CHAPTER - 3

ELECTRICAL MOTORS

Three electrical motors are being used in TUCATU:

1. Main motor
2. Steering wheel motor
3. Ultrasonic sensor motor

3.1 Main Motor

Main motor is a wiper motor. This motor is produced as having 2-step speed. Since continuous speed control is aimed in this project one of these speeds is chosen. Electrical characteristics of the motor are:

- Working voltage : 12 V
- Current (unload) : 1,2 A
- Current (load) : 1,6 A
- Motor type : Permanent magnetic DC motor

3.1.1 Main Motor Drive Circuit

Two direction movement and continuous speed control is needed. It is a fact that the inspection of the direction and the speed will be done by computer. A specific H-Bridge is designed as a motor drive circuit. The stages of this design are given below.

H-Bridge is basically formed of four transistors as shown in Figure 3.1.

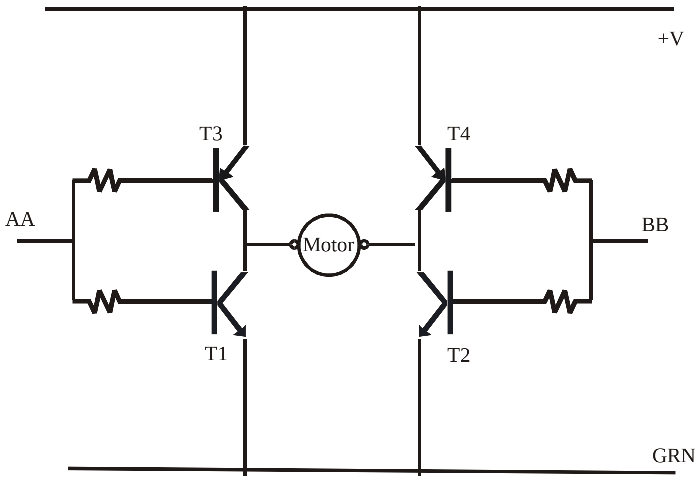


Figure 3.1: H-Bridge

Only one pair of the transistors is on saturation state at a time while other pair is at cut-off state. The motor connected to this pair of transistors rotates either on CW or CCW. For example, to keep the pair T3-T2 on and T4-T1 off, the point BB should be connected to +V and the point AA to the ground.

There is no possibility to directly connect the PIA ports to the points AA and BB. Since the outputs of PIA ports have capability of 5V and 10 mA sink/source current, some additional circuit is necessary.

The first stage of addition is voltage amplifier. The second one is current amplifier. In order to increase h_{fe} of power transistors, the Darlington method is used. The voltage amplifiers are added before AA and BB points. The final schematic of H-Bridge is given in Figure 3.2. Electrical specification of MA and MB points are suitable to PIA outputs.

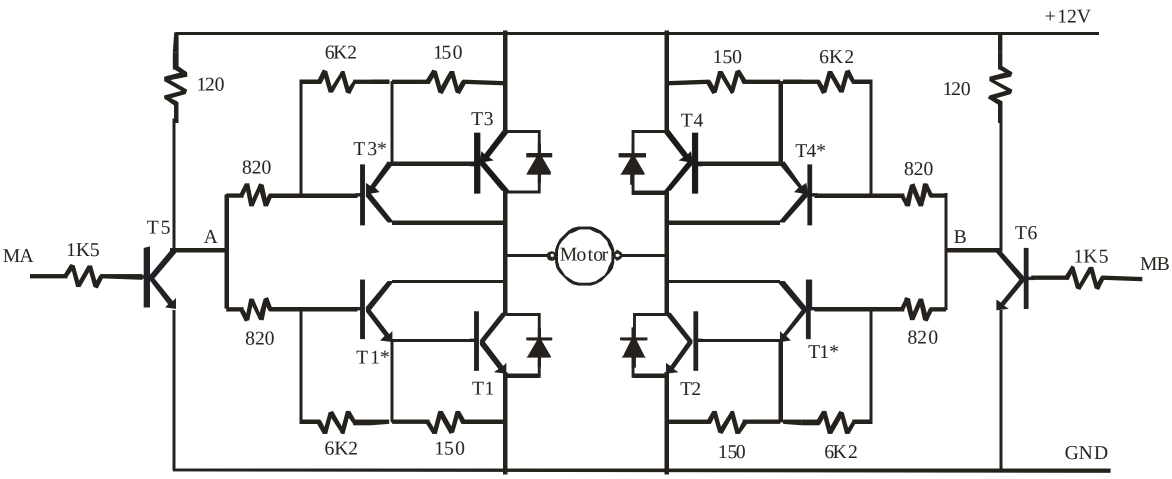


Figure 3.2: Final Design of H-Bridge

Diodes are used for protection of power transistors against a negative pick. The names of the components are listed below:

T1 – T2	: BD243
T3 – T4	: BD244
T1* - T2*	: BC547
T3* - T4*	: BC557
T5 – T6	: BC547

The value of the resistors is calculated according to the working condition of the circuit.

3.1.2 Main Motor Speed Control

The most efficient method for the speed control of the main motor is considered to be PWM. By the help of this method, power usage is minimized. The technique of PWM usage instead of using linear amplifiers is a more preferred method by means of power loss. Due to the fact that TUCATU has a limited power source (rechargeable battery), the voltage of motor control circuit is to be cut when the robot does not move. For this purpose, the circuit in Figure 3.3 is designed.

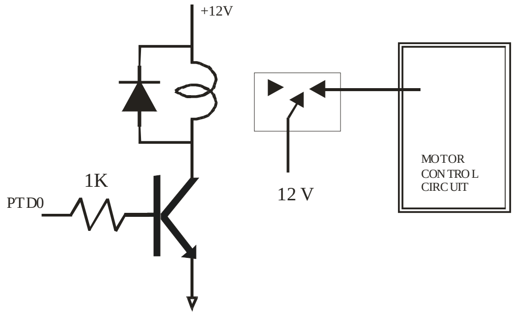


Figure 3.3: Motor Control Circuit

The connections between the motor control circuit and the micro controller are given in Table 3.1.

Table 3.1: Motor control connection

Connector J2	Motor Control Circuit	PTD
J2 (37)	MD	PTD0
J2 (35)	MB	PTD4
J2 (36)	MA	PTD3

It is important to control the speed of the robot while going forwards; that is why the speed control is done in the forward direction. The speed control in the backward direction is not considered.

The program about the motor speed control is given in the Chapter 9.2.2.

3.2 Steering Wheel Motor

A stepper motor is used for the motion of the steering wheel motor. This stepper motor is taken from a printer which is broken down. The method of the microcontroller to generate the signals required to drive the stepper motor is chosen. Thus, the hardware costs are minimized. The circuit designed is shown in Figure 3.4.

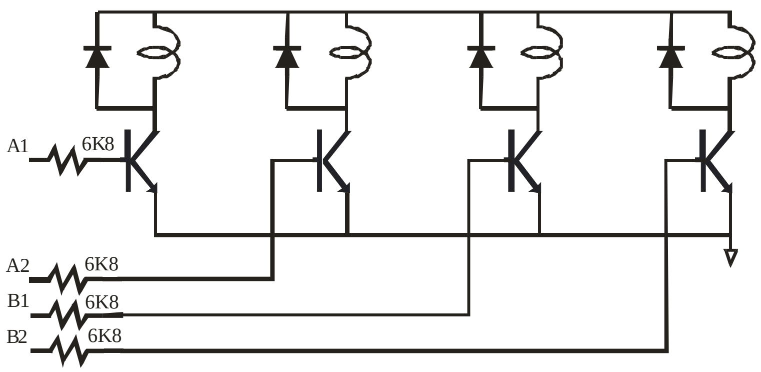


Figure 3.4**:** Step motor drive circuit

Instead of using discrete transistors and diodes, ULN2004 transistor array is used. The forms of the signals providing the movement of stepper motor are shown in Figure 3.5.

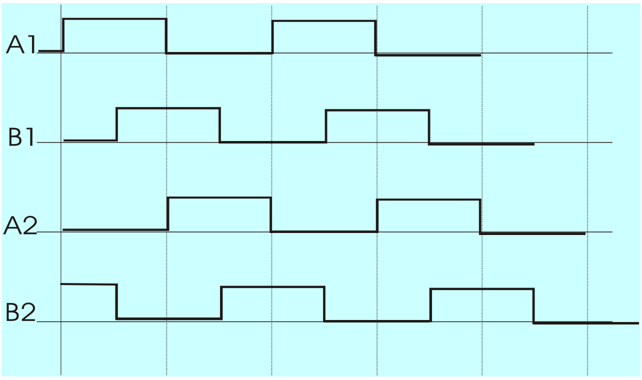


Figure 3.5: The form of stepper motor control signals

The program written in order to produce these signals is given in the Chapter 9.2.3

Step angle of the motor is 9 degree. As mentioned in Chapter 2, a gearbox is connected between steering wheel motor and front wheel. The reduction ratio of gearbox is 60. Therefore 2 steps are required for 1 degree steering angle.

The connections between the steering stepper motor and the microcontroller is given in Table 3.2.

Table 3.2 : The connections between the steering stepper motor and the micro controller

Connector	Steering Wheel Step Motor	Port
J2 (4)	RA1	PTA4
J2 (3)	RA2	PTA5
J2 (2)	RB1	PTA6
J2 (1)	RB2	PTA7

3.3 Ultrasonic Sensor Motor

The ultrasonic sensor motor is also a stepper motor and taken from an old hard disk drive. The hardware used is the same as the one used in the steering wheel motor. The signals providing the movement of the motor are again produced by software.

Step angle of the motor is 9 degree.

The connections between the sensor stepper motor and the microcontroller is given in Table 3.3.

Table 3.3: The connections between the sensor stepper motor and the micro controller

Connector	Steering Wheel Step Motor	Port
J2(7)	DA1	PTA0
J2(8)	DA2	PTA1
J2(6)	DB1	PTA2
J2(5)	DB2	PTA3

CHAPTER - 4

SENSORS

There are two sensors in TUCATU.

1. Path Measurement Sensor
2. Obstacle Sensor

4.1 Path Measurement Sensor

An original sensor is attached to the front wheel in order to find the distance travelled along the path forwards and backwards. The technical scheme of the sensor is shown in Figure 4.1.

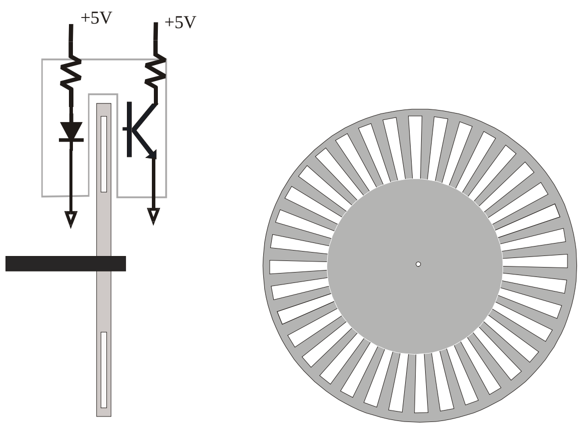


Figure 4.1: Path Measurement Sensor

A 3,5 inch floppy disk is used as the disc of the path measurement sensor. Sixteen holes are cut with the intervals of 22, 5 degree.

$$\text{resolution} = \frac{2 \times 11{,}25 \times 3{,}14}{16} = 4{,}41 \text{ cm}$$

As calculated above every one-interrupt shows 4,41 cm distance passed along the path. The information of the direction given to the main motor circuit decides whether the robot is going forwards or backwards. The software related to this sensor is given in the Chapter 9.2.4.

4.2 Obstacle Sensor

Ultrasonic receiver and sender are used as obstacle sensors. The ultrasonic receiver and sender are made resemble to radar and connected to sensor stepper motor as shown in Figure 4.2.

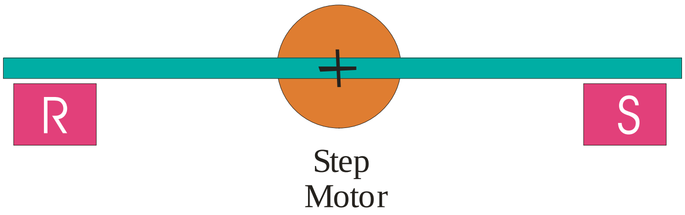


Figure 4.2: Obstacle Sensor

The aim of the obstacle sensor is to detect the obstacles in front and to measure the distance to the walls on the left and right sides. For these purposes, while moving, sensor system is at the 0 ° angle position to detect the obstacles in front of the robot. There are two methods to measure the distance by using ultrasonic receiver and sender:

1. Analog
2. Digital

In the analog method, the sender generates a signal at a certain frequency and amplitude. The signal on the output of the receiver is the signal reflected from the obstacle. The amplitude of this signal is related to the distance of the obstacle to the receiver.

In the digital method, the sender generates a signal at a certain frequency for a certain time. The time the signal generated is recorded. The delay is calculated when the receiver gets the reflected signal. The delay helps to calculate the distance of the obstacle to the sensor.

Digital method is chosen for this project. The required wave form is generated by microcontroller in terms of software. In order to amplify very low level signal of receiver two cascade amplifier have been used. The related circuit is shown in Figure 4.3.

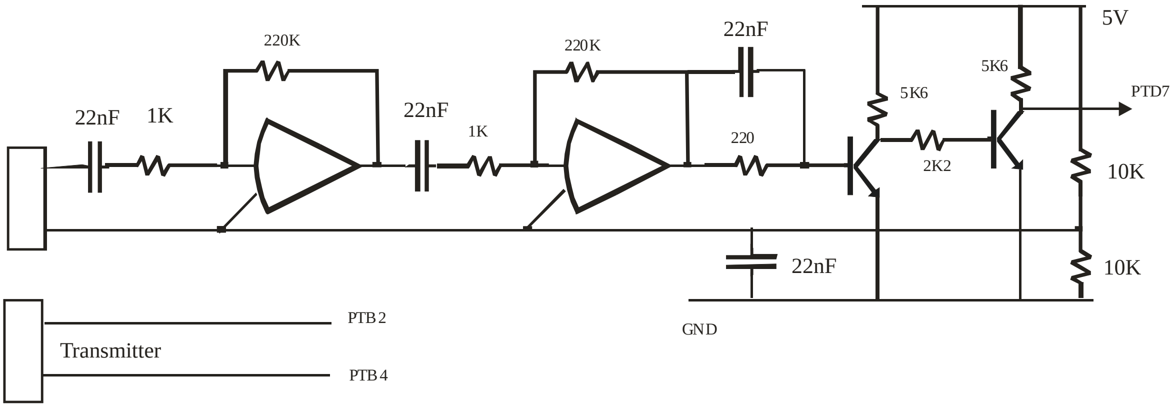


Figure 4.3: Ultrasonic receiver and transmitter circuit

Signal generation and detection software is given in Chapter 9.2.5.

CHAPTER - 5

LIGHT LEVEL MEASUREMENT

TUCATU has the capability of measuring the light level of environment. As a result of this measurement TUCATU decides whether or not the head light should be on.

In order to measure the light level a photo resistor is included. The circuit is so simple and given in Figure 5.1.

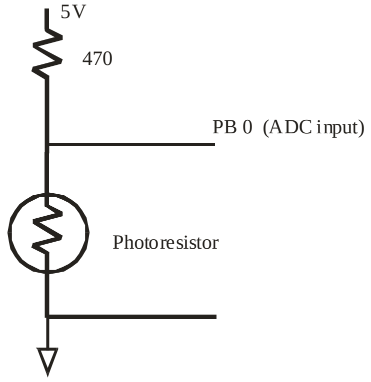


Figure 5.1: Light Level Measurement Circuit

As seen in Figure 5.1 the output of this light level measurement circuit is connected to ADC of MC6809. The related software is given in Chapter 9.2.7

If the light level is assumed to be low, TUCATU turns on the headlight. The headlight circuit is given in Figure 5.2

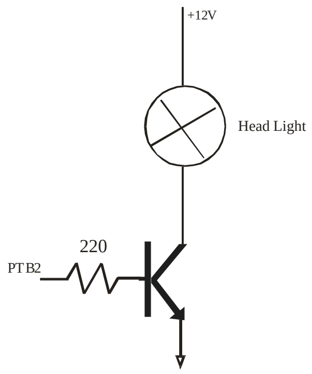


Figure 5.2: Head light control circuit

CHAPTER - 6

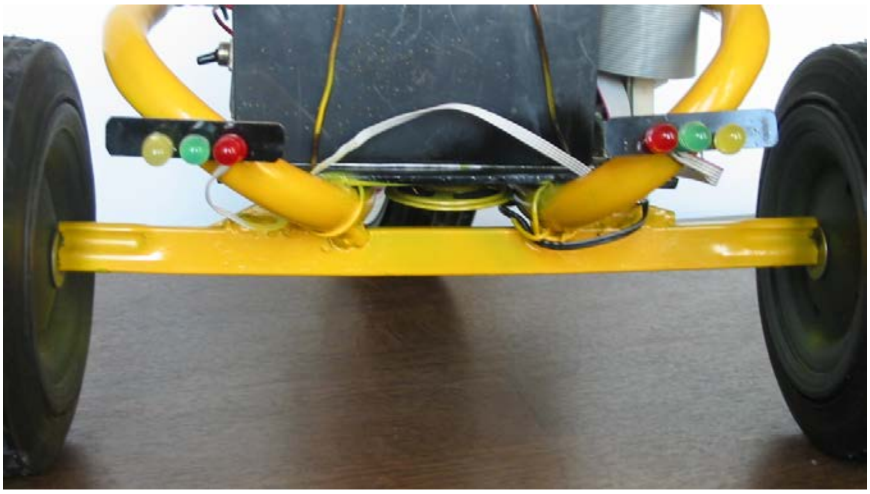
ALARMS AND SIGNALS

During the data entry from the remote control some indicators are necessary. These indicators confirm if the data is valid or not. Signal lights are also needed to inform the user about the action of TUCATU. Some warnings and alarms are needed in certain circumstances. Thus, a light and a voice alarm system have been installed.

The light signal system has been used for debugging purpose.

6.1 Signals

The signal light consisting of 3 LEDs each, are placed on the back right and back left sides of TUCATU to inform the people around while turning left and right. The circuit designed is given in Figure 6.1 and the picture is given in Picture 6.1



Picture 6.1: Signal system

Red **:** indicates stop condition

Green **:** indicates forward motion

Yellow **:** indicates turning direction

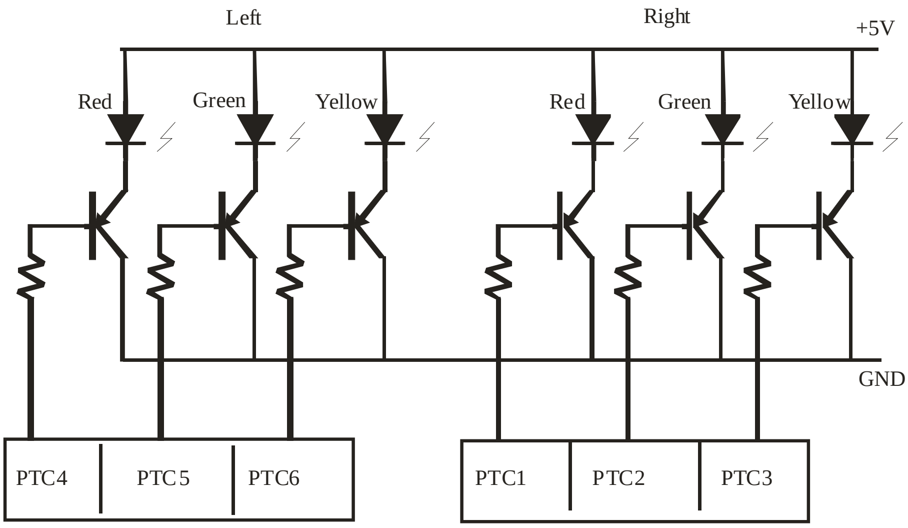


Figure 6.1: Light signal circuit

6.2 Voice

When TUCATU receives a key press from the IR remote control, she generates a short beep.

If the master makes a mistake during the data entry (e.g. missing parameters, wrong sequence) TUCATU generates "di da, di da, dit, dit" signal.

TUCATU also has the facility to alarm whenever she encounters an obstacle. The block diagram of the voice system is given in Figure 6.2.

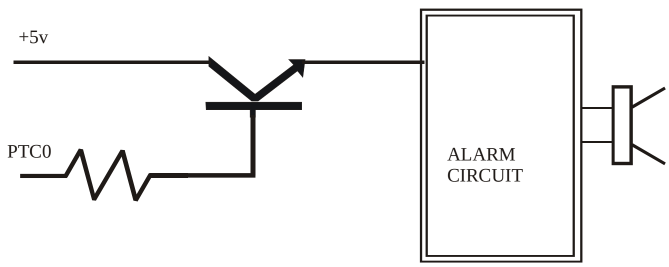


Figure 6.2: Voice Alarm System

The voice is controlled by the signals coming from PTC0 of MC6809.

CHAPTER - 7

REMOTE CONTROL AND TEACHING

It is planned TUCATU to have three working modes:

1. Free
2. Training / Teaching
3. Playback

In the free mode, the master moves TUCATU freely. The motion is not recorded.

In the training mode, a master teaches the required movements. Although a special “teaching” keyboard or a PC would have been used, in this project, it is preferred to use a TV remote control. This solution provides flexibility beside low cost. Each training action named “role” consists of 42 segments. Segment simply means one of the motion types and the speed information. TUCATU can store up to 96 roles one of each is 256 Byte.

In the playback mode, TUCATU repeats what she learnt. The master may choose one of the recorded programs from 0 to 9. For the time being, 10 roles are considered sufficient.

7.1 IR Remote Control

It is known that two international standards are being used with TV IR remote controls:

1. RC5
2. RECS 80

In RC5 standard, transitions between 1 and 0 determine the logical state whereas in RECS 80 standard duration of pulse is constant and space duration determines if it is logical 1 or 0. In Figure 7.1 RECS 80 format is shown.

