Control Logic and Small Signal Interface of a Kinetic Sculpture

Submitted By
Marc Weintraub

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR AN UNDERGRADUATE THESIS WITH A

BACHELOR OF ELECTRICAL ENGINEERING

School of Engineering
Tufts University
Medford, Massachusetts

May 2004

Signature of Author:
Marc Weintraub

Certified By:
Associate Professor Joseph P. Noonan
Department of Electrical Engineering
Tufts University

Committee:
Professor Chris Rogers
Department of Mechanical Engineering
Tufts University

Committee:
Professor Chorng Hwa Chang
Department of Electrical Engineering
Tufts University

© Copyright 2004, by Marc Weintraub
0. Abstract: ........................................................................................................................ 5

I. Introduction: .................................................................................................................. 6

II. Background: ................................................................................................................. 7

1. Inputs: ............................................................................................................................ 7
   A. Simon: ....................................................................................................................... 8
   B. Laser Maze: .............................................................................................................. 9
   C. Distance Piano: ....................................................................................................... 10
   D. Grip: ....................................................................................................................... 11
   E. Noise: ...................................................................................................................... 12

III. Design and Implementation: ................................................................................... 15

1. Microcontrollers and Interface: .................................................................................. 15
2. Simon: ........................................................................................................................ 16
   A. Circuit Implementation: ......................................................................................... 16
3. Laser Maze & Analog to Digital conversion ................................................................. 22
   A. Circuit Design and Implementation: ...................................................................... 22
   B. Code - Analog to Digital Conversion: .................................................................... 25
   C. Code – Laser Maze PIC A: .................................................................................... 26
   D. Code – Laser Maze PIC B: .................................................................................... 29
4. Distance Piano & Music (Timer Interrupts with Multiple Timer Registers) ............... 31
   A. Circuit Design and Implementation: ...................................................................... 31
   B. Code - Timer Interrupts: ........................................................................................ 32
5. Grip ................................................................................................................................ 38
   A. Circuit Design and Implementation: ...................................................................... 38
6. Noise ............................................................................................................................. 42
   A. Circuit Design and Implementation: ...................................................................... 42
7. Master ............................................................................................................................ 46
   A. Circuit Design and Implementation: ...................................................................... 46
   B. Code – Pulse Width Modulation (PWM): ............................................................... 49
   C. Code – Master: ...................................................................................................... 50

IV. Analysis: .................................................................................................................... 52

V. Conclusion: ................................................................................................................. 53

VI. Bibliography: ............................................................................................................ 54

VII. Appendix: ................................................................................................................. 55

Master PIC Code ................................................................................................................ 56
Simon Code ....................................................................................................................... 71
Distance Piano Code ....................................................................................................... 78
Grip Code ........................................................................................................................ 93
Laser Maze Code – PIC A .............................................................................................. 100
Laser Maze Code – PIC B .............................................................................................. 105
Noise Code ..................................................................................................................... 110
Abstract:

The ‘Kinetic Sculpture’ was a project worked on for one year, during the fall semester of 2003 to spring semester of 2004. The basic idea originated from the well known Rube Goldberg machine, a sculpture in which a ball rolls down a track and causes events to occur. Since the ‘Kinetic Sculpture’ was designed for children, stimulating tasks were incorporated with each of the events to keep their attention.

In order to make such a sculpture to work, electro-mechanical components interfaced to microcontrollers were necessary. This report focuses on the implementation and discussion of the control logic and small signal interface to the kinetic sculpture. A total of six modules were implemented, one to control the mechanics of the sculpture and five to be the brains of five different input tasks. PIC microcontrollers were used for this project, and interfaced on custom built boards that fulfill the requirements of each individual module. The interface boards and code implemented for each module will be thoroughly explained and discussed.
I. Introduction:

In the beginning of the fall semester of 2003, 6 students of various majors under the supervision of the robotics academy came together to form a project team. Uniting the skills of mechanical engineering, electrical engineering, engineering psychology and child development, the result of a weeks brainstorm was an interactive kinetic sculpture designed for children. The general concept behind the sculpture was to have a ball roll down a track, sequentially going through five different events that would be triggered by five different inputs. The ball would be stopped at different points of the sculpture before an event, and upon completion of a task the sculpture would allow the ball to continue along the track.

Various ideas were discussed for the events and the inputs. The events had to be interesting enough to grab the children’s attention, and the input tasks could not be too complicated to figure out. Weighing out these factors with what was possible for the engineers to implement, the following inputs and events were agreed upon:

Inputs: Simon, Laser Maze, Distance Piano, Grip, Noise.
Events: Elevator, Gate, Turnstile, Funnel, Accelerator.

Specific details of each input game and event will be discussed in the next section.

The Electrical Engineering aspect of this project was to select the appropriate electronic components, design and implement interfaces to make everything work from electricity drawn through a 120V AC source. This can be broken into two tasks:

1. Supplying the appropriate power to electro-mechanical components within the events and inputs.
2. Controlling the events and inputs by sending appropriate signals to other modules and electrical components.

This paper will deal with the latter subject.

II. Background:

1. Inputs:

![Input Devices](image)

Figure A1: Input Devices

The five input games were meant to stimulate the children by having them match patterns through different means of interactions, ranging from pressing buttons to making noise. The games were to also give different types of feedback, varying from musical tones to displaying attractive light patterns.
A. Simon:

Figure A2. Simon Input

Like its predecessor, the objective of Simon is to repeat a sequence of four signals that are displayed at the beginning of the game. This game challenges the player by testing their ability to memorize a sequential pattern, which is randomly chosen and displayed on the buttons. The player must repeat the pattern by pressing the buttons in the correct order they were displayed in order to win. A colorful pattern is displayed on the buttons upon victory. If an incorrect button is pressed at any point of the game, a red light is lit to signal a wrong input. The game displays another pattern and allows the player to try again, until victory is achieved.
B. Laser Maze:

Laser Maze is the technologically advanced version of the classical “connect-the-dots”. It tests the player’s ability to steadily tracing a pattern with a laser pointer. Ten dots are printed on the board of the input device, which trace out a star. The correct tracing order is indicated by a number located by the dot, labeled from one to ten. When the player shines the laser over a dot, a light next to it is illuminated if it was in the right sequence. Otherwise, all the lights that have been on turn off and the game indicates the player has lost and must start over. Once the player successfully illuminates every point, the lights blink in a manner to indicate victory.
C. Distance Piano:

Distance Piano is a musical note matching game. However, instead of a piano, the player is to match the note by keeping their hand a certain distance away from the platform of the input device. At the beginning of the game, a tone is played and a corresponding note in a stave printed on the device lights up. There are a total of 9 lights, with two lights indicating whether the note is in treble or bass clef. Once the clef is randomly chosen, an arbitrary note of that clef is played on the speaker and displayed on the lights. Since there are 7 notes on each clef with one note overlapping, there are a total of 13 possible notes that could be chosen from. When the player puts their hand over the platform, a certain tone is played and the corresponding note in the stave lights up. Once the player believes the note matches with that played at the beginning, he/she is to push a button which would check to see whether the input is correct or not. If it is correct, a rising scale is played and one of three victory status lights is lit. Three notes must be
matched successfully in order to win the game, when the game will play a tune from the classical square soft game “Final Fantasy”. Whenever a note is unsuccessfully matched, a low toned beep indicates the input was wrong and no new lights light up for the victory status.

**D. Grip:**

![Figure A5. Grip Input](image)

The Grip is a simple game that requires the player to press a trigger on a device until the amount of compression corresponds to a level indicated by a flashing light on the bar of the game. The bar consists of a total of eight lights, made up of four different color lights. From top to bottom, the color is ordered red, orange, yellow and green. When the grip is pressed all the way, all the lights are lit. The level on the light bar decreases as the grip is slowly released. The game randomly chooses one of the eight lights on the bar, and has it flash for the entire game. When the level of the bar reaches that light, the flashing ceases. When the player is confident that he/she has reached the right level, a lighted button located on the input device is to be pressed to indicate the
completion of the task. If the match is correct, the lights scroll up and down on the bar to indicate a victory. If the match was incorrect, the game indicates the input was wrong and lets the player try again to match the same flashing light.

E. Noise:

![Figure A6. Noise Input](image)

The Noise input is similar to the grip in terms of output. It has seven lights made up of three different colors. The objective is to match the middle light by producing an appropriate sound level. The player is to produce noise of with a moderate degree of loudness for a short amount of time until victory is achieved. Unlike the other games, there is no wrong input. The lights act as a decibel meter until the task is completed, when a unique victory pattern would be displayed.
2. Events:

All the events are to occur in sequence after each input. In the main sculpture, a ball first enters an elevator platform where it sits until an input game is completed. Upon victory of the game, the elevator will raise the ball to the top of the sculpture and drop it off onto an elevated track. The elevator shaft then goes back down to its original position. The ball would continue to roll until it is stopped by a gate, which will remain closed until the second input game is played and won. The ball then enters a turnstile, which is activated upon victory of the third game. The turnstile rotates 90 degrees and transports the ball to the next part of the track which leads it to the entrance of a funnel. Completion
of the fourth game allows the ball to proceed into the funnel, where it swirls around until it reaches the bottom. When the last game is completed, the ball is rolled into an accelerator which shoots it around a vast loop spanning the side of the sculpture. The ball gets back on track, and stops at the entrance of the elevator where it originally started off.
III. Design and Implementation:

1. Microcontrollers and Interface:

   A total of six modules were needed to control the five inputs and events. Each module required a microcontroller, and circuitry to interface it with other various components. Since the algorithms of these modules deal mostly with turning on lights and checking 1-bit inputs (or channels), no expensive microcontrollers with pre-built compilers or fancy features were necessary. The PIC 16F8xx/16F8x series microcontrollers were chosen due to its low cost and easily accessible software and reference manuals.

   The 16F88 and 16F877A were the most used PICs in this project, since both have multiple analog-to-digital converters (ADC) and more than one timer register. These were necessary features for some of the modules. The fall back of these PICs were that they weren’t JDM compatible, a simple programmer that could be built with off the shelf components. A warp13 programmer from Newfound electronics was purchased to overcome this problem.

   The PICs on each module require the following: A 5 volt power source, a ground connection, a value on the Master Clear (MCLR) pin, and a capacitor with very fast switching characteristics. Every module has 5 volts connected to the Vdd pin(s), ground to Vss pin(s) and a 0.1 uF tantalum capacitor in parallel to the 5 volt power source. The capacitor is needed to allow the PIC to change values on the Ports at very high speeds. Besides the PIC 16F88 (which has an internal oscillator), the PICs need an external oscillator. A crystal 4 MHz oscillator was connected to pins CLK1 and CLK2. Master Clear (MCLR)’, the enable pin of the PIC was connected to 5 volts through a 470 ohm
current limiting resistor. Whenever power was supplied to the modules the PIC would be active. 470 ohm current limiting resistors were used throughout every module to keep the current around 10mA, a safe but yet sufficient value for PICs and LEDs.

2. Simon:

A. Circuit Implementation:

Figure B1: Simon Circuit Diagram

The main attractions of the Simon input device are four large colorful illuminated buttons, which act as the inputs and outputs of the game. Since there is no need for any special features such as analog-to-digital conversion, the classical 16F84A was used. The buttons are Single-Pole Double-Throw (SPDT) switches, embedded on a 12 volt lamp. The double-throw ends of the SPDTs were connected to the 5 volt source and ground in a...
manner that when the button was released, the voltage on the single-pole would be ground. The single poles of each of the switches were connected to input ports on the PIC through 470 ohm current limiting resistors. Since the PIC outputs could not power the 12 volt lamps by themselves, N-FETS (NDP603AL) were used as a 5 volt switch that would turn on a 12 volt circuit. The output of the PIC was connected to the gate terminal, and the lamp was connected in series with the drain and a 12 volt source. The source end of the N-FET was connected to ground. Whenever the PIC outputted 5 volts, current would flow from the drain to source, supplying power to the 12 volt lamp.

Two outputs from the PIC were connected to a Red LED and a Green LED, lights used to indicate wrong and victory respectively. These outputs were also connected to two of eight leads on an RJ45 jack, which would serve as the primary interface between the Simon circuit and the Master. Another lead from the RJ45 was designated as the Activation Bit, a signal from the main sculpture that would tell the Simon circuit to begin the game. A reset button was installed, with one end of the switch connected to 5 volts and another to ground through a resistor. Another resistor connected the output of the switch to an input port of the PIC. Since the switch wasn’t an SPDT, this circuitry was necessary to send 0 volts on the output when the button is released, and 5 volts when pressed.

![Figure B2: Simon Circuit](image)
B. Code:

**SIMON INPUT**

![Flowchart Diagram]

*Figure B2: Simon Code Flow*
The game generates a pattern, displays that pattern, checks for inputs and gives feedback to the player. A total of six output ports and six input ports on the microcontroller are required to make this possible. All four bits of Port A, bits B4 and B5 on Port B are configured as inputs, and the remaining six bits on Port B are configured as outputs. Since the pattern in Simon is a sequence of four outputs, four registers holding each data were needed. These were declared and labeled as DATAONE, DATATWO, DATATHREE and DATAFOUR. An inherent timer register TMR0 is also initialized, which begins counting in modulus 256 when the code runs. The code begins in the ‘passive’ function, where it would constantly check for an activation bit sent from the Master on bit 5 of Port B.

