Speach

1 Introduction (слайд 0)

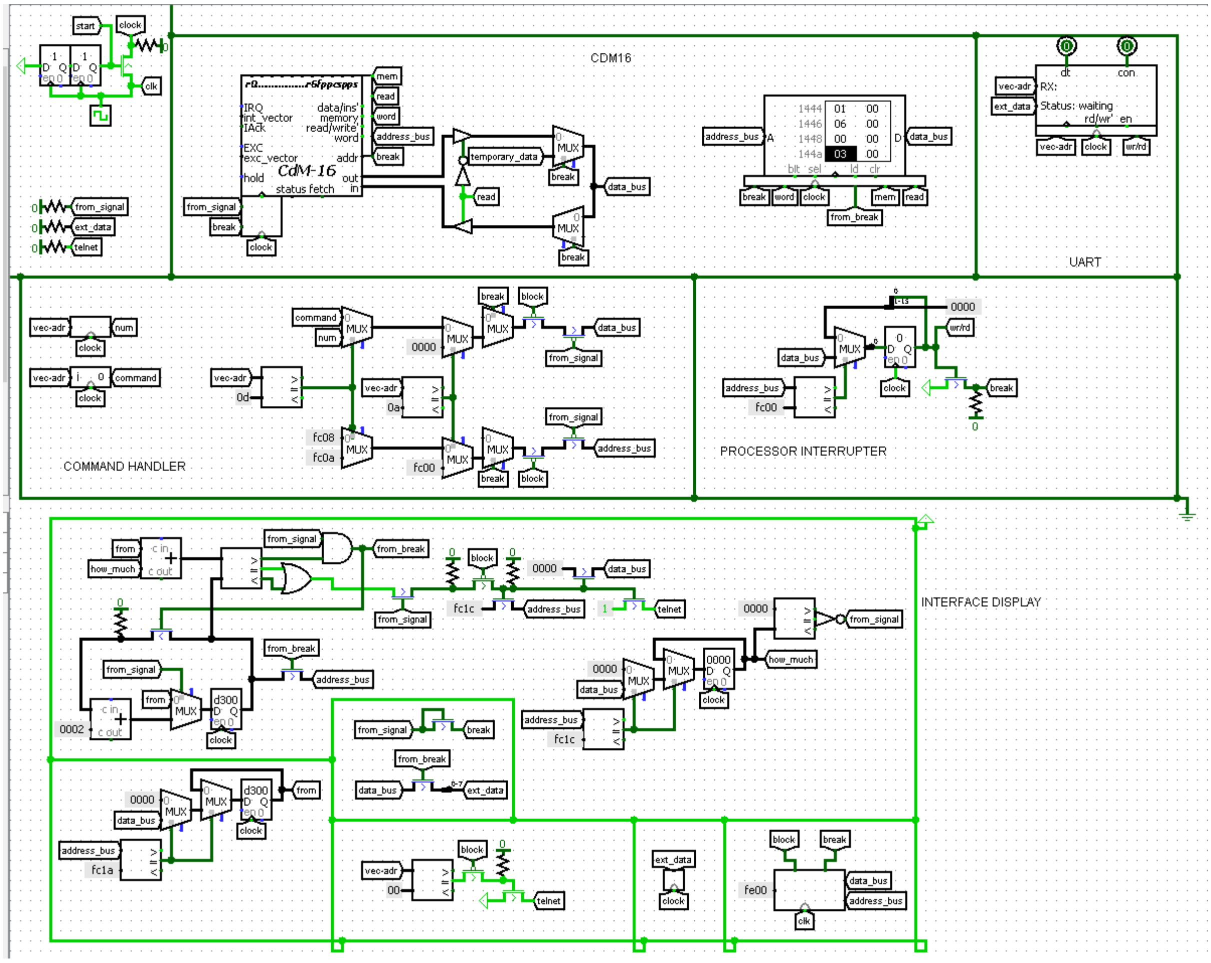
Ladies and gentlemen. We would like to show you our project.

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.

