Halil BAYDAR - 2017400297

I used c++ programming lenguage because of pointer facilities and my project consists of only one main.cpp file. I have seven variables in the type of unsigned short which represents seven 16 bit registers and fifteen pointers that point their addresses with according to their types: For example, unsigned char pointer points the most 8 significant bit of 16 bit register and unsigned short pointer points the whole bit of 16 bit register. I have multiple arrays to check the given input to figure out what it is register or register with breaked(w[si]) or not(reg_array, bit16_array, bit8_array, bit16_array_with_breaked, bit16_array_with_b_and_breaked, bit16_array_with_w_and_breaked). There is a variable array that keep the name and type of given variables and named as index keeper array to holds index of variables in the memory. I have tokenizer helper function(**void** tokenizer(string s))that take the string parameter to reads the given input properly. It returns line code line by line for process and it is pushed into line array (vector<string> line_array).In first of all, I reads all given input to put whole code into array(vector<string> code_array) and also variables into the memory. And I also read whole input again to process all given code with regard to the first string in per-line. Morever, Mov, Add and Sub functions work similar, in the first step they traverse variable array(vector<pair<string, string>> variable_array) to find out if variable is in the process or not and then they check the destination operand in the left hand side of comma is direct register addresing or indirect register addresing operand by traversing arrays I mentioned above and then check memory addresing operand(by looking for line_array[0][0]’s characters) and finally controls the destination operand whether it is variable or not to continue the process. If destination operand is register, helper functions (**unsigned short** *return_pointer(string s); //returns 16 bit register and **unsigned char** *get8Bit(string s); *//returns 8 bit register*) return pointer and it is got into process. And if line_array[0][0]==’w’ or line_array[0][0]==’b’, it is decided to as a memory so helper functions(**int** calculate_index(**int** arr_num); //calculate memory index) is called and index is computed and memory location value is processed. And finally if variable is in the process, for loop gives index of var in the memory and it is calculated as (memory[index]+(memory[index+1]<<8/*the most 8 significant bits are in the higher index of memory*/)) to be processed.According to the destination operand some boolean variables (is_bit16, is_bit8)are changed and there is a helper function which returns the value of the right hand side of the comma by eliminating stuffs and calculating requested value, if there is no issue, and which works with regard to them((**int** return_value_of_right_hand_side(**int** index))),so value is copied from source operand to destinaiton in move and also similar in sub and add instructions. If the process is over successfully if there is no an error , Addition and subtruction instructions fallow same way but in addition at the end of the process flags are changed according to result of it(thanks to helper funciton change_flags_only_one_paramether(). Mul and Div functions also works similar with each other because there is only source operand in the line and this functions checks operand like I mentioned above to get into the process. If 16 bit process is in the process, destination register is immidately calculated like that(*pdx<<16+pax/*the most 16 significant bit of the result of process should be in dx register*/) for division instruction not mul. At the end of the process div and mov functions modify flags according to the result of operation and if there is overflow in the div operation this code gives an error message with the code line in the input(thanks to helper function (control_overflow(int a, int b, int byte)). And, xor, and, or instructions also work similar, so I checked the type of destination operand similar way, this code puts left operand into the process with the value (like value^=source_value, value|=source_value) and flags are changed. Rcr,rcl,shl,shr functions are same in terms of left operand and also right operand, so they can be processed with cl register and immediate value lower than 31d number so (**int return_value()**) which is helper function returns the value in the reight hand side and it is processed with it. In the case of rcl function, Cf is equlized with the most significant bit of value and value is shifted left by one to be added the previous value of CF to value (after shitfted, value|=the previous value of CF). In the case of rcr function , (unsigned short tt = CF << sizeof(*ptr) * 8 - 1); this lets CF shifted right as much as the byte size of value to be added as the MSB into value. CF=value&1 it makes Cf equal to the LSB of the value and the previous value is pushed into the value as the MSB(value|=tt) after one bit shifted.In the case of shr function, value is shifted to right so long as destination value and CF is become equal the LSB of value. In the case of shl function, value is shifted to left so long as destination value and CF is become equal the MSB of value.Inc and dec functions controls destination operands like other I mentioned above and increases or decreases the destination operand by one and at the end of changes flags are changed with according to flags. Push function works only with word size operand and put the value into the memory by splitting the most 8 significant and the least 8 significant bits of the value. Cmp function controls destination operand like other functions I mentioned above and changes flags. Jmp functions traverse code array that keeps whole given code to find label and checks the related flags and it equlizes iterator with label index and it jumps. Int 21h function only checks ah register if it has 01, the funciton reads the input then puts it into al register ,and if it has 02 this function writes dl register’s ascii value to screen. If iterator reaches to the int 20h function, this program ends up.