

JIT translator

Generated by Doxygen 1.9.4



<b>1 Class Index</b>	<b>1</b>
1.1 Class List	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Class Documentation</b>	<b>5</b>
3.1 assembly_code Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Member Data Documentation	5
3.1.2.1 code	5
3.1.2.2 position	5
3.1.2.3 size	6
3.2 cvt_u_int64_t_int Union Reference	6
3.2.1 Detailed Description	6
3.2.2 Member Data Documentation	6
3.2.2.1 extended_address	6
3.2.2.2 rel_addr	6
3.3 label_table Struct Reference	7
3.4 opcode Struct Reference	7
3.4.1 Detailed Description	7
3.4.2 Member Data Documentation	7
3.4.2.1 code	7
3.4.2.2 size	7
3.5 stack Struct Reference	8
3.6 stack_node Struct Reference	8
<b>4 File Documentation</b>	<b>9</b>
4.1 label_table.h	9
4.2 log.h	10
4.3 src/translator.h File Reference	11
4.3.1 Detailed Description	14
4.3.2 Macro Definition Documentation	14
4.3.2.1 HOST_MEMORY_COUNT	14
4.3.2.2 MIN_DST_CODE_SIZE	14
4.3.2.3 PAGESIZE	14
4.3.2.4 TRANSLATE_PUSH_SIZE	14
4.3.2.5 UNKNOWN	15
4.3.3 Enumeration Type Documentation	15
4.3.3.1 HOST_ASSEMBLY_REG_ID	15
4.3.3.2 HOST_STACK_OP_CODES	15
4.3.3.3 REG_MASK	15
4.3.3.4 TRANSLATION_ERROR	15

4.3.3.5 VCMPPD_COMPARISONS_CODE	15
4.3.3.6 WORD_SIZE	15
4.3.3.7 X86_ASSEMBLY_OPCODES	16
4.3.3.8 X86_ASSEMBLY_OPCODES_SIZE	16
4.3.4 Function Documentation	16
4.3.4.1 assembly_code_aligned_init()	16
4.3.4.2 assembly_code_init()	16
4.3.4.3 command_line_handler()	17
4.3.4.4 cvt_host_reg_id_to_native()	17
4.3.4.5 double_printf()	18
4.3.4.6 double_scanf()	18
4.3.4.7 execute_start()	18
4.3.4.8 jmp_n_call_handler()	19
4.3.4.9 link_label()	19
4.3.4.10 load_code()	20
4.3.4.11 make_label_table()	20
4.3.4.12 translate_ahead_jmp_n_call()	21
4.3.4.13 translate_arithmetic_op()	21
4.3.4.14 translate_cycle()	21
4.3.4.15 translate_jmp_n_call()	22
4.3.4.16 translate_load_rsp()	22
4.3.4.17 translate_push()	23
4.3.4.18 translate_push_r()	23
4.3.4.19 translate_ret()	23
4.3.4.20 translate_save_rsp()	24
4.3.4.21 translate_stdout()	24
4.3.4.22 translate_two_pop_for_cmp()	24
4.3.4.23 translation_start()	24
4.3.4.24 traslate_rel32_label()	25
4.3.4.25 write_command()	25
4.4 translator.h	25

# Chapter 1

## Class Index

### 1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">assembly_code</a>	5
<a href="#">cvt_u_int64_t_int</a>	6
<a href="#">label_table</a>	7
<a href="#">opcode</a>	7
<a href="#">stack</a>	8
<a href="#">stack_node</a>	8



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all documented files with brief descriptions:

lib/ <a href="#">label_table.h</a> . . . . .	9
lib/ <a href="#">log.h</a> . . . . .	10
src/ <a href="#">translator.h</a> The header of translator, containing all used functions in JIT . . . . .	11





## Chapter 3

# Class Documentation

### 3.1 `assembly_code` Struct Reference

```
#include <translator.h>
```

#### Public Attributes

- `char * code`
- `int position`
- `size_t size`

#### 3.1.1 Detailed Description

Structure containing the assembled code and host (source) assembler code

#### 3.1.2 Member Data Documentation

##### 3.1.2.1 `code`

```
char* assembly_code::code
```

Buffer to contation code

##### 3.1.2.2 `position`

```
int assembly_code::position
```

position of current opcode in buffer

### 3.1.2.3 size

```
size_t assembly_code::size
```

size of buffer

The documentation for this struct was generated from the following file:

- [src/translator.h](#)

## 3.2 cvt\_u\_int64\_t\_int Union Reference

```
#include <translator.h>
```

### Public Attributes

- int [rel\\_addr](#)
- u\_int64\_t [extended\\_address](#)

### 3.2.1 Detailed Description

Union to zero-extended conversation from int to u\_int64\_t

### 3.2.2 Member Data Documentation

#### 3.2.2.1 extended\_address

```
u_int64_t cvt_u_int64_t_int::extended_address
```

Zero extended value.

#### 3.2.2.2 rel\_addr

```
int cvt_u_int64_t_int::rel_addr
```

Value to convert. The name of the variable is because it is used primarily for address translation.

The documentation for this union was generated from the following file:

- [src/translator.h](#)

## 3.3 label\_table Struct Reference

### Public Attributes

- int **size**
- [stack](#) \* **elems**

The documentation for this struct was generated from the following file:

- lib/label\_table.h

## 3.4 opcode Struct Reference

```
#include <translator.h>
```

### Public Attributes

- u\_int64\_t **code**
- int **size**

### 3.4.1 Detailed Description

Structure used to simplify writing opcodes

### 3.4.2 Member Data Documentation

#### 3.4.2.1 code

```
u_int64_t opcode::code
```

Opcode, that should be written in the buffer

#### 3.4.2.2 size

```
int opcode::size
```

Size of this opcode

The documentation for this struct was generated from the following file:

- src/[translator.h](#)

## 3.5 `stack` Struct Reference

### Public Attributes

- `int` **size**
- `int` **capacity**
- `stack_node` \* **data**

The documentation for this struct was generated from the following file:

- `lib/label_table.h`

## 3.6 `stack_node` Struct Reference

### Public Attributes

- `int` **label**
- `int` **jmp**
- `size_t` **code\_pos**

The documentation for this struct was generated from the following file:

- `lib/label_table.h`

## Chapter 4

# File Documentation

### 4.1 label\_table.h

```
1 #if !defined LABEL_TABLE_INCLUDED
2
3 #include <stdint>
4 #include <stdlib.h>
5 #include <immintrin.h>
6 #define LABEL_TABLE_INCLUDED
7
8 #define HT_ERROR -1
9 #define NOT_FOUND -1
10
11 #define CYCLE 1
12 // 128 * 16 * 32 * 16
13 // TODO: Rework for dynamic value of size
14 #define MIN_STACK_SIZE 16
15 #define MIN_LABEL_TABLE_SIZE 128
16
17 #define STACK self->elems
18
19
20 struct stack_node
21 {
22     int label; //
23     int jmp;
24     size_t code_pos;
25 };
26
27 struct stack
28 {
29     int size;
30     int capacity;
31     stack_node* data;
32 };
33
34 struct label_table // Hash table for immediate value
35 {
36     int size;
37     stack* elems;
38 };
39
40
41 //+=====| STACK |======+
42
43
44 void stack_init(stack* const __restrict stack, const size_t size)
45     __attribute__((nonnull(1)));
46
47
48 int stack_push(stack* const __restrict stack,
49               const int key,
50               const int data)
51     __attribute__((nonnull(1)));
52
53 int stack_destr(stack* const __restrict stack)
54     __attribute__((nonnull(1)));
55 //-----
56
57
58 int label_table_init(label_table* const __restrict self)
```

```

59     __attribute__((nonnull(1)));
60
61 void label_table_manual_destr(label_table* const __restrict self)
62     __attribute__((nonnull(1)));
63
64
65 void label_table_add(label_table* const __restrict self,
66                     const int key,
67                     const int data)
68     __attribute__((always_inline, nonnull(1)));
69
70
71 void set_all_cycles(label_table* self,
72                    const int indx,
73                    const size_t code_pos,
74                    const int label_pos)
75     __attribute__((nonnull(1)));
76
77 inline int label_table_search_by_label(label_table* const __restrict self,
78                                       const int label_pos)
79     __attribute__((always_inline, hot));
80
81
82 size_t get_code_pos_by_jmp(label_table* const __restrict self,
83                           const int label_pos,
84                           const int jmp_pos)
85     __attribute__((hot, nonnull(1)));
86
87 size_t* get_code_pos_ptr_by_jmp(label_table* const __restrict self,
88                                const int label_pos,
89                                const int jmp_pos)
90     __attribute__((hot, nonnull(1)));
91
92 void label_table__destr(label_table* const __restrict self)
93     __attribute__((nonnull(1)));
94
95 inline void label_table_add(label_table* const __restrict self,
96                             const int label,
97                             const int jmp)
98 {
99     int indx = _mm_crc32_u32(label, 0xDEDED) % MIN_LABEL_TABLE_SIZE;
100    // printf("indx =%d\n", indx);
101    stack_push(&self->elems[indx], label, jmp);
102 }
103
104 inline int label_table_search_by_label(label_table* const __restrict self,
105                                       const int label_pos)
106 {
107     int indx = _mm_crc32_u32(label_pos, 0xDEDED) % MIN_LABEL_TABLE_SIZE;
108
109     if (STACK[indx].size == 0) {
110         return NOT_FOUND;
111     }
112
113     return indx;
114 };
115 };
116
117 #endif

```

## 4.2 log.h

```

1 #ifndef LOG_INCLUDED
2 #define LOG_INCLUDED
3 #include <stdio.h>
4 #include <stdarg.h>
5 #include <stdlib.h>
6 #include <execinfo.h>
7
8 #define RED "\u001b[31m"
9
10 #define FATAL_RED "\u001b[31;1m"
11 #define GREEN "\u001b[32m"
12 #define YELLOW "\u001b[33m"
13 #define BLUE "\u001b[34m"
14 #define MAGENTA "\u001b[35m"
15 #define CYAN "\u001b[36m"
16 #define END "\u001b[0m"
17
18 #define ERROR(condition, ret_val, ...)
19     if(condition) {
20         ErrorPrint(__VA_ARGS__);
21         return ret_val;
22     }

```

```

22     }
23
24 #define RET_IF(condition, ret_val)          \
25 if(condition) {                          \
26     return ret_val;                      \
27 }
28
29 void SetLogFile(FILE* log_file = nullptr);
30
31 void ResetLogFile();
32
33 void ResetAllLogFiles();
34
35 int PrintToLog(const char* format, ...);
36
37 FILE* GetCurrentLogFile();
38
39 #define ErrorPrint(...)                  \
40 ErrorPrint_(__PRETTY_FUNCTION__, __LINE__, __FILE__, __VA_ARGS__); \
41
42 #define PrettyPrint(...)                  \
43 PrettyPrint_(__PRETTY_FUNCTION__, __LINE__, __FILE__, __VA_ARGS__); \
44
45 int PrettyPrint_(const char* function, const int line, const char* file, const char* format, ...);
46
47 int ErrorPrint_(const char* function, const int line, const char* file, const char* format, ...);
48
49 #endif

```

## 4.3 src/translator.h File Reference

The header of translator, containing all used functions in JIT.

```

#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <log.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <label_table.h>

```

### Classes

- struct [opcode](#)
- union [cvt\\_u\\_int64\\_t\\_int](#)
- struct [assembly\\_code](#)

### Macros

- #define [PAGESIZE](#) 4096
- #define [HOST\\_MEMORY\\_COUNT](#) 993
- #define [TRANSLATE\\_PUSH\\_SIZE](#) 23
- #define [UNKNOWN](#) 0
- #define [MIN\\_DST\\_CODE\\_SIZE](#) 2 << 14
- #define [BYTE](#)(val) val \* 8
- #define [OPSIZE](#)(op\_code\_name) SIZEOF\_##op\_code\_name