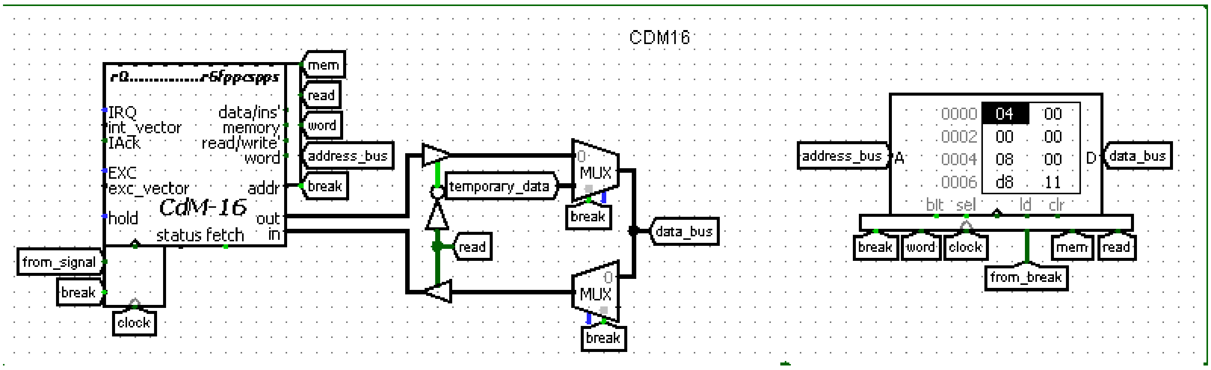
The interface of our program is displayed in the terminal using the telnet protocol.

4 Technical characteristics(1 слайд)



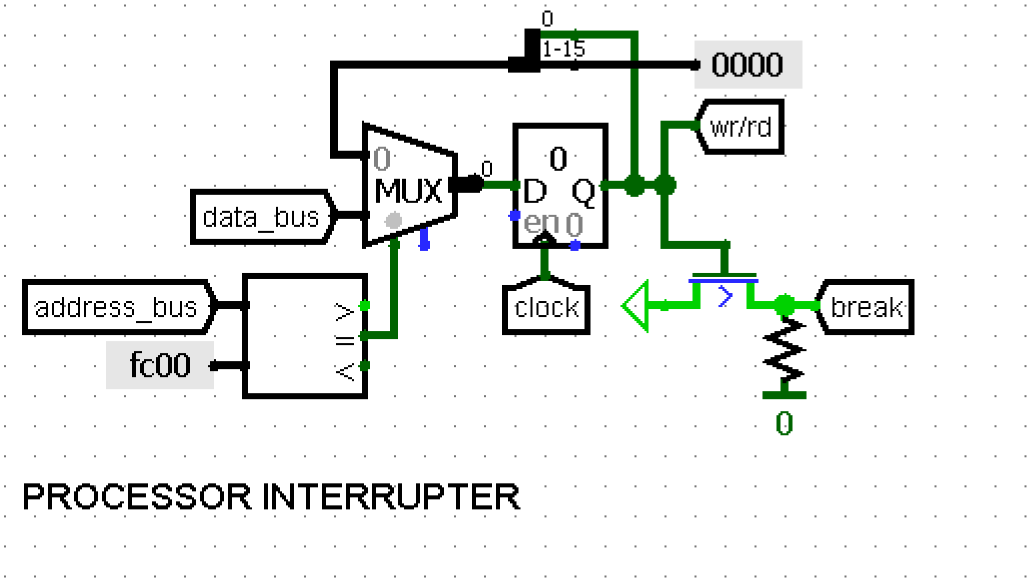
Now you can see a general view of the circuit. Let's see how it works in detail.

CDM16(2 слайд)



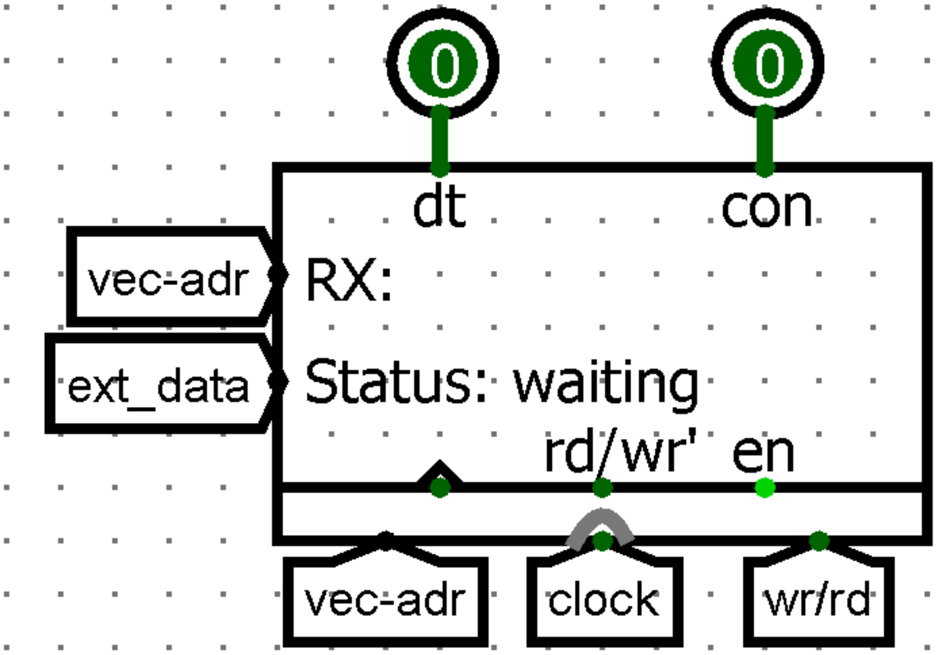
We are doing this project with a cdm16 processor using Von Neumann architecture. We chose it because it is much more user friendly than the cdm8/8e. Also on cdm16 we have the ability to write code using C language and compile it for cdm16 architecture.

