**MINISTRY OF SCIENCE AND HIGHER EDUCATION OF THE RUSSIAN FEDERATION FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION**

**"NOVOSIBIRSK NATIONAL RESEARCH UNIVERSITY STATE UNIVERSITY"**

**(NOVOSIBIRSK STATE UNIVERSITY, NSU)**

09.03.01 - Informatics and Computer Engineering

Focus (profile): Software Engineering and Computer Science

TERM PAPER

**Authors**

Rebrin Sergey 23215

Chebotareva Anna 23215

Fitkulin Ildar 23215

Job topic:

# "The Game of TV-Tennis"

Novosibirsk, 2024

## Contents

1. Introduction
2. Problem statement
3. Description of hardware part
    3.1. Display panel, bats, the ball – the game appearance for user
    3.2. Screen
    3.3. 8rows
    3.4. Row-main
    3.5. Color
    3.6. Control
    3.7. Keyboard
    3.8. Lstep
    3.9. Rstep
    3.10. Collision
    3.11. Onplace
    3.12. AI
    3.13. Steps
    3.14. Faults
    3.15. CDM-8 Processor
4. Description of software part
5. User manual
6. Conclusion

## Introduction

Our project is about making a TV-Tennis game by ourselves.

We have implemented a 32x32 video display, a kinematic controller that moves the bats and the ball over the display, also we use a set of 7-segment displays to show the current scores for the two players and the amount of hitted balls in one sequence.

We have 2 modes: the first is a competition between two real players, the second is a competition between real player and computer.

We also have written a program that predicts the ball path through the display and determines position for the computer’s bat to hit it back for using the second mode of our game.

To implement this we used Logisim and assembler (low-level language for CDM-8 Processor). Logisim is a special tool for making circuits, which are essential in the hardware part of a TV-Tennis game. Some circuits work due to the assembler's help. We used assembler to write a program, which predicts ball trajectory for the computer's bat.

Logisim part contains the main circuit – tennis.circ, and ai.circ, klava.circ, CDM-8-mark5-full.circ as essential additional subcircuits.

It is recommended to play with 512 or 256 Hz clock speed.

## Problem statement

Our additions to the basic technical tasks

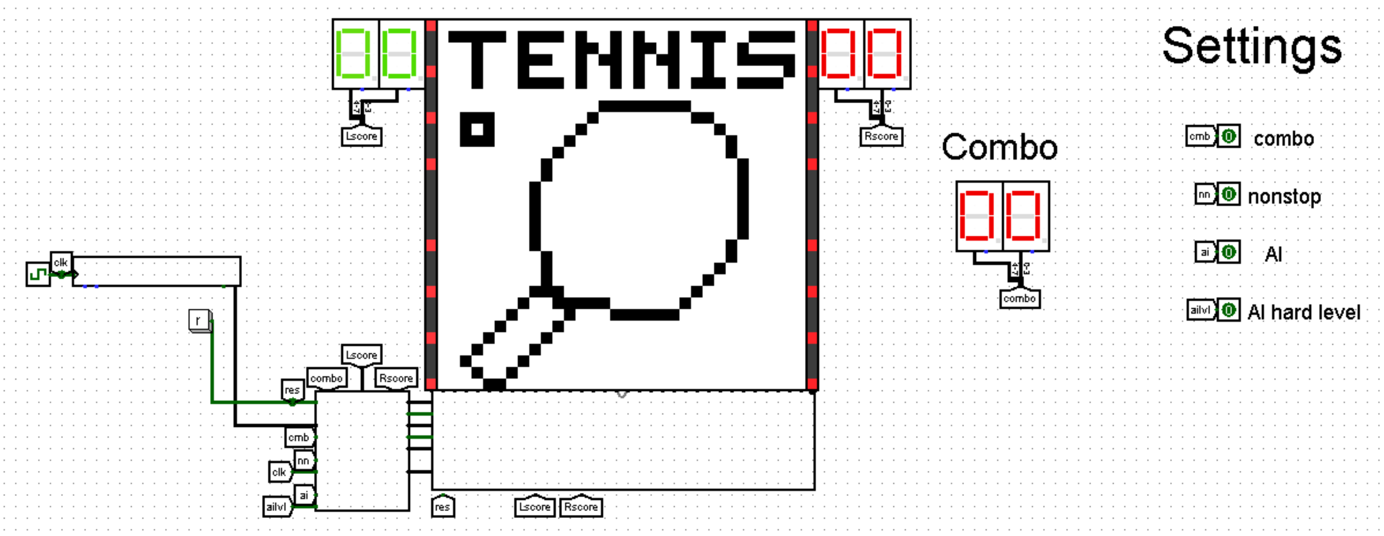
1. 2-player mode
2. Choosing the difficulty for the bat
3. Winning the ball
4. The mechanics of combo

Our plan of work on the project:

1. Set up the screen to display the bat and the ball
2. Configure the movement of the ball and the control of the bat
3. Write code for the CDM8 to control the robot with the second bat
4. Improve the appearance of the circuit and the game

## Description of hardware part

### Display panel, bats, the ball – the game appearance for user



We have the display panel made up of 1024 pixels, arranged into 32 columns of 32 pixels each. The columns are numbered 0..31 from left to right, and the pixels in each column are numbered 0..31 from bottom to top.

During the game process we show 2 bats and a ball against a plain background. The ball is drawn as a single pixel, and each bat is drawn as a vertical line made up of 3 adjacent pixels in a single column. Bats can be only in the 2nd and 29th columns, ball can appear anywhere on the display.

We need to know where the ball and bats are expected to be on the screen. So we need four 5-bit numbers to specify the positions of the ball (ballX, ballY – numbers from 0 to 31, the numbers of the column and the row) and the two bats (leftbatY, rightbatY – the numbers of the row, where the bottom ends of bats are expected to be). This information is calculated in the kinematic controller.

Also we have two columns to the left and right from the main screen, which are just used as decoration.

