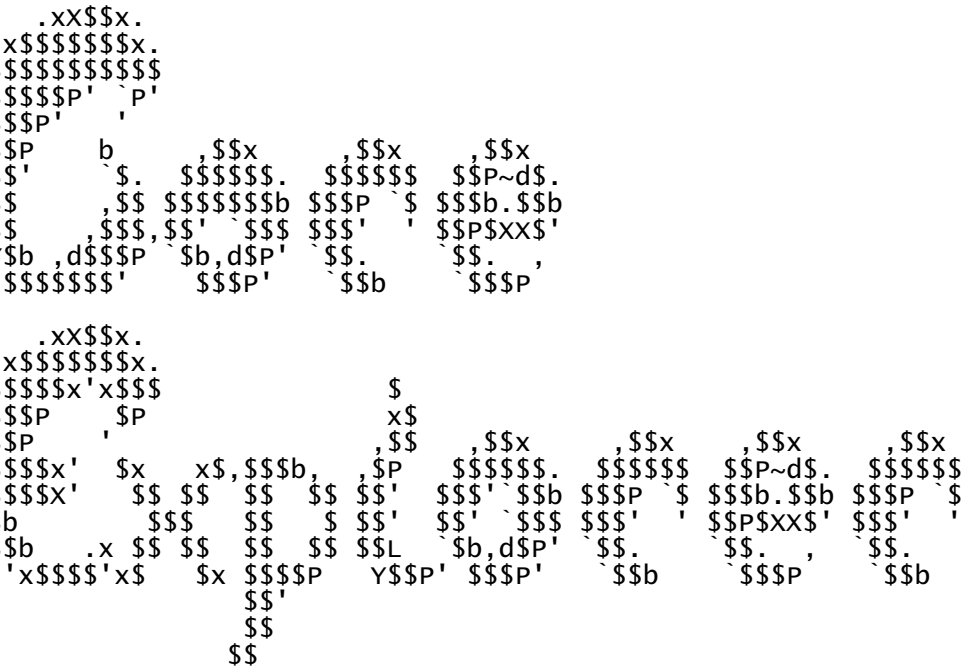

Issue 1
16/10/05

This is the first issue of the The Core Explorer. The brand new newsletter to promote the game CoreWar among new players. We will start from scratch and explain a lot of different strategies, tips and tricks on how to become more skilled in the game.

Roy's Top 10 Resources:

For new players its proven to be very hard to find good resources, here is our
top 10 list of resources and tutorials:

- The beginners guide to Redcode.....http://vyznev.net/corewar/guide.html
- Steve Bailey's Guides.....http://corewar.co.uk/sgb.zip
- Phillip Kendall.....http://www.srcf.ucam.org/~pak21/corewar/resource.html
- The Core Warrior.....http://www.corewar.info/corewarrior/corewarrior.htm
- Steve Morrell.....http://www.koth.org/info/chapter1.html
- Corewar Lexion.....http://www.corewar.info/lexicon
- Fizmo's CoreWar site.....http://www.corewar.info
- John Metcalfs Library.....http://corewar.co.uk/library.htm
- KOTH.org (Info Section).....http://www.koth.org/info.html
- rec.games.corewar (+FAQ).....http://groups.google.com/group/rec.games.corewar

Last but not least, join us on IRC!

Server: irc.koth.org:6667
Channel: #corewars

Did we forget good resources? Mail us!

Exploring replicators

Hi eveybody! This time, we're going to discuss the basics of silk engine for replication and multiprocessing in papers. After we go through some basics,we're going to show you how to make a simple warrior (paper) for beginners hill and we're going to submit it to 94b hill at

http://sal.math.ualberta.ca to test it. So, let's begin. The first thing that you must learn is how to create some desired number of processes at a certain place in the core. It is essential to know this if you want to make warriors that need certain number of processes to function properly, such as most papers. Let's say you want to make two processes. What are you going to do? Simple. Just write spl 1, 0 and the two processes will drop to the next instruction. If you want to have 4 processes, what then?
Well, since 4 = 2^2 = 2*2, you can just repeat the process once again.
So, you'll write down:

```
spl 1, 0
spl 1, 0
;(four processes here)
```