PROCESSOR INTERRUPTER(3 СЛАЙД)



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

UART(4 СЛАЙД)

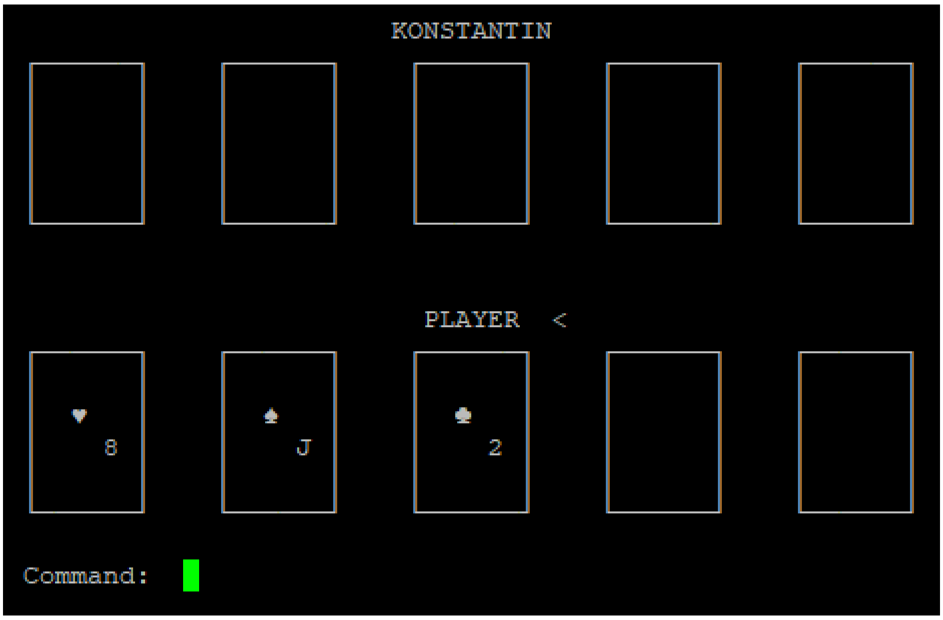


What do we need to see on the terminal? Five cards of the bot Konstantin (AI) and five cards of the player. And also a line into which the player will then enter commands indicating his decisions.

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)](https://github.com/cdm-processors/logisim-uart)). The part of the circuit responsible for Telnet communication is shown **there**.

(5 слайд)

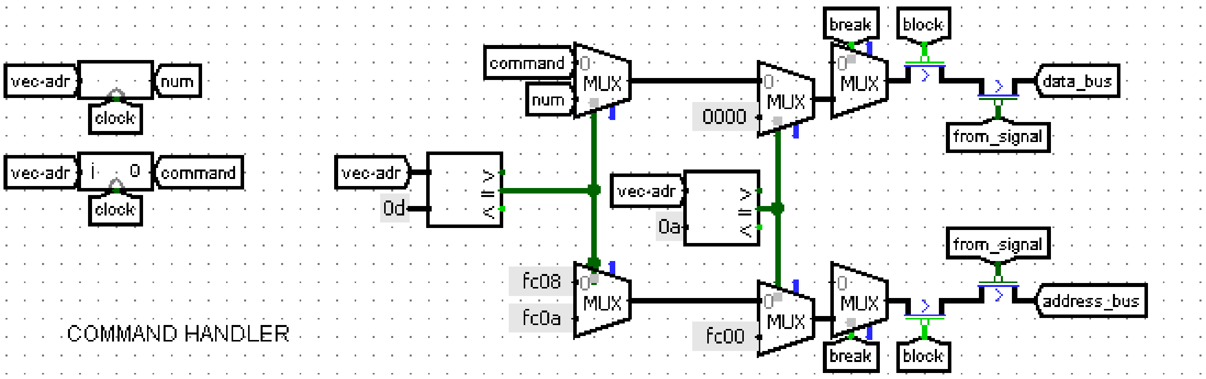
Thanks to this, we can now implement the output to the terminal of the interface described earlier:



```
                 KONSTANTIN

                   PLAYER  <

  ♥        ♠        ♣
    8        J        2

Command:
```

