Core War Frequently Asked Questions (rec.games.corewar FAQ)

These are the Frequently Asked Questions (and answers) from the Usenet newsgroup rec.games.corewar. A plain text version of this document is posted every two weeks. The latest hypertext version is available at http://homepages.paradise.net.nz/~anton/cw/corewar-faq.html and the latest plain text version is available at http://homepages.paradise.net.nz/~anton/cw/corewar-faq.txt.

This document is currently being maintained by Anton Marsden (anton@paradise.net.nz).

Last modified: Tue Feb 29 19:42:21 NZDT 2000

To Do

- Change "silk" definition - it is possible to write a silk in ICWS '88. Get some example code
- Add the new No-PSpace '94 hill location
- Make question 17 easier to understand. Add a state diagram?
- Add info about infinite hills, related games (C-Robots, Tierra?, ...)
- New question: How do I know if my warrior is any good? Refer to beginners' benchmarks, etc.
- Add a Who's Who list?
- Would very much like someone to compile a collection of the "revolutionary" warriors so that beginners can see how the game has developed over the years. Mail me if interested.

What's New

- Updated silk definition.
- Minor URL changes.
- Added the online location of Dewdney's articles
- Changed ftp://www.koth.org/corewar/ to ftp://www.koth.org/pub/corewar/
- Changed primary location of FAQ (again!)
- Changed Philip Kendall's home page address.
- Updated list server information

Table of Contents

1. What is Core War?

Core War is a game played by two or more programs (and vicariously by their authors) written in an assembly language called Redcode and run in a virtual computer called MARS (for Memory Array Redcode Simulator). The object of the game is to cause all processes of the opposing program to terminate, leaving your program in sole posession of the machine.

There are Core War systems available for most computer platforms. Redcode has been standardised by the ICWS, and is therefore transportable between all standard Core War systems.

The system in which the programs run is quite simple. The *core* (the memory of the simulated computer) is a continuous array of instructions, empty except for the competing programs. The core wraps around, so that after the last instruction comes the first one again.

There are no absolute addresses in Core War. That is, the address `0` doesn't mean the first instruction in the memory, but the instruction that contains the address `0`. The next instruction is 1, and the previous one obviously `-1`. However, all numbers are treated as positive, and are in the range `0` to `CORESIZE-1` where `CORESIZE` is the amount of memory locations in the core - this means that `-1` would be treated as `CORESIZE-1` in any arithmetic operations, eg. `3218 + 7856 = (3218 + 7856)` **`mod`** `CORESIZE`. Many people get confused by this, and it is particularly important when using the `SLT` instruction. Note that the source code of a program can still contain negative numbers, but if you start using instructions like `DIV #-2, #5` it is important to know what effect they will have when executed.

The basic unit of memory in Core War is one instruction. Each Redcode instruction contains three parts:

- the opcode
- the source address (a.k.a. the A-field)
- the destination address (a.k.a. the B-field)

The execution of the programs is equally simple. The MARS executes one instruction at a time, and then proceeds to the next one in the memory, unless the instruction explicitly tells it to jump to another address. If there is more than one program running, (as is usual) the programs execute alternately, one instruction at a time. The execution of each instruction takes the same time, one cycle, whether it is `MOV`, `DIV` or even `DAT` (which kills the process).

Each program may have several processes running. These processes are stored in a task queue. When it is the program's turn to execute an instruction it dequeues a process and executes the corresponding instruction. Processes that are not killed during the execution of the instruction are put back into the task queue. Processes created by a `SPL` instruction are added to the task queue *after* the creating process is put back into the task queue.

[ToC]

---

2. Is it "Core War" or "Core Wars"?

Both terms are used. Early references were to Core War. Later references seem to use Core Wars. I prefer "Core War" to refer to the game in general, "core wars" to refer to more than one specific battle.

[ToC]

---

3. Where can I find more information about Core War?

Core War was first described in the *Core War Guidelines* of March, 1984 by D. G. Jones and A. K. Dewdney of the Department of Computer Science at The University of Western Ontario (Canada). Dewdney wrote several "Computer Recreations" articles in *Scientific American* which discussed Core War, starting with the May 1984 article. Those articles are contained in two anthologies:

Author	Title	Published	ISBN	Library of Congress Call Number
Dewdney, A. K.	The Armchair Universe: An Exploration of Computer Worlds	New York: W. H. Freeman © 1988	0-7167-1939-8	QA76.6 .D517 1988
Dewdney, A. K.	The Magic Machine: A Handbook of Computer Sorcery	New York: W. H. Freeman © 1990	0-7167-2125-2 (Hardcover), 0-7167-2144-9 (Paperback)	QA76.6 .D5173 1990

A.K. Dewdney's articles are still the most readable introduction to Core War, even though the Redcode dialect described in there is no longer current. You can view a scanned version of the articles at http://www.koth.org/info/sciam/. For those who are interested, Dewdney has a home page at http://www.csd.uwo.ca/faculty/akd/.