Generally speaking, if you want to create 2^n processes, you just make a chain of n instructions of the form: spl 1, y. Note that you can use the bfield of the spl instruction to store some value that you might need later on, or to do some corecoloring (by using predecrements or postincrements), since that won't affect the creation of processes. Now, what if we want to make x processes running in parallel at a certain spot, and x <> 2^n? Well, there is a way to do that, too. First, you need to write down the desired number of porcesses in binary code. If you don't already know how, here is an example:

```
139 = 69*2 +       1
69 = 34*2  +       1
34 = 17*2  +       0
17 = 8*2   +       1
8  = 4*2   +       0
4  = 2*2   +       0
2  = 1*2   +       0
1  = 0*0   +       1
```

And, thus, 139 = 10001011 =
(10) (2)

1*2^7 + 0*2^6 + 0*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0

I hope that you understand how to do it, now. So, why do we need this? Well, you need to write down (x-1) in binary code and then make a chain of instructions where you interpret each "1" in the result as spl 1, 0 and each "0" as a mov.i -1, 0. So, if you want to have 9 processes on the desired spot, you need to do the following:

9-1 = 8 = 1000
(2)

```
spl 1, 0      \
mov.i -1, 0    \   this chain of spl and mov
mov.i -1, 0    /   instructions creates 9 processes
mov.i -1, 0   /
```

So, now we know how to create as many processes as we might need. Good. Let's continue...
Let's make the most basic paper:

```
00      spl 1, 0
01      spl 1, 0
02      spl pstep1, 0
03      mov >-1, }-1
04      mov <-2, {1
05      jmp pstep2, 0
```

The first spl/mov part utilizes a method called: silk copying. Most of todays papers use it. How does this thing work? Well, we first generate 4 processes. Then the execution of instruction 02 begins. What does it do? It makes some new processes pstep cells away from our current location. But there is nothing there! Only empty core, dat instructions, or who knows what else! So, why do we do this? Because we can. :0 The order of execution is really

important here. After the spl instruction at location 02 is executed, what will be the next instruction to be executed. The one right below! And we use that instruction to copy the code of this little paper to the place where our processes are waiting - pstep1 cells away. How do we do that? Well, I have to remind you that there are 4 processes sitting on that mov >-1, }-1 at location 03. After the first process executes the mov instruction, the code looks like this:

```
00      spl 1, 0
01      spl 1, 0
02      spl pstep1+1, 1
03      mov >-1, }-1
04      mov <-2, {1
05      jmp pstep2, 0
.....
.....
(2+pstep1) spl pstep1, 0
.....
.....
```

After the second execution:

```
00      spl 1, 0
01      spl 1, 0
02      spl pstep1+2, 2
03      mov >-1, }-1
04      mov <-2, {1
05      jmp pstep2, 0
.....
.....
(2+pstep1) spl pstep1, 0
(3+pstep1) mov >-1, }-1
.....
.....
```

After the executions of that mov instruction have finished, we'll get:

```
00      spl 1, 0
01      spl 1, 0
02      spl pstep1+2, 2
03      mov >-1, }-1
04      mov <-2, {1
05      jmp pstep2, 0
.....
.....
(2+pstep1) spl pstep1, 0
;here are the processes that are next to be executed
(3+pstep1) mov >-1, }-1
(4+pstep1) mov <-2, {1
(5+pstep1) jmp pstep2, 0
.....
.....
```

So, we managed to copy the code just in time. The second part of our little paper does the same as the first two instructions, only in reversed - the code is first copied and then we jump to the copy. This is done because we want to close our paper - to prevent the processes from leaking onto unknown part of core (that can be full of spl's or something else we don't like). Why do we do silk copying? Well, it is a little faster AND safer. Why safer? Well, we sometimes have to fight those nasty scanners that are always looking for our copies to attack them and disable them. Well, if we first copy the code, we are giving our opponent the chance to find it and attack it before we jump to it and execute it. That's why silk engine works better for combat purposes. I hope that you understood this article well and that you'll be able to make some simple silk-style papers yourself now. Well, to make a good combat paper, you'll also need some attack, so you'll have to add some bombing lines to the above code and also increase the number of processes that you give to one copy. We'll try to show you how to prepare yourfuture papers for real fighting in the following article.

