



\*166166\*



\*G00001\*

## **ELECTRICAL MOP**

### **THE SPECIFICATION**

#### **A THE TITLE**

##### **ELECTRICAL MOP**

#### **B THE DESCRIPTION**

##### **1: Purpose of the Electrical Mop**

It is microcontroller based and can mop the floor autonomously. The mop can start, move and stop in the ways to accommodate different floor environments.

##### **2: The Sequence of the Movements of the Mop**

Case 1: In the open floor. Initially, the mop is in the idle state and near a wall. When started, it will move ahead until it meets the wall in front. It will turn right if there is a wall on the left side, or turn left if there is a wall on the right side. The turning will continue until the mop is turned 180 degree when the direction of the movement is just reversed. the mop move ahead until it meets the wall in front again. It reverse the move direction by turning right if the immediate past turn is in left or turn left if the immediate past turn is in right 180 degree. The above sequence continues until the mop is in the corner of the wall or it is stopped by the user. The move trajectories is shown in Fig. 1 and Fig. 2.

Case 2: In the open floor. Initially, the mop is in the idle state and far from the wall which sensor can not detect. When it is started, it will move ahead until it meets the wall; then it will turn left 180 degree first and the following movement patterns are the same as those in Case 1. The move trajectory is shown in Fig.3.

Case 3: Among the sofa and furniture. Initially, the mop is in idle. When started, it moves ahead, turn left/ right 90 degree when it meets an obstacle, and then go ahead. Turn left /right degree again when it meets an obstacle again. The move trajectory is shown in Fig. 4.

### **3: Mechanical Structure**

This microcontroller based electrical mop is made up by a body and a paper mop which is attached to the rear part of the body. It is showed in Fig 5,6 & 7.

#### **a The Paper Mop**

It has a rectangular wiper and a long handle. The wiper is about 23cm long and 9cm wide. It consists of an upper plastic layer and a lower rubber layer. When it is in use, a special made cleaning paper will cover the whole rubber layer and then folded and inserted into the clip or catcher at the upper plastic layer. As the electrical mop move around a room, this wiper with the special made paper will clean up the floor.

The handle is about 12 cm long and 2 cm in diameter. It is connected to the upper plastic layer of the wiper and the other end is attached to the rear part of the body of the electrical mop. It also provides light contact pressure to the wiper against the floor.

#### **b The Body of the Electrical Mop**

The Main body of the electrical mop are for the accommodation of the rechargeable batteries PCB board sensors and the rear part is attached by the handle of the body mop. It is showed in Fig 5, 6 & 7.

The differential drive system with two independent motors/wheels is used to drive the mop.

Four castor wheels are installed to provide the additional support for the weight and to stabilise the movement of the mop.

**Fig 5 : The structure of the Electrical Mop Front view**

- a: PCB Board**
- b: Battery Rechargeable**
- c: Left and right motors to control the wheels**
- d: Left and right wheels**
- e: Sensor C**

**Fig 6 : The Structure of the Electrical Mop Top View**

- a: The wiper of the paper mop. It is covered by a special cleaning paper to clean the floor. It is about 23cm x 9cm.
- b: The Handle of the Paper Mop
- c: The left and right wheels
- d: Sensor R. It is at the right side of the body
- e: Sensor L. It is at the left side of the body
- f: Sensor C. It is at the Front of the body
- g: The Main Body
- h: The special cleaning paper for cleaning the floor

**Fig 7 : The Structure of the Electrical Mop Side View**

- a: The wiper of the paper mop
- b. The body of the paper mop.
- c. The handle of the paper mop . It connects to the wiper on the body of the electrical mop.
- d. The right wheel.
- e. The sensor C : It is at the Front of the Electrical Mop
- f.. The sensor R: It is at the Right side of the Body
- g. The special cleaning paper to clean the floor

#### **4: Electronics Hardware Design**

The control system must be able to control the mop to move in the ways specified in Fig 1 to 4 . In view of this requirement, the microcontroller MCU H8/330 HITACHI is chosen. It has two PWM ports which are very useful to simultaneously control two motors, one is in the left called Motor\_L and the other is in the right called Motor\_R. Three sensors are installed in the central front called Sensor\_C, in the left called Sensor\_L and in the right called Sensor\_R to feedback the information such as the wall/obstacle in front, in the left or in the right of the mop. To record the position of the mop and velocities of two motors which are important parameters for the control system, an encoder is equipped with each motor.