[ToC]

---

4. Core War has changed since Dewdney’s articles. Where do I get a copy of the current instruction set?

A draft of the official standard (ICWS’88) is available as ftp://www.koth.org/pub/corewar/documents/standards/redcode-icws-88.Z. This document is formatted awkwardly and contains ambiguous statements. For a more approachable intro to Redcode, take a look at Mark Durham’s tutorials, ftp://www.koth.org/pub/corewar/documents/tutorial.1.Z and ftp://www.koth.org/pub/corewar/documents/tutorial.2.Z.

Steven Morrell has prepared a more practically oriented Redcode tutorial that discusses different warrior classes with lots of example code. This and various other tutorials can be found at http://www.koth.org/info.html.

Even though ICWS’88 is still the "official" standard, you will find that most people are playing by ICWS’94 draft rules and extensions.

[ToC]

---

5. What is ICWS’94? Which simulators support ICWS’94?

There is an ongoing discussion about future enhancements to the Redcode language. A proposed new standard, dubbed ICWS’94, is currently being evaluated. A major change is the addition of "instruction modifiers" that allow instructions to modify A-field, B-field or both. Also new is a new addressing modes and unrestricted opcode and addressing mode combination ("no illegal instructions"). ICWS’94 is backwards compatible; i.e. ICWS’88 warriors will run correctly on an ICWS’94 system. Take a look at the ICWS’94 draft at ftp://www.koth.org/pub/corewar/documents/icws94.0202.Z for more information. There is a HTML version of this document available at http://www.koth.org/info/icws94.html. You can try out the new standard by submitting warriors to the ’94 hills of the KotH servers. Two corewar systems currently support ICWS’94, pMARS (many platforms) and Redcoder (Mac), both available at ftp://www.koth.org/pub/corewar. Note that Redcoder only supports a subset of ICWS’94.

[ToC]

---

6. What is the ICWS?

About one year after Core War first appeared in *Scientific American*, the "International Core War Society" (ICWS) was established. Since that time, the ICWS has been responsible for the creation and maintenance of Core War standards and the running of Core War tournaments. There have been six annual tournaments and two standards (ICWS’86 and ICWS’88). The ICWS is no longer active.

[ToC]

---

7. What is *Core Warrior*?

Following in the tradition of the *Core War News Letter*, *Push Off*, and *The 94 Warrior*, *Core Warrior* is a newsletter about strategies and current standings in Core War. Started in October 1995, back issues of *Core Warrior* (and the other newsletters) are available at http://para.inria.fr/~doligez/corewar/. There is also a *Core Warrior* index page at http://www.kendalls.demon.co.uk/pak21/corewar/warrior.html which has a summary of the contents of each issue of *Core Warrior*. Many of the earlier issues contain useful information for beginners.

[ToC]

8. Where are the Core War archives?

Many documents such as the guidelines and the ICWS standards along with previous tournament Redcode entries and complete Core War systems are available via anonymous ftp from ftp://ftp.csua.berkeley.edu/pub/corewar. Also, most of past rec.games.corewar postings (including Redcode source listings) are archived there. Jon Blow (blojo@csua.berkeley.edu) is the archive administrator. When uploading to /pub/corewar/incoming, ask Jon to move your upload to the appropriate directory and announce it on the net.

This site is mirrored at:

- ftp://www.koth.org/pub/corewar/
- ftp://ftp.inria.fr/INRIA/Projects/para/doligez/cw/mirror

The plain text version of this FAQ is automatically archived by news.answers (but this version is probably out-of-date).

[ToC]

9. Where can I find a Core War system for . . . ?

Core War systems are available via anonymous FTP from www.koth.org in the corewar/systems directory. Currently, there are UNIX, IBM PC-compatible, Macintosh, and Amiga Core War systems available there. It is a good idea to check ftp://www.koth.org/pub/corewar/incoming for program updates first.

CAUTION! There are many, many Core War systems available which are **NOT** ICWS’88 (or even ICWS’86) compatible available at various archive sites other than www.koth.org. Generally, the older the program - the less likely it will be ICWS compatible.

