Novosibirsk State University

**4COM1042 [Computing Platforms]**

**Co-design Group**

# Project B “The Game of TV-Tennis”

Anton Pimonov 21931 & Mikhail Komarov 21931

2022

**Contents**

1. Introducton
2. Overview
3. Hardware
4. Software
5. Conclusion

## **Introduction**

For our group project we were able to choose one theme from three basic ones or come up with something original. We chose TV-Tennis, because we didn’t find other two basic themes interesting and because we think TV-Tennis great, yet simple example of using everything we have passed through the program of our course.

As you can see, we successfully implemented TV-Tennis on Logisim + CdM-8 platform (Logisim for circuits, CdM-8 for code).

We divided our showcase in 3 parts: overview, hardware and software.  
We’ll begin with overview.

## **Overview**

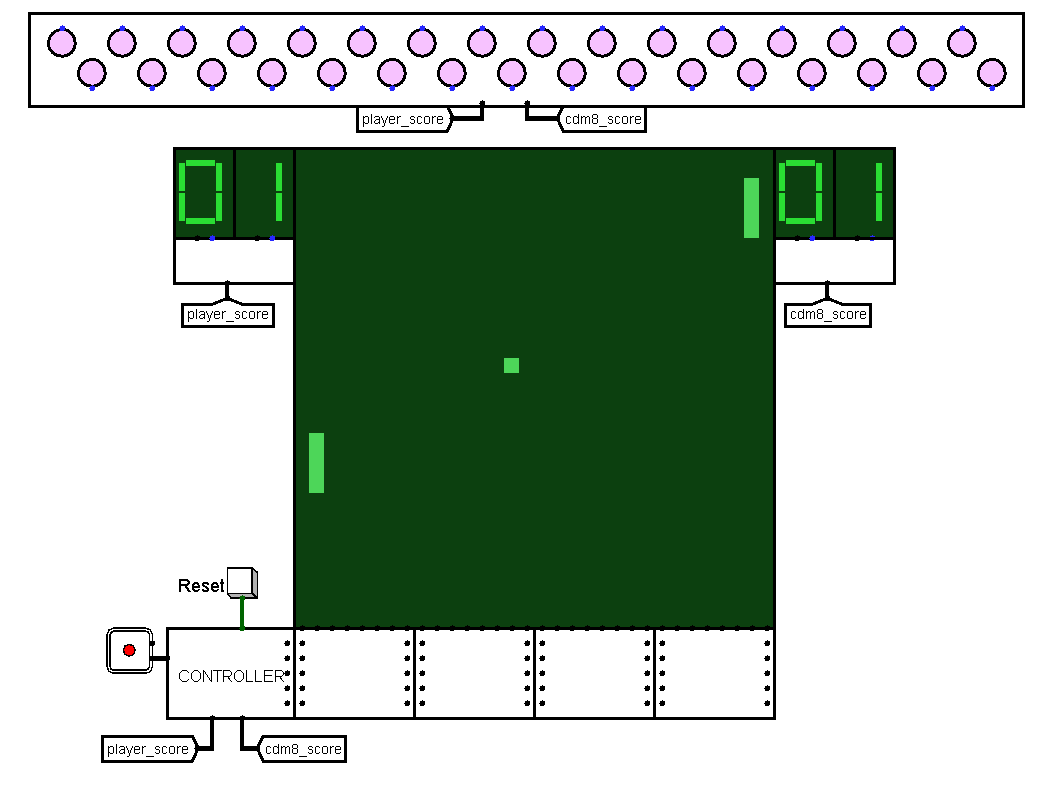
Let’s split components that player is able to see on “user” and “technical”.

“User” components: video display, joystick, restart button, score.  
These pieces have meaning to player, he is able to see what happens because of display and score, to control the bat because of joystick, and to restart the game because of restart button.

“Technical” components: video chip and kinematic controller.  
These pieces do all work, this is where everything being calculated (ball movement, bats movement etc). It has nothing to do with the player, he just does the inputs and get the results, he has no need to see or understand what happens between those things.

All of this you can see on the next page (main circuit, actually).

Now let’s move to the Hardware part.



## **Hardware**

### Display

What do we need to see on display? The ball and the bats.

How can we display it? By deciding what the ball and the bats are.