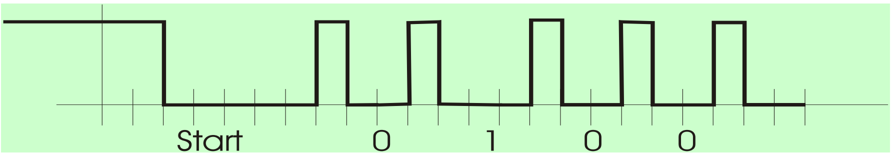


Figure 7.1: RECS 80 Format

7.2 IR Transmitter and Receiver

In this project, since the remote control used in Sony Systems is chosen, RECS 80 standard is used. This remote control can send 12 bits = 4096 different commands. For this application, 8-bit data is considered to be sufficient.TK19 IC optical sensor produced for this purpose is used to receive the signals the remote control sends. The output of TK19 is connected to the microcontroller for sending interrupts. One more interrupt input is needed since IRQ input is reserved for path measurement sensor. T1CH0 (PTD6) is initialized and used for this purpose.The signal to be decoded is connected to PTD1. The connection of TK19 to the microcontroller is shown in Figure 7.2.

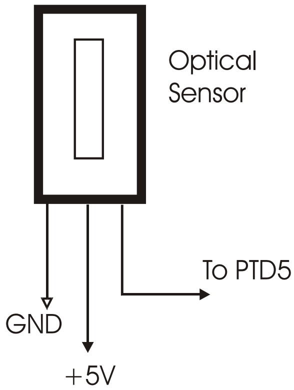


Figure 7.2: The Optical IC

7.3 Decoding of IR Remote Control Signals

The signal received must be decoded. The decoding process is implemented by a program. No additional hardware is used. Codes of the signals coming from the IR remote control are given in Table 7.1.

Table 7.1: Codes of IR control unit

Name of the key	Code of the key	First 8 bit in Hex	Function in TUCATU
0	1010 0001 0000	91	Number
1	0000 0001 0000	01	Number
2	1000 0001 0000	81	Number
3	0100 0001 0000	41	Number
4	1100 0001 0000	C1	Number
5	0010 0001 0000	21	Number
6	1010 0001 0000	A1	Number
7	0110 0001 0000	61	Number
8	1110 0001 0000	E1	Number
9	0001 0001 0000	11	Number
TLX	1111 1101 0000	FD	Destination
TV	0001 1101 0000	1D	Speed Motion
-/--	1011 1001 0000	B9	
i+	0101 1101 0000	5D	Training mode
↔↕	1011 1100 0100	BC	
OK	1010 0111 0000	A7	Flash
→	1010 0101 0000	A5	Speed Steering
Θ	1010 1001 0000	A9	Stop
Mute	0010 1001 0000	29	Step Number
Menu	0000 0111 0000	07	Menu
◄	0011 0011 1000	33	Left turn
►	0111 0011 1000	73	Right turn
▲	1011 0011 1000	B3	Forward
▼	1111 0011 1000	F3	Backward
Prog. Up	0000 1001 0000	09	
Prog Down	1000 1001 0000	89	
Volume Up	0100 1001 0000	49	Speed up
Volume Down	1100 1001 0000	C9	Speed down

The program written for decoding the signals coming from the IR remote control is given in the Chapter 9.2.6.

In the next chapter, all capabilities of TUCATU and how to use TUCATU will be explained in detail.

CHAPTER - 8

HOW TO USE TUCATU

When the power is turned on, the system starts in free mode. TUCATU is now can be directed via a remote control handset by her master. As said before, in free mode, all capabilities of TUCATU may be examined without recording the motion to the FLASH. Functions of the keys on the remote control are given in the previous chapter. Now, how to use these functions will be explained.

For better understanding picture of the remote control is given in Figure 8.1.

Figure 8.1: The Remote Control

The master may simply press the ▲ key and let TUCATU go forward at the minumum speed. She then may use **volume up** and **volume down** keys for speeding up and down TUCATU. Another way to determine the motion speed is to use **TV** key. The master should first press TV then a speed number between 1 and 9. If she presses a non-number key she will get an error message like “di da di da dit dit”. In this case, she must press TV key again, then a number and then ▲. TUCATU now goes forward at the required speed. Motion speed may be changed whenever the master wants even if TUCATU is moving. What worths to realize is that TUCATU does not immidiately reaches the given speed. She may gradually speed up as the conclusion of PWM usage. This provides comfort.

TUCATU stops going when the master presses the **Θ** button. She slows down and finally stops in seconds.

▼ key is for backward motion. TUCATU goes backward when after this key is pressed. There is no speed control on backwards.

On forward and backward motion a destination value can also be entered before the motion. To do this, 4 digit numbers is to be entered after **TLX** key is pressed. TUCATU, in this case, will go up to this destination and stop by self. If the master knows the distance TUCATU to go, this function may be necessary.

◄ and ► keys are used for left and right directions respectively. When one of these keys is pressed TUCATU begins turning to the required direction at the minumum steering speed. The master can enter steering speed with the key → and a number between 1 and 9. A non-number entrance will conclude in error. → , a number and ◄ or ► keys should be pressed respectively for steering speed change.

Another concept with right and left motion is the number of steps that the stepper motor will have. Like destination in forward and backward motion, number of steps may also be determined before TUCATU turns right or left. In order to enter the two digit value for number of steering steps the master should first press the **MUTE** key. TUCATU turns required number of steps and stops turning.

TUCATU not only goes forward and backward or turns right and left, she also may turn right or left while going forward or backward. Right forward, right backward, left forward, left backward motions includes the functions explained above, too.

Some easy usage of the remote control is thought and implemented. Below is given some combinations:

TUCATU turns **left** if ◄ key is pressed. She stops turning if

1- Θ key is pressed.
2- ◄ key is pressed.
3- ▲ key is pressed. She just goes forward.

TUCATU turns **right** if ► key is pressed. She stops turning if

1- Θ key is pressed.
2- ► key is pressed.
3- ▲ key is pressed. She just goes forward.

TUCATU goes **right forward** if ▲ then ► key is pressed. Then if,

1- ► key is pressed, she goes just forward.
2- ◄ key is pressed, she goes left forward.
3- ▲ key is pressed, she goes just forward.
4- ▼ key is pressed, she gradually slows down and stops,then goes just backward.

TUCATU goes **left forward** if ▲ then ◄ key is pressed. Then if,

1- ◄ key is pressed, she stops turning and goes just forward.
2- ► key is pressed, she goes right forward.
3- ▲ key is pressed, she stops turning and goes just forward.
4- ▼ key is pressed, she gradually slows down and stops,then goes just backward.

TUCATU goes **right backward** if ▼ then ► key is pressed. Then if,

1- ► key is pressed, she stops turning and goes just backward.
2- ◄ key is pressed, she goes left backward.
3- ▼ key is pressed, she goes just backward.
4- ▲ key is pressed, she stops and goes just forward.

TUCATU goes **left backward** if ▼ then ◄ key is pressed. Then if,

1- ◄ key is pressed, she stops turning and goes just backward.
2- ► key is pressed, she goes right backward.
3- ▼ key is pressed, she stops turning and goes just backward.
4- ▲ key is pressed, she stops and goes just forward.

Motion speed, steering speed, number of steps and destination values are applicable in those combinations.

i+ key is reserved for switching to the training / teaching mode. The master presses this key intending to record the following sequence of motion. All types of motions with their parameters like motion speed, steering speed, destination or number of steps are written to Flash after the master finishes and presses **OK** button. A number from 0 to 9 should also be entered immidiately after the OK key. This number specifies where the **role** is written in Flash. Although it seems only 10 roles (0-9) may be recorded to the Flash it is just because we chose to use only one digit number. 96 roles consisting 42 segments of 256 Byte would have taken place in Flash memory if 2 digit entries have been allowed.

In the playback mode, it is possible to select and run the programs taught before by the key **MENU** and the number (0 – 9) where the role was recorded into. In this mode, TUCATU stops and alarms if there is any obstacle in front of her. She continues her motion in case the obstacle disappears.

TUCATU turns her headlight on whether the environment is dark. She turns it off when there is light enough to see around.

CHAPTER - 9

SOFTWARE

The software is composed of a main program waiting for interrupts. Special functioned programs running under the management of Interrupt Service Routines (ISR) are called.

9.1 Operation Modes

The management program is implemented for three modes:

1. Free mode
2. Training mode
3. Playback mode

9.1.1 Free Mode

In this mode, master, examines all capabilities of TUCATU. He can test all motion types, speed up and speed down features. He also can examine data entry features of TUCATU such as a given step number for steering, speed value for forward motion etc.

TUCATU evaluates the signals coming from the remote control and starts doing the movements according to the instruction given:

1. Moving forward at a given speed
2. Moving backward at a predefined speed
3. Turning right and left while going forward
4. Turning right and left while going backward
5. Changing steering speed during right, left, right forward, left forward motion
6. Gradually slowing down
7. Stopping

No data is recorded at the end of the free mode.

9.1.2 Training Mode

In addition to the features in free mode, TUCATU measures and records how far she has gone in forward and backward directions and writes the needed values to the Flash in order to work in the playback mode.

9.1.3 Playback Mode

TUCATU moves according to the programs taught or loaded before. She generates a voice alarm in case there is an obstacle in front of her.

9.2 Programs

The program which is developed for this project has almost 2400 lines of assembly code and the size of the object code is about 9 KByte. In this section, only important parts of the programs and the flowcharts are given. The names of these programs are:

1. Main program
2. Motion motor program
3. Steering motor program
4. Path sensor program
5. Obstacle detection program
6. IR remote control decoding program
7. Light level measurement program
8. Data entry programs
9. Motion speed control program
10. Stop and end of segment programs
11. Flash erase and write programs
12. Training program

9.2.1 Main Program

The core of the software is the Main program. This program is the operating program of TUCATU. Mode selection and running of required program is organized by the main program. The flowchart of the main program is given in Figure 9.1

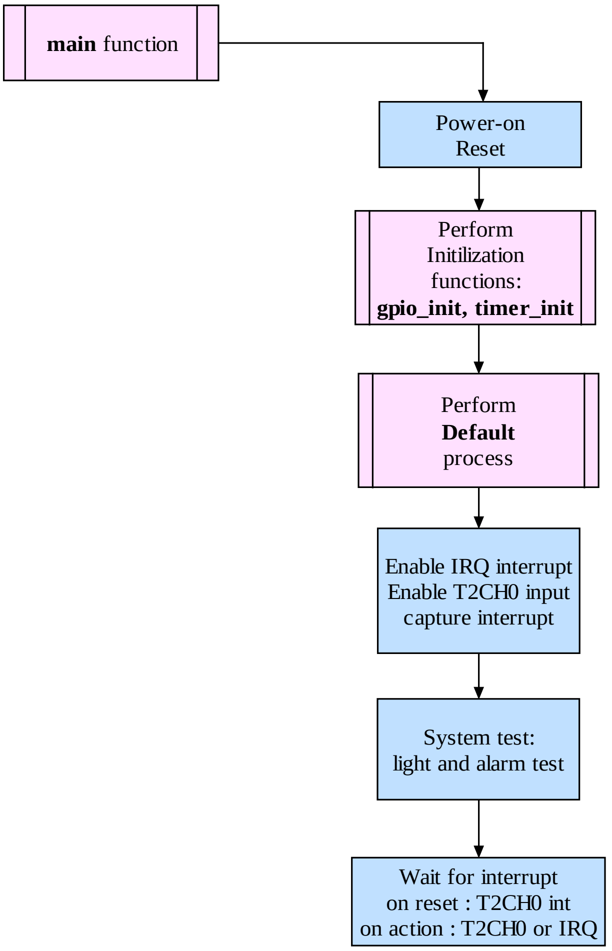


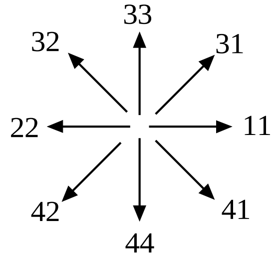
Figure 9.1: The flowchart of the Main program

The main program is given as follows:

```
* ------------------------------------------------------------- *
* Tucatu Main                                                    *
* ------------------------------------------------------------- *

Main:
            rsp                         ; stack pointer reset ($00FF)
            clra                        ; register init
            clrx
            sta   internal_error        ; clear internal errors counter
            mov   #$31,CONFIG1          ; MCU runs w/o LVI and COP support
            jsr   gpio_init             ; GPIO initialization
            jsr   timer_init            ; TIM initialization

            jsr   default               ; Default values

            lda   #$00                  ; Test
            sta   PTC
            lda   #$FE
            sta   PTB

            jsr   one_second
            mov   #$7F,PTC
            mov   #$00,PTB

            clr   mode                  ; mode=0
            clr   function              ; clear function code
            clr   segment               ; clear segment number

            lda   T2SC1                 ; T2SC1 is read
            lda   #$08                  ; %00001000
            sta   T2SC1                 ; T2SC1 CHOF flag cleared,interrupt-off

            lda   #$04
            sta   INTSCR                ; IRQ Interrup Enable

            lda   T2SC0
            mov   #$48,T2SC0            ; Timer Input Capture Interrupt Enable
            cli                         ; Enable all interrupt

bekle       bra   bekle                 ; Wait for interrupt
```

The second part of the main program is considered as a dispatcher. The flowchart of this part is given in Figure 9.2.

The variable motion_type has the information about the type of motion. It is zero if there is no action. Below is the motion types of the directions.



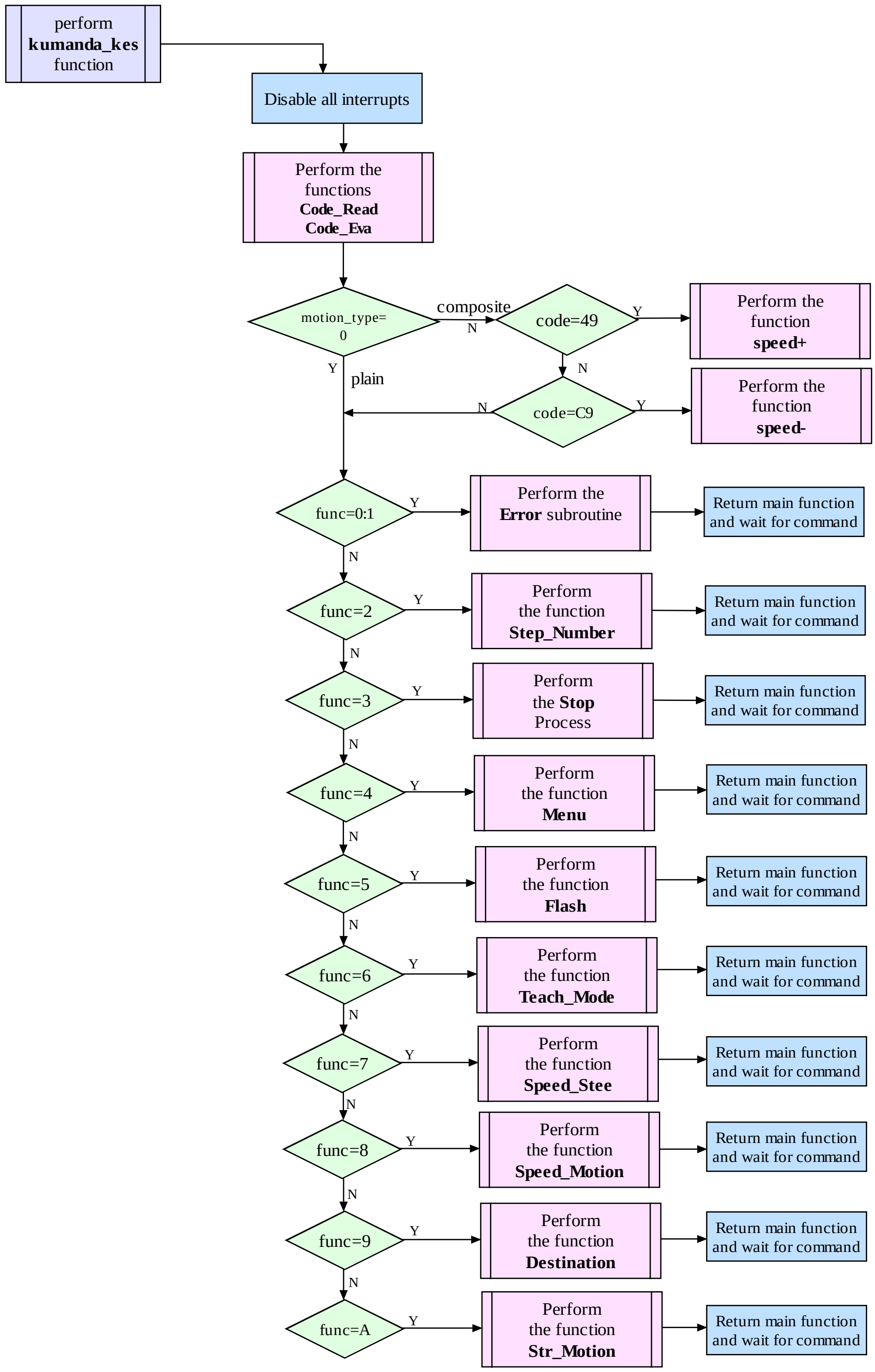


Figure 9.2: The flowchart of the dispatcher program

The source code of the program is as follows:

```
* ------------------------------------------------------------ *
* REMOTE CONTROL INTERRUPT                                     *
* ------------------------------------------------------------ *

Kumanda_Kes:
            sei                       ; Disable all interrupt
            pshh
            lda   T2SC0               ;
            mov   #$08,T2SC0          ; Disable timer ch=1 int and clear int flag
            jsr   code_read           ; Read code
            lda   motion_type
            beq   yali                ; Plain motion
            bra   composite
yali        jmp   yalin


* ------------------------------------------------------------ *
* Composite Motion                                             *
* ------------------------------------------------------------ *

speed+      lda   SPDMOTION
            cmp   #$6
            bge   sinir
            jsr   eof_segment
            lda   SPDMOTION
            inca
            sta   SPDMOTION
            sta   old_SPD
            ldhx  T1CH0H
            aix   #7F
            aix   #7F
            sthx  T1CH0H
            sthx  speed
hopa        jmp   rtf_int
sinir       mov   #$6,SPDMOTION
            mov   #$6,old_SPD
            bra   hopa

speed-      lda   SPDMOTION
            cmp   #$0
            ble   sinira
            jsr   eof_segment
            lda   SPDMOTION
            deca
            sta   SPDMOTION
            sta   old_SPD
            ldhx  T1CH0H
            aix   #-7F
            aix   #-7F
            sthx  T1CH0H
            sthx  speed
hoppa       jmp   rtf_int
sinira      mov   #$0,SPDMOTION
            bra   hoppa

durdur      jsr   stop
            jmp   rtf_int

composite
            mov   #$1,compos
            lda   code
            cmp   #$A9                ; If Stop key is pressed
            beq   durdur
            cmp   #$49
```

```
            beq   speed+              ; Speed up
            cmp   #$C9
            beq   speed-              ; Speed down

Duz_Yan     lda   code
            bra   yalin_2
            jmp   rtf_int

* -------------------------------------------------------------- *
* PLAIN MOTION - Parameters entry                                *
*               No connection with previous motion               *
* -------------------------------------------------------------- *

yalin       clr   compos
            lda   code
            sta   code_old
            clr   stop_flag

yalin_2     lda   function
            beq   hataya              ; unused code
            cmp   #$1
            beq   number              ; 0-9
            cmp   #$2
            beq   Step_num            ; Entry for step number; MUTE
            cmp   #$3
            beq   Sto                 ; Stop the action; STOP
            cmp   #$4
            beq   Men                 ; Jump to stored programs; MENU
            cmp   #$5
            beq   Flas                ; Store the last action; OK
            cmp   #$6
            beq   Teach_Mode          ; (i)
            cmp   #$7
            beq   Speed_Ste           ; Entry for steering speed;
            cmp   #$8
            beq   Speed_Motio         ; Entry for motion speed; TV
            cmp   #$9
            beq   Destinat            ; Four digit data entry for destination TLX

* -------------------------------------------------------------- *
* Plain Motion                                                   *
* -------------------------------------------------------------- *

            cmp   #$A
            beq   Str_motio           ; Straight motion
            cmp   #$B
            beq   Ste_motio           ; Direction Control Motion

* -------------------------------------------------------------- *
* Unknown code and function will be done                         *
* -------------------------------------------------------------- *

            bra   rtf_int

* -------------------------------------------------------------- *
* data entry error                                               *
* -------------------------------------------------------------- *

hataya      jsr   hata
            bra   rtf_int

Number      jsr   hata
            bra   rtf_int
```

```
Teach_Mode  ldhx  #$0100              ; RAM is to be cleared from 100 to 200

            clra
sil_        sta   ,x
            aix   #$1
            cmphx #$0200              ; 256 B 42 segments may be written
                                      ; between 100-200
            bne    sil_
            mov    #$00,segment
            mov    #$AA,mode          ; teach mode ->AA
            bra   rtf_int

Sto         jsr   Stop                ; Call Stop motion Subroutine
            bra   rtf_int

Str_Motio   jsr   Str_Motion          ; Call Straight Motion Subroutine
            bra   rtf_int

Ste_Motio   sta   code_old
            jsr   hata                ; ste motion cancelled
            bra   rtf_int

Speed_Ste   jsr   Speed_Stee          ; Call Steering Speed Read Subroutine
            bra   rtf_int

Speed_Motio jsr   Speed_Motion        ; Call Straight Motion SpeedRead Subroutine
            bra   rtf_int

Step_Num    jsr   Step_Number         ; Call Step number Read Subroutine
            bra   rtf_int

Men         jsr   Menu                ; Call Menu Subroutine
            bra   rtf_int

flas        jsr   flash               ; Call flash write subroutine
            bra   rtf_int

Destinat    jsr   destination         ; Call Destination data entry and convert
                                      ; Subroutine
            bra   rtf_int

rtf_int     lda   T2SC0
            mov   #$48,T2SC0
            pulh
            cli
            rti
```

9.2.2 Motion Motor Program

The main motor program consists of direction control , speed control by PWM and power control routines. Main motor program includes a decision part, forward and backward programs. The decision program also covers right and left motion. The flowcharts of these programs are given in Figure 9.3, Figure 9.4, and Figure 9.5 respectively.