Once the activation bit is set, the game enters the ‘generate’ function where the pattern to be mimicked is generated. Since there is no random function generator in the PIC instruction set architecture, the value on the timer register TMR0 is used. The value on TMR0 is captured, and stored into a separate register, Clock. This is a random number because it is indeterminate how long the code remains in the ‘passive’ function, in which TMR0 constantly increments in modulus 256. The 8 bit value in Clock is broken into four segments, each of two bits that are then decoded into a four bit value. This was done by taking the least two significant bits, and sending them to the ‘decoder’ function which would returns a four bit value with only one bit set. The two bits select which of the four bits to be set. The four bit value is then loaded onto DATAONE. The Clock register is shifted twice to the right, and the new two least significant bits are sent to the decoder to acquire another 4 bit value which is loaded onto DATATWO. The process is repeated for DATATHREE and DATAFOUR.
At the point where all the data is set, the game displays the pattern in the ‘displayPattern’ function. Each data is sequentially loaded onto the lower four bits of Port B for a brief period of time. However, since the microprocessor runs at very high speeds (4 million instruction cycles/second), a delay loop is required to keep the outputs on for enough time to be seen by the player. A delay loop is a segment of code that iterates for a given amount of counts to create a pause in the program. In order to have the code wait for a second, 4 million instructions need to be performed. A register holding a count value isn’t sufficient enough, since it can only hold 256 counts. Hence, a simple double loop with four NOPs\(^2\) was used to implement the ‘wait’ function, which delayed the program by a sufficient amount of time. The hex values 0xFF (decimal for 256) were loaded onto the counts for the inner and outer loop. The value in DATAONE is loaded onto the output port, and ‘wait’ would be called. The output port is cleared, and ‘wait’ would be called again. This repeats for the three other data values.

Once all four data segments of the pattern are displayed, the code enters a state checking for the player’s input. If the input matches DATAONE, the output bit corresponding to that on Port B is set. If it doesn’t match, the ‘wrong’ function is called, which sets bit 6 (labeled Wrong bit) on Port B and makes the code jump back to the ‘generate’ function. The checking is done by using an XOR function, which performs an exclusive-or operation on two registers. If two registers are equal, then the output is a zero and the Z-bit in the status register is set. The program checks this Z-bit to see whether the input register matches the DATAONE register.

An important issue in Simon was to make sure that the code doesn’t jump to check the next input while the button was still pressed from the previous input. This was

---

\(^2\) No Operation: Command that takes up 1 clock cycle but has no operation. Good for delays.
overcome by replacing the original ‘inputcheck’ function by two checks, one to check the input and another to check whether the button was released or not. As long as if the input matched the previous data, the code would continuously loop until the input is reset. It is then when the next ‘inputcheck’ takes place.

After four correct checks, the game calls the ‘victory’ function which outputs a pattern to the lower four bits of Port B, sets bit 7 (labeled victory bit) for a certain amount of delay and returns to the main function. The win pattern is a simple cycle that lights each output one at a time. The delay for each light to be on is shorter than the delay for ‘displayPattern’, since ‘wait2’ is called instead. ‘wait2’ has the value 0x77 (decimal for 120) loaded on the loop counters instead of 0xFF. Once in the main state, if input bit 4 (reset input) on Port A is set then the code jumps back to the ‘generate’ function.
3. Laser Maze & Analog to Digital conversion

A. Circuit Design and Implementation:

The way to acquire a value from a light source is with a photocell. A photocell decreases in resistance as the intensity of light on its surface increases. Hence, a simple voltage divider would give an analog output that would be proportional to the light intensity. A photocell ranging from 40 ohms to 20K ohms was chosen, and the

Figure B3: Laser Maze Circuit Diagram
appropriate resistor value to be connected in series with it was determined through breadboard experimentation. A simple test program that converts analog-to-digital then displays that value on the 8-bit Port B was implemented and loaded onto a 16F88 PIC. The outputs were connected to eight LEDs, and the output of a voltage divider circuit consisting of a photocell and a potentiometer was connected to the ADC³ channel of the PIC.

![Voltage Divider Diagram](image)

Figure B4: Voltage Divider

The breadboard was placed in a brightly lit environment, and the potentiometer was adjusted until the LED displaying the most significant bit on Port B was off. A laser was shined on the sensor, and the potentiometer was adjusted again until the most significant LED turned on. The circuit was now set so that even in a bright room the sensor wouldn’t trigger the most significant bit unless the laser hits it. The resistance of the potentiometer was set to roughly 270 ohms, and resistors of that value were obtained to construct Laser Maze.

The pattern to be traced by the laser was a star, and hence ten ADC channels were needed to connect to photocells. Since most PICs are limited to eight ADC ports, two 18-port 16F88 PICs were chosen to implement Laser Maze. Five ADC ports were designated for the photocells on each PIC, and five output ports were designated to display a corresponding LED. The two PICs also needed ports to communicate with each other. Two input pins on each of the PICs were used to read the appropriate status of the other

---

³ ADC = Analog-to-Digital Conversion. This term will be used from now on.
PIC, and one pin on each were used for the Victory and Wrong signals to be sent to the Master. The PIC with the lower 5 sensors and LEDs (sensor-LED pairs were labeled from 0 to 9) was called PIC A, and was made to output the Wrong bit on bit 7 of Port B. The other PIC was called PIC B and had the Victory bit on bit 7 of Port B.

Figure B5: Sensor and LED interface of Laser Maze

Figure B6: Laser Maze Circuit
B. Code - Analog to Digital Conversion:

The key function for this module is the ‘a2d’, which is used in many other modules. Before anything, the ADC control registers ADCON1 and ADCON0, and ADC select register ANSEL needed to be initialized appropriately. ADCON1 controls the format of the 10-bit digital output from the ADC, and together with ADCON0 control the acquisition time for the ADC process. A longer acquisition time gives better results. Tosc/64 was selected, where Tosc is the time for one clock cycle. The format for the 10-bit output was set so that the eight most significant bits would be stored into the ADC output register, ADRESH. The ANSEL register was loaded with the hex value 0x1F, representing 00011111 in binary. The 1’s set the corresponding bits on Port A to ADC channels. It is important that the bits selected as ADC channels are set to inputs. The ADCON0 register alone controls when to begin the ADC, indicates when it stops, and selects which channel to evaluate. Before the ‘a2d’ function is called, the ADCON0 register is set so that the ADC is processed on the appropriate channel. The ‘GO/DONE*’ bit is set to begin the ADC, and the same bit is polled for to check when the process is done. The value in ADRESH is then stored into another register ready for evaluation.
C. Code – Laser Maze PIC A:

Figure B7: Laser Maze PIC A Flow Graph
This PIC checks for five analog inputs, and compares them to a certain threshold. If that threshold exceeds a certain value, the input is checked and if it is the expected value then a corresponding output bit would be set. This would all be performed in the ‘checkInputs’ function. Five ACD channels are constantly evaluated, until one of the channels return a value with the most significant bit set. This would mean a laser had struck the photocell. Since the five checks are performed within microseconds, there will not be a situation where more than one channel would return a value with the most significant bit set. Five bits in the ‘thisInput’ register are used to represent each channel. For example, bit 0 is set when ADC channel 0 gets a value that sets the most significant bit. The objective of Laser Maze is to sequentially connect 10 dots. Hence, the code needs to make sure that ADC channel 0 gets triggered first, then ADC channel 1, and so on. A register ‘nextInput’ is used to check the input, which is initialized at the beginning with bit 0 set. If no input is acquired (‘thisInput’ is zero), then the check function iterates continuously. Once an input is acquired and ‘thisInput’ matches with ‘nextInput’, a corresponding output bit is set. Otherwise, the code goes to the ‘Wrong’ function which outputs a bit to signal a wrong input, and then resets the game. The ‘Wrong’ function is also called when PIC B signals to PIC A (this code) that a wrong input has been acquired. Input bit 5 on Port B is used to check for this. When PIC B signals to PIC A on bit 6 of Port B, the code enters the ‘Victory’ function. Here, the code waits until bit 6 is reset, which means PIC B will begin its unique display on the five output pins. PIC A would then begin its sequence, synchronized with PIC B.
Like Simon, timing was an issue for Laser Maze. However, the ADC from the laser striking a photocell is not constant like the output of a button being pressed. Even though a relatively long sampling rate was chosen for the ADC, it still leaves room for possible glitches. Hence, a delay loop is used at the very end of the check. This makes the ‘checkInput’ function evaluate the values on the sensors around a second, just about enough time for a player to move a laser point onto a sensor, acknowledge a feedback light, and then move the pointer away. If the laser point is kept on for longer, it would be considered a wrong input but then again this feature made the game more interesting.
D. Code – Laser Maze PIC B:

**LASER MAZE INPUT PIC B**

1. Reset Register Values
2. Check Inputs
   - No Status
3. Convert Analog data on Channel 0 to Digital
   - Does Value exceed certain threshold?
     - Yes: Set bit 0 of 'thisinput' register
     - No
4. Convert Analog data on Channel 1 to Digital
   - Does Value exceed certain threshold?
     - Yes: Set bit 1 of 'thisinput' register
     - No
5. Convert Analog data on Channel 2 to Digital
   - Does Value exceed certain threshold?
     - Yes: Set bit 2 of 'thisinput' register
     - No
6. Convert Analog data on Channel 3 to Digital
   - Does Value exceed certain threshold?
     - Yes: Set bit 3 of 'thisinput' register
     - No
7. Convert Analog data on Channel 4 to Digital
   - Does Value exceed certain threshold?
     - Yes: Set bit 4 of 'thisinput' register
     - No
8. Check if all 5 bits of Port B are set on PIC A
   - Yes: Send 'thisinput' to Port B
9. Does 'thisinput' match 'nextinput'?
   - Yes: Increment value of 'nextinput'
   - No
10. Victory Display Victory Sequence
    - Send signal to Master PIC
    - Send signal to Port A

**Figure B8: Laser Maze PIC B Flow Graph**
The code for PIC B is identical to PIC A, with a few exceptions. In the ‘checkInputs’ function, the code checks to see if the last of the five outputs on PIC A is set. This would indicate that all outputs have been set on PIC A. If an input is received on PIC B when the outputs on PIC A are not complete, the code enters into the ‘Wrong’ function which signals PIC A that a wrong input was received. Once all outputs are set on PIC B, the ‘Victory’ function is called, and a victory signal is sent to Master. The last of the five outputs on PIC B is connected to PIC A, and acts as a signal to indicate the game is complete.
4. Distance Piano & Music (Timer Interrupts with Multiple Timer Registers)

A. Circuit Design and Implementation:

Figure B9: Distance Piano Circuit Diagram

The Distance Piano required many output ports, and hence the PIC 16F877A, a massive PIC with a total of 34 I/O pins was used. It was interfaced with a total of 16 LEDs, a yellow light up button, a speaker and a distance sensor. 9 LEDs display the note, 2 display the clefs, and 5 represent the status. The GP2D12 sensor, a pre-built IR
transmitter and receiver calibrated to output an analog voltage dependent on distance, was used for this input device. In addition to the tantalum capacitor, a 10uF electrolyte capacitor was connected in parallel to the 5 volt source to prevent the distance sensor from causing too much current into the PIC. N-Type MOSFETS were used to interface the speaker and the 12 volt lamp in the illuminated button. The speaker was connected to 5 volts through a 20 ohm resistor, and to the drain of the MOSFET. The gate was connected to a pin on the PIC designated for the speaker output.

![Figure B10: Distance Piano Circuit](image)

**B. Code - Timer Interrupts:**

The Distance Piano needed to produce musical output, which requires accurate timing. Sound is a vibration of air waves, and different frequencies produce different tones. In order to produce given tones, the PIC needed to output a square wave of a given frequency. This was accomplished through the use of timer interrupts. Like in Simon, the Timer is a simple counter that increments until the value is 255. Once that value is
reached, it resets itself and an interrupt flag is set. This will be taken advantage of in the ‘playNote’ function to produce tones of given frequencies.

The ‘playNote’ function plays a given note for a given duration. Since there are two independent elements that require time, two different timer registers, TMR0 and TMR2 are used. TMR0 is used as the timer for the duration, and TMR2 is used for half the period of the note. A value is loaded onto a register labeled ‘duration’ and ‘tempNote’ before the ‘playNote’ function is actually called. These store the value of the duration and note respectively.

TMR2 is set to have a period given by the value in ‘tempNote’. Instead of the default 255, TMR2 would only count to the specified value and reset itself. The code enters a loop, which outputs 0 on bit 4 of Port C, the output used for the speaker. The code keeps on looping until an interrupt flag for TMR2 is set. The code then outputs 1, resets the interrupt flag and enters another loop that checks for an interrupt flag for TMR2. The TMR2 interrupt flag is reset, and a TMR0 interrupt flag is checked for. If it is set, the value in the ‘duration’ register is decremented, the interrupt is reset, and the code returns to the loop that outputs 0 to the speaker. Once the value in ‘duration’ is zero, the code exits the ‘playNote’ function.

