Assembly_to_byte code

Jump to bottom

k-off edited this page on Sep 28, 2019 · 5 revisions

File structure

To understand the rules of translation from source code into byte-code, we have to take a look at the structure of the file `.cor`.

Let's translate one of the champions with the provided `asm`:

```
.name       "Batman"
.comment    "This city needs me"

loop:
        sti r1, %:live, %1
live:
        live %0
        ld %0, r2
        zjmp %:loop
```

This is how the `.cor` content looks like:

MAGIC HEADER

CHAMPION NAME

NULL

CHAMPION EXEC CODE SIZE

```
00ea 83f3 4261 746d 616e 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0016 5468 6973
2063 6974 7920 6e65 6564 7320 6d65 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

CHAMPION COMMENT

```
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

NULL

CHAMPION EXEC CODE

```
0b68 0100 0700 0101 0000 0000 0290 0000
0000 0209 ffed
```

Magic header

first four bytes of the file are "magic header"

It is defined in the `COREWAR_EXEC_MAGIC` constant in the file `op.h` as `0xea83f3`.

What is magic header why use it?

Magic header is a signature of the file and means file is of a certain type (make analogy to the extension).

If you see a file with `.cor` extension, you assume it is a corewar champion. The virtual machine will also check the magic header and will only execute the file if it does contain one.

Champion name

Next 128 bytes are champion's name. 128 is the value of constant `PROG_NAME_LENGTH` from `op.h` file.

If the actual name is shorter than 128 bytes. The remaining space will be filled in with trailing zeroes.

Each character is written to the file as it's ASCII value:

Character	`B`	`a`	`t`	`m`	`a`	`n`
ASCII-code	`0x42`	`0x61`	`0x74`	`0x6d`	`0x61`	`0x6e`

NULL

Next four bytes are NULL-bytes. Their goal is to be present at this place and to be NULL. If they are not, file is invalid.

Champion exec code size

Next four bytes represent the size of champion's executable code (**only executable part of champion**).

The virtual machine checks whether executable code size doesn't exceed `CHAMP_MAX_SIZE` value from `op.h` file (`682` bytes).

Champion comment

Next 2048 bytes represent champions comment. It is same as champion name excepting max size `COMMENT_LENGTH`.

NULL

Next four bytes are NULL-bytes.

Champion exec code

The last part of the file is executable code.

No trailing zeroes in this part.

Operations encoding

We will need two tables to understand how encoding works.

Operations table

Code	Name	Argument #1	Argument #2	Argument #3	Codage octet	Size `T_DIR`
0x01	`live`	`T_DIR`	—	—	no	4

Code	Name	Argument #1	Argument #2	Argument #3	Codage octet	Size `T_DIR`
0x02	`ld`	`T_DIR` / `T_IND`	`T_REG`	—	present	4
0x03	`st`	`T_REG`	`T_REG` / `T_IND`	—	present	4
0x04	`add`	`T_REG`	`T_REG`	`T_REG`	present	4
0x05	`sub`	`T_REG`	`T_REG`	`T_REG`	present	4
0x06	`and`	`T_REG` / `T_DIR` / `T_IND`	`T_REG` / `T_DIR` / `T_IND`	`T_REG`	present	4
0x07	`or`	`T_REG` / `T_DIR` / `T_IND`	`T_REG` / `T_DIR` / `T_IND`	`T_REG`	present	4
0x08	`xor`	`T_REG` / `T_DIR` / `T_IND`	`T_REG` / `T_DIR` / `T_IND`	`T_REG`	present	4
0x09	`zjmp`	`T_DIR`	—	—	no	2
0x0a	`ldi`	`T_REG` / `T_DIR` / `T_IND`	`T_REG` / `T_DIR`	`T_REG`	present	2
0x0b	`sti`	`T_REG`	`T_REG` / `T_DIR` / `T_IND`	`T_REG` / `T_DIR`	present	2
0x0c	`fork`	`T_DIR`	—	—	no	2
0x0d	`lld`	`T_DIR` / `T_IND`	`T_REG`	—	present	4

Code	Name	Argument #1	Argument #2	Argument #3	Codage octet	Size `T_DIR`
0x0e	`lldi`	`T_REG` / `T_DIR` / `T_IND`	`T_REG` / `T_DIR`	`T_REG`	present	2
0x0f	`lfork`	`T_DIR`	—	—	no	2
0x10	`aff`	`T_REG`	—	—	present	4

> **What is «Size `T_DIR`» stands for?**
>
> In short, to read/writes correct amount of bytes from/to vm memory.
>
> In more detail it will be covered in corresponding chapter.