*Some usefull defines to decrease amount of code.*

## Enumerations

- enum [WORD\\_SIZE](#) : u\_int64\_t { **DWORD** = 4 , **QWORD** = 8 , **XMMWORD** = 16 }
- enum [VCMPPD\\_COMPARISONS\\_CODE](#) : u\_int64\_t { **EQUAL** = 0 , **LESS** = 17 , **GREATER** = 30 }
- enum [REG\\_MASK](#) : u\_int64\_t { **XMM0\_EXTEND** = 0x44 , **XMM5\_BASE** = 0x2C , **XMM5\_EXTEND** = 0x6C }
- enum [TRANSLATION\\_ERROR](#) { **CVT\_ERROR** = -0xDEd , **MALLOC\_ERROR** , **FILE\_OPENING\_ERROR** , **UNKNOWN\_FILE** }
- enum [HOST\\_STACK\\_OP\\_CODES](#) {  
**PUSH** = 1 , **POP** = 2 , **IN** = 3 , **OUT** = 4 ,  
**MUL** = 5 , **ADD** = 6 , **SUB** = 7 , **DIV** = 8 ,  
**HLT** = 10 , **JB** = 11 , **CALL** = 12 , **RET** = 13 ,  
**JA** = 14 , **JMP** = 15 , **SQRT** = 16 , **JE** = 17 ,  
**POPM** = 2 | 0x70 , **POPR** = 2 | 0x60 , **POPRM** = 2 | 0x35 , **PUSHM** = 1 | 0x70 ,  
**PUSHR** = 1 | 0x60 , **PUSHRM** = 1 | 0x35 }
- enum [HOST\\_ASSEMBLY\\_REG\\_ID](#) : u\_int64\_t { **AX** = 1 , **BX** , **CX** , **DX** }
- enum [X86\\_ASSEMBLY\\_OPCODES](#) : u\_int64\_t {  
**RAX** = 0 , **RBX** = 3 , **RCX** = 1 , **RDx** = 2 ,  
**XMM0** = RAX , **XMM1** = RCX , **XMM2** = RDx , **XMM3** = RBX ,  
**XMM4** , **VMOVQ\_XMM\_RSP\_IMM** = 0x0024007EFAC5 , **VMOVQ\_RSP\_XMM** = 0x2400D6F9C5 , **SUB\_RSP\_IMM** = 0x00EC8348 ,  
**ADD\_RSP\_IMM** = 0x00C48348 , **VMOVQ\_RSP\_IMM\_XMM** = 0x002400D6F9C5 , **ADDSD\_XMM\_RSP\_IMM** = 0x002400580FF2 , **SUBSD\_XMM\_RSP\_IMM** = 0x0024005C0FF2 ,  
**MULSD\_XMM\_RSP\_IMM** = 0x002400590FF2 , **DIVSD\_XMM\_RSP\_IMM** = 0x0024005E0FF2 , **VSQRTPD\_XMM0\_XMM0** = 0xC051F9C5 , **VMOVQ\_XMM5\_R13\_B\_IMM** = 0x006D7E7AC1C4 ,  
**VMOVQ\_XMM5\_R13\_D\_IMM** = 0xAD7E7AC1C4 , **VMOVQ\_R13\_B\_IMM\_XMM5** = 0x006DD679C1C4 ,  
**VMOVQ\_R13\_D\_IMM\_XMM5** = 0xADD679C1C4 , **JMP\_REL32** = 0x00000000E9 ,  
**JE\_REL32** = 0x00000000840F , **VCMPPD\_XMM5\_XMM0\_XMM5** = 0x00EDC2F9C5 , **CMP\_R14D\_3** = 0x03FE8341 , **CMP\_R14D\_1** = 0x01FE8341 ,  
**LEA\_RDI\_RSP\_16** = 0x10247C8D48 , **MOV\_EDI\_0** = 0x00000000BF , **MOVMSKPD\_R14D\_XMM5** = 0xF5500F4466 , **MOV\_RDI\_RSP** = 0xE78948 ,  
**NATIVE\_RET** = 0xC3 , **NATIVE\_CALL** = 0x00000000E8 , **MOV\_R15\_RSP** = 0xE78949 , **MOV\_RSP\_R15** = 0xFC894C ,  
**MOV\_R14** = 0xBE49 , **MOV\_TO\_STACK\_R14** = 0x2434894C , **MOV\_R13** = 0xBD49 }
- enum [X86\\_ASSEMBLY\\_OPCODES\\_SIZE](#) {  
**sizeof\_VMOVQ\_RSP\_XMM** = 5 , **sizeof\_VMOVQ\_RSP\_IMM\_XMM** = 6 , **sizeof\_SUB\_RSP\_IMM** = 4 ,  
**sizeof\_ADD\_RSP\_IMM** = 4 ,  
**sizeof\_ADDSD\_XMM\_RSP\_IMM** = 6 , **sizeof\_MOV\_R15\_RSP** = 3 , **sizeof\_MOV\_RSP\_R15** = 3 ,  
**sizeof\_NATIVE\_CALL** = 5 ,  
**sizeof\_MOV\_R14** = 2 , **sizeof\_MOV\_TO\_STACK\_R14** = 4 , **sizeof\_VMOVQ\_XMM\_RSP\_IMM** = 6 ,  
**sizeof\_VMOVQ\_XMM5\_R13\_B\_IMM** = 6 ,  
**sizeof\_VMOVQ\_XMM5\_R13\_D\_IMM** = 5 , **sizeof\_MOV\_R13** = 2 , **sizeof\_JMP\_REL32** = 5 , **sizeof\_JE\_REL32** = 6 ,  
**sizeof\_VCMPPD\_XMM5\_XMM0\_XMM5** = 5 , **sizeof\_MOVMSKPD\_R14D\_XMM5** = 5 , **sizeof\_CMP\_R14D\_1** = 4 , **sizeof\_CMP\_R14D\_3** = 4 ,  
**sizeof\_MOV\_RDI\_RSP** = 3 , **sizeof\_MOV\_EDI\_0** = 5 , **sizeof\_VSQRTPD\_XMM0\_XMM0** = 4 ,  
**sizeof\_LEA\_RDI\_RSP\_16** = 5 ,  
**sizeof\_VMOVQ\_R13\_B\_IMM\_XMM5** = 6 , **sizeof\_VMOVQ\_R13\_D\_IMM\_XMM5** = 5 , **sizeof\_RET** = 1  
}

## Functions

- int [assembly\\_code\\_init](#) ([assembly\\_code](#) \*const \_\_restrict self, const size\_t size) \_\_attribute\_\_((nonnull(1)))  
*function to initialize [assembly\\_code](#) structure*
- int [load\\_code](#) (const char \*const \_\_restrict src\_file\_name, [assembly\\_code](#) \*const \_\_restrict src\_code\_save) \_\_attribute\_\_((nonnull(1)))  
*function to read code from file and load it to struct field of [assembly\\_code](#)*



- int void [make\\_label\\_table](#) ([assembly\\_code](#) \*const \_\_restrict src\_code, [label\\_table](#) \*const \_\_restrict table) \_\_attribute\_\_((nonnull(1)))  
*function for making so-called [label\\_table](#) - structure (hash table) contating all info about labels (such as position in source code)*
- int void int [assembly\\_code\\_aligned\\_init](#) ([assembly\\_code](#) \*const \_\_restrict self, const size\_t alignment, const size\_t size) \_\_attribute\_\_((nonnull(1)))  
*same as [assembly\\_code\\_init\(\)](#), but the memory allocated using [aligned\\_alloc\(\)](#) function*
- void [link\\_label](#) ([assembly\\_code](#) \*const \_\_restrict dst\_code, [label\\_table](#) \*const \_\_restrict table, const int search\_indx, const int label\_pos) \_\_attribute\_\_((nonnull(1)))  
*link label with appropriate jmp or call*
- void void [traslate\\_rel32\\_label](#) ([assembly\\_code](#) \*const \_\_restrict dst\_code, const size\_t jmp\_pos) \_\_attribute\_\_((nonnull(1)))  
*translate relative/non-relative jump and call instruction*
- opcode [translate\\_jmp\\_n\\_call](#) ([assembly\\_code](#) \*const \_\_restrict dst\_code, const int jmp\_n\_call\_code) \_\_attribute\_\_((nonnull(1)))  
*translate conditional and non-conditional jump and call.*
- void [execute\\_start](#) (char \*const \_\_restrict execution\_buffer, const int time\_flag) \_\_attribute\_\_((nonnull(1)))  
*start execution of translated code*
- void [translation\\_start](#) (const char \*const \_\_restrict src\_file\_name, [assembly\\_code](#) \*const \_\_restrict dst\_↵ buffer, const int time\_flag) \_\_attribute\_\_((nonnull(1)))  
*main function that start translation*
- void int void [command\\_line\\_handler](#) (int argc, char \*argv[]) \_\_attribute\_\_((nonnull(2)))  
*handle command line parameters*
- int [double\\_printf](#) (double \*value)  
*simple function, that I declared to simplify translating stdout function in my assembler language*
- int [double\\_scanf](#) (double \*value)  
*simple function, same as [double\\_printf](#), but for stdin*
- void [write\\_command](#) ([assembly\\_code](#) \*const \_\_restrict dst\_code, opcode operation\_code) \_\_attribute\_\_((always\_inline))  
*main function that write translated instructions in destination buffer*
- void **nonnull** (1)))
- void [translate\\_load\\_rsp](#) ([assembly\\_code](#) \*const \_\_restrict dst\_code) \_\_attribute\_\_((always\_inline))  
*translate the instruction "mov rsp, r15". Here in r15 the previous value of rsp is saved*
- u\_int64\_t [cvt\\_host\\_reg\\_id\\_to\\_native](#) (const int host\_reg\_id, const u\_int64\_t suffix, const u\_int64\_t offset) \_\_attribute\_\_((always\_inline))  
*translate my assembler register individual number to x86 register encoding with this expression (XMM(1-4) << offset) | suffix;*
- void [translate\\_push](#) ([assembly\\_code](#) \*const \_\_restrict dst\_code, const u\_int64\_t data) \_\_attribute\_\_((nonnull(1)))  
*translate "push x", where x - double value to x86 instruction*
- void [translate\\_push\\_r](#) ([assembly\\_code](#) \*const \_\_restrict dst\_code, const int reg\_id) \_\_attribute\_\_((nonnull(1)))  
*translate "push xx", where xx is ax, bx, cx or dx register in my assembler language*
- void [translate\\_cycle](#) ([assembly\\_code](#) \*const \_\_restrict dst\_code, [label\\_table](#) \*const \_\_restrict table, const int label\_pos, const int jmp\_n\_call\_pos, const int jmp\_n\_call\_code) \_\_attribute\_\_((nonnull(1)))  
*translate cycled jump or call (label is before jump or call)*
- void [translate\\_ahead\\_jmp\\_n\\_call](#) ([assembly\\_code](#) \*const \_\_restrict dst\_code, [label\\_table](#) \*const \_\_restrict table, const int label\_pos, const int jmp\_n\_call\_pos, const int jmp\_n\_call\_code) \_\_attribute\_\_((nonnull(1)))  
*translate jump or call that is before label. The relative address of label is setted as 0.*
- void [jmp\\_n\\_call\\_handler](#) ([assembly\\_code](#) \*const \_\_restrict dst\_code, [label\\_table](#) \*const \_\_restrict table, const int label\_pos, const int jmp\_n\_call\_pos, const int jmp\_n\_call\_code) \_\_attribute\_\_((nonnull(1)))  
*translate jump or call. This function use [translate\\_ahead\\_jmp\\_n\\_call\(\)](#), [translate\\_push\(\)](#), and [translate\\_rel32\\_label\(\)](#) functions.*
- void [translate\\_save\\_rsp](#) ([assembly\\_code](#) \*const \_\_restrict dst\_code) \_\_attribute\_\_((nonnull(1)))