A total of seventeen notes (A-G on treble clef, B-G on bass clef, A# and G# for both clefs) are used for this module. The values to be loaded on ‘tempNote’ were calculated by the following formula for all seventeen notes:
\[ PR2 = \left( \frac{Fosc}{(4 \times 2 \times F_{note} \times TMR2 \text{ Prescale})} \right) - 1 \]  

PR2 is the period needed to be loaded on the register to produce a given period in real time. Fosc is the frequency of the oscillator, 4MHz. The TMR2 prescale is set to 4 so that values of TMR2 in real time correspond to milliseconds. A table of frequencies\(^5\) was used to get the appropriate frequency for each note, the value needed for Fnote. The value PR2 for each note is loaded onto registers labeled with the name of the note. This makes it easier to access each note later in the code.

The following formula (derived from the previous) was used to calculate the timing for the duration:

\[ \text{duration} = \left( \frac{Fosc \times \text{realTime}}{(4 \times 256 \times TMR0 \text{ Prescale})} \right) - 1 \]

The prescale for TMR0 is set to 32, so that the maximum possible ‘duration’ would cause a 2 second delay. Several desired real time values were converted into ‘duration’ values, such as 1 second and half a second. A new wait function, implemented using TMR0 interrupts is used from now on, since time delays of precise real time values can be used.

---

\(^4\) Formula from [http://www.piclist.com/techref/microchip/16F877/pwm.htm](http://www.piclist.com/techref/microchip/16F877/pwm.htm). Values were rearranged for this specific application.

\(^5\) [http://tomscarff.tripod.com/midi_analyser/midi_note_frequency.htm](http://tomscarff.tripod.com/midi_analyser/midi_note_frequency.htm)
C. Code – Distance Sensor:

Figure B11: Distance Piano Flow Graph
Like Simon, a timer register is used as the random number generator. TMR2 is used because of the lower prescale. One of 7 notes and a clef need to be chosen for this game. A register labeled ‘Clock’ captures the value in the TMR2 register, and the least two significant bits are decoded in the ‘decoder’ function to generate a 4 bit output with one bit set. The third bit of the ‘Clock’ register is used to determine whether to shift the 4 bits three times, so that all seven notes of a given clef could possibly be chosen. This value is stored in the register labeled ‘noteReg’. The fourth bit of the ‘Clock’ register is used to choose the clef, 0 for bass and 1 for treble. Once the choices are made, the code enters the ‘displayAndPlay’ function which outputs the clef and note to the LEDs. Port B and bit 0 of Port C are used to output the notes to the LEDs. The range of notes in treble and bass clef are displayed as follows:

<table>
<thead>
<tr>
<th>Bass Clef</th>
<th>Treble Clef</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Bass Clef" /></td>
<td><img src="image" alt="Treble Clef" /></td>
</tr>
</tbody>
</table>

Figure B12: Note display for different Clefs

If bass is chosen for the clef, then the value on ‘noteReg’ is simply loaded onto Port B later in the code, and the ‘getBassNote’ function is called. In this function, a bass note is selected according to which bit is set in ‘noteReg’. If treble is chosen, ‘getTrebleNote’ is called and a treble note is selected. ‘noteReg’ is then shifted twice to the left. Since ‘noteReg’ is only 8-bits, if bit 6 is set then it would roll off the register after the second shift and ‘noteReg’ would be zero.

The value ‘255’ is loaded onto the ‘duration’ register, and the ‘playNote’ function is called. ‘playNote’ not only plays the note for a given duration, but outputs the
appropriate LED corresponding to the note. The value in ‘noteReg’ is checked to see if it is zero, in which case bit 0 on Port C is set (bit 0 on Port C and Port B together are the outputs to the 9 LEDs). Otherwise, the value on ‘noteReg’ is loaded on to Port B. The ‘playNote’ then plays the note for about 2 seconds. This is the note the player must match, and the values are stored so that they can be checked later.

The code enters the ‘pianoState’, which calls the ‘a2d’ function. Only ADC channel 0, the pin connected to the output of the distance sensor is evaluated. The value of the conversion is stored in register ‘adVal’, which is masked so that the 3 least significant bits are discarded. A note is selected depending on this value, and played for a very short duration of time. The ‘pianoState’ keeps on looping, producing a tone based on the value in ‘adVal’. If the masked value in ‘adVal’ is zero, no tone is played.

An input, coming from an illuminated button on pin 1 of Port A is checked for in the ‘pianoState’. Once this is set, the code enters a check sequence that compares the last note played in the ‘pianoState’ to that played at the beginning. If there is a match, the code enters the correct sequence which plays a rising scale, then sets one of three win status bits. The game must be won three times, and hence all three win status bits must be set. If the values don’t match, the code enters the ‘wrong’ function which sets bit 2 of Port D, the wrong bit that lights a red LED and a signal to the Master. The code goes back to the generate function, and the sequence is repeated until 2 other notes are matched. When all three notes are matched, the ‘victory’ function is called. Bit 3 of Port D, which lights a green LED and signals the Master is set and a tune from the game ‘Final Fantasy’ is played.
5. Grip

A. Circuit Design and Implementation:

A slot car grip is used as the main input for this device. The grip varies in resistance, as the trigger is pressed. When the grip is fully pressed, the output resistance is zero ohms. When it is released, it reads infinite resistance. However, when the grip is pressed the resistance varies along a 40 ohm range. The maximum power current flowing through the resistors would be 5 volts / 40 ohms, which gives 0.125 amps of current. 5 volts * 0.125 amps gives 0.625 watts, which the resistors would need to tolerate. A voltage divider was implemented, with the grip in series with three 1/4 watt 120 ohm resistors in parallel, to form a 40 ohm resistor with ¼ watt tolerance.
The output was connected to ADC channel 0 of the PIC. Eight LEDs were connected to Port B of the PIC.
B. Code:

The code generates a random number, and depending on that number selects a bit to be matched. The code flashes the bit throughout the game, until the grip input matches the flashing light. At this point, the light stops flashing and if an illuminated button is pressed the code indicates victory. Otherwise the code indicates the input is wrong. The ‘generate’ function gets a random number and sets one of eight bits in the ‘rightInput’,
similar to Distance Piano. When the third bit of the ‘Clock’ register is set, the four bit output from the ‘decoder’ function is shifted four bits to the left instead of three.

The code checks the input from the grip in the ‘getInput’ function, which converts the analog data on ADC channel 0 to a digital value, which is stored on register ‘adVal’. A register ‘checkInputs’ is initialized to zero at the beginning of the ‘getInput’ function. The code compares the five most significant bits of ‘adVal’ to decreasing hexadecimal values beginning with 0x80. Before each comparison, a bit is set in the ‘checkInputs’ register. Once there is a match, the code calls the ‘setPort’ function which outputs the value on ‘checkInputs’ to Port B. The ‘setPort’ function is also responsible for making the bit to be matched flash on the outputs. If the bit set in ‘rightInput’ is set in the ‘checkInputs’ register, the bit remains solid and does not flash. The value in ‘checkInputs’ is simply loaded to Port B. If it is not set, the bit in ‘rightInput’ is set and reset for a given amount of time. A ‘toggle’ register holds a value, which is complimented every 255 counts. The ‘toggle’ register determines whether to set the bit in ‘rightInput’ or to reset it.

When an input from the illuminated button, connected to input bit 1 of Port A is set, the code enters a check sequence to see if the value in ‘rightInput’ is set in ‘checkInputs’. If it is, a ‘victory’ is called which displays a pattern of LEDs scrolling up and down on Port B, and sends a victory signal to Master. If not, the ‘wrong’ function is called and a wrong signal is sent to Master. The code returns to the ‘getInput’ function.
6. Noise

A. Circuit Design and Implementation:

A dielectric microphone was used to get noise input. A 1K resistor was connected to 5 volts and to the positive lead of the microphone. The output of the microphone was connected through a 10uf coupling capacitor to the positive terminal of a 741 op-amp, which was set up to have a gain of 100. High values were chosen for the resistors because the current produced from the microphone is very small. +5 volts and -5 volts were used to power the op-amp. As stated previously, sound is a vibration of air waves. Hence, the output is not a constant analog value, but rather an oscillating signal. Since a steady
measurable voltage proportional to sound intensity was desired, a rectifier was used to smooth the output. A diode was connected from the output of the op-amp to the ADC channel of the PIC, and to a 2 nf capacitor connected to ground. The diode truncates the negative voltage, and the capacitor holds the peak value for a brief moment of time. The 2nf capacitor was chosen through experimentation, with different capacitors tested to see which would hold the output for a sufficient amount of time.
B. Code:

Noise

Set up ADC Ports
Start Timer
Fix noise level to be matched

Is Activation bit set?

Yes

Reset 2 second timer

Get Value from ADC channel 0

Mask lower 3 bits of ADC return value

Set ‘checkInput’ to hold a pattern corresponding to value from ADC

Set Port B

Does the Noise level match?

No

Yes

Fell 2 seconds?

No

Yes

Set Victory bit
Display unique pattern on LEDs

The code for noise is identical to that of Grip, but without the ‘wrong’, ‘set bit’, ‘check input’, and ‘generate’ functions. A fixed value is to be matched for a brief amount of time, so no random number is needed to be generated. The only changes made to the code were in the ‘setPort’ function, which now also acts as the check function. Once an input is acquired, it is compared to the correct value. If it matches, the code starts decrementing a counter, ‘duration’ as long as the input matches the correct value. If the input value no longer matches the correct value, the ‘duration’ register is reloaded with the value it began with. Timer interrupts are used, so the ‘duration’ is decremented every
255 counts in the TMR0 register. Once the ‘duration’ register reaches zero, this indicates the player has produced noise at the right level for over the required time, and the ‘victory’ function is called. A victory pattern slightly different from that of Grip is displayed, and a victory bit is sent to Master.
7. Master

A. Circuit Design and Implementation:

The Master module is perhaps the most important amongst all modules. It takes in signals two signals from each input module, and sends out a signal. It also controls every event in the main sculpture: Elevator, Gate, Turnstile, Funnel, and Accelerator. The 16F877A was used for this, since a lot of pins were needed. All 34 I/O pins are used were
this module. Seven switches from the main sculpture indicating the location of the ball were connected to inputs of the PIC, and a total of seven outputs were used to control every single event.

The first event, the elevator required an interface to a bidirectional DC motor circuit which takes two inputs. When one of the inputs is set to zero, the circuit would cause the motor to rotate in one direction, and when the other input is zero, the motor would rotate in the other direction. Two switches were needed to determine the status of the elevator, one on the top and one on the bottom of the elevator shaft. Another switch was needed to tell the PIC if a ball had entered the elevator shaft, and the sculpture needed to begin.

The Gate, Funnel and Accelerator stop the ball with a solenoid. These were referred to as solenoids A, B and C in the code. A switch was installed before each of the solenoids, and the PIC was connected to the gate of a MOSFET that would power the solenoid whenever 5 volts was outputted. Another output was connected to the accelerator module, which turns on a motor that spins in one direction. The turnstile module also required an input connected to a switch on the track before the ball enters it, and an output signal that tells it to activate. When the turnstile module receives the activation signal, it activates a stepper motor that turns for 90 degrees, and turns back 90 degrees.

The Master also takes in and sends signals to input devices through RJ45 jacks. Five RJ45 jacks were mounted, with the following pin out:

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Activate</td>
<td>Ground</td>
<td>Victory</td>
<td>Ground</td>
<td>Wrong</td>
<td>+5V</td>
<td>-5V</td>
<td>+5V</td>
</tr>
</tbody>
</table>
A total of ten inputs, five victory and five wrong bits were interfaced with the PIC, along with five activation bits. The Master module also controls the power supplied to the input stations, since the PIC on the Master needs to be on before any of the other input PICs. An output from the PIC is connected to the gate of an N-Type MOSFET, with the source connected to ground and drain connected to the ground lead of all the input devices. 12 volts and 5 volts would be sent to the input devices, but without a ground to complete the circuit there would be no current flow.

A speaker was also connected to the PIC on Master through an N-Type MOSFET and a 10 ohm resistor. This was needed to give the players audio feedback when either a correct or wrong input is received.