The whole display – pixel panel, 1024 pixels. In fact, this is 32 columns of pixels, counted from 0 to 31. Each column has a 32-bit input pin. If Nth bit is 1, Nth pixel turns on.

Now we can represent the ball by just turning on any pixel on the display, because the ball is just a single pixel. The player bat – three pixels in 1st column, player is able to move it up and down. The bot bat – three pixels in 30th column, same rules of movement, just like to player.

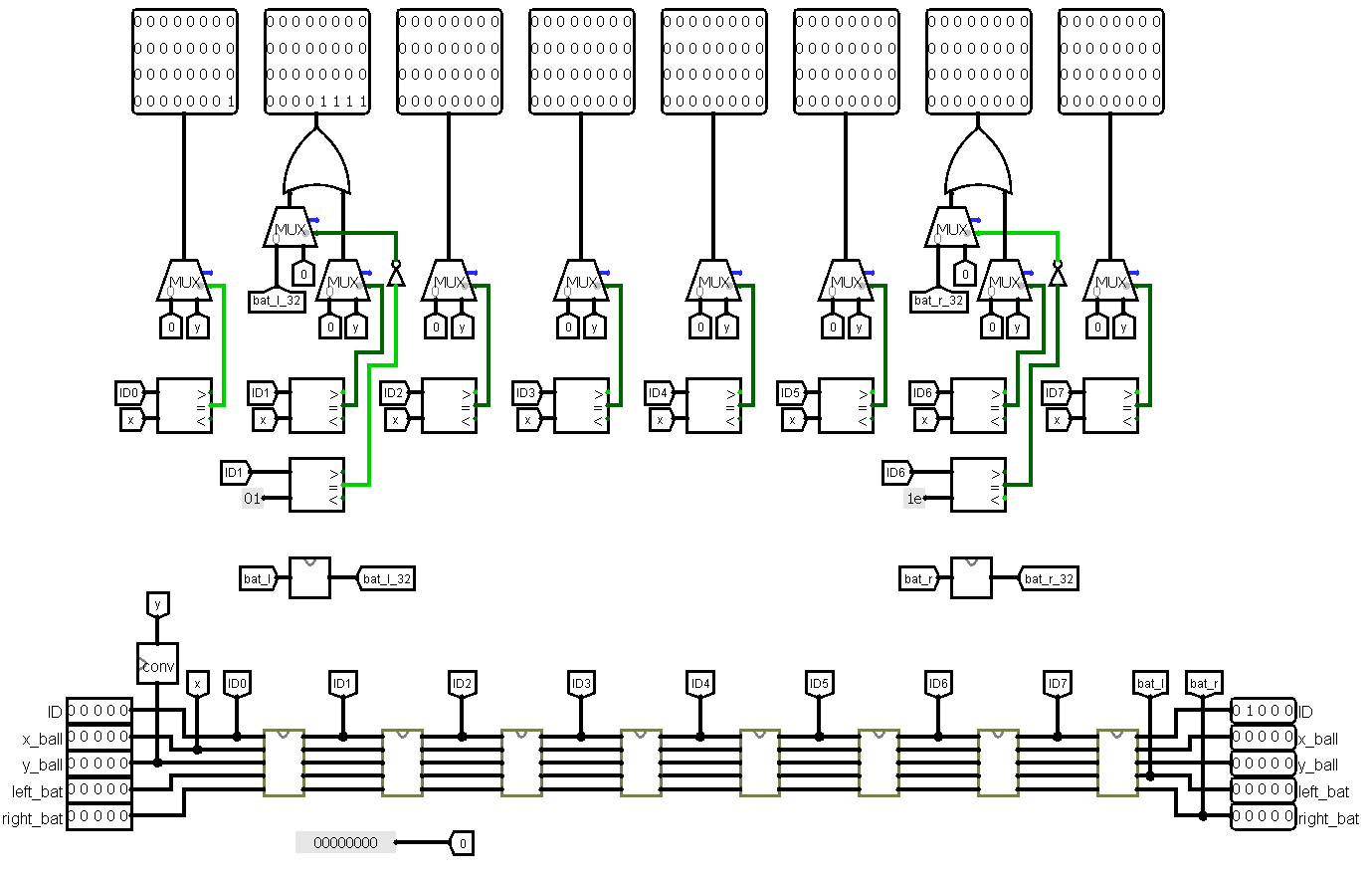
But it’s not enough to display the ball, we need to make it move. For this we need to know X and Y coordinates of the ball right now and the speed of the ball (speedX and speedY). This information represented by 5-bit numbers. More about it will be in “kinematic controller” part.

