Лабораторная работа № 1. Стековая виртуальная машина

22 февраля 2024 г.

Сергей Виленский, ИУ9-62Б

Цель работы

Целью данной работы является написание интерпретатора ассемблера модельного компьютера, который будет использоваться в последующих лабораторных как целевой язык.

Реализация интерпретатора

```
// константы
`define MAX_LEXEM_COUNT 1024
`define MAX_LEXEM_SIZE  32
`define MAX_LABELS      512
`define MEMMORY_LIMIT   1_000_000

module main;
    // имена файлов
    localparam targerProgramFile    = "lab1.asm";
    localparam stdinFile            = "stdin.txt";

    // лексемы
    reg [7:0]   lexemsArray[0:`MAX_LEXEM_COUNT-1][0:`MAX_LEXEM_SIZE-1];
    integer     lexemsSizesArray[0:`MAX_LEXEM_COUNT-1];
    integer     lexemsCount = 0;

    // метки
    reg [7:0]       labelsArray[0:`MAX_LABELS-1][0:`MAX_LEXEM_SIZE-1];
    integer         labelsSizesArray[0:`MAX_LABELS-1];
    integer signed  labelsValuesArray[0:`MAX_LABELS-1];
    integer         labelsCount = 0;

    // фактическая память машины
```

```
    integer signed  memmory[0:`MEMMORY_LIMIT-1];    // память машины
    integer         program_size;  // размер программы в памяти машины
    integer         reg_IP;     // указатель инструкций (instruction pointer)
    integer         reg_SP;     // указатель стека (stack pointer)
    integer signed  reg_FP;     // указатель базы (frame pointer)
    integer signed  reg_RV;     // возвращаемое значение (return value)

    // вспомогательные переменные при работе со стеком
    integer signed x, y, z, N, a, v;

    // ошибка, выдаваемая при недостаточном размере
    //  стека при выпонении некоторой операции
    localparam STACK_ERROR =
        "RuntimeError: line~%d: %s: In stack must \
be at least %d element, but %d found.\n";

    // ASCII-коды нужных сиволов
    localparam CHAR_TAB         = 9;
    localparam CHAR_NEWLINE     = 10;
    localparam CHAR_SPACE       = 32;
    localparam CHAR_PLUS        = 43;
    localparam CHAR_MINUS       = 45;
    localparam CHAR_HYPHEN      = 45;
    localparam CHAR_0           = 48;
    localparam CHAR_9           = 57;
    localparam CHAR_COLON       = 58;
    localparam CHAR_SEMICOLON   = 59;
    localparam CHAR_A           = 65;
    localparam CHAR_B           = 66;
    localparam CHAR_C           = 67;
    localparam CHAR_D           = 68;
    localparam CHAR_E           = 69;
    localparam CHAR_F           = 70;
    localparam CHAR_G           = 71;
    localparam CHAR_H           = 72;
    localparam CHAR_I           = 73;
    localparam CHAR_J           = 74;
    localparam CHAR_K           = 75;
    localparam CHAR_L           = 76;
    localparam CHAR_M           = 77;
    localparam CHAR_N           = 78;
    localparam CHAR_O           = 79;
    localparam CHAR_P           = 80;
    localparam CHAR_Q           = 81;
    localparam CHAR_R           = 82;
    localparam CHAR_S           = 83;
```

```
localparam CHAR_T           = 84;
localparam CHAR_U           = 85;
localparam CHAR_V           = 86;
localparam CHAR_W           = 87;
localparam CHAR_X           = 88;
localparam CHAR_Y           = 89;
localparam CHAR_Z           = 90;
localparam CHAR_UNDERSCORE  = 95;
localparam CHAR_a           = 97;
localparam CHAR_z           = 122;

// коды встроенных команд
localparam CMD_ADD          = -1;
localparam CMD_SUB          = -2;
localparam CMD_MUL          = -3;
localparam CMD_DIV          = -4;
localparam CMD_MOD          = -5;
localparam CMD_NEG          = -6;
localparam CMD_BITAND       = -7;
localparam CMD_BITOR        = -8;
localparam CMD_BITNOT       = -9;
localparam CMD_LSHIFT       = -10;
localparam CMD_RSHIFT       = -11;
localparam CMD_DUP          = -12;
localparam CMD_DROP         = -13;
localparam CMD_SWAP         = -14;
localparam CMD_ROT          = -15;
localparam CMD_OVER         = -16;
localparam CMD_DROPN        = -17;
localparam CMD_PUSHN        = -18;
localparam CMD_LOAD         = -19;
localparam CMD_SAVE         = -20;
localparam CMD_GETIP        = -21;
localparam CMD_SETIP        = -22;
localparam CMD_GETSP        = -23;
localparam CMD_SETSP        = -24;
localparam CMD_GETFP        = -25;
localparam CMD_SETFP        = -26;
localparam CMD_GETRV        = -27;
localparam CMD_SETRV        = -28;
localparam CMD_CMP          = -29;
localparam CMD_JMP          = -22;
localparam CMD_JLT          = -30;
localparam CMD_JGT          = -31;
localparam CMD_JEQ          = -32;
localparam CMD_JLE          = -33;
```

```
localparam CMD_JGE          = -34;
localparam CMD_JNE          = -35;
localparam CMD_CALL         = -36;
localparam CMD_RETN         = -37;
localparam CMD_IN           = -38;
localparam CMD_OUT          = -39;
localparam CMD_HALT         = -40;

// вспомогательные переменные
integer fd, i, j, k, current;
reg [7:0] char;
reg isComment, isEqual, isNegativ, wasFound;

initial begin

    // <<<<<===== первый этап - парсинг =====>>>>>
    lexemsSizesArray[0] = 0;  // обнуляем размер первой лексемы

    // открытие файла программы
    fd = $fopen(targerProgramFile, "r");
    // проверка на существование файла
    if (!fd) begin
        $write("FileError: File '%s' unexists.\n", targerProgramFile);
        $finish(1);
    end

    // чтение файла
    while (!$feof(fd)) begin
        char = $fgetc(fd);

        if (isComment == 1) begin // игнорируем, если это коммент
        end else if (  // завершаем считывание прошлой лексемы
            char == CHAR_SEMICOLON  ||
