AssemblerEmulator - Nikola Bebic

AUTHOR
Version
Sat Aug 26 2017

Table of Contents

Table of contents

AssemblerEmulator

System software

School of Electrical Engineering

University of Belgrade

Quick Start

This is a school project for the system software class. The purpose of the project is to write the assembler for the MicroRISC language and the emulator which would execute the programs produced by the assembler.

Installation

Just run the `build.sh` script and everything should be ready to go

Usage

Run the `run.sh` script with the -f option with the name of the `.ss` input file. The script will run the assembler with the input file, generate the output file, and run the emulator with that output file.

Example:

```
run.sh -f helloworld.ss
```

Input file:

```
org 0x0
.rodata.0
    dd stack - 4
    dd print_string_interrupt
    dd 30 dup dummy

org 128
.data.0
out:
    dd 0
in:
    dd 0

.bss
stack:
    DW 0x100 DUP ?

.text
dummy: ret

.global _start
_start:
    call hello
    load r0, #0
    int r0

PRINT_STRING_INTERRUPT def 1

hello:
    load r0, #hello_string
    load r15, #PRINT_STRING_INTERRUPT
    int r15 ; calls print_string_interrupt
    ret

print_string:
    loadub r1, [r0]
    jz r1, skip
    store r1, out
    load r1, #1
    add r0, r0, r1
    jmp print_string
skip:
    ret;

print_string_interrupt:
    call print_string
    ret

.rodata.1
hello_string:
    DB 'H'
    DB 'e'
    DB 'l'
    DB 'l'
    DB 'o'
    DB ','
    DB ' '
    DB 'W'
    DB 'o'
    DB 'r'
    DB 'l'
    DB 'd'
    DB '!'
    DB 10 ; CR
    DB 13 ; LF
    DB 0  ; end of string
```

Output:

```
Hello, World!
```

MicroRISC Specification

Architecture

32-bit RISC processor

32-bit virtual address space, addressable unit - byte, little-endian

No floating point arithmetic

Registers

16 32-bit general purpose registers, `R0` - `R15`

32-bit program counter: `PC`

32-bit stack pointer: `SP` . Stack grows towards higher addresses, stack pointer points to the word at the top of the stack

Constant terms

Constant terms can containt the following:

- Literals
- Arithmetic operators (+, -, *, /)
- Subexpressions with parentheses

Literals are signed decimal, binary or hexadecimal integers, or ASCII characters, as well as named constants or labels

Labels can can contain letters, digits, and symbol _ , and can not start with a letter

There is a predefined symbol `$` , which represents the address of the current instruction

Address modes

- Immediate: `#constant_term`
- Register direct: `Ri`
- Register indirect: `[Ri]`
- Register indirect with offset: `[Ri + offset]`. `offset` is a constant term
- PC relative: `$constant_term`. This is treated as register indirect with offset. Constant term must contain at least one label

Instructions

Instruction format:

```
[label:] instruction [operand0, operand1, operand2] [; comment]
```

Flow control instructions

Instruction	*Address modes*	*Comment*
`INT op`	Register direct	Generates a software interrupt. Interrupt entry is in the register
`JMP op`	Memory direct, register indirect, register indirect with offset	Jumps to given address
`CALL op`	Memory direct, register indirect, register indirect with offset	Calls a subroutine. `PC` is pushed to the stack
`RET`	None	Returns from subroutine
`JZ reg, op`	`reg`: Register direct, `op`: Memory direct, register indirect, register indirect with offset	Jumps to `op` if `reg == 0`
`JNZ reg, op`	`reg`: Register direct, `op`: Memory direct, register indirect, register indirect with offset	Jumps to `op` if `reg != 0`
`JGZ reg, op`	`reg`: Register direct, `op`: Memory direct, register indirect, register indirect with offset	Jumps to `op` if `reg > 0`
`JGEZ reg, op`	`reg`: Register direct, `op`: Memory direct, register indirect, register indirect with offset	Jumps to `op` if `reg >= 0`
`JLZ reg, op`	`reg`: Register direct, `op`: Memory direct, register indirect, register indirect with offset	Jumps to `op` if `reg < 0`
`JLEZ reg, op`	`reg`: Register direct, `op`: Memory direct, register indirect, register indirect with offset	Jumps to `op` if `reg < 0`

Load/Store instructions

Load, sizes of operands:

- Unsigned byte, suffix: `UB`

- Signed byte, suffix: `SB`
- Unsigned word, suffix: `UW`
- Signed word, suffix: `SW`
- Double word, no suffix

Store, sizes of the operands:

- Byte, suffix: `B`
- Word, suffix: `W`
- Double word, no suffix

Size of word is 2 bytes, and size of double word is 2 words

Instruction	***Address modes***	***Comment***
`LOAD reg, op`	`reg` : Register direct, `op` : All	Loads the data into the register
`STORE reg, op`	`reg` : Register direct, `op` : All except immediate	Stores the data from the register

Stack instructions

- 32-bit double word is always pushed to the stack, and popped from the stack

Instruction	***Address modes***	***Comment***
`PUSH reg`	Register direct	Pushes the register to the stack
`POP reg`	Register direct	Pops the register from the stack

ALU instructions

- Work only on 32-bit operands
- Signed arithmetic

Instruction	***Address modes***	***Comment***
`ADD reg0, reg1, reg2`	Register direct	`reg0 = reg1 + reg2`
`SUB reg0, reg1, reg2`	Register direct	`reg0 = reg1 - reg2`
`MUL reg0, reg1, reg2`	Register direct	`reg0 = reg1 * reg2`
`DIV reg0, reg1, reg2`	Register direct	`reg0 = reg1 / reg2`
`MOD reg0, reg1, reg2`	Register direct	`reg0 = reg1 % reg2`
`AND reg0, reg1, reg2`	Register direct	`reg0 = reg1 & reg2`
`OR reg0, reg1, reg2`	Register direct	`reg0 = reg1 | reg2`
`XOR reg0, reg1, reg2`	Register direct	`reg0 = reg1 ^ reg2`
`NOT reg0, reg1`	Register direct	`reg0 = ~reg1`
`ASL reg0, reg1, reg2`	Register direct	`reg0 = reg1 << reg2`
`ASR reg0, reg1, reg2`	Register direct	`reg0 = reg1 >> reg2`

Data definition

Format:

```
[label:] definition data_specifier [, ...] [; comment]
```

Possible definitions:

- `DB` - defines a byte
- `DW` - defines a word

- `DD` - defines a double word

Data specifiers:

```
constant_term [ DUP constant_term | ? ]
```

- `DUP` - First constant term denotes how many times the second constant term will occur
- `?` - Undefined value

Directives:

Named constant definition:

```
symbol DEF constant_expression [; comment]
```

Origin directive:

```
ORG constant_expression [; comment]
```

Segments

- `.text[.number]` - section containing the program code
- `.data[.number]` - section containing initialized data
- `.rodata[.number]` - section containing read only data
- `.bss[.number]` - section containing uninitialized data

Interrupts:

- IV table starts at the address 0 and has 32 entries
- During the interrupt execution, no hardware interrupt can happen
- Executing `INT 0` will end the program
- Entry 0 in the IVT contains the starting value of the stack pointer
- Entry 3 in the IVT contains the address of the error interrupt routine
- Entry 4 in the IVT contains the address of the timer interrupt routine. This routine is called every `0,1s`
- Entry 5 in the IVT contains the address of the keyboard interrupt routine. This routine is called every time a key is pressed

Two registers are mapped in the address space, right after the IV table.

The first register is mapped to the address `128` and is the `stdout` register. Every time a value is written to this register, it will be written on the standard output stream

The second register is mapped to the address `132` and is the `stdin` register. Every time a keyboard interrupt happens, this register will contain the ASCII code of the hit character. The value can be read more than once. New interrupts will not happen until the value is read at least once

Licence

Namespace Index

Namespace List

Here is a list of all namespaces with brief descriptions:

bnssassembler 26
bnssemulator 53
consoleio 70
cxxopts 72
cxxopts::anonymous_namespace{cxxopts.h} 75
cxxopts::values 78
std 80
z85 81

Hierarchical Index

Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

bnssemulator::AssemblerOutput 111
bnssemulator::CommandLineHelper 120
bnssassembler::CommandLineHelper 122
bnssemulator::compare_pair_difference< T, Pred > 124
bnssemulator::compare_pair_first< T1, T2, Pred > 125
bnssemulator::compare_pair_second< T1, T2, Pred > 126
bnssemulator::Context 130
bnssassembler::Data 142
bnssassembler::DataTypeParser 151
bnssassembler::DataTypeParser::DataTypeParserStaticData 153
enable_shared_from_this
  cxxopts::Value 516
    cxxopts::values::standard_value< T > 450
    cxxopts::values::standard_value< T > 450

  cxxopts::Value 516

exception
  bnssassembler::AssemblerException 108
    bnssassembler::FirstPassException 190
    bnssassembler::ParserException 356
    bnssassembler::SecondPassException 413

  bnssassembler::MessageException 276
    bnssassembler::DivisionByZeroException 161
    bnssassembler::IncorrectLabelException 211
    bnssassembler::InvalidDataDefinitionException 237
    bnssassembler::InvalidDataTypeException 238
    bnssassembler::InvalidExpressionException 239
    bnssassembler::NonExistingSymbolException 295

  bnssemulator::MessageException 274
  cxxopts::OptionException 329
    cxxopts::OptionParseException 331
      cxxopts::argument_incorrect_type 105
      cxxopts::argument_incorrect_type 105
      cxxopts::missing_argument_exception 286
      cxxopts::missing_argument_exception 286
      cxxopts::option_not_exists_exception 317
      cxxopts::option_not_exists_exception 317
      cxxopts::option_not_has_argument_exception 318
      cxxopts::option_not_has_argument_exception 318
      cxxopts::option_not_present_exception 319
      cxxopts::option_not_present_exception 319
      cxxopts::option_required_exception 320

cxxopts::option_required_exception 320
cxxopts::option_requires_argument_exception 321
cxxopts::option_requires_argument_exception 321

cxxopts::OptionParseException 331
cxxopts::OptionSpecException 346
cxxopts::invalid_option_format_error 236
cxxopts::invalid_option_format_error 236
cxxopts::option_exists_error 316
cxxopts::option_exists_error 316

cxxopts::OptionSpecException 346

cxxopts::OptionException 329

bnssemulator::Executer 162
bnssemulator::AluExecuter 101
bnssemulator::AddExecuter 85
bnssemulator::AndExecuter 104
bnssemulator::AslExecuter 106
bnssemulator::AsrExecuter 107
bnssemulator::DivideExecuter 154
bnssemulator::ModuloExecuter 287
bnssemulator::MultiplyExecuter 289
bnssemulator::OrExecuter 347
bnssemulator::SubtractExecuter 472
bnssemulator::XorExecuter 522

bnssemulator::CallExecuter 116
bnssemulator::ConditionalJumpExecuter 127
bnssemulator::JgezExecuter 240
bnssemulator::JgzExecuter 242
bnssemulator::JlezExecuter 244
bnssemulator::JlzExecuter 246
bnssemulator::JnzExecuter 249
bnssemulator::JzExecuter 251

bnssemulator::IntExecuter 234
bnssemulator::JmpExecuter 248
bnssemulator::LoadExecuter 265
bnssemulator::NotExecuter 297
bnssemulator::PopExecuter 358
bnssemulator::PushExecuter 364
bnssemulator::RetExecuter 402
bnssemulator::StoreExecuter 457

bnssassembler::Expression 164
bnssassembler::Literal 259
bnssassembler::Operation 308
bnssassembler::AddOperation 87
bnssassembler::DivideOperation 156

bnssassembler::MultiplyOperation 290
bnssassembler::SubtractOperation 473

bnssassembler::Symbol 480

bnssassembler::ExpressionBuilder 168
bnssassembler::ExpressionToken 170
bnssassembler::LiteralToken 261
bnssassembler::OperationToken 314
bnssassembler::AddToken 98
bnssassembler::ClosingBraceToken 117
bnssassembler::DivideToken 158
bnssassembler::MultiplyToken 292
bnssassembler::OpeningBraceToken 299
bnssassembler::SubtractToken 477

bnssassembler::SymbolToken 504

bnssassembler::ExpressionTokenFactory 174
bnssassembler::ExpressionTokenFactory::ExpressionTokenFactoryData 176
bnssemulator::FileReader 178
bnssassembler::FileReader 180
bnssassembler::FileWriter 182
bnssassembler::FirstPass 184
bnssassembler::FirstPassData 186
std::hash< bnssassembler::InstructionCode > 196
std::hash< bnssassembler::SectionData > 197
std::hash< bnssassembler::SectionType > 198
std::hash< bnssassembler::SymbolDefinition > 199
std::hash< bnssemulator::InstructionCode > 200
std::hash< bnssemulator::SectionType > 201
cxxopts::HelpGroupDetails 202
cxxopts::HelpOptionDetails 203
bnssemulator::InstructionBitField 212
bnssassembler::InstructionBitField 214
bnssemulator::InstructionBitFieldUnion 216
bnssassembler::InstructionBitFieldUnion 217
bnssassembler::InstructionCodeParser 218
bnssassembler::InstructionCodeParser::InstructionCodeParserStaticData 220
bnssassembler::InstructionParser 225
bnssassembler::AluInstructionParser 103
bnssassembler::ConditionalJumpInstructionParser 129
bnssassembler::InterruptInstructionParser 233
bnssassembler::LoadInstructionParser 267
bnssassembler::NoOperandInstructionParser 296
bnssassembler::NotInstructionParser 298
bnssassembler::StackInstructionParser 449
bnssassembler::StoreInstructionParser 459
bnssassembler::UndonditionalJumpInstructionParser 515

bnssemulator::KeyboardListener....253
bnssassembler::LineParser....256
    bnssassembler::DataDefinitionLineParser....145
    bnssassembler::GlobalSymbolsLineParser....192
    bnssassembler::InstructionLineParser....222
    bnssassembler::OrgDirectiveLineParser....348
    bnssassembler::SectionStartLineParser....430
    bnssassembler::SymbolDefinitionLineParser....496

map
    bnssemulator::AddressSpace....89

bnssassembler::MicroRiscExpression....278
bnssassembler::Operand....302
    bnssassembler::Immediate....205
    bnssassembler::MemoryDirect....268
    bnssassembler::RegisterDirect....373
    bnssassembler::RegisterIndirect....378
    bnssassembler::RegisterIndirectOffset....381

bnssassembler::OperandParser....305
    bnssassembler::ImmediateParser....209
    bnssassembler::MemoryDirectParser....272
    bnssassembler::RegisterDirectParser....376
    bnssassembler::RegisterIndirectOffsetParser....386
    bnssassembler::RegisterIndirectParser....388

cxxopts::OptionAdder....322
cxxopts::OptionDetails....325
cxxopts::Options....333
bnssassembler::Parser....353
    bnssassembler::MicroRiscParser....282

bnssemulator::Processor....359
bnssemulator::Processor::ProcessorStaticData....362
bnssemulator::Register....365
bnssassembler::RegisterParser....390
bnssassembler::RegisterParser::RegisterParserStaticData....392
bnssassembler::RelocationRecord....393
bnssemulator::RelocationRecord....398
bnssassembler::SecondPass....403
bnssassembler::SecondPassData....405
bnssassembler::SectionData....415
bnssemulator::SectionData....424
bnssassembler::SectionTypeParser....440
bnssassembler::SectionTypeParser::SectionTypeParserData....442
bnssemulator::StringHelper....460
bnssassembler::StringHelper....466

bnssemulator::SymbolData 486
bnssassembler::SymbolData 489
bnssassembler::SymbolDefinition 493
bnssemulator::TimerListener 508
bnssassembler::Token 509
    bnssassembler::DataDefinitionToken 147
    bnssassembler::GlobalSymbolsToken 194
    bnssassembler::InstructionToken 228
    bnssassembler::LabelToken 254
    bnssassembler::OrgDirectiveToken 350
    bnssassembler::SectionStartToken 432
    bnssassembler::SymbolDefinitionToken 498

cxxopts::values::type_is_container< T > 513
cxxopts::values::type_is_container< std::vector< T > > 514
unordered_map
    bnssassembler::SymbolTable 501

cxxopts::values::value_has_arg< T > 520
cxxopts::values::value_has_arg< bool > 521
vector
    bnssassembler::SectionTable 435
    bnssemulator::Segment 444

Class Index

Class List

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

bnssemulator::AddExecuter (Class representing the executer for the add instruction) 85
bnssassembler::AddOperation (Class implementing the behaviour of the + operator in expressions) 87
bnssemulator::AddressSpace (Class representing the address space of the emulator) 89
bnssassembler::AddToken (Token class representing the + operation) 98
bnssemulator::AluExecuter (Base class used for executing ALU instructions) 101
bnssassembler::AluInstructionParser (Class representing the parser for ALU instructions) ..103
bnssemulator::AndExecuter (Class representing the executer for the and instruction) 104
cxxopts::argument_incorrect_type 105
bnssemulator::AslExecuter (Class representing the executer for the asl instruction) 106
bnssemulator::AsrExecuter (Class representing the executer for the asr instruction) 107
bnssassembler::AssemblerException (Class representing the custom exception for the assembler) 108
bnssemulator::AssemblerOutput (Class representing the output from the assembler) 111
bnssemulator::CallExecuter (Class representing the executer for the call instruction) 116
bnssassembler::ClosingBraceToken (Token class representing the opening brace) 117
bnssemulator::CommandLineHelper (Utility class used for parsing the command line) 120
bnssassembler::CommandLineHelper (Utility class used to parse the command line) 122
bnssemulator::compare_pair_difference< T, Pred > 124
bnssemulator::compare_pair_first< T1, T2, Pred > 125
bnssemulator::compare_pair_second< T1, T2, Pred > 126
bnssemulator::ConditionalJumpExecuter (Base executer for conditional jump instructions) .127
bnssassembler::ConditionalJumpInstructionParser (Class representing the parser for conditional jump instructions) 129
bnssemulator::Context (Class representing the context of the processor) 130
bnssassembler::Data (Class representing the MicroRISC data) 142
bnssassembler::DataDefinitionLineParser (Class used for parsing data definitions) 145
bnssassembler::DataDefinitionToken (Class representing the data definition token) 147
bnssassembler::DataTypeParser (Utility class used for parsing data types) 151
bnssassembler::DataTypeParser::DataTypeParserStaticData 153
bnssemulator::DivideExecuter (Class representing the executer of the divide instruction) 154
bnssassembler::DivideOperation (Class implementing the behaviour of the / operator in expressions) 156
bnssassembler::DivideToken (Token class representing the / operation) 158
bnssassembler::DivisionByZeroException (Exception class representing division by zero) 161
bnssemulator::Executer (Base class used for executing instructions) 162
bnssassembler::Expression (Class representing the math expression) 164
bnssassembler::ExpressionBuilder (Utility class used for building math expressions) 168
bnssassembler::ExpressionToken (Class representing the token found in infix and postfix expressions) 170
bnssassembler::ExpressionTokenFactory (Utility class used for creating the ExpressionToken objects) 174
bnssassembler::ExpressionTokenFactory::ExpressionTokenFactoryData 176
bnssemulator::FileReader (Utility class used for reading assembler output from the file) 178

bnssassembler::FileReader (Utility class providing methods for reading the file) 180
bnssassembler::FileWriter (Utility class used to write the assembler result to a file) 182
bnssassembler::FirstPass (Class representing the executor of the first pass) 184
bnssassembler::FirstPassData (Class representing the data that the two-pass assembler will modify in the first pass) 186
bnssassembler::FirstPassException (Represents an exception that happend during the assembler first pass) 190
bnssassembler::GlobalSymbolsLineParser (Class used for parsing information about global symbols) 192
bnssassembler::GlobalSymbolsToken (Class representing the global symbols token) 194
std::hash< bnssassembler::InstructionCode > 196
std::hash< bnssassembler::SectionData > 197
std::hash< bnssassembler::SectionType > 198
std::hash< bnssassembler::SymbolDefinition > 199
std::hash< bnssemulator::InstructionCode > 200
std::hash< bnssemulator::SectionType > 201
cxxopts::HelpGroupDetails 202
cxxopts::HelpOptionDetails 203
bnssassembler::Immediate (Class representing the immediate operand) 205
bnssassembler::ImmediateParser (Class representing the parser for the immediate operands) 209
bnssassembler::IncorrectLabelException (Exception representing the incorrect label) 211
bnssemulator::InstructionBitField (Bit field that enables easier manipulation of instructions) 212
bnssassembler::InstructionBitField (Bit field that enables easier manipulation of instructions) 214
bnssemulator::InstructionBitFieldUnion (Union that enables easier manipulation of the instruction bit field) 216
bnssassembler::InstructionBitFieldUnion (Union that enables easier manipulation of the instruction bit field) 217
bnssassembler::InstructionCodeParser (Utility class used for parsing instruction codes) 218
bnssassembler::InstructionCodeParser::InstructionCodeParserStaticData 220
bnssassembler::InstructionLineParser (Class used for parsing instructions) 222
bnssassembler::InstructionParser (Abstract lass used for parsing one instruction) 225
bnssassembler::InstructionToken (Class representing the instruction in an assembler source file) 228
bnssassembler::InterruptInstructionParser (Class representing the parser for the interrupt instruction) 233
bnssemulator::IntExecuter (Class representing the executer for the int instruction) 234
cxxopts::invalid_option_format_error 236
bnssassembler::InvalidDataDefinitionException (Exception representing invalid data definition) 237
bnssassembler::InvalidDataTypeException (Exception representing the invalid data type) ...238
bnssassembler::InvalidExpressionException (Exception representing the invalid expression) 239
bnssemulator::JgezExecuter (Class representing the executer for the jgez instruction) 240
bnssemulator::JgzExecuter (Class representing the executer for the jgz instruction) 242
bnssemulator::JlezExecuter (Class representing the executer for the jlez instruction) 244
bnssemulator::JlzExecuter (Class representing the executer for the jlz instruction) 246
bnssemulator::JmpExecuter (Class representing the executer for the jmp instruction) 248
bnssemulator::JnzExecuter (Class representing the executer for the jnz instruction) 249

bnssemulator::JzExecuter (Class representing the executer for the jz instruction) 251
bnssemulator::KeyboardListener (Class representing the keyboard listener thread) 253
bnssassembler::LabelToken (Class representing the label token) 254
bnssassembler::LineParser (Chain of command abstract class used for parsing one line of file) 256
bnssassembler::Literal (Class representing the literal value) 259
bnssassembler::LiteralToken (Token class representing a math literal value) 261
bnssemulator::LoadExecuter (Class representing the executer for the load instruction) 265
bnssassembler::LoadInstructionParser (Class representing the load instruction parser) 267
bnssassembler::MemoryDirect (Class representing the memory direct operand) 268
bnssassembler::MemoryDirectParser (Class representing the parser for the memory direct operand) 272
bnssemulator::MessageException (Represents an exception with a string message) 274
bnssassembler::MessageException (Represents an exception with a string message) 276
bnssassembler::MicroRiscExpression (Adapter class for Expression) 278
bnssassembler::MicroRiscParser (Class representing the parser for the MicroRISC assembly) 282
cxxopts::missing_argument_exception 286
bnssemulator::ModuloExecuter (Class representing the executer for the modulo instruction) 287
bnssemulator::MultiplyExecuter (Class representing the executer for the multiply instruction) 289
bnssassembler::MultiplyOperation (Class implementing the behaviour of the * operator in expressions) 290
bnssassembler::MultiplyToken (Token class representing the * operation) 292
bnssassembler::NonExistingSymbolException (Exception representing the non existing symbol) 295
bnssassembler::NoOperandInstructionParser (Class representing the parser for the instruction without operands) 296
bnssemulator::NotExecuter (Class representing the executer for the not instruction) 297
bnssassembler::NotInstructionParser (Class representing the parser for the not instruction) 298
bnssassembler::OpeningBraceToken (Token class representing the opening brace) 299
bnssassembler::Operand (Class representing one operand in an instruction) 302
bnssassembler::OperandParser (Chain of command class used to parse operands of the instructions) 305
bnssassembler::Operation (Class representing the mathematical operation with two operands) 308
bnssassembler::OperationToken (Token class representing a math operator) 314
cxxopts::option_exists_error 316
cxxopts::option_not_exists_exception 317
cxxopts::option_not_has_argument_exception 318
cxxopts::option_not_present_exception 319
cxxopts::option_required_exception 320
cxxopts::option_requires_argument_exception 321
cxxopts::OptionAdder 322
cxxopts::OptionDetails 325
cxxopts::OptionException 329
cxxopts::OptionParseException 331
cxxopts::Options 333
cxxopts::OptionSpecException 346

bnssemulator::OrExecuter (Class representing the executer for the or instruction) 347
bnssassembler::OrgDirectiveLineParser (Class representing a line parser for the origin directive) 348
bnssassembler::OrgDirectiveToken (Class representing the origin directive token) 350
bnssassembler::Parser (Abstract class representing a text parser) 353
bnssassembler::ParserException (Represents an exception that happend during the parsing of the file) 356
bnssemulator::PopExecuter (Class representing the executer for the pop instruction) 358
bnssemulator::Processor (Class representing the processor) 359
bnssemulator::Processor::ProcessorStaticData 362
bnssemulator::PushExecuter (Class representing the executer for the push instruction) 364
bnssemulator::Register (Class representing the register) 365
bnssassembler::RegisterDirect (Class representing the register direct operand) 373
bnssassembler::RegisterDirectParser (Class representing the parser for the register direct operand) 376
bnssassembler::RegisterIndirect (Class representing the register indirect operand) 378
bnssassembler::RegisterIndirectOffset (Class representing the register indirect operand with offset) 381
bnssassembler::RegisterIndirectOffsetParser (Class representing the parser for the register indirect operand with offset) 386
bnssassembler::RegisterIndirectParser (Class representing the parser for the register indirect operand) 388
bnssassembler::RegisterParser (Utility class used for parsing registers) 390
bnssassembler::RegisterParser::RegisterParserStaticData 392
bnssassembler::RelocationRecord (Class representing one relocation record) 393
bnssemulator::RelocationRecord (Class representing one relocation record) 398
bnssemulator::RetExecuter (Class representing the executer for ret instruction) 402
bnssassembler::SecondPass (Utility class executing the second pass) 403
bnssassembler::SecondPassData (Class representing the data that will be updated during the second pass) 405
bnssassembler::SecondPassException (Represents an exception that happened during the assembler second pass) 413
bnssassembler::SectionData (Class representing the data about one section) 415
bnssemulator::SectionData (Class representing the data about one section) 424
bnssassembler::SectionStartLineParser (Class used for parsing section start definitions) 430
bnssassembler::SectionStartToken (Class representing the section start token) 432
bnssassembler::SectionTable (Class representing the table of sections) 435
bnssassembler::SectionTypeParser (Utility class representing the parser for the section types) 440
bnssassembler::SectionTypeParser::SectionTypeParserData 442
bnssemulator::Segment (Class representing one segment of memory) 444
bnssassembler::StackInstructionParser (Class representing the parser for stack instructions) 449
cxxopts::values::standard_value< T > 450
bnssemulator::StoreExecuter (Class representing the executer for the store instruction) 457
bnssassembler::StoreInstructionParser (Class representing the parser for the store instruction) 459
bnssemulator::StringHelper (Utility class providing helper methods for std::string class) 460
bnssassembler::StringHelper (Utility class providing helper methods for std::string class) 466
bnssemulator::SubtractExecuter (Class representing the executer for the subtract instruction) 472

bnssassembler::SubtractOperation (Class implementing the behaviour of the - operator in expressions) 473
bnssassembler::SubtractToken (Token class representing the - operation) 477
bnssassembler::Symbol (Class representing a symbol inside an expression) 480
bnssemulator::SymbolData (Class representing data about one symbol) 486
bnssassembler::SymbolData (Class representing data about one symbol) 489
bnssassembler::SymbolDefinition (Class representing a symbol definition) 493
bnssassembler::SymbolDefinitionLineParser (Class used for parsing symbol definitions) 496
bnssassembler::SymbolDefinitionToken (Class representing the symbol definition token) 498
bnssassembler::SymbolTable (Class representing the symbol table) 501
bnssassembler::SymbolToken (Token class representing a math symbol) 504
bnssemulator::TimerListener (Class representing a listener for the timer events) 508
bnssassembler::Token (Class representing one token of the assembler source file) 509
cxxopts::values::type_is_container< T > 513
cxxopts::values::type_is_container< std::vector< T > > 514
bnssassembler::UndonditionalJumpInstructionParser (Class representing the parser for the unconditional jump instructions) 515
cxxopts::Value 516
cxxopts::values::value_has_arg< T > 520
cxxopts::values::value_has_arg< bool > 521
bnssemulator::XorExecuter (Class representing the executer for the xor instruction) 522

File Index

File List

Here is a list of all files with brief descriptions:

Code/Assembler/Include/AddOperation.h 523
Code/Assembler/Include/AddressMode.h 524
Code/Assembler/Include/AddToken.h 526
Code/Assembler/Include/AluInstructionParser.h 527
Code/Assembler/Include/AssemblerException.h 528
Code/Assembler/Include/ClosingBraceToken.h 529
Code/Assembler/Include/CommandLineHelper.h 530
Code/Assembler/Include/CommonRegexes.h 532
Code/Assembler/Include/ConditionalJumpInstructionParser.h 534
Code/Assembler/Include/cxxopts.h 535
Code/Assembler/Include/Data.h 539
Code/Assembler/Include/DataDefinitionLineParser.h 540
Code/Assembler/Include/DataDefinitionToken.h 541
Code/Assembler/Include/DataType.h 542
Code/Assembler/Include/DataTypeParser.h 544
Code/Assembler/Include/DivideOperation.h 545
Code/Assembler/Include/DivideToken.h 546
Code/Assembler/Include/DivisionByZeroException.h 547
Code/Assembler/Include/Expression.h 548
Code/Assembler/Include/ExpressionBuilder.h 549
Code/Assembler/Include/ExpressionToken.h 550
Code/Assembler/Include/ExpressionTokenFactory.h 551
Code/Assembler/Include/FileReader.h 552
Code/Assembler/Include/FileWriter.h 554
Code/Assembler/Include/FirstPass.h 555
Code/Assembler/Include/FirstPassData.h 556
Code/Assembler/Include/FirstPassException.h 557
Code/Assembler/Include/GlobalSymbolsLineParser.h 558
Code/Assembler/Include/GlobalSymbolToken.h 559
Code/Assembler/Include/Immediate.h 560
Code/Assembler/Include/ImmediateParser.h 561
Code/Assembler/Include/IncorrectLabelException.h 562
Code/Assembler/Include/InstructionBitField.h 563
Code/Assembler/Include/InstructionBitFieldUnion.h 565
Code/Assembler/Include/InstructionCode.h 567
Code/Assembler/Include/InstructionCodeParser.h 569
Code/Assembler/Include/InstructionLineParser.h 570
Code/Assembler/Include/InstructionParser.h 571
Code/Assembler/Include/InstructionToken.h 572
Code/Assembler/Include/InterruptInstructionParser.h 573
Code/Assembler/Include/InvalidDataDefinitionException.h 574
Code/Assembler/Include/InvalidDataTypeException.h 575
Code/Assembler/Include/InvalidExpressionException.h 576

Code/Assembler/Include/LabelToken.h 577
Code/Assembler/Include/LineParser.h 578
Code/Assembler/Include/Literal.h 579
Code/Assembler/Include/LiteralToken.h 580
Code/Assembler/Include/LoadInstructionParser.h 581
Code/Assembler/Include/MemoryDirect.h 582
Code/Assembler/Include/MemoryDirectParser.h 583
Code/Assembler/Include/MessageException.h 584
Code/Assembler/Include/MicroRiscExpression.h 586
Code/Assembler/Include/MicroRiscParser.h 587
Code/Assembler/Include/MultiplyOperation.h 588
Code/Assembler/Include/MultiplyToken.h 589
Code/Assembler/Include/NonExistingSymbolException.h 590
Code/Assembler/Include/NoOperandInstructionParser.h 591
Code/Assembler/Include/NotInstructionParser.h 592
Code/Assembler/Include/OpeningBraceToken.h 593
Code/Assembler/Include/Operand.h 594
Code/Assembler/Include/OperandParser.h 595
Code/Assembler/Include/OperandType.h 596
Code/Assembler/Include/Operation.h 598
Code/Assembler/Include/OperationToken.h 599
Code/Assembler/Include/OrgDirectiveLineParser.h 600
Code/Assembler/Include/OrgDirectiveToken.h 601
Code/Assembler/Include/Parser.h 602
Code/Assembler/Include/ParserException.h 603
Code/Assembler/Include/PrintHelpers.h 604
Code/Assembler/Include/Register.h 605
Code/Assembler/Include/RegisterDirect.h 607
Code/Assembler/Include/RegisterDirectParser.h 608
Code/Assembler/Include/RegisterIndirect.h 609
Code/Assembler/Include/RegisterIndirectOffset.h 610
Code/Assembler/Include/RegisterIndirectOffsetParser.h 611
Code/Assembler/Include/RegisterIndirectParser.h 612
Code/Assembler/Include/RegisterParser.h 613
Code/Assembler/Include/RelocationRecord.h 614
Code/Assembler/Include/SecondPass.h 616
Code/Assembler/Include/SecondPassData.h 617
Code/Assembler/Include/SecondPassException.h 618
Code/Assembler/Include/SectionData.h 619
Code/Assembler/Include/SectionStartLineParser.h 621
Code/Assembler/Include/SectionStartToken.h 622
Code/Assembler/Include/SectionTable.h 623
Code/Assembler/Include/SectionType.h 624
Code/Assembler/Include/SectionTypeParser.h 626
Code/Assembler/Include/StackInstructionParser.h 627
Code/Assembler/Include/StoreInstructionParser.h 628
Code/Assembler/Include/StringHelper.h 629
Code/Assembler/Include/SubtractOperation.h 631
Code/Assembler/Include/SubtractToken.h 632

Code/Assembler/Include/Symbol.h 633
Code/Assembler/Include/SymbolData.h 634
Code/Assembler/Include/SymbolDefinition.h 636
Code/Assembler/Include/SymbolDefinitionLineParser.h 637
Code/Assembler/Include/SymbolDefinitionToken.h 638
Code/Assembler/Include/SymbolTable.h 639
Code/Assembler/Include/SymbolToken.h 640
Code/Assembler/Include/Token.h 641
Code/Assembler/Include/UnconditionalJumpInstructionParser.h 642
Code/Assembler/Include/z85.h 643
Code/Assembler/Include/z85_cpp.h 655
Code/Assembler/Source/AddOperation.cpp 657
Code/Assembler/Source/AddToken.cpp 658
Code/Assembler/Source/AluInstructionParser.cpp 659
Code/Assembler/Source/AssemblerException.cpp 660
Code/Assembler/Source/ClosingBraceToken.cpp 661
Code/Assembler/Source/CommandLineHelper.cpp 662
Code/Assembler/Source/ConditionalJumpInstructionParser.cpp 664
Code/Assembler/Source/Data.cpp 665
Code/Assembler/Source/DataDefinitionLineParser.cpp 666
Code/Assembler/Source/DataDefinitionToken.cpp 667
Code/Assembler/Source/DataTypeParser.cpp 668
Code/Assembler/Source/DivideOperation.cpp 669
Code/Assembler/Source/DivideToken.cpp 670
Code/Assembler/Source/DivisionByZeroException.cpp 671
Code/Assembler/Source/Expression.cpp 672
Code/Assembler/Source/ExpressionBuilder.cpp 673
Code/Assembler/Source/ExpressionTokenFactory.cpp 674
Code/Assembler/Source/FileReader.cpp 675
Code/Assembler/Source/FileWriter.cpp 677
Code/Assembler/Source/FirstPass.cpp 678
Code/Assembler/Source/FirstPassData.cpp 679
Code/Assembler/Source/FirstPassException.cpp 680
Code/Assembler/Source/GlobalSymbolsLineParser.cpp 681
Code/Assembler/Source/GlobalSymbolToken.cpp 682
Code/Assembler/Source/Immediate.cpp 683
Code/Assembler/Source/ImmediateParser.cpp 684
Code/Assembler/Source/IncorrectLabelException.cpp 685
Code/Assembler/Source/InstructionCodeParser.cpp 686
Code/Assembler/Source/InstructionLineParser.cpp 687
Code/Assembler/Source/InstructionParser.cpp 688
Code/Assembler/Source/InstructionToken.cpp 689
Code/Assembler/Source/InterruptInstructionParser.cpp 690
Code/Assembler/Source/InvalidDataDefinitionException.cpp 691
Code/Assembler/Source/InvalidDataTypeException.cpp 692
Code/Assembler/Source/InvalidExpressionException.cpp 693
Code/Assembler/Source/LabelToken.cpp 694
Code/Assembler/Source/LineParser.cpp 695
Code/Assembler/Source/Literal.cpp 696

Code/Assembler/Source/LiteralToken.cpp 697
Code/Assembler/Source/LoadInstructionParser.cpp 698
Code/Assembler/Source/Main.cpp 699
Code/Assembler/Source/MemoryDirect.cpp 701
Code/Assembler/Source/MemoryDirectParser.cpp 702
Code/Assembler/Source/MessageException.cpp 703
Code/Assembler/Source/MicroRiscExpression.cpp 705
Code/Assembler/Source/MicroRiscParser.cpp 706
Code/Assembler/Source/MultiplyOperation.cpp 707
Code/Assembler/Source/MultiplyToken.cpp 708
Code/Assembler/Source/NonExistingSymbolException.cpp 709
Code/Assembler/Source/NotInstructionParser.cpp 710
Code/Assembler/Source/OpeningBraceToken.cpp 711
Code/Assembler/Source/Operand.cpp 712
Code/Assembler/Source/OperandParser.cpp 713
Code/Assembler/Source/Operation.cpp 714
Code/Assembler/Source/OperationToken.cpp 715
Code/Assembler/Source/OrgDirectiveLineParser.cpp 716
Code/Assembler/Source/OrgDirectiveToken.cpp 717
Code/Assembler/Source/Parser.cpp 718
Code/Assembler/Source/ParserException.cpp 719
Code/Assembler/Source/RegisterDirect.cpp 720
Code/Assembler/Source/RegisterDirectParser.cpp 721
Code/Assembler/Source/RegisterIndirect.cpp 722
Code/Assembler/Source/RegisterIndirectOffset.cpp 723
Code/Assembler/Source/RegisterIndirectOffsetParser.cpp 724
Code/Assembler/Source/RegisterIndirectParser.cpp 725
Code/Assembler/Source/RegisterParser.cpp 726
Code/Assembler/Source/RelocationRecord.cpp 727
Code/Assembler/Source/SecondPass.cpp 729
Code/Assembler/Source/SecondPassData.cpp 730
Code/Assembler/Source/SecondPassException.cpp 731
Code/Assembler/Source/SectionData.cpp 732
Code/Assembler/Source/SectionStartLineParser.cpp 734
Code/Assembler/Source/SectionStartToken.cpp 735
Code/Assembler/Source/SectionTable.cpp 736
Code/Assembler/Source/SectionTypeParser.cpp 737
Code/Assembler/Source/StackInstructionParser.cpp 738
Code/Assembler/Source/StoreInstructionParser.cpp 739
Code/Assembler/Source/StringHelper.cpp 740
Code/Assembler/Source/SubtractOperation.cpp 742
Code/Assembler/Source/SubtractToken.cpp 743
Code/Assembler/Source/Symbol.cpp 744
Code/Assembler/Source/SymbolData.cpp 745
Code/Assembler/Source/SymbolDefinition.cpp 747
Code/Assembler/Source/SymbolDefinitionLineParser.cpp 748
Code/Assembler/Source/SymbolDefinitionToken.cpp 749
Code/Assembler/Source/SymbolTable.cpp 750
Code/Assembler/Source/SymbolToken.cpp 751

Code/Assembler/Source/Token.cpp 752
Code/Assembler/Source/UnconditionalJumpInstructionParser.cpp 753
Code/Assembler/Source/z85.cpp 754
Code/Assembler/Source/z85_impl.cpp 770
Code/Emulator/Include/AddExecuter.h 772
Code/Emulator/Include/Address.h 773
Code/Emulator/Include/AddressMode.h 525
Code/Emulator/Include/AddressSpace.h 774
Code/Emulator/Include/AluExecuter.h 775
Code/Emulator/Include/AndExecuter.h 776
Code/Emulator/Include/AslExecuter.h 777
Code/Emulator/Include/AsrExecuter.h 778
Code/Emulator/Include/AssemblerOutput.h 779
Code/Emulator/Include/CallExecuter.h 780
Code/Emulator/Include/CommandLineHelper.h 531
Code/Emulator/Include/CommonRegexes.h 533
Code/Emulator/Include/ConditionalJumpExecuter.h 781
Code/Emulator/Include/ConsoleInputOutput.h 782
Code/Emulator/Include/Context.h 783
Code/Emulator/Include/cxxopts.h 537
Code/Emulator/Include/DataType.h 543
Code/Emulator/Include/DivideExecuter.h 784
Code/Emulator/Include/Executer.h 785
Code/Emulator/Include/FileReader.h 553
Code/Emulator/Include/InstructionBitField.h 564
Code/Emulator/Include/InstructionBitFieldUnion.h 566
Code/Emulator/Include/InstructionCode.h 568
Code/Emulator/Include/IntExecuter.h 786
Code/Emulator/Include/JgezExecuter.h 787
Code/Emulator/Include/JgzExecuter.h 788
Code/Emulator/Include/JlezExecuter.h 789
Code/Emulator/Include/JlzExecuter.h 790
Code/Emulator/Include/JmpExecuter.h 791
Code/Emulator/Include/JnzExecuter.h 792
Code/Emulator/Include/JzExecuter.h 793
Code/Emulator/Include/KeyboardListener.h 794
Code/Emulator/Include/LoadExecuter.h 795
Code/Emulator/Include/MessageException.h 585
Code/Emulator/Include/ModuloExecuter.h 796
Code/Emulator/Include/MultiplyExecuter.h 797
Code/Emulator/Include/NotExecuter.h 798
Code/Emulator/Include/OperandType.h 597
Code/Emulator/Include/OrExecuter.h 799
Code/Emulator/Include/PopExecuter.h 800
Code/Emulator/Include/Processor.h 801
Code/Emulator/Include/PushExecuter.h 802
Code/Emulator/Include/Register.h 606
Code/Emulator/Include/RelocationRecord.h 615
Code/Emulator/Include/RetExecuter.h 803

Code/Emulator/Include/SectionData.h 620
Code/Emulator/Include/SectionType.h 625
Code/Emulator/Include/Segment.h 804
Code/Emulator/Include/StlHelper.h 805
Code/Emulator/Include/StoreExecuter.h 806
Code/Emulator/Include/StringHelper.h 630
Code/Emulator/Include/SubtractExecuter.h 807
Code/Emulator/Include/SymbolData.h 635
Code/Emulator/Include/TimerListener.h 808
Code/Emulator/Include/XorExecuter.h 809
Code/Emulator/Include/z85.h 649
Code/Emulator/Include/z85_cpp.h 656
Code/Emulator/Source/AddExecuter.cpp 810
Code/Emulator/Source/AddressSpace.cpp 811
Code/Emulator/Source/AluExecuter.cpp 812
Code/Emulator/Source/AndExecuter.cpp 813
Code/Emulator/Source/AslExecuter.cpp 814
Code/Emulator/Source/AsrExecuter.cpp 815
Code/Emulator/Source/AssemblerOutput.cpp 816
Code/Emulator/Source/CallExecuter.cpp 817
Code/Emulator/Source/CommandLineHelper.cpp 663
Code/Emulator/Source/ConditionalJumpExecuter.cpp 818
Code/Emulator/Source/ConsoleInputOutput.cpp 819
Code/Emulator/Source/Context.cpp 820
Code/Emulator/Source/DivideExecuter.cpp 821
Code/Emulator/Source/FileReader.cpp 676
Code/Emulator/Source/IntExecuter.cpp 822
Code/Emulator/Source/JgezExecuter.cpp 823
Code/Emulator/Source/JgzExecuter.cpp 824
Code/Emulator/Source/JlezExecuter.cpp 825
Code/Emulator/Source/JlzExecuter.cpp 826
Code/Emulator/Source/JmpExecuter.cpp 827
Code/Emulator/Source/JnzExecuter.cpp 828
Code/Emulator/Source/JzExecuter.cpp 829
Code/Emulator/Source/KeyboardListener.cpp 830
Code/Emulator/Source/LoadExecuter.cpp 831
Code/Emulator/Source/Main.cpp 700
Code/Emulator/Source/MessageException.cpp 704
Code/Emulator/Source/ModuloExecuter.cpp 832
Code/Emulator/Source/MultiplyExecuter.cpp 833
Code/Emulator/Source/NotExecuter.cpp 834
Code/Emulator/Source/OrExecuter.cpp 835
Code/Emulator/Source/PopExecuter.cpp 836
Code/Emulator/Source/Processor.cpp 837
Code/Emulator/Source/PushExecuter.cpp 838
Code/Emulator/Source/Register.cpp 839
Code/Emulator/Source/RelocationRecord.cpp 728
Code/Emulator/Source/RetExecuter.cpp 840
Code/Emulator/Source/SectionData.cpp 733

Code/Emulator/Source/Segment.cpp 841
Code/Emulator/Source/StoreExecuter.cpp 842
Code/Emulator/Source/StringHelper.cpp 741
Code/Emulator/Source/SubtractExecuter.cpp 843
Code/Emulator/Source/SymbolData.cpp 746
Code/Emulator/Source/TimerListener.cpp 844
Code/Emulator/Source/XorExecuter.cpp 845
Code/Emulator/Source/z85.cpp 762
Code/Emulator/Source/z85_impl.cpp 771