Figure B20: Master Circuit-Front View
B. Code – Pulse Width Modulation (PWM):

Sending full power to the elevator circuit would cause it to shoot up uncontrollably. PWM is a way of varying power to a circuit (in this case a motor) by changing the duty cycle of an input square wave. The functions ‘pwmDown’ and ‘pwmUp’ are set to output a square wave of 2KHz. ‘pwmUp’ is identical to ‘playNote’, but the only difference is that instead of a certain duration, a switch causes it to exit the function. The appropriate value for the period to be loaded onto the TMR2 register frequency was calculated with the formula used for musical notes.

\[ PR2 = \left( \frac{\text{Fosc}}{4 \times 2 \times \text{Fpwm} \times \text{TMR2 Prescale}} \right) - 1 \]

Since the code for ‘playNote’ is used for ‘pwmDown’, the duty cycle is 50%. More power is required to bring the motor shaft up, and hence a higher duty cycle is needed for ‘pwmUp’. The period for TMR2 is set to half of what it was for ‘pwmDown’, and a total of four segments of outputs are displayed. One segment outputting a 0, and three segments outputting 1. This gives a 75% duty cycle.
C. Code – Master:

The Master code begins by properly initializing registers and setting output ports. Musical notes are set just like in the Distance Piano module, along with the ‘playNote’ and ‘wait’ functions. The Master code sets bit 5 on Port A, the pin that controls the power supplied to the input stations, and then enters a function ‘checkInputs’ that checks the switches for the ball’s location within the sculpture. When the ball is acknowledged, the code activates the input connected to the RJ45 jack that corresponds to a specific event.
Nothing is done until the game returns a ‘Victory’ signal, upon which the Master sets bits to cause an event to happen. Whenever a ‘Victory’ signal is received, audio feedback is given. Whenever a ‘Wrong’ signal is received, a simple wrong tone is played.

The first thing the code checks is whether the elevator shaft is down. If not, it brings it down until the lower elevator switch is pressed. This is done in the ‘moveShaftDown’ function, which call the PWM functions described earlier to regulate the speed of the elevator. When the switch before the elevator is activated, the code enters the ‘event1’ state which waits for input 1 to be played and won. The elevator shaft is brought to the top with the ‘moveShaftUp’ function, and then waits for enough time for the ball to roll off onto the track. The elevator shaft then goes back down.

The code enters the ‘event2’ state when the switch before the ‘Gate’ solenoid, solenoid A is set. Once a ‘victory’ signal is received from input 2, a bit is set to activate solenoid A for two seconds. The bit is then cleared, and the solenoid moves back to its rest position. The ‘event3’ state is triggered by a switch before the turnstile, and sets a bit for a second to activate the turnstile module upon victory of input 3. The turnstile module has a built in PIC, with code loaded to control the stepper motor. The turnstile PIC, like the input devices remains in a passive state until an activation bit is set.

States ‘event4’ and ‘event5’ state are identical to ‘event2’. In the ‘event5’ state, however, another bit is set to activate the accelerator module. The accelerator is turned on half a second before solenoid C is activated, to gain enough speed to shoot the ball around a loop. The solenoid and accelerator are left on for two seconds, and are both simultaneously deactivated.
Every time an event occurs, a bit corresponding to the event in progress in register ‘eventCnt’ is set. Once five bits are set in this register, the Master code enters the ‘complete’ function which plays a song to indicate everything has been complete. This song is a tune from the Super Nintendo Entertainment System game, Super Mario World.

**IV. Analysis:**

The major decisions made for this aspect of the project involved in choosing the microcontrollers, the architecture of how small signals were interfaced with the sculpture and a general approach to making everything to work.

One of the drawbacks of the PICs used was that the code needed to be implemented in a low level assembly language. Although this was good for many simple applications such as lighting LEDs and activating other modules, an Object-Oriented PIC may have saved time and coding complexity for the algorithms in the input device games. A simple command in either C or Java would easily perform functions such as waits and musical output, functions that needed to be implemented from scratch in assembly.

The 16 series PICs, however, were cheap and required no special kits or boards that it needed to be interfaced to. A kit on each module could have resulted in expensive and unnecessary spending. The PICs were obtained for free from samples offered by Microchip, and even purchased PICs were no more expensive than $7.00. The benefit of the PICs was its size and availability in a dual inline package, which made it easier for bread board prototyping. Since very fundamental components were used for the entire project, the degrees of freedom for the design of each module, something important when dealing with very fundamental components.
If one microcontroller was to be used to control the logic of everything instead of multiple PICs controlling each individual module, something like the OO-PIC could have made sense. However, the physical implementation of this architecture would be a nightmare. The Distance Piano alone would require 24 wires just for signals between the microcontroller and input module. Of course, buffer chips and decoders could be used to minimize the wires, but at this point another PIC would be a cheaper and easier solution.

V. Conclusion:

Through thorough designing and patient testing, the control logic and small signal elements of the kinetic sculpture were successfully implemented. The decision to build the modules with very fundamental components made the task more challenging, but also made it worthwhile. Much was learned over the past year through constant trial and error which could never be acquired from text books. Working on a project with a team and applying engineering knowledge for problem solving was a great experience that everyone should have at least once in their academic career.
VI. Bibliography:

1. Microchip’s main website: www.microchip.com
   a. 16F84A data sheet:
      nodeId=1999&dty=Data+Sheets&ssUserText=PIC16F84A
   b. 16F877A data sheet:
      nodeId=1999&dty=Data+Sheets&ssUserText=PIC16F877A
   c. 16F88 data sheet:
      nodeId=1999&dty=Data+Sheets&ssUserText=PIC16F88

2. Analog to Digital Conversion sample code:
   http://www.piclist.com/techref/microchip/16f877adsamp.htm

3. Pulse Width Modulation set up instructions:
   http://www.piclist.com/techref/microchip/16F877/pwm.htm

4. Digikey website: www.digikey.com

5. N-type Mosfet characteristics:
   http://www.fairchildsemi.com/ds/ND/NDP603AL.pdf

VII. Appendix:
Master PIC Code

;####################################################################
; Robotics Academy - Kinetic Sculpture (2003-2004)
; Jason Adrian & Marc Weintraub
; Date Finalized: 4/10/04
;
; Target PIC: 16F877A
; Program Name: MasterPIC
; Purpose: Control electronic components of sculpture
; Description: Interfaces with everything
;
;####################################################################

; bit mapping:
; ***INPUTS***
; -switches-
; B0 = elevator lower switch
; B1 = elevator upper switch
; B2 = switchA (b4 solenoidA)
; B3 = switchT (b4 turnstile)
; -event/input station bits
; B4 = switchB (b4 solenoidB)
; B5 = switchC (b4 solenoidC)
; B6 = input1 victory
; B7 = input1 wrong
;
; C0 = input2 victory
; C1 = input2 wrong
; C2 = input3 victory
; C3 = input3 wrong
; C4 = input4 victory
; C5 = input4 wrong
; C6 = input5 victory
; C7 = input5 wrong
;
; E2 = input1 start: sculpture start
;
; ***OUTPUTS***
; A0 = input1 activate
; A1 = input2 activate
; A2 = input3 activate
; A3 = input4 activate
; A4 = not used
; A5 = supply input stations with power
;
; D0 = elevator DOWN
; D1 = elevator UP
; D2 = solenoidA ACTIVATE
; D3 = turnstile ACTIVATE
; D4 = solenoidB ACTIVATE
; D5 = solenoidC ACTIVATE
; D6 = accelerator ACTIVATE
; D7 = speaker OUTPUT
;
; E0 = input5 activate
; E1 = big blue light ACTIVATE

LIST p=16F877A
include "P16F877A.inc"

;notes
Bbass EQU 0x4D
Cbass EQU 0x48
Dbass EQU 0x49
Ebass EQU 0x4A
Fbass EQU 0x4B
Gbass EQU 0x4C
Abass EQU 0x40
Btreb EQU 0x41
Ctreb EQU 0x42
Dtreb EQU 0x43
Etreb EQU 0x44
Ftreb EQU 0x45
Gtreb EQU 0x46

duration EQU 0x52
oneSec EQU 0x53
check EQU 0x56
eventCnt EQU 0x57
tempNote EQU 0x58
count0 EQU 0x4E
count1 EQU 0x4F
half EQU 0x60
quart EQU 0x61
eighth EQU 0x62
A#treb EQU 0x63
G#treb EQU 0x64
A#bass EQU 0x65
G#bass EQU 0x66
halfR EQU 0x67
quartR EQU 0x68
eighthR EQU 0x69

setup
  org 0x000
  ; set values to notes
  ; set base clef
  movlw d'252'
  movwf Bbass
movlw d'238'
movwf Cbass
movlw d'212'
movwf Dbass
movlw d'189'
movwf Ebass
movlw d'178'
movwf Fbass
movlw d'159'
movwf Gbass
movlw d'150'
movwf G#bass

; set treble clef
movlw d'141'
movwf Atreb
movlw d'134'
movwf A#treb
movlw d'126'
movwf Btreb
movlw d'119'
movwf Ctreb
movlw d'105'
movwf Dtreb
movlw d'94'
movwf Etreb
movlw d'88'
movwf Etreb
movlw d'79'
movwf Gtreb
movlw d'75'
movwf G#treb

; load one sec
movlw d'121'
movwf oneSec

movlw d'61'
movwf half
movlw d'31'
movwf quart
movlw d'16'
movwf eighth

movlw d'69'
movwf halfR
movlw d'39'
movwf quartR
movlw d'24'
movwf eighthR

; set IO ports
bsf STATUS,RP0 ;bank 1
bcf STATUS,RP1

; input ports
movlw 0xFF
movwf TRISB  ;portb inputs

movwf TRISC  ;portc inputs
bsf TRISE,2
 ; output ports

clrf TRISA  ;porta
clrf TRISD  ;portd
bcf TRISE,0
bcf TRISE,1
 ; disable a2d channels

movlw 0x07
movwf ADCON1

; set timer0
movlw 0x84
movwf OPTION_REG
bcf STATUS,RP0  ;bank 0

; set timer2
movlw 0x05
movwf T2CON

; main function

main
 ; set initial output conditions
bcf PORTE,1
clrf PORTA
movlw 0x03
movwf PORTD
clrf eventCnt
clrf check

movlw d'30'
movwf duration
call wait

; supply power to inputStations
bsf PORTA,5

; check the inputs from the scupture to get the ball status.
; once the input is received, the program should jump to the
; appropriate state to handle the event the ball will enter
; in the scupture

checkInputs

; --sculpture inputs--

btfss PORTB,0  ; bring elevator shaft down
        call moveShaftDown

btfsc PORTE,2  ; goto event1: start elevator
        goto event1
btfsc PORTB,2 ; goto event2
  goto event2
btfsc PORTB,3 ; goto event3
  goto event3
btfsc PORTB,4 ; goto event4
  goto event4
btfsc PORTB,5 ; goto event5
  goto event5
movlw 0x1F ; check to see if all events occurred
  xorwf eventCnt,0
  movwf check
  movf check,1
  btfsc STATUS,Z ; if so, go to the complete sequence
  goto complete

; move the elevator down until switch B0 is hit
moveShaftDown
  bcf PORTD,0
  movlw d'10'
  movwf duration
  call wait
  call pwmDown
  return

; move the elevator up until switch B1 is hit
moveShaftUp
  bcf PORTD,1
  movlw d'10'
  movwf duration
  call wait
  call pwmUp
  return

; wait for input, move elevator to top, then bring it back down
event1
  bsf PORTA,0 ; activate input1
  ; if wrong, wait till kid gets it right
  btfsc PORTB,7
  call wrongTone
  btfsc PORTB,6
  goto event1
  call correctTone
call moveShaftUp
movf oneSec,0
movwf duration
call wait
movf oneSec,0
movwf duration
call wait
call moveShaftDown

bcf PORTA,0 ; deactivate input1
bsf eventCnt,0 ; set event checker
movf oneSec,0
movwf duration
call wait
goto checkInputs

; wait for input, activate solenoidA

event2
bsf PORTA,1 ; activate input2
btfsc PORTC,1
call wrongTone
btfss PORTC,0
goto event2
call correctTone
bsf PORTD,2 ; turn on solenoidA
movf oneSec,0
movwf duration
call wait
movf oneSec,0
movwf duration
call wait
bcf PORTD,2 ; turn on solenoidA
bcf PORTA,1 ; deactivate input2
bsf eventCnt,1 ; set event checker
movf oneSec,0
movwf duration
call wait
goto checkInputs

; wait for input, activate turnstile

event3
bsf PORTA,2 ; activate input3
btfsc PORTC,3
call wrongTone
btfss PORTC,2
  goto event3
  
call correctTone

  bsf PORTD,3  ; activate turnstile
  movf oneSec,0
  movwf duration
  call wait
  bcf PORTD,3  ; shut off turnstile
  bcf PORTA,2  ; deactivate input3
  bsf eventCnt,2 ; set event checker

  movf oneSec,0
  movwf duration
  call wait

  goto checkInputs

; wait for input, activate solenoidB
  event4
  bsf PORTA,3  ; activate input4
  btfsc PORTC,5
  call wrongTone
  btfss PORTC,4
  goto event4
  
call correctTone

  bsf PORTD,4  ; turn on solenoidB
  movf oneSec,0
  movwf duration
  call wait
  movf oneSec,0
  movwf duration
  call wait
  bcf PORTD,4  ; turn off solenoidB
  bcf PORTA,3  ; deactivate input4
  bsf eventCnt,3 ; set event checker

  movf oneSec,3
  movwf duration
  call wait

  goto checkInputs

; wait for input, activate solenoidC
  event5
  bsf PORTE,0  ; activate input5
  btfsc PORTC,7
  call wrongTone
btfss PORTC,6
goto event5
call correctTone

bsf PORTD,6 ; turn on accelerator
movlw d'61'
movwf duration
call wait
bsf PORTD,5 ; turn on solenoidC
movlw d'30'
movwf duration
call wait
movf oneSec,0
movwf duration
call wait
bcf PORTD,5 ; turn off solenoidC
bcf PORTD,6 ; turn off accelerator
bcf PORTE,0 ; deactivate input5
bsf eventCnt,4 ; set event checker
movf oneSec,0
movwf duration
call wait
goto checkInputs

