RETROACTIVE DESIGN

Reinventing the Pac-Man

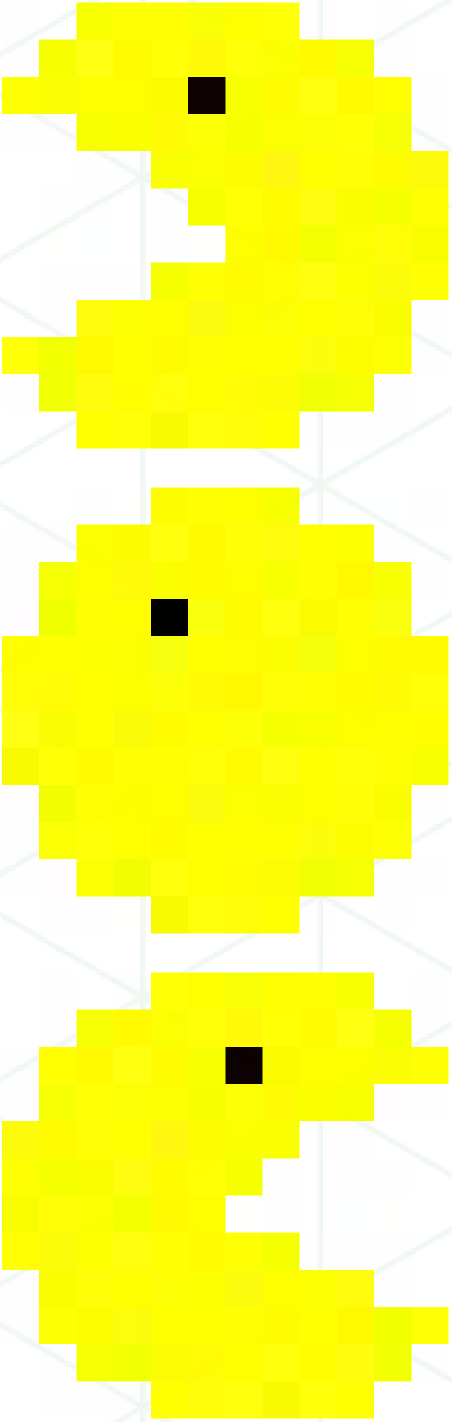
Ariosto Ferro   Todd Cronin
Jeff Thacker   Zach Wilcox

University of Utah Fall 2012 • ECE 3710

Pac-Man 2.0

Switched application from Asteroids to Pac-Man

- Vector graphics were too complicated
- Asteroids had too complicated of movement patterns, required trig
- Switched to glyph based VGA graphics
- Pacman works very well with glyphs and uses grid based movement



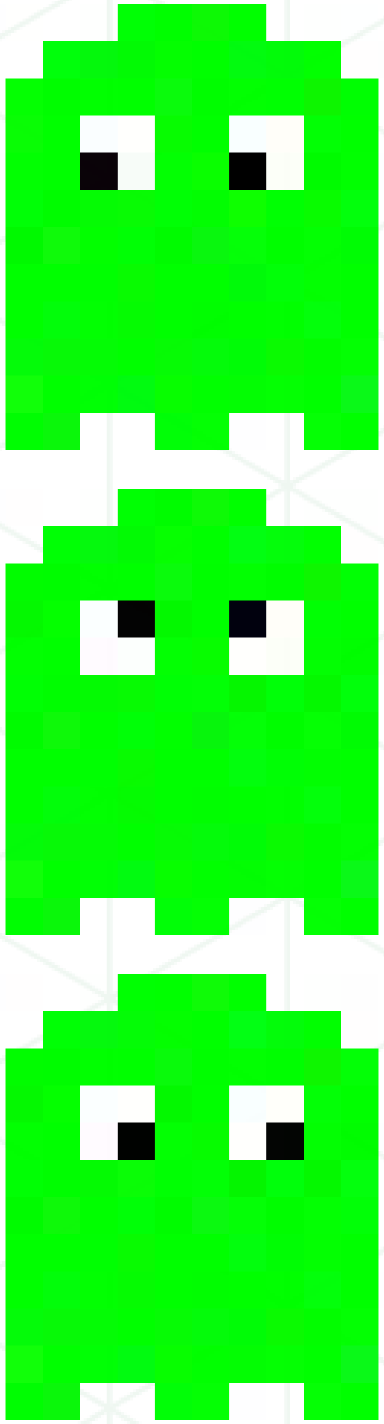
APPLICATION

Basic Game Logic:

- Game Loop
  - Move ghosts and Pac-Man by one grid square
  - Update frame buffer with new game state
  - Output sounds and update game timers
  - Jump to top of game loop or return to menu if game over