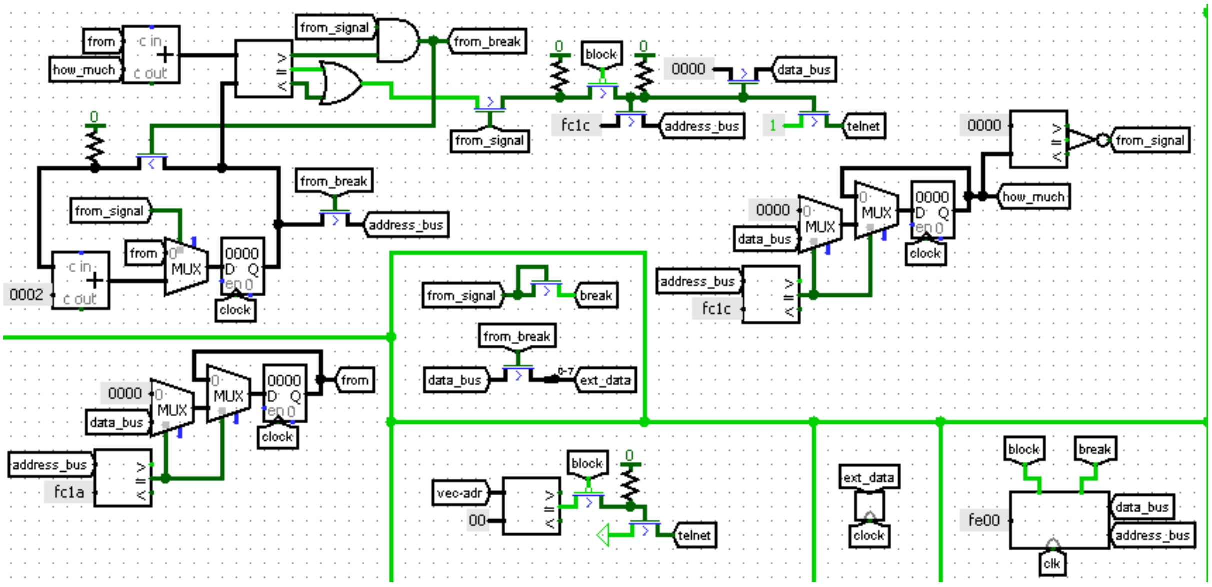
It’s look like this

COMMAND_HANDLER6 СЛАЙД)



It reads the input data you have entered into the terminal and writes the numeric value to the 0xfc08 and 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.

INTERFACE DISPLAY(7 СЛАЙД)