If you are looking for an ICWS’94 simulator, get pMARS, which is available for many platforms and can be downloaded from:

- ftp://ftp.csua.berkeley.edu/pub/corewar (original site)
- ftp://www.koth.org/corewar (koth.org mirror)
- ftp://ftp.inria.fr/INRIA/Projects/para/doligez/cw/mirror (Planar mirror)
- http://www.nc5.infi.net/~wtnewton/corewar/ (Terry Newton)
- ftp://members.aol.com/ofechner/corewar (Fechter)

Notes:

- If you have trouble running pMARS with a graphical display under Win95 then check out http://www.koth.org/pmars.html which should have a pointer to the latest compilation of pMARS for this environment.
- RPMs for the Alpha, PowerPC, Sparc and i386 can be found at ftp://ftp.inria.fr/INRIA/Projects/para/doligez/cw/pmars-rpm/

Reviews of Core War systems would be greatly appreciated in the newsgroup and in the newsletter.

Below is a not necessarily complete or up-to-date list of what's available at www.koth.org:

MADgic41.lzh	corewar for the Amiga, v4.1
MAD4041.lzh	older version?
MAD50B.lha	corewar for the Amiga, beta version 5.0
Redcoder-21.hqx	corewar for the Mac, supports ICWS’88 and ’94 (without extensions)
core-11.hqx	corewar for the Mac
core-wars-simulator.hqx	same as core-11.hqx?
corewar_unix_x11.tar.Z	corewar for UNIX/X-windows, ICWS’86 but not ICWS’88 compatible
koth31.tar.Z	corewar for UNIX/X-windows. This program ran the former KotH server at intel.com
koth.shar.Z	older version
kothpc.zip	port of older version of KotH to the PC
deluxe20c.tar.Z	corewar for UNIX (broken X-windows or curses) and PC
mars.tar.Z	corewar for UNIX, likely not ICWS’88 compatible
icons.zip	corewar icons for MS-Windows
macrored.zip	a redcode macro-preprocessor (PC)
c88v49.zip	PC corewar, textmode display
mars88.zip	PC corewar, graphics mode display
corwp302.zip	PC corewar, textmode display, slowish
mercury2.zip	PC corewar written in assembly, fast!
mtourn11.zip	tournament scheduler for mercury (req. 4DOS)
pmars08s.zip	portable system, ICWS’88 and ’94, runs on UNIX, PC, Mac, Amiga. C source archive
pmars08s.tar.Z	same as above
pmars08.zip	PC executables with graphics display, req 386+
macpmars02.sit.hqx	pMARS executable for Mac (port of version 0.2) buggy, no display
MacpMARS1.99a.cpt.hqx	port of v0.8 for the Mac, with display and debugger
MacpMARS1.0s.cpt.hqx	C source (MPW, ThinkC) for Mac frontend
pvms08.zip	pMARS v0.8 for VMS build files/help (req. pmars08s.zip)
ApMARS03.lha	pMARS executable for Amiga (port of version 0.3.1)
wincor11.zip	MS-Windows system, shareware ($15)

[ToC]

10. Where can I find warrior code?

To learn the game, it is a good idea to study previously posted warrior code. The FTP archives have code in the ftp://www.koth.org/pub/corewar/redcode directory. A clearly organized on-line warrior collection is available at the Core War web sites (see below).

[ToC]

11. I do not have FTP. How do I get all this great stuff?

There is an FTP email server at bitftp@pucc.princeton.edu. Send email with a subject and body text of "help" (without the quotes) for more information on its usage. Note that many FTP email gateways are shutting down due to abuse. To get a current list of FTP email servers, look at the *Accessing the Internet by E-mail FAQ* posted to news.answers. If you don't have access to Usenet, you can retrieve this FAQ one of the following ways:

- Send mail to mail-server@rtfm.mit.edu with the body containing "`send usenet/news.answers/internet-services/access-via-email`".
- Send mail to mailbase@mailbase.ac.uk with the body containing "`send lis-iis e-access-inet.txt`".

[ToC]

12. I do not have access to Usenet. How do I post and receive news?

To receive rec.games.corewar articles by email, join the COREWAR-L list run on the Koth.Org list processor. To join, send the message

SUB COREWAR-L FirstName LastName

to listproc@koth.org. You can send mail to corewar-l@koth.org to post even if you are not a member of the list. Responsible for the listserver is Scott J. Ellentuch (ttsg@ttsg.com).

Servers that allow you to post (but not receive) articles are available. Refer to the *Accessing the Internet by E-Mail FAQ* for more information.

[ToC]

13. Are there any Core War related WWW sites?

You bet. Each of the two KotH sites sport a world-wide web server. Stormking's Core War page is http://www.koth.org; pizza's is http://www.ecst.csuchico.edu/~pizza/koth . Damien Doligez (a.k.a. Planar) has a web page that features convenient access to regular newsletters (*Push Off, The '94*

Warrior, *Core Warrior*) and a well organized library of warriors: http://para.inria.fr/~doligez/corewar/. Convenient for U.S. users, this site is also mirrored at koth.org.

