Fall 1994

Biofeedback Gait Training Auditory Vs. Visual Techniques

Timothy D. Hiemenz
Old Dominion University

BIOFEEDBACK GAIT TRAINING:
AUDITORY vs. VISUAL TECHNIQUES

Timothy D. Hiemenz

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment
of the Requirements for the Degree of

MASTER OF SCIENCE

ELECTRICAL ENGINEERING

OLD DOMINION UNIVERSITY

December 1994

Approved by:

Linda L. Vahala (Director)

Martin D. Meyer

John W. Stoughton

Martha L. Walker

ABSTRACT

**BIOFEEDBACK GAIT TRAINING:
AUDITORY vs. VISUAL TECHNIQUES**

Timothy D. Hiemenz
Old Dominion University, 1994
Director: Dr. Linda Vahala

Training lower extremity amputees to walk normally is quite a difficult task. Amputees must wear a prosthesis so they can walk at all. This allows them some mobility, but their walking pattern may be unnatural. If their gait is temporally asymmetric, they need to exert more energy to move about. This research was initiated to help lower extremity amputees to walk more efficiently using biofeedback gait training. Two types of feedback were developed and tested to determine which method gave the most understandable feedback, validating its use in a clinical setting.

A normal gait cycle uses the lower limbs to create a locomotive pattern which is very energy efficient. If one leg requires more time to step than the other leg, the gait cycle is asymmetrical. This will require the amputee to do a considerable amount of work to walk. Due to the lopsided temporal patterns of the normal and amputated legs, the amputee will tire quickly. Training amputees to balance their

gait cycle will lessen the amount of energy required to maneuver and allow them to walk for longer periods of time.

This study includes the design and experimentation of a Biofeedback Gait Trainer. The device includes modules for training with auditory or visual feedback. The biofeedback in each case is meant to help the subject realize their deviation and strive to correct it by responding to the feedback signals. Some gait trainers have been designed with audio feedback and others with visual feedback. These are usually very expensive and therefore uncommon contraptions. This thesis describes the development of a low-cost gait training aid and determines whether a patient understands auditory or visual feedback signals better. Lower extremity amputees were tested with the gait timer, to acquire an accurate measure of their initial walking pattern. Then, they were trained with auditory feedback and visual feedback. Under these circumstances, their temporal gait patterns were examined, in order to determine which feedback technique provided the more understandable information.

ACKNOWLEDGEMENTS

I wish to thank my parents, Dr. and Mrs. Donald W. Hiemenz, for their guidance, understanding, and support. Without the education they provided me, my undergraduate and graduate degrees would have been only a dream. I want to extend a great appreciation to my grandfather, Mr. Anthony S. Noto, as well, for triggering my interest in Electronics, and influencing me so greatly.

I would also like to thank my advisor, Dr. Linda L. Vahala, for her support. Special thanks to Dr. Marty D. Meyer and Martha L. Walker, M.S.P.T, for their wisdom and guidance. Dr. Meyer helped me through to the end of the project, so I thank him tremendously. And without the knowledge and support from Martha Walker in the Old Dominion Motion Analysis Center, none of this would have been possible.

Thanks to the staff of Riverside Prosthetics and Orthotics, Newport News, VA for their interest and involvement in acquiring subjects. And a special thanks to the amputees who helped out by allowing themselves to be tested and trained with this new device.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Pathology of Amputees

Almost every person that has had one of their legs amputated wears a prosthesis. Their prosthesis is necessary if they want to walk or do any number of things that requires them to stand. Most of these amputees once had both of their legs and know what it means to walk normally. Due to their condition, they no longer walk as they did, and often find themselves struggling to do what others take for granted.

Wearing a prosthesis allows the amputee to move about, but also creates new problems. They have no sense of knowing exactly when they make contact with the floor when they step with their prosthetic leg, so walking with a prosthesis takes extra caution. Amputees must also swing their prosthesis to where they want to tread, being careful not to take too large of a step. They can easily lose their balance if they move ahead in too large of a step. Since there are no muscles in the prosthesis, as there are in a normal leg, the amputee must be certain that they are taking a safe step. Becoming a little off balance when bearing all body weight solely on the prosthesis could lead to the person falling over.

The prosthesis may feel like a very foreign object to the amputee. It is dead weight that the person may feel strapped to. In order to walk, an amputee may swing the prosthesis in a half circle motion in order to place it for the next step. Ideally, the leg should swing directly forward, but the prosthesis may drag if the knee is not raised high enough during a step. Not realizing this, the amputee may swing their prosthesis out and around to stop the dragging. That, in effect, makes the step time longer and the gait cycle more askew. Taking all of these factors into consideration, one can understand why it may require more time for the amputated leg to step than it does for the good leg, therefore making the temporal gait pattern asymmetric. Also, if the person feels insecure about putting all of their weight on the prosthesis, he will tend to step quickly to get off of it as soon as possible.

1.2 Gait Analyzers

Training prosthesis wearing amputees to walk better is quite difficult. Most physical therapists can work with the amputees to guide them to step more correctly, based on observed motions, but their advice may not always be accurate. Rarely can they spot small differences in their step timing because they do not have perfectly calibrated vision.

> Even very small aberrations in the walking pattern are conspicuously evident to an observer without the recourse to any instrumentation and observational gait analysis is still the most widely applied method of assessment in clinical use. Such purely objective methods are necessarily prone to observer error, which may arise from inadequate training, the limitations of visual perceptive ability and perhaps a subconscious personal bias towards an expected or sought-for-result.(Law 115)

Since small step timing differences are hard to detect, the use of electronic gait analyzers has become more popular. The difficulty of recording accurate gait data has limited the ability of clinicians to rehabilitate patients effectively,(Roth 10) so many have looked to technological advances for precise measurement of step times and distances. This may be due to the fact that the temporal and distance factors are among the most important and the most available, of all the different aspects of gait.(Roth 10)

A number of gait measurement devices have been developed.(Roth 10) Most devices that study gait patterns, will measure time and distance between steps.(Wall et al. 187) This is convenient because velocity and other information can be calculated from the data. Many of these systems use calibrated walkways where the patient must walk in order for data to be taken.(Bajd and Kralj 8; Gabel et al. 543; Crouse et al. 65; Gifford and Hutton 45) Many of these walkways are made to be portable. Other systems would film the feet from beneath a transparent walkway, but these have their difficulties.(Wall et al. 187) Many motion analysis laboratories use video data from a number of cameras to analyze gait data, but in these systems, foot/floor contact data is inaccurate. Some systems connect different contraptions to the feet to detect movement. One system in the literature describes how strips of tape were attached to the back of the feet. The strips of tape had evenly spaced holes punched along the whole length. The system detected the speed of the tape when it moved through an optical sensor to calculate the temporal gait data.(Law 117) Others use foot switches attached to or built into a pair of shoes, which trigger when foot/floor contact is made.(Bajd and Kralj 129) These are all useful devices to provide the clinician with useful information about the patient's gait pattern. The clinician then uses the data to instruct the patient how to walk better. The patient is not given the details of his gait pattern deviations, because he would not understand them.

The cost of the devices are also a factor. Only a few motion laboratories can afford expensive equipment, and for daily clinical use, a less expensive system is required.(Bajd and Kralj 129) Previous systems were "bulky, cumbersome, and expensive."(Roth 15) In 1980, the most likely possibility of inexpensive gait measurement would have involved a personal computer.(Bajd and Kralj 131) This should still be true, because most laboratories and clinics already have a personal computer, and their prices have dropped while the number of features have increased.

1.3 Gait Trainers

Other systems that have been designed actually give feedback to a patient as they walk. One such system uses a speaker, a tone generator, and a monitor to inform the patient how well they are walking. It requires the use of a special walkway where the patient must walk.(Hirokawa and Matsumura 8) The monitor will give them visual feedback by showing them the actual and desired locations of their steps. Depending on the pitch of the tone, the patient is informed if they are stepping too quickly, too slowly, or on time.(Hirokawa and Matsumura 10) Another system used audible feedback to train a patient to walk with symmetrical step lengths. Again the pitch of the tone informed the subject of their abnormality. (Gabel et al. 543) The continuous biofeedback helps the person recognize their problems and allows them to try to overcome them, because the biofeedback offers a type of therapy that normally is not available, and grants the person some participation in their recovery.(Seeger et al. 364)

A low cost method for biofeedback gait training would give more patients the ability to receive treatment, because more facilities could afford the equipment. A useful way to convey the information back to the patient must be determined in order for them to realize and adjust for the deviations of the gait cycle. This necessitates that the most effective type of biofeedback be used to achieve the best results.

1.4 Objective and Overview

This research was initiated to determine whether visual or auditory signals provide the more understandable biofeedback. A gait timer was designed using a microcontroller and remote signalling foot switches. The device was then enhanced to include two types of biofeedback. One biofeedback method uses a timer and a buzzer that will sound if a step has not been taken by a certain amount of time. This auditory method is meant to provide the patient with immediate knowledge of their abnormality. The other method uses visual biofeedback. It will turn on a number of lights representing the difference in the step times of each leg, so that symmetry can be achieved. At the end of each stride, the patient will be able to see where deviations occur, and be able to adjust for another attempt.

This thesis includes a discussion of gait analysis in Chapter 2, in order to present related medical terminology and information. In Chapter 3, the design of the gait trainer, including its circuitry and capabilities, is presented. Next, Chapter 4 describes the software programs that were written and used to allow the device to work as described in the most user-friendly environment available. The device was then tested using both the biofeedback methods described. Patients were trained with the apparatus and the data was examined to determine the advantages of each method. The results of the testing and training sessions are presented in Chapter 5. In

Chapter 6, conclusions are made as well as some recommendations for future developments.

CHAPTER 2

GAIT ANALYSIS

2.1 Introduction

In order to understand the concepts of gait analysis that have been researched in this paper, it will be helpful to explain some medical terminology. Gait is defined as the repetitive limb motion that propels the body forward, or more simply walking. "Human gait is an intricate combination of the neuromuscular and skeletal systems working together to produce a smooth and efficient form of locomotion."(Cheung et al. 131) When a person walks, one leg swings forward while the other is planted. Then, after that foot is planted, the other swings forward. Walking requires that both feet be in contact with the floor before another step is taken. If a person takes a step before the other has reached the floor, they are in the running mode of locomotion. This paper will only describe and study the phases of walking.

After discussing some of the terminology used in gait analysis, it will be helpful to address the pathology of leg amputees. This chapter also examines the importance of efficient locomotion and how amputees walk with a prosthesis.

Then, the topic of biofeedback as a clinical aid is considered.

2.2 Terminology

A gait cycle consists of sequential steps that start and end with an initial contact of a foot with the floor. Therefore, a gait cycle starting with a left foot/floor contact would end with the next left foot/floor contact. In this case a right foot/floor contact would occur between the two left foot/floor contacts. Ideally, the right foot/floor contact would occur at the midpoint of the cycle.

This gait cycle can also be thought of as a stride. A stride is made up of two steps, one with each foot (see Figure 1). The stride time, or full gait cycle time, is the amount of time for a person to make two sequential floor contacts with the same foot. In a normal gait, the step times for each foot would be the same, and equal half of the stride time. Temporal gait analysis consists of measuring the step times of the patient, and studying the irregularities.

2.3 Pathological Gait due to Amputation

A leg amputee has been deprived of the neuromuscular and skeletal components for proper locomotion, and must learn to walk while wearing a prosthesis or with the aid of crutches.(Cheung et al. 131) Perry states that less energy is required if a well-fitted prosthesis is used instead of

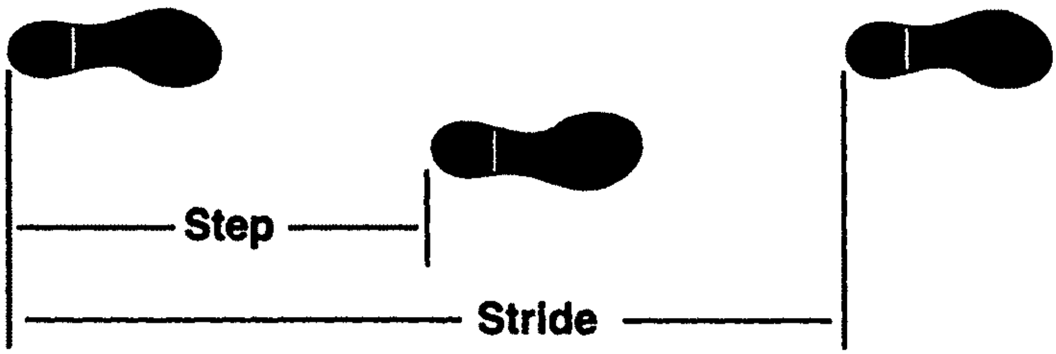

Figure 1 - Stride vs. Step

crutches.(475) But even a well-fitted prosthesis will not allow an amputee to walk in the most efficient manner.(Laughman et al. 129)

A person who has had a leg amputated below the knee joint is referred to as a BK amputee. An AK amputee has had the leg amputated above the knee. The extent of the amputation is an important factor in determining how abnormal a gait will be.(Cheung et al. 132) Either way, wearing a prosthesis allows an amputee to move about while having their hands free. But, their gait cycle is almost always askew, because the prosthesis does not behave as well as a normal leg.

Normal human locomotion is very energy efficient. It allows minimal energy to propel the body forward, because the movements of the legs, joints, and even the upper body flow so naturally.(Knight et al. 306) Due to the amputee's irregular locomotor system, they walk "at a higher metabolic cost than normal."(Cheung et al. 132) If an amputee learns to use the prosthesis as efficiently as possible, they should be able to reduce the amount of energy expended. Seeger et al. state:

> Training to achieve a functionally efficient and cosmetically smooth pattern of gait is a high priority in the habilitation (or rehabilitation) of patients with a physical handicap involving the lower limbs.(364)

Efficiency depends on keeping the rhythm, and not turning the locomotion into a series of disjoint steps.

