JIT translator

Generated by Doxygen 1.9.4

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

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

```
#if !defined LABEL_TABLE_INCLUDED

#include <cstdint>
#include <stdlib.h>
#include <immintrin.h>
#define LABEL_TABLE_INCLUDED

#define HT_ERROR -1
#define NOT_FOUND -1

#define CYCLE 1
// 128 * 16 * 32 * 16
// TODO: Rework for dynamic value of size
#define MIN_STACK_SIZE       16
#define MIN_LABEL_TABLE_SIZE 128

#define STACK self->elems


struct stack_node
{
        int    label; //
        int    jmp;
        size_t code_pos;
};

struct stack
{
        int             size;
        int             capacity;
        stack_node*     data;
};

struct label_table // Hash table for immediate value
{
        int     size;
        stack*  elems;
};


//+====================| STACK |============================+


void stack_init(stack* const __restrict stack, const size_t size)
       __attribute__((nonnull(1)));


int stack_push(stack* const __restrict stack,
               const int               key,
               const int               data)
        __attribute__((nonnull(1)));

int stack_destr(stack* const __restrict stack)
        __attribute__((nonnull(1)));
//-========================================================-


int label_table_init(label_table* const __restrict self)
```

```
       __attribute__((nonnull(1)));

void label_table_manual_destr(label_table* const __restrict self)
       __attribute__((nonnull(1)));


void label_table_add(label_table* const __restrict  self,
                     const int                      key,
                     const int                      data)
        __attribute__((always_inline, nonnull(1)));


void set_all_cycles(label_table* self,
                    const int    indx,
                    const size_t code_pos,
                    const int    label_pos)
       __attribute__((nonnull(1)));

inline int label_table_search_by_label(label_table* const __restrict self,
                                       const int                     label_pos)
      __attribute__((always_inline, hot));


size_t get_code_pos_by_jmp(label_table* const __restrict self,
                           const int                     label_pos,
                           const int                     jmp_pos)
       __attribute__((hot, nonnull(1)));

size_t* get_code_pos_ptr_by_jmp(label_table* const __restrict self,
                                const int                     label_pos,
                                const int                     jmp_pos)
       __attribute__((hot, nonnull(1)));

void label_table__destr(label_table* const __restrict self)
        __attribute__((nonnull(1)));

inline void label_table_add(label_table* const __restrict  self,
                            const int                      label,
                            const int                      jmp)
{
        int indx = _mm_crc32_u32(label, 0xDED) % MIN_LABEL_TABLE_SIZE;
         //         printf("indx =%d\n", indx);
         stack_push(&self->elems[indx], label, jmp);
}

inline int label_table_search_by_label(label_table* const __restrict self,
                                       const int                     label_pos)
{
         int indx = _mm_crc32_u32(label_pos, 0xDED) % MIN_LABEL_TABLE_SIZE;

         if (STACK[indx].size == 0) {
                 return NOT_FOUND;
         }

         return indx;

};

#endif
```

4.2 log.h

```
#ifndef LOG_INCLUDED
#define LOG_INCLUDED
#include <stdio.h>
#include <stdarg.h>
#include <stdlib.h>
#include <execinfo.h>

#define RED "\u001b[31m"

#define FATAL_RED "\u001b[31;1m"
#define GREEN "\u001b[32m"
#define YELLOW "\u001b[33m"
#define BLUE "\u001b[34m"
#define MAGENTA "\u001b[35m"
#define CYAN    "\u001b[36m"
#define END  "\u001b[0m"

#define ERROR(condition, ret_val,...)                       \
    if(condition) {                                         \
        ErrorPrint(__VA_ARGS__);                            \
        return ret_val;                                     \
    }

#define RET_IF(condition, ret_val)                    \
if(condition) {                                       \
    return ret_val;                                   \
}

void  SetLogFile(FILE* log_file = nullptr);

void  ResetLogFile();

void  ResetAllLogFiles();

int   PrintToLog(const char* format, ...);

FILE* GetCurrentLogFile();

#define ErrorPrint(...)                                                 \
ErrorPrint_(__PRETTY_FUNCTION__, __LINE__, __FILE__, __VA_ARGS__);

#define PrettyPrint(...)                                                  \
PrettyPrint_(__PRETTY_FUNCTION__, __LINE__, __FILE__, __VA_ARGS__);

int PrettyPrint_(const char* function, const int line, const char* file, const char* format, ...);

int ErrorPrint_(const char* function, const int line, const char* file, const char* format, ...);

#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** = 0x↩ F5500F4466 , **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 begining 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 contating 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

self	object to initialize
alignment	alignment of allocated memory
size	size of alocated memory for structure field self->buffer

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

self	object to initialize
size	size of alocated memory for structure field self->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

argc	command line argument count
argv	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

host_reg↩_id	my assembler register individual number
suffix	suffix needed in above expression
offset	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

pointer	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

pointer	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

execution_buffer	buffer, whre translated code is stored
time_flag	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

dst_code	buffer, where translated code will be written
table	label_table is used here to save current jump with appropriate label
label_pos	label position in source code. This value is used to search in label_table to "link" current jump with his label.
jmp_n_call_pos	jump or call position in source code. This value is also used for search
jmp_n_call_code	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

dst_code	buffer, where translated code will be injected
table	label_table structure, and for table the make_label_table() function must be called
search_indx	index in table->elem array. When the make_label_table() function was called, labels were added to the label_table structure. In the future, when translating the code, the hash table is searched using the label_table_search_by_label() 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 label_table structure.
label_pos	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

src_file_name	name of source file to read
src_code_save	object of structure assembly_code to save source code

Returns

FILE_OPENING_ERROR - if cannot open the file. MALLOC_ERROR - if memory allocation error detected, else return 0.

Parameters

src_file_name	name of source file to translate
src_code_save	object of assembly_code 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) contating all info about labels (such as position in source code)

Parameters

src_code	source code where labels will be looked up
table	object of label_table to save info about labels

4.3.4.12 translate_ahead_jmp_n_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 )  [inline]
```

