# Corewar

Understanding the Operation
By jyeo

# live - Live (opcode 0x01)

Duration to execute : **10cycles**

I'M ALIVE!!!! takes 1 parameter

S (D4) Modify carry : **No** OCP : **No** % IDX_MOD : **No** (Address Restriction)

Report the player as alive

1st Param is the player id

**NEW UPDATE : the value of the 1st param is an integer**

Example : (D4)

```
live %-1
```

→ live takes the 1st param which is -1 and report it as alive.
-1 is the id of the first player thus it will report player 1 as alive

```
Player -1 : tester
  Last live :                    10
  Lives in current period :       1
```

→ The vm report player 1 as alive once.

```
live %-2
```

→ live takes the 1st param which is -2 and report it as alive.
-2 is the id of the second player thus it will report player 2 as alive

```
Player -1 : tester
  Last live :                     0
  Lives in current period :       0

Player -2 : zork
  Last live :                    41
  Lives in current period :       2
```

→ The vm report player 2 as alive twice and non for player 1
Because "tester" live %-2 report player 2 as alive
and "zork" do live on itself so it will report itself as alive

If still cannot understand it, to be simple : this operation is crucial to survive!

# ld - Load Direct (opcode 0x02)

Duration to execute : **5cycles**

Save a value into register, takes 2 parameter

S (ID | D4), D (RG) Modify carry : **Yes** OCP : **Yes** % IDX_MOD : **Yes** (Address Restriction)

if S = ID, Save the memory from RAM into REGISTER specified in 2nd param.

if S = D4, Save the value as it is (as a number if you don't understand what i mean) into REGISTER specified in 2nd param.

**NEW UPDATE : the value of the 1st param is an integer**

Example : (ID), (RG)

```
ld 3, r3
st r3, 59
```

ld takes the memory from the address of ID 3 and save it into Register 3

*(st is to store the value into RAM, or we can say that it is copy and paste the result from r3 into RAM for us to see the result, we will look at this operation on the next page, so take it as it is for now.)*

| 0 | 1 | 2 | **3** | 4 | 5 | 6 | 7 | 8 | 9 |

(PC)

```
02 d0 00 03 03 03 70 03 00 3b
03 03 03 70 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
```

Read REG_SIZE bytes from the address from the1st param which is PC + (3 % IDX_MOD) = **3** and save it into r3

This shows that r3 now contain that value.

Example : (D4), (RG)

```
ld %100, r3
st r3, 57
```

ld takes the value from D2 and save it into Register 3

```
02 90 00 00 00 64 03 03 70 03 00 39
00 00 00 64 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
```

This shows that r3 now contain the value of 100 in hexadecimal which is 0x64

If still cannot understand it, to be simple : this operation is the same as SAVE a file into a FOLDER

# st - Store Direct (opcode 0x03)

Duration to execute : **25cycles**

Get a value from Source and store it into Destination takes 2 parameter

S (RG), D (RG | ID) Modify carry : **No** OCP : **Yes** % IDX_MOD : **Yes** (Address Restriction)

if D = RG, Store the value from Source into a Register specified in 2nd param.

if S = ID, Store the value from Source into an address in RAM specified in 2nd param.

This operation feels like we are doing COPY and PASTE

**NEW UPDATE : if the 2nd param is ID, the value is a short**

Example : (RG), (RG) &&
Example : (RG), (ID)

```
ld %100, r3
st r3, r4
st r4, 53
```

Base on the previous page, we understand that ld save the value into register, so now we will have the value of 0x64 in r3.
We will now copy and paste this value from r3 into r4.
And we will copy and paste this value from r4 into RAM pointed by ID 53
PC + (ID % IDX_MOD)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

(PC)

```
02 90 00 00 00 64 03 03 50 03 04 03 70 04 00 35
00 00 00 64 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

**Address 0x64**

COPY and PASTE the value of 0x64 into 11 + (53 % IDX_MOD) = **64**

This shows that the value from r3 is copied to r4 and the value of r4 is now copied into RAM at address of **0x64**

If still cannot understand it, to be simple : this operation is the same as COPY and PASTE, nothing more, nothing less

# add - Addition (opcode 0x04)

Duration to execute : **10cycles**

Add the 1st and 2nd param and store it in 3rd param

S (RG), S (RG), D (RG)  Modify carry : **Yes**  OCP : **Yes**  % IDX_MOD : **No** (Address Restriction)

This operation simply add (+) the 1st with 2nd param and save that value into 3rd param.

Example : (RG), (RG), (RG)