- translate "mov r15, rsp" in the beginning of the buffer to save return address*

  - void `translate_stdout` (`assembly_code` \*const \_\_restrict dst\_code) \_\_attribute\_\_((nonnull(1)))

*translate "out" instruction in my assembler language. Out just print the top value in the stack.*
- void `translate_two_pop_for_cmp` (`assembly_code` \*const \_\_restrict dst\_code, const int jmp\_code) \_\_attribute\_\_((nonnull(1)))

*translate two sequential pop for comparison for conditional jump*
- void `translate_ret` (`assembly_code` \*const \_\_restrict dst\_code) \_\_attribute\_\_((nonnull(1)))

*translate "ret" instruction to leave buffer execution*
- void `translate_arithmetic_op` (`assembly_code` \*const \_\_restrict dst\_code, const int op\_id) \_\_attribute\_\_((nonnull(1)))

*translate arithmetic operations like add, sub, mul, div.*

## Variables

- `opcode` `always_inline`

### 4.3.1 Detailed Description

The header of translator, containing all used functions in JIT.

### 4.3.2 Macro Definition Documentation

#### 4.3.2.1 HOST\_MEMORY\_COUNT

```
#define HOST_MEMORY_COUNT 993
```

All memory indexes that used in host assembler

#### 4.3.2.2 MIN\_DST\_CODE\_SIZE

```
#define MIN_DST_CODE_SIZE 2 << 14
```

Size of execution buffer (currently this is fixed value)

#### 4.3.2.3 PAGESIZE

```
#define PAGESIZE 4096
```

Default linux page size. Needed to mprotect correct work

#### 4.3.2.4 TRANSLATE\_PUSH\_SIZE

```
#define TRANSLATE_PUSH_SIZE 23
```

Size of opcode of one push register, needed to correct call

#### 4.3.2.5 UNKNOWN

```
#define UNKNOWN 0
```

Define to mark unknown yet label

### 4.3.3 Enumeration Type Documentation

#### 4.3.3.1 HOST\_ASSEMBLY\_REG\_ID

```
enum HOST_ASSEMBLY_REG_ID : u_int64_t
```

My own register "opcodes" (masks to be exact)

#### 4.3.3.2 HOST\_STACK\_OP\_CODES

```
enum HOST_STACK_OP_CODES
```

Opcodes for my CPU emulator (soft-CPU)

#### 4.3.3.3 REG\_MASK

```
enum REG_MASK : u_int64_t
```

Enum containing register code in x86 Assembler

#### 4.3.3.4 TRANSLATION\_ERROR

```
enum TRANSLATION_ERROR
```

Enum for error handling

#### 4.3.3.5 VCMPPD\_COMPARISONS\_CODE

```
enum VCMPPD_COMPARISONS_CODE : u_int64_t
```

Enum containing comparison "code" for VCMPPD instruction

#### 4.3.3.6 WORD\_SIZE

```
enum WORD_SIZE : u_int64_t
```

Enum containing word sizes

#### 4.3.3.7 X86\_ASSEMBLY\_OPCODES

```
enum X86_ASSEMBLY_OPCODES : u_int64_t
```

Enum containing all x86 opcodes

#### 4.3.3.8 X86\_ASSEMBLY\_OPCODES\_SIZE

```
enum X86_ASSEMBLY_OPCODES_SIZE
```

Enum containing the size (in bytes) of each x86 opcodes, used by program

### 4.3.4 Function Documentation

#### 4.3.4.1 assembly\_code\_aligned\_init()

```
int void int assembly_code_aligned_init (
    assembly_code *const __restrict self,
    const size_t alignment,
    const size_t size )
```

same as [assembly\\_code\\_init\(\)](#), but the memory allocated using `aligned_alloc()` function

##### Parameters

<i>self</i>	object to initialize
<i>alignment</i>	alignment of allocated memory
<i>size</i>	size of allocated memory for structure field <code>self-&gt;buffer</code>

##### Returns

return `MALLOC_ERROR` (see `TRANSLATION_ERROR`) if allocation is failed. Return 0 if succeed.

##### See also

[TRANSLATION\\_ERROR](#)

#### 4.3.4.2 assembly\_code\_init()

```
int assembly_code_init (
    assembly_code *const __restrict self,
    const size_t size )
```

function to initialize [assembly\\_code](#) structure

## Parameters

<i>self</i>	object to initialize
<i>size</i>	size of allocated memory for structure field <i>self</i> ->buffer

## Returns

return MALLOC\_ERROR (see TRANSLATION\_ERROR) if allocation is failed. Return 0 if succeed.

## See also

[TRANSLATION\\_ERROR](#)

## 4.3.4.3 command\_line\_handler()

```
void int void command_line_handler (
    int argc,
    char * argv[] )
```

handle command line parameters

## Parameters

<i>argc</i>	command line argument count
<i>argv</i>	array of strings with this arguments

## 4.3.4.4 cvt\_host\_reg\_id\_to\_native()

```
u_int64_t cvt_host_reg_id_to_native (
    const int host_reg_id,
    const u_int64_t suffix,
    const u_int64_t offset ) [inline]
```

translate my assembler register individual number to x86 register encoding with this expression (XMM(1-4) << offset) | suffix;

## Parameters

<i>host_reg_id</i>	my assembler register individual number
<i>suffix</i>	suffix needed in above expression
<i>offset</i>	offset needed in above expression

**Returns**

x86 register encoding

**4.3.4.5 double\_printf()**

```
int double_printf (
    double * value )
```

simple function, that I declared to simplify translating stdout function in my assembler language

**Parameters**

<i>pointer</i>	to value, that will be printed
----------------	--------------------------------

**Returns**

as a standart printf

**4.3.4.6 double\_scanf()**

```
int double_scanf (
    double * value )
```

simple function, same as double\_printf, but for stdin

**Parameters**

<i>pointer</i>	to value, where the inputted value will be stored
----------------	---

**Returns**

as a standart scanf

**4.3.4.7 execute\_start()**

```
void execute_start (
    char *const __restrict execution_buffer,
    const int time_flag )
```

start execution of translated code

## Parameters

<i>execution_buffer</i>	buffer, where translated code is stored
<i>time_flag</i>	flag that flag indicating whether to measure the execution time

## 4.3.4.8 jmp\_n\_call\_handler()

```
void jmp_n_call_handler (
    assembly_code *const __restrict dst_code,
    label_table *const __restrict table,
    const int label_pos,
    const int jmp_n_call_pos,
    const int jmp_n_call_code ) [inline]
```

translate jump or call. This function use [translate\\_ahead\\_jmp\\_n\\_call\(\)](#), [translate\\_push\(\)](#), and [translate\\_rel32\\_label\(\)](#) functions.