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

Parameters

dst_code	buffer, where translated code will be written
table	label_table is used here to save current jump with appropriate label
label_pos	label position in source code. This value is used to search in label_table to "link" current jump with his label
jmp_n_call_pos	jump or call position in source code. This value is also used for search
jmp_n_call_code	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

dst_code	buffer, where translated code will be written
op_id	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

dst_code	buffer, where translated code will be written
table	label table structure. Need to find labels position by knowing jump position (in this case it is posible because the label is before jump)
label_pos	in source assembler code. This value will be used to find the appropriate to label to current jump or call.
jmp_n_call_pos	position of jump or call in source assembler code. This value also used to find the appropriate to this jump label
jmp_n_call_code	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

dst_code	buffer, where translated code will be injected
jmp_n_call_code	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

dst_code	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

dst_code	buffer, where translated code will be written
data	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

dst_code	buffer, where translated code will be written
reg_id	my assembler language register indeficator

4.3.4.19 translate_ret()

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

translate "ret" instruction to leave buffer execution

Parameters

dst_code	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 begining of the buffer to save return address

Parameters

dst_code	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

dst_code	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

dst_code	buffer, where translated code will be written
jmp_code	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

src_file_name	name of source file to translate
dst_code	buffer, where translated code will be injected
time_flag	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

dst_code	buffer, where translated code will be injected
jmp_pos	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

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

4.4 translator.h

Go to the documentation of this file.