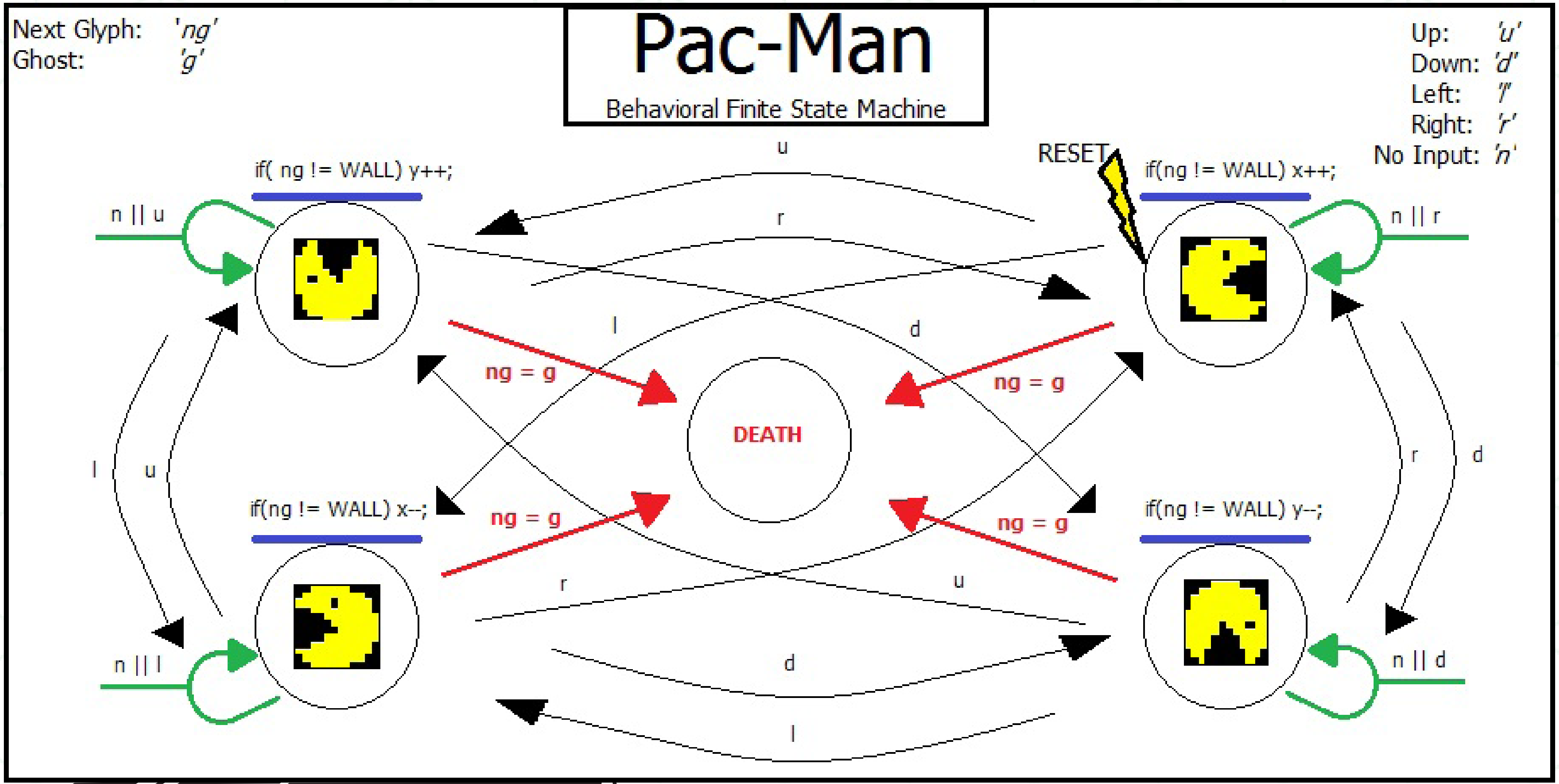
Characters require movement algorithms

- Pac-Man
  - Dependent on user input
  - Restricted by walls
- Ghosts (Ari, Jeff, Todd, Zach)
  - Move towards a target position, also restricted by walls
    - Target position can be current position of Pac-Man
      or or a position off screen during 'scatter mode'



Application

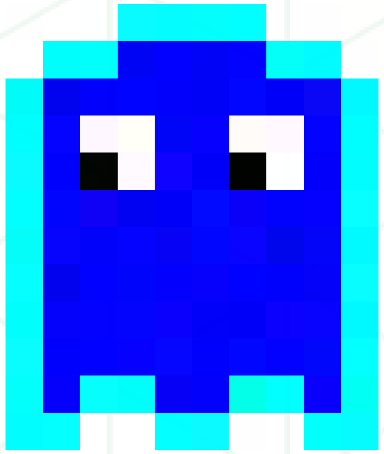
Pac-Man and ghost behavior can be represented by a FSM

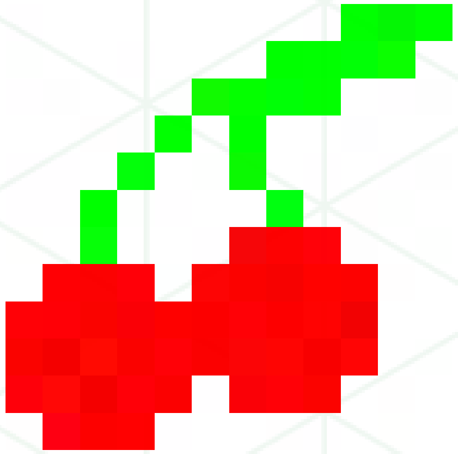


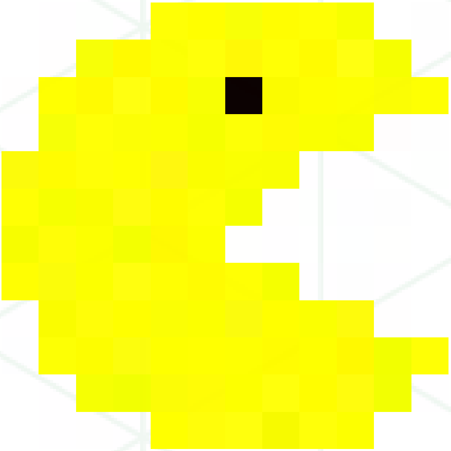
APPLICATION

Uses several types of glyphs

- 9 Glyphs for the Walls
- 4 Glyphs for Pac-Man with mouth open (1 for each position)
- 4 Glyphs for Pac-Man with closed mouth
- 3 Glyphs to ‘Melt’ Pac-Man when he dies.
- 1 Glyph for normal pill
- 1 Glyph for Super Pill
- 16 Glyphs for ghost (4 Ghost, 4 Directions of Eyes)
- 8 Glyphs for SuperPill Ghost (2 Color State, 4 Directions)
- Examples of our glyphs can be found through out our presentation.



