

Operations												
Opcode	Name	Arguments			Binary	Hexa	Description	Carry	Codage Octal	Cycles	Label size	
1	lve	T_DIR	-	-	00000001	0x01	alive	0	0	10	4	
2	ld	T_DIR,T_DIR,IND	T_REG	-	00000010	0x02	load	1 vers 0	1	5	4	
3	st	T_REG	T_REG,T_DIR,IND	-	00000011	0x03	store	0	5	5	4	
4	add	T_REG	T_REG	T_REG	00000100	0x04	addition	1 vers 0	1	10	4	
5	sub	T_REG	T_REG	T_REG	00000101	0x05	subtraction	1 vers 0	1	10	4	
6 and	T_REG,T_DIR,T_DIR,IND	T_REG,T_DIR,T_DIR,IND	T_REG	-	00000110	0x06	and r1,r2,r3 r1 & r2 > r3	1 vers 0	1	6	4	
7	or	T_REG,T_DIR,T_DIR,IND	T_REG,T_DIR,T_DIR,IND	T_REG	00000111	0x07	or r1,r2,r3 r1 r2 > r3	1 vers 0	6	6	4	
8	xor	T_REG,T_DIR,T_DIR,IND	T_REG,T_DIR,T_DIR,IND	T_REG	00001000	0x08	xor r1,r2,r3 r1 ^ r2 > r3	1 vers 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_DIR,IND	T_REG,T_DIR,T_DIR	T_REG	00001010	0x0A	load index	0	1	25	2	
11	sti	T_REG	T_REG,T_DIR,T_DIR,IND	T_REG,T_DIR,T_DIR	00001011	0x0B	store index	0	25	2	2	
12	fork	T_DIR	-	-	00001100	0x0C	fork	0	0	800	2	
13	lfd	T_DIR,T_DIR	T_REG	-	00001101	0x0D	long load	1 vers 0	1	20	4	
14	lfdi	T_REG,T_DIR,T_DIR,T_DIR,IND	T_REG,T_DIR,T_DIR,IND	T_REG	00001110	0x0E	long load index	1 vers 0	1	30	2	
15	fork	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	Encode: (octet)	Значение
T_REG	r	01	1	Перекрут rx (раз с = число, количество вращений от 1 до REG_NUMBER)
T_DIR	%	10	24	Магистр 2-го д-ла и зависимость от label
T_IND		11	2	Переключатель на число вращения T_IND от PC и отсчитав 4-й бит
Label				abcedefghijklmnopqrstuvwxyz: 0123456789

.COR FILE STRUCTURE		
size in bytes	description	
4	magic	
PROG_NAME_LENGTH	bot name	if (PROG_NAME_LENGTH + 1) % 4 != 0 => выравнивание = 4 - (PROG_NAME_LENGTH + 1) % 4 - переписна на выравнивание
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 <code>jmp</code>
PC	1	Позиция процесса (адрес)
Registers	REG_NUMBER	(своего рода буфер на) REG_NUMBER регистра (переменных), каждый из которых, занимает REG_SIZE байт, в которые процессор (PC) может записывать значения

Virtual Machine		
Name	Description short	Description long
live	"жизнь" процесса	Выполнение 2 операции. 1. Засчитывает, что процесс (если выполнялся ранее) завершился или нет. 2. Засчитывает, что жидк. носитель (если выполнялся ранее) закончил с номером операции, что этот жидк. носитель, который заводит как аргумент (T_REG).
ld	исполненная загрузка	Если первый аргумент T_REG, то зачитывает значение первого аргумента в T_REG. Если первый аргумент T_IND, то считывает T_IND переводимое значение в T_REG, а потом читает из памяти, что находится в T_REG, это значение. Значение T_REG переводимое значение в память. Если записано 0 битов в биты в T_REG, то значение в T_REG не меняется. Если записано 0 битов в биты в T_REG, то значение в T_REG не меняется на 0.
st	исполненная запись	Значение T_REG (первый аргумент) записывается. Если второй аргумент T_IND, то в память по адресу (текущая позиция PC плюс (T_IND * IDX_MOD)) записывается значение T_REG. Если второй аргумент T_REG, то в регистр, по этому номеру. Значение T_REG (первый аргумент) записывается в T_REG. Если второй аргумент T_IND, то в память по адресу (текущая позиция PC плюс (T_IND * IDX_MOD)) записывается значение T_REG. Если второй аргумент T_REG, то в регистр, по этому номеру.
add	сложение	Если записано 0 битов в биты в T_REG, то значение в T_REG не меняется. Если записано 0 битов в биты в T_REG, то значение в T_REG не меняется на 0.
sub	вычитание	Результат (первый минус второй аргумент) записывается в T_REG. Значение T_REG переводимое значение в память. Если записано 0 битов в биты в T_REG, то значение в T_REG не меняется. Если записано 0 битов в биты в T_REG, то значение в T_REG не меняется на 0.
and	логическое «И»	Применяет «И» для первого аргумента с записанным результатом в третий аргумент. Меняет carry на 1, если результат операции был 0. Меняет carry на 0, если результат операции был 1.
or	логическое «ИЛИ»	Аналогично и только с изменением на 1.
xor	исключающее «ИЛИ»	Аналогично и только с изменением на 1.
zfp	исполненный переход	Переводит PC в текущую позицию на T_REG * IDX_MOD если carry равен 1. Если позиция (первый минус второй аргумент) в T_REG * IDX_MOD (плюс текущая позиция PC) считается 4 бита и записывается в третий аргумент. Если первый аргумент T_IND, то значение первого аргумента для операции будет: 4 бита считывается с позиции (T_IND * IDX_MOD) плюс текущая позиция PC.
ldi	исполненная загрузка по индексу	Значение T_REG (первый аргумент) записывается в память. Значение T_REG (второй аргумент) записывается в T_REG. Если второй аргумент T_IND, то значение для, что вместо второго аргумента, в уроне подставляется те 4 бита, которые мы берем из памяти (T_IND * IDX_MOD).
sti	исполненная запись по индексу	Значение T_REG (первый аргумент) записывается в память. Значение T_REG (второй аргумент) записывается в T_REG. Если второй аргумент T_IND, то значение для, что вместо второго аргумента, в уроне подставляется те 4 бита, которые мы берем из памяти (T_IND * IDX_MOD).
fork	новый процесс	Значение T_REG (первый аргумент) записывается в память. Значение T_REG (второй аргумент) записывается в T_REG. Если второй аргумент T_IND, то значение для, что вместо второго аргумента, в уроне подставляется те 4 бита, которые мы берем из памяти (T_IND * IDX_MOD).
ldi	LONG ld	Аналогично ld но без T_IND * IDX_MOD. Считает, что отрицательный сдвиг работает неправильно. При аргументе T_IND считывает и записывает в T_REG 4 бита, а только 2.
ldi	LONG i	Аналогично ld но без T_IND * IDX_MOD (то значение первого аргумента (первый аргумент плюс второй аргумент) плюс позиция PC). При операции (T_IND * IDX_MOD) IDX_MOD не так учитывается. Если записано 0 битов в биты в T_REG, то значение в T_REG не меняется. Если записано 0 битов в биты в T_REG, то значение в T_REG не меняется на 0.
fork	LONG fork	Аналогично fork но без T_IND * IDX_MOD.
gdt	выход записан на экран	Значение из регистров * 256 выводится на экран как ASCII символ.