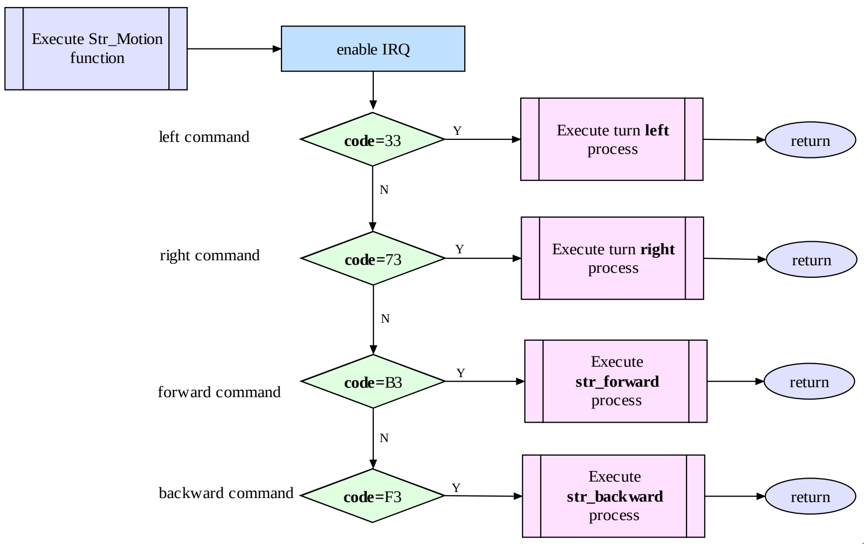


Figure 9.3: The flowchart of motion decision program

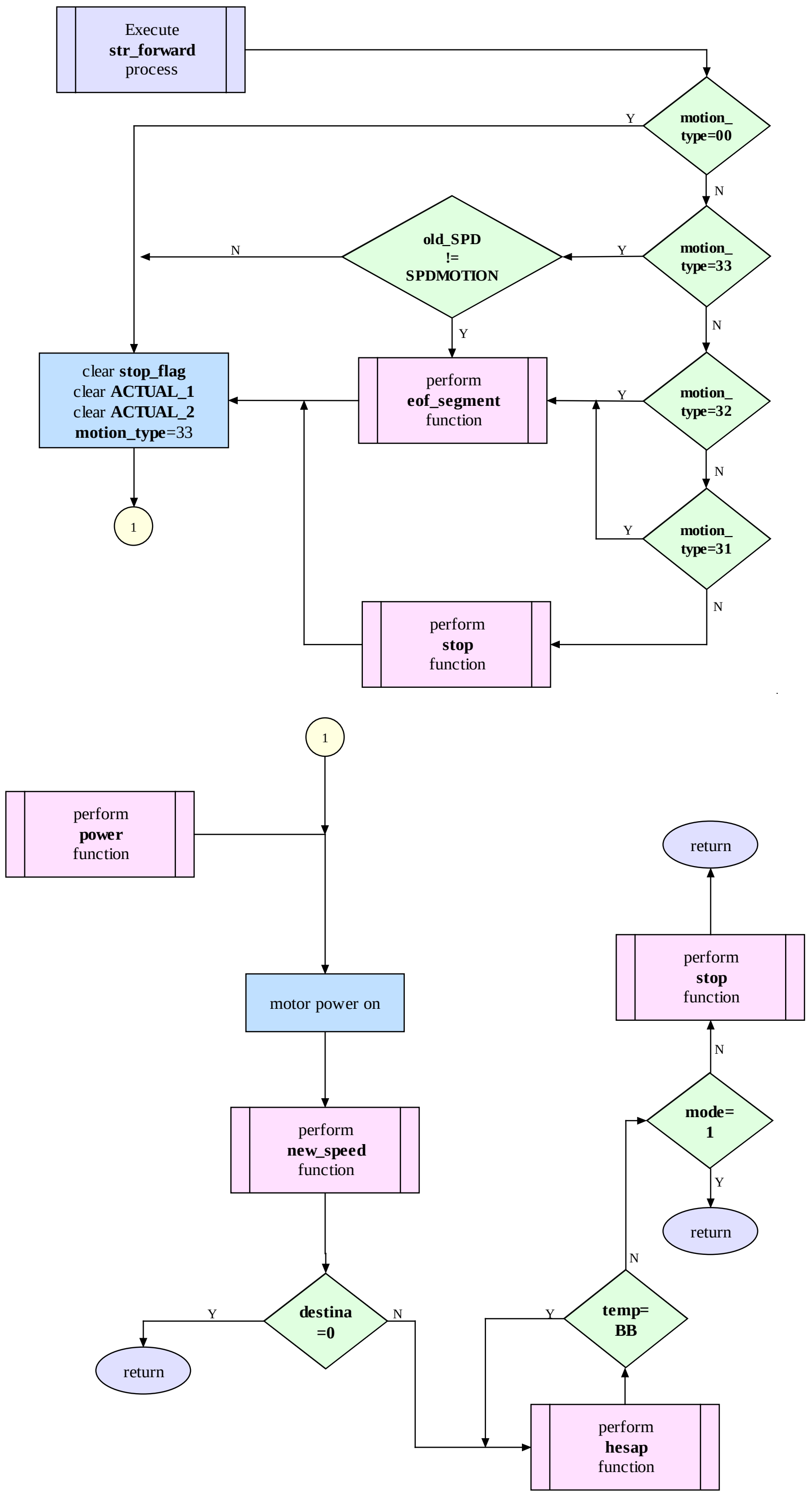


Figure 9.4 : The flowchart of straight forward motion control program

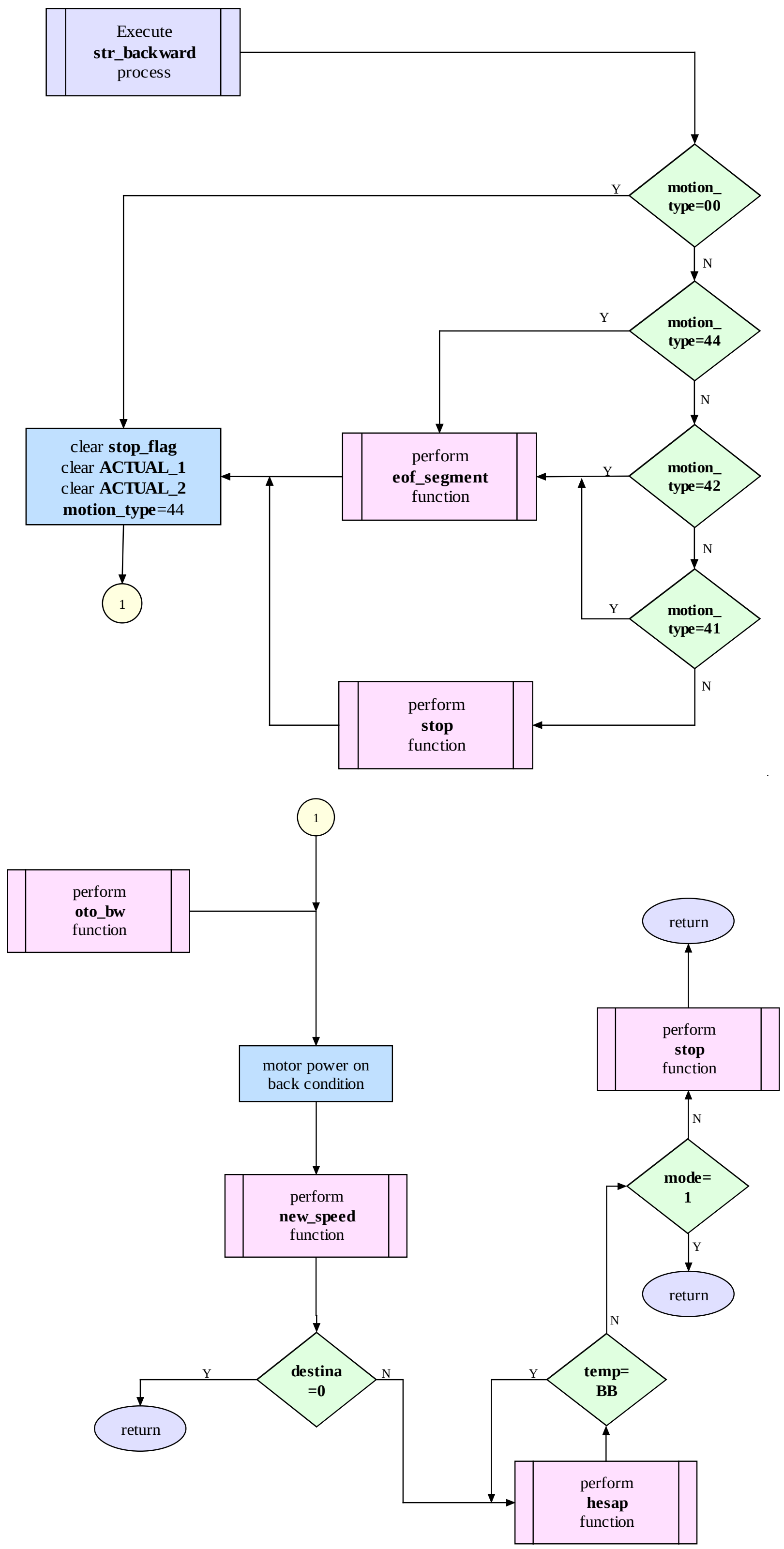


Figure 9.5: The flowchart of backward motion control program

The source code of the decision program is given below:

```
* ------------------------------------------------------------ *
* STR_MOTION - Straight Motion Control Subroutine              *
*             Tucatu moves forward or backward (B3, F3)        *
*             Turn steering right or left (73, 33)             *
*             Left forward B2, Right forward B4                *
*             left backward B1, Right backward B5              *
*             Steering turn right or left                      *
* ------------------------------------------------------------ *

Str_Motion:
            mov   #$5B,PTC
            mov   #$5B,BPTC          ; Green LEDs are on
            lda   #$04
            sta   INTSCR             ; IRQ Interrup Enable
            lda   code
            cmp   #$33
            beq   Lefte              ; Turn left, Code is $33
            cmp   #$73
            beq   Righte             ; Turn right, Code is $73
            cmp   #$B3
            beq   Str_forward        ; Str_forward, Code is $B3
            cmp   #$F3
            bra   Str_backward
            jsr   da
            jsr   da
            rts

Lefte       jmp   Left
righte      jmp   Right
```

The source codes of backward and forward motion program as well as related programs are given as below.

```
Str_backward:
            lda   motion_type
            beq   str_bw               ; motion_type 0 => start motion
            cmp   #$44                 ; motion_type 44 => rts
            beq   git_1
            cmp   #$42                 ; motion_type 42,41
                                       ; eof_segment, motion_type<-44, rts
            beq   str_bw_
            cmp   #$41
            beq   str_bw_
            jsr   stop                 ; motion_type 33,31,32,11,22 -> jsr stop
            clr   stop_flag
str_bw      mov   #$44,motion_type
            clr   ACTUAL_1             ; Reset actual
            clr   ACTUAL_2
oto_bw      lda   #$19                 ; D0, D3, D4 output,others input
            sta   DDRD
            lda   #$09                 ; Turn on power and back condition
            sta   PTD
            cli                        ; enable interrupt
            mov   #0,T1CH0H            ; PWM out = 0, mode= autonomous=>
                                       ; start motion. Come from Menu.
            mov   #0,T1CH0L            ; TUCATU moves back
            lda   destina
            beq   git_1                ; If destination value is not present
cont        bsr   hesap
            lda   temp
            cmp   #$BB
            beq   cont
```

```
            lda   mode                 ; does not stop if mode=autonomous
            cmp   #$1
            beq   git_1
            jsr   stop                 ; reach end of destination
            rts
git_1       rts
str_bw_     jsr  eof_segment
            bra  str_bw

* (Destination - actual) > 0 continue ; 16 bit compare

hesap       ldhx  dest_1
            cphx  ACTUAL_1
            blo   kucuk
            bhi   buyuk
esit        lda   #$00                 ; destination = actual
            sta   temp
            bra   durak
buyuk       lda   #$BB                 ; destination > actual
            sta   temp
            bra   durak
kucuk       lda   #$CC                 ; destination < actual
            sta   temp
durak       rts


Str_forward:
            lda   motion_type
            beq   str_2
            cmp   #$33
            beq   str__                ;motion_type      33 => rts
            cmp   #$32                 ;motion_type      32,31 => eof_segment
            beq   str_1
            cmp   #$31
            beq   str_1

            jsr   stop                 ;motion_type     11,22,44,41,42 =>stop
            motor by stop
            bra   str_2


str__
            lda   old_SPD
            cmp   SPDMOTION
            beq   bitti
            jsr   eof_segment

str_2       clr   ACTUAL_1             ; Reset actual
            clr   ACTUAL_2
            clr   stop_flag
            mov   #$33,motion_type
            lda   #$19
            sta   DDRD

power       lda   #1                   ; Motor Power on
            sta   PTD

speed_up    cli
            clrx
            clra
oto_fw      jsr   new_speed
```

```
*Motion motor reach to maximum speed

bitti       lda   destina           ; destination flag, 0 if there is no value
            beq   bitim


cont_1      bsr   hesap
            lda   temp
            cmp   #$BB
            beq   cont_1
            lda   mode
            cmp   #$1
            beq   bitim
            jsr   stop

bitim       mov   SPDMOTION,old_SPD
            rts

str_1       lda   old_SPD
            cmp   SPDMOTION
            bne   str__
            jsr   eof_segment
            bra   str_2
```

9.2.3 Steering Wheel Motor Program

Steering wheel motor is a stepper motor. This program is used for stepper motor control. The abilities of this program are:

- Direction control
- Number of steps
- Speed control

The flowchart of the steering wheel motor control program consists of two programs; Left and right. The flowcharts of these programs are given in Figure 9.6 and Figure 9.7.

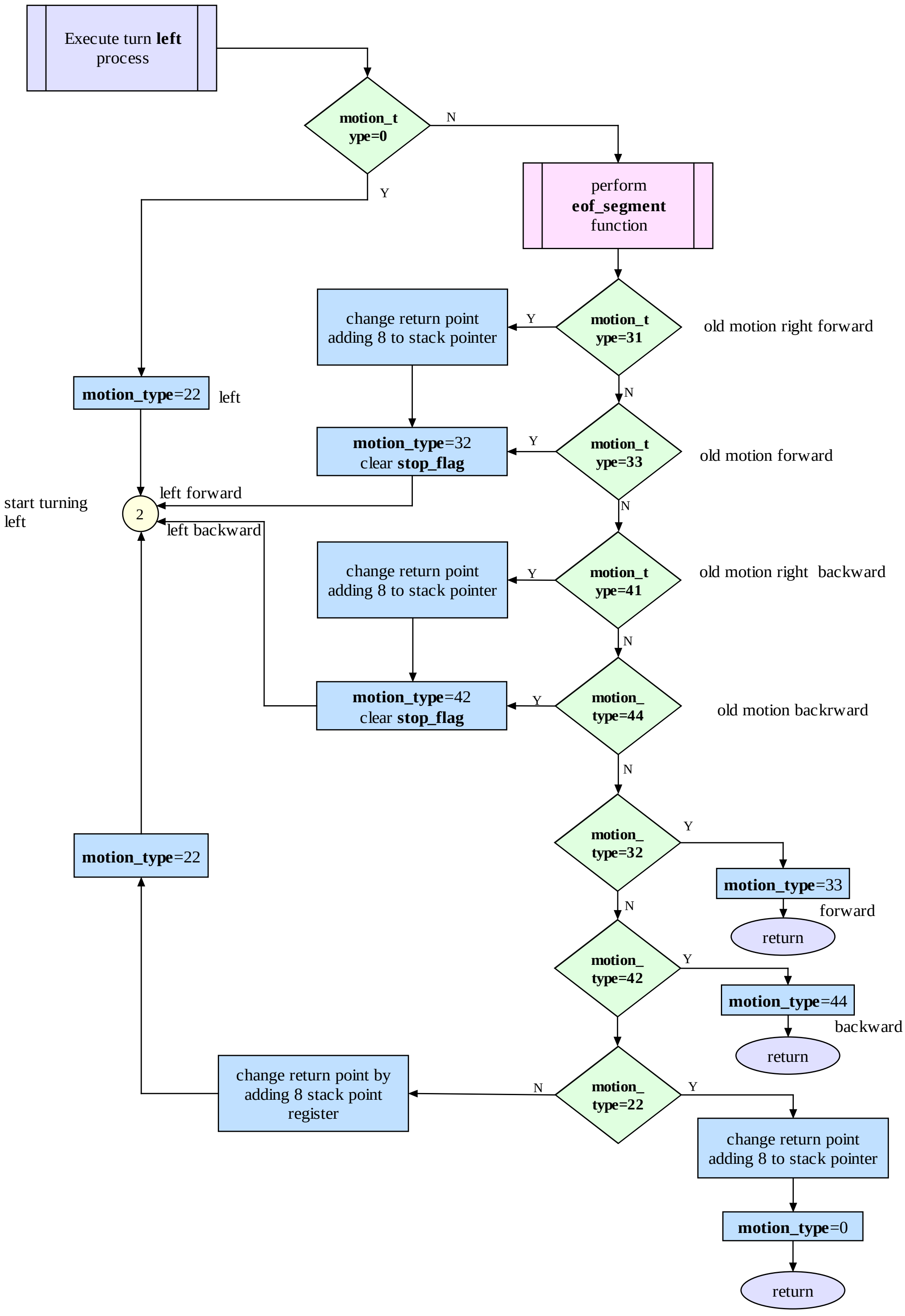


Figure 9.6-a : The flowchart of the steering wheel motor control program (left, part-1)

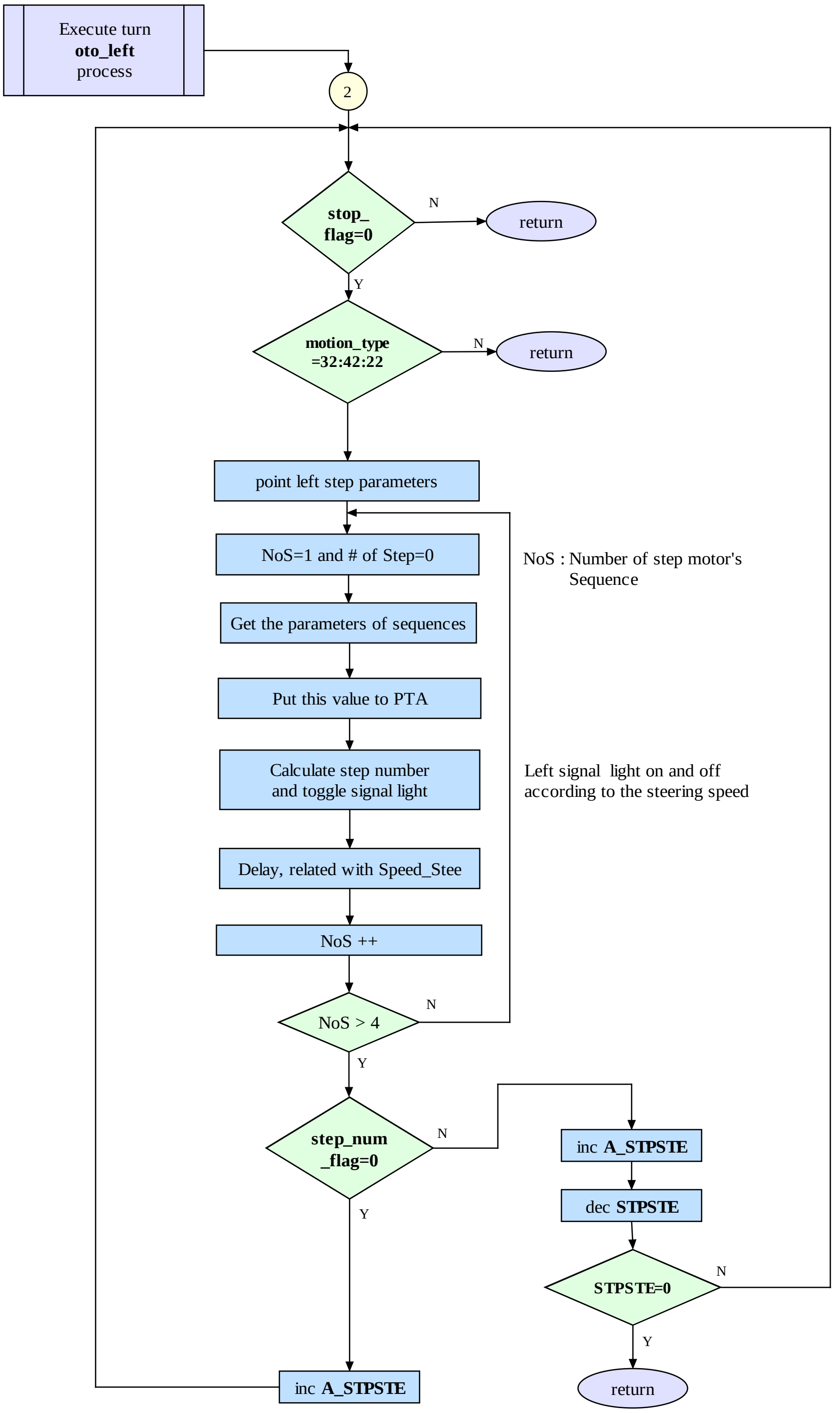


Figure 9.6-b : The flowchart of the steering wheel motor control program (left, part-2)

