

To Design a Bipedal Robot

By

Hamdi bin Mohd Daud

A project dissertation submitted in partial fulfillment of
the requirements for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

December 2007

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

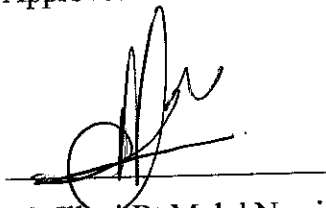
**To Design a
Bipedal Robot**

by

Hamdi Bin Mohd Daud

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Approved:



Ms Illani Bt Mohd Nawawi

Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK

December 2007

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

A handwritten signature in black ink, consisting of several overlapping strokes, positioned above a horizontal line.

Hamdi Bin Mohd Daud

ABSTRACT

The design of a bipedal robot literally involves walking gait and control of balance. Controlling the direction of balance for a two legged walking robot typically means mimicking the human form and its walking locomotion. The ultimate aim is to maintain an upright torso while advancing one leg in front of the other continuously without falling. The bipedal structure forces the writer to consider a combination of factors such as weight balancing, mechanism and degrees of freedom before designing the robot prototype. The writer has to apply fundamental knowledge about servo motor actuator, pulse width modulation, and PIC programming in accomplishing the project. Independent work through self-discipline, self-management and job co-ordination also needed to be exercise while undertaking the project. This report consists of five important chapters that cover the project introduction, review of related literature, project methodology, discussion of result, and finally the conclusion. The introductory part briefly discusses the background study of the project and the existing studies on designing a bipedal robot, more focus on the problem statement that covers problem identification and the significant of the projects, as well as the objectives of the project are stated and it describes more on the relevance and the feasibility of the project within the scope and time frame given. The review of related literature on the project is enclosed in chapter two while the project methodology is covered in chapter three of the report. Findings related to the project along with its discussions are stated in chapter four of the report together with the conclusion and references used during the research works.

ACKNOWLEDGEMENTS

In The name of Allah the Beneficent, the Merciful

An utmost gratitude to Ms Illani Bt Mohd Nawi for allowing the writer to do a Final Year Project under her supervisions. Even with a tight time schedule, support and good advices were always given which is the key factor of finishing the project in time. Without her relentless effort to guide and supervise, the project will not have been successful.

Also thanks to Dr Taj Mohammad Baloch as the student advisor for this project. His advice and recommendation for this project helped the writer to conduct this project smoothly.

Special thanks to Mr. Muhamad Aidil Jazmi for the references of his project as a guideline for the writer to understand and meet this project requirement. He has given numerous inspirations for the writer to carry on the project and been very helpful in assisting the writer in a lot of difficult time.

Last but not least, an utmost gratitude to father Hj Mohd Daud Bin Hj Yaakob and mother Hjh Hamidah Bt Mohamad which has brought the writer up and be good role models throughout these years.

Also the writer is very grateful to friends who have been supportive during the period of this project were held. Not to forget to all others who have help in the project directly and indirectly and might not be mentioned here.

Thank you for all the supports given and only god may repay them

TABLE OF CONTENTS

| | |
|--|----|
| LIST OF FIGURES | ix |
| LIST OF ABBREVIATIONS..... | x |
| CHAPTER 1 INTRODUCTION..... | 1 |
| 1.1 Background of study | 1 |
| 1.2 Problem Statement..... | 1 |
| 1.2.1 Problem Identification..... | 1 |
| 1.2.2 Significance of the Project | 2 |
| 1.3 Objective | 2 |
| 1.4 Scope of Study | 2 |
| CHAPTER 2 LITERATURE REVIEW | 3 |
| 2.1 Biped Walking | 3 |
| 2.1.1 Static Walking | 3 |
| 2.1.2 Dynamic Walking..... | 5 |
| 2.2 Degree of Freedom | 5 |
| 2.3 Mechanical Actuator..... | 7 |
| 2.3.1 Servo Motor..... | 7 |
| 2.4 Pulse Width Modulation | 11 |
| CHAPTER 3 METHODOLOGY | 12 |
| 3.1 Project Procedure | 13 |
| 3.2 Material | 13 |
| 3.3 Tools and Software | 14 |
| CHAPTER 4 RESULT AND DISCUSSION..... | 15 |
| 4.1 Bipedal Model Description..... | 15 |
| 4.1.1 Robot Design and Degree of Freedom | 15 |
| 4.1.2 Mechanical | 17 |
| 4.1.3 Electrical..... | 19 |
| 4.2 System Description | 21 |
| 4.2.1 Servo Controller | 21 |
| 4.2.2 Walking Controller..... | 25 |
| 4.3 Discussion | 26 |
| 4.3.1 Servo Controller C Program..... | 26 |

| | |
|---|----|
| 4.3.2 Walking Controller C Program | 28 |
| 4.3.3 Computer Instruction C Program | 29 |
| CHAPTER 5 CONCLUSION AND RECOMENDATION | 31 |
| REFERENCES | 32 |
| APPENDICES | 34 |
| Appendix A PROJECT GANTT CHART | 35 |
| Appendix B BIPEDAL ROBOT SCHEMATIC BOARD..... | 36 |
| Appendix C FUTABA S3001 STANDARD SERVO SPECIFICATION | 37 |
| Appendix D PIC 16F877 DATA SHEET | 38 |
| Appendix E MAX232I DUAL DRIVERS/RECEIVERS DATA SHEET | 39 |
| Appendix F SERVO CONTROLLER C PROGRAM..... | 40 |
| Appendix G WALKING CONTROLLER C PROGRAM..... | 44 |
| Appendix H COMPUTER INSTRUCTION C PROGRAM..... | 50 |

LIST OF FIGURES

| | |
|--|----|
| Figure 1 : Static walking and condition to be in stable position..... | 4 |
| Figure 2 : The projection of center of gravity in static walking..... | 4 |
| Figure 3 : The center of gravity in static walking (front view) | 5 |
| Figure 4 : Types of planar mechanical linkage & relation with Degree of Freedom | 6 |
| Figure 5 : Common servo motor..... | 7 |
| Figure 6 : Input pulse signal and servo motor output position | 8 |
| Figure 7 : Effect of duty cycle in pulse width modulation | 11 |
| Figure 8 : Flowchart of procedures done..... | 12 |
| Figure 9 : Obtained Futaba S3001 servos..... | 13 |
| Figure 10 : Obtained PIC 16F877 microcontroller..... | 14 |
| Figure 11 : Joints and links of the robot leg | 15 |
| Figure 12 : Degree of freedom in x-y-z plane | 16 |
| Figure 13 : 1mm L-shape stainless steel frame | 17 |
| Figure 14 : 10 obtained Futaba S3001 servos..... | 17 |
| Figure 15 : Bipedal Leg Prototype (Front view) | 18 |
| Figure 16 : Bipedal Leg Prototype (Rear view) | 18 |
| Figure 17 : Schematic board for controller with 2 PIC 16F877 | 19 |
| Figure 18 : Circuit testing for PCB implementation..... | 20 |
| Figure 19 : Printed Circuit Board layout for the circuit schematic | 20 |
| Figure 20 : Walking Controller and Servo Controller association | 21 |
| Figure 21 : PWM output from servo controller..... | 22 |
| Figure 22 : Servo controller program flowchart | 24 |
| Figure 23 : Flowchart for walking controller program..... | 25 |
| Figure 24 : Servo Controller pulse width modulation generator | 27 |
| Figure 25 : Servo variable representation on the bipedal robot..... | 28 |

LIST OF ABBREVIATIONS

AC - Alternate Current

DC - Direct Current

DOF - Degree Of Freedom

EEPROM - Electrically Erasable Programmable Read-Only

EIA - Energy Information Administration

MHz -Megahertz

ms - milliseconds

us - microseconds

PWM - Pulse Width Modulation

PIC - Programmable Intelligent Computer

PCB - Printed Circuit Board

RAM - Random Access Memory

+ve - Positive

-ve - Negative

CHAPTER 1

INTRODUCTION

1.1 Background of study

This project is about designing a biped robot that can walk. By definition, bipedal is standing or moving for example by walking, running, or hopping, with only two appendages which are typically legs. An animal or machine that usually moves in a bipedal manner is known as a biped, meaning "two feet" [12]. As for walking, one of the robot feet should be in front of another, with at least one foot on the ground at any time. This walking exercise usually is an active process, requiring constant adjustment of balance.

For nearly the whole of the 20th century, bipedal robots were very difficult to construct [12]. Robots which could move usually did so using wheels, treads, or multiple legs. Increasingly cheap and compact computing power, however, has made two-legged robots more feasible. The introduction of ASIMO [9], developed by Honda and Kondo Robot [10] by Kondo made us realize what we are capable of.

1.2 Problem Statement

1.2.1 *Problem Identification*

In recent years the interest to study the bipedal walking has grown and the demand for build bipedal robots has increase. Bipedal robots are more versatile than conventional quadruped or wheeled robots, but they tend to tip over easily. To solve this problem, the stability of a biped robot needs to be maintained during walking.

1.2.2 Significance of the Project

This project contributes to a lot variety of purposes, from the investigation of the theories on bipedal walking to the design of humanoid robot. Nowadays a lot of robots were design to help with human live. With an advance bipedal walking, robots can surpass the limitation faced before by walking steadily, climbing steps, become more versatile and reach further more by evolving to humanoid robot. The design also can be used for disable person that lost their freedom to move freely and improve their living.

1.3 Objective

Generally the main objective of this project is to design a prototype of a bipedal robot using a PIC microcontroller. The robot should be able to walk, tilt and bend the two legs given. The bipedal robot must be controlled using suitable PIC microcontroller to achieve the objective mentioned. The specifications of the prototype are that it shall be around 0.5 meter in height covered from feet to waist, mounted microcontroller circuit programmed to perform the task required by the robot to walk without falling. In addition the PIC microcontroller can also be programmed to read any sensor that is required to accomplish the goal. The fully working prototype can be reconfigured to follow certain order, such as avoiding any blockage or even dancing.

1.4 Scope of Study

After consideration of all the necessary procedure and problems that might occurs, this project has been divided into two categories, which are hardware development and software development. The hardware covers the entire prototype mechanical and electrical component while the software mainly focuses on the system architecture or microcontroller programming. In the first semester, the author will be involved mainly in hardware research. For the second semester, the author will continue on the hardware and software development. Experimentation and testing is necessary to avoid any project setback. Good project management technique and usage of time constraint efficiently must be adopted in order to complete the project.

CHAPTER 2

LITERATURE REVIEW

In order to obtain relevant and beneficial information regarding this project, the author carried out some relevant literature review, such as by referring journals, websites and etc. The information is very important for the development of the system as it provides theories, concepts as well as techniques that be utilized throughout this project.

2.1 Biped Walking

In order to understand the mechanical bipedal robots mechanics design, is necessary first to understand the biped walking process or biped locomotion. This area has been studied for a long time, but until this past few years, new generation of robots that walk on two legs were introduced thanks to the fast development of computers and microcontroller.

First, there were robots that used static walking. The control architecture had to make sure that the projection of the center of gravity on the ground was always inside the foot support area. This approach is the basic and the first step throughout bipedal development.

2.1.1 *Static Walking*

Static walking assumes that the robot is statically stable [8]. This mean that, at any time, if all motion is stopped the robot will stay indefinitely in a stable position. It is necessary that the projection of the center of gravity of the robot on the ground must be contained within the foot support area (see Fig. 1, 2 and 3). This condition can be achieved by widening the support area of the foot surface

to support the whole body weight or increasing the foot weight to support the balance of the upper body. Also, walking speed must be low so that the inertia forces are negligible.

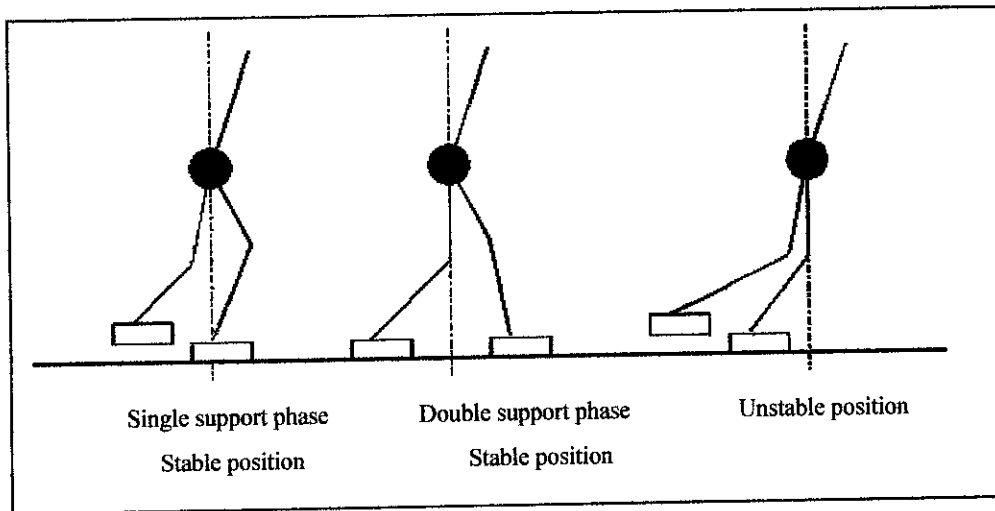


Figure 1 : Static walking and condition to be in stable position [8]

Biped with static walking requires large feet, strong ankle joints and can achieve only slow walking speeds. The technique adopted is based on the principle that the important aspect in the static walking locomotion is not the perfect respect of a given trajectory, but the displacement of the body from one point to another without falling.

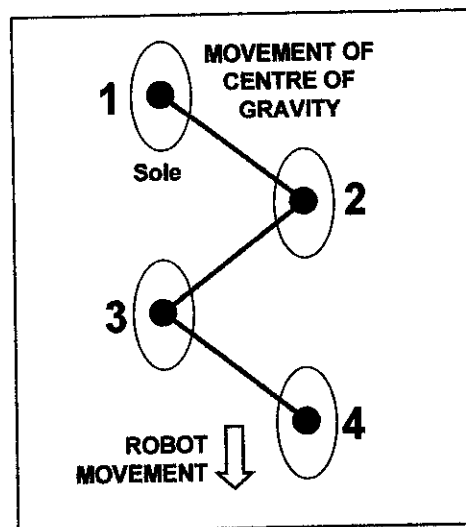


Figure 2 : The projection of center of gravity in static walking [7]

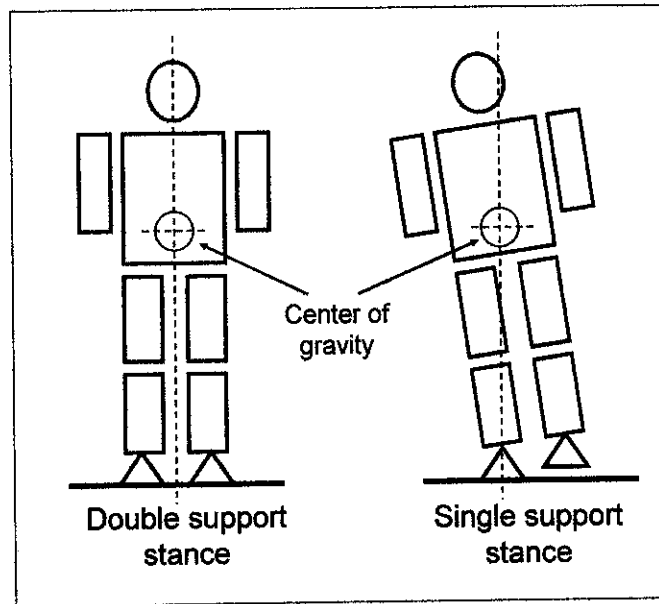


Figure 3 : The center of gravity in static walking (front view).

2.1.2 *Dynamic Walking*

Dynamic walking of a biped allows the center of gravity of the robot to be outside the support region for limited amounts of time. There is no absolute criterion that determines whether the dynamic walking is stable or not. Indeed a biped can be designed to recover from different kinds of instabilities, depending on the walking surfaces. However, dynamic walking condition can only be achieved if the robot has active ankle or knee joints that are used to stabilize the robot body along with a stabilizer sensor [8], which required a lot of funding in research. To keep this research in adequate budget, the student will only limit the study on static walking.

2.2 **Degree of Freedom**

A system with several body parts would have a combined Degree Of Freedom (DOF) that is the sum of the DOFs of the bodies, less the internal constraints they may have on relative motion. The term degree of freedom is used to describe the number of parameters needed to specify the spatial pose of a linkage [13].

Mechanical linkages are a series of rigid links connected with joints to form a closed chain, or a series of closed chains. Each link has two or more joints, and the joints have various degrees of freedom to allow motion between the links. A linkage is called a mechanism if two or more links are movable with respect to a fixed link. Mechanical linkages are usually designed to take an input and produce a different output, altering the motion, velocity, acceleration, and applying mechanical advantage.

The most common linkages have one degree of freedom, meaning that there is one input that produces one output motion. Most linkages are also planar, meaning all the motion takes place in one plane. Spatial linkages (non-planar) are more difficult to design and therefore not as common.

Kutzbach-Gruebler's equation is used to calculate the degrees of freedom of linkages. The number of degrees of freedom in a linkage is also called mobility. Figure 4 shows a simplified version of the Kutzbach-Gruebler's equation for planar linkages:

$$m = 3(n - 1) - 2j$$

m = mobility = degrees of freedom

n = number of links (including a single ground link)

j = number of mechanical joints (pin or slider joint)

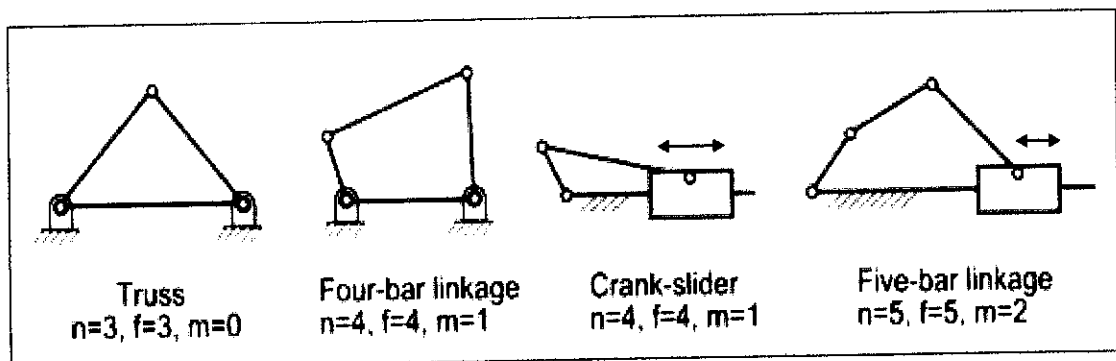


Figure 4 : Types of planar mechanical linkage and relation with Degree of Freedom [13]

2.3 Mechanical Actuator

Electrical actuators are required in any bipedal robot design for body part movements. Electrical actuators are electro-mechanical hardware such as solenoids and servo motors. Depending on the mode of powering and controlling, they can be controlled directly by a computer platform attached to a servo controller board or autonomously by a dedicated PIC. The main principle behind every electrical actuator is that motion is induced by the application of an electrically created magnetic field to a ferrous core. The strength and direction of the magnetic field determines the speed and direction of rotation or motion.

2.3.1 Servo Motor

Servo motors are geared dc motors with positional control feedback and are used for position control. The shaft of the motor can be positioned or rotated through 180 degrees. They are commonly used in the hobby market for controlling model cars, airplanes, boats, and helicopters.

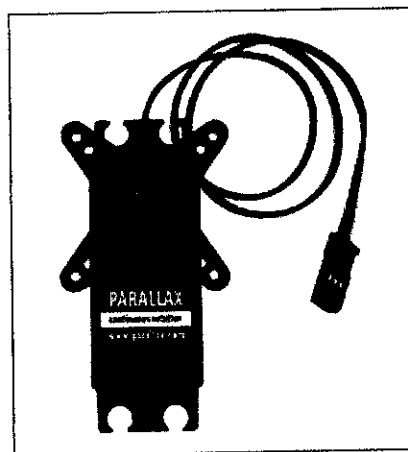


Figure 5 : Common servo motor [11]

Servo motors are available in a number of stock sizes. Standard servo typically has 3 wire outlets, which is differentiated by 3 different colors varies to particular manufacturers. Ideally two of them are for power, ranging from 4 to 6 volts and ground. The third wire feeds a position control signal to the motor, and the control signal is a variable-width pulse.

Servo Motor Control

The servo motor has some control circuits and a potentiometer that is connected to the output shaft [14]. The amount of power applied to the motor is proportional to the distance it needs to travel. So, if the shaft needs to turn a large distance, the motor will run at full speed. If it needs to turn only a small amount, the motor will run at a slower speed. This is called proportional control. The angle is determined by the duration of a pulse that is applied to the control wire. This is called Pulse Coded Modulation. A neutral, midrange positional pulse is a 1.5ms pulse, which is sent 50 times a second (20 ms period) to the motor. This pulse signal will cause the shaft to locate itself at the midway position of 90 degrees. The shaft rotation on a servo motor is limited to approximately 180 degrees (± 90 degrees from center position). A 1ms pulse will rotate the shaft all the way to the left, while a 2ms pulse will turn the shaft all the way to the right. By varying the pulse width between 1 and 2 ms, the servo motor shaft can be rotated to any degree position within its range.

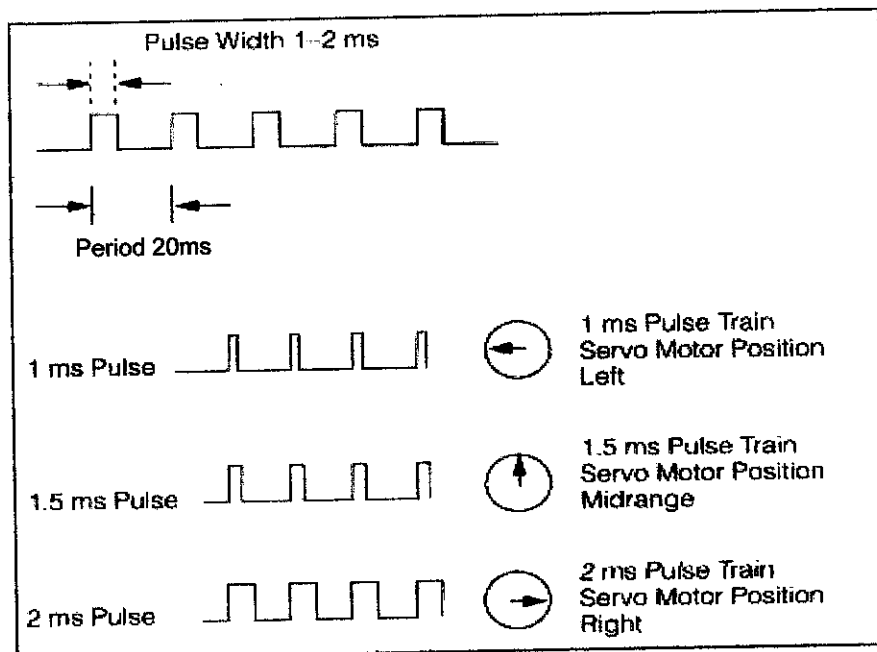


Figure 6 : Input pulse signal and servo motor output position

Servo Motor Selection

In choosing a suitable servo motor for this project, various servo motor specifications were researched. Servos are distinguished by a number of aspects and performance features that should be considered.