## Parameters

<i>dst_code</i>	buffer, where translated code will be written
<i>table</i>	<a href="#">label_table</a> is used here to save current jump with appropriate label
<i>label_pos</i>	label position in source code. This value is used to search in <a href="#">label_table</a> to "link" current jump with his label.
<i>jmp_n_call_pos</i>	jump or call position in source code. This value is also used for search
<i>jmp_n_call_code</i>	indefify which instruction translate (unconditional jump, conditional jump,

## 4.3.4.9 link\_label()

```
void link_label (
    assembly_code *const __restrict dst_code,
    label_table *const __restrict table,
    const int search_indx,
    const int label_pos )
```

link label with appropriate jmp or call

## Parameters

<i>dst_code</i>	buffer, where translated code will be injected
<i>table</i>	<a href="#">label_table</a> structure, and for table the <a href="#">make_label_table()</a> function must be called
<i>search_indx</i>	index in table->elem array. When the <a href="#">make_label_table()</a> function was called, labels were added to the <a href="#">label_table</a> structure. In the future, when translating the code, the hash table is searched using the <a href="#">label_table_search_by_label()</a> function, that is, in fact, we check whether the position of the code that we are translating is a label or not. This index is responsible for the position of the label in the <a href="#">label_table</a> structure.
<i>label_pos</i>	label position (offset in source buffer) in source buffer

#### 4.3.4.10 load\_code()

```
int load_code (
    const char *const __restrict src_file_name,
    assembly_code *const __restrict src_code_save )
```

function to read code from file and load it to struct field of [assembly\\_code](#)

load source code to buffer

##### Parameters

<i>src_file_name</i>	name of source file to read
<i>src_code_save</i>	object of structure <a href="#">assembly_code</a> to save source code

##### Returns

FILE\_OPENING\_ERROR - if cannot open the file. MALLOC\_ERROR - if memory allocation error detected, else return 0.

##### Parameters

<i>src_file_name</i>	name of source file to translate
<i>src_code_save</i>	object of <a href="#">assembly_code</a> structure, where the source code will be saved

##### Returns

FILE\_OPENING\_ERROR - if file opening error happened. MALLOC\_ERROR - if memory allocation error happened, else return 0. For more info see TRANSLATION\_ERROR.

##### See also

[TRANSLATION\\_ERROR](#)

#### 4.3.4.11 make\_label\_table()

```
int void make_label_table (
    assembly_code *const __restrict src_code,
    label_table *const __restrict table )
```

function for making so-called [label\\_table](#) - structure (hash table) containing all info about labels (such as position in source code)



## Parameters

<i>src_code</i>	source code where labels will be looked up
<i>table</i>	object of <a href="#">label_table</a> to save info about labels

**4.3.4.12 translate\_ahead\_jump\_n\_call()**

```
void translate_ahead_jump_n_call (
    assembly_code *const __restrict dst_code,
    label_table *const __restrict table,
    const int label_pos,
    const int jmp_n_call_pos,
    const int jmp_n_call_code ) [inline]
```

translate jump or call that is before label. The relative address of label is setted as 0.

## Parameters

<i>dst_code</i>	buffer, where translated code will be written
<i>table</i>	<a href="#">label_table</a> is used here to save current jump with appropriate label
<i>label_pos</i>	label position in source code. This value is used to search in <a href="#">label_table</a> to "link" current jump with his label
<i>jmp_n_call_pos</i>	jump or call position in source code. This value is also used for search
<i>jmp_n_call_code</i>	indefify which instruction translate (unconditional jump, conditional jump, call)

**4.3.4.13 translate\_arithmetic\_op()**

```
void translate_arithmetic_op (
    assembly_code *const __restrict dst_code,
    const int op_id ) [inline]
```

translate arithmetic operations like add, sub, mul, div.

## Parameters

<i>dst_code</i>	buffer, where translated code will be written
<i>op_id</i>	indeficator of which operator translate (sub, add, mul or div)

**4.3.4.14 translate\_cycle()**

```
void translate_cycle (
    assembly_code *const __restrict dst_code,
```

```

label_table *const __restrict table,
const int label_pos,
const int jmp_n_call_pos,
const int jmp_n_call_code ) [inline]

```

translate cycled jump or call (label is before jump or call)

#### Parameters

<i>dst_code</i>	buffer, where translated code will be written
<i>table</i>	label table structure. Need to find labels position by knowing jump position (in this case it is possible because the label is before jump)
<i>label_pos</i>	in source assembler code. This value will be used to find the appropriate to label to current jump or call.
<i>jmp_n_call_pos</i>	position of jump or call in source assembler code. This value also used to find the appropriate to this jump label
<i>jmp_n_call_code</i>	indefify which instruction translate (unconditional jump, conditional jump, call)

#### 4.3.4.15 translate\_jmp\_n\_call()

```

opcode translate_jmp_n_call (
    assembly_code *const __restrict dst_code,
    const int jmp_n_call_code )

```

translate conditional and non-conditional jump and call.

#### Parameters

<i>dst_code</i>	buffer, where translated code will be injected
<i>jmp_n_call_code</i>	a number indicating whether the instruction is conditional or non-conditional jump or call

#### Returns

opcode structure, that contain translated code and size of this chunk of code

#### 4.3.4.16 translate\_load\_rsp()

```

void translate_load_rsp (
    assembly_code *const __restrict dst_code ) [inline]

```

translate the instruction "mov rsp, r15". Here in r15 the previous value of rsp is saved

#### Parameters

<i>dst_code</i>	buffer, where translated code will be written
-----------------	---

#### 4.3.4.17 translate\_push()

```
void translate_push (
    assembly_code *const __restrict dst_code,
    const u_int64_t data ) [inline]
```

translate "push x", where x - double value to x86 instruction

##### Parameters

<i>dst_code</i>	buffer, where translated code will be written
<i>data</i>	converted to u_int64_t double value

#### 4.3.4.18 translate\_push\_r()

```
void translate_push_r (
    assembly_code *const __restrict dst_code,
    const int reg_id ) [inline]
```

translate "push xx", where xx is ax, bx, cx or dx register in my assembler language

##### Parameters

<i>dst_code</i>	buffer, where translated code will be written
<i>reg_id</i>	my assembler language register indicator

#### 4.3.4.19 translate\_ret()

```
void translate_ret (
    assembly_code *const __restrict dst_code ) [inline]
```

translate "ret" instruction to leave buffer execution

##### Parameters

<i>dst_code</i>	buffer, where translated code will be written
-----------------	---

#### 4.3.4.20 translate\_save\_rsp()

```
void translate_save_rsp (
    assembly_code *const __restrict dst_code ) [inline]
```

translate "mov r15, rsp" in the beginning of the buffer to save return address

##### Parameters

<i>dst_code</i>	buffer, where translated code will be written
-----------------	---

#### 4.3.4.21 translate\_stdout()

```
void translate_stdout (
    assembly_code *const __restrict dst_code ) [inline]
```

translate "out" instruction in my assembler language. Out just print the top value in the stack.