            char == CHAR_NEWLINE    ||
            char == CHAR_TAB        ||
            char == CHAR_SPACE
        ) begin
            if (lexemsSizesArray[lexemsCount] != 0) begin
                ++lexemsCount;
                lexemsSizesArray[lexemsCount] = 0;
            end
        end else begin  // следующий символ лексемы

            // проверка на переполнение массива лексем
            if (lexemsCount >= `MAX_LEXEM_COUNT) begin
                $write("SyntaxError: Too many lexems. Max lexem count - %d.\n",
```

```
                                                        `MAX_LEXEM_COUNT);
            $finish(1);
        end

        // проверка на превышение размера лексем
        if (lexemsSizesArray[lexemsCount] >= `MAX_LEXEM_SIZE) begin
            $write("SyntaxError: Too long lexem - ");
            for (i = 0; i != lexemsSizesArray[lexemsCount]; ++i) begin
                $write("%s", lexemsArray[lexemsCount][i]);
            end
            $write(". Max lexem size - %d.\n", `MAX_LEXEM_SIZE);
            $finish(1);
        end

        // приписываем последней лексеме новый символ
        lexemsArray[lexemsCount][lexemsSizesArray[lexemsCount]] = char;
        ++lexemsSizesArray[lexemsCount];
    end

    if (char == CHAR_SEMICOLON) begin  // начало коммента
        isComment = 1;
    end else if (char == CHAR_NEWLINE) begin  // конец коммента
        isComment = 0;
    end
end
// при чтении файла считывается символ с кодом 255. Странно. Удаляем.
--lexemsSizesArray[lexemsCount];
// если чтение последней лексемы не закончено, заканчиваем
if (lexemsSizesArray[lexemsCount] > 0) begin
    ++lexemsCount;
end
$fclose(fd);  // закрыаем файл

// // <<<<<======= ДЕБАГ: печать массива считанных лексем
// for (i = 0; i != lexemsCount; ++i) begin
//     for (j = 0; j != lexemsSizesArray[i]; ++j) begin
//         $write("%s", lexemsArray[i][j]);
//     end
//     $write("\n---\n");
// end


// program_size по умолчанию имеет индекс 0 в массиве.
// Остальные мнемотики и метки не привязаны к индексам массива.
labelsArray[0][0] = CHAR_P;
labelsArray[0][1] = CHAR_R;
```

```
labelsArray[0][2] = CHAR_O;
labelsArray[0][3] = CHAR_G;
labelsArray[0][4] = CHAR_R;
labelsArray[0][5] = CHAR_A;
labelsArray[0][6] = CHAR_M;
labelsArray[0][7] = CHAR_UNDERSCORE;
labelsArray[0][8] = CHAR_S;
labelsArray[0][9] = CHAR_I;
labelsArray[0][10] = CHAR_Z;
labelsArray[0][11] = CHAR_E;
labelsSizesArray[0] = 12;
labelsCount = 1;

// 41 встроенная мнемотика.....................
labelsArray[labelsCount][0] = CHAR_A;
labelsArray[labelsCount][1] = CHAR_D;
labelsArray[labelsCount][2] = CHAR_D;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_ADD;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_S;
labelsArray[labelsCount][1] = CHAR_U;
labelsArray[labelsCount][2] = CHAR_B;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_SUB;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_M;
labelsArray[labelsCount][1] = CHAR_U;
labelsArray[labelsCount][2] = CHAR_L;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_MUL;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_D;
labelsArray[labelsCount][1] = CHAR_I;
labelsArray[labelsCount][2] = CHAR_V;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_DIV;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_M;
labelsArray[labelsCount][1] = CHAR_O;
labelsArray[labelsCount][2] = CHAR_D;
labelsSizesArray[labelsCount] = 3;
```

```
labelsValuesArray[labelsCount] = CMD_MOD;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_N;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_G;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_NEG;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_B;
labelsArray[labelsCount][1] = CHAR_I;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_A;
labelsArray[labelsCount][4] = CHAR_N;
labelsArray[labelsCount][5] = CHAR_D;
labelsSizesArray[labelsCount] = 6;
labelsValuesArray[labelsCount] = CMD_BITAND;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_B;
labelsArray[labelsCount][1] = CHAR_I;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_O;
labelsArray[labelsCount][4] = CHAR_R;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_BITOR;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_B;
labelsArray[labelsCount][1] = CHAR_O;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_N;
labelsArray[labelsCount][4] = CHAR_O;
labelsArray[labelsCount][5] = CHAR_T;
labelsSizesArray[labelsCount] = 6;
labelsValuesArray[labelsCount] = CMD_BITNOT;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_L;
labelsArray[labelsCount][1] = CHAR_S;
labelsArray[labelsCount][2] = CHAR_H;
labelsArray[labelsCount][3] = CHAR_I;
labelsArray[labelsCount][4] = CHAR_F;
labelsArray[labelsCount][5] = CHAR_T;
labelsSizesArray[labelsCount] = 6;
```

```
labelsValuesArray[labelsCount] = CMD_LSHIFT;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_R;
labelsArray[labelsCount][1] = CHAR_S;
labelsArray[labelsCount][2] = CHAR_H;
labelsArray[labelsCount][3] = CHAR_I;
labelsArray[labelsCount][4] = CHAR_F;
labelsArray[labelsCount][5] = CHAR_T;
labelsSizesArray[labelsCount] = 6;
labelsValuesArray[labelsCount] = CMD_RSHIFT;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_D;
labelsArray[labelsCount][1] = CHAR_U;
labelsArray[labelsCount][2] = CHAR_P;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_DUP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_D;
labelsArray[labelsCount][1] = CHAR_R;
