDD    TOC3,X
        STD     TOC3,X
        BRA     TOC3RTI

LASTHI0 BCLR    TCTL1,X BIT4
        LDD     High0
        ADDD    TOC3,X
        STD     TOC3,X

TOC3RTI RTI

*****************************************************************************
*           Interupt Service Routine - RTI_ISR
*****************************************************************************
RTI_ISR LDX     #BASE
        BRCLR   TFLG2,X BIT6 RTI_RTI

        LDAA    #BIT6
```

```
        STAA    TFLG2,X

        LDX     COUNT     ; Increase the COUNT (FOR TESTING ONLY)
        INX
        STX     COUNT

        LDAA    MODE      ; If in walking mode
        BEQ     RTI_RTI

        LDX     W_COUNT   ; Start counting
        INX
        STX     W_COUNT

        CPX     #1250     ; If in the walkin subroutine for > 5 secs
        BLT     RTI_RTI

        LDAA    #1      ; Timeout, tranform back to tank
        STAA    Timeout

        LDX     #0       ; Reset the counter
        STX     W_COUNT

RTI_RTI RTI
*********************************************************************
*           Tank Moving Mechanism Control - TMMC
*       Change the direction of the tracks according to Dir0 & Dir1
*       Calculate High0 & High1 according to Track0 & Track1
*********************************************************************
TMMC    PSHA
        PSHB
        PSHX
        LDX     #BASE

* Left Track
LEFT1   LDAA    Dir0
        BNE     BACK0
FORW0   BCLR    PORTD,X BIT5
        BRA     Set_H0
BACK0   BSET    PORTD,X BIT5

Set_H0  LDAA    Track0
        LDAB    #$FF
        MUL
        STD     High0

* Right Track
RIGHT1  LDAA    Dir1
        BNE     BACK1
FORW1   BCLR    PORTD,X BIT4
        BRA     Set_H1
BACK1   BSET    PORTD,X BIT4

Set_H1  LDAA    Track1
        LDAB    #$FF
        MUL
        STD     High1

        PULX
        PULB
        PULA
        RTS
*********************************************************************
*       Sensors Data Gathering - Sensors
*       Gather all sensors information
*********************************************************************
Sensors PSHA
        PSHX
        LDX     #BASE

        LDAA    $4000
        STAA    Positn
```

```
        LDAA    $5000
        STAA    Touch
        LDAA    $6000
        STAA    Bump

        LDAA    #$FF                ; Turn on IR LED
        STAA    IR_ADR

        LDAA    #%00010000            ; Start sampling Channel 1 - 4
        STAA    ADCTL,X

CHECK   BRCLR   ADCTL,X BIT7 CHECK

        LDAA    ADR1,X
        STAA    IR0
        LDAA    ADR2,X
        STAA    IR1
        LDAA    ADR3,X
        STAA    IR2
        LDAA    ADR4,X
        STAA    IR3

* Select the first CDS
        LDAA    OUTPUT3
        ANDA    #%11111000
        STAA    CDS_SEL
        STAA    OUTPUT3

*         LDAA    #%00000000
*         STAA    CDS_SEL

        LDAA    #%00010100
        STAA    ADCTL,X

CHECK2  BRCLR   ADCTL,X BIT7 CHECK2

        LDAA    ADR1,X
        STAA    IR4
        LDAA    ADR2,X
        STAA    IR5
        LDAA    ADR3,X
        STAA    CDS0
        LDAA    ADR4,X
        STAA    BATTERY

*        LDAA    #0              ; Turn off the IR LED
*        STAA    IR_ADR
* Select the second CDS

        LDAA    OUTPUT3
        ANDA    #%11111000
        EORA    #%00000001
        STAA    CDS_SEL
        STAA    OUTPUT3
*         LDAA    #%00000001
*         STAA    CDS_SEL