There are circuits designed to interface the infra-red sensors and the encoders with the MCU. Sensor interface circuit is done according to the wiring diagram of the products; the decoder HCTL 2016 is used to counts the pulses generated from decoders and send them to the MCU. The drivers UDN2953 are used to drive two motors; the drive chips are controlled by the PWM signals from MCU. A LED bar driven by a MCU port is used for the purpose of debugging and troubleshooting. The program is stored in a EPROM M27C256.. The address decoding for all the devices is done by the address bus and a GAL chip GAL16V8B . The program for the GAL chip MacAble language is listed in Appendix 1..

The following Table. 1 list the port and address allocations in the circuit design.

| Address Allocations |            | Port Allocations                |        |               |
|---------------------|------------|---------------------------------|--------|---------------|
| Device              | Address    | Purpose                         | Port   | Pin           |
| EPROM               | 0000-7FFFF | Sensor_L IRQ4                   | P8/IN  | P8.5          |
| 2016 Res            | C000       | Sensor_C IRQ3                   | P8/IN  | P8.4          |
|                     |            | Sensor_R IRQ5                   | P8/IN  | P8.6          |
| 2016_R<br>High Byte | D000       | Motor_R PWM<br>OE 0: ON; 1: OFF | P4/OUT | P4.7          |
| Low Byte            | D001       | PH 0: FW, 1: BW                 | P4/OUT | P4.4          |
| 2016_L<br>High Byte | A000       | Motor_L PWM<br>OE 0: ON; 1: OFF | P4/OUT | P4.6          |
| Low Byte            | A001       | PH 0: FW, 1: BW                 | P4/OUT | P4.5          |
|                     |            | LED                             | P6/OUT | P6.0-<br>P6.7 |

Table 1: Port and Address Allocation

## 5. Software Design for Robot Soft 1

### 5.1: The Control Scheme

The main part of the software is to implement the control algorithm, combined with the sensor signals which reflect the status of the mop movement and surrounding environment.

The control algorithm must make sure that two motors run synchronously so that mop can move straight, turn right, turn left or stop in the ways required. Fig.8 shows the basic control loops used in this project.

In the Fig 8 control loops,

$n_{dl}$  : the desired speed of the left motor

$n_{dr}$  : the desired speed of the right motor

$n_l$  : the actual speed of the left motor

$n_r$  : the actual speed of the right motor

$K_p$  : proportional gain

$K_i$  : integral gain

*Bias*: turning speeds the motors

PWM\_L: the PWM value of the left motor

PWM\_R: the PWM value of the right motor

The equations of the above control is:

$$PWM\_L = \int_0^t K_p [(n_{dl} - n_l) - K_i \int_0^t (n_l - n_r + bias) dt] dt$$

$$PWM\_R = \int_0^t K_p [(n_{dr} - n_r) + K_i \int_0^t (n_l - n_r + bias) dt] dt$$

In the above, the PWM\_L and PWM\_R is the outputs of the controller which determine the duty ratio of the PWM signals, and thus determine the speed the motor speeds. The lowest and maximum values are limited by MCU. (the values of 0 and ff as required by the MCU). The velocity is measured by counts/sample; and the acceleration is measured by counts/sample/sample.