Just keep reading!

Nenad Tomasev

Core Explorer Warrior #1

Creating a Paper

Now you have a general idea about how a paper works, but you still don't know how to create one. Actually it is quite simple. There are only some building blocks, that you have to glue together and there is your paper.

The first building block consists of the so-called frontend-silk (fsilk):

```
fsilk1  spl     pStep1,         0
        mov.i   > fsilk1,       } fsilk1
```

You can add as many fsilks to the first one and it will still work, if you have the right number of processes (we will come back to later to the question of processes). So the following would work very fine:

```
fsilk1  spl     pStep1,         0
        mov.i   > fsilk1,       } fsilk1

fsilk2  spl     pStep2,         0
        mov.i   > fsilk2,       } fsilk2
        ...
```

There are other building blocks like a backend-silk (bsilk). As the name suggests it should be used at the end of a paper:

```
        ...
        mov.i   > fsilk2,       } bsilk1
bsilk1  jmp     pStep3,         0
```

So far we have:

```
fsilk1  spl     pStep1,         0
        mov.i   > fsilk1,       } fsilk1

fsilk2  spl     pStep2,         0
        mov.i   > fsilk2,       } fsilk2

        mov.i   > fsilk2,       } bsilk1
bsilk1  jmp     pStep3,         0
```

This paper has 6 lines, so it needs 6 processes to work properly:

```
pGo     spl     1
        mov.i   0,              -1
        spl     1

fsilk1  spl     pStep1,         0
        mov.i   > fsilk1,       } fsilk1

fsilk2  spl     pStep2,         0
        mov.i   > fsilk2,       } fsilk2

        mov.i   > fsilk2,       } bsilk1
bsilk1  jmp     pStep3,         0
```

Both fsilks copy first split to a new location (pStep1 and pStep2 instructions away) and copy then the part of the paper there, that follows the silk. Let me explain that a little bit more exactly.

fsilk1 copies the complete paper, but fsilk2 doesn't. It copies all instructions between fsilk2 and bsilk1 and (!) the two following instructions. Usually the two following instructions are just some dat's, that might overwrite an opponent.

So far we have a quite uninteresting paper. It copies itself around and hopes, that this is enough. Maybe we should throw some bombs:

```
fsilk1  spl     pStep1,         0
        mov.i   > fsilk1,       } fsilk1

fsilk2  spl     pStep2,         0
        mov.i   > fsilk2,       } fsilk2

        mov.i   bomb,           < bStep

        mov.i   > fsilk2,       } bsilk1
bsilk1  jmp     pStep3,         0

bomb    dat.f   < 1,            { 1
```

Since we now have 8 lines, we need 8 processes to make the paper work properly:

```
pGo     spl     1
        spl     1
        spl     1
...
```

But how does the bombing work? At first you must know, that you could include any kind of instruction between the various silks, but usually it is best to insert it before the last silk.

So we still have the two fsilks, which copy the paper around, but before the bsilk is executed, we have to deal with the bombing. We not only throw one bomb, but 8, because we have 8 processes for the paper. To avoid, that we throw 8 bombs all to the same position, we have to change the target pointer (bStep). Because of the predecrement, we throw 8 bombs below the position bStep indicates.

Before we can test this paper, we should find some constants for pStep1, pStep2, pStep3 and bStep:

```
        pStep1  EQU     2341
        pStep2  EQU     2421
        pStep3  EQU     3241
        bStep   EQU     100

pGo     spl     1
        spl     1
        spl     1

fsilk1  spl     pStep1,         0
        mov.i   > fsilk1,       } fsilk1

fsilk2  spl     pStep2,         0
        mov.i   > fsilk2,       } fsilk2

        mov.i   bomb,           < bStep

        mov.i   < fsilk2,       { bsilk1
bsilk1  jmp     pStep3,         0

bomb    dat.f   < 1,            { 1
```

This version scores about 118 against WilFiz. That is not that bad.