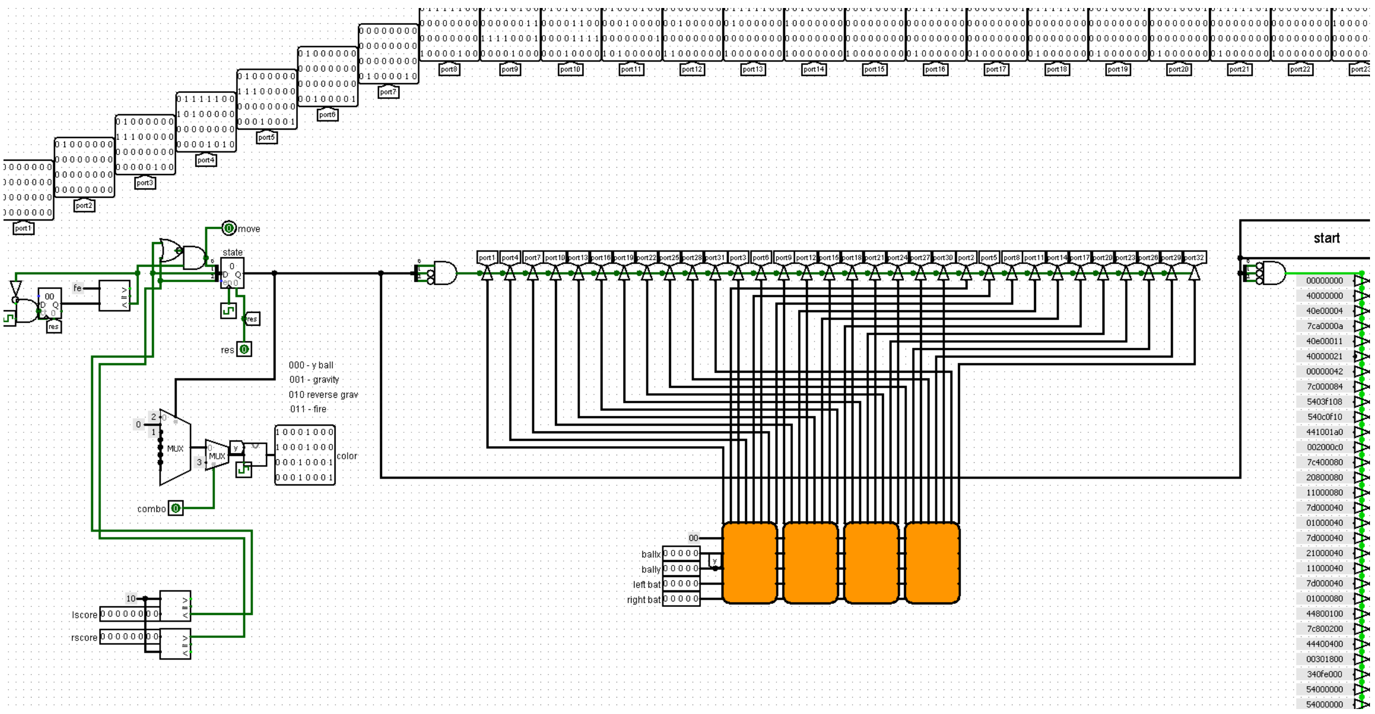
There are two screens (made by two 7-segment displays) for showing a player's score and one for showing “combo” – amount of hitted ball in one sequence before the first goal.

And we have some settings – turning on “combo” (it changes mode when we have some hitted balls in one sequence), “nonstop”-mode – ball bounces from the wall without innings, “AI” – turning on AI (playing with computer mode), and difficult AI-level.

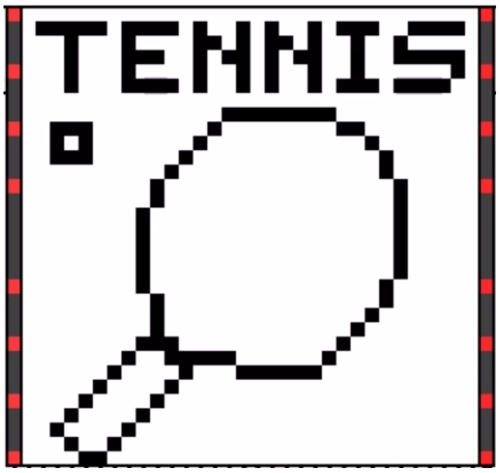
## Screen

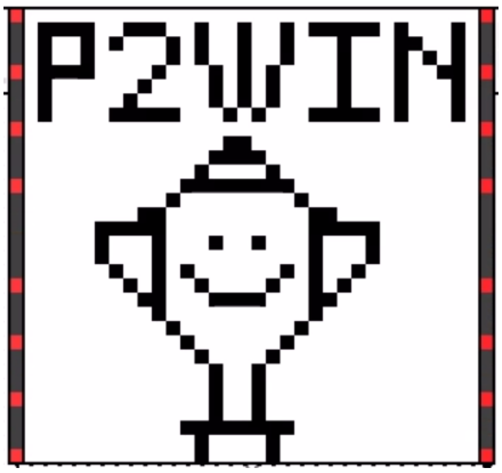
In “screen”-scheme we have 8 inputs:

- 1-bit “reset” input, which means restarting game;
- 1-bit “combo” input, which means the state of “combo”-mode;
- two 8-bits inputs with players score;
- four 5-bits inputs: ball_x, ball_y, left bat coordinate, right bat coordinate. This data is transferred to the video subsystem named “8rows” in our project.



This circuit is the tool for showing all elements like score, display with bats and ball, and start- and end-screens (specified by constants).





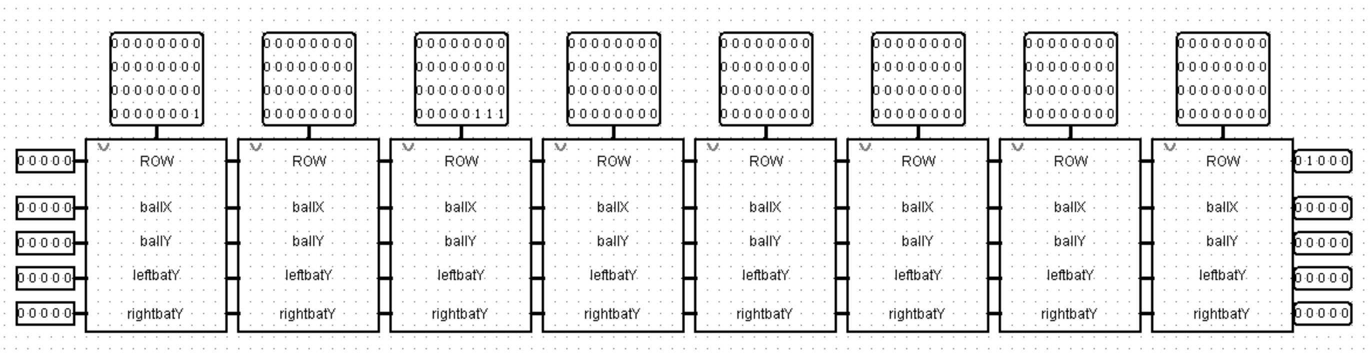
Also here we compare the score with its maximum value to know, do we need to stop bats and ball or not.

