

Virtual Machine

Dr. Nguyen Hua Phung

February 2021

1 Introduction

Virtual machine is a software program that can run on a physical machine to function as a virtual computer system with its own set of instructions together with other components.

Virtual machine has many different applications:

- Simulator/emulator of a real physical computer system so that softwares can be executed on. VirtualBox or VMWare is such an example.
- New platform to execute its program equally in different physical machines. Java virtual machine is used to execute Java program equally in different computer systems.
- Applications use complex data. Microsoft Word is such an application which uses many different kinds of data (text, image, table, format,...)

The following sections will describe the structure and set of instructions in a virtual machine used for assignments in this course.

2 Virtual machine structure

The virtual machine includes a CPU, a memory bank for code, a static memory bank, a cache and a stack for return addresses. The CPU executes instructions in the memory bank of code which keeps the code of executed program; the static memory keeps the variables used in the program; the stack of return addresses keeps return addresses of call instructions; the cache, which is used to optimize, keeps a map of address and value

The CPU has 16 registers including 1 special (IP) and 15 general registers (from R1 to R15). The IP register keeps an address in the memory bank for code. The general registers can keep an address or data in integer, float, or boolean type.

To use the virtual machine, a program written in a text file including a chain of instructions described in Section 3 must be firstly loaded into the memory bank for code. Each instruction is put in a slot of the memory bank for code

which has 65536 slots addressed from 0. The virtual machine then lets the CPU repeatedly perform the following steps:

1. read the instruction at address stored in IP register which is always initialized to 0.
2. increase the address in IP register by 1.
3. execute the instruction read in step 1. The meaning of each instruction described in Section 3 presents how to execute the corresponding instruction.

The static memory has also 65536 slots, each of which can keep data in integer, float, or boolean type. The slots in the static memory are addressed from 0 to 65535. The value in each slot can be changed during the execution of the program.

The stack of return address just keeps the address of instructions in the memory bank for code. The size of the stack is 1000.

The next section will describe the detail of the instructions used by the virtual machine.

3 Set of instructions

An instruction is written in a line and always starts with a code. In addition, an instruction may have zero, one or two operands. The first operand of the code, if any, will be separated with the code by exactly one space and the second operand of the code, if any, will be separated with the first operand by a comma and a space. There is no other tailing character.


In the following description of instruction, an operand, whose name is "dest", can be a general register only while an operand whose name is "src" can be a general register or a literal. An integer literal can be written as a decimal integer constant in C++; a float literal can be written by a sequence of digits and a dot and another sequence of digits. These sequences cannot be empty. A boolean literal is true or false. An address literal can be written like an integer constant but ending with the character A.

There are 6 groups of instructions which will be described in the following subsections. In each subsection, the first paragraph describes the expected type of operands of the instructions in this group. Then, all instructions in this group are listed where the first part (before colon) is the instruction and the second part (after colon) is the meaning of the corresponding instruction.

3.1 Arithmetic instructions

All instructions in this group use two operands as input and put the result in the first operand. The type of these operands can be integer or float and the type of the result is integer just when both input operands are integer, otherwise, the type of the result is float.

The instructions in this group include:

- **Add** dest, src: $\text{dest} = \text{dest} + \text{src}$
- **Minus** dest, src: $\text{dest} = \text{dest} - \text{src}$
- **Mul** dest, src: $\text{dest} = \text{dest} * \text{src}$
- **Div** dest, src: $\text{dest} = \text{dest} / \text{src}$ 

3.2 Comparison instructions

The type of the operands of these instructions can be integer, float or boolean type. If the operand type is boolean, both operands must be in the same type. Otherwise, the operand type can be mixed. The result type always is boolean type.

The instructions in this group include:

- **CmpEQ** dest, src: if $(\text{dest} == \text{src})$ $\text{dest} = \text{true}$ else $\text{dest} = \text{false}$
- **CmpNE** dest, src: if $(\text{dest} != \text{src})$ $\text{dest} = \text{true}$ else $\text{dest} = \text{false}$
- **CmpLT** dest, src: if $(\text{dest} < \text{src})$ $\text{dest} = \text{true}$ else $\text{dest} = \text{false}$
- **CmpLE** dest, src: if $(\text{dest} \leq \text{src})$ $\text{dest} = \text{true}$ else $\text{dest} = \text{false}$
- **CmpGT** dest, src: if $(\text{dest} > \text{src})$ $\text{dest} = \text{true}$ else $\text{dest} = \text{false}$
- **CmpGE** dest, src: if $(\text{dest} \geq \text{src})$ $\text{dest} = \text{true}$ else $\text{dest} = \text{false}$

3.3 Logical instructions

The type of the operands of these instructions can be boolean type only and the result type is also boolean.

The instructions in this group include:

- **Not** dest: $\text{dest} = !\text{dest}$
- **And** dest, src: $\text{dest} = \text{dest} \&\& \text{src}$
- **Or** dest, src: $\text{dest} = \text{dest} || \text{src}$

3.4 Load and store instructions

The type of **src** and **dest** are specified in each instruction

The instructions in this group include:

- **Move** dest, src: $\text{dest} = \text{src}$; the type of **src** and **dest** can be any type
- **Load** dest, src: $\text{dest} = *\text{src}$; the type of **src** is address while the type of **dest** can be any type. The adress used in this instruction is the address in the static memory.

- **Store** dest, src: *dest = src ; the type of **dest** is address while the type of **src** can be any type. The address used in this instruction is the address in the static memory.

3.5 Sequence control instructions

The type of **src** is address type while the type of **dest** is boolean. The address used in this group is the address in the memory bank for code.

The instructions in this group include:

- **Jump** src: IP = src
- **JumpIf** dest, src: if (dest) IP = src
- **Call** src: push value of IP register onto stack and then IP = src
- **Return**: pop value out of stack and change value of IP register to it.
- **Halt**: stop running

3.6 Input and Output instructions

The type of **dest** of the instructions in this group can be integer, float or boolean while the type of **src** of the instructions in this group can be any type (integer, float, boolean or address).

The instructions in this group include:

- **Input** dest: cin >> dest
- **Output** src: cout << src

4 Error

An exception must be thrown with the address of the instruction which has the error described as follows:

4.1 Loading error

The loading error happens when the virtual loads the input program from a text file.

- InvalidInstruction(address): an instruction does not comply with the description of an instruction or the code in the instruction is not listed or the number of operands does not match with the description of the corresponding code. For example, the instruction has an extra space before the code or after the last operand.
- InvalidOperand(address): this error happens when

- operand **dest** does not receive a general register, or
- operand **src** does not receive a general register or a literal, or
- the literal in the instruction does not comply with the description of literal in Section 3.

4.2 Execution Error

The execution error happens when the CPU executes an instruction in the memory bank for code.

- `TypeMismatch(address)`: the input operands of the instruction does not comply with the expected types of operands.
- `InvalidInput(address)`: the input value of **Input** instruction does not comply with the description of a literal in Section 3.
- `InvalidDestination(address)`: an address given in **src** operand of the instruction **Jump**, **JumpIf**, or **Call** is not in the program.
- `DivideByZero(address)`: the value of **src** operand of **Div** instruction is 0(int type) or 0.0(float type).
- `StackFull(address)`: the stack of the virtual machine becomes full so it cannot accept the push of the instruction at the address.

Assume that there is no other kind of error.