##### Parameters

<i>dst_code</i>	buffer, where translated code will be written
-----------------	---

#### 4.3.4.22 translate\_two\_pop\_for\_cmp()

```
void translate_two_pop_for_cmp (
    assembly_code *const __restrict dst_code,
    const int jmp_code ) [inline]
```

translate two sequential pop for comparison for conditional jump

##### Parameters

<i>dst_code</i>	buffer, where translated code will be written
<i>jmp_code</i>	indefify which instruction translate (unconditional jump, conditional jump,

#### 4.3.4.23 translation\_start()

```
void translation_start (
    const char *const __restrict src_file_name,
    assembly_code *const __restrict dst_buffer,
    const int time_flag )
```

main function that start translation

## Parameters

<i>src_file_name</i>	name of source file to translate
<i>dst_code</i>	buffer, where translated code will be injected
<i>time_flag</i>	flag that flag indicating whether to measure the translation time

## 4.3.4.24 traslate\_rel32\_label()

```
void void traslate_rel32_label (
    assembly_code *const __restrict dst_code,
    const size_t jmp_pos )
```

translate relative/non-relative jump and call instruction

## Parameters

<i>dst_code</i>	buffer, where translated code will be injected
<i>jmp_pos</i>	jmp or call position in destination buffer. When processing labels, it may be that jmp (call, conditional jmp) comes first and then the label. In this case, jmp (call) is translated in advance, and when the label is reached, this function is called, and the already translated jmp (call) receives the address of this label.

## 4.3.4.25 write\_command()

```
void write_command (
    assembly_code *const __restrict dst_code,
    opcode operation_code ) [inline]
```

main function that write translated instructions in destination buffer

## Parameters

<i>dst_code</i>	buffer, where translated code will be written
<i>operation_code</i>	object of opcode structure. Define the command, that will be written

## 4.4 translator.h

[Go to the documentation of this file.](#)

```
1
6 #include <stdlib.h>
7 #include <string.h>
8 #include <stdio.h>
9 #include <log.h>
10 #include <sys/types.h>
11 #include <sys/mman.h>
```

```

12 #include <label_table.h>
13
14 #define PAGESIZE 4096
15 #define HOST_MEMORY_COUNT 993
16 #define TRANSLATE_PUSH_SIZE 23
17 #define UNKNOWN 0
18 #define MIN_DST_CODE_SIZE 2 « 14
24 #define BYTE(val) val * 8
25 #define OPSIZE(op_code_name) SIZEOF_##op_code_name
26
27
28 //+=====| WORD SIZES |=====+
29
32 enum WORD_SIZE : u_int64_t {
33     DWORD = 4,
34     QWORD = 8,
35     XMMWORD = 16,
36 };
37
38 //-----
39
40 //+=====| COMPARISONS |=====+
41
44 enum VCMPPD_COMPARISONS_CODE : u_int64_t {
45     EQUAL = 0,
46     LESS = 17,
47     GREATER = 30
48 };
49
50 //-----
51
52
53 //+=====| MASK FOR XMM REGISTER |=====+
54
57 enum REG_MASK : u_int64_t {
58     XMM0_EXTEND = 0x44,
59     XMM5_BASE = 0x2C,
60     XMM5_EXTEND = 0x6C
61 };
62
63 //-----
64
65
66 //+=====| TRANSLATION ERRORS |=====+
67
70 enum TRANSLATION_ERROR {
71     CVT_ERROR = -0xDEAD,
72     MALLOC_ERROR,
73     FILE_OPENING_ERROR,
74     UNKNOWN_FILE
75 };
76
77
78 //+=====| HOST ASSEMBLER OPCODES |=====+
79
82 enum HOST_STACK_OP_CODES { // C++11
83     PUSH = 1,
84     POP = 2,
85     IN = 3,
86     OUT = 4,
87     MUL = 5,
88     ADD = 6,
89     SUB = 7,
90     DIV = 8,
91     HLT = 10,
92     JB = 11,
93     CALL = 12,
94     RET = 13,
95     JA = 14,
96     JMP = 15,
97     SQRT = 16,
98     JE = 17,
99     POPM = 2 | 0x70,
100     POPR = 2 | 0x60,
101     POPRM = 2 | 0x35, // Pop to memory with register index
102     PUSHM = 1 | 0x70,
103     PUSHR = 1 | 0x60,
104     PUSHRM = 1 | 0x35
105 };
106 };
107
110 enum HOST_ASSEMBLY_REG_ID : u_int64_t {
111     AX = 1,
112     BX,
113     CX,
114     DX
115 };

```

```

116
117
118 //-----
119
120 //+=====| NATIVE ASSEMBLER |=====+
121
124 enum X86_ASSEMBLY_OPCODES : u_int64_t {
125
126 //+=====| REG_IDS |=====+
127 //
128 RAX = 0,
129 RBX = 3, //
130 RCX = 1,
131 RDX = 2,
132
133 XMM0 = RAX,
134 XMM1 = RCX,
135 XMM2 = RDX,
136 XMM3 = RBX,
137 XMM4,
138
139 //-----
140
141 //+=====| OP_CODES |=====+
142
143 // NOT THAT ALL OP_CODES REVERSED BECAUSE OF LITTLE ENDIAN
144 /*
145
146 //-----+
147 // .-----v v v
148 " . vmovq xmm(0-5), [rsp - 8] | XMM Encoding
149 || | +-----^ | xmm0 = 0x44 = 01000100b
150 ^ ^ ^ | xmm1 = 0x4C = 01010100b ... */
151 VMOVQ_XMM_RSP_IMM = 0x0024007EFAC5, //
152 /*
153 +-----+
154 / +-----+ | XMM Encoding:
155 . / | | | | xmm0 = 0x4
156 | . V V | | xmm1 = 0xC ..
157 | | vmovq [rsp], xmm(0-5)
158 ^ ^
159 VMOVQ_RSP_XMM = 0x2400D6F9C5, //
160 /*
161 +-----+
162 / |
163 . V
164 | sub rsp, imm
165 ^
166 SUB_RSP_IMM = 0x00EC8348, //
167 ADD_RSP_IMM = 0x00C48348, //
168 /*
169 .-----+ +-----< XMM Encoding: see +++
170 / +-----+ | |
171 | / V V V |
172 | | vmovq [rsp + 00], xmm(0-4)
173 | | .-----^
174 V V V
175 VMOVQ_RSP_IMM_XMM = 0x002400D6F9C5, //
176 /*
177 ^ ^ ^
178 | | |
179 vmovq code
180
181 .-----+ +-----< Encoding: See ++
182 / .-----+ | |
183 . / V V V |
184 | | addsd [rsp + 00], xmm(0-4)
185 | | .-----^
186 | | /
187 ADDSD_XMM_RSP_IMM = 0x002400580FF2,
188 /*
189 ^ ^ ^
190 | | |
191 addsd
192 SUBSD_XMM_RSP_IMM = 0x0024005C0FF2,
193 MULSD_XMM_RSP_IMM = 0x002400590FF2,
194 DIVSD_XMM_RSP_IMM = 0x0024005E0FF2,
195
196 /*
197 +-----+
198 / / / / |
199 . . . . +-----+
200 | | | | V V V V V V V V
201 ^ ^ ^ ^ vsqrtpd xmm0, xmm0
202 VSQRTPD_XMM0_XMM0 = 0xC051F9C5,
203 /*
204 Used to memory reference
205
206 +-----+
207 / |
208 . V
209