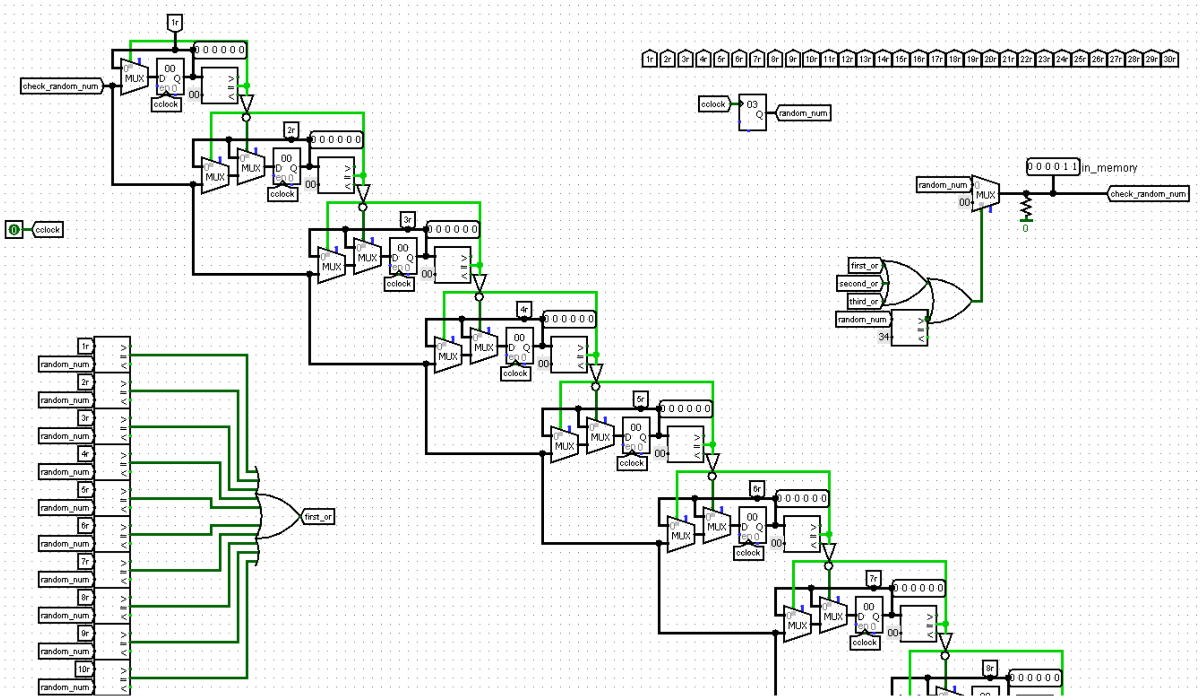


We also need to output values such as the values and suits of cards, whose turn it is, who won, etc. To do this in the program we write in one memory cell(0xfc1a) the pointer to the sequence in memory that we want to output, and in another(0xfc1c) - its length, then the processor stops and waits for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

RANDOM CARDS(8 СЛАЙД)

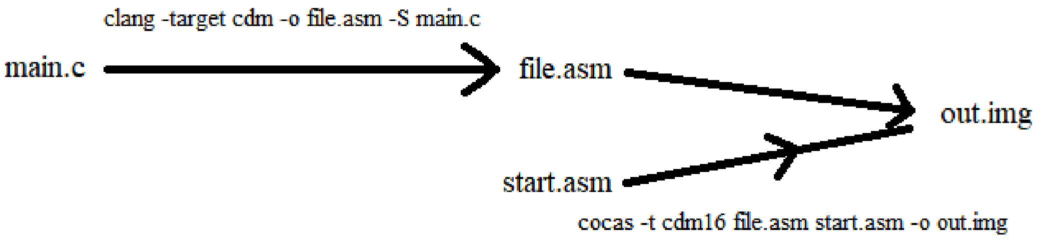
The cards in the hands of the player and bot are created using a random number generator.At the next generation the number is written to the corresponding register; there are 30 registers in total, since we have only 3 rounds, in each of which the player and the bot have a total of 10 cards in their hands. Then it inserts them into the memory cell

SUIT_VALUE(0xfe00).



5 Functional part **(9 СЛАЙД)**

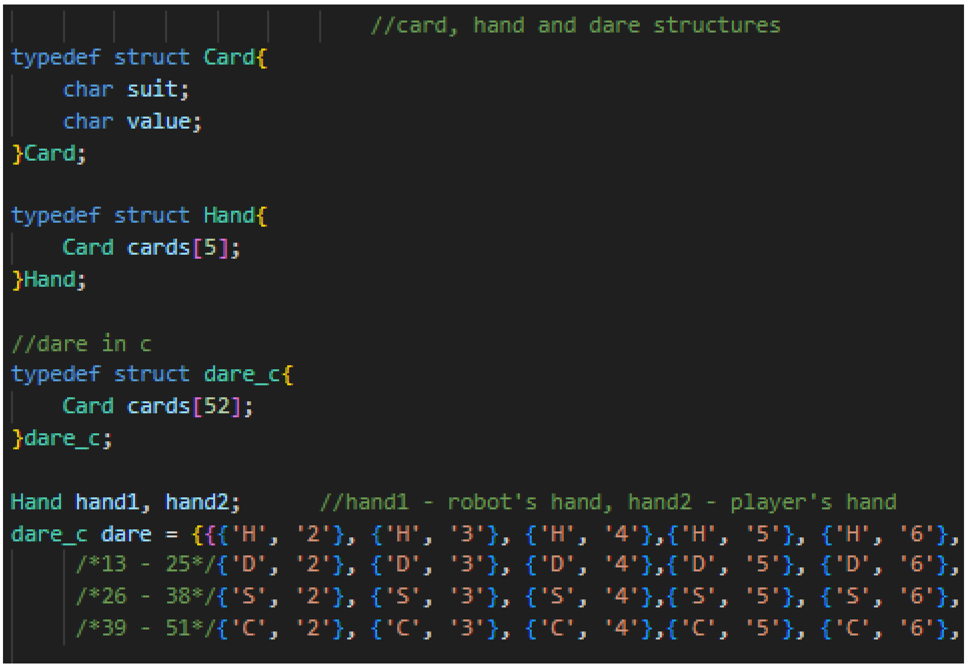
In this project we decided to write the functional part in C (specifically the "main.c" file) and compile it for the cdm architecture in “file.asm” using clang. Also we have a file "start.asm", which is necessary when starting "file.asm", and it contains arrays and variables used in "file.asm" and “main.c”. Next, we link these two files and turn it into an "out.img".



```
clang -target cdm -o file.asm -S main.c
```

