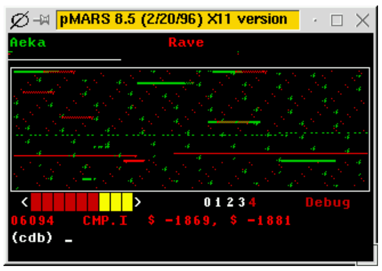
Core War

Core War



```
06094   CMP.I  $ -1869, $ -1881
(cdb) _
```

A game of Core War running under the pMARS simulator

Original author(s)	D. G. Jones & A. K. Dewdney
Initial release	1984
Type	Programming game

Core War is a programming game created by D. G. Jones and A. K. Dewdney in which two or more battle programs (called "warriors") compete for control of a virtual computer. These battle programs are written in an abstract assembly language called *Redcode*. At the beginning of a game, each battle program is loaded into memory at a random location, after which each program executes one instruction in turn. The object of the game is to cause the processes of opposing programs to terminate (which happens if they execute an invalid instruction), leaving the victorious program in sole possession of the machine.

Gameplay

Key features

No numeric instruction values

> The Redcode standard leaves the underlying representation of instruction codes undefined and provides no means for programs to directly access it. Arithmetic operations may only be done on the two address fields contained in each instruction. The only operations supported on the instruction codes themselves are copying and comparing for equality.

No absolute addressing

> Only relative addressing is used. All addresses are interpreted as offsets relative to the instruction containing them. That is, address *0* always refers to the currently executing instruction, address *1* to the instruction after it, and so on. Since the address space wraps around, it is in fact impossible for a Redcode program to determine its absolute address.

Low level multiprocessing

> Instead of a single instruction pointer a Redcode simulator has a *process queue* for each program containing a variable number of instruction pointers which the simulator cycles through. Each program starts with only one process, but new processes may be added to the queue using the `SPL` instruction. A process dies when it executes a `DAT` instruction or performs a division by zero. A program is considered dead when it has no more processes left.

No external access

Redcode and the MARS architecture provide no input or output functions. The simulator is a closed system, with the only input being the initial values of the memory and the process queues, and the only output being the outcome of the battle, i.e., which programs had surviving processes. Of course, the simulator may still allow external inspection and modification of the memory while the simulation is running.

Constant instruction length and time

Each Redcode instruction occupies exactly one memory slot and takes exactly one cycle to execute. The rate at which a process executes instructions, however, depends on the number of other processes in the queue, as processing time is shared equally. The memory is addressed in units of one instruction.

Relatively few instructions

The earliest published version of Redcode defined only 8 instructions. The ICWS-86 standard increased the number to 10 while the ICWS-88 standard increased it to 11. The currently used ICWS-94 standard has 16 instructions. However, Redcode supports a number of different addressing modes and (from ICWS-94) instruction modifiers which increase the actual number of operations possible to 7168.

All addresses are valid

All numbers in Redcode are treated as unsigned integers, and the maximum integer value is set to equal one less than the number of memory locations. Therefore, there is a one-to-one correspondence between numbers and memory locations. Numbers that would fall outside the valid range are wrapped around according to the usual rules of modulo arithmetic.

Circular memory

As a consequence of the above and the lack of absolute addressing, the memory space (or *core*) appears to the programs in it as a circle with no definite start or end. A process that encounters no invalid or jump instructions can continue executing successive instructions endlessly, eventually returning to the instruction where it started.

Versions of Redcode

A number of versions of Redcode exist. The earliest version described by A. K. Dewdney[] differs in many respects from the later standards established by the International Core War Society, and could be considered a different, albeit related, language. The form of Redcode most commonly used today is based on a draft standard submitted to the ICWS in 1994 that was never formally accepted, as the ICWS had become effectively defunct around that time. Development of Redcode, however, has continued in an informal manner, chiefly via online forums such as the `rec.games.corewar`[1] newsgroup.