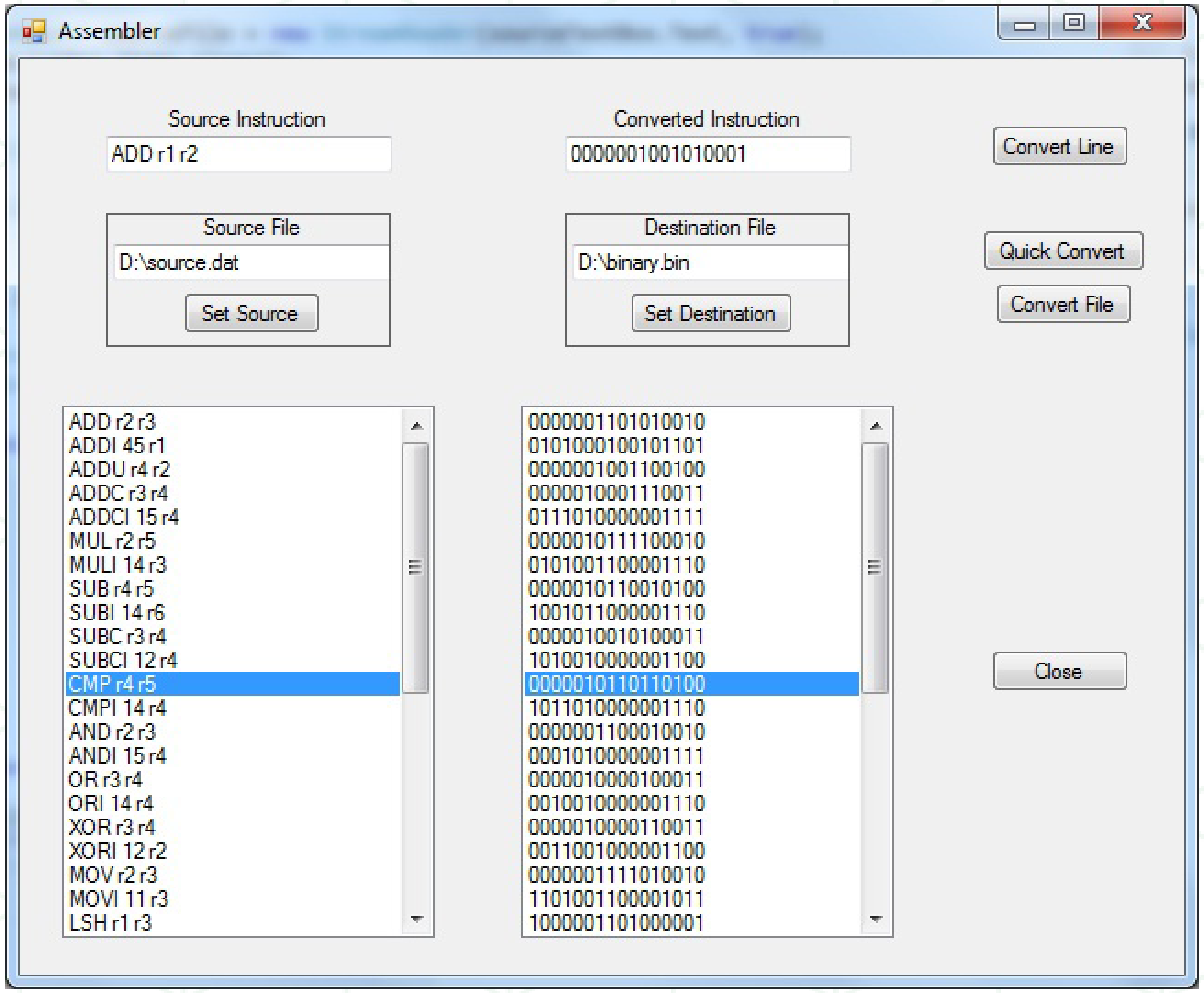




Assembler

GUI Interface

- Windows form C# Application
- Individual instructions or file-at-a-time
- Supports all instructions supported by the processor.
- Instructions have the following conventions:
  1. Registers start with ‘r’, **ex:** r12
  2. Immediates are number only.
  3. Special registers are: sp, ra, fp



Source Instruction: ADD r1 r2

Converted Instruction: 0000001001010001

Source File: D:\source.dat

Destination File: D:\binary.bin

```
ADD r2 r3       0000001101010010
ADDI 45 r1      0101000100101101
ADDU r4 r2      0000001001100100
ADDC r3 r4      0000010001110011
ADDCI 15 r4     0111010000001111
MUL r2 r5       0000010111100010
MULI 14 r3      0101001100001110
SUB r4 r5       0000010110010100
SUBI 14 r6      1001011000001110
SUBC r3 r4      0000010010100011
SUBCI 12 r4     1010010000001100
CMP r4 r5       0000010110110100
CMPI 14 r4      1011010000001110
AND r2 r3       0000001100010010
ANDI 15 r4      0001010000001111
OR r3 r4        0000010000100011
ORI 14 r4       0010010000001110
XOR r3 r4       0000010000110011
XORI 12 r2      0011001000001100
MOV r2 r3       0000001111010010
MOVI 11 r3      1101001100001011
LSH r1 r3       1000001101000001
```

