VILNIUS GEDIMINAS
TECHNICAL UNIVERSITY
FACULTY OF FUNDAMENTAL SCIENCES

Virtual Machine

LECTURER: DR. PAVEL STEFANOVIČ

VILNIUS, 2017

5 laboratory work (1)

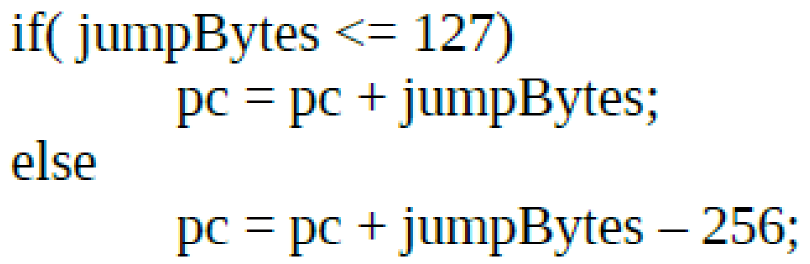
- Using programming language of your choice make a virtual machine that:
  - has 16 registers *R0-R15* (8 bits wide);
  - has PC (program counter) register that points to the currently executed instruction. After execution it has to increase by 2. In case of jump instructions, jump bytes (displacement address) is added to PC, thus virtual machine performs jump to different address;
  - has 8 bit addressing mode: it addresses program memory at byte level;
  - has 256 byte program memory;
  - has flag register for holding 1 bit of data – *EOF* mark for *IN* command.

5 laboratory work (2)

- The task includes two files:
    - **decryptor.bin**, which contains binary program for execution;
    - **q1_encr.txt** – encrypted file that machine reads byte by byte when it executes *IN* command.
- Task is considered as completed when machine reads input and by executing given program performs input decryption and outputs English text – a sentence of known scientist.
- You can use any programming language, but *C/C++* this time is recommended. All specification of VM will be given in *C* language.

5 laboratory work (3)

- If You will not use *C/C++* it is possible that signed byte is not available in your language. Then, for calculation *JMP* addresses (which are relative to the current program counter) use following expression:



```
if( jumpBytes <= 127)
        pc = pc + jumpBytes;
else
        pc = pc + jumpBytes – 256;
```

- Note: when reading file for *IN* command, open it using BINARY mode. By coincidence it contains byte values that in text mode denotes end of the file, thus file in text mode might not be read until the end.

Specification of Virtual machine (1)

- There are 2 types of the commands:
  - **1 type (command have two parameters).** These commands typically work with registers and contain register references:

8 bits Command code	
Bits 7-4 Register 2	Bits 3-0 Register 1

- Some commands, like *LSL*, *INC* does not have register *R2*, so value in bits 7-4 then is ignored.

Specification of Virtual machine (2)

- **2 type (command have one parameter):** 1 byte command code, 1 bytes of signed address to add to PC and jump to resulting memory address or constant (for *MOVC* command)

8 bits Command code
8 bits Address of jump or constant

Commands of Virtual machine (1)

Command	Cmd code	Comment
INC Reg	0x01	Increments any register referred by Reg
DEC Reg	0x02	Decrements any register referred by Reg
MOV Reg1, Reg2	0x03	Copies register Reg2 to other register Reg1
MOVC byte const	0x04	Copies byte constant to the register Reg 0. This is type 2 command
LSL Reg	0x05	Shift to the left.
LSR Reg	0x06	Shift to the right.
JMP addr	0x07	Jumps to relative address by adding command type 2 signed value (addr) to program counter (PC). This and all the jumps are type 2 commands.
JZ addr	0x08	Same as JMP, but if zero flag is set
JNZ addr	0x09	Same as JMP, but if zero flag is not set

Commands of Virtual machine (2)

