# 1.0 Objectives

In this lab you will create a video game in assembly language that runs on the i281 simulator.

# 2.0 Parts List

| **Quantity** | **Item** |
| --- | --- |
| 1 | Computer with access to the internet |
| 1 | Internet browser running the [i281 Simulator](https://www.ece.iastate.edu/~alexs/classes/i281_CPU/i281_CPU_Software.zip)  |
| - | Access to the Quartus Program |
| 1 | FPGA Board |

# 3.0 Background

For this lab you will be using the i281 simulator to code your video game. Below is a reminder of the i281 assembly instructions that were introduced in lab 10.

## 3.1 Assembly Instructions

Assembly is a low-level programming language that strongly corresponds to the machine code of the underlying computer architecture.

| **Instruction** | **Function** | **Use Format** |
| --- | --- | --- |
| NOOP | No operation | NOOP |
| INPUTC  | Input into code memory | INPUTC [ destination ] |
| INPUTCF | Input into code memory with an offset specified by a register | INPUTCF [ destination + offset ] |
| INPUTD  | Input into data memory | INPUTD [ destination ] |
| INPUTDF | Input into data memory with an offset specified by a register | INPUTDF [ destination + offset ] |
| MOVE  | Move the contents of one register into another | MOVE register1, register2 |
| LOADI  | Load immediate value | LOADI register, number |
| ADD  | Add two registers | ADD register1, register2 |
| **Instruction** | **Function** | **Use Format** |
| ADDI  | Add an immediate value to a register | ADDI register, number |
| SUB  | Subtract two registers | SUB register1, register2 |
| SUBI  | Subtract an immediate value from a register | SUB2 register, number |
| LOAD  | Load from a data memory address into a register | LOAD register, [ source ] |
| LOADF  | Load with an offset specified by another register | LOADF register, [ source + offset ] |
| STORE  | Store a register into a data memory address | STORE register, [ source ] |
| STOREF  | Store with an offset specified by another register | STOREF [ source + offset ], register |
| SHIFTL  | Shift left all bits in a register | SHIFTL register |
| SHIFTR  | Shift right all bits in a register | SHIFTR register |
| CMP  | Compare the values in two registers | CMP register1, register2 |
| JUMP  | Jump unconditionally to a specified address | JUMP destination  |
| BRE  | Branch if equal | BRE destination  |
| BRZ  | Branch if zero | BRZ destination  |
| BRNE  | Branch if not equal | BRNE destination  |
| BRNZ  | Branch if not zero | BRNZ destination  |
| BRG  | Branch if greater | BRG destination  |
| BRGE  | Branch if greater than or equal | BRGE destination  |

## 3.2 Formatting Assembly Code for the i281 Simulator

The simulator expects the code to be formatted in a certain way. In order to run your program, ensure that you follow these rules:

* Start your program with a .data header. This lets the compiler know what variables to store in the data memory.
* Variables are declared in the following format:

name-of-variable BYTE value-of-variable

Within this format value can be ? if it is unknown. It will default to 0.

* After your variables are declared, begin your program with the .code header. This tells the compiler where your code begins.
* When accessing the data memory in the program use the following format:

[ name-of-variable ]

* The four registers are named A, B, C, and D
* Locations can be specified in the program using the format

Address-label: instruction

# 4.0 Activity

You will create the video game in several steps. Each of them implements a component of the overall design. Watch the short video that introduces the game before you proceed.

## 4.1 Up Down Position Change

* Use switch 6 to move the bar up and down.
* Download its assembly code and study it.
* Modify the sample program that ships with the simulator.
* Load and run the Up-Down code so that the vertical bar displays in the fifth 7-segment display rather than the eighth.
* Demonstrate the working program to the TA.



Figure C: Up-Down sample program.

##

## **4.**2Left Right Summon Object

* Modify the Left-Right sample program that comes with the simulator to display a moving  instead of the horizontal bar .
* Which parts of the program did you modify?
* Show your modified version to the TA.



Figure B: Left-Right sample program.

##

##

## **4.**3 **Create Objects**

* Modify the Left-Right sample program such that the horizontal bar no longer bounces off the two sides. Instead it should start from the left, travel four positions to the right, and then repeat on the left again.
* This animation on the 7-segment displays should run in an infinite loop.
* Show your progress to the TA.



Figure A: Bit representation for a seven-segment display.

## 4.4 Cycle Creation

* This part requires you to combine the inputs from parts 4.2 and 4.3.
* Write an assembly program that animates a moving  object.
* This object should start on the left, move four positions to the right, and then reappear on the left, similar to 4.4.
* The new challenge here is to alternate the start position of the object between up and down on each left-right cycle. That is, the initial placement should be down  but after the object reaches the right-most 7-segment display it should reappear on the left in the up configuration .
* This two-step animation should run in an infinite loop.
* Show your code and sample output to the TA.

## 4.5 Add User Input

* Take your code from 4.4 and combine it with the unedited Up-Down example from the simulator. That is, add a user controlled vertical bar that is displayed on the right-most 7-segment display.
* Use switch 6 to control the bar.
* Don’t implement collision detection between the square and the bar yet.
* Demonstrate the working program to the TA.

##

## 4.6 Collision Detection

* Add a check to see if the user’s vertical bar will collide with the square.
* Remember the shape in the rightmost segment will need to ensure that it doesn’t override the player’s sprite if they’re successful in avoiding collision.
* If a collision is detected the game is over and END should be displayed on 3 of the seven-segment displays.
* If the player was successful in avoiding a collision, the animation should display both the square object and the player’s vertical bar in the right-most 7-segment display. See the code for PONG on how to handle this.
* Demonstrate the final version of your video game to the TA.