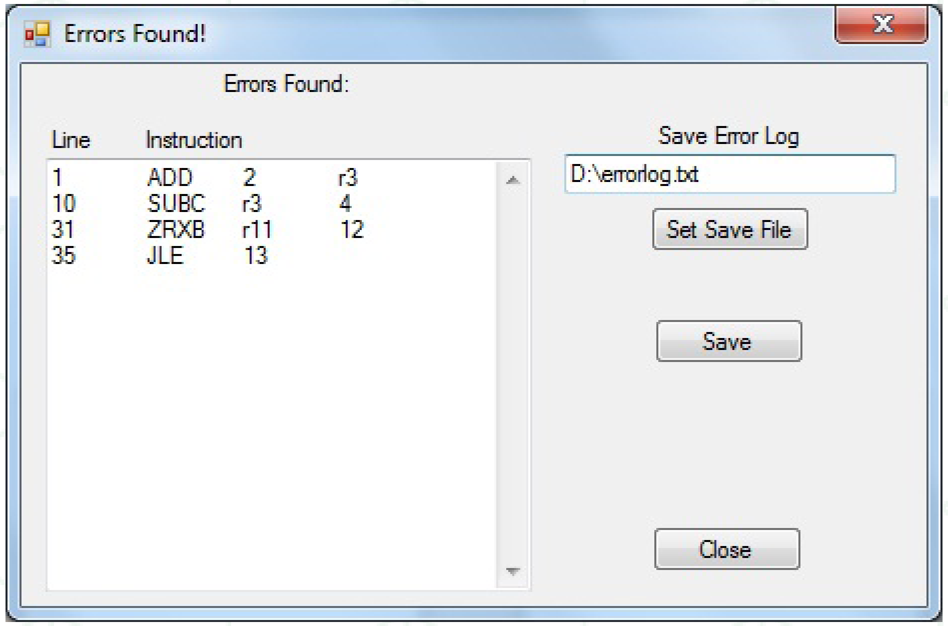
Assembler

Features

- Assembler ignores all commented lines.
- Assembler reads labels and calculates branch offsets automatically.

Features

- Generates a list of line numbers of unrecognized instructions.
- Writes all error messages into a log.
- Quick convert creates an output file without error checking.

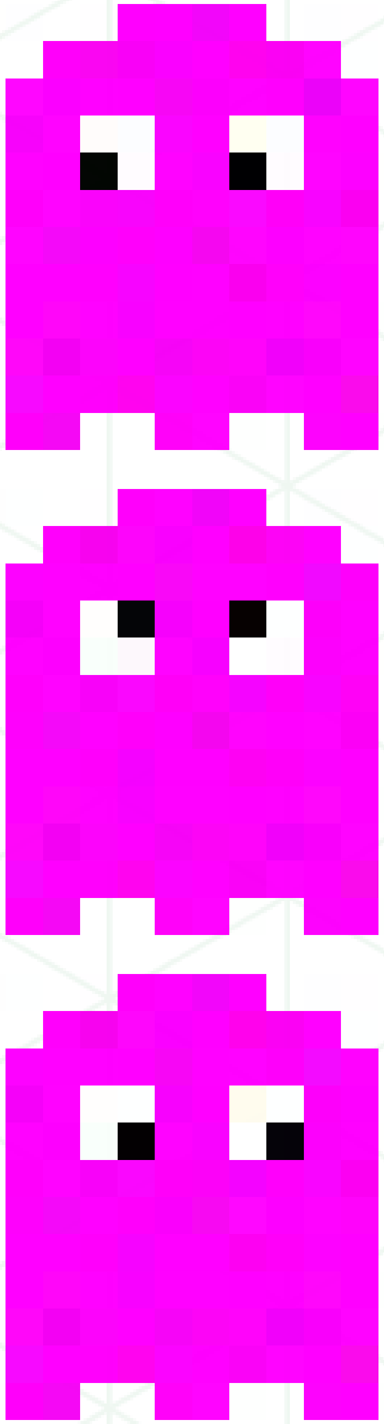


Errors Found!

Errors Found:

```
Line      Instruction
1         ADD     2       r3
10        SUBC    r3      4
31        ZRXB    r11     12
35        JLE     13
```

Save Error Log: D:\errorlog.txt



PROCESSOR

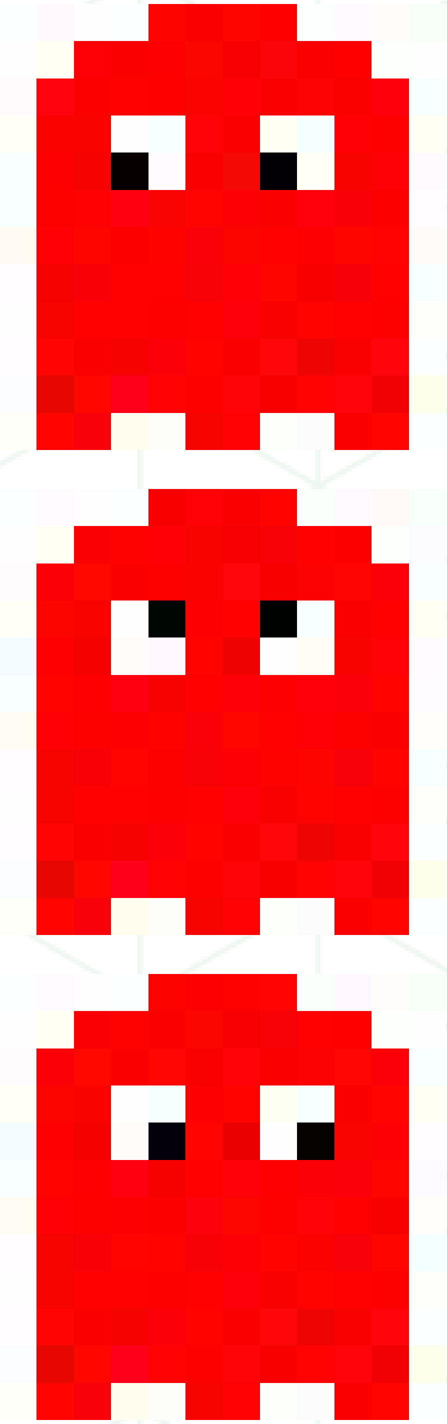
Based on CR-16 Architecture

- 16 Bit architecture. 16 registers. 13 general purpose
- 50MHz clock cycle
- All instructions take three clock cycles to complete
- 32Kbytes of RAM. (16384 16-bit words)
- Supports all of the following instructions:

