Note of the Author

This manual is intended to be a tutorial for a programmer who wants to start playing the Core War. You can find my address at the end of this manual. You can write to me if you need more information. Keep in mind that I'm not an English speaking person (I'm Italian) and my English is far from perfection, so you are authorized to correct the mistakes that fill this paper.

Introduction

If you are interested in the Core War you are supposed to be a programer. So the best way to describe it is to show you the algorithm that evaluates the Redcode instructions (Redcode is the assembly-like language the Core Wars programs are written with).

Before showing the algorithm there is the need to introduce some concepts. The following dissertation is based on the ICWS Standard '86. A new Standard is under discussion, but at this moment I have no news about it. However the proposed new Standard resembles very closely the current one : there is only a new instruction and some modifications in the multitasking handling algorithm. The proposed new Standard will be discussed at the end of this manual.

1. THE CORE

The Core is the memory of the MARS virtual computer. The dimensions of the Core are not constrained by any current standard. In the tournaments held by ICWS and its branch sections it is in use to adopt the Core size of 8192 locations, numbered from 0 to 8191. At the beginning of a battle every location of the Core is initialized to the value of 0 (DAT #0 #0, see ahead for the explanation of this construct). Two programs, elsewhere referred as 'warriors', are loaded at random position in the Core. Each warrior doesn't know where its opponent is loaded. It is clear that the only factor which makes a battle different from another one is the distance between the place the two warriors are loaded at. So, to simply the loading algorithm, it is possible to load always the first warrior at the Core location 0, and to load the other warrior at a randomly choosen location. The fact that Redcode hasn't the absolute addressing, implies that there is no way to get advantage by this algorithm.

Obviously the codes of the two warriors mustn't overlap. In the ICWS tournaments that a minimum distance between the warriors must be observed at loading time : usually it consists of 2048 locations. In a Core of 8192 cells observing a distance of 2048 (at the head and at the tail of a warrior) implies that the number of the different battles playable for each couple of warriors is 8192 - 2 * 2048 - length (first warrior) - length (second warrior) < 4096. There are programs which execute all the possible battles between each couple of warriors. These programs are the better way to determine the real strength of a warrior. However they require a very fast computer; the QL is inadequate for this task.

2. THE INSTRUCTIONS

There are 10 different instructions in Redcode. They have a constant format. It consists of 5 fields :

- Opcode Range 0 .. 10
- A-Mode Range 0 .. 3
- A-Field Range 0 .. CORE-SIZE
- B-Mode Range 0 .. 3

- B-Field Range 0 .. CORE-SIZE

Each instruction takes exactly one Core location. Not all the instructions use all the two fields.

2.1 Opcodes

Each opcode corresponds to one instruction. They are the following :

MNEMONIC	OPCODE	FIELDS USED	MEANING
DAT	0	B	(THE)Invalid instruction. It contains data
MOV	1	A B	Copy the A location into the B one
ADD	2	A B	B location := A location + B location
SUB	3	A B	B location := A location - B location
JMP	4	A	Jump to A location
JMZ	5	A B	Jump to A location if B = 0
JMN	6	A B	Jump to A location if B <> 0
DJN	7	A B	Decrement B and jump to A if B <> 0
CMP	8	A B	If A = B skip the next instruction
SPL	10	B	Start a new program at B

The opcode 9 is not used (historical reasons : an old instruction wiped out from the Standard).

2.2 Addressing modes

There are 4 different addressing modes.

MODE	SYMBOL	MEANING
Immediate	#	The following field is considerated as a number Example : CMP #12 #32 It compares 12 with 32 (It is meaningless)
Relative	NONE	The folloving field is cosiderated as an offset from the current location. Example : MOV #12 5 moves the number 12 five locations ahead in the Core. Negative numbers are allowed. In a 8192 locations Core -1 is equivalent to 8191, -2 to 8190 ...
Indirect	@	The field is considerated as in the relative addressing mode, but the location so reached is read to get the effective operand. Example : MOV #12 @5. MARS reads the location CURRENT + 5 and gets its content (say 14). This is used as the displacement to perform a relative addressing operation : the 12 is written to location (CURRENT + 5) + 14.
Predecremented	<	This acts as the indirect addressing mode. The indirect only difference is that the content of the location reached by the first relative addressing id decremented before using it. Example : MOV #12 <5 It is read the location CURRENT + 5 (again, assume it contains 14). Its content is decremented by one and it used to perform another relative addressing to location (CURRENT + 5) + 13.