2.4 Biofeedback Gait Training

Biofeedback training is a method to help patients realize and attempt to overcome their abnormalities. Biofeedback gait training can be used to help patients with a pathological gait pattern. The training can be done over long periods of time so that an accurate record of a patients gait pattern can be recorded throughout the development of disease or the continuance of physical therapy.(Gifford and Hughes 301) This study deals with the temporal patterns of gait, so only the timing of the gait cycle is measured. The researchers attempted to train AK and BK amputees on a treadmill using biofeedback gait training. Two modes of training were used and compared. Patients were trained with an auditory feedback system, and a visual feedback system. The progress of the patients was compared to see which method gave the best results, and therefore used in future training sessions.

CHAPTER 3

HARDWARE DESIGN

3.1 Introduction

This chapter lists the important elements of a advantageous biofeedback gait trainer. It then describes the design and implementation of the gait timer. It explains the purpose and functionality of the components of the timer and proves it's accuracy using test data and a stopwatch. Then the auditory biofeedback module is described, including the hardware and operation. Finally, the visual biofeedback module is described, including the usefulness of the display.

3.2 Design Considerations

Many factors need to be considered when designing a practical Biofeedback Gait Trainer. In order to improve a person's gait, it is necessary to have a measuring device which will not interfere with the natural movements of the body. A design that lets a subject walk at any pace without interruption would allow the subject to walk more normally, and not have to worry about the measurements being taken. The unit must also be useful for many people, ranging in height, weight, age, and mobility. Additionally, it should give the clinician valuable information in a user-friendly environment. Other considerations include limited number of external components that provide accurate results and useful feedback with minimal delay.

In order to give the subject as much freedom as possible, remote transmitters and receivers were incorporated into the design. This allows the subject to walk without being constrained to a certain walkway or have to worry about wires connecting them to a computer. Therefore, the step times measured will be more accurate, because the subject will have enough time and mobility to settle into their regular stride.

3.3 Gait Timer

Foot/floor contact is detected when the subject steps on the switches placed on the feet. The foot switches can be placed inside or attached to the bottom of the shoes. The switches are paper thin and less than an inch square, so they will work with any size shoe. They only need the slightest amount of pressure to trigger, so children can also successfully use the device.

Wires connect to the foot switches and run up the back of the legs to the transmitter unit. The two wires are long enough for even the tallest subjects and can be secured to the legs with velcro straps or rubber bands. The wires connect to the transmitter unit which is strapped to the lower back with

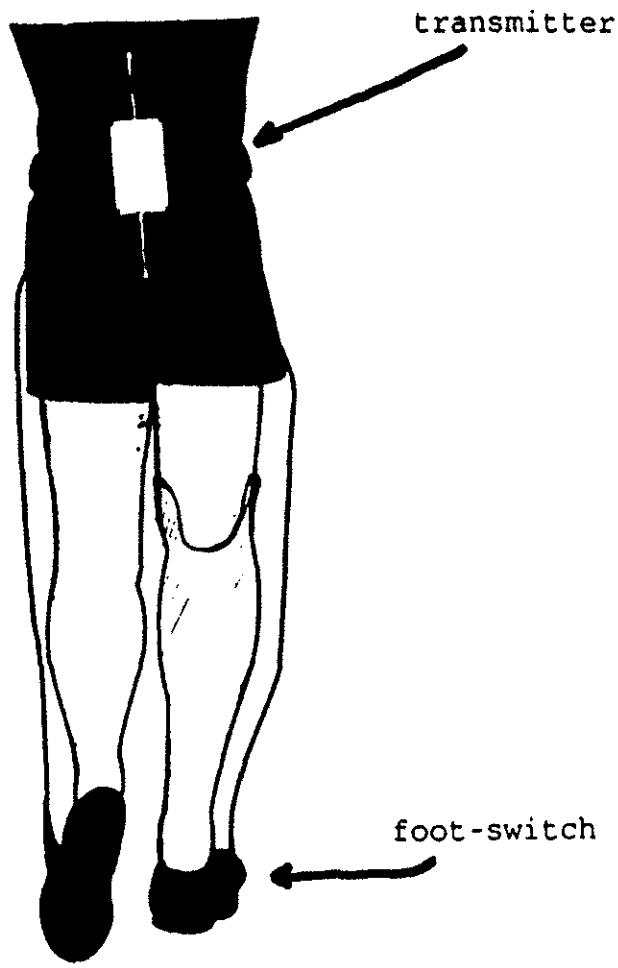

Figure 2 - Remote Transmitter Unit

a belt, as shown in Figure 2. This transmitter sends an on/off signal for each foot switch so the receiver knows when the subject's feet are on the floor. The transmitter can send the two signals to the receiver from over 20 feet away, so step times can be measured in large rooms or hallways.

The transmitter straps onto the back with a large, adjustable belt. It uses two 9 volt batteries for the transmitter circuitry and has a power switch to save battery power. Figure 3 shows a diagram of the transmitter layout. Both transmitter circuits were purchased to work with their respective receiver circuits and to operate at different frequencies, so that they do not interfere with each other. One operates at 27 MHz and the other at 49 MHz, and both can be transmitted and received simultaneously. The receivers output a logical zero (low) until the switches are closed. Then a logical one (high) can be observed which indicates that foot/floor contact has been made.

The two receiver outputs are wired into a Motorola 68HC11 EVBU microcontroller, as shown in Figure 4. The microcontroller is the brain of the gait timer/trainer. Its internal clock, inputs, and outputs are used in the design. The receiver outputs are connected to 2 input capture lines on the microcontroller. When these inputs sense a rising edge, the internal clock time is stored to registers in memory. The clock is pre-scaled so that each count takes 8 μs, and an entire clock cycle takes 524.3 ms, approximately one half of

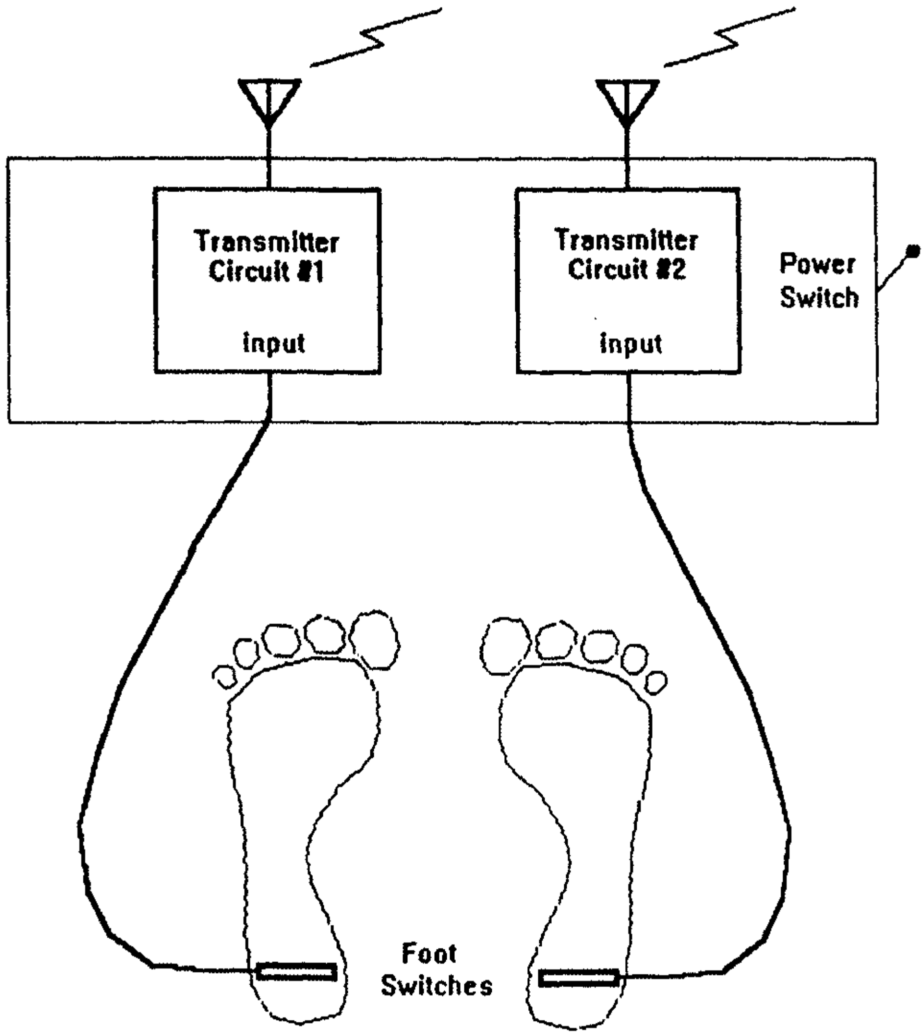

Figure 3 - Diagram of Transmitter Layout

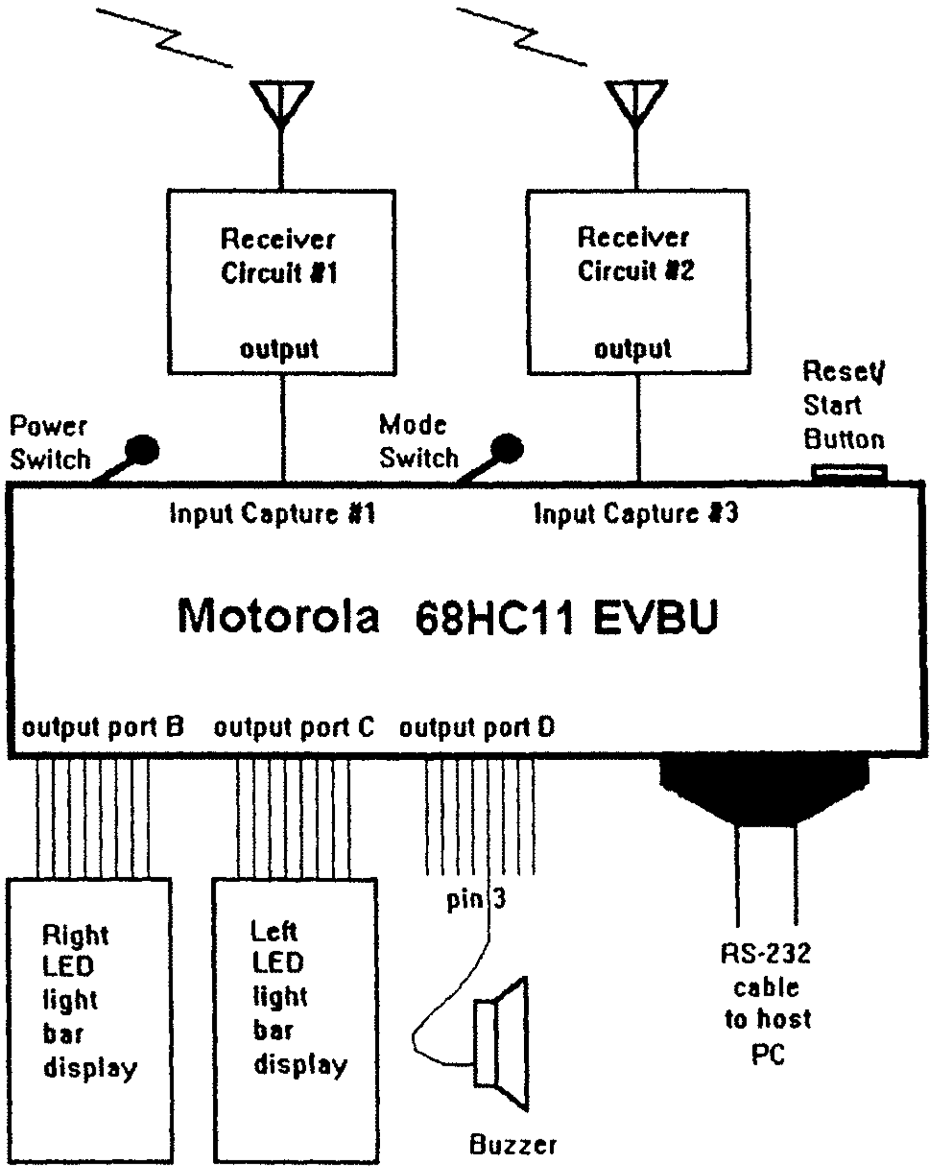

Figure 4 - Diagram of Receiver Layout

a second. Since a step may take more time than that, two extra registers are used to store roll-over counts, to keep track of time when the clock overflows. The times that are stored represent the foot/floor contact times of the subject's trial. When the difference between foot/floor contact times is calculated, step times can be found. The step times are calculated with the code stored in the microcontroller's Electrically Erasable Programmable Read Only Memory (EEPROM). They are determined in less than a millisecond, because the microcontoller's clock rate is so high. The microcontroller is interfaced with a PC and the step times are printed to the computer's monitor after every stride. The times are in hexadecimal and represent the number of clock counts between the respective footsteps.(see Figure 5) In order to calculate the time in seconds, the following equation must be used:

$$t_{(sec)} = count_{(hex)} \rightarrow count_{(dec)} * 8 * 10^{-6} \qquad (3.1)$$

Since this is an awkward operation to do repetitively, a computer program was written to make it easier, which is described in chapter 4. After printing the times on the screen the device is ready to take more measurements without any noticeable delay, so the patient can walk freely without having to wait for the equipment to reset. This remote signalling is instantaneous and grants considerable freedom to the subject. Therefore, the subject can learn to feel comfortable while wearing the device, so they will walk as

```
Hit RETURN to start.

OK!
0001 BB56
is RIGHT time
0001 5A16
is LEFT time
0001 A1B3
is RIGHT time
0001 43AE
is LEFT time
0001 9E81
is RIGHT time
0001 4B01
is LEFT time
0001 FCFC
is RIGHT time
0001 351C
is LEFT time
```

Figure 5 - Sample of Hexadecimal Step Times

they regularly would, giving the clinician the most exact data on their gait patterns.