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

```
#include <label_table.h>

#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


//+================| WORD SIZES |==================+

enum WORD_SIZE : u_int64_t {
        DWORD   = 4,
        QWORD   = 8,
        XMMWORD = 16,
};

//-================================================-

//+===============| COMPARISONS |==================+

enum VCMPPD_COMPARISONS_CODE : u_int64_t {
        EQUAL = 0,
        LESS = 17,
        GREATER = 30
};

//-================================================-


//+===========| MASK FOR XMM REGISTER |============+

enum REG_MASK : u_int64_t {
        XMM0_EXTEND = 0x44,
        XMM5_BASE   = 0x2C,
        XMM5_EXTEND = 0x6C
};

//-================================================-


//+============| TRANSLATION ERRORS |==============+

enum TRANSLATION_ERROR {
  CVT_ERROR = -0xDED,
  MALLOC_ERROR,
  FILE_OPENING_ERROR,
  UNKNOWN_FILE
};


//+===========| HOST ASSEMBLER OPCODES |============+

enum HOST_STACK_OP_CODES { // C++11
  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, // Pop to memory with register index
  PUSHM = 1  | 0x70,
  PUSHR = 1  | 0x60,
  PUSHRM = 1 | 0x35

};

enum HOST_ASSEMBLY_REG_ID : u_int64_t {
  AX = 1,
  BX,
  CX,
  DX
};
```

```


//-===============================================-

//+=============| NATIVE ASSEMBLER |==============+

enum X86_ASSEMBLY_OPCODES : u_int64_t {

  //+===============| REG_IDS |===================+
  //
  RAX = 0,
  RBX = 3, //
  RCX = 1,
  RDX = 2,

  XMM0 = RAX,
  XMM1 = RCX,
  XMM2 = RDX,
  XMM3 = RBX,
  XMM4,

  //-===============================================-

  //+================| OP_CODES |===================+

  // NOT THAT ALL OP_CODES REVERSED BECAUSE OF LITTLE ENDIAN
          /*
                          -------------------------+-+
                         // .------------------v   v v
                        ,, .  vmovq xmm(0-5), [rsp - 8]     | XMM Encoding
                        || | +------^                       | xmm0 = 0x44 = 01000100b
                        ^^ ^ ^                              | xmm1 = 0x4C = 01010100b  ...  */
  VMOVQ_XMM_RSP_IMM = 0x0024007EFAC5, //                                         ^
          /*                                                                     |
                      +---------+                                                |
                     /   +------+---------+    | XMM Encoding:                   |
                    .  /        |         |    | xmm0 = 0x4                      |
                    | .         V         v    | xmm1 = 0xC ..                   |
                    | | vmovq [rsp], xmm(0-5)                                    |
                    ^ ^                                                          |    */
  VMOVQ_RSP_XMM = 0x2400D6F9C5, //                                               |
          /*                                                                     |
                     +---------+                                                 |
                    /          |                                                 |
                   .           V                                                 |
                   | sub rsp, imm                                                |
                   ^                                                             |    */
  SUB_RSP_IMM  = 0x00EC8348, //                                                  |
  ADD_RSP_IMM  = 0x00C48348, //                                                  |
          /*                                                                     |
                         .----------------+     +---------< XMM Encoding: see  +++
                        /  --------+      |     |                                |
                        | /        V      V     V                                |
                        | | vmovq [rsp + 00], xmm(0-4)                           |
                        | |  .----------------^                                  |
                        V V  V                                                   |     */
  VMOVQ_RSP_IMM_XMM = 0x002400D6F9C5,  //                                        |
          /*                  ^ ^ ^                                              |
                              | | |                                              |
                              vmovq code                                         |
                                                                                 |
                           .----------------+     +------------< Encoding: See  ++
                          /   .-------+     |     |
                        ./ ./         V     V     V
                        |  |  addsd [rsp + 00], xmm(0-4)
                        |  |  .------------------^
                        |  | /                                                          */
  ADDSD_XMM_RSP_IMM = 0x002400580FF2,
          /*                  ^ ^ ^
                              | | |
                              addsd                                                     */
  SUBSD_XMM_RSP_IMM = 0x0024005C0FF2,
  MULSD_XMM_RSP_IMM = 0x002400590FF2,
  DIVSD_XMM_RSP_IMM = 0x0024005E0FF2,

          /*              +-+-+-+------------+
                         / / / /             |
                        . . . .      +-+-+-+-+-+-+-+-+
                        | | | |      V V V V V V V V V
                        ^ ^ ^ ^      vsqrtpd xmm0, xmm0                                  */
  VSQRTPD_XMM0_XMM0 = 0xC051F9C5,
          /*
            Used to memory reference

                             +-----------------------+
                            /                        |
                           .                         V
                           |     vmovq xmm5, [r13 + imm]
                           ^                                                             */
  VMOVQ_XMM5_R13_B_IMM = 0x006D7E7AC1C4,
          /*
            Identical, but the imm is double word (4 bytes)
                                                                                         */
  VMOVQ_XMM5_R13_D_IMM = 0xAD7E7AC1C4,

  VMOVQ_R13_B_IMM_XMM5 = 0x006DD679C1C4,

          /* !NOTE!: SIZEOF VMOVQ_R13_D_IMM_XMM is 9 bytes,
                     so i have to split it, and cannot use the last byte.
                     It is suit for my case
                                                                                         */

  VMOVQ_R13_D_IMM_XMM5  = 0xADD679C1C4,
          /* Relative jump
                   +-+-+-+--------
                  / / / /        |
                 . . . .         V
                 | | | |    jmp rel32
                 ^ ^ ^ ^                                                                 */
  JMP_REL32  = 0x00000000E9,
  JE_REL32   = 0x00000000840F,
          /*                  +-------------------------+
                             /                          |
                            .                           V