        LDAA    #%00000110
        STAA    ADCTL,X
        LDAA    #6
WAIT3   DECA
        BNE     WAIT3

        LDAA    ADR1,X
        STAA    CDS1
* Select the third CDS
        LDAA    OUTPUT3
        ANDA    #%11111000
        EORA    #%00000010
        STAA    CDS_SEL
```

```
        STAA    OUTPUT3
*         LDAA    #%00000010
*         STAA    CDS_SEL

        LDAA    #%00000110
        STAA    ADCTL,X
        LDAA    #6
WAIT4   DECA
        BNE     WAIT4
        LDAA    ADR1,X
        STAA    CDS2

* Select the forth CDS
        LDAA    OUTPUT3
        ANDA    #%11111000
        EORA    #%00000011
        STAA    CDS_SEL
        STAA    OUTPUT3
*         LDAA    #%00000011
*         STAA    CDS_SEL

        LDAA    #%00000110
        STAA     ADCTL,X
        LDAA    #6
WAIT5   DECA
        BNE     WAIT5
        LDAA    ADR1,X
        STAA    CDS3

* Select the fifth CDS
        LDAA    OUTPUT3
        ANDA    #%11111000
        EORA    #%00000100
        STAA    CDS_SEL
        STAA    OUTPUT3
*         LDAA    #%00000100
*         STAA    CDS_SEL


        LDAA    #%00000110
        STAA     ADCTL,X
        LDAA    #6
WAIT6   DECA
        BNE     WAIT6
        LDAA    ADR1,X
        STAA    CDS4

        PULX
        PULA
        RTS
*******************************************************************************
*******************************************************************************
*                       SUBROUTINE - InitSCI
* Description: This subroutine initializes the BAUD rate to 9600 and
*              sets up the SCI port for 1 start bit, 8 data bits and
*              1 stop bit.  It also enables the transmitter and receiver.
*              Effected registers are BAUD, SCCR1, and SCCR2.
* Input         : None.
* Output        : Initializes SCI.
* Destroys      : None.
* Calls         : None.
*******************************************************************************
*
InitSCI PSHA                            ; Save contents of A register

        LDAA    #$30                    ; Set BAUD rate to 9600
        STAA    BAUD
        LDAA    #$00                    ; Set SCI Mode to 1 start bit,
        STAA    SCCR1                   ;     8 data bits, and 1 stop bit.
        LDAA    #%00001100              ; Enable SCI Transmitter
        STAA    SCCR2                   ;     and Receiver
```

```
       PULA                            ; Restore A register
       RTS                             ; Return from subtoutine
*
*********************************************************************
*       Set the mode of the robot (track or walking) - M_SET
*       Determine which sensors are the front sensors and the rear sensors
*       .... etc. So that I can use the same subroutine in either walking
*       or tank mode.
*********************************************************************
M_SET   PSHX
       PSHA
       LDAA    MODE
       BNE     M_WALK

* Tank Mode
M_TANK  LDX     #IR2
       STX     FR_IR
       LDX     #IR3
       STX     FL_IR
       LDX     #IR5
       STX     R_IR
       LDAA    #BUMP1
       STAA    FL_BUMP
       LDAA    #BUMP2
       STAA    RL_BUMP
       LDAA    #BUMP3
       STAA    FR_BUMP
       LDAA    #BUMP4
       STAA    RR_BUMP
       BRA     M_SET_E

* Walking Mode
M_WALK  LDX     #IR0
       STX     FR_IR
       LDX     #IR1
       STX     FL_IR
       LDX     #IR4
       STX     R_IR
       LDAA    #BUMP3
       STAA    FL_BUMP
       LDAA    #BUMP1
       STAA    RL_BUMP
       LDAA    #BUMP4
       STAA    FR_BUMP
       LDAA    #BUMP2
       STAA    RR_BUMP

M_SET_E PULA
       PULX
       RTS
*********************************************************************
*       Collision Avoidance - C_AVOID
*********************************************************************
C_AVOID PSHA
       PSHB
       PSHX