### video\_section

Inputs: ID, x\_ball, y\_ball, left\_bat, right\_bat (5 bit)

Outputs: ID, x\_ball, y\_ball, left\_bat, right\_bat (5 bit), 8 column outputs (32 bit)

There are 4 circuits like this under display, each one has 5 5-bit input pins, 5 5-bit and 8 32-bit output pins, connected to columns. 5-bit inputs and outputs carry information about column number, X and Y coordinates of the ball and coordinates of the bats. If X coordinate of the ball and column number are equal, bit equal to Y coordinate of the ball goes 1 and turns on the pixel. This is how we display the ball. There is also bat-check: if column number equal to 1 or 30, we are turning on 4 pixels, position of lower one we get from input, other ones are just 3 pixels above.

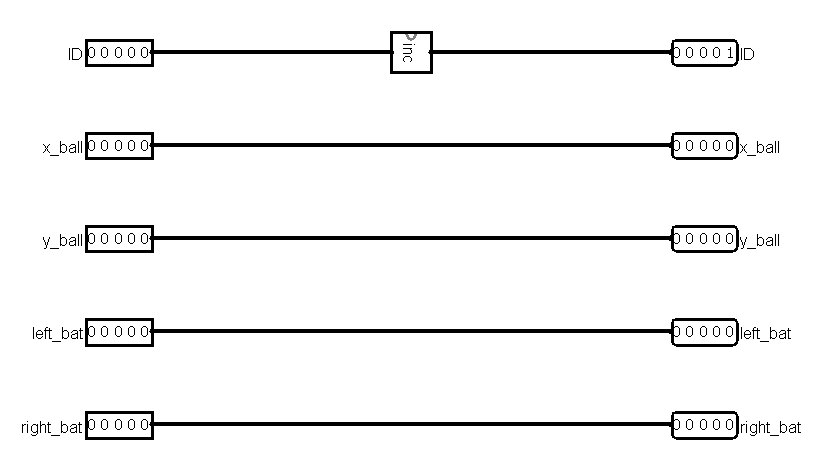


### video\_chip

Inputs: ID, x\_ball, y\_ball, left\_bat, right\_bat (5 bit)

Outputs: ID, x\_ball, y\_ball, left\_bat, right\_bat (5 bit)

This little circuit has 5-bit inputs and outputs identical to video\_section circuit. The only task of this circuit: it increases ID (column number) by 1.



### Video chip

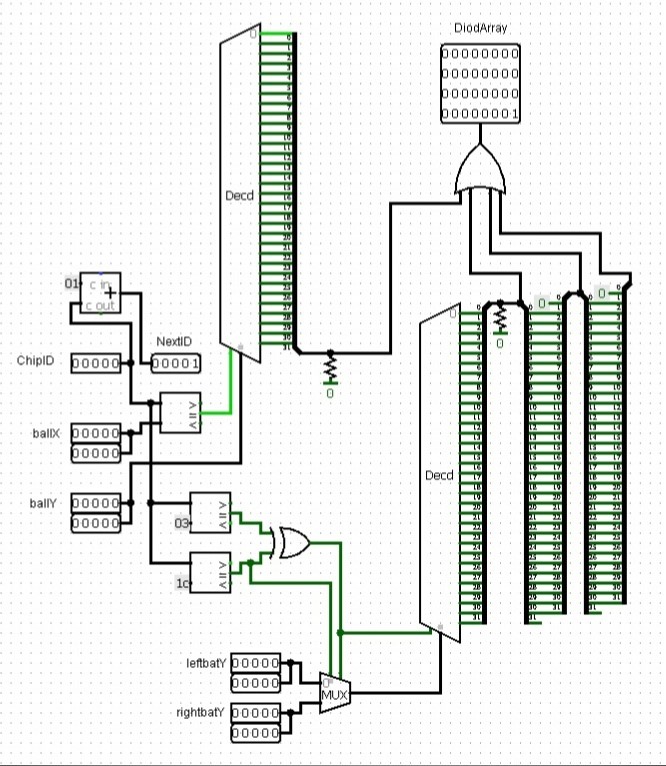
Every video chip has ballX, ballY, leftbatY, rightbatY input pins on its West side and ballX, ballY, leftbatY, rightbatY output pins on its East side, and the bit patterns on each of them pass through each video chip unchanged.