## 8rows

This scheme is the video subsystem, it has five 5-bit inputs: ROW(number of the row), ballX, ballY, leftbatY, rightbatY. The display panel is powered by a group of 32 video chips (named "row-main"), with all five of these inputs linked to each video chip. To minimize wiring, the video chips are interconnected in a chain configuration. The chips are positioned in a chain from West to East, where the chip on the western end receives all input data and forwards it to the next chip in the sequence.

Instead of chaining 32 individual video 8 chips are chained together to make the circuit easier. 4 of these sections are then chained in "screen".

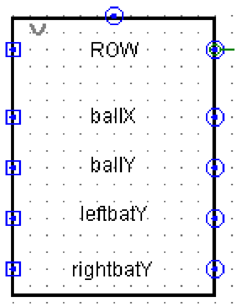
Each section is indicated in the circuit diagram by a rectangle containing text values of corresponding data.



## Row-main (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, all of this data pass through chips unchanged.

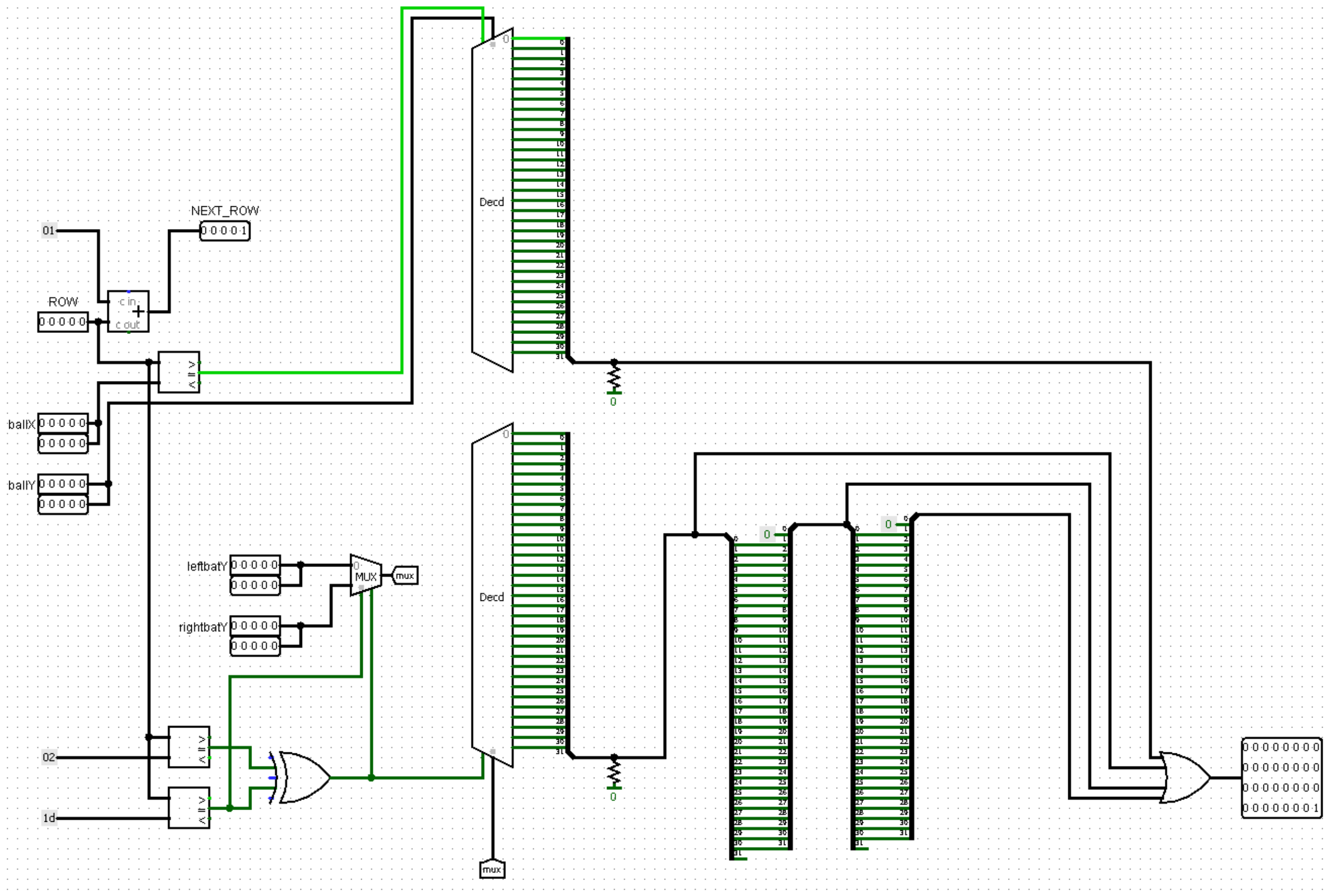
Each video chip drives a specific column of the display panel, so each chip needs to 'know' which column it is responsible for.



Each video chip has an additional 5-bit input pin labeled ROW (number of columns) on its West side, and an additional corresponding output pin on its East side. Value of ROW increases by 1 passing through the chip. We pass 0b00000 to the first chip like a constant for being sure that all chips will drive the correct column. This way all 32 chips number themselves automatically.