                            |  vcmppd xmm5, xmm0, xmm5, 0 - Comparison mod
                            ^                                                            */
  VCMPPD_XMM5_XMM0_XMM5 = 0x00EDC2F9C5,
          /*       +-+-+-+-----------+
                  / / / /            |
                 . . . .       +-+-+-+-+-+
                 | | | |       | | | | | |
                 | | | |       V V V V V V
                 | | | |       cmp r14d, 3    - Note that r14d contain the mask, and the mask 3
                 | | | |                        mean that for first double and second double in xmm
                 | | | |                        register the comparison condition is true
                 | | | |                        Because we don’t use the hight half of xmm register
                 | | | |                        The second bit, ,will be always set. So this is Why we use
                 ^ ^ ^ ^                        number 3, not 1                           */
  CMP_R14D_3 = 0x03FE8341,
  CMP_R14D_1 = 0x01FE8341,


  LEA_RDI_RSP_16 = 0x10247C8D48,
          /*
                   +-+-+-+----------------+
                  / / / /                 |
                 . . . .                  V
                 | | | |         mov edi, 0
                 ^ ^ ^ ^                                                                   */
  MOV_EDI_0  = 0x00000000BF,

          /*
                             +-+-+-+-+-----------+
                            / / / / /            |
                           . . . . .   +-+-+-+-+-+-+-+-+-+
                           | | | | |   v V v v v v v v v v
                           ^ ^ ^ ^ ^   movmskpd r14d, xmm5                                  */
 MOVMSKPD_R14D_XMM5    = 0xF5500F4466,
          /*
             To call scanf
                             +-+-+-----------+
                            / / /            |
                           . . .       +-+-+-+-+-+
                           | | |       V V V V V V
                           ^ ^ ^       mov rdi, rsp                                          */
 MOV_RDI_RSP           = 0xE78948,

 NATIVE_RET        = 0xC3,

          /*            .-.-.-.-----------< Rel 32 immediate address
                        | | | |
                        v v v v                                                               */
  NATIVE_CALL       = 0x00000000E8,

  MOV_R15_RSP       = 0xE78949, // Full opcode because instruction is used once
  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

};

//-================================================-

struct opcode
{
        u_int64_t code;
        int       size;
};

union cvt_u_int64_t_int {
        int       rel_addr;
        u_int64_t extended_address;
};


struct assembly_code
{
        char*  code;
        int    position;
        size_t size;
};

int assembly_code_init(assembly_code* const __restrict self,
                       const size_t                    size)
        __attribute__((nonnull(1)));

int load_code(const char*    const __restrict src_file_name,
              assembly_code* const __restrict src_code_save)
        __attribute__((nonnull(1,2)));


//+=========================| FUNCTIONS DECLARATIONS |================================+

void make_label_table(assembly_code* const __restrict src_code,
                      label_table*   const __restrict table)
        __attribute__((nonnull(1,2)));

int assembly_code_aligned_init(assembly_code* const __restrict self,
                               const size_t                    alignment,
                               const size_t                    size)
        __attribute__((nonnull(1)));


void link_label(assembly_code* const __restrict dst_code,
                label_table* const   __restrict table,
                const int                       search_indx,
                const int                       label_pos)
        __attribute__((nonnull(1,2)));

void traslate_rel32_label(assembly_code* const __restrict dst_code,
                          const size_t                    jmp_pos)
        __attribute__((nonnull(1)));


opcode translate_jmp_n_call(assembly_code* const __restrict dst_code,
                            const int                       jmp_n_call_code)
        __attribute__((nonnull(1), always_inline));


void execute_start(char* const __restrict execution_buffer,
                   const int              time_flag)
        __attribute__((nonnull(1)));

void translation_start(const char *const __restrict    src_file_name,
                       assembly_code *const __restrict dst_buffer,
                       const int                       time_flag)
        __attribute__((nonnull(1,2)));

int load_code(const char    *const __restrict    src_file_name,
              assembly_code *const __restrict src_code_save)
         __attribute__((nonnull(1,2)));

void command_line_handler(int argc, char* argv[])
         __attribute__((nonnull(2)));

extern "C" int double_printf(double* value);

extern "C" int double_scanf (double* value);


//+======================| INLINE FUNCTIONS DECLARATIONS |===========================+


inline void write_command(assembly_code* const __restrict  dst_code,
                          opcode                           operation_code)
         __attribute__((always_inline, nonnull(1)));


inline void translate_load_rsp(assembly_code* const __restrict dst_code)
         __attribute__((always_inline, nonnull(1)));


inline 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));

inline void translate_push(assembly_code* const __restrict dst_code,
                           const u_int64_t                 data)
         __attribute__((nonnull(1), always_inline));

inline void translate_push_r(assembly_code* const __restrict dst_code,
                             const int                       reg_id)
         __attribute__((nonnull(1), always_inline));

inline 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(1,2), always_inline));

inline 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(1,2), always_inline));

inline 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(1,2), always_inline));

inline void translate_save_rsp(assembly_code* const __restrict dst_code)
         __attribute__((nonnull(1), always_inline));

inline void translate_stdout(assembly_code* const __restrict dst_code)
        __attribute__((nonnull(1), always_inline));

inline void translate_two_pop_for_cmp(assembly_code* const __restrict dst_code,
                                      const int                       jmp_code)
        __attribute__((nonnull(1), always_inline));


inline void translate_ret(assembly_code* const __restrict dst_code)
         __attribute__((nonnull(1), always_inline));


inline void translate_arithmetic_op(assembly_code* const __restrict dst_code,
                                    const int                       op_id)
        __attribute__((nonnull(1), always_inline));

//-=====================================================================-


```

Index