labelsArray[labelsCount][2] = CHAR_O;
labelsArray[labelsCount][3] = CHAR_P;
labelsSizesArray[labelsCount] = 4;
labelsValuesArray[labelsCount] = CMD_DROP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_S;
labelsArray[labelsCount][1] = CHAR_W;
labelsArray[labelsCount][2] = CHAR_A;
labelsArray[labelsCount][3] = CHAR_P;
labelsSizesArray[labelsCount] = 4;
labelsValuesArray[labelsCount] = CMD_SWAP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_R;
labelsArray[labelsCount][1] = CHAR_O;
labelsArray[labelsCount][2] = CHAR_T;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_ROT;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_O;
labelsArray[labelsCount][1] = CHAR_V;
labelsArray[labelsCount][2] = CHAR_E;
```

```
labelsArray[labelsCount][3] = CHAR_R;
labelsSizesArray[labelsCount] = 4;
labelsValuesArray[labelsCount] = CMD_OVER;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_D;
labelsArray[labelsCount][1] = CHAR_R;
labelsArray[labelsCount][2] = CHAR_O;
labelsArray[labelsCount][3] = CHAR_P;
labelsArray[labelsCount][4] = CHAR_N;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_DROPN;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_P;
labelsArray[labelsCount][1] = CHAR_U;
labelsArray[labelsCount][2] = CHAR_S;
labelsArray[labelsCount][3] = CHAR_H;
labelsArray[labelsCount][4] = CHAR_N;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_PUSHN;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_L;
labelsArray[labelsCount][1] = CHAR_O;
labelsArray[labelsCount][2] = CHAR_A;
labelsArray[labelsCount][3] = CHAR_D;
labelsSizesArray[labelsCount] = 4;
labelsValuesArray[labelsCount] = CMD_LOAD;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_S;
labelsArray[labelsCount][1] = CHAR_A;
labelsArray[labelsCount][2] = CHAR_V;
labelsArray[labelsCount][3] = CHAR_E;
labelsSizesArray[labelsCount] = 4;
labelsValuesArray[labelsCount] = CMD_SAVE;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_G;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_I;
labelsArray[labelsCount][4] = CHAR_P;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_GETIP;
```

```
++labelsCount;

labelsArray[labelsCount][0] = CHAR_S;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_I;
labelsArray[labelsCount][4] = CHAR_P;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_SETIP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_G;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_S;
labelsArray[labelsCount][4] = CHAR_P;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_GETSP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_S;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_S;
labelsArray[labelsCount][4] = CHAR_P;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_SETSP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_G;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_F;
labelsArray[labelsCount][4] = CHAR_P;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_GETFP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_S;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_F;
labelsArray[labelsCount][4] = CHAR_P;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_SETFP;
++labelsCount;
```

```
labelsArray[labelsCount][0] = CHAR_G;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_R;
labelsArray[labelsCount][4] = CHAR_V;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_GETRV;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_S;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_R;
labelsArray[labelsCount][4] = CHAR_V;
labelsSizesArray[labelsCount] = 5;
labelsValuesArray[labelsCount] = CMD_SETRV;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_C;
labelsArray[labelsCount][1] = CHAR_M;
labelsArray[labelsCount][2] = CHAR_P;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_CMP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_J;
labelsArray[labelsCount][1] = CHAR_M;
labelsArray[labelsCount][2] = CHAR_P;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_JMP;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_J;
labelsArray[labelsCount][1] = CHAR_L;
labelsArray[labelsCount][2] = CHAR_T;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_JLT;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_J;
labelsArray[labelsCount][1] = CHAR_G;
labelsArray[labelsCount][2] = CHAR_T;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_JGT;
++labelsCount;
```

```
labelsArray[labelsCount][0] = CHAR_J;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_Q;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_JEQ;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_J;
labelsArray[labelsCount][1] = CHAR_L;
labelsArray[labelsCount][2] = CHAR_E;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_JLE;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_J;
labelsArray[labelsCount][1] = CHAR_G;
labelsArray[labelsCount][2] = CHAR_E;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_JGE;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_J;
labelsArray[labelsCount][1] = CHAR_N;
labelsArray[labelsCount][2] = CHAR_E;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_JNE;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_C;
labelsArray[labelsCount][1] = CHAR_A;
labelsArray[labelsCount][2] = CHAR_L;
labelsArray[labelsCount][3] = CHAR_L;
labelsSizesArray[labelsCount] = 4;
labelsValuesArray[labelsCount] = CMD_CALL;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_R;
labelsArray[labelsCount][1] = CHAR_E;
