MINISTRY OF SCIENCE AND HIGHER EDUCATION OF THE RUSSIAN FEDERATION
FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION
"NOVOSIBIRSK NATIONAL RESEARCH STATE UNIVERSITY"
(NOVOSIBIRSK STATE UNIVERSITY, NSU)

09.03.01 - Informatics and Computer Engineering
Focus (profile): Software Engineering and Computer Science

TERM PAPER

Job topic: **‘POKER’**

Kuzminov Maksim
Chepik Andrew
Baev Michael

Novosibirsk
2024

1 Terms and abbreviations

“fold” command	The "fold" command means that the player or bot has surrendered.
“raise x” command	The command "raise x" means that the player or bot has raised its bid by x.
“call” command	The "call" command means that the player or bot has raised its bid to the opponent's level.
“check” command	The "check" command means that the player or bot continues to play the game without raising the bid.
“VAR_NAME(ADDRESS)”	If you see such a definition (e.g. SUIT_VALUE(0xfe00)), here is what it means: the name of a variable or array in the “start.asm” file is written in big letters, and its address in memory is written in brackets.
(WORD)	The word capitalized and bolded in brackets means a reference to this object. This means that in the future we can refer to this object via "**WORD**".

2 Introduction

Our project called "Poker" is based on the card game of the same name. We came up with our own simplified version of poker based on "five-card draw". The player and the bot named Konstantin each have 5 cards, 3 of which are visible at the beginning of the round. Depending on the cards, decisions have to be made ("fold", "raise", "check" and "call"). The one who has the strongest combination in the round takes the whole amount from the bid to himself. The one who has the highest balance at the end of the game wins.

The hardware part of our project is implemented using the "cdm 16" processor, which we studied during the Digital Platforms course.

The interface of our program is displayed in the terminal using the telnet protocol. You can learn more about this and the rules of the game in the technical specification.

3 Purpose and area of application

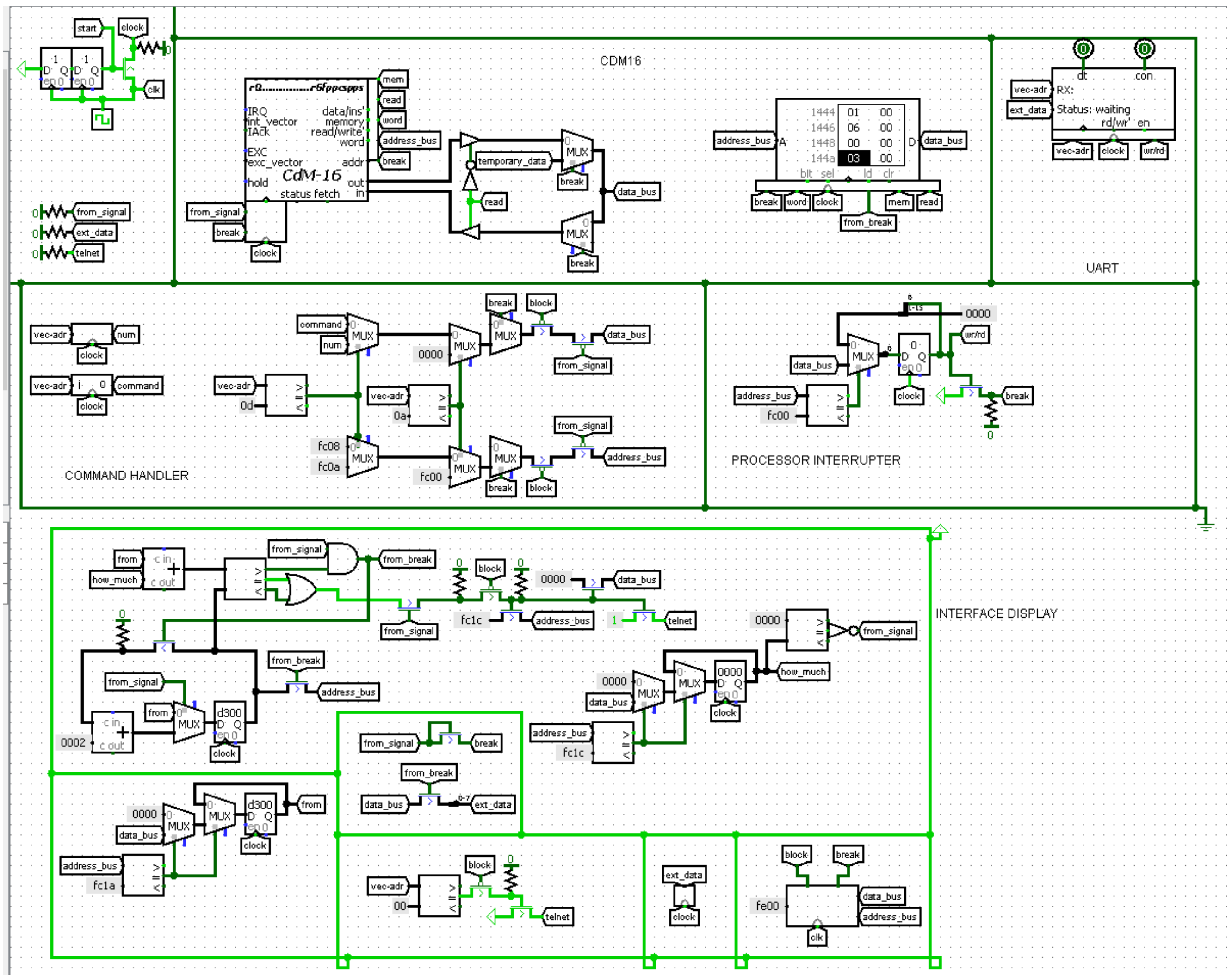
The purpose:

Our application is designed to provide the user with the ability to play poker in a virtual environment.

Area of application:

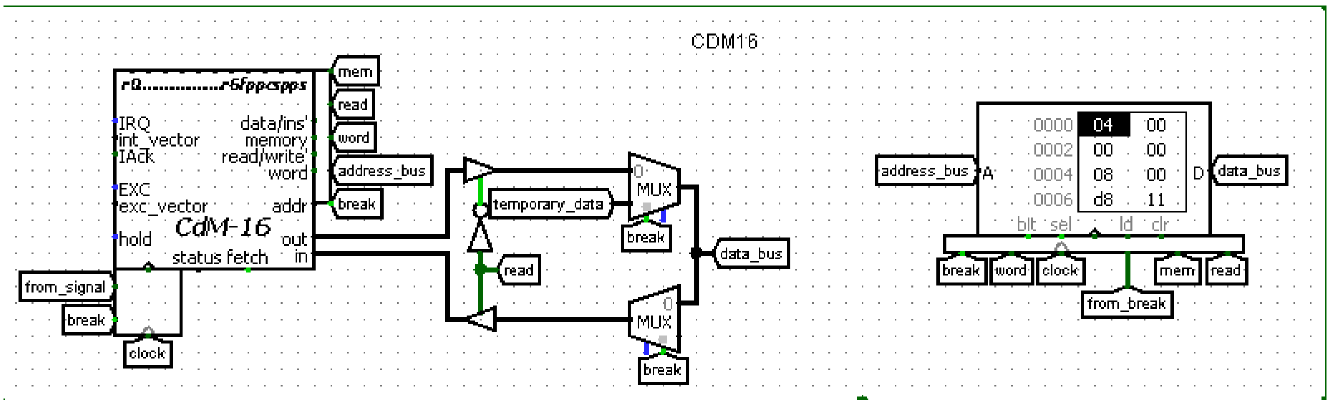
Application is designed for a wide range of users interested in the game of poker. This can range from beginners who want to learn the rules of the game to experienced players. Also our app can be used to teach poker strategy and analyze the game.