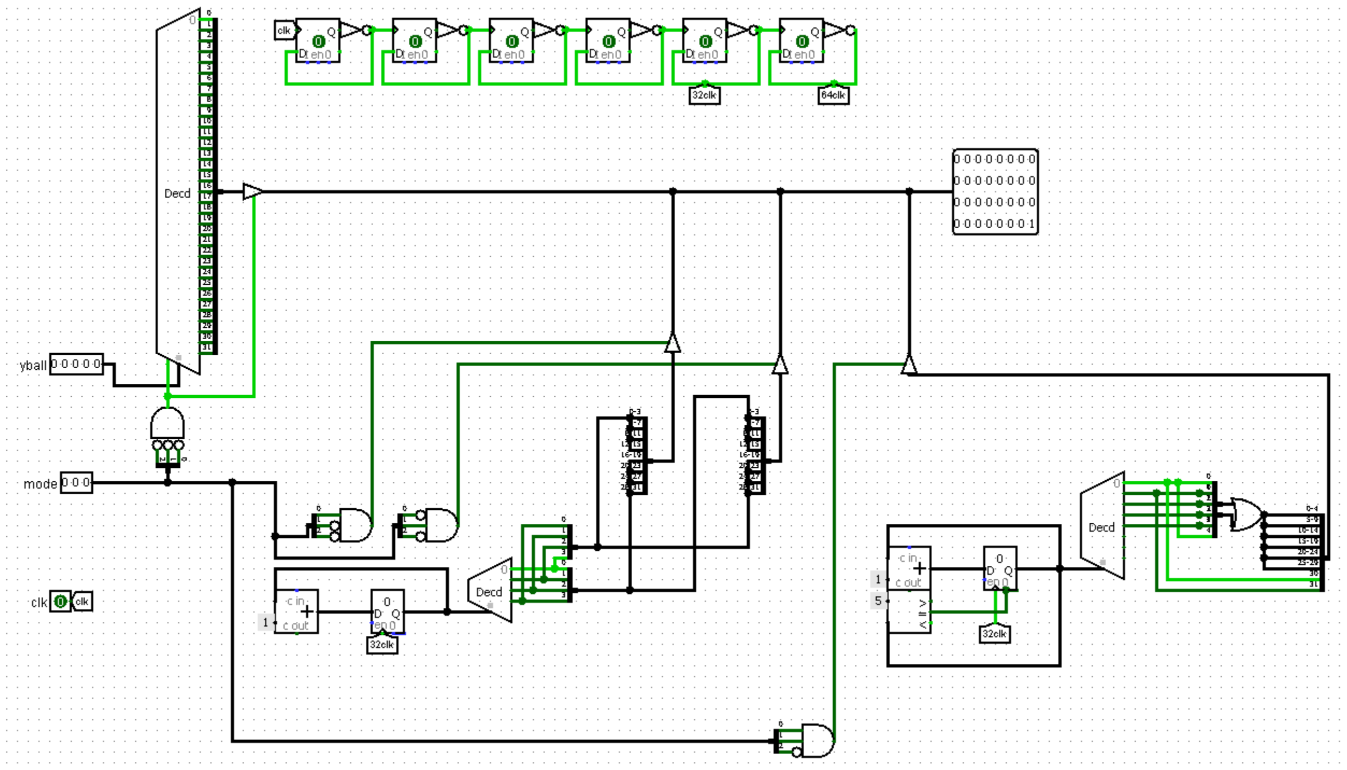
Each chip has the possibility to display a ball, and the possibility to show a single bat. The ball's position is received on input pins ballX and ballY. Each chip compares its chipID to ballX, and if they are equal it lights up pixel number ballY in its column of the display.

The bats appear only in fixed columns, so we remember their column numbers (2 and 29) in every chip as two 5-bit constants: 0b00010 and 0b11101. These two constants are compared with the ROW, so if the video chip that has 00010 as its ROW it shows the left bat with its bottom pixel in row leftbatY, and the video chip that has 11101 as its ROW shows the right bat with its bottom pixel in row rightbatY.



Of course no chip will ever show both bats, but the ball can be in the same columns with bats. So it is easier to keep information about the ball and both bats in every scheme.

## Color



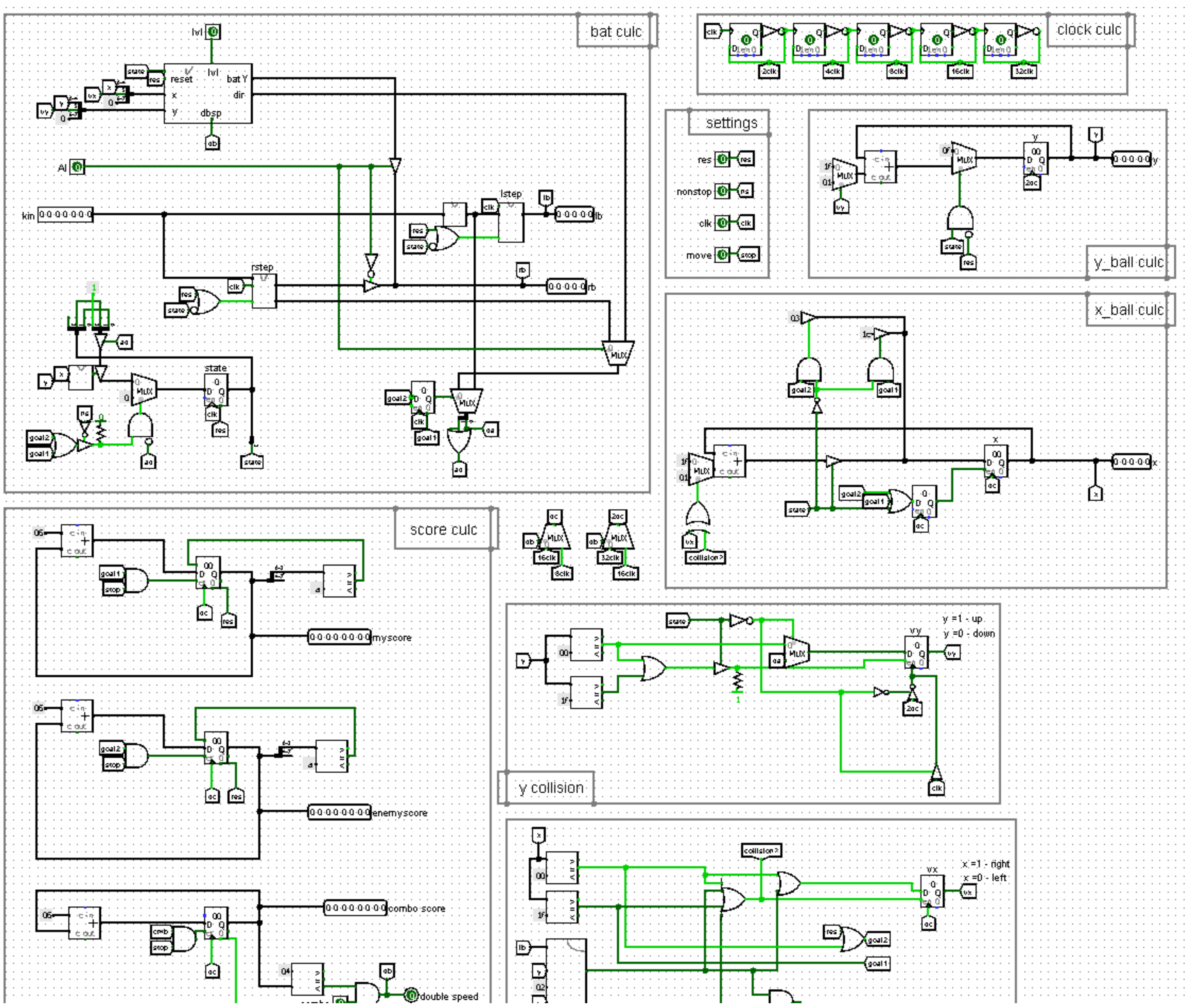
This circuit is implemented for two columns on the left and right sides from the main screen, where we have one or more red pixels running through those columns. When “combo”-mode is turned off and when we haven’t enough hitted balls in one sequence we have only one red pixel which follows the ball y-coordinate (one of the inputs).