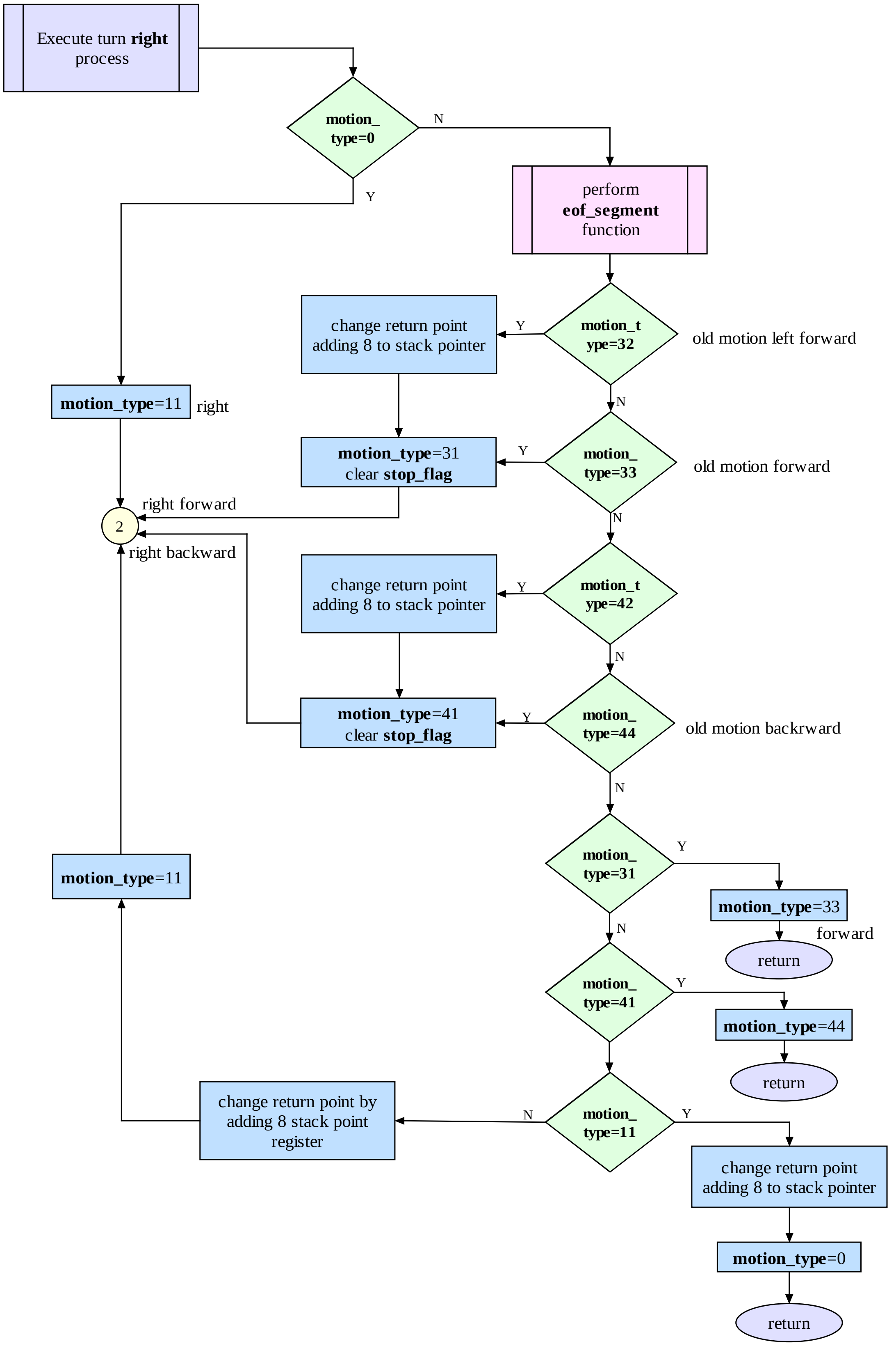


Figure 9.7-a : The flowchart of the steering wheel motor control program (right, part-1)

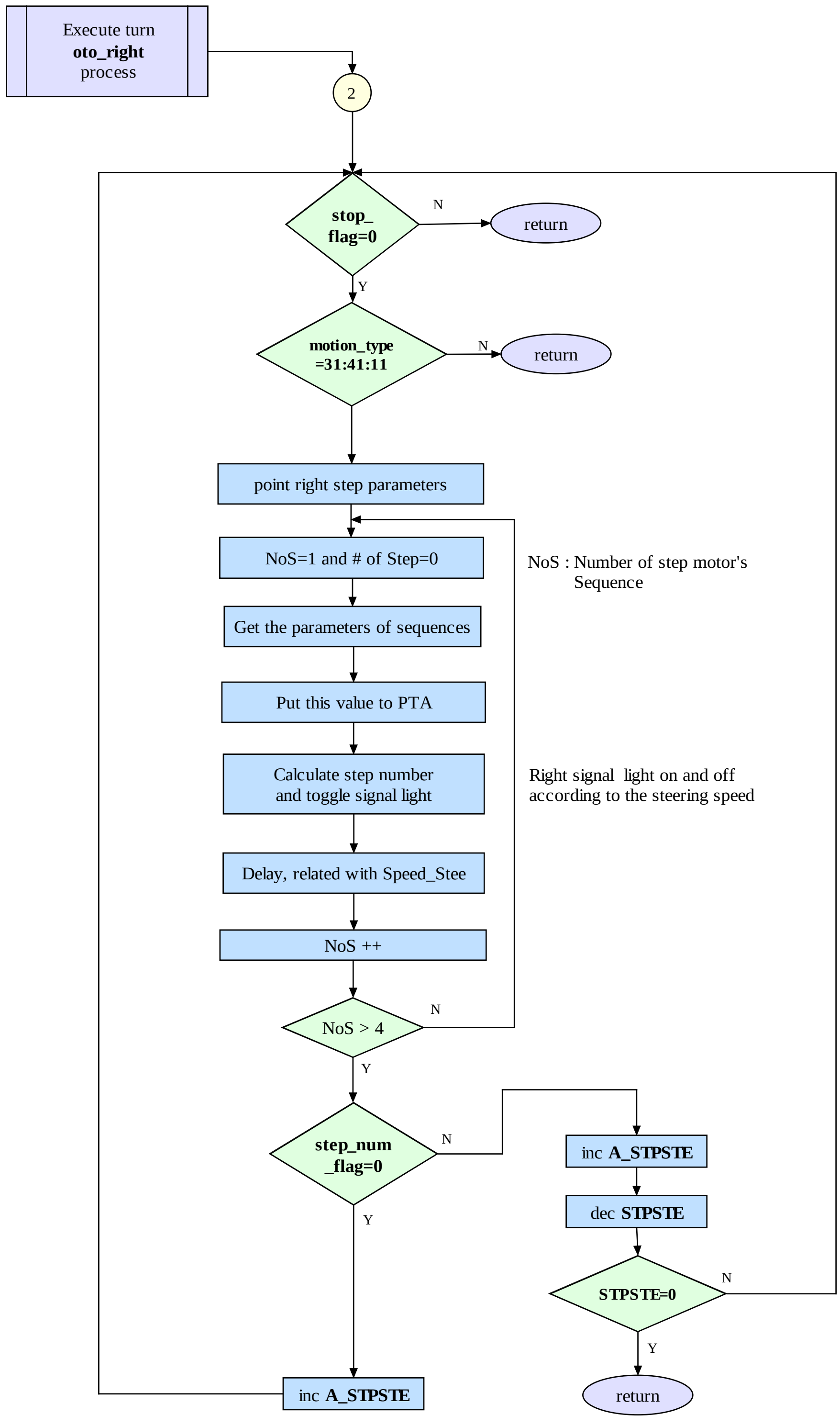


Figure 9.7-b : The flowchart of the steering wheel motor control program (right, part-2)

The source code of the right and left motion control programs and related programs are given below:

```
* ---------------------------------------------------------------- *
* Steering Control                                                  *
* ---------------------------------------------------------------- *

left:        lda   motion_type
             beq   left__
             jsr   eof_segment

             lda   motion_type
             cmp   #$33
             beq   fw_left_

             cmp   #$44
             beq   bw_left_

             cmp   #$32
             beq   _fw

             cmp   #$42
             beq   _bw

             cmp   #$31
             beq   fw_left

             cmp   #$41
             beq   bw_left

             cmp   #$22
             beq   left_dur

             ais   #$08
             bra   left__

fw_left      ais   #$08
fw_left_     mov   #$32,motion_type
             clr   stop_flag
             bra   left_
bw_left      ais   #$08
bw_left_     mov   #$42,motion_type
             clr   stop_flag
             bra   left_
_fw
             mov   #$33,motion_type
             bra   dur_ccw
_bw
             mov   #$44,motion_type
             bra   dur_ccw

left_dur     ais    #$08
             mov    #$00,motion_type
             bra   dur_ccw

left__
             mov   #$22,motion_type

left_        lda   #$19
             sta   DDRD
             cli
             lda   T2SC0
             mov   #$48,T2SC0          ; enable remote control interrupt
```

```
oto_left    lda   BPTA
            and   #$F0
            sta   BPTA
            lda   sinyal
            sta   temp
            bclr  6,PTC                ; left LED on
            mov   #$AA,temp_1          ; LED flag set
STE_CCW     ldhx  #STECCW              ; CCW signal forms start address
LOOP_1

            lda   stop_flag
            bne   dur_ccw
            lda   motion_type
            and   #$0F
            cmpa  #$02
            bne   dur_ccw
            dbnz  temp,ilerisi_c
            lda   temp_1
            cmp   #$AA
            beq   sondur_ccw
yak_ccw     lda   sinyal
            sta   temp
            bclr  6,PTC                ; left LED on
            mov   #$AA,temp_1          ; LED flag set
            bra   ilerisi_c
sondur_ccw  lda   sinyal
            sta   temp
            bset  6,PTC                ; left LED off
            mov   #$BB,temp_1          ; LED flag off
ilerisi_c   lda   BPTA                 ; Step motor drive port buffer
            and   #$0F
            ora   0,x
            sta   BPTA
            sta   PTA

            bsr   wait
            incx
            cpx   #$30
            blt   LOOP_1
            lda   step_num_flag        ; if step number = 0, do 1 step
            beq   single_ccw
            inc   A_STPSTE
            dbnz  STPSTE,STE_CCW       ; Go till STPSTE: Step number
            rts
single_ccw
            inc   A_STPSTE
            bra   STE_CCW

dur_ccw     rts

wait        clr   count+1
            lda   SPDSTEE              ; Step motor speed. Speed $00 ... $55
                                       ; Speed 0,1,2,3...8 (8 : 1f,  0 : 55)
            sta   count

m_wait      dbnz  count+1,m_wait
            dbnz  count,m_wait
            rts

h_second   clr    say+1
           clr    say+2
           mov    #2,say

hsecond    dbnz   say+2,hsecond
           dbnz   say+1,hsecond
           dbnz   say,hsecond
           rts
```

```
right:
            lda   motion_type
            beq   right__
            jsr   eof_segment

            lda   motion_type
            cmp   #$33
            beq   fw_right_
            cmp   #$44
            beq   bw_right_
            cmp   #$31
            beq   fw_
            cmp   #$41
            beq   bw_
            cmp   #$32
            beq   fw_right
            cmp   #$42
            beq   bw_right

            cmp   #$11
            beq   right_dur

            ais   #$08

            bra   right__

fw_right    ais   #$08
fw_right_   mov   #$31,motion_type
            clr   stop_flag
            bra   right_
bw_right    ais   #$08
bw_right_   mov   #$41,motion_type
            clr   stop_flag
            bra   right_
fw_         mov   #$33,motion_type
            bra   dur_cw
bw_         mov   #$44,motion_type
            bra   dur_cw
right_dur   ais   #$08
            mov   #$00,motion_type
            bra   dur_cw

right__     mov   #$11,motion_type

right_      lda   #$19
            sta   DDRD
            cli
            lda   T2SC0
            mov   #$48,T2SC0          ; enable remote control interrupt
oto_right   lda   BPTA
            and   #$F0
            sta   BPTA
            lda   sinyal              ; signal timing
            sta   temp
            bclr  3,PTC               ; right LED on
            mov   #$AA,temp_1         ; LED ON flag set
STE_CW      ldhx  #STECW              ; CW signal forms start address
LOOP_2
            lda   stop_flag
            bne   dur_cw
            lda   motion_type
            and   #$0F
            cmpa  #$01
            bne   dur_cw
            dbnz  temp,ilerisi
            lda   temp_1
            cmp   #$AA
            beq   sondur_cw
```

```
yak_cw      lda   sinyal
            sta   temp
            bclr  3,PTC             ; right LED on
            mov   #$AA,temp_1       ; LED flag on
            bra   ilerisi
sondur_cw   lda   sinyal
            sta   temp
            bset  3,PTC             ; right LED off
            mov   #$BB,temp_1       ; LED flag off
ilerisi     lda   BPTA              ; Step motor drive port buffer
            and   #$0F
            ora   0,x
            sta   BPTA
            sta   PTA
            jsr   wait
            incx
            cpx   #$2C
            blt   LOOP_2
            lda   step_num_flag     ; if step number flag = 0, do 1 step
            beq   single_cw
            inc   A_STPSTE
            dbnz  STPSTE,STE_CW     ; STPSTE: Step number
            rts

single_cw
            inc   A_STPSTE
            bra   STE_CW

dur_cw      rts
```

9.2.4 Path Measurement Sensor Program

The value of the path travelled is calculated by the number of interrupts coming from the path measurement sensor. IRQ input is used for this part of the application.

The path measurement counter is cleared when an action starts.

```
* -------------------------------------------------------------- *
* YOL_KES  - Path measurement interrupt routine                  *
* -------------------------------------------------------------- *

YOL_KES:    sei
            pshh
            ldhx  ACTUAL_1
            aix   #$1
            sthx  ACTUAL_1
            lda   mode
            cbeqa #$1,attla_adc

return      pulh
            cli
            rti
```

9.2.5 Obstacle Detection Program

The flowchart of the obstacle detection program is given in Figure 9.8.

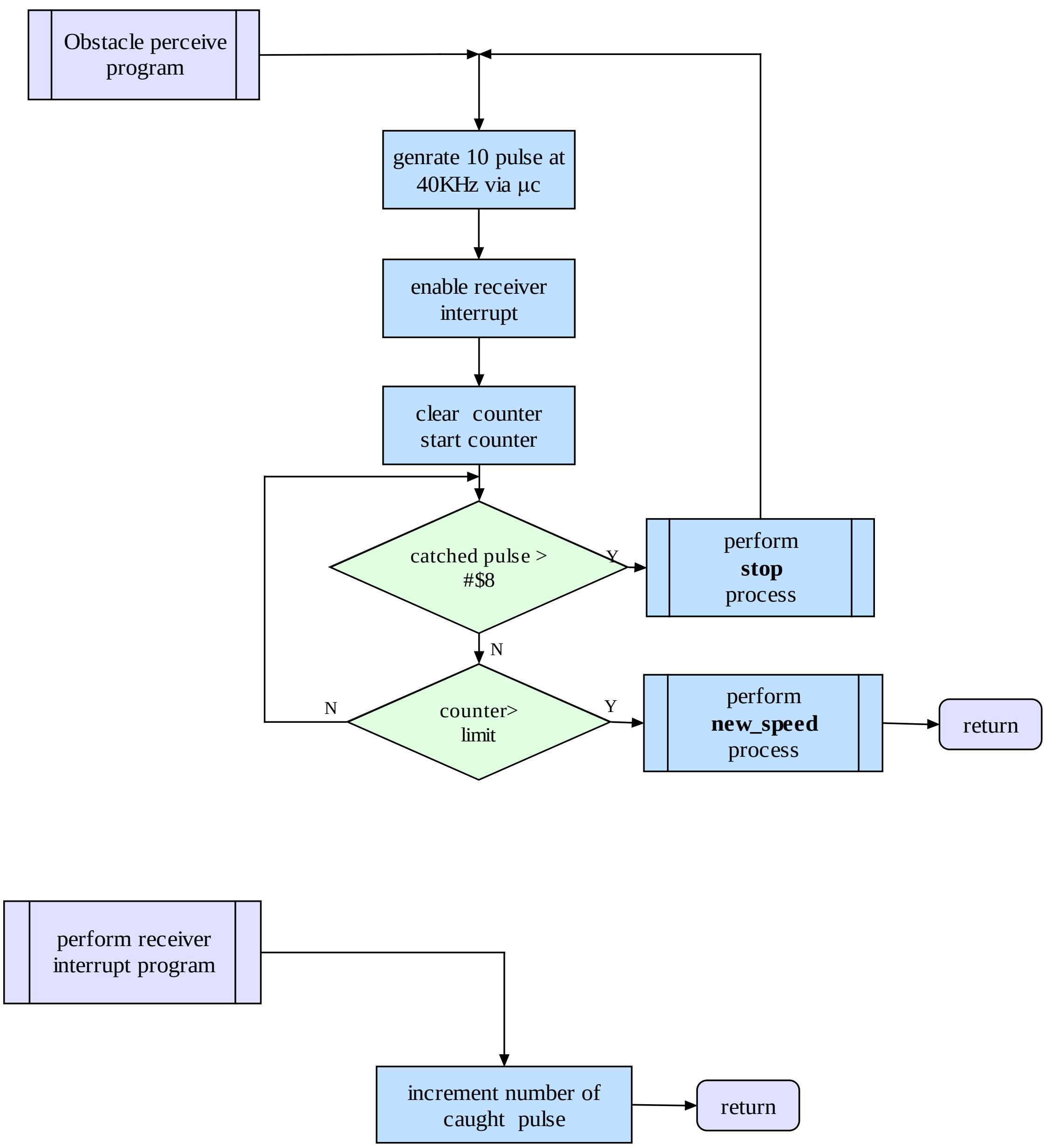


Figure 9.8: The flowchart of the program

The source code of the program is given below:

```
ult_sen
            lda     motion_type
            and     #$F0
            cmp     #$30
            bne     return
            ldx     #$14
            bset    2,PTB   ;set    PTB2
            bclr    4,PTB   ;clear  PTB4

_40KHz      lda     #$5
            dbnza   *
            nop
            lda    PTB
            eor    #$14
            sta    PTB
            decx
            bne    _40KHz
            lda    T2SC1
            mov    #$48,T2SC1         ;enable   intterrupt for ultrasonic sensor
            cli                       ;enable interrupts

wait_mod:
            ldhx   #$061A             ; 25000 cycle ~0,01 sn (3,4/2=1,7m
                                      ; sensor range)
back_con    lda    ult_con
            cmp    #$08               ; catch 8 pulse
            bge    block_
            aix    #-1
            cphx   #$00
            bne    back_con
            lda    T2SC1              ; disable  interrupt(Timer2 channel 1)
            mov    #$08,T2SC1
            clr    ult_con
            clr    ult_flag
            sei
            bset   0,PTD
            jsr    new_speed
            mov    #$1,mode
            bra    return
block_      sei
            lda    T2SC1
            mov    #$08,T2SC1
            lda    motion_type
            psha                      ;save motion_type
            lda    SPDMOTION
            psha
            mov    #$FF,ult_flag      ;set ult_flag if there is a block
            mov    #$BB,mode          ;IRQ on
            jsr    stop
            clr    old_SPD
            mov    #$1,SPDMOTION
            clr    stop_flag
            pula
            sta    SPDMOTION
            pula
            sta    motion_type
            clr    ult_con
            clr    ult_flag
            bra    ult_sen
```

9.2.6 IR Remote Control Decoding Program

The IR remote control signal is received by a receiver circuit. The output of this circuit is on TTL level. The output of this device is connected to two points: PTD1 and PTD6. PTD6 generates an interrupt and PTD1 reads this signal. The flowchart of the program is given in Figure 9.9.