Each video chip drives a specific column of the display panel, and each chip needs to ‘know’ which column it is driving, so the chips are numbered 0..31 like the columns. We give each video chip an additional 5-bit input pin labelled chipID on its West side, and add a corresponding output pin on its East side, but this time we add 1 to the number before outputting it again. Then when we connect the chips in a chain we make sure to input 0b00000 on the chipID pin of the West-most chip (= column 0), which adds 1 and so sends the signal 0b00001 to the next chip in the chain. This way all 32 chips number themselves automatically.

Each chip includes circuitry to display a ball, and circuitry to display a single bat.

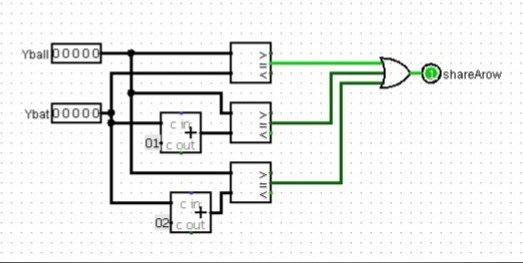
The ball’s position is received on input pins ballX and ballY. Each chip compares its chipID to ballX, and if they are equal it lights up pixel number ballY in its column of the display. Because the bats are in fixed columns we can ‘hard-wire’ their column numbers (3 and 28) into the video subsystem. Every video chip contains two 5-bit constants: 0b00011 and 0b11100. These two constants are compared with the chipID, so the video chip that has 00011 as its chipID displays the left bat with its bottom pixel in row leftbatY, and the video chip that has 11100 as its chipID displays the right bat with its bottom pixel in row rightbatY.

Also the video chip checks the ball and the bats for collisions using the comparators and decoders to send the result to the computer.



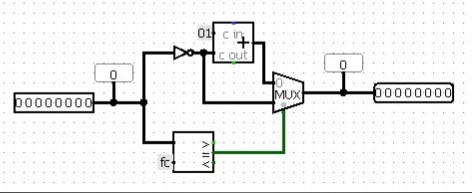
### Inrange

Collision of the ball and the bat happens when coordinates Yball and Ybat (or Ybat +1 or +2) are equal. This chip checks both of the coordinates and rises the signal if the collision occurs.



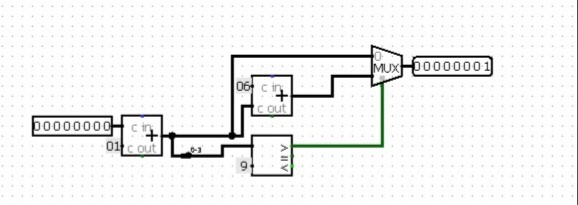
### Vinvert

The velocity (vxy) of the ball has a horizontal x-component (vx) and a vertical y-component (vy). Though each of these is represented by a 8-bit 2’s complement (signed) number, we will use Vx and Vy only in range -4..+3 in order to interpret it as 3 bit 2’s complement numbers later. A positive x velocity means that the ball is travelling from left to right across the space, and a positive y velocity means that the ball is travelling from bottom to top. Negative velocities of course mean that the ball is travelling in the opposite direction. So, this chip inverts the velocity of the ball considering these conditions.



### BCDinc

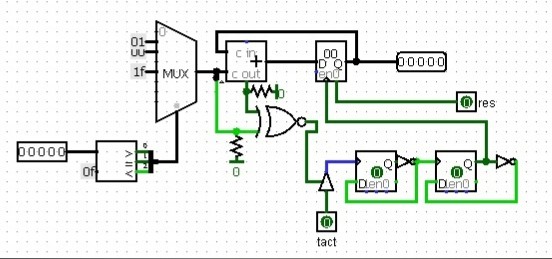
Logisim uses octal notation 8-bit numbers to representante the scores. That's why we made a chip transforming number in the decimal numeral system by adding 6 to numbers which are not equal in both numeral systems.