```
ld %100, r1
ld %100, r2
add r1, r2, r3
st r3, 45
```

Save number 100 into r1
Save number 100 into r2
Add r1 + r2 and store it in r3
Show it in the RAM

```
02 90 00 00 00 64 01 02 90 00 00 00 64 02 04 54 01 02 03 03 70 03 00 2d
00 00 00 c8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

r1 (100) + r2 (100) = 200 0xc8
If inside the register contains memory which is hexadecimal, they work the same as well

If still cannot understand it, to be simple : this operation is already very simple

# sub - Subtraction (opcode 0x05)

Duration to execute : **10cycles**

Sub the 1st and 2nd param and store it in 3rd param

S (RG), S (RG), D (RG) Modify carry : **Yes** OCP : **Yes** % IDX_MOD : **No** (Address Restriction)

This operation simply subtract (-) the 1st with 2nd param and save that value into 3rd param.

No example, it works the same as the previous operation

# and - Logical And (opcode 0x06)

Duration to execute : **6cycles**

Bitwise operation and (&) between the 1st and 2nd param and store it in 3rd param

S (RG | ID | D4), S (RG | ID | D4), D (RG)  Modify carry : **Yes**  OCP : **Yes**  % IDX_MOD : **Yes** (Address Restriction)

This operation simply do the (&) operation between the 1st with 2nd param and save that value into 3rd param.

Address Restriction only applied on ID

D4 is the value as it is (as a number)

**NEW UPDATE : the value of 1st param & 2nd param is an integer**

No example, it works the same as the previous operation

# or - Logical OR (opcode 0x07)

Duration to execute : **6cycles**

Bitwise operation or (|) between the 1st and 2nd param and store it in 3rd param

S (RG | ID | D4), S (RG | ID | D4), D (RG)  Modify carry : **Yes**  OCP : **Yes**  % IDX_MOD : **Yes** (Address Restriction)

This operation simply do the (|) operation between the 1st with 2nd param and save that value into 3rd param.

Address Restriction only applied on ID

D4 is the value as it is (as a number)

**NEW UPDATE : the value of 1st param & 2nd param is an integer**

No example, it works the same as the previous operation

# xor - Logical XOR (opcode 0x08)

Duration to execute : **6cycles**

Bitwise operation or (^) between the 1st and 2nd param and store it in 3rd param

S (RG | ID | D4), S (RG | ID | D4), D (RG)  Modify carry : **Yes**  OCP : **Yes**  % IDX_MOD : **Yes** (Address Restriction)

This operation simply do the (^) operation between the 1st with 2nd param and save that value into 3rd param.

Address Restriction only applied on ID

D4 is the value as it is (as a number)

**NEW UPDATE : the value of 1st param & 2nd param is an integer**

No example, it works the same as the previous operation

# zjmp - Jump (opcode 0x09)

Duration to execute : **20cycles**

Jump to the address specified in 1st param

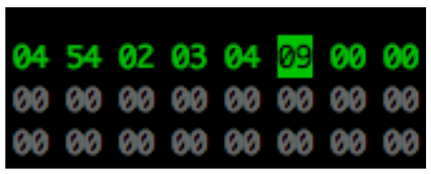
S (D2) Modify carry : **No** OCP : **No** % IDX_MOD : **Yes** (Address Restriction)

This operation will make the PROCESS jump or teleport into another address in the RAM,
With proper planning, this could make the PROCESS into a looping state
zjmp will only do the jump if carry is in the state 1, if else it will not jump.
**NEW UPDATE : the value of 1st param is a short**

Example : (D2)

```
add r2, r3, r4
zjmp %0
```

→ The r2, r3, by default is 0, so by adding this 2 param, we will get the result of 0
This will modify the carry from 0 to 1.
zjmp take the address from D2, for this example is PC + (0 % IDX_MOD) = **0**




→ zjmp is jumping to address 0 from the current location, which is the same place.
This will make the process into an infinite looping that stays in the same location

If still cannot understand it, to be simple :
this operation can teleport your process to anywhere within the distance permitted by IDX_MOD if carry == 1

# ldi - Load Index (opcode 0x0a)

Duration to execute : **25cycles**

Save a value into register, takes 3 parameter

S (RG | ID | D2), S (RG | D2), D (RG)  Modify carry : **Yes**  OCP : **Yes**  % IDX_MOD : **Yes** (Address Restriction)

Add value from the 1st and 2nd param and read the address located from the result to save into the REGISTER specified at the 3rd param.