Strategy

Warriors are commonly divided into a number of broad categories, although actual warriors may often combine the behavior of two or more of these. Three of the common strategies (*replicator*, *scanner* and *bomber*) are also known as paper, scissors and stone, since their performance against each other approximates that of their namesakes in the well-known playground game.[]

Paper (or replicator)

A replicator makes repeated copies of itself and executes them in parallel, eventually filling the entire core with copies of its code. Replicators are hard to kill, but often have difficulty killing their opponents. Replicators therefore tend to score a lot of ties, particularly against other replicators.

A **silk** is a special type of very rapid replicator, named after *Silk Warrior*[2] by Juha Pohjalainen. Most modern replicators are of this type. Silk replicators use parallel execution to copy their entire code with one instruction, and begin execution of the copy before it is finished.[3]

Scissors (or scanner)

A scanner is designed to beat replicators. A scanner does not attack blindly, but tries to locate its enemy before launching a targeted attack. This makes it more effective against hard-to-kill opponents like replicators, but also leaves it vulnerable to decoys. A scanner usually bombs memory with `SPL 0` instructions. This causes the enemy to create a huge number of processes which do nothing but create more processes, slowing down useful processes. When the enemy becomes so slow that it is unable to do anything useful, the memory is bombed with `DAT` instructions. Scanners are also generally more complex, and therefore larger and more fragile, than other types of warriors.[4]

A **one-shot** is a very simple scanner that only scans the core until it finds the first target, and then permanently switches to an attack strategy, usually a core clear. *Myrmidon*[5] by Roy van Rijn is an example of a oneshot.

Stone (or bomber)

A bomber blindly copies a "bomb" at regular intervals in the core, hoping to hit the enemy. The bomb is often a `DAT` instruction, although other instructions, or even multi-instruction bombs, may be used. A bomber can be small and fast, and they gain an extra edge over scanning opponents since the bombs also serve as convenient distractions. Bombers are often combined with imp spirals to gain extra resiliency against replicators.

Vampire (or pit-trapper)

A vampire tries to make its opponent's processes jump into a piece of its own code called a "pit". Vampires can be based on either bombers or scanners. A major weakness of vampires is that they can be easily attacked indirectly, since they must by necessity scatter pointers to their code all over the core. Their attacks are also slow, as it takes an extra round for the processes to reach the pit. *myVamp*[6] by Paulsson is an example of a vampire.

Imp

Imps are named after the first ever published warrior, *Imp*[7] by A. K. Dewdney, a trivial one-instruction mobile warrior that continually copies its sole instruction just ahead of its instruction pointer. Imps are hard to kill but next to useless for offense. Their use lies in the fact that they can easily be spawned in large numbers, and may survive even if the rest of the warrior is killed.

An **imp ring** (or **imp spiral**) consists of imps spaced at equal intervals around the core and executing alternately. The imps at each arm of the ring/spiral copy their instruction to the next arm, where it is immediately executed again. Rings and spirals are even harder to kill than simple imps, and they even have a (small) chance of killing warriors not protected against them. The number of arms in an imp ring or spiral must be relatively prime with the size of the core.

Quickscanner (or q-scan)

A quickscanner attempts to catch its opponent early by using a very fast unrolled scanning loop. Quickscanning is an early-game strategy, and always requires some other strategy as a backup. Adding a quickscanning component to a warrior can improve its score against long warriors such as other quickscanners. However, the unrolled scan can only target a limited number of locations, and is unlikely to catch a small opponent.

Core clear

A core clear sequentially overwrites every instruction in the core, sometimes even including itself. Core clears are not very common as stand-alone warriors, but are often used as an end-game strategy by bombers and scanners.

Core War Programming

With an understanding of *Core War* strategies, a programmer can create a warrior to achieve certain goals. Revolutionary ideas come once in a while; most of the time, however, programmers base their programs on already published warriors. Using optimizers such as OptiMax or core-step optimizer tools, a more effective warrior can be created.