The source code of the IR decoding program is given bellow.

```
* -------------------------------------------------------------- *
* CODE_READ    - Code_Read is designed as a subroutine           *
*                Code Value is stored in "CODE"                  *
*                Code_read works with Code_Eva                   *
*                Code_Eva return Function                        *
* -------------------------------------------------------------- *

Code_Read:

poll        clr   code
            clr   shift
            lda   PTD
            and   #2
            beq   poll                ; Wait for remote signal
on_1        clrx
st_p        lda   PTD
            and   #2
            beq   start
            mov   #3,count
j_2         dbnz  count,j_2
            cpx   #$FF
            beq   st_p
            incx
            bra   st_p
start       cpx   #$44                ; time spent in logical 1
            blo   poll
            clrx
back_1      lda   PTD
            and   #2
            bne   to_1
            mov   #3,count
j_3         dbnz  count,j_3
            incx
            cpx   #$B2                ; upperlimit of start bit: AC+5
            bhi   poll
            bra   back_1
to_1        cpx   #$A7                ; lowerlimit of start bit: AC-5
            blo   poll
```

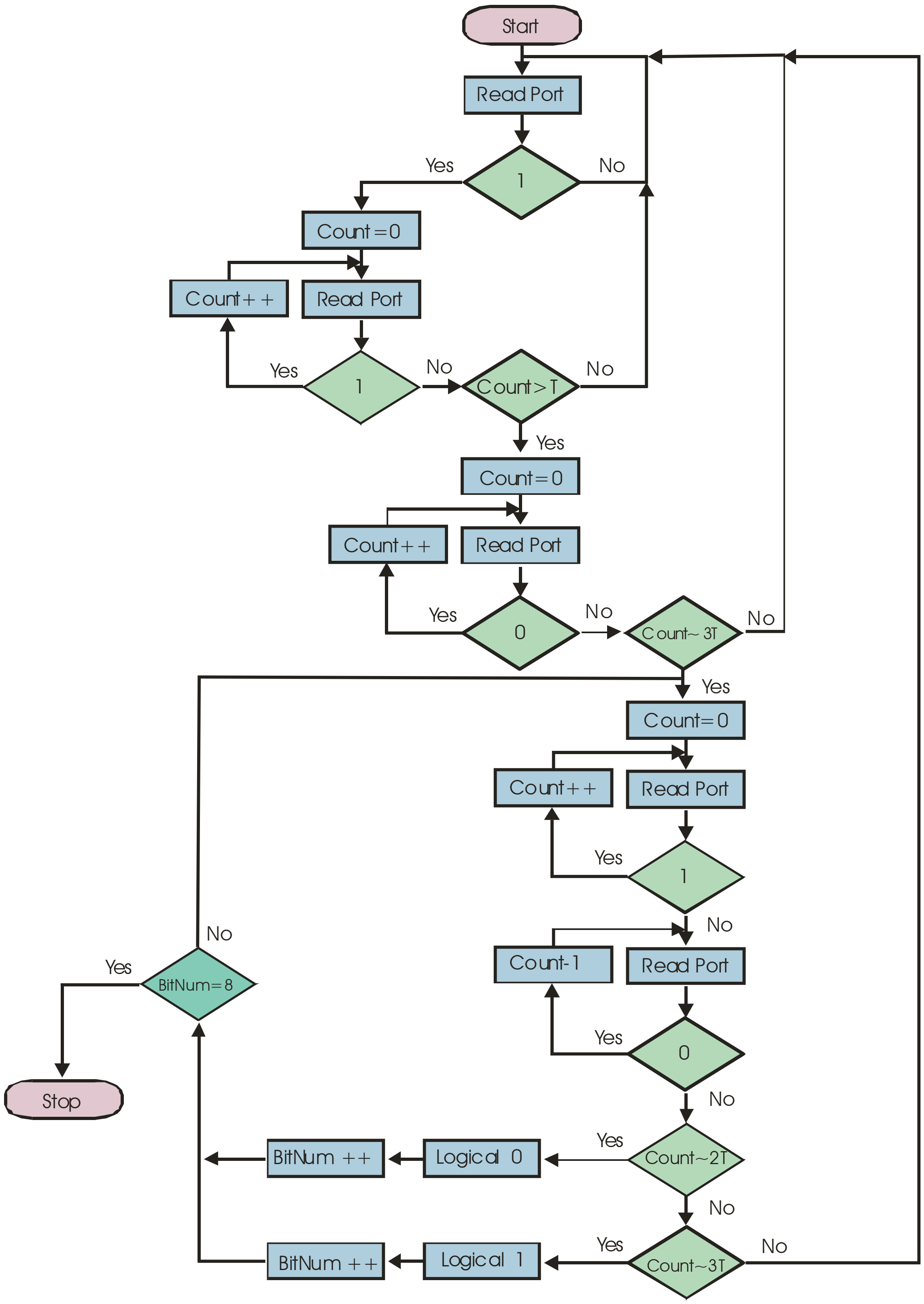


Figure 9. 9: The flowchart of the IR Code Decoder program

```
BASLA       clr   code
            clr   shift
back        clrx
back_2      lda   PTD
            and   #2
            beq   to_0
            mov   #3,count
j_4         dbnz  count,j_4
            incx
            cpx   #$2A             ; upperlimit of the time spent in 1: 24+5
            bhi   poll
            bra   back_2

to_0        cpx   #$1D             ; lowerlimit of the time spent in 1: 24-5
            blo   poll

back_3      lda   PTD
            and   #2
            bne   next
            mov   #3,count
j_5         dbnz  count,j_5
            incx
            cpx   #$82             ; upperlimit of 2T : 7C+5
            bhi   poll
            bra   back_3

next        cpx   #$4C             ; lowerlimit of T
            blo   poll

            cpx   #$57             ; upperlimit of T
            blo   zero
            cpx   #$77             ; (7C-5) ile (52+5) arası
            blo   poll

one         sec
            bra   jump
zero        clc
jump        rol   code             ; Remote control code
            ldx   shift
            incx
            stx   shift
            cpx   #$08
            blt   back

pol         lda   PTD
            and   #2
            beq   pol              ; logical 1?

on_12       ldhx  #$3000
st_p2       lda   PTD
            and   #2
            beq   pol
            aix   #-1
            cphx  #$0000
            beq   code_eva
            bra   st_p2
```

```
* ------------------------------------------------------------- *
* CODE_EVA - Remote Control Code Evaluation Routine             *
*            After Remote Code Reader Routine                   *
*            Code is in "CODE"                                  *
*            The evaluation of the code is in "function"        *
* ------------------------------------------------------------- *

code_eva    jsr   di                   ; Key press sound

            mov   #1,function          ; function=1
            lda   code                 ; Remote control code
            tax                        ; copy of acc
            and   #$0F                 ; filtering
            cmp   #$01                 ; Number codes are $91, 01, 81, 41, C1
            beq   donus                ; Number            21, A1, 61, E1, 11

            inc   function             ; function=2
            txa                        ; refresh acc
            cmp   #$29
            beq   Donus                ; Step Number, Code is $29

            inc   function             ; function=3
            txa                        ; refresh acc
            cmp   #$A9                 ; Stop
            beq   Donus                ; Stop, Code is $A9

            inc   function             ; function=4
            txa                        ; refresh acc
            cmp   #$07                 ; Go to stored programs
            beq   donus                ; Menu, Code is $07

            inc   function             ; function=5
            txa
            cmp   #$A7                 ; Write to Flash
            beq   donus                ; Flash, Code is $A7

            inc   function             ; function=6
            txa
            cmp   #$5D                 ; Switch to teaching mode
            beq   donus                ; Teach_Mode, Code is $5D

            inc   function             ; function=7
            txa
            cmp   #$A5                 ; Speed of Steering motor
            beq   donus                ; Speed_Ste, Code is $A5

            inc   function             ; function=8
            txa
            cmp   #$1D                 ; Speed of motion
            beq   donus                ; Speed_Motion, Code is $1D

            inc   function             ; function=9
            txa
            cmp   #$FD                 ; Destination
            beq   donus                ; Destination, Code is $FD

            inc   function             ; function=10
            txa                        ;
            and   #$0F
            cmp   #$03                 ; straight motion             (+)
            beq   donus                ; Str_motion, Codes are $33, $73, $B3, $F3

            inc   function             ; function=11
            txa
            cmp   #$09
            beq   donus                ; Steering and motion control
            txa
            cmp   #$49
```

```
            beq   donus              ; Ste_motion, Codes are $09, $89, $49, $C9
            txa
            cmp   #$89
            beq   donus
            txa
            cmp   #$C9
            beq   donus

            mov   #0,function        ; Unused code
donus       rts
```

9.2.7 Light Level Measurement

The source code of the program is given below:

```
con_adc     clr   adc_step
            lda   #$00
            sta   ADSCR              ; ADSCR : adc int disable; single
                                     ; conversion; 0 for PTB0
read_back   lda   ADSCR              ; check ADSCR until CoCo bit is set
            and   #$80
            beq   read_back          ; read_back loop
            lda   ADR                ; conversion result in ADR
            cmp   #$AA
            bhi   light
            lda   #$00
            sta   PTB
            bra   ult_sen
light       lda   #$FF
            sta   PTB
```

9.2.8 Data Entry Programs

There are four data entry programs, named:

- Destination
- Step number
- Motion speed
- Steering speed

The details of these programs are given in this section.

9.2.8.1 Destination Value Entry

This program reads four digit value then converts this value into two digit hexadecimal number as the destination value. The flow chart of this program is given in Figure 9.10.

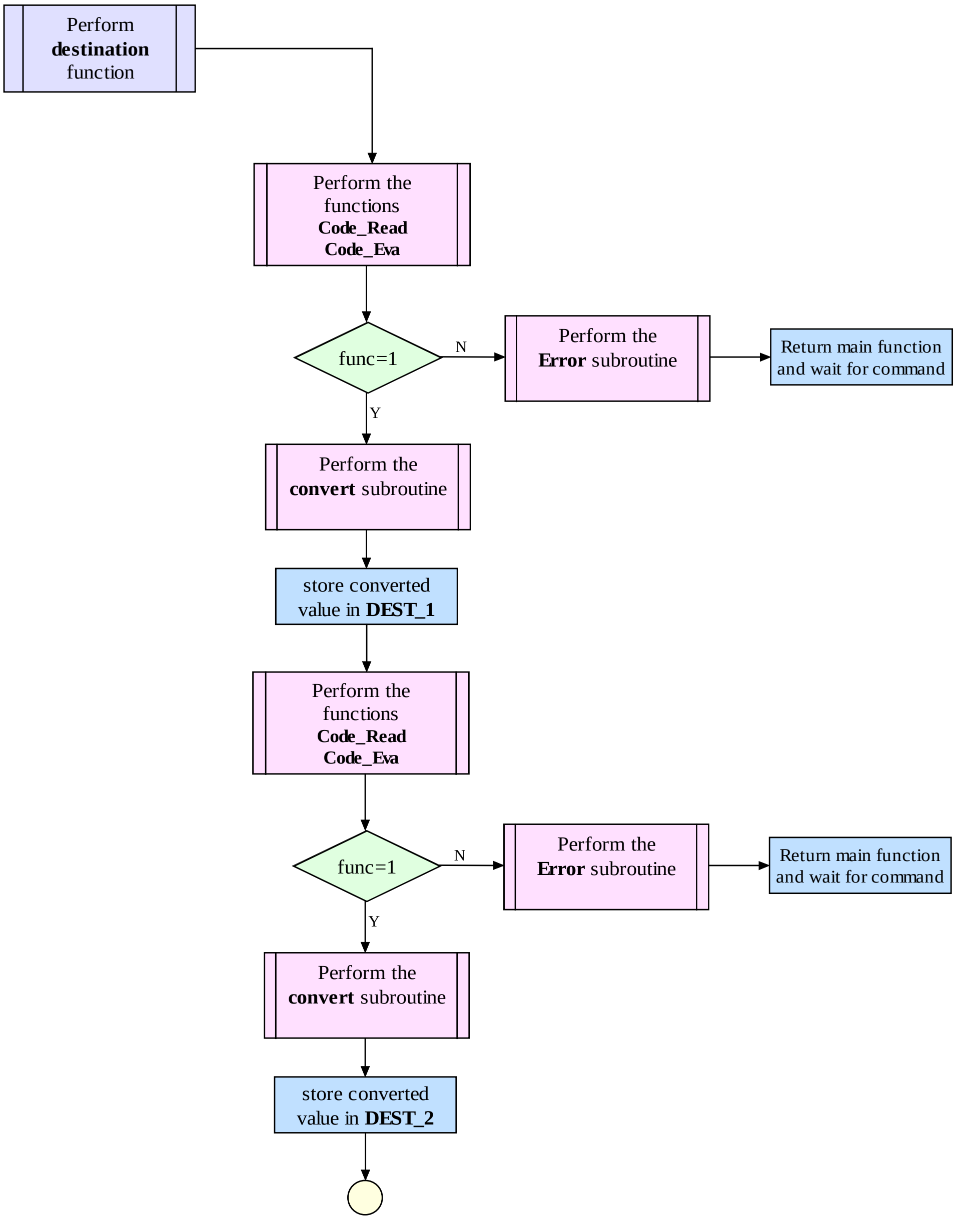


Figure 9. 10-a: The flowchart of the Destination Entry program (part-1)

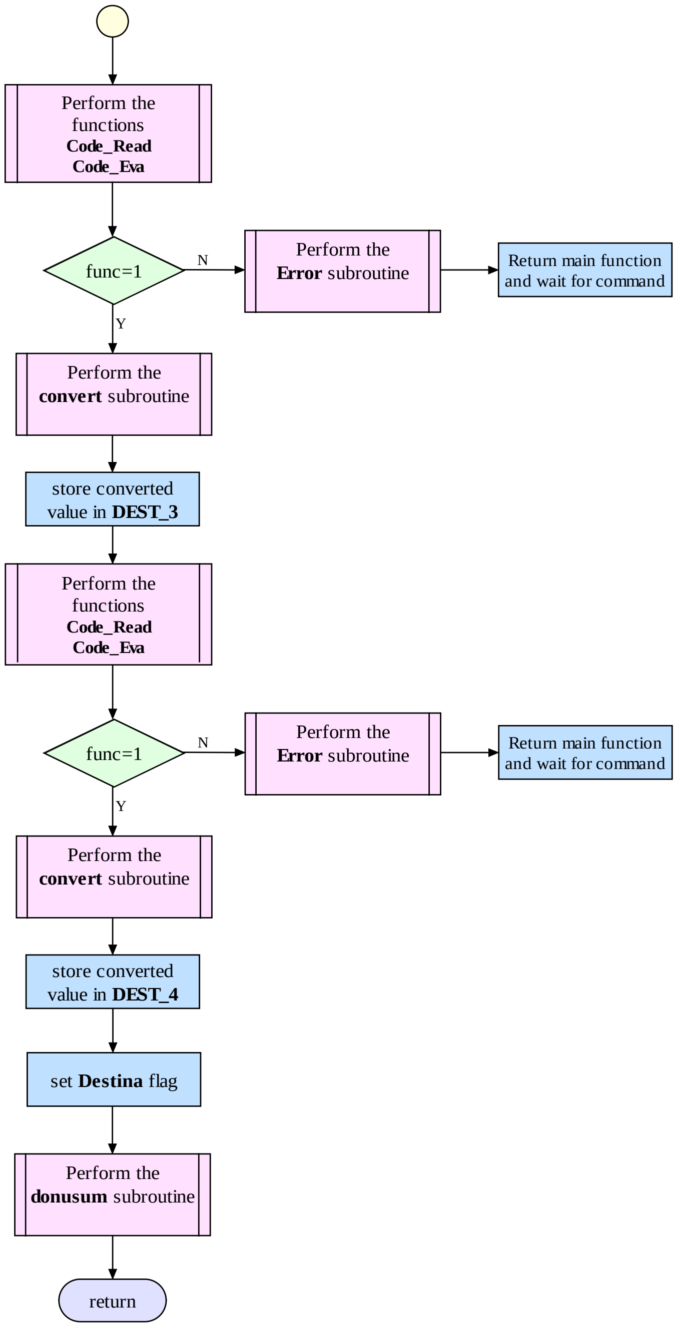


Figure 9. 10-b: The flowchart of the Destination Entry program (part-2)

The source code of the program is given blow:

```
* ------------------------------------------------------------- *
* DESTINATION - Read destination routine: 4 digit value         *
*               Store (DESTINA_1, DESTINA_2, DESTINA_3, DESTINA_4)
*               Goto OKU for next action                        *
* ------------------------------------------------------------- *

Destination:
            clr   DESTINA_1
            clr   DESTINA_2
            clr   DESTINA_3
            clr   DESTINA_4
            clr   Destina             ; No Destination values
            jsr   code_read           ; Read the MSD of destination
            sei                       ; Disable all interrupt
            lda   T2SC0
            mov   #$08,T2SC0
            lda   function
            cmp   #1
            bne   neg_5
            bsr   convert             ; convert code to number
            sta   DESTINA_1
            jsr   code_read           ; Read the Second digit of destination
            lda   function
            cmp   #1
            bne   neg_5
            bsr   convert             ; convert code to number
            sta   DESTINA_2
            jsr   code_read           ; Read the third digit of destination
            lda   function
            cmp   #1
            bne   neg_5
            bsr   convert             ; convert code to number
            sta   DESTINA_3
            jsr   code_read           ; Read the LSD of destination
            lda   function
            cmp   #1
            bne   neg_5
            bsr   convert             ; convert code to number
            sta   DESTINA_4
            mov   #$44,Destina        ; A value is entered into destination
            bsr   donusum
            bra   next_1

neg_5       jsr   hata
next_1      cli                       ; enable all interrupt
            lda   T2SC0
            mov   #$48,T2SC0
            rts

* ------------------------------------------------------------- *
* CONVERT - Convert IR data to number                           *
*           Read code, return number in ACC                     *
* ------------------------------------------------------------- *

convert     lda   code
            and   #$F0
            lsra
            lsra
            lsra
            lsra
            sta   temp
            and   #$2
            asla
            sta   temp_1
            lda   temp
```

```
            and   #$4
            lsra
            ora   temp_1
            sta   temp_2
            lda   temp
            and   #$1
            asla
            asla
            asla
            sta   temp_1
            lda   temp
            and   #$08
            lsra
            lsra
            lsra
            ora   temp_1
            ora   temp_2
            inca
            cmp   #$0A
            bne   atlat
            clra
atlat       rts

* -----------------------------------------------------------------------*
* DONUSUM: digit decimal number will be converted to hexadecimal          *
* 4 digit will be respectively in DESTINA_1, DESTINA_2, DESTINA_3, DESTINA_4 *
* Result will be in DEST_1 ve DEST_2                                     *
* ---------------------------------------------------------------------- *

Donusum:
            clr   DEST_1
            clr   DEST_2

* Birler basamağı: Aynen sonuca katıldı

            lda   DESTINA_4
            sta   DEST_2

* Onlar basamağı 10 ile çarpılıp sonuca katıldı

            lda   DESTINA_3
            ldx   #$0A                ; 10 ile çarma
            mul                       ; sonuç X + A da
            sta   temp_3
            stx   temp_2
            bsr   topla

* Yüzler basamağı 100 ile çarpılıp sonuca katıldı

            lda   DESTINA_2
            ldx   #$64                ; 100 ile çarma
            mul                       ; sonuç X + A da
            sta   temp_3
            stx   temp_2
            bsr   topla

* Binler basamağı 1000 ile çarpılıp sonuca katıldı
* 1000 ile çarpma iki aşamalı gerçekeleşebilir : 125*8

            lda   DESTINA_1
            ldx   #$7D                ; 125 ile çarma
            mul                       ; sonuç X + A da
            sta   temp_3
            stx   temp_2
            sta   temp_4              ; yedek

            asl   temp_3
            asl   temp_3
```

```
            asl   temp_3
            asl   temp_2
            asl   temp_2
            asl   temp_2
            lda   temp_4
            lsra
            lsra
            lsra
            lsra
            lsra
            ora   temp_2
            sta   temp_2
            bsr   topla
            rts

topla       lda   temp_3
            add   DEST_2
            sta   DEST_2
            lda   temp_2
            adc   DEST_1
            sta   DEST_1
            rts
```

9.2.8.2 Step Number Entry

The step number of steering can be given in two digit value. This program read these two digit value and convert into hexadecimal value. The flow chart of the step number entry program is given in Figure 9.11.

Source code of the step number entry program is blow.

```
* --------------------------------------------------------------- *
* STEPNUMBER  - Read step number routine : 2 digit value          *
*               Store (STEPNUM_1, STEPNUM_2                       *
*               Goto OKU for next action                          *
* --------------------------------------------------------------- *

Step_Number:
            clr   STEPNUM_1
            clr   STEPNUM_2
            clr   Step_num_flag       ; No Step number value
            jsr   code_read           ; Read the MSD of step number
            sei                       ; disable all interrupt
            lda   T2SC0
            mov   #$48,T2SC0
            lda   function
            cmp   #1
            bne   neg_8
            jsr   convert             ; convert code to number
            sta   STEPNUM_1
            jsr   code_read           ; Read the LSB digit of step number
            lda   function
            cmp   #1
            bne   neg_8
            jsr   convert             ; convert code to number
            sta   STEPNUM_2
            mov   #$FF,Step_num_flag  ; A value is entered into Step Number Flag
            lda   STEPNUM_1
            ldx   #$0A
            mul
```

```
                add   STEPNUM_2
                sta   STPSTE              ; 8 bit Number of step
                cli                       ; enable all interrupt
                lda   T2SC0
                mov   #$48,T2SC0
                bra   next_2

neg_8           jsr   hata
next_2          cli                       ; enable all interrupt
                lda   T2SC0
                mov   #$48,T2SC0
                rts
```

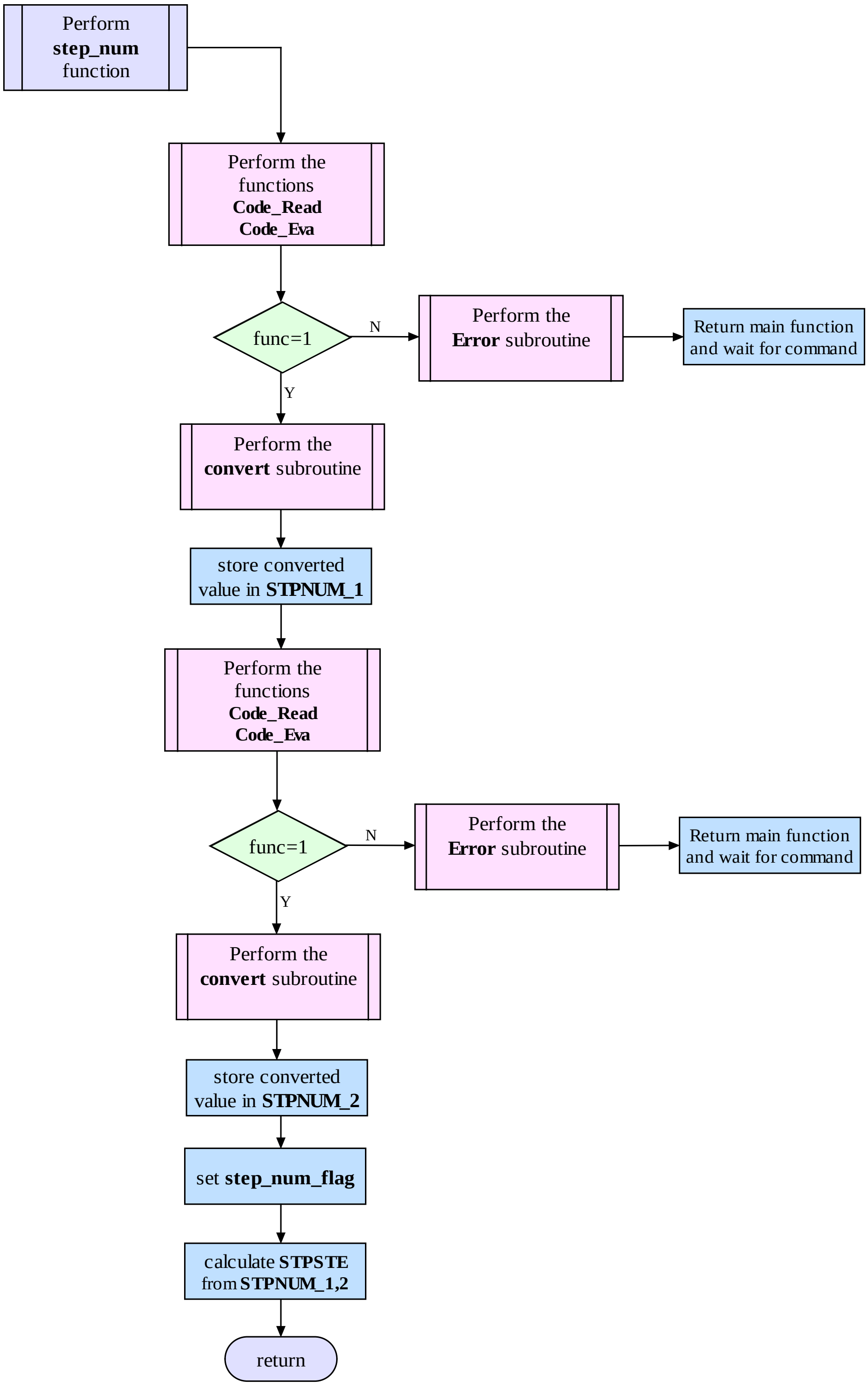


Figure 9. 11: The flowchart of the Step Number Entry program

9.2.8.3 Motion Speed Entry

Motion speed is one digit value and it is read by motion speed program. The program is given blow.

```
* ------------------------------------------------------------- *
* SPEED_MOTION- Read speed of motion routine : 1 digit          *
*              Stote speed in to SPMOTION                       *
*              Goto OKU for next action                         *
* ------------------------------------------------------------- *

Speed_Motion:
            jsr   code_read          ;
            sei                       ; Disable all interrupt
            lda   T2SC0
            mov   #$08,T2SC0
            lda   function
            cmp   #1
            bne   neg_2
            jsr   convert             ; convert code to number on ACC
            cmp   #$7
            blt   atla_14
            lda   #$06
atla_14     ldhx  #$0000
            sta   temp
            sta   SPDMOTION
            beq   atla_15
ekle        aix   #$7F
            aix   #$7F
            dbnz  temp,ekle
atla_15     sthx  Speed               ; 00 = 0,  FE = 1, 1FD=2, 2FC=3, 3FB=4,
4FA=5, 5F9=6
            jsr   q_second
            bra   next_3

neg_2       jsr   hata
next_3      cli                       ; enable all interrupt
            lda   T2SC0
            mov   #$48,T2SC0
            rts
```