### LStepper

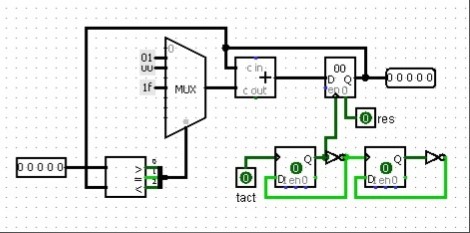
The Joystick's movement in Logisim is very coarse. We made this chip to make it a bit slower for the left bat.

Joystick’s display takes 30 pixels where 15 is the middle. Chip compares the pixel's address and moves the bat up or down gradually with the help of d-triggers (devices for slowing down the clocks).



### RStepper

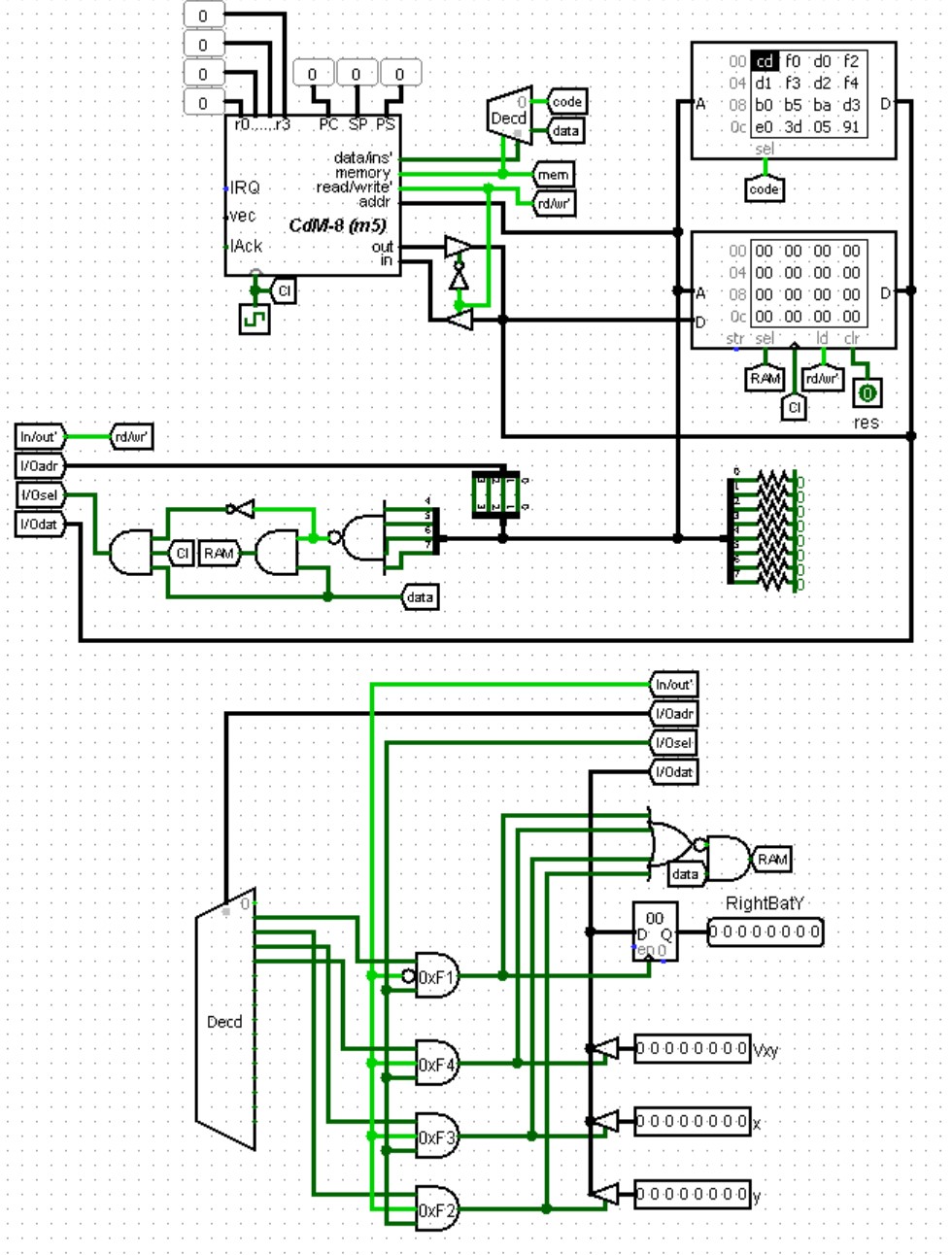
The usage of RStepper is the same as the previous chip with the only one difference - we compare the last bat’s address with the previous one.