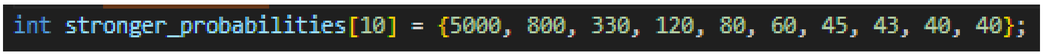
```
cocas -t cdm16 file.asm start.asm -o out.img
```

(10 СЛАЙД) At the very beginning of the file "main.c" 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 dare structures
typedef struct Card{
    char suit;
    char value;
}Card;

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

//dare in c
typedef struct dare_c{
    Card cards[52];
}dare_c;

Hand hand1, hand2;      //hand1 - robot's hand, hand2 - player's hand
dare_c dare = {{{'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'},
```

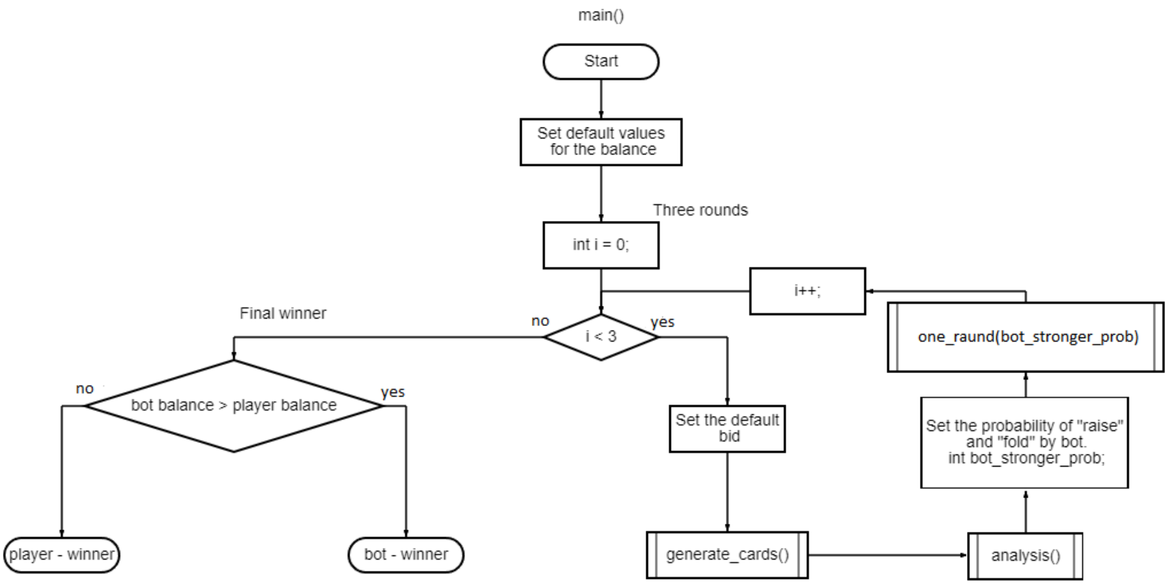
Next, we initialize the global array. It shows the probability that the player has a better combination than the bot.



