Operations

Opcode	Name	Arguments			Binary	Hexa	Description	Carry	Codage Octal	Cycles	Label size
1	**live**	T_DIR	-	-	00000001	0x01	alive	0	0	10	4
2	**ld**	T_DIR | T_IND	T_REG	-	00000010	0x02	load	1 или 0	1	5	4
3	**st**	T_REG	T_REG | T_IND	-	00000011	0x03	store	0	1	5	4
4	**add**	T_REG	T_REG	T_REG	00000100	0x04	addition	1 или 0	1	10	4
5	**sub**	T_REG	T_REG	T_REG	00000101	0x05	substraction	1 или 0	1	10	4
6	**and**	T_REG | T_DIR | T_IND	T_REG | T_DIR | T_IND	T_REG	00000110	0x06	and r1, r2, r3 r1 & r2 -> r3	1 или 0	1	6	4
7	**or**	T_REG | T_DIR | T_IND	T_REG | T_DIR | T_IND	T_REG	00000111	0x07	or r1, r2, r3 r1 | r2 -> r3	1 или 0	1	6	4
8	**xor**	T_REG | T_DIR | T_IND	T_REG | T_DIR | T_IND	T_REG	00001000	0x08	xor r1, r2, r3 r1 ^ r2 -> r3	1 или 0	1	6	4
9	**zjmp**	T_DIR	-	-	00001001	0x09	jump if carry == 1	0	0	20	2
10	**ldi**	T_REG | T_DIR | T_IND	T_REG | T_DIR	T_REG	00001010	0x0A	load index	0	1	25	2
11	**sti**	T_REG	T_REG | T_DIR | T_IND	T_REG | T_DIR	00001011	0x0B	store index	0	1	25	2
12	**fork**	T_DIR	-	-	00001100	0x0C	fork	0	0	800	2
13	**lld**	T_DIR | T_IND	T_REG	-	00001101	0x0D	long load	1 или 0	1	10	4
14	**lldi**	T_REG | T_DIR | T_IND	T_REG | T_DIR	T_REG	00001110	0x0E	long load index	1 или 0	1	50	2
15	**lfork**	T_DIR	-	-	00001111	0x0F	long fork	0	0	1000	2
16	**aff**	T_REG	-	-	00010000	0x10	aff	0	1	2	4

Arguments

Name	Sign	Binary code	Encod: (octet)	Значення
T_REG	r	01	1	Регистр rx (где x = число, которое находится в диапазоне от 1 до REG_NUMBER)
T_DIR	%	10	2-4	Число 2 или 4 байта в зависимости от label
T_IND		11	2	Перейдёт на число указаное в T_IND от PC и считает 4 байта
Label				abcdefghijklmnopqrstuvwxyz_0123456789

.COR FILE STRUCTURE

size in bytes	description	
4	magic	
PROG_NAME_LENGTH	bot name	if ((PROG_NAME_LENGTH + 1) % 4 != 0) => вирівнювання = 4 - (PROG_NAME_LENGTH + 1) % 4 - перевірка на вирівнювання
4	NULL	
4	size of executable code	
COMMENT_LENGTH	bot comment	
4	NULL	
N	executable code	

Processus value for each champions

Name	Qty	Descriptions
Carry	1	Флаг, который меняется некоторыми инструкциями и используется в **zjmp**
PC	1	Позиция процесса (каретки)
Registres	REG_NUMBER	(своего рода буфер на) REG_NUMBER регистров (переменных), каждый из которых, занимает REG_SIZE байт, в которые процессы (PC) могут записывать значения