**NEW UPDATE : if the 1st or 2nd param is D2, it is a short, if it is ID, it is an integer**

Example : (D2), (D2), (RG)

```
ldi %4, %4, r3
st r3, 57
```

ldi takes the number of 4 (1st param) + 4 (2nd param) = 8

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | **8** | 9 | 10 | 11 |

(PC)

```
0a a4 00 04 00 04 03 03 70 03 00 39
70 03 00 39 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
```

Read REG_SIZE bytes from the address in MEMORY from the result of the addition which is

PC + (8 % IDX_MOD) = **8**

This shows that r3 now contain that value.

---

Example : (ID), (D2), (RG)

```
ldi 3, %4, r3
st r3, 57
```

ldi takes the ID value of 3, get that value located from the address in MEMORY as 1st param

which is **0x3000403 = 50332675**

Add this value with 2nd param (4) - > **50332675 + 4 = 50332679**

| 0 | 1 | 2 | **3** | 4 | 5 | 6 | **7** | 8 | 9 | 10 | 11 |

(PC)

```
0a e4 00 03 00 04 03 03 70 03 00 39
03 70 03 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
```

Read REG_SIZE bytes from the address in MEMORY from the result of the addition which is

PC + (**50332679** % IDX_MOD) = **7**

This shows that r3 now contain that value.

Example : (RG), (D2), (RG) &&

Example : (RG), (RG), (RG) &&

Example : (ID), (RG), (RG)

No example, it will perform addition according to what is the value inside the RG

If still cannot understand it, to be simple :

this operation works the same as ld but the location where it gets the value is more complicated :(

# sti - Store Index (opcode 0x0b)

Duration to execute : **25cycles**

Store a value into register, takes 3 parameter

S (RG), S (RG | ID | D2), D (RG | D2), Modify carry : **No** OCP : **Yes** % IDX_MOD : **Yes** (Address Restriction)

Get the value from the REGISTER specified at 1st param and store it into the address from the result of the add between 1st and 2nd param.

**NEW UPDATE : if the 2nd or 3rd param is D2, it is a short, if it is ID, it is an integer**

Example : (RG), (D2), (D2)

```
ld %100, r3
sti r3, %4, %53
```

Store the value of 100 into r3
COPY the value of r3 and PASTE it into the address in MEMORY pointed by result of **4 + 53**
**7** + (**57** % IDX_MOD) = **64**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | **7** | 8 | 9 | 10 | 11 | 12 | 13|

(PC)

```
02 90 00 00 00 64 03 0b 68 03 00 04 00 35
00 00 00 64 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

The value is PASTED here in memory 0x64

---

Example : (RG), (ID), (D2)

```
ld %100, r3
sti r3, 5, %57
```

Store the value of 100 into r3
COPY the value of r3 and PASTE it into the address in MEMORY pointed by result of **ID 5 + 53**
**ID 5** is **0x390000 = 3735552 + 57 = 3735609**
**7** + (**3735605** % IDX_MOD) = **64**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | **7** | 8 | 9 | 10 | 11 | **12** | 13| 14| 15|

(PC)

```
02 90 00 00 00 64 03 0b 78 03 00 05 00 39 00 00
00 00 00 64 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

The value is PASTED here in memory 0x64

Example : (RG), (RG), (D2) &&

Example : (RG), (RG), (RG) &&

Example : (RG), (ID), (RG)

No example, it will perform addition according to what is the value inside the RG

If still cannot understand it, to be simple :

this operation works the same as st but the location where it PASTE the value is more complicated :(

# fork - Fork (opcode 0x0c)

Duration to execute : **800cycles**

Generates a new process takes 1 parameter

S (D2) Modify carry : **No** OCP : **No** % IDX_MOD : **Yes** (Address Restriction)

Create a new process and put it into the address in MEMORY pointed by the 1st param.