After the calibrations and experiments, the parameters used in the controller are chosen as shown in the table below:

| Main Parameters           | Value | Unit                 |
|---------------------------|-------|----------------------|
| Sampling Time (T)         | 10    | ms                   |
| Desired Speed/Left Motor  | 300   | counts/sample        |
| Desired Speed/Right Motor | 300   | counts/sample        |
| Acceleration/Deceleration | 10    | counts/sample/sample |
| Proportional Gain         | 01    |                      |
| Integral Gain             | 0.1   |                      |
| 45 Degree Turn Counts     | 47712 | counts               |
| Bias of Left Motor        | 300   | counts/sample        |
| Bias of right Motor       | 300   | counts/sample        |

The PPR of the encoder is 180; which means one revolts of the motor will produce 720 counts. After calculation, the above desired motor speed roughly corresponds to the mop speed 120mm/s. Because the slippage of the wheel and the capacity of the battery, the actual speed may be lower than this value.

## **5.2: Robot Soft 1 Programe**

In the program, the H8/330 Timer 1 is used as the sampling time generator. In every 10 ms intervals, the speeds of the motors are updated, the control algorithm is processed and the PWM values are fed into the PWM times. In addition, the interruption of the center sensors is provided; the status of the left and right sensors are feedback using polling method. The flow charts of the program are showed on fig 9 to 12, based on the mop movements specified, hardware configuration and the control algorithms. The source codes of the program can be found in Appendix 2.

**Fig 9 Flow Chart of the main programme**

**Fig 10 Flow Chart of the Interruptions**

**Fig 11 Flow Chart of the 10 m/s Sampling Interrupts**

**Fig 12 Flow Chart of the Sensor C Interruptions**

## APPENDIX I

## ROBOTSOFT I : SOFTWARE PROGRAMME

## A PROGRAM FOR ADDRESS DECODING GAL ABLE FILE

```

module      mop
title       'addressing decoding for mop project'

mop         device      'p16v8r'

A15,A14,A13,A12      pin    2,3,4,5;
O5,O4,O3,O2          pin    19,18,17,16  istype 'com,invert';

H,L,Z,X      =      1,0,.z,.,x;

equations

!O5=A15&A14&A13&!A12;
!O4=A15&A14&!A13&A12;
!O3=A15&!A14&A13&!A12;
!O2=A15&A14&!A13&!A12;

test_vectors ([A15,A14,A13,A12] -->[O5,O4,O3,O2])

[1,1,1,0]-->[0,1,1,1];
[1,1,0,1]-->[1,0,1,1];
[1,0,1,0]-->[1,1,0,1];
[1,1,0,0]-->[1,1,1,0];

end

```

## Appendix 2

### ROBOTSOFT I SOFTWARE PROGRAMME

#### B PROGRAM OF THE CONTROLLER

---

Mop speed control--moves in a zigzag way or turn right around all the way in a closed trajectory . In a zigzag way, the mop goes straight when there is no obstacle, then first time only turn left 180 when sensor L and R is disabled and the mop meets the wall; turn left 180 when sensor R is on and the mop meets the wall; turn right 180 when sensor L is on and the mop meets the wall; Continue until it is in the corner of the wall.

---

```

*****
;
*****REGISTERS*****
;
*****
; *---port*
p4ddr .equ h'ffb5
p4dr .equ h'ffb7
p6ddr .equ h'ffb9
p6dr .equ h'ffbb
p8ddr .equ h'ffbd
p8dr .equ h'ffbf
p9ddr .equ h'ffc0
p9dr .equ h'ffc1

; *---16-bit timer*
tier .equ h'ff90
tcsr .equ h'ff91
ocra .equ h'ff94
tcr .equ h'ff96
to cr .equ h'ff97
frc .equ h'ff92

; *---8-bit timer*
tcr0 .equ h'ffc8
tcsr0 .equ h'ffc9
tcora0 .equ h'ffca
;tcnt0 .equ h'ffcc