[ToC]

---

14. What is KotH? How do I enter?

King Of The Hill (KotH) is an ongoing Core War tournament available to anyone with email. You enter by submitting via email a Redcode program (warrior) with special comment lines. You will receive a reply indicating how well your program did against the current top programs "on the hill".

There are two styles of KotH tournaments, "classical" and "multi-warrior". The "classical" KotH is a one-on-one tournament, that is your warrior will play 100 battles against each of the 20 other programs currently on the Hill. You receive 3 points for each win and 1 point for each tie. (The existing programs do not replay each other, but their previous battles are recalled.) All scores are updated to reflect your battles and all 21 programs are ranked from high to low. If you are number 21 you are pushed off the Hill, if you are higher than 21 someone else is pushed off.

In "multi-warrior" KotH, all warriors on the hill fight each other at the same time. Score calculation is a bit more complex than for the one-on-one tournament. Briefly, points are awarded based on how many warriors survive until the end of a round. A warrior that survives by itself gets more points than a warrior that survives together with other warriors. Points are calculated from the formula `(W*W-1)/S`, where `W` is the total number of warriors and `S` the number of surviving warriors. The pMARS documentation has more information on multi-warrior scoring.

The idea for an email-based Core War server came from David Lee. The original KotH was developed and run by William Shubert at Intel starting in 1991, and discontinued after almost three years of service. Currently, KotHs based on Bill's UNIX scripts but offering a wider variety of hills are are running at two sites: koth@koth.org is maintained by Scott J. Ellentuch (tuc@ttsg.com) and pizza@ecst.csuchico.edu by Thomas H. Davies (sd@ecst.csuchico.edu). Up until May '95, the two sites provided overlapping services, i.e. the some of the hill types were offered by both "pizza" and "stormking". To conserve resources, the different hill types are now divided up among the sites. The way you submit warriors to both KotHs is pretty much the same. Therefore, the entry rules described below apply to both "pizza" and "stormking" unless otherwise noted.

Entry Rules for King of the Hill Corewar

- Write a corewar program. KotH is fully ICWS '94 compatible, EXCEPT that a comma (",") is required between two arguments.
- Put a line starting with "`;redcode`" (or "`;redcode-94`", etc., see below) at the top of your program. This **MUST** be the first line. Anything before it will be lost. If you wish to receive mail on every new entrant, use "`;redcode verbose`". Otherwise you will only receive mail if a challenger makes it onto the hill. Use "`;redcode quiet`" if you wish to receive mail only when you get shoved off the hill.
  Additionally, adding "`;name <program name>`" and "`;author <your name>`" will be helpful in the performance reports. Do **NOT** have a line beginning with "`;address`" in your code; this will confuse the mail daemon and you won't get mail back. Using "`;name`" is mandatory on the Pizza hills.
  In addition, it would be nice if you have lines beginning with "`;strategy`" that describe the algorithm you use.

There are currently seven separate hills you can select by starting your program with `;redcode-94`, `;redcode-b`, `;redcode-lp`, `;redcode-x`, `;redcode`, `;redcode-94x` or `;redcode-94m`. The former four run at "pizza", the latter three at "stormking". More information on these hills is listed below.

- Mail this file to koth@koth.org or pizza@ecst.csuchico.edu. "Pizza" requires a subject of "`koth`" (use the `-s` flag on most mailers).
- Within a few minutes you should get mail back telling you whether your program assembled correctly or not. If it did assemble correctly, sit back and wait; if not, make the change required and re-submit.
- In an hour or so you should get more mail telling you how your program performed against the current top 20 (or 10) programs. If no news arrives during that time, don't worry; entries are put in a queue and run through the tournament one at a time. A backlog may develop. Be patient.

If your program makes it onto the hill, you will get mail every time a new program makes it onto the hill. If this is too much mail, you can use "`;redcode[-??] quiet`" when you first mail in your program; then you will only get mail when you make it on the top 25 list or when you are knocked off. Using "`;redcode[-??] verbose`" will give you even more mail; here you get mail every time a new challenger arrives, even if they don't make it onto the top 25 list.

Often programmers want to try out slight variations in their programs. If you already have a program named "`foo V1.0`" on the hill, adding the line "`;kill foo`" to a new program will automatically bump `foo 1.0` off the hill. Just "`;kill`" will remove all of your programs when you submit the new one. The server kills programs by assigning an impossibly low score; it may therefore take another successful challenge before a killed program is actually removed from the hill.