Price

In this project, at least 10 servos for 10 degree of freedom are needed to make the robot walk. To avoid short in budget, it is suggested that the price of each servo is lower than RM50 so that the total cost of the servos will not exceed RM500.

Torque

Torque delivered by the servo depends on the size and weight of the robot. The servo needs to have enough power to do the job that is needed. With the earlier robot design, it is predetermine that servo with 3.5kg/cm torque should be able to lift a 500gram leg. Lighter material is needed to reduce the weight of the robot to minimize the torque required including the weight of the servo motor and circuits.

Weight

More powerful motors tend to be heavier and larger. To keep the robot light, the servos needed to be light but delivered enough power to move the joints of the robot. To minimize the mass of the robot, total weight of all 10 servo needed is less than 500 gram, which is less than 50 gram each.

Speed

The speed of the servo refers to the maximum speed at which it can turn. This is usually measured in seconds per 60°. If it is 0.5 seconds

per 60°, then it will take 1.5 seconds to turn 180°. This feature is very important in static walking as the robot needed to be balanced all the time and operate in specific speed. However the speed of the servo can be adjusted in controller programmer

Voltage

A number of servos and other controller-compatible devices operate at different voltages, such as 4.8, 6.0 or 7.2 volts. To avoid complication in power supply circuit, it is very useful to obtain servos that run at the same voltage, which is possibly at 6V.

Gear Type

There are plastic, metal, and titanium gear types available. The nylon gear is adequate for lower torque. It is very important not to force the servo to overrun its limitation as it can damage the gear of the servo.

Size

The servo to act as the wrist or knee of a legged robot, then its size may be an issue. If the servo is small indeed, it is possible to install it directly to the knee joint and if the servo is too big, mounting it at the trunk of the robot is most likely and using something along the lines of tendon-like wires to transmit its motion.

The scope of this project is to use PIC microcontroller and the student will limit the electronic controllers for the electrical actuators to those that use pulse width modulation, which can be generate easily by microcontroller. The pulse width modulation technique enables a capable electrical motor to adjust speed and direction of rotation based on the width of pulses received

2.4 Pulse Width Modulation

Pulse width modulation (PWM) of a signal or power source involves the modulation of its duty cycle, at constant frequency. The pulse duration can be varied by changing the duty cycle as shown in Figure 7 below. This type of modulation can be used to control such servo since it will respond to desired pulse duration.

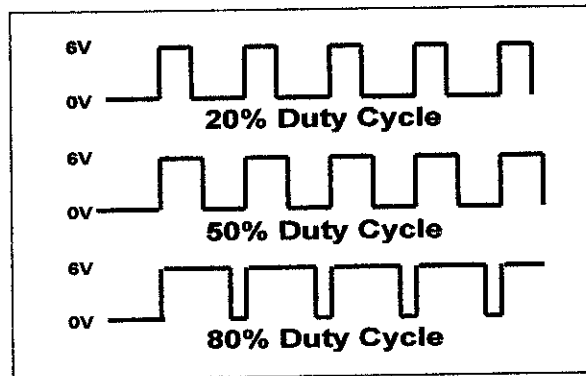


Figure 7 : Effect of duty cycle in pulse width modulation

CHAPTER 3
METHODOLOGY

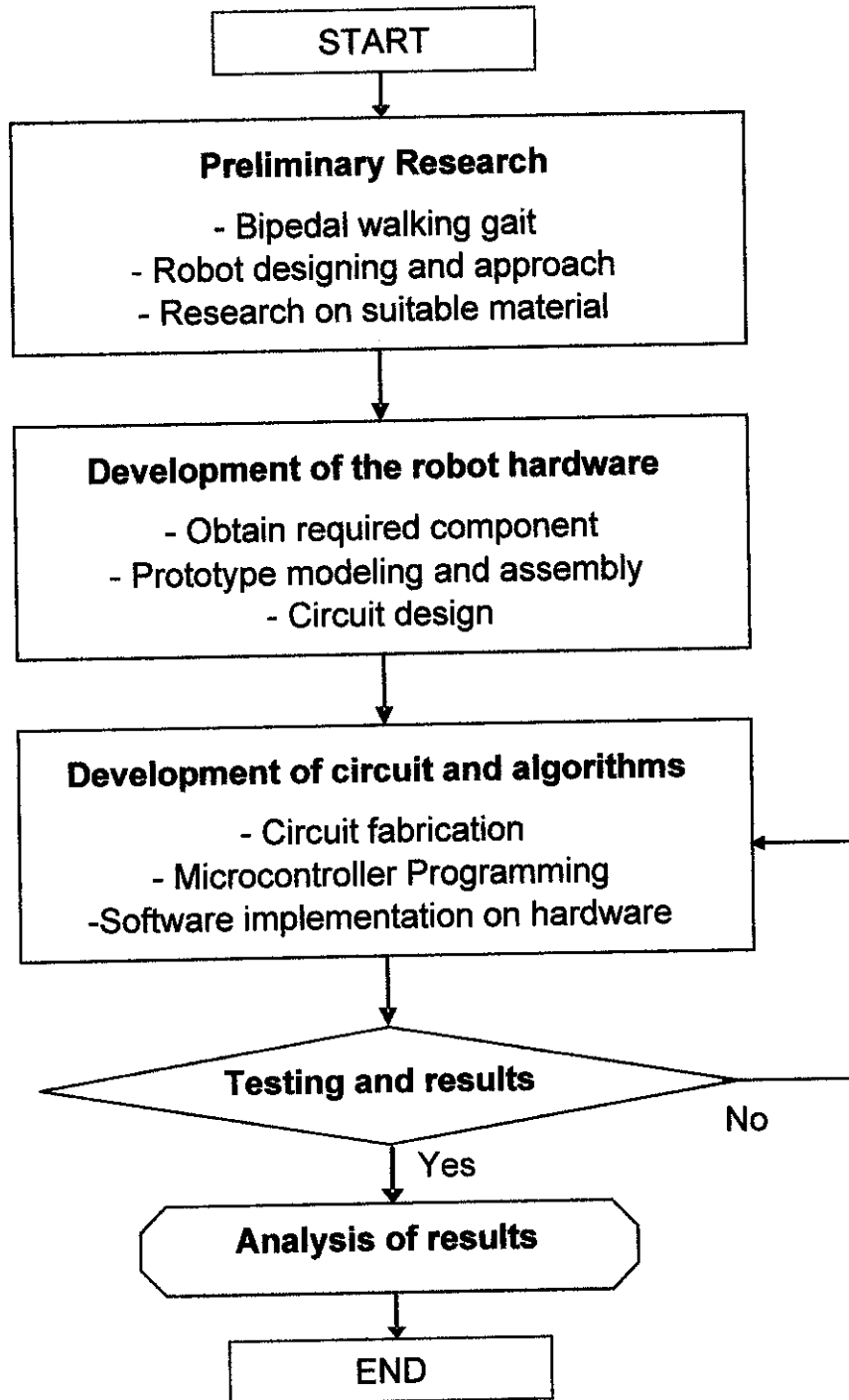


Figure 8 : Flowchart of procedures done

3.1 Project Procedure

Researches on walking gait and suitable hardware to be implemented on the biped robot were done as a preliminary step before making any decision toward robot development and construction. The structure of the robot, including the leg, foot, actuator, controller and electrical circuit are identified as hardware while the programming language to control the robot movement is known as software. After all the required hardware components were obtained and assembled together to construct the bipedal robot prototype, the next step is to design the controller circuit using suitable PIC microcontroller. Once the controller is designed and fabricated, software development was continued with the controller programming. The next step is to implement the programming into the controller by connecting all the inputs and outputs connection and a few configurations were verified. Then experiment shall be performed to test the prototype functionality and to identify any occurring problem. If the prototype is not operational, modification on programming and circuit test have to be done. Finally the result of the functional prototype will be analyzed to maximize the performance.

3.2 Material

Ten Futaba S3001 servos were successfully obtained for the project as shown in figure 9. All the servos were ordered from Singapore through a Kyosho Toykar hobby shop located at Berjaya Times Square, Kuala Lumpur.

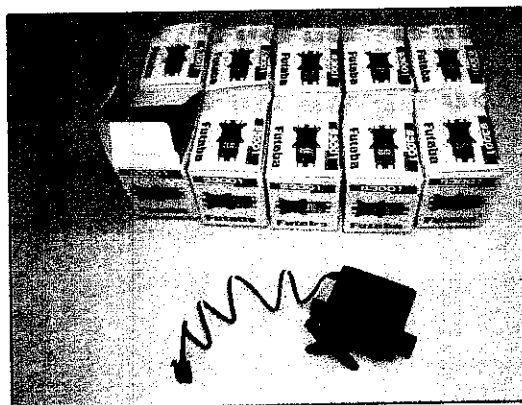


Figure 9 : Obtained Futaba S3001 servos

1mm L-shape stainless steel frames were obtained from local hardware store. Two PIC 16F877 microcontrollers were obtained from State Electronic Trading, a local electronic store at Ipoh, Perak. Other electronic parts required for the project such as capacitors, crystal oscillator, wires, male connector and circuit board were obtained from the Electronic Storage Room in the Electrical Department.

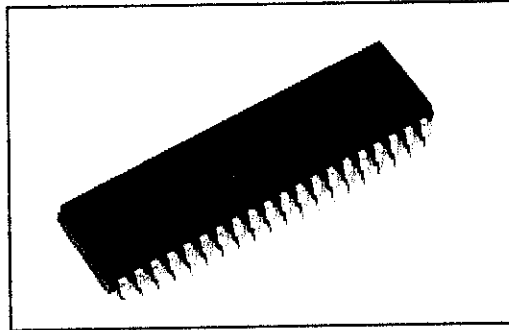


Figure 10 : Obtained PIC 16F877 microcontroller

3.3 Tools and Software

This project required the student to build the working prototype of the bipedal robot. To achieve this, several workshop tools are required to build the body of the robot such as measurement tapes, screwdriver, pliers, saw, cutting tool and drill. The printed circuit board (PCB) for the controller circuit of the robot was fabricated in the PCB fabrication Lab and Electronic Lab was used to access several tools such as multimeter, oscillator, power supply, welding tool and etc. The circuit PCB layout was drawn using EAGLE layout editor version 4.13 and the program was compile into the controller using Microchip MPLAB IDE version 7.40 software which is available in the Microprocessor Lab.

CHAPTER 4

RESULT AND DISCUSSION

4.1 Bipedal Model Description

4.1.1 Robot Design and Degree of Freedom

After considering all the movement needed from the biped locomotion and static walking, the robot leg can be design and structured. There will be 10 main joints connecting the whole robot leg at the torso, hip, knee and ankle as shown in figure 11. The torso will link both torso joint for sideways movement. There would be 2 separate links connecting to each hip joint. In each leg, the hip joint will be connected to the knee joint by a link at the thigh. The shank will link the knee joint to the ankle joint. Notice that there will be 2 joints in each ankle which can be linked by a small body part. These 2 joints acts in different direction, one for displacement to the front and the other is to the side to redirect the center of gravity of the upper body.

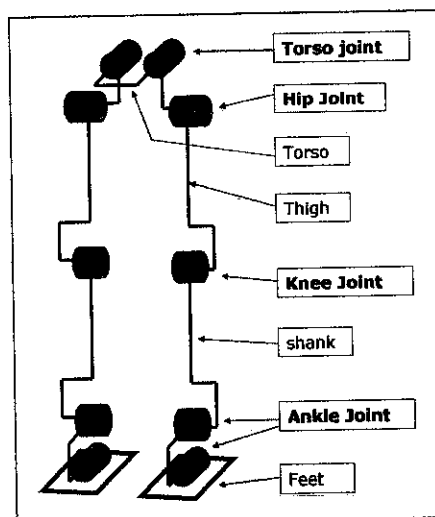


Figure 11 : Joints and links of the robot leg

Referring to figure 11 the robot leg shall have 10 mechanical joints that will be actuated by 10 servos and linked by 11 body part which are the torso, 2 thighs, 2 shanks, 2 ankles and also the foot. To find the degrees of freedom of the bipedal robot, Kutzbach-Gruebler's equation of a planar mechanical linkage was used to calculate the DOF, where total joints are 10 and links are 11;

m = mobility = Degree of Freedom

$n = 11$ = number of links (including a single ground link)

$j = 10$ = number of mechanical joints (pin or slider joint)

$$m = 3(n - 1) - 2j$$

$m = 10$ DOFs

The robot legs shall be able to move in 5 degrees of freedom in each leg, as shown in figure 12. Each leg must have 3 DOFs respectively to the x-y plane at the hip, knee and ankle joints. These 3 joint in each leg will mostly used to shift the displacement of the body forward. Meanwhile 2 DOFs in y-z plane in each leg at the torso and knee joint are use to shift the body sideway for balancing purpose. As conclusion, the bipedal robot shall have total of 10 DOFs in x-y-z plane respectively.

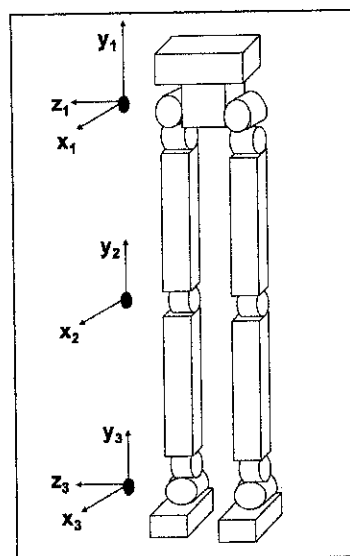


Figure 12 : Degree of freedom in x-y-z plane

4.1.2 Mechanical

Ten servos and 1mm L-shape stainless steel frame were used to construct the bipedal robot leg. The stainless steel was measured and divided into 11 parts before it were cut, bended and shaped to fit accordingly to all the servos, links and joints. The stainless steel frames that were formed were shown in figure 13.

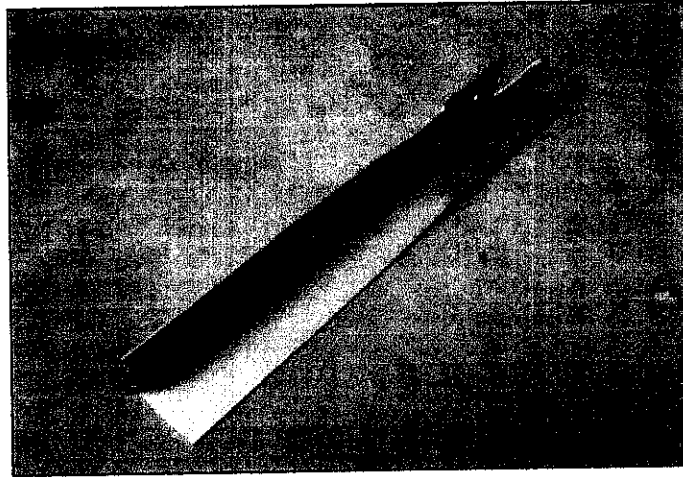


Figure 13 : 1mm L-shape stainless steel frame

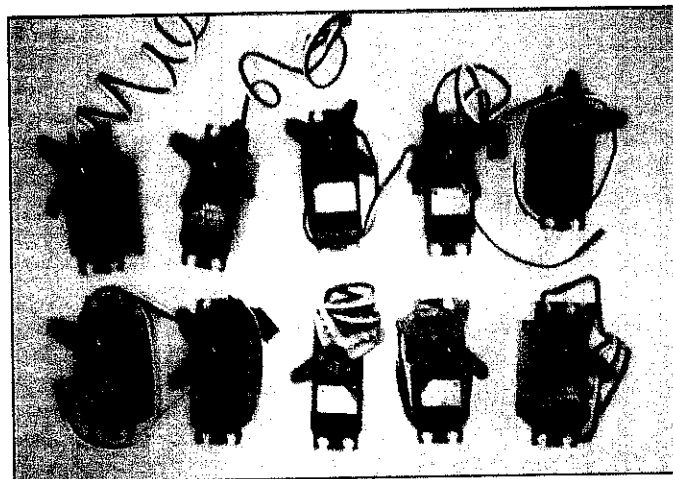


Figure 14 : 10 obtained Futaba S3001 servos

The entire stainless steel frame was then linked together with the obtained Futaba servos as shown in figure 14 using screws, nuts and washers. All the body parts was then assembled together to produce the bipedal robot prototype as shown in figure 15 and 16.

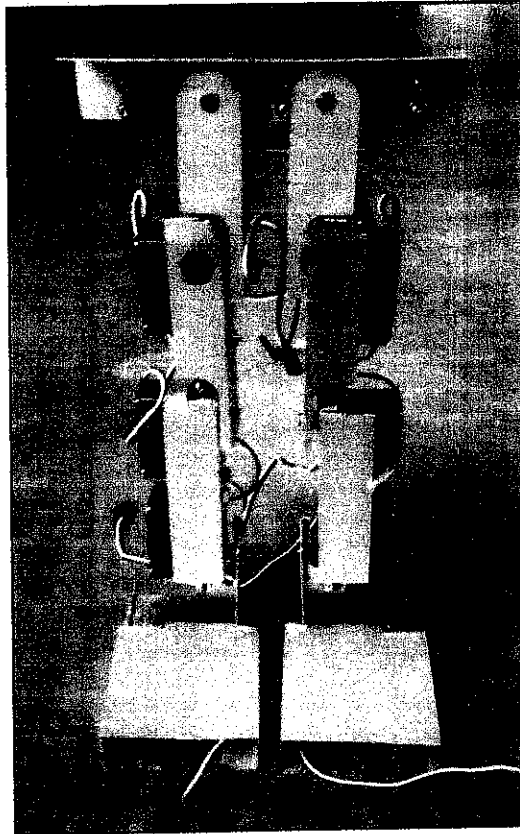


Figure 15 : Bipedal Leg Prototype (Front view)

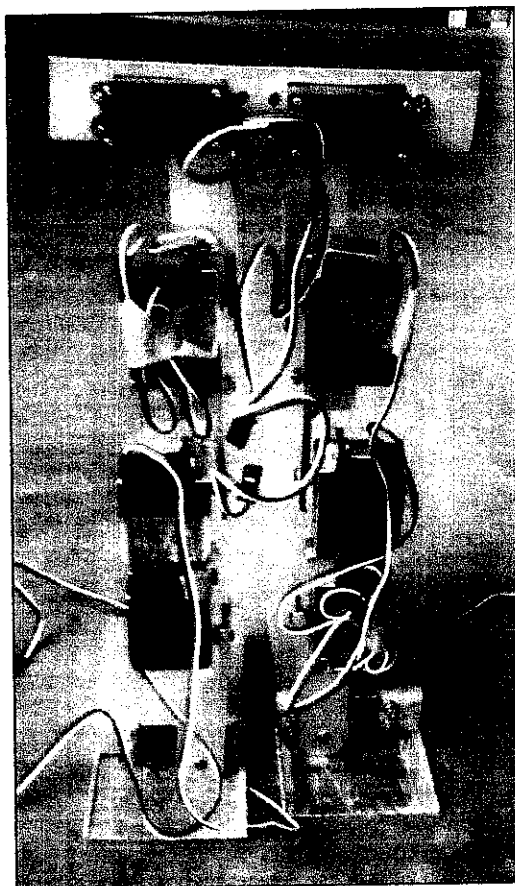


Figure 16 : Bipedal Leg Prototype (Rear view)

4.1.3 Electrical

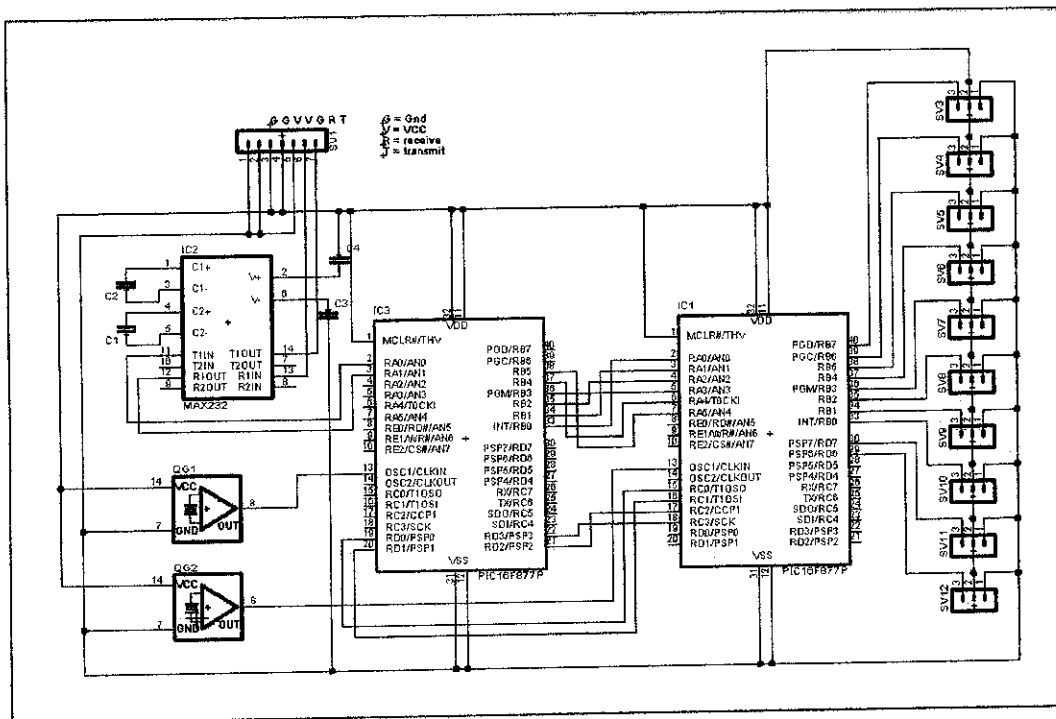


Figure 17 : Schematic board for controller with 2 PIC 16F877

The controller circuit board utilized two PIC 16F877, one is known as Servo Controller and the other is Walking Controller as shown in figure 17. The Servo Controller is connected directly to the servos and handles the task of controlling the 10 servos. It will read the output instruction from the Walking Controller and continuously update the position for each servo at a time. However the Walking Controller will not have to worry about the PWM output controlling the servo but instead focuses on walking timing and patterns that need to be achieved. To comply with this condition, different clock timing for each PIC have to be introduced because if both PICs have the same clock timings they were unable to communicate with each other. Different clock timing will gift PIC2 enough time to retrieve data from PIC1. A 4 MHz crystal oscillator is installed to the Servo Controller while the Walking Controller will use an 8 MHz oscillator. For the receiver, MAX232 chip manufactured by MAXIM is used. The MAX232 is a dual driver (transmitter and receiver) that includes a capacitive voltage generator to supply standard Energy Information Administration (EIA)-232 voltage levels

from a single 5V supply. The receiver converts EIA-232 inputs to $\pm 5V$ TTL or CMOS levels, which to be sent to the controller from the computer.