### CDM-8 Processor

A Harvard-architecture uses two separate memories, one for program code and one for data, each of which has its own separate address space. The main advantage of Harvard over von Neumann lies in the possibility of using both memories simultaneously (giving greater speed). Using a Harvard architecture we may use 512 bytes of memory for larger system designs, one half for code and the other half for data.

The main differences lie in the use of load and store instructions, because we now need to specify which memory we are addressing. In our work with platforms we will treat code memory as read-only, because overwriting code at run-time can be disastrous. A Harvard architecture requires the CPU to indicate which memory (code or data) it needs to use in each clock cycle.

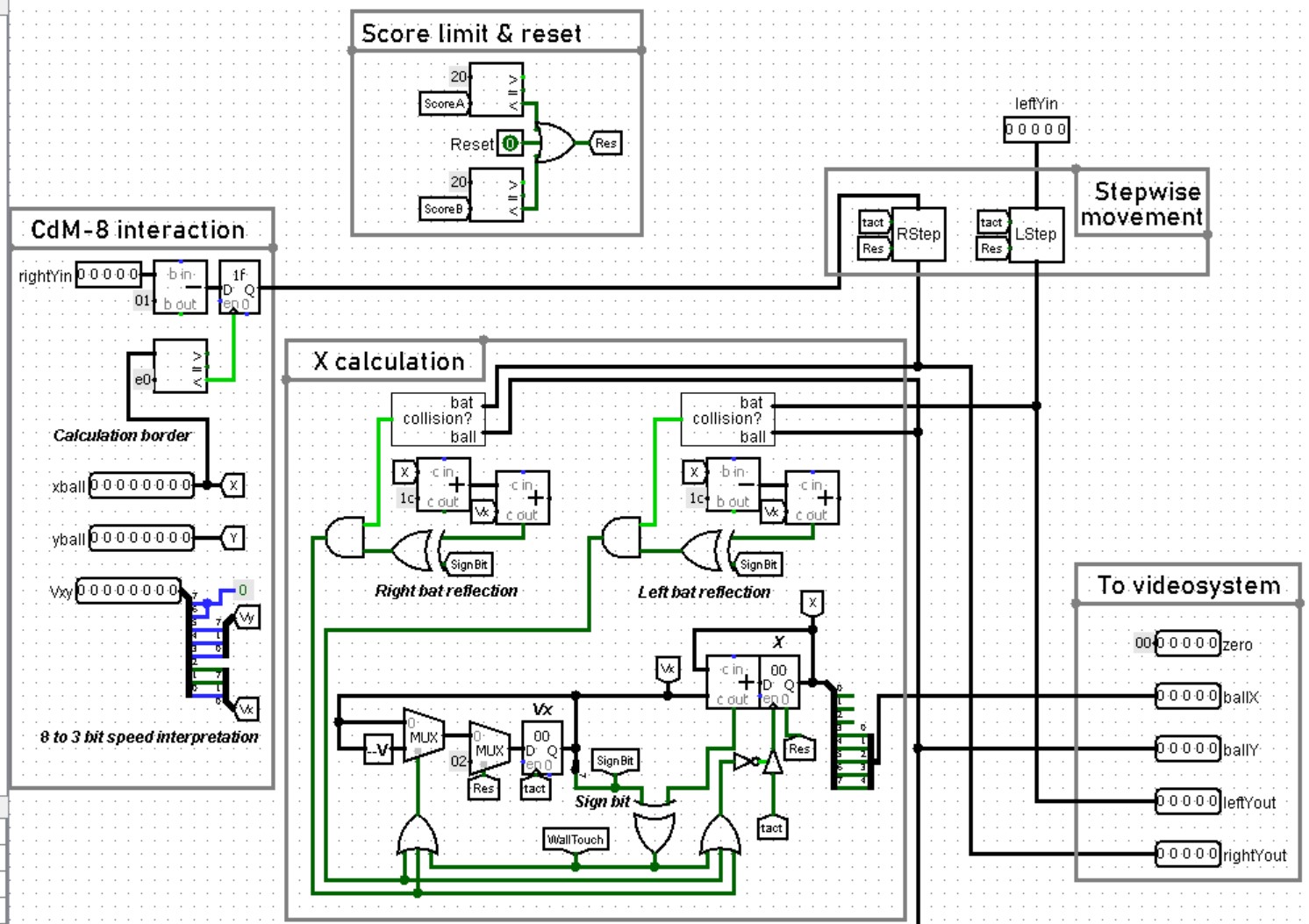


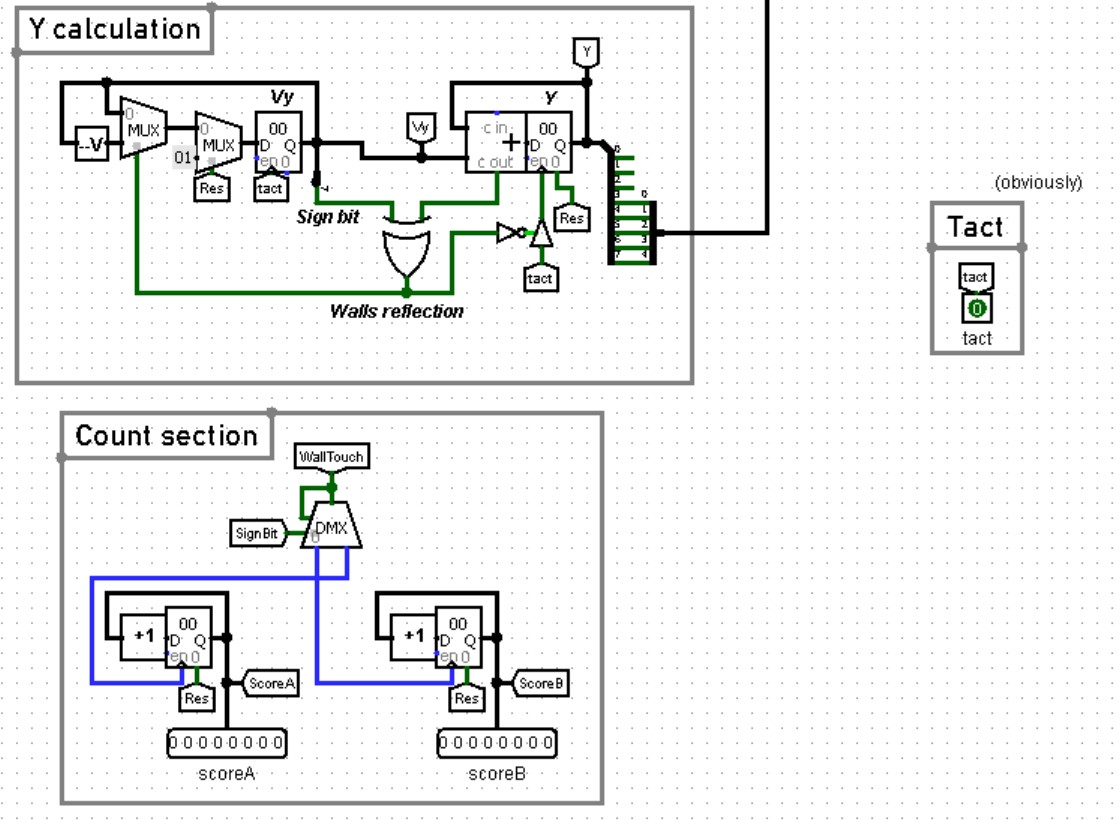
### Kinematic controller

The work of the kinematic controller is to move the ball by updating its coordinates, to block the right bat when the ball is on the right-hand side of the screen, and to report the positions of the ball and two bats to other parts of the system (program and video subsystem).