Namespace Documentation

bnssassembler Namespace Reference

Classes

- class **AddOperation**
- *Class implementing the behaviour of the + operator in expressions.* class **AddToken**
- ***Token** class representing the + operation.* class **AluInstructionParser**
- *Class representing the parser for ALU instructions.* class **AssemblerException**
- *Class representing the custom exception for the assembler.* class **ClosingBraceToken**
- ***Token** class representing the opening brace.* class **CommandLineHelper**
- *Utility class used to parse the command line.* class **ConditionalJumpInstructionParser**
- *Class representing the parser for conditional jump instructions.* class **Data**
- *Class representing the MicroRISC data.* class **DataDefinitionLineParser**
- *Class used for parsing data definitions.* class **DataDefinitionToken**
- *Class representing the data definition token.* class **DataTypeParser**
- *Utility class used for parsing data types.* class **DivideOperation**
- *Class implementing the behaviour of the / operator in expressions.* class **DivideToken**
- ***Token** class representing the / operation.* class **DivisionByZeroException**
- *Exception class representing division by zero.* class **Expression**
- *Class representing the math expression.* class **ExpressionBuilder**
- *Utility class used for building math expressions.* class **ExpressionToken**
- *Class representing the token found in infix and postfix expressions.* class **ExpressionTokenFactory**
- *Utility class used for creating the **ExpressionToken** objects.* class **FileReader**
- *Utility class providing methods for reading the file.* class **FileWriter**
- *Utility class used to write the assembler result to a file.* class **FirstPass**
- *Class representing the executor of the first pass.* class **FirstPassData**
- *Class representing the data that the two-pass assembler will modify in the first pass.* class **FirstPassException**
- *Represents an exception that happend during the assembler first pass.* class **GlobalSymbolsLineParser**
- *Class used for parsing information about global symbols.* class **GlobalSymbolsToken**
- *Class representing the global symbols token.* class **Immediate**
- *Class representing the immediate operand.* class **ImmediateParser**
- *Class representing the parser for the immediate operands.* class **IncorrectLabelException**
- *Exception representing the incorrect label.* struct **InstructionBitField**
- *Bit field that enables easier manipulation of instructions.* union **InstructionBitFieldUnion**
- *Union that enables easier manipulation of the instruction bit field.* class **InstructionCodeParser**
- *Utility class used for parsing instruction codes.* class **InstructionLineParser**
- *Class used for parsing instructions.* class **InstructionParser**
- *Abstract lass used for parsing one instruction.* class **InstructionToken**
- *Class representing the instruction in an assembler source file.* class **InterruptInstructionParser**
- *Class representing the parser for the interrupt instruction.* class **InvalidDataDefinitionException**
- *Exception representing invalid data definition.* class **InvalidDataTypeException**
- *Exception representing the invalid data type.* class **InvalidExpressionException**
- *Exception representing the invalid expression.* class **LabelToken**
- *Class representing the label token.* class **LineParser**
- *Chain of command abstract class used for parsing one line of file.* class **Literal**
- *Class representing the literal value.* class **LiteralToken**
- ***Token** class representing a math literal value.* class **LoadInstructionParser**
- *Class representing the load instruction parser.* class **MemoryDirect**
- *Class representing the memory direct operand.* class **MemoryDirectParser**
- *Class representing the parser for the memory direct operand.* class **MessageException**

- *Represents an exception with a string message.* class **MicroRiscExpression**
- *Adapter class for **Expression**.* class **MicroRiscParser**
- *Class representing the parser for the MicroRISC assembly.* class **MultiplyOperation**
- *Class implementing the behaviour of the * operator in expressions.* class **MultiplyToken**
- ***Token** class representing the * operation.* class **NonExistingSymbolException**
- *Exception representing the non existing symbol.* class **NoOperandInstructionParser**
- *Class representing the parser for the instruction without operands.* class **NotInstructionParser**
- *Class representing the parser for the not instruction.* class **OpeningBraceToken**
- ***Token** class representing the opening brace.* class **Operand**
- *Class representing one operand in an instruction.* class **OperandParser**
- *Chain of command class used to parse operands of the instructions.* class **Operation**
- *Class representing the mathematical operation with two operands.* class **OperationToken**
- ***Token** class representing a math operator.* class **OrgDirectiveLineParser**
- *Class representing a line parser for the origin directive.* class **OrgDirectiveToken**
- *Class representing the origin directive token.* class **Parser**
- *Abstract class representing a text parser.* class **ParserException**
- *Represents an exception that happend during the parsing of the file.* class **RegisterDirect**
- *Class representing the register direct operand.* class **RegisterDirectParser**
- *Class representing the parser for the register direct operand.* class **RegisterIndirect**
- *Class representing the register indirect operand.* class **RegisterIndirectOffset**
- *Class representing the register indirect operand with offset.* class **RegisterIndirectOffsetParser**
- *Class representing the parser for the register indirect operand with offset.* class **RegisterIndirectParser**
- *Class representing the parser for the register indirect operand.* class **RegisterParser**
- *Utility class used for parsing registers.* class **RelocationRecord**
- *Class representing one relocation record.* class **SecondPass**
- *Utility class executing the second pass.* class **SecondPassData**
- *Class representing the data that will be updated during the second pass.* class **SecondPassException**
- *Represents an exception that happened during the assembler second pass.* class **SectionData**
- *Class representing the data about one section.* class **SectionStartLineParser**
- *Class used for parsing section start definitions.* class **SectionStartToken**
- *Class representing the section start token.* class **SectionTable**
- *Class representing the table of sections.* class **SectionTypeParser**
- *Utility class representing the parser for the section types.* class **StackInstructionParser**
- *Class representing the parser for stack instructions.* class **StoreInstructionParser**
- *Class representing the parser for the store instruction.* class **StringHelper**
- *Utility class providing helper methods for std::string class.* class **SubtractOperation**
- *Class implementing the behaviour of the - operator in expressions.* class **SubtractToken**
- ***Token** class representing the - operation.* class **Symbol**
- *Class representing a symbol inside an expression.* class **SymbolData**
- *Class representing data about one symbol.* class **SymbolDefinition**
- *Class representing a symbol definition.* class **SymbolDefinitionLineParser**
- *Class used for parsing symbol definitions.* class **SymbolDefinitionToken**
- *Class representing the symbol definition token.* class **SymbolTable**
- *Class representing the symbol table.* class **SymbolToken**
- ***Token** class representing a math symbol.* class **Token**
- *Class representing one token of the assembler source file.* class **UndonditionalJumpInstructionParser**

Class representing the parser for the unconditional jump instructions. Enumerations

- enum **AddressMode** { **IMMEDIATE** = 0b100, **REGISTER_DIRECT** = 0b000, **MEMORY_DIRECT** = 0b110, **REGISTER_INDIRECT** = 0b010, **REGISTER_INDIRECT_OFFSET** = 0b111 }*Enum representing the address mode.*
- enum **DataType** { **DOUBLE_WORD** = 0, **WORD**, **BYTE** }*Enum representing a data type.*

- enum **InstructionCode** : int8_t { **INT** = 0x00, **JMP** = 0x02, **CALL** = 0x03, **RET** = 0x01, **JZ** = 0x04, **JNZ** = 0x05, **JGZ** = 0x06, **JGEZ** = 0x07, **JLZ** = 0x08, **JLEZ** = 0x09, **LOAD** = 0x10, **STORE** = 0x11, **PUSH** = 0x20, **POP** = 0x21, **ADD** = 0x30, **SUB** = 0x31, **MUL** = 0x32, **DIV** = 0x33, **MOD** = 0x34, **AND** = 0x35, **OR** = 0x36, **XOR** = 0x37, **NOT** = 0x38, **ASL** = 0x39, **ASR** = 0x3A }*Enum representing the instruction code.*
- enum **OperandType** : int8_t { **DEFAULT** = 0b000, **UNSIGNED_BYTE** = 0b011, **SIGNED_BYTE** = 0b111, **REGULAR_BYTE** = 0b111, **UNSIGNED_WORD** = 0b001, **SIGNED_WORD** = 0b101, **REGULAR_WORD** = 0b101, **REGULAR_DOUBLE_WORD** = 0b000 }*Enum representing the operand type.*
- enum **Register** { **R0** = 0x00, **R1**, **R2**, **R3**, **R4**, **R5**, **R6**, **R7**, **R8**, **R9**, **R10**, **R11**, **R12**, **R13**, **R14**, **R15**, **SP** = 0x10, **PC** = 0x11, **NONE** = 0x1F }*Enum representing a register.*
- enum **SectionType** : int8_t { **TEXT** = 0, **DATA**, **RODATA**, **BSS** }*Enum representing the type of the section.*

Functions

- std::string **multiple** (unsigned char c, size_t times)
  Returns a string containing multiple of the same characters.
- std::string **multiple** (std::string s, size_t times)
  Returns a string containing multiple of the same strings.
- static void **split** (std::list< **RelocationRecord** > &original, std::list< **RelocationRecord** > &left, std::list< **RelocationRecord** > &right)
- **Data parseData** (std::string str)
  Parses the data from the string.
- static void **fixUnaryMinusStart** (std::string &infix_expression, std::regex token_extractor)
  Fixes the expression that starts with an unary minus sign.
- static std::list< std::shared_ptr< **ExpressionToken** > > **infixToPostfix** (std::string infix_expression)
  Builds a postfix expression from the infix string.
- static std::shared_ptr< **Expression** > **postfixToTree** (const std::list< std::shared_ptr< **ExpressionToken** >> &postfix_expression)
  Builds a tree from the postfix expression.
- static void **loadStoreFixup** (std::string &instruction, **OperandType** &type)
  Hack to fix the load and store instructions which can have various operands.
- static void **stripComment** (std::string &line, std::vector< std::string > one_line_comment_delimiters)
  Strips the comment from one line of the file.
- static std::string **extractLabel** (std::string &line, std::vector< std::string > label_delimiters)
  Extracts the label (if any) from the line.
- std::shared_ptr< **Operand** > **parsePcrel** (std::string str)
  Parses the input as a PC relative address.
- std::ostream & **operator<<** (std::ostream &os, const **RelocationRecord** &record)
- bool **operator==** (const **RelocationRecord** &lhs, const **RelocationRecord** &rhs)
- bool **operator!=** (const **RelocationRecord** &lhs, const **RelocationRecord** &rhs)
- std::ostream & **operator<<** (std::ostream &os, const **SecondPassData** &data)
- bool **operator==** (const **SectionData** &lhs, const **SectionData** &rhs) noexcept
- bool **operator!=** (const **SectionData** &lhs, const **SectionData** &rhs) noexcept
- bool **operator<** (const **SectionData** &lhs, const **SectionData** &rhs) noexcept
- bool **operator>** (const **SectionData** &lhs, const **SectionData** &rhs) noexcept
- bool **operator<=** (const **SectionData** &lhs, const **SectionData** &rhs) noexcept
- bool **operator>=** (const **SectionData** &lhs, const **SectionData** &rhs) noexcept
- static std::string **name** (**SectionType** type, bool indexed, size_t index)
- static void **writeDescription** (**SectionType** type, bool indexed, size_t index, bool org_valid, **uint32_t** org_address, size_t size)
- std::ostream & **operator<<** (std::ostream &os, const **SectionData** &data)

- std::ostream & **operator<<** (std::ostream &os, const **SectionTable** &section_table)
- static void **generateMaps** (const std::list< **RelocationRecord** > &source, std::unordered_map< size_t, std::pair< **RelocationRecord**, size_t >> &sections, std::unordered_map< std::string, std::pair< **RelocationRecord**, size_t >> &symbols)
- static void **exchange** (std::list< **RelocationRecord** > &left, std::list< **RelocationRecord** > &right)
- std::ostream & **operator<<** (std::ostream &os, const **SymbolData** &data)
- bool **operator==** (const **SymbolDefinition** &lhs, const **SymbolDefinition** &rhs)
- bool **operator!=** (const **SymbolDefinition** &lhs, const **SymbolDefinition** &rhs)
- bool **operator<** (const **SymbolDefinition** &lhs, const **SymbolDefinition** &rhs)
- bool **operator>** (const **SymbolDefinition** &lhs, const **SymbolDefinition** &rhs)
- bool **operator<=** (const **SymbolDefinition** &lhs, const **SymbolDefinition** &rhs)
- bool **operator>=** (const **SymbolDefinition** &lhs, const **SymbolDefinition** &rhs)
- std::ostream & **operator<<** (std::ostream &os, const **SymbolTable** &table)

Variables

- const std::string **ZERO** = "0"
- const std::string **DECIMAL** = "[1-9][0-9]*"
- const std::string **HEX** = "0x[0-9a-fA-F][0-9a-fA-F]*"
- const std::string **BINARY** = "0b[01][01]*"
- const std::string **OCT** = "0[0-7][0-7]*"
- const std::string **CHARACTER** = "'[[:print:]]'"
- const std::string **LITERAL** = "(" + **ZERO** + "|" + **DECIMAL** + "|" + **HEX** + "|" + **BINARY** + "|" + **OCT** + "|" + **CHARACTER** + ")"
- const std::string **OPERATOR** = "[-+*/()]"
- const std::string **SYMBOL** = "(([a-zA-Z_][a-zA-Z_0-9]*)|\\$)"
- const std::string **LABEL** = **SYMBOL**
- const std::string **CONSTANT_TERM** = "([[:space:]]*(" + LITERAL + "|" + **OPERATOR** + "|" + **SYMBOL** + ")[[:space:]]*)*"
- const std::string **ORG_DIRECTIVE** = "[Oo][Rr][Gg]"
- const std::string **SYMBOL_DEFINITION** = "[Dd][Ee][Ff]"
- const std::string **DUPLICATE_DIRECTIVE** = "[Dd][Uu][Pp]"
- const std::string **GLOBAL_DIRECTIVE** = "[.][Gg][Ll][Oo][Bb][Aa][Ll]"
- const std::string **COMMA_TOKENIZER** = "[[:space:]]*(.*?)[[:space:]]*,(.*)"
- const std::string **LAST_COMMA_TOKEN** = "[[:space:]]*(.*)[[:space:]]*"
- const std::regex **ZERO_REGEX** = std::regex(**ZERO**)
- const std::regex **DECIMAL_REGEX** = std::regex(**DECIMAL**)
- const std::regex **HEX_REGEX** = std::regex(**HEX**)
- const std::regex **BINARY_REGEX** = std::regex(**BINARY**)
- const std::regex **OCT_REGEX** = std::regex(**OCT**)
- const std::regex **CHARACTER_REGEX** = std::regex(**CHARACTER**)
- const std::regex **LITERAL_REGEX** = std::regex(**LITERAL**)
- const std::regex **OPERATOR_REGEX** = std::regex(**OPERATOR**)
- const std::regex **SYMBOL_REGEX** = std::regex(**SYMBOL**)
- const std::regex **LABEL_REGEX** = std::regex(**LABEL**)
- const std::regex **CONSTANT_TERM_REGEX** = std::regex(**CONSTANT_TERM**)
- const std::regex **ORG_DIRECTIVE_REGEX** = std::regex(**ORG_DIRECTIVE**)
- const std::regex **SYMBOL_DEFINITION_REGEX** = std::regex(**SYMBOL_DEFINITION**)
- const std::regex **DUPLICATE_DIRECTIVE_REGEX** = std::regex(**DUPLICATE_DIRECTIVE**)
- const std::regex **GLOBAL_DIRECTIVE_REGEX** = std::regex(**GLOBAL_DIRECTIVE**)
- const std::regex **COMMA_TOKENIZER_REGEX** = std::regex(**COMMA_TOKENIZER**)
- const std::regex **LAST_COMMA_TOKEN_REGEX** = std::regex(**LAST_COMMA_TOKEN**)
- const std::string **UPPER_LEFT** = "\u2554"
- const std::string **UPPER_RIGHT** = "\u2557"
- const std::string **LOWER_LEFT** = "\u255a"
- const std::string **LOWER_RIGHT** = "\u255d"
- const std::string **HORIZONTAL** = "\u2550"

- const std::string **VERTICAL** = "\u2551"
- const std::string **T_LEFT** = "\u2563"
- const std::string **T_RIGHT** = "\u2560"
- const std::string **T_UP** = "\u2569"
- const std::string **T_DOWN** = "\u2566"
- const std::string **ALL_FOUR** = "\u256c"
- const size_t **NUM_OF_REGISTERS** = 16
  Number of all purpose registers (excluding PC and SP)

Enumeration Type Documentation

enum bnssassembler::AddressMode

Enum representing the address mode.

Enumerator:

IMMEDIATE	
REGISTER_DIRECT	
MEMORY_DIRECT	
REGISTER_INDIRECT	
REGISTER_INDIRECT_OFFSET	

Definition at line 9 of file AddressMode.h.

```
                     {
        IMMEDIATE                   = 0b100,
        REGISTER_DIRECT             = 0b000,
        MEMORY_DIRECT               = 0b110,
        REGISTER_INDIRECT           = 0b010,
        REGISTER_INDIRECT_OFFSET    = 0b111
    };
```

enum bnssassembler::DataType

Enum representing a data type.

Enumerator:

DOUBLE_WORD	32bit value
WORD	16bit value
BYTE	8bit value

Definition at line 9 of file DataType.h.

```
                  {
        DOUBLE_WORD = 0,
        WORD,
        BYTE
    };
```

enum bnssassembler::InstructionCode : int8_t

Enum representing the instruction code.

Enumerator:

INT	
JMP	
CALL	
RET	
JZ	
JNZ	
JGZ	
JGEZ	
JLZ	
JLEZ	
LOAD	
STORE	
PUSH	
POP	
ADD	
SUB	
MUL	
DIV	
MOD	
AND	
OR	
XOR	
NOT	
ASL	
ASR	

Definition at line 12 of file InstructionCode.h.

```
                             : int8_t {
        INT   = 0x00,
        JMP   = 0x02,
        CALL  = 0x03,
        RET   = 0x01,
        JZ    = 0x04,
        JNZ   = 0x05,
        JGZ   = 0x06,
        JGEZ  = 0x07,
        JLZ   = 0x08,
        JLEZ  = 0x09,

        LOAD  = 0x10,
        STORE = 0x11,

        PUSH  = 0x20,
        POP   = 0x21,

        ADD   = 0x30,
        SUB   = 0x31,
        MUL   = 0x32,
        DIV   = 0x33,
        MOD   = 0x34,
        AND   = 0x35,
        OR    = 0x36,
        XOR   = 0x37,
        NOT   = 0x38,
        ASL   = 0x39,
        ASR   = 0x3A
    };
```

enum bnssassembler::OperandType : int8_t

Enum representing the operand type.

Enumerator:

DEFAULT	
UNSIGNED_BYTE	
SIGNED_BYTE	
REGULAR_BYTE	
UNSIGNED_WORD	
SIGNED_WORD	
REGULAR_WORD	
REGULAR_DOUBLE_WORD	

Definition at line 10 of file OperandType.h.

```
   10                     : int8_t {
   11         DEFAULT                  = 0b000,
   12         UNSIGNED_BYTE            = 0b011,
   13         SIGNED_BYTE              = 0b111,
   14         REGULAR_BYTE             = 0b111,
   15         UNSIGNED_WORD            = 0b001,
   16         SIGNED_WORD              = 0b101,
   17         REGULAR_WORD             = 0b101,
   18         REGULAR_DOUBLE_WORD      = 0b000
   19     };
```

enum bnssassembler::Register

Enum representing a register.

Enumerator:

R0	
R1	
R2	
R3	
R4	
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
R13	
R14	
R15	
SP	
PC	
NONE	

Definition at line 16 of file Register.h.

```
                 {
        R0 = 0x00,
        R1,
        R2,
        R3,
        R4,
        R5,
        R6,
        R7,
        R8,
        R9,
        R10,
        R11,
        R12,
        R13,
        R14,
        R15,
        SP = 0x10,
        PC = 0x11,
        NONE = 0x1F
    };
```

enum bnssassembler::SectionType : int8_t

Enum representing the type of the section.

Enumerator:

TEXT	Text section
DATA	**Data** section
RODATA	Read-only data section
BSS	Block started by symbol section

Definition at line 11 of file SectionType.h.

```
                    : int8_t {
        TEXT = 0,
        DATA,
        RODATA,
        BSS
    };
```

Function Documentation

static void bnssassembler::exchange (std::list< RelocationRecord > & *left*, std::list< RelocationRecord > & *right*) `[static]`

Definition at line 39 of file SubtractOperation.cpp.

Referenced by bnssassembler::SubtractOperation::generateRelocations().

```
{
        std::list<RelocationRecord> for_right;
        std::list<RelocationRecord> for_left;

        for (auto &element : left) {
            if (element.opposite()) {
                for_right.push_back(element);
            }
```

```
   47         }
   48
   49         for (auto &element : right) {
   50             if (element.opposite()) {
   51                 for_left.push_back(element);
   52             }
   53         }
   54
   55         for (auto &element : for_right) {
   56             element.toggleOpposite();
   57             right.push_back(element);
   58             left.remove(element);
   59         }
   60
   61         for (auto &element : for_left) {
   62             element.toggleOpposite();
   63             left.push_back(element);
   64             right.remove(element);
   65         }
   66     }
```

static std::string bnssassembler::extractLabel (std::string & *line*, std::vector< std::string > *label_delimiters*)`[static]`

Extracts the label (if any) from the line.

Parameters:

line	Reference to the line. After this method does not contain the label
label_delimiters	Strings that separate the label from the rest of the line

Definition at line 30 of file Parser.cpp.

References bnssassembler::StringHelper::join(), and LABEL.

Referenced by bnssassembler::Parser::parse().

```
   30
{
   31         auto delimiters = StringHelper::join(label_delimiters, "|");
   32         static std::regex regex("[[:space:]]*(.*)[[:space:]]*(" + delimiters +
")[[:space:]]*(.*)[[:space:]]*");
   33         static std::regex characters_fix(".*'" + delimiters + "'.*");
   34
   35         if (regex_match(line, regex) && !regex_match(line, characters_fix)) {
   36             static std::regex label_regex("[[:space:]]*(" + LABEL +
")[[:space:]]*");
   37             auto ret = regex_replace(line, regex, "$1");
   38             if (regex_match(ret, label_regex)) {
   39                 ret = regex_replace(ret, label_regex, "$1");
   40             }
   41             else {
   42                 throw IncorrectLabelException(ret);
   43             }
   44
   45             line = regex_replace(line, regex, "$3");
   46             return ret;
   47         }
   48
   49         return "";
   50
   51     }
```

static void bnssassembler::fixUnaryMinusStart (std::string & *infix_expression*, std::regex *token_extractor*)`[static]`

Fixes the expression that starts with an unary minus sign.

Parameters:

infix_expression	Reference to the expression that will be fixed
token_extractor	Regex used to extract the first token from the string

Definition at line 16 of file ExpressionBuilder.cpp.

Referenced by infixToPostfix().

```
{
        if (regex_match(infix_expression, token_extractor)) {
            auto token_string = regex_replace(infix_expression,
token_extractor, "$1");
            if (token_string == "-") {
                infix_expression = "0 " + infix_expression;
            }
        }
    }
```

static void bnssassembler::generateMaps (const std::list< RelocationRecord > & *source*, std::unordered_map< size_t, std::pair< RelocationRecord, size_t >> & *sections*, std::unordered_map< std::string, std::pair< RelocationRecord, size_t >> & *symbols*)`[static]`

Definition at line 18 of file SubtractOperation.cpp.

Referenced by bnssassembler::SubtractOperation::generateRelocations().

```
{
        for (auto &element : source) {
            if (element.section()) {
                if (sections.count(element.sectionIndex()) > 0) {
                    sections[element.sectionIndex()].second++;
                }
                else {
                    sections[element.sectionIndex()] =
std::make_pair(element, 1);
                }
            }
            else {
                if (symbols.count(element.symbolName()) > 0) {
                    symbols[element.symbolName()].second++;
                }
                else {
                    symbols[element.symbolName()] = std::make_pair(element,
1);
                }
            }
        }
    }
```

static std::list<std::shared_ptr<ExpressionToken> > bnssassembler::infixToPostfix (std::string *infix_expression*)`[static]`

Builds a postfix expression from the infix string.

Parameters:

infix_expression	Infix expression string

Returns:

Postfix expression list of tokens

Definition at line 30 of file ExpressionBuilder.cpp.

References bnssassembler::ExpressionTokenFactory::create(), fixUnaryMinusStart(), LITERAL, OPERATOR, bnssassembler::ExpressionBuilder::popToPostfix(), and SYMBOL.

Referenced by bnssassembler::ExpressionBuilder::build().

```
    {
        std::list<std::shared_ptr<ExpressionToken>> ret;
        std::stack<std::shared_ptr<ExpressionToken>> stack;
        auto rank = 0;

        static std::regex end_of_infix("[[:space:]]*");
        static std::regex token_extractor("[[:space:]]*(" + LITERAL + "|" + OPERATOR + "|" + SYMBOL + ")(.*)[[:space:]]*");

        fixUnaryMinusStart(infix_expression, token_extractor);

        while (true) {
            if (infix_expression.size() == 0 || regex_match(infix_expression, end_of_infix)) {
                break;
            }

            if (!regex_match(infix_expression, token_extractor)) {
                throw InvalidExpressionException();
            }

            auto token_string = regex_replace(infix_expression, token_extractor, "$1");
            infix_expression = regex_replace(infix_expression, token_extractor, "$5");
            auto token = ExpressionTokenFactory::create(token_string);
            token->process(ret, stack, rank);
        }

        while (!stack.empty()) {
            ExpressionBuilder::popToPostfix(ret, stack, rank);
        }

        if (rank != 1) {
            throw MessageException("Invalid expression - too many operands");
        }

        return ret;
    }
```

static void bnssassembler::loadStoreFixup (std::string & *instruction*, OperandType & *type*) `[static]`

Hack to fix the load and store instructions which can have various operands.

Parameters:

instruction	String that should be fixed
type	**Operand** type of the instruction that should be set

Load and store instructions can have suffices UB, SB, UW, SW, B and W. Those are still the same instructions with the same instruction codes, but different operand types. This function fixes the string containing the instruction, making it look like it is a regular load or store, but sets the operand type to the specific type. If the instruction is not load or store, this function does nothing

Definition at line 29 of file InstructionLineParser.cpp.

References REGULAR_BYTE, REGULAR_DOUBLE_WORD, REGULAR_WORD, SIGNED_BYTE, SIGNED_WORD, UNSIGNED_BYTE, and UNSIGNED_WORD.

Referenced by bnssassembler::InstructionLineParser::parse().

```
   29                                                                       {
   30         transform(instruction.begin(), instruction.end(), instruction.begin(), tolower);
   31         if (instruction == "loadub") {
   32             instruction = "load";
   33             type = UNSIGNED_BYTE;
   34         }
   35         else if (instruction == "loadsb") {
```

```
   36                 instruction = "load";
   37                 type = SIGNED_BYTE;
   38             }
   39             else if (instruction == "loaduw") {
   40                 instruction = "load";
   41                 type = UNSIGNED_WORD;
   42             }
   43             else if (instruction == "loadsw") {
   44                 instruction = "load";
   45                 type = SIGNED_WORD;
   46             }
   47             else if (instruction == "load") {
   48                 type = REGULAR_DOUBLE_WORD;
   49             }
   50             else if (instruction == "storeb") {
   51                 instruction = "store";
   52                 type = REGULAR_BYTE;
   53             }
   54             else if (instruction == "storew") {
   55                 instruction = "store";
   56                 type = REGULAR_WORD;
   57             }
   58             else if (instruction == "store") {
   59                 type = REGULAR_DOUBLE_WORD;
   60             }
   61         }
```

std::string bnssassembler::multiple (unsigned char *c*, size_t *times*)`[inline]`

Returns a string containing multiple of the same characters.

Parameters:

c	Character
times	Number of times this character should be in the string

Returns:

String containing all the characters

Definition at line 37 of file PrintHelpers.h.

Referenced by operator<<(), and writeDescription().

```
   37                                                                    {
   38             std::string ret;
   39             for (size_t i = 0; i < times; i++) {
   40                 ret += c;
   41             }
   42 
   43             return ret;
   44         }
```

std::string bnssassembler::multiple (std::string *s*, size_t *times*)`[inline]`

Returns a string containing multiple of the same strings.

Parameters:

s	String
times	Number of times this string should be in the returning string

Returns:

String containing all the strings

Definition at line 52 of file PrintHelpers.h.

```
   52                                                                {
   53             std::string ret;
   54             for (size_t i = 0; i < times; i++) {
   55                 ret += s;
   56             }
```

```

    return ret;
}
```

static std::string bnssassembler::name (SectionType *type*, bool *indexed*, size_t *index*) `[static]`

Definition at line 123 of file SectionData.cpp.

References BSS, DATA, RODATA, and TEXT.

Referenced by cxxopts::OptionDetails::as(), cxxopts::Options::operator[](), cxxopts::OptionAdder::OptionAdder(), bnssassembler::SymbolDefinitionLineParser::parse(), cxxopts::Options::parse(), bnssassembler::MicroRiscExpression::setValue(), and writeDescription().

```
                                                            {
    std::string ret(" .");
    switch (type) {
    case TEXT:
        ret += "text";
        break;
    case DATA:
        ret += "data";
        break;
    case RODATA:
        ret += "rodata";
        break;
    case BSS:
        ret += "bss";
        break;
    default:
        break;
    }

    if (indexed) {
        ret += "." + std::to_string(index);
    }

    return ret;
}
```

bool bnssassembler::operator!= (const SymbolDefinition & *lhs*, const SymbolDefinition & *rhs*)

Definition at line 19 of file SymbolDefinition.cpp.

```
{
    return !(lhs == rhs);
}
```

bool bnssassembler::operator!= (const RelocationRecord & *lhs*, const RelocationRecord & *rhs*)

Definition at line 69 of file RelocationRecord.cpp.

```
{
    return !(lhs == rhs);
}
```

bool bnssassembler::operator!= (const SectionData & *lhs*, const SectionData & *rhs*) `[noexcept]`

Definition at line 83 of file SectionData.cpp.

```
{
   84         return !(lhs == rhs);
   85     }
```

bool bnssassembler::operator< (const SymbolDefinition & *lhs*, const SymbolDefinition & *rhs*)

Definition at line 23 of file SymbolDefinition.cpp.

References bnssassembler::SymbolDefinition::name_.

```
   23
{
   24         return lhs.name_ < rhs.name_;
   25     }
```

bool bnssassembler::operator< (const SectionData & *lhs*, const SectionData & *rhs*)`[noexcept]`

Definition at line 87 of file SectionData.cpp.

```
   87
{
   88         if (lhs.type_ < rhs.type_) {
   89             return true;
   90         }
   91
   92         if (lhs.type_ > rhs.type_) {
   93             return false;
   94         }
   95
   96         if (!lhs.indexed_ && rhs.indexed_) {
   97             return true;
   98         }
   99
  100         if (lhs.indexed_ && !rhs.indexed_) {
  101             return false;
  102         }
  103
  104         if (lhs.indexed_) {
  105             return lhs.index_ < rhs.index_;
  106         }
  107
  108         return false;
  109     }
```

std::ostream& bnssassembler::operator<< (std::ostream & *os*, const SymbolTable & *table*)

Parameters:

os	Stream where the content will be written
table	**Data** that will be written

Definition at line 22 of file SymbolTable.cpp.

References ALL_FOUR, HORIZONTAL, LOWER_LEFT, LOWER_RIGHT, multiple(), T_DOWN, T_LEFT, T_RIGHT, T_UP, UPPER_LEFT, UPPER_RIGHT, and VERTICAL.

```
   22                                                                    {
   23         std::cout << UPPER_LEFT << multiple(HORIZONTAL, 81) << UPPER_RIGHT <<
std::endl;
   24         std::cout << VERTICAL << UPPER_LEFT << multiple(HORIZONTAL, 79) <<
UPPER_RIGHT << VERTICAL << std::endl;
   25         std::cout << VERTICAL << VERTICAL << std::setw(79) << std::left << "
Symbol table:" << VERTICAL << VERTICAL << std::endl;
   26         std::cout << VERTICAL << LOWER_LEFT << multiple(HORIZONTAL, 79) <<
LOWER_RIGHT << VERTICAL << std::endl;
   27         std::cout << T_RIGHT << multiple(HORIZONTAL, 47) << T_DOWN <<
multiple(HORIZONTAL, 9) << T_DOWN << multiple(HORIZONTAL, 8) << T_DOWN <<
multiple(HORIZONTAL, 14) << T_LEFT << std::endl;
```

```
   28         std::cout << VERTICAL << "                     Name
" << VERTICAL << " Section " << VERTICAL << " Offset " << VERTICAL << " Global/Local
" << VERTICAL << std::endl;
   29         std::cout << T_RIGHT << multiple(HORIZONTAL, 47) << ALL_FOUR <<
multiple(HORIZONTAL, 9) << ALL_FOUR << multiple(HORIZONTAL, 8) << ALL_FOUR <<
multiple(HORIZONTAL, 14) << T_LEFT << std::endl;
   30
   31         os << table.size() << std::endl;
   32         for (auto &entry : table) {
   33             os << entry.second << std::endl;
   34         }
   35
   36         std::cout << LOWER_LEFT << multiple(HORIZONTAL, 47) << T_UP <<
multiple(HORIZONTAL, 9) << T_UP << multiple(HORIZONTAL, 8) << T_UP <<
multiple(HORIZONTAL, 14) << LOWER_RIGHT << std::endl;
   37
   38         return os;
   39     }
```

std::ostream& bnssassembler::operator<< (std::ostream & *os*, const SymbolData & *data*)

Parameters:

os	Stream where the content will be written
data	**Data** that will be written

Definition at line 30 of file SymbolData.cpp.

References bnssassembler::SymbolData::local_, bnssassembler::SymbolData::name_, bnssassembler::SymbolData::offset_, bnssassembler::SymbolData::section_index_, and VERTICAL.

```
   30                                                                     {
   31         os << data.name_ << std::endl;
   32         os << data.section_index_ << std::endl;
   33         os << data.offset_ << std::endl;
   34         os << data.local_ << std::endl;
   35
   36         std::cout << VERTICAL << " " << std::setw(46) << std::left << data.name_
<< VERTICAL << " " << std::setw(8) << std::left << data.section_index_ << VERTICAL <<
" " << std::setw(7) << std::left << data.offset_ << VERTICAL << std::setw(14) <<
std::left << (data.local_ ? " Local" : " Global") << VERTICAL << std::endl;
   37
   38         return os;
   39     }
```

std::ostream& bnssassembler::operator<< (std::ostream & *os*, const RelocationRecord & *record*)

Parameters:

os	Stream where the content will be written
record	**Data** that will be written

Definition at line 39 of file RelocationRecord.cpp.

References bnssassembler::RelocationRecord::absolute_, bnssassembler::RelocationRecord::offset_, bnssassembler::RelocationRecord::section_, bnssassembler::RelocationRecord::section_index_, bnssassembler::RelocationRecord::symbol_name_, and VERTICAL.

```
   39
{
   40         os << record.offset_ << std::endl;
   41         os << record.absolute_ << std::endl;
   42         os << record.section_ << std::endl;
   43         if (record.section_) {
   44             os << record.section_index_ << std::endl;
   45         }
   46         else {
   47             os << record.symbol_name_ << std::endl;
   48         }
```

```
        std::cout << VERTICAL << " " << std::setw(7) << std::left << 
record.offset_ << VERTICAL << " " << (record.absolute_ ? "Absolute" : "Relative") << 
" " << VERTICAL << " ";
        if (record.section_) {
            std::cout << std::setw(8) << std::left << 
std::to_string(record.section_index_) + "." << VERTICAL << std::setw(51) << " " << 
VERTICAL << std::endl;
        }
        else {
            std::cout << std::setw(8) << " " << VERTICAL << std::setw(51) << 
std::left << record.symbol_name_ << VERTICAL << std::endl;
        }

        return os;
    }
```

std::ostream& bnssassembler::operator<< (std::ostream & *os*, const SectionTable & *section_table*)

Parameters:

os	Stream where the content will be written
section_table	**Data** that will be written

Definition at line 53 of file SectionTable.cpp.

References HORIZONTAL, LOWER_LEFT, LOWER_RIGHT, multiple(), UPPER_LEFT, UPPER_RIGHT, and VERTICAL.

```
{
        os << section_table.size() << std::endl;

        std::cout << UPPER_LEFT << multiple(HORIZONTAL, 81) << UPPER_RIGHT << 
std::endl;
        std::cout << VERTICAL << UPPER_LEFT << multiple(HORIZONTAL, 79) << 
UPPER_RIGHT << VERTICAL << std::endl;
        std::cout << VERTICAL << VERTICAL << std::setw(79) << std::left << " 
Section table:" << VERTICAL << VERTICAL << std::endl;
        std::cout << VERTICAL << LOWER_LEFT << multiple(HORIZONTAL, 79) << 
LOWER_RIGHT << VERTICAL << std::endl;
        std::cout << LOWER_LEFT << multiple(HORIZONTAL, 81) << LOWER_RIGHT << 
std::endl;

        for (auto &section : section_table) {
            os << section << std::endl;
        }

        std::cout << std::endl << std::endl;

        return os;
    }
```

std::ostream& bnssassembler::operator<< (std::ostream & *os*, const SecondPassData & *data*)

Parameters:

os	Stream where the content will be written
data	**Data** that will be written

Definition at line 83 of file SecondPassData.cpp.

References HORIZONTAL, bnssassembler::SecondPassData::imported_symbols_, LOWER_LEFT, LOWER_RIGHT, multiple(), bnssassembler::SecondPassData::section_table_, bnssassembler::SecondPassData::symbol_table_, T_LEFT, T_RIGHT, UPPER_LEFT, UPPER_RIGHT, and VERTICAL.

```
                                                                    {
        os << data.imported_symbols_.size() << std::endl;

```

```
   86         std::cout << UPPER_LEFT << multiple(HORIZONTAL, 81) << UPPER_RIGHT << std::endl;
   87         std::cout << VERTICAL << UPPER_LEFT << multiple(HORIZONTAL, 79) << UPPER_RIGHT << VERTICAL << std::endl;
   88         std::cout << VERTICAL << VERTICAL << std::setw(79) << std::left << " Imported symbols:" << VERTICAL << VERTICAL << std::endl;
   89         std::cout << VERTICAL << LOWER_LEFT << multiple(HORIZONTAL, 79) << LOWER_RIGHT << VERTICAL << std::endl;
   90         std::cout << T_RIGHT << multiple(HORIZONTAL, 81) << T_LEFT << std::endl;
   91 
   92         for (auto &symbol : data.imported_symbols_) {
   93             os << symbol << std::endl;
   94             std::cout << VERTICAL << " " << std::setw(80) << std::left << symbol << VERTICAL << std::endl;
   95         }
   96 
   97         std::cout << LOWER_LEFT << multiple(HORIZONTAL, 81) << LOWER_RIGHT << std::endl << std::endl << std::endl;
   98         os << data.section_table_ << std::endl;
   99         os << data.symbol_table_ << std::endl;
  100 
  101         return os;
  102     }
```

std::ostream& bnssassembler::operator<< (std::ostream & *os*, const SectionData & *data*)

Parameters:

os	Stream where the content will be written
data	**Data** that will be written

Definition at line 156 of file SectionData.cpp.

References ALL_FOUR, bnssassembler::SectionData::data_, HORIZONTAL, bnssassembler::SectionData::index_, bnssassembler::SectionData::indexed_, bnssassembler::SectionData::location_counter_, LOWER_LEFT, LOWER_RIGHT, multiple(), bnssassembler::StringHelper::numberFormat(), bnssassembler::SectionData::org_address_, bnssassembler::SectionData::org_valid_, bnssassembler::SectionData::relocation_records_, T_DOWN, T_LEFT, T_RIGHT, T_UP, bnssassembler::StringHelper::toHexString(), bnssassembler::SectionData::type_, VERTICAL, and writeDescription().

```
  156                                                                    {
  157         os << data.type_ << std::endl;
  158         os << data.indexed_ << std::endl;
  159         if (data.indexed_) {
  160             os << data.index_ << std::endl;
  161         }
  162 
  163         os << data.org_valid_ << std::endl;
  164         if (data.org_valid_) {
  165             os << data.org_address_ << std::endl;
  166         }
  167 
  168         os << data.location_counter_ << std::endl;
  169         os << data.data_.size() << std::endl;
  170         for (auto &entry : data.data_) {
  171             os << StringHelper::numberFormat(entry) << std::endl;
  172         }
  173 
  174         writeDescription(data.type_, data.indexed_, data.index_, data.org_valid_, data.org_address_, data.location_counter_);
  175 
  176         std::cout << VERTICAL << " ";
  177 
  178         size_t i;
  179         for (i = 0; i < data.data_.size(); i++) {
  180             auto entry = data.data_[i];
  181             if (i % 16 == 0 && i != 0) {
  182                 std::cout << VERTICAL << std::endl << VERTICAL << " ";
  183             }
  184 
  185             std::cout << StringHelper::toHexString(entry) << " ";
  186         }
```

```

        for (; i % 16 != 0 || i == 0; i++) {
            std::cout << "     ";
        }

        std::cout << VERTICAL << std::endl;

        std::cout << T_RIGHT << multiple(HORIZONTAL, 81) << T_LEFT << std::endl;
        std::cout << VERTICAL << std::setw(81) << std::left << " Relocation table:" << VERTICAL << std::endl;
        std::cout << T_RIGHT << multiple(HORIZONTAL, 8) << T_DOWN << multiple(HORIZONTAL, 10) << T_DOWN << multiple(HORIZONTAL, 9) << T_DOWN << multiple(HORIZONTAL, 51) << T_LEFT << std::endl;

        std::cout << VERTICAL << " Offset " << VERTICAL << " Absolute " << VERTICAL << " Section " << VERTICAL << "                      Symbol                        " << VERTICAL << std::endl;
        std::cout << T_RIGHT << multiple(HORIZONTAL, 8) << ALL_FOUR << multiple(HORIZONTAL, 10) << ALL_FOUR << multiple(HORIZONTAL, 9) << ALL_FOUR << multiple(HORIZONTAL, 51) << T_LEFT << std::endl;

        os << data.relocation_records_.size() << std::endl;
        for (auto &record : data.relocation_records_) {
            os << record << std::endl;
        }

        std::cout << LOWER_LEFT << multiple(HORIZONTAL, 8) << T_UP << multiple(HORIZONTAL, 10) << T_UP << multiple(HORIZONTAL, 9) << T_UP << multiple(HORIZONTAL, 51) << LOWER_RIGHT << std::endl;

        return os;
    }
```

bool bnssassembler::operator<= (const SymbolDefinition & ***lhs*****, const SymbolDefinition &** ***rhs*****)**

Definition at line 31 of file SymbolDefinition.cpp.

```
{
        return !(lhs > rhs);
    }
```

bool bnssassembler::operator<= (const SectionData & ***lhs*****, const SectionData &** ***rhs*****)`[noexcept]`**

Definition at line 115 of file SectionData.cpp.

```
{
        return !(lhs > rhs);
    }
```

bool bnssassembler::operator== (const SymbolDefinition & ***lhs*****, const SymbolDefinition &** ***rhs*****)**

Definition at line 15 of file SymbolDefinition.cpp.

References bnssassembler::SymbolDefinition::name_.

```
{
        return lhs.name_ == rhs.name_;
    }
```

bool bnssassembler::operator== (const RelocationRecord & ***lhs*****, const RelocationRecord &** ***rhs*****)**

Definition at line 61 of file RelocationRecord.cpp.

References bnssassembler::RelocationRecord::absolute_, bnssassembler::RelocationRecord::offset_, bnssassembler::RelocationRecord::section_, bnssassembler::RelocationRecord::section_index_, and bnssassembler::RelocationRecord::symbol_name_.

```
{
        return
            lhs.offset_ == rhs.offset_ &&
            lhs.absolute_ == rhs.absolute_ &&
            lhs.section_ == rhs.section_ &&
            (lhs.section_ ? lhs.section_index_ == rhs.section_index_ : lhs.symbol_name_ == rhs.symbol_name_);
    }
```

bool bnssassembler::operator== (const SectionData & *lhs*, const SectionData & *rhs*) `[noexcept]`

Definition at line 79 of file SectionData.cpp.

```
{
        return lhs.type_ == rhs.type_ && lhs.indexed_ == rhs.indexed_ && (lhs.indexed_ ? lhs.index_ == rhs.index_ : true);
    }
```

bool bnssassembler::operator> (const SymbolDefinition & *lhs*, const SymbolDefinition & *rhs*)

Definition at line 27 of file SymbolDefinition.cpp.

```
{
        return !(lhs < rhs || lhs == rhs);
    }
```

bool bnssassembler::operator> (const SectionData & *lhs*, const SectionData & *rhs*) `[noexcept]`

Definition at line 111 of file SectionData.cpp.

```
{
        return !(lhs < rhs || lhs == rhs);
    }
```

bool bnssassembler::operator>= (const SymbolDefinition & *lhs*, const SymbolDefinition & *rhs*)

Definition at line 35 of file SymbolDefinition.cpp.

```
{
        return !(lhs < rhs);
    }
```

bool bnssassembler::operator>= (const SectionData & *lhs*, const SectionData & *rhs*) `[noexcept]`

Definition at line 119 of file SectionData.cpp.

```
{
        return !(lhs < rhs);
    }
```

Data bnssassembler::parseData (std::string *str*)

Parses the data from the string.

Parameters:

str	String that will be parsed

Returns:

Parsed data

Exceptions:

Throws	if the data could not be parsed

Definition at line 19 of file DataDefinitionLineParser.cpp.

References bnssassembler::ExpressionBuilder::build(), CONSTANT_TERM, DUPLICATE_DIRECTIVE, and bnssassembler::DataTypeParser::parse().

Referenced by bnssassembler::DataDefinitionLineParser::parse().

```
   19                           {
   20         static std::regex splitter("(.*)" + DUPLICATE_DIRECTIVE + "(.*)");
   21         static std::regex left_regex("[[:space:]]*([Dd][BbWwDd])(" +
CONSTANT_TERM + ")");
   22         static std::regex uninitialized_value("[[:space:]]*\\?[[:space:]]*");
   23 
   24         auto left = str;
   25         std::string right;
   26 
   27         if (regex_match(str, splitter)) {
   28             left = regex_replace(str, splitter, "$1");
   29             right = regex_replace(str, splitter, "$2");
   30         }
   31 
   32         if (!regex_match(left, left_regex)) {
   33             throw InvalidDataDefinitionException(str);
   34         }
   35 
   36         auto data_type_string = regex_replace(left, left_regex, "$1");
   37         auto left_expression_string = regex_replace(left, left_regex, "$2");
   38 
   39         auto data_type = DataTypeParser::parse(data_type_string);
   40         auto left_expression =
ExpressionBuilder::build(left_expression_string);
   41 
   42         if (right.empty()) {
   43             return Data(data_type, left_expression,
ExpressionBuilder::build("1"));
   44         }
   45 
   46         if (regex_match(right, uninitialized_value)) {
   47             return Data(data_type, left_expression);
   48         }
   49 
   50         auto right_expression = ExpressionBuilder::build(right);
   51         return Data(data_type, right_expression, left_expression);
   52     }
```

std::shared_ptr<Operand> bnssassembler::parsePcrel (std::string *str*)

Parses the input as a PC relative address.