; play victory sequence, turn on big blue light
complete
movf oneSec,0
movwf duration
call wait
bsf PORTE,1 ; turn on big blue light
call victorySong ; play victory song
bcf PORTE,1
goto main

; victory song
victorySong

movf eighthR,0
movwf duration
movf Gbass,0
movwf tempNote
call playNote
movf eighthR,0
movwf duration
call wait
movf eighthR,0
movwf duration
movf Gbass,0
movwf tempNote
call playNote

movf eighthR,0
movwf duration
call wait

movf eighthR,0
duration
movf Ebass,0
movwf tempNote
call playNote

movlw d'8'
duration
call wait

movf eighth,0
duration
movf Gbass,0
tempNote
call playNote

movf eighthR,0
duration
call wait

movf eighth,0
duration
movf Ebass,0
tempNote
call playNote

movlw d'8'
duration
call wait

movf eighth,0
duration
movf Gbass,0
tempNote
call playNote

movlw d'8'
duration
call wait

movf eighth,0
duration
movf Ebass,0
tempNote
call playNote

movlw d'8'
duration
call wait
movlw d'8'
movwf duration
call wait

movf eighth,0
movwf duration
movf Dbass,0
movwf tempNote
call playNote

movlw d'8'
movwf duration
call wait

movf eighth,0
movwf duration
movf Gbass,0
movwf tempNote
call playNote

movf quartR,0
movwf duration
call wait

movf eighth,0
movwf duration
movf Ebass,0
movwf tempNote
call playNote

movlw d'8'
movwf duration
call wait

movf eighth,0
movwf duration
movf Dtreb,0
movwf tempNote
call playNote

movlw d'8'
movwf duration
call wait

movf eighth,0
movwf duration
movf Etreb,0
movwf tempNote
call playNote

movlw d'8'
movwf duration
call wait
movf eighth,0
movwf duration
movf Dtreb,0
movwf tempNote
call playNote

movlw d'8'
movwf duration
call wait

movf eighth,0
movwf duration
movf Etreb,0
movwf tempNote
call playNote

movlw d'8'
movwf duration
call wait

movf eighth,0
movwf duration
movf Dtreb,0
movwf tempNote
call playNote

movlw d'16'
movwf duration
call wait

movlw d'10'
movwf duration
movf Gbass,0
movwf tempNote
call playNote

movlw d'2'
movwf duration
call wait

movlw d'10'
movwf duration
movf Btreb,0
movwf tempNote
call playNote

movlw d'2'
movwf duration
call wait

movlw d'10'
movwf duration
movf Atreb,0
movwf tempNote
call playNote
movlw d'4'
movwf duration
call wait
movf oneSec,0
movwf duration
movf Gbass,0
movwf tempNote
call playNote
movlw d'4'
movwf duration
call wait
movlw d'4'
movwf duration
movf Gbass,0
movwf tempNote
call playNote
movlw d'4'
movwf duration
movf Atreb,0
movwf tempNote
call playNote
movlw d'4'
movwf duration
movf Btreb,0
movwf tempNote
call playNote
movlw d'4'
movwf duration
movf Ctreb,0
movwf tempNote
call playNote
movlw d'4'
movwf duration
movf Dtreb,0
movwf tempNote
call playNote
movlw d'4'
movwf duration
movf Etreb,0
movwf tempNote
call playNote
movlw d'4'
movwf duration
movf Ftreb,0
movwf tempNote
call playNote
movlw d'161'
movwf duration
movf Gtreb,0
movwf tempNote
call playNote

return

; tone for correct input
correctTone

movf eighthR,0
movwf duration
movf Btreb,0
movwf tempNote
call playNote

movf half,0
movwf duration
movf Gbass,0
movwf tempNote
call playNote

return

; tone for wrong input
wrongTone

movf eighthR,0
movwf duration
movf Bbass,0
movwf tempNote
call playNote

movlw d'8'
movwf duration
call wait

movf oneSec,0
movwf duration
movf Bbass,0
movwf tempNote
call playNote

movlw d'161'
movwf duration
call wait

return

playNote ; produce square wave of certain frequency

movf tempNote,0
bsf STATUS,RP0
movwf PR2
bcf STATUS,RP0
bsf PORTD,7
bcf INTCON,TMR0IF ; clear interrupt bit

out0 ; check interrupt flags

btfss PIR1,TMR2IF
goto    out0
bcf    PIR1,TMR2IF
bcf    PORTD,7
out1    ; check interrupt flags
btfss  PIR1,TMR2IF
    goto  out1
bcf    PIR1,TMR2IF
bsf    PORTD,7
btfss  INTCON,TMR0IF    ; skip if interrupt is not enabled
    goto  out0
bcf    INTCON,TMR0IF    ; clear interrupt bit
decfsz duration
    goto  out0
return

pwmUp    ; 75% duty cycle square wave at 2KHz
movlw    d'31'
bssf    STATUS,RP0
movwf    PR2
bcf    STATUS,RP0
bsf    PORTD,1
out0u0    ; check interrupt flags
btfss  PIR1,TMR2IF
    goto  out0u0
bcf    PIR1,TMR2IF
bcf    PORTD,1
out1u0    ; check interrupt flags
btfss  PIR1,TMR2IF
    goto  out1u0
out1u1    ; check interrupt flags
btfss  PIR1,TMR2IF
    goto  out1u1
out1u2    ; check interrupt flags
btfss  PIR1,TMR2IF
    goto  out1u2
bcf    PIR1,TMR2IF
bsf    PORTD,1
btfss  PORTB,1    ; skip if switch is not pressed
    goto  out0u0
return

pwmDown    ; 50% duty cycle square wave at 2KHz
movlw    d'63'
bssf    STATUS,RP0
movwf    PR2
bcf STATUS,RP0
bsf PORTD,0

; check interrupt flags
btfss PIR1,TMR2IF
goto out0d

bcf PIR1,TMR2IF
bcf PORTD,0

; check interrupt flags
btfss PIR1,TMR2IF
goto out1d

bcf PIR1,TMR2IF
bsf PORTD,0

btfss PORTB,0 ; skip if switch is not pressed
goto out0d
return

; wait for a given duration of time
wait ; produce square wave of certain frequency
bcf STATUS,RP0
bcf INTCON,TMR0IF ; clear interrupt bit

wait4int
btfss INTCON,TMR0IF ; skip if interrupt is not enabled
goto wait4int
bcf INTCON,TMR0IF ; clear interrupt bit
decfsz duration
goto wait4int
return

endz
end
Simon Code

; Robotics Academy - Kinetic Sculpture (2003-2004)
; Jason Adrian & Marc Weintraub
; Date Finalized: 4/10/04
; Target PIC: 16F84A
; Program Name: Simon
; Purpose: Input device
; Description: Simon game. A pattern of 4 lights
; are displayed, which need to be matched to win.
;

LIST p=16F84A
include "P16F84a.inc"
_CONFIG_0x3d18
org 0x000
DATAONE EQU 0x16
DATATWO EQU 0x17
DATATHREE EQU 0x18
DATAFOUR EQU 0x19
INPUT EQU 0x1c
clock EQU 0x1d
count EQU 0x21
select EQU 0x22
decoded EQU 0x23
waitCnt EQU 0x24
win EQU 0x25
PREVINPUT EQU 0x26

;SET UP INPUT and OUTPUT PORTS
bsf STATUS, RP0 ;go to bank b
; set A to input
movlw 0x0F
movwf TRISA
; set B to input and output
movlw 0x30
movwf TRISA
; enable timer
movlw 0x81
movwf OPTION_REG
bcf STATUS, RP0 ;go to bank a

clr PORTB

passive
movf PORTB,0
movwf INPUT
btfss INPUT,5 ;check for activation bit
goto passive

goto generate
; wait until activation bit is set

main
movf PORTB,0
movwf INPUT
btfss INPUT,4 ;check for reset bit
goto main

;-----generate data to be outputted and matched
;according to timer0

generate
; set variable to data 1

movf TMR0, 0
movwf clock
movwf select
call decode
movf decoded, 0
movwf DATAONE

; set variable to data 2
rrf clock
rrf clock
movf clock,0
movwf select
call decode
movf decoded, 0
movwf DATATWO

; set variable to data 3
rrf clock
rrf clock
movf clock,0
movwf select
call decode
movf decoded, 0
movwf DATATHREE

; set variable to data 4
rrf clock
rrf clock
movf clock,0
movwf select
call decode
movf decoded, 0
movwf DATAFOUR

;-----display the pattern onto the buttons
displayPattern

call    wait
movf    DATAONE, 0
movwf    PORTB

call    wait

clrf    PORTB
call    wait
movf    DATATWO, 0
movwf    PORTB

call    wait

clrf    PORTB
call    wait
movf    DATATHREE, 0
movwf    PORTB
call    wait

clrf    PORTB
call    wait
movf    DATAFOUR, 0
movwf    PORTB
call    wait

clrf    PORTB

; check input to see if it matches DATAONE
inputcheck1
movf    PORTA, 0
movwf    INPUT

xorwf    DATAONE, 1
btfs    STATUS, Z
goto    inputcheck2
movf    INPUT, 1
btfss    STATUS, Z
goto   gameover

goto    inputcheck1

; do not do anything as long as button is pressed
inputcheck2

call    wait2
movf    INPUT, 0
movwf    PREVINPUT
movwf    PORTB
movf    PORTA, 0
movwf    INPUT
xorwf    PREVINPUT, 1
btfs    STATUS, Z
goto    inputcheck2

clrf    PORTB

; check input to see if it matches DATATWO
inputcheck2a
    movf PORTA, 0
    movwf INPUT

    xorwf DATATWO,1
    btfsc STATUS, Z
    goto inputcheck3
    movf INPUT, 1
    btfss STATUS, Z
    goto gameover
    goto inputcheck2a

; do not do anything as long as button is pressed
inputcheck3
    call wait2
    movf INPUT,0
    movwf PREVINPUT
    movwf PORTB
    movf PORTA, 0
    movwf INPUT
    xorwf PREVINPUT,1
    btfsc STATUS, Z
    goto inputcheck3
    clrf PORTB

; check input to see if it matches DATATHREE
inputcheck3a
    movf PORTA, 0
    movwf INPUT

    xorwf DATATHREE,1
    btfsc STATUS, Z
    goto inputcheck4
    movf INPUT, 1
    btfss STATUS, Z
    goto gameover
    goto inputcheck3a

; do not do anything as long as button is pressed
inputcheck4
    call wait2
    movf INPUT,0
    movwf PREVINPUT
    movwf PORTB
    movf PORTA, 0
    movwf INPUT
    xorwf PREVINPUT,1
    btfsc STATUS, Z
    goto inputcheck4
    clrf PORTB
; check input to see if it matches DATAFOUR
inputcheck
  movf PORTA, 0
  movwf INPUT

  xorwf DATAFOUR, 1
  btfsc STATUS, Z
  goto victory
  movf INPUT, 1
  btfss STATUS, Z
  goto gameover

  goto inputcheck4a

; do not do anything as long as button is pressed
inputchecklast

  call waitC
  movf INPUT, 0
  movwf PREVINPUT
  movwf PORTB
  movf PORTA, 0
  movwf PORTB
  movf INPUT
  xorwf PREVINPUT, 1
  btfsc STATUS, Z
  goto inputchecklast

  clrf PORTB
  call wait2
  call wait2

  movlw 0x03
  movwf win

;-----display a pattern on the buttons to indicate victory
winPattern

  movlw 0x01
  movwf PORTB
  call wait2

  movlw 0x02
  movwf PORTB
  call wait2
movlw 0x08
movwf PORTB
call wait2

movlw 0x04
movwf PORTB
call wait2
decfsz win,1
goto winPattern

; activate victory bit
clrf PORTB
bsf PORTB, 7
call wait
call wait
clrf PORTB
goto main

; ----- decoder function. 2-4 bit decoder
decode
clrf decoded
btfsc select, 1
goto MSB1
goto MSB0