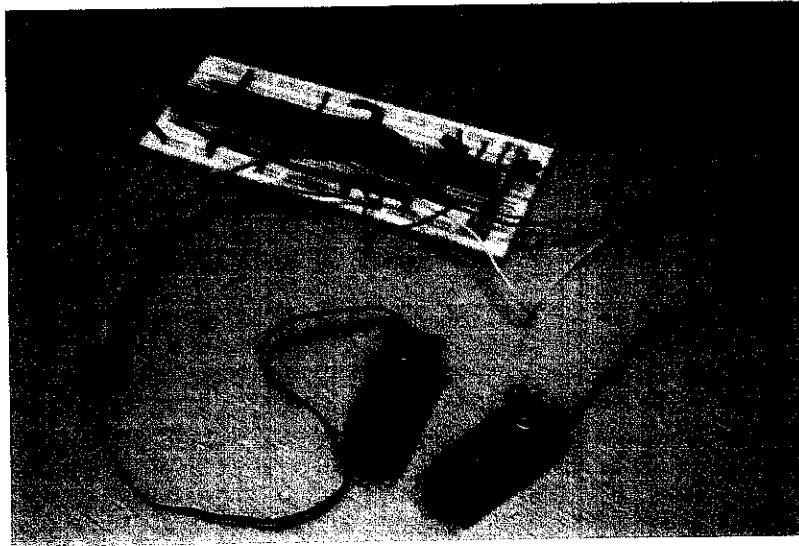


Figure 18 : Circuit testing for PCB implementation

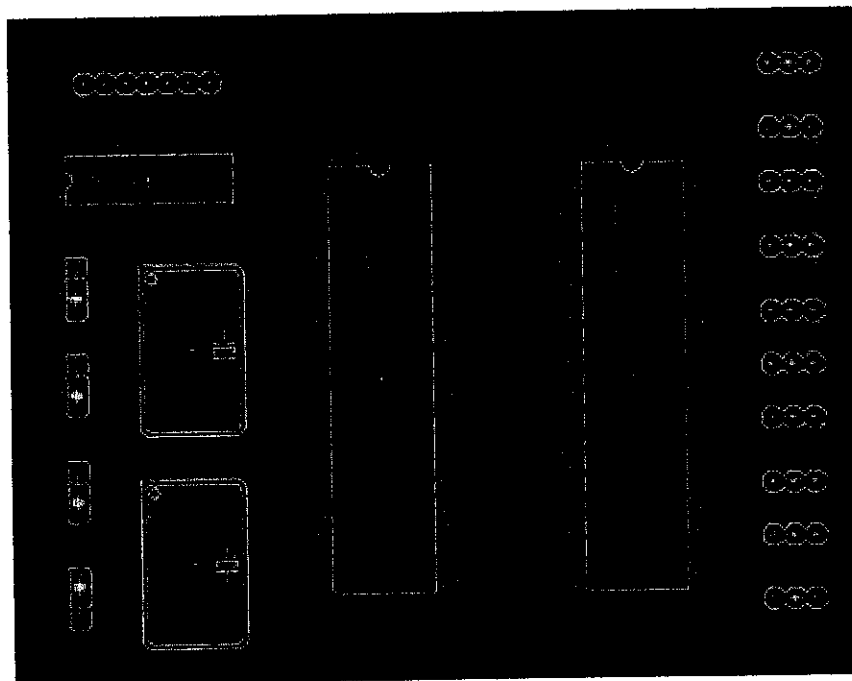


Figure 19 : Printed Circuit Board layout for the circuit schematic

The Printed Circuit Board layout as shown in Figure 19 was generated from the schematic board, used to fabricate a Printed Circuit Board where the electronic components will be installed and welded. The circuit was tested before implementation on the printed circuit board as shown in Figure 18. The circuit will be then mounted on the bipedal robot and connects to the servos.

4.2 System Description

System controlling the bipedal robot is divided into two parts which are the Servo Controller and the Walking Controller. The Servo Controller is connected directly to the entire servos while generating Pulse Width Modulation to vary the servo position. The Walking Controller selects the servo to be controlled and instruct the servo to achieve the required angle or position. This will produce a specialization in each controller with each of them performing the previous stated task, shown in Figure 20.

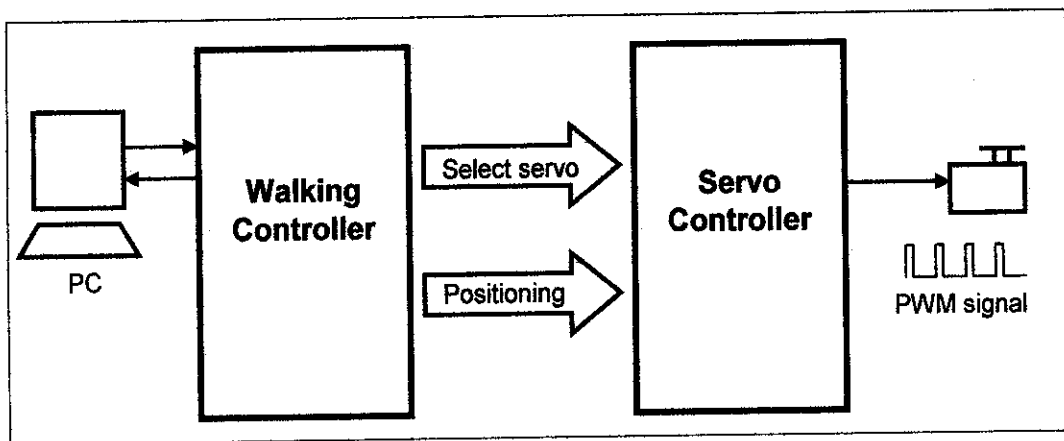


Figure 20 : Walking Controller and Servo Controller association

With this arrangement, the Servo Controller will only have to focus on generating the required PWM to the servo while the Walking Controller will only have to worry about the walking gait. On the other hand by using this configuration, the programming code will also be reduced significantly. This is due to elimination of instruction set on servo selection, positioning and memory allocation.

4.2.1 Servo Controller

Servos are control using pulse width modulation (PWM) to allocate the desired position which is easy to produce by the microcontroller. Maximum pulse width to control a servo is 2ms and the minimum pulse width of 1ms with the frequency of 50Hz. Frequency of 50Hz will produce 20ms for each period. To avoid overlapping among the servos, a 2ms pulse slot is always allocated for

each servo so that other servo pulse width will never overlap with each other even if the pulse width for a servo does not reach 2ms. Meaning with a maximum pulse width of 2ms for each servo, there would be 18ms of idle pulse width is divided to other nine servos with different time allocation. Figure 21 may ease the comprehension.

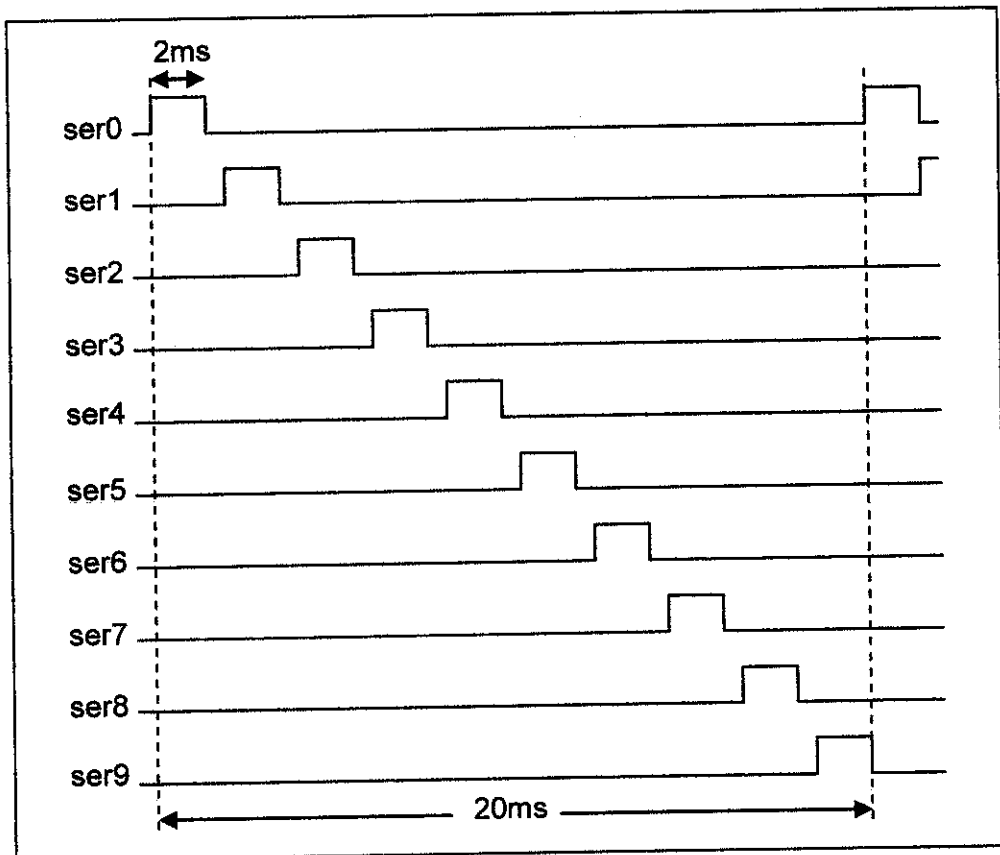


Figure 21 : PWM output from servo controller

This pattern is use so that one servo can be processed at a time and simplifies the walking algorithm. With PWM, the duration of a pulse can be varies from 1ms to 2ms to control the servo movement.

To achieve the PWM output from the controller, servo program was written as shown in Figure 22 explaining the servo controller program flow. There are ten globally declared variable servos 0 to 9 that stores the required servo position, these variables are retrieves from the Walking Controller output and updated to allocate the desired pulse width. The controller will update only one servo at a

single time. Programming on the servo PIC was finalized first before working on the walking control PIC. In this project, the ranges of motion of the servo are not the full 180 degrees but instead only 90 degrees.

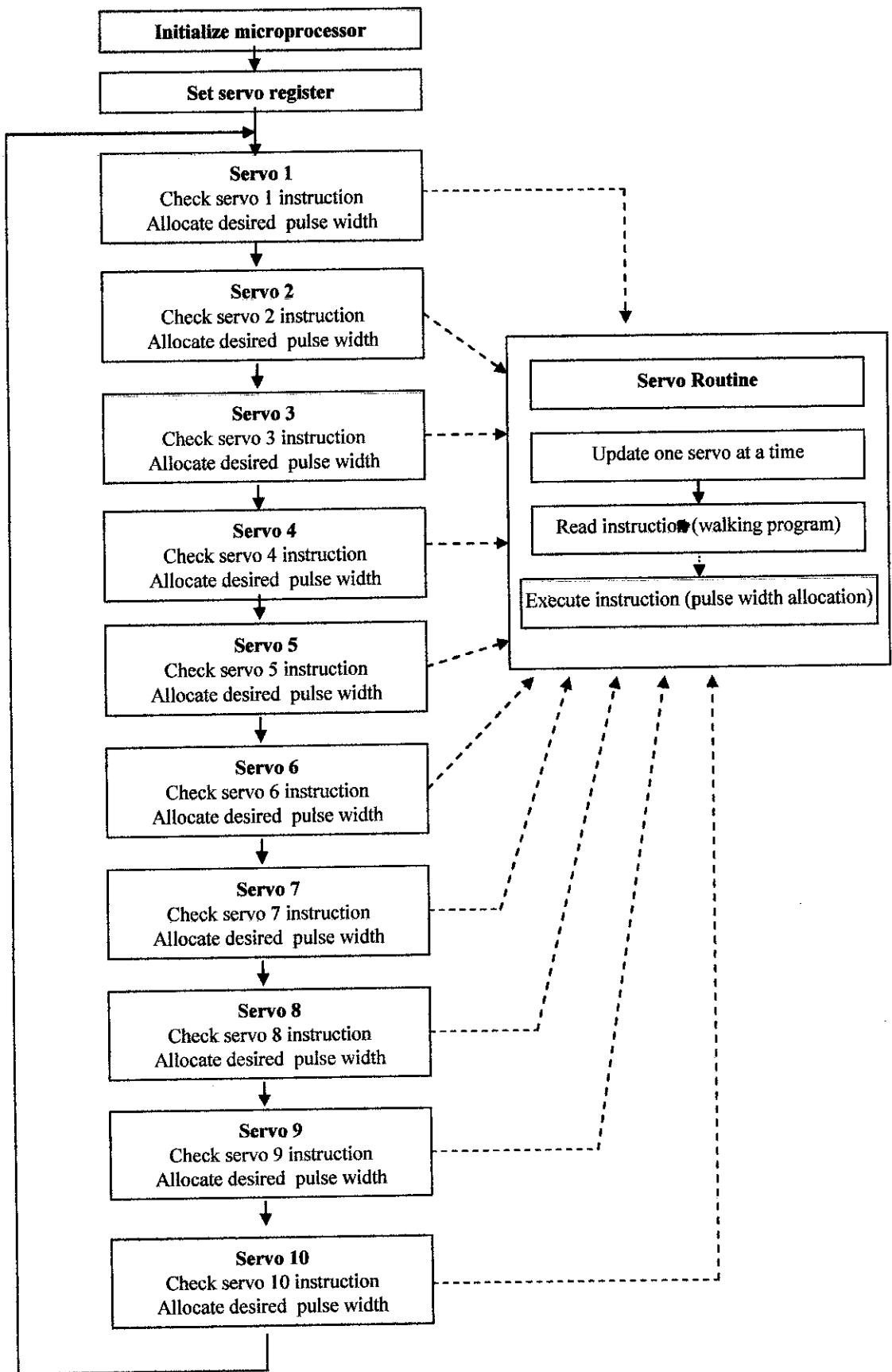


Figure 22 : Servo controller program flowchart

4.2.2 Walking Controller

The control PIC will not have to worry about PWM controlling the servo but instead focuses on walking timing and patterns that need to be achieved. The controller indicates which servo to operate at a time and instruct the servo controller what to do later. Figure 23 below shows the simplified flowchart for walking controller.

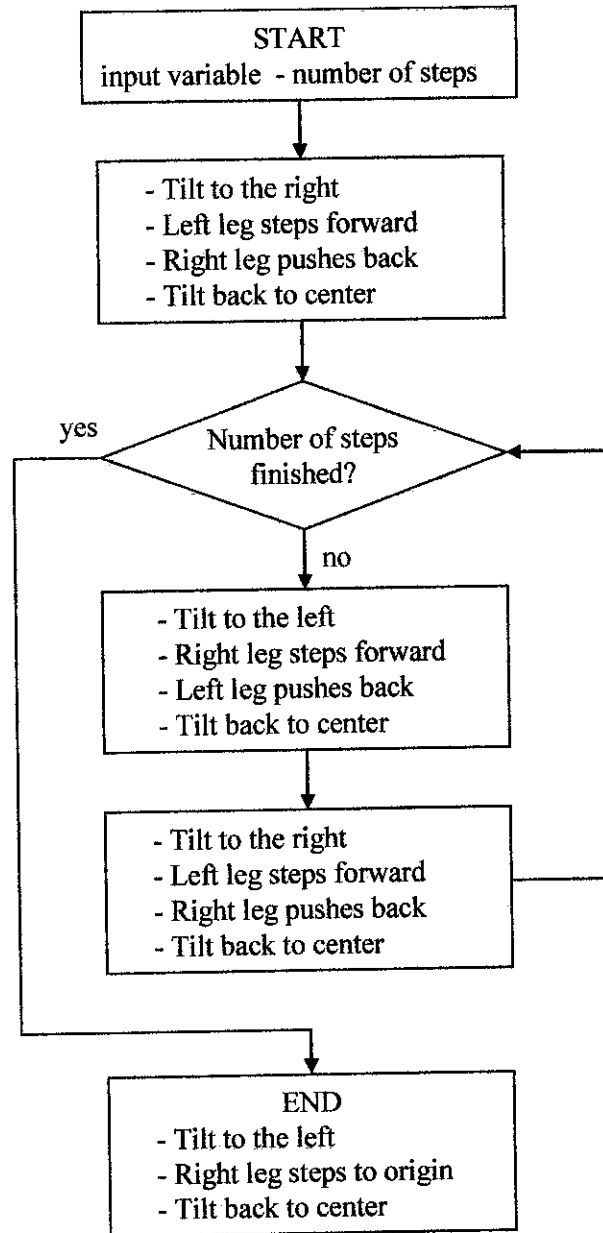


Figure 23 : Flowchart for walking controller program

4.3 Discussion

4.3.1 Servo Controller C Program

By using this program code, the controller will read the data received from PORT C input to determine which servo to be control at a single time and then retrieve the desired position for the servo from PORT A. The Servos 0 to 9 are set as integer variable serv0 to serv9 accordingly, which represent a 6 bit (64 levels) position value. This mean servo position value can be varies from 0 to 63 levels.

```
int serv0,serv1,serv2,serv3,serv4,serv5,serv6,serv7,serv8,serv9;    //Set integer
void update()                //Update function
{
  int ser,pos;                //Set servo and position as integer
  ser=PORTC;                  //Read data from PORTC and determine servo
  pos=PORTA;                  //Read data from PORT A for position
  if(ser==0){serv0=pos;}     //Select servo 0 if 0 and put position value in serv0
  else if(ser==1){serv1=pos;} //Select servo 1 if 1 and put position value in serv1
  else if(ser==2){serv2=pos;} //Select servo 2 if 2 and put position value in serv2
  else if(ser==3){serv3=pos;} //Select servo 3 if 3 and put position value in serv3
  else if(ser==4){serv4=pos;} //Select servo 4 if 4 and put position value in serv4
  else if(ser==5){serv5=pos;} //Select servo 5 if 5 and put position value in serv5
  else if(ser==6){serv6=pos;} //Select servo 6 if 6 and put position value in serv6
  else if(ser==7){serv7=pos;} //Select servo 7 if 7 and put position value in serv7
  else if(ser==8){serv8=pos;} //Select servo 8 if 8 and put position value in serv8
  else if(ser==9){serv9=pos;} //Select servo 9 if 9 and put position value in serv9
}
```

The next program code is the pulse width modulation generator for the Servo Controller. The program will retrieve the input variable serv0 to serv9 and calculate the pulse width to be given to the selected servo.

```
//servo 0
update();                //Go to Update function
del1=serv0*4;           //Delay 1 value (pulse duration delay)
del2=255-del1;         //Delay 2 value (delay for neutral)

PORTB=0x80;            //Output to servo 0 at RB7
PORTD=0x00;           //No output at PORT D
delay_us(980);        //Minimum pulse width of 0.98ms
delay_us(del1);       //Use Delay 1 to delay the pulse width
delay_us(del1);       //Use Delay 1 to delay the pulse width
delay_us(del1);       //Use Delay 1 to delay the pulse width
```

```

delay_us(del1);           //Use Delay 1 to delay the pulse width
PORTB=0x00;              //No output to servo 0
PORTD=0x00;              //No output at PORT D
delay_us(del2);          //Use Delay 2 to delay for 0 amplitude
delay_us(del2);          //Use Delay 2 to delay for 0 amplitude
delay_us(del2);          //Use Delay 2 to delay for 0 amplitude
delay_us(del2);          //Use Delay 2 to delay for 0 amplitude

```

Figure 24 below simplify the configuration on how the program will determine and locate pulse width for servo 0 and servo 1. Note that *del1* which varies from 0 to 1008 represent the enlarged input variable *ser0* to *serv9* that varies from 0 to 63 (amplification of 4). To maintain a 2ms period gap between each servo, output bit 0 will be delay 4 times by $255-del1$, represent by *del2*.