Parameters:

str	String representation of an operand

Returns:

Pointer to the parsed operand or nullptr if the input is not a PC relative address

Exceptions:

Throws	if the string could not be parsed

Definition at line 16 of file RegisterIndirectOffsetParser.cpp.

References bnssassembler::ExpressionBuilder::build(), CONSTANT_TERM, and PC.

Referenced by bnssassembler::RegisterIndirectOffsetParser::parse().

```
                                                        {
            static std::regex regex("[[:space:]]*\\$(" + CONSTANT_TERM + ")");
            static std::regex 
not_pcrel("[[:space:]]*\\$[[:space:]]*[-+*/].*[[:space:]]*");

            if (!regex_match(str, regex) || regex_match(str, not_pcrel)) {
                return nullptr;
            }

            auto address_string = regex_replace(str, regex, "$1");
            auto address = ExpressionBuilder::build(address_string);
            return std::make_shared<RegisterIndirectOffset>(PC, address, true);
        }
```

static std::shared_ptr<Expression> bnssassembler::postfixToTree (const std::list< std::shared_ptr< ExpressionToken >> & *postfix_expression*)`[static]`

Builds a tree from the postfix expression.

Parameters:

postfix_expression	Postfix expression

Returns:

Pointer to the root of the tree

Definition at line 71 of file ExpressionBuilder.cpp.

Referenced by bnssassembler::ExpressionBuilder::build().

```
{
            if (postfix_expression.size() == 0) {
                return nullptr;
            }

            std::stack<std::reference_wrapper<std::shared_ptr<Expression>>> 
stack;
            std::shared_ptr<Expression> root = nullptr;
            stack.push(root);
            for (auto iterator = postfix_expression.rbegin(); iterator != 
postfix_expression.rend(); ++iterator) {
                if (stack.empty()) {
                    throw MessageException("Invalid expression - not enough 
operators");
                }

                std::shared_ptr<Expression> &curr = stack.top();
                stack.pop();
                curr = iterator->get()->create();
                curr->pushChildren(stack);
            }

            return root;
        }
```

static void bnssassembler::split (std::list< RelocationRecord > & *original*, std::list< RelocationRecord > & *left*, std::list< RelocationRecord > & *right*)`[static]`

Definition at line 11 of file AddOperation.cpp.

Referenced by bnssassembler::AddOperation::generateRelocations().

```
   11
{
   12         for (auto &element : original) {
   13             if (element.opposite()) {
   14                 element.toggleOpposite();
   15                 right.push_back(element);
   16             }
   17             else {
   18                 left.push_back(element);
   19             }
   20         }
   21     }
```

static void bnssassembler::stripComment (std::string & *line*, std::vector< std::string > *one_line_comment_delimiters*)`[static]`

Strips the comment from one line of the file.

Parameters:

line	Line of the file
one_line_comment_delimiters	Delimiters for one-line comments

Definition at line 18 of file Parser.cpp.

References bnssassembler::StringHelper::join().

Referenced by bnssassembler::Parser::parse().

```
   18
{
   19         auto delimiters = StringHelper::join(one_line_comment_delimiters,
"|");
   20         std::regex regex("(.*?)(" + delimiters + ").*");
   21
   22         line = regex_replace(line, regex, "$1");
   23     }
```

static void bnssassembler::writeDescription (SectionType *type*, bool *indexed*, size_t *index*, bool *org_valid*, uint32_t *org_address*, size_t *size*)`[static]`

Definition at line 149 of file SectionData.cpp.

References HORIZONTAL, multiple(), name(), T_LEFT, T_RIGHT, bnssassembler::StringHelper::toHexString(), UPPER_LEFT, UPPER_RIGHT, and VERTICAL.

Referenced by operator<<().

```
  149
{
  150         std::cout << UPPER_LEFT << multiple(HORIZONTAL, 81) << UPPER_RIGHT <<
std::endl;
  151         auto description = name(type, indexed, index) + " size: " +
StringHelper::toHexString(size) + (org_valid ? " ORG: " +
StringHelper::toHexString(org_address) : "");
  152         std::cout << VERTICAL << std::setw(81) << std::left << description <<
VERTICAL << std::endl;
  153         std::cout << T_RIGHT << multiple(HORIZONTAL, 81) << T_LEFT << std::endl;
  154     }
```

Variable Documentation

const std::string bnssassembler::ALL_FOUR = "\u256c"

Definition at line 29 of file PrintHelpers.h.

Referenced by operator<<().

const std::string bnssassembler::BINARY = "0b[01][01]*"

Definition at line 10 of file CommonRegexes.h.

const std::regex bnssassembler::BINARY_REGEX = std::regex(BINARY)

Definition at line 36 of file CommonRegexes.h.

Referenced by bnssassembler::StringHelper::parseNumber().

const std::string bnssassembler::CHARACTER = "'[[:print:]]'"

Definition at line 12 of file CommonRegexes.h.

const std::regex bnssassembler::CHARACTER_REGEX = std::regex(CHARACTER)

Definition at line 38 of file CommonRegexes.h.

Referenced by bnssassembler::StringHelper::parseNumber().

const std::string bnssassembler::COMMA_TOKENIZER = "[[:space:]]*(.*?)[[:space:]]*,(.*)"

Definition at line 30 of file CommonRegexes.h.

const std::regex bnssassembler::COMMA_TOKENIZER_REGEX = std::regex(COMMA_TOKENIZER)

Definition at line 51 of file CommonRegexes.h.

Referenced by bnssassembler::DataDefinitionLineParser::parse(), and bnssassembler::InstructionParser::parse().

const std::string bnssassembler::CONSTANT_TERM = "([[:space:]]*(" + LITERAL + "|" + OPERATOR + "|" + SYMBOL + ")[[:space:]]*)*"

Definition at line 23 of file CommonRegexes.h.

Referenced by bnssassembler::ImmediateParser::parse(), bnssassembler::SymbolDefinitionLineParser::parse(), bnssassembler::OrgDirectiveLineParser::parse(), parseData(), and parsePcrel().

const std::regex bnssassembler::CONSTANT_TERM_REGEX = std::regex(CONSTANT_TERM)

Definition at line 44 of file CommonRegexes.h.

Referenced by bnssassembler::MemoryDirectParser::parse().

const std::string bnssassembler::DECIMAL = "[1-9][0-9]*"

Definition at line 8 of file CommonRegexes.h.

const std::regex bnssassembler::DECIMAL_REGEX = std::regex(DECIMAL)

Definition at line 34 of file CommonRegexes.h.

Referenced by bnssassembler::StringHelper::parseNumber().

const std::string bnssassembler::DUPLICATE_DIRECTIVE = "[Dd][Uu][Pp]"

Definition at line 27 of file CommonRegexes.h.

Referenced by parseData().

const std::regex bnssassembler::DUPLICATE_DIRECTIVE_REGEX = std::regex(DUPLICATE_DIRECTIVE)

Definition at line 48 of file CommonRegexes.h.

const std::string bnssassembler::GLOBAL_DIRECTIVE = "[.][Gg][Ll][Oo][Bb][Aa][Ll]"

Definition at line 28 of file CommonRegexes.h.

Referenced by bnssassembler::GlobalSymbolsLineParser::parse().

const std::regex bnssassembler::GLOBAL_DIRECTIVE_REGEX = std::regex(GLOBAL_DIRECTIVE)

Definition at line 49 of file CommonRegexes.h.

const std::string bnssassembler::HEX = "0x[0-9a-fA-F][0-9a-fA-F]*"

Definition at line 9 of file CommonRegexes.h.

const std::regex bnssassembler::HEX_REGEX = std::regex(HEX)

Definition at line 35 of file CommonRegexes.h.

Referenced by bnssassembler::StringHelper::parseNumber().

const std::string bnssassembler::HORIZONTAL = "\u2550"

Definition at line 23 of file PrintHelpers.h.

Referenced by operator<<(), and writeDescription().

const std::string bnssassembler::LABEL = SYMBOL

Definition at line 22 of file CommonRegexes.h.

Referenced by extractLabel().

const std::regex bnssassembler::LABEL_REGEX = std::regex(LABEL)

Definition at line 43 of file CommonRegexes.h.

const std::string bnssassembler::LAST_COMMA_TOKEN = "[[:space:]]*(.*)[[:space:]]*"

Definition at line 31 of file CommonRegexes.h.

const std::regex bnssassembler::LAST_COMMA_TOKEN_REGEX = std::regex(LAST_COMMA_TOKEN)

Definition at line 52 of file CommonRegexes.h.

Referenced by bnssassembler::DataDefinitionLineParser::parse(), and bnssassembler::InstructionParser::parse().

const std::string bnssassembler::LITERAL = "(" + ZERO + "|" + DECIMAL + "|" + HEX + "|" + BINARY + "|" + OCT + "|" + CHARACTER + ")"

Definition at line 17 of file CommonRegexes.h.

Referenced by infixToPostfix().

const std::regex bnssassembler::LITERAL_REGEX = std::regex(LITERAL)

Definition at line 40 of file CommonRegexes.h.

Referenced by bnssassembler::ExpressionTokenFactory::create().

const std::string bnssassembler::LOWER_LEFT = "\u255a"

Definition at line 21 of file PrintHelpers.h.

Referenced by operator<<().

const std::string bnssassembler::LOWER_RIGHT = "\u255d"

Definition at line 22 of file PrintHelpers.h.

Referenced by operator<<().

const size_t bnssassembler::NUM_OF_REGISTERS = 16

Number of all purpose registers (excluding PC and SP)

Definition at line 11 of file Register.h.

Referenced by bnssassembler::RegisterParser::RegisterParserStaticData::RegisterParserStaticData().

const std::string bnssassembler::OCT = "0[0-7][0-7]*"

Definition at line 11 of file CommonRegexes.h.

const std::regex bnssassembler::OCT_REGEX = std::regex(OCT)

Definition at line 37 of file CommonRegexes.h.

Referenced by bnssassembler::StringHelper::parseNumber().

const std::string bnssassembler::OPERATOR = "[-+*/()]"

Definition at line 20 of file CommonRegexes.h.

Referenced by infixToPostfix().

const std::regex bnssassembler::OPERATOR_REGEX = std::regex(OPERATOR)

Definition at line 41 of file CommonRegexes.h.

Referenced by bnssassembler::ExpressionTokenFactory::create().

const std::string bnssassembler::ORG_DIRECTIVE = "[Oo][Rr][Gg]"

Definition at line 25 of file CommonRegexes.h.

Referenced by bnssassembler::OrgDirectiveLineParser::parse().

const std::regex bnssassembler::ORG_DIRECTIVE_REGEX = std::regex(ORG_DIRECTIVE)

Definition at line 46 of file CommonRegexes.h.

const std::string bnssassembler::SYMBOL = "(([a-zA-Z_][a-zA-Z_0-9]*)|\\$)"

Definition at line 21 of file CommonRegexes.h.

Referenced by infixToPostfix(), bnssassembler::SymbolDefinitionLineParser::parse(), and bnssassembler::GlobalSymbolsLineParser::parse().

const std::string bnssassembler::SYMBOL_DEFINITION = "[Dd][Ee][Ff]"

Definition at line 26 of file CommonRegexes.h.

Referenced by bnssassembler::SymbolDefinitionLineParser::parse().

const std::regex bnssassembler::SYMBOL_DEFINITION_REGEX = std::regex(SYMBOL_DEFINITION)

Definition at line 47 of file CommonRegexes.h.

const std::regex bnssassembler::SYMBOL_REGEX = std::regex(SYMBOL)

Definition at line 42 of file CommonRegexes.h.

Referenced by bnssassembler::ExpressionTokenFactory::create().

const std::string bnssassembler::T_DOWN = "\u2566"

Definition at line 28 of file PrintHelpers.h.

Referenced by operator<<().

const std::string bnssassembler::T_LEFT = "\u2563"

Definition at line 25 of file PrintHelpers.h.

Referenced by operator<<(), and writeDescription().

const std::string bnssassembler::T_RIGHT = "\u2560"

Definition at line 26 of file PrintHelpers.h.

Referenced by operator<<(), and writeDescription().

const std::string bnssassembler::T_UP = "\u2569"

Definition at line 27 of file PrintHelpers.h.

Referenced by operator<<().

const std::string bnssassembler::UPPER_LEFT = "\u2554"

Definition at line 19 of file PrintHelpers.h.

Referenced by operator<<(), and writeDescription().

const std::string bnssassembler::UPPER_RIGHT = "\u2557"

Definition at line 20 of file PrintHelpers.h.

Referenced by operator<<(), and writeDescription().

const std::string bnssassembler::VERTICAL = "\u2551"

Definition at line 24 of file PrintHelpers.h.

Referenced by operator<<(), and writeDescription().

const std::string bnssassembler::ZERO = "0"

Definition at line 7 of file CommonRegexes.h.

const std::regex bnssassembler::ZERO_REGEX = std::regex(ZERO)

Definition at line 33 of file CommonRegexes.h.

Referenced by bnssassembler::StringHelper::parseNumber().

bnssemulator Namespace Reference

Classes

- class **AddExecuter**
- *Class representing the executer for the add instruction.* class **AddressSpace**
- *Class representing the address space of the emulator.* class **AluExecuter**
- *Base class used for executing ALU instructions.* class **AndExecuter**
- *Class representing the executer for the and instruction.* class **AslExecuter**
- *Class representing the executer for the asl instruction.* class **AsrExecuter**
- *Class representing the executer for the asr instruction.* class **AssemblerOutput**
- *Class representing the output from the assembler.* class **CallExecuter**
- *Class representing the executer for the call instruction.* class **CommandLineHelper**
- *Utility class used for parsing the command line.* struct **compare_pair_difference**
- struct **compare_pair_first**
- struct **compare_pair_second**
- class **ConditionalJumpExecuter**
- *Base executer for conditional jump instructions.* class **Context**
- *Class representing the context of the processor.* class **DivideExecuter**
- *Class representing the executer of the divide instruction.* class **Executer**
- *Base class used for executing instructions.* class **FileReader**
- *Utility class used for reading assembler output from the file.* struct **InstructionBitField**
- *Bit field that enables easier manipulation of instructions.* union **InstructionBitFieldUnion**
- *Union that enables easier manipulation of the instruction bit field.* class **IntExecuter**
- *Class representing the executer for the int instruction.* class **JgezExecuter**
- *Class representing the executer for the jgez instruction.* class **JgzExecuter**
- *Class representing the executer for the jgz instruction.* class **JlezExecuter**
- *Class representing the executer for the jlez instruction.* class **JlzExecuter**
- *Class representing the executer for the jlz instruction.* class **JmpExecuter**
- *Class representing the executer for the jmp instruction.* class **JnzExecuter**
- *Class representing the executer for the jnz instruction.* class **JzExecuter**
- *Class representing the executer for the jz instruction.* class **KeyboardListener**
- *Class representing the keyboard listener thread.* class **LoadExecuter**
- *Class representing the executer for the load instruction.* class **MessageException**
- *Represents an exception with a string message.* class **ModuloExecuter**
- *Class representing the executer for the modulo instruction.* class **MultiplyExecuter**
- *Class representing the executer for the multiply instruction.* class **NotExecuter**
- *Class representing the executer for the not instruction.* class **OrExecuter**
- *Class representing the executer for the or instruction.* class **PopExecuter**
- *Class representing the executer for the pop instruction.* class **Processor**
- *Class representing the processor.* class **PushExecuter**
- *Class representing the executer for the push instruction.* class **Register**
- *Class representing the register.* class **RelocationRecord**
- *Class representing one relocation record.* class **RetExecuter**
- *Class representing the executer for ret instruction.* class **SectionData**
- *Class representing the data about one section.* class **Segment**
- *Class representing one segment of memory.* class **StoreExecuter**
- *Class representing the executer for the store instruction.* class **StringHelper**
- *Utility class providing helper methods for std::string class.* class **SubtractExecuter**
- *Class representing the executer for the subtract instruction.* class **SymbolData**
- *Class representing data about one symbol.* class **TimerListener**
- *Class representing a listener for the timer events.* class **XorExecuter**

Class representing the executer for the xor instruction. Enumerations

- enum **AddressMode** : uint32_t { **IMMEDIATE** = 0b100, **REGISTER_DIRECT** = 0b000, **MEMORY_DIRECT** = 0b110, **REGISTER_INDIRECT** = 0b010, **REGISTER_INDIRECT_OFFSET** = 0b111 }*Enum representing the address mode.*
- enum **DataType** : int8_t { **DOUBLE_WORD** = 0, **WORD**, **BYTE** }*Enum representing a data type.*
- enum **InstructionCode** : int8_t { **INT** = 0x00, **JMP** = 0x02, **CALL** = 0x03, **RET** = 0x01, **JZ** = 0x04, **JNZ** = 0x05, **JGZ** = 0x06, **JGEZ** = 0x07, **JLZ** = 0x08, **JLEZ** = 0x09, **LOAD** = 0x10, **STORE** = 0x11, **PUSH** = 0x20, **POP** = 0x21, **ADD** = 0x30, **SUB** = 0x31, **MUL** = 0x32, **DIV** = 0x33, **MOD** = 0x34, **AND** = 0x35, **OR** = 0x36, **XOR** = 0x37, **NOT** = 0x38, **ASL** = 0x39, **ASR** = 0x3A }*Enum representing the instruction code.*
- enum **OperandType** : int8_t { **DEFAULT** = 0b000, **UNSIGNED_BYTE** = 0b011, **SIGNED_BYTE** = 0b111, **REGULAR_BYTE** = 0b111, **UNSIGNED_WORD** = 0b001, **SIGNED_WORD** = 0b101, **REGULAR_WORD** = 0b101, **REGULAR_DOUBLE_WORD** = 0b000 }*Enum representing the operand type.*
- enum **SectionType** : int8_t { **TEXT** = 0, **DATA**, **RODATA**, **BSS** }*Enum representing the type of the section.*

Functions

- static void **removeEmpty** (std::vector< **SectionData** > &section_table)
- static bool **checkOverlaps** (const std::vector< **SectionData** > &section_table)
- static std::list< std::pair< **uint32_t**, **uint32_t** > > **getAvailable** (const std::vector< **SectionData** > &section_table)
- static void **generateAddresses** (std::vector< **SectionData** > &section_table, std::list< std::pair< **uint32_t**, **uint32_t** >> &available)
- std::istream & **operator>>** (std::istream &is, **AssemblerOutput** &data)
- static size_t **getRegisterIndex** (**InstructionBitField** instruction, size_t register_index)
  Gets the index of the register in the array of registers.
- static **uint32_t fill** (**OperandType** type, int32_t operand)
- static **InstructionCode opcode** (**InstructionBitField** instruction)
- **Register operator+** (const **Register** &lhs, const **Register** &rhs) noexcept
- **Register operator-** (const **Register** &lhs, const **Register** &rhs) noexcept
- **Register operator*** (const **Register** &lhs, const **Register** &rhs) noexcept
- **Register operator/** (const **Register** &lhs, const **Register** &rhs) noexcept
- **Register operator%** (const **Register** &lhs, const **Register** &rhs) noexcept
- **Register operator &** (const **Register** &lhs, const **Register** &rhs) noexcept
- **Register operator|** (const **Register** &lhs, const **Register** &rhs) noexcept
- **Register operator^** (const **Register** &lhs, const **Register** &rhs) noexcept
- **Register operator<<** (const **Register** &lhs, const **Register** &rhs) noexcept
- **Register operator>>** (const **Register** &lhs, const **Register** &rhs) noexcept
- std::istream & **operator>>** (std::istream &is, **RelocationRecord** &data)
- std::istream & **operator>>** (std::istream &is, **SectionData** &data)
- std::istream & **operator>>** (std::istream &is, **SymbolData** &data)

Variables

- const size_t **BLOCK_BITS** = 16
- const **uint32_t PAGE_MASK** = ~0 << **BLOCK_BITS**
- const **uint32_t OFFSET_MASK** = **~PAGE_MASK**
- const size_t **BLOCK_SIZE** = **OFFSET_MASK** + 1
- const std::string **ZERO** = "0"
- const std::string **DECIMAL** = "[1-9][0-9]*"
- const std::string **HEX** = "0x[0-9a-fA-F][0-9a-fA-F]*"
- const std::string **BINARY** = "0b[01][01]*"
- const std::string **OCT** = "0[0-7][0-7]*"
- const std::string **CHARACTER** = "'[[:print:]]'"

- const std::string **LITERAL** = "(" + **ZERO** + "|" + **DECIMAL** + "|" + **HEX** + "|" + **BINARY** + "|" + **OCT** + "|" + **CHARACTER** + ")"
- const std::string **OPERATOR** = "[-+*/()]"
- const std::string **SYMBOL** = "(([a-zA-Z_][a-zA-Z_0-9]*)|\\$)"
- const std::string **LABEL** = **SYMBOL**
- const std::string **CONSTANT_TERM** = "([[:space:]]*(" + LITERAL + "|" + **OPERATOR** + "|" + **SYMBOL** + ")[[:space:]]*)*"
- const std::string **ORG_DIRECTIVE** = "[Oo][Rr][Gg]"
- const std::string **SYMBOL_DEFINITION** = "[Dd][Ee][Ff]"
- const std::string **DUPLICATE_DIRECTIVE** = "[Dd][Uu][Pp]"
- const std::string **GLOBAL_DIRECTIVE** = "[.][Gg][Ll][Oo][Bb][Aa][Ll]"
- const std::string **COMMA_TOKENIZER** = "[[:space:]]*(.*?)[[:space:]]*,(.*)"
- const std::string **LAST_COMMA_TOKEN** = "[[:space:]]*(.*)[[:space:]]*"
- const std::regex **ZERO_REGEX** = std::regex(**ZERO**)
- const std::regex **DECIMAL_REGEX** = std::regex(**DECIMAL**)
- const std::regex **HEX_REGEX** = std::regex(**HEX**)
- const std::regex **BINARY_REGEX** = std::regex(**BINARY**)
- const std::regex **OCT_REGEX** = std::regex(**OCT**)
- const std::regex **CHARACTER_REGEX** = std::regex(**CHARACTER**)
- const std::regex **LITERAL_REGEX** = std::regex(**LITERAL**)
- const std::regex **OPERATOR_REGEX** = std::regex(**OPERATOR**)
- const std::regex **SYMBOL_REGEX** = std::regex(**SYMBOL**)
- const std::regex **LABEL_REGEX** = std::regex(**LABEL**)
- const std::regex **CONSTANT_TERM_REGEX** = std::regex(**CONSTANT_TERM**)
- const std::regex **ORG_DIRECTIVE_REGEX** = std::regex(**ORG_DIRECTIVE**)
- const std::regex **SYMBOL_DEFINITION_REGEX** = std::regex(**SYMBOL_DEFINITION**)
- const std::regex **DUPLICATE_DIRECTIVE_REGEX** = std::regex(**DUPLICATE_DIRECTIVE**)
- const std::regex **GLOBAL_DIRECTIVE_REGEX** = std::regex(**GLOBAL_DIRECTIVE**)
- const std::regex **COMMA_TOKENIZER_REGEX** = std::regex(**COMMA_TOKENIZER**)
- const std::regex **LAST_COMMA_TOKEN_REGEX** = std::regex(**LAST_COMMA_TOKEN**)
- static const int32_t **UNSIGNED_BYTE_MASK** = 0x000000ff
- static const int32_t **UNSIGNED_WORD_MASK** = 0x0000ffff
- static const int32_t **SIGNED_BYTE_TEST** = 0x00000080
- static const int32_t **SIGNED_WORD_TEST** = 0x00008000
- static const int32_t **SIGNED_BYTE_FILL** = 0xffffff00
- static const int32_t **SIGNED_WORD_FILL** = 0xffff0000
- static const uint64_t **TOP_32_BITS** = ~static_cast<uint64_t>(0) << 32

Enumeration Type Documentation

enum bnssemulator::AddressMode : uint32_t

Enum representing the address mode.

Enumerator:

IMMEDIATE	
REGISTER_DIRECT	
MEMORY_DIRECT	
REGISTER_INDIRECT	
REGISTER_INDI	

RECT_OFFSET	

Definition at line 10 of file AddressMode.h.

```
                       : uint32_t {
        IMMEDIATE = 0b100,
        REGISTER_DIRECT = 0b000,
        MEMORY_DIRECT = 0b110,
        REGISTER_INDIRECT = 0b010,
        REGISTER_INDIRECT_OFFSET = 0b111
    };
```

enum bnssemulator::DataType : int8_t

Enum representing a data type.

Enumerator:

DOUBLE_WORD	32bit value
WORD	16bit value
BYTE	8bit value

Definition at line 9 of file DataType.h.

```
                    : int8_t {
        DOUBLE_WORD = 0,
        WORD,
        BYTE
    };
```

enum bnssemulator::InstructionCode : int8_t

Enum representing the instruction code.

Enumerator:

INT	
JMP	
CALL	
RET	
JZ	
JNZ	
JGZ	
JGEZ	
JLZ	
JLEZ	
LOAD	
STORE	
PUSH	
POP	
ADD	
SUB	
MUL	
DIV	
MOD	
AND	
OR	

XOR	
NOT	
ASL	
ASR	

Definition at line 12 of file InstructionCode.h.

```
                        : int8_t {
        INT   = 0x00,
        JMP   = 0x02,
        CALL  = 0x03,
        RET   = 0x01,
        JZ    = 0x04,
        JNZ   = 0x05,
        JGZ   = 0x06,
        JGEZ  = 0x07,
        JLZ   = 0x08,
        JLEZ  = 0x09,

        LOAD  = 0x10,
        STORE = 0x11,

        PUSH  = 0x20,
        POP   = 0x21,

        ADD   = 0x30,
        SUB   = 0x31,
        MUL   = 0x32,
        DIV   = 0x33,
        MOD   = 0x34,
        AND   = 0x35,
        OR    = 0x36,
        XOR   = 0x37,
        NOT   = 0x38,
        ASL   = 0x39,
        ASR   = 0x3A
    };
```

enum bnssemulator::OperandType : int8_t

Enum representing the operand type.

Enumerator:

DEFAULT	
UNSIGNED_BYTE	
SIGNED_BYTE	
REGULAR_BYTE	
UNSIGNED_WORD	
SIGNED_WORD	
REGULAR_WORD	
REGULAR_DOUBLE_WORD	

Definition at line 10 of file OperandType.h.

```
                    : int8_t {
        DEFAULT = 0b000,
        UNSIGNED_BYTE = 0b011,
        SIGNED_BYTE = 0b111,
        REGULAR_BYTE = 0b111,
        UNSIGNED_WORD = 0b001,
        SIGNED_WORD = 0b101,
        REGULAR_WORD = 0b101,
        REGULAR_DOUBLE_WORD = 0b000
    };
```

enum bnssemulator::SectionType : int8_t

Enum representing the type of the section.

Enumerator:

TEXT	Text section
DATA	Data section
RODATA	Read-only data section
BSS	Block started by symbol section

Definition at line 11 of file SectionType.h.

```
                      : int8_t {
        TEXT = 0,
        DATA,
        RODATA,
        BSS
    };
```

Function Documentation

static bool bnssemulator::checkOverlaps (const std::vector< SectionData > & *section_table*) `[static]`

Definition at line 23 of file AddressSpace.cpp.

Referenced by bnssemulator::AddressSpace::AddressSpace().

```
                                                                                  {
        std::vector<std::pair<uint32_t, uint32_t>> check;
        for (auto &section : section_table) {
            if (section.hasAddress()) {
                check.push_back(std::make_pair(section.address(),
section.address() + section.size() - 1));
            }
        }

        sort(check.begin(), check.end(), compare_pair_first<uint32_t,
uint32_t>());

        for (size_t i = 1; i < check.size(); i++) {
            if (check[i - 1].second >= check[i].first) {
                return true;
            }
        }

        return false;
    }
```

static uint32_t bnssemulator::fill (OperandType *type*, int32_t *operand*) `[static]`

Definition at line 17 of file LoadExecuter.cpp.

References REGULAR_DOUBLE_WORD, SIGNED_BYTE, SIGNED_BYTE_FILL, SIGNED_WORD, SIGNED_WORD_FILL, bnssemulator::StringHelper::toHexString(), UNSIGNED_BYTE, UNSIGNED_BYTE_MASK, UNSIGNED_WORD, and UNSIGNED_WORD_MASK.

Referenced by bnssemulator::LoadExecuter::execute().

```
   17                                                                {
   18         switch (type) {
   19         case UNSIGNED_BYTE:
   20             return operand & UNSIGNED_BYTE_MASK;
   21         case SIGNED_BYTE:
   22             if ((operand & SIGNED_BYTE_TEST) != 0) {
   23                 return operand | SIGNED_BYTE_FILL;
   24             }
   25 
   26             return operand;
   27         case UNSIGNED_WORD:
   28             return operand & UNSIGNED_WORD_MASK;
   29         case SIGNED_WORD:
   30             if ((operand & SIGNED_WORD_TEST) != 0) {
   31                 return operand | SIGNED_WORD_FILL;
   32             }
   33 
   34             return operand;
   35         case REGULAR_DOUBLE_WORD:
   36             return operand;
   37         default:
   38             throw MessageException("Invalid operand type: " +
StringHelper::toHexString(static_cast<int8_t>(type)));
   39         }
   40     }
```

static void bnssemulator::generateAddresses (std::vector< SectionData > & *section_table*, std::list< std::pair< uint32_t, uint32_t >> & *available*) `[static]`

Definition at line 77 of file AddressSpace.cpp.

Referenced by bnssemulator::AddressSpace::AddressSpace().

```
   77 
{
   78         for (auto &section : section_table) {
   79             if (!section.hasAddress()) {
   80                 auto found = false;
   81                 for (auto iterator = available.begin(); iterator !=
available.end(); ++iterator) {
   82                     if (iterator->second - iterator->first >= section.size())
{
   83                         found = true;
   84                         section.address(iterator->first);
   85                         iterator->first += section.size();
   86                         if (iterator->first == iterator->second) {
   87                             available.erase(iterator);
   88                         }
   89 
   90                         break;
   91                     }
   92                 }
   93 
   94                 if (!found) {
   95                     throw MessageException("There is not enough space for all
the sections");
   96                 }
   97             }
   98         }
   99     }
```

static std::list<std::pair<uint32_t, uint32_t> > bnssemulator::getAvailable (const std::vector< SectionData > & *section_table*) `[static]`

Definition at line 42 of file AddressSpace.cpp.

Referenced by bnssemulator::AddressSpace::AddressSpace().

```
   42
{
   43         std::set<std::pair<uint32_t, uint32_t>, compare_pair_first<uint32_t,
uint32_t>> set{{0, 0xffffffff}};
   44         for (auto &section : section_table) {
   45             if (section.hasAddress()) {
   46                 auto section_pair = std::make_pair(section.address(),
section.address() + section.size() - 1);
   47                 auto upper = set.upper_bound(section_pair);
   48                 --upper;
   49
   50                 auto left = std::make_pair(upper->first, section_pair.first -
1);
   51                 auto right = std::make_pair(section_pair.second + 1,
upper->second);
   52                 set.erase(upper);
   53
   54                 if (left.first != left.second + 1) {
   55                     set.insert(left);
   56                 }
   57
   58                 if (right.first != right.second + 1) {
   59                     set.insert(right);
   60                 }
   61             }
   62         }
   63
   64         std::list<std::pair<uint32_t, uint32_t>> ret;
   65         for (auto &entry : set) {
   66             ret.push_back(entry);
   67         }
   68
   69         return ret;
   70     }
```

static size_t bnssemulator::getRegisterIndex (InstructionBitField *instruction*, size_t *register_index*) `[static]`

Gets the index of the register in the array of registers.

Parameters:

instruction	Instruction
register_index	Index of the register in the instruction

Returns:

Index of the register in the array

Definition at line 81 of file Context.cpp.

References bnssemulator::StringHelper::numberFormat(), bnssemulator::InstructionBitField::register0, bnssemulator::InstructionBitField::register1, and bnssemulator::InstructionBitField::register2.

Referenced by bnssemulator::Context::getOperand(), and bnssemulator::Context::getOperandAddress().

```
   81
{
   82         switch (register_index) {
   83         case 0:
   84             return instruction.register0;
   85         case 1:
   86             return instruction.register1;
   87         case 2:
   88             return instruction.register2;
   89         default:
   90             throw MessageException("Invalid register index: " +
StringHelper::numberFormat(register_index));
   91         }
   92     }
```

static InstructionCode bnssemulator::opcode (InstructionBitField *instruction*)`[static]`

Definition at line 72 of file Processor.cpp.

References bnssemulator::InstructionBitField::operation_code.

Referenced by bnssemulator::Processor::executeInstruction().

```
   72                                                                {
   73         return static_cast<InstructionCode>(instruction.operation_code);
   74     }
```

Register bnssemulator::operator& (const Register & *lhs*, const Register & *rhs*)`[noexcept]`

Definition at line 131 of file Register.cpp.

References bnssemulator::Register::Register().

```
  131                                                                       {
  132         return Register(lhs.value_ & rhs.value_);
  133     }
```

Register bnssemulator::operator% (const Register & *lhs*, const Register & *rhs*)`[noexcept]`

Definition at line 127 of file Register.cpp.

References bnssemulator::Register::Register().

```
  127                                                                       {
  128         return Register(lhs.value_ % rhs.value_);
  129     }
```

Register bnssemulator::operator* (const Register & *lhs*, const Register & *rhs*)`[noexcept]`

Definition at line 112 of file Register.cpp.

References bnssemulator::Register::Register(), and TOP_32_BITS.

```
  112                                                                       {
  113         auto result_value = static_cast<int64_t>(lhs.value_) +
static_cast<int64_t>(rhs.value_);
  114         auto left = static_cast<bool>(lhs.value_ & INT32_MIN);
  115         auto right = static_cast<bool>(rhs.value_ & INT32_MIN);
  116         auto result = static_cast<bool>(result_value & INT32_MIN);
  117
  118         auto flags = ((result_value & TOP_32_BITS) != 0) || (!left && !right &&
result);
  119
  120         return Register(static_cast<int32_t>(result_value), flags, flags);
  121     }
```

Register bnssemulator::operator+ (const Register & *lhs*, const Register & *rhs*)`[noexcept]`

Definition at line 97 of file Register.cpp.

References bnssemulator::Register::Register().

```
   97                                                                       {
   98         auto result_value = static_cast<int64_t>(lhs.value_) +
static_cast<int64_t>(rhs.value_);
   99         auto left = static_cast<bool>(lhs.value_ & INT32_MIN);
  100         auto right = static_cast<bool>(rhs.value_ & INT32_MIN);
  101         auto result = static_cast<bool>(result_value & INT32_MIN);
  102
```

```
        auto flags = (left && right && !result) || (!left && !right && result);

        return Register(static_cast<int32_t>(result_value), flags, flags);
    }
```

Register bnssemulator::operator- (const Register & *lhs*, const Register & *rhs*)`[noexcept]`

Definition at line 108 of file Register.cpp.

```
                                                                           {
        return lhs + -rhs;
    }
```

Register bnssemulator::operator/ (const Register & *lhs*, const Register & *rhs*)`[noexcept]`

Definition at line 123 of file Register.cpp.

References bnssemulator::Register::Register().

```
                                                                           {
        return Register(lhs.value_ / rhs.value_);
    }
```

Register bnssemulator::operator<< (const Register & *lhs*, const Register & *rhs*)`[noexcept]`

Definition at line 143 of file Register.cpp.

References bnssemulator::Register::Register(), TOP_32_BITS, and bnssemulator::Register::value_.

```
                                                                            {
        auto shift = rhs.value_ % 32;
        auto result = lhs.value_ << shift;

        auto carry = (result & TOP_32_BITS) != 0;

        return Register(result, carry, false);
    }
```

std::istream& bnssemulator::operator>> (std::istream & *is*, RelocationRecord & *data*)

Parameters:

is	Input stream
data	Reference to the object that should be loaded

Returns:

Input stream

Definition at line 5 of file RelocationRecord.cpp.

References bnssemulator::RelocationRecord::absolute_, bnssemulator::RelocationRecord::offset_, bnssemulator::RelocationRecord::section_, bnssemulator::RelocationRecord::section_index_, and bnssemulator::RelocationRecord::symbol_name_.

```
                                                                     {
        is >> data.offset_;
        is >> data.absolute_;
        is >> data.section_;
        if (data.section_) {
            is >> data.section_index_;
        }
        else {
            is >> data.symbol_name_;
        }
```

```
   15
   16         return is;
   17     }
```

std::istream& bnssemulator::operator>> (std::istream & *is*, SectionData & *data*)

Parameters:

is	Input stream
data	Reference to the object that should be loaded

Returns:

Input stream

Definition at line 6 of file SectionData.cpp.

References bnssemulator::SectionData::data_, bnssemulator::SectionData::index_, bnssemulator::SectionData::indexed_, bnssemulator::SectionData::location_counter_, bnssemulator::SectionData::org_address_, bnssemulator::SectionData::org_valid_, bnssemulator::SectionData::relocation_records_, and bnssemulator::SectionData::type_.

```
    6                                                                 {
    7         int type;
    8         is >> type;
    9         data.type_ = static_cast<SectionType>(type);
   10         is >> data.indexed_;
   11         if (data.indexed_) {
   12             is >> data.index_;
   13         }
   14
   15         is >> data.org_valid_;
   16         if (data.org_valid_) {
   17             is >> data.org_address_;
   18         }
   19
   20         is >> data.location_counter_;
   21         size_t data_size;
   22         is >> data_size;
   23         for (size_t i = 0; i < data_size; i++) {
   24             int data_byte;
   25             is >> data_byte;
   26             data.data_.push_back(static_cast<int8_t>(data_byte));
   27         }
   28
   29         size_t relocation_records_size;
   30         is >> relocation_records_size;
   31         for (size_t i = 0; i < relocation_records_size; i++) {
   32             RelocationRecord relocation_record;
   33             is >> relocation_record;
   34             data.relocation_records_.push_back(relocation_record);
   35         }
   36
   37         return is;
   38     }
```

std::istream& bnssemulator::operator>> (std::istream & *is*, AssemblerOutput & *data*)

Parameters:

is	Input stream
data	Reference to the object that should be loaded

Returns:

Input stream

Definition at line 7 of file AssemblerOutput.cpp.

References bnssemulator::AssemblerOutput::imported_symbols_, bnssemulator::AssemblerOutput::section_table_, and bnssemulator::AssemblerOutput::symbol_table_.

```
    7                                                                      {
```

```
  8         size_t num_of_imported_symbols;
  9         is >> num_of_imported_symbols;
 10         for (size_t i = 0; i < num_of_imported_symbols; i++) {
 11             std::string symbol;
 12             is >> symbol;
 13             data.imported_symbols_.insert(symbol);
 14         }
 15 
 16         size_t section_table_size;
 17         is >> section_table_size;
 18         for (size_t i = 0; i < section_table_size; i++) {
 19             SectionData section;
 20             is >> section;
 21             data.section_table_.push_back(section);
 22         }
 23 
 24         size_t symbol_table_size;
 25         is >> symbol_table_size;
 26         for (size_t i = 0; i < symbol_table_size; i++) {
 27             SymbolData symbol;
 28             is >> symbol;
 29             data.symbol_table_[symbol.name()] = symbol;
 30         }
 31 
 32         return is;
 33     }
```

std::istream& bnssemulator::operator>> (std::istream & *is*, SymbolData & *data*)

Parameters:

is	Input stream
data	Reference to the object that should be loaded

Returns:

Input stream

Definition at line 17 of file SymbolData.cpp.

References bnssemulator::SymbolData::local_, bnssemulator::SymbolData::name_, bnssemulator::SymbolData::offset_, and bnssemulator::SymbolData::section_index_.

```
 17                                                                 {
 18         is >> data.name_;
 19         is >> data.section_index_;
 20         is >> data.offset_;
 21         is >> data.local_;
 22 
 23         return is;
 24     }
```

Register bnssemulator::operator>> (const Register & *lhs*, const Register & *rhs*) `[noexcept]`

Definition at line 152 of file Register.cpp.

References bnssemulator::Register::Register(), and bnssemulator::Register::value_.

```
152                                                                            {
153         auto shift = rhs.value_ % 32;
154         auto result = lhs.value_ >> shift;
155 
156         auto back = result << shift;
157         auto carry = lhs.value_ != back;
158 
159         return Register(result, carry, false);
160     }
```

Register bnssemulator::operator^ (const Register & *lhs*, const Register & *rhs*) `[noexcept]`

Definition at line 139 of file Register.cpp.

References bnssemulator::Register::Register().

```
                                                                            {
        return Register(lhs.value_ ^ rhs.value_);
    }
```

Register bnssemulator::operator| (const Register & *lhs*, const Register & *rhs*) [noexcept]

Definition at line 135 of file Register.cpp.

References bnssemulator::Register::Register().

```
                                                                            {
        return Register(lhs.value_ | rhs.value_);
    }
```

static void bnssemulator::removeEmpty (std::vector< SectionData > & *section_table*) [static]

Definition at line 12 of file AddressSpace.cpp.

Referenced by bnssemulator::AddressSpace::AddressSpace().

```
                                                                    {
        size_t j = 0;
        for (size_t i = 0; i < section_table.size(); i++) {
            if (section_table[i].size() != 0) {
                section_table[j++] = section_table[i];
            }
        }

        section_table.resize(j);
    }
```

Variable Documentation

const std::string bnssemulator::BINARY = "0b[01][01]*"

Definition at line 10 of file CommonRegexes.h.

const std::regex bnssemulator::BINARY_REGEX = std::regex(BINARY)

Definition at line 36 of file CommonRegexes.h.

Referenced by bnssemulator::StringHelper::parseNumber().

const size_t bnssemulator::BLOCK_BITS = 16

Definition at line 9 of file Address.h.

const size_t bnssemulator::BLOCK_SIZE = OFFSET_MASK + 1

Definition at line 12 of file Address.h.

const std::string bnssemulator::CHARACTER = "'[[:print:]]'"

Definition at line 12 of file CommonRegexes.h.

const std::regex bnssemulator::CHARACTER_REGEX = std::regex(CHARACTER)

Definition at line 38 of file CommonRegexes.h.

Referenced by bnssemulator::StringHelper::parseNumber().

const std::string bnssemulator::COMMA_TOKENIZER = "[[:space:]]*(.*?)[[:space:]]*,(.*)"

Definition at line 30 of file CommonRegexes.h.

const std::regex bnssemulator::COMMA_TOKENIZER_REGEX = std::regex(COMMA_TOKENIZER)

Definition at line 51 of file CommonRegexes.h.

const std::string bnssemulator::CONSTANT_TERM = "([[:space:]]*(" + LITERAL + "|" + OPERATOR + "|" + SYMBOL + ")[[:space:]]*)*"

Definition at line 23 of file CommonRegexes.h.

const std::regex bnssemulator::CONSTANT_TERM_REGEX = std::regex(CONSTANT_TERM)

Definition at line 44 of file CommonRegexes.h.

const std::string bnssemulator::DECIMAL = "[1-9][0-9]*"

Definition at line 8 of file CommonRegexes.h.

const std::regex bnssemulator::DECIMAL_REGEX = std::regex(DECIMAL)

Definition at line 34 of file CommonRegexes.h.

Referenced by bnssemulator::StringHelper::parseNumber().

const std::string bnssemulator::DUPLICATE_DIRECTIVE = "[Dd][Uu][Pp]"

Definition at line 27 of file CommonRegexes.h.

const std::regex bnssemulator::DUPLICATE_DIRECTIVE_REGEX = std::regex(DUPLICATE_DIRECTIVE)

Definition at line 48 of file CommonRegexes.h.

const std::string bnssemulator::GLOBAL_DIRECTIVE = "[.][Gg][Ll][Oo][Bb][Aa][Ll]"

Definition at line 28 of file CommonRegexes.h.

const std::regex bnssemulator::GLOBAL_DIRECTIVE_REGEX = std::regex(GLOBAL_DIRECTIVE)

Definition at line 49 of file CommonRegexes.h.

const std::string bnssemulator::HEX = "0x[0-9a-fA-F][0-9a-fA-F]*"

Definition at line 9 of file CommonRegexes.h.

const std::regex bnssemulator::HEX_REGEX = std::regex(HEX)

Definition at line 35 of file CommonRegexes.h.

Referenced by bnssemulator::StringHelper::parseNumber().

const std::string bnssemulator::LABEL = SYMBOL

Definition at line 22 of file CommonRegexes.h.

const std::regex bnssemulator::LABEL_REGEX = std::regex(LABEL)

Definition at line 43 of file CommonRegexes.h.

const std::string bnssemulator::LAST_COMMA_TOKEN = "[[:space:]]*(.*)[[:space:]]*"

Definition at line 31 of file CommonRegexes.h.

const std::regex bnssemulator::LAST_COMMA_TOKEN_REGEX = std::regex(LAST_COMMA_TOKEN)

Definition at line 52 of file CommonRegexes.h.

const std::string bnssemulator::LITERAL = "(" + ZERO + "|" + DECIMAL + "|" + HEX + "|" + BINARY + "|" + OCT + "|" + CHARACTER + ")"

Definition at line 17 of file CommonRegexes.h.

const std::regex bnssemulator::LITERAL_REGEX = std::regex(LITERAL)

Definition at line 40 of file CommonRegexes.h.

const std::string bnssemulator::OCT = "0[0-7][0-7]*"

Definition at line 11 of file CommonRegexes.h.

const std::regex bnssemulator::OCT_REGEX = std::regex(OCT)

Definition at line 37 of file CommonRegexes.h.