Jens Gutzeit

___ SAL 94b update: 13.10.2005.___

So, let's have a closer look at the beginners hill at SAL.
There are many papers currently on the hill, and scanners have
taken advantage of that, and rule the hill. Dragonfly, test, Svarog,
Yetima, Xenocitrum... Maybe you should consider submitting a scanner
and join the feast. However, it is usually a better approach to attack
the warriors from the top part of the hill, because that guarantees
a longer supply of food for your warrior. Most of the successfull
scanners there are oneshots, and Dragonfly is a real hit. Doesn't it
bother you that it rules the hill in such a superior way? Don't you want
to bring it down? Perhaps you should try some aggressive stone/imps?
Maybe not? Let's have a closer look at the scores... Dragonfly loses heavily
to some warrios. Why? Because they have a good decoy placement. So, this
is the advice: make your warrior more resistant against oneshots, and
you'll rule the hill! Maybe it would be best to attack with a good scanner
with some extra oneshot protection: large decoys, boot, corecoloring, etc.
However, stone/imps and stone/papers could also do well, if they are
designed carefully. We'll show you how to make all these warriors soon!
Our paper from this issue didn't enter the hill, because it had no anti-
imp weaponry, and also no qscan (you'll learn about that later). Try to
analize it and improve it. It will be a good exercise. In the meantime,
try conquering the 94b hill! And remember: the most important thing is
strategy: who do we want to defeat with our warrior? Where do we intend
to get the points to enter... etc.
More in the next issue of CoreExplorer. Stay tuned.

Status 94b hill @ SAL:

#	%W	%L	%T	Name	Author	Score	Age
1	42.8	38.9	18.3	Dragonfly	S.Fernandes	146.7	1
2	40.3	42.0	17.7	test	Andreas Scholta	138.7	3
3	39.9	41.6	18.6	Svarog	Nenad Tomasev	138.1	104
4	30.3	24.1	45.7	Yatima v2.0.5	Jens Gutzeit	136.4	72
5	39.0	42.8	18.2	Xenocitrum_v0.8	inversed	135.3	16
6	27.0	19.2	53.9	D'n'B [v0.3]	inversed	134.7	8
7	38.4	42.8	18.8	Unknown	Neo	133.9	190
8	38.2	43.1	18.6	think twice v.2 (..	el kauka	133.4	45
9	40.3	47.4	12.4	Star	Sascha Zapf	133.2	235
10	23.5	14.3	62.2	Snare_Rush_v0.7	Inversed	132.8	9
11	28.8	25.3	46.0	The Silent Death	Anonymous	132.2	86
12	25.4	20.7	53.8	stealthbomb	Fizmo and Neutrino	130.1	103
13	26.8	23.5	49.7	Blotter	inversed	130.1	5
14	37.2	44.6	18.3	Blindfolded	Miz	129.8	14
15	26.9	24.5	48.7	Tom	David Moore	129.3	218
16	28.0	26.8	45.2	3[sm]md v0.2	inversed	129.1	2
17	36.4	45.6	18.1	dx42e	inversed	127.2	12
18	24.1	21.2	54.7	Barkosta[v0.3]	inversed	126.9	18
19	22.7	20.0	57.3	Enigma 2.1	brx/Roy	125.3	200
20	33.7	42.2	24.1	Spark_v0.4	inversed	125.2	21
21	20.7	16.4	62.9	Everybody must ge..	madjester	125.0	140
22	22.0	19.3	58.7	Biomass_v0.8	inversed	124.7	19
23	23.0	21.7	55.3	Envane [v0.5]	inversed	124.2	7
24	19.8	16.6	63.5	Break Down and Cry	Jens Gutzeit	123.0	73
25	31.7	42.3	26.1	bloodhunter	elkauka	121.0	4

Questions? Articles? Suggestions? Compliments? Mail us!
Authors: Nenad <tomasev at nspoint.net>, Roy <roy dot van dot rijn at gmail>