The device was tested to confirm that it indeed measured times as accurately as believed. The switches were set up on a table top near a stopwatch and the host PC. The timing program was started and the left switch was triggered. After observing a time of 30 seconds on the stopwatch, the right switch was contacted. Then, 30 seconds were allowed to elapse before the left switch was contacted again. This was repeated many times, and the hexadecimal times were stored in a log file. When done, the times in the log file were converted to seconds. The times were all very close to the 30 second test time with an average of 30.142 seconds and a standard deviation of .059. This small error can be attributed to the human factor of pressing the switches at the exact time, and the gait timer can be believed to work as planned.

3.4 Auditory Feedback Module

The auditory feedback design incorporates a buzzer into the design. As shown in Figure 4, the buzzer is connected to output port D, pin 3 on the microcontroller. It will produce a high pitch sound when a high logic level is output to that pin. The auditory feedback module is just an extension of the gait timer described above. The buzzer can be set to make an undesirable noise after a user defined time has elapsed, if a foot/floor contact had not yet been made. The noise will continue until the appropriate foot has touched the floor. This pace can be set by the clinician to a value which will help the patient in their gait training. A different pace can be set for each foot, because a symmetric temporal gait pattern is not expected immediately. If the pace times are set to the hexadecimal number FFFFFF, as they are on power up, no feedback occurs and the program acts just like the gait timer described above, in that it only prints the step times and never gives the patient any auditory feedback.

The pace is controlled by the output compare features on the microcontroller. These execute certain segments of code once a certain clock count has been reached. The code does not execute if the foot/floor contact was made before or at the set pace, and no sound will be heard in that case. The subject will want to step as quickly as possible after the noise starts, because it will turn off the irritating buzzing, which will continue until the proper foot is planted. The timing of the next pace will not start until the current foot/floor contact has been made, so the patient will not feel like they are falling behind. The longer that the device is making noise, the longer it will irritate the patient. Ideally, it will speed up their bad leg, and therefore help to correct their abnormality. With this continuous feedback, the patient will be able to realize the problem with their gait pattern, and physically and mentally strive to correct it.

3.5 Visual Feedback Module

The visual feedback design uses two vertical Light Emitting Diode (LED) light bars to provide the subject with information about their temporal gait pattern. Here the object of the feedback is not to have the patient step at certain times, but to equalize the left and right foot step times, no matter what they are. Once they are spending the same amount of time in each step, their temporal gait pattern is symmetric, and they can go on to learn to walk faster if necessary. Each of the two light bars consist of 8 bright LED's and correspond to the left and right step times.(see Figure 6) The display box consists of simple circuitry to allow the LED's to work from a 9 volt DC power supply. The LED's are wired into two 8 pin output ports on the micro-controller, as shown in Figure 4. Port B is assigned to display the feedback for the right foot, while port C is configured for the left. When the output pins go high, the LED's turn on. The program finds the difference between the left and right step times and scales it. Then it turns on a certain number of LED's on the side of the display that corresponds to the side with the longer step time. The foot with the shorter step time will not have any lights glowing on its display. The LED's remain lit until the next 2 foot/floor contacts are observed. Then a new step difference will be calculated and displayed in the same way. The program does not need to know which step time will take longer, prior to

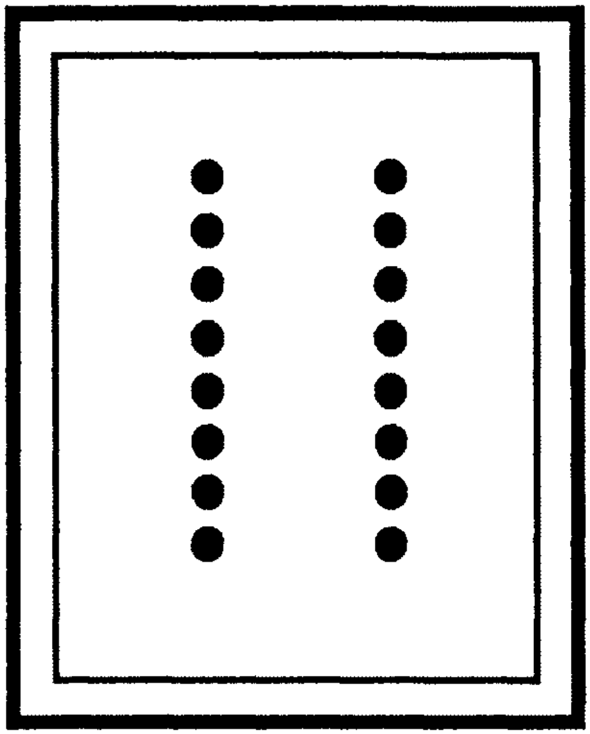

Figure 6 - LED Light Bar Display

its execution. It will find the difference and determine which foot took longer to contact the floor.

These light bar displays provide useful information for the subject. If the subject sees a fully lit display, he will know one of his legs is taking considerably longer to step. As the training continues, the subject can try to lessen the number of LED's that are turned on by stepping faster with the appropriate foot. When only a few LED's at the bottom of the display are lit, then the person is very close to walking symmetrically. This continuous feedback allows the patient to observe his improvements, and make an effort to try harder. The display can also be setup to allow only the clinician to see the step time difference, so they can measure improvements or instruct the subject in a certain manner. Either way, it supplies useful feedback to the patient, so they can improve their gait.

CHAPTER 4

SOFTWARE

4.1 Introduction

The different software used in the gait timer/trainer is described in this chapter, including a discussion of the usefulness of the program used to connect a host PC to the device. Two Pascal executable programs that were created to make the system user-friendly are described next. Finally, the assembly language programs that were written and down-loaded into the microcontroller to make it act as a biofeedback gait timer/trainer are presented.

4.2 Connecting to the host PC

First, the computer must be able to communicate with the microcontroller, so it can down-load assembly language programs and output data. The software chosen for this application was Procomm, which allows a PC to act as a terminal for microcontrollers or remote computers using a modem. Procomm has many advantages associated with it. It is simple to use and takes up less than 2 kilobytes of disk space on the hard drive of the PC. With a few keystrokes, it also allows the user to enter the DOS shell or any text editor.

One useful feature is that it can create a log file that will contain all text that appears on the monitor between the start and the end of the log file creation. This is used when a gait training trial is to be performed on a patient. Before the patient starts to walk, a log file should be opened and named. While the person walks, the step times that appear (in hexadecimal) on the screen will be saved in that file. At the end of the trial, the log file can be closed, and examined with any text editor.

Procomm's main purpose is to down-load the assembly language code to the microcontroller so that it functions as desired. With the proper keystrokes, the gait trainer modules can be down-loaded into the microcontroller's RAM. They can then be moved into the EEPROM, so a push of the reset button on the microcontroller will start the program. Once a module is loaded into the EEPROM, the device can be shut off and it will still be able to execute on the next power up. Procomm is also used to modify the memory locations in the microcontroller's RAM so that the pace times used in the auditory feedback module can be set by the clinician before the training is started.

4.3 Analyzing Step Times

The software for analyzing the data must give accurate results and useful information to the clinician trying to train the patient to walk better. For this reason, the DOS shell in Procomm is used to get to the command line of the computer and execute the applications Pascal programs. One program, called **G.exe**, that was written using the Turbo Pascal compiler, examines a log file and converts all the hexadecimal times into seconds using equation (3.1). This program can be used after a trial was recorded in a log file. When executed, the program asks for information about the patient, including their name, age, and sex. It then asks for the date that they were tested, and the name of the log file that represents their step data.(see Figure 7) It asks for the personal information so that it can print it out as a header to all the statistical data. The program examines the log file and extracts all valid times and determines which step times are for the left foot and which are for the right foot. These times are converted into seconds using equation (3.1). The difference between the step times for each gait cycle is also calculated, as well as the time for the full stride. The average left and right step times are calculated, including the standard deviation of each. The average step time difference and stride times are also calculated. Bajd and Kralj state "For some patients... symmetry quotients certainly represent precious data when the effects of therapy or

```
Please provide some information about the patient.
What is their name? Robert D. Lynn
How old are they? 4_
Are they male or female? male
What is the date they were tested? October 22, 1994

Enter the name of the log file (without .log):
testamp
```

Figure 7 - G.exe program input screen

assessment of a prosthesis are to be determined."(132) Therefore, the percentage of time spent on the right foot and the left foot are also found, because this is a good indication of the symmetry of the gait pattern. This is calculated by dividing the average step times for each foot by the average stride time and multiplying by 100 percent. The result is the percentage of time spent on the appropriate foot. After printing the personal information, the program lists all the statistical information described above.(see Figure 8) It then waits for a key-press or a print-screen command, and continues by listing all the step time data for each gait cycle. Using the information provided, a clinician can determine the extent of the patients progress and/or determine a suitable value at which to set the pace times, if using the auditory feedback module.

4.4 Setting Pace Times

In order to set the pace times in the auditory feedback module, the clinician must know the hexadecimal count that will represent the desired time in seconds. The count must be entered into specific memory locations in the microcontrollers RAM. To facilitate this process, a pascal program **C.exe** was written. This program can be executed out of the DOS shell in Procomm. When started, it prompts the user for a time in seconds that will be used as the pace time. When a time is entered, the program instantly returns an equivalent

```
Step Timing Analysis
--------------------
Subject: Robert D. Lynn
Age: 42  Sex: male
Tested on: October 22, 1994
----------------------------------------------------------------
MEAN LEFT STEP TIME (s): 0.671
LEFT STANDARD DEVIATION: 0.027

MEAN RIGHT STEP TIME (s): 0.914
RIGHT STANDARD DEVIATION: 0.078


MEAN FULL GAIT CYCLE (s): 1.584

MEAN STEP DIFFERENCE (s): 0.243


AMOUNT OF GAIT CYCLE ON LEFT: 57.67%
AMOUNT OF GAIT CYCLE ON RIGHT: 42.33%



Press ENTER to continue or PRINTSCREEN for a hardcopy...
```

Figure 8 - G.exe program output screen

```
Enter the pace (in seconds) to be converted.
.67
Use 014726 as the pace.




Press ENTER to continue or PRINTSCREEN for a hardcopy...
```

Figure 9 - C.exe program execution

hexadecimal number that the microcontroller can use to set the pace properly.(see Figure 9) A memory modify command of memory locations 0024-0026 and 0027-0029 allows the left and right pace times to be set. Both of the programs described above were written with the understanding that the user will not always be the most computer literate person, so they were made as easy to use as possible.

4.5 Feedback Modules

The auditory feedback program and the visual feedback program work in much the same way, but still have some noteworthy differences. A flowchart of the gait timer with its modules is shown in Figure 10. The programs start by initializing all aspects of the device. The clock is pre-scaled by a factor of 16 and then the memory locations of the interrupt routines are stored to the vector jump tables. Next, the input and output lines are configured and reset. Then, the program asks for the enter key to be pressed so that the training may start. Once that happens the device waits for a right foot/floor contact before proceeding. This is just a starting point for the program so it knows to look for a left foot/floor contact next. When one occurs, the time is stored to RAM. Then it looks for a right foot/floor contact, and stores that time to RAM. Once another left contact occurs, the time is stored, and the step times calculated. The time for the right step is found by subtracting the first

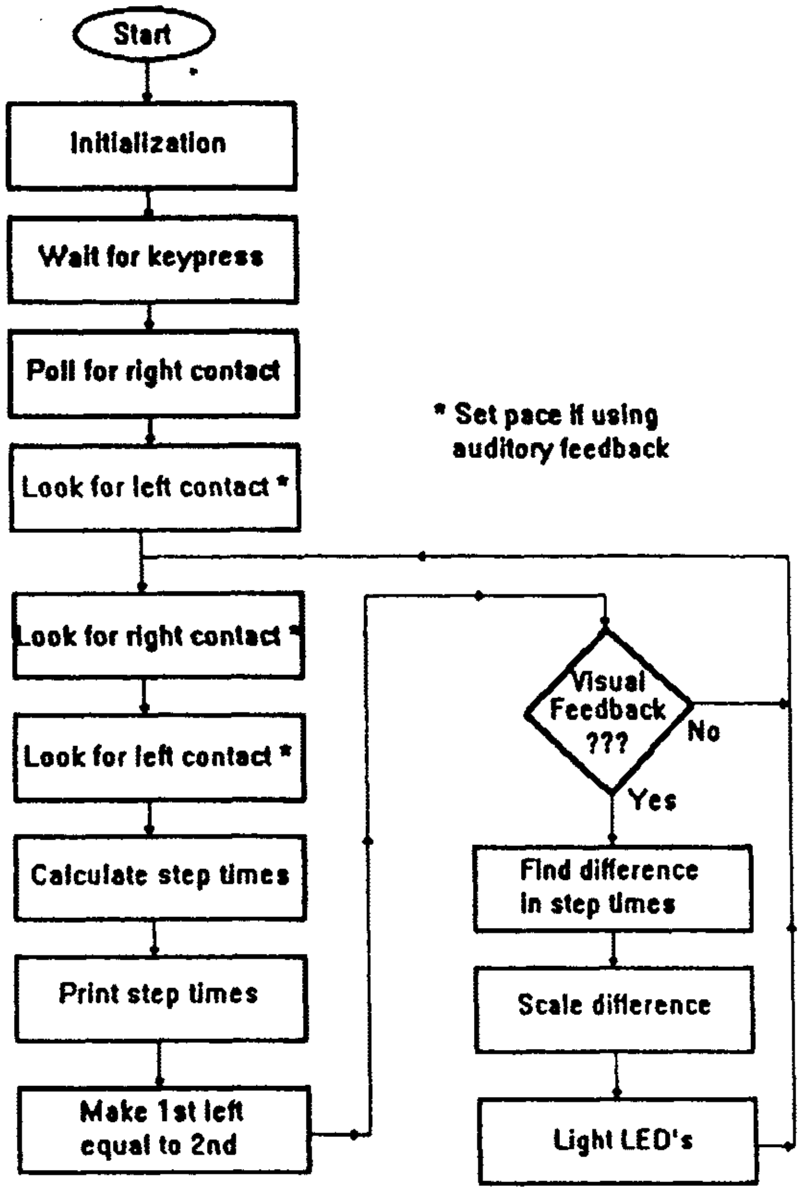

Figure 10 - Gait Timer Flowchart

left contact time from the right contact time. Similarly, the left step time is computed by subtracting the right contact time from the second left contact time. The program then prints these step times to the screen. It moves the second left contact time to the location in memory where the first left contact time was stored, and sets itself to restart at the point in the program where it looks for a right contact. This is all done in about a millisecond, which is fast compared to normal walking times, and allows the person to continue with their natural stride. The process repeats itself until the microcontroller is reset.