The complete operations table

Arguments table

The second table contains codes of types of arguments, and size of arguments.

Type	Assembly	Code	Size
`T_REG`	`r`	`01`	1 byte
`T_DIR`	`%`	`10`	`T_DIR` size
`T_IND`	—	`11`	2 bytes

The complete arguments table

> **Registries and their sizes**
>
> There are two characteristics of a registry

> Name of registry (`r1`, `r2`...) has size of 1 byte and is laced in the byte-code. But the registry itself is 4 byte big, as defined in the `REG_SIZE` constant, and is a variable of cursor.

> **`T_DIR` arguments size**
>
> As we can see in the operations table, size of arguments of type `T_DIR` is not fixed and depends on operation. but file `op.h` contains a constant `T_DIR` defined with value — 4:
>
> ```
> # define IND_SIZE     2
> # define REG_SIZE     4
> # define DIR_SIZE     REG_SIZE
> ```
>
> What is the logic?
>
> Operations with size 2 for `T_DIR` argument use this argument as a relative address (as an argument of type `T_IND`), and `T_IND` size is always 2 bytes.

Encoding algorithm

Each operation in byte-code has the following structure:

1. Operation code — 1 byte
2. Encoding byte for saving arguments types (if needed) — 1 byte
3. Arguments (see sizes in the table)

> **Encoding byte for arguments types**
>
> We can check whether it is needed for current operation from the operations table. If operation has one single argument and it's type is `T_DIR`, then encoding byte is not used. For all other operations encoding byte must be present.

Let's encode some operations:

```
loop:
        sti r1, %:live, %1
live:
        live %0
        ld %0, r2
        zjmp %:loop
```

Operation #1

```
loop:
        sti r1, %:live, %1
```

Define size for all parts of operation.

Operation code	Encoding byte	Argument #1	Argument #2	Argument #3
1 byte	1 byte	1 byte	2 bytes	2 bytes

Find values for all parts of operation.

Operation code

The code for each operation can be found in the operations table (1-16). For `sti` it is `11`.

Encoding byte for the arguments types

Lets write encoding byte in **binary form**. Left most pair of bits stand for type of argument #1, next pair - for type of argument #2, third pair - for type of argument #3. Last pair is always `00`. Codes for different types can be found in **Arguments table**.

Argument #1	Argument #2	Argument #3	—	The whole byte	Decimal	Hex
`T_REG`	`T_DIR`	`T_DIR`	—	—		

Argument #1	Argument #2	Argument #3	—	The whole byte	Decimal	Hex
`01`	`10`	`10`	`00`	`01101000`		

Argument-registry `T_REG`

Convert the number of registry `r1` into `0x01`.

Argument-label `T_DIR`

Label is converted into a number, which represents distance in bytes from the current position.

Label `live` points to the next operation, we know that current operation is 7 bytes, so the value to write is 7.

This value must be written on 2 bytes — `0x0007`.

Argument-number `T_DIR`

In this case we just take value and write as is on 2 bytes.

Final output for this operation

System	Operation code	Encoding byte	Argument #1	Argument #2	Argument #3	
`Decimal`	`11`	`104`	`1`	`7`	`1`	
`Hex`	`0x0B`	`0x68`	`0x0001`	`0x0007`	`0x01`	

Operation #2

Repeat for the next operation:

```
live:
        live %0
```

The only major difference is that this operation doesn't have the encoding byte for argument types:

Operation code	Argument #1
1 byte	4 bytes

Final output in hex — `01 0000 0000`.

Instruction #3

```
        ld %0, r2
```

Operation code	Encoding byte	Argument #1	Argument #2
1 byte	1 byte	4 bytes	1 byte

Final output in hex — `02 90 00 00 00 00 02`.

Instruction #4

```
        zjmp %:loop
```

Operation code for `zjmp` is `0x09`.

Encoding byte is not used here

Operation code	Argument #1
1 byte	2 bytes

Final output in hex — `09 ff ed`

Result

Executable code of the champion will look like:

```
0b68 0100 0700 0101 0000 0000 0290 0000
0000 0209 ffed
```


Clone this wiki locally

```
https://github.com/k-off/Corewar.wiki.git
```