labelsArray[labelsCount][2] = CHAR_T;
labelsArray[labelsCount][3] = CHAR_N;
labelsSizesArray[labelsCount] = 4;
labelsValuesArray[labelsCount] = CMD_RETN;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_I;
```

```
labelsArray[labelsCount][1] = CHAR_N;
labelsSizesArray[labelsCount] = 2;
labelsValuesArray[labelsCount] = CMD_IN;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_O;
labelsArray[labelsCount][1] = CHAR_U;
labelsArray[labelsCount][2] = CHAR_T;
labelsSizesArray[labelsCount] = 3;
labelsValuesArray[labelsCount] = CMD_OUT;
++labelsCount;

labelsArray[labelsCount][0] = CHAR_H;
labelsArray[labelsCount][1] = CHAR_A;
labelsArray[labelsCount][2] = CHAR_L;
labelsArray[labelsCount][3] = CHAR_T;
labelsSizesArray[labelsCount] = 4;
labelsValuesArray[labelsCount] = CMD_HALT;
++labelsCount;


// <<<<<===== второй этап - анализ (проходы) и исполнение =====>>>>>

// первый проход ==========>
current = 256;  // указание
for (i = 0; i != lexemsCount; ++i) begin
    if (lexemsArray[i][0] == CHAR_COLON) begin

        // проверка синтаксиса метки
        // проверка первого символа идентификатора
        if (
            lexemsSizesArray[i] < 2 ||
            !(
                CHAR_a <= lexemsArray[i][1] &&
                          lexemsArray[i][1] <= CHAR_z ||
                CHAR_A <= lexemsArray[i][1] &&
                          lexemsArray[i][1] <= CHAR_Z ||
                lexemsArray[i][1] == CHAR_UNDERSCORE
            )
        ) begin
            $write("SyntaxError: Invalid label syntax - '");
            for (j = 0; j != lexemsSizesArray[i]; ++j) begin
                $write("%s", lexemsArray[i][j]);
            end
            $write("'. Must be - ':[a-zA-Z_][a-zA-Z0-9_-]*'.\n");
            $finish(1);
```

```
end
// проверка остальных символов идентификатора
for (j = 2; j != lexemsSizesArray[i]; ++j) begin
    if (!(
        CHAR_a <= lexemsArray[i][j] &&
                  lexemsArray[i][j] <= CHAR_z  ||
        CHAR_A <= lexemsArray[i][j] &&
                  lexemsArray[i][j] <= CHAR_Z  ||
        CHAR_0 <= lexemsArray[i][j] &&
                  lexemsArray[i][j] <= CHAR_9  ||
        lexemsArray[i][j] == CHAR_UNDERSCORE                          ||
        lexemsArray[i][j] == CHAR_HYPHEN
    )) begin
        $write("SyntaxError: Invalid label syntax - '");
        for (k = 0; k != lexemsSizesArray[i]; ++k) begin
            $write("%s", lexemsArray[i][k]);
        end
        $write("'. Must be - ':[a-zA-Z_][a-zA-Z0-9_-]*'.\n");
        $finish(1);
    end
end
// проверка уникальности метки
for (j = 0; j != labelsCount; ++j) begin
    if (lexemsSizesArray[i] == labelsSizesArray[j]) begin
        isEqual = 1;
        for (k = 0; k != lexemsSizesArray[i]; ++k) begin
            if (lexemsArray[i][k] != labelsArray[j][k]) begin
                isEqual = 0;
                k = lexemsSizesArray[i] - 1;  // break
            end
        end
        if (isEqual == 1) begin
            $write("SyntaxError: This label yet exists - ");
            for (k = 0; k != lexemsSizesArray[i]; ++k) begin
                $write("%s", lexemsArray[i][k]);
            end
            $write(".\n");
            $finish(1);
        end
    end
end
// проверка на переполнение массива меток
if (labelsCount == `MAX_LABELS) begin
    $write("SyntaxError: Too many labels. Max size - %d.\n",
                                                    `MAX_LABELS);
    $finish(1);
```

```
        end

        // переписываем метку
        labelsSizesArray[labelsCount] = lexemsSizesArray[i] - 1;
        for (j = 1; j != lexemsSizesArray[i]; ++j) begin
            labelsArray[labelsCount][j - 1] = lexemsArray[i][j];
        end
        // сохраняем адрес памяти
        labelsValuesArray[labelsCount] = current;
        ++labelsCount;
        --current;  // не учитываем метки при подсчете слов программы
    end
    ++current;
end
// записываем значение переменной PROGRAM_SIZE
program_size = current;
labelsValuesArray[0] = program_size;

// // <<<<<======= ДЕБАГ: печать массива сохраненный мнемотик и меток
// for (i = 0; i != labelsCount; ++i) begin
//     for (j = 0; j != labelsSizesArray[i]; ++j) begin
//         $write("%s", labelsArray[i][j]);
//     end
//     $write(": value-%d\n", labelsValuesArray[i]);
// end

// второй проход ==========>
current = 256;
for (i = 0; i != lexemsCount; ++i) begin

    // игнорируем метки
    if (lexemsArray[i][0] == CHAR_COLON) begin
        --current;  // не оставляем дыр в памяти от меток
    end else if (  // распознаем первый симол числа
        lexemsArray[i][0] == CHAR_PLUS  ||
        lexemsArray[i][0] == CHAR_MINUS ||
        CHAR_0 <= lexemsArray[i][0] &&
                  lexemsArray[i][0] <= CHAR_9
    ) begin
        isNegativ = 0;
        j = 0;
        if (lexemsArray[i][0] == CHAR_MINUS) begin
            isNegativ = 1;
            j = 1;
        end else if (lexemsArray[i][0] == CHAR_PLUS) begin
            j = 1;
```

```
        end
        // пустая последовательность десятичным цифр
        if (j == lexemsSizesArray[i]) begin
            $write("SyntaxError: Invalid digit - '");
            for (k = 0; k != lexemsSizesArray[i]; ++k) begin
                $write("%s", lexemsArray[i][k]);
            end
            $write("'. Must be - '[+-]?[0-9]+'.\n");
            $finish(1);
        end
        // преобразуем строку в число
        memmory[current] = 0;
