Core War

Marcin Puc, Piotr Komorowski

1. Project description

Core War is a game in which two concurrently executing programs compete in a shared memory of a virtual computer („The Core”). In order to defeat the opponent, its program must be forced to execute a special instruction or an illegal operation.

The programs are implemented in a special assembly-like language which provides a set of instructions performing memory manipulation. Partaking in the duel requires a player to write a program („a warrior”) in the provided language, compile it and execute it in the virtual machine. For this project, a language, a compiler and a virtual machine will be created.

2. Definitions

A virtual computer supporting the game comprises a block of memory whose cells contain instructions. The block is cyclical i.e. after the last cell the execution returns to the first cell. Programs are loaded into the block at random positions. During the execution only relative addressing is available – an absolute address of any cell cannot be retrieved.

Every executed program includes one or more processes, where a process is defined as a pointer to a block cell. Processes for each program are stored in queues and called alternately. A „warrior” loses when all his processes have been removed from the queue. If the number of executed cycles exceeds a set value before one of the warriors emerges victorious, the duel ends with a tie.

Instructions consist of three main elements: *OpCode*, *A-field* and *B-field*. The OpCode identifies a physical instruction executed by the VM, while the A- and B-fields store numerical data e.g. jump address.

3. Programming language

In the original version of the game programs are written in „Redcode”, a language developed by D. G. Jones and A. K. Dewdney. Its instruction set and syntax will serve as a model in this project.

Example instructions:

- `MOV` – transfers data between memory locations
- `ADD` – arithmetic addition
- `MUL` – arithmetic multiplication
- `JMP` – jump to another instruction
- `FRK` – create new process
- `KIL` – end current process and remove it from its queue

4. Compiler

The compiler analyses program code written in a human-readable form of the language and converts it to a file to be processed by the VM.

Example compiler functionalities:

- replacing labels with relative instruction addresses
- precalculation of addresses to a format usable by the VM
- error detection

5. Virtual machine

The VM accepts compiled files and executes programs by loading their code into memory. Internally, the memory block is implemented as an array of instructions. The VM initializes warriors by adding one process pointing to a specified instruction to each program’s queue. Subsequently, the game is conducted according to the rules in the *Definitions* section. After execution the VM displays the duel outcome.