Warriors can also be generated by genetic algorithms or genetic programming. Programs that integrate this evolutionary technique are known as *evolvers*. Several evolvers were introduced by the *Core War* community and tend to focus on generating warriors for smaller core settings. The latest evolver with significant success was *µGP*[8] which produced some of the most successful nano and tiny warriors. Nevertheless, evolutionary strategy still needs to prove its effectiveness on larger core settings.[9]

Development

Core War was inspired by a self-replicating program called Creeper and a subsequent program called Reaper that destroyed copies of Creeper.[] Creeper was created by Bob Thomas at BBN.[10] Dewdney was not aware of the origin of Creeper and Reaper and refers to them as a rumor originating from Darwin and the worm experiments of Shoch and Hupp. The 1984 Scientific American article on *Core War*[] nevertheless cites the game Darwin, played by Victor A. Vyssotsky, Robert Morris, and Douglas McIlroy at Bell Labs in 1961. The word "Core" in the name comes from magnetic-core memory, an obsolete random-access memory technology.

The first description of the Redcode language was published in March 1984, in *Core War Guidelines* by D. G. Jones and A. K. Dewdney.[] The game was introduced to the public in May 1984, in an article written by Dewdney in *Scientific American*. Dewdney revisited *Core War* in his "Computer Recreations" column in March 1985,[11] and again in January 1987.[12]

The International Core Wars Society (ICWS) was founded in 1985, one year after Dewdney's original article. The ICWS published new standards for the Redcode language in 1986 and 1988, and proposed an update in 1994 that was never formally set as the new standard.[13] Nonetheless, the 1994 draft was commonly adopted and extended, and forms the basis of the *de facto* standard for Redcode today. The ICWS was directed by Mark Clarkson (1985–1987), William R. Buckley (1987–1992), and Jon Newman (1992–); currently the ICWS is defunct.[14]

Redcode

```
0000:  ADD.AB  #   4, $   3
0001:  MOV.F   $   2, @   2
0002:  JMP.B   $  -2, $   0
0003:  DAT.F   #   0, #   0
```

Assembled ICWS-94 style Redcode

Redcode is the programming language used in Core War. It is executed by a virtual machine known as a *Memory Array Redcode Simulator*, or *MARS*. The design of Redcode is loosely based on actual CISC assembly languages of the early 1980s, but contains several features not usually found in actual computer systems.

Both Redcode and the MARS environment are designed to provide a simple and abstract platform without the complexity of actual computers and processors. Although Redcode is meant to resemble an ordinary CISC assembly language, it differs in many ways from "real" assembly.

References

External links

- Core War - the war of the programmers (http://corewar.co.uk/)
- The Core War Info page (http://www.corewar.info/)
- The Beginner's Guide to Redcode (http://vyznev.net/corewar/guide.html)
- Annotated Draft of the Proposed 1994 Core War Standard (http://corewar.co.uk/icws94.htm)
- The Core War Bibliography (http://corewar.co.uk/biblio.htm)

Article Sources and Contributors

Core War *Source*: http://en.wikipedia.org/w/index.php?oldid=562480081 *Contributors*: .digamma, Alexei Kopylov, Alexios Chouchoulas, Angusmclellan, Atlant, Bart133, Brainfsck, Cwolfsheep, Cyclopia, Cyp, Darklilac, David72486, Davidhorman, Dbabbitt, Dennis714, Dijxtra, Divide, Ehird, Enchanter, Epbr123, Ffrreeaakk, Fplay, Frap, Gortu, Ilmari Karonen, Impomatic, Iridescent, Japanese Searobin, Jclark256, Jfmantis, John Metcalf, Joule36e5, Judgesurreal777, Kocio, Lexor, Lotje, Magister Mathematicae, Master Thief Garrett, Maxim Razin, Mclarkson, Michael Hardy, Mika1h, Mintguy, Mjchang, Morte, Nattfodd, Oniscoid, OoS, Pak21, Paul Stansifer, Pi is 3.14159, RJFJR, Rabarberski, Rat at WikiFur, Reyk, Rw63phi, Sayembara, Sfghoul, Size J Battery, Smjg, Spud Gun, Squillero, Svick, Taejo, TenOfAllTrades, Thumperward, Tuankiet65, Tyomitch, Wilke, William R. Buckley, WilliamKF, Wtshymanski, Yonkie, ZeroOne, Zsinj, 159 anonymous edits

Image Sources, Licenses and Contributors

Image:Core War PMars Screenshot.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Core_War_PMars_Screenshot.png *License*: unknown *Contributors*: Original uploader was Ilmari Karonen at en.wikipedia

License