```

```
;*---PWM*
```

```
left_tcr .equ    h'ffa0
right_tcr .equ   h'ffa4
left_dtr .equ    h'ffa1
right_dtr .equ   h'ffa5
```

```
;*---Interrupt*
```

```
syscr .equ    h'ffc4
iscr .equ     h'ffc6
ier .equ      h'ffc7
```

```
*****
;
*****ADDRESSES*****
;
*****
```

```
;*---Encoder 2016*
```

```
left_2016_hi .equ    h'a000 ;high byte of count of left motor
left_2016_lo .equ    h'a001 ;low byte of count of left motor
right_2016_hi .equ   h'd000 ;high byte of count of right motor
right_2016_lo .equ   h'd001 ;low byte of count of left motor
rst_2016 .equ      h'c000 ;reset/clear the counter
```

```
*****
;
*****DEFINED RAM LOCATIONS*****
;
*****
```

```
int_ram .equ    h'fd80
left_count .equ  int_ram + 2
right_count .equ  int_ram + 4
past_left_count .equ  int_ram + 6
past_right_count .equ  int_ram + 8
left_error .equ   int_ram + 13
right_error .equ   int_ram + 15
left_speed .equ   int_ram + 18
right_speed .equ   int_ram + 20
int_error .equ    int_ram + 22
integral .equ     int_ram + 24
left_d_speed .equ  int_ram + 26
right_d_speed .equ  int_ram + 28
status .equ       int_ram + 30
left_bias .equ    int_ram + 31
right_bias .equ   int_ram + 33
left_dir_flag .equ  int_ram + 35
right_dir_flag .equ  int_ram + 36
display_count .equ  int_ram + 39
```

```

stop_flag      .equ    int_ram + 40
turn_flag      .equ    int_ram + 41
past_left_duty .equ    int_ram + 42
past_right_duty .equ    int_ram + 44
left_duty      .equ    int_ram + 46
right_duty     .equ    int_ram + 48
divide         .equ    int_ram + 50
bias_l         .equ    int_ram + 52
bias_r         .equ    int_ram + 54
turn_count     .equ    int_ram + 56
number_turn    .equ    int_ram + 58
turn_dir_flag  .equ    int_ram + 59
first_time_flag .equ    int_ram + 60
move_flag      .equ    int_ram + 61
k_45           .equ    int_ram + 62
count_45       .equ    int_ram + 64

```

```

;*****
;*****DEFINED CONSTANTS*****
;*****
desired_speed  .equ    h'012c ;70% of previous h'0082
desired_count  .equ    h'e5e4
;sample_period .equ    h'62   ;10ms sampling period counts
sample_period  .equ    h'93   ;15ms sampling period counts
display_period .equ    h'7fff ;interval unit for display/0.1sec
;from testing, 3125d (0c35h) only produce 50us time
prop_kl        .equ    h'01   ;proportional gain for left motor
prop_kr        .equ    h'01   ;proportional gain for right motor
int_k          .equ    h'01   ;integral gain
acc            .equ    h'000a ;acceleration
dcc           .equ    h'000a ;deceleration
high_limit     .equ    h'00fa ;high limit of PWM duty ratio
low_limit      .equ    h'0000 ;low limit of PWM duty reatio
scale_l        .equ    h'04 ;h'02 ;scale factor of left motor error
scale_r        .equ    h'05 ;h'01 ;scale factor of right motor error
scale_int      .equ    h'04 ;h'04 ;scale factor of error integral
turn_l         .equ    h'012c ;turn speed of left motor
turn_r         .equ    h'012c ;turn speed of right motor
;count_45      .equ    h'ba60 ;counts of 45 degree(smart turn)
;count_45      .equ    h'cb52 ;counts of 45 degree(smart turn)
;count_45      .equ    h'c2d9 ;counts of 45 degree(smart turn)
;              ( h'c842)