Referenced by bnssemulator::StringHelper::parseNumber().

const uint32_t bnssemulator::OFFSET_MASK = ~PAGE_MASK

Definition at line 11 of file Address.h.

const std::string bnssemulator::OPERATOR = "[-+*/()]"

Definition at line 20 of file CommonRegexes.h.

const std::regex bnssemulator::OPERATOR_REGEX = std::regex(OPERATOR)

Definition at line 41 of file CommonRegexes.h.

const std::string bnssemulator::ORG_DIRECTIVE = "[Oo][Rr][Gg]"

Definition at line 25 of file CommonRegexes.h.

const std::regex bnssemulator::ORG_DIRECTIVE_REGEX = std::regex(ORG_DIRECTIVE)

Definition at line 46 of file CommonRegexes.h.

const uint32_t bnssemulator::PAGE_MASK = ~0 << BLOCK_BITS

Definition at line 10 of file Address.h.

const int32_t bnssemulator::SIGNED_BYTE_FILL = 0xffffff00 `[static]`

Definition at line 14 of file LoadExecuter.cpp.

Referenced by fill().

const int32_t bnssemulator::SIGNED_BYTE_TEST = 0x00000080 `[static]`

Definition at line 11 of file LoadExecuter.cpp.

const int32_t bnssemulator::SIGNED_WORD_FILL = 0xffff0000 `[static]`

Definition at line 15 of file LoadExecuter.cpp.

Referenced by fill().

const int32_t bnssemulator::SIGNED_WORD_TEST = 0x00008000 `[static]`

Definition at line 12 of file LoadExecuter.cpp.

const std::string bnssemulator::SYMBOL = "(([a-zA-Z_][a-zA-Z_0-9]*)|\\$)"

Definition at line 21 of file CommonRegexes.h.

const std::string bnssemulator::SYMBOL_DEFINITION = "[Dd][Ee][Ff]"

Definition at line 26 of file CommonRegexes.h.

const std::regex bnssemulator::SYMBOL_DEFINITION_REGEX = std::regex(SYMBOL_DEFINITION)

Definition at line 47 of file CommonRegexes.h.

const std::regex bnssemulator::SYMBOL_REGEX = std::regex(SYMBOL)

Definition at line 42 of file CommonRegexes.h.

const uint64_t bnssemulator::TOP_32_BITS = ~static_cast<uint64_t>(0) << 32 `[static]`

Definition at line 5 of file Register.cpp.

Referenced by operator*(), and operator<<().

const int32_t bnssemulator::UNSIGNED_BYTE_MASK = 0x000000ff `[static]`

Definition at line 8 of file LoadExecuter.cpp.

Referenced by fill().

const int32_t bnssemulator::UNSIGNED_WORD_MASK = 0x0000ffff `[static]`

Definition at line 9 of file LoadExecuter.cpp.

Referenced by fill().

const std::string bnssemulator::ZERO = "0"

Definition at line 7 of file CommonRegexes.h.

const std::regex bnssemulator::ZERO_REGEX = std::regex(ZERO)

Definition at line 33 of file CommonRegexes.h.

Referenced by bnssemulator::StringHelper::parseNumber().

consoleio Namespace Reference

Functions

- bool **keyboardHit** ()
- int **getCharacter** ()
- static void **restore** ()
- static void **initialize** ()

Variables

- static struct termios **old_termios**
- static bool **initialized** = false
- static const int **STDIN_DESCRIPTOR_ID** = 0

Function Documentation

int consoleio::getCharacter ()

Definition at line 66 of file ConsoleInputOutput.cpp.

References initialize().

Referenced by bnssemulator::KeyboardListener::listen().

```
   66                        {
   67 #ifdef _MSC_VER
   68         return _getch();
   69 #else
   70         initialize();
   71         int r;
   72         unsigned char c;
   73         if ((r = read(0, &c, sizeof(c))) < 0) {
   74             return r;
   75         }
   76         else {
   77             return c;
   78         }
   79 #endif
   80     }
```

static void consoleio::initialize () `[static]`

Definition at line 28 of file ConsoleInputOutput.cpp.

References old_termios, and restore().

Referenced by getCharacter(), and keyboardHit().

```
   28                              {
   29         if (!initialized) {
   30             initialized = true;
   31 
   32             // Get the file descriptor for standard input terminal
   33             tcgetattr(STDIN_DESCRIPTOR_ID, &old_termios);
   34 
   35             // Create a copy of the descriptor to work on
   36             struct termios new_termios;
   37             memcpy(&new_termios, &old_termios, sizeof(new_termios));
   38 
   39             // Restore old descriptor on exit
   40             atexit(restore);
   41 
   42             // Unset the ECHO and ICANON flags, and set the descriptor
   43             // Unsetting the ECHO flag disables the output of characters to
terminal
```

```
   44             // when they are typed
   45             // Unsetting the ICANON flag makes the read function read characters
directly
   46             // from console, without the need for a newline
   47             new_termios.c_lflag &= ~(ICANON | ECHO);
   48             tcsetattr(STDIN_DESCRIPTOR_ID, TCSANOW, &new_termios);
   49         }
   50     }
```

bool consoleio::keyboardHit ()

Definition at line 53 of file ConsoleInputOutput.cpp.

References initialize().

Referenced by bnssemulator::KeyboardListener::listen().

```
   53                    {
   54 #ifdef _MSC_VER
   55         return _kbhit();
   56 #else
   57         initialize();
   58         struct timeval tv = { 0L, 0L };
   59         fd_set fds;
   60         FD_ZERO(&fds);
   61         FD_SET(0, &fds);
   62         return select(1, &fds, NULL, NULL, &tv);
   63 #endif
   64     }
```

static void consoleio::restore () `[static]`

Definition at line 24 of file ConsoleInputOutput.cpp.

References old_termios.

Referenced by initialize().

```
   24                       {
   25         tcsetattr(STDIN_DESCRIPTOR_ID, TCSANOW, &old_termios);
   26     }
```

Variable Documentation

bool consoleio::initialized = false `[static]`

Definition at line 21 of file ConsoleInputOutput.cpp.

struct termios consoleio::old_termios `[static]`

Definition at line 20 of file ConsoleInputOutput.cpp.

Referenced by initialize(), and restore().

const int consoleio::STDIN_DESCRIPTOR_ID = 0 `[static]`

Definition at line 22 of file ConsoleInputOutput.cpp.

cxxopts Namespace Reference

Namespaces

- **anonymous_namespace{cxxopts.h}**
- **values**

Classes

- class **argument_incorrect_type**
- struct **HelpGroupDetails**
- struct **HelpOptionDetails**
- class **invalid_option_format_error**
- class **missing_argument_exception**
- class **option_exists_error**
- class **option_not_exists_exception**
- class **option_not_has_argument_exception**
- class **option_not_present_exception**
- class **option_required_exception**
- class **option_requires_argument_exception**
- class **OptionAdder**
- class **OptionDetails**
- class **OptionException**
- class **OptionParseException**
- class **Options**
- class **OptionSpecException**
- class **Value**

Typedefs

- typedef std::string **String**

Functions

- template<typename T > T **toLocalString** (T &&t)
- size_t **stringLength** (const **String** &s)
- **String** & **stringAppend** (**String** &s, **String** a)
- **String** & **stringAppend** (**String** &s, size_t n, char c)
- template<typename Iterator > **String** & **stringAppend** (**String** &s, Iterator begin, Iterator end)
- template<typename T > std::string **toUTF8String** (T &&t)
- bool **empty** (const std::string &s)
- template<typename T > std::shared_ptr< **Value** > **value** ()
- template<typename T > std::shared_ptr< **Value** > **value** (T &t)
- void **check_required** (const **Options** &options, const std::vector< std::string > &required)

Typedef Documentation

typedef std::string cxxopts::String

Definition at line 184 of file cxxopts.h.

Function Documentation

void cxxopts::check_required (const Options & *options*, const std::vector< std::string > & *required*)`[inline]`

Definition at line 849 of file cxxopts.h.

References cxxopts::Options::count().

Referenced by cxxopts::OptionAdder::OptionAdder().

```
    {
        for (auto& r : required)
        {
            if (options.count(r) == 0)
            {
                throw option_required_exception(r);
            }
        }
    }
```

bool cxxopts::empty (const std::string & *s*)`[inline]`

Definition at line 230 of file cxxopts.h.

Referenced by cxxopts::Options::generate_group_help(), and cxxopts::OptionAdder::OptionAdder().

```
    {
        return s.empty();
    }
```

String & cxxopts::stringAppend (String & *s*, String *a*)`[inline]`

Definition at line 202 of file cxxopts.h.

Referenced by cxxopts::anonymous_namespace{cxxopts.h}::format_description(), and cxxopts::OptionAdder::OptionAdder().

```
    {
        return s.append(std::move(a));
    }
```

String & cxxopts::stringAppend (String & *s*, size_t *n*, char *c*)`[inline]`

Definition at line 209 of file cxxopts.h.

```
    {
        return s.append(n, c);
    }
```

template<typename Iterator > String & cxxopts::stringAppend (String & *s*, Iterator *begin*, Iterator *end*)

Definition at line 216 of file cxxopts.h.

```
    {
        return s.append(begin, end);
    }
```

size_t cxxopts::stringLength (const String & *s*)`[inline]`

Definition at line 195 of file cxxopts.h.

Referenced by cxxopts::Options::help_one_group(), and cxxopts::OptionAdder::OptionAdder().

```
    {
        return s.length();
    }
```

template<typename T > T cxxopts::toLocalString (T && *t*)

Definition at line 188 of file cxxopts.h.

Referenced by cxxopts::Options::add_option(), cxxopts::anonymous_namespace{cxxopts.h}::format_description(), cxxopts::anonymous_namespace{cxxopts.h}::format_option(), cxxopts::Options::help(), cxxopts::Options::help_one_group(), and cxxopts::OptionAdder::OptionAdder().

```
    {
        return t;
    }
```

template<typename T > std::string cxxopts::toUTF8String (T && *t*)

Definition at line 223 of file cxxopts.h.

Referenced by cxxopts::Options::help(), and cxxopts::OptionAdder::OptionAdder().

```
    {
        return std::forward<T>(t);
    }
```

template<typename T > std::shared_ptr< Value > cxxopts::value ()

Definition at line 567 of file cxxopts.h.

Referenced by cxxopts::Options::add_option(), cxxopts::values::standard_value< T >::default_value(), cxxopts::values::standard_value< T >::get(), cxxopts::values::standard_value< T >::implicit_value(), bnssassembler::LiteralToken::LiteralToken(), cxxopts::option_required_exception::option_required_exception(), cxxopts::OptionAdder::OptionAdder(), cxxopts::Options::positional_help(), bnssemulator::Context::timerTriggered(), Z85_decode_unsafe(), and Z85_encode_unsafe().

```
    {
        return std::make_shared<values::standard_value<T>>();
    }
```

template<typename T > std::shared_ptr< Value > cxxopts::value (T & *t*)

Definition at line 574 of file cxxopts.h.

```
    {
        return std::make_shared<values::standard_value<T>>(&t);
    }
```

cxxopts::anonymous_namespace{cxxopts.h} Namespace Reference

Functions

- std::basic_regex< char > **option_matcher** ("--([[:alnum:]][-_[:alnum:]]+)(=(.*))?|-([[:alnum:]]+)")
- std::basic_regex< char > **option_specifier** ("(([[:alnum:]]),)?([[:alnum:]][-_[:alnum:]]*)?")
- **String format_option** (const **HelpOptionDetails** &o)
- **String format_description** (const **HelpOptionDetails** &o, size_t start, size_t width)

Variables

- constexpr int **OPTION_LONGEST** = 30
- constexpr int **OPTION_DESC_GAP** = 2

Function Documentation

String cxxopts::anonymous_namespace{cxxopts.h}::format_description (const HelpOptionDetails & *o*, size_t *start*, size_t *width*)

Definition at line 918 of file cxxopts.h.

References cxxopts::HelpOptionDetails::default_value, cxxopts::HelpOptionDetails::desc, cxxopts::HelpOptionDetails::has_default, cxxopts::stringAppend(), and cxxopts::toLocalString().

Referenced by format_option(), cxxopts::Options::help_one_group(), and cxxopts::OptionAdder::OptionAdder().

```
        {
            auto desc = o.desc;

            if (o.has_default)
            {
                desc += toLocalString(" (default: " + o.default_value + ")");
            }

            String result;

            auto current = std::begin(desc);
            auto startLine = current;
            auto lastSpace = current;

            auto size = size_t{};

            while (current != std::end(desc))
            {
                if (*current == ' ')
                {
                    lastSpace = current;
                }

                if (size > width)
                {
                    if (lastSpace == startLine)
                    {
                        stringAppend(result, startLine, current + 1);
                        stringAppend(result, "\n");
                        stringAppend(result, start, ' ');
                        startLine = current + 1;
                        lastSpace = startLine;
                    }
                    else
                    {
                        stringAppend(result, startLine, lastSpace);
                        stringAppend(result, "\n");
                        stringAppend(result, start, ' ');
                        startLine = lastSpace + 1;
                    }
                    size = 0;
                }
                else
                {
                    ++size;
                }

                ++current;
            }

            //append whatever is left
            stringAppend(result, startLine, current);

            return result;
        }
```

String cxxopts::anonymous_namespace{cxxopts.h}::format_option (const HelpOptionDetails & *o*)

Definition at line 876 of file cxxopts.h.

References cxxopts::HelpOptionDetails::arg_help, format_description(), cxxopts::HelpOptionDetails::has_arg, cxxopts::HelpOptionDetails::has_implicit, cxxopts::HelpOptionDetails::implicit_value, cxxopts::HelpOptionDetails::l, cxxopts::HelpOptionDetails::s, and cxxopts::toLocalString().

Referenced by cxxopts::Options::help_one_group(), and cxxopts::OptionAdder::OptionAdder().

```
        {
            auto& s = o.s;
            auto& l = o.l;

            String result = "  ";

            if (s.size() > 0)
            {
                result += "-" + toLocalString(s) + ",";
            }
            else
            {
                result += "   ";
            }

            if (l.size() > 0)
            {
                result += " --" + toLocalString(l);
            }

            if (o.has_arg)
            {
                auto arg = o.arg_help.size() > 0 ? toLocalString(o.arg_help) : 
"arg";

                if (o.has_implicit)
                {
                    result += " [=" + arg + "(=" + 
toLocalString(o.implicit_value) + ")]";
                }
                else
                {
                    result += " " + arg;
                }
            }

            return result;
        }
```

std::basic_regex< char > cxxopts::anonymous_namespace{cxxopts.h}::option_matcher ("--([[:alnum:]][-_[:alnum:]]+)(=(.*))?|-([[:alnum:]]+)")

Referenced by cxxopts::OptionAdder::OptionAdder(), and cxxopts::Options::parse().

std::basic_regex< char > cxxopts::anonymous_namespace{cxxopts.h}::option_specifier ("(([[:alnum:]]),)?([[:alnum:]][-_[:alnum:]]*)?")

Referenced by cxxopts::OptionAdder::operator()(), and cxxopts::OptionAdder::OptionAdder().

Variable Documentation

constexpr int cxxopts::anonymous_namespace{cxxopts.h}::OPTION_DESC_GAP = 2

Definition at line 866 of file cxxopts.h.

Referenced by cxxopts::Options::help_one_group(), and cxxopts::OptionAdder::OptionAdder().

constexpr int cxxopts::anonymous_namespace{cxxopts.h}::OPTION_LONGEST = 30

Definition at line 865 of file cxxopts.h.

Referenced by cxxopts::OptionAdder::OptionAdder().

cxxopts::values Namespace Reference

Classes

- class **standard_value**
- struct **type_is_container**
- struct **type_is_container< std::vector< T > >**
- struct **value_has_arg**
- struct **value_has_arg< bool >**

Functions

- template<typename T > void **parse_value** (const std::string &text, T &**value**)
- void **parse_value** (const std::string &, bool &**value**)
- void **parse_value** (const std::string &text, std::string &**value**)
- template<typename T > void **parse_value** (const std::string &text, std::vector< T > &**value**)

Function Documentation

template<typename T > void cxxopts::values::parse_value (const std::string & *text*, T & *value*)

Definition at line 412 of file cxxopts.h.

```
        {
            std::istringstream is(text);
            if (!(is >> value))
            {
                throw argument_incorrect_type(text);
            }

            if (is.rdbuf()->in_avail() != 0)
            {
                throw argument_incorrect_type(text);
            }
        }
```

void cxxopts::values::parse_value (const std::string & , bool & *value*)`[inline]`

Definition at line 428 of file cxxopts.h.

```
        {
            value = true;
        }
```

void cxxopts::values::parse_value (const std::string & *text*, std::string & *value*)`[inline]`

Definition at line 435 of file cxxopts.h.

```
        {
            value = text;
        }
```

template<typename T > void cxxopts::values::parse_value (const std::string & *text*, std::vector< T > & *value*)

Definition at line 442 of file cxxopts.h.

Referenced by cxxopts::option_required_exception::option_required_exception(), and cxxopts::values::standard_value< T >::parse().

```
        {
            T v;
            parse_value(text, v);
            value.push_back(v);
        }
```

std Namespace Reference

Classes

- struct **hash< bnssassembler::InstructionCode >**
- struct **hash< bnssassembler::SectionData >**
- struct **hash< bnssassembler::SectionType >**
- struct **hash< bnssassembler::SymbolDefinition >**
- struct **hash< bnssemulator::InstructionCode >**
- struct **hash< bnssemulator::SectionType >**

z85 Namespace Reference

Functions

- std::string **encode_with_padding** (const char *source, size_t inputSize)
  Encodes 'inputSize' bytes from 'source'. If 'inputSize' is not divisible by 4 with no remainder, 'source' is padded.
- std::string **encode_with_padding** (const std::string &source)
- std::string **encode_with_padding** (const char *) **Z85_DELETE_FUNCTION_DEFINITION**
- std::string **decode_with_padding** (const char *source, size_t inputSize)
  *Decodes 'inputSize' printable symbols from 'source', encoded with **encode_with_padding()**.*
- std::string **decode_with_padding** (const std::string &source)
- std::string **decode_with_padding** (const char *) **Z85_DELETE_FUNCTION_DEFINITION**
- std::string **encode** (const char *source, size_t inputSize)
  Encodes 'inputSize' bytes from 'source'. If 'inputSize' is not divisible by 4 with no remainder, empty string is retured.
- std::string **encode** (const std::string &source)
- std::string **encode** (const char *) **Z85_DELETE_FUNCTION_DEFINITION**
- std::string **decode** (const char *source, size_t inputSize)
  Decodes 'inputSize' printable symbols from 'source'. If 'inputSize' is not divisible by 5 with no remainder, empty string is returned.
- std::string **decode** (const std::string &source)
- std::string **decode** (const char *) **Z85_DELETE_FUNCTION_DEFINITION**

Function Documentation

std::string z85::decode (const char * *source*, size_t *inputSize*)

Decodes 'inputSize' printable symbols from 'source'. If 'inputSize' is not divisible by 5 with no remainder, empty string is returned.

Parameters:

source	in, input buffer (printable string to be decoded)
inputSize	in, number of symbols to be decoded

Returns:

decoded string

Definition at line 115 of file z85_impl.cpp.

References Z85_decode(), and Z85_decode_bound().

Referenced by decode().

```
116     {
117         if (!source || inputSize == 0)
118         {
119             return std::string();
120         }
121 
122         std::string buf;
123         buf.resize(Z85_decode_bound(inputSize));
124 
125         const size_t decodedBytes = Z85_decode(source, &buf[0], inputSize);
126         if (decodedBytes == 0)
127         {
128             assert(!"wrong input size");
129             return std::string();
130         }
```

```

        return buf;
    }
```

std::string z85::decode (const std::string & *source*)

Definition at line 135 of file z85_impl.cpp.

References decode().

```
    {
        return decode(source.c_str(), source.size());
    }
```

std::string z85::decode (const char *)

std::string z85::decode_with_padding (const char * *source*, size_t *inputSize*)

Decodes 'inputSize' printable symbols from 'source', encoded with **encode_with_padding()**.

Parameters:

source	in, input buffer (printable string to be decoded)
inputSize	in, number of symbols to be decoded

Returns:

decoded string

Definition at line 62 of file z85_impl.cpp.

References Z85_decode_with_padding(), and Z85_decode_with_padding_bound().

Referenced by decode_with_padding(), and bnssemulator::FileReader::parse().

```
    {
        if (!source || inputSize == 0)
        {
            return std::string();
        }

        const size_t bufSize = Z85_decode_with_padding_bound(source, inputSize);
        if (bufSize == 0)
        {
            assert(!"wrong padding");
            return std::string();
        }

        std::string buf;
        buf.resize(bufSize);

        const size_t decodedBytes = Z85_decode_with_padding(source, &buf[0], inputSize);
        assert(decodedBytes == buf.size()); (void)decodedBytes;

        return buf;
    }
```

std::string z85::decode_with_padding (const std::string & *source*)

Definition at line 85 of file z85_impl.cpp.

References decode_with_padding().

```
    {
        return decode_with_padding(source.c_str(), source.size());
    }
```

std::string z85::decode_with_padding (const char *)

std::string z85::encode (const char * *source*, size_t *inputSize*)

Encodes 'inputSize' bytes from 'source'. If 'inputSize' is not divisible by 4 with no remainder, empty string is retured.

Parameters:

source	in, input buffer (binary string to be encoded)
inputSize	in, number of bytes to be encoded

Returns:

printable string

Definition at line 90 of file z85_impl.cpp.

References Z85_encode(), and Z85_encode_bound().

Referenced by encode().

```
    {
        if (!source || inputSize == 0)
        {
            return std::string();
        }

        std::string buf;
        buf.resize(Z85_encode_bound(inputSize));

        const size_t encodedBytes = Z85_encode(source, &buf[0], inputSize);
        if (encodedBytes == 0)
        {
            assert(!"wrong input size");
            return std::string();
        }

        return buf;
    }
```

std::string z85::encode (const std::string & *source*)

Definition at line 110 of file z85_impl.cpp.

References encode().

```
    {
        return encode(source.c_str(), source.size());
    }
```

std::string z85::encode (const char *)

std::string z85::encode_with_padding (const char * *source*, size_t *inputSize*)

Encodes 'inputSize' bytes from 'source'. If 'inputSize' is not divisible by 4 with no remainder, 'source' is padded.

Parameters:

source	in, input buffer (binary string to be encoded)
inputSize	in, number of bytes to be encoded

Returns:

printable string

Definition at line 41 of file z85_impl.cpp.

References Z85_encode_with_padding(), and Z85_encode_with_padding_bound().

Referenced by encode_with_padding(), and bnssassembler::FileWriter::write().

```
{
    if (!source || inputSize == 0)
    {
        return std::string();
    }

    std::string buf;
    buf.resize(Z85_encode_with_padding_bound(inputSize));

    const size_t encodedBytes = Z85_encode_with_padding(source, &buf[0], inputSize);
    assert(encodedBytes == buf.size()); (void)encodedBytes;

    return buf;
}
```

std::string z85::encode_with_padding (const std::string & *source*)

Definition at line 57 of file z85_impl.cpp.

References encode_with_padding().

```
{
    return encode_with_padding(source.c_str(), source.size());
}
```

std::string z85::encode_with_padding (const char *)

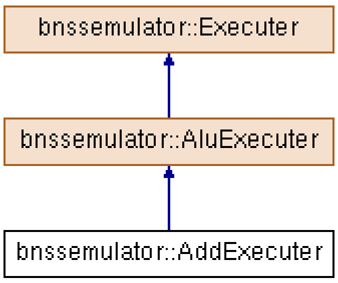
Class Documentation

bnssemulator::AddExecuter Class Reference

Class representing the executer for the add instruction.

`#include <AddExecuter.h>`

Inheritance diagram for bnssemulator::AddExecuter:



Protected Member Functions

- void **execute** (**Register** &dst, const **Register** &lhs, const **Register** &rhs) const override
  Executes the ALU instruction.

Additional Inherited Members

Detailed Description

Class representing the executer for the add instruction.

Definition at line 10 of file AddExecuter.h.

Member Function Documentation

void bnssemulator::AddExecuter::execute (Register & *dst*, const Register & *lhs*, const Register & *rhs*) const `[override]`, `[protected]`, `[virtual]`

Executes the ALU instruction.

Parameters:

dst	Reference to the destination register
lhs	Left operand register
rhs	Right operand register

Implements **bnssemulator::AluExecuter** (*p.102*).

Definition at line 5 of file AddExecuter.cpp.

```
    5
{
    6         dst = lhs + rhs;
    7     }
```

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**AddExecuter.h**

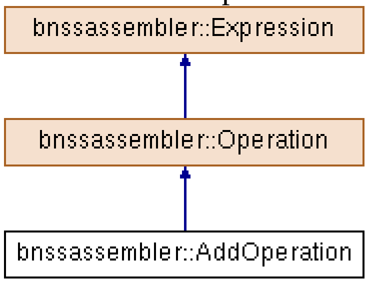
- Code/Emulator/Source/**AddExecuter.cpp**

bnssassembler::AddOperation Class Reference

Class implementing the behaviour of the + operator in expressions.

`#include <AddOperation.h>`

Inheritance diagram for bnssassembler::AddOperation:



Public Member Functions

- std::list< **RelocationRecord** > **generateRelocations** () const override
  Generates the relocation records for the subtree.

Protected Member Functions

- int32_t **calculate** (int32_t lhs, int32_t rhs) const noexcept override
  Calculates the value of the subexpression.

Detailed Description

Class implementing the behaviour of the + operator in expressions.

Definition at line 10 of file AddOperation.h.

Member Function Documentation

int32_t bnssassembler::AddOperation::calculate (int32_t *lhs*, int32_t *rhs*) const`[override]`, `[protected]`, `[virtual]`, `[noexcept]`

Calculates the value of the subexpression.

Parameters:

lhs	Left side of the operator
rhs	Right side of the operator

Returns:

Result of the operation

Exceptions:

Throws	if the expression can not be evaluated (example: division by zero)

Implements **bnssassembler::Operation** (*p.309*).

Definition at line 7 of file AddOperation.cpp.

```
                                                                    {
            return lhs + rhs;
     }
```

std::list< RelocationRecord > bnssassembler::AddOperation::generateRelocations () const`[override]`, `[virtual]`

Generates the relocation records for the subtree.

Returns:

Collection of relocation records

Reimplemented from **bnssassembler::Expression** (*p.165*).

Definition at line 23 of file AddOperation.cpp.

References bnssassembler::SubtractOperation::generateRelocations(), bnssassembler::Operation::generateRelocations(), bnssassembler::Operation::left(), bnssassembler::Operation::right(), and bnssassembler::split().

```
                                                                     {
        auto original = Operation::generateRelocations();
        std::list<RelocationRecord> left;
        std::list<RelocationRecord> right;

        split(original, left, right);
        return SubtractOperation::generateRelocations(left, right);
    }
```

The documentation for this class was generated from the following files:

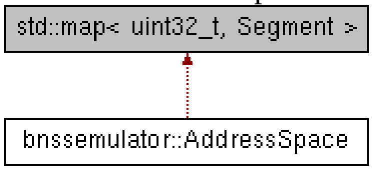
- Code/Assembler/Include/**AddOperation.h**
- Code/Assembler/Source/**AddOperation.cpp**

bnssemulator::AddressSpace Class Reference

Class representing the address space of the emulator.

`#include <AddressSpace.h>`

Inheritance diagram for bnssemulator::AddressSpace:



Public Member Functions

- **AddressSpace** (std::vector< **SectionData** > &&section_table, const std::unordered_map< std::string, **SymbolData** > symbol_table)
  Constructs an address space from the section table.
- **InstructionBitField getInstruction** (**uint32_t** address) const
  Gets the instruction at the specified address.
- int32_t **getSecondWordOfInstruction** (**uint32_t** address) const
  Gets the second word of the instruction at the specified address.
- uint8_t **get8bitData** (**uint32_t** address) const
  Gets 8 bits of data at the specified address.
- uint16_t **get16bitData** (**uint32_t** address) const
  Gets 16 bits of data at the specified address.
- **uint32_t get32bitData** (**uint32_t** address) const
  Gets 32 bits of data at the specified address.
- void **set8bitData** (**uint32_t** address, uint8_t data)
  Sets 8 bits of data at the specified address.
- void **set16bitData** (**uint32_t** address, uint16_t data)
  Sets 16 bits of data at the specified address.
- void **set32bitData** (**uint32_t** address, **uint32_t** data)
  Sets 32 bits of data at the specified address.
- **uint32_t initialStackPointer** () const
  Gets the initial value of the stack pointer.
- size_t **errorInterrupt** () const noexcept
  Gets the entry of the error interrupt routine.
- size_t **timerInterrupt** () const noexcept
  Gets the entry of the timer interrupt routine.
- size_t **keyboardInterrupt** () const noexcept
  Gets the entry of the keyboard interrupt routine.
- **uint32_t getInterrupt** (**uint32_t** entry) const noexcept
  Gets the address of the interrupt routine at the specified entry.
- bool **stdinRead** () const noexcept
  Check whether the standard input has been read.
- void **writeToStdin** (char character) noexcept
  Writes a character to stdin.

Private Member Functions

- **Segment** & **segment** (**uint32_t** address)
- const **Segment** & **segment** (**uint32_t** address) const

Private Attributes

- **uint32_t stdout_address_** = 128
- **uint32_t stdin_address_** = 132
- size_t **error_interrupt_** = 3
- size_t **timer_interrupt_** = 4
- size_t **keyboard_interrupt_** = 5
- bool **stdin_read_** = true

Detailed Description

Class representing the address space of the emulator.

Definition at line 16 of file AddressSpace.h.

Constructor & Destructor Documentation

bnssemulator::AddressSpace::AddressSpace (std::vector< SectionData > && *section_table*, const std::unordered_map< std::string, SymbolData > *symbol_table*) `[explicit]`

Constructs an address space from the section table.

Parameters:

section_table	Section table
symbol_table	Symbol table

Definition at line 101 of file AddressSpace.cpp.

References bnssemulator::checkOverlaps(), bnssemulator::generateAddresses(), bnssemulator::getAvailable(), and bnssemulator::removeEmpty().

```
  101
{
  102         removeEmpty(section_table);
  103
  104         if (checkOverlaps(section_table)) {
  105             throw MessageException("Sections are overlapping");
  106         }
  107
  108         auto available = getAvailable(section_table);
  109         generateAddresses(section_table, available);
  110
  111         for (auto &section : section_table) {
  112             insert(make_pair(section.address(), Segment(section.address(),
section.size(), section.type(), move(section.data()))));
  113         }
  114
  115         for (auto &section : section_table) {
  116             for (auto &relocation_entry : section.relocations()) {
  117                 uint32_t relocation;
  118
  119                 if (relocation_entry.section()) {
  120                     relocation =
section_table.at(relocation_entry.sectionIndex()).address();
  121                 }
  122                 else {
  123                     auto &symbol =
symbol_table.at(relocation_entry.symbolName());
  124                     relocation =
section_table.at(symbol.sectionIndex()).address() + symbol.offset();
  125                 }
  126
  127                 if (!relocation_entry.absolute()) {
```

```
  128                     relocation -= section.address() +
relocation_entry.offset();
  129                 }
  130
  131                 at(section.address()).relocate(section.address() +
relocation_entry.offset(), relocation);
  132             }
  133         }
  134     }
```

Member Function Documentation

size_t bnssemulator::AddressSpace::errorInterrupt () const`[noexcept]`

Gets the entry of the error interrupt routine.

Returns:

Entry of the error interrupt routine

Definition at line 189 of file AddressSpace.cpp.

References error_interrupt_.

Referenced by bnssemulator::Context::jumpToErrorInterrupt().

```
  189                                                       {
  190         return error_interrupt_;
  191     }
```

uint16_t bnssemulator::AddressSpace::get16bitData (uint32_t *address*) const

Gets 16 bits of data at the specified address.

Parameters:

address	Address

Returns:

Data

Definition at line 152 of file AddressSpace.cpp.

References get8bitData().

Referenced by get32bitData(), and bnssemulator::Context::getOperand().

```
  152                                                              {
  153         return get8bitData(address) |
(static_cast<uint16_t>(get8bitData(address + 1)) << 8);
  154     }
```

uint32_t bnssemulator::AddressSpace::get32bitData (uint32_t *address*) const

Gets 32 bits of data at the specified address.

Parameters:

address	Address

Returns:

Data

Definition at line 156 of file AddressSpace.cpp.

References get16bitData().

Referenced by getInterrupt(), bnssemulator::Context::getOperand(), initialStackPointer(), and bnssemulator::Context::popFromStack().

```
  156                                                                {
  157         return get16bitData(address) |
(static_cast<uint32_t>(get16bitData(address + 2)) << 16);
  158     }
```

uint8_t bnssemulator::AddressSpace::get8bitData (uint32_t *address*) const

Gets 8 bits of data at the specified address.

Parameters:

address	Address

Returns:

Data

Definition at line 144 of file AddressSpace.cpp.

References bnssemulator::Segment::readData(), segment(), stdin_address_, and stdin_read_.

Referenced by get16bitData(), and bnssemulator::Context::getOperand().

```
  144                                                              {
  145         if (address == stdin_address_) {
  146             stdin_read_ = true;
  147         }
  148
  149         return segment(address).readData(address);
  150     }
```

InstructionBitField bnssemulator::AddressSpace::getInstruction (uint32_t *address*) const

Gets the instruction at the specified address.

Parameters:

address	Address

Returns:

Instruction

Definition at line 136 of file AddressSpace.cpp.

References bnssemulator::Segment::getInstruction(), and segment().

Referenced by bnssemulator::Context::getInstruction().

```
  136                                                                              {
  137         return segment(address).getInstruction(address);
  138     }
```

uint32_t bnssemulator::AddressSpace::getInterrupt (uint32_t *entry*) const`[noexcept]`

Gets the address of the interrupt routine at the specified entry.

Parameters:

entry	Entry

Returns:

Address of the interrupt routine

Definition at line 201 of file AddressSpace.cpp.

References get32bitData().

Referenced by bnssemulator::Context::jumpToInterrupt().

```
  201                                                                       {
  202         try {
  203             return get32bitData(entry * 4);
  204         }
  205         catch (...) {
  206             return 0;
  207         }
  208     }
```

int32_t bnssemulator::AddressSpace::getSecondWordOfInstruction (uint32_t *address*) const

Gets the second word of the instruction at the specified address.

Parameters:

address	Address

Returns:

Second word of the instruction

Definition at line 140 of file AddressSpace.cpp.

References bnssemulator::Segment::getSecondWordOfInstruction(), and segment().

Referenced by bnssemulator::Context::getSecondWordOfInstruction().

```
  140                                                                             {
  141         return segment(address).getSecondWordOfInstruction(address);
  142     }
```

uint32_t bnssemulator::AddressSpace::initialStackPointer () const

Gets the initial value of the stack pointer.

Returns:

Initial value of the stack pointer

Exceptions:

Throws	if the initial value of the stack pointer is not defined

Definition at line 180 of file AddressSpace.cpp.

References get32bitData().

Referenced by bnssemulator::Context::Context().

```
  180                                                        {
  181         try {
  182             return get32bitData(0);
  183         }
  184         catch (...) {
  185             throw MessageException("Initial stack pointer value is not
defined");
  186         }
  187     }
```

size_t bnssemulator::AddressSpace::keyboardInterrupt () const `[noexcept]`

Gets the entry of the keyboard interrupt routine.

Returns:

Entry of the keyboard interrupt routine

Definition at line 197 of file AddressSpace.cpp.

References keyboard_interrupt_.

Referenced by bnssemulator::Context::jumpToKeyboardInterrupt().

```
                                                          {
        return keyboard_interrupt_;
    }
```

Segment & bnssemulator::AddressSpace::segment (uint32_t *address*)`[private]`

Definition at line 218 of file AddressSpace.cpp.

References bnssemulator::StringHelper::toHexString().

Referenced by get8bitData(), getInstruction(), getSecondWordOfInstruction(), segment(), and set8bitData().

```
                                                    {
        auto upper = upper_bound(address);
        if (upper == begin()) {
            throw std::runtime_error("The address " +
StringHelper::toHexString(address) + " is out of emulated scope");
        }

        --upper;
        return upper->second;
    }
```

const Segment & bnssemulator::AddressSpace::segment (uint32_t *address*) const`[private]`

Definition at line 228 of file AddressSpace.cpp.

References segment().

```
                                                                  {
        return const_cast<AddressSpace &>(*this).segment(address);
    }
```

void bnssemulator::AddressSpace::set16bitData (uint32_t *address*, uint16_t *data*)

Sets 16 bits of data at the specified address.

Parameters:

address	Address
data	Data

Definition at line 170 of file AddressSpace.cpp.

References set8bitData().

Referenced by bnssemulator::StoreExecuter::execute(), and set32bitData().

```
                                                                     {
        set8bitData(address, static_cast<uint8_t>(data & 0x00ff));
        set8bitData(address + 1, static_cast<uint8_t>((data & 0xff00) >> 8));
    }
```

void bnssemulator::AddressSpace::set32bitData (uint32_t *address*, uint32_t *data*)

Sets 32 bits of data at the specified address.

Parameters:

address	Address

data	Data

Definition at line 175 of file AddressSpace.cpp.

References set16bitData().

Referenced by bnssemulator::StoreExecuter::execute(), and bnssemulator::Context::pushToStack().

```
                                                                    {
        set16bitData(address, static_cast<uint16_t>(data & 0x0000ffff));
        set16bitData(address + 2, static_cast<uint16_t>((data & 0xffff0000) >>
16));
    }
```

void bnssemulator::AddressSpace::set8bitData (uint32_t *address*, uint8_t *data*)

Sets 8 bits of data at the specified address.

Parameters:

address	Address
data	Data

Definition at line 160 of file AddressSpace.cpp.

References segment(), stdin_address_, stdin_read_, stdout_address_, and bnssemulator::Segment::writeData().

Referenced by bnssemulator::StoreExecuter::execute(), set16bitData(), and writeToStdin().

```
                                                                  {
        segment(address).writeData(address, data);
        if (address == stdout_address_) {
            std::cout << data;
        }
        else if (address == stdin_address_) {
            stdin_read_ = false;
        }
    }
```

bool bnssemulator::AddressSpace::stdinRead () const [noexcept]

Check whether the standard input has been read.

Returns:

Whether the standard input has been read

Definition at line 210 of file AddressSpace.cpp.

References stdin_read_.

```
                                              {
        return stdin_read_;
    }
```

size_t bnssemulator::AddressSpace::timerInterrupt () const [noexcept]

Gets the entry of the timer interrupt routine.

Returns:

Entry of the timer interrupt routine

Definition at line 193 of file AddressSpace.cpp.

References timer_interrupt_.

Referenced by bnssemulator::Context::jumpToTimerInterrupt().

```
{
    return timer_interrupt_;
}
```

void bnssemulator::AddressSpace::writeToStdin (char *character*) [noexcept]

Writes a character to stdin.

Definition at line 214 of file AddressSpace.cpp.

References set8bitData(), and stdin_address_.

Referenced by bnssemulator::Context::jumpToKeyboardInterrupt().

```
{
    set8bitData(stdin_address_, character);
}
```

Member Data Documentation

size_t bnssemulator::AddressSpace::error_interrupt_ = 3 [private]

Definition at line 130 of file AddressSpace.h.

Referenced by errorInterrupt().

size_t bnssemulator::AddressSpace::keyboard_interrupt_ = 5 [private]

Definition at line 132 of file AddressSpace.h.

Referenced by keyboardInterrupt().

uint32_t bnssemulator::AddressSpace::stdin_address_ = 132 [private]

Definition at line 128 of file AddressSpace.h.

Referenced by get8bitData(), set8bitData(), and writeToStdin().

bool bnssemulator::AddressSpace::stdin_read_ = true [mutable], [private]

Definition at line 134 of file AddressSpace.h.

Referenced by get8bitData(), set8bitData(), and stdinRead().

uint32_t bnssemulator::AddressSpace::stdout_address_ = 128 [private]

Definition at line 127 of file AddressSpace.h.

Referenced by set8bitData().

size_t bnssemulator::AddressSpace::timer_interrupt_ = 4 [private]

Definition at line 131 of file AddressSpace.h.

Referenced by timerInterrupt().

The documentation for this class was generated from the following files:

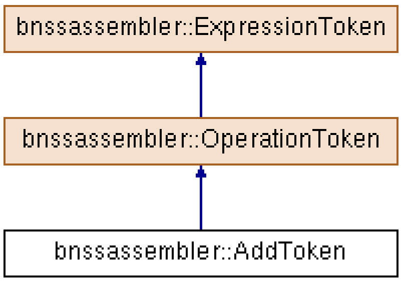
- Code/Emulator/Include/**AddressSpace.h**
- Code/Emulator/Source/**AddressSpace.cpp**

bnssassembler::AddToken Class Reference

Token class representing the + operation.

`#include <AddToken.h>`

Inheritance diagram for bnssassembler::AddToken:



Public Member Functions

- int **inputPriority** () const noexcept override
  Gets the input priority of the token.
- int **stackPriority** () const noexcept override
  Gets the stack priority of the token.
- int **rank** () const noexcept override
  Gets the rank of the token.
- std::string **operation** () const noexcept override
- std::shared_ptr< **Expression** > **create** () const override
  Creates an expression object out of the token.

Protected Member Functions

- std::shared_ptr< **ExpressionToken** > **clone** (std::string param) const override
  Clones the current object, using the string provided.

Detailed Description

Token class representing the + operation.

Definition at line 10 of file AddToken.h.

Member Function Documentation

std::shared_ptr< ExpressionToken > bnssassembler::AddToken::clone (std::string *param*) const `[override]`, `[protected]`, `[virtual]`

Clones the current object, using the string provided.

Parameters:

param	String that will be used to construct the new object

Returns:

Pointer to the cloned object

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 26 of file AddToken.cpp.

```
                                                                        {
        return std::make_shared<AddToken>();
    }
```

std::shared_ptr< Expression > bnssassembler::AddToken::create () const `[override]`, `[virtual]`

Creates an expression object out of the token.

Returns:

Pointer to the expression

Exceptions:

Throws	if the token has no corresponding expression object

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 22 of file AddToken.cpp.

```
                                                    {
        return std::make_shared<AddOperation>();
    }
```

int bnssassembler::AddToken::inputPriority () const `[override]`, `[virtual]`, `[noexcept]`

Gets the input priority of the token.

Returns:

Input priority of the token

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 6 of file AddToken.cpp.

```
                                              {
        return 2;
    }
```

std::string bnssassembler::AddToken::operation () const `[override]`, `[virtual]`, `[noexcept]`

Implements **bnssassembler::OperationToken** (*p.314*).

Definition at line 18 of file AddToken.cpp.

```
                                                {
        return "+";
    }
```

int bnssassembler::AddToken::rank () const `[override]`, `[virtual]`, `[noexcept]`

Gets the rank of the token.

Returns:

Rank of the token

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 14 of file AddToken.cpp.

```
                                      {
        return -1;
    }
```

int bnssassembler::AddToken::stackPriority () const `[override]`, `[virtual]`, `[noexcept]`

Gets the stack priority of the token.

Returns:

Stack priority of the token

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 10 of file AddToken.cpp.

```
                                                   {
        return 2;
    }
```

The documentation for this class was generated from the following files:

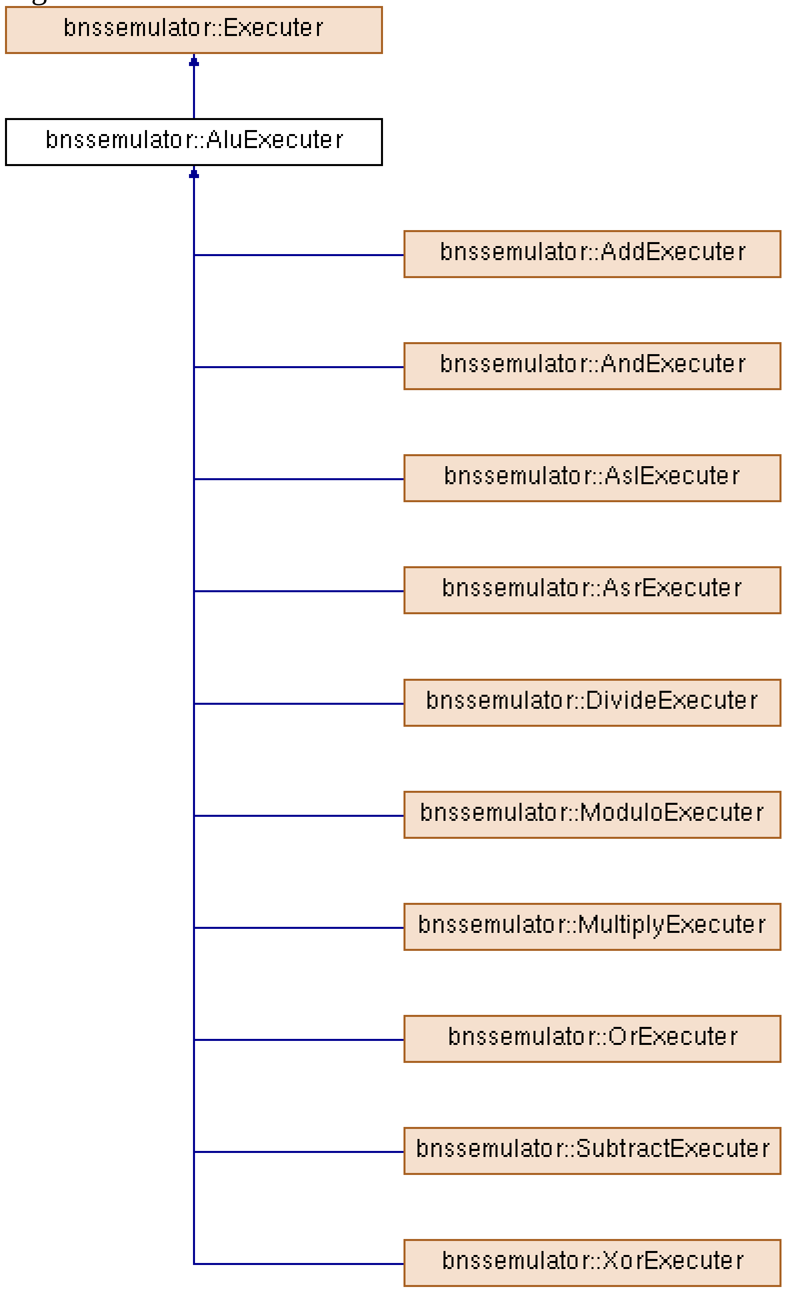
- Code/Assembler/Include/**AddToken.h**
- Code/Assembler/Source/**AddToken.cpp**

bnssemulator::AluExecuter Class Reference

Base class used for executing ALU instructions.

```
#include <AluExecuter.h>
```

Inheritance diagram for bnssemulator::AluExecuter:


bnssemulator::Executer
bnssemulator::AluExecuter
bnssemulator::AddExecuter
bnssemulator::AndExecuter
bnssemulator::AslExecuter
bnssemulator::AsrExecuter
bnssemulator::DivideExecuter
bnssemulator::ModuloExecuter
bnssemulator::MultiplyExecuter
bnssemulator::OrExecuter
bnssemulator::SubtractExecuter
bnssemulator::XorExecuter

Public Member Functions

- void **execute** (**InstructionBitField** instruction, **Context** &context) const override
  Executes the instruction.