```

```

205             |      vmovq xmm5, [r13 + imm]
206             ^
207 VMOVQ_XMM5_R13_B_IMM = 0x006D7E7AC1C4,
208 /*
209     Identical, but the imm is double word (4 bytes)
210 */
211 VMOVQ_XMM5_R13_D_IMM = 0xAD7E7AC1C4,
212
213 VMOVQ_R13_B_IMM_XMM5 = 0x006DD679C1C4,
214
215 /* !NOTE!: SIZEOF VMOVQ_R13_D_IMM_XMM is 9 bytes,
216    so i have to split it, and cannot use the last byte.
217    It is suit for my case
218 */
219
220 VMOVQ_R13_D_IMM_XMM5 = 0xADD679C1C4,
221 /* Relative jump
222    +-----+
223    / / / / |
224    . . . . V
225    | | | | jmp rel32
226    ^ ^ ^ ^
227 */
227 JMP_REL32 = 0x00000000E9,
228 JE_REL32 = 0x00000000840F,
229 /*
230    +-----+
231    / |
232    . V
233    | vmppd xmm5, xmm0, xmm5, 0 - Comparison mod
234    ^
235 */
234 VCMPPD_XMM5_XMM0_XMM5 = 0x00EDC2F9C5,
235 /*
236    +-----+
237    / / / / |
238    . . . . +-----+
239    | | | | | | | |
240    | | | | V V V V V V - Note that r14d contain the mask, and the mask 3
241    | | | | cmp r14d, 3 mean that for first double and second double in xmm
242    | | | | register the comparison condition is true
243    | | | | Because we don't use the high half of xmm register
244    | | | | The second bit, ,will be always set. So this is Why we
245 use ^ ^ ^ ^ number 3, not 1
246 CMP_R14D_3 = 0x03FE8341,
247 CMP_R14D_1 = 0x01FE8341,
248
249
250 LEA_RDI_RSP_16 = 0x10247C8D48,
251 /*
252    +-----+
253    / / / / |
254    . . . . V
255    | | | | mov edi, 0
256    ^ ^ ^ ^
257 */
257 MOV_EDI_0 = 0x00000000BF,
258
259 /*
260    +-----+
261    / / / / |
262    . . . . +-----+
263    | | | | | | | | v V v v v v v v v v
264    ^ ^ ^ ^ ^ ^ ^ ^ movmskpd r14d, xmm5
265 */
265 MOVMSKPD_R14D_XMM5 = 0xF5500F4466,
266 /*
267     To call scanf
268    +-----+
269    / / / |
270    . . . +-----+
271    | | | V V V V V V
272    ^ ^ ^ mov rdi, rsp
273 */
273 MOV_RDI_RSP = 0xE78948,
274
275 NATIVE_RET = 0xC3,
276
277 /*
278    .-.-.-----< Rel 32 immediate address
279    | | | |
280    v v v v
281 */
280 NATIVE_CALL = 0x00000000E8,
281
282 MOV_R15_RSP = 0xE78949, // Full opcode because instruction is used once
283 MOV_RSP_R15 = 0xFC894C,
284 MOV_R14 = 0xBE49,
285 MOV_TO_STACK_R14 = 0x2434894C,
286 MOV_R13 = 0xBD49,
287
288 //-----
289 };
290

```



```

291
292 enum X86_ASSEMBLY_OPCODES_SIZE {
293     sizeof_VMOVQ_RSP_XMM      = 5,
294     sizeof_VMOVQ_RSP_IMM_XMM  = 6,
295     sizeof_SUB_RSP_IMM        = 4,
296     sizeof_ADD_RSP_IMM        = 4,
297     sizeof_ADDSD_XMM_RSP_IMM  = 6,
298     sizeof_MOVR15_RSP         = 3,
299     sizeof_MOVR15_RSP_R15     = 3,
300     sizeof_NATIVE_CALL        = 5,
301     sizeof_MOVR14             = 2,
302     sizeof_MOV_TO_STACK_R14    = 4,
303     sizeof_VMOVQ_XMM_RSP_IMM  = 6,
304     sizeof_VMOVQ_XMM5_R13_B_IMM = 6,
305     sizeof_VMOVQ_XMM5_R13_D_IMM = 5,
306     sizeof_MOVR13             = 2,
307     sizeof_JMP_REL32          = 5,
308     sizeof_JE_REL32           = 6,
309     sizeof_VCMPPD_XMM5_XMM0_XMM5 = 5,
310     sizeof_MOVMASKPD_R14D_XMM5 = 5,
311     sizeof_CMP_R14D_1         = 4,
312     sizeof_CMP_R14D_3         = 4,
313     sizeof_MOVRDI_RSP         = 3,
314     sizeof_MOVEDI_0           = 5,
315     sizeof_VSQRTPD_XMM0_XMM0  = 4,
316     sizeof_LEA_RDI_RSP_16     = 5,
317     sizeof_VMOVQ_R13_B_IMM_XMM5 = 6,
318     sizeof_VMOVQ_R13_D_IMM_XMM5 = 5,
319     sizeof_RET                 = 1
320 };
321
322 //-----
323 struct opcode
324 {
325     u_int64_t code;
326     int size;
327 };
328
329 union cvt_u_int64_t_int {
330     int rel_addr;
331     u_int64_t extended_address;
332 };
333
334 struct assembly_code
335 {
336     char* code;
337     int position;
338     size_t size;
339 };
340
341 int assembly_code_init(assembly_code* const __restrict self,
342     const size_t size)
343     __attribute__((nonnull(1)));
344
345 int load_code(const char* const __restrict src_file_name,
346     assembly_code* const __restrict src_code_save)
347     __attribute__((nonnull(1,2)));
348
349 //+=====| FUNCTIONS DECLARATIONS |=====+
350
351 void make_label_table(assembly_code* const __restrict src_code,
352     label_table* const __restrict table)
353     __attribute__((nonnull(1,2)));
354
355 int assembly_code_aligned_init(assembly_code* const __restrict self,
356     const size_t alignment,
357     const size_t size)
358     __attribute__((nonnull(1)));
359
360 void link_label(assembly_code* const __restrict dst_code,
361     label_table* const __restrict table,
362     const int search_idx,
363     const int label_pos)
364     __attribute__((nonnull(1,2)));
365
366 void traslate_rel32_label(assembly_code* const __restrict dst_code,
367     const size_t jmp_pos)
368     __attribute__((nonnull(1)));
369
370 opcode translate_jmp_n_call(assembly_code* const __restrict dst_code,
371     const int jmp_n_call_code)