The internal clock on the microcontroller will overflow every 524.3 ms. When this happens, a subroutine is set up to keep track of the number of overflows, so the program is not hindered. When using pace times as in the auditory feedback module, this subroutine also checks the pace and determines when the appropriate amount of time has elapsed.

The auditory feedback program uses this pace time that is stored in RAM to determine when to buzz. When a foot/floor contact occurs, the contact time is stored, and then the device waits the appropriate pace time before it turns on the buzzer. Once the buzzer is activated, it takes the other foot to contact the floor to turn it off. If that foot makes contact with the floor before the buzzer is activated, it will not turn on. Instead, it will be set to go off at a new time for the other foot. This process will continue as long as the patient wants to be trained, and will only stop by resetting the microcontroller.

The visual feedback program operates without using pace times. It stores the left and right contact times and prints out the respective step times as before, but it also calculates the difference between the two step times. Once this difference is found, it is compared to many scale factors. These scale factors represent certain amounts of time divided up non-linearly. Depending on the comparison, the subroutine selects a number of LED's that need to be lit up on the light bar to represent the difference in the step times. For each light bar, the bottom six LED's represent a tenth of a second, so if five were on, the step time difference would be more than .5 seconds. The seventh LED signifies .75 seconds, and if all eight LED's turn on, the step time difference is greater or equal to one second. The subroutine also determines which step time was longer, and turns on the side of the light bar that has the longer step time. The light bar stays on until the next stride when a new difference is found. This process also repeats until the microcontroller is reset.

For both of the feedback modules, Procomm can be set up to be recording the printed step times to a specific log file. The **G.exe** program can be run on a log file created by any of the gait training modules. This will allow the clinician to see the progress of the patient and the effect of the biofeedback with little additional effort. All programs described and/or written for this project can be found in the Appendix. **G.pas** and **C.pas** are the hexadecimal conversion programs. The code for the auditory feedback module is stored in files **buzzd1** and **buzzd2**. The code for the visual feedback module is stored in programs **flash1** and **flash2**. These programs are stored in two parts due to the amount of RAM on the microcontroller. The EEPROM can hold 512 bytes, but the largest available block of RAM on this model of the microcontroller is only 256 bytes. In order to load the EEPROM with a *move* command, the code must be loaded into RAM and then moved into it. Therefore, the program had to be split into two separate parts and loaded one at a time into the EEPROM.

CHAPTER 5

EXPERIMENTAL RESULTS

5.1 Introduction

This chapter describes the testing and training of prosthetic wearing amputees. The data generated from the testing was used to determine which type of biofeedback is more advantageous. First, the procedure for timing and testing the subjects is discussed. Then, the usefulness of training on a treadmill is explored. Next, the results are examined for each of the subjects that were trained. Finally, a few observations about the results are made.

5.2 Procedure

Two subjects participated in the testing of the biofeedback gait trainer. Each was an amputee that had been wearing a prosthesis since surgery. After signing consent forms to participate in the experimentation, they were examined by a Physical Therapist. Subject A is a teenage female left leg AK amputee. Subject B is an adult male right leg BK amputee. Both subjects had been walking with prosthetics since amputation and had learned to maneuver with them without too much trouble. Therefore their temporal gait patterns were not as distorted as those patients who have recently had an amputation and are still adjusting to their prosthesis. Nevertheless, an untrained eye could easily observe temporal gait asymmetry in both of the subjects. For this reason, they were found to be acceptable subjects for testing the biofeedback gait trainer.

The subjects were asked to walk on a treadmill for the timing and training. Initially, the treadmill was set at the slowest possible speed, then the speed was increased until the subject said they were walking at their normal pace. After stepping off the treadmill, the transmitter was strapped on and the foot switches attached to the underside of their shoes. First, the device was used to get an accurate measure of their temporal gait pattern. The treadmill was started, and the device measured the step times as the subject walked. No feedback was given at this time because this was only meant to measure the asymmetry of the subject's gait. Finally, the subjects were tested with one of the feedback modules, and then the other, while the treadmill moved at the same speed. Subject A was trained with visual signals first, and Subject B was trained with auditory signals first.

For the auditory feedback, the pace was determined from the data recorded in the timing session. It was chosen to be in the range of values less than the pathological leg average step time and greater than the normal leg average step time.

Immediate symmetry was not to be expected, so the pace setting gave the subjects some leeway. The subjects were given directions on how to understand and react to the feedback signals, and what it meant if they heard the buzzing. Then, they were trained with the auditory feedback and the step times were recorded as they went. The subjects were only trained for approximately 2 minutes, because it would be undesirable in this experiment to have training carry over to the next module. The main purpose of the testing was to determine how well the subjects understood the feedback, not to determine how long a person must train to walk correctly.

For the visual biofeedback training module, the treadmill moved at the same speed as the subjects watched the LED light bar display set up in front of them. They were given instructions on how to interpret the visual feedback signals, and they started the training. Again, their step times were recorded as the training proceeded. They were only given feedback for approximately 2 minutes, in case they had not yet trained with the auditory feedback.

5.3 Usefulness of Treadmill Training

The training was done on a treadmill for many reasons. First, it would allow the subject to walk for a long period of time without having to stop. If walking down a hall or in a large room the subject would have to stop and turn around sometimes. Using the treadmill allows the training to continue without interruption. Also, the treadmill assured a constant pace at which the subject had to walk. This insured the stride times would be relatively constant, while changing the step times with the biofeedback signals. Since the clinician can regulate the speed of the treadmill, future training sessions could include variations in the speed to further train the subject. Also, the subject can feel safe while walking on the treadmill because it is equipped with a rail that they can hold on to. Without the use of a treadmill, the subject would have to be guarded for stability by a clinician. Finally, using the treadmill also guarantees the feedback signals will reach the subject. It would be hard to walk far away and still be able to see or hear the feedback signals. The advantages of using a treadmill for the gait training are quite evident, and it seemed to be a good decision.

5.4 Subject A

Subject A was trained following the procedure previously described, using the visual feedback first. The step times were stored in a log file for further analysis by the gait program, **G.exe**. Subject A was trained using the visual feedback module. The LED light bar display was positioned in front of the treadmill so the subject could easily see the lights. The step times for the visual feedback training were stored in a log file. From the initial timing of subject A, a pace was chosen for use in the auditory biofeedback module. The pace times were set at .70 seconds for both legs. This was based on the average left and right step times, which were .770 and .539 seconds, respectively. This pace was chosen because the pathological limb was on the left side and that side is not expected to move symmetrically immediately. The hexadecimal representation of the pace time was calculated using the program **C.exe**, and the value was loaded into RAM on the microcontroller. Finally, the subject was ready for training with the auditory biofeedback module. The step times of this training were also recorded into a file. All of the average timing and training data found on subject A's temporal gait pattern from executing the gait program is listed in Table 1.

5.5 Subject B

Subject B was also trained, but used the auditory feedback module first. After timing subject B, a pace of .67 seconds was chosen for both legs. This was also based on the average left and right step times, which were .668 and .730 seconds, respectively. The hexadecimal count representing the pace time was loaded into RAM on the microcontroller. Then, the subject was trained using the auditory biofeedback module, and the step times were recorded. Subject B then used the visual feedback module for training and these step times were

Table 1 - Subject A's Gait Data

Trial	Left Step	Right Step	Step Diff.	% on Left	%on Right
Timing	.770s	.539s	.231s	41.19	58.81
Auditory	.729s	.579s	.150s	44.25	55.75
Visual	.696s	.612s	.084s	46.79	53.21

Improvement due to auditory feedback - 35.06%

Improvement due to visual feedback - 63.64%

Table 2 - Subject B's Gait Data

Trial	Left Step	Right Step	Step Diff.	% on Left	%on Right
Timing	.668s	.730s	.062s	52.21	47.79
Auditory	.674s	.709s	.035s	51.27	48.73
Visual	.669s	.689s	.020s	50.74	49.26

Improvement due to auditory feedback - 43.55%

Improvement due to visual feedback - 67.74%

also stored. Table 2 shows the temporal gait data accumulated from subject B's timing and training using the gait trainer.

5.6 Observations

Studying the data on subject A gives an indication that she initially had a slightly asymmetrical gait pattern. Although not terribly abnormal, it could be noticed by any observer, and was easily detected by the timing device. First, the visual feedback was used, and a great amount of improvement was observed. Training for 2 minutes with visual feedback decreased the average difference in step times from .231 to .084 seconds. This represents an improvement of 63.64%, based on the equation:

$$Improvement = \frac{T_{du} - T_{df}}{T_{du}} * 100\% \tag{5.1}$$

where

T_{du} = usual difference in step times, and

T_{df} = difference in step times using feedback.

When the auditory feedback was used, less improvement was observed. After setting the pace times and training for about 2 minutes, the average step time difference decreased to .150 seconds. Using equation (5.1), this represents an improvement of 35.06%.

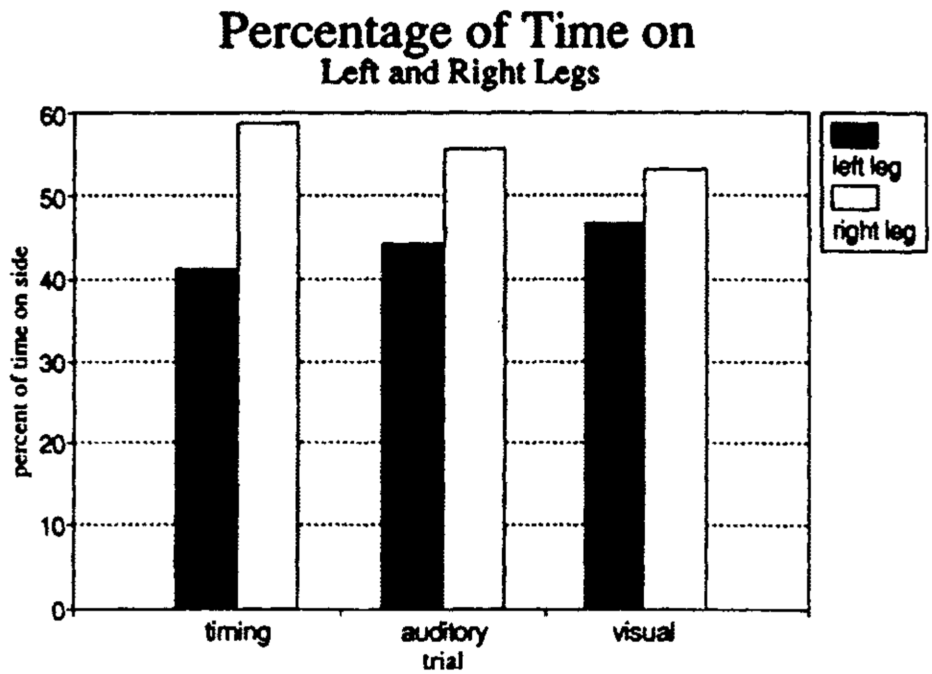

Figure 11 - Subject A's Symmetry Quotients

Figure 11 shows each of the trials of testing and training subject A. It presents the symmetry quotient for each trial. Normal temporal gait patterns would have symmetry quotients of 50% for each leg. The figure shows the symmetry quotients got close to normal when using auditory feedback, and even closer when using visual feedback. By examining subject B's gait data, it is easily seen that the pathological limb is on the right side. The temporal asymmetry was barely noticeable to observers, but the device found the deviations. As with subject A, the auditory feedback module improved subject B's gait in the 2 minute trial period. The average step time difference decreased from .062 seconds to .035 seconds, an improvement of 43.55%, was found using equation (5.1).

Using visual feedback for a 2 minute trial period increased the improvement again. The step time difference decreased to .020 seconds. Calculating the improvement using equation (5.1) gives an improvement of 67.74%.

Figure 12 shows the symmetry quotients for each of subject B's trials. The figure shows how his symmetry quotient was already close to normal even before feedback training. The figure also shows the improvement in the subject's temporal pattern with each of the feedback trials.

For both subjects, the gait timer system measured and recorded their temporal gait patterns whether using biofeedback or not. Using the gait programs to study the trials gives an indication of their improvement while using feedback. Both subjects improved with the auditory feedback, but an increased improvement was observed when using the visual feedback. This can easily be deduced from Tables 1 and 2 and from Figures 6 and 7. It is believed that either of the modules could be used to train the subjects, but the visual feedback technique provides the most understandable signalling.