When combo-mode is turned on and we have at least five hitted balls in one sequence we have more red pixels running faster than in normal-mode.

## Control

This circuit is the main circuit in the kinematic part. Here we have calculation of bats coordinates (bat culc), ball coordinates (y_ball culc, x_ball culc), auxiliary speed calculation (clock culc), also we have score calculation (score culc), and checking ball collisions with bats and walls (y collision and x collision).

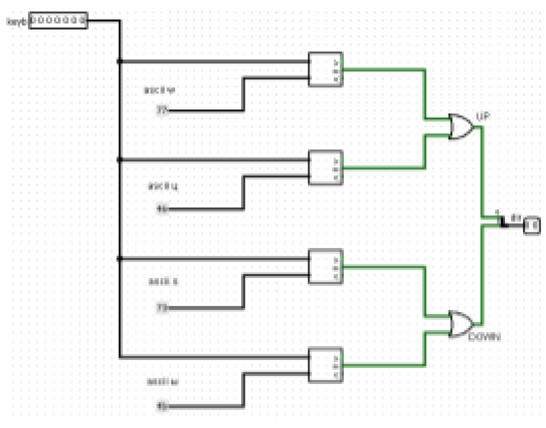
The function of this part is calculating coordinates of each object and score calculation at any moment. This circuit has 8 inputs: reset, nonstop-mode, AI-mode and its level, clock, combo-mode, data from keyboard and “move” – it is responsible for the state of moving (1 – all objects stop, 0 – move).



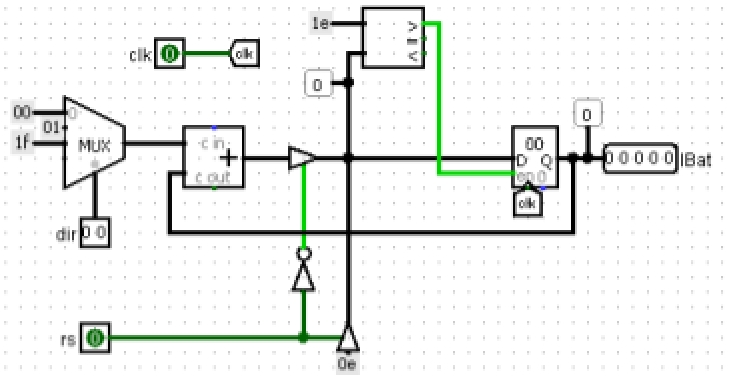
*Clock culc:* just using d-triggers to make the clock slower by each step.

*Settings:* just saving input information from settings.

*Bat culc:* in this part of the circuit we process data from the keyboard (by the circuit “keyboard” for left but and rstep for right bat) to understand the direction of the bats moving. It allows moving both bats, but if AI-mode is turned on, we use AI to calculate right bat coordinates and move it. So here we use lstep and rstep for gradually changing bats positions.

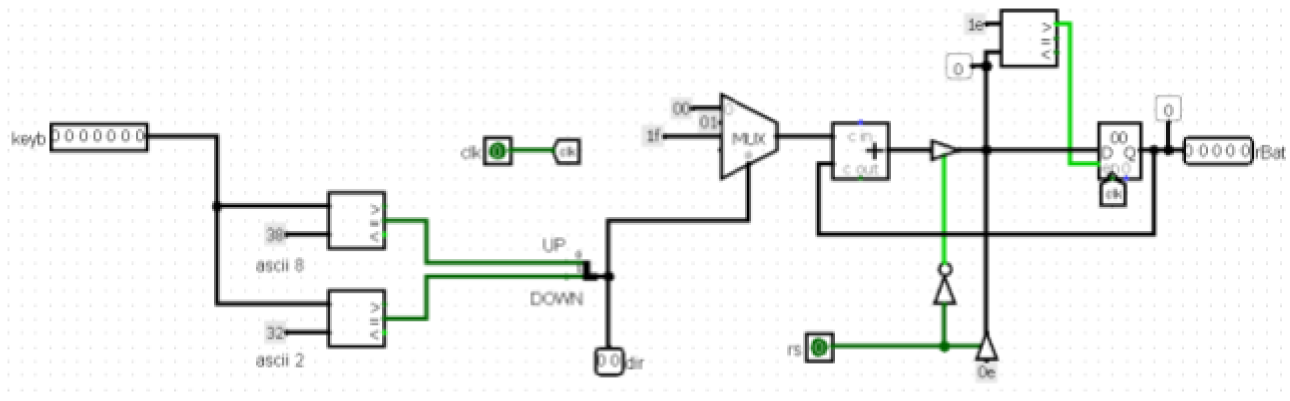


keyboard



lstep

So processing data from the keyboard is just comparing the ASCII-value of the entered character with needed ASCII-value. “w” (or “ц”) is for moving the left bat up, “s” (or “ы”) is for moving the left bat down. “2” and “8” are the same for the right bat. Then “lstep” and “rstep” save current bats positions in registers and change them by 1 (increase or decrease) depending on the data from the keyboard, then pass this information in “control”.



step

Also here we make “innings”-positions (after goals, if nonstop-mode is turned off, the ball is given to the player who missed the ball).