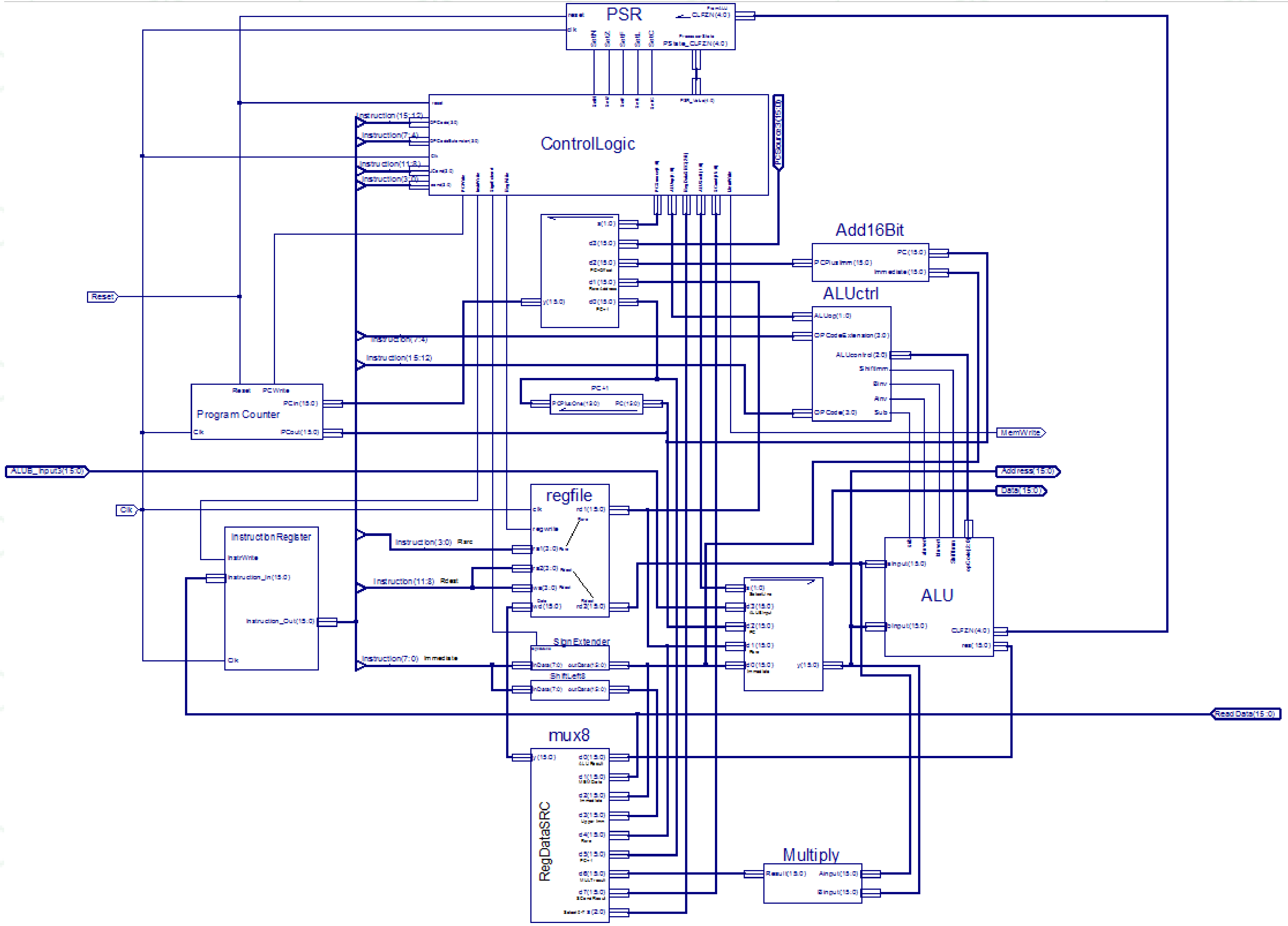
ADD / ADDI
ADDU / ADDUI
ADDC / ADDCI
MUL / MULI
SUB / SUBI
CMP / CMPI

AND / ANDI
OR / ORI
XOR / XORI
MOV / MOVI
LSH / LSHI
ASHU / ASHUI

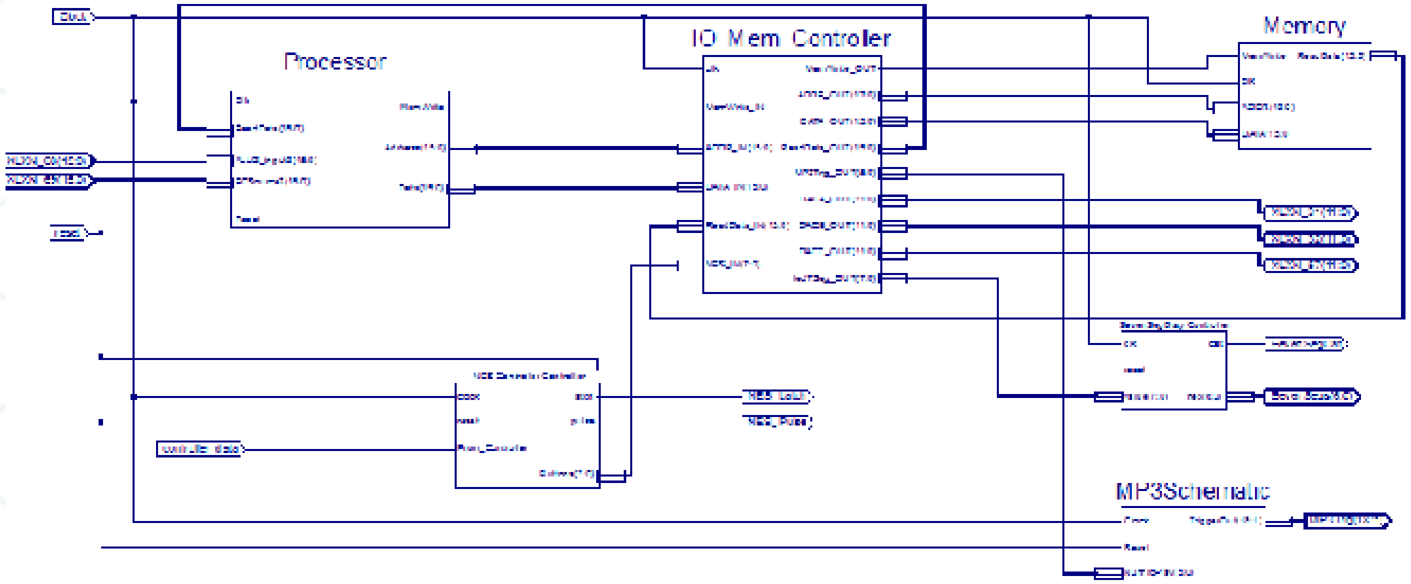
LUI
LOAD
STOR
Scond
Bcond
Jcond



Processor


PSR
reset
clk
CLFZN(4:0)
PSstate_CLFZN(4:0)
ControlLogic
reset
Instruction(15:12)
Instruction(7:4)
Instruction(11:8)
Instruction(3:0)
PCSource3(15:0)
a(1:0)
d3(15:0)
d2(15:0)
d1(15:0)
d0(15:0)
y(15:0)
Add16Bit
PC(15:0)
PCPlusImm(15:0)
Immediate(15:0)
ALUctrl
ALUop(1:0)
OPCodeExtension(3:0)
ALUcontrol(2:0)
ShiftImm
Binv
Ainv
OPCode(3:0)
Sub
Reset
Instruction(7:4)
Instruction(15:12)
PC+1
PCPlusOne(15:0)
PC(15:0)
Program Counter
Reset
PCWrite
PCin(15:0)
Clk
PCout(15:0)
MemWrite
ALUB_inputs(15:0)
Address(15:0)
Data(15:0)
Clk
regfile
clk
regwrite
ra1(3:0)
ra2(3:0)
wa(3:0)
wd(15:0)
rd1(15:0)