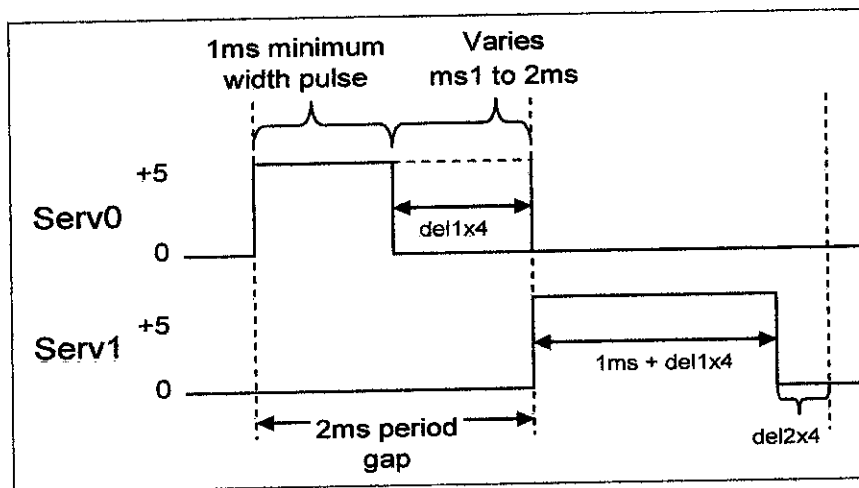


Figure 24 : Servo Controller pulse width modulation generator

4.3.2 Walking Controller C Program

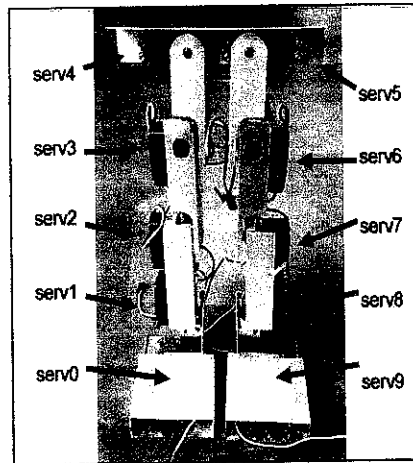


Figure 25 : Servo variable representation on the bipedal robot

Figure 24 shows the servo variable representation for each servo located at the bipedal robot. The program below is for tilting purpose which is to put the center of gravity of the whole robot body into either one foot. The tilt process only involves 4 servos which are ser0, ser4, ser5 and ser9. These servos will shift the body of the robot to the side, either right or left. To tilt to the left, ser0 and ser 9 current pulse widths will be reduced to rotate the shaft of the servo in anticlockwise direction. Meanwhile ser4 and ser5 current pulse width will be increased to rotate the shaft of the servo in clockwise direction. Actions on these 4 servos will shift the torso of the robot body to the left resulting the left foot to accumulate the center of gravity of the robot. Like wise for tilting to the right, the process will be the same but only with the opposite shaft rotation direction.

```

void ser0459(int op,int del)           //ser0459 function for tilt
                                        //set integer op for operation and del for level
{
    if(op=='+')                         //select if operation is + or tilt to the left
    {
        ser0=ser0-del;                 //decrease current servo 0 position by del level
        ser4=ser4+del;                 //increase current servo 4 position by del level
        ser5=ser5+del;                 //increase current servo 5 position by del level
        ser9=ser9-del;                 //decrease current servo 9 position by del level
    }
    else if(op=='-')                   //select if operation is - or tilt to the left
    {
        ser0=ser0+del;                 //increase current servo 0 position by del level
        ser4=ser4-del;                 //decrease current servo 4 position by del level
        ser5=ser5-del;                 //decrease current servo 5 position by del level
        ser9=ser9+del;                 //increase current servo 9 position by del level
    }
}

```

The next program is to keep the feet and torso at the same plane or parallel to each other, even if the leg is lift up. This action is very important for the robot to have smooth landing during walking by directing the foot to be parallel to the ground. This routine will be updated every time when servo 6 and servo 7 moves.

```

void serv8()                //serv8 routine
{
    signed pos6,pos7,temp;   //signed integer pos6,pos7 and temp
    pos6=sc6-ser6;          //pos6 is servo 6 center position – current position
    pos7=sc7-ser7;          //pos7 is servo 7 center position – current position
    temp=sc8;                //temp is servo 8 center position
    temp=temp+pos7+pos6;     //calculate temp for changes
    ser8=temp;               //servo 8 position equal to temp value
}

```

The next program is written to command the left leg for bending or stretching. To bend the leg, servo 6 pulse width will be increase by 1 level and servo 7's pulse width will be decrease by 2 levels. This action will rotate the servo 6's shaft to clockwise direction and servo 7 shaft to anticlockwise direction which resulting the left leg to bend. Different rotation direction of the servos will make the leg to stretches back to normal. The program is also applied to the right leg with different servo selection.

```

void bleft(int dir)         //bleft routine for bend to left and set variable direction
{
    if(dir=='-')            //Select if to shorten the leg if variable input is –ve sign
    {
        ser6=ser6+1;        //increase current servo 6 position by 1 level
        ser7=ser7-2;        //decline current servo 7 position by 2 level
        serv8();             //Call serv8 routine
    }
    else if(dir=='+')       //Select if to stretches leg if variable input is +ve sign
    {
        ser6=ser6-1;        //decrease current servo 6 position by 1 level
        ser7=ser7+2;        //increase current servo 6 position by 2 level
        serv8();             //Call serv8 routine
    }
}

```

4.3.3 Computer Instruction C Program

This C program is the main program for the user instruction to the robot, and will be execute using the MPLAB. The program will ask the user to select 6

available actions for the robot to achieve by inputting a number ranging 1 to 6 using the keyboard. If the input is 4, 5 or 6 the program will further ask the user to enter the number of steps to be taken by the robot ranging from 1 to 9 steps.

```

printf("\nSelect action : ");
do {
    act=getc();
} while (act !='0' && act !='2' && act !='3' && act !='4' && act !='5' && act !='6' && act
!='7' && act !='8' && act !='9');
printf("%c\r",act);

if(act=='0'){bend('-',90,6);}
else if(act=='1'){bend('+',90,6);}
else if(act=='2'){tilt('-',90,5);}
else if(act=='3'){tilt('+',90,5);}
else if(act=='4')
{
    printf("\nFast walk\r");
    printf("\nNumber of steps : ");
    do {
        s=getc();
    } while (act !='1' && act !='2' && act !='3' && act !='4' && act !='5' && act
!='6' && act !='7' && act !='8' && act !='9');
    printf("%c\r",s);
    s=convert(s);
    walk(30,s,4);
    }
else if(act=='5')
{
    printf("\nMed walk\r");
    printf("\nNumber of steps : ");
    do {
        s=getc();
    } while (act !='1' && act !='2' && act !='3' && act !='4' && act !='5' && act
!='6' && act !='7' && act !='8' && act !='9');
    printf("%c\r",s);
    s=convert(s);
    walk(60,s,4);
    }
else if(act=='6')
{
    printf("\nSlow walk\r");
    printf("\nNumber of steps : ");
    do {
        s=getc();
    } while (act !='1' && act !='2' && act !='3' && act !='4' && act !='5' && act
!='6' && act !='7' && act !='8' && act !='9');
    printf("%c\r",s);
    s=convert(s);
    walk(250,s,5);
    }
else if(act=='r'){reset();}
else {send();}
} while (TRUE);
}

```

CHAPTER 5

CONCLUSION AND RECOMENDATION

The twentieth century was a period of radical change from almost every aspect of scientific discoveries. Increasingly cheap and compact computing power, however, has made two-legged robots more feasible. Throughout the project student are required to do a lot of research related to bipedal robot, make a lot of decision in choosing suitable material and equipment and also anticipate in design process. At the end of the project, the bipedal robot can be used to investigate the theories of bipedal walking and can contribute to more advance studies on humanoid robot.

As recommendation due to this project breakdown, several different approaches needed to be taken toward this project completion. To avoid failure of the controller circuit, each servo motor needed to be initially tested at each joints before integrating it all together one by one. The student also needed to understand clearly the operation of the servo before designing the servo controller. This project methodology has to be revised for a better outcome. As an additional recommendation a single microcontroller can be used instead of two to produce the same output, by storing the servo selection and position data in the controller temporary registers. The bipedal robot's 6 bits servo positioning resolution can also be refine to have a much improve and smooth walking gait. Further development, would be designing it to be autonomous with addition of sensors and camera. After it had become an autonomous walker (able to walk on any surface), next we can add a remote controller so that the robot can be controlled remotely while the robot deal with obstacles. In the future, human can be replaced by machine that can move like human and such robot can spare human life by doing harmful task or work in the most horrible surroundings.

REFERENCES

- [1] Harprit Sandhu, (1997). *“An Introduction to Robotics”*. Nexus Special Interest Ltd, Nexus House, Boundary way, Hemel Hempstead, England.
- [2] Saeed b Niku, (2001). *“Introduction to Robotics, Analysis, System, Application”*. Prentice Hall Inc, Upper Saddle River, New Jersey.
- [3] Arthur G. Erdman, George N. Sandor, (1984). *“Mechanism Design: Analysis and Synthesis”*. Prentice-Hall.
- [4] Barnett, Cox and O’Cull, (2004), *“Embedded C Programming and the Microchip PIC”*. 1st Edition, Thomson Delmar Learning
- [5] Mazidi, M.A, McKinlay, R.D., Causey, D., (2008), *“PIC Microcontroller and Embedded Systems”*. Pearson Prentice Hall.
- [6] Huang, (2005). *“PIC Microcontroller: An Introduction to Software and Hardware Interfacing”*. Thomson Delmar Learning.
- [7] Muhamad Aidil Jazmi, (2006). *“Microprocessor 2 Project Technical Report”*. Institute Technology of Petronas, Perak, Malaysia.
(<http://aidilj.tripod.com/>)
- [8] Erik V. Cuevas, Daniel Zaldivar, Raul Rojas, (2005). *“Bipedal Robot Description Technical Report”*. Freie Universität Berlin, Institut für Informatik Takustr, Berlin, Germany, Universidad de Guadalajara Av. Revolución, Guadalajara, Mexico.
(<http://page.mi.fu-berlin.de/~zaldivar/files/tr-b-04-19.pdf>)
- [9] (2007). *“Asimo the World Most Advance Robot”*.
(<http://www.asimo.honda.com>)
- [10] (2003-2007). *“Kondo robot”*
(<http://www.kondo-robot.com>)
- [11] (2007). *“Futaba Standard Analog Servo”*
(<http://www.futaba-rc.com/servos/index.html>)

- [12] (2005-2007). “Bipedalism” (<http://en.wikipedia.org/wiki/Biped>)
- [13] (2005-2007). “Mechanical linkage”.
(http://en.wikipedia.org/wiki/Linkage_%28mechanical%29#References)
- [14] (2007). “What a Servo: A Quick Tutorial”
(<http://www.seattlerobotics.org/guide/servos.html>)
- [15] Ben Fry and Casey Reas, (2001-2007). “Processing”.
(<http://processing.org/reference/index.html>)

APPENDICES

APPENDIX A – PROJECT GANTT CHART

APPENDIX B – BIPEDAL ROBOT SCHEMATIC BOARD

APPENDIX C – FUTABA S3001 STANDARD SERVO SPECIFICATION

APPENDIX D – PIC16F877 DATA SHEET

APPENDIX E - MAX232I DUAL DRIVERS/RECEIVERS DATA SHEET

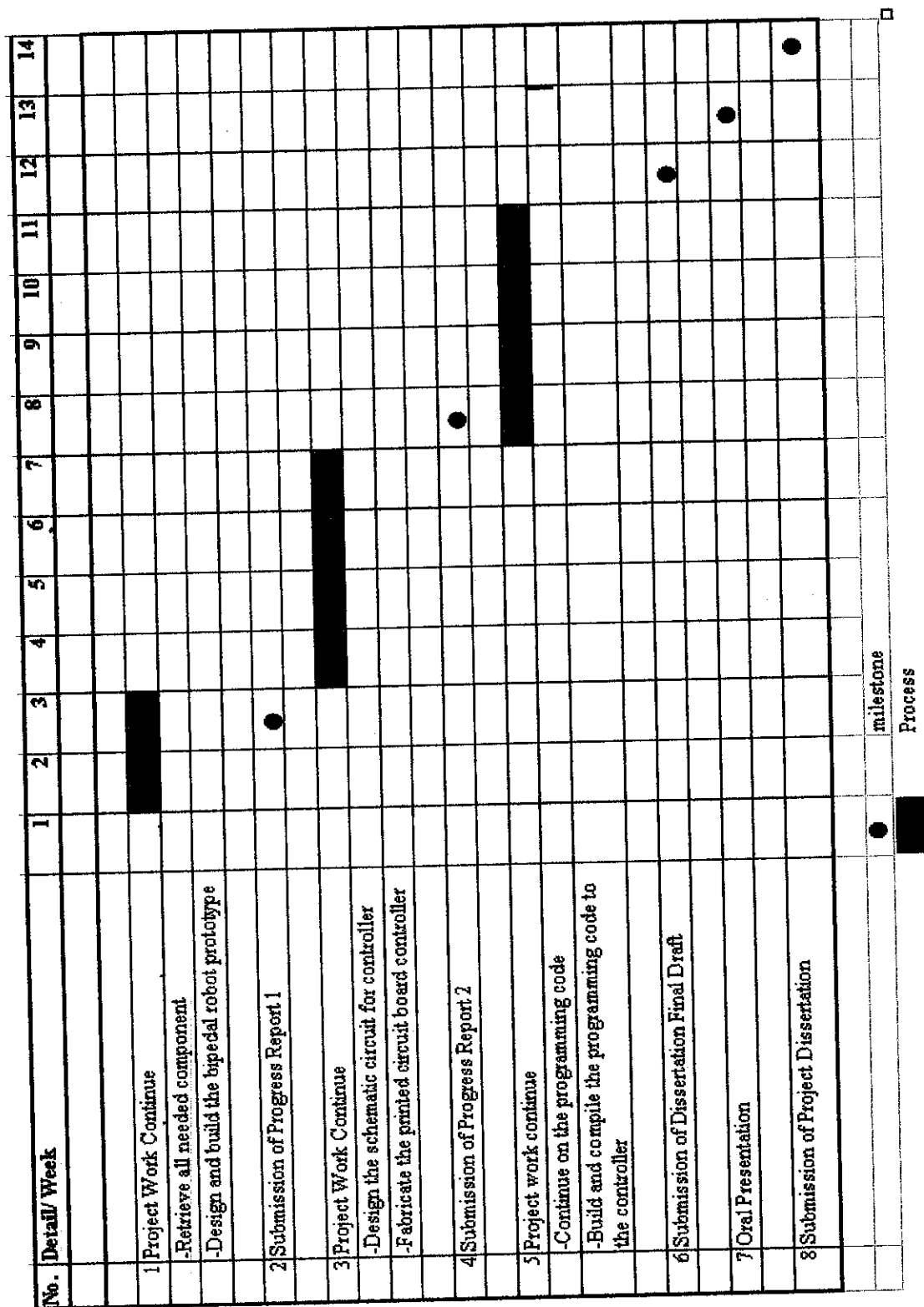
APPENDIX F – SERVO CONTROLLER C PROGRAM

APPENDIX G – WALKING CONTROLLER C PROGRAM

APPENDIX H – COMPUTER INSTRUCTION C PROGRAM

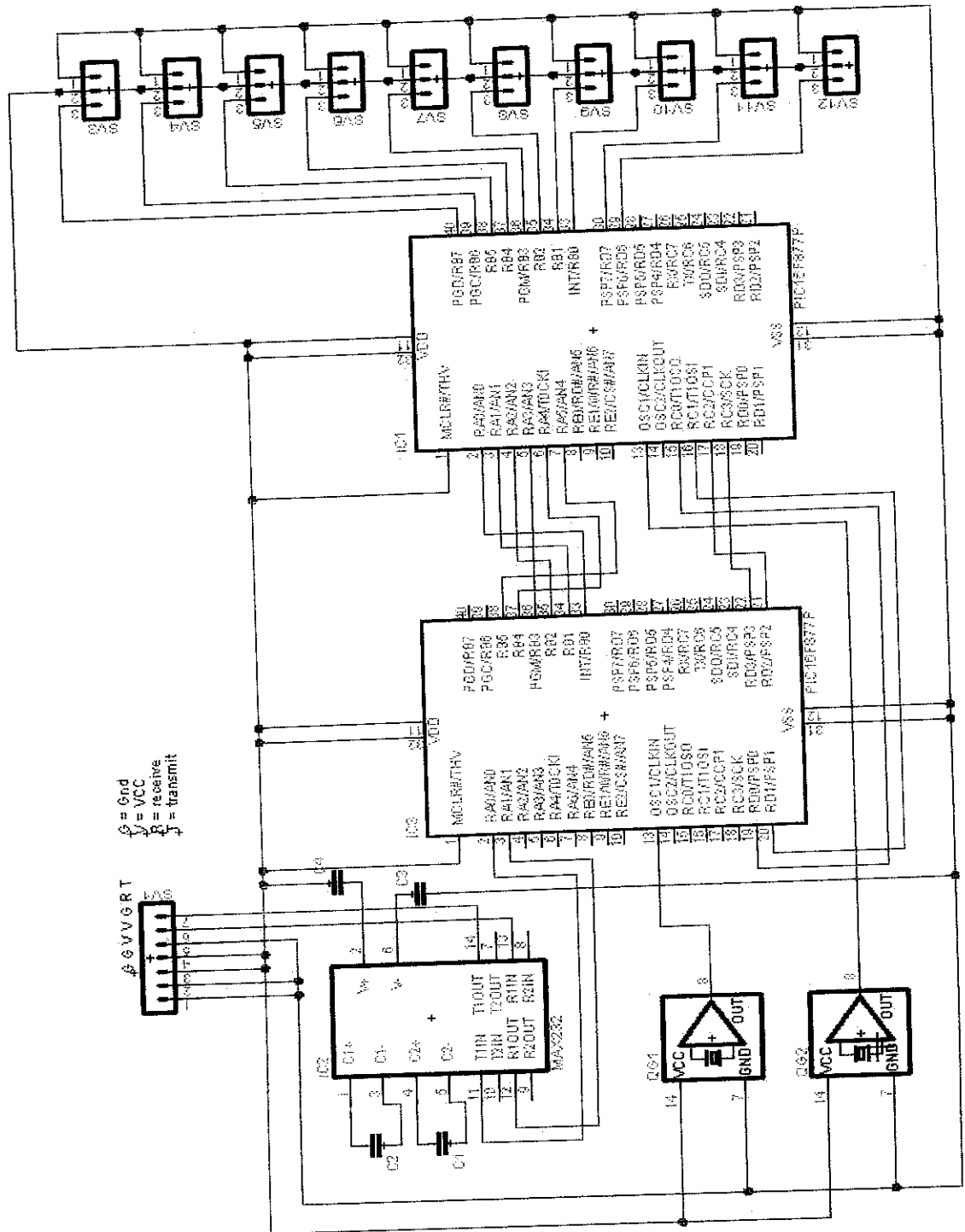
APPENDIX A

PROJECT GANTT CHART



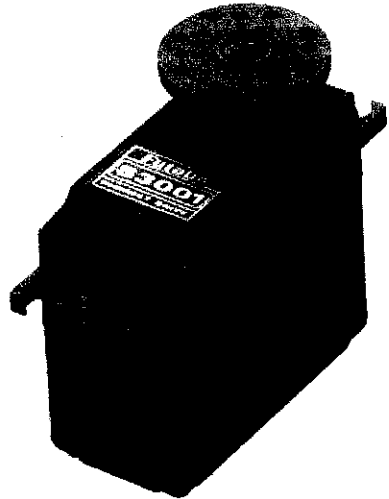
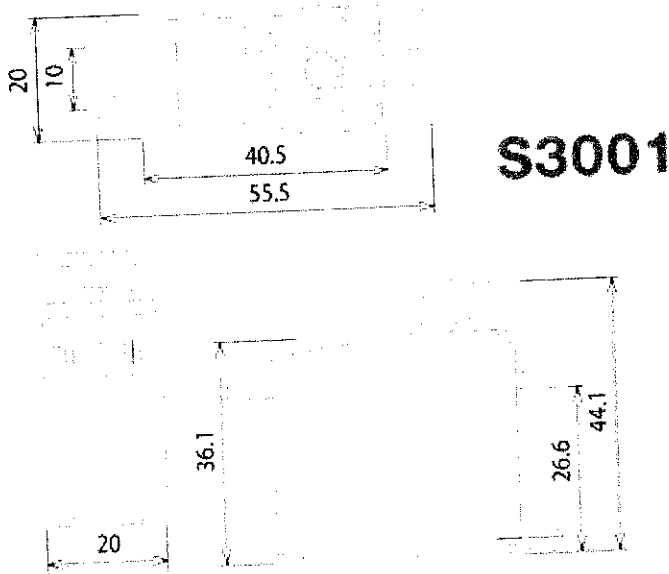
APPENDIX B

BIPEDAL ROBOT SCHEMATIC BOARD



APPENDIX C

FUTABA S3001 STANDARD SERVO SPECIFICATION



S3001 is a Standard Servo with a Ball Bearing on the output shaft. It has a "J" connector and uses a nylon gears. This servo can produce high-current draw from your batteries. If using NiMH or LiPo batteries, make sure they are capable of delivering sufficient amps.