*y_ball culc:* works like “lstep”, but taking into account the state of combo-mode (ball can change its position faster).

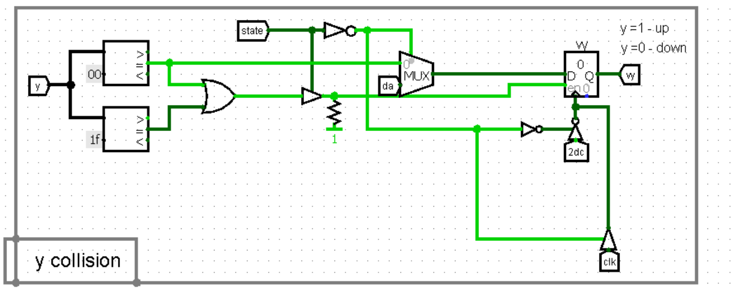
*x_ball culc:* works like “y_ball culc”, but also it sends the ball to the 3rd to 28th columns after goals.

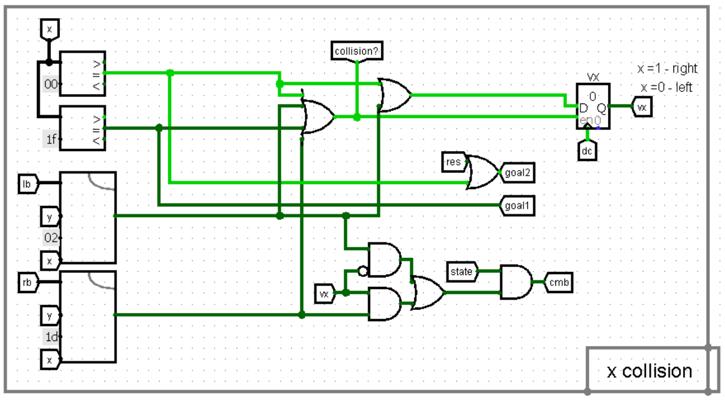
These circuits make the ball move after goals the next way: ball waits when the nearest bat moves and then ball moves in the same direction as the bat.

*score culc:* saving current score and increasing it by one after goals, or resets to zero after restarting the game.

*y collision:* check collisions with “roof” and “floor” by comparing ball y-coordinate with 0 and 31 and change ball moving direction (1 – moving up, 0 – moving down)

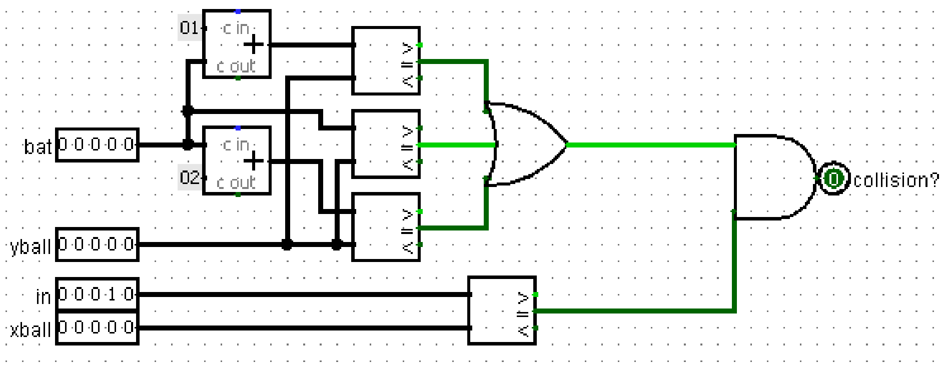
*x collision:* check collisions with bats and walls. Collisions with walls are checked by comparing the ball x-coordinate with 0 and 31.





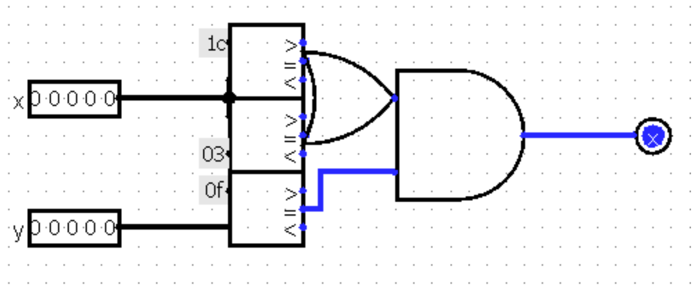
For checking collisions with bats we have an auxiliary scheme “collision” with 4 input pins: bat coordinate, ball y and x coordinates, and the number of the column necessary for checking bat. It calculates coordinates of all of 3 bat pixels and compares them with ball coordinates.

If there is a collision with walls it will be perceived as a goal and if nonstop-mode is turned on ball will just reflect as with bats collision, but if nonstop-mode is turned off then the ball will stand near to the bat which has missed the ball.