Protected Member Functions

- virtual void **execute** (**Register** &dst, const **Register** &lhs, const **Register** &rhs) const =0
  Executes the ALU instruction.

Detailed Description

Base class used for executing ALU instructions.

Definition at line 10 of file AluExecuter.h.

Member Function Documentation

void bnssemulator::AluExecuter::execute (InstructionBitField *instruction*, Context & *context*) const`[override]`, `[virtual]`

Executes the instruction.

Parameters:

instruction	Instruction
context	**Processor** context

Implements **bnssemulator::Executer** (*p.163*).

Definition at line 5 of file AluExecuter.cpp.

References bnssemulator::Context::getRegister(), bnssemulator::InstructionBitField::register0, bnssemulator::InstructionBitField::register1, and bnssemulator::InstructionBitField::register2.

```
{
        auto &dst = context.getRegister(instruction.register0);
        auto &lhs = context.getRegister(instruction.register1);
        auto &rhs = context.getRegister(instruction.register2);

        execute(dst, lhs, rhs);
    }
```

virtual void bnssemulator::AluExecuter::execute (Register & *dst*, const Register & *lhs*, const Register & *rhs*) const`[protected]`, `[pure virtual]`

Executes the ALU instruction.

Parameters:

dst	Reference to the destination register
lhs	Left operand register
rhs	Right operand register

Implemented in **bnssemulator::AddExecuter** (*p.85*), **bnssemulator::AndExecuter** (*p.104*), **bnssemulator::AslExecuter** (*p.106*), **bnssemulator::AsrExecuter** (*p.107*), **bnssemulator::DivideExecuter** (*p.154*), **bnssemulator::ModuloExecuter** (*p.287*), **bnssemulator::MultiplyExecuter** (*p.289*), **bnssemulator::OrExecuter** (*p.347*), **bnssemulator::SubtractExecuter** (*p.472*), and **bnssemulator::XorExecuter** (*p.522*).

The documentation for this class was generated from the following files:

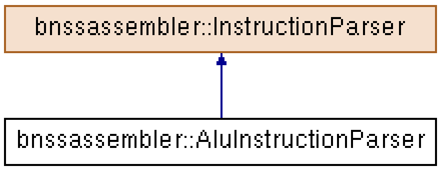
- Code/Emulator/Include/**AluExecuter.h**
- Code/Emulator/Source/**AluExecuter.cpp**

bnssassembler::AluInstructionParser Class Reference

Class representing the parser for ALU instructions.

`#include <AluInstructionParser.h>`

Inheritance diagram for bnssassembler::AluInstructionParser:


bnssassembler::InstructionParser
bnssassembler::AluInstructionParser

Public Member Functions

- **AluInstructionParser** () noexcept
  *Constructs an **AluInstructionParser** object.*

Additional Inherited Members

Detailed Description

Class representing the parser for ALU instructions.

Definition at line 10 of file AluInstructionParser.h.

Constructor & Destructor Documentation

bnssassembler::AluInstructionParser::AluInstructionParser () [noexcept]

Constructs an **AluInstructionParser** object.

Definition at line 6 of file AluInstructionParser.cpp.

References bnssassembler::InstructionParser::operands_.

```
                                                        {
        operands_.push_back(std::make_shared<RegisterDirectParser>());
        operands_.push_back(std::make_shared<RegisterDirectParser>());
        operands_.push_back(std::make_shared<RegisterDirectParser>());
    }
```

The documentation for this class was generated from the following files:

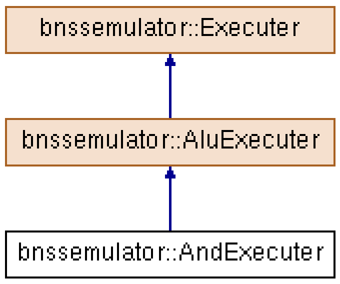
- Code/Assembler/Include/**AluInstructionParser.h**
- Code/Assembler/Source/**AluInstructionParser.cpp**

bnssemulator::AndExecuter Class Reference

Class representing the executer for the and instruction.

`#include <AndExecuter.h>`

Inheritance diagram for bnssemulator::AndExecuter:



Protected Member Functions

- void **execute** (**Register** &dst, const **Register** &lhs, const **Register** &rhs) const override
  Executes the ALU instruction.

Additional Inherited Members

Detailed Description

Class representing the executer for the and instruction.

Definition at line 10 of file AndExecuter.h.

Member Function Documentation

void bnssemulator::AndExecuter::execute (Register & *dst*, const Register & *lhs*, const Register & *rhs*) const `[override]`, `[protected]`, `[virtual]`

Executes the ALU instruction.

Parameters:

dst	Reference to the destination register
lhs	Left operand register
rhs	Right operand register

Implements **bnssemulator::AluExecuter** (*p.102*).

Definition at line 5 of file AndExecuter.cpp.

```
    5
{
    6          dst = lhs & rhs;
    7      }
```

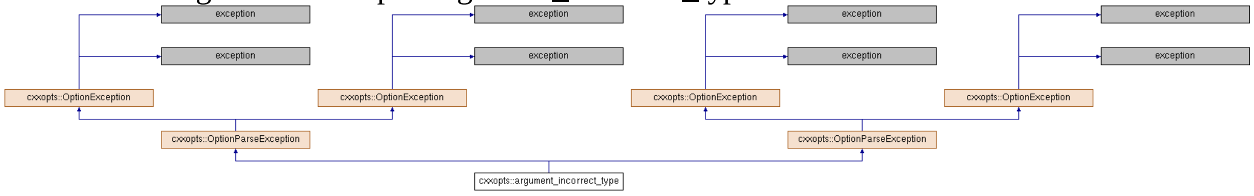
The documentation for this class was generated from the following files:

- Code/Emulator/Include/**AndExecuter.h**
- Code/Emulator/Source/**AndExecuter.cpp**

cxxopts::argument_incorrect_type Class Reference

`#include <cxxopts.h>`

Inheritance diagram for cxxopts::argument_incorrect_type:


exception
exception
cxxopts::OptionException
cxxopts::OptionParseException
cxxopts::argument_incorrect_type

Public Member Functions

- **argument_incorrect_type** (const std::string &arg)
- **argument_incorrect_type** (const std::string &arg)

Detailed Description

Definition at line 382 of file cxxopts.h.

Constructor & Destructor Documentation

cxxopts::argument_incorrect_type::argument_incorrect_type (const std::string & *arg*) `[inline]`

Definition at line 386 of file cxxopts.h.

```
                : OptionParseException(
                    "Argument '" + arg + "' failed to parse"
                )
            {
            }
```

cxxopts::argument_incorrect_type::argument_incorrect_type (const std::string & *arg*) `[inline]`

Definition at line 386 of file cxxopts.h.

```
                : OptionParseException(
                    "Argument '" + arg + "' failed to parse"
                )
            {
            }
```

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

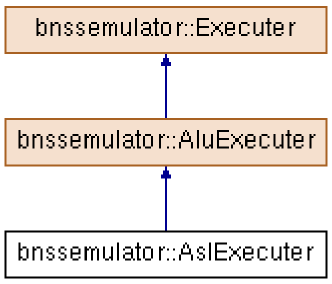
- Code/Assembler/Include/**cxxopts.h**

bnssemulator::AslExecuter Class Reference

Class representing the executer for the asl instruction.

`#include <AslExecuter.h>`

Inheritance diagram for bnssemulator::AslExecuter:



Protected Member Functions

- void **execute** (**Register** &dst, const **Register** &lhs, const **Register** &rhs) const override
  Executes the ALU instruction.

Additional Inherited Members

Detailed Description

Class representing the executer for the asl instruction.

Definition at line 10 of file AslExecuter.h.

Member Function Documentation

void bnssemulator::AslExecuter::execute (Register & *dst*, const Register & *lhs*, const Register & *rhs*) const `[override]`, `[protected]`, `[virtual]`

Executes the ALU instruction.

Parameters:

dst	Reference to the destination register
lhs	Left operand register
rhs	Right operand register

Implements **bnssemulator::AluExecuter** (*p.102*).

Definition at line 5 of file AslExecuter.cpp.

```
{
            dst = lhs << rhs;
        }
```

The documentation for this class was generated from the following files:

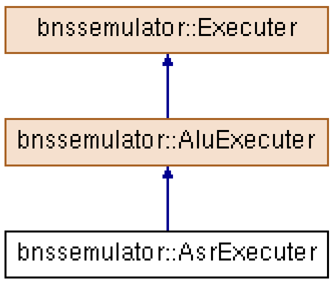
- Code/Emulator/Include/**AslExecuter.h**
- Code/Emulator/Source/**AslExecuter.cpp**

bnssemulator::AsrExecuter Class Reference

Class representing the executer for the asr instruction.

`#include <AsrExecuter.h>`

Inheritance diagram for bnssemulator::AsrExecuter:



Protected Member Functions

- void **execute** (**Register** &dst, const **Register** &lhs, const **Register** &rhs) const override
  Executes the ALU instruction.

Additional Inherited Members

Detailed Description

Class representing the executer for the asr instruction.

Definition at line 10 of file AsrExecuter.h.

Member Function Documentation

void bnssemulator::AsrExecuter::execute (Register & *dst*, const Register & *lhs*, const Register & *rhs*) const [override], [protected], [virtual]

Executes the ALU instruction.

Parameters:

dst	Reference to the destination register
lhs	Left operand register
rhs	Right operand register

Implements **bnssemulator::AluExecuter** (*p.102*).

Definition at line 5 of file AsrExecuter.cpp.

```
{
            dst = lhs >> rhs;
        }
```

The documentation for this class was generated from the following files:

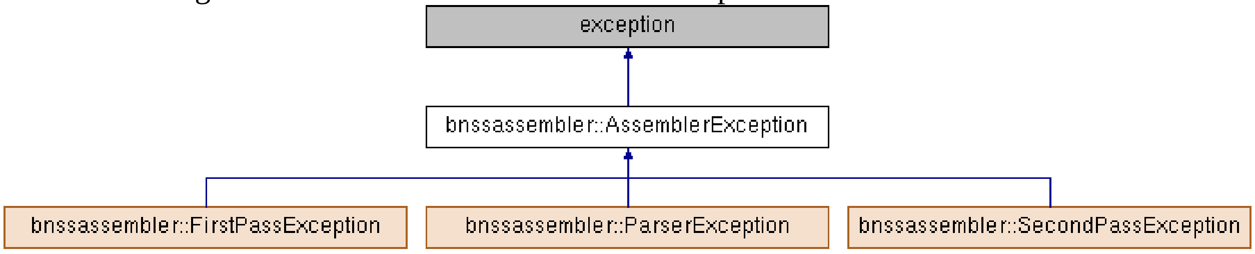
- Code/Emulator/Include/**AsrExecuter.h**
- Code/Emulator/Source/**AsrExecuter.cpp**

bnssassembler::AssemblerException Class Reference

Class representing the custom exception for the assembler.

`#include <AssemblerException.h>`

Inheritance diagram for bnssassembler::AssemblerException:


exception
bnssassembler::AssemblerException
bnssassembler::FirstPassException
bnssassembler::ParserException
bnssassembler::SecondPassException

Public Member Functions

- **AssemblerException** (size_t line_number, std::string line) noexcept
  *Constructs an **AssemblerException** object.*
- std::string **message** () const noexcept
  *Gets the message. Note that this should be used instead of **what()***
- const char * **what** () const noexcept override

Protected Member Functions

- virtual std::string **messageBody** () const noexcept=0
  Returns the actual message body of the exception.

Private Attributes

- size_t **line_number_**
- std::string **line_**

Detailed Description

Class representing the custom exception for the assembler.

Definition at line 11 of file AssemblerException.h.

Constructor & Destructor Documentation

bnssassembler::AssemblerException::AssemblerException (size_t *line_number*, std::string *line*) `[explicit]`, `[noexcept]`

Constructs an **AssemblerException** object.

Parameters:

line_number	Number of the line in the source file which triggered the exception
line	Line of the source file which triggered the exception

Definition at line 6 of file AssemblerException.cpp.

```
    6 : line_number_(line_number), line_(line) {}
```

Member Function Documentation

std::string bnssassembler::AssemblerException::message () const [noexcept]

Gets the message. Note that this should be used instead of **what()**

Returns:

The message of the exception

Definition at line 8 of file AssemblerException.cpp.

References line_, line_number_, messageBody(), and bnssassembler::StringHelper::numberFormat().

Referenced by main(), and what().

```
                                                          {
        return
            "Error in line " + StringHelper::numberFormat(line_number_) + "\n"
+
            line_ + "\n" +
            messageBody();
    }
```

virtual std::string bnssassembler::AssemblerException::messageBody () const [protected], [pure virtual], [noexcept]

Returns the actual message body of the exception.

Implemented in **bnssassembler::FirstPassException** (*p.191*), **bnssassembler::ParserException** (*p.357*), and **bnssassembler::SecondPassException** (*p.414*).

Referenced by message().

const char * bnssassembler::AssemblerException::what () const [override], [noexcept]

Definition at line 15 of file AssemblerException.cpp.

References message().

```
                                                          {
        return message().c_str();
    }
```

Member Data Documentation

std::string bnssassembler::AssemblerException::line_ [private]

Definition at line 33 of file AssemblerException.h.

Referenced by message().

size_t bnssassembler::AssemblerException::line_number_ [private]

Definition at line 32 of file AssemblerException.h.

Referenced by message().

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**AssemblerException.h**
- Code/Assembler/Source/**AssemblerException.cpp**

bnssemulator::AssemblerOutput Class Reference

Class representing the output from the assembler.
`#include <AssemblerOutput.h>`

Public Member Functions

- std::vector< **SectionData** > & **sectionTable** () noexcept
  Gets the section table.
- const std::vector< **SectionData** > & **sectionTable** () const noexcept
  Gets the section table.
- std::unordered_map< std::string, **SymbolData** > & **symbolTable** () noexcept
  Gets the symbol table.
- const std::unordered_map< std::string, **SymbolData** > & **symbolTable** () const noexcept
  Gets the symbol table.
- bool **importedSymbolsExist** () const noexcept
  Checks if there are imported symbols.
- std::vector< std::string > **importedSymbolsAsVector** () const noexcept
  Gets the imported symbols as a vector of strings.
- **uint32_t startOfProgram** (std::string start_symbol) const
  Gets the address of the start of the program.

Private Attributes

- std::unordered_set< std::string > **imported_symbols_**
- std::vector< **SectionData** > **section_table_**
- std::unordered_map< std::string, **SymbolData** > **symbol_table_**

Friends

- std::istream & **operator>>** (std::istream &is, **AssemblerOutput** &data)
  Loads the object from stream.

Detailed Description

Class representing the output from the assembler.

Definition at line 15 of file AssemblerOutput.h.

Member Function Documentation

std::vector< std::string > bnssemulator::AssemblerOutput::importedSymbolsAsVector () const`[noexcept]`

Gets the imported symbols as a vector of strings.

Returns:

Imported symbols as a vector of strings

Definition at line 55 of file AssemblerOutput.cpp.

References imported_symbols_.

```
{
        std::vector<std::string> ret;
        for (auto &symbol : imported_symbols_) {
            ret.push_back(symbol);
        }

        return ret;
    }
```

bool bnssemulator::AssemblerOutput::importedSymbolsExist () const `[noexcept]`

Checks if there are imported symbols.

Returns:

Whether there are imported symbols

Definition at line 51 of file AssemblerOutput.cpp.

References imported_symbols_.

```
{
        return imported_symbols_.size() != 0;
    }
```

std::vector< SectionData > & bnssemulator::AssemblerOutput::sectionTable ()`[noexcept]`

Gets the section table.

Returns:

Section table

Definition at line 35 of file AssemblerOutput.cpp.

References section_table_.

Referenced by sectionTable().

```
{
        return section_table_;
    }
```

const std::vector< SectionData > & bnssemulator::AssemblerOutput::sectionTable () const`[noexcept]`

Gets the section table.

Returns:

Section table

Definition at line 39 of file AssemblerOutput.cpp.

References sectionTable().

```
{
        return const_cast<AssemblerOutput &>(*this).sectionTable();
    }
```

uint32_t bnssemulator::AssemblerOutput::startOfProgram (std::string *start_symbol*) const

Gets the address of the start of the program.

Parameters:

start_symbol	Symbol representing the start of the program

Returns:

Address of the start of program

Exceptions:

Throws	if there is no start of program

Definition at line 64 of file AssemblerOutput.cpp.

References section_table_, and symbol_table_.

```
                                                                             {
           if (symbol_table_.count(start_symbol) == 0) {
               throw MessageException("The " + start_symbol + " symbol is not
defined");
           }

           auto symbol = symbol_table_.at(start_symbol);
           return section_table_[symbol.sectionIndex()].address() +
symbol.offset();
       }
```

std::unordered_map< std::string, SymbolData > & bnssemulator::AssemblerOutput::symbolTable () [noexcept]

Gets the symbol table.

Returns:

Symbol table

Definition at line 43 of file AssemblerOutput.cpp.

References symbol_table_.

Referenced by symbolTable().

```
{
           return symbol_table_;
       }
```

const std::unordered_map< std::string, SymbolData > & bnssemulator::AssemblerOutput::symbolTable () const [noexcept]

Gets the symbol table.

Returns:

Symbol table

Definition at line 47 of file AssemblerOutput.cpp.

References symbolTable().

```
{
           return const_cast<AssemblerOutput &>(*this).symbolTable();
       }
```

Friends And Related Function Documentation

std::istream& operator>> (std::istream & *is*, AssemblerOutput & *data*) [friend]

Loads the object from stream.

Parameters:

is	Input stream
data	Reference to the object that should be loaded

Returns:

Input stream

Definition at line 7 of file AssemblerOutput.cpp.

```
                                                                     {
       size_t num_of_imported_symbols;
       is >> num_of_imported_symbols;
       for (size_t i = 0; i < num_of_imported_symbols; i++) {
           std::string symbol;
           is >> symbol;
           data.imported_symbols_.insert(symbol);
       }

       size_t section_table_size;
       is >> section_table_size;
       for (size_t i = 0; i < section_table_size; i++) {
           SectionData section;
           is >> section;
           data.section_table_.push_back(section);
       }

       size_t symbol_table_size;
       is >> symbol_table_size;
       for (size_t i = 0; i < symbol_table_size; i++) {
           SymbolData symbol;
           is >> symbol;
           data.symbol_table_[symbol.name()] = symbol;
       }

       return is;
   }
```

Member Data Documentation

std::unordered_set<std::string> bnssemulator::AssemblerOutput::imported_symbols_ `[private]`

Definition at line 69 of file AssemblerOutput.h.

Referenced by importedSymbolsAsVector(), importedSymbolsExist(), and bnssemulator::operator>>().

std::vector<SectionData> bnssemulator::AssemblerOutput::section_table_ `[private]`

Definition at line 70 of file AssemblerOutput.h.

Referenced by bnssemulator::operator>>(), sectionTable(), and startOfProgram().

std::unordered_map<std::string, SymbolData> bnssemulator::AssemblerOutput::symbol_table_ `[private]`

Definition at line 71 of file AssemblerOutput.h.

Referenced by bnssemulator::operator>>(), startOfProgram(), and symbolTable().

The documentation for this class was generated from the following files:

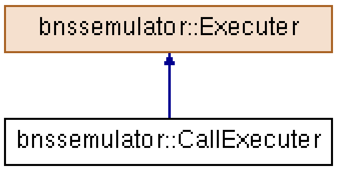
- Code/Emulator/Include/**AssemblerOutput.h**
- Code/Emulator/Source/**AssemblerOutput.cpp**

bnssemulator::CallExecuter Class Reference

Class representing the executer for the call instruction.

`#include <CallExecuter.h>`

Inheritance diagram for bnssemulator::CallExecuter:



Public Member Functions

- void **execute** (**InstructionBitField** instruction, **Context** &context) const override
  Executes the instruction.

Detailed Description

Class representing the executer for the call instruction.

Definition at line 10 of file CallExecuter.h.

Member Function Documentation

void bnssemulator::CallExecuter::execute (InstructionBitField *instruction*, Context & *context*) const `[override]`, `[virtual]`

Executes the instruction.

Parameters:

instruction	Instruction
context	**Processor** context

Implements **bnssemulator::Executer** (*p.163*).

Definition at line 5 of file CallExecuter.cpp.

References bnssemulator::Context::getOperandAddress(), and bnssemulator::Context::jumpToSubroutine().

```
{
        auto address = context.getOperandAddress(instruction, 0);
        context.jumpToSubroutine(address);
    }
```

The documentation for this class was generated from the following files:

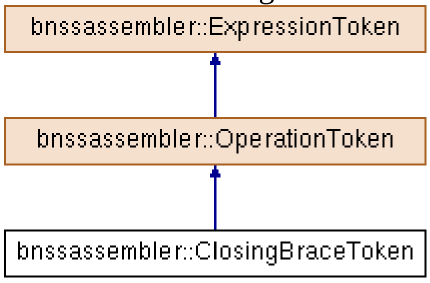
- Code/Emulator/Include/**CallExecuter.h**
- Code/Emulator/Source/**CallExecuter.cpp**

bnssassembler::ClosingBraceToken Class Reference

Token class representing the opening brace.

`#include <ClosingBraceToken.h>`

Inheritance diagram for bnssassembler::ClosingBraceToken:


bnssassembler::ExpressionToken
bnssassembler::OperationToken
bnssassembler::ClosingBraceToken

Public Member Functions

- int **inputPriority** () const noexcept override
  Gets the input priority of the token.
- int **stackPriority** () const noexcept override
  Gets the stack priority of the token.
- int **rank** () const noexcept override
  Gets the rank of the token.
- std::string **operation** () const noexcept override
- std::shared_ptr< **Expression** > **create** () const override
  Creates an expression object out of the token.

Protected Member Functions

- std::shared_ptr< **ExpressionToken** > **clone** (std::string param) const override
  Clones the current object, using the string provided.
- bool **isClosingBrace** () const noexcept override
  Checks if the operator is the closing brace (closing brace should not be on the stack)

Detailed Description

Token class representing the opening brace.

Definition at line 10 of file ClosingBraceToken.h.

Member Function Documentation

std::shared_ptr< ExpressionToken > bnssassembler::ClosingBraceToken::clone (std::string *param*) const `[override]`, `[protected]`, `[virtual]`

Clones the current object, using the string provided.

Parameters:

param	String that will be used to construct the new object

Returns:

Pointer to the cloned object

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 26 of file ClosingBraceToken.cpp.

```
{
        return std::make_shared<ClosingBraceToken>();
    }
```

std::shared_ptr< Expression > bnssassembler::ClosingBraceToken::create () const`[override]`, `[virtual]`

Creates an expression object out of the token.

Returns:

Pointer to the expression

Exceptions:

Throws	if the token has no corresponding expression object

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 22 of file ClosingBraceToken.cpp.

```
                                                             {
        throw MessageException("Internal error - Closing brace in postfix");
    }
```

int bnssassembler::ClosingBraceToken::inputPriority () const`[override]`, `[virtual]`, `[noexcept]`

Gets the input priority of the token.

Returns:

Input priority of the token

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 6 of file ClosingBraceToken.cpp.

```
                                                       {
        return 1;
    }
```

bool bnssassembler::ClosingBraceToken::isClosingBrace () const`[override]`, `[protected]`, `[virtual]`, `[noexcept]`

Checks if the operator is the closing brace (closing brace should not be on the stack)

Returns:

Whether the operator is the closing brace

Reimplemented from **bnssassembler::OperationToken** (*p.314*).

Definition at line 30 of file ClosingBraceToken.cpp.

```
                                                         {
        return true;
    }
```

std::string bnssassembler::ClosingBraceToken::operation () const`[override]`, `[virtual]`, `[noexcept]`

Implements **bnssassembler::OperationToken** (*p.314*).

Definition at line 18 of file ClosingBraceToken.cpp.

```
                                                    {
        return ")";
    }
```

int bnssassembler::ClosingBraceToken::rank () const `[override]`, `[virtual]`, `[noexcept]`

Gets the rank of the token.

Returns:

Rank of the token

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 14 of file ClosingBraceToken.cpp.

```
                                          {
        return 0;
    }
```

int bnssassembler::ClosingBraceToken::stackPriority () const `[override]`, `[virtual]`, `[noexcept]`

Gets the stack priority of the token.

Returns:

Stack priority of the token

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 10 of file ClosingBraceToken.cpp.

```
                                                  {
        return 0;
    }
```

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**ClosingBraceToken.h**
- Code/Assembler/Source/**ClosingBraceToken.cpp**

bnssemulator::CommandLineHelper Class Reference

Utility class used for parsing the command line.
`#include <CommandLineHelper.h>`

Static Public Member Functions

- static std::string **parse** (int argc, char *argv[])
  Parses the command line.

Private Member Functions

- **CommandLineHelper** ()=delete
- **CommandLineHelper** (**CommandLineHelper** &)=delete
- void **operator=** (**CommandLineHelper** &)=delete

Detailed Description

Utility class used for parsing the command line.

Definition at line 11 of file CommandLineHelper.h.

Constructor & Destructor Documentation

bnssemulator::CommandLineHelper::CommandLineHelper () `[private]`, `[delete]`

bnssemulator::CommandLineHelper::CommandLineHelper (CommandLineHelper &) `[private]`, `[delete]`

Member Function Documentation

void bnssemulator::CommandLineHelper::operator= (CommandLineHelper &) `[private]`, `[delete]`

std::string bnssemulator::CommandLineHelper::parse (int *argc*, char * *argv*[]) `[static]`

Parses the command line.

Parameters:

argc	Arguments count
argv	Arguments vector

Returns:

Input file name

Definition at line 7 of file CommandLineHelper.cpp.

References cxxopts::Options::add_options(), cxxopts::Options::count(), cxxopts::Options::help(), and cxxopts::Options::parse().

Referenced by main().

```
    7                                                                                  {
        cxxopts::Options options(argv[0], "Emulator\nSystem software\nSchool of Electrical Engineering\nUniversity of Belgrade\nCopyright (c) 2017 Nikola Bebic\n");
        options.add_options()
            ("i,input", "Specifies input file", cxxopts::value<std::string>()->default_value("out.out"))
            ("h,help", "Prints help");

        options.parse(argc, argv);

        if (options.count("help")) {
            std::cout << options.help() << std::endl;
            exit(0);
        }

        return options["input"].as<std::string>();
    }
```

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**CommandLineHelper.h**
- Code/Emulator/Source/**CommandLineHelper.cpp**

bnssassembler::CommandLineHelper Class Reference

Utility class used to parse the command line.
`#include <CommandLineHelper.h>`

Static Public Member Functions

- static std::pair< std::string, std::string > **parse** (int argc, char *argv[])
  Parses the command line.

Private Member Functions

- **CommandLineHelper** ()=delete
- **CommandLineHelper** (**CommandLineHelper** &)=delete
- void **operator=** (**CommandLineHelper** &)=delete

Detailed Description

Utility class used to parse the command line.

Definition at line 11 of file CommandLineHelper.h.

Constructor & Destructor Documentation

bnssassembler::CommandLineHelper::CommandLineHelper () `[private]`, `[delete]`

bnssassembler::CommandLineHelper::CommandLineHelper (CommandLineHelper &) `[private]`, `[delete]`

Member Function Documentation

void bnssassembler::CommandLineHelper::operator= (CommandLineHelper &) `[private]`, `[delete]`

std::pair< std::string, std::string > bnssassembler::CommandLineHelper::parse (int *argc*, char * *argv*[]) `[static]`

Parses the command line.

Parameters:

argc	Arguments count
argv	Arguments vector

Returns:

Pair of strings - input and output file names

Definition at line 7 of file CommandLineHelper.cpp.

References cxxopts::Options::add_options(), cxxopts::Options::count(), cxxopts::Options::help(), and cxxopts::Options::parse().

Referenced by main().

```
      7
{
    cxxopts::Options options(argv[0], "Assembler\nSystem software\nSchool of Electrical Engineering\nUniversity of Belgrade\nCopyright (c) 2017 Nikola Bebic\n");
    options.add_options()
        ("i,input", "Specifies input file", cxxopts::value<std::string>()->default_value("in.ss"))
        ("o,output", "Specifies output file", cxxopts::value<std::string>()->default_value("out.out"))
        ("h,help", "Prints help");

    options.parse(argc, argv);

    if (options.count("help")) {
        std::cout << options.help() << std::endl;
        exit(0);
    }

    return make_pair(options["input"].as<std::string>(), options["output"].as<std::string>());
}
```

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**CommandLineHelper.h**
- Code/Assembler/Source/**CommandLineHelper.cpp**

bnssemulator::compare_pair_difference< T, Pred > Struct Template Reference

`#include <StlHelper.h>`

Public Member Functions

- bool **operator()** (const std::pair< T, T > &left, const std::pair< T, T > &right) const

---

Detailed Description

template<class T, class Pred = std::less<T>>

struct bnssemulator::compare_pair_difference< T, Pred >

Definition at line 24 of file StlHelper.h.

Member Function Documentation

template<class T , class Pred = std::less<T>> bool bnssemulator::compare_pair_difference< T, Pred >::operator() (const std::pair< T, T > & *left*, const std::pair< T, T > & *right*) const `[inline]`

Definition at line 25 of file StlHelper.h.

```
{
            Pred p;
            return p(left.second - left.first, right.second - right.first);
        }
```

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

- Code/Emulator/Include/**StlHelper.h**

bnssemulator::compare_pair_first< T1, T2, Pred > Struct Template Reference

```
#include <StlHelper.h>
```

Public Member Functions

- bool **operator()** (const std::pair< T1, T2 > &left, const std::pair< T1, T2 > &right) const

---

Detailed Description

template<class T1, class T2, class Pred = std::less<T1>>

struct bnssemulator::compare_pair_first< T1, T2, Pred >

Definition at line 8 of file StlHelper.h.

Member Function Documentation

template<class T1 , class T2 , class Pred = std::less<T1>> bool bnssemulator::compare_pair_first< T1, T2, Pred >::operator() (const std::pair< T1, T2 > & *left*, const std::pair< T1, T2 > & *right*) const `[inline]`

Definition at line 9 of file StlHelper.h.

```
{
            Pred p;
            return p(left.first, right.first);
        }
```

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

- Code/Emulator/Include/**StlHelper.h**

bnssemulator::compare_pair_second< T1, T2, Pred > Struct Template Reference

`#include <StlHelper.h>`

Public Member Functions

- bool **operator()** (const std::pair< T1, T2 > &left, const std::pair< T1, T2 > &right) const

---

Detailed Description

template<class T1, class T2, class Pred = std::less<T2>>

struct bnssemulator::compare_pair_second< T1, T2, Pred >

Definition at line 16 of file StlHelper.h.

Member Function Documentation

template<class T1 , class T2 , class Pred = std::less<T2>> bool bnssemulator::compare_pair_second< T1, T2, Pred >::operator() (const std::pair< T1, T2 > & *left*, const std::pair< T1, T2 > & *right*) const`[inline]`

Definition at line 17 of file StlHelper.h.

```
{
                Pred p;
                return p(left.second, right.second);
        }
```

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

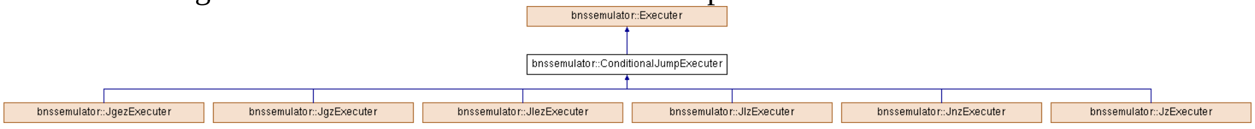
- Code/Emulator/Include/**StlHelper.h**

bnssemulator::ConditionalJumpExecuter Class Reference

Base executer for conditional jump instructions.

`#include <ConditionalJumpExecuter.h>`

Inheritance diagram for bnssemulator::ConditionalJumpExecuter:


bnssemulator::Executer
bnssemulator::ConditionalJumpExecuter
bnssemulator::JgezExecuter
bnssemulator::JgzExecuter
bnssemulator::JlezExecuter
bnssemulator::JlzExecuter
bnssemulator::JnzExecuter
bnssemulator::JzExecuter

Public Member Functions

- void **execute** (**InstructionBitField** instruction, **Context** &context) const override
  Executes the instruction.

Protected Member Functions

- virtual bool **test** (bool negative, bool zero, bool overflow, bool carry) const noexcept=0
  Tests whether the jump should happen.

Detailed Description

Base executer for conditional jump instructions.

Definition at line 10 of file ConditionalJumpExecuter.h.

Member Function Documentation

void bnssemulator::ConditionalJumpExecuter::execute (InstructionBitField *instruction*, Context & *context*) const`[override]`, `[virtual]`

Executes the instruction.

Parameters:

instruction	Instruction
context	**Processor** context

Implements **bnssemulator::Executer** (*p.163*).

Definition at line 5 of file ConditionalJumpExecuter.cpp.

References bnssemulator::Context::getOperandAddress(), bnssemulator::Context::getRegister(), bnssemulator::Context::jumpTo(), bnssemulator::InstructionBitField::register0, and test().

```
{
        auto address = context.getOperandAddress(instruction, 1);
        auto &reg = context.getRegister(instruction.register0);

        if (test(reg.negativeFlag(), reg.zeroFlag(), reg.overflowFlag(),
reg.carryFlag())) {
            context.jumpTo(address);
        }
    }
```

virtual bool bnssemulator::ConditionalJumpExecuter::test (bool *negative*, bool *zero*, bool *overflow*, bool *carry*) const`[protected]`, `[pure virtual]`, `[noexcept]`

Tests whether the jump should happen.

Parameters:

negative	Negative flag of the register
zero	Zero flag of the register
overflow	Overflow flag of the register
carry	Carry flag of the register

Returns:

Whether the jump should happen

Implemented in **bnssemulator::JgezExecuter** (*p.240*), **bnssemulator::JgzExecuter** (*p.242*), **bnssemulator::JlezExecuter** (*p.244*), **bnssemulator::JlzExecuter** (*p.246*), **bnssemulator::JnzExecuter** (*p.249*), and **bnssemulator::JzExecuter** (*p.251*).

Referenced by execute().

The documentation for this class was generated from the following files:

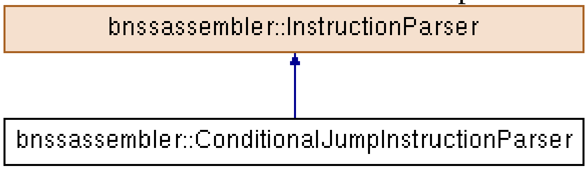
- Code/Emulator/Include/**ConditionalJumpExecuter.h**
- Code/Emulator/Source/**ConditionalJumpExecuter.cpp**

bnssassembler::ConditionalJumpInstructionParser Class Reference

Class representing the parser for conditional jump instructions.

`#include <ConditionalJumpInstructionParser.h>`

Inheritance diagram for bnssassembler::ConditionalJumpInstructionParser:


bnssassembler::InstructionParser
bnssassembler::ConditionalJumpInstructionParser

Public Member Functions

- **ConditionalJumpInstructionParser** () noexcept
  Constructs a ***ConditionalJumpInstructionParser*** *object.*

Additional Inherited Members

Detailed Description

Class representing the parser for conditional jump instructions.

Definition at line 10 of file ConditionalJumpInstructionParser.h.

Constructor & Destructor Documentation

bnssassembler::ConditionalJumpInstructionParser::ConditionalJumpInstructionParser () `[noexcept]`

Constructs a **ConditionalJumpInstructionParser** object.

Definition at line 9 of file ConditionalJumpInstructionParser.cpp.

References bnssassembler::InstructionParser::operands_.

```
{
        operands_.push_back(std::make_shared<RegisterDirectParser>());
        auto memdir = std::make_shared<MemoryDirectParser>();
        auto regindpom = std::make_shared<RegisterIndirectOffsetParser>();
        auto regind = std::make_shared<RegisterIndirectParser>();

        memdir->next(regindpom);
        regindpom->next(regind);

        operands_.push_back(memdir);
    }
```

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**ConditionalJumpInstructionParser.h**
- Code/Assembler/Source/**ConditionalJumpInstructionParser.cpp**

bnssemulator::Context Class Reference

Class representing the context of the processor.
`#include <Context.h>`

Public Member Functions

- **Context** (**AssemblerOutput** &&assembler_output)
  *Constructs a **Context** object.*
- void **pushToStack** (int32_t value)
  Pushes a value to the stack.
- int32_t **popFromStack** ()
  Pops a value from the stack.
- **Register** & **getRegister** (size_t index) noexcept
  Gets the reference to the register.
- const **Register** & **getRegister** (size_t index) const noexcept
  Gets the reference to the register.
- **AddressSpace** & **addressSpace** () noexcept
  Gets the address space.
- const **AddressSpace** & **addressSpace** () const noexcept
  Gets the address space.
- void **jumpTo** (**uint32_t** address)
  Jumps to address.
- void **jumpToSubroutine** (**uint32_t** address)
  Jumps to subroutine at address.
- void **jumpToInterrupt** (size_t entry)
  Jumps to interrupt routine at the specified entry.
- void **returnFromSubroutine** ()
  Returns from subroutine.
- int32_t **getOperand** (**InstructionBitField** instruction, size_t register_index, size_t num_of_bytes=4)
  Gets the operand based on the instruction.
- int32_t **getOperandAddress** (**InstructionBitField** instruction, size_t register_index)
  Gets the address of the operand based on the instruction.
- **InstructionBitField getInstruction** ()
  Gets the instruction to execute.
- int32_t **getSecondWordOfInstruction** ()
  Gets the second word of the instruction.
- void **finishProgram** () noexcept
  Finishes the program.
- bool **programFinished** () const noexcept
  Checks whether the program finished.
- bool **insideInterrupt** () const noexcept
  Checks whether the program is executing an interrupt routine.
- void **pressCharacter** (char character) noexcept
  Presses a character.
- bool **hasCharacters** () const noexcept
  Checks whether there are any characters that should be written to stdin.
- char **getCharacter** () noexcept

Gets the first character to be written to stdin.

- bool **timerTriggered** () const noexcept
  Checks if the timer was triggered.
- void **timerTriggered** (bool value) noexcept
  Sets the timer triggered flag.
- void **jumpToErrorInterrupt** () noexcept
  Jumps to error interrupt.
- void **jumpToTimerInterrupt** () noexcept
  Jumps to timer interrupt.
- void **jumpToKeyboardInterrupt** () noexcept
  Jumps to keyboard interrupt.

Private Attributes

- **AddressSpace address_space_**
- std::vector< **Register** > **registers_**
- **Register** & **stack_pointer_**
- **Register** & **program_counter_**
- bool **inside_interrupt_** = false
- size_t **interrupt_call_stack_depth_** = 0
- std::mutex **characters_mutex_**
- std::queue< char > **characters_**
- std::mutex **timer_mutex_**
- bool **timer_triggered_** = false
- bool **end_of_program_** = false

Detailed Description

Class representing the context of the processor.

Definition at line 15 of file Context.h.

Constructor & Destructor Documentation

bnssemulator::Context::Context (AssemblerOutput && *assembler_output*) `[explicit]`

Constructs a **Context** object.

Parameters:

assembler_output	R-value reference to the assembler output object

Exceptions:

Throws	if the context can not be constructed

Definition at line 8 of file Context.cpp.

References address_space_, bnssemulator::AddressSpace::initialStackPointer(), bnssemulator::StringHelper::join(), program_counter_, stack_pointer_, and bnssemulator::Register::value().

```
    8                                                       : 
address_space_(move(assembler_output.sectionTable()), 
assembler_output.symbolTable()), registers_(18), stack_pointer_(registers_[16]), 
program_counter_(registers_[17]) {
    9         if (assembler_output.importedSymbolsExist()) {
                  throw MessageException("Can not resolve imported symbols: " + StringHelper::join(assembler_output.importedSymbolsAsVector(), ", "));
              }

              stack_pointer_.value(address_space_.initialStackPointer());
              program_counter_.value(assembler_output.startOfProgram("_start"));
          }
```

Member Function Documentation

AddressSpace & bnssemulator::Context::addressSpace () `[noexcept]`

Gets the address space.

Returns:

Address space

Definition at line 37 of file Context.cpp.

References address_space_.

Referenced by addressSpace(), and bnssemulator::StoreExecuter::execute().

```
                                                {
        return address_space_;
    }
```

const AddressSpace & bnssemulator::Context::addressSpace () const `[noexcept]`

Gets the address space.

Returns:

Address space

Definition at line 41 of file Context.cpp.

References addressSpace().

```
                                                            {
        return const_cast<Context &>(*this).addressSpace();
    }
```

void bnssemulator::Context::finishProgram () `[noexcept]`

Finishes the program.

Definition at line 153 of file Context.cpp.

References end_of_program_.

Referenced by bnssemulator::IntExecuter::execute(), and bnssemulator::Processor::executeProgram().

```
                                       {
        end_of_program_ = true;
    }
```

char bnssemulator::Context::getCharacter () `[noexcept]`

Gets the first character to be written to stdin.

Returns:

Character to be written to stdin

Definition at line 178 of file Context.cpp.

References characters_, and characters_mutex_.

Referenced by jumpToKeyboardInterrupt().

```
                                     {
        characters_mutex_.lock();
        auto ret = characters_.front();
        characters_.pop();
        characters_mutex_.unlock();
        return ret;
    }
```

InstructionBitField bnssemulator::Context::getInstruction ()

Gets the instruction to execute.

Returns:

Instruction

Definition at line 140 of file Context.cpp.

References address_space_, bnssemulator::AddressSpace::getInstruction(), and program_counter_.

Referenced by bnssemulator::Processor::executeInstruction().

```
                                             {
        auto ret = address_space_.getInstruction(program_counter_);
        program_counter_ += 4;
        return ret;
    }
```

int32_t bnssemulator::Context::getOperand (InstructionBitField *instruction*, size_t *register_index*, size_t *num_of_bytes* = 4)

Gets the operand based on the instruction.

Parameters:

instruction	Instruction
register_index	Index of the register to use in case of register address modes
num_of_bytes	Number of bytes of the operand

Returns:

Operand

Definition at line 94 of file Context.cpp.

References bnssemulator::InstructionBitField::address_mode, address_space_, bnssemulator::AddressSpace::get16bitData(), bnssemulator::AddressSpace::get32bitData(), bnssemulator::AddressSpace::get8bitData(), getOperandAddress(), bnssemulator::getRegisterIndex(), getSecondWordOfInstruction(), bnssemulator::IMMEDIATE, bnssemulator::MEMORY_DIRECT, bnssemulator::REGISTER_DIRECT, bnssemulator::REGISTER_INDIRECT, bnssemulator::REGISTER_INDIRECT_OFFSET, registers_, and bnssemulator::StringHelper::toHexString().

Referenced by bnssemulator::LoadExecuter::execute().

```
{
        int32_t val;
        switch (instruction.address_mode) {
        case IMMEDIATE:
            val = getSecondWordOfInstruction();
            return val;
        case REGISTER_DIRECT:
```

```
  101               return registers_[getRegisterIndex(instruction, register_index)];
  102           case MEMORY_DIRECT:
  103           case REGISTER_INDIRECT:
  104           case REGISTER_INDIRECT_OFFSET:
  105               val = getOperandAddress(instruction, register_index);
  106               switch (num_of_bytes) {
  107               case 1:
  108                   return address_space_.get8bitData(val);
  109               case 2:
  110                   return address_space_.get16bitData(val);
  111               case 4:
  112                   return address_space_.get32bitData(val);
  113               default:
  114                   throw MessageException("Invalid number of bytes");
  115               }
  116           default:
  117               throw MessageException("Invalid address mode: " +
StringHelper::toHexString(instruction.address_mode));
  118           }
  119       }
```

int32_t bnssemulator::Context::getOperandAddress (InstructionBitField *instruction*, size_t *register_index*)

Gets the address of the operand based on the instruction.

Parameters:

instruction	Instruction
register_index	Index of the register to use in case of register address modes

Returns:

Address of the operand

Definition at line 121 of file Context.cpp.

References bnssemulator::InstructionBitField::address_mode, bnssemulator::getRegisterIndex(), getSecondWordOfInstruction(), bnssemulator::MEMORY_DIRECT, bnssemulator::REGISTER_INDIRECT, bnssemulator::REGISTER_INDIRECT_OFFSET, registers_, and bnssemulator::StringHelper::toHexString().

Referenced by bnssemulator::CallExecuter::execute(), bnssemulator::StoreExecuter::execute(), bnssemulator::JmpExecuter::execute(), bnssemulator::ConditionalJumpExecuter::execute(), and getOperand().

```
  121
{
  122           // ReSharper disable once CppJoinDeclarationAndAssignment
  123           uint32_t second_word;
  124
  125           switch (instruction.address_mode) {
  126           case MEMORY_DIRECT:
  127               return getSecondWordOfInstruction();
  128           case REGISTER_INDIRECT:
  129               return registers_[getRegisterIndex(instruction, register_index)];
  130           case REGISTER_INDIRECT_OFFSET:
  131               // ReSharper disable once CppJoinDeclarationAndAssignment
  132               second_word = getSecondWordOfInstruction();
  133               return
static_cast<uint32_t>(registers_[getRegisterIndex(instruction, register_index)]) +
second_word;
  134           default:
  135               throw MessageException("Invalid address mode: " +
StringHelper::toHexString(instruction.address_mode));
  136           }
  137       }
```

Register & bnssemulator::Context::getRegister (size_t *index*)`[noexcept]`

Gets the reference to the register.