        for (j = j; j != lexemsSizesArray[i]; ++j) begin
            memmory[current] *= 10;
            memmory[current] += (lexemsArray[i][j] - CHAR_0);
        end
        if (isNegativ == 1) begin
            memmory[current] *= -1;
        end
    end else if (  // распознаем первый символ идентификатора
        CHAR_a <= lexemsArray[i][0] &&
                  lexemsArray[i][0] <= CHAR_z ||
        CHAR_A <= lexemsArray[i][0] &&
                  lexemsArray[i][0] <= CHAR_Z ||
        lexemsArray[i][0] == CHAR_UNDERSCORE
    ) begin
        // распознаем остальные симолы идентификатора
        for (j = 1; j != lexemsSizesArray[i]; ++j) begin
            if (!(
                CHAR_a <= lexemsArray[i][j] &&
                          lexemsArray[i][j] <= CHAR_z  ||
                CHAR_A <= lexemsArray[i][j] &&
                          lexemsArray[i][j] <= CHAR_Z  ||
                CHAR_0 <= lexemsArray[i][j] &&
                          lexemsArray[i][j] <= CHAR_9  ||
                lexemsArray[i][j] == CHAR_UNDERSCORE                          ||
                lexemsArray[i][j] == CHAR_HYPHEN
            )) begin
                $write("SyntaxError: Invalid identificator syntax - '");
                for (k = 0; k != lexemsSizesArray[i]; ++k) begin
                    $write("%s", lexemsArray[i][k]);
                end
                $write("'. Must be - ':[a-zA-Z_][a-zA-Z0-9_-]*'.\n");
                $finish(1);
            end
        end
```

```
        // ищем среди записанных мнемоник и меток нужную
        wasFound = 0;
        for (j = 0; j != labelsCount; ++j) begin
            if (lexemsSizesArray[i] == labelsSizesArray[j]) begin
                isEqual = 1;
                for (k = 0; k != lexemsSizesArray[i]; ++k) begin
                    if (lexemsArray[i][k] != labelsArray[j][k]) begin
                        isEqual = 0;
                        k = lexemsSizesArray[i] - 1;  // break
                    end
                end
                if (isEqual == 1) begin
                    wasFound = 1;
                    // подставляем мнемотику или метку
                    memmory[current] = labelsValuesArray[j];
                    j = labelsCount - 1;  // break
                end
            end
        end
        // ошибка, если идентификатор не найден
        if (wasFound == 0) begin
            $write("SyntaxError: Identificator not exists - '");
            for (j = 0; j != lexemsSizesArray[i]; ++j) begin
                $write("%s", lexemsArray[i][j]);
            end
            $write("'.\n");
            $finish(1);
        end
    end else begin  // лексема не распознана
        $write("SyntaxError: Lexem did not recognized - '");
        for (j = 0; j != lexemsSizesArray[i]; ++j) begin
            $write("%s", lexemsArray[i][j]);
        end
        $write("'. Must be digit - '[+-]?[0-9]+' ");
        $write("or identificator - '[a-zA-Z_][a-zA-Z0-9_-]*'.\n");
        $finish(1);
    end

    // // <<<<<======== ДЕБАГ: итеративная печать исходной лексемы и
    // //               соответсвующего ей записанного в память числа
    // for (j = 0; j != lexemsSizesArray[i]; ++j) begin
    //     $write("%s", lexemsArray[i][j]);
    // end
    // $write("\t%d\n", memmory[current]);
```

```
    ++current;
end


// инциализация состояния старта виртуальной машины
                            // начиная с адреса 256 загружены
                            //          слова пользовательской программы
reg_IP = 256;               // будет выполнена самая первая инструкция
reg_SP = `MEMMORY_LIMIT;    // прочитать со стека ничего нельзя, записать можно
                            // значение регистров FP и RV не определено


// открываем файл с входными данными
fd = $fopen(stdinFile, "r");
if (!fd) begin
    $write("FileError: File '%s' unexists.\n", stdinFile);
    $finish(1);
end

// главный исполняющий цикл машины
forever begin
    // // <<<<<======= ДЕБАГ: печать выполняемой команды и состояния стека
    // $write("lexem~%d\tcmd=%d | ", (reg_IP - 256), memmory[reg_IP]);
    // for (k = `MEMMORY_LIMIT - 1; k != reg_SP - 1; --k) begin
    //     $write("%d ", memmory[k]);
    // end
    // $write(" ...\n");

    // проверка на валидность значения регистра IP
    if (!(256 <= reg_IP && reg_IP < (256 + program_size))) begin
        $write("RuntimeError: register IP must be between 256 and %d.",
                                                (256 + program_size));
        $write(" Your register IP - %d.\n", reg_IP);
        $finish(1);
    end
    // проверка на переполненность стека ;)
    if (!((256 + program_size) <= reg_SP)) begin
        $write("RuntimeError: line~%d: stack overflow.\n", (reg_IP - 256));
        $finish(1);
    end
    // проверка на валидность значения регистра SP
    //      (по идее, никогда не пригодится, достаточно прочих проверок)
    if (!(reg_SP <= `MEMMORY_LIMIT)) begin
        $write("RuntimeError: line~%d: register SP must be between",
                                                    (reg_IP - 256));
        $write(" %d and %d.", (256 + program_size), `MEMMORY_LIMIT);
```

```
    $write(" Your register SP - %d.\n", reg_SP);
    $finish(1);
end

// выбор соответсвующей иструкции по следующему слову в памяти машины
case (memmory[reg_IP])

    CMD_ADD : begin
        // проверка на возможность считывания
        //      нужного количества слов со стека
        if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