FEATURES: Single Top Ball Bearing, 3-pole motor, same mounting as S148 and S9201 (individual rectangular grommets),

SPECS:

Length : 1.6" (41mm)

Width : 0.8" (20mm)

Height : 1.4" (35mm)

Weight : 1.6oz (45.1g)

Torque : 33 oz-in at 4.8V 42 oz-in. at 6V

Transit : 0.28 sec/60° @ 4.8V .22 sec/60° @ 6V

APPENDIX D
PIC 16F877 DATA SHEET

APPENDIX E
MAX232I DUAL DRIVERS/RECEIVERS DATA SHEET



PIC16F87X

28/40-pin 8-Bit CMOS FLASH Microcontrollers

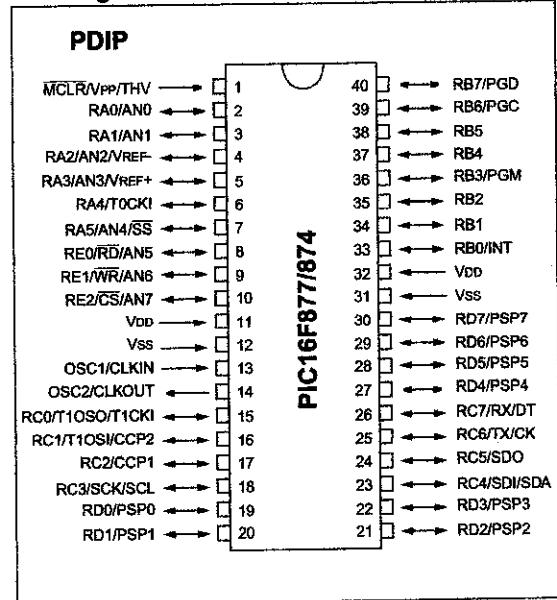
Devices Included in this Data Sheet:

- PIC16F873
- PIC16F876
- PIC16F874
- PIC16F877

Microcontroller Core Features:

- High-performance RISC CPU
- Only 35 single word instructions to learn
- All single cycle instructions except for program branches which are two cycle
- Operating speed: DC - 20 MHz clock input
DC - 200 ns instruction cycle
- Up to 8K x 14 words of FLASH Program Memory,
Up to 368 x 8 bytes of Data Memory (RAM)
Up to 256 x 8 bytes of EEPROM data memory
- Pinout compatible to the PIC16C73B/74B/76/77
- Interrupt capability (up to 14 sources)
- Eight level deep hardware stack
- Direct, indirect and relative addressing modes
- Power-on Reset (POR)
- Power-up Timer (PWRT) and
Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC
oscillator for reliable operation
- Programmable code-protection
- Power saving SLEEP mode
- Selectable oscillator options
- Low-power, high-speed CMOS FLASH/EEPROM
technology
- Fully static design
- In-Circuit Serial Programming™ (ICSP) via two
pins
- Single 5V In-Circuit Serial Programming capability
- In-Circuit Debugging via two pins
- Processor read/write access to program memory
- Wide operating voltage range: 2.0V to 5.5V
- High Sink/Source Current: 25 mA
- Commercial and Industrial temperature ranges
- Low-power consumption:
 - < 2 mA typical @ 5V, 4 MHz
 - 20 µA typical @ 3V, 32 kHz
 - < 1 µA typical standby current

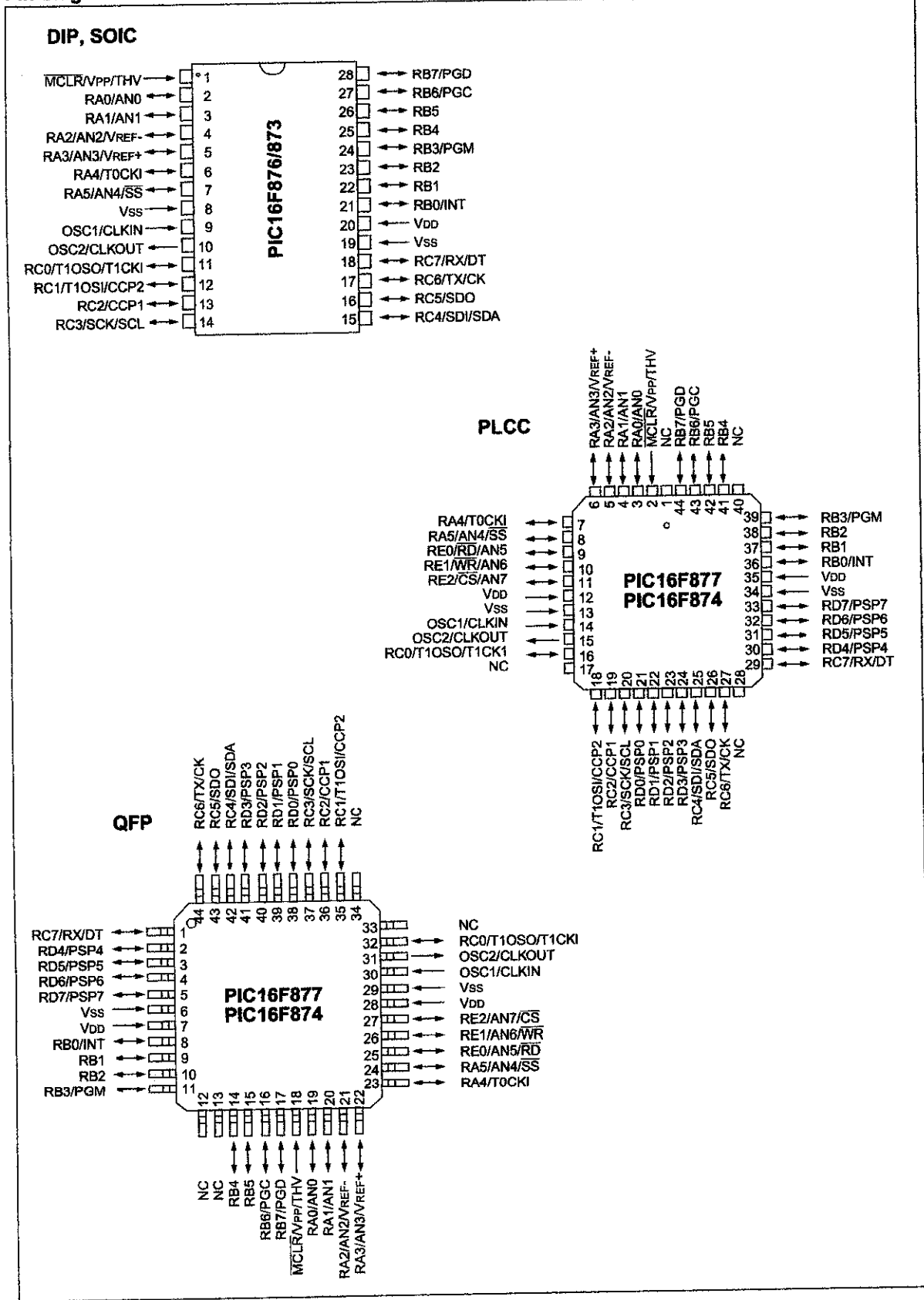
Pin Diagram



Peripheral Features:

- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler,
can be incremented during sleep via external
crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period
register, prescaler and postscaler
- Two Capture, Compare, PWM modules
 - Capture is 16-bit, max. resolution is 12.5 ns
 - Compare is 16-bit, max. resolution is 200 ns
 - PWM max. resolution is 10-bit
- 10-bit multi-channel Analog-to-Digital converter
- Synchronous Serial Port (SSP) with SPI™ (Master
Mode) and I²C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver
Transmitter (USART/SCI) with 9-bit address
detection
- Parallel Slave Port (PSP) 8-bits wide, with
external RD, WR and CS controls (40/44-pin only)
- Brown-out detection circuitry for
Brown-out Reset (BOR)

Pin Diagrams



PIC16F87X

| Key Features PICmicro™ Mid-Range Reference Manual (DS33023) | PIC16F873 | PIC16F874 | PIC16F876 | PIC16F877 |
|--|-------------------------|-------------------------|-------------------------|-------------------------|
| Operating Frequency | DC - 20 MHz | DC - 20 MHz | DC - 20 MHz | DC - 20 MHz |
| Resets (and Delays) | POR, BOR (PWRT, OST) | POR, BOR (PWRT, OST) | POR, BOR (PWRT, OST) | POR, BOR (PWRT, OST) |
| FLASH Program Memory (14-bit words) | 4K | 4K | 8K | 8K |
| Data Memory (bytes) | 192 | 192 | 368 | 368 |
| EEPROM Data Memory | 128 | 128 | 256 | 256 |
| Interrupts | 13 | 14 | 13 | 14 |
| I/O Ports | Ports A,B,C | Ports A,B,C,D,E | Ports A,B,C | Ports A,B,C,D,E |
| Timers | 3 | 3 | 3 | 3 |
| Capture/Compare/PWM modules | 2 | 2 | 2 | 2 |
| Serial Communications | MSSP, USART | MSSP, USART | MSSP, USART | MSSP, USART |
| Parallel Communications | — | PSP | — | PSP |
| 10-bit Analog-to-Digital Module | 5 input channels | 8 input channels | 5 input channels | 8 input channels |
| Instruction Set | 35 Instructions | 35 Instructions | 35 Instructions | 35 Instructions |

Table of Contents

| | |
|--|-----|
| 1.0 Device Overview | 5 |
| 2.0 Memory Organization | 11 |
| 3.0 I/O Ports | 29 |
| 4.0 Data EEPROM and FLASH Program Memory | 41 |
| 5.0 Timer0 Module | 47 |
| 6.0 Timer1 Module | 51 |
| 7.0 Timer2 Module | 55 |
| 8.0 Capture/Compare/PWM (CCP) Module(s) | 57 |
| 9.0 Master Synchronous Serial Port (MSSP) Module | 63 |
| 10.0 Universal Synchronous Asynchronous Receiver Transmitter (USART) | 95 |
| 11.0 Analog-to-Digital Converter (A/D) Module | 111 |
| 12.0 Special Features of the CPU | 121 |
| 13.0 Instruction Set Summary | 137 |
| 14.0 Development Support | 145 |
| 15.0 Electrical Characteristics | 151 |
| 16.0 DC and AC Characteristics Graphs and Tables | 173 |
| 17.0 Packaging Information | 175 |
| Appendix A: Revision History | 183 |
| Appendix B: Device Differences | 183 |
| Appendix C: Conversion Considerations | 183 |
| Index | 185 |
| On-Line Support | 191 |
| Product Identification System | 193 |

To Our Valued Customers

Most Current Data Sheet

To obtain the most up-to-date version of this data sheet, please register at our Worldwide Web site at:

<http://www.microchip.com>

You can determine the version of a data sheet by examining its literature number found on the bottom outside corner of any page. The last character of the literature number is the version number. e.g., DS30000A is version A of document DS30000.

New Customer Notification System

Register on our web site (www.microchip.com/cn) to receive the most current information on our products.

Errata

An errata sheet may exist for current devices, describing minor operational differences (from the data sheet) and recommended workarounds. As device/documentation issues become known to us, we will publish an errata sheet. The errata will specify the revision of silicon and revision of document to which it applies.

To determine if an errata sheet exists for a particular device, please check with one of the following:

- Microchip's Worldwide Web site; <http://www.microchip.com>
- Your local Microchip sales office (see last page)
- The Microchip Corporate Literature Center; U.S. FAX: (480) 786-7277

When contacting a sales office or the literature center, please specify which device, revision of silicon and data sheet (include literature number) you are using.

Corrections to this Data Sheet

We constantly strive to improve the quality of all our products and documentation. We have spent a great deal of time to ensure that this document is correct. However, we realize that we may have missed a few things. If you find any information that is missing or appears in error, please:

- Fill out and mail in the reader response form in the back of this data sheet.
- E-mail us at webmaster@microchip.com.

We appreciate your assistance in making this a better document.

PIC16F87X

1.0 DEVICE OVERVIEW

This document contains device-specific information. Additional information may be found in the PICmicro™ Mid-Range Reference Manual, (DS33023), which may be obtained from your local Microchip Sales Representative or downloaded from the Microchip website. The Reference Manual should be considered a complementary document to this data sheet, and is highly recommended reading for a better understanding of the device architecture and operation of the peripheral modules.

There are four devices (PIC16F873, PIC16F874, PIC16F876 and PIC16F877) covered by this data sheet. The PIC16F876/873 devices come in 28-pin packages and the PIC16F877/874 devices come in 40-pin packages. The 28-pin devices do not have a Parallel Slave Port implemented.

The following two figures are device block diagrams sorted by pin number; 28-pin for Figure 1-1 and 40-pin for Figure 1-2. The 28-pin and 40-pin pinouts are listed in Table 1-1 and Table 1-2, respectively.

FIGURE 1-1: PIC16F873 AND PIC16F876 BLOCK DIAGRAM

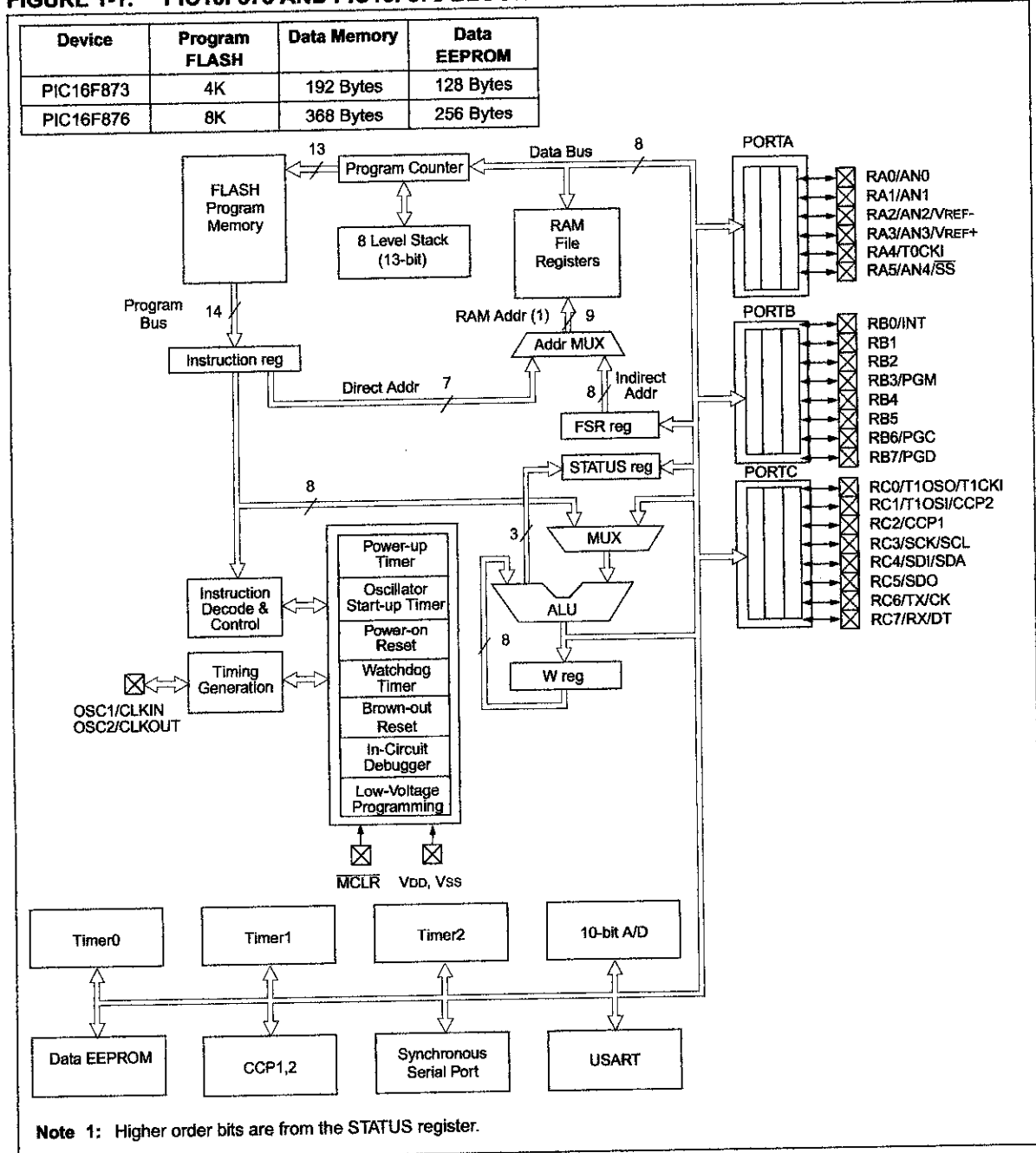
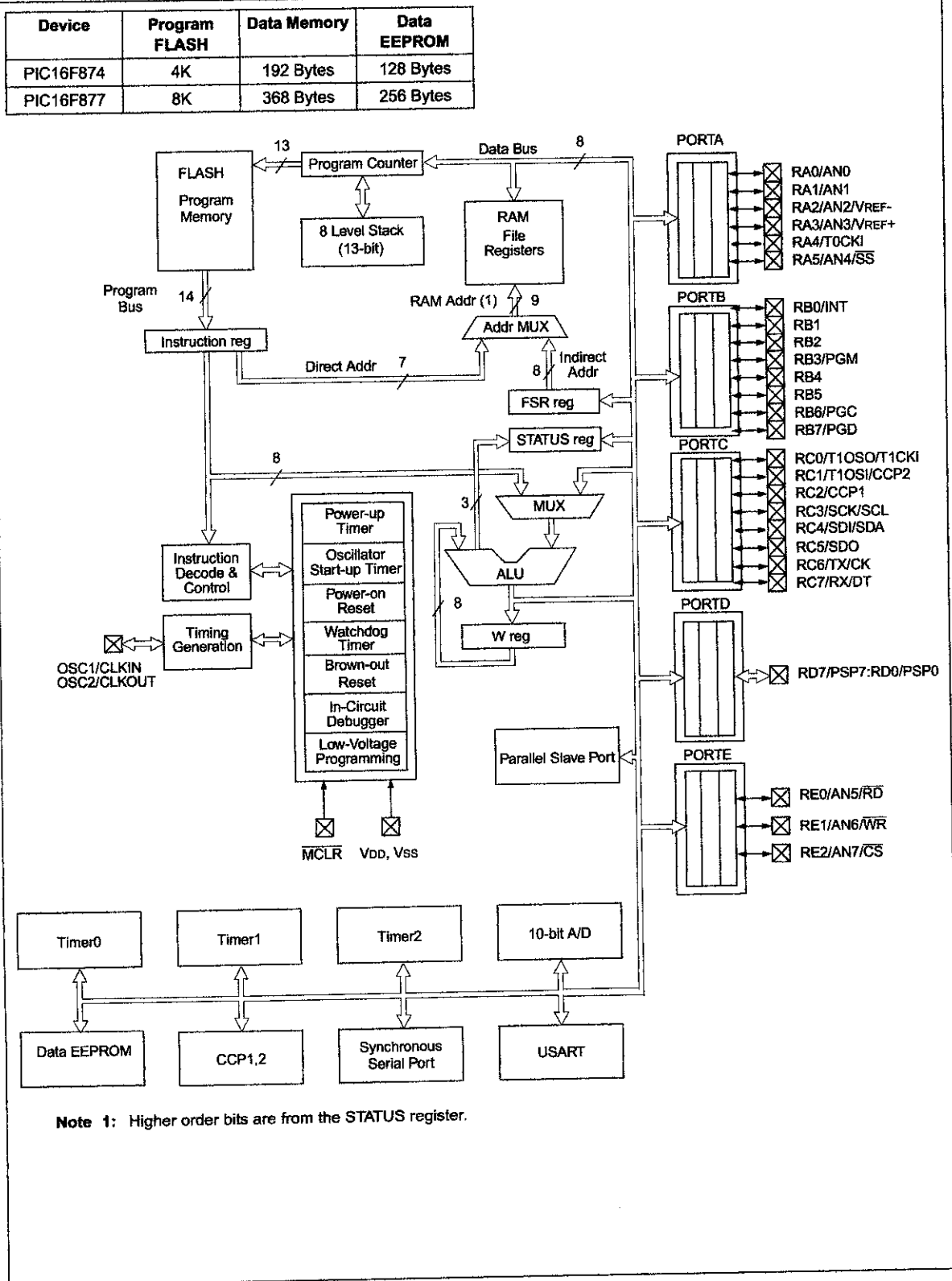


FIGURE 1-2: PIC16F874 AND PIC16F877 BLOCK DIAGRAM



PIC16F87X

TABLE 1-1: PIC16F873 AND PIC16F876 PINOUT DESCRIPTION

| Pin Name | DIP Pin# | SOIC Pin# | I/O/P Type | Buffer Type | Description |
|-----------------|----------|-----------|------------|------------------------|---|
| OSC1/CLKIN | 9 | 9 | I | ST/CMOS ⁽³⁾ | Oscillator crystal input/external clock source input. |
| OSC2/CLKOUT | 10 | 10 | O | — | Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, the OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate. |
| MCLR/VPP/THV | 1 | 1 | I/P | ST | Master clear (reset) input or programming voltage input or high voltage test mode control. This pin is an active low reset to the device. |
| RA0/AN0 | 2 | 2 | I/O | TTL | <p>PORTA is a bi-directional I/O port.</p> <p>RA0 can also be analog input0</p> <p>RA1 can also be analog input1</p> <p>RA2 can also be analog input2 or negative analog reference voltage</p> <p>RA3 can also be analog input3 or positive analog reference voltage</p> <p>RA4 can also be the clock input to the Timer0 module. Output is open drain type.</p> <p>RA5 can also be analog input4 or the slave select for the synchronous serial port.</p> |
| RA1/AN1 | 3 | 3 | I/O | TTL | |
| RA2/AN2/VREF- | 4 | 4 | I/O | TTL | |
| RA3/AN3/VREF+ | 5 | 5 | I/O | TTL | |
| RA4/T0CKI | 6 | 6 | I/O | ST | |
| RA5/SS/AN4 | 7 | 7 | I/O | TTL | |
| RB0/INT | 21 | 21 | I/O | TTL/ST ⁽¹⁾ | <p>PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.</p> <p>RB0 can also be the external interrupt pin.</p> <p>RB3 can also be the low voltage programming input</p> <p>Interrupt on change pin.</p> <p>Interrupt on change pin.</p> <p>Interrupt on change pin or In-Circuit Debugger pin. Serial programming clock.</p> <p>Interrupt on change pin or In-Circuit Debugger pin. Serial programming data.</p> |
| RB1 | 22 | 22 | I/O | TTL | |
| RB2 | 23 | 23 | I/O | TTL | |
| RB3/PGM | 24 | 24 | I/O | TTL | |
| RB4 | 25 | 25 | I/O | TTL | |
| RB5 | 26 | 26 | I/O | TTL | |
| RB6/PGC | 27 | 27 | I/O | TTL/ST ⁽²⁾ | |
| RB7/PGD | 28 | 28 | I/O | TTL/ST ⁽²⁾ | |
| RC0/T1OSO/T1CKI | 11 | 11 | I/O | ST | <p>PORTC is a bi-directional I/O port.</p> <p>RC0 can also be the Timer1 oscillator output or Timer1 clock input.</p> <p>RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output.</p> <p>RC2 can also be the Capture1 input/Compare1 output/PWM1 output.</p> <p>RC3 can also be the synchronous serial clock input/output for both SPI and I²C modes.</p> <p>RC4 can also be the SPI Data In (SPI mode) or data I/O (I²C mode).</p> <p>RC5 can also be the SPI Data Out (SPI mode).</p> <p>RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.</p> <p>RC7 can also be the USART Asynchronous Receive or Synchronous Data.</p> |
| RC1/T1OSI/CCP2 | 12 | 12 | I/O | ST | |
| RC2/CCP1 | 13 | 13 | I/O | ST | |
| RC3/SCK/SCL | 14 | 14 | I/O | ST | |
| RC4/SDI/SDA | 15 | 15 | I/O | ST | |
| RC5/SDO | 16 | 16 | I/O | ST | |
| RC6/TX/CK | 17 | 17 | I/O | ST | |
| RC7/RX/DT | 18 | 18 | I/O | ST | |
| Vss | 8, 19 | 8, 19 | P | — | Ground reference for logic and I/O pins. |
| VDD | 20 | 20 | P | — | Positive supply for logic and I/O pins. |

Legend: I = input O = output I/O = input/output P = power
 — = Not used TTL = TTL input ST = Schmitt Trigger input

- Note** 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
 2: This buffer is a Schmitt Trigger input when used in serial programming mode.
 3: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