Sample Entry

```
;redcode
;name Dwarf
;author A. K. Dewdney
;strategy Throw DAT bombs around memory, hitting every 4th memory cell.
;strategy This program was presented in the first Corewar article.
bomb  DAT   #0
dwarf ADD   #4,    bomb
      MOV   bomb, @bomb
      JMP   dwarf
      END   dwarf          ; Programs start at the first line unless
                           ; an "END start" pseudo-op appears to indicate
                           ; the first logical instruction.  Also, nothing
                           ; after the END instruction will be assembled.
```

Hill Name	**Hill Size**	**Core Size**	**Max. Processes**	**Duration Before Tie**	**Max. Entry Length**	**Min. Distance**	**Rounds Fought**	**Instr. Set**
Pizza's ICWS '94 Draft Hill (Accessed with "`;redcode-94`")	25	8000	8000	80000	100	100	200	Extended ICWS '94 Draft
Pizza's Beginner's Hill (Accessed with "`;redcode-b`")	25	8000	8000	80000	100	100	200	Extended ICWS '94 Draft
Pizza's Experimental (Small) Hill (Accessed with "`;redcode-x`")	25	800	800	8000	20	20	200	Extended ICWS '94 Draft
Pizza's Limited Process (LP) Hill (Accessed with "`;redcode-lp`")	25	8000	8	80000	200	200	200	Extended ICWS '94 Draft
Stormking's ICWS '88 Standard Hill (Accessed with "`;redcode`")	20	8000	8000	80000	100	100	250	ICWS '88
Stormking's ICWS '94 No Pspace Hill (Accessed with "`;redcode-94nop`")	20	8000	8000	80000	100	100	250	ICWS '94
Stormking's ICWS '94 Experimental (Big) Hill (Accessed with "`;redcode-94x`")	20	55440	55440	500000	200	200	250	Extended ICWS '94 Draft
Stormking's ICWS '94 Multi-Warrior Hill (Accessed with "`;redcode-94m`")	10	8000	8000	80000	100	100	200	Extended ICWS '94 Draft

Note: Warriors on the beginner's hill are retired at age 100.

If you just want to get a status report without actually challenging the hills, send email with "`;status`" as the message body (and don't forget "`Subject: koth`" for "pizza"). If you send mail to "pizza" with "`Subject: koth help`" you will receive instructions that may be more up to date than those contained in this document.

At "stormking", a message body with "`;help`" will return brief instructions. If you submit code containing a "`;test`" line, your warrior will be assembled but not actually pitted against the warriors on the hill.

At "pizza", you can use "`;redcode[-??] test`" to do a test challenge of the Hill without affecting the status of the Hill. These challenges can be used to see how well your warrior does against the current Hill warriors.

All hills run portable MARS (pMARS) version 0.8, a platform-independent Core War system available at www.koth.org.

The ’94 and ’94x hills allow five experimental opcodes and three experimental addressing modes currently not covered in the ICWS’94 draft document:

- `LDP` - Load P-Space
- `STP` - Store P-Space
- `SEQ` - Skip if EQual (synonym for `CMP`)
- `SNE` - Skip if Not Equal
- `NOP` - (No OPeration)
- `*` - indirect using A-field as pointer
- `{` - predecrement indirect using A-field
- `}` - postincrement indirect using A-field

[ToC]

---

15. Is it `DAT 0, 0` or `DAT #0, #0`? How do I compare to core?

Core is initialized to `DAT 0, 0`. This is an illegal instruction (in source code) under ICWS’88 rules and strictly compliant assemblers (such as KotH or pmars -8) will not let you have a `DAT 0, 0` instruction in your source code - only `DAT #0, #0`. So this begs the question, how to compare something to see if it is empty core. The answer is, most likely the instruction before your first instruction and the instruction after your last instruction are both `DAT 0, 0`. You can use them, or any other likely unmodified instructions, for comparison. Note that under ICWS’94, `DAT 0, 0` is a legal instruction.

[ToC]

---

16. How does `SLT` (Skip if Less Than) work?

`SLT` gives some people trouble because of the way modular arithmetic works. It is important to note that all negative numbers are converted to positive numbers before a battles begins. Example: `-1` becomes `M-1` where `M` is the memory size (core size).

Once you realize that all numbers are treated as positive, it is clear what is meant by "less than". It should also be clear that no number is less than zero.