            $write(STACK_ERROR, (reg_IP - 256),
                    "ADD", 2, (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end

        // считываем со стека два слова (числа)
        y = memmory[reg_SP++];
        x = memmory[reg_SP++];
        // помещаем на вершину стека результат вычислений
        memmory[--reg_SP] = x + y;
        // переходим к следующей инструкции
        ++reg_IP;
    end

    CMD_SUB : begin
        if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
            $write(STACK_ERROR, (reg_IP - 256),
                    "SUB", 2, (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end

        y = memmory[reg_SP++];
        x = memmory[reg_SP++];
        memmory[--reg_SP] = x - y;
        ++reg_IP;
    end

    CMD_MUL : begin
        if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
            $write(STACK_ERROR, (reg_IP - 256),
                    "MUL", 2, (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end

        y = memmory[reg_SP++];
```

```
    x = memmory[reg_SP++];
    memmory[--reg_SP] = x * y;
    ++reg_IP;
end

CMD_DIV : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "DIV", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    y = memmory[reg_SP++];
    x = memmory[reg_SP++];

    if (y == 0) begin
        $write("RuntimeError: line~%d: Division by zero.\n",
                                                (reg_IP - 256));
        $finish(1);
    end

    memmory[--reg_SP] = x / y;
    ++reg_IP;
end

CMD_MOD : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "MOD", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    y = memmory[reg_SP++];
    x = memmory[reg_SP++];

    if (y == 0) begin
        $write("RuntimeError: line~%d: Division by zero.\n",
                                                (reg_IP - 256));
        $finish(1);
    end

    memmory[--reg_SP] = x % y;
    ++reg_IP;
end

CMD_NEG : begin
```

```
    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "NEG", 1, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    x = memmory[reg_SP++];
    memmory[--reg_SP] = -x;
    ++reg_IP;
end

CMD_BITAND : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "BITAND", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    y = memmory[reg_SP++];
    x = memmory[reg_SP++];
    memmory[--reg_SP] = x & y;
    ++reg_IP;
end

CMD_BITOR : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "BITOR", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    y = memmory[reg_SP++];
    x = memmory[reg_SP++];
    memmory[--reg_SP] = x | y;
    ++reg_IP;
end

CMD_BITNOT : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "BITNOT", 1, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    x = memmory[reg_SP++];
    memmory[--reg_SP] = ~x;
```

```
    ++reg_IP;
end

CMD_LSHIFT : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "LSHIFT", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    y = memmory[reg_SP++];
    x = memmory[reg_SP++];
    memmory[--reg_SP] = x << y;
    ++reg_IP;
end

CMD_RSHIFT : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "RSHIFT", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    y = memmory[reg_SP++];
    x = memmory[reg_SP++];
    memmory[--reg_SP] = x >> y;
    ++reg_IP;
end

CMD_DUP : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "DUP", 1, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    x = memmory[reg_SP++];
    memmory[--reg_SP] = x;
    memmory[--reg_SP] = x;
    ++reg_IP;
end

CMD_DROP : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "DROP", 1, (`MEMMORY_LIMIT - reg_SP));
```

```
            $finish(1);
        end

        x = memmory[reg_SP++];
        ++reg_IP;
    end

    CMD_SWAP : begin
        if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
            $write(STACK_ERROR, (reg_IP - 256),
                    "SWAP", 2, (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end

        y = memmory[reg_SP++];
        x = memmory[reg_SP++];
        memmory[--reg_SP] = y;
        memmory[--reg_SP] = x;
        ++reg_IP;
    end

    CMD_ROT : begin
        if ((`MEMMORY_LIMIT - reg_SP) < 3) begin
            $write(STACK_ERROR, (reg_IP - 256),
                    "ROT", 3, (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end

        z = memmory[reg_SP++];
        y = memmory[reg_SP++];
        x = memmory[reg_SP++];
        memmory[--reg_SP] = y;
        memmory[--reg_SP] = z;
        memmory[--reg_SP] = x;
        ++reg_IP;
    end

    CMD_OVER : begin
        if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
            $write(STACK_ERROR, (reg_IP - 256),
                    "OVER", 2, (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end

        y = memmory[reg_SP++];
        x = memmory[reg_SP++];
```

```
                    memmory[--reg_SP] = x;
                    memmory[--reg_SP] = y;
                    memmory[--reg_SP] = x;
                    ++reg_IP;
                end

                CMD_DROPN : begin
                    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
                        $write(STACK_ERROR, (reg_IP - 256),
                                "DROPN", 1, (`MEMMORY_LIMIT - reg_SP));
                        $finish(1);
                    end
                    if ((`MEMMORY_LIMIT - reg_SP) < memmory[reg_SP] + 1) begin
                        $write(STACK_ERROR, (reg_IP - 256),