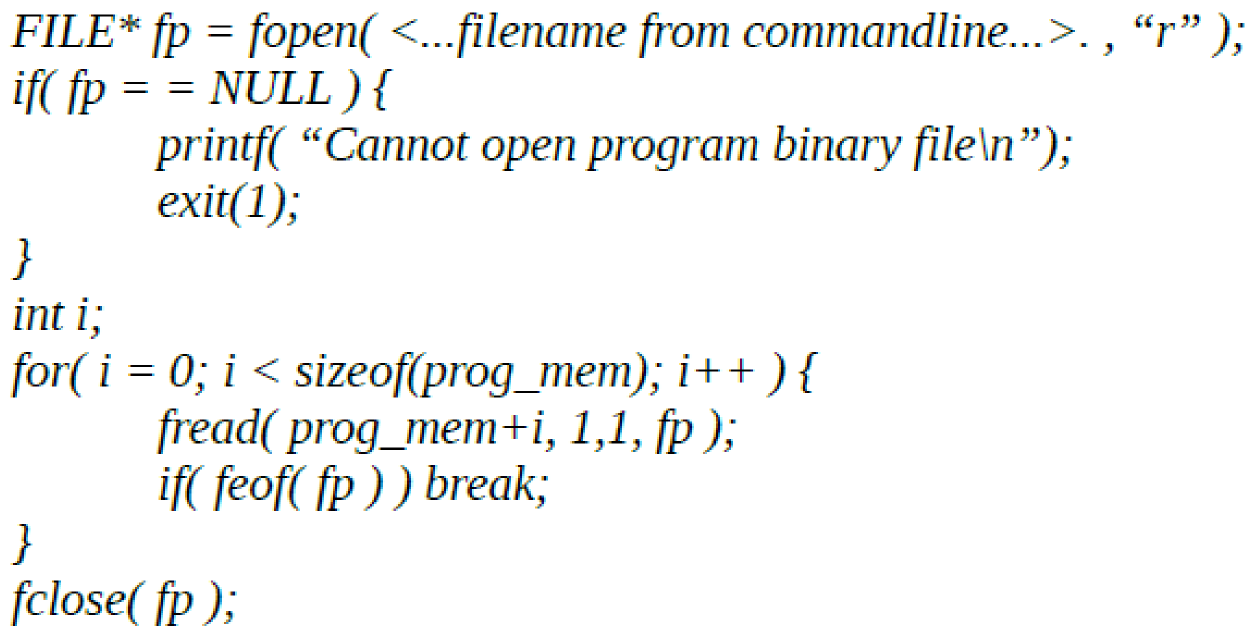
JFE	0x0A	Same as JMP, but if file end with IN command has been reached. See IN command
RET	0x0B	Exits execution of VM code
ADD Reg1, Reg2	0x0C	Adds Reg2 content to Reg1 leaving result in Reg1
SUB Reg1,Reg2	0x0D	Subtracts Reg2 from Reg1, stores result to Reg1
XOR Reg1,Reg2	0x0E	Bitwise XOR (exclusive OR) operation, result is stored into Reg1
OR Reg1,Reg2	0x0F	Bitwise OR operation, result is stored into Reg1
IN Reg	0x10	Reads 1 byte from input data file and sets file end flag if end of file has been reached
OUT Reg	0x11	Outputs contents of register Reg to the screen

Possible implementation scenario in C language: (1)

- Define registers as an array of char: *unsigned char regs[16];*
- Define program memory as an array of char: *char prog_mem[256];*
- Define flag register as simple int variable, as luckily Your machine does not have *INT* command, which would have to push Your flag register to the stack (note, that stack as well as data memory is absent here as well): *int ieof_flag*;

Possible implementation scenario in C language: (2)

- Fill program memory using *fopen*, *fread* and *fclose* functions (defined in stdio.h). Because of reason that in machine binary program there are *ASCII* control codes *(<0x20)*, so the program have to open in binary mode ("*rb*"):



```
FILE* fp = fopen( <...filename from commandline...>. , "r" );
if( fp = = NULL ) {
        printf( "Cannot open program binary file\n");
        exit(1);
}
int i;
for( i = 0; i < sizeof(prog_mem); i++ ) {
        fread( prog_mem+i, 1,1, fp );
        if( feof( fp ) ) break;
}
fclose( fp );
```

Possible implementation scenario in C language: (3)

- Define VM command as structure:

```
typedef struct {
        char code;
        char cop1;
} VMCommand;
```

- Your PC register might be simple pointer of command type. On the run You can assign it to the start of program memory. Note, that on each increase of PC in this case You will skip by size of *VMCommand* structure which is 2 bytes in this case:

```
VMCommand* r_pc;
r_pc = (VMCommand*) prog_mem;
```