```

```

*****
;
*****INITIALIZATION*****
;
*****
;
        .section vector,data,locate=h'0

; *---reset vector*

        .org    h'00
        .data.w  initialise

; *---NMI interrupt vector*

        .org    h'0006
        .data.w  SELEC

; *---o/p compare A: 8-bit timer 0 10ms interrupt vector*

        .org    h'26
        .data.w  INT_SAMPLING

; *---center sensor interrupt vector

        .org    h'0e
        .data.w  SET_TURN

*****
;
***** PROGRAM *****
;
*****
;

        .section init,code,locate=h'40

*****
;
;INITIALIZE *
;
*****
initialise:
; *--- initialize satck pointer

        mov.w    #h'ff7f,sp    ;stack pointer

; *--- initialise ports *

        mov.b    #h'ff,r1l
        mov.b    r1l,@p4ddr    ;config P4 output)

```

```

mov.b  r1l,@p6ddr      ;config P6 output

mov.b  #h'00,r1l
mov.b  r1l,@p8ddr      ;config. P8 input

;---initialise pwm *

mov.w  #h'00fa,r1
mov.b  r1l,@left_tcr
mov.b  r1l,@right_tcr  ;enable PWMs,)
mov.b  r1h,@left_dtr
mov.b  r1h,@right_dtr

;---initialize ccr and unmask interruptions *
andc   #h'7f,ccr      ;unmask irq

;---initialize interrupts *

mov.w  #h'08ff,r1
mov.b  r1l,@iscr      ;set IRQs falling edge triggered
mov.b  r1h,@ier       ;enable center sensor (IRQ3)

;---initialise 8-bit timer *
mov.b  #sample_period,r0l
mov.b  r0l,@tcora0    ;time constant for 10msec

mov.b  #h'00,r0l
mov.b  r0l,@tcsr0     ;clear the interrupt flag

mov.b  #h'4b,r0l
mov.b  r0l,@tcr0      ;enable the interrupt, clk/1024

;---initialise 2016 (clear to zero),CCR and RAMs recording counts

mov.w  #h'0000,r0
mov.w  r0,@rst_2016
mov.w  r0,@past_left_count
mov.w  r0,@past_right_count
mov.w  r0,@left_count
mov.w  r0,@right_count
mov.w  r0,@left_speed
mov.w  r0,@right_speed
mov.w  r0,@integral
mov.w  r0,@int_error

```

```

mov.w r0,@past_left_duty
mov.w r0,@past_right_duty
mov.w r0,@left_duty
mov.w r0,@right_duty
mov.w r0,@bias_l
mov.w r0,@bias_r
mov.w r0,@turn_count
mov.b r0l,@stop_flag
mov.b r0l,@turn_flag
mov.b r0l,@turn_dir_flag
mov.b r0l,@first_time_flag
mov.b r0l,@move_flag
mov.b r0l,@p6dr
mov.b r0l,@number_turn
mov.w #acc,r0
mov.w r0,@left_d_speed
mov.w r0,@right_d_speed
mov.b #h'04,r0h
mov.b r0h,@k_45
mov.w #h'c2d9,r0
mov.w r0,@count_45

```

```

;*****
;*****MAIN PROGRAM*****
;*****

```

```

start:    bclr  #5,@p4dr    ;set left motor forward
          bclr  #4,@p4dr    ;set right motor forward

```

```

run:      mov.w @left_duty,r0
          mov.w @right_duty,r1
          mov.b r0l,@left_dtr
          mov.b r1l,@right_dtr    ;run

```

```

bra      run

```

```

;*****
;*****INTERRUPTS *****
;*****

```

```

INT_SAMPLING:

```

```

push r0
push r1
push r2

```

```

orc    #h'80,ccr
bclr   #6,@tcsr0
andc   #h'7f,ccr           ;unmask interrupt

jsr    @GET_SPEED           ;get the left/right speed
jsr    @GET_DUTY            ;get the left/right duty from the control

```

```

pop     r2
pop     r1
pop     r0

```

```

rte

```

```

SET_TURN:  push    r0
           push    r1
           push    r2

           orc     #h'80,ccr
           mov.b   #h'00,r0l
           mov.b   r0l,@ier
           andc    #h'7f,ccr           ;unmask interrupts

           mov.b   @turn_flag,r0l
           cmp.b   #h'01,r0l
           beq     go_back

           mov.b   #h'01,r0l
           mov.b   r0l,@turn_flag      ;set the turn

           mov.b   @move_flag,r1h
           cmp.b   #h'01,r1h
           beq     go_back

           mov.b   @first_time_flag,r0l
           cmp.b   #h'00,r0l
           beq     first_time

more_time: mov.b   @p8dr,r0l
           and     #h'60,r0l
           cmp.b   #h'00,r0l
           beq     go_zig              ;Sensor_R/L (P8.6/5) is on?