C4      LDX     FL_IR
       LDAA    0,X
       SUBA    #120
       CLV
       BGT     C1
       NEGA
       LDAB    #7
       MUL
       STAB    CA_R
       LDAA    #0
       STAA    Dir1
       BRA     C2
```

```
C1      LDAB    #25
        MUL
        STAB    CA_R
        LDAA    #1
        STAA    Dir1

C2      LDX     FR_IR
        LDAA    0,X
        SUBA    #120
        CLV
        BGT     C3
        NEGA
        LDAB    #7
        MUL
        STAB    CA_L
        LDAA    #0
        STAA    Dir0
        BRA     CA_E

C3      LDAB    #25
        MUL
        STAB    CA_L
        LDAA    #1
        STAA    Dir0

CA_E    PULX
        PULB
        PULA
        RTS
*

*
*************************************************************************
*************************************************************************
*       Arbitrator in Tank mode - MODE0
*************************************************************************
MODE0   PSHA
        PSHB
        PSHX
* Bump sensor check
FL_C    LDAA    B_ADR
        ANDA    FL_BUMP
        BNE     FR_C
FL_B    LDAA    #1
        STAA    Dir1
        STAA    Dir0

        LDAA    #$00
        STAA    Track0
        LDAA    #$FF
        STAA    Track1

        JSR     TMMC

FL_BC   LDAA    B_ADR
        ANDA    FL_BUMP
        BEQ     FL_BC

        LDAA    #$05
FL_BD1  LDX     #$FFFF
FL_BD2  DEX
        BNE     FL_BD2
        DECA
        BNE     FL_BD1

        BRA     TRAP_


FR_C    LDAA    B_ADR
        ANDA    FR_BUMP
        BNE     RL_C
```

```
FR_B     LDAA    #1
         STAA    Dir1
         STAA    Dir0

         LDAA    #$FF
         STAA    Track0
         LDAA    #$00
         STAA    Track1

         JSR     TMMC

FR_BC    LDAA    B_ADR
         ANDA    FR_BUMP
         BEQ     FR_BC

         LDAA    #$05
FR_BD1   LDX     #$FFFF
FR_BD2   DEX
         BNE     FR_BD2
         DECA
         BNE     FR_BD1

TRAP_    BRA     TRAP

RL_C     LDAA    B_ADR
         ANDA    RL_BUMP
         BNE     RR_C
RL_B     LDAA    #0
         STAA    Dir1
         STAA    Dir0

         LDAA    #$00
         STAA    Track0
         LDAA    #$FF
         STAA    Track1

         JSR     TMMC

RL_BC    LDAA    B_ADR
         ANDA    RL_BUMP
         BEQ     RL_BC

         LDAA    #$05
RL_BD1   LDX     #$FFFF
RL_BD2   DEX
         BNE     RL_BD2
         DECA
         BNE     RL_BD1

         BRA     TRAP


RR_C     LDAA    B_ADR
         ANDA    RR_BUMP
         BNE     TRAP
RR_B     LDAA    #0
         STAA    Dir1
         STAA    Dir0

         LDAA    #$FF
         STAA    Track0
         LDAA    #$00
         STAA    Track1

         JSR     TMMC
RR_BC    LDAA    B_ADR
         ANDA    RR_BUMP
         BEQ     RR_BC

         LDAA    #$05
RR_BD1   LDX     #$FFFF
```

```
RR_BD2  DEX
        BNE     RR_BD2
        DECA
        BNE     RR_BD1
* if traped, turn; otherwise following the IR reading
TRAP    LDAA    CA_L
        ANDA    #%11100000
        BNE     MODE0_E

        LDAA    CA_R
        ANDA    #%11100000
        BNE     MODE0_E


        LDX     #BASE           ; If traps, randomly transform
        LDAA    TCNT,X
        ANDA    #%11110000
        BNE     M0_M4