3. THE ALGORITHM

The following algorithm is given in a psuedo-programming language. It will be easy to translate it into one of the available programing languages. The original (italian) version of this algorithm was published on MC Microcomputer, number 76, July/August 1988 and it was written by Nicola Baldini, Andrea Giotti, Claudio Bizzarri. I'm translating it in English and introducing some minor changes.

It is worth remembering that a WORD is a record whose fields are : WORD.OPCODE, WORD.A.MODE, WORD.A.FIELD, WORD.B.MODE, WORD.B.FIELD.

At the purpose of keeping simple the algorithm I'll adopt this notation : | Z | is equal to Z modulo CORE.SIZE.

```
---- REMark get the PC of one of the processes which constitute the
warrior currently in execution. Details of this operation will given
in the next chapter.
Get PC from the Program Counters List
CURRENT := CORE [ PC ]
---- REMark now evaluate the addressing mode of the first operand
evaluate CURRENT.A.MODE :
0 : Immediate
       LOCATION.1 := PC
1 : Relative
       LOCATION.1 := | PC + CURRENT.A.FIELD |
2 : Indirect
       LOCATION.1 := | PC + CURRENT.A.FIELD +
                       CORE [ | PC + CURRENT.A.FIELD | ].B.FIELD |
3 : Predecremented
       POINTER.1  := | PC + CURRENT.A.FIELD |
       LOCATION.1 := | POINTER.1 - 1 + CORE [ POINTER.1 ].B.FIELD |


---- REMark evaluate the addressing mode of the second opearand
evaluate CURRENT.B.MODE :
0 : Immediate
       LOCATION.2 := PC
1 : Relative
       LOCATION.2 := | PC + CURRENT.B.FIELD |
2 : Indirect
       LOCATION.2 := | PC + CURRENT.B.FIELD +
                       CORE [ | PC + CURRENT.B.FIELD | ].B.FIELD |
3 : Predecremented
       POINTER.2 := | PC + CURRENT.A.FIELD |
       LOCATION.2 := | POINTER.2 - 1 + CORE [ POINTER.2 ].B.FIELD |

---- REMark Fetch the locations interested by the instruction
       Note that the whole words are copied
WORD.1 := CORE [ LOCATION.1 ]
WORD.2 := CORE [ LOCATION.2 ]

---- REMark calculate the values of A.FIELD and B.FIELD
if CURRENT.A.MODE = 0 THEN A.VALUE := WORD.1.A.FIELD
                      ELSE A.VALUE := WORD.1.B.FIELD
B.VALUE := WORD.2.B.FIELD
```

```
---- REMark perform the decrement on A.FIELD and B.FIELD
if CURRENT.A.MODE = 3 THEN
    CORE [ POINTER.1 ].B.FIELD := | CORE [ POINTER.1 ].B.FIELD - 1 |
if CURRENT.B.MODE = 3 THEN
    CORE [ POINTER.2 ].B.FIELD := | CORE [ POINTER.2 ].B.FIELD - 1 |

---- REMark increase PC
PC := | PC + 1 |

---- REMark eventually execute the instruction
evaluate CURRENT.OPCODE :
0 : DAT
       Terminate this process
1 : MOV
       if CURRENT.A.MODE = 0 or
          CURRENT.B.MODE = 0
           then CORE [ LOCATION.2 ].B.FIELD := A.VALUE
           else CORE [ LOCATION.2 ] := WORD.1
2 : ADD
       CORE [ LOCATION.2 ].B.FIELD := | B.VALUE + A.VALUE |
3 : SUB
       CORE [ LOCATION.2 ].B.FIELD := | B.VALUE - A.VALUE |
4 : JMP
       PC := LOCATION.1
5 : JMZ
       if B.VALUE = 0 then PC := LOCATION.1
6 : JMN
       if B.VALUE <> 0 then PC := LOCATION.1
7 : DJN
       CORE [ LOCATION.2 ].B.FIELD :=
                        | CORE [ LOCATION.2 ].B.FIELD - 1 |
       if | B.VALUE - 1 | <> 0 then PC := LOCATION.1
8 : CMP
       if CURRENT.A.MODE = 0 or
          CURRENT.B.MODE = 0
           then        if A.VALUE = B.VALUE then PC := | PC + 1 |
           else        if WORD.2  = WORD.1  then PC := | PC + 1 |
10 : SPL
       if this warrior has less than 64 active processes then
           generate a new process and let its PC := LOCATION.2


store PC in the Program Counters List

END OF INSTRUCTION EVALUATION
```

4. PROCESSES HANDLING

In the algorithm above there are some obscure points that it has to manage with the processes. I'll try to clarify them.