Assembler errors		
Validation	Type	Error message
name	NET STOKH	Syntax error at token (TOKEN0004 0001) LABEL "2"
name	NET MAMBI	Syntax error at token (TOKEN0010 0154) ENDLINE
name	NET ЗАПЯТЫЙКАВЫЙ	Syntax error at token (TOKEN0009 0001) END "IPI"
name	NET ПЕРЕКРЫВАЮЩАЯСЯКАВЫЙ	Lexical error at [2:10]
name	NET КАБЫЧКА	Syntax error at token (TOKEN0001 0007) INSTRUCTION "FOOK"
comment	NET КАБЫЧКА	Lexical error at [2:10]
comment	NET ЗАПЯТЫЙКАВЫЙКАБЫЧ	Syntax error at token (TOKEN0009 0001) END "IPI"
comment	NET ПЕРЕКРЫВАЮЩАЯКАБЫЧ	Lexical error at [2:10]
comment	NET СТРОК	Syntax error at token (TOKEN0004 0001) LABEL "2"
comment	NET МЯМИ	Syntax error at token (TOKEN0001 0154) ENDLINE
comment	NET КАВЫЧКА	Syntax error at token (TOKEN0004 0001) LABEL "2"
commands	NET команды указывающей в аргументе T. END	No such label file while attempting to dereference token (TOKEN0004 0154) DIRECT LABEL "IPI" via "sv"
commands	NET команды указывающей LABEL	Invalid instruction at token (TOKEN0005 0003) INSTRUCTION " *"
commands	указано больше аргументов	Syntax error at token (TOKEN0007 0155) DIRECT "IPI"
commands	указан не существующий аргумент	Invalid parameter 2 type register for instruction file
commands	в команде	Syntax error at token (TOKEN0004 0003) INSTRUCTION " *"
label	указан LABEL без команды	Syntax error at token (TOKEN0005 0003) END "IPI"
label	два раза name	Syntax error at [2:10]
comment	два раза comment	Syntax error at token (TOKEN0004 0003) COMMENT "COMMENT" comment"

ZORK EXPLANATION					
operation	note	value	hexa	byte No	
sti				0	Здесь OPCODE
	codage	01 10 10 00 = 0x68	68	1	Кодировка: 1. пер. 3-м. 3. регистр
	arg1	r1	01	2	1 - первый регистр r1 = 0
	arg2	%live	01	3	выделено байт 3 = 4
			01	4	Здесь значение (00)
	arg3	1%	00	5	выделено байт 5 = 6
			01	6	Здесь значение (01)
and			06	7	Здесь OPCODE and6
	codage	01 10 01 00 = 0x64	64	8	Кодировка: 1. пер. 3-м. 3. регистр
	arg1	r1	01	9	Здесь значение (01) (1 байт для рег)
	arg2	%0	00	10	для direct и выделится 4 байта
			00	11	и значение у нас такое, так что
			00	12	00 00 00 00
			00	13	00000000 00000000 00000000 00000000
	arg3	r1	01	14	7-й регистр передан, выделено 1 байт
			01	15	Здесь OPCODE not6
	arg1	%1	00	16	not arg1 выделена 4 байта
			00	17	(до так надо, написано в таблице)
			01	18	и значение 01
zmp			09	20	Здесь OPCODE and5
	arg1	%live	01	21	
			B	22	выделено arg 5