        JSR     M0_M1
        BRA     END


M0_M4   LDX     #BASE           ; Randomly turn left or right when traped
        LDAA    TCNT,X
        ANDA    #%10000000
        BEQ     M9
        LDAA    #0
        STAA    Dir0
        LDAA    #1
        STAA    Dir1
        BRA     M10

M9      LDAA    #0
        STAA    Dir1
        LDAA    #1
        STAA    Dir0

M10     LDAA    #$80
        STAA    Track0
        STAA    Track1

        JSR     TMMC

        LDAA    #05
M31     LDX     #$FFFF
M30     DEX
        BNE     M30
        DECA
        BNE     M31
        BRA     END


* following the IR reading for collision avoidance
MODE0_E LDAA    CA_L
        STAA    Track0
        LDAA    CA_R
        STAA    Track1
        JSR     TMMC
END     PULX
        PULB
        PULA
        RTS



*******************************************************************
*******************************************************************
*       Transform from Tank mode to Walking mode - M0_M1
*******************************************************************
```

```
M0_M1   PSHA                    ; Return if already in mode 1
        LDAA    MODE
        BNE     M0_M1_E

        LDAA    #$00            ; Stop the tracks
        STAA    Track0
        STAA    Track1

        JSR     TMMC

        LDAA    #%00101010      ; Set the leg direction
        STAA    LEG_DIR
CK       LDAA    T_ADR          ; move three legs until they touch the ground
        ANDA    #%00010101
        BEQ     ST1
        STAA    LEG_EN
        BRA     CK
ST1      LDAA    T_ADR          ; move other three legs likewise
        ANDA    #%00101010
        BEQ     ST2
        STAA    LEG_EN
        BRA     ST1
ST2      LDAA    #0
        STAA    LEG_EN

        LDAA    #%00101010      ; Set the walking pattern
        STAA    WALK_P
M0_M1_E LDAA    #1              ; Set mode to 1
        STAA    MODE
        PULA
        RTS
*************************************************************************
*       Transform from Walking mode to Tank mode - M1_M0
*************************************************************************
M1_M0   PSHA                    ; Return if already in mode 0
        LDAA    MODE
        BEQ     M1_M0_E

        LDAA    #%00010101      ; Set direction of the legs
        STAA    LEG_DIR
CH       LDAA    POS_ADR        ; Move all the legs until all the position
        ANDA    #%00111111      ; sensors are closed
        EORA    #%00111111
        BEQ     SP
        STAA    LEG_EN
        BRA     CH
SP       LDAA    #0
        STAA    LEG_EN

M1_M0_E LDAA    #0
        STAA    MODE
        PULA

        RTS
*************************************************************************
*       Walking algorithm - WALK
*************************************************************************
WALK    PSHA
        PSHB

        LDD     #0              ; Reset W_count
        STD     W_COUNT
        LDAA    Timeout         ; Check if the Timeout flag is set by the RTI
        BNE     W3
        LDAA    WALK_P          ; move the legs according to the walking pattern

        STAA    LEG_EN
W1       LDAA    T_ADR          ; keep moving until the legs touch the round
        ANDA    WALK_P
        EORA    WALK_P
        BEQ     W2
```

```
        STAA    LEG_EN
        LDAA    Timeout         ; Check for timeout
        BNE     W3
        BRA     W1
W2      LDAA    WALK_P          ; negate and update the walking pattern
        EORA    #%00111111      ; move the other legs forward
        STAA    LEG_EN
        STAA    WALK_P
* Wait until all three legs left the ground
W5      LDAA    T_ADR
        ANDA    WALK_P
        BEQ     W5

W4      LDAA    T_ADR           ; move until the legs touch the round
        ANDA    WALK_P
        BEQ     W3
        STAA    LEG_EN
        LDAA    Timeout         ; Check for timeout
        BNE     W3
        BRA     W4
W3      LDAA    #0              ; Stop all the legs
        STAA    LEG_EN
        PULB
        PULA
        RTS