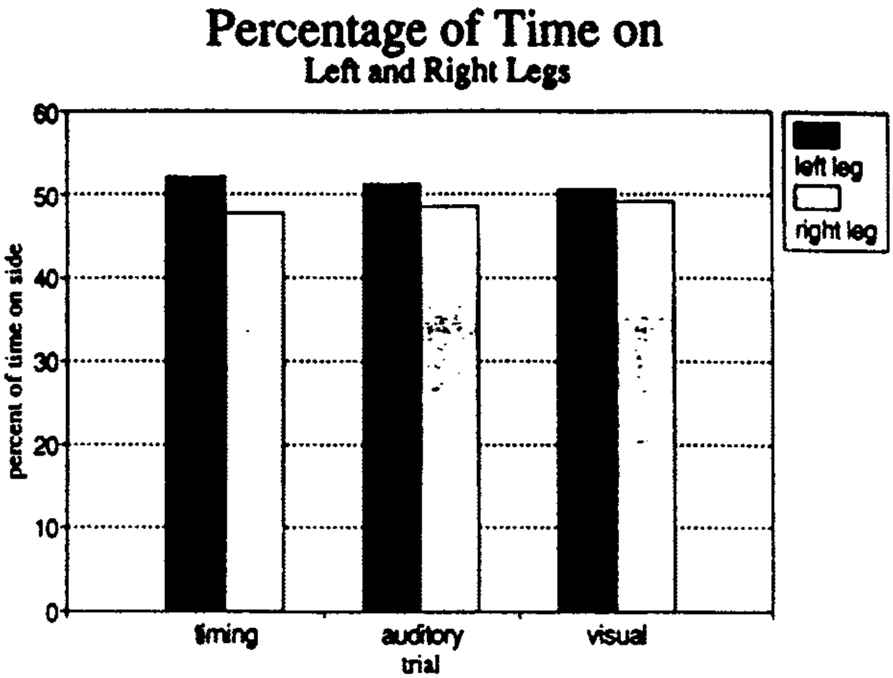

Figure 12 - Subject B's Symmetry Quotients

CHAPTER 6

CONCLUSIONS

6.1 Discussion

The purpose of this research was to determine whether auditory or visual signals provide more understandable biofeedback. The determining factor in the decision was the comparison of the overall response of subjects to the feedback signals, over a short period of time. In short, which method relayed the most useful and understandable information to the subject. Both biofeedback methods were developed to train a person to walk with more evenly timed footsteps. Amputees wearing prosthetics on one leg were chosen as the test group because they often have asymmetry in the timing patterns of their gait. By measuring their temporal gait pattern before and while being trained, we could determine the response to the feedback signals, validating their effectiveness.

When they were trained with each of the biofeedback methods, timing measurements were recorded. This allowed a comparison of the training period data to the original gait pattern data. An assessment of the improvement was made from the decrease in the average step time difference in the temporal patterns of the subject's gait. For each of the training modules, great improvement was found, because the average step time difference reduced dramatically. But, by studying the data in chapter 5, it is evident that the subjects made better improvement while training with the visual feedback signals than with the auditory signals.

This could be verified by observing the training sessions. When using the auditory biofeedback module, the subjects' reactions to the buzzer were unpredictable. Throughout the training session, the buzzer could be heard on random steps, signifying the information was not completely training the subjects properly. Contrary to this, the visual training allowed the subject's to correct themselves within a few strides, and continue to walk better for the rest of the trial.

The symmetry quotients of the subjects' gait were also determined for the initial timing and each of the biofeedback techniques. This data highlights the amount of temporal asymmetry through initial timing and feedback training. A normal temporal gait pattern should have symmetry quotients of 50% for each leg. From graphs of the symmetry quotients in chapter 5, it can be seen that these subjects do not have normal temporal gait patterns. It is also evident that symmetry increased with each of the biofeedback techniques. But, their symmetry quotients improve the most when using the visual biofeedback module. Therefore, it is believed that the visual technique was the most understandable to the subjects, and allows them to react to their temporal discrepancies better.

Clearly, the visual feedback technique provided the subjects with better biofeedback signals. The response to the LED light bar display was much better that the response to the buzzer. This can be seen from the trial data, the improvement factors, and the symmetry quotients.

The subjects were asked to assess both of the biofeedback methods. Each subject stated that the visual feedback gave them a better understanding of their abnormality. They said that the auditory signals were a little more confusing, and required more concentration. They also said the LED light bar display was better, because the LED's stayed on until the next stride was made, therefore giving them better information. These statements concur with the recorded data discussed earlier.

With subsequent training sessions, it is believed that a normal temporal gait pattern can be achieved using either of the biofeedback modules. But, the visual technique would be used, because the response of the patients to the LED light bar display was better. Using the auditory feedback module would prove to be a waste of time, because the improvements are made at a slower rate.

Since the visual feedback signals provided better biofeedback to the subjects, all biofeedback devices to be designed should use this type of feedback. But, for patients with poor vision, auditory signals could be implemented into the design. This would allow the device to be used by more people, and this is a major concern when designing training aids to be used in a clinical environment.

6.2 Conclusion

This thesis contains the development of a biofeedback gait trainer with two types of biofeedback. The timing and signalling has been proven to operate as desired. The system is capable of timing and training subjects having a pathological gait pattern. The temporal gait patterns of lower extremity amputees wearing prosthetics were tested. Training the subjects using each type of feedback attempted to make their step times more symmetrical, and therefore their locomotion more energy efficient. Both types of biofeedback provided the subjects with signals that allowed them to improve their gait pattern, but the visual feedback provided them with the most understandable feedback. This is due to the fact that the feedback signals were easily interpreted. This allowed the subjects to realize their deviations and try to correct them. Therefore, the visual biofeedback module should be used in all of the subsequent training sessions.

Overall, the study proved to be interesting. The reactions of humans to visual signals is fascinating, and in this gait trainer system, they are put to good use. The device worked well, and can be used in the future to time and train patients with any kind of pathological gait pattern.

6.3 Future Considerations

This study was only intended to determine which method of biofeedback was best. Out of all the human senses, vision and hearing are the easiest to signal. Therefore, only visual and auditory signals were incorporated into the design. To test the methods, the temporal gait factor was measured and trained. Out of all the parameters of human gait, this is a relatively easy factor to measure, but it is also a very important one. It may be interesting to study a patient fully trained using the visual feedback method, and seeing how they respond to the auditory signals. If, after a certain amount of training, they stop improving, the auditory feedback could be used to see if any further improvements can be made. It may be possible, because the auditory feedback code depends on exact pace timing, and not just temporal symmetry.

Another important factor is step distance, which this device did not measure. A future goal to increase the usefulness of this biofeedback gait trainer would be to measure the spatial factors. One method may use the speed of the treadmill and the subject's total stride time to determine the stride distance. Nevertheless, this is something to be considered, and was not attempted at the time of this paper.

Another consideration is the transmission of the feedback signals to the subject. If the subject wore a helmet with visual feedback signals set in front of his eyes, he would be able to move off the treadmill and walk around normally. This requires the microcontroller to remotely transmit the feedback signals back to the subject. For obvious cost and development reasons, this was also not attempted in this study.

Finally, a reasonable future endeavor. It would be nice to have the step times print on the screen in seconds, instead of the hexadecimal counts. This requires that the hexadecimal time be converted to seconds before printing. This was not attempted, because there was not enough memory on the microcontroller to do such an operation. If the microcontroller included more memory in the EEPROM or RAM, this modification could be incorporated. That would allow the clinician to view the step times as they occur, and the **C.exe** program would no longer be needed.

BIBLIOGRAPHY

Bajd, T., A. Kralj.(1980). "Simple Kinematic Gait Measurements." Journal of Biomedical Engineering 2(2): 129-132.

Cheung, C., J.C. Wall, S. Zelin.(1983). "A Micro-computer based system for measuring Temporal Asymmetry in Amputee Gait." Prosthetics and Orthotics International 7(3):131-140.

Crouse, J., J.C. Wall, A.E. Marble.(1987). "Measurement of the Temporal and Spatial Parameters of Gait using a Microcomputer based system." Journal of Biomedical Engineering 9(1):64-68.

Fernie, G., J. Holden, M. Soto.(1978). "Biofeedback Training of Knee Control in the Above-Knee Amputee." American Journal of Physical Medicine 57(4):161-166.

Gabel, R.H., R.C. Johnston, R.D. Crowinshield.(1979). "A Gait Analyzer/Trainer Instrumentation System." Journal of Biomechanics 12(7):543-549.

Gifford, G., J. Hughes.(1983). "A Gait Analysis System in Clinical Practice." Journal of Biomedical Engineering 5(4):297-301.

Gifford, G.E., W.C. Hutton.(1980). "Microprocessor Controlled System for Evaluating Treatments for Disabilities Affecting the Lower Limbs." Journal of Biomedical Engineering 2(1):45-48.

Hirokawa, S., K. Matsumura.(1989). "Biofeedback Gait Training System for Temporal and Distance Factors." Medical and Biological Engineering and Computing 27(1):8-13.

Knight, M., S. Nade, W. Ongley.(1983). "The 'Hollywood Gaitrack': a Method for Measuring Temporal and Distance Factors of Gait." Medical and Biological Engineering and Computing 21(3):306-310.

Laughman, R.K., et al. "Prosthetic Gait Training using the concept of Biofeedback and Instrumented Gait Analysis." Control Aspects of Prosthetics and Orthotics. Proc. of IFAC Symposium. 7-9 May 1982. Ohio, USA: 129-132.

Law, H.T.(1987). "Microcomputer-based Low-cost Method for Measurement of Spatial and Temporal Patterns of Gait." Journal of Biomedical Engineering 9(2):115-120.

Perry, J. Gait Analysis: Normal and Pathological Function. New York, NY: McGraw-Hill Medical Publishing Group, 1992.

Roth, E.J., et al.(1990). "The Timer-Logger-Communicator Gait Monitor: Recording Temporal Gait Parameters using a Portable Computerized Device." International Disability Studies 12(1):10-16.

Seeger, B.R., D.J. Caudrey, J.R. Scholes.(1981). "Biofeedback Therapy to Achieve Symmetrical Gait in Hemiplegic Cerebral Palsied Children." Archives of Physical Medicine and Rehabilitation 62(8):364-368.

Wall, J.C., J. Charteris, J.W. Hoare.(1978). "An Automated On-line System for Measuring the Temporal Patterns of Foot/Floor Contact." Journal of Medical Engineering Technology 2(4):187-190.