MSB1
btfsc select, 0
bsf decoded, 3
btfss select, 0
bsf decoded, 2
return

MSB0
btfsc select, 0
bsf decoded, 1
btfss select, 0
bsf decoded, 0
return

; ----- delay generator
wait
movlw 0xFF ; load count1 with decimal 100
movwf waitCnt
d1a
movlw 0xFF ; load count2 with decimal 100
movwf count
d2a
NOP
NOP
NOP
NOP
nop
decfsz count,1
goto d2a
decfsz waitCnt,1
goto d1a
return
;-----delay generator with less delay time

wait2   
    movlw 0x77
    movwf waitCnt

  d1b
    movlw 0x77
    movwf count

  d2b
    NOP
    NOP
    NOP
    NOP
    decfsz count,1
    goto d2b
    decfsz waitCnt,1
    goto d1b
    return

;-----delay generator with even less delay time

waitC   
    movlw 0x10
    movwf waitCnt

  d1c
    movlw 0x33
    movwf count

  d2c
    decfsz count,1
    goto d2c
    decfsz waitCnt,1
    goto d1c
    return

endz
    end
Distance Piano Code

; Robotics Academy - Kinetic Sculpture (2003-2004)
; Jason Adrian & Marc Weintraub
; Date Finalized: 4/10/04
; Target PIC: 16F877A
; Program Name: Distance Piano
; Purpose: Input device
; Description: A game where the player must match
; a given note by putting their hand a certain
; distance away from the sensor. There will be three
; three notes.
;
; PORTB & 4msb of PORTD are outputs to the notes.
; C7-C5 are count outputs
; C4 is square wave output
; D3, D2 are Victory and Wrong outputs respectively
; A0 is Analog input
; A1 is Digital input
; A2 is activation bit

LIST p=16F877A
include "P16F877A.inc"

; notes
; declare registers

clock EQU 0x21
select EQU 0x22
decoded EQU 0x23

Atreb EQU 0x40
Btreb EQU 0x41
Ctreb EQU 0x42
Dtreb EQU 0x43
Etreb EQU 0x44
Ftreb EQU 0x45
Gtreb EQU 0x46

adVal EQU 0x47
Bbass EQU 0x39
Cbass EQU 0x48
Dbass EQU 0x49
Ebass EQU 0x4A
Fbass EQU 0x4B
Gbass EQU 0x4C

tempNote EQU 0x4D
count0 EQU  0x4E
count1 EQU  0x4F
count EQU  0x50
waitCnt EQU  0x51
duration EQU  0x52
oneSec EQU  0x53
noteReg EQU  0x54
winStat EQU  0x55
check EQU  0x56
setLight EQU  0x57
toggle EQU  0x58
half EQU  0x60
quart EQU  0x61
eighth EQU  0x62
A#treb EQU  0x63
G#treb EQU  0x64
A#bass EQU  0x65
G#bass EQU  0x66
halfR EQU  0x67
quartR EQU  0x68
eighthR EQU  0x69
thisNote EQU  0x6A
lightReg EQU  0x6B

setup
org 0x000
;  clrf  winStat
;  goto  checkInput

; set values to notes
;  set base clef
movlw d'268'
movwf A#bass
movlw d'252'
movwf Bbass
movlw d'238'
movwf Cbass
movlw d'212'
movwf Dbass
movlw d'189'
movwf Ebass
movlw d'178'
movwf Fbass
movlw d'159'
movwf Gbass
movlw d'150'
movwf G#bass

;  set treble clef
movlw d'141'
movwf Atreb
movlw d'134'
movwf A#treb
movlw d'126'
movwf Btreb
movlw d'119'
movwf Ctreb
movlw d'105'
movwf Dtreb
movlw d'94'
movwf Etreb
movlw d'88'
movwf Ftreb
movlw d'79'
movwf Gtreb
movlw d'75'
movwf G#treb

; load one sec
movlw d'121'
movwf oneSec

movlw d'61'
movwf half
movlw d'31'
movwf quart
movlw d'16'
movwf eighth

movlw d'69'
movwf halfR
movlw d'39'
movwf quartR
movlw d'24'
movwf eighthR

; set IO ports
bsf STATUS,RP0 ;bank 1
bcf STATUS,RP1

movlw 0x07
movwf TRISA ;2 inputs on porta
clrf TRIB ;portb outputs
clrf TRISC ;portc outputs
clrf TRISD ;portd outputs

; set timer2, timer0 and a2d
movlw 0x4E
movwf ADCON1
movlw 0x84
movwf OPTION_REG
bcf STATUS,RP0 ;bank 0
movlw 0x41 ;Fosc/4 A/D ch0
movwf ADCON0
movlw 0x05
movwf T2CON
; main function. clear all ports and enter passive state
main
    clrf PORTB ; output LED to notes
    clrf PORTC ; C0 = 9th note, C1,C2 = treble, bass
                ; C5,C6,C7 = win status
    clrf PORTD
    clrf winStat
    clrf noteReg
    clrf toggle

; nothing should happen until the game is activated
passive
    btfss PORTA,2 ; check for activation bit
    goto passive

; generate a value according to the time
; the current time would determine which note
; and whether the note is base or treble

generate
    movlw 0xE0
    andwf PORTC,1
    movf TMR2, 0
    movwf clock
    movwf select
    call decode
    movf decoded, 0
    btfsc clock, 3
    call shiftThree
    movf decoded,0
    movwf noteReg

; treble or base
    btfsc clock, 4
    bsf PORTC,1
    btfss clock, 4
    bsf PORTC,2

; display the note on the LEDs and play the note, on the speaker
; this tells the player which note he/she should match

displayAndPlay
    btfsc PORTC,1
    call setTreb
    movf noteReg,0
    movwf setLight
    movwf lightReg
    btfsc PORTC,1
    call getNoteTreb
    btfsc PORTC,2
    call getNoteBass

    movlw d'255'
movwf    duration
movf    noteReg,0
movwf    tempNote
call    playNote

clr    PORTB
bcf    PORTC,0
clr    thisNote

goto pianoState

pianoState

bcf PORTA,3

; do a2d
call a2d
movlw 0xF8   ; mask bits
andwf adVal   ; least significant 3 bits become zero

movlw 0x05
movwf duration

; if ye high play ye note
; set the light
movlw 0x40
btfss toggle,0
movlw 0x00
movwf setLight
; set the note
movf Atreb,0
btfss toggle,0
movf Gtreb,0
movwf tempNote
movlw 0x88
xorwf adVal,0
movwf check
btfsc STATUS, Z
call playNote

; if ye high play ye note
movlw 0x40
btfss toggle,0
movlw 0x00
movwf setLight
movf Atreb,0
btfss toggle,0
movf Gtreb,0
movwf tempNote
movlw 0x80
xorwf adVal,0
movwf check
btfsc STATUS, Z
call playNote

; if ye high play ye note
movlw 0x40
btfss toggle,0
movlw 0x00
movwf setLight
movf Atreb,0
btfss toggle,0
movf Gtreb,0
movwf tempNote
movlw 0x78
xorwf adVal,0
movwf check
btfsc STATUS, Z
call playNote

; if ye high play ye note
movlw 0x20
btfss toggle,0
movlw 0x80
movwf setLight
movf Gbass,0
btfss toggle,0
movf Ftreb,0
movwf tempNote
movlw 0x70
xorwf adVal,0
movwf check
btfsc STATUS, Z
call playNote

; if ye high play ye note
movlw 0x20
btfss toggle,0
movlw 0x80
movwf setLight
movf Gbass,0
btfss toggle,0
movf Ftreb,0
movwf tempNote
movlw 0x68
xorwf adVal,0
movwf check
btfsc STATUS, Z
call playNote

; if ye high play ye note
movlw 0x20
btfss toggle,0
movlw 0x80
movwf setLight
movf Gbass,0
btfss toggle,0
movf Ftreb,0
movwf tempNote
movlw 0x60
xorwf adVal,0
movwf check
btfsc STATUS, Z
call playNote

; if ye high play ye note
movlw 0x10
btfs toggle,0
movlw 0x40
movwf setLight
movf Fbass,0
btfs toggle,0
movf Etreb,0
movwf tempNote
movlw 0x58
xorwf adVal,0
movwf check
btfsc STATUS, Z
call playNote

; if ye high play ye note
movlw 0x10
btfs toggle,0
movlw 0x40
movwf setLight
movf Fbass,0
btfs toggle,0
movf Etreb,0
movwf tempNote
movlw 0x50
xorwf adVal,0
movwf check
btfsc STATUS, Z
call playNote

; if ye high play ye note
movlw 0x08
btfs toggle,0
movlw 0x20
movwf setLight
movf Ebass,0
btfs toggle,0
movf Dtreb,0
movwf tempNote
movlw 0x48
xorwf adVal,0
movwf check
btfsc STATUS, Z
call playNote

; if ye high play ye note
movlw 0x08
btfs toggle,0
movlw  0x20
movwf  setLight
movf   Ebass,0
btfs   toggle,0
movf   Dtreb,0
movwf  tempNote
movlw  0x40
xorwf  adVal,0
movwf  check
btfsc  STATUS, Z
    call      playNote

; if ye high play ye note
movlw  0x04
btfs   toggle,0
movlw  0x10
movwf  setLight
movf   Dbass,0
btfs   toggle,0
movf   Ctreb,0
movwf  tempNote
movlw  0x38
xorwf  adVal,0
movwf  check
btfsc  STATUS, Z
    call      playNote

; if ye high play ye note
movlw  0x02
btfs   toggle,0
movlw  0x08
movwf  setLight
movf   Cbass,0
btfs   toggle,0
movf   Btreb,0
movwf  tempNote
movlw  0x30
xorwf  adVal,0
movwf  check
btfsc  STATUS, Z
    call      playNote

; if ye high play ye note
movlw  0x01
btfs   toggle,0
movlw  0x04
movwf  setLight
movf   Bbass,0
btfs   toggle,0
movf   Atreb,0
movwf  tempNote
movlw  0x28
xorwf  adVal,0
movwf  check
btfsc  STATUS, Z
    call      playNote
; if ye high play ye note
movlw 0x01
btfss toggle,0
movlw 0x04
movwf setLight
movf Bbass,0
btfss toggle,0
movf Atreb,0
movwf tempNote
movlw 0x20
xorwf adVal,0
movwf check
btfsc STATUS, Z
    call playNote
clrf PORTB
btfsc PORTA,1
    goto checkInput
    goto pianoState

checkInput
    bsf PORTA,3
    movf thisNote,0
    xorwf noteReg,0
    movwf check
    btfss STATUS, Z
    goto wrong
    call oneWin
    incf winStat,1
    movlw 0x01
    xorwf winStat,0
    movwf check
    btfsc STATUS, Z
    bsf PORTC,5
    movlw 0x02
    xorwf winStat,0
    movwf check
    btfsc STATUS, Z
    bsf PORTC,6
    movlw 0x03
    xorwf winStat,0
    movwf check
    btfsc STATUS, Z
    goto victory
    clrf PORTB
    clrf noteReg
    clrf lightReg
movlw d'121'
 movwf duration
 call rest

goto generate

bsf PORTC,3

movlw d'8'
 movwf duration
 call rest

movlw d'8'
 movwf duration
 movf Gbass,0
 movwf tempNote
 call playNote

movlw d'8'
 movwf duration
 movf Atreb,0
 movwf tempNote
 call playNote

movlw d'8'
 movwf duration
 movf Btreb,0
 movwf tempNote
 call playNote

movlw d'8'
 movwf duration
 movf Ctreb,0
 movwf tempNote
 call playNote

movlw d'8'
 movwf duration
 movf Dtreb,0
 movwf tempNote
 call playNote

movlw d'8'
 movwf duration
 movf Etreb,0
 movwf tempNote
 call playNote

movlw d'8'
 movwf duration
 movf Ftreb,0
 movwf tempNote
 call playNote

movlw d'8'
 movwf duration
movf Gtreb,0
movwf tempNote
call playNote

bcf PORTA,3

movlw d'31'
movwf duration
call rest

bcf PORTC,3

return

wrong

movlw d'20'
movwf duration
call rest
bcf PORTA,3
bsf PORTD,2
movlw d'121'
movwf duration
call rest
bcf PORTD,2
movf lightReg,0
movwf noteReg
goto displayAndPlay

victory

bcf PORTA,3
movlw d'10'
movwf duration
call rest
bcf PORTA,3
bsf PORTC,7

movf eighth,0
movwf duration
movf Ctreb,0
movf tempNote
call playNote

movlw d'8'
movwf duration
call rest

movf eighth,0
movwf duration
movf Ctreb,0
movf tempNote
call playNote

movlw d'8'
movwf duration
call rest
movf eighth,0
duration
movf Ctreb,0
tempNote
call playNote

movlw d'8'
duration
movwf tempNote
call rest

movf quart,0
duration
movf Ctreb,0
tempNote
call playNote

movf eighthR,0
duration
movwf tempNote
call rest

movf quart,0
duration
movf G#bass,0
tempNote
call playNote

movf eighthR,0
duration
movwf tempNote
call rest

movf quart,0
duration
movf A#treb,0
tempNote
call playNote

movf eighthR,0
duration
movwf tempNote
call rest

movf eighth,0
duration
movf Ctreb,0
tempNote
call playNote

movf eighthR,0
duration
movwf tempNote
call rest

movf eighth,0
duration
movf A#treb,0
tempNote
call playNote
movf half,0
movwf duration
movf Ctreb,0
movwf tempNote
call playNote
bsf PORTD,3
movlw d'121'
movwf duration
bcf PORTD,3
call rest
clr PORTB
clr PORTC
goto passive