Possible implementation scenario in C language: (4)

- Before each command execution You may extract register 1 and register 2 references and use them as required. Define register references at the top:

```
int iReg1, iReg2;
```

- Extract the from command, having in mind that PC register is pointing to Your current command already:

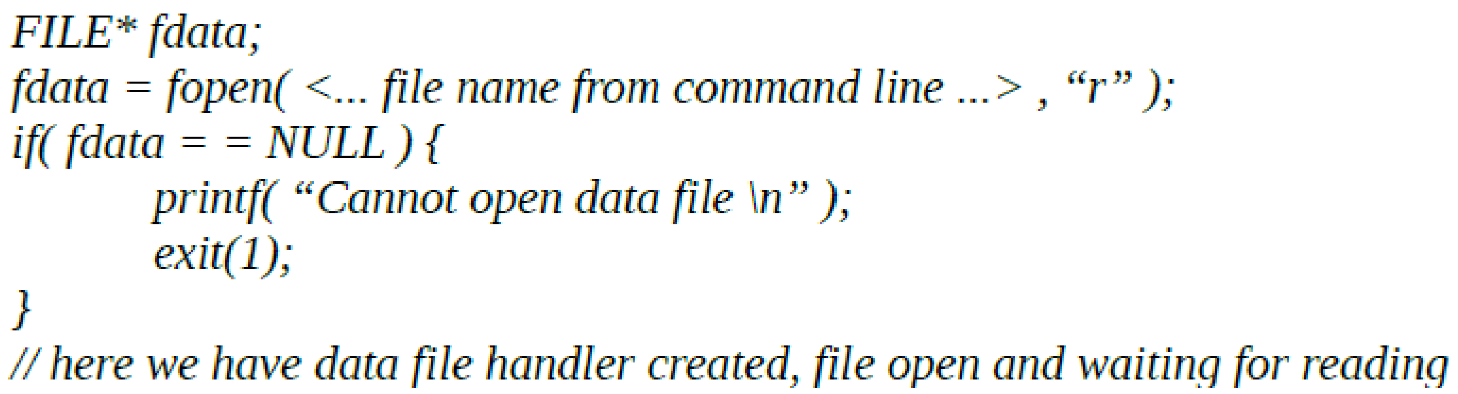
```
iReg1 = r_pc->cop1 & 0x0F;
iReg2 = r_pc->( cop1 & 0xF0 ) >> 4 ; // or divide by 16 which is the same, as You know
```

- After each command execution You typically increase PC, note that it will increase by 2 bytes, as *C* calculates it having in mind the size of structure *VMCommand*. This is called pointer arithmetic's:

```
r_pc++; // Now r_pc points to the next command
```

Possible implementation scenario in C language: (5)

- There is a special treatment for *IN* command which is not typical *IN* for a CPU. *IN* reads byte from the data file and puts it into register referred by the command. Means, *IN R1* reads byte into *regs[1]*. You can open data file prior the execution starts using simple fopen:



```
FILE* fdata;
fdata = fopen( <... file name from command line ...> , “r” );
if( fdata = = NULL ) {
        printf( “Cannot open data file \n” );
        exit(1);
}
// here we have data file handler created, file open and waiting for reading
```

Possible implementation scenario in C language: (6)

- So, when You are processing *IN* command, having in mind that You have been set *iReg* with correctly obtained value from the command, code piece for *IN* might be simply as follows:

```
fread( regs[ iReg1], 1, 1, fdata );
if( feof( fdata ) ) {
        ieof_flag = 1;
} else {
        ieof_flag = 0;
}
```

- The flag's value should be analyzed by *JFE* command – i.e. command should be executed only if the flag is set.

Possible implementation scenario in C language: (7)

- There is a little notification about *JMP* command: It has 1 byte value after command code to obtain jump address. The value is in bytes, and it is signed. Problem is that we have defined *r_pc* register as a pointer to the *VMCommand*. To deal with this, You have to convert *r_pc* to the pointer of char type, add byte value from the command and convert it to *VMCommand* pointer back (Note, that *r_pc->cop1*, as the second byte of command of type 2 might contain negative value):

```
r_pc = (VMCommand*) (  (char*) r_pc + r_pc->cop1 );
```