```

```

        mov.b  #h'ff,r1l
        mov.b  r1l,@stop_flag      ;If on, stop the mop
        bra    go_back

go_zig:  mov.b  @turn_dir_flag,r0l
        not    r0l
        mov.b  r0l,@turn_dir_flag  ;swap turn flag
        bra    go_back

first_time: mov.b  #h'ff,r0l
        mov.b  r0l,@first_time_flag ;change the first time flag
        mov.b  @p8dr,r0l
        and    #h'20,r0l
        cmp.b  #h'00,r0l
        bne    go_back             ; Sensor_L (P8.5) is on?
        mov.b  #h'ff,r0l
        mov.b  r0l,@turn_dir_flag  ;set turn_direction-left flag

go_back:  mov.b  #h'08,r0h
        mov.b  r0h,@ier             ;enable IRQ3

        pop    r2
        pop    r1
        pop    r0

        rte

SELEC:
        push   r0
        push   r1
        push   r2

        mov.b  #h'02,r1h
        mov.b  r1h,@k_45            ;number of 45 degree (turn 90)

        mov.w  #h'c842,r0
        mov.w  r0,@count_45

        mov.b  #h'01,r1h
        mov.b  r1h,@move_flag

        mov.b  #h'00,r0l
        mov.b  r0l,@turn_dir_flag  ;set turn_direction-right flag

```

```
andc    #h'7f,ccr          ;unmask interrupts
```

```
pop     r2
pop     r1
pop     r0
```

```
rte
```

```
*****
;
**SUBROUTINES*
*****
;
```

```
*****
;*****Read the speeds of two motors*****
```

```
GET_SPEED:  mov.b @left_2016_hi,r0h    ;GET THE SPEED/PAST SPEED
            mov.b @left_2016_lo,r0l
            mov.w r0,@left_count      ;read left counts
            mov.b @right_2016_hi,r0h
            mov.b @right_2016_lo,r0l
            mov.w r0,@right_count     ;read right counts
```

```
left_calcu: mov.w @past_left_count,r0
            mov.w @left_count,r1
            sub.w r0,r1
            cmp.b #h'00,r1h
            blt  l_negative
            mov.w r1,@left_speed
            jmp  @right_calcu
```

```
l_negative: jsr  @COMPLE
            mov.w r1,@left_speed      ;get left speed
```

```
right_calcu: mov.w @past_right_count,r0
            mov.w @right_count,r1
            sub.w r0,r1
            cmp.b #h'00,r1h
            blt  r_negative
            mov.w r1,@right_speed
            jmp  @store
```

```
r_negative: jsr  @COMPLE
            mov.w r1,@right_speed     ;get right speed
```

```

store:    mov.w @left_count,r0
           mov.w r0,@past_left_count
           mov.w @right_count,r0
           mov.w r0,@past_right_count

           rts

```

```

;*****
;*****Get the duties for two motors*****

```

```

GET_DUTY:  mov.b @stop_flag,r0l
           cmp.b #h'ff,r0l
           bne  check_turn      ;stop flag is on (mop in the corner?)