rd2(15:0)
Instruction Register
InstrWrite
Instruction_In(15:0)
Instruction_Out(15:0)
Clk
Instruction(3:0) Rsrc
Instruction(11:8) Rdest
Instruction(7:0) Immediate
Sign Extender
inData(7:0)
outData(15:0)
ShiftLeft8
inData(7:0)
outData(15:0)
a(1:0)
d3(15:0)
d2(15:0)
d1(15:0)
d0(15:0)
y(15:0)
ALU
aInput(15:0)
bInput(15:0)
CLFZN(4:0)
res(15:0)
Read Data(15:0)
mux8
RegDataSRC
y(15:0)
d0(15:0)
d1(15:0)
d2(15:0)
d3(15:0)
d4(15:0)
d5(15:0)
d6(15:0)
d7(15:0)
Multiply
Result(15:0)
Ainput(15:0)
Binput(15:0)

I/O

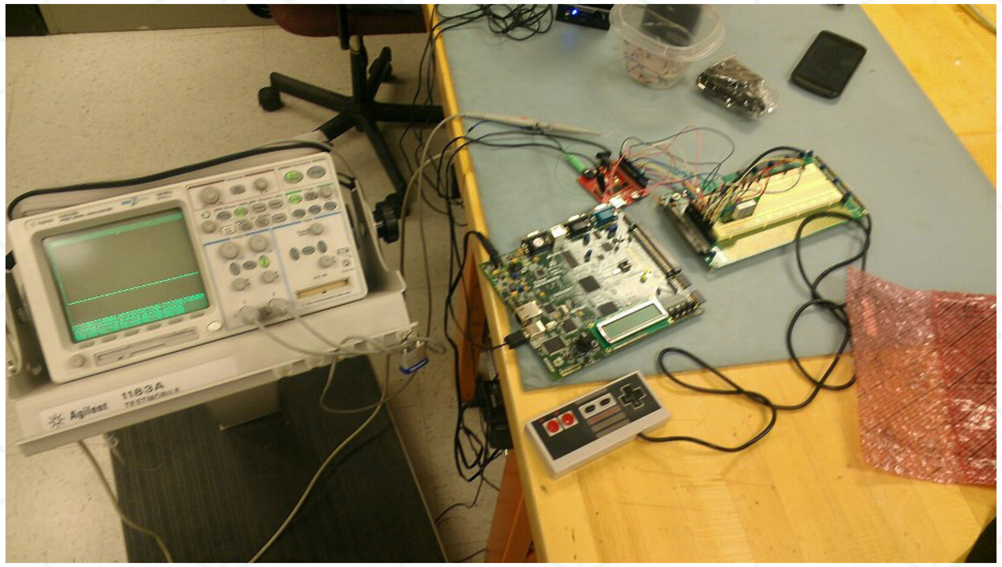


Four basic I/O devices

- NES Controllers
- MP3 Trigger
- VGA Graphics
- Seven Segment Display

Memory Mapped I/O interface

- Processor reads/writes to memory addresses
- One address for each controller button
- One address to activate MP3 trigger
- Frame buffer for VGA
- One address to output to 7-Seg Display