TABLE 1-2: PIC16F874 AND PIC16F877 PINOUT DESCRIPTION

| Pin Name | DIP Pin# | PLCC Pin# | QFP Pin# | I/O/P Type | Buffer Type | Description |
|-----------------|----------|-----------|----------|------------|------------------------|--|
| OSC1/CLKIN | 13 | 14 | 30 | I | ST/CMOS ⁽⁴⁾ | Oscillator crystal input/external clock source input. |
| OSC2/CLKOUT | 14 | 15 | 31 | O | — | Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate. |
| MCLR/VPP/THV | 1 | 2 | 18 | I/P | ST | Master clear (reset) input or programming voltage input or high voltage test mode control. This pin is an active low reset to the device. |
| RA0/AN0 | 2 | 3 | 19 | I/O | TTL | PORTA is a bi-directional I/O port. RA0 can also be analog input0 RA1 can also be analog input1 RA2 can also be analog input2 or negative analog reference voltage RA3 can also be analog input3 or positive analog reference voltage RA4 can also be the clock input to the Timer0 timer/counter. Output is open drain type. RA5 can also be analog input4 or the slave select for the synchronous serial port. |
| RA1/AN1 | 3 | 4 | 20 | I/O | TTL | |
| RA2/AN2/VREF- | 4 | 5 | 21 | I/O | TTL | |
| RA3/AN3/VREF+ | 5 | 6 | 22 | I/O | TTL | |
| RA4/T0CKI | 6 | 7 | 23 | I/O | ST | |
| RA5/SS/AN4 | 7 | 8 | 24 | I/O | TTL | |
| RB0/INT | 33 | 36 | 8 | I/O | TTL/ST ⁽¹⁾ | PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs. RB0 can also be the external interrupt pin. RB3 can also be the low voltage programming input Interrupt on change pin. Interrupt on change pin. Interrupt on change pin or In-Circuit Debugger pin. Serial programming clock. Interrupt on change pin or In-Circuit Debugger pin. Serial programming data. |
| RB1 | 34 | 37 | 9 | I/O | TTL | |
| RB2 | 35 | 38 | 10 | I/O | TTL | |
| RB3/PGM | 36 | 39 | 11 | I/O | TTL | |
| RB4 | 37 | 41 | 14 | I/O | TTL | |
| RB5 | 38 | 42 | 15 | I/O | TTL | |
| RB6/PGC | 39 | 43 | 16 | I/O | TTL/ST ⁽²⁾ | |
| RB7/PGD | 40 | 44 | 17 | I/O | TTL/ST ⁽²⁾ | |
| RC0/T1OSO/T1CKI | 15 | 16 | 32 | I/O | ST | PORTC is a bi-directional I/O port. RC0 can also be the Timer1 oscillator output or a Timer1 clock input. RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output. RC2 can also be the Capture1 input/Compare1 output/PWM1 output. RC3 can also be the synchronous serial clock input/output for both SPI and I ² C modes. RC4 can also be the SPI Data In (SPI mode) or data I/O (I ² C mode). RC5 can also be the SPI Data Out (SPI mode). RC6 can also be the USART Asynchronous Transmit or Synchronous Clock. RC7 can also be the USART Asynchronous Receive or Synchronous Data. |
| RC1/T1OSI/CCP2 | 16 | 18 | 35 | I/O | ST | |
| RC2/CCP1 | 17 | 19 | 36 | I/O | ST | |
| RC3/SCK/SCL | 18 | 20 | 37 | I/O | ST | |
| RC4/SDI/SDA | 23 | 25 | 42 | I/O | ST | |
| RC5/SDO | 24 | 26 | 43 | I/O | ST | |
| RC6/TX/CK | 25 | 27 | 44 | I/O | ST | |
| RC7/RX/DT | 26 | 29 | 1 | I/O | ST | |

Legend: I = input O = output I/O = input/output P = power
 — = Not used TTL = TTL input ST = Schmitt Trigger input

- Note**
- 1: This buffer is a Schmitt Trigger input when configured as an external interrupt.
 - 2: This buffer is a Schmitt Trigger input when used in serial programming mode.
 - 3: This buffer is a Schmitt Trigger input when configured as general purpose I/O and a TTL input when used in the Parallel Slave Port mode (for interfacing to a microprocessor bus).
 - 4: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

TABLE 1-2: PIC16F874 AND PIC16F877 PINOUT DESCRIPTION (CONTINUED)

| Pin Name | DIP Pin# | PLCC Pin# | QFP Pin# | I/O/P Type | Buffer Type | Description |
|---------------------------|----------|------------|-------------|------------|-----------------------|---|
| RD0/PSP0 | 19 | 21 | 38 | I/O | ST/TTL ⁽³⁾ | PORTD is a bi-directional I/O port or parallel slave port when interfacing to a microprocessor bus. |
| RD1/PSP1 | 20 | 22 | 39 | I/O | ST/TTL ⁽³⁾ | |
| RD2/PSP2 | 21 | 23 | 40 | I/O | ST/TTL ⁽³⁾ | |
| RD3/PSP3 | 22 | 24 | 41 | I/O | ST/TTL ⁽³⁾ | |
| RD4/PSP4 | 27 | 30 | 2 | I/O | ST/TTL ⁽³⁾ | |
| RD5/PSP5 | 28 | 31 | 3 | I/O | ST/TTL ⁽³⁾ | |
| RD6/PSP6 | 29 | 32 | 4 | I/O | ST/TTL ⁽³⁾ | |
| RD7/PSP7 | 30 | 33 | 5 | I/O | ST/TTL ⁽³⁾ | |
| RE0/ \overline{RD} /AN5 | 8 | 9 | 25 | I/O | ST/TTL ⁽³⁾ | PORTE is a bi-directional I/O port. RE0 can also be read control for the parallel slave port, or analog input5. RE1 can also be write control for the parallel slave port, or analog input6. RE2 can also be select control for the parallel slave port, or analog input7. |
| RE1/ \overline{WR} /AN6 | 9 | 10 | 26 | I/O | ST/TTL ⁽³⁾ | |
| RE2/ \overline{CS} /AN7 | 10 | 11 | 27 | I/O | ST/TTL ⁽³⁾ | |
| Vss | 12,31 | 13,34 | 6,29 | P | — | Ground reference for logic and I/O pins. |
| VDD | 11,32 | 12,35 | 7,28 | P | — | Positive supply for logic and I/O pins. |
| NC | — | 1,17,28,40 | 12,13,33,34 | | — | These pins are not internally connected. These pins should be left unconnected. |

Legend: I = input O = output I/O = input/output P = power
 — = Not used TTL = TTL input ST = Schmitt Trigger input

- Note**
- 1: This buffer is a Schmitt Trigger input when configured as an external interrupt.
 - 2: This buffer is a Schmitt Trigger input when used in serial programming mode.
 - 3: This buffer is a Schmitt Trigger input when configured as general purpose I/O and a TTL input when used in the Parallel Slave Port mode (for interfacing to a microprocessor bus).
 - 4: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

NOTES:

2.0 MEMORY ORGANIZATION

There are three memory blocks in each of these PICmicro MCUs. The Program Memory and Data Memory have separate buses so that concurrent access can occur and is detailed in this section. The EEPROM data memory block is detailed in Section 4.0.

Additional information on device memory may be found in the PICmicro™ Mid-Range Reference Manual, (DS33023).

2.1 Program Memory Organization

The PIC16F87X devices have a 13-bit program counter capable of addressing an 8K x 14 program memory space. The PIC16F877/876 devices have 8K x 14 words of FLASH program memory and the PIC16F873/874 devices have 4K x 14. Accessing a location above the physically implemented address will cause a wrap-around.

The reset vector is at 0000h and the interrupt vector is at 0004h.

FIGURE 2-1: PIC16F877/876 PROGRAM MEMORY MAP AND STACK

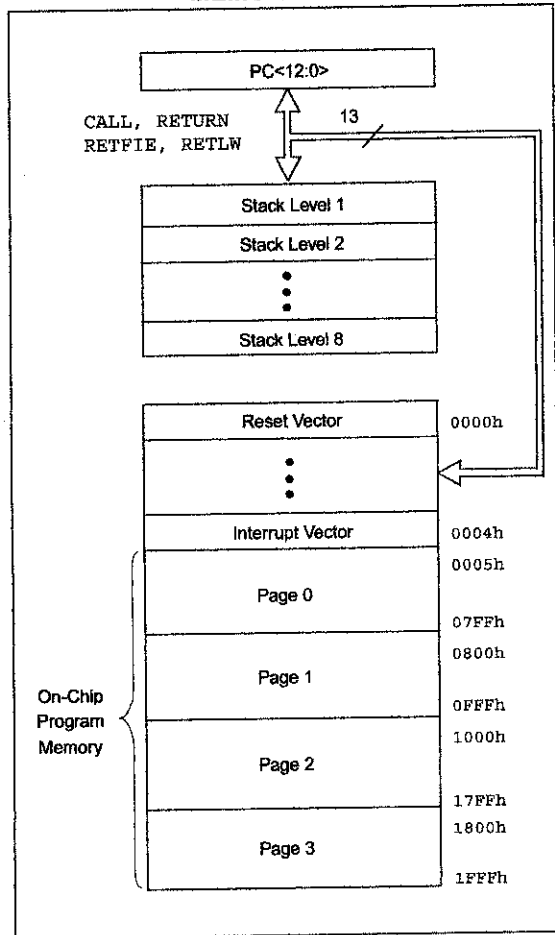
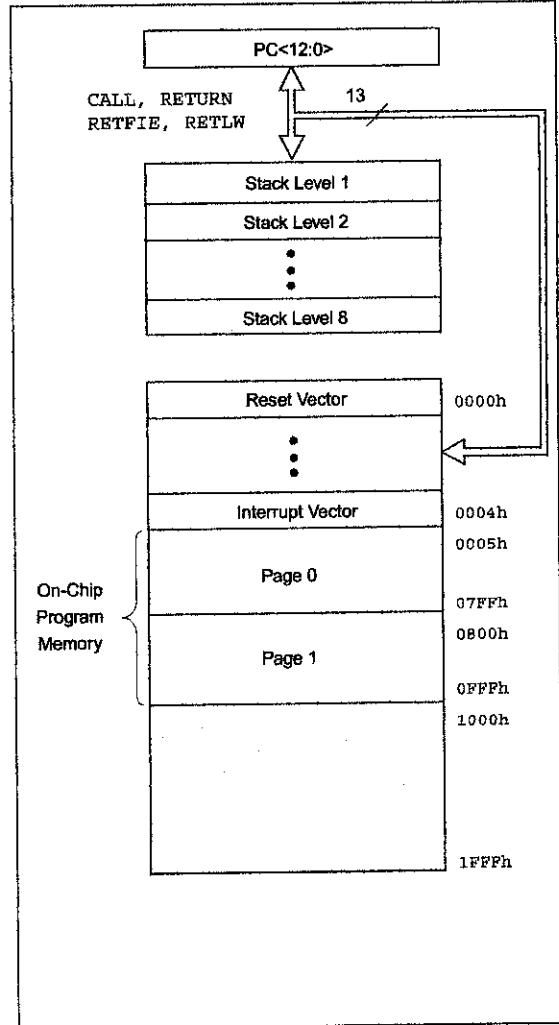


FIGURE 2-2: PIC16F874/873 PROGRAM MEMORY MAP AND STACK



PIC16F87X

2.2 Data Memory Organization

The data memory is partitioned into multiple banks which contain the General Purpose Registers and the Special Function Registers. Bits RP1 (STATUS<6>) and RP0 (STATUS<5>) are the bank select bits.

| RP1:RP0 | Bank |
|---------|------|
| 00 | 0 |
| 01 | 1 |
| 10 | 2 |
| 11 | 3 |

Each bank extends up to 7Fh (128 bytes). The lower locations of each bank are reserved for the Special Function Registers. Above the Special Function Registers are General Purpose Registers, implemented as static RAM. All implemented banks contain Special Function Registers. Some "high use" Special Function Registers from one bank may be mirrored in another bank for code reduction and quicker access.

| |
|--|
| Note: EEPROM Data Memory description can be found in Section 4.0 of this Data Sheet |
|--|

2.2.1 GENERAL PURPOSE REGISTER FILE

The register file can be accessed either directly, or indirectly through the File Select Register FSR.

FIGURE 2-3: PIC16F877/876 REGISTER FILE MAP

| | | | | | | File Address | |
|-----------------------------------|-----|-----------------------------------|------------|-----------------------------------|--------------|-----------------------------------|--------------|
| Indirect addr. ^(*) | 00h | Indirect addr. ^(*) | 80h | Indirect addr. ^(*) | 100h | Indirect addr. ^(*) | 180h |
| TMR0 | 01h | OPTION_REG | 81h | TMR0 | 101h | OPTION_REG | 181h |
| PCL | 02h | PCL | 82h | PCL | 102h | PCL | 182h |
| STATUS | 03h | STATUS | 83h | STATUS | 103h | STATUS | 183h |
| FSR | 04h | FSR | 84h | FSR | 104h | FSR | 184h |
| PORTA | 05h | TRISA | 85h | | 105h | | 185h |
| PORTB | 06h | TRISB | 86h | PORTB | 106h | TRISB | 186h |
| PORTC | 07h | TRISC | 87h | | 107h | | 187h |
| PORTD ⁽¹⁾ | 08h | TRISD ⁽¹⁾ | 88h | | 108h | | 188h |
| PORTE ⁽¹⁾ | 09h | TRISE ⁽¹⁾ | 89h | | 109h | | 189h |
| PCLATH | 0Ah | PCLATH | 8Ah | PCLATH | 10Ah | PCLATH | 18Ah |
| INTCON | 0Bh | INTCON | 8Bh | INTCON | 10Bh | INTCON | 18Bh |
| PIR1 | 0Ch | PIE1 | 8Ch | EEDATA | 10Ch | EECON1 | 18Ch |
| PIR2 | 0Dh | PIE2 | 8Dh | EEADR | 10Dh | EECON2 | 18Dh |
| TMR1L | 0Eh | PCON | 8Eh | EEDATH | 10Eh | Reserved ⁽²⁾ | 18Eh |
| TMR1H | 0Fh | | 8Fh | EEADRH | 10Fh | Reserved ⁽²⁾ | 18Fh |
| T1CON | 10h | | 90h | | 110h | | 190h |
| TMR2 | 11h | SSPCON2 | 91h | | 111h | | 191h |
| T2CON | 12h | PR2 | 92h | | 112h | | 192h |
| SSPBUF | 13h | SSPADD | 93h | | 113h | | 193h |
| SSPCON | 14h | SSPSTAT | 94h | | 114h | | 194h |
| CCPR1L | 15h | | 95h | | 115h | | 195h |
| CCPR1H | 16h | | 96h | | 116h | | 196h |
| CCP1CON | 17h | | 97h | General Purpose Register 16 Bytes | 117h | General Purpose Register 16 Bytes | 197h |
| RCSTA | 18h | TXSTA | 98h | | 118h | | 198h |
| TXREG | 19h | SPBRG | 99h | | 119h | | 199h |
| RCREG | 1Ah | | 9Ah | | 11Ah | | 19Ah |
| CCPR2L | 1Bh | | 9Bh | | 11Bh | | 19Bh |
| CCPR2H | 1Ch | | 9Ch | | 11Ch | | 19Ch |
| CCP2CON | 1Dh | | 9Dh | | 11Dh | | 19Dh |
| ADRESH | 1Eh | ADRESL | 9Eh | | 11Eh | | 19Eh |
| ADCON0 | 1Fh | ADCON1 | 9Fh | | 11Fh | | 19Fh |
| | 20h | | A0h | | 120h | | 1A0h |
| General Purpose Register 96 Bytes | | General Purpose Register 80 Bytes | | General Purpose Register 80 Bytes | | General Purpose Register 80 Bytes | |
| | 7Fh | accesses 70h-7Fh | EFh F0h | accesses 70h-7Fh | 16Fh 170h | accesses 70h - 7Fh | 1EFh 1F0h |
| Bank 0 | | Bank 1 | FFh | Bank 2 | 17Fh | Bank 3 | |

— Unimplemented data memory locations, read as '0'.

* Not a physical register.

Note 1: These registers are not implemented on 28-pin devices.

2: These registers are reserved, maintain these registers clear.

FIGURE 2-4: PIC16F874/873 REGISTER FILE MAP

| | | | | | | | | File Address |
|-------------------------------|-----|-------------------------------|-----|-------------------------------|------|-------------------------------|------|--------------|
| Indirect addr. ^(*) | 00h | Indirect addr. ^(*) | 80h | Indirect addr. ^(*) | 100h | Indirect addr. ^(*) | 180h | |
| TMR0 | 01h | OPTION REG | 81h | TMR0 | 101h | OPTION REG | 181h | |
| PCL | 02h | PCL | 82h | PCL | 102h | PCL | 182h | |
| STATUS | 03h | STATUS | 83h | STATUS | 103h | STATUS | 183h | |
| FSR | 04h | FSR | 84h | FSR | 104h | FSR | 184h | |
| PORTA | 05h | TRISA | 85h | | 105h | | 185h | |
| PORTB | 06h | TRISB | 86h | PORTB | 106h | TRISB | 186h | |
| PORTC | 07h | TRISC | 87h | | 107h | | 187h | |
| PORTD ⁽¹⁾ | 08h | TRISD ⁽¹⁾ | 88h | | 108h | | 188h | |
| PORTE ⁽¹⁾ | 09h | TRISE ⁽¹⁾ | 89h | | 109h | | 189h | |
| PCLATH | 0Ah | PCLATH | 8Ah | PCLATH | 10Ah | PCLATH | 18Ah | |
| INTCON | 0Bh | INTCON | 8Bh | INTCON | 10Bh | INTCON | 18Bh | |
| PIR1 | 0Ch | PIE1 | 8Ch | EEDATA | 10Ch | EECON1 | 18Ch | |
| PIR2 | 0Dh | PIE2 | 8Dh | EEDADR | 10Dh | EECON2 | 18Dh | |
| TMR1L | 0Eh | PCON | 8Eh | EEDATH | 10Eh | Reserved ⁽²⁾ | 18Eh | |
| TMR1H | 0Fh | | 8Fh | EEADRH | 10Fh | Reserved ⁽²⁾ | 18Fh | |
| T1CON | 10h | | 90h | | 110h | | 190h | |
| TMR2 | 11h | SSPCON2 | 91h | | | | | |
| T2CON | 12h | PR2 | 92h | | | | | |
| SSPBUF | 13h | SSPADD | 93h | | | | | |
| SSPCON | 14h | SSPSTAT | 94h | | | | | |
| CCPR1L | 15h | | 95h | | | | | |
| CCPR1H | 16h | | 96h | | | | | |
| CCP1CON | 17h | | 97h | | | | | |
| RCSTA | 18h | TXSTA | 98h | | | | | |
| TXREG | 19h | SPBRG | 99h | | | | | |
| RCREG | 1Ah | | 9Ah | | | | | |
| CCPR2L | 1Bh | | 9Bh | | | | | |
| CCPR2H | 1Ch | | 9Ch | | | | | |
| CCP2CON | 1Dh | | 9Dh | | | | | |
| ADRESH | 1Eh | ADRESL | 9Eh | | | | | |
| ADCON0 | 1Fh | ADCON1 | 9Fh | | | | | |
| | 20h | | A0h | | 120h | | 1A0h | |
| General Purpose Register | | General Purpose Register | | accesses 20h-7Fh | | accesses A0h - FFh | | |
| 96 Bytes | | 96 Bytes | | | 16Fh | | 1EFh | |
| | 7Fh | | FFh | | 170h | | 1F0h | |
| Bank 0 | | Bank 1 | | Bank 2 | 17Fh | Bank 3 | 1FFh | |

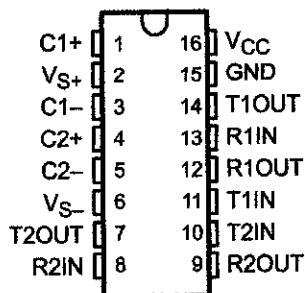
Unimplemented data memory locations, read as '0'.
 * Not a physical register.
Note 1: These registers are not implemented on 28-pin devices.
2: These registers are reserved, maintain these registers clear.

MAX232, MAX232I DUAL EIA-232 DRIVERS/RECEIVERS

SLLS0471 – FEBRUARY 1989 – REVISED OCTOBER 2002

- Meet or Exceed TIA/EIA-232-F and ITU Recommendation V.28
- Operate With Single 5-V Power Supply
- Operate Up to 120 kbit/s
- Two Drivers and Two Receivers
- ± 30 -V Input Levels
- Low Supply Current . . . 8 mA Typical
- Designed to be Interchangeable With Maxim MAX232
- ESD Protection Exceeds JESD 22 – 2000-V Human-Body Model (A114-A)
- Applications
 - TIA/EIA-232-F
 - Battery-Powered Systems
 - Terminals
 - Modems
 - Computers

MAX232 . . . D, DW, N, OR NS PACKAGE
MAX232I . . . D, DW, OR N PACKAGE
(TOP VIEW)



description/ordering information

The MAX232 is a dual driver/receiver that includes a capacitive voltage generator to supply EIA-232 voltage levels from a single 5-V supply. Each receiver converts EIA-232 inputs to 5-V TTL/CMOS levels. These receivers have a typical threshold of 1.3 V and a typical hysteresis of 0.5 V, and can accept ± 30 -V inputs. Each driver converts TTL/CMOS input levels into EIA-232 levels. The driver, receiver, and voltage-generator functions are available as cells in the Texas Instruments LinASIC™ library.

ORDERING INFORMATION

| TA | PACKAGE† | | ORDERABLE PART NUMBER | TOP-SIDE MARKING |
|---------------|---------------|---------------|-----------------------|------------------|
| 0°C to 70°C | PDIP (N) | Tube | MAX232N | MAX232N |
| | SOIC (D) | Tube | MAX232D | MAX232 |
| | | Tape and reel | MAX232DR | |
| | SOIC (DW) | Tube | MAX232DW | MAX232 |
| | | Tape and reel | MAX232DWR | |
| SOP (NS) | Tape and reel | MAX232NSR | MAX232 | |
| -40°C to 85°C | PDIP (N) | Tube | MAX232IN | MAX232IN |
| | SOIC (D) | Tube | MAX232ID | MAX232I |
| | | Tape and reel | MAX232IDR | |
| | SOIC (DW) | Tube | MAX232IDW | MAX232I |
| | | Tape and reel | MAX232IDWR | |

† Package drawings, standard packing quantities, thermal data, symbolization, and PCB design guidelines are available at www.ti.com/sc/package.



Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

LinASIC is a trademark of Texas Instruments.

PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

**TEXAS
INSTRUMENTS**

POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

Copyright © 2002, Texas Instruments Incorporated

Function Tables

EACH DRIVER

| INPUT T ₁ N | OUTPUT T ₁ OUT |
|---------------------------|------------------------------|
| L | H |
| H | L |

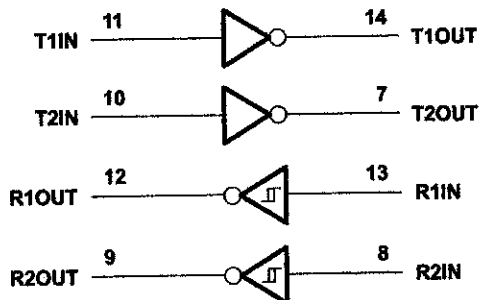
H = high level, L = low level

EACH RECEIVER

| INPUT R ₁ N | OUTPUT R ₁ OUT |
|---------------------------|------------------------------|
| L | H |
| H | L |

H = high level, L = low level

Logic diagram (positive logic)



MAX232, MAX232I DUAL EIA-232 DRIVERS/RECEIVERS

SLLS0471 – FEBRUARY 1989 – REVISED OCTOBER 2002

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)[†]

| | |
|--|--------------------------------------|
| Input supply voltage range, V_{CC} (see Note 1) | -0.3 V to 6 V |
| Positive output supply voltage range, V_{S+} | $V_{CC} - 0.3$ V to 15 V |
| Negative output supply voltage range, V_{S-} | -0.3 V to -15 V |
| Input voltage range, V_I : Driver | -0.3 V to $V_{CC} + 0.3$ V |
| Receiver | ±30 V |
| Output voltage range, V_O : T1OUT, T2OUT | $V_{S-} - 0.3$ V to $V_{S+} + 0.3$ V |
| R1OUT, R2OUT | -0.3 V to $V_{CC} + 0.3$ V |
| Short-circuit duration: T1OUT, T2OUT | Unlimited |
| Package thermal impedance, θ_{JA} (see Note 2): D package | 73°C/W |
| DW package | 57°C/W |
| N package | 67°C/W |
| NS package | 64°C/W |
| Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds | 260°C |
| Storage temperature range, T_{stg} | -65°C to 150°C |

[†] Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTE 1: All voltage values are with respect to network ground terminal.

2. The package thermal impedance is calculated in accordance with JESD 51-7.

recommended operating conditions

| | | MIN | NOM | MAX | UNIT |
|------------|---------------------------------------|---------|-----|-----|------|
| V_{CC} | Supply voltage | 4.5 | 5 | 5.5 | V |
| V_{IH} | High-level input voltage (T1IN, T2IN) | 2 | | | V |
| V_{IL} | Low-level input voltage (T1IN, T2IN) | | | 0.8 | V |
| R1IN, R2IN | Receiver input voltage | | | ±30 | V |
| T_A | Operating free-air temperature | MAX232 | 0 | 70 | °C |
| | | MAX232I | -40 | 85 | |

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted) (see Note 3 and Figure 4)

| PARAMETER | TEST CONDITIONS | MIN | TYP [‡] | MAX | UNIT |
|-------------------------|---|-----|------------------|-----|------|
| I_{CC} Supply current | $V_{CC} = 5.5$ V, All outputs open, $T_A = 25^\circ\text{C}$ | | 8 | 10 | mA |

[‡] All typical values are at $V_{CC} = 5$ V and $T_A = 25^\circ\text{C}$.

NOTE 3: Test conditions are C1–C4 = 1 μF at $V_{CC} = 5$ V \pm 0.5 V.



POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

30471 - FEBRUARY 1989 - REVISED OCTOBER 2002

DRIVER SECTION

Electrical characteristics over recommended ranges of supply voltage and operating free-air temperature range (see Note 3)

| PARAMETER | | TEST CONDITIONS | MIN | TYP† | MAX | UNIT |
|-----------------|------------------------------|--|-----|------|-----|------|
| VOH | High-level output voltage | T1OUT, T2OUT RL = 3 kΩ to GND | 5 | 7 | | V |
| VOL | Low-level output voltage‡ | T1OUT, T2OUT RL = 3 kΩ to GND | | -7 | -5 | V |
| RO | Output resistance | T1OUT, T2OUT V _{S+} = V _{S-} = 0, V _O = ±2 V | 300 | | | Ω |
| I _{SS} | Short-circuit output current | T1OUT, T2OUT V _{CC} = 5.5 V, V _O = 0 | | ±10 | | mA |
| | Short-circuit input current | T1IN, T2IN V _I = 0 | | | 200 | μA |

† Typical values are at V_{CC} = 5 V, T_A = 25°C.
 ‡ The algebraic convention, in which the least positive (most negative) value is designated minimum, is used in this data sheet for logic voltage levels only.

§ Not more than one output should be shorted at a time.

TE 3: Test conditions are C1-C4 = 1 μF at V_{CC} = 5 V ± 0.5 V.

Switching characteristics, V_{CC} = 5 V, T_A = 25°C (see Note 3)

| PARAMETER | | TEST CONDITIONS | MIN | TYP | MAX | UNIT |
|-------------------|------------------------------------|------------------------------------|-----|-----|-----|--------|
| t _r | Driver slew rate | RL = 3 kΩ to 7 kΩ, See Figure 2 | | | 30 | V/μs |
| t _{z(t)} | Driver transition region slew rate | See Figure 3 | | 3 | | V/μs |
| | Data rate | One TOUT switching | | 120 | | kbit/s |

TE 3: Test conditions are C1-C4 = 1 μF at V_{CC} = 5 V ± 0.5 V.

RECEIVER SECTION

Electrical characteristics over recommended ranges of supply voltage and operating free-air temperature range (see Note 3)

| PARAMETER | | TEST CONDITIONS | MIN | TYP† | MAX | UNIT |
|------------------|---|--|-----|------|-----|------|
| VOH | High-level output voltage | R1OUT, R2OUT I _{OH} = -1 mA | 3.5 | | | V |
| VOL | Low-level output voltage‡ | R1OUT, R2OUT I _{OL} = 3.2 mA | | | 0.4 | V |
| V _{I+} | Receiver positive-going input threshold voltage | R1IN, R2IN V _{CC} = 5 V, T _A = 25°C | | 1.7 | 2.4 | V |
| V _{I-} | Receiver negative-going input threshold voltage | R1IN, R2IN V _{CC} = 5 V, T _A = 25°C | 0.8 | 1.2 | | V |
| V _{IYS} | Input hysteresis voltage | R1IN, R2IN V _{CC} = 5 V | 0.2 | 0.5 | 1 | V |
| | Receiver input resistance | R1IN, R2IN V _{CC} = 5, T _A = 25°C | 3 | 5 | 7 | kΩ |

† Typical values are at V_{CC} = 5 V, T_A = 25°C.

‡ The algebraic convention, in which the least positive (most negative) value is designated minimum, is used in this data sheet for logic voltage levels only.

TE 3: Test conditions are C1-C4 = 1 μF at V_{CC} = 5 V ± 0.5 V.

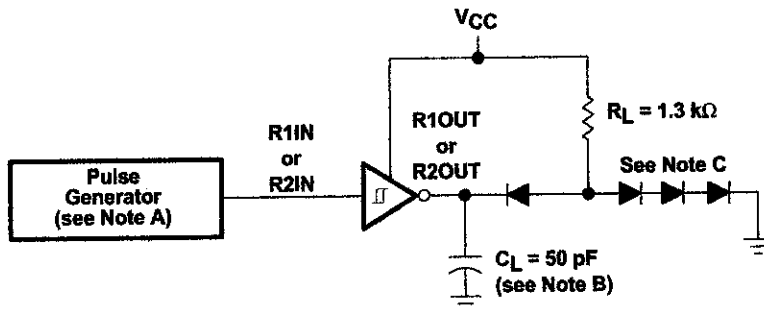
Switching characteristics, V_{CC} = 5 V, T_A = 25°C (see Note 3 and Figure 1)

| PARAMETER | | TYP | UNIT |
|---------------------|--|-----|------|
| t _{PLH(R)} | Receiver propagation delay time, low- to high-level output | 500 | ns |
| t _{PHL(R)} | Receiver propagation delay time, high- to low-level output | 500 | ns |

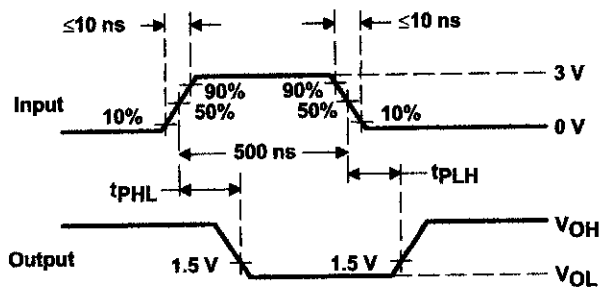
TE 3: Test conditions are C1-C4 = 1 μF at V_{CC} = 5 V ± 0.5 V.



PARAMETER MEASUREMENT INFORMATION



TEST CIRCUIT



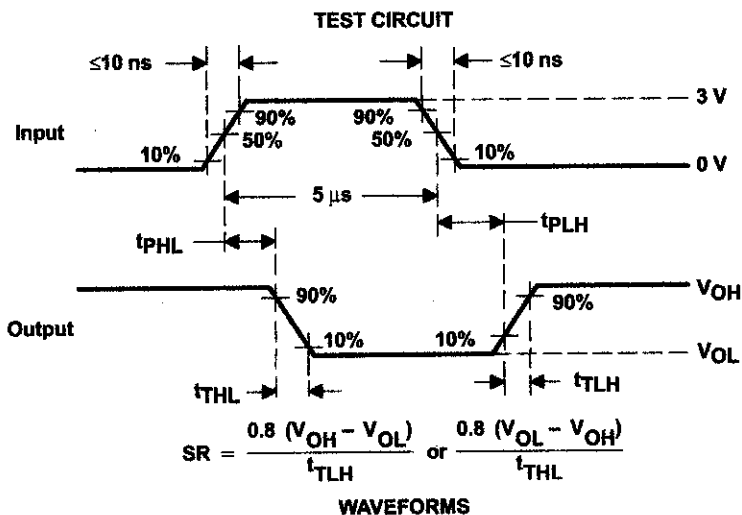
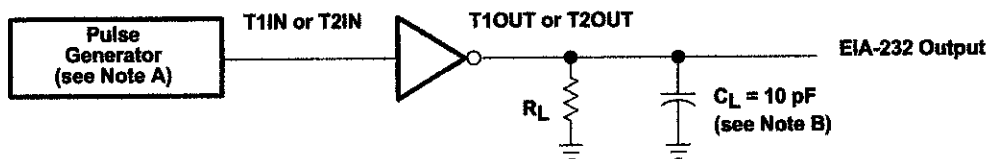
WAVEFORMS

- NOTES: A. The pulse generator has the following characteristics: $Z_O = 50 \Omega$, duty cycle $\leq 50\%$.
 B. C_L includes probe and jig capacitance.
 C. All diodes are 1N3064 or equivalent.

Figure 1. Receiver Test Circuit and Waveforms for t_{PHL} and t_{PLH} Measurements

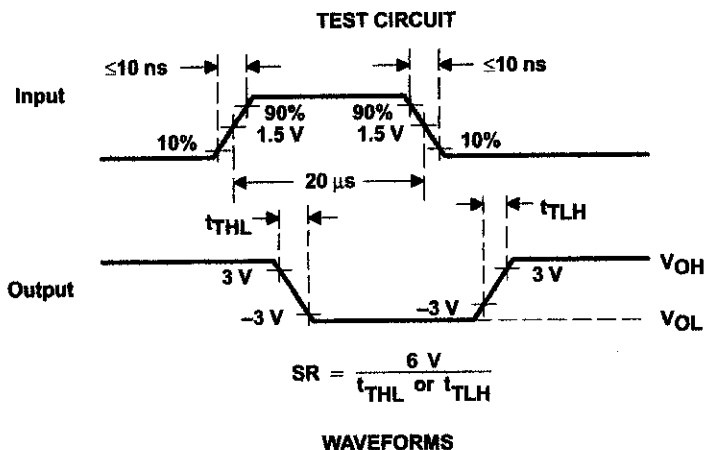
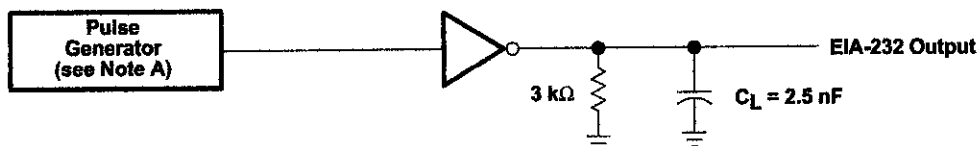
30471—FEBRUARY 1989—REVISED OCTOBER 2002

PARAMETER MEASUREMENT INFORMATION



- NOTES: A. The pulse generator has the following characteristics: $Z_O = 50 \Omega$, duty cycle $\leq 50\%$.
B. C_L includes probe and jig capacitance.

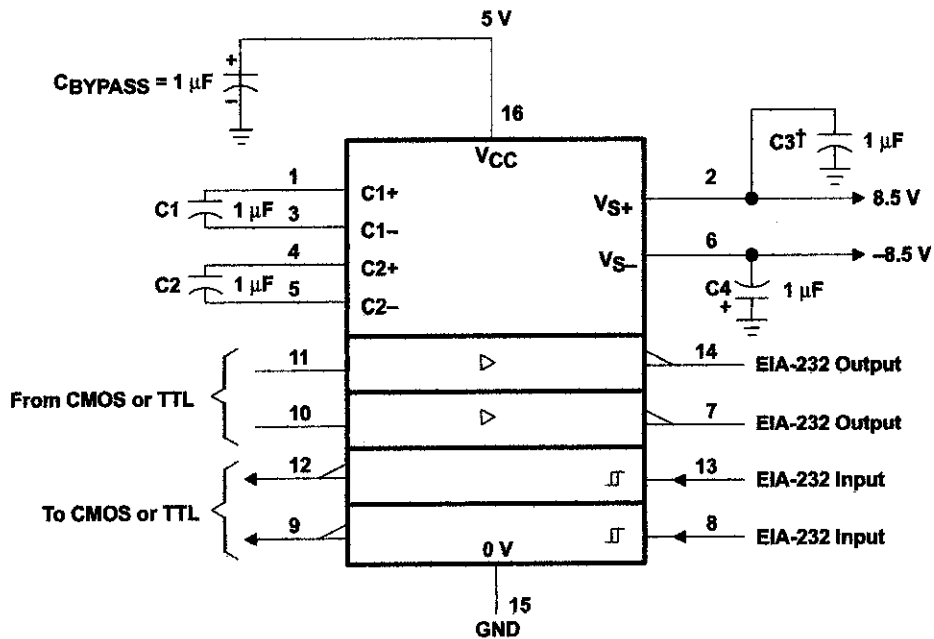
Figure 2. Driver Test Circuit and Waveforms for t_{PHL} and t_{PLH} Measurements (5- μ s Input)



- NOTE A: The pulse generator has the following characteristics: $Z_O = 50 \Omega$, duty cycle $\leq 50\%$.

Figure 3. Test Circuit and Waveforms for t_{THL} and t_{TLH} Measurements (20- μ s Input)

APPLICATION INFORMATION



† C3 can be connected to V_{CC} or GND.

Figure 4. Typical Operating Circuit

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

APPENDIX F

SERVO CONTROLLER C PROGRAM

```

//Servo program

#include <16f877.h>           //Standard include for 16F877 Chip
#fuses HS,NOPROTECT,NOWDT,NOBROWNOUT,NOLVP
#use delay(clock=4000000)    //Delay clock is 4Mhz
#byte PORTA = 5             //Port A is "File Address 05H" in "Bank 0"
#byte PORTB = 6             //Port B is "File Address 06H" in "Bank 0"
#byte PORTC = 7             //Port C is "File Address 07H" in "Bank 0"
#byte PORTD = 8             //Port D is "File Address 08H" in "Bank 0"
#define ALL_OUT 0           //Constant to set data direction register to output
#define ALL_IN 0xFF        //Constant to set data direction register to input

int serv0,serv1,serv2,serv3,serv4,serv5,serv6,serv7,serv8,serv9; //Set integer

void update()                //Update function
{
    int ser,pos;              //Set servo and position as integer
    ser=PORTC;                //Read data from PORTC and determine servo
    pos=PORTA;                //Read data from PORT A for position
    if(ser==0){serv0=pos;}    //Select servo 0 if 0 and put position value in serv0
    else if(ser==1){serv1=pos;} //Select servo 1 if 1 and put position value in serv1
    else if(ser==2){serv2=pos;} //Select servo 2 if 2 and put position value in serv2
    else if(ser==3){serv3=pos;} //Select servo 3 if 3 and put position value in serv3
    else if(ser==4){serv4=pos;} //Select servo 4 if 4 and put position value in serv4
    else if(ser==5){serv5=pos;} //Select servo 5 if 5 and put position value in serv5
    else if(ser==6){serv6=pos;} //Select servo 6 if 6 and put position value in serv6
    else if(ser==7){serv7=pos;} //Select servo 7 if 7 and put position value in serv7
    else if(ser==8){serv8=pos;} //Select servo 8 if 8 and put position value in serv8
    else if(ser==9){serv9=pos;} //Select servo 9 if 9 and put position value in serv9
}

main () {
    int del1,del2;           //Set integer delay 1 and delay 2

    set_tris_A(ALL_IN);     //Set all bits(byte) in Port A to input
    set_tris_C(ALL_IN);     //Set all bits(byte) in Port C to input
    set_tris_B(ALL_OUT);    //Set all bits(byte) in Port B to output
    set_tris_D(ALL_OUT);    //Set all bits(byte) in Port D to output

    delay_ms(10);
    //initialize servo center position
    serv0=37;                //Default center position for servo 0
    serv1=37;                //Default center position for servo 1
    serv2=37;                //Default center position for servo 2
    serv3=37;                //Default center position for servo 3
    serv4=37;                //Default center position for servo 4
    serv5=37;                //Default center position for servo 5
    serv6=37;                //Default center position for servo 6
    serv7=27;                //Default center position for servo 7
    serv8=37;                //Default center position for servo 8
    serv9=37;                //Default center position for servo 9
    delay_ms(10);

    do {
        //pulse width modulation generator

        //servo 0
        update();            //Go to Update function
        del1=serv0*4;        //Delay 1 value (pulse duration delay)
        del2=255-del1;       //Delay 2 value (delay for neutral)

        PORTB=0x80;          //Output to servo 0 at RB7
        PORTD=0x00;          //No output at PORT D
        delay_us(980);       //Minimum pulse width of 0.98ms
    }
}

```

```

delay_us(del1); //Use Delay 1 to delay the pulse width
delay_us(del1); //Use Delay 1 to delay the pulse width
delay_us(del1); //Use Delay 1 to delay the pulse width
delay_us(del1); //Use Delay 1 to delay the pulse width

PORTB=0x00; //No output to servo 0
PORTD=0x00; //No output at PORT D
delay_us(del2); //Use Delay 2 to delay for 0 amplitude
delay_us(del2); //Use Delay 2 to delay for 0 amplitude
delay_us(del2); //Use Delay 2 to delay for 0 amplitude
delay_us(del2); //Use Delay 2 to delay for 0 amplitude

//servo 1 //Go to Update function
update(); //Delay 1 value (pulse duration delay)
del1=serv1*4; //Delay 2 value (delay for neutral)
del2=255-del1; //Output to servo 1 at RB6
PORTB=0x40; //No output at PORT D
PORTD=0x00; //Minimum pulse width of 0.98ms
delay_us(980); //Use Delay 1 to delay the pulse width
delay_us(del1); //Use Delay 1 to delay the pulse width
delay_us(del1); //Use Delay 1 to delay the pulse width
delay_us(del1); //Use Delay 1 to delay the pulse width
delay_us(del1); //Use Delay 1 to delay the pulse width

PORTB=0x00; //No output to servo 1
PORTD=0x00; //No output at PORT D
delay_us(del2); //Use Delay 2 to delay for 0 amplitude
delay_us(del2); //Use Delay 2 to delay for 0 amplitude
delay_us(del2); //Use Delay 2 to delay for 0 amplitude
delay_us(del2); //Use Delay 2 to delay for 0 amplitude

//servo 2 //Go to Update function
update(); //Delay 1 value (pulse duration delay)
del1=serv2*4; //Delay 2 value (delay for neutral)
del2=255-del1;

PORTB=0x20; //Output to servo 2 at RB5
PORTD=0x00; //No output at PORT D
delay_us(980); //Minimum pulse width of 0.98ms
delay_us(del1); //Use Delay 1 to delay the pulse width
delay_us(del1); //Use Delay 1 to delay the pulse width
delay_us(del1); //Use Delay 1 to delay the pulse width
delay_us(del1); //Use Delay 1 to delay the pulse width

PORTB=0x00; //No output to servo 2
PORTD=0x00; //No output at PORT D
delay_us(del2); //Use Delay 2 to delay for 0 amplitude
delay_us(del2); //Use Delay 2 to delay for 0 amplitude
delay_us(del2); //Use Delay 2 to delay for 0 amplitude
delay_us(del2); //Use Delay 2 to delay for 0 amplitude

//servo 3 //Go to Update function
update(); //Delay 1 value (pulse duration delay)
del1=serv3*4; //Delay 2 value (delay for neutral)
del2=255-del1;

PORTB=0x10; //Output to servo 3 at RB4
PORTD=0x00; //No output at PORT D
delay_us(980); //Minimum pulse width of 0.98ms
delay_us(del1); //Use Delay 1 to delay the pulse width
delay_us(del1); //Use Delay 1 to delay the pulse width
delay_us(del1); //Use Delay 1 to delay the pulse width
delay_us(del1); //Use Delay 1 to delay the pulse width

PORTB=0x00; //No output to servo 3
PORTD=0x00; //No output at PORT D
delay_us(del2); //Use Delay 2 to delay for 0 amplitude
delay_us(del2); //Use Delay 2 to delay for 0 amplitude
delay_us(del2); //Use Delay 2 to delay for 0 amplitude
delay_us(del2); //Use Delay 2 to delay for 0 amplitude

//servo 4 //Go to Update function
update(); //Delay 1 value (pulse duration delay)
del1=serv4*4; //Delay 2 value (delay for neutral)
del2=255-del1;