Parameters:

index	Index of the register

Returns:

Reference to the register

Definition at line 29 of file Context.cpp.

References registers_.

Referenced by bnssemulator::AluExecuter::execute(), bnssemulator::StoreExecuter::execute(), bnssemulator::PushExecuter::execute(), bnssemulator::PopExecuter::execute(), bnssemulator::ConditionalJumpExecuter::execute(), bnssemulator::LoadExecuter::execute(), bnssemulator::IntExecuter::execute(), bnssemulator::NotExecuter::execute(), and getRegister().

```
                                                    {
        return registers_[index];
    }
```

const Register & bnssemulator::Context::getRegister (size_t *index*) const[noexcept]

Gets the reference to the register.

Parameters:

index	Index of the register

Returns:

Reference to the register

Definition at line 33 of file Context.cpp.

References getRegister().

```
                                                              {
        return const_cast<Context &>(*this).getRegister(index);
    }
```

int32_t bnssemulator::Context::getSecondWordOfInstruction ()

Gets the second word of the instruction.

Returns:

Second word of the instruction

Definition at line 147 of file Context.cpp.

References address_space_, bnssemulator::AddressSpace::getSecondWordOfInstruction(), and program_counter_.

Referenced by getOperand(), and getOperandAddress().

```
                                                   {
        auto ret =
address_space_.getSecondWordOfInstruction(program_counter_);
        program_counter_ += 4;
        return ret;
    }
```

bool bnssemulator::Context::hasCharacters () const[noexcept]

Checks whether there are any characters that should be written to stdin.

Definition at line 171 of file Context.cpp.

References characters_, and characters_mutex_.

Referenced by bnssemulator::Processor::executeProgram().

```
171                                         {
172         characters_mutex_.lock();
173         auto ret = !characters_.empty();
174         characters_mutex_.unlock();
175         return ret;
176     }
```

bool bnssemulator::Context::insideInterrupt () const `[noexcept]`

Checks whether the program is executing an interrupt routine.

Returns:

Whether the program is executing an interrupt routine

Definition at line 161 of file Context.cpp.

References inside_interrupt_.

Referenced by bnssemulator::Processor::executeProgram().

```
161                                           {
162         return inside_interrupt_;
163     }
```

void bnssemulator::Context::jumpTo (uint32_t *address*)

Jumps to address.

Parameters:

address	Address

Definition at line 46 of file Context.cpp.

References program_counter_.

Referenced by bnssemulator::ConditionalJumpExecuter::execute(), bnssemulator::JmpExecuter::execute(), and jumpToSubroutine().

```
 46                                   {
 47         program_counter_ = address;
 48     }
```

void bnssemulator::Context::jumpToErrorInterrupt () `[noexcept]`

Jumps to error interrupt.

Definition at line 199 of file Context.cpp.

References address_space_, bnssemulator::AddressSpace::errorInterrupt(), and jumpToInterrupt().

Referenced by bnssemulator::Processor::executeProgram().

```
199                                          {
200         jumpToInterrupt(address_space_.errorInterrupt());
201     }
```

void bnssemulator::Context::jumpToInterrupt (size_t *entry*)

Jumps to interrupt routine at the specified entry.

Parameters:

entry	Entry

Definition at line 59 of file Context.cpp.

References address_space_, bnssemulator::AddressSpace::getInterrupt(), inside_interrupt_, and jumpToSubroutine().

Referenced by bnssemulator::IntExecuter::execute(), jumpToErrorInterrupt(), jumpToKeyboardInterrupt(), and jumpToTimerInterrupt().

```
                                          {
        inside_interrupt_ = true;
        jumpToSubroutine(address_space_.getInterrupt(entry));
    }
```

void bnssemulator::Context::jumpToKeyboardInterrupt () [noexcept]

Jumps to keyboard interrupt.

Definition at line 208 of file Context.cpp.

References address_space_, getCharacter(), jumpToInterrupt(), bnssemulator::AddressSpace::keyboardInterrupt(), and bnssemulator::AddressSpace::writeToStdin().

Referenced by bnssemulator::Processor::executeProgram().

```
                                               {
        address_space_.writeToStdin(getCharacter());
        jumpToInterrupt(address_space_.keyboardInterrupt());
    }
```

void bnssemulator::Context::jumpToSubroutine (uint32_t *address*)

Jumps to subroutine at address.

Parameters:

address	Address

Definition at line 50 of file Context.cpp.

References inside_interrupt_, interrupt_call_stack_depth_, jumpTo(), program_counter_, and pushToStack().

Referenced by bnssemulator::CallExecuter::execute(), and jumpToInterrupt().

```
                                               {
        if (inside_interrupt_) {
            interrupt_call_stack_depth_++;
        }

        pushToStack(program_counter_);
        jumpTo(address);
    }
```

void bnssemulator::Context::jumpToTimerInterrupt () [noexcept]

Jumps to timer interrupt.

Definition at line 203 of file Context.cpp.

References address_space_, jumpToInterrupt(), timer_triggered_, and bnssemulator::AddressSpace::timerInterrupt().

Referenced by bnssemulator::Processor::executeProgram().

```
                                            {
        timer_triggered_ = false;
        jumpToInterrupt(address_space_.timerInterrupt());
    }
```

int32_t bnssemulator::Context::popFromStack ()

Pops a value from the stack.

Returns:

Popped value

Exceptions:

Throws	if stack underflow happens

Definition at line 23 of file Context.cpp.

References address_space_, bnssemulator::AddressSpace::get32bitData(), and stack_pointer_.

Referenced by bnssemulator::PopExecuter::execute(), and returnFromSubroutine().

```
                                {
        int32_t ret = address_space_.get32bitData(stack_pointer_);
        stack_pointer_ -= 4;
        return ret;
    }
```

void bnssemulator::Context::pressCharacter (char *character*) [noexcept]

Presses a character.

Parameters:

character	Character

Definition at line 165 of file Context.cpp.

References characters_, and characters_mutex_.

Referenced by bnssemulator::KeyboardListener::listen().

```
                                                        {
        characters_mutex_.lock();
        characters_.push(character);
        characters_mutex_.unlock();
    }
```

bool bnssemulator::Context::programFinished () const [noexcept]

Checks whether the program finished.

Returns:

Whether the program finished

Definition at line 157 of file Context.cpp.

References end_of_program_.

Referenced by bnssemulator::Processor::executeProgram(), bnssemulator::TimerListener::listen(), and bnssemulator::KeyboardListener::listen().

```
                                                {
        return end_of_program_;
    }
```

void bnssemulator::Context::pushToStack (int32_t *value*)

Pushes a value to the stack.

Parameters:

value	Value to be pushed

Exceptions:

Throws	if stack overflow happens

Definition at line 17 of file Context.cpp.

References address_space_, bnssemulator::AddressSpace::set32bitData(), and stack_pointer_.

Referenced by bnssemulator::PushExecuter::execute(), and jumpToSubroutine().

```
                                    {
        stack_pointer_ += 4;
        address_space_.set32bitData(stack_pointer_, value);
    }
```

void bnssemulator::Context::returnFromSubroutine ()

Returns from subroutine.

Definition at line 64 of file Context.cpp.

References inside_interrupt_, interrupt_call_stack_depth_, popFromStack(), and program_counter_.

Referenced by bnssemulator::RetExecuter::execute().

```
                                {
        if (inside_interrupt_) {
            interrupt_call_stack_depth_--;
            if (interrupt_call_stack_depth_ == 0) {
                inside_interrupt_ = false;
            }
        }

        program_counter_ = popFromStack();
    }
```

bool bnssemulator::Context::timerTriggered () const `[noexcept]`

Checks if the timer was triggered.

Returns:

Whether the timer has been triggered

Definition at line 186 of file Context.cpp.

References timer_mutex_, and timer_triggered_.

Referenced by bnssemulator::Processor::executeProgram(), and bnssemulator::TimerListener::listen().

```
                                          {
        timer_mutex_.lock();
        auto ret = timer_triggered_;
        timer_mutex_.unlock();
        return ret;
    }
```

void bnssemulator::Context::timerTriggered (bool *value*) `[noexcept]`

Sets the timer triggered flag.

Parameters:

value	Flag

Definition at line 193 of file Context.cpp.

References timer_mutex_, timer_triggered_, and cxxopts::value().

```
                                               {
        timer_mutex_.lock();
        timer_triggered_ = value;
        timer_mutex_.unlock();
    }
```

Member Data Documentation

AddressSpace bnssemulator::Context::address_space_ `[private]`

Definition at line 177 of file Context.h.

Referenced by addressSpace(), Context(), getInstruction(), getOperand(), getSecondWordOfInstruction(), jumpToErrorInterrupt(), jumpToInterrupt(), jumpToKeyboardInterrupt(), jumpToTimerInterrupt(), popFromStack(), and pushToStack().

std::queue<char> bnssemulator::Context::characters_ `[private]`

Definition at line 186 of file Context.h.

Referenced by getCharacter(), hasCharacters(), and pressCharacter().

std::mutex bnssemulator::Context::characters_mutex_ `[mutable]`, `[private]`

Definition at line 185 of file Context.h.

Referenced by getCharacter(), hasCharacters(), and pressCharacter().

bool bnssemulator::Context::end_of_program_ = false `[private]`

Definition at line 191 of file Context.h.

Referenced by finishProgram(), and programFinished().

bool bnssemulator::Context::inside_interrupt_ = false `[private]`

Definition at line 182 of file Context.h.

Referenced by insideInterrupt(), jumpToInterrupt(), jumpToSubroutine(), and returnFromSubroutine().

size_t bnssemulator::Context::interrupt_call_stack_depth_ = 0 `[private]`

Definition at line 183 of file Context.h.

Referenced by jumpToSubroutine(), and returnFromSubroutine().

Register& bnssemulator::Context::program_counter_ `[private]`

Definition at line 180 of file Context.h.

Referenced by Context(), getInstruction(), getSecondWordOfInstruction(), jumpTo(), jumpToSubroutine(), and returnFromSubroutine().

std::vector<Register> bnssemulator::Context::registers_ `[private]`

Definition at line 178 of file Context.h.

Referenced by getOperand(), getOperandAddress(), and getRegister().

Register& bnssemulator::Context::stack_pointer_ `[private]`

Definition at line 179 of file Context.h.

Referenced by Context(), popFromStack(), and pushToStack().

std::mutex bnssemulator::Context::timer_mutex_ `[mutable]`, `[private]`

Definition at line 188 of file Context.h.

Referenced by timerTriggered().

bool bnssemulator::Context::timer_triggered_ = false `[private]`

Definition at line 189 of file Context.h.

Referenced by jumpToTimerInterrupt(), and timerTriggered().

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**Context.h**
- Code/Emulator/Source/**Context.cpp**

bnssassembler::Data Class Reference

Class representing the MicroRISC data.
`#include <Data.h>`

Public Member Functions

- **Data** (**DataType type**, **MicroRiscExpression value**, **MicroRiscExpression count**) noexcept
  Constructs a ***Data*** *object.*
- **Data** (**DataType type**, **MicroRiscExpression count**) noexcept
  Constructs an uninitialized ***Data*** *object.*
- **DataType type** () const noexcept
  Get the type of the data.
- bool **initialized** () const noexcept
  Check whether the data is initialized.
- **MicroRiscExpression value** () const noexcept
  Get the value of the data.
- **MicroRiscExpression count** () const noexcept
  Get how many times the data should repeat.

Private Attributes

- **DataType type_**
- bool **initialized_** = true
- **MicroRiscExpression value_**
- **MicroRiscExpression count_**

Detailed Description

Class representing the MicroRISC data.

Definition at line 11 of file Data.h.

Constructor & Destructor Documentation

bnssassembler::Data::Data (DataType *type*, MicroRiscExpression *value*, MicroRiscExpression *count*)`[noexcept]`

Constructs a **Data** object.

Parameters:

type	Type of the data
value	Value of the data
count	How many times the data will repeat

Definition at line 5 of file Data.cpp.

```
    5 : type_(type), value_(value), count_(count) {}
```

bnssassembler::Data::Data (DataType *type*, MicroRiscExpression *count*)`[explicit]`, `[noexcept]`

Constructs an uninitialized **Data** object.

Parameters:

type	Type of the data
count	How many times the data will repeat

Definition at line 7 of file Data.cpp.

```
    7 : type_(type), initialized_(false), value_(nullptr), count_(count) {}
```

Member Function Documentation

MicroRiscExpression bnssassembler::Data::count () const [noexcept]

Get how many times the data should repeat.

Returns:

How many times should the data repeat

Definition at line 21 of file Data.cpp.

References count_.

```
   21                                                     {
   22         return count_;
   23     }
```

bool bnssassembler::Data::initialized () const [noexcept]

Check whether the data is initialized.

Returns:

Whether the data is initialized

Definition at line 13 of file Data.cpp.

References initialized_.

```
   13                                            {
   14         return initialized_;
   15     }
```

DataType bnssassembler::Data::type () const [noexcept]

Get the type of the data.

Returns:

Type of the data

Definition at line 9 of file Data.cpp.

References type_.

```
    9                                        {
   10         return type_;
   11     }
```

MicroRiscExpression bnssassembler::Data::value () const [noexcept]

Get the value of the data.

Returns:

value of the data

Definition at line 17 of file Data.cpp.

References value_.

```
                                                         {
            return value_;
        }
```

Member Data Documentation

MicroRiscExpression bnssassembler::Data::count_ `[private]`

Definition at line 55 of file Data.h.

Referenced by count().

bool bnssassembler::Data::initialized_ = true `[private]`

Definition at line 53 of file Data.h.

Referenced by initialized().

DataType bnssassembler::Data::type_ `[private]`

Definition at line 52 of file Data.h.

Referenced by type().

MicroRiscExpression bnssassembler::Data::value_ `[private]`

Definition at line 54 of file Data.h.

Referenced by value().

The documentation for this class was generated from the following files:

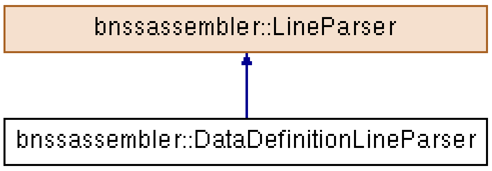
- Code/Assembler/Include/**Data.h**
- Code/Assembler/Source/**Data.cpp**

bnssassembler::DataDefinitionLineParser Class Reference

Class used for parsing data definitions.

`#include <DataDefinitionLineParser.h>`

Inheritance diagram for bnssassembler::DataDefinitionLineParser:


bnssassembler::LineParser
bnssassembler::DataDefinitionLineParser

Protected Member Functions

- std::shared_ptr< **Token** > **parse** (const std::string &line, size_t line_number, std::string initial_line) const override
  Parses one line of the file. Does not call the next parser in chain.

Additional Inherited Members

Detailed Description

Class used for parsing data definitions.

Definition at line 10 of file DataDefinitionLineParser.h.

Member Function Documentation

std::shared_ptr< Token > bnssassembler::DataDefinitionLineParser::parse (const std::string & *line*, size_t *line_number*, std::string *initial_line*) const `[override]`, `[protected]`, `[virtual]`

Parses one line of the file. Does not call the next parser in chain.

Parameters:

line	Line to parse
line_number	Number of the line that is parsed
initial_line	Initial line that is parsed

Returns:

Extracted token from line or nullptr if the parser failed parsing the line

Exceptions:

Throws	if the parser failed and identified the error

Implements **bnssassembler::LineParser** (*p.257*).

Definition at line 54 of file DataDefinitionLineParser.cpp.

References bnssassembler::COMMA_TOKENIZER_REGEX, bnssassembler::LAST_COMMA_TOKEN_REGEX, and bnssassembler::parseData().

```
{
        auto parsed_line = line;
        std::vector<Data> data_vector;

        static std::regex comma("(.*)','(.*)");

        while (true) {
            if (!regex_match(parsed_line, COMMA_TOKENIZER_REGEX) || regex_match(parsed_line, comma)) {
                break;
            }

            auto comma_token = regex_replace(parsed_line, COMMA_TOKENIZER_REGEX, "$1");
            parsed_line = regex_replace(parsed_line, COMMA_TOKENIZER_REGEX, "$2");

            try {
                data_vector.push_back(parseData(comma_token));
            }
            catch (InvalidDataDefinitionException &) {
                return nullptr;
            }
            catch (InvalidDataTypeException &) {
                return nullptr;
            }
        }

        if (!regex_match(parsed_line, LAST_COMMA_TOKEN_REGEX)) {
            return nullptr;
        }

        auto comma_token = regex_replace(parsed_line, LAST_COMMA_TOKEN_REGEX, "$1");
        try {
            data_vector.push_back(parseData(comma_token));
        }
        catch (InvalidDataDefinitionException &) {
            return nullptr;
        }
        catch (InvalidDataTypeException &) {
            return nullptr;
        }

        return std::make_shared<DataDefinitionToken>(data_vector, line_number, initial_line);
    }
```

The documentation for this class was generated from the following files:

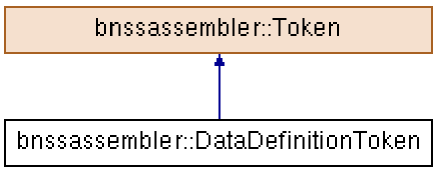
- Code/Assembler/Include/**DataDefinitionLineParser.h**
- Code/Assembler/Source/**DataDefinitionLineParser.cpp**

bnssassembler::DataDefinitionToken Class Reference

Class representing the data definition token.

`#include <DataDefinitionToken.h>`

Inheritance diagram for bnssassembler::DataDefinitionToken:



Public Member Functions

- **DataDefinitionToken** (std::vector< **Data** > data, size_t line_number, std::string **line**) noexcept
  *Constructs a **DataDefinitionToken** object.*
- void **resolveSymbolDefinitions** (std::unordered_set< **SymbolDefinition** > symbols) noexcept override
  Resolves symbol definitions in a token.
- void **firstPass** (**FirstPassData** &data) const override
  Executes the first pass over the token.
- void **secondPass** (**SecondPassData** &data) const override
  Executes the second pass over the token.
- void **resolveSymbolTable** (const **SymbolTable** &symbol_table) noexcept override
  Resolves the symbols from the symbol table and updates relocation info.
- void **resolveImports** (std::unordered_set< std::string > imported_symbols) noexcept override
  Resolves the imported symbols and updates relocation info.

Private Member Functions

- size_t **dataSize** () const noexcept
  Gets the size of the data defined with this token.

Private Attributes

- std::vector< **Data** > **data_**

Detailed Description

Class representing the data definition token.

Definition at line 12 of file DataDefinitionToken.h.

Constructor & Destructor Documentation

bnssassembler::DataDefinitionToken::DataDefinitionToken (std::vector< Data > *data*, size_t *line_number*, std::string *line*) `[noexcept]`

Constructs a **DataDefinitionToken** object.

Parameters:

data	**Data** that is defined

line_number	Number of the line where data is defined
line	Line where data is defined

Definition at line 8 of file DataDefinitionToken.cpp.

```
: Token(line_number, line), data_(data) {}
```

Member Function Documentation

size_t bnssassembler::DataDefinitionToken::dataSize () const `[private]`, `[noexcept]`

Gets the size of the data defined with this token.

Returns:

Size of the data defined

Definition at line 77 of file DataDefinitionToken.cpp.

References data_, and bnssassembler::DataTypeParser::size().

Referenced by firstPass().

```
                                                {
        size_t ret = 0;

        for (auto &data : data_) {
            ret += data.count().value() * DataTypeParser::size(data.type());
        }

        return ret;
    }
```

void bnssassembler::DataDefinitionToken::firstPass (FirstPassData & *data*) const `[override]`, `[virtual]`

Executes the first pass over the token.

Parameters:

data	**Data** that the token will modify

Implements **bnssassembler::Token** (*p.510*).

Definition at line 22 of file DataDefinitionToken.cpp.

References dataSize(), and bnssassembler::FirstPassData::incLocationCounter().

```
                                                            {
        data.incLocationCounter(dataSize());
    }
```

void bnssassembler::DataDefinitionToken::resolveImports (std::unordered_set< std::string > *imported_symbols*) `[override]`, `[virtual]`, `[noexcept]`

Resolves the imported symbols and updates relocation info.

Parameters:

imported_symbols	Collection of imported symbols

Reimplemented from **bnssassembler::Token** (*p.511*).

Definition at line 67 of file DataDefinitionToken.cpp.

References data_.

```
   67
{
   68         for (auto &data: data_) {
   69             if (data.initialized()) {
   70                 data.value().resolveImports(imported_symbols);
   71             }
   72 
   73             data.count().resolveImports(imported_symbols);
   74         }
   75     }
```

void bnssassembler::DataDefinitionToken::resolveSymbolDefinitions (std::unordered_set< SymbolDefinition > *symbols*)`[override]`, `[virtual]`, `[noexcept]`

Resolves symbol definitions in a token.

Parameters:

symbols	Vector od symbol definitions that should be resolved

Reimplemented from **bnssassembler::Token** (*p.511*).

Definition at line 10 of file DataDefinitionToken.cpp.

```
   10
{
   11         for (auto &symbol : symbols) {
   12             for (auto &data : data_) {
   13                 if (data.initialized()) {
   14                     data.value().setValue(symbol.name(), 
symbol.expression());
   15                 }
   16 
   17                 data.count().setValue(symbol.name(), symbol.expression());
   18             }
   19         }
   20     }
```

void bnssassembler::DataDefinitionToken::resolveSymbolTable (const SymbolTable & *symbol_table*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the symbols from the symbol table and updates relocation info.

Parameters:

symbol_table	**Symbol** table

Reimplemented from **bnssassembler::Token** (*p.511*).

Definition at line 57 of file DataDefinitionToken.cpp.

References data_.

```
   57
{
   58         for (auto &data: data_) {
   59             if (data.initialized()) {
   60                 data.value().resolveSymbolTable(symbol_table);
   61             }
   62 
   63             data.count().resolveSymbolTable(symbol_table);
   64         }
   65     }
```

void bnssassembler::DataDefinitionToken::secondPass (SecondPassData & *data*) const`[override]`, `[virtual]`

Executes the second pass over the token.

Parameters:

data	**Data** that the token will modify

Implements **bnssassembler::Token** (*p.512*).

Definition at line 26 of file DataDefinitionToken.cpp.

References bnssassembler::SecondPassData::addData(), bnssassembler::BSS, bnssassembler::BYTE, bnssassembler::SecondPassData::currentSectionType(), bnssassembler::DATA, data_, bnssassembler::DOUBLE_WORD, bnssassembler::RODATA, and bnssassembler::WORD.

```
   26                                                                    {
   27         for (auto &data_definition : data_) {
   28             if (data_definition.initialized()) {
   29                 if (data.currentSectionType() != DATA &&
data.currentSectionType() != RODATA) {
   30                     throw MessageException("Initialized values can only exist
in DATA and RODATA sections");
   31                 }
   32
   33                 for (auto i = 0; i < data_definition.count().value(); i++) {
   34                     switch (data_definition.type()) {
   35                     case DOUBLE_WORD:
   36
data.addData(static_cast<uint32_t>(data_definition.value().value()),
data_definition.value().generateRelocations());
   37                         break;
   38                     case WORD:
   39
data.addData(static_cast<uint16_t>(data_definition.value().value()),
data_definition.value().generateRelocations());
   40                         break;
   41                     case BYTE:
   42
data.addData(static_cast<uint8_t>(data_definition.value().value()),
data_definition.value().generateRelocations());
   43                         break;
   44                     default:
   45                         throw MessageException("Data type not implemented");
   46                     }
   47                 }
   48             }
   49             else {
   50                 if (data.currentSectionType() != BSS) {
   51                     throw MessageException("Uninitialized values can only exist
in the BSS section");
   52                 }
   53             }
   54         }
   55     }
```

Member Data Documentation

std::vector<Data> bnssassembler::DataDefinitionToken::data_ `[private]`

Definition at line 28 of file DataDefinitionToken.h.

Referenced by dataSize(), resolveImports(), resolveSymbolTable(), and secondPass().

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**DataDefinitionToken.h**
- Code/Assembler/Source/**DataDefinitionToken.cpp**

bnssassembler::DataTypeParser Class Reference

Utility class used for parsing data types.
`#include <DataTypeParser.h>`

Classes

- struct **DataTypeParserStaticData**

Static Public Member Functions

- static **DataType parse** (std::string str)
  Parses the datatype from string.
- static size_t **size** (**DataType** data)
  Returns the size of the data type in bytes.

Private Member Functions

- **DataTypeParser** ()=delete
- **DataTypeParser** (**DataTypeParser** &)=delete
- void **operator=** (**DataTypeParser** &)=delete

Static Private Member Functions

- static **DataTypeParserStaticData** & **staticData** () noexcept

---

Detailed Description

Utility class used for parsing data types.

Definition at line 11 of file DataTypeParser.h.

Constructor & Destructor Documentation

bnssassembler::DataTypeParser::DataTypeParser () `[private]`, `[delete]`

bnssassembler::DataTypeParser::DataTypeParser (DataTypeParser &) `[private]`, `[delete]`

Member Function Documentation

void bnssassembler::DataTypeParser::operator= (DataTypeParser &) `[private]`, `[delete]`

DataType bnssassembler::DataTypeParser::parse (std::string *str*) `[static]`

Parses the datatype from string.

Parameters:

str	String containing the data type

Returns:

Parsed data type

Definition at line 8 of file DataTypeParser.cpp.

References bnssassembler::DataTypeParser::DataTypeParserStaticData::map, and staticData().

Referenced by bnssassembler::parseData().

```
                                       {
        transform(str.begin(), str.end(), str.begin(), tolower);

        if (staticData().map.count(str) == 0) {
            throw InvalidDataTypeException(str);
        }

        return staticData().map[str];
    }
```

size_t bnssassembler::DataTypeParser::size (DataType *data*)`[static]`

Returns the size of the data type in bytes.

Parameters:

data	DataType

Definition at line 18 of file DataTypeParser.cpp.

References bnssassembler::BYTE, bnssassembler::DOUBLE_WORD, and bnssassembler::WORD.

Referenced by bnssassembler::DataDefinitionToken::dataSize().

```
                                    {
        switch (data) {
        case BYTE: return 1;
        case WORD: return 2;
        case DOUBLE_WORD: return 4;
        default: throw MessageException("DataType not yet implemented");
        }
    }
```

DataTypeParser::DataTypeParserStaticData & bnssassembler::DataTypeParser::staticData ()`[static]`, `[private]`, `[noexcept]`

Definition at line 33 of file DataTypeParser.cpp.

Referenced by parse().

```
{
        static DataTypeParserStaticData static_data;
        return static_data;
    }
```

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**DataTypeParser.h**
- Code/Assembler/Source/**DataTypeParser.cpp**

bnssassembler::DataTypeParser::DataTypeParserStaticData Struct Reference

Public Member Functions

- **DataTypeParserStaticData** ()

Public Attributes

- std::unordered_map< std::string, **DataType** > **map**

Detailed Description

Definition at line 26 of file DataTypeParser.h.

Constructor & Destructor Documentation

bnssassembler::DataTypeParser::DataTypeParserStaticData::DataTypeParserStaticData ()

Definition at line 27 of file DataTypeParser.cpp.

References bnssassembler::BYTE, bnssassembler::DOUBLE_WORD, and bnssassembler::WORD.

```
                                                                  {
        map["db"] = BYTE;
        map["dw"] = WORD;
        map["dd"] = DOUBLE_WORD;
    }
```

Member Data Documentation

std::unordered_map<std::string, DataType> bnssassembler::DataTypeParser::DataTypeParserStaticData::map

Definition at line 27 of file DataTypeParser.h.

Referenced by bnssassembler::DataTypeParser::parse().

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

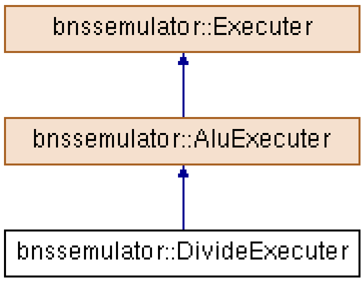
- Code/Assembler/Include/**DataTypeParser.h**
- Code/Assembler/Source/**DataTypeParser.cpp**

bnssemulator::DivideExecuter Class Reference

Class representing the executer of the divide instruction.

`#include <DivideExecuter.h>`

Inheritance diagram for bnssemulator::DivideExecuter:



Protected Member Functions

- void **execute** (**Register** &dst, const **Register** &lhs, const **Register** &rhs) const override
  Executes the ALU instruction.

Additional Inherited Members

Detailed Description

Class representing the executer of the divide instruction.

Definition at line 10 of file DivideExecuter.h.

Member Function Documentation

void bnssemulator::DivideExecuter::execute (Register & *dst*, const Register & *lhs*, const Register & *rhs*) const `[override]`, `[protected]`, `[virtual]`

Executes the ALU instruction.

Parameters:

dst	Reference to the destination register
lhs	Left operand register
rhs	Right operand register

Implements **bnssemulator::AluExecuter** (*p.102*).

Definition at line 6 of file DivideExecuter.cpp.

```
{
        if (rhs == 0) {
            throw MessageException("Division by zero");
        }

        dst = lhs / rhs;
    }
```

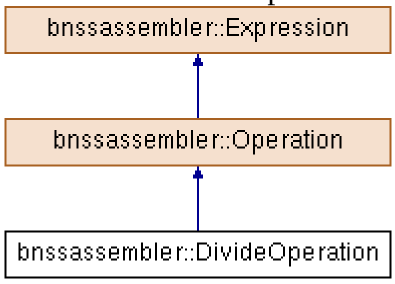
The documentation for this class was generated from the following files:

- Code/Emulator/Include/**DivideExecuter.h**

- Code/Emulator/Source/**DivideExecuter.cpp**

bnssassembler::DivideOperation Class Reference

Class implementing the behaviour of the / operator in expressions.
`#include <DivideOperation.h>`
Inheritance diagram for bnssassembler::DivideOperation:



Public Member Functions

- bool **validate** () const noexcept override
  Validates the expression.

Protected Member Functions

- int32_t **calculate** (int32_t lhs, int32_t rhs) const override
  Calculates the value of the subexpression.

Detailed Description

Class implementing the behaviour of the / operator in expressions.

Definition at line 10 of file DivideOperation.h.

Member Function Documentation

int32_t bnssassembler::DivideOperation::calculate (int32_t *lhs*, int32_t *rhs*) const`[override]`, `[protected]`, `[virtual]`

Calculates the value of the subexpression.

Parameters:

lhs	Left side of the operator
rhs	Right side of the operator

Returns:

Result of the operation

Exceptions:

Throws	if the expression can not be evaluated (example: division by zero)

Implements **bnssassembler::Operation** (*p.309*).

Definition at line 10 of file DivideOperation.cpp.

```
                                                                       {
        if (rhs == 0) {
            throw DivisionByZeroException();
        }

        return lhs / rhs;
```

```
    }
```

bool bnssassembler::DivideOperation::validate () const`[override]`, `[virtual]`, `[noexcept]`

Validates the expression.

Returns:

Boolean value indicating whether the expression is correct

Reimplemented from **bnssassembler::Expression** (*p.167*).

Definition at line 6 of file DivideOperation.cpp.

References bnssassembler::Operation::containsSymbol().

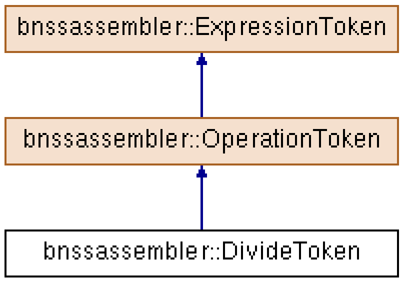
```
                                         {
        return !containsSymbol();
    }
```

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**DivideOperation.h**
- Code/Assembler/Source/**DivideOperation.cpp**

bnssassembler::DivideToken Class Reference

Token class representing the / operation.
`#include <DivideToken.h>`
Inheritance diagram for bnssassembler::DivideToken:


bnssassembler::ExpressionToken
bnssassembler::OperationToken
bnssassembler::DivideToken

Public Member Functions

- int **inputPriority** () const noexcept override
  Gets the input priority of the token.
- int **stackPriority** () const noexcept override
  Gets the stack priority of the token.
- int **rank** () const noexcept override
  Gets the rank of the token.
- std::string **operation** () const noexcept override
- std::shared_ptr< **Expression** > **create** () const override
  Creates an expression object out of the token.

Protected Member Functions

- std::shared_ptr< **ExpressionToken** > **clone** (std::string param) const override
  Clones the current object, using the string provided.

Detailed Description

Token class representing the / operation.

Definition at line 10 of file DivideToken.h.

Member Function Documentation

std::shared_ptr< ExpressionToken > bnssassembler::DivideToken::clone (std::string *param*) const `[override]`, `[protected]`, `[virtual]`

Clones the current object, using the string provided.

Parameters:

param	String that will be used to construct the new object

Returns:

Pointer to the cloned object

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 26 of file DivideToken.cpp.

```
                                                                    {
        return std::make_shared<DivideToken>();
    }
```

std::shared_ptr< Expression > bnssassembler::DivideToken::create () const`[override]`, `[virtual]`

Creates an expression object out of the token.

Returns:

Pointer to the expression

Exceptions:

Throws	if the token has no corresponding expression object

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 22 of file DivideToken.cpp.

```
                                                   {
        return std::make_shared<DivideOperation>();
    }
```

int bnssassembler::DivideToken::inputPriority () const`[override]`, `[virtual]`, `[noexcept]`

Gets the input priority of the token.

Returns:

Input priority of the token

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 6 of file DivideToken.cpp.

```
                                                  {
        return 3;
    }
```

std::string bnssassembler::DivideToken::operation () const`[override]`, `[virtual]`, `[noexcept]`

Implements **bnssassembler::OperationToken** (*p.314*).

Definition at line 18 of file DivideToken.cpp.

```
                                                    {
        return "/";
    }
```

int bnssassembler::DivideToken::rank () const`[override]`, `[virtual]`, `[noexcept]`

Gets the rank of the token.

Returns:

Rank of the token

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 14 of file DivideToken.cpp.

```
                                         {
        return 3;
    }
```

int bnssassembler::DivideToken::stackPriority () const `[override]`, `[virtual]`, `[noexcept]`

Gets the stack priority of the token.

Returns:

Stack priority of the token

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 10 of file DivideToken.cpp.

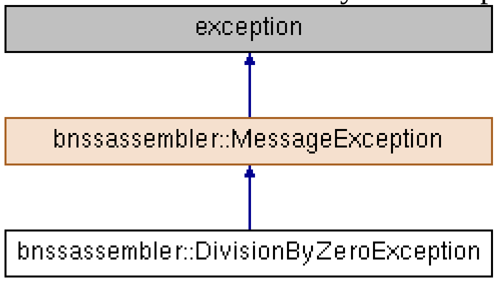
```
                                              {
        return 3;
    }
```

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**DivideToken.h**
- Code/Assembler/Source/**DivideToken.cpp**

bnssassembler::DivisionByZeroException Class Reference

Exception class representing division by zero.
`#include <DivisionByZeroException.h>`
Inheritance diagram for bnssassembler::DivisionByZeroException:


exception
bnssassembler::MessageException
bnssassembler::DivisionByZeroException

Public Member Functions

- **DivisionByZeroException** () noexcept
  *Constructs a **DivisionByZeroException** object.*

Detailed Description

Exception class representing division by zero.

Definition at line 10 of file DivisionByZeroException.h.

Constructor & Destructor Documentation

bnssassembler::DivisionByZeroException::DivisionByZeroException () `[noexcept]`

Constructs a **DivisionByZeroException** object.

Definition at line 5 of file DivisionByZeroException.cpp.

```
    5 : MessageException("Error: Division by zero") {}
```

The documentation for this class was generated from the following files:

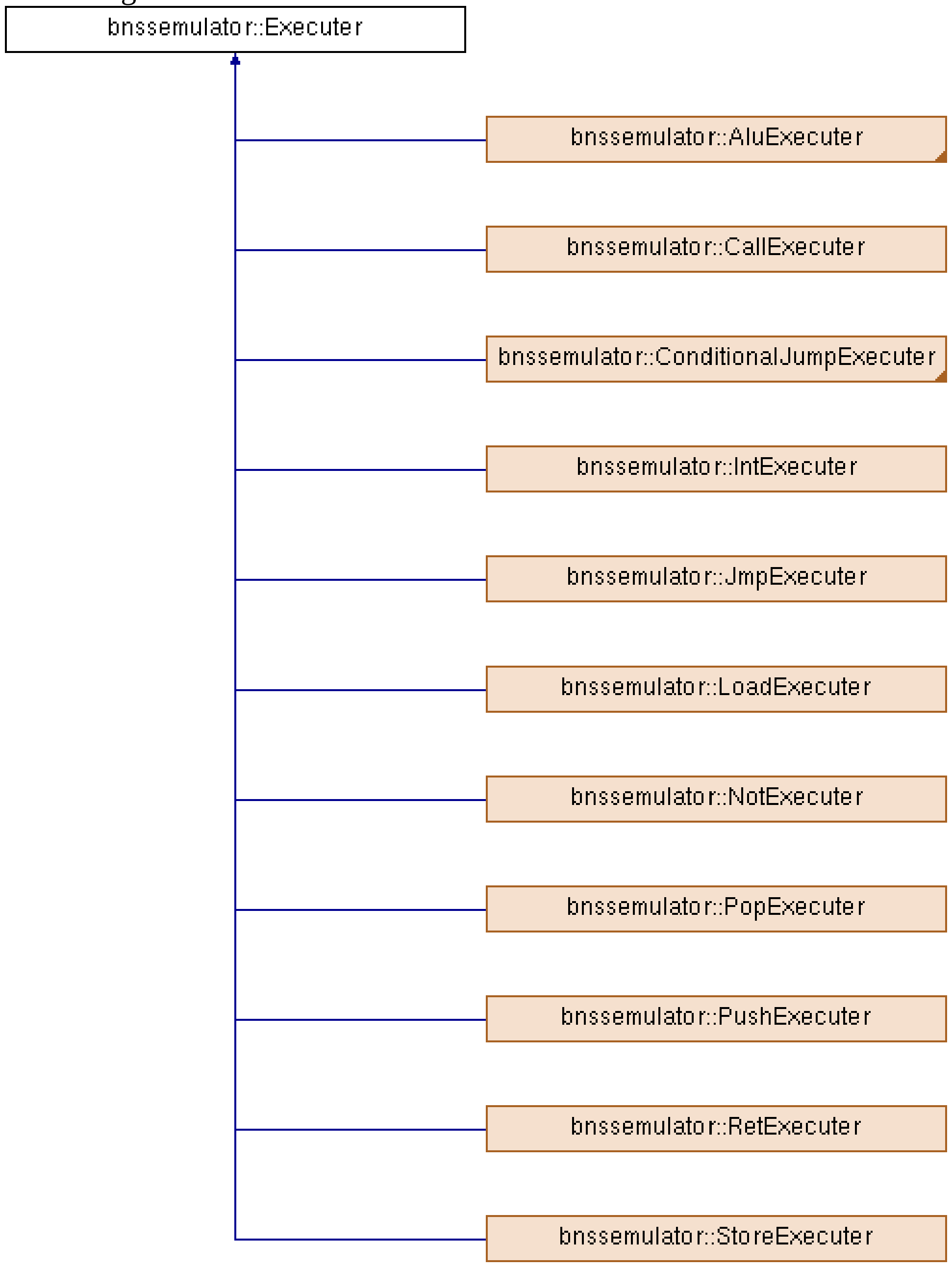
- Code/Assembler/Include/**DivisionByZeroException.h**
- Code/Assembler/Source/**DivisionByZeroException.cpp**

bnssemulator::Executer Class Reference

Base class used for executing instructions.

`#include <Executer.h>`

Inheritance diagram for bnssemulator::Executer:


bnssemulator::Executer
bnssemulator::AluExecuter
bnssemulator::CallExecuter
bnssemulator::ConditionalJumpExecuter
bnssemulator::IntExecuter
bnssemulator::JmpExecuter
bnssemulator::LoadExecuter
bnssemulator::NotExecuter
bnssemulator::PopExecuter
bnssemulator::PushExecuter
bnssemulator::RetExecuter
bnssemulator::StoreExecuter

Public Member Functions

- virtual void **execute** (**InstructionBitField** instruction, **Context** &context) const =0
  Executes the instruction.
- virtual **~Executer** ()=default

Detailed Description

Base class used for executing instructions.

Definition at line 11 of file Executer.h.

Constructor & Destructor Documentation

virtual bnssemulator::Executer::~Executer () `[virtual]`, `[default]`

Member Function Documentation

virtual void bnssemulator::Executer::execute (InstructionBitField *instruction*, Context & *context*) const `[pure virtual]`

Executes the instruction.

Parameters:

instruction	Instruction
context	**Processor** context

Implemented in **bnssemulator::AluExecuter** (*p.102*), **bnssemulator::CallExecuter** (*p.116*), **bnssemulator::ConditionalJumpExecuter** (*p.127*), **bnssemulator::IntExecuter** (*p.234*), **bnssemulator::JmpExecuter** (*p.248*), **bnssemulator::LoadExecuter** (*p.265*), **bnssemulator::NotExecuter** (*p.297*), **bnssemulator::PopExecuter** (*p.358*), **bnssemulator::PushExecuter** (*p.364*), **bnssemulator::RetExecuter** (*p.402*), and **bnssemulator::StoreExecuter** (*p.457*).

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

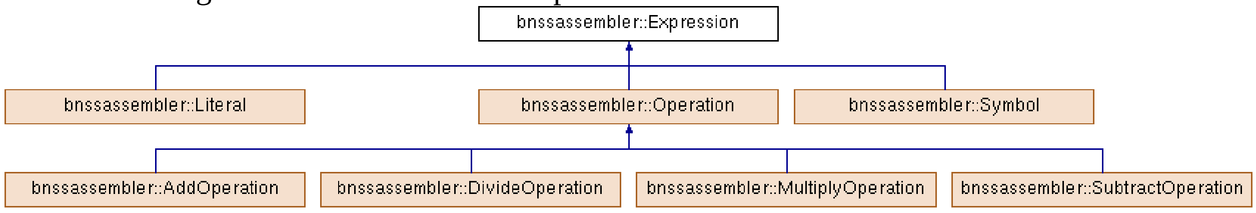
- Code/Emulator/Include/**Executer.h**

bnssassembler::Expression Class Reference

Class representing the math expression.

`#include <Expression.h>`

Inheritance diagram for bnssassembler::Expression:


bnssassembler::Expression
bnssassembler::Literal
bnssassembler::Operation
bnssassembler::Symbol
bnssassembler::AddOperation
bnssassembler::DivideOperation
bnssassembler::MultiplyOperation
bnssassembler::SubtractOperation

Public Member Functions

- virtual int32_t **value** () const =0
  Evaluates the expression.
- virtual bool **setValue** (std::string symbol, std::shared_ptr< **Expression** > **value**) noexcept
  Traverses the subtree and sets the value for the symbol.
- virtual bool **validate** () const noexcept
  Validates the expression.
- virtual bool **containsSymbol** () const noexcept
  *Tests whether the expression contains a **Symbol**.*
- virtual int **symbolCount** () const noexcept
  Counts the symbols in the expression.
- virtual void **pushChildren** (std::stack< std::reference_wrapper< std::shared_ptr< **Expression** >>> &stack) const noexcept
  Pushes the children to the stack.
- virtual void **resolveSymbolTable** (const **SymbolTable** &symbol_table) noexcept
  Resolves the symbols from the symbol table and sets the relocation info.
- virtual void **resolveImports** (std::unordered_set< std::string > imported_symbols) noexcept
  Resolves the imported symbols and sets the relocation info.
- virtual void **resolveCurrentPcSymbol** (size_t section_index, size_t offset) noexcept
  Resolves the current PC symbol and sets the relocation info.
- virtual std::list< **RelocationRecord** > **generateRelocations** () const
  Generates the relocation records for the subtree.
- virtual **~Expression** ()=default

Detailed Description

Class representing the math expression.

Definition at line 16 of file Expression.h.

Constructor & Destructor Documentation

virtual bnssassembler::Expression::~Expression () `[virtual]`, `[default]`

Member Function Documentation

bool bnssassembler::Expression::containsSymbol () const `[virtual]`, `[noexcept]`

Tests whether the expression contains a **Symbol**.

Returns:

Boolean value indicating whether the expression contains a **Symbol**

Reimplemented in **bnssassembler::Operation** (*p.309*), **bnssassembler::Symbol** (*p.481*), and **bnssassembler::SubtractOperation** (*p.474*).

Definition at line 14 of file Expression.cpp.

```
                                            {
        // Default: Does not contain symbol
        return false;
    }
```

std::list< RelocationRecord > bnssassembler::Expression::generateRelocations () const `[virtual]`

Generates the relocation records for the subtree.

Returns:

Collection of relocation records

Reimplemented in **bnssassembler::Operation** (*p.309*), **bnssassembler::Symbol** (*p.481*), **bnssassembler::SubtractOperation** (*p.474*), and **bnssassembler::AddOperation** (*p.88*).

Definition at line 40 of file Expression.cpp.

```
                                                                   {
        // Default: Return empty list
        return std::list<RelocationRecord>();
    }
```

void bnssassembler::Expression::pushChildren (std::stack< std::reference_wrapper< std::shared_ptr< Expression >>> & *stack*) const `[virtual]`, `[noexcept]`

Pushes the children to the stack.

Parameters:

stack	Reference to the stack

Reimplemented in **bnssassembler::Operation** (*p.310*).

Definition at line 24 of file Expression.cpp.

```
{
        // Default: Do nothing
    }
```

void bnssassembler::Expression::resolveCurrentPcSymbol (size_t *section_index*, size_t *offset*) `[virtual]`, `[noexcept]`

Resolves the current PC symbol and sets the relocation info.

Parameters:

section_index	Current PC section
offset	PC address in relation to the current section beginning

Reimplemented in **bnssassembler::Operation** (*p.311*), and **bnssassembler::Symbol** (*p.482*).

Definition at line 36 of file Expression.cpp.

```
   36
{
   37         // Default: Do nothing
   38     }
```

void bnssassembler::Expression::resolveImports (std::unordered_set< std::string > *imported_symbols*)`[virtual]`, `[noexcept]`

Resolves the imported symbols and sets the relocation info.