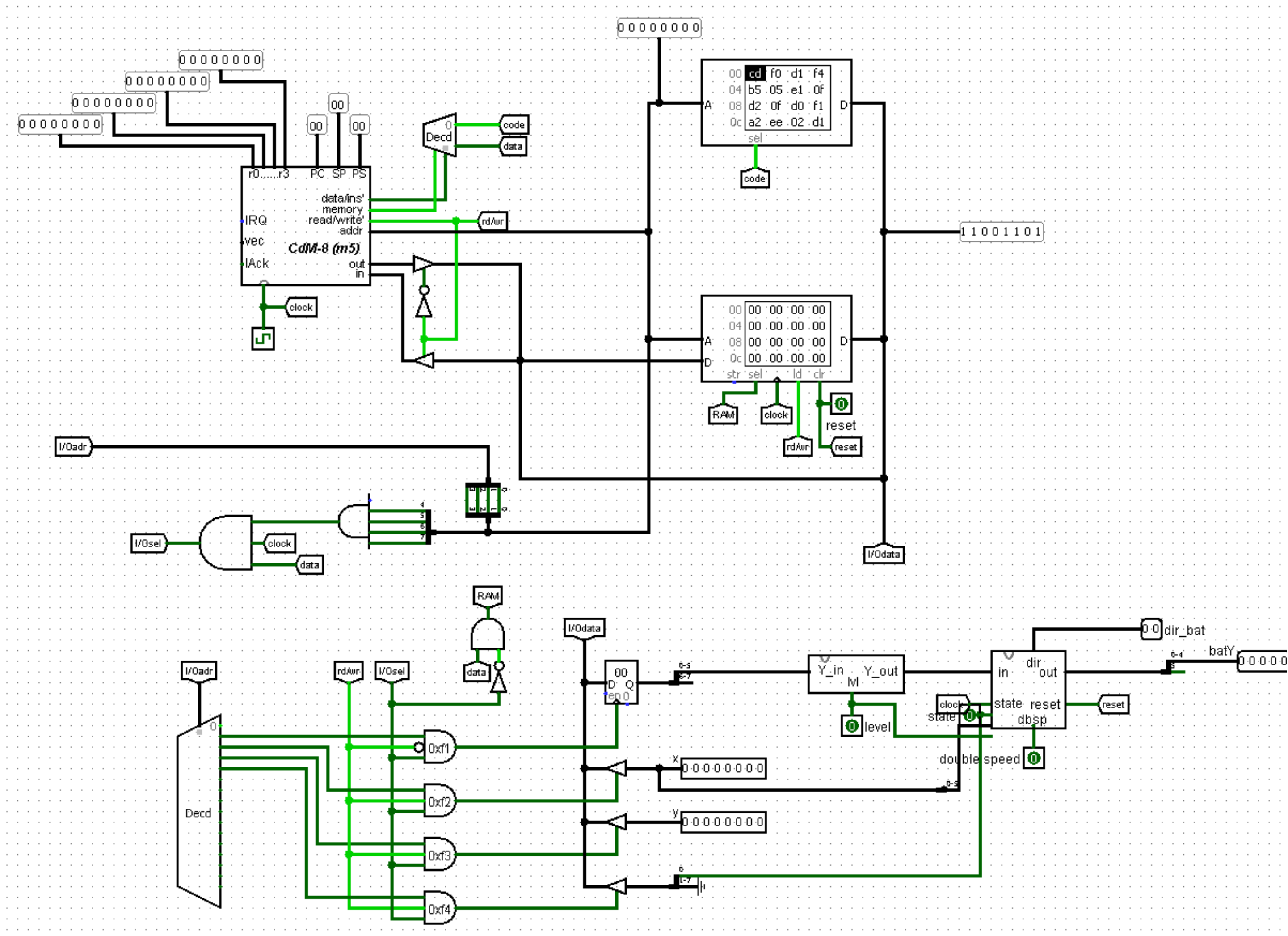
## Onplace

This is the additional scheme made in order to check that innings work as needed. Because sometimes they can just refuse to work, so we check ball position again.

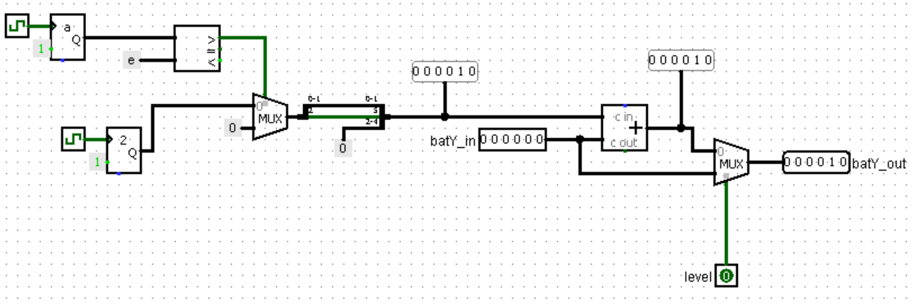


## AI

Here is the main AI-scheme. We use CDM-8 Processor with a Harvard-architecture. ROM is for code, RAM is for data. Description of code is in this part. It sets the result (predicted y-coordinate for the right bat) in 0xf1 address and takes x and y ball coordinates in 0xf2 and 0xf3. Also it takes the “state” input in 0xf4.

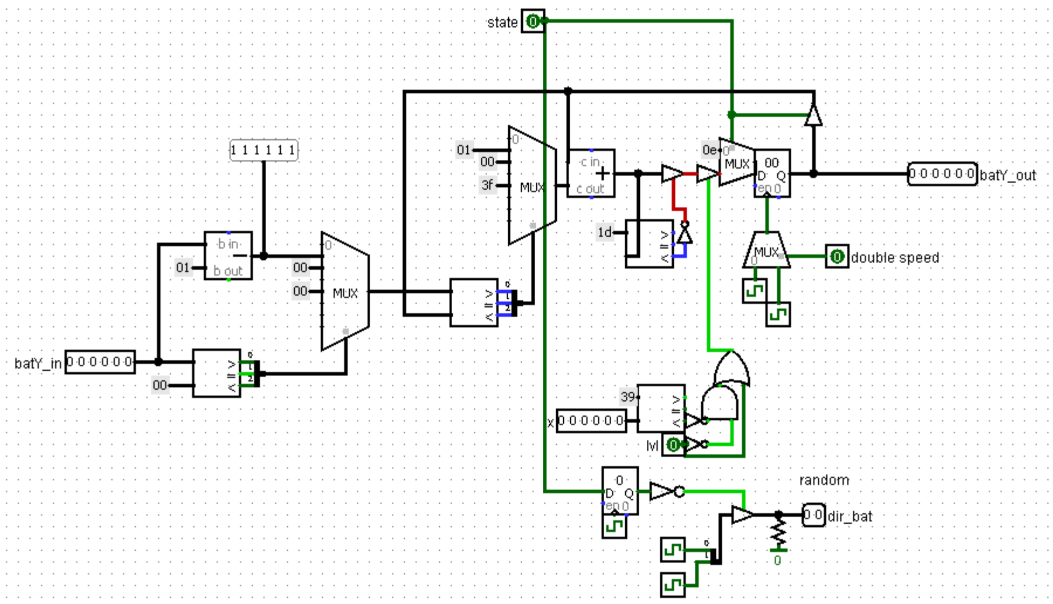


Then the result is remembered in the register and sent to the “faults”-circuit. It has 2 input pins (real predicted bat y-coordinate and level of the AI).