```



```

PORTB=0x08;          //Output to servo 4 at RB3
PORTD=0x00;          //No output at PORT D
delay_us(980);       //Minimum pulse width of 0.98ms
delay_us(del1);      //Use Delay 1 to delay the pulse width
delay_us(del1);      //Use Delay 1 to delay the pulse width
delay_us(del1);      //Use Delay 1 to delay the pulse width
delay_us(del1);      //Use Delay 1 to delay the pulse width

PORTB=0x00;          //No output to servo 4
PORTD=0x00;          //No output at PORT D
delay_us(del2);      //Use Delay 2 to delay for 0 amplitude
delay_us(del2);      //Use Delay 2 to delay for 0 amplitude
delay_us(del2);      //Use Delay 2 to delay for 0 amplitude
delay_us(del2);      //Use Delay 2 to delay for 0 amplitude

//servo 5
update();            //Go to Update function
del1=serv5*4;        //Delay 1 value (pulse duration delay)
del2=255-del1;       //Delay 2 value (delay for neutral)
PORTB=0x04;          //Output to servo 5 at RB2
PORTD=0x00;          //No output at PORT D

delay_us(980);       //Minimum pulse width of 0.98ms
delay_us(del1);      //Use Delay 1 to delay the pulse width
delay_us(del1);      //Use Delay 1 to delay the pulse width
delay_us(del1);      //Use Delay 1 to delay the pulse width
delay_us(del1);      //Use Delay 1 to delay the pulse width

PORTB=0x00;          //No output to servo 5
PORTD=0x00;          //No output at PORT D
delay_us(del2);      //Use Delay 2 to delay for 0 amplitude
delay_us(del2);      //Use Delay 2 to delay for 0 amplitude
delay_us(del2);      //Use Delay 2 to delay for 0 amplitude
delay_us(del2);      //Use Delay 2 to delay for 0 amplitude

//servo 6
update();            //Go to Update function
del1=serv6*4;        //Delay 1 value (pulse duration delay)
del2=255-del1;       //Delay 2 value (delay for neutral)

PORTB=0x02;          //Output to servo 6 at RB1
PORTD=0x00;          //No output at PORT D
delay_us(980);       //Minimum pulse width of 0.98ms
delay_us(del1);      //Use Delay 1 to delay the pulse width
delay_us(del1);      //Use Delay 1 to delay the pulse width
delay_us(del1);      //Use Delay 1 to delay the pulse width
delay_us(del1);      //Use Delay 1 to delay the pulse width

PORTB=0x00;          //No output to servo 6
PORTD=0x00;          //No output at PORT D
delay_us(del2);      //Use Delay 2 to delay for 0 amplitude
delay_us(del2);      //Use Delay 2 to delay for 0 amplitude
delay_us(del2);      //Use Delay 2 to delay for 0 amplitude
delay_us(del2);      //Use Delay 2 to delay for 0 amplitude

//servo 7
update();            //Go to Update function
del1=serv7*4;        //Delay 1 value (pulse duration delay)
del2=255-del1;       //Delay 2 value (delay for neutral)

PORTB=0x01;          //Output to servo 7 at RB0
PORTD=0x00;          //No output at PORT D
delay_us(980);       //Minimum pulse width of 0.98ms
delay_us(del1);      //Use Delay 1 to delay the pulse width
delay_us(del1);      //Use Delay 1 to delay the pulse width
delay_us(del1);      //Use Delay 1 to delay the pulse width
delay_us(del1);      //Use Delay 1 to delay the pulse width

PORTB=0x00;          //No output to servo 7
PORTD=0x00;          //No output at PORT D
delay_us(del2);      //Use Delay 2 to delay for 0 amplitude
delay_us(del2);      //Use Delay 2 to delay for 0 amplitude
delay_us(del2);      //Use Delay 2 to delay for 0 amplitude
delay_us(del2);      //Use Delay 2 to delay for 0 amplitude

```

```

//servo 8
update();
del1=serv8*4;
del2=255-del1;

PORTB=0x00;
PORTD=0x80;
delay_us(980);
delay_us(del1);
delay_us(del1);
delay_us(del1);
delay_us(del1);

PORTB=0x00;
PORTD=0x00;
delay_us(del2);
delay_us(del2);
delay_us(del2);
delay_us(del2);

//servo 9
update();
del1=serv9*4;
del2=255-del1;

PORTB=0x00;
PORTD=0x40;
delay_us(980);
delay_us(del1);
delay_us(del1);
delay_us(del1);
delay_us(del1);

PORTB=0x00;
PORTD=0x00;
delay_us(del2);
delay_us(del2);
delay_us(del2);
delay_us(del2);

} while (TRUE);
}

//Go to Update function
//Delay 1 value (pulse duration delay)
//Delay 2 value (delay for neutral)

//No output at PORT B
//Output to servo 8 at RD7
//Minimum pulse width of 0.98ms
//Use Delay 1 to delay the pulse width
//Use Delay 1 to delay the pulse width
//Use Delay 1 to delay the pulse width
//Use Delay 1 to delay the pulse width

//No output at PORT B
//No output to servo 8
//Use Delay 2 to delay for 0 amplitude
//Use Delay 2 to delay for 0 amplitude
//Use Delay 2 to delay for 0 amplitude
//Use Delay 2 to delay for 0 amplitude

//Go to Update function
//Delay 1 value (pulse duration delay)
//Delay 2 value (delay for neutral)

//No output at PORT B
//Output to servo 0 at RD6
//Minimum pulse width of 0.98ms
//Use Delay 1 to delay the pulse width
//Use Delay 1 to delay the pulse width
//Use Delay 1 to delay the pulse width
//Use Delay 1 to delay the pulse width

//No output at PORT B
//No output to servo 9
//Use Delay 2 to delay for 0 amplitude
//Use Delay 2 to delay for 0 amplitude
//Use Delay 2 to delay for 0 amplitude
//Use Delay 2 to delay for 0 amplitude

//Endless routines (non stop operation)

```

APPENDIX G

WALKING CONTROLLER C PROGRAM

```

//Walking program

#include <16f877.h>           //Standard Include for 16F877 Chip
#fuses HS,NOPROTECT,NOWDT,NOBROWNOUT,NOLVP
#use delay(clock=8000000)    //Delay clock is 8MHz
#use rs232(baud=9600, xmit=PIN_A0, rcv=PIN_A1) //Set up RS232
#byte PORTB = 6              //Port B is "File Address 06H" in "Bank 0"
#byte PORTD = 8              //Port D is "File Address 08H" in "Bank 0"
#define ALL_OUT 0            //Constant to set data direction register to output
#define ALL_IN 0xFF         //Constant to set data direction register to input

//Set integer for servos output variables for position (level)
int ser0,ser1,ser2,ser3,ser4,ser5,ser6,ser7,ser8,ser9;

//Set adjusted servo center position (6 bits, 0-63 level resolution, default center is 37)
int sc0=37,sc1=37,sc2=37,sc3=37,sc4=37,sc5=37,sc6=37,sc7=37,sc8=37,sc9=37;

//put values onto buses/transmit to servo controller
void output(int serv,int posi) //Output function for variables servo and position
{
    PORTD=serv;                //PORTD output is servo selection
    PORTB=posi;                //PORTB output is servo position
    delay_ms(2);               //Delay for 2 ms
}

//put default value to servo registers/stand straight

void reset()                   //Reset function for default center position
{
    ser0=sc0;                  //Servo 0 center position is 37 (from default set values)
    ser1=sc1;                  //Servo 0 center position is 37
    ser2=sc2;                  //Servo 0 center position is 37
    ser3=sc3;                  //Servo 0 center position is 37
    ser4=sc4;                  //Servo 0 center position is 27
    ser5=sc5;                  //Servo 0 center position is 37
    ser6=sc6;                  //Servo 0 center position is 37
    ser7=sc7;                  //Servo 0 center position is 37
    ser8=sc8;                  //Servo 0 center position is 37
    ser9=sc9;                  //Servo 0 center position is 37
}

//calls the routine which send data/servos move as registers value
//update one servo at a time with 2ms gap between each (at output function)

void send()                    //Send function
{
    output(0,ser0);            //Call output function (put position value for servo 0)
    output(1,ser1);            //Call output function (put position value for servo 1)
    output(2,ser2);            //Call output function (put position value for servo 2)
    output(3,ser3);            //Call output function (put position value for servo 3)
    output(4,ser4);            //Call output function (put position value for servo 4)
    output(5,ser5);            //Call output function (put position value for servo 5)
    output(6,ser6);            //Call output function (put position value for servo 6)
    output(7,ser7);            //Call output function (put position value for servo 7)
    output(8,ser8);            //Call output function (put position value for servo 8)
    output(9,ser9);            //Call output function (put position value for servo 9)
}

//returns the decimal value from ascii character
//variable value is input character to be insert by user (number of steps)

int convert(int value)         //variable convert is result
{
    int result;
    if(value=='0'){result=0;} //if input variable is 0, result is decimal value 0
    if(value=='1'){result=1;} //if input variable is 1, result is decimal value 1
}

```

```

if(value=='2'){result=2;} //if input variable is 2, result is decimal value 2
if(value=='3'){result=3;} //if input variable is 3, result is decimal value 3
if(value=='4'){result=4;} //if input variable is 4, result is decimal value 4
if(value=='5'){result=5;} //if input variable is 5, result is decimal value 5
if(value=='6'){result=6;} //if input variable is 6, result is decimal value 6
if(value=='7'){result=7;} //if input variable is 7, result is decimal value 7
if(value=='8'){result=8;} //if input variable is 8, result is decimal value 8
if(value=='9'){result=9;} //if input variable is 9, result is decimal value 9
return result; //return result as reply
}

//position of servo 0, 4, 5, and 9 because position are dependent
//'+ ' tilt left, '-' tilt right

void ser0459(int op,int del) //ser0459 function for tilt
//set integer op for operation and del for level
{
//select if operation is + or tilt to the left
if(op=='+')
{
ser0=ser0-del; //decrease current servo 0 position by del level
ser4=ser4+del; //increase current servo 4 position by del level
ser5=ser5+del; //increase current servo 5 position by del level
ser9=ser9-del; //decrease current servo 9 position by del level
}
else if(op=='-') //select if operation is - or tilt to the left
{
ser0=ser0+del; //increase current servo 0 position by del level
ser4=ser4-del; //decrease current servo 4 position by del level
ser5=ser5-del; //decrease current servo 5 position by del level
ser9=ser9+del; //increase current servo 9 position by del level
}
}
//serv8 and serv1 routines automatically check for values so that the torso and feet are parallel
//this routines will makes the feet horizontally parallel to torso
//always uses these routines! Do not manually set ser8 and ser1 Except for special occasions

void serv8() //serv8 routine
{
signed pos6,pos7,temp; //signed integer pos6,pos7 and temp
pos6=sc6-ser6; //pos6 is servo 6 center position - current position
pos7=sc7-ser7; //pos7 is servo 7 center position - current position
temp=sc8; //temp is servo 8 center position
temp=temp+pos7+pos6; //calculate temp for changes
ser8=temp; //servo 8 position equal to temp value
}
void serv1() //serv1 routine
{
signed pos3,pos2,temp; //signed integer pos3,pos2 and temp
pos3=sc3-ser3; //pos3 is servo 3 center position - current position
pos2=sc2-ser2; //pos2 is servo 2 center position - current position
temp=sc1; //temp is servo 1 center position
temp=temp+pos2+pos3; //calculate temp for changes
ser1=temp; //servo 1 position equal to temp value
}
//routine to bend left leg, not a convention
// '-' to make it shorter, '+' it stretches

void bleft(int dir) //bleft routine for bend to left and set variable direction
{
//Select if to shorten the leg if variable input is -ve sign
if(dir=='-')
{
ser6=ser6+1; //increase current servo 6 position by 1 level
ser7=ser7-2; //decline current servo 7 position by 2 level
serv8(); //Call serv8 routine
}
else if(dir=='+') //Select if to stretches leg if variable input is +ve sign
{
ser6=ser6-1; //decrease current servo 6 position by 1 level
ser7=ser7+2; //increase current servo 6 position by 2 level
serv8(); //Call serv8 routine
}
}
//likewise for right leg

```

```

void bright(int dir) //bright function for bend to right and set integer direction
{
  if(dir=='-') //Select if to shorten leg if variable input is - ve sign
  {
    ser3=ser3-1; //decrease current servo 3 position by 1 level
    ser2=ser2+2; //increase current servo 2 position by 1 level
    serv1(); //Call serv1 routine
  }
  else if(dir=='+') //Select if to stretches leg if variable input is +ve sign
  {
    ser3=ser3+1; //increase current servo 3 position by 1 level
    ser2=ser2-2; //decline current servo 2 position by 1 level
    serv1(); //Call serv1 routine
  }
}

//'- to bend, '+' to unbend legs

void bend(int dir,int dela,int j) //bend function for leg bending
//variable dir (+ve if stretch, -ve if shorten)
//variable dela for hold time (control speed)
//variable j for magnitude
{
  int i; //set integer i
  for (i=1;i<=j;++i) //do for j times (depend on magnitude)
  {
    bleft(dir); //Call bleft routine and put variable dir
    bright(dir); //Call bright routine and put variable dir
    send(); //Call send function
    delay_ms(dela); //hold for variable dela ms
  }
}

//'- to tilt right, '+' to tilt left
void tilt(int dir,int dela,int j) //tilt function for tilting process
//variable dir for direction (+ve if left, -ve if right)
//variable dela for hold time (speed)
//variable j for magnitude
{
  int i; //set integer i
  for (i=1;i<=j;++i) //do for j times (depend on magnitude)
  {
    ser0459(dir,1); //Call ser0459 routine and use input variable dir
    send(); //Call send function
    delay_ms(dela); //hold for input variable dela in millisecond
  }
}

//walking routine
//walk(speed, number of steps, tilt magnitude)

void walk(int del,int steps,int ti) //walk routine, set variable for delay, steps and tilt
{
  int i;
  int k;
  //slight forward movement of upper torso //do for 5 times (for smooth movement do 1 by 1)
  for (i=1;i<=5;++i)
  {
    ser6=ser6+1; //decrease servo 6 position by 1 level
    ser3=ser3-1; //increase servo 3 position by 1 level
    send(); //Call send function
    delay_ms(del); //hold for del variable in millisecond
  }

  Hold current position //new servo 6 center position value
  sc6=ser6; //new servo 3 center position value
  sc3=ser3;

  //first step, tilt to the right, left leg steps forward
  //START

  //tilt to right to put right feet into center of gravity //call tilt function
  tilt('-',del,ti); //insert variable -ve sign to tilt to right, del and tilt

  //bend left leg to rise left feet for first step

```

```

for (i=1;i<=12;++i) //do until 12 times(for smooth movement)
{
    bleft('-'); //call bleft function and give -ve sign
    send(); //call send function
    delay_ms(del); //hold for variable del value in milliseconds
}

//stretches back the left leg
//the thigh stretches back for only 2/3 of origin
//the left shank stretches to origin and the left leg became straight.
for (i=1;i<=8;++i) //do until 8 times(for smooth movement)
{
    ser6=ser6-1; //decrease servo 6 position by 1 level
    ser7=ser7+3; //increase servo 7 position by 3 levels
    serv8(); //call serv8 routines
}

//meanwhile the right leg pushes backward to shift the entire body to front
if(i==1||i==3||i==5||i==7) //do at the same time for every 2 interval
{
    ser3=ser3+1; //increase servo 3 position by 1 level
    serv1(); //call serv1 routines
}
send(); //call send function
delay_ms(del); //hold for variable del in milliseconds
}

//tilt back to center with left foot kept upfront
tilt('+',del,ti); //call tilt function (+ve value to tilt to left)
//tilt to the left, right leg steps forward
for(k=1;k<=steps;++k) //Do until reaches number of steps
{
    //tilt to left to put left feet into center of gravity
    tilt('+',del,ti); //call tilt function
    //insert variable +ve sign to tilt to left, del and tilt
}

//stretches back left leg to normal
for (i=1;i<=4;++i) //do until 4 times(for smooth movement)
{
    ser6=ser6-1; //decrease servo 6 position by 1 level
    serv8(); //call serv8 routines
}

//bend right leg to rise right feet
bright('-'); //call bright function and give -ve sign
send(); //call send function
delay_ms(del); //hold for variable del value in milliseconds
}

//more rise on the right feet
for (i=1;i<=8;++i) //do until 8 times(for smooth movement)
{
    if(i>0&& i<5) //do for every count 1 to 4 (0<i<5)
    {
        ser3=ser3-1; //decrease servo 3 position by 1 level
    }
    ser3=ser3-1; //decrease servo 3 position by 1 level
    ser2=ser2+1; //increase servo 2 position by 1 level
    serv1(); //call serv1 routines
    send(); //call send function
    delay_ms(del); //hold for variable del value in milliseconds
}

//stretches back the right leg
//the thigh stretches back for only 50% of origin
//the right shank stretches to origin and the right leg became straight.
for (i=1;i<=8;++i) //do until 8 times(for smooth movement)
{
    ser3=ser3+1; //increase servo 3 position by 1 level
    ser2=ser2-2; //decrease servo 2 position by 2 level
    serv1(); //call serv1 routines
}

//left leg pushes backward to shift the entire body to front
if(i==1||i==3||i==5||i==7) //do at the same time for every 2 interval
{
    ser6=ser6-1; //decrease servo 6 position by 1 level
}

```

```

serv8(); //call serv8 routines
}
send(); //call send function
delay_ms(del); //hold for variable del value in milliseconds
}

//tilt back to center with right foot kept upfront
tilt('-',del,ti); //call tilt function
//insert variable -ve sign to tilt to right, del and tilt

//tilt to the right, left leg steps forward

//tilt to right to put right feet into center of gravity
tilt('-',del,ti); //call tilt function
//insert variable -ve sign to tilt to right, del and tilt

//stretches back right leg to normal
for (i=1;i<=4;++i) //do until 4 times(for smooth movement)
{
ser3=ser3+1; //increase servo 3 position by 1 level
serv1(); //call serv1 routines

bleft('-'); //bend left leg to rise left feet
send(); //call bleft function and give -ve sign
delay_ms(del); //call send function
//hold for variable del value in milliseconds
}

//more rise on the left feet
for (i=1;i<=8;++i) //do until 8 times(for smooth movement)
{
if(i>0&& i<5) //do for every count 1 to 4 (0<i<5)
{
ser6=ser6+1; //increase servo 6 position by 1 level
}
ser6=ser6+1; //increase servo 6 position by 1 level
ser7=ser7-1; //decrease servo 7 position by 1 level
serv8(); //call serv8 routines
send(); //call send function
delay_ms(del); //hold for variable del value in milliseconds
}

//stretches back the left leg
//the thigh stretches back for only 50% of origin
//the left shank stretches to origin and the right leg became straight.

for (i=1;i<=8;++i) //do until 8 times(for smooth movement)
{
ser6=ser6-1; //decrease servo 6 position by 1 level
ser7=ser7+2; //increase servo 7 position by 2 level
serv8(); //call serv8 routines

//right leg pushes slightly forward to maintain center of gravity
if(i==1||i==3||i==5||i==7) //do at the same time for every 2 interval
{
ser3=ser3+1; //increase servo 3 position by 1 level
serv1(); //call serv1 routines
}

send(); //call send function
delay_ms(del); //hold for variable del value in milliseconds
}

//tilt back to center with left foot kept upfront
tilt('+',del,ti); //call tilt function
//insert variable +ve sign to tilt to left, del and tilt
}

//stop routine with left leg at front

//tilt to left to put left feet into center of gravity
tilt('+',del,ti); //call tilt function
//insert variable +ve sign to tilt to left, del and tilt

//stretches back left leg to origin
for (i=1;i<=4;++i) //do until 4 times(for smooth movement)
{

```

```

        ser6=ser6-1;
        serv8();

        //bend right leg to rise right feet
        bright('-');
        send();
        delay_ms(del);
    }

    //more rise on the right feet
    for (i=1;i<=4;++i)
    {
        ser3=ser3-1;
        serv1();
        send();
        delay_ms(del);
    }

    //stretches back the right leg
    //the right thigh stretches for only 50% of origin but became parallel to left leg
    //the left shank stretches to origin and the left leg became straight

    for (i=1;i<=4;++i)
    {
        bright('+');
        send();
        delay_ms(del);
    }

    //tilt back to center with both feet at origin
    tilt('-',del,ti);

    //done positioning

    for (i=1;i<=5;++i)
    {
        //slight backward movement of upper torso to it origin
        ser6=ser6-1;
        ser3=ser3+1;
        send();
        delay_ms(del);
    }

    sc6=ser6;
    sc3=ser3;
}

```

```

//decrease servo 6 position by 1 level
//call serv8 routines

//call bright function and give -ve sign
//call send function
//hold for variable del value in milliseconds

//do until 4 times(for smooth movement)
//decrease servo 3 position by 1 level
//call serv1 routines
//call send function
//hold for variable del in milliseconds

//do until 4 times(for smooth movement)
//call send function
//do at the same time for every 2 interval

//call tilt function
//insert variable --ve sign to tilt to right, del and tilt

//do for 5 times(for smooth movement)
//decrease servo 6 position by 1 level
//increase servo 3 position by 1 level
//Call send function
//hold for del variable in millisecond

//new servo 6 center position
//new servo 3 center position

```


APPENDIX H

COMPUTER INSTRUCTION C PROGRAM

```

main () {

    int i,s;
    int act;

    set_tris_B(ALL_OUT);
    set_tris_D(ALL_OUT);
    set_tris_C(ALL_OUT);
    set_tris_E(ALL_OUT);

    reset();
    delay_ms(1000);
    send();
    do {
        i=0;
        s=0;

        //ask user to select action
        //if input is 0; the robot will bend the leg
        //if input is 1; the robot will stretch the leg
        //if input is 2; the robot will tilt the leg to the right
        //if input is 3; the robot will tilt the leg to the left
        //if input is 4; ask user to enter number of step (1-9) for fast walk
        //if input is 5; ask user to enter number of step (1-9) for medium walk
        //if input is 6; ask user to enter number of step (1-9) for slow walk

        printf("\nSelect action : ");
        do {
            act=getc();
        } while (act !='0' && act !='2' && act !='3' && act !='4'&& act !='5'&& act !='6'&& act
            !='7'&& act !='8'&& act !='9');
        printf("%c\r",act);

        if(act=='0'){bend('-',90,6);}
        else if(act=='1'){bend('+',90,6);}
        else if(act=='2'){tilt('-',90,5);}
        else if(act=='3'){tilt('+',90,5);}
        else if(act=='4')
        {
            printf("\nFast walk\r");
            printf("\nNumber of steps : ");

            do {
                s=getc();
            } while (act !='1'&& act !='2'&&act !='3'&&act !='4'&&act !='5'&&act
                !='6'&&act !='7'&& act !='8'&&act !='9');
            printf("%c\r",s);
            s=convert(s);
            walk(30,s,4);
        }
        else if(act=='5')
        {
            printf("\nMed walk\r");
            printf("\nNumber of steps : ");
            do {
                s=getc();
            } while (act !='1'&& act !='2'&&act !='3'&&act !='4'&&act !='5'&&act
                !='6'&&act !='7'&& act !='8'&&act !='9');
            printf("%c\r",s);
            s=convert(s);
            walk(60,s,4);
        }
        else if(act=='6')
        {
            printf("\nSlow walk\r");
            printf("\nNumber of steps : ");

```

```
do {
    s=getc();
    } while (act !='1'&& act !='2'&&act !='3'&&act !='4'&&act !='5'&&act
!= '6'&&act !='7'&& act !='8'&&act !='9');
printf("%c\n",s);
s=convert(s);
walk(250,s,5);
}
else if(act=='r'){reset();}
else {send();}
} while (TRUE);
}
```