```
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.

void main():(11 слайд

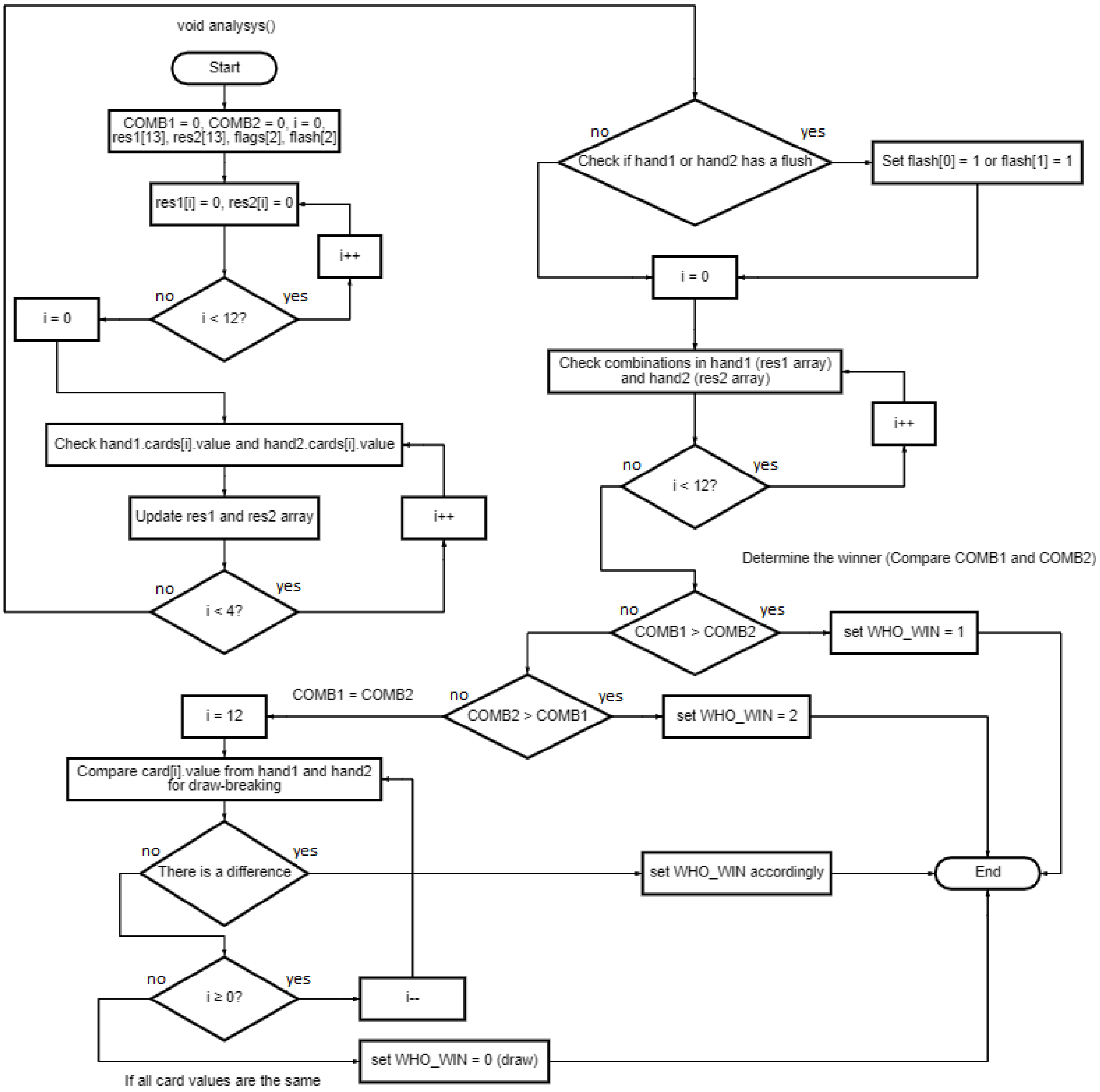


The first function is “main”. First of all, we set a default value for the player's balance and the bot's balance which is 2,000 each.. Then we start a triple loop because it's only three rounds. In each iteration we set the standard value of the player's and bot's bids (10 and 5 accordingly). Then we execute the function “generate_cards()”,then we run an “analysis()” function to define hand combinations. Then we set the probability that the bot will raise the bid and that he will surrender. After the loop, we reveal the winner.

void generate_cards():