[ToC]

---

17. What is the difference between in-register and in-memory evaluation?

These terms refer to the way instruction operands are evaluated. The ’88 Redcode standard ICWS’88 is unclear about whether a simulator should "buffer" the result of A-operand evaluation before the B-operand is evaluated. Simulators that do buffer are said to use in-register evaluation, those that don’t, in-memory evaluation. ICWS’94 clears this confusion by mandating in-register evaluation. Instructions that execute differently under these two forms of evaluation are `MOV`, `ADD`, `SUB`, `MUL`, `DIV` and `MOD` where *the effective address of the A-operand is modified by evaluation of the B-operand*. This is best illustrated by an example:

```
L1  mov L2, <L2
L2  dat #0, #1
```

Under in-register evaluation, the `L2` instruction is saved in a buffer before the `L2` memory location is decremented by evaluation of the B-operand of `L1`. The saved `DAT #0,#1` instruction is then written to `L2`, leaving it unchanged.

Under in-memory evaluation, the `L2` instruction is not buffered and thus decremented by evaluation of the B-operand. After execution of `L1`, `L2` changes to `DAT #0,#0`.

[ToC]

18. What is P-space?

P-space is an area of memory which only your program’s processes can access. The contents of each memory location are preserved between rounds in a multi-round match.

P-space is in many ways different from the core. First of all, each P-space location can only store one number, not a whole instruction. Also, the addressing in P-space is absolute, ie. the P-space address `1` is always `1` regardless of where in the core the instruction containing it is. And last but not least, P-space can only be accessed by two special instructions, `LDP` and `STP`.

The syntax of these two instructions is a bit unusual. `STP`, for example, has an ordinary value in the core as its source, which is put into the P-space field pointed to by the destination. So the P-space location isn’t determined by the destination *address*, but by its *value*, ie. the value that would be overwritten if this were a `MOV`. So `STP.AB #Q, #R` would put the *value* Q into the *P-space field* R **mod** `PSPACESIZE`. Similarly,

```
        stp.b  2, 3
        dat    0, 0
        dat    0, 9
        dat    0, 7
```

would put the value 9 into the P-space field `7`. `LDP` works the same way, except that now the source is a P-space field and the destination a core instruction. The P-space location `0` is a special location. It is initialised to a special value before each round. This value is:

- `-1` (or `CORESIZE-1`) at the beginning of the first round
- `0` if the program died in the previous round
- The number of surviving programs if the program did not die in the previous round

This means that for one-on-one matches, loss=0, win=1 and tie=2.

The default size of P-space is 1/16 of the core size. This size is the value of the predefined variable `PSPACESIZE`. The addresses in the P-space wrap around just like in the core, ie. you can store a value from `0` to `CORESIZE-1` in each P-space location. All P-space values (except for location `0`) are `0` initially.

[ToC]

---

19. What does "Missing `;assert ..`" in my message from KotH mean?

This means you have omitted an "`;assert`" line in your submission. "`;assert`" is used to specify which environments your code will work under or was designed for. For example, if your warrior was written for the '94 draft hill then you can put:

```
;assert CORESIZE==8000
```

in your code, meaning that an error will occur if you attempt you compile the code for a different core size. If you don't want to use the features of "`;assert`" and you want to get rid of the annoying warning just put:

```
;assert 1
```

in your code, which means it will compile unconditionally.

[ToC]

---

20. How should I format my code?

The way you format your code is really your own choice in the end. If you are new to the game then use the style you feel most comfortable with. However, using a common format helps others to understand your code quicker. Most players tend to use the following conventions when writing code:

- use lower case for location names and opcode names
- don't add opcode modifiers if you don't need to, eg. `add.ab #1, #2` is the same as `add #1, #2`
- use whitespace after every comma
- use tabs to align the opcodes, the instruction field(s) and any comments
- do not use **$** (direct addressing mode) or **:** (suffix of some labels)

[ToC]

---

21. Are there any other Core War related resources I should know about?

Using genetic algorithms to generate warriors has been attempted by a number of people. There are a number of resources available for people who are interested in doing some experimentation:

- There is a Core War genetic algorithms mailing list at corewar-ga-request@sunsite.auc.dk. The FTP site for this list is at ftp://sunsite.auc.dk/pub/local/corewar/. The administrator is Martin Pedersen (martin.pedersen@person.dk).
- Jason’s Core War project page (http://www.avalon.net/~jboer/projects/corewar/corewar.html) contains a C program which can evolve warriors, along with some previously generated warriors.
- *Core Wars Genetics: The Evolution of Predation* by John Perry can be found at http://www.koth.org/evolving_warriors.html.
- The most recent paper on Core War evolution is by Ryan Coleman and can be found at http://www.geocities.com/CapeCanaveral/Launchpad/6898/paper.html.

[ToC]

---

22. What does (expression or term of your choice) mean?

Here is a selected glossary of terms. If you have a definition and/or term you wish to see here, please send it to me.

(References to an X-like program mean that the term X is derived from the specific program X and has become a generic term).