4 Technical characteristics



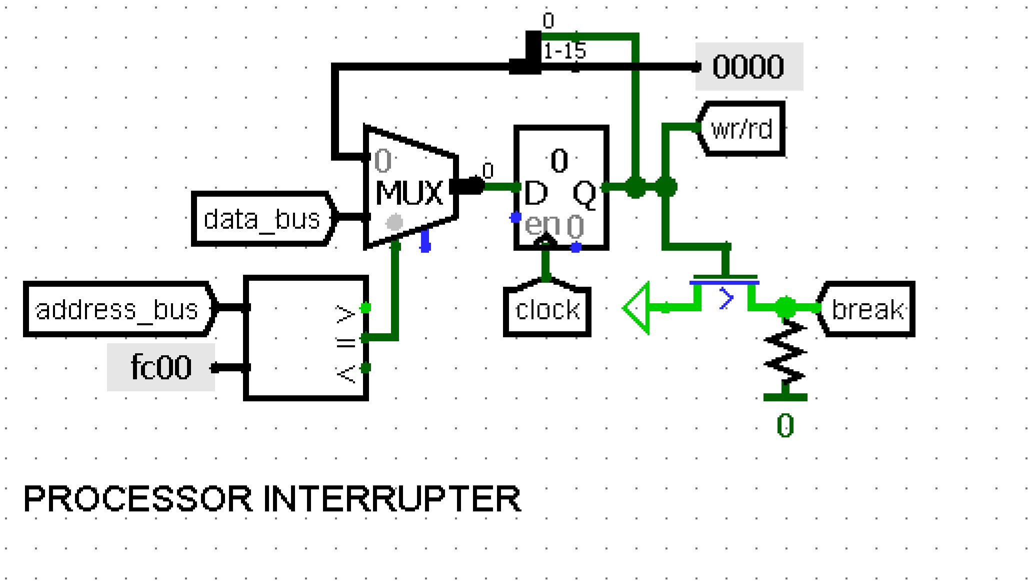
Now you can see a general view of the circuit. Let's see how it works in detail.

1. CDM-16



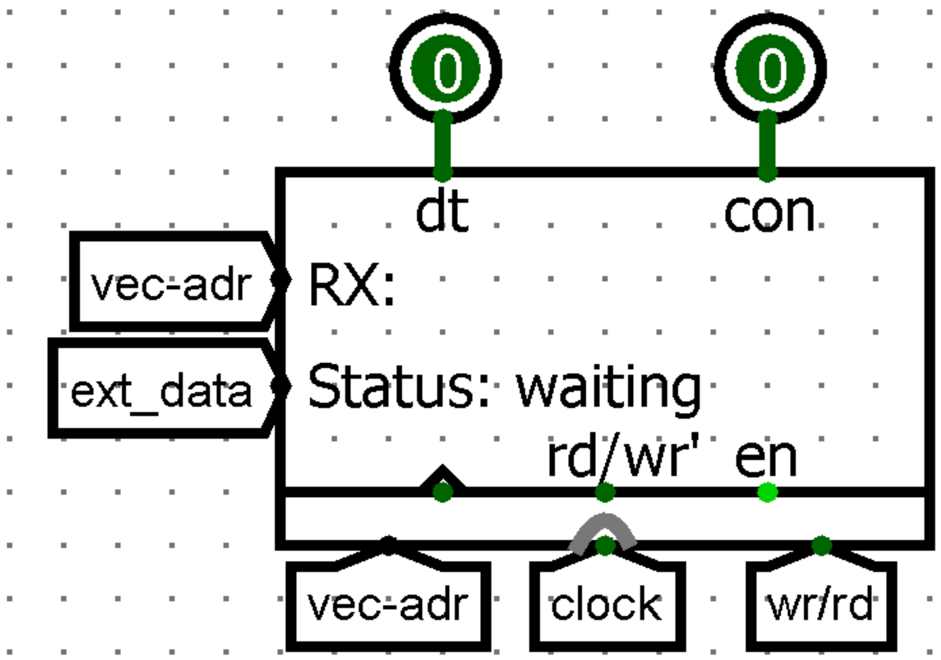
We are doing this project with a cdm16 processor using Von Neumann architecture. We chose it because it is much more user friendly than the cdm8/8e. Also on cdm16 we have the ability to write code using C language and compile it for cdm16 architecture.

2. PROCESSOR INTERRUPTER



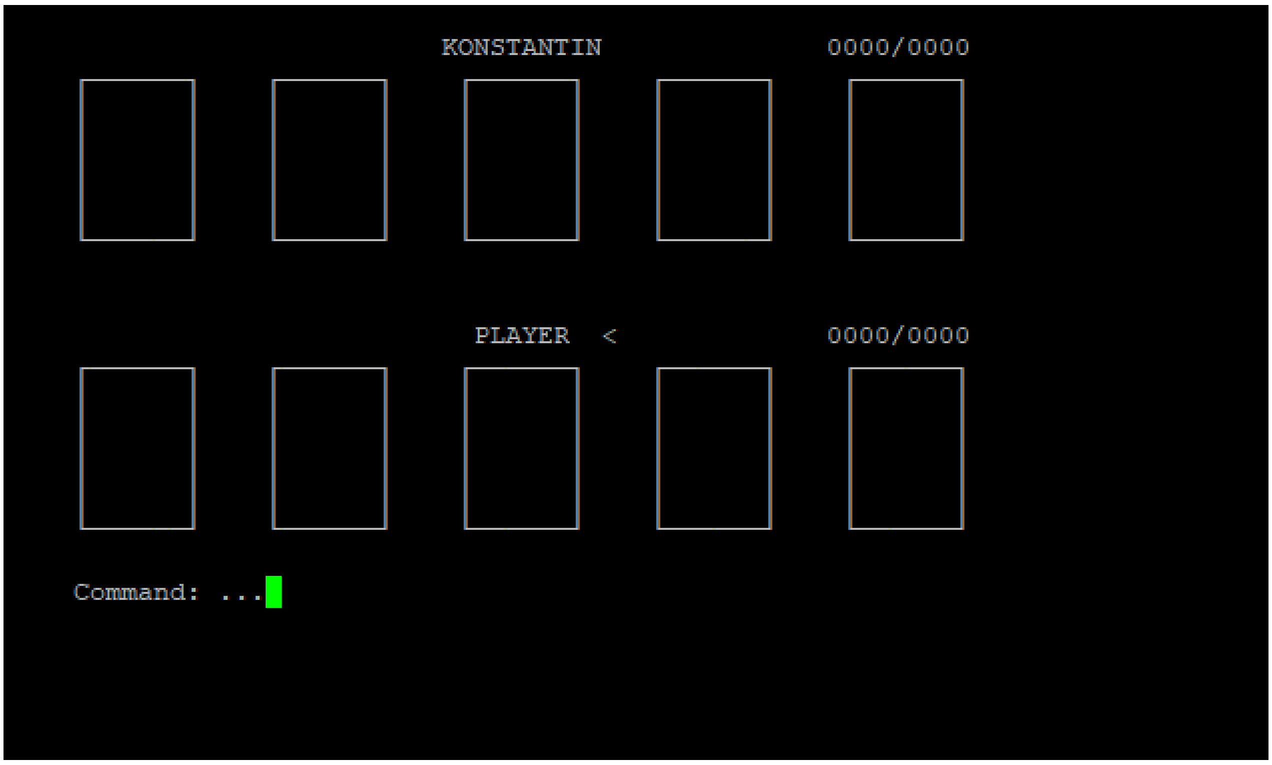
This part of our circuitry is responsible for stopping the processor. More specifically, when we set RD_WR(0xfc00) to 1 in our program, we stop the clock access to the processor.

3. UART



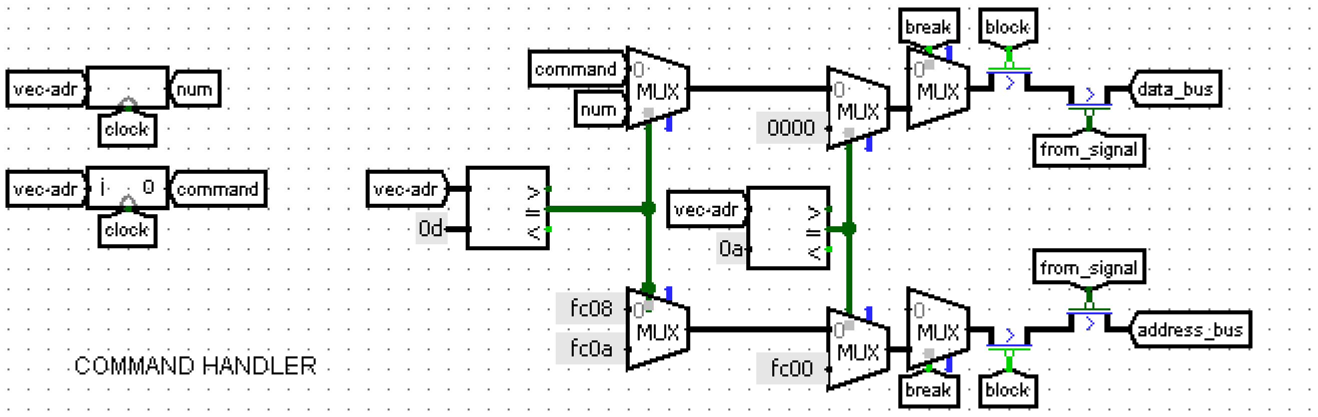
What do we need to see on the terminal? Five cards of the bot Konstantin (AI) and five cards of the player. And also a line into which the player will then enter commands indicating his decisions. Our terminal is based on the Telnet network protocol, with which our circuit interacts through the Logisim UART library (cdm-processors/logisim-uart: General purpose asynchronous buffered receiver/transmitter. (github.com)). The part of the circuit responsible for Telnet communication is shown above. To connect to our game you must type "127.0.0.1 25" into a telnet terminal