```

```

434     __attribute__((nonnull(1), always_inline));
435
436
441 void execute_start(char* const __restrict execution_buffer,
442                   const int time_flag)
443     __attribute__((nonnull(1)));
444
451 void translation_start(const char *const __restrict src_file_name,
452                      assembly_code *const __restrict dst_buffer,
453                      const int time_flag)
454     __attribute__((nonnull(1,2)));
455
464 int load_code(const char *const __restrict src_file_name,
465              assembly_code *const __restrict src_code_save)
466     __attribute__((nonnull(1,2)));
467
473 void command_line_handler(int argc, char* argv[])
474     __attribute__((nonnull(2)));
475
481 extern "C" int double_printf(double* value);
482
487 extern "C" int double_scanf (double* value);
488
489
490 //+=====| INLINE FUNCTIONS DECLARATIONS |=====+
491
492
497 inline void write_command(assembly_code* const __restrict dst_code,
498                          opcode operation_code)
499     __attribute__((always_inline, nonnull(1)));
500
501
506 inline void translate_load_rsp(assembly_code* const __restrict dst_code)
507     __attribute__((always_inline, nonnull(1)));
508
509
517 inline u_int64_t cvt_host_reg_id_to_native(const int host_reg_id,
518                                             const u_int64_t suffix,
519                                             const u_int64_t offset)
520     __attribute__((always_inline));
521
527 inline void translate_push(assembly_code* const __restrict dst_code,
528                           const u_int64_t data)
529     __attribute__((nonnull(1), always_inline));
530
535 inline void translate_push_r(assembly_code* const __restrict dst_code,
536                             const int reg_id)
537     __attribute__((nonnull(1), always_inline));
538
551 inline void translate_cycle(assembly_code* const __restrict dst_code,
552                             label_table* const __restrict table,
553                             const int label_pos,
554                             const int jmp_n_call_pos,
555                             const int jmp_n_call_code)
556     __attribute__((nonnull(1,2), always_inline));
557
568 inline void translate_ahead_jmp_n_call(assembly_code* const __restrict dst_code,
569                                       label_table* const __restrict table,
570                                       const int label_pos,
571                                       const int jmp_n_call_pos,
572                                       const int jmp_n_call_code)
573     __attribute__((nonnull(1,2), always_inline));
574
585 inline void jmp_n_call_handler(assembly_code* const __restrict dst_code,
586                               label_table* const __restrict table,
587                               const int label_pos,
588                               const int jmp_n_call_pos,
589                               const int jmp_n_call_code)
590     __attribute__((nonnull(1,2), always_inline));
591
596 inline void translate_save_rsp(assembly_code* const __restrict dst_code)
597     __attribute__((nonnull(1), always_inline));
598
603 inline void translate_stdout(assembly_code* const __restrict dst_code)
604     __attribute__((nonnull(1), always_inline));
605
611 inline void translate_two_pop_for_cmp(assembly_code* const __restrict dst_code,
612                                       const int jmp_code)
613     __attribute__((nonnull(1), always_inline));
614
615
620 inline void translate_ret(assembly_code* const __restrict dst_code)
621     __attribute__((nonnull(1), always_inline));
622
623
629 inline void translate_arithmetic_op(assembly_code* const __restrict dst_code,
630                                     const int op_id)

```

```
631     __attribute__((nonnull(1), always_inline));  
632  
633 //-----  
634  
635
```



# Index

assembly\_code, [5](#)  
    code, [5](#)  
    position, [5](#)  
    size, [5](#)  
assembly\_code\_aligned\_init  
    translator.h, [16](#)  
assembly\_code\_init  
    translator.h, [16](#)  
  
code  
    assembly\_code, [5](#)  
    opcode, [7](#)  
command\_line\_handler  
    translator.h, [17](#)  
cvt\_host\_reg\_id\_to\_native  
    translator.h, [17](#)  
cvt\_u\_int64\_t\_int, [6](#)  
    extended\_address, [6](#)  
    rel\_addr, [6](#)  
  
double\_printf  
    translator.h, [18](#)  
double\_scanf  
    translator.h, [18](#)  
  
execute\_start  
    translator.h, [18](#)  
extended\_address  
    cvt\_u\_int64\_t\_int, [6](#)  
  
HOST\_ASSEMBLY\_REG\_ID  
    translator.h, [15](#)  
HOST\_MEMORY\_COUNT  
    translator.h, [14](#)  
HOST\_STACK\_OP\_CODES  
    translator.h, [15](#)  
  
jmp\_n\_call\_handler  
    translator.h, [19](#)  
  
label\_table, [7](#)  
lib/label\_table.h, [9](#)  
lib/log.h, [10](#)  
link\_label  
    translator.h, [19](#)  
load\_code  
    translator.h, [20](#)  
  
make\_label\_table  
    translator.h, [20](#)  
MIN\_DST\_CODE\_SIZE  
    translator.h, [14](#)  
  
opcode, [7](#)  
    code, [7](#)  
    size, [7](#)  
  
PAGESIZE  
    translator.h, [14](#)  
position  
    assembly\_code, [5](#)  
  
REG\_MASK  
    translator.h, [15](#)  
rel\_addr  
    cvt\_u\_int64\_t\_int, [6](#)  
  
size  
    assembly\_code, [5](#)  
    opcode, [7](#)  
src/translator.h, [11](#), [25](#)  
stack, [8](#)  
stack\_node, [8](#)  
  
translate\_ahead\_jmp\_n\_call  
    translator.h, [21](#)  
translate\_arithmetic\_op  
    translator.h, [21](#)  
translate\_cycle  
    translator.h, [21](#)  
translate\_jmp\_n\_call  
    translator.h, [22](#)  
translate\_load\_rsp  
    translator.h, [22](#)  
translate\_push  
    translator.h, [23](#)  
translate\_push\_r  
    translator.h, [23](#)  
TRANSLATE\_PUSH\_SIZE  
    translator.h, [14](#)  
translate\_ret  
    translator.h, [23](#)  
translate\_save\_rsp  
    translator.h, [23](#)  
translate\_stdout  
    translator.h, [24](#)  
translate\_two\_pop\_for\_cmp  
    translator.h, [24](#)  
TRANSLATION\_ERROR  
    translator.h, [15](#)  
translation\_start  
    translator.h, [24](#)

translator.h  
   assembly\_code\_aligned\_init, 16  
   assembly\_code\_init, 16  
   command\_line\_handler, 17  
   cvt\_host\_reg\_id\_to\_native, 17  
   double\_printf, 18  
   double\_scanf, 18  
   execute\_start, 18  
   HOST\_ASSEMBLY\_REG\_ID, 15  
   HOST\_MEMORY\_COUNT, 14  
   HOST\_STACK\_OP\_CODES, 15  
   jmp\_n\_call\_handler, 19  
   link\_label, 19  
   load\_code, 20  
   make\_label\_table, 20  
   MIN\_DST\_CODE\_SIZE, 14  
   PAGESIZE, 14  
   REG\_MASK, 15  
   translate\_ahead\_jmp\_n\_call, 21  
   translate\_arithmetic\_op, 21  
   translate\_cycle, 21  
   translate\_jmp\_n\_call, 22  
   translate\_load\_rsp, 22  
   translate\_push, 23  
   translate\_push\_r, 23  
   TRANSLATE\_PUSH\_SIZE, 14  
   translate\_ret, 23  
   translate\_save\_rsp, 23  
   translate\_stdout, 24  
   translate\_two\_pop\_for\_cmp, 24  
   TRANSLATION\_ERROR, 15  
   translation\_start, 24  
   traslate\_rel32\_label, 25  
   UNKNOWN, 14  
   VCMPPD\_COMPARISONS\_CODE, 15  
   WORD\_SIZE, 15  
   write\_command, 25  
   X86\_ASSEMBLY\_OPCODES, 15  
   X86\_ASSEMBLY\_OPCODES\_SIZE, 16  
 traslate\_rel32\_label  
   translator.h, 25  
  
 UNKNOWN  
   translator.h, 14  
  
 VCMPPD\_COMPARISONS\_CODE  
   translator.h, 15  
  
 WORD\_SIZE  
   translator.h, 15  
 write\_command  
   translator.h, 25  
  
 X86\_ASSEMBLY\_OPCODES  
   translator.h, 15  
 X86\_ASSEMBLY\_OPCODES\_SIZE  
   translator.h, 16