The controller updates the coordinates and velocity of the ball, and reports them to the program, which uses these values to predict the flight path of the ball and to move its bat accordingly. The kinematic controller needs to know the position of both bats to determine whether the ball has hit either of them, and is also responsible for ‘blocking’ the right bat when the ball is in the right-hand half of the screen.

Also there are a score counter and chip that stops the game when human or computer reaches 20 scores.





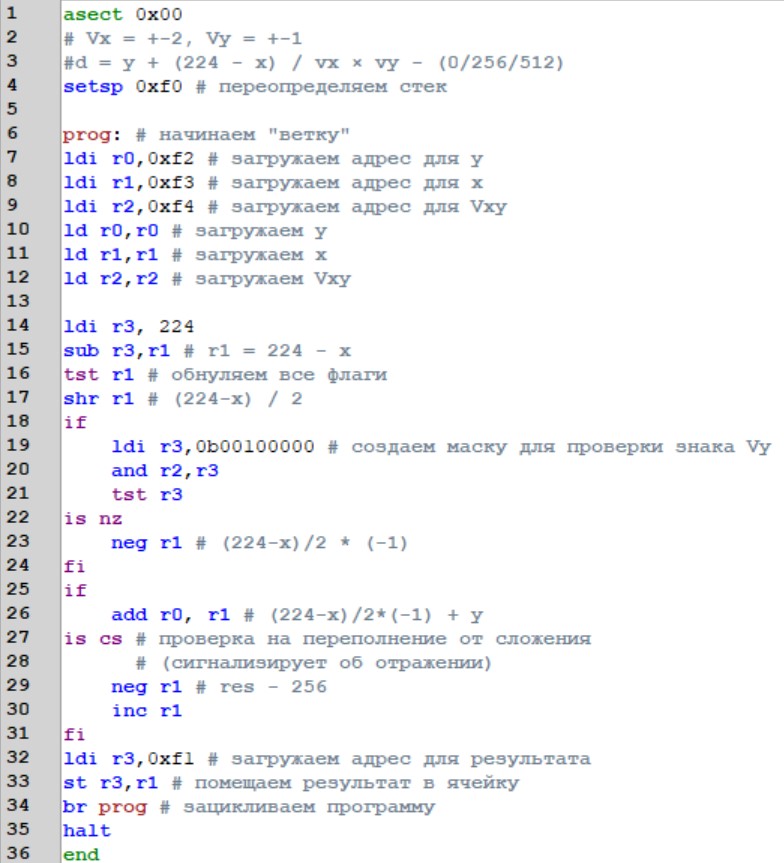
## Description of software part

The program has to predict ball’s trajectory given its current position and velocity. It should determine the point at which the ball will cross the 28th vertical of the display (counting from 0) and write its y-coordinate to the IO-3 register. It must do so reasonably quickly because if in the meantime the ball travels as far as the median point of the display, the controller will not copy the content of IO-3 to its eatsern pin, and as a result the right bat will not move and the ball will likely hit the right wall.

Instead of imitating the controller by adding small increments to the coordinates and working out the next location until we approach the bat, we must compute the final location of the ball at the point of crossing in one step.

The distance between the ball and the right bat along the horizontal axis is 224 – x (here and below we will measure any distances in sub-pixels to scale the board up to the size 256x256). If the horizontal velocity of the ball is vx, it will take the ball (224 – x)/vx controller clock cycles to reach the bat line. Its vertical displacement over this time will be (224 – x)/vx × vy. This displacement, when added to the current y-coordinate defines the intersection point on the bat line at which the ball will arrive. Or rather would arrive if we could guarantee that it would not collide with the horizontal wall.

In the current design no object other than the ball ever moves in the x-direction; this means that vx has a constant magnitude. The controller sets vx to –2 (110 in 3-bit 2’s complement) upon reset, so the absolute value of the horizontal speed stays at 2. On the other hand, vy is a 3-bit number as well, so its absolute value could be at most 4. This means that we can easily both divide and multiply in the program. We divide by 2 by executing the right shift, and we multiply by 2 and 4 by doing left shifts; finally multiplication by 3 boils down to a shift and an add.



## Conclusion

Finally, we found the work with the project very interesting and productive. Our group practised with the writing assembler’s code, creating chips and collecting them.We realized TV-Tennis with the updated joystick’s movement and made the game looks more presentable because of this update.