Thanks to this, we can now implement the output to the terminal of the interface described above:

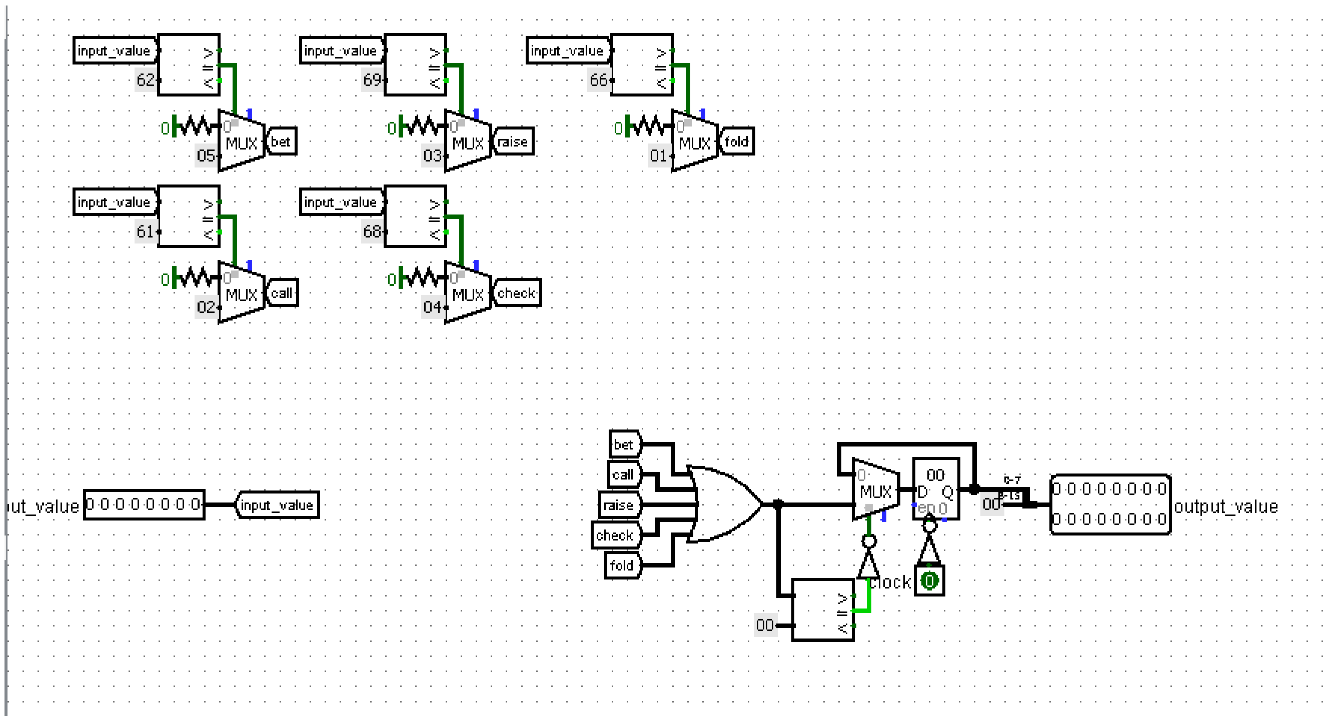


```
                    KONSTANTIN              0000/0000

                     PLAYER  <              0000/0000

Command: ...
```

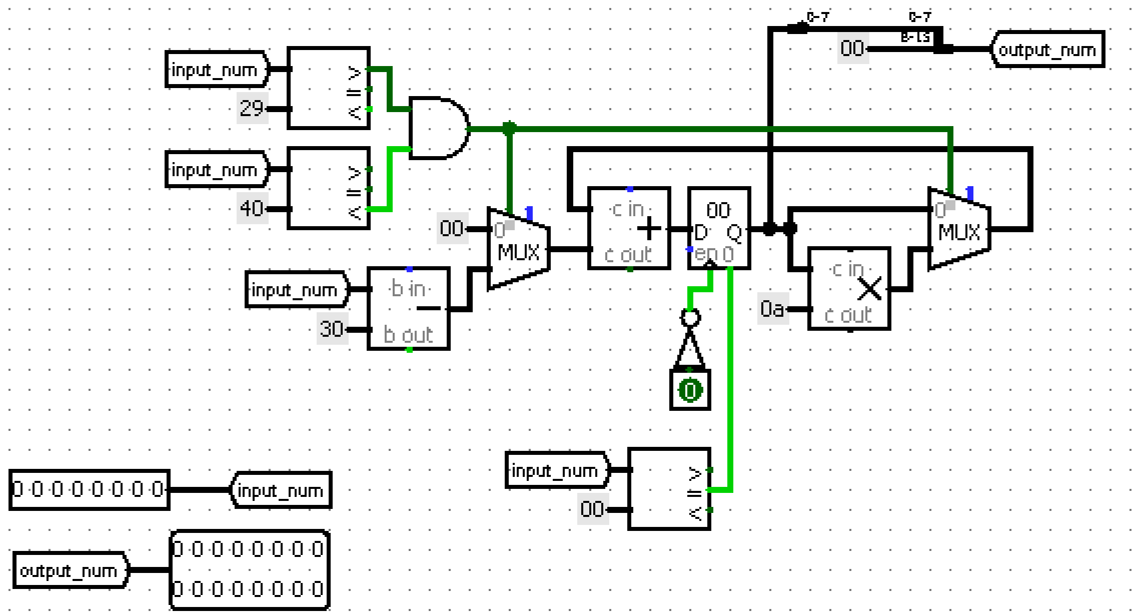
4. COMMAND HANDLER



Main image of Command Handler **(CHM)**



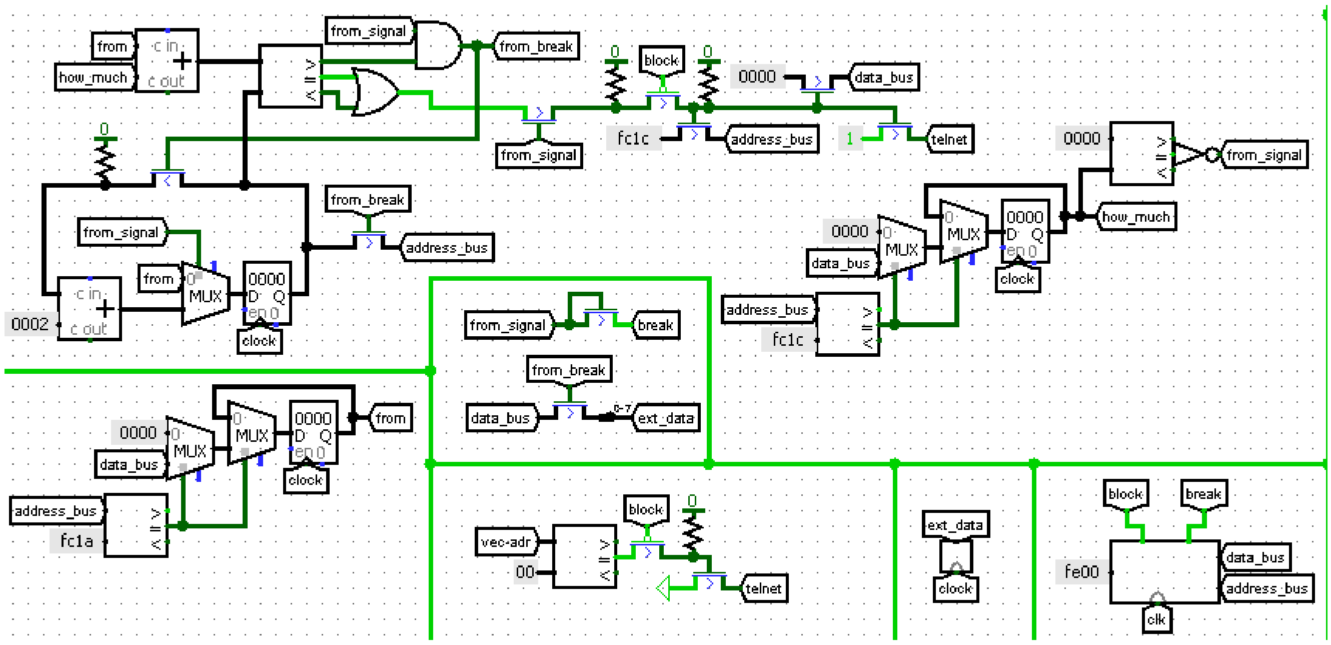
The "Commands" subcircuit located on the CHM on the top left