9.2.8.4 Steering Speed Entry

Steering speed is one digit value and it is read by motion speed program. The program is given blow and flow chart is given in Figure 9.12.

```
* ------------------------------------------------------------- *
* SPEED_STEE  - Read speed of steering routine : 1 digit        *
*              Calculate delay for this speed                   *
*              Stote delay in to SPDDIRSTE                      *
*              Goto OKU for next action                         *
* ------------------------------------------------------------- *

Speed_Stee:
            jsr   code_read           ; Read speed of steering
            sei                       ; Disable all interrupt
            lda   T2SC0
```

```
        mov   #$08,T2SC0
        lda   function
        cmp   #1
        bne   neg_1
        jsr   convert               ; convert code to number
        sta   Speed_st              ; 0... 9
        inca
        ldhx  #$0066
```

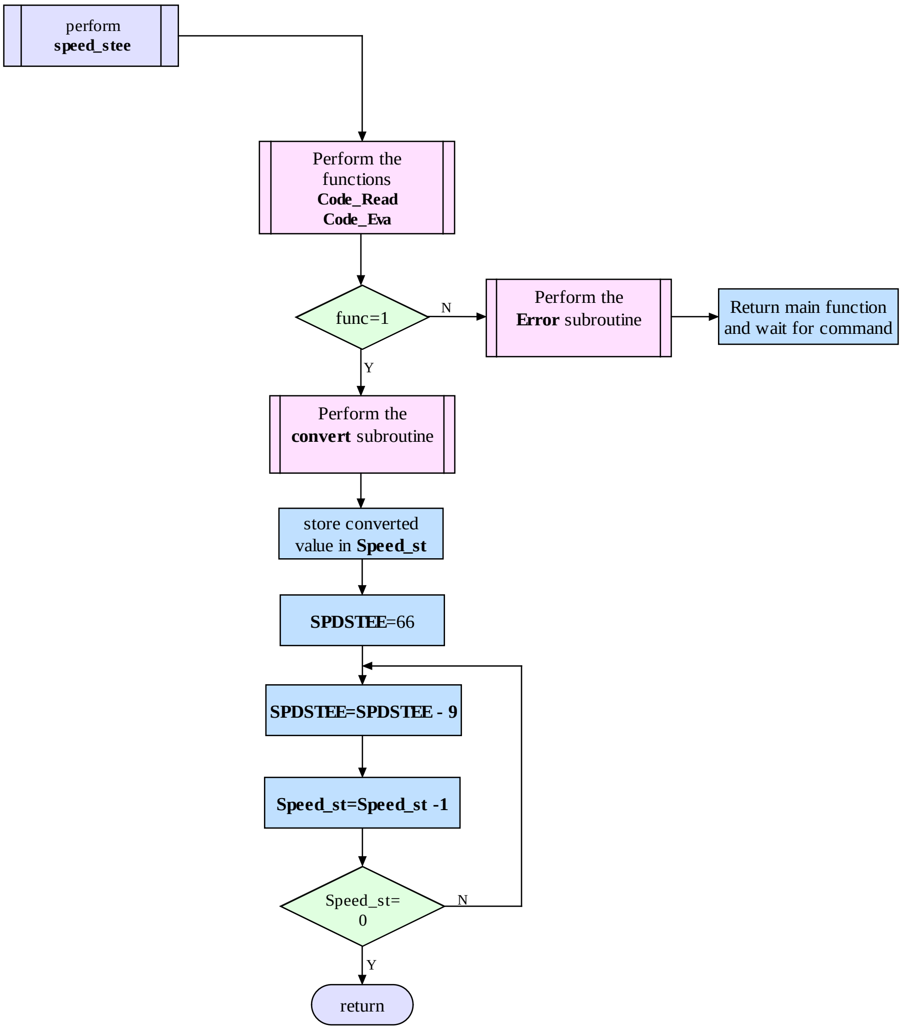


Figure 9. 12: The flowchart of the Steering speed Entry program

```
eksit   aix   #-9
        deca
        bne   eksit
        stx   SPDSTEE               ; Adjustment, multiplied by 2  !!!
neg_1   ldhx  #$E02F
        mov   Speed_st,temp
        inc   temp
looop   aix   #$1
        dbnz  temp,looop
        lda   0,x
        sta   sinyal
        cli                         ; enable all interrupt
        lda   T2SC0
        mov   #$48,T2SC0
        rts
```

9.2.9 Motion Speed Control Programs

There are three speed control programs;

- Speed up
- Speed down

Source code of this program are as follows.

9.2.9.1 Speed Up

Whenever "Speed up" key is pressed, this program is activated. This program is increase the speed of main program by one step. Program controls the highest speed.

```
speed+      lda   SPDMOTION
            cmp   #$6
            bge   sinir
            jsr   eof_segment
            lda   SPDMOTION
            inca
            sta   SPDMOTION
            sta   old_SPD
            ldhx  T1CH0H
            aix   #7F
            aix   #7F
            sthx  T1CH0H
            sthx  speed
hopa        jmp   rtf_int
sinir       mov   #$6,SPDMOTION
            mov   #$6,old_SPD
            bra   hopa
```

9.2.9.2 Speed Down

Whenever "Speed down" key is pressed, this program is activated. This program is decrease the speed of main program by one step. Program controls the lowest speed.

```
speed-      lda   SPDMOTION
            cmp   #$0
            ble   sinira
            jsr   eof_segment
            lda   SPDMOTION
            deca
            sta   SPDMOTION
            sta   old_SPD
            ldhx  T1CH0H
            aix   #-7F
            aix   #-7F
            sthx  T1CH0H
            sthx  speed
hoppa       jmp   rtf_int
```

```
sinira      mov   #$0,SPDMOTION
            bra   hoppa

durdur      jsr   stop
            jmp   rtf_int
```

9.2.10 Stop and End of Segment

Stop and End of Segment programs are prepared for stop motion and storing segment values.

9.2.10.1 Stop

Stop program, stop the main motor. If a forward type motion is in action, slow down process taking in account. The flow chart of the stop program is given in Figure 9.13.

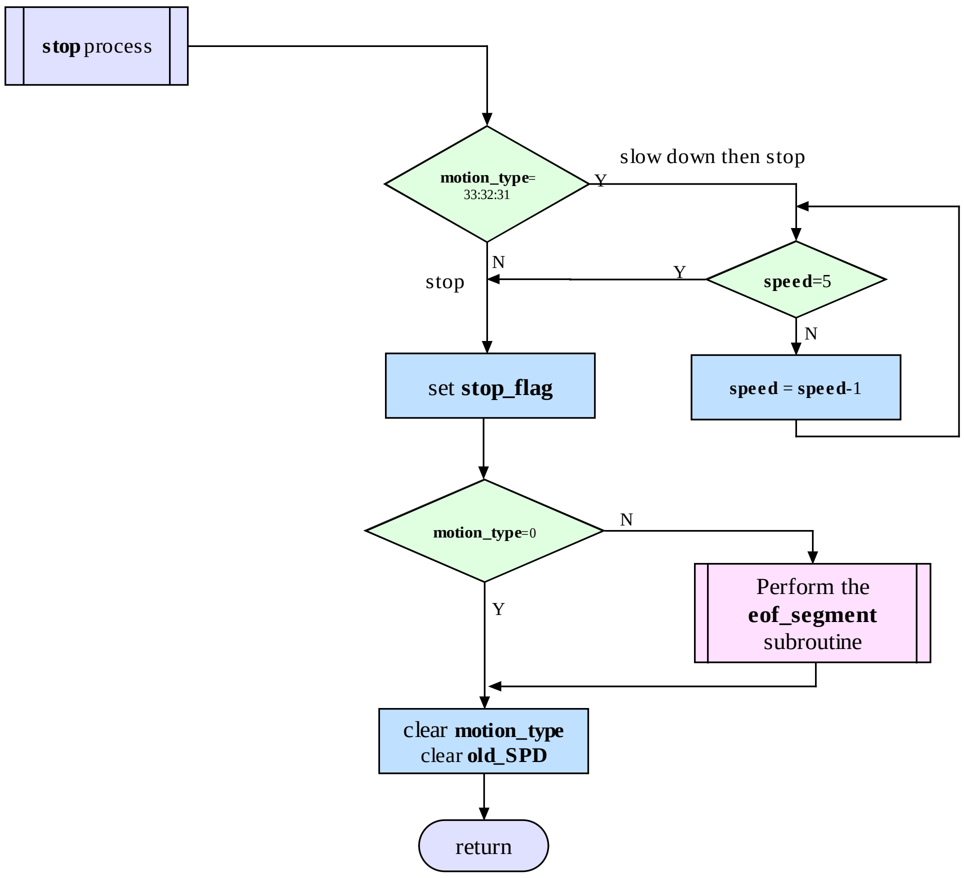


Figure 9. 13: The flowchart of the Stop program

The stop program source code is blow.

```
* ------------------------------------------------------------- *
* STOP  - Stop Control Subroutine                               *
*         90 right Forward, 91 Right Backward, 92 Left forward,  *
*         93 Left Backward                                      *
*         CC left, CE right, CD forward, CF backward            *
*         End of segment                                        *
* ------------------------------------------------------------- *

Stop:
            lda   motion_type
            cmp   #$31              ; Right + forward
            beq   yavasla
            cmp   #$32              ; Left + forward
            beq   yavasla
            cmp   #$33              ; forward
            beq   yavasla
            bra   dur

yavasla     clrx
            clra
            ldhx  Speed             ; last speed
geri_2      cphx  #0005
            bls   dur
            aix   #-01
            sthx  T1CH0H            ; TUCATU moves forward until stop
            jsr   gecik
            bra   geri_2

dur         sthx  speed
            lda   #$F0
            sta   PTD
            lda   #$00
            sta   PTA
            mov   #$6D,BPTC         ; RED lights are on
            mov   #$6D,PTC
            mov   #$11,stop_flag    ; Indicate a stop action
            lda   motion_type
            beq   next_7
            bsr   eof_segment

next_7      clr   motion_type
            clr   old_SPD
            rts
```

9.2.10.2 End of Segment

At the end of each segment, segment values are written in RAM area. The flowchart of the “End of Segment” program is given in Figure 9.14.

The source code of the program is given as follows.

```
* ------------------------------------------------------------- *
* END of SEGMENT - Write segment parameters into RAM            *
* ------------------------------------------------------------- *

eof_segment:
            lda   ult_flag
            cbeqa #$FF,zipla4
            lda   mode
            cmp   #$AA
            bne   zipla3
            ldhx  #$100               ; finding segment start address
            lda   segment
            beq   zipla
artir       aix   #$6
            deca
            bne   artir
zipla
            lda   motion_type
            sta   0,x                 ; type of action
            lda   old_SPD
            sta   1,x
            lda   SPDSTEE             ; Speed of Steering
            sta   2,x
            lda   A_STPSTE
            sta   3,x                 ; Number of steps
            lda   ACTUAL_1
```

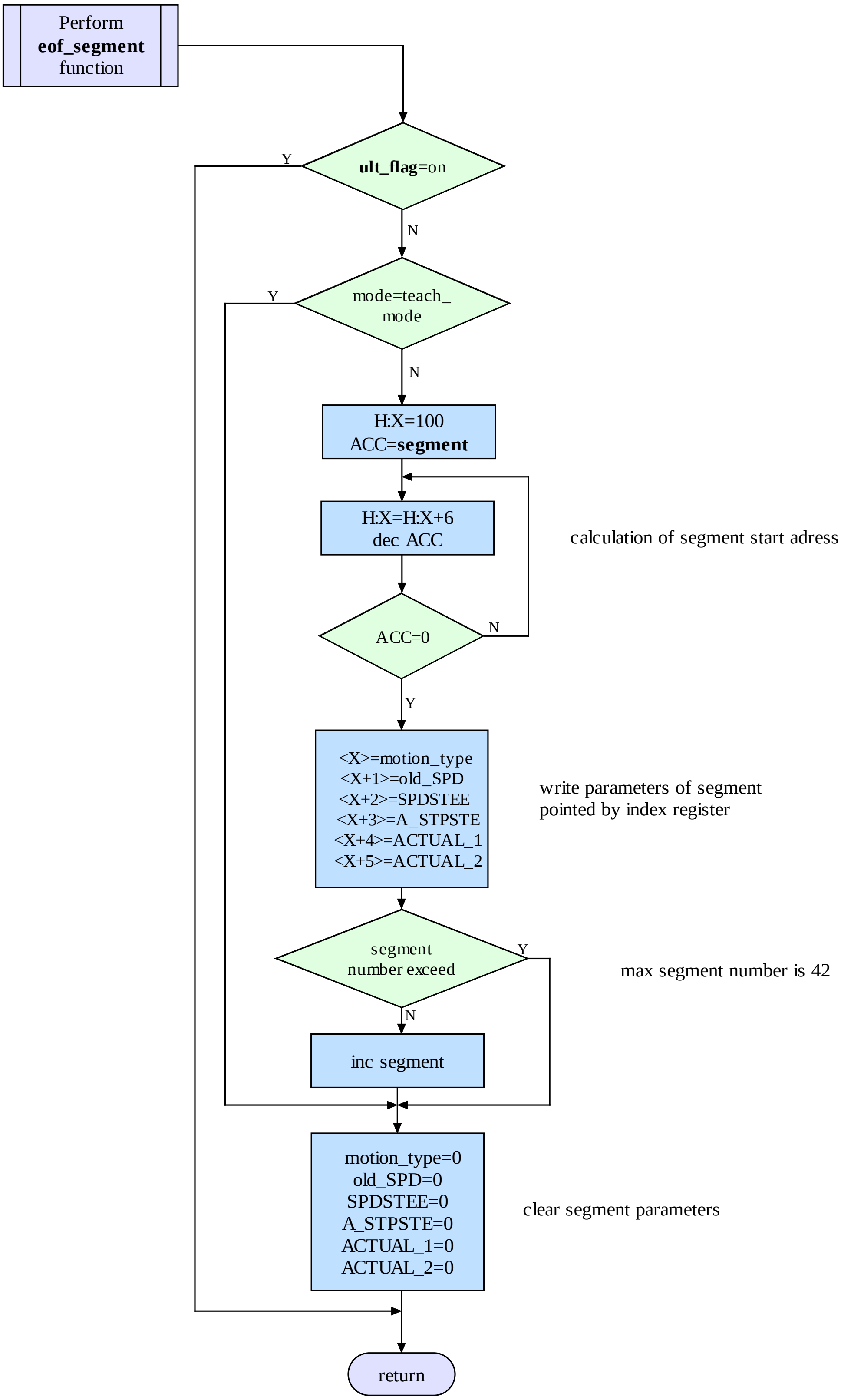


Figure 9. 14: The flowchart of the End of Segment program

```
            sta   4,x
            lda   ACTUAL_2              ; Value of destination msb
            sta   5,x                   ; Value of destination lsb
            lda   segment
            cmp   #$42
            ble   zipla2
            jsr   di
            jsr   di
            bra   zipla3
zipla2      inc   segment

zipla3      lda   #$00                  ; number of steps for steering

            clr   code_old
            sta   step_num_flag         ; Step number flag
            sta   Dest_1
            sta   Dest_2
            sta   Destina               ; Destination flag
            sta   ACTUAL_1
            sta   ACTUAL_2
            sta   STPSTE
            sta   A_STPSTE
            lda   #$00
            sta   PTA                   ; Step motor initial values
            sta   BPTA                  ; Step motor buffer

zipla4
            jsr   da                    ;
            rts
```

9.2.11 Flash Erase and Write

For playback activity, role and segment parameters must be stored into Flash. In order to do this, Flash_erase, flash_write programs are written. In this part, flowcharts and source code of these programs will be seen.

9.2.11.1 Flash Erase

In order to write a data or a program into flash, related flash area must be erased. Erase program must be in Ram are. First of all Flash_Erase program transfer into RAM area, then run this program.

The transfer and flash program flow chart is given in Figure 9.15 and Figure 9.16. The source code of this program is blow.

```
* ------------------------------------------------------------- *
* ERASE_FLASH - First move flash erase program into RAM         *
*               Then run erase-flash program                    *
* ------------------------------------------------------------- *

erase_flash:
            ldhx  #Flash_Erase-1
            sthx  bas                  ; start address of flash_erase program
            ldhx  #sil_son
            sthx  son                  ; end address of flash_erase program

* ----- Block move  ----- *

            ldhx  son                 ; end address of block
devam       lda   ,x
```

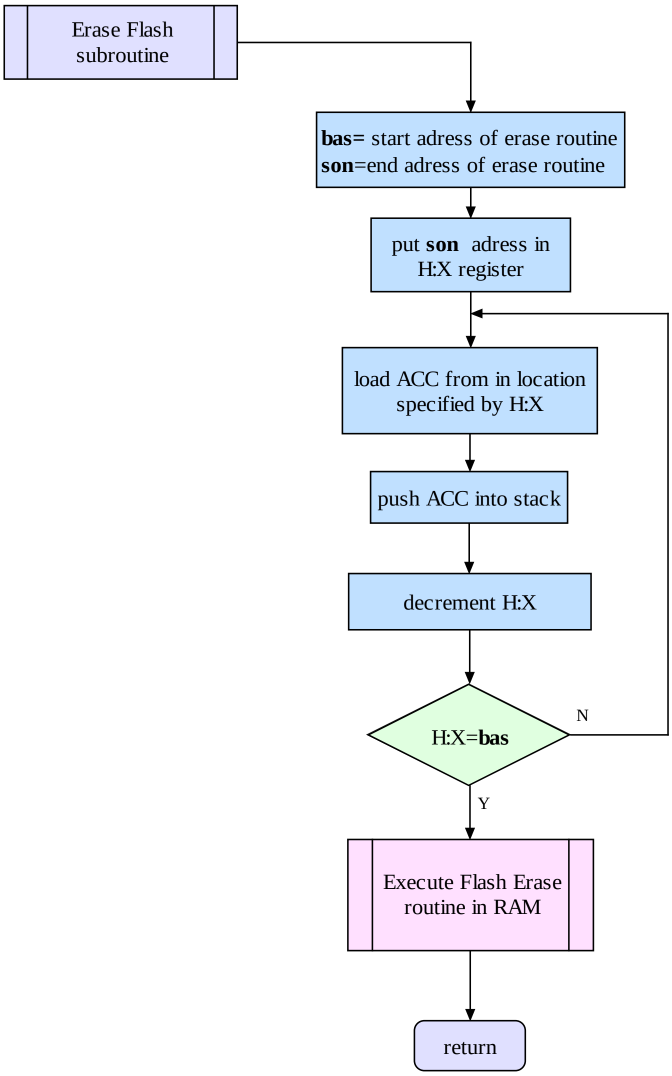


Figure 9. 15: The flowchart of the Transfer and Flash Erase program

```
            psha
            aix   #-1
            cmphx bas                    ; start address of block
            bne   devam

            ldhx  flash_start_adr         ; start address of flash
            sthx  temp
            tsx                           ; Start address of the erease program
sthx  temp_2                              ; Save program start address
            jsr   ,x                      ; Run flash-erase program for first 128B
            ldhx  flash_start_adr
            aix   #$7F
            incx
            sthx  temp                   ; second area
            ldhx  temp_2
            jsr   ,x                     ; Run flash-erase program for second 128B
            ais   #{sil_son-Flash_Erase+1}
            rts
```