Binary launch

One of several means to start an imp-spiral running. The fastest launch technique, but requires the most code. See also `JMP/ADD` Launch and Vector Launch.

```
impsize   equ 2667
example   spl 4               ; extend by adding spl 8, spl 16, etc.
          spl 2
          jmp imp+(0*impsize) ; jmp’s execute in order
          jmp imp+(1*impsize)
          spl 2
          jmp imp+(2*impsize)
          jmp imp+(3*impsize)
imp       mov 0, impsize      ; in ’94 use -> mov.i #0, impsize
```

Bootstrapping

Strategy of copying the active portion of the program away from the initial location, leaving a decoy behind and making the relocated program as small as possible.

B-Scanners

Scanners which only recognize non-zero B-fields.

```
example   add #10, scan
scan      jmz example, 10
```

c

Measure of speed, equal to one location per cycle. Speed of light.

CMP-Scanner

A Scanner which uses a CMP instruction to look for opponents.

```
example   add step, scan
scan      cmp 10, 30
          jmp attack
          jmp example
step      dat #20, #20
```

Colour

Property of bombs making them visible to scanners, causing them to attack useless locations, thus slowing them down.

```
example   dat #100
```

Core-Clear

Code that sequentially overwrites core with DAT instructions; usually the last part of a program.

Decoys

Bogus or unused instructions meant to slow down scanners. Typically, DATs with non-zero B-fields.

Decrement Resistant

Property of warriors making them functional (or at least partially functional) when overrun by a DJN-stream.

DJN-Stream (also DJN-Train)

Using a DJN command to rapidly decrement core locations.

```
example   ...
          ...
          djn example, <4000
```

Dwarf

The prototypical small bomber.

Gate-busting (also gate-crashing)

technique to "interweave" a decrement-resistant imp-spiral (e.g. MOV 0, 2668) with a standard one to overrun imp-gates.

Hybrids

warriors that combine two or more of the basic strategies, either in sequence (e.g. stone->paper) or in parallel (e.g. imp/stone).

Imp

Program which only uses the MOV instruction.

```
example   mov 0, 1
```

or

```
example   mov 0, 2
          mov 0, 2
```

Imp-Gate

A location in core which is bombed or decremented continuously so that an Imp can not pass. Also used to describe the program-code which maintains the gate.

```
example   ...
          ...
          spl 0, <example
          dat <example, #0
```

Imp-Ring

A minimal Imp-Spiral.

```
D         EQU (CORESIZE+1)/3
a         mov 0, D   ; copy self to b
b         mov 0, D   ; copy self to c
c         mov 0, D   ; copy self to a+1
```

Imp-Spiral

An Imp-like program with two or more processes supporting each other. A three-point spiral, with six processes running in this sequence:

```
D         EQU (CORESIZE+1)/3
a         mov 0, D   ; copy self to b
b         mov 0, D   ; copy self to c
c         mov 0, D   ; copy self to a+1
a+1       mov 0, D   ; copy self to b+1
b+1       mov 0, D   ; copy self to c+1
c+1       mov 0, D   ; copy self to a+2
```

Incendiary Bomb

A type of Stun bomb which creates a `SPL 0` carpet.

```
example   spl 0, 8
          mov -1, <-1
```

`JMP`/`ADD` Launch

one of several means to start an imp-spiral running. The slowest launch technique, but requires the least code. See also Binary Launch and Vector Launch.

```
IMPSIZE   EQU 2667
example   spl 1               ; extend by adding more spl 1’s
          spl 1
          spl 2
          jmp @0, imp
          add #IMPSIZE, -1    ; bump address by impsize after each jmp
          dat #0, #0          ; excess processes must die!
imp       mov 0, IMPSIZE      ; in ’94 use -> mov.i #0,IMPSIZE
```

Mirror

see reflection.

On-axis/off-axis

On-axis scanners compare two locations M/2 apart, where M is the memory size. Off-axis scanners use some other separation.

Optimal Constants

(also optima-type constants) Bomb or scan increments chosen to cover core most effectively, i.e. leaving gaps of uniform size. Programs to calculate optimal constants and lists of optimal numbers are available at www.koth.org.

Paper

A Paper-like program is one which replicates itself many times. Part of the Scissors (beats) Paper (beats) Stone (beats Scissors) analogy.

P-Warrior

A warrior which uses the results of previous round(s) in order to determine which strategy it will use.

Pit-Trapper

(also Slaver, Vampire). A program which enslaves another. Usually accomplished by bombing with `JMP`s to a `SPL 0` pit with an optional core-clear routine.

Q^2 Scan

A modern version of the Quick Scan where anything found is attacked almost immediately.