*************************************************************************
*       TURNING LEFT - TURN_L
*       Change the direction of the legs and walking pattern and then call
*       the walk subroutine
*************************************************************************
TURN_L  PSHA
        LDAA    #LEFT
        STAA    LEG_DIR
        LDAA    WALK_P
        EORA    #RIGHT
        STAA    WALK_P
        JSR     WALK
        LDAA    WALK_P
        EORA    #RIGHT
        STAA    WALK_P
        INC     T_C
        PULA
        RTS
*************************************************************************
*       TURNING RIGHT - TURN_R
*       Change the direction of the legs and walking pattern and then call
*       the walk subroutine
*************************************************************************
TURN_R  PSHA
        LDAA    #RIGHT
        STAA    LEG_DIR
        LDAA    WALK_P
        EORA    #LEFT
        STAA    WALK_P
        JSR     WALK
        LDAA    WALK_P
        EORA    #LEFT
        STAA    WALK_P
        INC     T_C
        PULA
        RTS
*************************************************************************
*       Walking FORWARD - WALK_F
*       Change the direction of the legs and walking pattern and then call
*       the walk subroutine
*************************************************************************
WALK_F  PSHA
        LDAA    #FORWARD
        STAA    LEG_DIR
        JSR     WALK
```

```
        PULA
        RTS
*******************************************************************************
*       Walking BACKWARD - WALK_B
*       Change the direction of the legs and walking pattern and then call
*       the walk subroutine
*******************************************************************************
WALK_B  PSHA
        LDAA    #BACKWARD
        STAA    LEG_DIR
        JSR     WALK
        LDAA    WALK_P
        EORA    #%00111111
        STAA    WALK_P
        PULA
        RTS

*******************************************************************************
*       Arbitrator in WALKING mode - MODE1
*******************************************************************************
MODE1   PSHA
        PSHB
        PSHX

* Check the bump sensors
FL_C1   LDAA    B_ADR
        ANDA    FL_BUMP
        BNE     FR_C1
        JSR     TURN_R
        BRA     FL_C1

FR_C1   LDAA    B_ADR
        ANDA    FR_BUMP
        BNE     RL_C1
        JSR     TURN_L
        BRA     FR_C1

RL_C1   LDAA    B_ADR
        ANDA    RL_BUMP
        BNE     RR_C1
        JSR     TURN_L
        BRA     RL_C1

RR_C1   LDAA    B_ADR
        ANDA    RR_BUMP
        BNE     TRAP1
        JSR     TURN_R
        BRA     RR_C1

* if traped, turn; otherwise following the IR reading
TRAP1   JSR     D_MOTN          ; spin around and detect human
        LDAA    CA_L
        ANDA    #%11100000
        BNE     MODE1_E

        LDAA    CA_R
        ANDA    #%11100000
        BNE     MODE1_E

*        JSR     D_MOTN
        LDAA    HUMAN           ; jiggle the tail if see a human
        BNE     H1
        JSR     S_TAIL

H1      LDX     #BASE           ; Maybe transform back to tank
        LDAA    TCNT,X
        ANDA    #%00010000
        BNE     M1_T

        JSR     M1_M0
        BRA     END1
```

```
M1_T    LDX     #BASE           ; Or turn left or right
        LDAA    TCNT,X
        ANDA    #%10000000
        BEQ     M1_9

        JSR     TURN_L
        JSR     TURN_L
        JSR     TURN_L
        JSR     TURN_L
        BRA     MODE1_E

M1_9    JSR     TURN_R
        JSR     TURN_R
        JSR     TURN_R
        JSR     TURN_R

* following the IR reading for collision avoidance
MODE1_E LDAA    Dir0
        EORA    Dir1
        BEQ     M1_1
        LDAA    Dir0
        BEQ     M1_R
M1_L    LDAA    HUMAN       ; Whenever see a human in front of it,
        BNE     H2          ; jiggle the tail
        JSR     S_TAIL