APPENDIX

```
Program G(input,output);

{ Timothy Hiemenz, Pascal program completed 9-7-94. This
program uses a log file created by Procomm of a trial  using
the biofeedback gait trainer on a subject. The  training aid
prints hex times to the screen as the subject walks. If the
trial is stored in a log file (with a .log extension) then the
file can be scanned with this program to read all times,
determine which foot they are for, convert them to seconds,
and print them to the screen. This program will also give
average step time for each foot, the step difference between
feet, and other useful statistics. Initially the program asks
for information about the patient, which is printed with the
statistics to give it a clinical look. }


uses Crt;


type hex=array[1..8] of char;{array for storing hex numbers}
     hexlist=array[1..50] of hex;{ can store upto 50 steps }
     declist=array[1..50] of real; { 50 steps in seconds }


var  nl,nr:integer;              { # of left and right steps }
     name,age,sex,date:string;{ data about subject }
     lh,rh:hexlist;              {array of 50 lefts and rights}
     ld,rd,                      { in hex and decimal }
     dd:declist;            { 50 step differences in decimal }
     good:boolean;          { var to determine if good file }


Procedure Getinfo(var name,age,sex,date:string);

{ This procedure asks some information about the subject so it
can be printed later with the results. It asks for their name,
age, sex, and the date on which they were tested. }

begin
     clrscr;                        { clear the screen }
     write('Please provide some information ');
     writeln('about the patient.');
     writeln;
     write('What is their name? ');
     readln(name);                  { get their name }
     writeln;
     write('How old are they? ');
     readln(age);                   { get their age }
     writeln;
     write('Are they male or female? ');
     readln(sex);                   { get their sex }
     writeln;
```

```
     write('What is the date they were tested? ');
     readln(date);                      { get the test date }
     writeln;
     writeln;
end;
{ end of Procedure Getinfo }


Procedure Getfile(var nl,nr:integer;var lh,rh:hexlist;
                  var good:boolean);

{ This procedure asks the user for the name of the .log file
which contains the test data on the subject the are studying.
It also checks the file for good data, and extracts the times
into hex arrays for conversion later. }

var  filename,                 { name of .log file }
     line:string;              { line of text being checked }
     test:text;                { text variable for reading file }
     i,j,k:integer;            { dummy variables }
     right:boolean;            { var for determining which foot }

begin
     right:=true;                      { initialize vars }
     nr:=0;
     nl:=0;
     write('Enter the name of the log ');
     writeln('file (without .log):');
     writeln;
     readln(filename);                 { get filename }
     if filename<>'' then
     begin
          filename:=filename+'.log';
          assign(test,filename);
          reset(test);                 { setup to read file }
          repeat
                readln(test,line);
                if line<>'' then
                begin  { check lines in file for times }
                     for k:=1 to 3 do
                     if (line[1]='>') or (line[1]='R') or
                        (line[1]='L') then
                        for i:=1 to 9 do
                        line[i]:=line[i+1];
                     if ((line[1]<':') and (line[1]>'/'))
                     then
                     begin
                          if right then nr:=nr+1
                          else nl:=nl+1;
                          {see if right or left}
                          good:=true;
                          for i:=1 to 4 do
```

```
                              { store hex times }
                              if right then
                                 rh[nr,i]:=line[i]
                                 { store rights }
                              else
                                 lh[nl,i]:=line[i];
                                 { store lefts }
                              for i:=6 to 9 do
                              if right then
                                 rh[nr,i-1]:=line[i]
                              else
                                 lh[nl,i-1]:=line[i];
                              right:=not right;
                              { set for next foot }
                         end;
                    end;
          until eof(test); {check lines until end of file }
          close(test);     { close file }
     end;
end;
{ end of Procedure Getfile }


Function Getvalue(tempch:char):integer;

{ This function takes a character in hexadecimal and returns
the appropriate integer decimal value assosiated with it }

var  tempva:integer;        { temporary value }

begin
     case tempch of         {case statement to find decimal #}
          '0':tempva:=0;
          '1':tempva:=1;
          '2':tempva:=2;
          '3':tempva:=3;
          '4':tempva:=4;
          '5':tempva:=5;
          '6':tempva:=6;
          '7':tempva:=7;
          '8':tempva:=8;
          '9':tempva:=9;
          'A':tempva:=10;
          'B':tempva:=11;
          'C':tempva:=12;
          'D':tempva:=13;
          'E':tempva:=14;
          'F':tempva:=15;
     end;
     getvalue:=tempva;          { return value }
end;
{ end of Function Getvalue }
```

```
Procedure Convert(nl,nr:integer;lh,rh:hexlist;
                  var ld,rd,dd:declist);

{ This procedure uses function Getvalue and together they
convert the hexadecimal prescaled microcontroller clock count
to a decimal time in seconds. }

type val=array[1..8] of integer; {array of integer digits}

var  value:val;        { value of hex digit }
     i,j:integer;      { looping variables }

begin
     for j:=1 to nl do   { repeat for all left steps }
     begin
          for i:=1 to 8 do    { examine all hex digits }
          value[i]:=Getvalue(lh[j,i]);
          ld[j]:=0;           { convert to seconds }
          ld[j]:=ld[j]+value[1]*268.435456;
          ld[j]:=ld[j]+value[2]*16.777216;
          ld[j]:=ld[j]+value[3]*1.048576;
          ld[j]:=ld[j]+value[4]*0.065536;
          ld[j]:=ld[j]+value[5]*0.004096;
          ld[j]:=ld[j]+value[6]*0.000256;
          ld[j]:=ld[j]+value[7]*0.000016;
          ld[j]:=ld[j]+value[8]*0.000001;
          ld[j]:=ld[j]*8;
     end;
     for j:=1 to nr do   { repeat for all right steps }
     begin
          for i:=1 to 8 do    { examine all hex digits }
          value[i]:=Getvalue(rh[j,i]);
          rd[j]:=0;           { convert to seconds }
          rd[j]:=rd[j]+value[1]*268.435456;
          rd[j]:=rd[j]+value[2]*16.777216;
          rd[j]:=rd[j]+value[3]*1.048576;
          rd[j]:=rd[j]+value[4]*0.065536;
          rd[j]:=rd[j]+value[5]*0.004096;
          rd[j]:=rd[j]+value[6]*0.000256;
          rd[j]:=rd[j]+value[7]*0.000016;
          rd[j]:=rd[j]+value[8]*0.000001;
          rd[j]:=rd[j]*8;
     end;
     for j:=1 to nr do        { loop for number of steps }
     begin
          dd[j]:=abs(ld[j]-rd[j]);
                { find difference in steps }
     end;
end;
{ end of Procedure Convert }
```

```
Procedure Print(nl,nr:integer;ld,rd,dd:declist;
                 name,age,sex,date:string);

{ This procedure prints the step times, the differences, and
the full gait cycles for the trials in the log file. It also
averages them and finds the standard deviation, as well as
find the percent of time spent in each step. It prints the
subjects information out first, so when a hardcopy is made,
the information is there. }

var  i,j,num:integer; { looping vars and number of steps }
     lave,rave,dave, { left, right, and difference averages}
     ldev,rdev:real; { left & right standard deviation }

begin
     num:=nl;
     if num>nr then num:=nr;  { determine number of steps }
     dave:=0;
     lave:=0;
     rave:=0;                 { initialize variables }
     ldev:=0;
     rdev:=0;
     clrscr;                            { clear the screen }
     writeln('STEP TIMING ANALYSIS');{ print subject info }
     writeln('--------------------');
     writeln('Subject: ',name);
     writeln('Age: ',age,'  Sex: ',sex);
      writeln('Tested on:  ',date);
     write('-----------------------------');
     writeln('-----------------------------');
     for i:=1 to nr do
     rave:=rave+rd[i];
     rave:=rave/nr;     { calculate average right step }
     for i:=1 to nl do
     lave:=lave+ld[i];
     lave:=lave/nl;     { calculate average left step }
     dave:=abs(lave-rave); { calculate average difference }
     for i:=1 to nr do
     rdev:=rdev+sqr(rd[i]-rave);
     rdev:=rdev/nr;
     rdev:=sqrt(rdev);{ calculate right standard deviation }
     for i:=1 to nl do
     ldev:=ldev+sqr(ld[i]-lave);
     ldev:=ldev/nl;
     ldev:=sqrt(ldev);{ calculate left standard deviation }
     { print results }
     writeln('MEAN LEFT STEP TIME (s): ',lave:6:3);
     writeln('LEFT STANDARD DEVIATION: ',ldev:6:3);
     writeln;
     writeln('MEAN RIGHT STEP TIME (s): ',rave:6:3);
     writeln('RIGHT STANDARD DEVIATION: ',rdev:6:3);
     writeln;
```

```
    writeln;
    writeln('MEAN FULL GAIT CYCLE (s): ',lave+rave:6:3);
    writeln;
    writeln('MEAN STEP DIFFERENCE (s): ',dave:6:3);
    writeln;
    writeln;
    write('AMOUNT OF GAIT CYCLE ON LEFT: ');
    writeln(100*rave/(rave+lave):5:2,'%');
    write('AMOUNT OF GAIT CYCLE ON RIGHT: ');
    writeln(100*lave/(rave+lave):5:2,'%');
    writeln;
    writeln;
    writeln;
    write('Press ENTER to continue or ');
    writeln('PRINTSCREEN for a hardcopy...');
    readln;                    { wait for enter keypress }
    clrscr;                    { clear screen }
    write('  LEFT STEP TIMES');
    write('   ');
    write('RIGHT STEP TIMES');
    write('   ');
    write('FULL GAIT CYCLE');
    write('   ');
    writeln('STEP DIFFERENCE');
    write('  ---------------');
    write('   ');
    write('----------------');
    write('   ');
    write('---------------');
    write('   ');
    writeln('---------------');
    for i:=1 to num do     { print data for each step }
    begin
         write('      ');
         write(ld[i]:6:3);
         write('               ');
         write(rd[i]:6:3);
         write('             ');
         write((rd[i]+ld[i]):6:3);
         write('             ');
         writeln(dd[i]:6:3);
    end;
    for i:=1 to 20-num do
    writeln;             { print spaces until end of screen }
    write('Press ENTER to continue or ');
    writeln('PRINTSCREEN for a hardcopy...');
    readln;                  { wait for enter keypress }
    clrscr;                  { clear screen }
end;
{ end of Procedure Print }
```

```
{main}
begin
     good:=false;{ initially assume file is no good }
     Getinfo(name,age,sex,date); { get subject info }
     Getfile(nl,nr,lh,rh,good);  { get file and data }
     if good then                { if file good continue }
     begin                       { if not abort program }
          Convert(nl,nr,lh,rh,ld,rd,dd);{ convert times and}
          Print(nl,nr,ld,rd,dd,name,age,sex,date); {print}
     end
     else
     begin              { if file not good abort program }
          writeln;
          write('This file does not contain ');
          writeln('any valid times.');
          writeln;
          writeln('PROGRAM ABORTED');
     end;
end.
{ end of program G }
```

```
Program C(input,output);

{ Timothy Hiemenz, Pascal program completed 9-7-94
  This program prompts the user for a time in seconds which
  is to be used as the pace for the auditory biofeedback
  gait trainer. The time is converted to a hex number
  representing the number of clock cycles in the
  microcontroller which equals the time in seconds. Then
  the converted time is printed to the screen so it can
  be loaded into the microcontroller's RAM. }

uses Crt;

type hex=array[1..6] of char;
{ array of char will hold converted result }

var  rtime:real;      { time in seconds }
     htime:hex;       { time in clock cycles }


Function Getchar(tempva:integer):char;

{ This function takes an integer and assigns
  a hex value to  it }

var  tempch:char;      { temporary value }

begin
     case tempva of            { case statement chooses }
          0:tempch:='0';       { the proper value }
          1:tempch:='1';
          2:tempch:='2';
          3:tempch:='3';
          4:tempch:='4';
          5:tempch:='5';
          6:tempch:='6';
          7:tempch:='7';
          8:tempch:='8';
          9:tempch:='9';
          10:tempch:='A';
          11:tempch:='B';
          12:tempch:='C';
          13:tempch:='D';
          14:tempch:='E';
          15:tempch:='F';
     end;
     getchar:=tempch;                { return proper value }
end;
{ end of function Getchar }
```

```
Procedure Gettime(var rtime:real);

{ This procedure reads the time (in seconds) from the
  keyboard, and divides it by the prescale factor in
  the microcontroller. It also divides the number by 256
  so that the internal representations of the number don't
  get too large. }

begin
     clrscr;                    { clear the screen }
     write('Enter the pace (in seconds) to be converted.');
     writeln;
     readln(rtime);            { ask for time }
     rtime:=rtime/0.000008;    { scale the time }
     rtime:=rtime/256.0;       { divide by 256 }
end;
{ end of Procedure Gettime }

Procedure Convert(rtime:real;var htime:hex);

{ This procedure takes the time in seconds and converts
  it to hex characters.}

var  i,                         { looping variable }
     lnum,num,rem:integer;      { integers used in conversion}
     rnum,rrem:real;            { reals used in conversion }

begin
     for i:=1 to 6 do           { loop to clear the }
     htime[i]:='0';             { hex characters }
     i:=4;                      { set i for low order }
     num:=trunc(rtime);         { num=int. part of rtime }
     rnum:=frac(rtime);         { rnum=frac. part of rtime }
     rnum:=rnum*256;            { multiply 256 back in }
     repeat                     { loop to find 4 low }
           lnum:=num;           { order hex characters }
           rem:=num mod 16;
           num:=num div 16;
           htime[i]:=getchar(rem);      { get digit }
           i:=i-1;                      { next digit }
     until num=0;
     rem:=lnum;
     i:=6;                      { set i for high order }
     num:=trunc(rnum);
     repeat                     { loop to find 2 high order }
           lnum:=num;           { hex characters }
           rem:=num mod 16;
           num:=num div 16;
           htime[i]:=getchar(rem);      { get digit }
           i:=i-1;                      { next digit }
     until num=0;
     rem:=lnum;
```

```
end;
{ end of Procedure Convert }

Procedure Print(htime:hex);

{ This procedure prints the hex time on the screen }

var  i:integer;           { looping variable }

begin
     writeln('Use ',htime,' as the pace.');  { print time}
     for i:=1 to 20 do                      { skip 20 lines }
     writeln;
     write('Press ENTER to continue or ');
     writeln('PRINTSCREEN for a hardcopy...');
     readln;                          { wait for enter key }
     clrscr;                          { clear the screen }
end;
{ end of Procedure Print }

{main}
begin
     Gettime(rtime);            { get the time in seconds }
     Convert(rtime,htime);      { convert it to hex }
     Print(htime);              { print out the hex time }
end.
{ end of program C }
```

```
*    Buzzd1 program created by Timothy Hiemenz, 9-7-94
*    This program is half of the auditory biofeedback
*    gait trainer. It can be used without storing a step
*    pace to just measure step times, or can be setup
*    to buzz when a foot should make contact with the
*    floor. The other half of the program is buzzd2 and both
*    have to be stored in EEPROM on the microcontroller.
*
*************************************************************
*
*       Equate Statements
*
o_reg1  equ     $0001
          * register for storing count of overflows
o_reg2  equ     $0000
          * register for storing count of overflows
cont    equ     $002b
          * register for counting number of steps
p       equ     $002c
          * variable to keep track of when to print
x       equ     $0023
          * keeps track of when to decrement pace
pacex1  equ     $0024
          * high order of pace 1
pace1   equ     $0025
          * low order of pace 1
pacex3  equ     $0027
          * high order of pace 3
pace3   equ     $0028
          * low order of pace 3
tflg1   equ     $1023
          * main timer interrupt flag #1
tflg2   equ     $1025
          * main timer interrupt flag #2
tmsk1   equ     $1022
          * main timer interrupt mask #1
tmsk2   equ     $1024
          * main timer interrupt mask #2
tctl1   equ     $1020
          * timer control register #1
tctl2   equ     $1021
          * timer control register #2
output  equ     $ffc1
          * subroutine to output hex time on screen
out     equ     $ffc7
          * subroutine to output characters on screen
outchar equ     $ffb8
          * subroutine to output a character
in      equ     $ffcd
          * subroutine to accept input from keyboard
porta   equ     $1000
          * port a is where ic's connect
```

```
portd   equ     $1008
          * port d is i/o port used for buzz