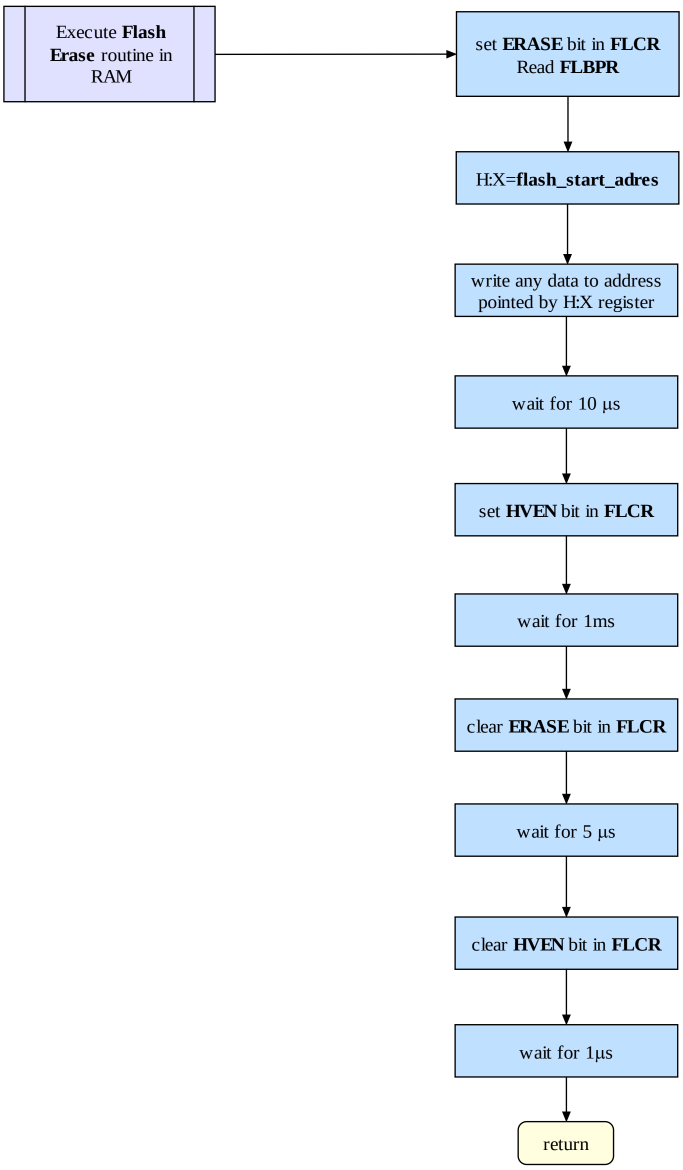


Figure 9. 16: The flowchart of the Flash Erase program

```
Flash_Erase:

* -----  Flash erase  ----- *

* 1. step :             ERASE<-1
            lda   #$02
            sta   FLCR

* 2. step :             Read FLBPR
            lda   FLBPR              ; read flash block protect register

* 3. step :             Write a dummy data into erased area
            ldhx  temp
            sta   ,x                 ; any address in the page

* 4. step :    Wait for 10us, each step is 400ns, so 10.000/400=25 step is needed
            lda   #$07
            nop
            nop
            dbnza  *

* 5. step :             HVEN<-1
            lda   #$A
            sta   FLCR

* 6. step               Wait for 1ms, 1.000.000/400=2.500 step is needed
            ldx   #$4
            nop
            nop
azalt       lda   #$CE
            dbnza  *
            nop
            dbnzx azalt

* 7. step :              ERASE<-0
            lda   #$8
            sta   FLCR

* 9. step :              Wait for 5us, 5000/400=13 step is needed
            lda   #$4
            dbnza *

* 9. step :              HVEN<-0
            clra
            sta   FLCR

* 10. step :             Wait for 1us, 1000/400=3 step is needed
            nop
            nop
sil_son     rts
```

9.2.11.2 Flash Write

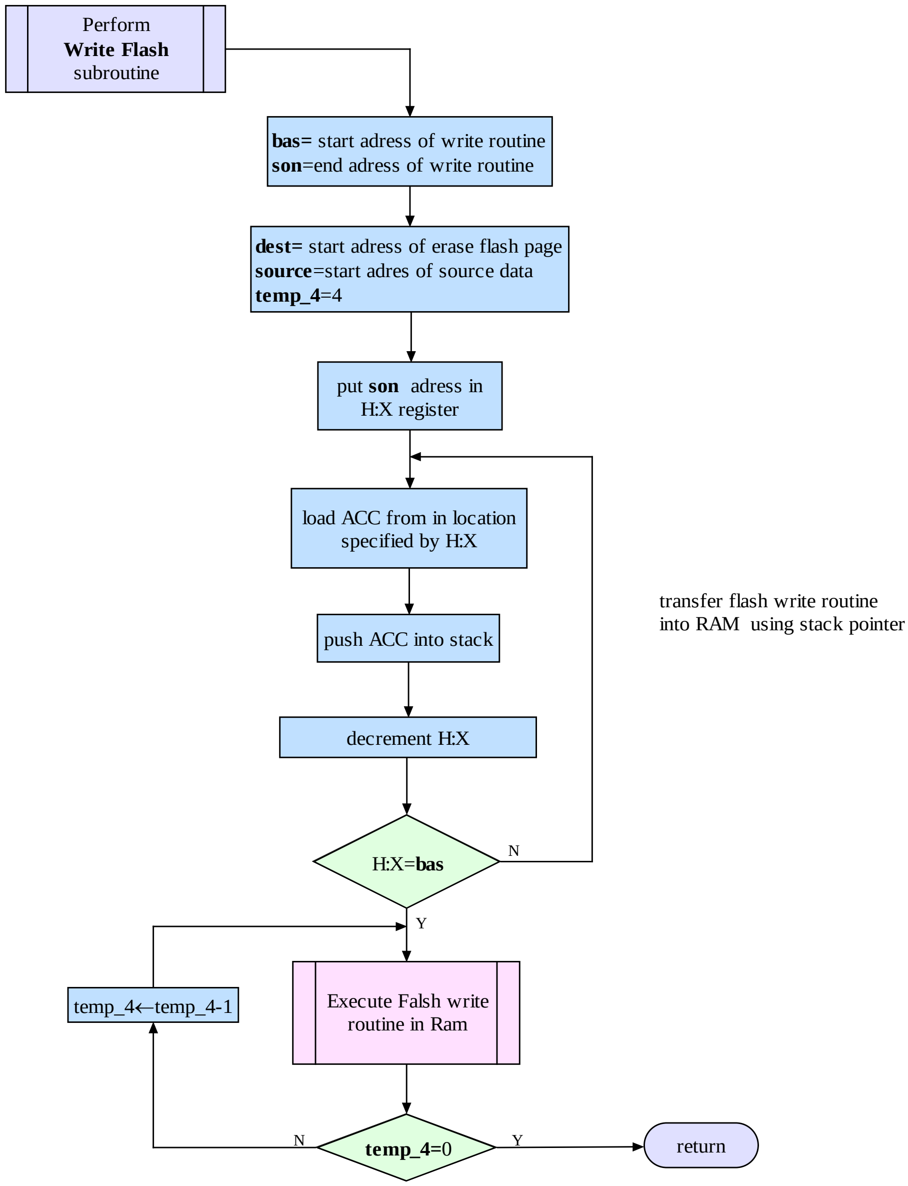


Figure 9. 17: The flowchart of the Transfer and Flash Write program

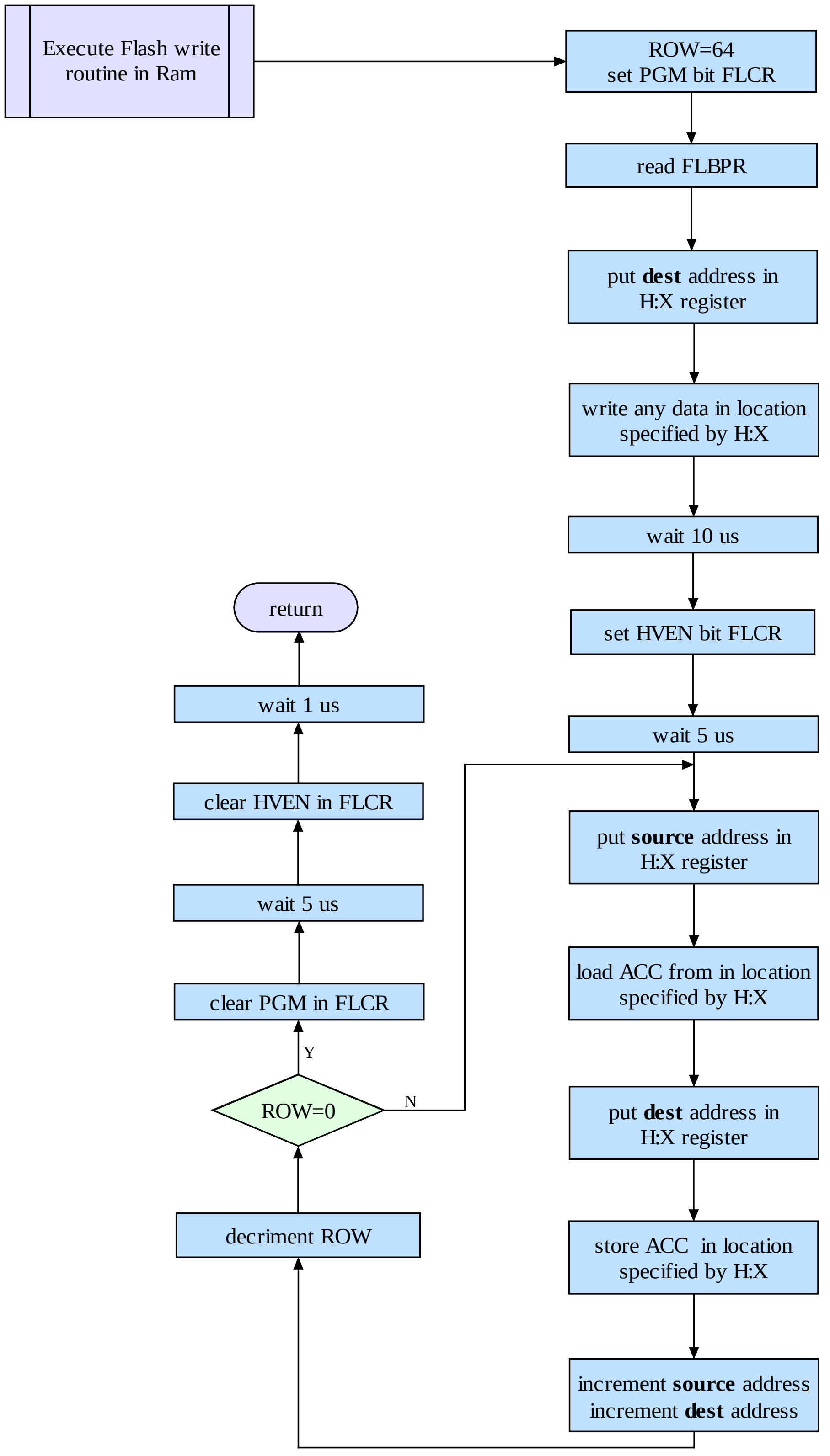


Figure 9. 18: The flowchart of the Flash Write program

```
* ------------------------------------------------------------- *
* WRITE_FLASH - First move flash write program into RAM         *
*            Then run write-flash program                        *
* ------------------------------------------------------------- *

Write_flash:

* ----- Block move ----- *

            ldhx  #RamWriteEE-1
            sthx  bas                    ; start address of flash_erase program
            ldhx  #PGM_son
            sthx  son                    ; end address of flash_erase program

            ldhx   son                   ; end address of block
devamm      lda    ,x
            psha
            aix    #-1
            cmphx  bas                   ; start address of block
            bne    devamm

            mov    #$4,temp_4
            ldhx   flash_start_adr      ; start address of role in FLASH
            sthx   dest
            ldhx   #$0100               ; start address of role in RAM
            sthx   source

            tsx                          ; Start address of flash_yaz program
            sthx  temp
dallan      jsr   ,x                     ; Run write operation for one block
            ldhx  temp
            dbnz  temp_4,dallan
            ais   #{PGM_son-RamWriteEE+1}
            rts

* -----  Flase write  ----- *

RamWriteEE:
            mov #Row_Size,Row

* 1. step                 PGM<-1
            lda   #1
            sta   FLCR

* 2. step                 read FLBPR
            lda   FLBPR                 ; read flash block protect register

* 3. step                 write any data into writen area
            ldhx   dest
            sta    ,x                   ; write any data

* 4. step                 wait for 10us, each step is 400ns, 10.000/400=25 step
is needed
            lda   #$07
            nop
            nop
            dbnza  *

* 5. step                 HVEN <- 1
            lda   #9
            sta   FLCR

* 6. step                 wait for 5us, 5000/400=13 step is needed
            lda   #$4
            dbnza *

RamWriteEE1:
```

```
* 7. step                   write data into Flash
            ldhx source
            lda  ,x
            ldhx dest
            sta ,x
            inc  dest+1
            bne  RamWriteEE2
            inc  dest

RamWriteEE2:
            inc  source+1
            bne  RamWriteEE3
            inc  source

* 9. step                   wait for 30-40us, 30000/400=16 step is needed

RamWriteEE3:
            lda   #$10
            dbnza *
            dbnz  Row,RamWriteEE1    ; 4us is needed after 64 byte write
                                       operation

* 9. step                   write all data of 64 byte data

* 10. step                  PGM<-0
            lda   #8
            sta   FLCR

* 11. step                  wait for 5us, 5000/400=13 step is needed
            lda   #$4
            dbnza *

* 12. step                  HVEN<-0
            clra
            sta   FLCR

* 13. step                  wait for 1us, 1000/400=3 step is needed
            nop
            nop
            nop
PGM_son     rts
```

9.2.12 Teach Mode

Teach mode or traning mode is one of the features of TUCATU. The flow chart of teach mode is given in Figure 9.19.

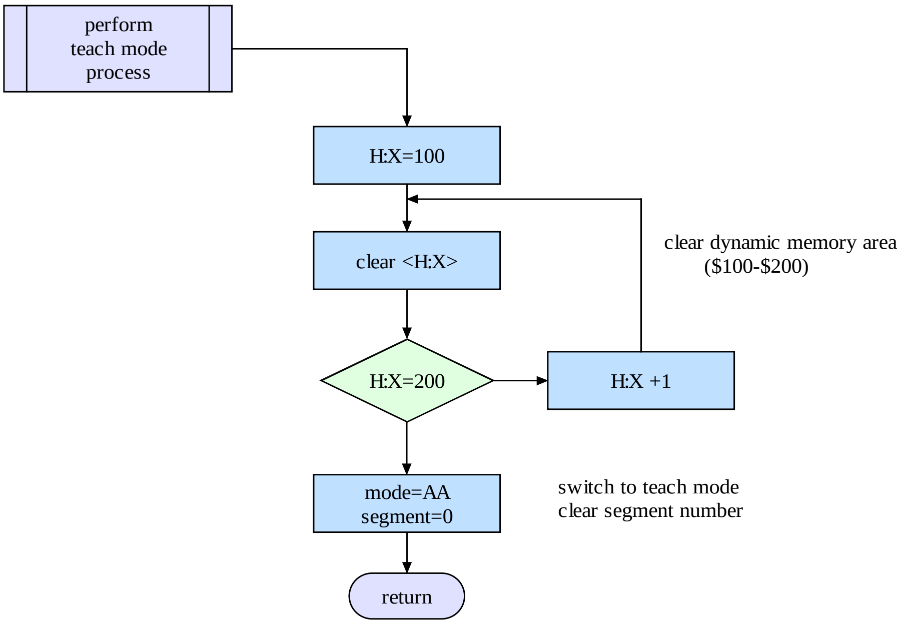


Figure 9. 19: The flowchart of the Teach mode program

The source code of teach mode is given blow.

```
* -------------------------------------------------------------- *
* Teaching Mode                                                  *
* -------------------------------------------------------------- *

            clr   mode              ; mode=0 ????!!! Teaching mode
                                    ; mode=1 Autonomous mode
            clr   function          ; clear function code
            clr   segment           ; clear segment number

            lda   T2SC1             ; T1SC1 okunda    ekleme tarihi 11_subat
            lda   #$08              ; %00001000       11_subat
            sta   T2SC1             ; T1SC1 CHOF bayragi silindi,interrupt-off
11_subat

            lda   #$04
            sta   INTSCR            ; IRQ Interrup Enable

            lda   T2SC0
            mov   #$48,T2SC0        ; Timer Input Capture Interrupt Enable
            cli                     ; Enable all interrupt

bekle       bra   bekle             ; Wait for interrupt
```

9.2.13 Playback Program

In the playback mode, master may select any role; TUCATU playbacks this role. The flow chart of the menu program is given blow.

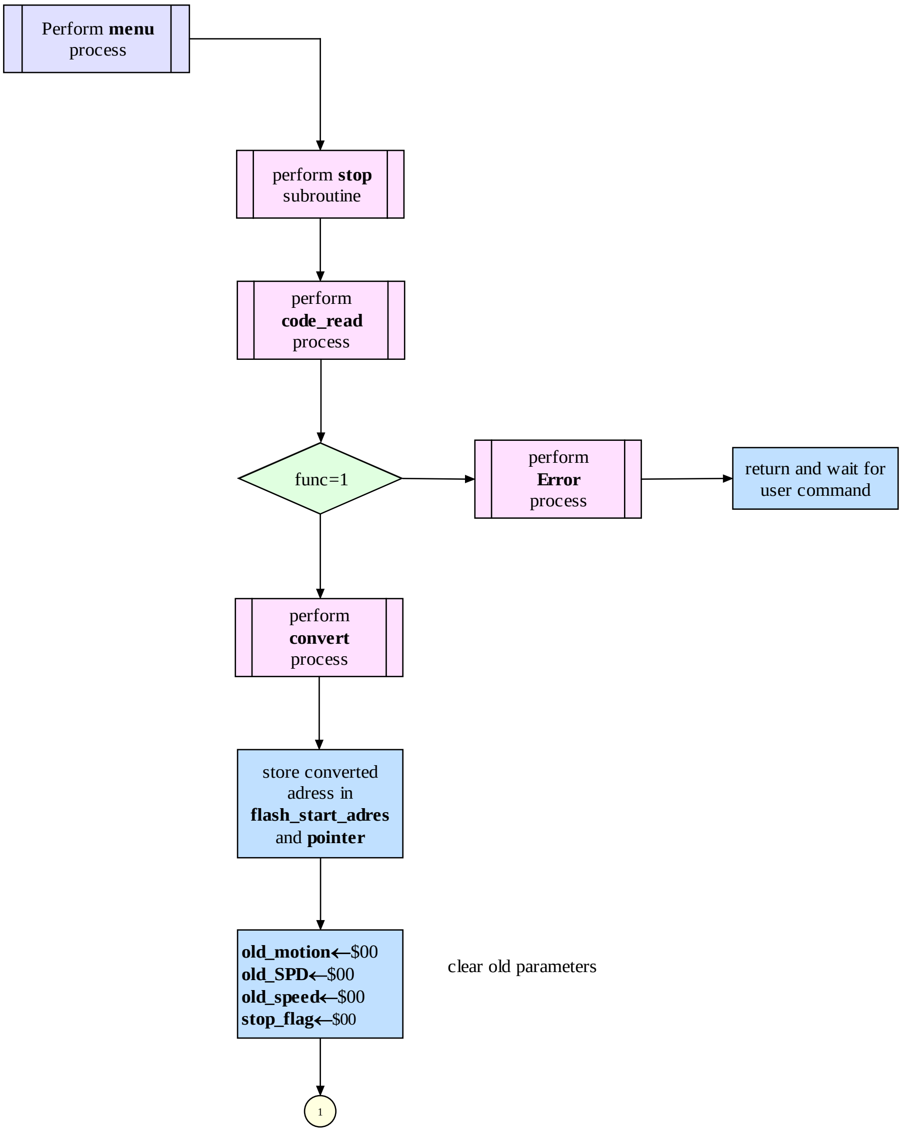


Figure 9. 20-a: The flowchart of the Menu program (part-1)

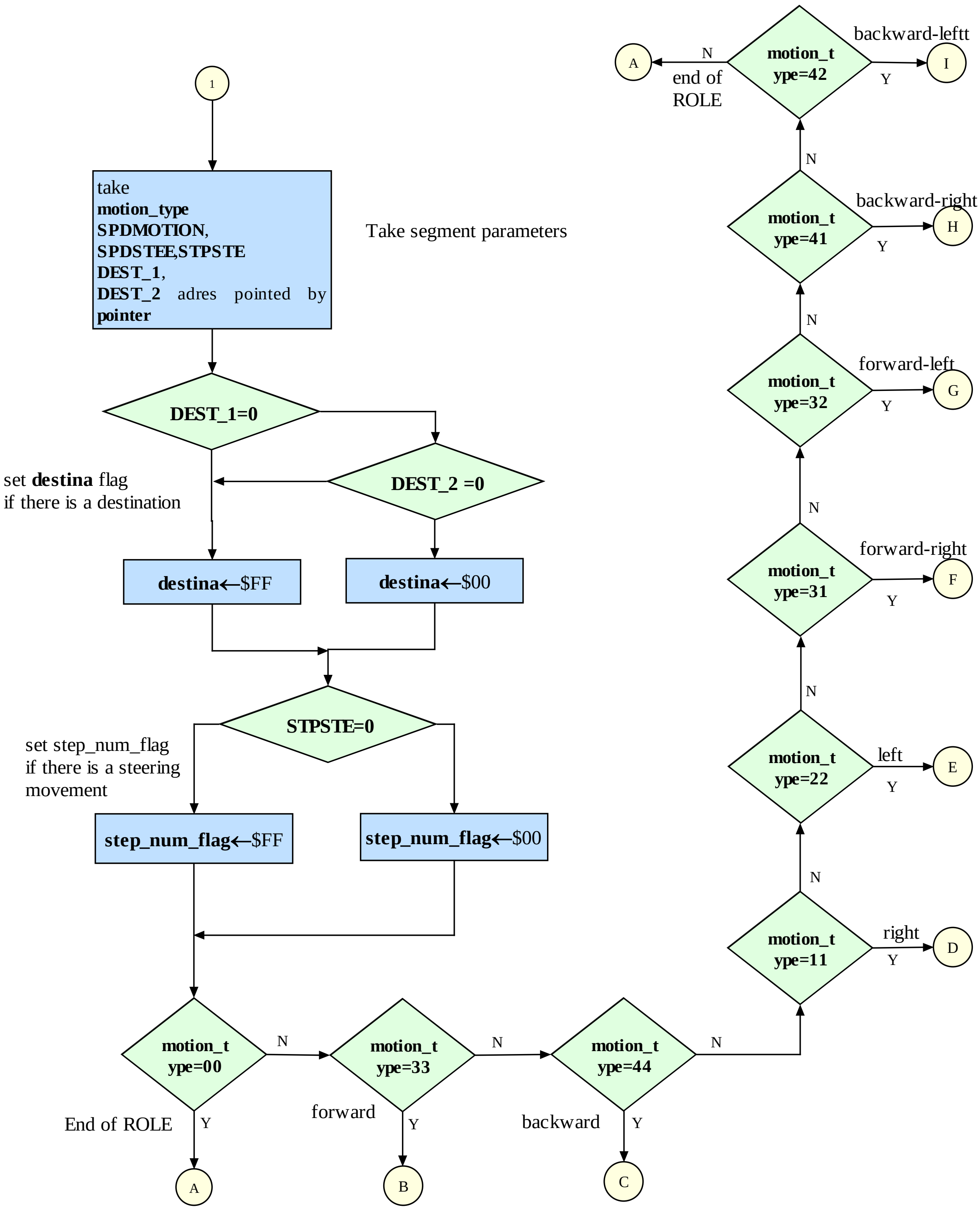


Figure 9. 20-b: The flowchart of the Menu program (part-2)