The "Bid" subcircuit located under the "Commands" subcircuit on the CHM

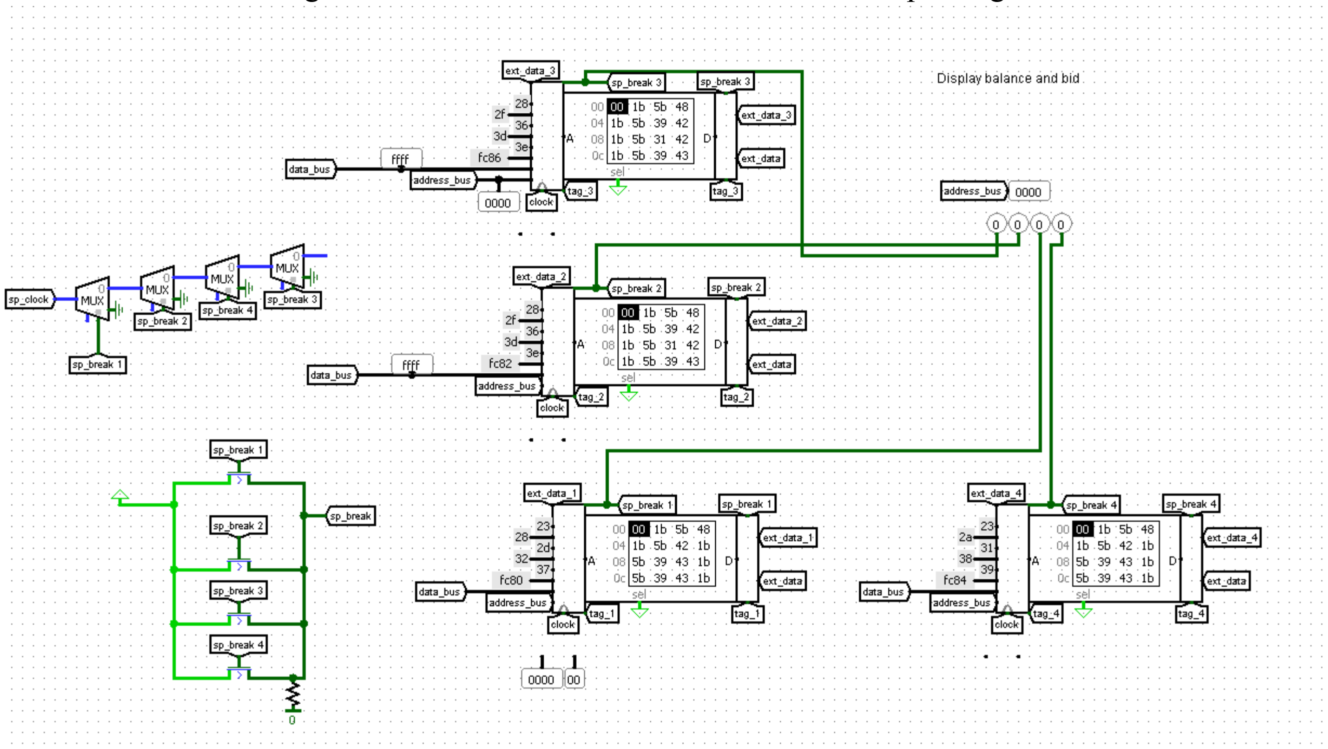
It reads the input data you have entered into the terminal and writes the numeric value to the COMMAND(0xfc08) and BID_COMMAND(0xfc0a) memory cells. A person can enter a total of 4 commands: "fold", "raise x", "check", "call", where x is a number from 0 to 20 inclusive. If incorrect data has been entered, the program will ask to enter it again.

5. INTERFACE DISPLAY

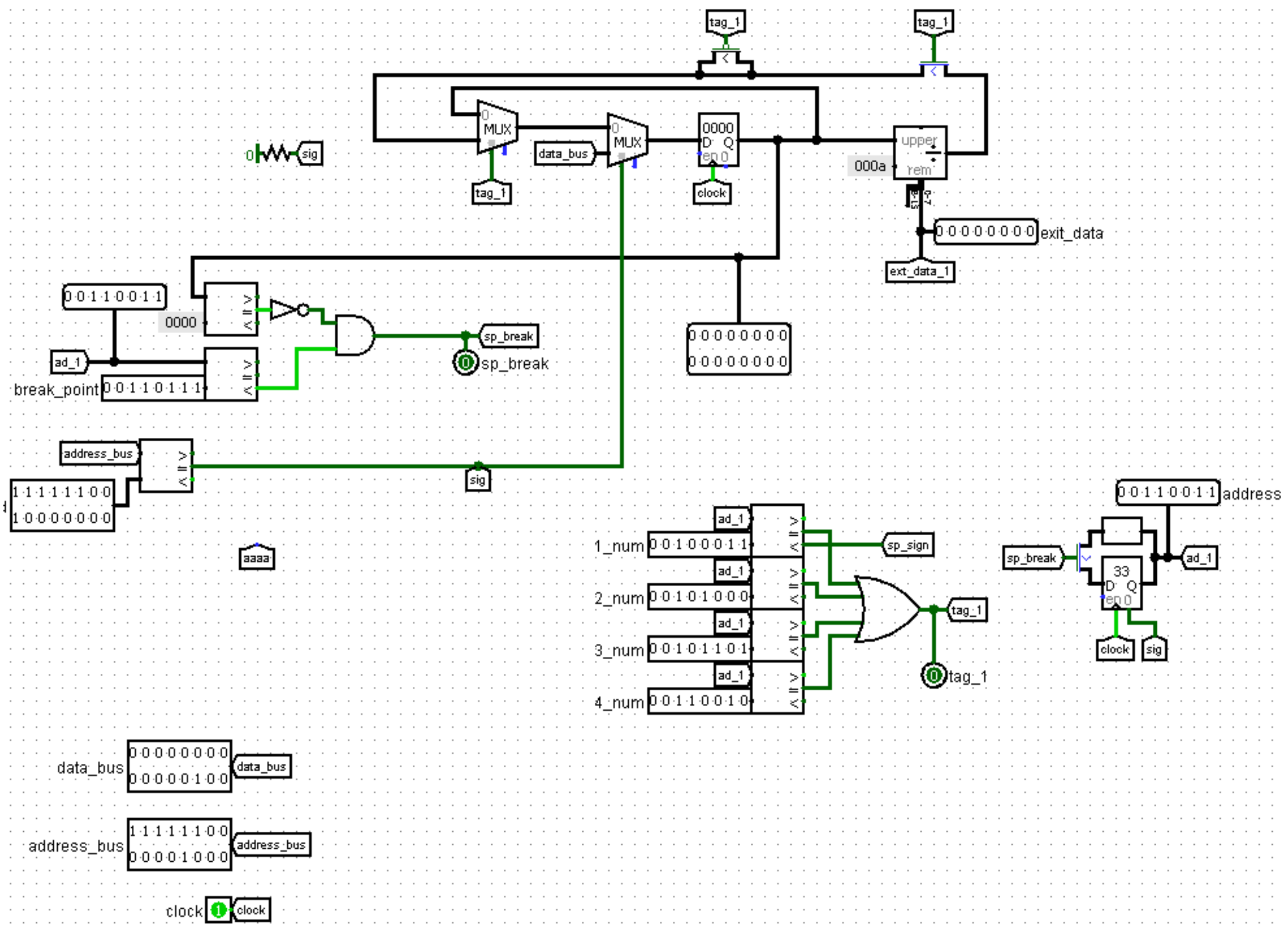


We also need to output values such as the values and suits of cards, whose turn it is, who won, etc. To do this in the program we write in one memory cell SEQUENCE_PTR(0xfc1a) the pointer to the sequence in memory that we want to output, and in another SEQUENCE_LEN(0xfc1c) - its length, then the processor stops and waits for the output to finish. After the output the processor starts its work again.