Parameters:

imported_symbols	Collection of imported symbols

Reimplemented in **bnssassembler::Operation** (*p.311*), and **bnssassembler::Symbol** (*p.482*).

Definition at line 32 of file Expression.cpp.

```
   32
{
   33         // Default: Do nothing
   34     }
```

void bnssassembler::Expression::resolveSymbolTable (const SymbolTable & *symbol_table*)`[virtual]`, `[noexcept]`

Resolves the symbols from the symbol table and sets the relocation info.

Parameters:

symbol_table	**Symbol** table

Reimplemented in **bnssassembler::Operation** (*p.311*), and **bnssassembler::Symbol** (*p.482*).

Definition at line 28 of file Expression.cpp.

```
   28
{
   29         // Default: Do nothing
   30     }
```

bool bnssassembler::Expression::setValue (std::string *symbol*, std::shared_ptr< Expression > *value*)`[virtual]`, `[noexcept]`

Traverses the subtree and sets the value for the symbol.

Parameters:

symbol	Name of the symbol
value	New value of the symbol

Returns:

Whether the symbol was found and the value was set

Reimplemented in **bnssassembler::Operation** (*p.312*), and **bnssassembler::Symbol** (*p.483*).

Definition at line 4 of file Expression.cpp.

```
    4
{
    5         // Default: No value set
```

```
        return false;
    }
```

int bnssassembler::Expression::symbolCount () const`[virtual]`, `[noexcept]`

Counts the symbols in the expression.

Returns:

Number of symbols in the expression

Reimplemented in **bnssassembler::Operation** (*p.312*), **bnssassembler::Symbol** (*p.483*), and **bnssassembler::SubtractOperation** (*p.475*).

Definition at line 19 of file Expression.cpp.

```
                                        {
        // Default: Does not contain any symbol
        return 0;
    }
```

bool bnssassembler::Expression::validate () const`[virtual]`, `[noexcept]`

Validates the expression.

Returns:

Boolean value indicating whether the expression is correct

Reimplemented in **bnssassembler::DivideOperation** (*p.157*), and **bnssassembler::MultiplyOperation** (*p.291*).

Definition at line 9 of file Expression.cpp.

```
                                      {
        // Default: Expression valid
        return true;
    }
```

virtual int32_t bnssassembler::Expression::value () const`[pure virtual]`

Evaluates the expression.

Exceptions:

Throws	if the expression has variables or could not be evaluated (for example, division by zero)

Implemented in **bnssassembler::Literal** (*p.259*), **bnssassembler::Operation** (*p.313*), and **bnssassembler::Symbol** (*p.484*).

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**Expression.h**
- Code/Assembler/Source/**Expression.cpp**

bnssassembler::ExpressionBuilder Class Reference

Utility class used for building math expressions.
`#include <ExpressionBuilder.h>`

Static Public Member Functions

- static **MicroRiscExpression build** (std::string infix_expression)
  *Builds one **MicroRiscExpression** object from the infix string.*
- static void **popToPostfix** (std::list< std::shared_ptr< **ExpressionToken** >> &output, std::stack< std::shared_ptr< **ExpressionToken** >> &stack, int &expression_rank)
  Pops one item from the stack to the postfix output, and updates the rank.

Private Member Functions

- **ExpressionBuilder** ()=delete
- **ExpressionBuilder** (**ExpressionBuilder** &)=delete
- void **operator=** (**ExpressionBuilder** &)=delete

Detailed Description

Utility class used for building math expressions.

Definition at line 12 of file ExpressionBuilder.h.

Constructor & Destructor Documentation

bnssassembler::ExpressionBuilder::ExpressionBuilder ()`[private]`**,** `[delete]`

bnssassembler::ExpressionBuilder::ExpressionBuilder (ExpressionBuilder &)`[private]`**,** `[delete]`

Member Function Documentation

MicroRiscExpression bnssassembler::ExpressionBuilder::build (std::string *infix_expression*)`[static]`

Builds one **MicroRiscExpression** object from the infix string.

Parameters:

infix_expression	Infix string

Definition at line 93 of file ExpressionBuilder.cpp.

References bnssassembler::infixToPostfix(), and bnssassembler::postfixToTree().

Referenced by bnssassembler::ImmediateParser::parse(), bnssassembler::SymbolDefinitionLineParser::parse(), bnssassembler::RegisterIndirectOffsetParser::parse(), bnssassembler::OrgDirectiveLineParser::parse(), bnssassembler::MemoryDirectParser::parse(), bnssassembler::parseData(), and bnssassembler::parsePcrel().

```
                                                                     {
        auto postfix = infixToPostfix(infix_expression);
        auto tree = postfixToTree(postfix);
        return MicroRiscExpression(tree);
    }
```

void bnssassembler::ExpressionBuilder::operator= (ExpressionBuilder &)[private], [delete]

void bnssassembler::ExpressionBuilder::popToPostfix (std::list< std::shared_ptr< ExpressionToken >> & *output*, std::stack< std::shared_ptr< ExpressionToken >> & *stack*, int & *expression_rank*)[static]

Pops one item from the stack to the postfix output, and updates the rank.

Parameters:

output	Postfix output
stack	Postfix stack
expression_rank	Postfix expression rank

Definition at line 99 of file ExpressionBuilder.cpp.

Referenced by bnssassembler::infixToPostfix(), and bnssassembler::OperationToken::process().

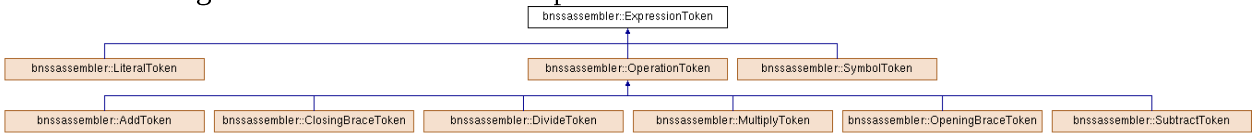
```
{
        if (stack.empty()) {
            throw MessageException("The opening brace is missing");
        }

        auto top = stack.top();
        stack.pop();
        output.push_back(top);

        expression_rank += top->rank();
        if (expression_rank < 1) {
            throw MessageException("The expression is invalid - too many 
operators");
        }
    }
```

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**ExpressionBuilder.h**
- Code/Assembler/Source/**ExpressionBuilder.cpp**

bnssassembler::ExpressionToken Class Reference

Class representing the token found in infix and postfix expressions.
`#include <ExpressionToken.h>`
Inheritance diagram for bnssassembler::ExpressionToken:


bnssassembler::ExpressionToken
bnssassembler::LiteralToken
bnssassembler::OperationToken
bnssassembler::SymbolToken
bnssassembler::AddToken
bnssassembler::ClosingBraceToken
bnssassembler::DivideToken
bnssassembler::MultiplyToken
bnssassembler::OpeningBraceToken
bnssassembler::SubtractToken

Public Member Functions

- virtual int **inputPriority** () const noexcept=0
  Gets the input priority of the token.
- virtual int **stackPriority** () const noexcept=0
  Gets the stack priority of the token.
- virtual int **rank** () const noexcept=0
  Gets the rank of the token.
- virtual void **process** (std::list< std::shared_ptr< **ExpressionToken** >> &output, std::stack< std::shared_ptr< **ExpressionToken** >> &stack, int &expression_rank) const =0
  Processes the current token.
- virtual std::shared_ptr< **Expression** > **create** () const =0
  Creates an expression object out of the token.
- virtual **~ExpressionToken** ()=default

Protected Member Functions

- virtual std::shared_ptr< **ExpressionToken** > **clone** (std::string param) const =0
  Clones the current object, using the string provided.
- **ExpressionToken** ()=default

Friends

- class **ExpressionTokenFactory**

Detailed Description

Class representing the token found in infix and postfix expressions.

Definition at line 13 of file ExpressionToken.h.

Constructor & Destructor Documentation

virtual bnssassembler::ExpressionToken::~ExpressionToken () `[virtual]`, `[default]`

bnssassembler::ExpressionToken::ExpressionToken () `[protected]`, `[default]`

Member Function Documentation

virtual std::shared_ptr<ExpressionToken> bnssassembler::ExpressionToken::clone (std::string *param*) const`[protected], [pure virtual]`

Clones the current object, using the string provided.

Parameters:

param	String that will be used to construct the new object

Returns:

Pointer to the cloned object

Implemented in **bnssassembler::LiteralToken** (*p.262*), **bnssassembler::SymbolToken** (*p.505*), **bnssassembler::AddToken** (*p.98*), **bnssassembler::ClosingBraceToken** (*p.117*), **bnssassembler::DivideToken** (*p.158*), **bnssassembler::MultiplyToken** (*p.292*), **bnssassembler::OpeningBraceToken** (*p.299*), and **bnssassembler::SubtractToken** (*p.477*).

Referenced by bnssassembler::OperationToken::process().

virtual std::shared_ptr<Expression> bnssassembler::ExpressionToken::create () const`[pure virtual]`

Creates an expression object out of the token.

Returns:

Pointer to the expression

Exceptions:

Throws	if the token has no corresponding expression object

Implemented in **bnssassembler::LiteralToken** (*p.262*), **bnssassembler::SymbolToken** (*p.505*), **bnssassembler::AddToken** (*p.99*), **bnssassembler::ClosingBraceToken** (*p.118*), **bnssassembler::DivideToken** (*p.159*), **bnssassembler::MultiplyToken** (*p.293*), **bnssassembler::OpeningBraceToken** (*p.300*), and **bnssassembler::SubtractToken** (*p.478*).

virtual int bnssassembler::ExpressionToken::inputPriority () const`[pure virtual], [noexcept]`

Gets the input priority of the token.

Returns:

Input priority of the token

Implemented in **bnssassembler::LiteralToken** (*p.262*), **bnssassembler::SymbolToken** (*p.505*), **bnssassembler::AddToken** (*p.99*), **bnssassembler::ClosingBraceToken** (*p.118*), **bnssassembler::DivideToken** (*p.159*), **bnssassembler::MultiplyToken** (*p.293*), **bnssassembler::OpeningBraceToken** (*p.300*), and **bnssassembler::SubtractToken** (*p.478*).

Referenced by bnssassembler::OperationToken::process().

virtual void bnssassembler::ExpressionToken::process (std::list< std::shared_ptr< ExpressionToken >> & *output*, std::stack< std::shared_ptr< ExpressionToken >> & *stack*, int & *expression_rank*) const`[pure virtual]`

Processes the current token.

Parameters:

output	Output list of tokens
stack	Helper stack of tokens
expression_rank	Rank of the expression

Implemented in **bnssassembler::LiteralToken** (*p.262*), **bnssassembler::SymbolToken** (*p.505*), and **bnssassembler::OperationToken** (*p.315*).

virtual int bnssassembler::ExpressionToken::rank () const`[pure virtual], [noexcept]`

Gets the rank of the token.

Returns:

Rank of the token

Implemented in **bnssassembler::LiteralToken** (*p.263*), **bnssassembler::SymbolToken** (*p.506*), **bnssassembler::AddToken** (*p.99*), **bnssassembler::ClosingBraceToken** (*p.119*), **bnssassembler::DivideToken** (*p.159*), **bnssassembler::MultiplyToken** (*p.293*), **bnssassembler::OpeningBraceToken** (*p.300*), and **bnssassembler::SubtractToken** (*p.478*).

virtual int bnssassembler::ExpressionToken::stackPriority () const`[pure virtual], [noexcept]`

Gets the stack priority of the token.

Returns:

Stack priority of the token

Implemented in **bnssassembler::LiteralToken** (*p.263*), **bnssassembler::SymbolToken** (*p.506*), **bnssassembler::AddToken** (*p.100*), **bnssassembler::ClosingBraceToken** (*p.119*), **bnssassembler::DivideToken** (*p.160*), **bnssassembler::MultiplyToken** (*p.294*), **bnssassembler::OpeningBraceToken** (*p.301*), and **bnssassembler::SubtractToken** (*p.479*).

Friends And Related Function Documentation

friend class ExpressionTokenFactory`[friend]`

Definition at line 59 of file ExpressionToken.h.

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

- Code/Assembler/Include/**ExpressionToken.h**

bnssassembler::ExpressionTokenFactory Class Reference

Utility class used for creating the **ExpressionToken** objects.
`#include <ExpressionTokenFactory.h>`

Classes

- struct **ExpressionTokenFactoryData**

Static Public Member Functions

- static std::shared_ptr< **ExpressionToken** > **create** (std::string param)
  Creates an expression token using the provided string.

Private Member Functions

- **ExpressionTokenFactory** ()=delete
- **ExpressionTokenFactory** (**ExpressionTokenFactory** &)=delete
- void **operator=** (**ExpressionTokenFactory** &)=delete

Static Private Member Functions

- static **ExpressionTokenFactoryData** & **staticData** () noexcept

---

Detailed Description

Utility class used for creating the **ExpressionToken** objects.

Definition at line 12 of file ExpressionTokenFactory.h.

Constructor & Destructor Documentation

bnssassembler::ExpressionTokenFactory::ExpressionTokenFactory () `[private]`, `[delete]`

bnssassembler::ExpressionTokenFactory::ExpressionTokenFactory (ExpressionTokenFactory &) `[private]`, `[delete]`

Member Function Documentation

std::shared_ptr< ExpressionToken > bnssassembler::ExpressionTokenFactory::create (std::string *param*) `[static]`

Creates an expression token using the provided string.

Parameters:

param	String used for token creation

Returns:

Pointer to the created token

Definition at line 16 of file ExpressionTokenFactory.cpp.

References bnssassembler::LITERAL_REGEX, bnssassembler::ExpressionTokenFactory::ExpressionTokenFactoryData::literals_, bnssassembler::OPERATOR_REGEX, bnssassembler::ExpressionTokenFactory::ExpressionTokenFactoryData::operators_, staticData(), bnssassembler::SYMBOL_REGEX, and bnssassembler::ExpressionTokenFactory::ExpressionTokenFactoryData::symbols_.

Referenced by bnssassembler::infixToPostfix().

```
   16
{
   17         if (regex_match(param, LITERAL_REGEX)) {
   18             return staticData().literals_->clone(param);
   19         }
   20
   21         if (regex_match(param, OPERATOR_REGEX)) {
   22             return staticData().operators_[param]->clone(param);
   23         }
   24
   25         if (regex_match(param, SYMBOL_REGEX)) {
   26             return staticData().symbols_->clone(param);
   27         }
   28
   29         throw MessageException("Invalid expression token: " + param);
   30     }
```

void bnssassembler::ExpressionTokenFactory::operator= (ExpressionTokenFactory &)`[private]`, `[delete]`

ExpressionTokenFactory::ExpressionTokenFactoryData & bnssassembler::ExpressionTokenFactory::staticData ()`[static]`, `[private]`, `[noexcept]`

Definition at line 32 of file ExpressionTokenFactory.cpp.

Referenced by create().

```
   32
{
   33         static ExpressionTokenFactoryData static_data;
   34         return static_data;
   35     }
```

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**ExpressionTokenFactory.h**
- Code/Assembler/Source/**ExpressionTokenFactory.cpp**

bnssassembler::ExpressionTokenFactory::ExpressionTokenFactoryData Struct Reference

Public Member Functions

- **ExpressionTokenFactoryData** () noexcept

Public Attributes

- std::unordered_map< std::string, std::shared_ptr< **ExpressionToken** > > **operators_**
- std::shared_ptr< **ExpressionToken** > **literals_**
- std::shared_ptr< **ExpressionToken** > **symbols_**

Detailed Description

Definition at line 21 of file ExpressionTokenFactory.h.

Constructor & Destructor Documentation

bnssassembler::ExpressionTokenFactory::ExpressionTokenFactoryData::ExpressionTokenFactoryData () `[noexcept]`

Definition at line 37 of file ExpressionTokenFactory.cpp.

```
   37
{
   38         literals_ = std::make_shared<LiteralToken>("1");
   39         symbols_ = std::make_shared<SymbolToken>("OvdeMoguStaHocuDaNapisem");
   40
   41         auto add = std::make_shared<AddToken>();
   42         operators_[add->operation()] = add;
   43         auto sub = std::make_shared<SubtractToken>();
   44         operators_[sub->operation()] = sub;
   45         auto mul = std::make_shared<MultiplyToken>();
   46         operators_[mul->operation()] = mul;
   47         auto div = std::make_shared<DivideToken>();
   48         operators_[div->operation()] = div;
   49         auto opb = std::make_shared<OpeningBraceToken>();
   50         operators_[opb->operation()] = opb;
   51         auto clb = std::make_shared<ClosingBraceToken>();
   52         operators_[clb->operation()] = clb;
   53     }
```

Member Data Documentation

std::shared_ptr<ExpressionToken> bnssassembler::ExpressionTokenFactory::ExpressionTokenFactoryData::literals_

Definition at line 23 of file ExpressionTokenFactory.h.

Referenced by bnssassembler::ExpressionTokenFactory::create().

std::unordered_map<std::string, std::shared_ptr<ExpressionToken> > bnssassembler::ExpressionTokenFactory::ExpressionTokenFactoryData::operators_

Definition at line 22 of file ExpressionTokenFactory.h.

Referenced by bnssassembler::ExpressionTokenFactory::create().

std::shared_ptr<ExpressionToken> bnssassembler::ExpressionTokenFactory::ExpressionTokenFactoryData::symbols_

Definition at line 24 of file ExpressionTokenFactory.h.

Referenced by bnssassembler::ExpressionTokenFactory::create().

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

- Code/Assembler/Include/**ExpressionTokenFactory.h**
- Code/Assembler/Source/**ExpressionTokenFactory.cpp**

bnssemulator::FileReader Class Reference

Utility class used for reading assembler output from the file.
`#include <FileReader.h>`

Static Public Member Functions

- static **AssemblerOutput parse** (std::string filename)
  Parses the input file.

Private Member Functions

- **FileReader** ()=delete
- **FileReader** (**FileReader** &)=delete
- void **operator=** (**FileReader** &)=delete

Detailed Description

Utility class used for reading assembler output from the file.

Definition at line 11 of file FileReader.h.

Constructor & Destructor Documentation

bnssemulator::FileReader::FileReader () `[private]`, `[delete]`

bnssemulator::FileReader::FileReader (FileReader &) `[private]`, `[delete]`

Member Function Documentation

void bnssemulator::FileReader::operator= (FileReader &) `[private]`, `[delete]`

AssemblerOutput bnssemulator::FileReader::parse (std::string *filename*) `[static]`

Parses the input file.

Parameters:

filename	Name of the input file

Returns:

Object containing the parsed input file

Definition at line 8 of file FileReader.cpp.

References z85::decode_with_padding(), and bnssemulator::StringHelper::fileToString().

Referenced by main().

```
                                                    {
        auto raw_file = StringHelper::fileToString(filename);
        auto decoded = z85::decode_with_padding(raw_file);
        std::stringstream stringstream(decoded);
        AssemblerOutput ret;
        stringstream >> ret;
        return ret;
    }
```

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**FileReader.h**
- Code/Emulator/Source/**FileReader.cpp**

bnssassembler::FileReader Class Reference

Utility class providing methods for reading the file.
`#include <FileReader.h>`

Static Public Member Functions

- static std::vector< std::string > **readAllLines** (std::string filename)
  Reads all lines of the file.

Private Member Functions

- **FileReader** ()=delete
- **FileReader** (**FileReader** &)=delete
- void **operator=** (**FileReader** &)=delete

Detailed Description

Utility class providing methods for reading the file.

Definition at line 11 of file FileReader.h.

Constructor & Destructor Documentation

bnssassembler::FileReader::FileReader () `[private]`, `[delete]`

bnssassembler::FileReader::FileReader (FileReader &) `[private]`, `[delete]`

Member Function Documentation

void bnssassembler::FileReader::operator= (FileReader &) `[private]`, `[delete]`

std::vector< std::string > bnssassembler::FileReader::readAllLines (std::string *filename*) `[static]`

Reads all lines of the file.

Parameters:

filename	Name of the file

Returns:

All lines of the file

Exceptions:

Throws	if the file does not exist or could not be opened for reading

Definition at line 6 of file FileReader.cpp.

References bnssassembler::StringHelper::fileToString(), and bnssassembler::StringHelper::split().

Referenced by main().

```
                                                              {
        auto raw_file = StringHelper::fileToString(filename);
        return StringHelper::split(raw_file, "\n");
```

```
}
```

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**FileReader.h**
- Code/Assembler/Source/**FileReader.cpp**

bnssassembler::FileWriter Class Reference

Utility class used to write the assembler result to a file.
`#include <FileWriter.h>`

Static Public Member Functions

- static void **write** (std::string filename, const **SecondPassData** &data)
  Writes the data to the file.

Private Member Functions

- **FileWriter** ()=delete
- **FileWriter** (**FileWriter** &)=delete
- void **operator=** (**FileWriter** &)=delete

Detailed Description

Utility class used to write the assembler result to a file.

Definition at line 11 of file FileWriter.h.

Constructor & Destructor Documentation

bnssassembler::FileWriter::FileWriter () `[private]`, `[delete]`

bnssassembler::FileWriter::FileWriter (FileWriter &) `[private]`, `[delete]`

Member Function Documentation

void bnssassembler::FileWriter::operator= (FileWriter &) `[private]`, `[delete]`

void bnssassembler::FileWriter::write (std::string *filename*, const SecondPassData & *data*) `[static]`

Writes the data to the file.

Parameters:

filename	Name of the file
data	**Data** to be written to the file

Exceptions:

Throws	in case of I/O error

Definition at line 9 of file FileWriter.cpp.

References z85::encode_with_padding().

Referenced by main().

```
    9                                                                        {
   10         std::ofstream out_file(filename, std::ofstream::binary);
   11         // TODO: Somehow write the input file name and timestamp
   12         std::ostringstream stringstream(std::ostringstream::out |
std::ostringstream::binary);
```

```
        stringstream << data;
        auto encoded = z85::encode_with_padding(stringstream.str());
        out_file << encoded;
    }
```

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**FileWriter.h**
- Code/Assembler/Source/**FileWriter.cpp**

bnssassembler::FirstPass Class Reference

Class representing the executor of the first pass.
`#include <FirstPass.h>`

Static Public Member Functions

- static **FirstPassData** **execute** (std::vector< std::shared_ptr< **Token** >> tokens)
  Executes the first pass.

Private Member Functions

- **FirstPass** ()=delete
- **FirstPass** (**FirstPass** &)=delete
- void **operator=** (**FirstPass** &)=delete

Detailed Description

Class representing the executor of the first pass.

Definition at line 11 of file FirstPass.h.

Constructor & Destructor Documentation

bnssassembler::FirstPass::FirstPass () `[private]`, `[delete]`

bnssassembler::FirstPass::FirstPass (FirstPass &) `[private]`, `[delete]`

Member Function Documentation

FirstPassData bnssassembler::FirstPass::execute (std::vector< std::shared_ptr< Token >> *tokens*) `[static]`

Executes the first pass.

Parameters:

tokens	Vector of parsed tokens

Returns:

FirstPassData object

Definition at line 7 of file FirstPass.cpp.

References bnssassembler::MessageException::message(), and bnssassembler::FirstPassData::symbolDefinitions().

Referenced by main().

```
                                                                          {
        FirstPassData ret;
        for (auto &token : tokens) {
            try {
                token->resolveSymbolDefinitions(ret.symbolDefinitions());
                token->firstPass(ret);
            }
            catch (MessageException &exception) {
                throw FirstPassException(token->lineNumber(), token->line(),
exception.message());
            }
        }

        return ret;
    }
```

void bnssassembler::FirstPass::operator= (FirstPass &) `[private]`, `[delete]`

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**FirstPass.h**
- Code/Assembler/Source/**FirstPass.cpp**

bnssassembler::FirstPassData Class Reference

Class representing the data that the two-pass assembler will modify in the first pass.
`#include <FirstPassData.h>`

Public Member Functions

- void **incLocationCounter** (size_t offset)
  Increases the location counter of the current section.
- void **insertSymbol** (std::string symbol)
  Inserts a symbol into the symbol table.
- void **insertSection** (**SectionType** type)
  Inserts a non-indexed section into the section table.
- void **insertSection** (**SectionType** type, size_t index)
  Inserts an indexed section into the section table.
- void **insertSymbolDefinition** (**SymbolDefinition** symbol)
  Inserts a symbol definition into the vector.
- std::unordered_set< **SymbolDefinition** > **symbolDefinitions** () const noexcept
  Get the symbol definitions.

Private Member Functions

- void **insertSection** (**SectionData** section_data)
  Inserts a section into the section table.

Private Attributes

- **SymbolTable symbol_table_**
- **SectionTable section_table_**
- std::unordered_set< **SymbolDefinition** > **symbol_definitions_**

Friends

- class **SecondPassData**

Detailed Description

Class representing the data that the two-pass assembler will modify in the first pass.

Definition at line 13 of file FirstPassData.h.

Member Function Documentation

void bnssassembler::FirstPassData::incLocationCounter (size_t *offset*)

Increases the location counter of the current section.

Definition at line 6 of file FirstPassData.cpp.

References section_table_.

Referenced by bnssassembler::DataDefinitionToken::firstPass(), and bnssassembler::InstructionToken::firstPass().

```
    6                                                   {
    7         if (section_table_.empty()) {
            throw MessageException("All data must be in sections");
        }

        section_table_.back().incLocationCounter(offset);
    }
```

void bnssassembler::FirstPassData::insertSection (SectionType *type*)

Inserts a non-indexed section into the section table.

Parameters:

type	Type of the section

Definition at line 27 of file FirstPassData.cpp.

Referenced by bnssassembler::SectionStartToken::firstPass(), and insertSection().

```
                                                    {
        insertSection(SectionData(type));
    }
```

void bnssassembler::FirstPassData::insertSection (SectionType *type*, size_t *index*)

Inserts an indexed section into the section table.

Parameters:

type	Type of the section
index	Index of the section

Definition at line 31 of file FirstPassData.cpp.

References insertSection().

```
                                                                    {
        insertSection(SectionData(type, index));
    }
```

void bnssassembler::FirstPassData::insertSection (SectionData *section_data*)[private]

Inserts a section into the section table.

Parameters:

section_data	Section data to be inserted

Definition at line 47 of file FirstPassData.cpp.

References section_table_.

```
                                                              {
        section_table_ += section_data;
    }
```

void bnssassembler::FirstPassData::insertSymbol (std::string *symbol*)

Inserts a symbol into the symbol table.

Definition at line 14 of file FirstPassData.cpp.

References bnssassembler::SymbolTable::contains(), section_table_, and symbol_table_.

Referenced by bnssassembler::LabelToken::firstPass().

```
                                                   {
        if (section_table_.empty()) {
            throw MessageException("All labels must be in sections");
```

```
   17         }
   18
   19         if (symbol_table_.contains(symbol)) {
   20             throw MessageException("Symbol " + symbol + " is already defined");
   21         }
   22
   23         SymbolData symbol_data(symbol, section_table_.size() - 1,
section_table_.back().locationCounter(), true);
   24         symbol_table_ += symbol_data;
   25     }
```

void bnssassembler::FirstPassData::insertSymbolDefinition (SymbolDefinition *symbol*)

Inserts a symbol definition into the vector.

Parameters:

symbol	**SymbolDefinition** object to insert

Definition at line 35 of file FirstPassData.cpp.

References bnssassembler::SymbolDefinition::name(), and symbol_definitions_.

Referenced by bnssassembler::SymbolDefinitionToken::firstPass().

```
   35                                                                       {
   36         if (symbol_definitions_.count(symbol) > 0) {
   37             throw MessageException("Symbol " + symbol.name() + " is already
defined");
   38         }
   39
   40         symbol_definitions_.insert(symbol);
   41     }
```

std::unordered_set< SymbolDefinition > bnssassembler::FirstPassData::symbolDefinitions () const`[noexcept]`

Get the symbol definitions.

Returns:

Symbol definitions

Definition at line 43 of file FirstPassData.cpp.

References symbol_definitions_.

Referenced by bnssassembler::FirstPass::execute().

```
   43
{
   44         return symbol_definitions_;
   45     }
```

Friends And Related Function Documentation

friend class SecondPassData`[friend]`

Definition at line 50 of file FirstPassData.h.

Member Data Documentation

SectionTable bnssassembler::FirstPassData::section_table_ `[private]`

Definition at line 53 of file FirstPassData.h.

Referenced by incLocationCounter(), insertSection(), and insertSymbol().

std::unordered_set<SymbolDefinition> bnssassembler::FirstPassData::symbol_definitions_ `[private]`

Definition at line 54 of file FirstPassData.h.

Referenced by insertSymbolDefinition(), and symbolDefinitions().

SymbolTable bnssassembler::FirstPassData::symbol_table_ `[private]`

Definition at line 52 of file FirstPassData.h.

Referenced by insertSymbol().

The documentation for this class was generated from the following files:

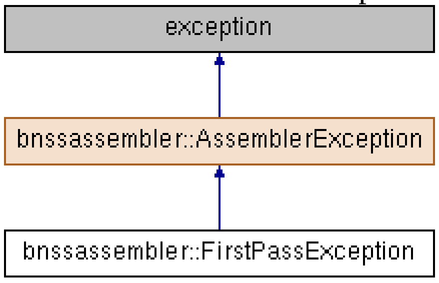
- Code/Assembler/Include/**FirstPassData.h**
- Code/Assembler/Source/**FirstPassData.cpp**

bnssassembler::FirstPassException Class Reference

Represents an exception that happend during the assembler first pass.

`#include <FirstPassException.h>`

Inheritance diagram for bnssassembler::FirstPassException:


exception
bnssassembler::AssemblerException
bnssassembler::FirstPassException

Public Member Functions

- **FirstPassException** (size_t line_number, std::string line, std::string specific_message) noexcept
  *Constructs a **FirstPassException** object.*

Protected Member Functions

- std::string **messageBody** () const noexcept override
  Returns the actual message body of the exception.

Private Attributes

- std::string **specific_message_**

Detailed Description

Represents an exception that happend during the assembler first pass.

Definition at line 10 of file FirstPassException.h.

Constructor & Destructor Documentation

bnssassembler::FirstPassException::FirstPassException (size_t *line_number*, std::string *line*, std::string *specific_message*) `[noexcept]`

Constructs a **FirstPassException** object.

Parameters:

line_number	Number of the line where the error happened
line	Line where the error happened
specific_message	Specific message about the error that happened

Definition at line 5 of file FirstPassException.cpp.

```
    5 : AssemblerException(line_number, line), specific_message_(specific_message) {}
```

Member Function Documentation

std::string bnssassembler::FirstPassException::messageBody () const `[override]`, `[protected]`, `[virtual]`, `[noexcept]`

Returns the actual message body of the exception.

Implements **bnssassembler::AssemblerException** (*p.109*).

Definition at line 7 of file FirstPassException.cpp.

References specific_message_.

```
    7                                                                  {
    8         return "Error during the first pass\n" + specific_message_;
    9     }
```

Member Data Documentation

std::string bnssassembler::FirstPassException::specific_message_ `[private]`

Definition at line 22 of file FirstPassException.h.

Referenced by messageBody().

The documentation for this class was generated from the following files:

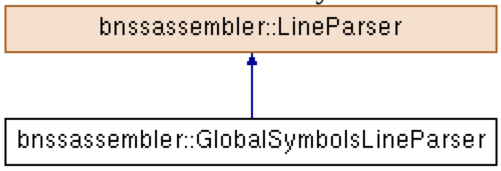
- Code/Assembler/Include/**FirstPassException.h**
- Code/Assembler/Source/**FirstPassException.cpp**

bnssassembler::GlobalSymbolsLineParser Class Reference

Class used for parsing information about global symbols.

`#include <GlobalSymbolsLineParser.h>`

Inheritance diagram for bnssassembler::GlobalSymbolsLineParser:


bnssassembler::LineParser
bnssassembler::GlobalSymbolsLineParser

Protected Member Functions

- std::shared_ptr< **Token** > **parse** (const std::string &line, size_t line_number, std::string initial_line) const override
  Parses one line of the file. Does not call the next parser in chain.

Additional Inherited Members

Detailed Description

Class used for parsing information about global symbols.

Definition at line 10 of file GlobalSymbolsLineParser.h.

Member Function Documentation

std::shared_ptr< Token > bnssassembler::GlobalSymbolsLineParser::parse (const std::string & *line*, size_t *line_number*, std::string *initial_line*) const `[override]`, `[protected]`, `[virtual]`

Parses one line of the file. Does not call the next parser in chain.

Parameters:

line	Line to parse
line_number	Number of the line that is parsed
initial_line	Initial line that is parsed

Returns:

Extracted token from line or nullptr if the parser failed parsing the line

Exceptions:

Throws	if the parser failed and identified the error

Implements **bnssassembler::LineParser** (*p.257*).

Definition at line 9 of file GlobalSymbolsLineParser.cpp.

References bnssassembler::GLOBAL_DIRECTIVE, bnssassembler::StringHelper::split(), and bnssassembler::SYMBOL.

```
{
        static std::regex regex("[[:space:]]*" + GLOBAL_DIRECTIVE +
"(.*)[[:space:]]*");
        static std::regex symbol_regex("[[:space:]]*(" + SYMBOL +
")[[:space:]]*");
```

```

        if (!regex_match(line, regex)) {
            return nullptr;
        }

        auto symbols_string = regex_replace(line, regex, "$1");
        auto symbols = StringHelper::split(symbols_string, ",");

        std::vector<std::string> ret;
        for (auto &symbol : symbols) {
            if (!regex_match(symbol, symbol_regex)) {
                throw MessageException(symbol + " is not a valid symbol");
            }

            ret.push_back(regex_replace(symbol, symbol_regex, "$1"));
        }

        return std::make_shared<GlobalSymbolsToken>(ret, line_number, initial_line);
    }
```

The documentation for this class was generated from the following files:

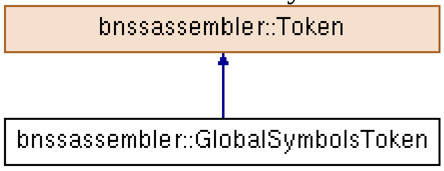
- Code/Assembler/Include/**GlobalSymbolsLineParser.h**
- Code/Assembler/Source/**GlobalSymbolsLineParser.cpp**

bnssassembler::GlobalSymbolsToken Class Reference

Class representing the global symbols token.

`#include <GlobalSymbolToken.h>`

Inheritance diagram for bnssassembler::GlobalSymbolsToken:



Public Member Functions

- **GlobalSymbolsToken** (std::vector< std::string > symbols, size_t line_number, std::string **line**) noexcept
  *Constructs a **GlobalSymbolsToken** object.*
- void **firstPass** (**FirstPassData** &data) const override
  Executes the first pass over the token.
- void **secondPass** (**SecondPassData** &data) const override
  Executes the second pass over the token.

Private Attributes

- std::vector< std::string > **symbols_**

Detailed Description

Class representing the global symbols token.

Definition at line 11 of file GlobalSymbolToken.h.

Constructor & Destructor Documentation

bnssassembler::GlobalSymbolsToken::GlobalSymbolsToken (std::vector< std::string > *symbols*, size_t *line_number*, std::string *line*) `[noexcept]`

Constructs a **GlobalSymbolsToken** object.

Parameters:

symbols	Vector of global symbols
line_number	Number of the line where the symbols are defined
line	Line where symbols are defined

Definition at line 6 of file GlobalSymbolToken.cpp.

```
    6 : Token(line_number, line), symbols_(symbols) {}
```

Member Function Documentation

void bnssassembler::GlobalSymbolsToken::firstPass (FirstPassData & *data*) const `[override]`, `[virtual]` Executes the first pass over the token.

Parameters:

data	**Data** that the token will modify

Implements **bnssassembler::Token** (*p.510*).

Definition at line 8 of file GlobalSymbolToken.cpp.

```
                                                                   {
        // Do nothing
    }
```

void bnssassembler::GlobalSymbolsToken::secondPass (SecondPassData & *data*) const`[override]`, `[virtual]`

Executes the second pass over the token.

Parameters:

data	**Data** that the token will modify

Implements **bnssassembler::Token** (*p.512*).

Definition at line 12 of file GlobalSymbolToken.cpp.

References bnssassembler::SecondPassData::contains(), bnssassembler::SecondPassData::exportSymbol(), bnssassembler::SecondPassData::insertImported(), and symbols_.

```
                                                                      {
        for (const auto &symbol : symbols_) {
            if (data.contains(symbol)) {
                data.exportSymbol(symbol);
            }
            else {
                data.insertImported(symbol);
            }
        }
    }
```

Member Data Documentation

std::vector<std::string> bnssassembler::GlobalSymbolsToken::symbols_`[private]`

Definition at line 24 of file GlobalSymbolToken.h.

Referenced by secondPass().

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**GlobalSymbolToken.h**
- Code/Assembler/Source/**GlobalSymbolToken.cpp**

std::hash< bnssassembler::InstructionCode > Struct Template Reference

`#include <InstructionCode.h>`

Public Member Functions

- size_t **operator()** (const **bnssassembler::InstructionCode** &code) const

---

Detailed Description

template<>

struct std::hash< bnssassembler::InstructionCode >

Definition at line 45 of file InstructionCode.h.

Member Function Documentation

size_t std::hash< bnssassembler::InstructionCode >::operator() (const bnssassembler::InstructionCode & *code*) const`[inline]`

Definition at line 49 of file InstructionCode.h.

```
{
        return hash<int8_t>()(static_cast<int8_t>(code));
    }
```

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

- Code/Assembler/Include/**InstructionCode.h**

std::hash< bnssassembler::SectionData > Struct Template Reference

`#include <SectionData.h>`

Public Member Functions

- size_t **operator()** (const **bnssassembler::SectionData** &section_data) const

Detailed Description

template<>

struct std::hash< bnssassembler::SectionData >

Definition at line 138 of file SectionData.h.

Member Function Documentation

size_t std::hash< bnssassembler::SectionData >::operator() (const bnssassembler::SectionData & *section_data*) const`[inline]`

Definition at line 142 of file SectionData.h.

References bnssassembler::SectionData::hash().

```
{
        return hash<size_t>()(section_data.hash());
    }
```

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

- Code/Assembler/Include/**SectionData.h**

std::hash< bnssassembler::SectionType > Struct Template Reference

`#include <SectionType.h>`

Public Member Functions

- size_t **operator()** (const **bnssassembler::SectionType** &type) const

---

Detailed Description

template<>

struct std::hash< bnssassembler::SectionType >

Definition at line 20 of file SectionType.h.

Member Function Documentation

size_t std::hash< bnssassembler::SectionType >::operator() (const bnssassembler::SectionType & *type*) const `[inline]`

Definition at line 24 of file SectionType.h.

```
{
        return hash<int8_t>()(static_cast<int8_t>(type));
    }
```

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

- Code/Assembler/Include/**SectionType.h**

std::hash< bnssassembler::SymbolDefinition > Struct Template Reference

`#include <SymbolDefinition.h>`

Public Member Functions

- size_t **operator()** (const **bnssassembler::SymbolDefinition** &symbol) const

Detailed Description

template<>

struct std::hash< bnssassembler::SymbolDefinition >

Definition at line 47 of file SymbolDefinition.h.

Member Function Documentation

size_t std::hash< bnssassembler::SymbolDefinition >::operator() (const bnssassembler::SymbolDefinition & *symbol*) const`[inline]`

Definition at line 51 of file SymbolDefinition.h.

References bnssassembler::SymbolDefinition::name().

```
   51
{
   52         return hash<string>()(symbol.name());
   53     }
```

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

- Code/Assembler/Include/**SymbolDefinition.h**

std::hash< bnssemulator::InstructionCode > Struct Template Reference

`#include <InstructionCode.h>`

Public Member Functions

- size_t **operator()** (const **bnssemulator::InstructionCode** &code) const

---

Detailed Description

template<>

struct std::hash< bnssemulator::InstructionCode >

Definition at line 45 of file InstructionCode.h.

Member Function Documentation

size_t std::hash< bnssemulator::InstructionCode >::operator() (const bnssemulator::InstructionCode & ***code*****) const**`[inline]`

Definition at line 49 of file InstructionCode.h.

```
{
        return hash<int8_t>()(static_cast<int8_t>(code));
    }
```

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

- Code/Emulator/Include/**InstructionCode.h**

std::hash< bnssemulator::SectionType > Struct Template Reference

`#include <SectionType.h>`

Public Member Functions

- size_t **operator()** (const **bnssemulator::SectionType** &type) const

Detailed Description

template<>

struct std::hash< bnssemulator::SectionType >

Definition at line 20 of file SectionType.h.

Member Function Documentation

size_t std::hash< bnssemulator::SectionType >::operator() (const bnssemulator::SectionType & *type*) const`[inline]`

Definition at line 24 of file SectionType.h.

```
{
        return hash<int8_t>()(static_cast<int8_t>(type));
    }
```

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

- Code/Emulator/Include/**SectionType.h**

cxxopts::HelpGroupDetails Struct Reference

```
#include <cxxopts.h>
```

Public Attributes

- std::string **name**
- std::string **description**
- std::vector< **HelpOptionDetails** > **options**

Detailed Description

Definition at line 660 of file cxxopts.h.

Member Data Documentation

std::string cxxopts::HelpGroupDetails::description

Definition at line 663 of file cxxopts.h.

std::string cxxopts::HelpGroupDetails::name

Definition at line 662 of file cxxopts.h.

std::vector< HelpOptionDetails > cxxopts::HelpGroupDetails::options

Definition at line 664 of file cxxopts.h.

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

- Code/Assembler/Include/**cxxopts.h**

cxxopts::HelpOptionDetails Struct Reference

`#include <cxxopts.h>`

Public Attributes

- std::string **s**
- std::string **l**
- **String desc**
- bool **has_arg**
- bool **has_default**
- std::string **default_value**
- bool **has_implicit**
- std::string **implicit_value**
- std::string **arg_help**
- bool **is_container**

Detailed Description

Definition at line 646 of file cxxopts.h.

Member Data Documentation

std::string cxxopts::HelpOptionDetails::arg_help

Definition at line 656 of file cxxopts.h.

Referenced by cxxopts::anonymous_namespace{cxxopts.h}::format_option(), and cxxopts::OptionAdder::OptionAdder().

std::string cxxopts::HelpOptionDetails::default_value

Definition at line 653 of file cxxopts.h.

Referenced by cxxopts::anonymous_namespace{cxxopts.h}::format_description(), and cxxopts::OptionAdder::OptionAdder().

String cxxopts::HelpOptionDetails::desc

Definition at line 650 of file cxxopts.h.

Referenced by cxxopts::anonymous_namespace{cxxopts.h}::format_description(), and cxxopts::OptionAdder::OptionAdder().

bool cxxopts::HelpOptionDetails::has_arg

Definition at line 651 of file cxxopts.h.

Referenced by cxxopts::Options::add_option(), cxxopts::anonymous_namespace{cxxopts.h}::format_option(), and cxxopts::OptionAdder::OptionAdder().

bool cxxopts::HelpOptionDetails::has_default

Definition at line 652 of file cxxopts.h.

Referenced by cxxopts::anonymous_namespace{cxxopts.h}::format_description(), and cxxopts::OptionAdder::OptionAdder().

bool cxxopts::HelpOptionDetails::has_implicit

Definition at line 654 of file cxxopts.h.

Referenced by cxxopts::anonymous_namespace{cxxopts.h}::format_option(), and cxxopts::OptionAdder::OptionAdder().

std::string cxxopts::HelpOptionDetails::implicit_value

Definition at line 655 of file cxxopts.h.

Referenced by cxxopts::anonymous_namespace{cxxopts.h}::format_option(), and cxxopts::OptionAdder::OptionAdder().

bool cxxopts::HelpOptionDetails::is_container

Definition at line 657 of file cxxopts.h.

std::string cxxopts::HelpOptionDetails::l

Definition at line 649 of file cxxopts.h.

Referenced by cxxopts::anonymous_namespace{cxxopts.h}::format_option(), and cxxopts::OptionAdder::OptionAdder().

std::string cxxopts::HelpOptionDetails::s

Definition at line 648 of file cxxopts.h.

Referenced by cxxopts::anonymous_namespace{cxxopts.h}::format_option(), and cxxopts::OptionAdder::OptionAdder().

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

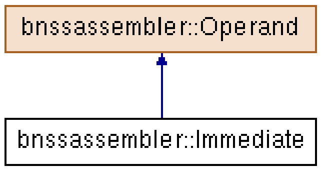
- Code/Assembler/Include/**cxxopts.h**

bnssassembler::Immediate Class Reference

Class representing the immediate operand.

`#include <Immediate.h>`

Inheritance diagram for bnssassembler::Immediate:



Public Member Functions

- **Immediate** (**MicroRiscExpression** value) noexcept
  *Constructs an **Immediate** object.*
- void **packToInstruction** (**InstructionBitFieldUnion** &instruction, **uint32_t** &second_word, std::list< **RelocationRecord** > &relocations) const override
  Packs the operand into the instruction.
- void **resolveSymbols** (std::unordered_set< **SymbolDefinition** > symbols) noexcept override
  Resolves the defined symbols in the expressions.
- void **resolveSymbolTable** (const **SymbolTable** &symbol_table) noexcept override
  Resolves the symbols from the symbol table and updates the relocation info.
- void **resolveImports** (std::unordered_set< std::string > imported_symbols) noexcept override
  Resolves the imported symbols and updates the relocation info.
- void **resolveCurrentPcSymbol** (size_t section_index, size_t offset) noexcept override
  Resolves the current PC symbol and sets the relocation info.
- **AddressMode addressMode** () const noexcept override
  Gets the address mode of the operand.

Private Attributes

- **MicroRiscExpression value_**

Detailed Description

Class representing the immediate operand.

Definition at line 11 of file Immediate.h.

Constructor & Destructor Documentation

bnssassembler::Immediate::Immediate (MicroRiscExpression *value*) `[explicit]`, `[noexcept]`

Constructs an **Immediate** object.

Parameters:

value	Value of the immediate operand

Definition at line 5 of file Immediate.cpp.

```
    5 : value_(value) {}
```

Member Function Documentation

AddressMode bnssassembler::Immediate::addressMode () const `[override]`, `[virtual]`, `[noexcept]`

Gets the address mode of the operand.

Returns:

Address mode of the operand

Implements **bnssassembler::Operand** (*p.302*).

Definition at line 31 of file Immediate.cpp.