;function to set the notes to the appropriate value
;if treble is selected in the generate function,
;the treble scale will be loaded

getNoteTreb
btsc noteReg,2
movf Atreb,0
btsc noteReg,3
movf Btreb,0
btsc noteReg,4
movf Ctreb,0
btsc noteReg,5
movf Dtreb,0
btsc noteReg,6
movf Etreb,0
btsc noteReg,7
movf Ftreb,0
movf noteReg,1
btsc STATUS,Z
movf Gtreb,0
movwf noteReg
cf toggle,0
return

;same for bass
getNoteBass
btsc noteReg,0
movf Bbass,0
btsc noteReg,1
movf Cbass,0
btsc noteReg,2
movf Dbass,0
btsc noteReg,3
movf Ebass,0
btsc noteReg,4
movf Fbass,0
btsc noteReg,5
movf Gbass,0
btsc noteReg,6
movf Atreb,0
movwf noteReg
bsf toggle,0
return

;function to set the note to treble.
;this is needed for the LED lights

setTreb
  btfsc noteReg,6
  clrf noteReg
  rlf noteReg,1
  rlf noteReg,1
  return

;function to shift the decoded register 3 bits
shiftThree
  bcf STATUS,C
  rlf decoded,1
  rlf decoded,1
  rlf decoded,1
  return

;function to decode timer register. This takes a binary 2 bit value and decodes
;it to 4 bits.
decode
  clrf decoded
  btfsc select, 1
  goto MSB1
  goto MSB0

MSB1
  btfsc select, 0
  bsf decoded, 3
  btfss select, 0
  bsf decoded, 2
  return

MSB0
  btfsc select, 0
  bsf decoded, 1
  btfss select, 0
  bsf decoded, 0
  return

;rest function. produces a delay of a certain amount of time
rest
  ; delay
  bcf STATUS,RP0
  bcf INTCON,TMR0IF ; clear interrupt bit
wait4int
  btfss INTCON,TMR0IF ; skip if interrupt is not enabled
  goto wait4int
  bcf INTCON,TMR0IF ; clear interrupt bit
decfsz duration
  goto wait4int
  return

;play note function. plays the given note
playNote ; produce square wave of certain frequency
    movf setLight,1
    btfsc STATUS,Z
    bsf PORTC,0
    btfss STATUS,Z
    bcf PORTC,0
    movf setLight,0
    movwf PORTB
    movf tempNote,0
    movwf thisNote
    bsf STATUS,RP0
    movwf PR2
    bcf STATUS,RP0
    bcf PORTC,4
    bcf INTCON,TMR0IF ; clear interrupt bit
    ;output 0
    out0 ; check interrupt flags
    btfss PIR1,TMR2IF
    goto out0
    bcf PIR1,TMR2IF
    bcf PORTC,4
    ;output 1
    out1 ; check interrupt flags
    btfss PIR1,TMR2IF
    goto out1
    bcf PIR1,TMR2IF
    bsf PORTC,4
    btfss INTCON,TMR0IF ; skip if interrupt is not enabled
    goto out0
    bcf INTCON,TMR0IF ; clear interrupt bit
    decfsz duration
    goto out0
    clrf PORTB
    bcf PORTC,0
    return

;analog to digital conversion
a2d
    bsf ADCON0,GO
newConv
    btfsc ADCON0,GO ;Wait for conversion to complete
    goto newConv
    movf ADRESH,W ; set adVal to 8 MSB of 10bit AD conversion
    movwf adVal
    return
endz
end
Grip Code

LIST p=16F88
include "P16F88.inc"

;Setting ports
duration EQU 0x40
rightInput EQU 0x41
adVal EQU 0x42
thisInput EQU 0x43
check EQU 0x44
toggle EQU 0x45
count EQU 0x46
rightVal EQU 0x47
checkInput EQU 0x48
compareInput EQU 0x49

org 0x000
bsf STATUS,RP0 ;bank 1
bcf STATUS,RP1
movlw 0x07
movwf TRISA
clrf TRISB ;portb output
movlw 0x60
movwf OSCCON
clrf ADCON1
movlw 0x84
movwf OPTION_REG
bcf STATUS,RP0 ;bank 0

;initialize variables
initialize

clrf TMR0
clrf PORTA
movlw 0x01
clrf adVal
clrf PORTB
movlw 0x41
movwf ADCON0
clf toggle
clf rightVal
clf rightInput

;wait until activation bit is set
passive
btfs PORTA,2 ;check for activation bit
goto passive

;------generate function
;generate a value according to timer
generate
clf rightInput
clf compareInput
movf TMR0, 0
andlw 0x78
movwf rightVal
call decode
btfs rightVal, 6
call shiftfour

;------get the input
;get input and display led that corresponds
getInput
bcf PORTA,3 ; turn off check input button light

; set all bits up to bit X if AD gives a certain value
clf thisInput
call a2d
movlw 0xF8
andwf adVal,1

clf checkInput
btfs adVal,7
call setPort

clf checkInput
bsf checkInput,0
bsf thisInput,0
movlw 0x78
xorwf adVal,0
movwf check
btfs STATUS, Z
call setPort

clf checkInput
bsf checkInput,1
bsf thisInput,1
movlw 0x70
xorwf adVal,0
movwf check
btfs STATUS, Z
call setPort

clf checkInput
bsf    checkInput, 2
bsf    thisInput,  2
movlw  0x68
xorwf  adVal,  0
movwf  check
btfsc  STATUS, Z
call   setPort

clrf   checkInput
bsf    checkInput, 3
bsf    thisInput,  3
movlw  0x60
xorwf  adVal,  0
movwf  check
btfsc  STATUS, Z
call   setPort
movlw  0x58
xorwf  adVal,  0
movwf  check
btfsc  STATUS, Z
call   setPort

clrf   checkInput
bsf    checkInput, 4
bsf    thisInput,  4
movlw  0x50
xorwf  adVal,  0
movwf  check
btfsc  STATUS, Z
call   setPort
movlw  0x48
xorwf  adVal,  0
movwf  check
btfsc  STATUS, Z
call   setPort

clrf   checkInput
bsf    checkInput, 5
bsf    thisInput,  5
movlw  0x40
xorwf  adVal,  0
movwf  check
btfsc  STATUS, Z
call   setPort

movlw  0xF0
andwf  adVal,  1

movlw  0x30
xorwf  adVal,  0
movwf  check
btfsc  STATUS, Z
call   setPort

clrf   checkInput
bsf    checkInput, 6
bsf    thisInput, 6
movlw  0x20
xorwf  adVal, 0
movwf  check
btfs  STATUS, Z
call  setPort
movlw  0x10
xorwf  adVal, 0
movwf  check
btfs  STATUS, Z
call  setPort
clrf  checkInput
bsf  checkInput, 7
bsf  thisInput, 7
movf  adVal, 0
btfs  STATUS, Z
call  setPort
btfs  PORTA, 1
goto  checkThis
goto  getInput

;-------check function
;check to see if the input is correct
checkThis
bsf  PORTA, 3
movf  rightInput, 0
xorwf  compareInput, 0
movwf  check
btfs  STATUS, Z
goto  wrong
call  victory
bsf  PORTA, 7 ;set Correct bit
call  wait
call  wait
bsf  PORTA, 7
clr  PORTB
goto  passive

;-------victory function
;display a pattern that indicate victory
victory
clr  PORTB
call  wait0
call  wait0
call  wait0
cbf  PORTA, 3
movlw  0x10
movwf  count
bsf  PORTB, 0
loop1
call  wait0
rlf  PORTB, 1
decfsz  count
goto     loop1

movlw   0x10
movwf   count

loop2

    call    wait0
    rrf    PORTB,1
decfsz  count
    goto    loop2
clf     PORTB
return

;------set the LEDs connected to port
;set values to PORTB
setPort

    movf    checkInput,0
    movwf   compareInput
    movf    thisInput,0
    movwf   PORTB
    btfs    toggle,0
    call    setBit
    btfs    toggle,0
    call    resetBit

    movf    rightInput,0
    xorwf   checkInput,0
    movwf   check
    btfs    STATUS, Z
    call    setBit

decfsz  count,1
return

;------set the correct LED
;show the LED needed to be matched
setBit

    movf    rightInput,0
    iorwf   PORTB,1
return

;------reset the correct LED
;turn the LED needed to be matched
resetBit

    comf    rightInput,0
    andwf   PORTB,1
return

;------analog to digital converter
a2d

    bsf   ADCON0,GO
newConv

    btfs    ADCON0,GO ;Wait for conversion to complete
goto newConv

movf ADRESH,W
movwf adVal       ; set adVal to 8 MSB of 10bit AD conversion

;------wrong function
wrong
clr PORTB
bsf PORTA,6       ;set WRONG bit
call wait
bcf PORTA,3
bcf PORTA,6
goto getInput

;------shift register 4 bits
shiftfour
bcf STATUS,C       ; clear carry bit so register isn't affected
rlf rightInput,1
rlf rightInput,1
rlf rightInput,1
rlf rightInput,1
return

;------decoder
; 2 to 4 bit decoder
decode
clr rightInput
btfsc rightVal, 5
goto MSB1
btfsc rightVal, 5
btfss rightVal, 5
bsf rightInput, 3
return

MSB1
btfsc rightVal, 4
bsf rightInput, 0
btfss rightVal, 4
bsf rightInput, 1
return

MSB0
btfsc rightVal, 5
bsf rightInput, 2
btfss rightVal, 5
bsf rightInput, 3
return

;------wait function
;wait for a fixed duration of time
wait
movlw 'd121'
movwf duration
bcf INTCON,TMR0IF ; clear interrupt bit

wait4int
btfss INTCON,TMR0IF ; skip if interrupt is not enabled
goto wait4int
bcf INTCON,TMR0IF ; clear interrupt bit
decfsz duration
goto wait4int
return

;------shorter wait
;wait for a shorter duration of time

wait0

movlw d'07'
movwf duration ; clear interrupt bit
bcf INTCON,TMR0IF

wait4int0

btfss INTCON,TMR0IF ; skip if interrupt is not enabled
goto wait4int
bcf INTCON,TMR0IF ; clear interrupt bit
decfsz duration
goto wait4int0
return

end
Laser Maze Code – PIC A

; Robotics Academy - Kinetic Sculpture (2003-2004)
; Jason Adrian & Marc Weintraub
; Date Finalized: 4/10/04
;
; Target PIC: 16F88
; Program Name: LaserMazeA
; Purpose: Input device
; Description: A game where the player must flash a laser over a sequence of sensors in a certain order to win. One of two PICS
;
LIST p=16F88
include "P16F88.inc"

;Setting ports
count EQU 0x40
nextInput EQU 0x41
adVal EQU 0x42
thisInput EQU 0x43
waitCnt EQU 0x44

org 0x000
bsf STATUS,RP0 ;bank 1
bcf STATUS,RP1
movlw 0x1F
movwf TRISB
movlw 0x60
movwf TRISB ;portb outputs
movwf OSCCON
movlw 0x40
movwf ADCON1
movlw 0x1F
movwf ANSEL ;activate all AD channels
bcf STATUS,RP0 ;bank 0

;------main function
;set initial values
main

movlw 0x01
movwf nextInput
clrf thisInput
clrf adVal
clrf PORTB

;------check values
;convert the value on half of the sensors from analog to digital
;also check status of PICB. If PICB returns a signal that a wrong input has been received then the function wrong will be called. If PICB returns a signal indicating that the game has been completed, the victory function is called
checkInputs

clrf    thisInput
; check status of PICB
btfsc   PORTB,5
goto    wrong
; check status of PICB
btfsc   PORTB,6
goto    victory

; check value on sensor0
movlw   0xC1
movwf   ADCON0
call    a2d
        btfsc   adVal,7
        bsf    thisInput,0

; check value on sensor1
movlw   0xC9
movwf   ADCON0
call    a2d
        btfsc   adVal,7
        bsf    thisInput,1

; check value on sensor2
movlw   0xD1
movwf   ADCON0
call    a2d
        btfsc   adVal,7
        bsf    thisInput,2

; check value on sensor3
movlw   0xD9
movwf   ADCON0
call    a2d
        btfsc   adVal,7
        bsf    thisInput,3

; check value on sensor4
movlw   0xE1
movwf   ADCON0
call    a2d
        btfsc   adVal,7
        bsf    thisInput,4

; check to see if an input was acquired. return to checkInputs if not
movf    thisInput,1
        btfsc   STATUS,Z
        goto    checkInputs

; check to see if the input is in sequence with the previous inputs
movf    nextInput,0
        xorwf   thisInput,1
        btfss   STATUS,Z
        goto    wrong
; set appropriate values to portB
movf nextInput,0
iorwf PORTB,1
rlf nextInput

call waitA
call waitA
goto checkInputs

;------analog to digital converter
a2d
bsf ADCON0,GO
newConv
btfsc ADCON0,GO ;Wait for conversion to complete
goto newConv
movf ADRESH,W
movwf adVal ; set adVal to 8 MSB of 10bit AD conversion
return