```

```

stop_all:  jsr  @dcc_right
           jsr  @dcc_left      ;decelerate two motors to stop
           mov.w #h'0000,r0
           mov.w r0,@bias_l
           mov.w r0,@bias_r
           jmp  @control

```

```

check_turn: mov.b @turn_flag,r0l
            cmp.b #h'01,r0l
            beq  turn
            jmp  @go_straight

```

```

turn:      mov.w @count_45,r0
            mov.w @turn_count,r1
            cmp.w r0,r1      ;reach 45?
            blo  check_dir
            mov.b @number_turn,r0h
            inc  r0h
            mov.b r0h,@number_turn

            mov.b @k_45,r2h
            cmp.b r2h,r0h
            bge  stop_turn
            mov.w #h'0000,r0
            mov.w r0,@turn_count

```

```

check_dir  mov.b @turn_dir_flag,r0l
            cmp.b #h'ff,r0l
            bne  turn_right   ;turn left or turn right?

```

```

turn_left:  jsr  @dcc_left      ;turn off the left motor
             mov.w #turn_l,r0
             mov.w r0,@bias_l   ;load bias for left motor

             mov.w #h'0000,r0
             mov.w r0,@bias_r

             mov.w @right_speed,r0
             mov.w @turn_count,r1
             add.w r0,r1
             mov.w r1,@turn_count ;accumulate the turn counts
             jmp  @control

turn_right: jsr  @dcc_right     ;turn off right motor
             mov.w #turn_r,r0
             mov.w r0,@bias_r   ;load bias for right motor
             mov.w #h'0000,r0
             mov.w r0,@bias_l

             mov.w @left_speed,r0
             mov.w @turn_count,r1
             add.w r0,r1
             mov.w r1,@turn_count ;accumulate the turn counts
             jmp  @control

stop_turn:  mov.w #h'0000,r0
             mov.b r0h,@turn_flag ;clear turn flag
             mov.b r0h,@number_turn ;clear the number of 45 degree
             mov.w r0,@turn_count ;clear the turn counts

go_straight: mov.w #h'0000,r0
              mov.w r0,@bias_l
              mov.w r0,@bias_r

              jsr  @acc_left
              jsr  @acc_right

control:    mov.w @left_speed, r0
             mov.w @left_d_speed, r1
             sub.w r0,r1
             jsr  @SCALE
             mov.w r1,@left_error ;get the left speed error

```

```

mov.w @right_speed,r0
mov.w @right_d_speed,r1
sub.w r0,r1
jsr @SCALE
mov.w r1,@right_error ;get the right speed error

mov.w @right_speed,r0
mov.w @left_speed,r1
sub.w r0,r1 ;two motors velocity difference
mov.w @bias_l,r0
add.w r0,r1
mov.w @bias_r,r0
sub.w r0,r1
mov.w @int_error,r0
add.w r0,r1
mov.w r1,@int_error ;integral of the difference
;mov.b #int_k,r0l
;jsr @MULTIPLE
mov.b #scale_int,r0h
mov.b r0h,@divide ;load the scale factor
jsr @SCALE
mov.w r1,@integral ;get the error of the integral loop

mov.w @integral,r0
mov.w @left_error,r1
sub.w r0,r1 ;get the error for left motor
mov.b #scale_l,r0h
mov.b r0h,@divide ;load the scale factor
jsr @SCALE
;mov.b #prop_kl,r0l
;jsr @MULTIPLE
mov.w @past_left_duty,r0
add.w r0,r1
jsr @LIMIT_DUTY
mov.w r1,@left_duty ;get the duty for left motor
mov.w r1,@past_left_duty

mov.w @integral,r0
mov.w @right_error,r1
add.w r0,r1 ;get the error for right motor
mov.b #scale_r,r0h
mov.b r0h,@divide
jsr @SCALE
;mov.b #prop_kr,r0l
;jsr @MULTIPLE

```

```

mov.w @past_right_duty,r0
add.w r0,r1
jsr @LIMIT_DUTY
mov.w r1,@right_duty ;get the duty for left motor
mov.w r1,@past_right_duty

rts

```

```

;*****
;** Acc right motor*****

```

```

acc_right:  mov.w #desired_speed,r0
            mov.w @right_d_speed,r1
            cmp.w r0,r1
            bge acc_r_re
            mov.w #acc, r0
            add.w r0,r1
            mov.w r1,@right_d_speed ;accelearte the right motor