                                "DROPN", (memmory[reg_SP] + 1),
                                (`MEMMORY_LIMIT - reg_SP));
                        $finish(1);
                    end

                    reg_SP += memmory[reg_SP] + 1;
                    ++reg_IP;
                end

                CMD_PUSHN : begin
                    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
                        $write(STACK_ERROR, (reg_IP - 256),
                                "PUSHN", 1, (`MEMMORY_LIMIT - reg_SP));
                        $finish(1);
                    end

                    reg_SP -= memmory[reg_SP] - 1;
                    ++reg_IP;
                end

                CMD_LOAD : begin
                    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
                        $write(STACK_ERROR, (reg_IP - 256),
                                "LOAD", 1, (`MEMMORY_LIMIT - reg_SP));
                        $finish(1);
                    end

                    a = memmory[reg_SP++];

                    if (!(256 <= a && a <= `MEMMORY_LIMIT)) begin
                        $write(
"RuntimeError: line~%d: LOAD: memmory adress must be between", (reg_IP - 256));
```

```
                    $write(" 256 and %d.", `MEMMORY_LIMIT);
                    $write(" Your request adress - %d.\n", a);
                    $finish(1);
                end

                memmory[--reg_SP] = memmory[a];
                ++reg_IP;
            end

            CMD_SAVE : begin
                if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
                    $write(STACK_ERROR, (reg_IP - 256),
                            "SAVE", 2, (`MEMMORY_LIMIT - reg_SP));
                    $finish(1);
                end

                v = memmory[reg_SP++];
                a = memmory[reg_SP++];

                if (!(256 + program_size <= a && a <= `MEMMORY_LIMIT)) begin
                    $write("RuntimeError: line~%d: SAVE:\
memmory adress must be between", (reg_IP - 256));
                    $write(" %d and %d.", 256 + program_size, `MEMMORY_LIMIT);
                    $write(" Your request adress - %d.\n", a);
                    $write("[ you cant rewrite your executing program ;) ].\n");
                    $finish(1);
                end

                memmory[a] = v;
                ++reg_IP;
            end

            CMD_GETIP : begin
                memmory[--reg_SP] = reg_IP++;
            end

            CMD_SETIP : begin
                if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
                    $write(STACK_ERROR, (reg_IP - 256),
                            "SETIP", 1, (`MEMMORY_LIMIT - reg_SP));
                    $finish(1);
                end

                reg_IP = memmory[reg_SP++];
            end
```

```
CMD_GETSP : begin
    a = reg_SP;
    memmory[--reg_SP] = a;
    ++reg_IP;
end

CMD_SETSP : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "SETSP", 1, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    reg_SP = memmory[reg_SP++];

    if (!(256 + program_size <= reg_SP &&
                                reg_SP <= `MEMMORY_LIMIT)) begin
        $write("RuntimeError: line~%d: SETSP: value of register");
        $write(" SP must be between %d and %d.",
                256 + program_size, `MEMMORY_LIMIT);
        $write("But register SP was set with %s.\n", reg_SP);
        $finish(1);
    end

    ++reg_IP;
end

CMD_GETFP : begin
    memmory[--reg_SP] = reg_FP;
    ++reg_IP;
end

CMD_SETFP : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "SETFP", 1, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    reg_FP = memmory[reg_SP++];
    ++reg_IP;
end

CMD_GETRV : begin
    memmory[--reg_SP] = reg_RV;
    ++reg_IP;
```

```
end

CMD_SETRV : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "SETRV", 1, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    reg_RV = memmory[reg_SP++];
    ++reg_IP;
end

CMD_CMP : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "CMP", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    y = memmory[reg_SP++];
    x = memmory[reg_SP++];

    if (x < y) begin
        memmory[--reg_SP] = -1;
    end else if (x == y) begin
        memmory[--reg_SP] = 0;
    end else begin
        memmory[--reg_SP] = 1;
    end

    ++reg_IP;
end

CMD_JMP : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "JMP", 1, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    reg_IP = memmory[reg_SP++];
end

CMD_JLT : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
```

```
            $write(STACK_ERROR, (reg_IP - 256),
                    "JLT", 2, (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end

        a = memmory[reg_SP++];
        x = memmory[reg_SP++];
        reg_IP = ((x < 0) ? a : (reg_IP + 1));
    end

    CMD_JGT : begin
        if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
            $write(STACK_ERROR, (reg_IP - 256),
                    "JGT", 2, (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end

        a = memmory[reg_SP++];
        x = memmory[reg_SP++];
        reg_IP = ((x > 0) ? a : (reg_IP + 1));
    end

    CMD_JEQ : begin
        if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
            $write(STACK_ERROR, (reg_IP - 256),
                    "JEQ", 2, (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end

        a = memmory[reg_SP++];
        x = memmory[reg_SP++];