;------wrong function
wrong
clrfr PORTB
bsf PORTB,7 ;set WRONG bit
call waitA
call waitA
call waitA
call waitA
bcf PORTB,7
goto main

;------victory function
;display pattern to indicate victory
victory
clrfr PORTB
btfsc PORTB,6
goto victory
call flash024
call flash13
call flash024
call flash13
call flash024
call flash13
call flash024
call flash13
call waitA
call waitA
call waitA
call waitA
goto main

;------display LEDs 0,2,4 for a certain time
flash024
movlw 0x15
movwf PORTB
call waitB
clrfr PORTB
call waitB
return

;------display LEDs 1,3 for a certain time
flash13
    movlw 0xA
    movwf PORTB
    call waitB
    clrf PORTB
    call waitB
return

;------delay generator
waitA
    movlw 0xFF
    movwf waitCnt
    d1a
    movlw 0xFF
    movwf count
    d2a
    NOP
    NOP
    NOP
    NOP
    decfsz count,1
    goto d2a
    decfsz waitCnt,1
    goto d1a
return

;------shorter delay
waitB
    movlw 0x77
    movwf waitCnt
    d1b
    movlw 0xAA
    movwf count
    d2b
    NOP
    NOP
    NOP
    NOP
    decfsz count,1
    goto d2b
    decfsz waitCnt,1
    goto d1b
return

;------even shorter delay
waitC
    movlw 0x10
    movwf waitCnt
    d1c
    movlw 0x33
    movwf count
    d2c
    decfsz count,1
goto d2c
decsz waitCnt,1
goto d1c
return

dend
Laser Maze Code – PIC B

LIST p=16F88
include "P16F88.inc"

;Setting ports
count EQU 0x40
nextInput EQU 0x41
adVal EQU 0x42
thisInput EQU 0x43
waitCnt EQU 0x44

org 0x000
bsf STATUS,RP0 ;bank 1
bcf STATUS,RP1
movlw 0x1F
movwf TRISB
movlw 0x60
movwf TRISB ;portb outputs
movwf OSCCON
movlw 0x40
movwf ADCON1
movlw 0x1F
movwf ANSEL ;activate all AD channels
bcf STATUS,RP0 ;bank 0

;-----main function
;set initial values
main
movlw 0x01
movwf nextInput
clrf thisInput
clrf adVal
clrf PORTB

;-----check values
;convert the value on half of the sensors from analog to digital
;also check status of PICB. If PICB returns a signal that a wrong input has
;been received then the function wrong will be called. If PICB returns
;a signal indicating that the game has been completed, the victory function
;is called
checkInputs

    clrf   thisInput;
    ; check status of PICB
btfsc   PORTB,5
    goto   wrong
    ; check status of PICB
btfsc   PORTB,6
    goto   victory

    ;check value on sensor5
movlw   0xC1
movwf   ADCON0
    call   a2d
btfsc   adVal,7
bsf   thisInput,0

    ;check value on sensor6
movlw   0xC9
movwf   ADCON0
    call   a2d
btfsc   adVal,7
bsf   thisInput,1

    ;check value on sensor7
movlw   0xD1
movwf   ADCON0
    call   a2d
btfsc   adVal,7
bsf   thisInput,2

    ;check value on sensor8
movlw   0xD9
movwf   ADCON0
    call   a2d
btfsc   adVal,7
bsf   thisInput,3

    ;check value on sensor9
movlw   0xE1
movwf   ADCON0
    call   a2d
btfsc   adVal,7
bsf   thisInput,4

    ; check to see if an input was acquired. return to checkInputs if not
movf   thisInput,1
btfsc   STATUS,Z
    goto   checkInputs

    ; check status of PICA. If PICA is not complete, any input is an error.
btfss   PORTB,6
    goto   wrong
; check to see if the input is in sequence with the previous inputs
movf nextInput,0
xorwf thisInput,1
btfss STATUS,Z
goto wrong

; set appropriate values to portB
movf nextInput,0
iorwf PORTB,1
rlf nextInput
call waitA
call waitA

; check to see if all the LEDs are lit at this point
btfsc PORTB,4
goto victory
goto checkInputs

;----analog to digital converter
a2d
bsf ADCON0,GO
newConv
btfsc ADCON0,GO ;Wait for conversion to complete
goto newConv

movf ADRESH,W
movwf adVal ; set adVal to 8 MSB of 10bit AD conversion
return

;-----wrong function
wrong
; tell PICA that something went wrong
clrf PORTB
bsf PORTB,5
call waitC
bcf PORTB,5
goto main

;-----victory function
;display pattern to indicate victory
victory
call waitC
cclf PORTB
call flash13
call flash024
call flash13
call flash024
call flash13
call flash024
bsf PORTB,7
call waitA
call waitA
call waitA
goto main

;------display LEDs 0,2,4 for a certain time
flash024
    movlw 0x15
    movwf PORTB
    call waitB
    clrf PORTB
    call waitB
    return

;------display LEDs 1,3 for a certain time
flash13
    movlw 0x0A
    movwf PORTB
    call waitB
    clrf PORTB
    call waitB
    return

;------delay generator
waitA
    movlw 0xFF
    movwf waitCnt
    d1a
    movlw 0xFF
    movwf count
    d2a
    NOP
    NOP
    NOP
    NOP
    decfsz count,1
    goto d2a
    decfsz waitCnt,1
    goto d1a
    return

;------shorter delay
waitB
    movlw 0x77
    movwf waitCnt
    d1b
    movlw 0xAA
    movwf count
    d2b
    NOP
    NOP
    NOP
    NOP
    decfsz count,1
    goto d2b
    decfsz waitCnt,1
    goto d1b
return

;-----even shorter delay
waitC
    movlw 0x10
    movwf waitCnt
  d1c
    movlw 0x33
    movwf count
  d2c
    decfsz count,1
    goto d2c
    decfsz waitCnt,1
    goto d1c
return

end
Noise Code

; Robotics Academy - Kinetic Sculpture (2003-2004)
; Jason Adrian & Marc Weintraub
; Date: 4/10/04
;
; Target PIC: 16F88
; Program Name: Noise
; Purpose: Input device
; Description: The player is to produce noise of a
certain loudness for a certain amount of time
;

LIST p=16F88
include "P16F88.inc"

; Setting ports
duration EQU 0x40
rightInput EQU 0x41
adVal EQU 0x42
thisInput EQU 0x43
check EQU 0x44
toggle EQU 0x45
count EQU 0x46
rightVal EQU 0x47
checkInput EQU 0x48
compareInput EQU 0x49

org 0x000
bsf STATUS,RP0 ;bank 1
bcf STATUS,RP1
movlw 0x07
movwf TRISA
clrf TRISB ;portb output
movlw 0x60
movwf OSCCON
clrf ADCON1
movlw 0x01
movwf ANSEL ;activate all AD channels
; set timer0
movlw 0x84
movwf OPTION_REG
bcf STATUS,RP0 ;bank 0

; initialize variables
initialize
clrf TMR0
clrf PORTA
movlw 0x01
clrf adVal
clrf PORTB
movlw 0x41
movwf ADCON0
  clrf toggle
  clrf rightVal
  clrf rightInput

; wait until activation bit is set
passive
  btfss PORTA,2 ; check for activation bit
  goto passive

  clrf rightInput
  bsf rightInput,3

setVal ; simply resets count
  movlw d'255'
  movwf duration

;----- get the input
; get input and display led that corresponds
getInput
  bcf PORTA,3 ; turn off 'check input button light

  ; set all bits up to bit X if AD gives a certain value
  clrf thisInput
  call a2d
  movlw 0xE0
  andwf adVal,1

  clrf checkInput
  movf adVal,1
  btfsc STATUS, Z
  goto setPort

  clrf checkInput
  bsf checkInput,0
  bsf thisInput,0
  movlw 0x20
  xorwf adVal,0
  movwf check
  btfsc STATUS, Z
  goto setPort

  clrf checkInput
  bsf checkInput,1
  bsf thisInput,1
  movlw 0x40
  xorwf adVal,0
  movwf check
  btfsc STATUS, Z
  goto setPort

  clrf checkInput
  bsf checkInput,2
  bsf thisInput,2
  movlw 0x60
  xorwf adVal,0
movwf check
btfsc STATUS, Z
  goto setPort

clrf checkInput
bsf checkInput, 3
bsf thisInput, 3
movlw 0x80
xorwf adVal, 0
movwf check
btfsc STATUS, Z
  goto setPort

clrf checkInput
bsf checkInput, 4
bsf thisInput, 4
movlw 0xA0
xorwf adVal, 0
movwf check
btfsc STATUS, Z
  goto setPort

clrf checkInput
bsf checkInput, 5
bsf thisInput, 5
movlw 0xC0
xorwf adVal, 0
movwf check
btfsc STATUS, Z
  goto setPort

clrf checkInput
bsf checkInput, 6
bsf thisInput, 6
movlw 0xE0
xorwf adVal, 0
movwf check
btfsc STATUS, Z
  goto setPort

;-----victory function
; display a pattern that indicate victory
victory
  clrf PORTB
  call wait0
  call wait0
  call wait0
  bcf PORTA, 3
  movlw 0x10
  movwf count
loop1
  movlw 0x81
  movwf PORTB

call wait0
movlw 0x42
movwf PORTB
call wait0
movlw 0x24
movwf PORTB
call wait0
movlw 0x18
movwf PORTB
call wait0
movlw 0x24
movwf PORTB
call wait0
movlw 0x42
movwf PORTB
call wait0

decfsz count
goto loop1

return

;------set the LEDs connected to port
;set values to PORTB
;wait until noise is activated for certain amount of time
setPort
movf checkInput,0
movwf compareInput
movf thisInput,0
movwf PORTB
movf rightInput,0
xorwf checkInput,0
movwf check
btfss STATUS, Z
goto reset

goto victory

;------analog to digital converter
a2d
bsf ADCON0,GO
newConv
btfsc ADCON0,GO ;Wait for conversion to complete
goto newConv

movf ADRESH,W
movwf adVal ; set adVal to 8 MSB of 10bit AD conversion
return

;-----wait function
;wait for a fixed duration of time

wait
    movlw d'121'
    movwf duration
    bcf INTCON,TMR0IF ; clear interrupt bit

wait4int
    btfss INTCON,TMR0IF ; skip if interrupt is not enabled
    goto wait4int
    bcf INTCON,TMR0IF ; clear interrupt bit
    decfsz duration
    goto wait4int
return

;-----shorter wait
;wait for a shorter duration of time

wait0
    movlw d'07'
    movwf duration
    bcf INTCON,TMR0IF ; clear interrupt bit

wait4int0
    btfss INTCON,TMR0IF ; skip if interrupt is not enabled
    goto wait4int0
    bcf INTCON,TMR0IF ; clear interrupt bit
    decfsz duration
    goto wait4int0
return

end
Turnstile Code

; Robotics Academy - Kinetic Sculpture (2003-2004)
; Jason Adrian & Marc Weintraub
; Date Finalized: 4/10/04
;
; Target PIC: 16F84A
; Program Name: Stepper Motor
; Purpose: Control the stepper motor used for the
; turnstile.
; Description: This program sequences the four
; coils in the stepper to rotate 120
; degrees one direction, pause, then reverse
; direction and return to the starting point
;
LIST p=16F84A
include "P16F84a.inc"
_CONFIG_ 0x3d18

;Setting ports
count1 EQU 0x40
count2 EQU 0x42
input EQU 0x48
looper EQU 0x46

org 0x00
bsf STATUS, RP0
movlw 0x0f ; A is input
movwf TRISA
clrf TRISB ; B is output
bcf STATUS, RP0

main
movf PORTA,0
movwf input
btfss input,0
goto main

movlw 0x13
movwf looper

forward
movlw 0x03 ; step sequence
movwf PORTB ; to motor driver circuitry

call wait

movlw 0x06
movwf PORTB

call wait
movlw 0x0C
movwf PORTB

call wait

movlw 0x09
movwf PORTB

call wait

decfsz looper,1
goto forward

; forward sequence
; repeat backwards

movlw 0x13
movwf looper

; pause

movlw 0x00
movwf PORTB

call wait
call wait
call wait
call wait
call wait
call wait
call wait
call wait
call wait
call wait
call wait
call wait
call wait
call wait

backward

movlw 0x09 ; step sequence
movwf PORTB ; to motor driver circuitry

call wait

movlw 0x0C
movwf PORTB

call wait

movlw 0x06
movwf PORTB
call wait
movlw 0x03
movwf PORTB
call wait
decfsz looper,1
goto backward
goto main

; delay loop

wait
movlw 0x66            ; else load count1 with
movwf count1
d1a
movlw 0x66          ; else load count2 with
movwf count2
d2a
decfsz count2,1
goto d2a
decfsz count1,1
goto d1a
return

end