Quick Scan

2c scan of a set group of core locations with bombing if anything is found. Both of the following codes snips scan 16 locations and check for a find. If anything is found, it is attacked, otherwise 16 more locations are scanned.

Example:

```
start
s1 for 8    ;'88 scan
       cmp  start+100*s1, start+100*s1+4000 ;check two locations
       mov  #start+100*s1-found, found      ;they differ so set pointer
rof
       jmn  attack,   found    ;if we have something, get it
s2 for 8
       cmp  start+100*(s2+6), start+100*(s2+6)+4000
       mov  #start+100*(s2+6)-found, found
rof
found  jmz  moveme,  #0        ;skip attack if qscan found nothing
attack cmp  @found,  start-1   ;does found points to empty space?
       add  #4000,   found     ;no, so point to correct location
       mov  start-1, @found    ;move a bomb
moveme jmp  0,       0
```

In ICWS'94, the quick scan code is more compact because of the `SNE` opcode:

```
start       ;'94 scan
s1 for 4
       sne  start+400*s1, start+400*s1+100     ;check two locations
       seq  start+400*s1+200, start+400*s1+300 ;check two locations
       mov  #start+400*s1-found, found       ;they differ so set pointer
rof
       jmn  which,   found     ;if we have something, get it
s2 for 4
       sne  start+400*(s2+4), start+400*(s2+4)+100
       seq  start+400*(s2+4)+200, start+400*(s2+4)+300
       mov  #start+400*(s2+4)-found-100, found
rof
found  jmz  moveme,  #0        ;skip attack if qscan found nothing
       add  #100,    -1        ;increment pointer till we get the
which  jmn  -1,      @found    ;right place
       mov  start-1, @found    ;move a bomb
moveme jmp  0,       0
```

Reflection

Copy of a program or program part, positioned to make the active program invisible to a `CMP`-scanner.

Replicator

Generic for Paper. A program which makes many copies of itself, each copy also making copies.

Self-Splitting

Strategy of amplifying the number of processes executing a piece of code.

```
example    spl 0
loop       add #10, example
           mov example, @example
           jmp loop
```

Scanner
    A program which searches through core for an opponent rather than bombing blindly.
Scissors
    A program designed to beat replicators, usually a (B-field scanning) vampire. Part of the Paper-Scissors-Stone analogy.
Self-Repair
    Ability of a program to fix it's own code after attack.
Silk
    A replicator which splits off a process to each new copy before actually copying the code. This allows it to replicate extremely quickly. Example:

```
        spl   1
        mov   -1, 0
        spl   1                ;generate 6 consecutive processes
silk    spl   3620,   #0       ;split to new copy
        mov   >-1, }-1      ;copy self to new location
        mov   bomb,   >2000 ;linear bombing
        mov   bomb,   }2042    ;A-indirect bombing for anti-vamp
        jmp   silk,   {silk    ;reset source pointer, make new copy
bomb    dat   >2667,  >5334  ;anti-imp bomb
```

Slaver
    see Pit-Trapper.
Stealth
    Property of programs, or program parts, which are invisible to scanners, accomplished by using zero B-fields and reflections.
Stone
    A Stone-like program designed to be a small bomber. Part of the Paper-Scissors-Stone analogy.
Stun
    A type of bomb which makes the opponent multiply useless processes, thus slowing it down. Example is referred to as a `SPL-JMP` bomb.

```
example    spl   0
           jmp   -1
```

Two-Pass Core-Clear (also `SPL`/`DAT` Core-Clear)
    core clear that fills core first with `SPL` instructions, then with `DAT`s. This is very effective in killing paper and certain imp-spiral variations.
Vampire
    see Pit-Trapper.
Vector Launch
    one of several means to start an imp-spiral running. As fast as Binary Launch, but requiring much less code. See also `JMP`/`ADD` Launch and Binary Launch. This example is one form of a Vector Launch:

```
        sz      EQU   2667

                spl   1
                spl   1
                jmp   @vt, }0
        vt      dat   #0, imp+0*sz ; start of vector table
                dat   #0, imp+1*sz
                dat   #0, imp+2*sz
                dat   #0, imp+3*sz ; end of vector table
        imp     mov.i #0, sz
```

[ToC]

---

23. Other questions?

Just ask in the rec.games.corewar newsgroup or contact me. If you are shy, check out the Core War archives first to see if your question has been answered before.

[ToC]

Credits

Additions, corrections, etc. to this document are solicited. Thanks in particular to the following people who have contributed major portions of this document:

- Mark Durham (wrote the original version of the FAQ)
- Paul Kline
- Randy Graham
- Stefan Strack (maintained a recent version of the FAQ)