```

```

acc_r_re:   rts

```

```

;*****
;** Acc left motor*****

```

```

acc_left:   mov.w #desired_speed,r0
            mov.w @left_d_speed,r1
            cmp.w r0,r1
            bge acc_l_re
            mov.w #acc, r0
            add.w r0,r1
            mov.w r1,@left_d_speed ;accelearte the right motor

```

```

acc_l_re:   rts

```

```

;*****
;** Dcc right motor*****

```

```

dcc_right:  mov.w #h'0000,r0
            mov.w @right_d_speed,r1
            cmp.w r0,r1
            ble dcc_r_re
            mov.w #dcc, r0
            sub.w r0,r1
            mov.w r1,@right_d_speed ;dceclearte the right motor

```

```

dcc_r_re:   rts

```

```

;*****
;
;*** Dcc left motor*****

```

```

dcc_left:  mov.w #h'0000,r0
           mov.w @left_d_speed,r1
           cmp.w r0,r1
           ble  dcc_l_re
           mov.w #dcc, r0
           sub.w r0,r1
           mov.w r1,@left_d_speed    ;dcelear the left motor

```

```

dcc_l_re:  rts

```

```

;*****
;
;*****Shift a signed r1 (division)*****

```

```

SCALE:    cmp.b #00,r1h
           bge  shift
           jsr  @COMPLETE
           jsr  @shifting
           jsr  @COMPLETE

           bra  complete

```

```

shift:    jsr  @shifting

```

```

complete: rts

```

```

;*****
;
;*****shifting_right/divide r1 *****

```

```

shifting: mov.b @divide,r2h

```

```

shift_again: shlr r1h
              rotr r1l
              dec  r2h
              cmp.b #h'00, r2h
              bne  shift_again

              rts

```

```

;*****
;*****Limit r1*****

```

```

LIMIT_DUTY:  mov.w  #high_limit,r2 DUTY RATION
              cmp.w  r2,r1
              blt   compare_lo
              mov.w  #high_limit,r1
;check:      mov.w  #h'ffff,r0
;            mov.w  r0,@p6dr
              bra   finish

```

```

compare_lo:  mov.w  #low_limit,r2
              cmp.w  r2,r1
              bge   finish
              mov.w  #low_limit,r1

```

```

finish:      rts

```

```

;*****
;*****Complement of r1*****

```

```

COMPLE:      not   r1l
              not   r1h
              mov.w #h'0001, r2
              add.w r2,r1          ;2's complement of the value
              rts

```

```

;*****
;*****Multiplication of r0l and r1*****

```

```

MULTIPLE:    cmp.b  #h'00, r1h
              blt   multi_neg
              mulxu r0l,r1
              bra   ending
multi_neg:    jsr    @COMPLE          ;If<0, get the positive value
              mulxu r0l,r1
              jsr    @COMPLE          ;return to negative after multiple
ending:      rts

```

```

.end

```

### **THE CLAIMS**

- (1) **The new way of using the paper mop.**  
**This microcontroller based robot uses the magic mop which attached to its body for clearing the floor. The robot provides the contact pressure through the handle of the mop to the specificial ly made cleaning paper against the floor.**
  
- (2) **The “Robot Soft I “ is the programmed software for the microcontroller in the robot. It characterised by controlling the robot to move in an orderly designed movement showed in Fig 1 to 4.**
  
- (3) **The Robot Soft I make the robot do a 180 degree turn. When the sensor detect an obstacle or a wall at a short distance in front of the robot, the robot will turn right for 180 degree then move forward . As it meets a wall again, the robot will turn left for another 180 degree. The robot will continue to turn right and left at 180 degree alternatively until a room is completely cleaned. This is the most Important movement of the software.**
  
- (4) **The Robot Soft I can also make the robot do a 90 degree turn either left or right in specifc condition so that the robot can more around the Furniture.**