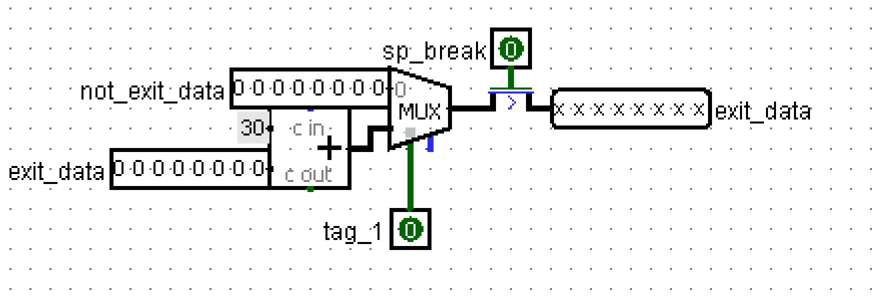
We also need to go to the "interface" subcircuit and set the "output.img" file in ROM.



Main image of the circuit for balance and bid output **(BBM)**



Subcircuit "print num" located to the left of the ROMs on BBM

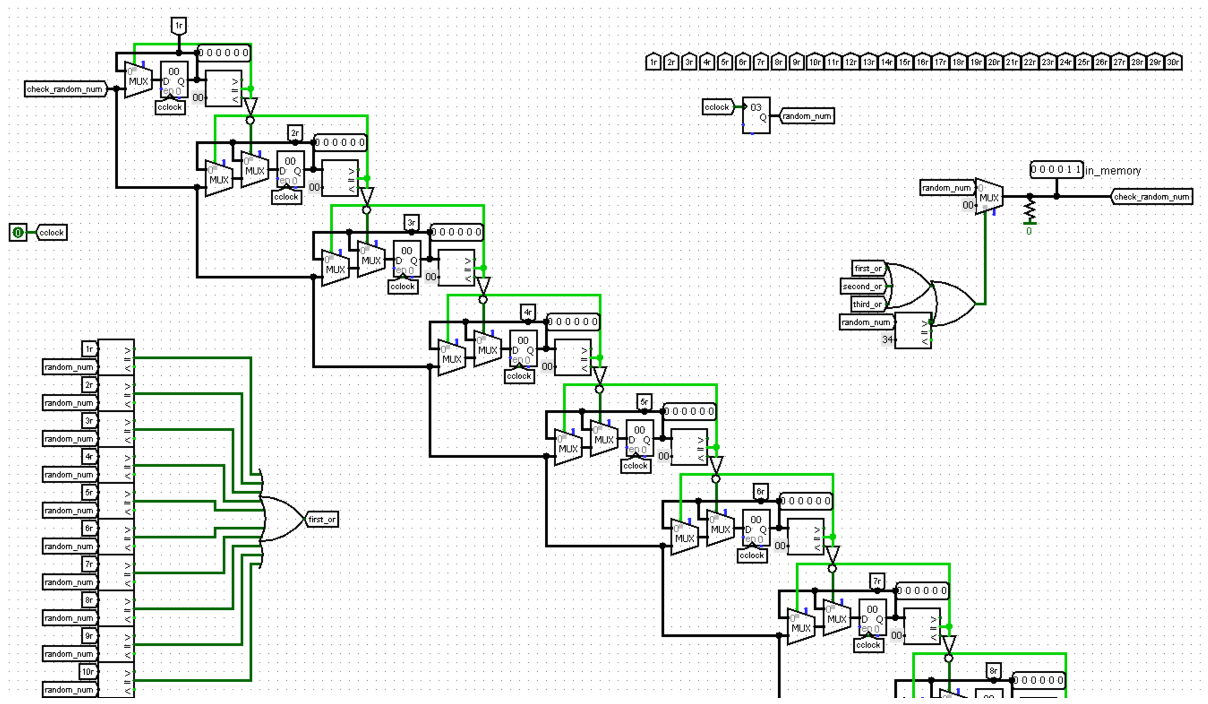


Subcircuit "again_print_num" located to the right of the ROMs on BBM

This part of the circuit is for displaying the balance and the bid. It takes the balance from memory and displays it by dividing by 10; The interaction with this subcircuit will be described a little later.

6. RANDOM CARDS

The cards in the hands of the player and bot are created using a random number generator. At the next generation the number is written to the corresponding register; there are 30 registers in total, since we have only 3 rounds, in each of which the player and the bot have a total of 10 cards in their hands. Checking the generated number is quite simple: whether the given number lies in the desired interval [1; 52] and whether the same number has been generated before or not. Then it inserts them into the array beginning in SUIT_VALUE(0xfe00).



Part of the circuit "Random_numbers". *Bottom left: check for coincidence with previous numbers; top right: registers and random number generator; diagonally: writing numbers into registers.*

7. Other subcircuits

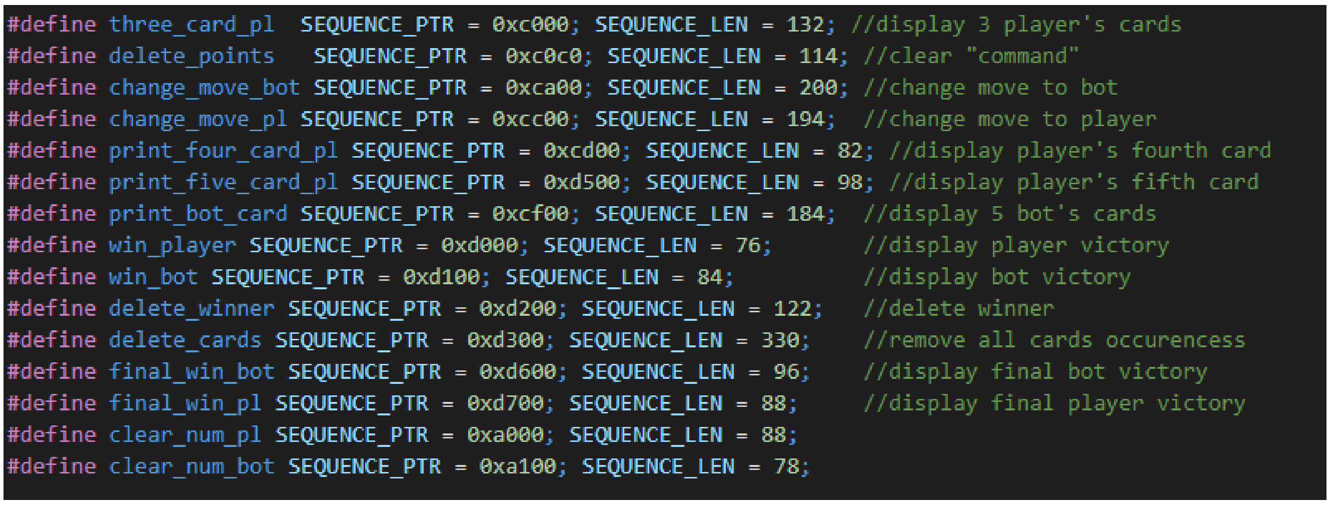
The other subcircuits, not described earlier, are needed only to remove unnecessary wires and elements, which dirty the look of our circuit.

5 Functional characteristics

In this project we decided to write the functional part in C ("main.c") and compile it for the cdm architecture in "file.asm". We do it using the command: "clang -target cdm -o file.asm -S main.c". Also we have a file "start.asm", which is necessary when starting "file.asm", and it contains arrays and variables used in "file.asm" and "main.c". Next, we link these two files and turn it into an "out.img" file using the command: "cocas -t cdm16 file.asm start.asm -o out.img".