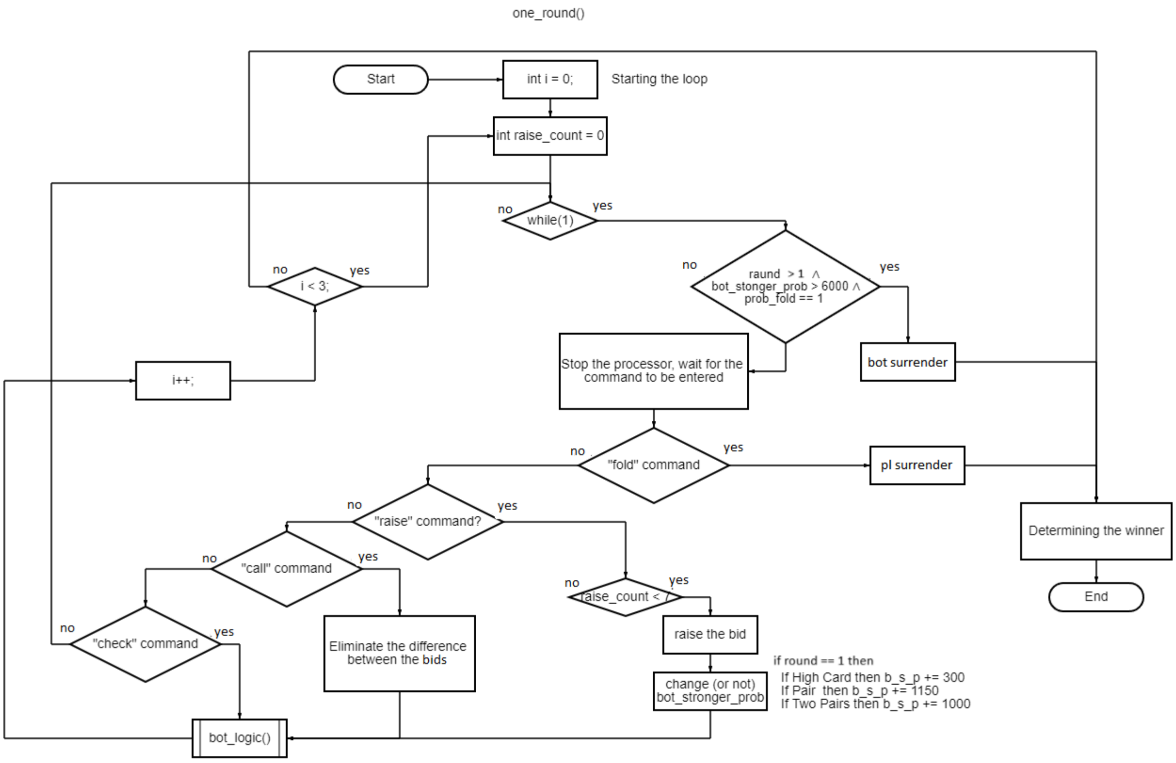
The next function is “generate_cards()”. With 30 random numbers in the 0xfe00 memory cell (described earlier), this function gives values**(10СЛАЙД)** for hand1 and hand2, indexing them by the dare array.

void analysis():(12СЛАЙД


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

....

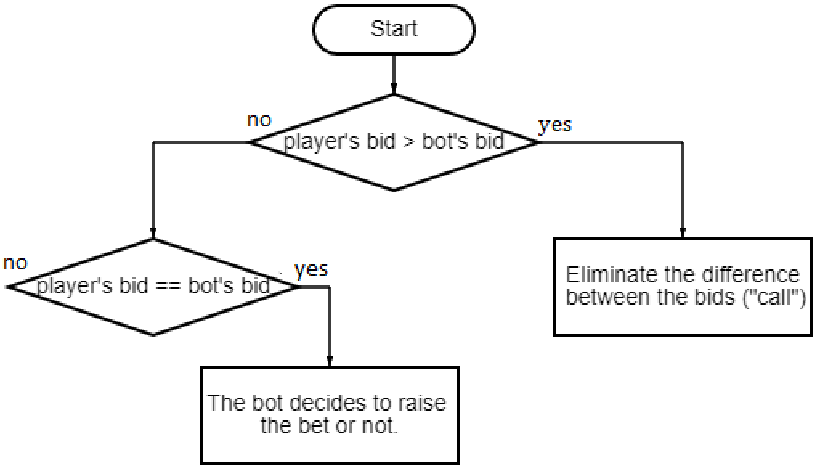
void one_round(int bot_stronger_prob):(13 слайд



if round == 1 then
If High Card then b_s_p += 300
If Pair then b_s_p += 1150
If Two Pairs then b_s_p += 1000

The next function "one_round()" is needed to process one game round. At the beginning, we start a triple loop. In it we start an infinite loop and check whether the bot will pass or not. If not, we ask the player to enter a command and depending on it we make decisions and execute the "bot_logic()" function.

(14 слайд


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.

The next function provides the logic for the bot's actions.

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. This concludes our presentation, thank you for your attention. We will be glad to hear your questions.