H2      JSR     TURN_L
        BRA     END1
M1_R    LDAA    HUMAN
        BNE     H3
        JSR     S_TAIL

H3      JSR     TURN_R
        BRA     END1

M1_1    LDAA    Dir0
        BEQ     M1_F
M1_B    LDAA    HUMAN
        BNE     H4
        JSR     S_TAIL

H4      JSR     WALK_B
        BRA     END1
M1_F    JSR     WALK_F

        LDAA    T_C             ; If turned too many times, transform back
        ANDA    #%11110000      ; to tank mode
        BEQ     END1
        LDAA    #0
        STAA    T_C
        JSR     M1_M0

END1    PULX
        PULB
        PULA
        RTS



*************************************************************************
*                       SUBROUTINE - InitRTI
*************************************************************************
InitRTI PSHA
        PSHX
        LDX     #BASE
        LDAA    #$00    ; interrupt every 4 ms
        STAA    PACTL,X
        LDAA    #%01000000
        STAA    TMSK2,X
```

```
        PULX
        PULA
        RTS

*************************************************************************
*                       SUBROUTINE - D_HUMAN
*       Spin around the detect human, stop when detected. Or keep moving
*       around after a while and no human around. Return 1 in Human when
*       detected
*************************************************************************

D_HUMAN PSHA
        PSHX
        PSHY

        LDAA    MODE
        BNE     T1
T0       LDAA    #0
        STAA    Dir1
        LDAA    #1
        STAA    Dir0

        LDAA    #$B0
        STAA    Track0
        STAA    Track1

        JSR     TMMC

        LDAB    #$0A
M0_M3    LDX     #$FFFF
M0_M     JSR     D_MOTN
        LDAA    HUMAN
        BNE     M0_M2
        DEX
        BNE     M0_M
        DECB
        BNE     M0_M3
        BRA     DH_E

T1       LDAB    #10
        JSR     TURN_L
        JSR     D_MOTN
        LDAA    HUMAN
        BNE     M0_M2
        DECB
        BNE     T1

        BRA     DH_E
M0_M2    JSR     M0_M1

DH_E     PULY
        PULX
        PULA
        RTS

*************************************************************************
*                       SUBROUTINE - D_MOTN
*       Reset the D-FF of the motion sensors. Wait for a while, if human
*       is detected, return 1 in Human.
*************************************************************************
D_MOTN   PSHA
        PSHY

        LDAA    #0
        STAA    HUMAN

        LDY     #OUTPUT3
        BCLR    0,Y BIT7
        LDAA    OUTPUT3
        STAA    MOTION
        BSET    0,Y BIT7
```

```
        LDAA    OUTPUT3
        STAA    MOTION

        LDAA    #$FF
D1       DECA
        BNE     D1

        LDAA    MOTION
        ANDA    #%00010000
        BEQ     DM_E
        LDAA    #1
        STAA    HUMAN

DM_E     PULY
        PULA
        RTS
*************************************************************************
*                       SUBROUTINE - S_TAIL
*       Jiggle the tail by moving the track in the rear end back and forth
*************************************************************************
S_TAIL   PSHA
        PSHB
        PSHX
        JSR     M1_M0

        LDAA    #0
        STAA    Track1
        LDAA    #$FF
        STAA    Track0

        LDAA    #0

        LDAB    #6
S_W3     LDX     #$1000
S_W2     STAA    Dir0
        JSR     TMMC
        DEX
        BNE     S_W2
        EORA    #%00000001
        DECB
        BNE     S_W3
        LDAA    #0
        STAA    Track0
        STAA    Track1

        JSR     TMMC

        JSR     M0_M1
ST_E     PULX
        PULB
        PULA
        RTS
```