NES Controller

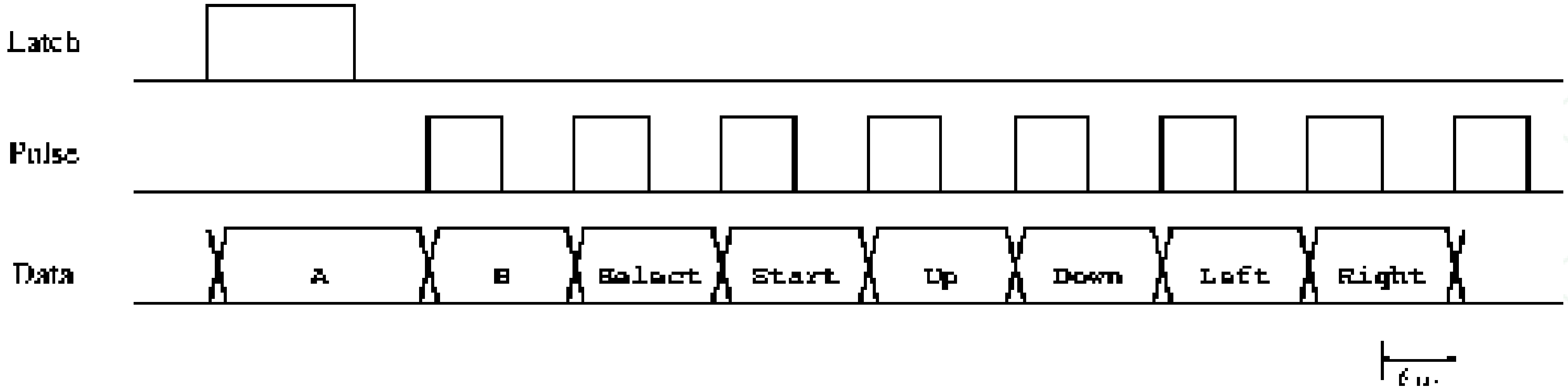
Basic Interface

- 3 Data Wires: Latch, Pulse, Data
- LATCH/PULSE are inputs to controller.
- DATA is output from controller
- Each Pulse is 6μs, 50% duty cycle
- Data is active low
- Controlled by Finite State Machine

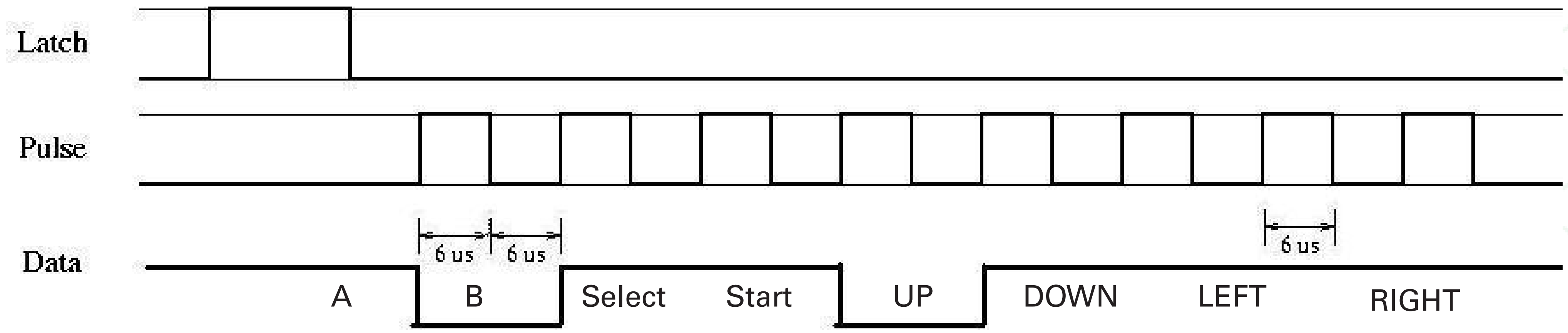
Finite State Machine

- FSM has 17 states, one for latch and two for each pulse.
- FSM rests in IDLE state until a 60Hz enable signal activates it.

NES Controller



- The waveform below illustrates the B and UP buttons being pressed on the NES controller.



MP3 Trigger

Basic Interface

- Plays a pre-loaded MP3 track when activated
- 18 command trigger lines or UART control
- UART and MP3 Trigger have 38.4KHz clock
- Trigger lines are active low
- Trigger functions are set with initialization file
- UART supports functions the triggers don’t

Controller

- Control logic mostly combinational
- Processor writes trigger number to I/O address
- Control logic sends low signal to trigger number
- One clock cycle is too fast — send low signal longer