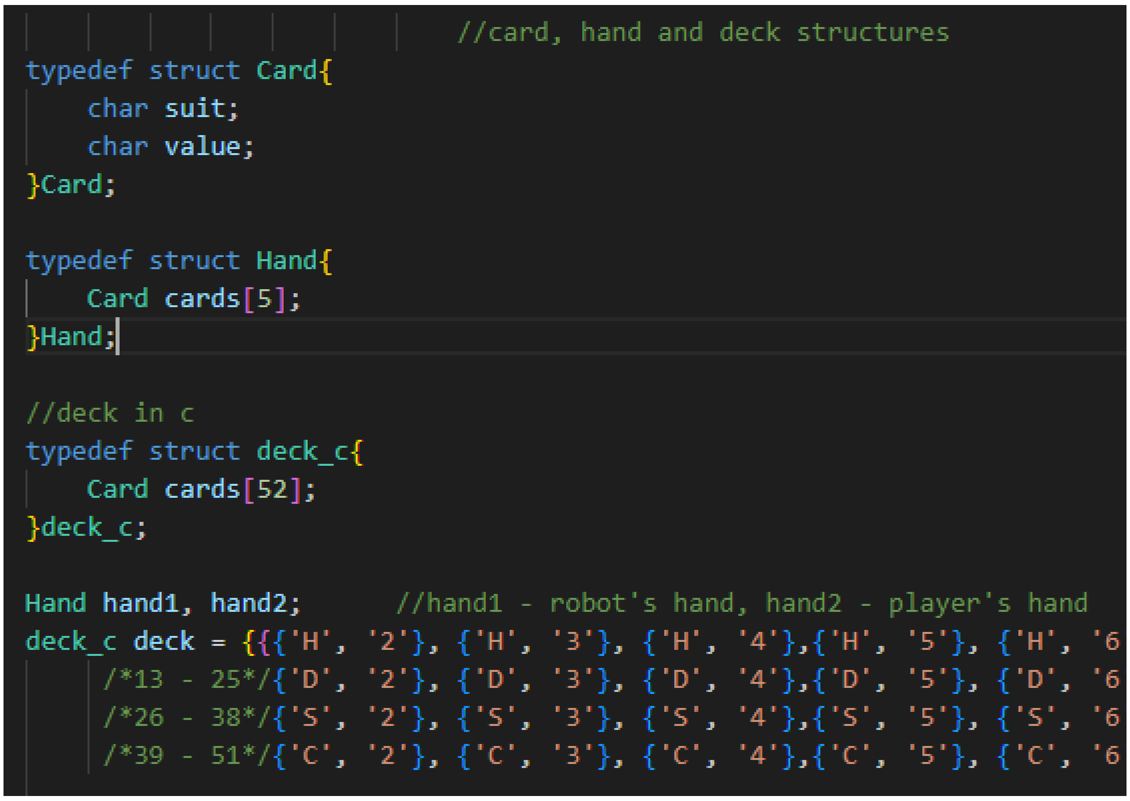
At the very beginning of the file "main.c" we, using the command: "extern volatile int", we include variables and arrays from "start.asm" into the file.

Part of the code below is designed to interact with the output of the interface described earlier:



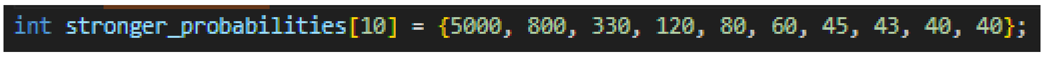
```
#define three_card_pl  SEQUENCE_PTR = 0xc000; SEQUENCE_LEN = 132; //display 3 player's cards
#define delete_points   SEQUENCE_PTR = 0xc0c0; SEQUENCE_LEN = 114; //clear "command"
#define change_move_bot SEQUENCE_PTR = 0xca00; SEQUENCE_LEN = 200; //change move to bot
#define change_move_pl SEQUENCE_PTR = 0xcc00; SEQUENCE_LEN = 194;  //change move to player
#define print_four_card_pl SEQUENCE_PTR = 0xcd00; SEQUENCE_LEN = 82; //display player's fourth card
#define print_five_card_pl SEQUENCE_PTR = 0xd500; SEQUENCE_LEN = 98; //display player's fifth card
#define print_bot_card SEQUENCE_PTR = 0xcf00; SEQUENCE_LEN = 184;  //display 5 bot's cards
#define win_player SEQUENCE_PTR = 0xd000; SEQUENCE_LEN = 76;       //display player victory
#define win_bot SEQUENCE_PTR = 0xd100; SEQUENCE_LEN = 84;          //display bot victory
#define delete_winner SEQUENCE_PTR = 0xd200; SEQUENCE_LEN = 122;   //delete winner
#define delete_cards SEQUENCE_PTR = 0xd300; SEQUENCE_LEN = 330;    //remove all cards occurencess
#define final_win_bot SEQUENCE_PTR = 0xd600; SEQUENCE_LEN = 96;    //display final bot victory
#define final_win_pl SEQUENCE_PTR = 0xd700; SEQUENCE_LEN = 88;     //display final player victory
#define clear_num_pl SEQUENCE_PTR = 0xa000; SEQUENCE_LEN = 88;
#define clear_num_bot SEQUENCE_PTR = 0xa100; SEQUENCE_LEN = 78;
```

Then we create the three structures we need to store the cards: player's hand (hand2) and bot’s hand (hand1). We also declare the value of the deck.



```
                              //card, hand and deck structures
typedef struct Card{
    char suit;
    char value;
}Card;

typedef struct Hand{
    Card cards[5];
}Hand;

//deck in c
typedef struct deck_c{
    Card cards[52];
}deck_c;

Hand hand1, hand2;      //hand1 - robot's hand, hand2 - player's hand
deck_c deck = {{{'H', '2'}, {'H', '3'}, {'H', '4'},{'H', '5'}, {'H', '6
    /*13 - 25*/{'D', '2'}, {'D', '3'}, {'D', '4'},{'D', '5'}, {'D', '6
    /*26 - 38*/{'S', '2'}, {'S', '3'}, {'S', '4'},{'S', '5'}, {'S', '6
    /*39 - 51*/{'C', '2'}, {'C', '3'}, {'C', '4'},{'C', '5'}, {'C', '6
```

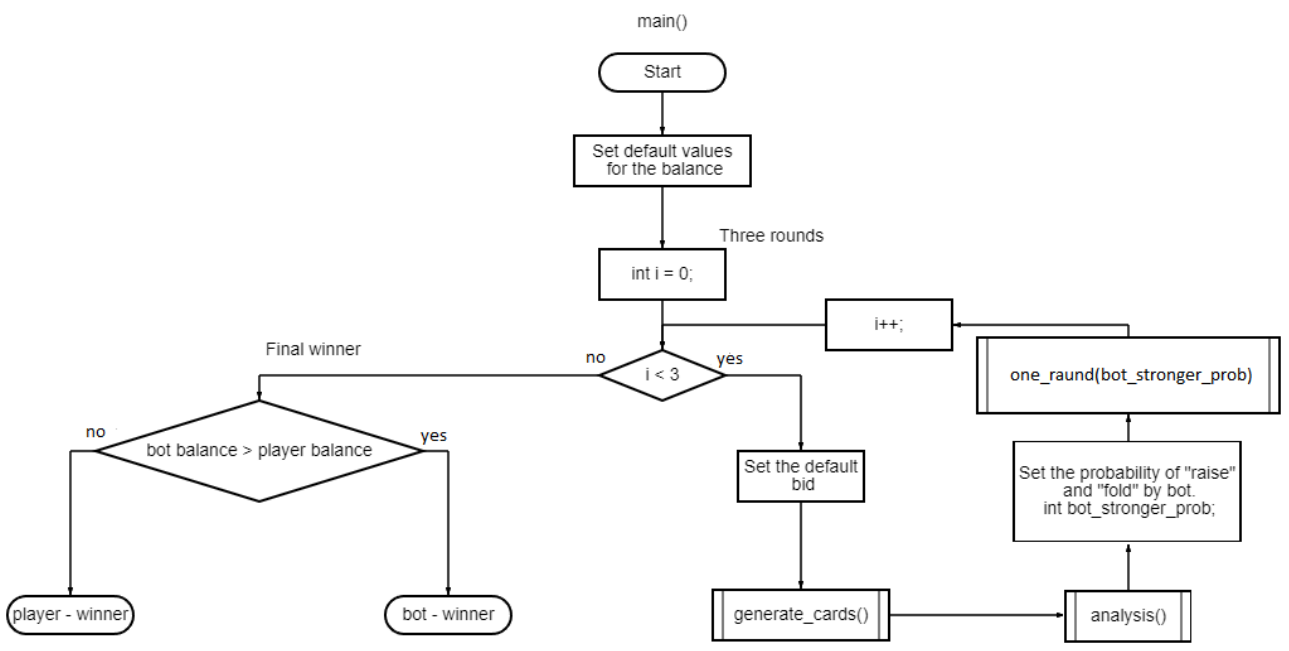
Next, we initialize the global array. It shows the probability that the player has a better combination than the bot.



```
int stronger_probabilities[10] = {5000, 800, 330, 120, 80, 60, 45, 43, 40, 40};
```

Now we're going to discuss the functions we need for the game.