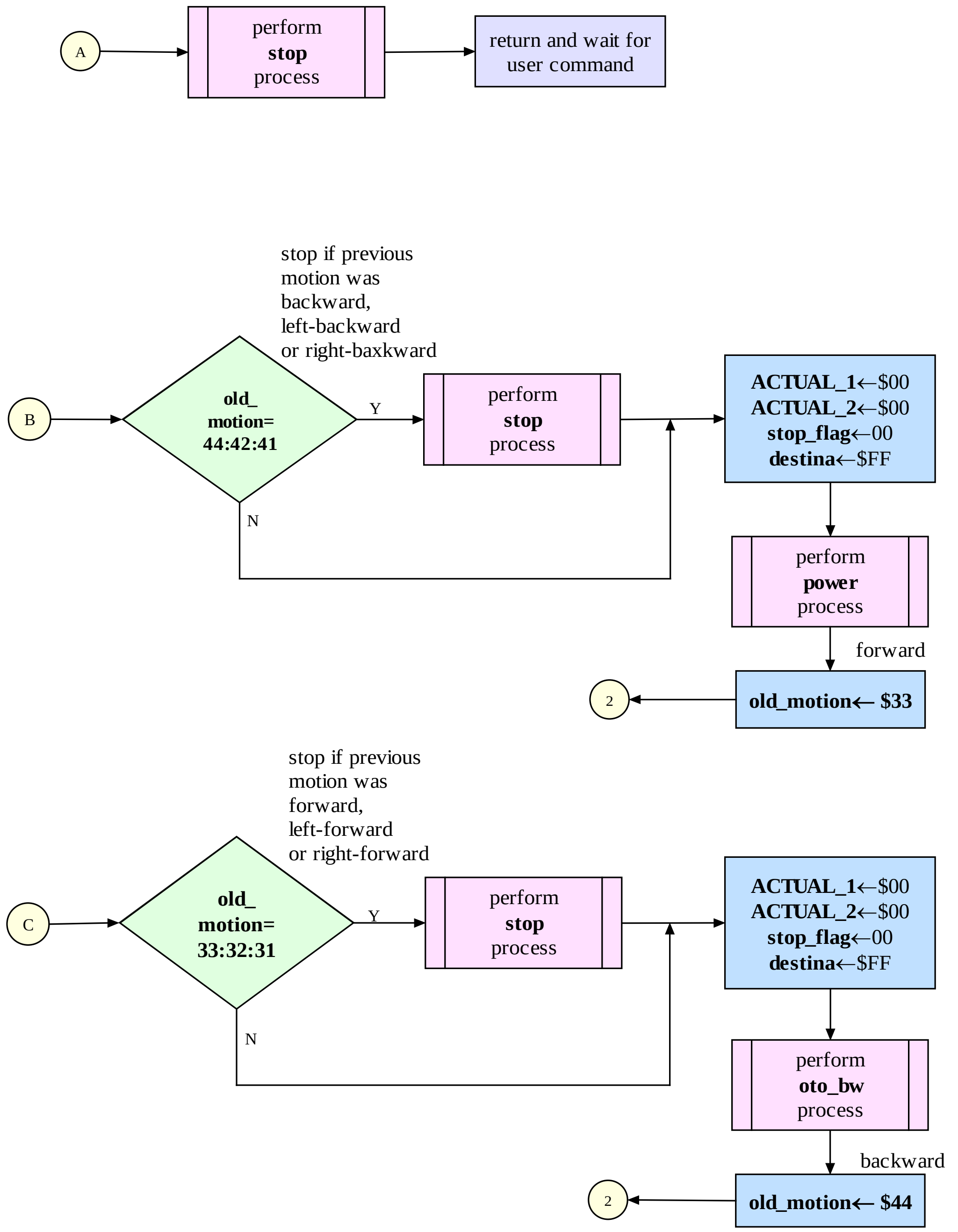


Figure 9. 20-c: The flowchart of the Menu program (part-3)

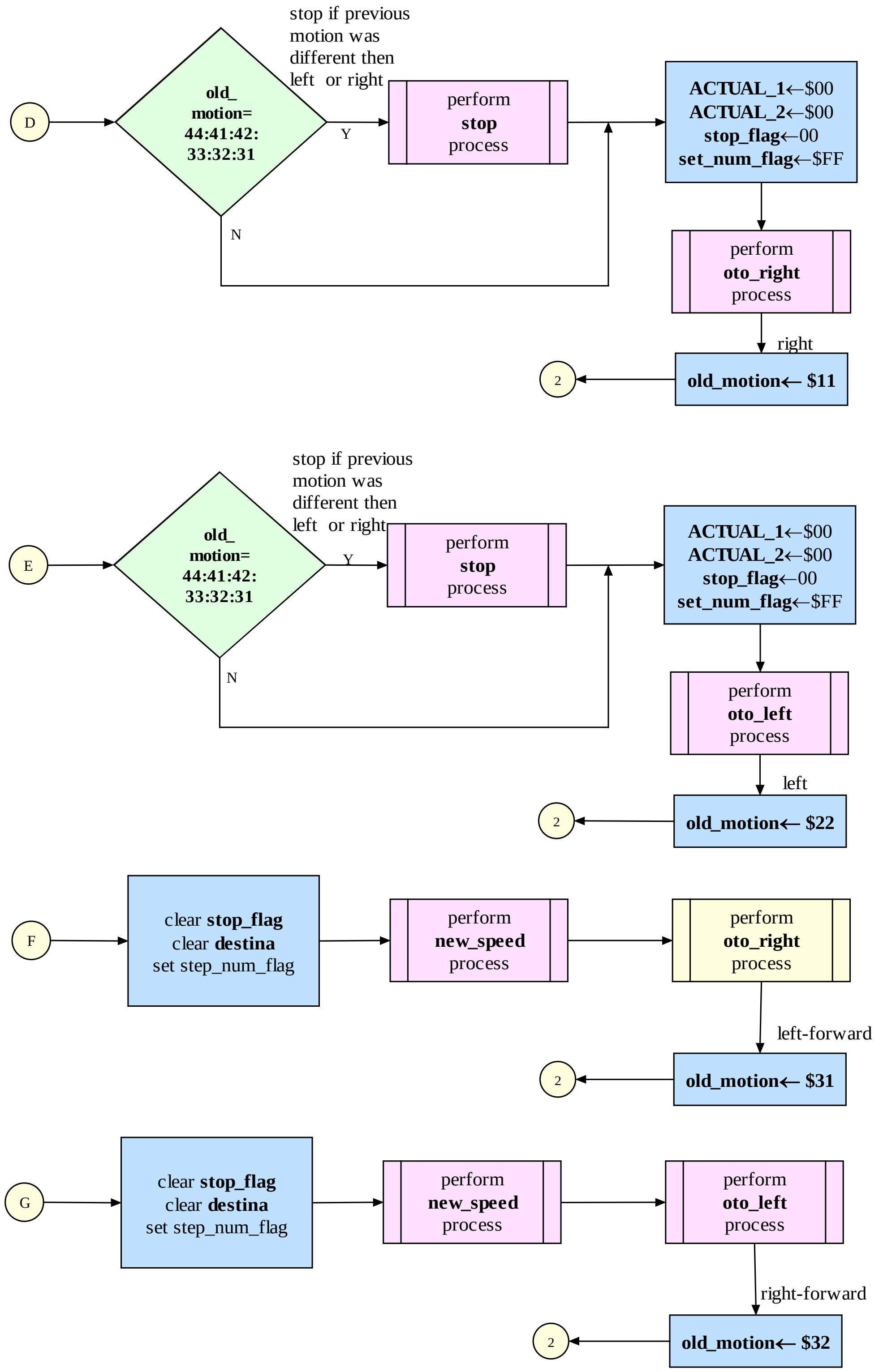


Figure 9.20-d: The flowchart of the Menu program (part-4)

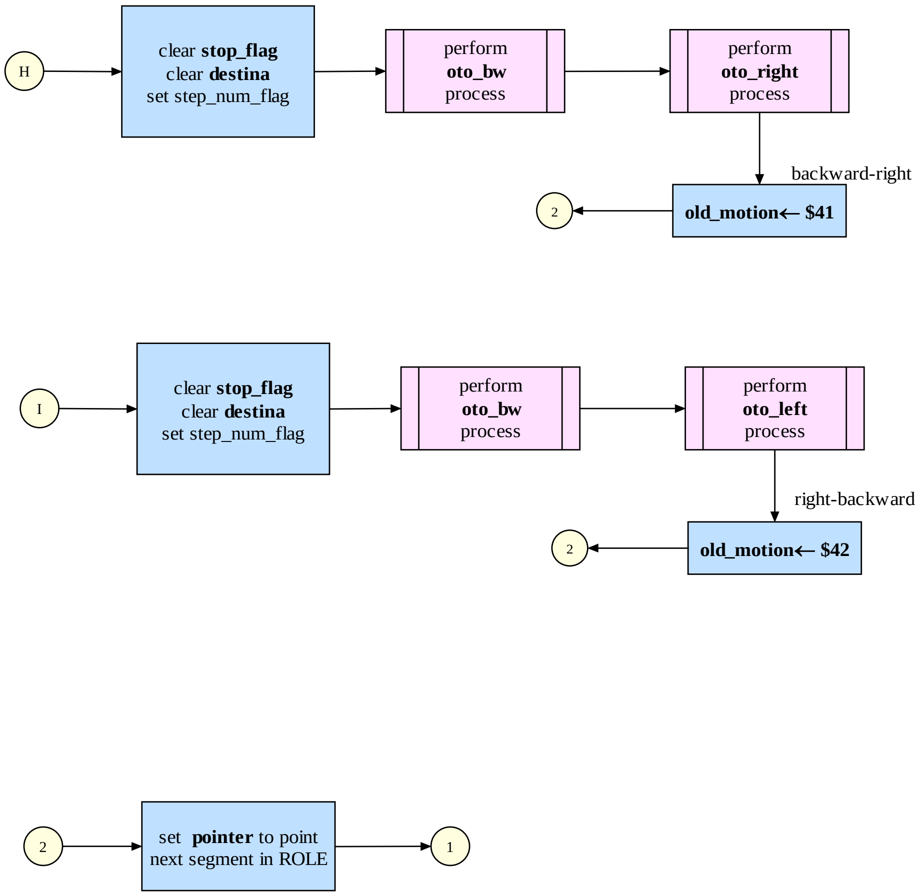


Figure 9.20-e: The flowchart of the Menu program (part-5)

The source code of menu program is given as follows.

```
* -------------------------------------------------------------- *
* MENU         - Read program number routine: 1 digit value      *
*                Calculate program address                       *
*                Jump to selected program                        *
* -------------------------------------------------------------- *
Menu:
            jsr   stop
            mov   #$1,mode              ; autonomous mode
            jsr   code_read             ; Read program number
            sei                         ; disable all interrupt
            lda   T2SC0                 ; disable T2SC0 interrupt and clear
                                         interrupt flag
            mov   #$08,T2SC0
            cli                         ;enable IRQ interrupt to count
            lda   function
            cmp   #1
            bne   neg_3
            jsr   convert               ; convert code to number
            asla                        ; address need two byte
            sta   role                  ; Role number * 2
            mov   #$E0,temp             ; Indirect address of flash
            sta   temp+1
            ldhx  temp                  ; Flash start address pointer
            lda   ,x
```

```
            psha
            lda  1,x
            psha
            pulx
            pulh
            sthx  flash_start_adr     ; Role start address
            sthx  pointer             ; Segment pointer
            clr   old_motion
            clr   old_SPD
            clr   old_speed
            clr   stop_flag
* -----  One segment parameters  ----- *

backk       lda   0,x                 ; type of action
            cmp   #$00
            beq   tamam
            sta   motion_type
            lda   1,x                 ; Speed of motion
            sta   SPDMOTION
            lda   2,x                 ; Speed of Steering
            sta   SPDSTEE
            lda   3,x                 ; Number of steps
            sta   STPSTE
            lda   4,x                 ; Value of destination msb
            sta   DEST_1
            lda   5,x                 ; Value of destination lsb
            sta   DEST_2
            ldhx  dest_1
            beq   dest_zero           ; Destination = 0
            mov   #$FF,destina
            bra   jump_1
dest_zero   clr   destina
jump_1      lda   STPSTE
            beq   stp_no_zero
            mov   #$FF,step_num_flag
            bra   jump_2
stp_no_zero clr   step_num_flag

* -----  Action ----- *

jump_2      lda   motion_type
            cmp   #$33
            beq   duz_ileri
            cmp   #$44
            beq   duz_geri
            cmp   #$11
            beq   duz_sag
            cmp   #$22
            beq   duz_sol
            cmp   #$31
            beq   sag_ileri
            cmp   #$32
            beq   sol_ileri
            cmp   #$41
            beq   sag_geri
            cmp   #$42
            beq   sol_geri

tamam       jmp   tamamm
neg_3       jmp   negg_3

duz_ileri    jmp   duz_ileri1
duz_geri     jmp   duz_geri1
duz_sag      jmp   duz_sag1
duz_sol      jmp   duz_sol1
sag_ileri    jmp   sag_ileri1
sol_ileri    jmp   sol_ileri1
sag_geri     jmp   sag_geri1
```

```
sol_geri      jmp   sol_geri1

duz_ileri1  lda   old_motion
            cmp   #$44
            beq   duz_ileri2
            cmp   #$42
            beq   duz_ileri2
            cmp   #$41
            beq   duz_ileri2
duz_ileri3  clr   stop_flag
            mov   #$FF,destina
            ldhx  pointer
            lda   4,x                   ; Value of destination msb
            sta   DEST_1
            lda   5,x                   ; Value of destination lsb
            sta   DEST_2
            clr   ACTUAL_1
            clr   ACTUAL_2
            jsr   power
            mov   #$33,old_motion
            mov   SPDMOTION,old_SPD
            jmp   bitis
duz_ileri2  jsr   stop
            bra   duz_ileri3

duz_geri1   lda   old_motion
            cmp   #$33
            beq   duz_geri2
            cmp   #$31
            beq   duz_geri2
            cmp   #$32
            beq   duz_geri2
duz_geri3   clr   stop_flag
            ldhx  pointer
            lda   4,x                   ; Value of destination msb
            sta   DEST_1
            lda   5,x                   ; Value of destination lsb
            sta   DEST_2
            mov   #$FF,destina
            clr   ACTUAL_1
            clr   ACTUAL_2
            jsr   oto_bw
            mov   #$44,old_motion
            jmp   bitis
duz_geri2   mov   #$33,motion_type
            jsr   stop
            mov   #$44,motion_type
            bra   duz_geri3

duz_sag1    lda   old_motion
            cmp   #$33
            beq   duz_sag2
            cmp   #$44
            beq   duz_sag2
            cmp   #$31
            beq   duz_sag2
            cmp   #$32
            beq   duz_sag2
            cmp   #$41
            beq   duz_sag2
            cmp   #$42
            beq   duz_sag2
duz_sag3    clr   stop_flag
            ldhx  pointer
            lda   2,x                   ; Speed of Steering
            sta   SPDSTEE
            lda   3,x                   ; Number of steps
            sta   STPSTE
```

```
            mov   #$11,motion_type
            mov   #$FF,step_num_flag
            jsr   oto_right
            mov   #$11,old_motion
            bra   bitis
duz_sag2    jsr   stop
            bra   duz_sag3

duz_sol1
            lda   old_motion
            cmp   #$33
            beq   duz_sol2
            cmp   #$44
            beq   duz_sol2
            cmp   #$31
            beq   duz_sol2
            cmp   #$32
            beq   duz_sol2
            cmp   #$41
            beq   duz_sol2
            cmp   #$42
            beq   duz_sol2
duz_sol3    clr   stop_flag
            ldhx  pointer
            lda   2,x                   ; Speed of Steering
            sta   SPDSTEE
            lda   3,x                   ; Number of steps
            sta   STPSTE
            mov   #$22,motion_type
            mov   #$FF,step_num_flag
            jsr   oto_left
            mov   #$22,old_motion
            bra   bitis
duz_sol2    jsr   stop
            bra   duz_sol3

sag_ileri1  clr   destina               ; ignore destination value
            clr   stop_flag
            bsr   new_speed
            jsr   oto_right
            mov   #$31,old_motion
            mov   SPDMOTION,old_SPD
            bra   bitis

sol_ileri1  clr   destina               ; ignore destination value
            clr   stop_flag
            bsr   new_speed
            jsr   oto_left
            mov   #$32,old_motion
            mov   SPDMOTION,old_SPD
            bra   bitis

sag_geri1   clr   destina               ; ignore destination value
            clr   stop_flag
            jsr   oto_bw
            jsr   oto_right
            mov   #$41,old_motion
            bra   bitis

sol_geri1   clr   destina               ; ignore destination value
            clr   stop_flag
            jsr   oto_bw
            jsr   oto_left
            mov   #$42,old_motion

bitis       ldhx  pointer               ; Role start address
            aix   #6
            sthx  pointer
```

```
            jmp   backk               ; Continue

tamamm      jsr   stop
            jsr   dududut
            jsr   default
            cli
            lda   T2SC0
            mov   #$48,T2SC0
            rts

negg_3      jsr   hata
            bra   tamamm

new_speed
            bsr   cal_speed

            ldhx  old_speed
            cphx  speed
            bhs   yavas

hizlan      cphx  speed
            bhs   son1
            aix   #02
            sthx  T1CH0H              ; TUCATU moves forward until stop
            jsr   gecik
            bra   hizlan

yavas       cphx  speed
            bls   son1
            aix   #-01
            sthx  T1CH0H              ; TUCATU moves forward until stop
            jsr   gecik
            bra   yavas

son1        mov   SPDMOTION,old_SPD
            rts
cal_speed
            ldhx  #$0000
            lda   SPDMOTION
            sta   temp
            beq   atla_16
ekle16      aix   #$7F
            aix   #$7F
            dbnz  temp,ekle16
atla_16     sthx  Speed

            ldhx  #$0000
            lda   old_SPD
            sta   temp
            beq   atla_17
ekle17      aix   #$7F
            aix   #$7F
            dbnz  temp,ekle17
atla_17     sthx  old_speed
            rts
```

CHAPTER - 10

CONCLUSION AND RECOMMENDATION

The main goal of the project was the realization of a low cost, multipurpose robot. The second goal was the usage of MC6808 as much as possible. The cost of the project is less than 75 €. The flash capability of MC6808 is used for teaching process. The traveled path, speed, direction and steering wheel angle values are stored in Flash. In the playback mode TUCATU reads the trajectory information from Flash.

The following features are given to TUCATU:

- **Movement:**
    - Backward and forward motion: Direction control is provided by H-Bridge circuit.
    - Left and right motion: Rotation control is provided by a stepper motor.
    - Speed control: Increase and decrease by using of PWM methods.

- **Path Measurement** : An optical sensor is used for the measurement of traveled path.

- **Obstacle Detection** : An ultrasonic sensor is mounted on a stepper motor for the detection of obstacles front.

- **Training and Playback:** Teaching process is done by a TV remote control. During teaching mode, all trajectory information is stored in Flash. In the playback operation TUCATU gets this information from Flash.

- **IR Communication :** An IR communication facility between TUCATU and **the** remote control is provided.

- **Light Level Measurement :** TUCATU can measure the light level of environment and decides whether or not to turn on the head light

- **Warning and Signal Systems :** Warning and signal systems are features of TUCATU

The obstacle detection system can measure a distance of 10-100 cm. Port A is used for stepper motors, Port C for warning and signaling, Port D for motor control, Port B for ADC and ultrasonic sensor.

TUCATU Project may be considered as an integration of five projects:

1. Motion control in 8 directions
2. IR remote control
3. Training and playback
4. Light level measurement
5. Distance measurement

All these are designed and realized in this project.

During the whole study, mechanical, electrical, electronic designs have been done with what we had. Any professional item and help was not involved. From this point, the project may be assumed as an original engineering study and application.

In the development phase, we have some difficulties, especially in real time system design. We used MC6802 development kits which are used in micro computer laboratory in ITU for overcoming these difficulties.

CHAPTER - 11

REFERENCES

[1] Adalı, E. ***Mikroişlemciler Mikrobilgisayarlar,*** Birsen Yay. 1998

[2] Adalı, E. ***Gerçek Zaman Sistemleri,*** Sistem Yayıncılık. 1996

[3] ***M68HC08 Microcontroller Technical Data***, Motorola Inc 2002

[4] ***M68HC08 Microcontroller Reference Manual***, Motorola Inc 2002

[5] Wagner Lipnharski ***"Infrared",***www.ustr.net/infrared/infrared1.shtml, UST Reseach Inc. Orlando, Florida, 1999

[6] Berger Lahr, ***"Formulas + Calculations for Optimum Selection of Stepmotor".***