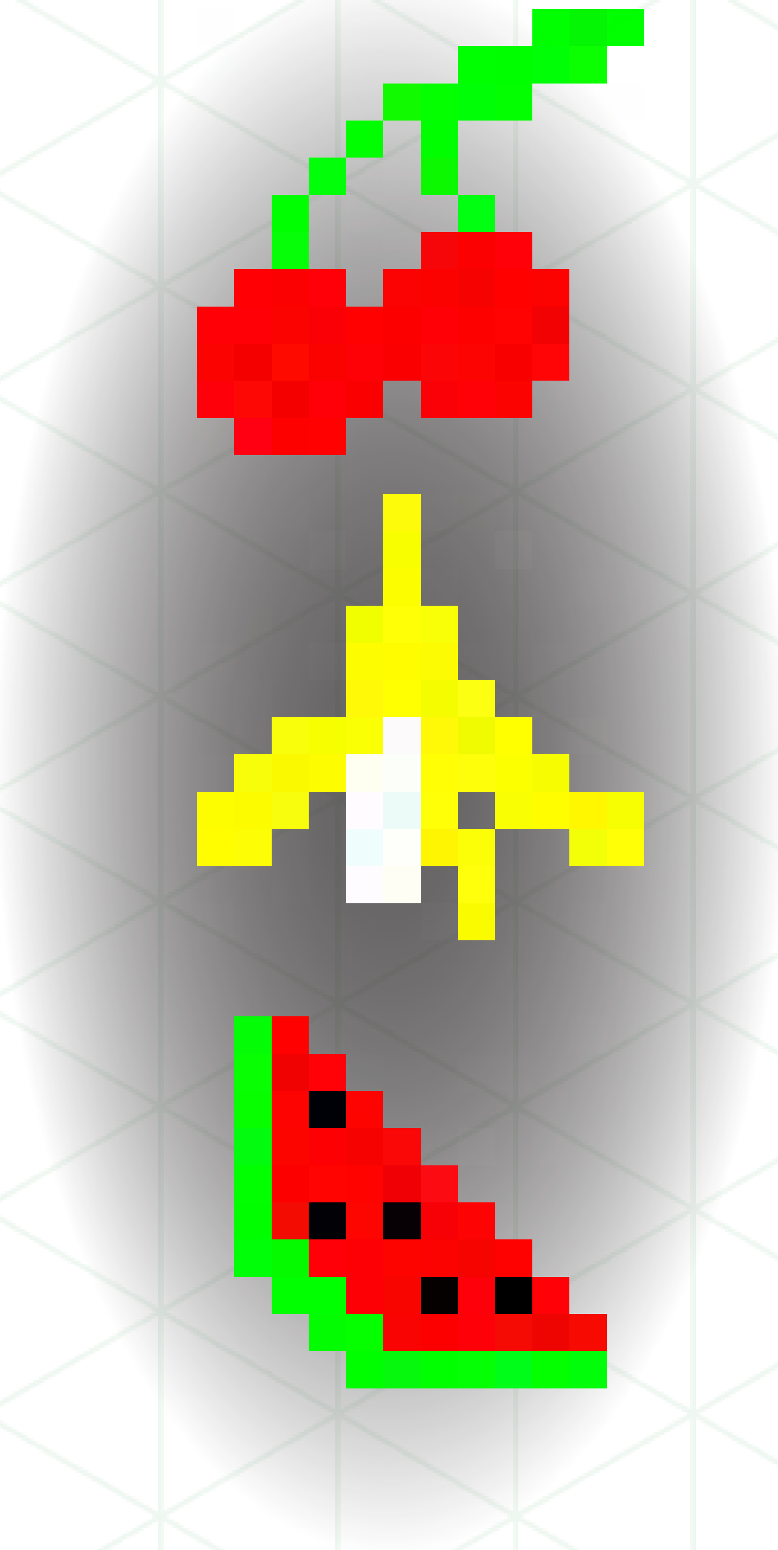
VGA Graphics

Switched from Vector to VGA

- Limited information on vector graphics
- On-board DACs are fairly slow
- No good scheme to update graphics

VGA

- Three-bit color, 8 total colors supported.
- 640x480 pixel resolution
- Glyph based graphics
- 12 x 12 pixel glyphs
- Glyphs are hard-coded into ROM.
- Graphics are stored in a 53x40 glyph frame buffer



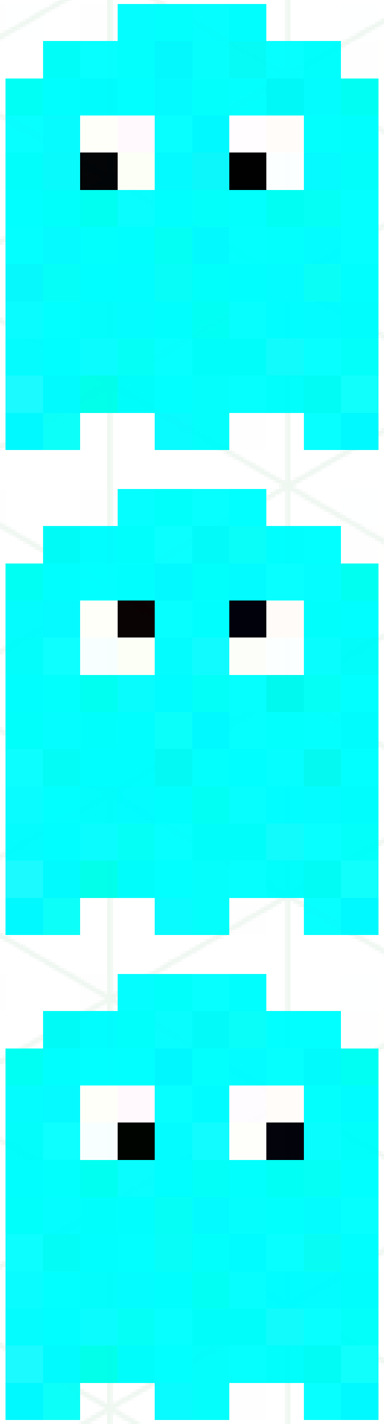
7-Segment Display

Used for testing/debugging

- Displays between 0x00 and 0xFF in hex.
- Memory mapped I/O interface.
- Processor writes 8-bit value to memory I/O address

Interface

- 8 data pin interface. 1 pin for each segment and a digit select
- When digit select is low the first digit is updated.
- When digit select is high the second digit is updated.
- Inputs must oscillate at ~1KHz to keep segment light on



Closing Thoughts

The next weeks are devoted to writing and testing the application.

This project was really exciting; the broad scope of tasks required to create a working processor was eye opening.

We would do it again.