        reg_IP = ((x == 0) ? a : (reg_IP + 1));
    end

    CMD_JLE : begin
        if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
            $write(STACK_ERROR, (reg_IP - 256),
                    "JLE", 2, (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end

        a = memmory[reg_SP++];
        x = memmory[reg_SP++];
        reg_IP = ((x <= 0) ? a : (reg_IP + 1));
    end
```

```
CMD_JGE : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "JGE", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    a = memmory[reg_SP++];
    x = memmory[reg_SP++];
    reg_IP = ((x >= 0) ? a : (reg_IP + 1));
end

CMD_JNE : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "JNE", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    a = memmory[reg_SP++];
    x = memmory[reg_SP++];
    reg_IP = ((x != 0) ? a : (reg_IP + 1));
end

CMD_CALL : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "CALL", 1, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end

    a = memmory[reg_SP++];
    memmory[--reg_SP] = reg_IP + 1;
    reg_IP = a;
end

CMD_RETN : begin
    if ((`MEMMORY_LIMIT - reg_SP) < 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "RETN", 2, (`MEMMORY_LIMIT - reg_SP));
        $finish(1);
    end
    if ((`MEMMORY_LIMIT - reg_SP) < memmory[reg_SP] + 2) begin
        $write(STACK_ERROR, (reg_IP - 256),
                "RETN", memmory[reg_SP] + 2,
                (`MEMMORY_LIMIT - reg_SP));
```

```
            $finish(1);
        end

        N = memmory[reg_SP++];
        a = memmory[reg_SP++];
        reg_SP += N;
        reg_IP = a;
    end

    CMD_IN : begin
        char = $fgetc(fd);
        memmory[--reg_SP] = char;
        ++reg_IP;
    end

    CMD_OUT : begin
        if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
            $write(STACK_ERROR, (reg_IP - 256),
                    "OUT", 1, (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end

        char = memmory[reg_SP++];
        $write("%s", char);
        ++reg_IP;
    end

    CMD_HALT : begin
        if ((`MEMMORY_LIMIT - reg_SP) < 1) begin
            $write(STACK_ERROR, (reg_IP - 256),
                    "HALT", 1, (`MEMMORY_LIMIT - reg_SP));
            $finish(1);
        end

        a = memmory[reg_SP++];
        $write("\n\nProgram terminated with code %d.\n", a);
        ++reg_IP;
        $finish;
    end

    default : begin
                                        // неотрицательное число
        if (memmory[reg_IP] >= 0) begin  // помещаем на вершину стека
            memmory[--reg_SP] = memmory[reg_IP++];
                        // ошибка, если слово не распознано ни как
        end else begin // интсрукция, ни как неотрицательное число
```

```
                    $write("RuntimeError: line~%d: Unknown command - '%d'.\n",
                                        (reg_SP - 256), memmory[reg_IP]);
                    $finish(1);
                end
            end
        endcase

    end
  end
endmodule
```

Реализация

```
main CALL
HALT


:main                                   ; ret
    0 0                                 ; ret wordCount isWord
    :loop
        IN                              ; ret wordCount isWord char

        DUP 10 CMP loop_exit JEQ        ; loop exit if '\n'
        DUP 13 CMP loop_exit JEQ        ; loop exit if '\r'

        DUP 32 CMP not_space JEQ        ; skip if ' '
        DUP 9 CMP not_space JEQ         ; skip if '\t'
        DUP 44 CMP not_space JEQ        ; skip if ','
        DUP 46 CMP not_space JEQ        ; skip if '.'
        DUP 59 CMP not_space JEQ        ; skip if ';'
        DUP 63 CMP not_space JEQ        ; skip if '?'
        DUP 33 CMP not_space JEQ        ; skip if '!'
            DROP                        ; ret wordCount isWord
            DUP 0 CMP not_first_char JNE    ; skip if char not first in wordCount
                DROP                    ; ret wordCount
                1 ADD                   ; ret wordCount+1
                1                       ; ret wordCount+1 isWord=1
            :not_first_char
                if_exit JMP
        :not_space                      ; ret wordCount isWord
            2 DROPN                     ; ret wordCount
            0                           ; ret wordCount isWord=0
        :if_exit                        ; ret wordCount isWord

        loop JMP
```

```
    :loop_exit                              ; ret wordCount isWord char
    2 DROPN                                 ; ret wordCount
    write_int CALL                          ; ret
    0 SWAP                                  ; 0 ret
    0 RETN                                  ; 0

:write_int      ; x ret
SWAP            ; ret x
DUP 10 MOD      ; ret x x%10
SWAP 10 DIV     ; ret x%10 x/10
DUP             ; ret x%10 x/10 x/10
write_int_skip_prefix JEQ
                ; ret x%10 x/10
write_int CALL  ; ret x%10
100500          ; ret x%10 100500

:write_int_skip_prefix
                ; ret x%10 ?
DROP            ; ret x%10
48 ADD OUT      ; ret
0 RETN          ; пусто
```

Тестирование

```
> iverilog -o lab1 lab1.vl
> vvp lab1
> 6 10 20 30 5 100 15  aA 1b  cAd,2eA.fj3;;hAi?g4k??lA!5mn  oA6 ?!!.pqrA!7stuA8vwxA9!yz
>
> 20
```

Вывод

В результате данной лабораторной работы были обретены навыки написания интерпретатора ассемблера модельного компьютера, который будет использоваться в последующих лабораторных как целевой язык, а также написания программ на языке модельного ассемблера.