1. **void main():**


main()
Start
Set default values for the balance
Three rounds
int i = 0;
i++;
Final winner
no
i < 3
yes
one_raund(bot_stronger_prob)
no
bot balance > player balance
yes
Set the default bid
Set the probability of "raise" and "fold" by bot. int bot_stronger_prob;
player - winner
bot - winner
generate_cards()
analysis()

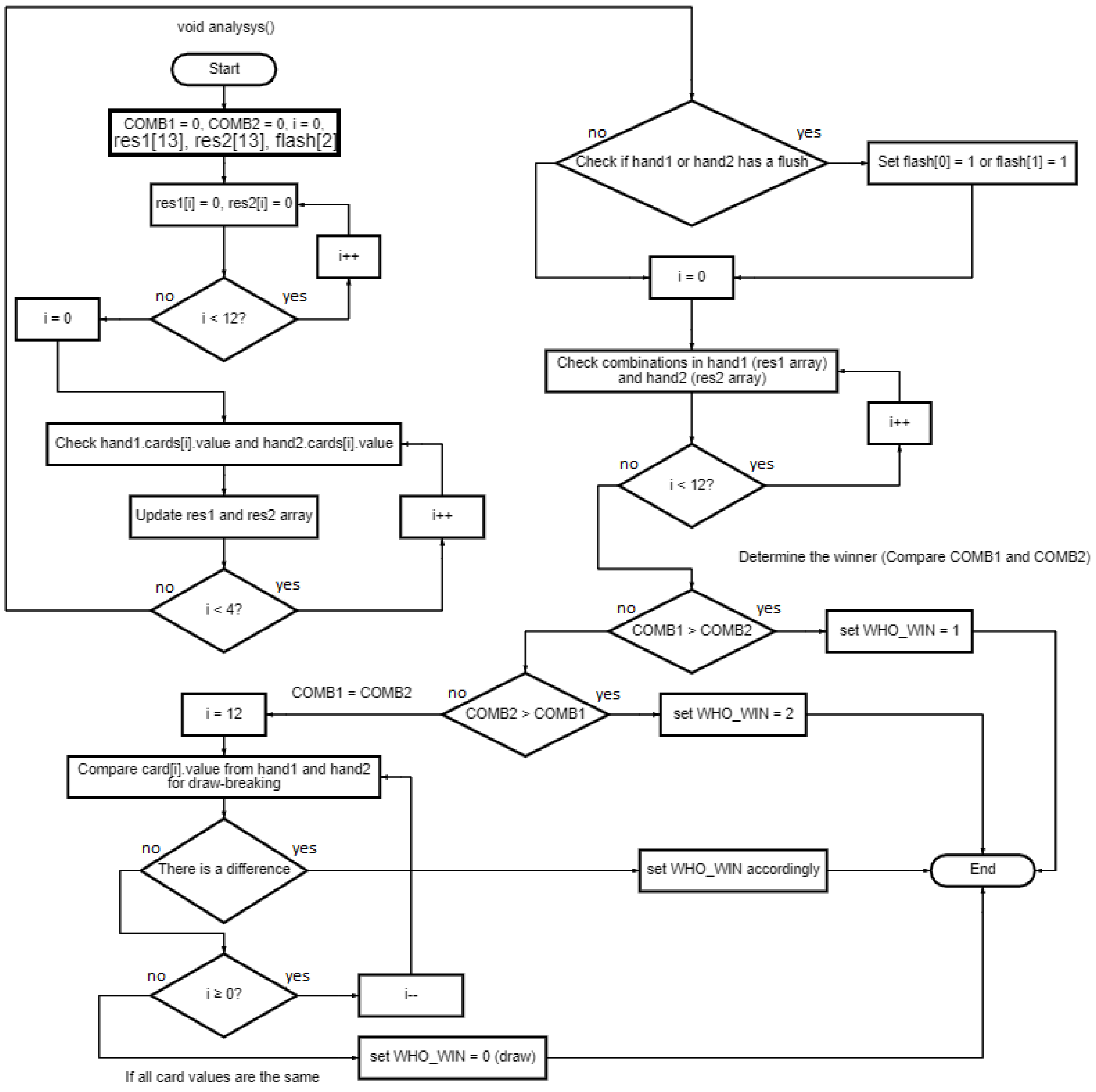
The default balance for player and bot is 2000

The "bot_stronger_prob" variable is indexed using the "stronger_probabilities" array. For example, if the bot combination is 1 (i.e., a “pair”), then "bot_stronger_prob = stronger_probabilities[1];". To set the probability that the bot will raise the bid, we take 6 numbers from the SUIT_VALUE(0xfe00) array, and if the number is odd, we raise it by 15, otherwise we move to the next number. Similarly, the probability that the bot will surrender, but only the check is carried out in the second round, and only if variable “bot_strongert_prob” > 6000.

2. **void generate_cards():**

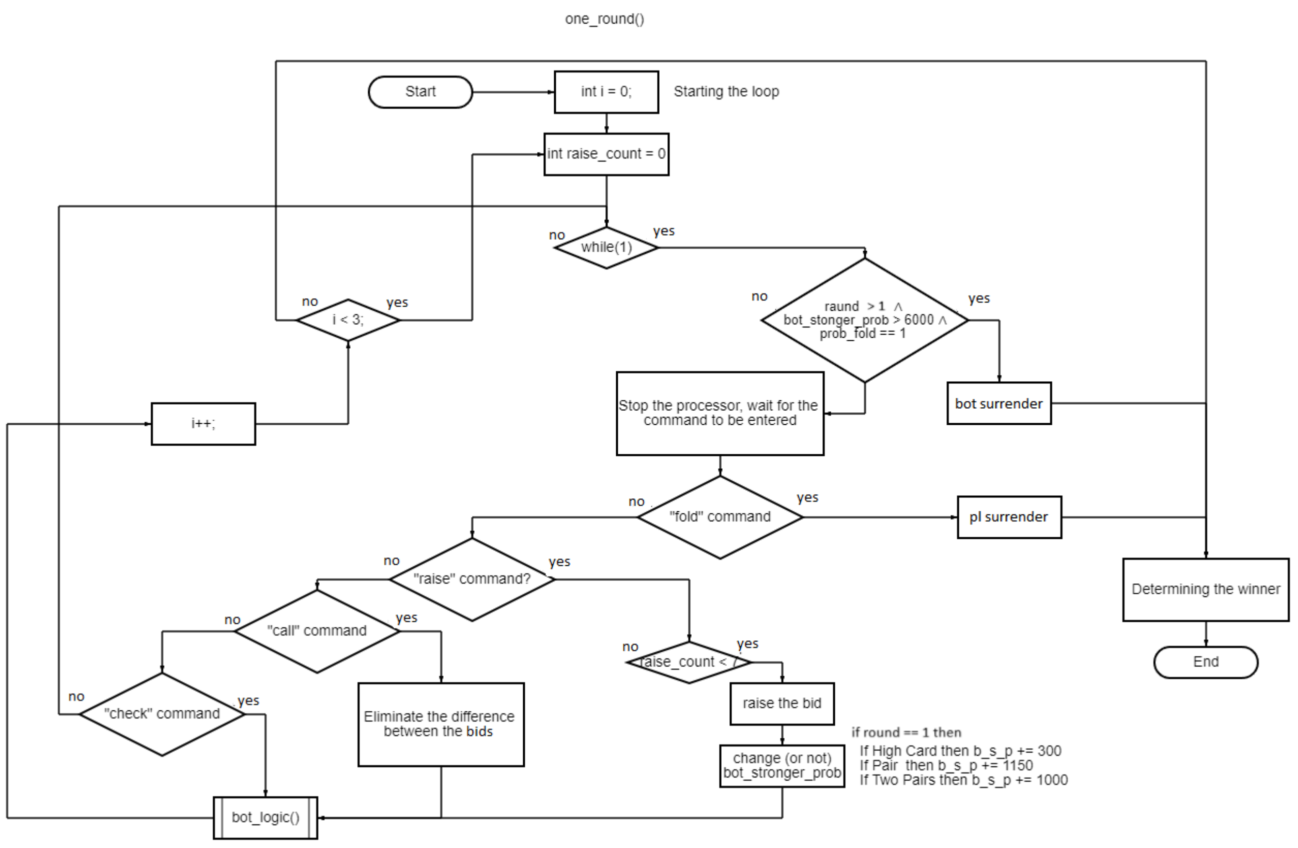
With 30 random numbers in SUIT_VALUE(0xfe00), this function gives values for hand1 and hand2, indexing them by the dare array defined earlier.