You can see in the pseudo-code for the SPL instruction that a warrior can't run more that 64 processes. If this limit is reached the SPL has no more effect : it becomes equivalent to the SuperBASIC "REMark".

The Program Counters List (PCL) is the place the processes' PC are stored in. Two PCLs are needed : one for each warrior. It more efficient to implement it as and array of 64 elements instead of a list : an array allows random access and there is no need of pointers, so it is less memory consuming and faster to access.

When the battle is started the PCLs contain only the PC of the two warriors.

```
PCL.1 ->   A1
PCL.2 ->   B1
```

The processes are executed in this order : $A_1\,B_1\,A_1\,B_1\,A_1\,B_1$ ...

When (for example) the first warrior executes a SPL the situation changes :

```
PCL.1 ->   A1  A2
PCL.2 ->   B1
```

and the processes are executed in this way : $A_1\,B_1\,A_2\,B_1\,A_1\,B_1\,A_2$ ... with the effect that the two A's processes run at half speed. When other SPLs are performed the modifications follow the same rule. However it is important to notice that in the PCL a new process PC must directly follow the PC of the process which has created it : suppose that A_{12} creates the process A_{30} ; the the PCL will be :

```
PCL.1 ->   A?? ... A12 A30 A?? ...
```

When a process executes a DAT instruction it is terminated and its PC is wiped out from the PCL. When a PCL becomes empty the corresponding warrior is defeated.

5. MORE ABOUT TOURNAMENTS

It is possible that, due to the modifications made during the battle to the code of the warriors or due to other factors, a battle won't terminate. To prevent this case a limit is applied on the number of the instruction executed per warrior : usually this limit amounts to 15000 instructions per warrior. It is possible that a battle will give a winner after, say, 100000 instructions, but this is a lot of time on a computer. If the time limit elapses without a winner the battle ends with a draw. In the tournaments a victory is usually rewarded with 3 points, a draw with 1 point, and a defeat with 0 points (as in the English Football League, if I'm right).

6. THE PROPOSED NEW STANDARD (PNS)

If the PNS be adopted the following modifications will be introduced :

- the new instruction SLT A B will appear with the following syntax : if A is less than B than skip next instruction.
- all the fields of an instruction will be accepted by the compiler, even if meaningless for the particular instruction.
- the SPL B will become SPL A (the B field can also be used but it is meaningless)
- the 64 processes per warriors limit will be dropped.
- a newly created process' PC will be placed at the back of the PCL

7. HOW TO GET THE MOST FROM REDCODE

Instead of using a DAT for storing pointers or counters or ..., you can store that in the unused field of the JMP instruction. Example :

```
START    MOV #123, COUNTER
         ...
COUNTER  JMP SOMEWHERE
```

If you look at the MARS algorithm you can see that the MOV #123, COUNTER will only modify the B field of the JMP at COUNTER. A following MOV BOMB, @COUNTER will only read the B field and all works well as if you were reading from a DAT. In this way you can save one instruction, because it is possible that the location at START won't be used anymore by your program after the startup : it is obvious that a small warrior is less vulnerable than a larger one (but it can be less smart, too).

Normally a bomb is a DAT #0, but what happens if the bomb is a SPL 0 ? What if the bomb is a JMP @0 ? Hint. Remember that the B field can be easily accessed by a warrior : it can contain and address to jump at ! The current world champion COWBOY uses this tecnique to capture the enemy processes, when the prisoners - bisons in the snare... - reach the number of 64 the cowboy can start to shot at them and win the battle.

8. CONCLUSION

I hope this paper will help you to start playing the Core War. In the floppy you have found this manual in, you can also find the MARS interpreter and compiler, plus some warrior. Try to understand how they work and what they do. PLAY THE CORE WAR !

Paolo Montrasio
via XXIV Maggio 49
20099 Sesto San Giovanni
MILANO
ITALY

telephone (Italian speakers or emergencies only) : 02 2487734
(20.00 PM)