Virtual Machine

Name	Description short	Description long
live	"жизнь" процесса	Выполняет **2** операции: **1.** Засчитывает, что процесс (который выполняет данную команду) жив. **2.** Засчитывает, что жив номер (если этот номер совпадает с номером игрока, то засчитает, что этот игрок жив), который заходит как аргумент (**T_DIR**).
ld	косвенная загрузка	Если первый аргумент **T_DIR**, то идёт просто запись первого аргумента в **T_REG**. Если первый аргумент **T_IND**, то сначала **T_IND** перезаписывается на **T_IND % IDX_MOD**, а потом идём на ячейку, от **текущей позиции** + это **значение**, с той позиции считываем **4** байта и записываем в **T_REG**. В зависимости от того, что записали в **T_REG** меняем **carry**. Если записали **0** меняем **carry** на **1**, если **не 0** меняем на **0**.
st	косвенная запись	Значение **T_REG** (первый аргумент) записывается: - Если второй аргумент **T_IND** - то в ячейку, по адресу (**текущая позиция PC** плюс (**T_IND % IDX_MOD**)) - Если второй аргумент **T_REG** - то в регистр, по этому номеру.
add	сложение	Результат (**первый** плюс **второй** аргумент) записывается в **третий**. В зависимости от того, что записали в **третий** меняем **carry**. Если записали **0** меняем **carry** на **1**, если **не 0** меняем на **0**.
sub	вычитание	Результат (**первый** минус **второй** аргумент) записывается в **третий**. В зависимости от того, что записали в **третий** меняем **carry**. Если записали **0** меняем **carry** на **1**, если **не 0** меняем на **0**.
and	побитовое <И>	Применяет **&** для первых двух аргументов и записывает результат в третий аргумент Меняет **carry** на **1** если результат операции был **0** или Меняет **carry** на **0** если результат операции был **не 0**
or	побитовое <ИЛИ>	Аналогично **and** только **&** меняется на **|**
xor	исключающее <ИЛИ>	Аналогично **and** только **&** меняется на **^**
zjmp	косвенный переход	Перемещает **PC** с текущей позиции на **T_DIR % IDX_MOD** если **carry** равен **1**
ldi	косвенная загрука по индексу	С позиции (((**первый** плюс **второй** аргумент) **% IDX_MOD**) плюс **текущая позиция PC**) считается **4** байта и записывается в **третий** аргумент. - Если **первый** аргумент **T_IND** - то **значение** первого аргумента для операции будет: **4** байта считанные с позиции ((**T_IND % IDX_MOD**) плюс **текущая позиция PC**).
sti	косвенная запись по индексу	Значение **T_REG** (первый аргумент) записывается в ячейку, по адресу (**текущая позиция PC** плюс ((**второй аргумент** плюс **третий аргумент**) **% IDX_MOD**)) - Если второй аргумент **T_IND** - то ясное дело, что вместо **второго аргумента**, в уровнение подставляются те **4** байта, которые мы берём из ячейки (**T_IND % IDX_MOD**).
fork	новый процесс	Значение ((**T_IND % IDX_MOD**) плюс **текущая позиция PC**) является **позицией**, на которой создаётся копия **текущего процесса**, со всеми его параметрами (кроме **позиции**).
lld	LONG ld	Аналогично **ld** но без % **IDX_MOD**. Стоит отметить, что оригинальный **corewar** работает не правильно. При аргументе **T_IND** считывает и записывает в **T_REG** не **4** байта, а только **2**.
lldi	LONG ldi	Аналогично **ldi** но без % **IDX_MOD** (это касается только операции ((**первый** аргумент плюс **второй**) плюс **позиция PC**), при операции (**T_IND** % **IDX_MOD**) **IDX_MOD** всё так же **учитывается**). И в зависимости от того, что записали в **третий** аргумент меняем **carry**. Если записали **0** меняем **carry** на **1**, если **не 0** меняем на **0**.
lfork	LONG fork	Аналогично **fork** но без **% IDX_MOD**
aff	вывод значения на экран	Значение из аргумента **% 256** выводиться на экран как **ASCII** символ.

Assembler errors

Validation	Type	Error message
name	нет стоки	Syntax error at token [TOKEN][004:001] LABEL "l2:"
name	нет имени	Syntax error at token [TOKEN][001:014] ENDLINE
name	нет закрывающейся кавычки	Syntax error at token [TOKEN][009:001] END "(null)"
name	нет открывающейся кавычки	Lexical error at [2:10]
name	нет кавычек	Syntax error at token [TOKEN][001:007] INSTRUCTION "zork"
comment	нет кавычек	Lexical error at [2:10]
comment	нет закрывающейся кавычки	Syntax error at token [TOKEN][009:001] END "(null)"
comment	нет открывающейся кавычки	Lexical error at [2:10]
comment	нет строки	Syntax error at token [TOKEN][004:001] LABEL "l2:"
comment	нет имени	Syntax error at token [TOKEN][001:014] ENDLINE
comands	нет команд	Syntax error at token [TOKEN][004:001] END "(null)"
comands	нет комманды указанной в аргументе T_IND	No such label live while attempting to dereference token [TOKEN][004:014] DIRECT_LABEL "%:live"
comands	нет комманды указанной в LABLE	Invalid instruction at token [TOKEN][005:003] INSTRUCTION "___"
comands	указанно больше аргументов	Syntax error at token [TOKEN][007:015] DIRECT "%1"
comands	указан не корректный аргумент	Invalid parameter 0 type register for instruction live
comands	в команде	Syntax error at token [TOKEN][004:025] INSTRUCTION "___"
lable	указан LABLE без команд	Syntax error at token [TOKEN][010:005] END "(null)"
name	два поля .name	Lexical error at [3:11]
comment	два поля .comment	Syntax error at token [TOKEN][004:001] COMMAND_COMMENT ".comment"

ZORK EXPLANATION

operation	note	value	hexa	byte №	
sti			0b	0	Здесь OPCODE
	codage	01 10 10 00 = 0x68	68	1	Кодировка: 1 - per, 2 -прям, 3 - прям
	arg1	r1	01	2	01 -- первый регистр r1 = 0;
	arg2	%:live	00	3	выделено байт 3 и 4
			0f	4	Здесь значение (0f)
	arg3	1%	00	5	выделено байт 5 и 6
			01	6	Здесь значение (01)
and			06	7	Здесь OPCODE 0x06
	codage	01 10 01 00 = 0x64	64	8	Кодировка: 1 - per, 2 -прям, 3 - регистр
	arg1	r1	01	9	Здесь значение (01) (1 байт для per)
	arg2	%0	00	10	для direct and выделяется 4 байта
			00	11	и в значении у нас ноль, так что
			00	12	00 00 00 00
			00	13	00000000 00000000 00000000 00000000
	arg3	r1	01	14	тут регистр первый, выделено 1 байт
live			01	15	Здесь OPCODE 0x01
	arg1	%1	00	16	под arg1 выделено 4 байта
			00	17	(бо так надо, написано в табличе)
			00	18	и значение 01
			01	19	
zjmp			09	20	Здесь OPCODE 0x09
	arg1	%:live	ff	21	12
			fb	22	прыгаем на -5