The new process keeps all the value in the REGISTER and CARRY, only the PC differs

**NEW UPDATE : the value of 1st param is a short**

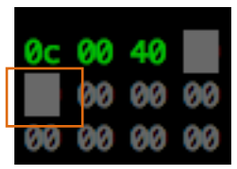
Example : (D2)

```
fork %64
```

→ straight forward, do a forking into the address in MEMORY pointed by the 1st param

PC + (64 % IDX_MOD) = **64**

(PC)




→ The new process is generates here in memory 0x64

If still cannot understand it, to be simple :

this operation is ***!!KAGE BUNSHIN NO JUTSU (Shadow Clone)!!***

# lld - Long Load Direct (opcode 0x0D)

Duration to execute : **10cycles**

Save a value into register, takes 2 parameter

S (ID | D4), D (RG) Modify carry : **Yes** OCP : **Yes** % IDX_MOD : **No** (Address Restriction)

Same as ld (0x02) but without address restriction

**NEW UPDATE : the value of the 1st param is an integer**

No example, it works ***TOTALLY*** the same as the ld

New observation :

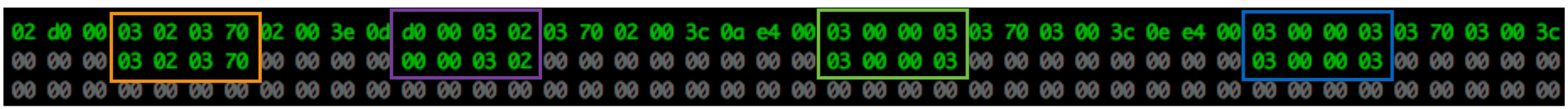
is ID is the 1st param, it will takes only 2 bytes from the pointed address to load it into the REGISTER, all other operation (ld, ldi, lldi) takes 4 bytes to load it.

by using the -v 4 flag in the corer, this will be the printed result :

ld 3, r2 ld 50463600 r2
ldi 3, %4, r3 ldi 50332675 4 r3
lld 3, r2 lld 770 r2
lldi 3, %4, r3 lldi 50332675 4 r3

```
ld 3, r2
st r2, 62
lld 3, r2
st r2, 60
ldi 3, %0, r3
st r3, 60
lldi 3, %0, r3
st r3, 60
```

As we can see from the result below, ldi is only reading 2 bytes from the address pointed by 1st param if it is param type ID
If the param type is D4, it will take the full integer value



# lldi - Long Load Index (opcode 0x0E)

Duration to execute : **10cycles**

Save a value into register, takes 3 parameter

S (RG | ID | D2), S (RG | D2), D (RG)  Modify carry : **Yes**  OCP : **Yes**  % IDX_MOD : **No** (Address Restriction)

Same as ldi (0x0a) but without address restriction

**NEW UPDATE : if the 1st or 2nd param is D2, it is a short, if it is ID, it is an integer**

No example, it works ***TOTALLY*** the same as the ldi

# lfork - Long Fork (opcode 0x0F)

Duration to execute : **1000cycles**

Generates a new process takes 1 parameter

S (D2) Modify carry : **No** OCP : **No** % IDX_MOD : **No** (Address Restriction)

Same as fork (0x0c) but without address restriction

**NEW UPDATE : the value of 1st param is a short**

No example, it works ***TOTALLY*** the same as the fork

# aff - Aff (opcode 0x10)

Duration to execute : **2cycles**

Useless operation, Really useless

S (RG) Modify carry : **No** OCP : **Yes** % IDX_MOD : **No** (Address Restriction)

Get the value from the REGISTER specified at 1st param DISPLAY it in Standard output in ASCII % 256

I found no way to modify carry it with aff as well. So it is really useless other then wasting space ad show message?

Example : (RG)

```
ld  %52, r3
ld  %53, r2
aff r3
aff r2
```

→

Store the value of 52 into r3
Store the value of 53 into r2
display the ASCII of (52 % 256) 52 which is 4 in Standard output
display the ASCII of (53 % 256) 53 which is 5 in Standard output

```
➜ vm_champs ./asm t.s && ./corewar -a t.cor
Writing output program to t.cor
Introducing contestants...
* Player 1, weighing 20 bytes, "" ("") !
Aff: 4
Aff: 5
Contestant 1, "", has won !
```