ddrd    equ     $1009
          * configure's port d for input or output
ri      equ     $b6e3
          * location of right time announcement
le      equ     $b6f2
          * location of left time announcement
ready   equ     $b7e3
          * location of initial announcement
ok      equ     $b7f9
          * location of response to enter key
init    equ     $e36c
          * buffalo device initialization subroutine
vecinit equ     $e34b
          * buffalo vector initialization subroutine
toc1    equ     $1016
          * timer output compare #1 register
toc3    equ     $101a
          * timer output compare #3 register
tic1    equ     $1010
          * timer input capture #1 register
tic3    equ     $1014
          * timer input capture #3 register
*
************************************************************
*
*    Initialization
*    Set up jump table, edge triggerring, clock * prescaling,
etc.
*
        org     $0100
          * store program at $0100 then move to $b600
        ldaa    #$83
          * load A with 10000011
        staa    tmsk2
          * set prescale of 16 and overflow enabled
        jsr     init
          * initialize output to computer
        jsr     vecinit
          * initialize vector table
        ldd     #$b700
          * set up oc1 interrupt vector
        std     $00e0
          * to jump to b700
        ldd     #$b70c
          * set up oc3 interrupt vector
        std     $00da
          * to jump to b70c
        ldd     #$b729
          * set up ic1 interrupt vector
```

```
        std     $00e9
          * to jump to b729
        ldd     #$b766
          * set up ic3 interrupt vector
        std     $00e3
          * to jump to b766
        ldd     #$b7a6
          * set up timer overflow interrupt
        std     $00d1
          * vector to jump to b7a6
        clr     tctl1
          * clear tctl1
        ldaa    #$11
          * sets up input captures to trigger
        staa    tctl2
          * on rising edge when 11 stored to tctl2
        ldaa    #$08
          * sets up d3 as output pin
        staa    ddrd
          * when 08 is stored to ddrd
        clr     portd
          * clear output pins on port d
        ldx     #ready
          * output inital announcement to ask
        jsr     out
          * for enter key to be pressed
        jsr     in
          * wait for keypress
        ldx     #ok
          * output response to keypress
        jsr     out
          * after enter is detected
poll    ldaa    porta
          * poll port a for right foot contact
        anda    #$01
          * so timing can start
        cmpa    #$01
          * with a left foot contact
        bne     poll
          * loop back if no right foot contact
        bsr     clear
          * call subroutine to clear registers
        ldaa    #$04
          * load A with 00000100 and store
start   staa    tflg1
          * to clear the ic1 flag and
        staa    tmsk1
          * set only ic1 flag to cause interrupt
        ldaa    #$80
          * load A with 10000000 and store to
        staa    tflg2
          * clear timer overflow flag
```

```
        cli
          * enables all interrupts
*
****************************************************************
*
*    Main loop where program waits for interrupts to occur.
*    It prints out step times after each left contact is
*    made (except the very first one)
*
main    ldaa    p
          * load A with value held by p
        cmpa    #$01
          * compare A with the number 1
        bne     main
          * if not equal then loop again
        sei
          * set interrupt mask
*
****************************************************************
*
*    Calculation of right foot swing time by subtracting
*    R-L1
*
        ldy     #$0005
          * setup Y to calculate starting with $0005
        ldx     #$000e
          * setup X to store swing time at $000e
        bsr     calc
          * branch to subroutine calc
        jsr     output
          * output first half of hex time
        jsr     output
          * output second half of hex time
        ldx     #ri
          * setup X to print right announcement
        jsr     out
          * output right swing announcement
*
****************************************************************
*
*    Calculation of left foot swing time by subtracting L2-R
*
        ldy     #$0009
          * setup Y to calculate starting with $0009
        ldx     #$0012
          * setup X to store swing time at $0012
        bsr     calc
          * branch to subroutine calc
        jsr     output
          * output first half of hex time
        jsr     output
          * output second half of hex time
```

```
        ldx     #1e
          * setup X to print left announcement
        jsr     out
          * output left swing announcement
*
****************************************************************
*
*    Move second left time to position where first left time
*    is stored and setup program to continue recording times
*
        ldd     $000a
          * load first half of hex time L2
        std     $0002
          * and store it to first half of L1
        ldd     $000c
          * load second half of time L2
        std     $0004
          * and store it to second half of L1
        clr     p
          * clear print variable
        ldaa    #$01
          * load A with 1 to setup ic3 after branch
        bra     start
          * branch back to main program to restart
*
****************************************************************
*
*    Subroutine to clear registers in initialization of
*    program
*
clear   clr     o_reg1
          * clear register o_reg1
        clr     o_reg2
          * clear register o_reg2
        clr     cont
          * clear cont variable
        clr     p
          * clear print variable
        clr     x
          * clear x variable
        rts
          * return from subroutine
*
****************************************************************
*
*    This procedure subtracts the values indexed by Y
*
loop    ldaa    4,y
          * load A with location indexed 4 from Y
        sbca    0,y
          * subtract with carry reg. at Y from A
        staa    $0c,y
```

```
        * store A to location indexed 12 from Y
    dey
        * decrement Y index once
    rts
        * return from subroutine
*
************************************************************
*
*    This procedure calculates the difference between two
*    times that are indexed by Y and stores the result
*
calc    bsr     loop
        * call subroutine loop
        bsr     loop
        * call subroutine loop
        bsr     loop
        * call subroutine loop
        bsr     loop
        * call subroutine loop
        rts
        * return from subroutine
*
************************************************************
*
*    Time announcements -- characters to be output after
*    times are printed to the screen
*
        org     $01e3
        * memory location of characters
        fcc     "is RIGHT time"
        fcb     $0d
        * carriage return character
        fcb     $04
        * end of transmission
        fcc     "is LEFT time"
        fcb     $0d
        * carriage return character
        fcb     $04
        * end of transmission
```

```
*    Buzzd2 program created by Timothy Hiemenz, 9-7-94
*    This program is half of the auditory biofeedback
*    gait trainer. It can be used without storing a step
*    pace to just measure step times, or can be setup
*    to buzz when a foot should make contact with the floor.
*    The other half of the program is buzzd1 and both have
*    to be stored in EEPROM on the microcontroller.
*
*************************************************************
*
*       Equate Statements
*
o_reg1  equ     $0001
          * register for storing count of overflows
o_reg2  equ     $0000
          * register for storing count of overflows
cont    equ     $002b
          * register for counting number of steps
p       equ     $002c
          * variable to keep track of when to print
L1      equ     $0002
          * $02-$05 stores 1st left contact time
R       equ     $0006
          * $06-$09 stores right contact time
L2      equ     $000a
          * $0a-$0d stores 2nd left contact time
c1      equ     $0020
          * extra compare register for oc1
c3      equ     $0021
          * extra compare register for oc3
w       equ     $0022
          * variable to keep track of which foot is next
x       equ     $0023
          * keeps track of when to decrement pace
pacex1  equ     $0024
          * extra pace register for oc1
pace1   equ     $0025
          * pace register for oc1
pacex3  equ     $0027
          * extra pace register for oc3
pace3   equ     $0028
          * pace register for oc3
tflg1   equ     $1023
          * main timer interrupt flag #1
tflg2   equ     $1025
          * main timer interrupt flag #2
tmsk1   equ     $1022
          * main timer interrupt mask #1
tmsk2   equ     $1024
          * main timer interrupt mask #2
tctl1   equ     $1020
          * timer control register #1
```

```
tctl2   equ     $1021
          * timer control register #2
toc1    equu    $1016
          * timer output compare register #1
toc3    equ     $101a
          * timer output compare register #3
tic1    equ     $1010
          * timer input capture register #1
tic3    equ     $1014
          * timer input capture register #3
portd   equ     $1008
          * port d is i/o port used for buzz
outchar equ     $ffb8
          * location of output subroutine

*
***********************************************************
*
*    oc1 interrupt routine to turn on buzz
*
        org     $0100
          * set memory location of routine
        sei
          * set interrupt mask
        bsr     buzz
          * turn on buzz
        ldaa    #$52
          * output an R to the screen
        jsr     outchar
          * to know to expect a right
        ldaa    #$01
          * setup for right foot contact
        bra     ret
          * branch for rest of commands are same

*
***********************************************************
*
*    oc3 interrupt routine to turn on buzz
*
        org     $010c
          * set memory location of routine
        sei
          * set interrupt mask
        bsr     buzz
          * turn on buzz
        ldaa    #$4c
          * output an L to the screen
        jsr     outchar
          * to know to expect a left
        ldaa    #$04
          * setup for left foot contact
```

```
ret     staa    tflg1
          * clear the appropriate flag
        staa    tmsk1
          * set the appropriate mask
        clr     x
          * stop decrementing pace
        rti
          * return and reenable interrupts

*
************************************************************
*
*    Subroutine to turn on buzzer connected to port d pin 3
*
buzz    ldaa    #$08
          * load A with 00001000
        staa    portd
          * store a 1 to pin d3 -- BUZZ
        rts
          * return from subroutine

*
************************************************************
*
*    ic1 interrupt routine -- for left foot contacts
*    stops buzzing, saves contact time, loads next pace
*
        org     $0129
          * set memory location of routine
        sei
          * set interrupt mask
        clr     portd
          * stop buzzing
        bsr     done
          * jump to check if 1st or 2nd left
        bsr     saveos
          * jump to store overflow registers
        ldaa    pacex1
          * load the extra pace register #1
        staa    c1
          * store it to the extra compare register
        ldd     tic1
          * load D with capture time
        std     2,y
          * store capture time
        addd    pace1
          * add pace to capture time
        std     toc1
          * store result in oc#1 register
        bvc     over
          * if too large be sure to
```

```
        inc     c1
          * increase the extra to keep correct
over    ldaa    #$07
          * load ASCII beep character into A
        jsr     outchar
          * output the beep
        clr     w
          * clearing w says current foot = left
        ldaa    #$01
          * load A with setup for right contact
        bra     return
          * branch to rest of routine

done    ldy     #L1
          * setup to store at L1
        ldaa    cont
          * check cont var
        cmpa    #$00
          * compare to 0
        beq     left1
          * if = then jump to left1
        ldy     #L2
          * setup to store at L2
        inc     p
          * increase print variable
left1   inc     cont
          * increase cont variable
        rts
          * return from subroutine

*
************************************************************
*
*    ic3 interrupt routine -- for right foot contacts
*    stops buzzing, stores contact time, loads next pace
*
        org     $0166
          * set memory loaction of routine
        sei
          * set interrupt mask
        clr     portd
          * stop buzzing
        ldy     #R
          * setup Y to store time in R
        bsr     saveos
          * jump to store overflow registers
        ldaa    pacex3
          * load the extra pace register #3
        staa    c3
          * store it to the extra compare register
        ldd     tic3
          * load D with capture time
```

```
        std     2,y
          * store capture time
        addd    pace3
          * add pace to capture time
        std     toc3
          * store result in oc#3 register
        bvc     flow
          * if too large be sure to
        inc     c3
          * increase the extra to keep correct
flow    ldaa    #$07
          * load ASCII beep character into A
        jsr     outchar
          * output the beep
        inc     w
          * incrementing w says foot = right
        ldaa    #$04
          * load A with setup for left foot contact
return  staa    tflg1
          * clears flags for appropriate foot
        staa    tmsk1
          * sets interrupt mask for appropriate foot
        inc     x
          * flag telling to start to decrement pace
        ldaa    #$80
          * load A with 10000000
        staa    tmsk2
          * to set timer overflow to cause interrupt
        staa    tflg2
          * and clear the timer overflow flag
        rti
          * return and reenable interrupts

saveos  ldd     o_reg2
          * load D with overflow register values
        std     0,y
          * store into proper place indexed by Y
        rts
          * return from subroutine
*
************************************************************
*
*    Timer overflow routine
*    keeps track of extra time digits and decrements extra
*    pace registers for appropriate feet
*
        org     $01a6
          * set memory location of routine
        sei
          * set interrupt mask
        ldaa    o_reg1
          * load A with value in o_reg1
```

```
        cmpa    #$ff
          * compare A to 11111111
        bne     endroll
          * if not equal goto endroll
        inc     o_reg2
          * increase o_reg2 once
endroll inc     o_reg1
          * increase o_reg1 once
        ldaa    x
          * load x variable to check whether
        cmpa    #$00
          * or not to be decrementing
        beq     end2
          * if not go to end2
        ldaa    w
          * if so, find out which foot
        cmpa    #$00
          * by testing variable w
        bne     right
          * if w not equal to 0 then right foot
left    ldaa    c1
          * load extra compare register
        dec     c1
          * decrement extra compare register
        cmpa    #$00
          * compare to 0
        bne     end2
          * branch to end2 if not equal
        ldaa    #$81
          * setup oc#1 and ic#3
        bra     skip
          * branch to skip over right procedure
right   ldaa    c3
          * load extra compare register
        dec     c3
          * decrement extra compare register
        cmpa    #$00
          * compare to 0
        bne     end2
          * branch to end2 if not equal
        ldaa    #$24
          * setup oc#3 and ic#1
skip    staa    tflg1
          * clear appropriate flags
        staa    tmsk1
          * and set appropriate mask
end2    ldaa    #$80
          * load A with 10000000
        staa    tflg2
          * clears timer overflow flag
        rti
          * return from interrupt
```

```
*
************************************************************
*
*    Announcements printed at beginning of program
*
ready   fcc     "Hit RETURN to start."
        fcb     $0d
          * carriage return character
        fcb     $04
          * end of transmission character
ok      fcc     "OK!"
        fcb     $0d
          * carriage return character
        fcb     $04
          * end of transmission character
```

```
*    Flash1 program created by Timothy Hiemenz, 9-7-94
*    This program is half of the visual biofeedback
*    gait trainer. It can be used without connecting
*    the light bar to just measure step times, or can
*    be setup to light a number of lights showing which
*    foot is taking longer to contact the floor. The other
*    half of the program is flash2 and both have to
*    be stored in EEPROM on the microcontroller.
*
*************************************************************
*
*       Equate Statements
*
lrdiff  equ     $b769
          * subroutine to find L-R step difference