This scheme (“faults”) makes AI a little bit silly if hard-level is turned off (in the other case coordinates stay unchanged). It generates random faults using a random number generator with a probability of slightly less than 50 percent and calculates new “predicted” y coordinates for the right bat.

Then the new coordinates are passed into the “steps”-scheme. It has 3 input pins: state of bat, its predicted coordinates and the state of combo-mode. Output pins: direction of bat and its new coordinates. This circuit is intended for gradually bat moving to the correct position because without this tool the right bat will move straight away to the correct position without pretending to move gradually.



This circuit works like "Istep" but with changing bat speed in combo-mode, because in other case the right bat will be too slow.

However, here is one more difference: in every moment we compare the real right bat position with the necessary position. If they are not equal, we increase or decrease the real position by 1 depending on the result of comparing.

Also we have to check our new "predicted" coordinates after adding "faults" to see if they are in the range 0-31 to avoid a situation when the bat is higher than the "roof" or below the "floor".

## CDM-8 Processor

CDM-8 Processor is the special tool for us to learn how to work with processors and e.t.c. We use a Harvard-architecture, which uses two separate memories, one for program code and one for data. Each memory has its own addresses space. Because of this we can process data faster due to using both memories simultaneously. A Harvard architecture requires the CPU to indicate which memory (code or data) it needs to use in each clock cycle. We may use 512 bytes of memory for larger system designs, one half for code and the other half for data.

## Description of software part

```
asect 0x00
setsp 0xf0
L4:
ldi r1,0xf4
ld r1,r1
if
     tst r1
is z
     ldi r2, 15
     ldi r0, 0xf1
     st r0, r2
     br L4
fi
ldi r1, 0xf2
ldi r2, 0xf3
ld r1, r1
ld r2, r2
move r1, r3
ldi r0, 0x1f
and r0, r1
ldi r0, 0x20
and r0, r3

if
     tst r3
is nz              # beq 0x3b
     move r2, r3
     ldi r0, 0x1f
     and r0, r2
     ldi r0, 0x20
     and r0, r3
     if
          tst r3
     is nz              # beq 0x2e
          ldi r0, 0x1d
          sub r1, r0
          neg r0
          shra r0
          ldi r3, 0x1d
          add r0, r2
          if
               cmp r3, r2
          is lt              # bge 0x2c
               sub r2, r3
               ldi r2, 0x1d
               sub r2, r3
               move r2, r3
          fi                  # br 0x3b
     else
          ldi r0, 0x1d
          sub r1, r0
          neg r0
          shra r0
          if
               cmp r2, r0                # bge 0x39
          is lt
               sub r0, r2                  # br 0x3b
          else
               sub r2, r0
               move r0, r2
          fi
     fi
     else
          ldi r2, 15
          ldi r0, 0xf1
          st r0, r2
          br L4
fi
  ldi r0, 0xf1
  st r0, r2
  br L4                    # br 0x2
  halt
end
```

This part is about writing the code which predicts ball trajectory for the computer's (the right) bat.

Firstly we check the bats position by setting the “state” in 0xf4 address and checking if it is 0 or 1. If it is 0 we set the center of the screen as necessary right bat position to avoid its random strange movements.

Then we set x and y ball coordinates in 0xf2 and 0xf3 addresses. Bits from 0 to 4 are describing x-coordinate, the 5th bit describes the direction of the ball (for x: 0 – left, 1 – right; for y: 0 – down, 1 – up).

Then we separate these parts: direction and real coordinates.

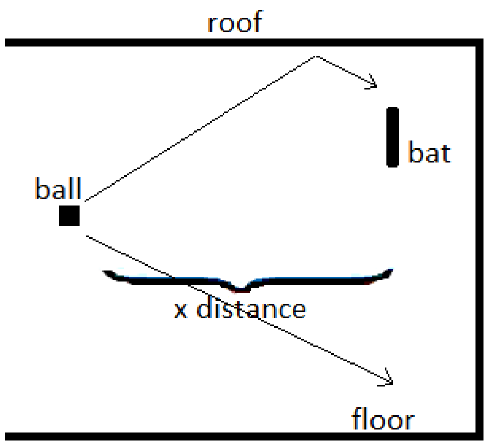
Then all of the instructions are for the case when the ball runs to the right side. We separate direction and real y-coordinate and check the y-direction.

We find the distance between the ball and the right bat, then we divide it by 2 to find the distance which ball runs before the collision with the right bat (because y coordinate changes by 1 pixel at the time when x coordinate changes by 2 pixels).

Then we check the collisions with the “roof” and the “floor” (depending on the ball direction) and find the real coordinate where the ball will be.

If the ball runs to the left side we set the center of the screen as “predicted position”.

Then we set the value which we have found in the 0xf1 address and that’s all.



## User Manual

After downloading the folder with our project, open the tennis.circ file. The main circuit open in front of you. In the upper panel, in the "Simulate" tab, hover over the "Tick frequency" item and select 512 Hz.

In the center of the screen there is a display on which the game will be shown. On the right side we have a window with settings that you can customize as you wish. To the left of the display is the game control. Click on the input field. Press ctrl + K to start the game.

The left bat is controlled by the "w" and "s" keys. If the 2-player mode is enabled, the right bat is controlled using the "2" and "8" keys. We do not recommend holding down the key in this mode.

If the bats are in inning mode, then move up or down to select the direction of the ball.

There is an "r" button at the bottom of the input field. Click on it to start the game again.

## Conclusion

We have implemented a “TV-Tennis game”. It was a really interesting job. We have put our knowledge into practice, learned new information and finally we have a ready project.

We used Assembler and Logisim in our project. It is a really useful and great experience for understanding computers and low-level programming. It’s an incredibly cool feeling when you realize that you understand how to do this or that thing.

So now we have a “Tennis game” with different modes, and we did it by ourselves. Its circuit has 15 subcircuits and the code has 73 lines. We think that our version of the game makes it much more interesting than the classic version.