3. void analysis():


void analysys()
Start
COMB1 = 0, COMB2 = 0, i = 0, res1[13], res2[13], flash[2]
res1[i] = 0, res2[i] = 0
i++
i < 12?
no
yes
i = 0
Check hand1.cards[i].value and hand2.cards[i].value
Update res1 and res2 array
i++
i < 4?
Check if hand1 or hand2 has a flush
Set flash[0] = 1 or flash[1] = 1
i = 0
Check combinations in hand1 (res1 array) and hand2 (res2 array)
i++
i < 12?
Determine the winner (Compare COMB1 and COMB2)
COMB1 > COMB2
set WHO_WIN = 1
COMB2 > COMB1
set WHO_WIN = 2
COMB1 = COMB2
i = 12
Compare card[i].value from hand1 and hand2 for draw-breaking
There is a difference
set WHO_WIN accordingly
End
i ≥ 0?
i--
set WHO_WIN = 0 (draw)
If all card values are the same

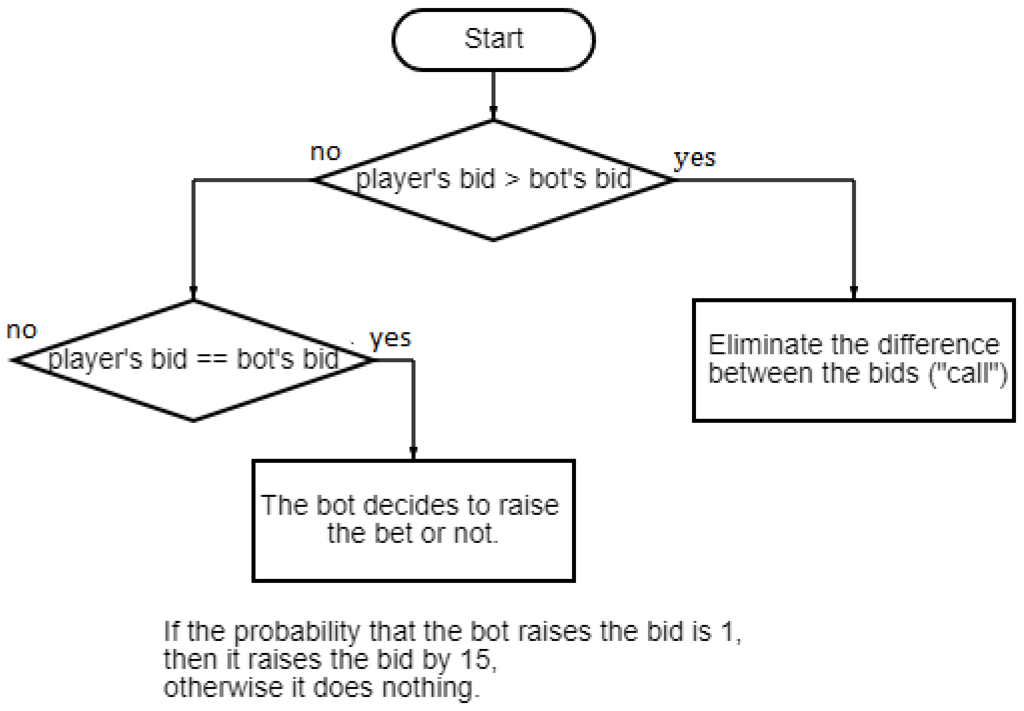
COMB1, COMB2, WHO_WIN - variables into which we will write the results of the combination analysis at the end of the function execution. res1, res2 - arrays, each cell of which can take the value 1 or 0 - there is a card in the combination or not, respectively.

4. void one_round(int bot_stronger_prob):

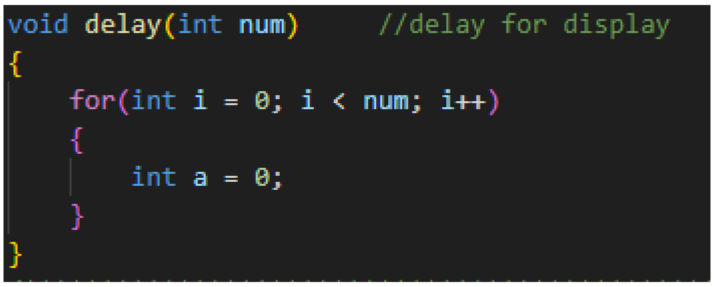


The function “one_round()” is needed to process one game round.

5. void bot_logic(int bot_stronger_prob):


Start
no
player's bid > bot's bid
yes
no
player's bid == bot's bid
yes
Eliminate the difference between the bids ("call")
The bot decides to raise the bet or not.
If the probability that the bot raises the bid is 1,
then it raises the bid by 15,
otherwise it does nothing.

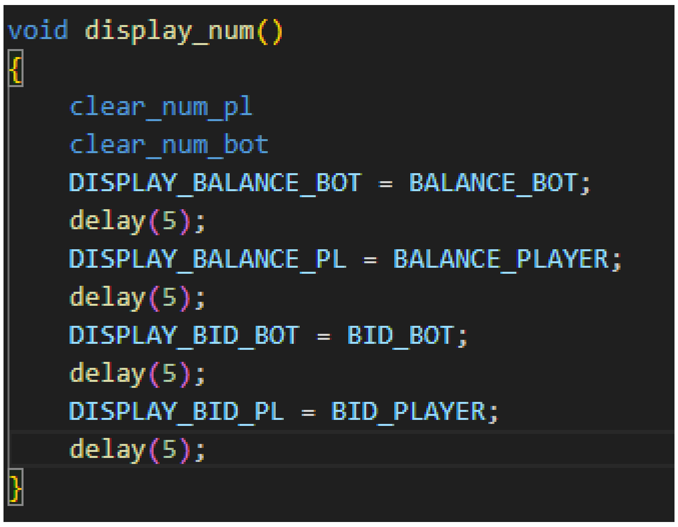
6. **void delay(int num):**



```
void delay(int num)     //delay for display
{
    for(int i = 0; i < num; i++)
    {
        int a = 0;
    }
}
```

This function takes a number and does a "for" loop on it. It is needed for the next function.

7. **void display_num():**



```
void display_num()
{
    clear_num_pl
    clear_num_bot
    DISPLAY_BALANCE_BOT = BALANCE_BOT;
    delay(5);
    DISPLAY_BALANCE_PL = BALANCE_PLAYER;
    delay(5);
    DISPLAY_BID_BOT = BID_BOT;
    delay(5);
    DISPLAY_BID_PL = BID_PLAYER;
    delay(5);
}
```

This part of the code is for the balance and bid output.

8. **Remaining functions:**

All the remaining functions not mentioned earlier are only needed for convenient interface output. More specifically, they substitute the required values into the cells in memory. They do not need a detailed description.

6 Conclusion

In conclusion, we managed to make a rather original project, implementing both software and hardware parts. Everything we wanted to do in our game, we implemented to the fullest extent. Of course, the game can still be supplemented with something, but we don't have any plans for that at the moment.

7 Sources used in developing

- logisim-uart: github.com/cdm-processors/logisim-uart
- C compiler for CdM-16: github.com/leadpogrommer/llvm-project-cdm
- cdm-devkit: github.com/dmitry-irtegov/cdm-devkit