ZAZ corewar only show aff with the flag -a and it work with any other combination of flags

```
Aff: 5
    02 90 00 00 00 34 03 02 90 00 00 00 35 02 10 40 03 10 40 02
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
➜ vm_champs ./asm t.s && ./corewar -v 4 -a t.cor
Writing output program to t.cor
Introducing contestants...
* Player 1, weighing 20 bytes, "" ("") !
P    1 | ld 52 r3
P    1 | ld 53 r2
Aff: 4
Aff: 5
Contestant 1, "", has won !
```

```
➜ vm_champs ./asm t.s && ./corewar -v 16 -a t.cor
Writing output program to t.cor
Introducing contestants...
* Player 1, weighing 20 bytes, "" ("") !
ADV 7 (0x0000 -> 0x0007) 02 90 00 00 00 34 03
ADV 7 (0x0007 -> 0x000e) 02 90 00 00 00 35 02
Aff: 4
ADV 3 (0x000e -> 0x0011) 10 40 03
Aff: 5
ADV 3 (0x0011 -> 0x0014) 10 40 02
Contestant 1, "", has won !
```

# Flag -v -4

```
live %-1
ld 3, r2
st r2, 100
add r2, r3, r5
sub r2, r3, r4
and r2, %3, r4
or r2, 3, r4
xor r2, 3, r4
zjmp %1
ldi %4, %4, r4
ldi 3, %4, r3
sti r5, %4, %53
sti r6, 5, %57
fork %1000
lld 3, r2
lldi 3, %4, r3
lfork %4005
live %-1
live %-2
aff r2
```

```
➜ vm_champs ./asm t.s && ./corewar -v 4 -a t.cor
Writing output program to t.cor
Introducing contestants...
* Player 1, weighing 107 bytes, "tester" ("test aff") !
P    1 | live -1
P    1 | ld 50463600 r2
P    1 | st r2 100
P    1 | add r2 r3 r5
P    1 | sub r2 r3 r4
P    1 | and 50463600 3 r4
P    1 | or 50463600 197640 r4
P    1 | xor 50463600 197641 r4
P    1 | zjmp 1 FAILED
P    1 | ldi 4 4 r4
       | -> load from 4 + 4 = 8 (with pc and mod 56)
P    1 | ldi 50332675 4 r3
       | -> load from 50332675 + 4 = 50332679 (with pc and mod 62)
P    1 | sti r5 4 53
       | -> store to 4 + 53 = 57 (with pc and mod 119)
P    1 | sti r6 3738627 57
       | -> store to 3738627 + 57 = 3738684 (with pc and mod 129)
P    1 | fork 1000 (564)
P    1 | lld 770 r2
P    1 | lldi 50332675 4 r3
       | -> load from 50332675 + 4 = 50332679 (with pc 50332763)
P    1 | lfork 4005 (4096)
P    3 | live -1
P    1 | live -1
P    3 | ld 50463600 r2
P    3 | st r2 100
P    1 | live -2
Aff:
P    3 | add r2 r3 r5
P    3 | sub r2 r3 r4
P    3 | and 50463600 3 r4
P    3 | or 50463600 197640 r4
P    3 | xor 50463600 197641 r4
P    3 | zjmp 1 FAILED
P    3 | ldi 4 4 r4
       | -> load from 4 + 4 = 8 (with pc and mod 56)
P    3 | ldi 50332675 4 r3
       | -> load from 50332675 + 4 = 50332679 (with pc and mod 62)
P    3 | sti r5 4 53
       | -> store to 4 + 53 = 57 (with pc and mod 119)
P    3 | sti r6 3738627 57
```

This is the Process Number, not the player ID

- live print out the value as it is.
- ld 1st param is ID so it prints out the value in decimal number.
- Every Param type in REG is printed as r[reg_num].
- Except : and, or, xor which the 1st and 2nd param can be any param type, so even if we put a REG as the param type, it will print the value of that REG.
- ldi and sti can have REG or other param type as well, which in this case, -v 4 will print the REG value. *(in conclusion, only operation with param type only accept REG will print it as r[reg_num]*
- zjmp will print FAILED if carry is 0 and OK if carry is 1
- ldi, sti, lldi will print out the calculation with the following format "param1 + param2 = result (with pc and mod [the result of the PC (result % IDX_MOD)]"
- fork print the value (pc + (value % IDX_MOD)).
- lfork print the value (pc + value) .
- the operation of aff does not shown here, the Aff there is cause by -a flag, but i still cannot figure where or when it should be flush out to standard output