side    equ     $0023
          * variable of which step time is longer
diff    equ     $001a
          * location of step time difference
o_reg1  equ     $0001
          * register for storing count of overflows
o_reg2  equ     $0000
          * register for storing count of overflows
cont    equ     $002b
          * register for counting number of steps
p       equ     $002c
          * variable to keep track of when to print
tflg1   equ     $1023
          * main timer interrupt flag #1
tflg2   equ     $1025
          * main timer interrupt flag #2
tmsk1   equ     $1022
          * main timer interrupt mask #1
tmsk2   equ     $1024
          * main timer interrupt mask #2
tctl2   equ     $1021
          * timer control register #2
output  equ     $ffc1
          * subroutine to output hex time on screen
out     equ     $ffc7
          * subroutine to output characters on screen
outchar equ     $ffb8
          * subroutine to output a character
in      equ     $ffcd
          * subroutine to accept input from keyboard
porta   equ     $1000
          * port a is where ic's connect
portb   equ     $1004
          * port b is an output port
portc   equ     $1003
          * port c is an i/o port for output
```

```
ddrc    equ     $1007
          * data direction register for port c
ri      equ     $b6e3
          * location of right time announcement
le      equ     $b6f2
          * location of left time announcement
ready   equ     $b6c8
          * location of initial announcement
ok      equ     $b6de
          * location of response to enter key
init    equ     $e36c
          * buffalo device initialization subroutine
vecinit equ     $e34b
          * buffalo vector initialization subroutine
tic1    equ     $1010
          * timer input capture #1 register
tic3    equ     $1014
          * timer input capture #3 register
*
************************************************************
*
*    Initialization
*    Sets up jump table, edge triggering, clock prescaling,
*    etc.
*
        org     $0100
          * store program at $0100 then move to $b600
        ldaa    #$83
          * load A with 10000011
        staa    tmsk2
          * set prescale of 16 and overflow enabled
        jsr     init
          * initialize output to computer
        jsr     vecinit
          * initialize vector table
        ldd     #$b700
          * set up ic1 interrupt vector
        std     $00e9
          * to point to b700
        ldd     #$b732
          * set up ic3 interrupt vector
        std     $00e3
          * to point to b732
        ldd     #$b756
          * set up timer overflow interrupt
        std     $00d1
          * vector to point to b756
        ldaa    #$11
          * sets up input captures to trigger
        staa    tctl2
          * on rising edge when 11 stored to tctl2
```

```
        ldaa    #$ff
          * sets up all pins of portc
        staa    ddrc
          * to be output ports
        ldx     #ready
          * output initial announcement to ask
        jsr     out
          * for enter key to be pressed
        jsr     in
          * wait for keypress
        ldx     #ok
          * output response to keypress
        jsr     out
          * after enter key is detected
poll    ldaa    porta
          * poll port a for right foot contact
        anda    #$01
          * so timing can start
        cmpa    #$01
          * with a left foot contact
        bne     poll
          * loop back if no right foot contact
        bsr     clear
          * call subroutine to clear registers
        ldaa    #$04
          * load A with 00000100 and store
start   staa    tflg1
          * to clear ic1 flag and
        staa    tmsk1
          * set only ic1 flag to cause interrupt
        ldaa    #$80
          * load A with 10000000 and store to
        staa    tflg2
          * clear timer overflow flag
        cli
          * enables all interrupts
*
************************************************************
*
*    Main loop where program waits for interrupts to occur.
*    It prints out step times after each left contact is
*    made. (except the very first one)
*
main    ldaa    p
          * load A with value held by p
        cmpa    #$01
          * compare A with the number 1
        bne     main
          * if cont is less then 2 loop again
        sei
          * set interrupt mask
*
```

```
************************************************************
*
*    Calculation of right foot swing time by subtracting
*    R-L1
*
        ldy     #$0005
          * setup Y to calculate starting with $0005
        ldx     #$000e
          * setup X to store swing time at $000e
        bsr     calc
          * branch to subroutine calc
        jsr     output
          * output first half of hex time
        jsr     output
          * output second half of hex time
        ldx     #ri
          * setup X to print right announcement
        jsr     out
          * output right swing announcement

*
************************************************************
*
*    Calculation of left foot swing time by subtracting L2-R
*
        ldy     #$0009
          * setup Y to calculate starting with $0009
        ldx     #$0012
          * setup X to store swing time at $0012
        bsr     calc
          * branch to subroutine calc
        jsr     output
          * output first half of hex time
        jsr     output
          * output second half of hex time
        ldx     #le
          * setup X to print left announcement
        jsr     out
          * output left swing announcement
*
************************************************************
*
*    Calculation of step time difference L-R or R-L
*    (depending which is larger) and then
*    move second left time to position where first left time
*    is stored and setup program to continue recording times
*
        jsr     lrdiff
          * call subroutine to calculate difference
        ldd     $000a
          * load first half of hex time L2
```

```
        std     $0002
          * and store it to first half of L1
        ldd     $000c
          * load second half of time L2
        std     $0004
          * and store it to second half of L1
        clr     p
          * clear print variable
        clr     side
          * clear the side variable
        ldaa    #$01
          * load A with 1 to setup ic3 after branch
        bra     start
          * branch back to main program to restart

*
************************************************************
*
*    Subroutine to reset registers
*
clear   clr     o_reg1
          * clear register o_reg1
        clr     o_reg2
          * clear register o_reg2
        clr     cont
          * clear cont variable
        clr     p
          * clear print variable
        clr     side
          * clear the side variable
        clr     portb
          * clear portb (right light bar)
        clr     portc
          * clear portc (left light bar)
        rts
          * return from subroutine
*
************************************************************
*
*    This procedure subtracts the values indexed by Y
*
loop    ldaa    4,y
          * load A with location indexed 4 from Y
        sbca    0,y
          * subtract with carry reg. at Y from A
        staa    $0c,y
          * store A to location indexed 12 from Y
        dey
          * decrement Y index once
        rts
          * return from subroutine
*
```

```
***********************************************************
*
*    This procedure calculates the difference between two
*    times that are indexed by Y and stores the result
*
calc    bsr     loop
          * call subroutine loop
        bsr     loop
          * call subroutine loop
        bsr     loop
          * call subroutine loop
        bsr     loop
          * call subroutine loop
        rts
          * return from subroutine
*
***********************************************************
*
*    Announcements printed at beginning of program
*
        org     $01c8
          * set memory location of characters
        fcc     "Hit RETURN to start."
        fcb     $0d
          * carriage return character
        fcb     $04
          * end of transmission
        fcc     "OK!"
        fcb     $0d
          * carriage return character
        fcb     $04
          * end of transmission
*
***********************************************************
*
*    Time announcements -- characters to be output after
*    times are printed to the screen
*
        org     $01e3
          * set memory location of characters
        fcc     "is RIGHT time"
        fcb     $0d
          * carriage return character
        fcb     $04
          * end of transmission
        fcc     "is LEFT time"
        fcb     $0d
          * carriage return character
        fcb     $04
          * end of transmission
```

```
*    Flash2 program created by Timothy Hiemenz, 9-7-94
*    This program is half of the visual biofeedback
*    gait trainer. It can be used without connecting
*    the light bar to just measure step times, or can
*    be setup to light a number of lights showing which
*    foot is taking longer to contact the floor. The other
*    half of the program is flash1 and both have to
*    be stored in EEPROM on the microcontroller.
*
************************************************************
*
*       Equate Statements
*
diff    equ     $001a
          * location of step time difference
side    equ     $0023
          * variable of which step time is longer
calc    equ     $b6b4
          * location of calc subroutine
o_reg1  equ     $0001
          * register for storing count of overflows
o_reg2  equ     $0000
          * register for storing count of overflows
cont    equ     $002b
          * register for counting number of steps
p       equ     $002c
          * variable to keep track of when to print
L1      equ     $0002
          * $02-$05 stores 1st left contact time
R       equ     $0006
          * $06-$09 stores right contact time
L2      equ     $000a
          * $0a-$0d stores 2nd left contact time
w       equ     $0022
          * variable to keep track of which foot is next
tflg1   equ     $1023
          * main timer interrupt flag #1
tflg2   equ     $1025
          * main timer interrupt flag #2
tmsk1   equ     $1022
          * main timer interrupt mask #1
tctl2   equ     $1021
          * timer control register #2
tic1    equ     $1010
          * timer input capture register #1
tic3    equ     $1014
          * timer input capture register #2
outchar equ     $ffb8
          * location of output subroutine
output  equ     $ffc1
          * subroutine to output hex time to screen
```

```
portb   equ     $1004
          * port b is output port
portc   equ     $1003
          * port c is i/o port used as output port
*
************************************************************
*
*    ic1 interrupt routine -- for left foot contacts
*    turns off lights, saves contact time, etc
*
        org     $0100
          * set memory location of routine
        sei
          * set interrupt mask
        clr     portb
          * clear output port b and c to
        clr     portc
          * turn of light bars
        bsr     done
          * jump to check if 1st or 2nd left
        bsr     saveos
          * jump to store overflow registers
        ldd     tic1
          * load D with capture time
        std     2,y
          * store capture time
        ldaa    #$07
          * load ASCII beep character into A
        jsr     outchar
          * output the beep
        clr     w
          * clear w variable says foot = left
        ldaa    #$01
          * load A with setup for right contact
        bra     return
          * branch to rest of routine

done    ldy     #L1
          * setup to store at L1
        ldaa    cont
          * check cont var
        cmpa    #$00
          * compare to 0
        beq     left1
          * if = then jump to left1
        ldy     #L2
          * setup to store at L2
        inc     p
          * increase print variable
left1   inc     cont
          * increase cont variable
```

```
        rts
          * return from subroutine
*
************************************************************
*
*    ic3 interrupt routine -- for right foot contacts
*    store contact times, etc
*
        sei
          * set interrupt mask
        ldy     #R
          * setup Y to store time in R
        bsr     saveos
          * jump to store overflow registers
        ldd     tic3
          * load D with capture time
        std     2,y
          * store capture time
        ldaa    #$07
          * load ASCII beep character into A
        jsr     outchar
          * output the beep
        inc     w
          * incrementing w says foot = right
        ldaa    #$04
          * load A with setup for left foot contact
return  staa    tflg1
          * clears flags for appropriate foot
        staa    tmsk1
          * sets interrupt mask for appropriate foot
        rti
          * return and reenable interrupts

saveos  ldd     o_reg2
          * load D with overflow register values
        std     0,y
          * store into proper place indexed by Y
        rts
          * return from subroutine
*
************************************************************
*
*    Timer overflow routine
*    keeps track of extra time digits
*
        sei
          * set interrupt mask
        ldaa    o_reg1
          * load A with value in o_reg1
        cmpa    #$ff
          * compare A to 11111111
```

```
        bne     endroll
          * if not equal goto endroll
        inc     o_reg2
          * increase o_reg2 once
endroll inc     o_reg1
          * increase o_reg1 once
        ldaa    #$80
          * load A with 10000000
        staa    tflg2
          * clears timer overflow flag
        rti
          * return from interrupt
*
************************************************************
*
*    Calculation of difference L-R or R-L depending on
*    which is larger
*
lrdiff  ldy     #$0011
          * start with taking L-R by indexing Y
        ldx     #diff
          * and indexing X to result
        jsr     calc
          * jump to subroutine calc
        ldaa    diff
          * load the first byte of the result
        cmpa    #$ff
          * compare to FF which means diff is negative
        bne     go
          * if positive L-R > 0
        inc     side
          * else R-L > 0 so increment side variable
*
************************************************************
*
*    Step time swap - this section swaps time locations
*    if L-R < 0, so the same calc procedure can be used to
*    find R-L
*
        ldd     $000e
        std     $001a
        ldd     $0010
        std     $001c
        ldd     $0012
        std     $000e
        ldd     $0014
        std     $0010
        ldd     $001a
        std     $0012
        ldd     $001c
        std     $0014
```

```
        bra     lrdiff
          * branch back to find difference R-L
*
************************************************************
*
*    Light bar control -- this section examines the step
*    time difference and by scaling the value, determines
*    how many lights on the display should be turned on.
*
go      ldx     #diff
          * index X to the step time difference
        ldd     1,x
          * load the second highest byte of diff
        cpd     #$01e8
          * compare diff to 1 seconds
        bge     eight
          * branch if >= to eight
        cpd     #$016e
          * compare diff to .75 seconds
        bge     seven
          * branch if >= to seven
        cpd     #$0124
          * compare diff to .6 second
        bge     six
          * branch if >= to six
        cpd     #$00f4
          * compare diff to .5 seconds
        bge     five
          * branch if >= to five
        cpd     #$00c3
          * compare diff to .4 seconds
        bge     four
          * branch if >= to four
        cpd     #$0092
          * compare diff to .3 seconds
        bge     three
          * branch if >= to three
        cpd     #$0061
          * compare diff to .2 seconds
        bge     two
          * branch if >= to two
        cpd     #$0030
          * compare diff to .1 seconds
        bge     one
          * branch if >= to one
        bra     store
          * branch to store
eight   ldaa    #$ff
          * turn on 8 lights
        bra     store
          * branch to store
```

```
seven   ldaa    #$7f
          * turn on 7 lights
        bra     store
          * branch to store
six     ldaa    #$3f
          * turn on 6 lights
        bra     store
          * branch to store
five    ldaa    #$1f
          * turn on 5 lights
        bra     store
          * branch to store
four    ldaa    #$0f
          * turn on 4 lights
        bra     store
          * branch to store
three   ldaa    #$07
          * turn on 3 lights
        bra     store
          * branch to store
two     ldaa    #$03
          * turn on 2 lights
        bra     store
          * branch to store
one     ldaa    #$01
          * turn on 1 light
store   ldab    side
          * determine which side to light
        cmpb    #$00
          * if side = 0 light left side
        beq     left
          * branch to left if = 0
        staa    portb
          * else light right side
        bra     end
          * branch to end
left    staa    portc
          * light left side
end     rts
          * return from subroutine
```