References bnssassembler::IMMEDIATE.

```
   31                                                           {
   32           return IMMEDIATE;
   33       }
```

void bnssassembler::Immediate::packToInstruction (InstructionBitFieldUnion & *instruction*, uint32_t & *second_word*, std::list< RelocationRecord > & *relocations*) const `[override]`, `[virtual]`

Packs the operand into the instruction.

Parameters:

instruction	Reference to the first word of the instruction containing the instruction info
second_word	Reference to the second word of the instruction containing the address/value/displacement
relocations	Reference to the list of relocation records

Implements **bnssassembler::Operand** (*p.303*).

Definition at line 7 of file Immediate.cpp.

References bnssassembler::InstructionBitField::address_mode, bnssassembler::InstructionBitFieldUnion::bit_field, bnssassembler::MicroRiscExpression::generateRelocations(), bnssassembler::IMMEDIATE, bnssassembler::MicroRiscExpression::value(), and value_.

```
    7
{
    8           instruction.bit_field.address_mode = IMMEDIATE;
    9           second_word = value_.value();
   10           relocations.splice(relocations.end(), value_.generateRelocations());
   11       }
```

void bnssassembler::Immediate::resolveCurrentPcSymbol (size_t *section_index*, size_t *offset*) `[override]`, `[virtual]`, `[noexcept]`

Resolves the current PC symbol and sets the relocation info.

Parameters:

section_index	Current PC section
offset	PC address in relation to the current section beginning

Reimplemented from **bnssassembler::Operand** (*p.303*).

Definition at line 27 of file Immediate.cpp.

References bnssassembler::MicroRiscExpression::resolveCurrentPcSymbol(), and value_.

```
   27
{
   28         value_.resolveCurrentPcSymbol(section_index, offset);
   29     }
```

void bnssassembler::Immediate::resolveImports (std::unordered_set< std::string > *imported_symbols*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the imported symbols and updates the relocation info.

Parameters:

imported_symbols	Collection of imported symbols

Reimplemented from **bnssassembler::Operand** (*p.303*).

Definition at line 23 of file Immediate.cpp.

References bnssassembler::MicroRiscExpression::resolveImports(), and value_.

```
   23
{
   24         value_.resolveImports(imported_symbols);
   25     }
```

void bnssassembler::Immediate::resolveSymbols (std::unordered_set< SymbolDefinition > *symbols*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the defined symbols in the expressions.

Parameters:

symbols	Collection of symbol definitions

Reimplemented from **bnssassembler::Operand** (*p.303*).

Definition at line 13 of file Immediate.cpp.

References bnssassembler::MicroRiscExpression::setValue(), and value_.

```
   13
{
   14         for (auto &symbol : symbols) {
   15             value_.setValue(symbol.name(), symbol.expression());
   16         }
   17     }
```

void bnssassembler::Immediate::resolveSymbolTable (const SymbolTable & *symbol_table*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the symbols from the symbol table and updates the relocation info.

Parameters:

symbol_table	**Symbol** table

Reimplemented from **bnssassembler::Operand** (*p.304*).

Definition at line 19 of file Immediate.cpp.

References bnssassembler::MicroRiscExpression::resolveSymbolTable(), and value_.

```
   19
{
   20         value_.resolveSymbolTable(symbol_table);
   21     }
```

Member Data Documentation

MicroRiscExpression bnssassembler::Immediate::value_ `[private]`

Definition at line 26 of file Immediate.h.

Referenced by packToInstruction(), resolveCurrentPcSymbol(), resolveImports(), resolveSymbols(), and resolveSymbolTable().

The documentation for this class was generated from the following files:

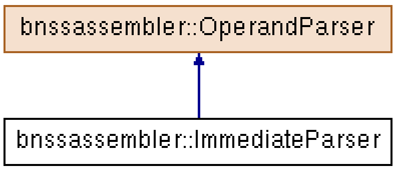
- Code/Assembler/Include/**Immediate.h**
- Code/Assembler/Source/**Immediate.cpp**

bnssassembler::ImmediateParser Class Reference

Class representing the parser for the immediate operands.

`#include <ImmediateParser.h>`

Inheritance diagram for bnssassembler::ImmediateParser:



Protected Member Functions

- std::shared_ptr< **Operand** > **parse** (std::string str) const override
  Parses one operand. Does not call the next parser if it fails.

Additional Inherited Members

Detailed Description

Class representing the parser for the immediate operands.

Definition at line 10 of file ImmediateParser.h.

Member Function Documentation

std::shared_ptr< Operand > bnssassembler::ImmediateParser::parse (std::string *str*) const`[override]`, `[protected]`, `[virtual]`

Parses one operand. Does not call the next parser if it fails.

Parameters:

str	**Operand** which should be parsed

Returns:

Pointer to the operand or nullptr, if the parser failed parsing

Exceptions:

Throws	if the parser fails but identifies the error

Implements **bnssassembler::OperandParser** (*p.306*).

Definition at line 9 of file ImmediateParser.cpp.

References bnssassembler::ExpressionBuilder::build(), and bnssassembler::CONSTANT_TERM.

```
                                                                        {
        static std::regex regex("#(" + CONSTANT_TERM + ")");

        if (!regex_match(str, regex)) {
            return nullptr;
        }

        auto constant_term_string = regex_replace(str, regex, "$1");
        auto expression = ExpressionBuilder::build(constant_term_string);
        return std::make_shared<Immediate>(expression);
    }
```

The documentation for this class was generated from the following files:

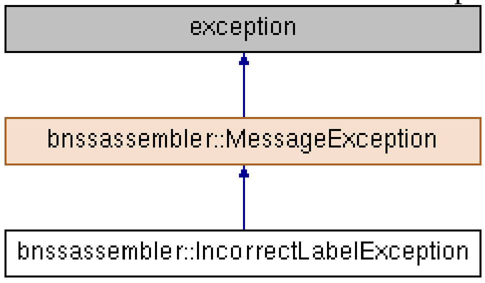
- Code/Assembler/Include/**ImmediateParser.h**
- Code/Assembler/Source/**ImmediateParser.cpp**

bnssassembler::IncorrectLabelException Class Reference

Exception representing the incorrect label.

```
#include <IncorrectLabelException.h>
```

Inheritance diagram for bnssassembler::IncorrectLabelException:


exception
bnssassembler::MessageException
bnssassembler::IncorrectLabelException

Public Member Functions

- **IncorrectLabelException** (std::string label) noexcept
  *Constructs an **IncorrectLabelException** object.*

Detailed Description

Exception representing the incorrect label.

Definition at line 11 of file IncorrectLabelException.h.

Constructor & Destructor Documentation

bnssassembler::IncorrectLabelException::IncorrectLabelException (std::string *label*) `[explicit]`, `[noexcept]`

Constructs an **IncorrectLabelException** object.

Parameters:

label	Label that was incorrect

Definition at line 5 of file IncorrectLabelException.cpp.

```
5 : MessageException("The label \"" + label + "\" is in incorrect format") {}
```

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**IncorrectLabelException.h**
- Code/Assembler/Source/**IncorrectLabelException.cpp**

bnssemulator::InstructionBitField Struct Reference

Bit field that enables easier manipulation of instructions.

```
#include <InstructionBitField.h>
```

Public Attributes

- **uint32_t operation_code**: 8
- **uint32_t address_mode**: 3
- **uint32_t register0**: 5
- **uint32_t register1**: 5
- **uint32_t register2**: 5
- **uint32_t type**: 3
- **uint32_t unused**: 3

Detailed Description

Bit field that enables easier manipulation of instructions.

Definition at line 10 of file InstructionBitField.h.

Member Data Documentation

uint32_t bnssemulator::InstructionBitField::address_mode

Definition at line 12 of file InstructionBitField.h.

Referenced by bnssemulator::StoreExecuter::execute(), bnssemulator::Context::getOperand(), and bnssemulator::Context::getOperandAddress().

uint32_t bnssemulator::InstructionBitField::operation_code

Definition at line 12 of file InstructionBitField.h.

Referenced by bnssemulator::opcode().

uint32_t bnssemulator::InstructionBitField::register0

Definition at line 12 of file InstructionBitField.h.

Referenced by bnssemulator::AluExecuter::execute(), bnssemulator::StoreExecuter::execute(), bnssemulator::PushExecuter::execute(), bnssemulator::PopExecuter::execute(), bnssemulator::ConditionalJumpExecuter::execute(), bnssemulator::LoadExecuter::execute(), bnssemulator::IntExecuter::execute(), bnssemulator::NotExecuter::execute(), and bnssemulator::getRegisterIndex().

uint32_t bnssemulator::InstructionBitField::register1

Definition at line 12 of file InstructionBitField.h.

Referenced by bnssemulator::AluExecuter::execute(), bnssemulator::StoreExecuter::execute(), bnssemulator::NotExecuter::execute(), and bnssemulator::getRegisterIndex().

uint32_t bnssemulator::InstructionBitField::register2

Definition at line 12 of file InstructionBitField.h.

Referenced by bnssemulator::AluExecuter::execute(), and bnssemulator::getRegisterIndex().

uint32_t bnssemulator::InstructionBitField::type

Definition at line 12 of file InstructionBitField.h.

Referenced by bnssemulator::LoadExecuter::execute(), and bnssemulator::StoreExecuter::execute().

uint32_t bnssemulator::InstructionBitField::unused

Definition at line 12 of file InstructionBitField.h.

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

- Code/Emulator/Include/**InstructionBitField.h**

bnssassembler::InstructionBitField Struct Reference

Bit field that enables easier manipulation of instructions.

```
#include <InstructionBitField.h>
```

Public Attributes

- **uint32_t operation_code**: 8
- **uint32_t address_mode**: 3
- **uint32_t register0**: 5
- **uint32_t register1**: 5
- **uint32_t register2**: 5
- **uint32_t type**: 3
- **uint32_t unused**: 3

Detailed Description

Bit field that enables easier manipulation of instructions.

Definition at line 10 of file InstructionBitField.h.

Member Data Documentation

uint32_t bnssassembler::InstructionBitField::address_mode

Definition at line 12 of file InstructionBitField.h.

Referenced by bnssassembler::InstructionToken::packInstruction(), bnssassembler::Immediate::packToInstruction(), bnssassembler::RegisterIndirect::packToInstruction(), bnssassembler::MemoryDirect::packToInstruction(), and bnssassembler::RegisterIndirectOffset::packToInstruction().

uint32_t bnssassembler::InstructionBitField::operation_code

Definition at line 12 of file InstructionBitField.h.

Referenced by bnssassembler::InstructionToken::packInstruction().

uint32_t bnssassembler::InstructionBitField::register0

Definition at line 12 of file InstructionBitField.h.

Referenced by bnssassembler::InstructionToken::packInstruction(), bnssassembler::RegisterDirect::packToInstruction(), bnssassembler::RegisterIndirect::packToInstruction(), and bnssassembler::RegisterIndirectOffset::packToInstruction().

uint32_t bnssassembler::InstructionBitField::register1

Definition at line 12 of file InstructionBitField.h.

Referenced by bnssassembler::InstructionToken::packInstruction(), bnssassembler::RegisterDirect::packToInstruction(), bnssassembler::RegisterIndirect::packToInstruction(), and bnssassembler::RegisterIndirectOffset::packToInstruction().

uint32_t bnssassembler::InstructionBitField::register2

Definition at line 12 of file InstructionBitField.h.

Referenced by bnssassembler::InstructionToken::packInstruction(), bnssassembler::RegisterDirect::packToInstruction(), bnssassembler::RegisterIndirect::packToInstruction(), and bnssassembler::RegisterIndirectOffset::packToInstruction().

uint32_t bnssassembler::InstructionBitField::type

Definition at line 12 of file InstructionBitField.h.

Referenced by bnssassembler::InstructionToken::packInstruction().

uint32_t bnssassembler::InstructionBitField::unused

Definition at line 12 of file InstructionBitField.h.

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

- Code/Assembler/Include/**InstructionBitField.h**

bnssemulator::InstructionBitFieldUnion Union Reference

Union that enables easier manipulation of the instruction bit field.
`#include <InstructionBitFieldUnion.h>`

Public Attributes

- **InstructionBitField bit_field**
- **uint32_t data**

Detailed Description

Union that enables easier manipulation of the instruction bit field.

Definition at line 10 of file InstructionBitFieldUnion.h.

Member Data Documentation

InstructionBitField bnssemulator::InstructionBitFieldUnion::bit_field

Definition at line 11 of file InstructionBitFieldUnion.h.

Referenced by bnssemulator::Segment::getInstruction().

uint32_t bnssemulator::InstructionBitFieldUnion::data

Definition at line 12 of file InstructionBitFieldUnion.h.

Referenced by bnssemulator::Segment::getInstruction().

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

- Code/Emulator/Include/**InstructionBitFieldUnion.h**

bnssassembler::InstructionBitFieldUnion Union Reference

Union that enables easier manipulation of the instruction bit field.

```
#include <InstructionBitFieldUnion.h>
```

Public Attributes

- **InstructionBitField bit_field**
- **uint32_t data**

Detailed Description

Union that enables easier manipulation of the instruction bit field.

Definition at line 10 of file InstructionBitFieldUnion.h.

Member Data Documentation

InstructionBitField bnssassembler::InstructionBitFieldUnion::bit_field

Definition at line 11 of file InstructionBitFieldUnion.h.

Referenced by bnssassembler::InstructionToken::packInstruction(), bnssassembler::Immediate::packToInstruction(), bnssassembler::RegisterIndirect::packToInstruction(), bnssassembler::RegisterDirect::packToInstruction(), bnssassembler::MemoryDirect::packToInstruction(), and bnssassembler::RegisterIndirectOffset::packToInstruction().

uint32_t bnssassembler::InstructionBitFieldUnion::data

Definition at line 12 of file InstructionBitFieldUnion.h.

Referenced by bnssassembler::InstructionToken::packInstruction().

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

- Code/Assembler/Include/**InstructionBitFieldUnion.h**

bnssassembler::InstructionCodeParser Class Reference

Utility class used for parsing instruction codes.

`#include <InstructionCodeParser.h>`

Classes

- struct **InstructionCodeParserStaticData**

Static Public Member Functions

- static **InstructionCode** **parse** (std::string str)
  Parses the instruction code.

Private Member Functions

- **InstructionCodeParser** ()=delete
- **InstructionCodeParser** (**InstructionCodeParser** &)=delete
- void **operator=** (**InstructionCodeParser** &)=delete

Static Private Member Functions

- static **InstructionCodeParserStaticData** & **staticData** () noexcept

---

Detailed Description

Utility class used for parsing instruction codes.

Definition at line 11 of file InstructionCodeParser.h.

Constructor & Destructor Documentation

bnssassembler::InstructionCodeParser::InstructionCodeParser () `[private]`, `[delete]`

bnssassembler::InstructionCodeParser::InstructionCodeParser (InstructionCodeParser &) `[private]`, `[delete]`

Member Function Documentation

void bnssassembler::InstructionCodeParser::operator= (InstructionCodeParser &) `[private]`, `[delete]`

InstructionCode bnssassembler::InstructionCodeParser::parse (std::string *str*) `[static]`

Parses the instruction code.

Parameters:

str	String representing the instruction code

Returns:

Instruction code

Exceptions:

Throws	if the instruction code is not valid

Definition at line 8 of file InstructionCodeParser.cpp.

References bnssassembler::InstructionCodeParser::InstructionCodeParserStaticData::map, and staticData().

Referenced by bnssassembler::InstructionLineParser::parse().

```
                                                                  {
        transform(str.begin(), str.end(), str.begin(), tolower);

        if (staticData().map.count(str) == 0) {
            throw MessageException(str + " is not an instruction code");
        }

        return staticData().map[str];
    }
```

InstructionCodeParser::InstructionCodeParserStaticData & bnssassembler::InstructionCodeParser::staticData () `[static]`, `[private]`, `[noexcept]`

Definition at line 18 of file InstructionCodeParser.cpp.

Referenced by parse().

```
{
        static InstructionCodeParserStaticData static_data;
        return static_data;
    }
```

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**InstructionCodeParser.h**
- Code/Assembler/Source/**InstructionCodeParser.cpp**

bnssassembler::InstructionCodeParser::InstructionCodeParserStaticData Struct Reference

Public Member Functions

- **InstructionCodeParserStaticData** () noexcept

Public Attributes

- std::unordered_map< std::string, **InstructionCode** > **map**

Detailed Description

Definition at line 21 of file InstructionCodeParser.h.

Constructor & Destructor Documentation

bnssassembler::InstructionCodeParser::InstructionCodeParserStaticData::InstructionCodeParserStaticData () `[noexcept]`

Definition at line 23 of file InstructionCodeParser.cpp.

References bnssassembler::ADD, bnssassembler::AND, bnssassembler::ASL, bnssassembler::ASR, bnssassembler::CALL, bnssassembler::DIV, bnssassembler::INT, bnssassembler::JGEZ, bnssassembler::JGZ, bnssassembler::JLEZ, bnssassembler::JLZ, bnssassembler::JMP, bnssassembler::JNZ, bnssassembler::JZ, bnssassembler::LOAD, bnssassembler::MOD, bnssassembler::MUL, bnssassembler::NOT, bnssassembler::OR, bnssassembler::POP, bnssassembler::PUSH, bnssassembler::RET, bnssassembler::STORE, bnssassembler::SUB, and bnssassembler::XOR.

```
   23
{
   24         map["int"]   = INT;
   25         map["jmp"]   = JMP;
   26         map["call"]  = CALL;
   27         map["ret"]   = RET;
   28         map["jz"]    = JZ;
   29         map["jnz"]   = JNZ;
   30         map["jgz"]   = JGZ;
   31         map["jgez"]  = JGEZ;
   32         map["jlz"]   = JLZ;
   33         map["jlez"]  = JLEZ;
   34
   35         map["load"]  = LOAD;
   36         map["store"] = STORE;
   37
   38         map["push"]  = PUSH;
   39         map["pop"]   = POP;
   40
   41         map["add"]   = ADD;
   42         map["sub"]   = SUB;
   43         map["mul"]   = MUL;
   44         map["div"]   = DIV;
   45         map["mod"]   = MOD;
   46         map["and"]   = AND;
   47         map["or"]    = OR;
   48         map["xor"]   = XOR;
   49         map["not"]   = NOT;
   50         map["asl"]   = ASL;
   51         map["asr"]   = ASR;
   52     }
```

Member Data Documentation

std::unordered_map<std::string, InstructionCode> bnssassembler::InstructionCodeParser::InstructionCodeParserStaticData::map

Definition at line 22 of file InstructionCodeParser.h.

Referenced by bnssassembler::InstructionCodeParser::parse().

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

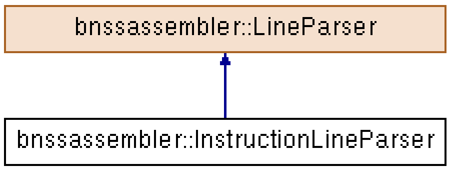
- Code/Assembler/Include/**InstructionCodeParser.h**
- Code/Assembler/Source/**InstructionCodeParser.cpp**

bnssassembler::InstructionLineParser Class Reference

Class used for parsing instructions.

`#include <InstructionLineParser.h>`

Inheritance diagram for bnssassembler::InstructionLineParser:


bnssassembler::LineParser
bnssassembler::InstructionLineParser

Public Member Functions

- **InstructionLineParser** () noexcept
  Constructs an ***InstructionLineParser*** *object.*

Protected Member Functions

- std::shared_ptr< **Token** > **parse** (const std::string &line, size_t line_number, std::string initial_line) const override
  Parses one line of the file. Does not call the next parser in chain.

Private Attributes

- std::unordered_map< **InstructionCode**, std::shared_ptr< **InstructionParser** > > **instructions_**

Detailed Description

Class used for parsing instructions.

Definition at line 14 of file InstructionLineParser.h.

Constructor & Destructor Documentation

bnssassembler::InstructionLineParser::InstructionLineParser () [noexcept]

Constructs an **InstructionLineParser** object.

Definition at line 63 of file InstructionLineParser.cpp.

References bnssassembler::ADD, bnssassembler::AND, bnssassembler::ASL, bnssassembler::ASR, bnssassembler::CALL, bnssassembler::DIV, instructions_, bnssassembler::INT, bnssassembler::JGEZ, bnssassembler::JGZ, bnssassembler::JLEZ, bnssassembler::JLZ, bnssassembler::JMP, bnssassembler::JNZ, bnssassembler::JZ, bnssassembler::LOAD, bnssassembler::MOD, bnssassembler::MUL, bnssassembler::NOT, bnssassembler::OR, bnssassembler::POP, bnssassembler::PUSH, bnssassembler::RET, bnssassembler::STORE, bnssassembler::SUB, and bnssassembler::XOR.

```
   63                                                        {
   64         instructions_[INT] = std::make_shared<InterruptInstructionParser>();
   65         instructions_[RET] = std::make_shared<NoOperandInstructionParser>();
   66
   67         auto uncond_jump =
std::make_shared<UndonditionalJumpInstructionParser>();
   68         instructions_[JMP] = uncond_jump;
   69         instructions_[CALL] = uncond_jump;
   70
   71         auto cond_jump = std::make_shared<ConditionalJumpInstructionParser>();
   72         instructions_[JZ] = cond_jump;
```

```
        instructions_[JNZ] = cond_jump;
        instructions_[JGZ] = cond_jump;
        instructions_[JGEZ] = cond_jump;
        instructions_[JLZ] = cond_jump;
        instructions_[JLEZ] = cond_jump;

        instructions_[LOAD] = std::make_shared<LoadInstructionParser>();
        instructions_[STORE] = std::make_shared<StoreInstructionParser>();

        auto stack_instruction = std::make_shared<StackInstructionParser>();
        instructions_[PUSH] = stack_instruction;
        instructions_[POP] = stack_instruction;

        auto alu_instruction = std::make_shared<AluInstructionParser>();
        instructions_[ADD] = alu_instruction;
        instructions_[SUB] = alu_instruction;
        instructions_[MUL] = alu_instruction;
        instructions_[DIV] = alu_instruction;
        instructions_[MOD] = alu_instruction;
        instructions_[AND] = alu_instruction;
        instructions_[OR] = alu_instruction;
        instructions_[XOR] = alu_instruction;
        instructions_[ASL] = alu_instruction;
        instructions_[ASR] = alu_instruction;

        instructions_[NOT] = std::make_shared<NotInstructionParser>();
    }
```

Member Function Documentation

std::shared_ptr< Token > bnssassembler::InstructionLineParser::parse (const std::string & *line*, size_t *line_number*, std::string *initial_line*) const`[override]`, `[protected]`, `[virtual]`

Parses one line of the file. Does not call the next parser in chain.

Parameters:

line	Line to parse
line_number	Number of the line that is parsed
initial_line	Initial line that is parsed

Returns:

Extracted token from line or nullptr if the parser failed parsing the line

Exceptions:

Throws	if the parser failed and identified the error

Implements **bnssassembler::LineParser** (*p.257*).

Definition at line 101 of file InstructionLineParser.cpp.

References bnssassembler::DEFAULT, instructions_, bnssassembler::loadStoreFixup(), and bnssassembler::InstructionCodeParser::parse().

```
{
        std::regex regex("[[:space:]]*([A-Za-z]*)(.*)[[:space:]]*");
        if (!regex_match(line, regex)) {
            return nullptr;
        }

        auto instruction_code_string = regex_replace(line, regex, "$1");
        auto operands_string = regex_replace(line, regex, "$2");

        auto type = DEFAULT;
        loadStoreFixup(instruction_code_string, type);

        InstructionCode instruction_code;

        try {
            instruction_code = InstructionCodeParser::parse(instruction_code_string);
        }
        catch (MessageException&) {
            return nullptr;
        }

        auto instruction_parser = instructions_.at(instruction_code);
        auto vector_of_operands = instruction_parser->parse(operands_string);
        return std::make_shared<InstructionToken>(line_number, initial_line, instruction_code, vector_of_operands, type);
    }
```

Member Data Documentation

std::unordered_map<InstructionCode, std::shared_ptr<InstructionParser> > bnssassembler::InstructionLineParser::instructions_`[private]`

Definition at line 23 of file InstructionLineParser.h.

Referenced by InstructionLineParser(), and parse().

The documentation for this class was generated from the following files:

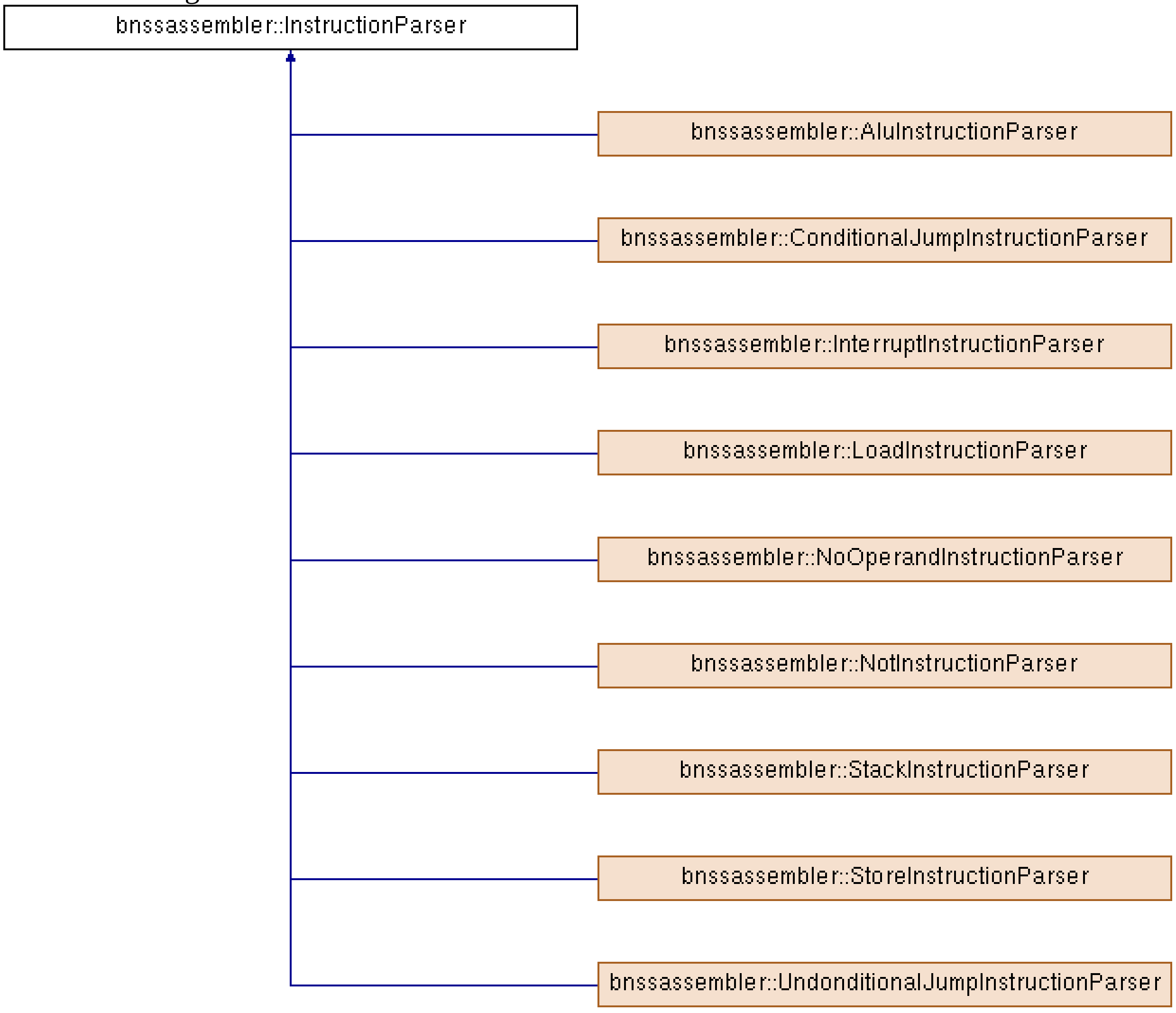
- Code/Assembler/Include/**InstructionLineParser.h**
- Code/Assembler/Source/**InstructionLineParser.cpp**

bnssassembler::InstructionParser Class Reference

Abstract lass used for parsing one instruction.

`#include <InstructionParser.h>`

Inheritance diagram for bnssassembler::InstructionParser:


bnssassembler::InstructionParser
bnssassembler::AluInstructionParser
bnssassembler::ConditionalJumpInstructionParser
bnssassembler::InterruptInstructionParser
bnssassembler::LoadInstructionParser
bnssassembler::NoOperandInstructionParser
bnssassembler::NotInstructionParser
bnssassembler::StackInstructionParser
bnssassembler::StoreInstructionParser
bnssassembler::UndonditionalJumpInstructionParser

Public Member Functions

- std::vector< std::shared_ptr< **Operand** > > **parse** (std::string str) const
  Parses the instruction.
- virtual **~InstructionParser** ()=0
  Abstract destructor so the class is abstract.

Protected Attributes

- std::vector< std::shared_ptr< **OperandParser** > > **operands_**
  The chains of operand parsers for all operands.

Detailed Description

Abstract lass used for parsing one instruction.

Definition at line 12 of file InstructionParser.h.

Constructor & Destructor Documentation

bnssassembler::InstructionParser::~InstructionParser () `[pure virtual]`

Abstract destructor so the class is abstract.

Definition at line 44 of file InstructionParser.cpp.

```
   44 {}
```

Member Function Documentation

std::vector< std::shared_ptr< Operand > > bnssassembler::InstructionParser::parse (std::string *str*) const

Parses the instruction.

Parameters:

str	String representing the instruction

Returns:

Vector of operands in the instruction

Exceptions:

Throws	if it fails parsing

Definition at line 8 of file InstructionParser.cpp.

References bnssassembler::COMMA_TOKENIZER_REGEX, bnssassembler::LAST_COMMA_TOKEN_REGEX, and operands_.

```
    8 
{
    9         std::vector<std::shared_ptr<Operand>> operands;
   10 
   11         for (size_t i = 0; i < operands_.size() - 1 && operands_.size() != 0; 
i++) {
   12             if (!regex_match(str, COMMA_TOKENIZER_REGEX)) {
   13                 throw MessageException("Invalid instruction format: " + str);
   14             }
   15 
   16             auto operand_string = regex_replace(str, COMMA_TOKENIZER_REGEX, 
"$1");
   17             str = regex_replace(str, COMMA_TOKENIZER_REGEX, "$2");
   18 
   19             auto operand = operands_[i]->tryParse(operand_string);
   20             if (operand == nullptr) {
   21                 throw MessageException("Invalid operand: " + operand_string);
   22             }
   23 
   24             operands.push_back(operand);
   25         }
   26 
   27         if (operands_.size() > 0) {
   28             if (!regex_match(str, LAST_COMMA_TOKEN_REGEX)) {
   29                 throw MessageException("Invalid instruction format: " + str);
   30             }
   31 
   32             auto operand_string = regex_replace(str, LAST_COMMA_TOKEN_REGEX, 
"$1");
   33             auto operand = operands_[operands_.size() - 
1]->tryParse(operand_string);
   34             if (operand == nullptr) {
   35                 throw MessageException("Invalid operand: " + operand_string);
   36             }
```

```

            operands.push_back(operand);
        }

        return operands;
    }
```

Member Data Documentation

std::vector<std::shared_ptr<OperandParser> > bnssassembler::InstructionParser::operands_ [protected]

The chains of operand parsers for all operands.

Definition at line 30 of file InstructionParser.h.

Referenced by bnssassembler::AluInstructionParser::AluInstructionParser(), bnssassembler::ConditionalJumpInstructionParser::ConditionalJumpInstructionParser(), bnssassembler::InterruptInstructionParser::InterruptInstructionParser(), bnssassembler::LoadInstructionParser::LoadInstructionParser(), bnssassembler::NotInstructionParser::NotInstructionParser(), parse(), bnssassembler::StackInstructionParser::StackInstructionParser(), bnssassembler::StoreInstructionParser::StoreInstructionParser(), and bnssassembler::UndonditionalJumpInstructionParser::UndonditionalJumpInstructionParser().

The documentation for this class was generated from the following files:

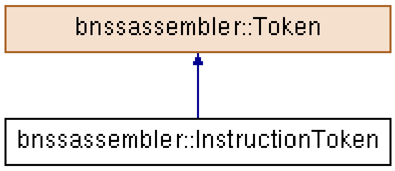
- Code/Assembler/Include/**InstructionParser.h**
- Code/Assembler/Source/**InstructionParser.cpp**

bnssassembler::InstructionToken Class Reference

Class representing the instruction in an assembler source file.

`#include <InstructionToken.h>`

Inheritance diagram for bnssassembler::InstructionToken:



Public Member Functions

- **InstructionToken** (size_t line_number, std::string **line**, **InstructionCode** code, std::vector< std::shared_ptr< **Operand** >> operands, **OperandType** type=**DEFAULT**) noexcept
  *Constructs an **InstructionToken** object.*
- void **resolveSymbolDefinitions** (std::unordered_set< **SymbolDefinition** > symbols) noexcept override
  Resolves symbol definitions in a token.
- void **firstPass** (**FirstPassData** &data) const override
  Executes the first pass over the token.
- void **secondPass** (**SecondPassData** &data) const override
  Executes the second pass over the token.
- void **resolveSymbolTable** (const **SymbolTable** &symbol_table) noexcept override
  Resolves the symbols from the symbol table and updates relocation info.
- void **resolveImports** (std::unordered_set< std::string > imported_symbols) noexcept override
  Resolves the imported symbols and updates relocation info.
- void **resolveCurrentPcSymbol** (size_t section_index, size_t offset) noexcept override
  Resolves the current PC symbol and sets the relocation info.

Private Member Functions

- size_t **length** () const
  Gets the length of the instruction in bytes.
- std::pair< **uint32_t**, std::pair< **uint32_t**, std::list< **RelocationRecord** > > > **packInstruction** () const

Private Attributes

- **InstructionCode code_**
- **OperandType type_**
- std::vector< std::shared_ptr< **Operand** > > **operands_**

Detailed Description

Class representing the instruction in an assembler source file.

Definition at line 16 of file InstructionToken.h.

Constructor & Destructor Documentation

bnssassembler::InstructionToken::InstructionToken (size_t *line_number*, std::string *line*, InstructionCode *code*, std::vector< std::shared_ptr< Operand >> *operands*, OperandType *type* = DEFAULT)[noexcept]

Constructs an **InstructionToken** object.

Parameters:

line_number	Number of the line where the instruction is
line	Line where the instruction is
code	Instruction code
operands	Vector of operands of the instruction
type	Type of the operand

Definition at line 9 of file InstructionToken.cpp.

```
: Token(line_number, line), code_(code), type_(type), operands_(operands) {}
```

Member Function Documentation

void bnssassembler::InstructionToken::firstPass (FirstPassData & *data*) const[override], [virtual]

Executes the first pass over the token.

Parameters:

data	**Data** that the token will modify

Implements **bnssassembler::Token** (*p.510*).

Definition at line 17 of file InstructionToken.cpp.

References bnssassembler::FirstPassData::incLocationCounter(), and length().

```
                                                        {
        data.incLocationCounter(length());
    }
```

size_t bnssassembler::InstructionToken::length () const[private]

Gets the length of the instruction in bytes.

Returns:

Length of the instruction in bytes

Definition at line 52 of file InstructionToken.cpp.

References bnssassembler::IMMEDIATE, bnssassembler::MEMORY_DIRECT, operands_, and bnssassembler::REGISTER_INDIRECT_OFFSET.

Referenced by firstPass(), and secondPass().

```
                                    {
        for (auto &operand : operands_) {
            if (
                operand->addressMode() == IMMEDIATE ||
                operand->addressMode() == MEMORY_DIRECT ||
                operand->addressMode() == REGISTER_INDIRECT_OFFSET)
                return 8;
        }
        return 4;
    }
```

std::pair< uint32_t, std::pair< uint32_t, std::list< RelocationRecord > > > bnssassembler::InstructionToken::packInstruction () const`[private]`

Definition at line 64 of file InstructionToken.cpp.

References bnssassembler::InstructionBitField::address_mode, bnssassembler::InstructionBitFieldUnion::bit_field, code_, bnssassembler::InstructionBitFieldUnion::data, bnssassembler::NONE, operands_, bnssassembler::InstructionBitField::operation_code, bnssassembler::InstructionBitField::register0, bnssassembler::InstructionBitField::register1, bnssassembler::InstructionBitField::register2, bnssassembler::REGISTER_DIRECT, bnssassembler::InstructionBitField::type, and type_.

Referenced by secondPass().

```
{
        std::pair<uint32_t, std::pair<uint32_t, std::list<RelocationRecord>>> ret;
        InstructionBitFieldUnion instruction;

        instruction.bit_field.operation_code = code_;
        instruction.bit_field.address_mode = REGISTER_DIRECT; // Default address mode
        instruction.bit_field.register0 = NONE;
        instruction.bit_field.register1 = NONE;
        instruction.bit_field.register2 = NONE;
        instruction.bit_field.type = type_;

        for (auto &operand : operands_) {
            operand->packToInstruction(instruction, ret.second.first, ret.second.second);
        }

        // ReSharper disable once CppSomeObjectMembersMightNotBeInitialized
        ret.first = instruction.data;
        return ret;
    }
```

void bnssassembler::InstructionToken::resolveCurrentPcSymbol (size_t *section_index*, size_t *offset*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the current PC symbol and sets the relocation info.

Parameters:

section_index	Current PC section
offset	PC address in relation to the current section beginning

Reimplemented from **bnssassembler::Token** (*p.510*).

Definition at line 46 of file InstructionToken.cpp.

References operands_.

```
{
        for (auto &operand: operands_) {
            operand->resolveCurrentPcSymbol(section_index, offset);
        }
    }
```

void bnssassembler::InstructionToken::resolveImports (std::unordered_set< std::string > *imported_symbols*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the imported symbols and updates relocation info.

Parameters:

imported_symbols	Collection of imported symbols

Reimplemented from **bnssassembler::Token** (*p.511*).

Definition at line 40 of file InstructionToken.cpp.

References operands_.

```
{
        for (auto &operand : operands_) {
            operand->resolveImports(imported_symbols);
        }
    }
```

void bnssassembler::InstructionToken::resolveSymbolDefinitions (std::unordered_set< SymbolDefinition > *symbols*)`[override]`, `[virtual]`, `[noexcept]`

Resolves symbol definitions in a token.

Parameters:

symbols	Vector od symbol definitions that should be resolved

Reimplemented from **bnssassembler::Token** (*p.511*).

Definition at line 11 of file InstructionToken.cpp.

```
{
        for (auto &operand : operands_) {
            operand->resolveSymbols(symbols);
        }
    }
```

void bnssassembler::InstructionToken::resolveSymbolTable (const SymbolTable & *symbol_table*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the symbols from the symbol table and updates relocation info.

Parameters:

symbol_table	**Symbol** table

Reimplemented from **bnssassembler::Token** (*p.511*).

Definition at line 34 of file InstructionToken.cpp.

References operands_.

```
{
        for (auto &operand : operands_) {
            operand->resolveSymbolTable(symbol_table);
        }
    }
```

void bnssassembler::InstructionToken::secondPass (SecondPassData & *data*) const`[override]`, `[virtual]`

Executes the second pass over the token.

Parameters:

data	**Data** that the token will modify

Implements **bnssassembler::Token** (*p.512*).

Definition at line 21 of file InstructionToken.cpp.

References bnssassembler::SecondPassData::addData(), bnssassembler::SecondPassData::currentSectionType(), length(), packInstruction(), and bnssassembler::TEXT.

```
                                                              {
        if (data.currentSectionType() != TEXT) {
            throw MessageException("Instructions can only exist in text 
sections");
        }

        auto pair = packInstruction();
        data.addData(pair.first, std::list<RelocationRecord>());

        if (length() == 8) {
            data.addData(pair.second.first, pair.second.second);
        }
    }
```

Member Data Documentation

InstructionCode bnssassembler::InstructionToken::code_ `[private]`

Definition at line 35 of file InstructionToken.h.

Referenced by packInstruction().

std::vector<std::shared_ptr<Operand> > bnssassembler::InstructionToken::operands_ `[private]`

Definition at line 37 of file InstructionToken.h.

Referenced by length(), packInstruction(), resolveCurrentPcSymbol(), resolveImports(), and resolveSymbolTable().

OperandType bnssassembler::InstructionToken::type_ `[private]`

Definition at line 36 of file InstructionToken.h.

Referenced by packInstruction().

The documentation for this class was generated from the following files:

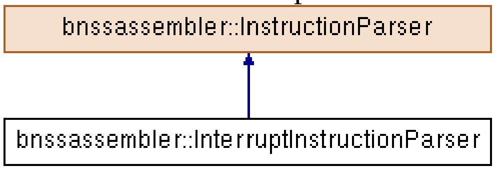
- Code/Assembler/Include/**InstructionToken.h**
- Code/Assembler/Source/**InstructionToken.cpp**

bnssassembler::InterruptInstructionParser Class Reference

Class representing the parser for the interrupt instruction.

`#include <InterruptInstructionParser.h>`

Inheritance diagram for bnssassembler::InterruptInstructionParser:


bnssassembler::InstructionParser
bnssassembler::InterruptInstructionParser

Public Member Functions

- **InterruptInstructionParser** () noexcept
  Constructs an ***InterruptInstructionParser*** *object.*

Additional Inherited Members

Detailed Description

Class representing the parser for the interrupt instruction.

Definition at line 10 of file InterruptInstructionParser.h.

Constructor & Destructor Documentation

bnssassembler::InterruptInstructionParser::InterruptInstructionParser () [noexcept]

Constructs an **InterruptInstructionParser** object.

Definition at line 6 of file InterruptInstructionParser.cpp.

References bnssassembler::InstructionParser::operands_.

```
                                                                                {
        operands_.push_back(std::make_shared<RegisterDirectParser>());
    }
```

The documentation for this class was generated from the following files:

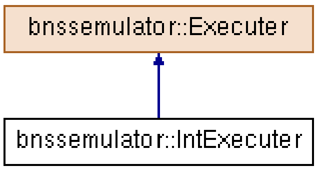
- Code/Assembler/Include/**InterruptInstructionParser.h**
- Code/Assembler/Source/**InterruptInstructionParser.cpp**

bnssemulator::IntExecuter Class Reference

Class representing the executer for the int instruction.

`#include <IntExecuter.h>`

Inheritance diagram for bnssemulator::IntExecuter:



Public Member Functions

- void **execute** (**InstructionBitField** instruction, **Context** &context) const override
  Executes the instruction.

Detailed Description

Class representing the executer for the int instruction.

Definition at line 10 of file IntExecuter.h.

Member Function Documentation

void bnssemulator::IntExecuter::execute (InstructionBitField *instruction*, Context & *context*) const`[override]`, `[virtual]`

Executes the instruction.

Parameters:

instruction	Instruction
context	**Processor** context

Implements **bnssemulator::Executer** (*p.163*).

Definition at line 5 of file IntExecuter.cpp.

References bnssemulator::Context::finishProgram(), bnssemulator::Context::getRegister(), bnssemulator::Context::jumpToInterrupt(), and bnssemulator::InstructionBitField::register0.

```
{
        auto &entry = context.getRegister(instruction.register0);

        if (entry == 0) {
            context.finishProgram();
            return;
        }

        context.jumpToInterrupt(entry);
    }
```

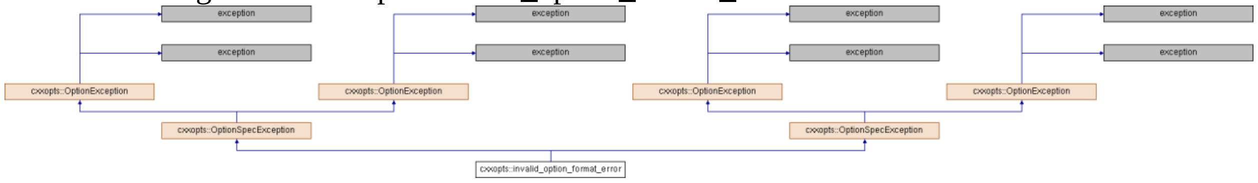
The documentation for this class was generated from the following files:

- Code/Emulator/Include/**IntExecuter.h**
- Code/Emulator/Source/**IntExecuter.cpp**

cxxopts::invalid_option_format_error Class Reference

`#include <cxxopts.h>`

Inheritance diagram for cxxopts::invalid_option_format_error:


exception
exception
cxxopts::OptionException
cxxopts::OptionSpecException
cxxopts::invalid_option_format_error

Public Member Functions

- **invalid_option_format_error** (const std::string &format)
- **invalid_option_format_error** (const std::string &format)

Detailed Description

Definition at line 322 of file cxxopts.h.

Constructor & Destructor Documentation

cxxopts::invalid_option_format_error::invalid_option_format_error (const std::string & *format*) `[inline]`

Definition at line 325 of file cxxopts.h.

```
326             : OptionSpecException("Invalid option format '" + format + "'")
327         {
328         }
```

cxxopts::invalid_option_format_error::invalid_option_format_error (const std::string & *format*) `[inline]`

Definition at line 325 of file cxxopts.h.

```
326             : OptionSpecException("Invalid option format '" + format + "'")
327         {
328         }
```

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

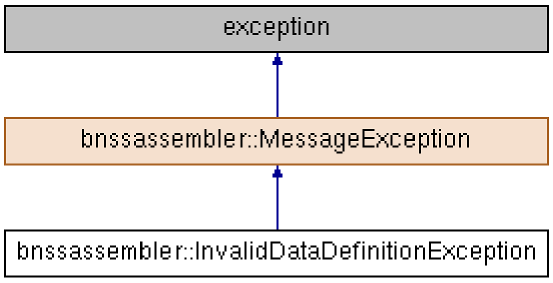
- Code/Assembler/Include/**cxxopts.h**

bnssassembler::InvalidDataDefinitionException Class Reference

Exception representing invalid data definition.

`#include <InvalidDataDefinitionException.h>`

Inheritance diagram for bnssassembler::InvalidDataDefinitionException:


exception
bnssassembler::MessageException
bnssassembler::InvalidDataDefinitionException

Public Member Functions

- **InvalidDataDefinitionException** (std::string data) noexcept
  *Constructs an **InvalidDataDefinitionException** object.*

Detailed Description

Exception representing invalid data definition.

Definition at line 10 of file InvalidDataDefinitionException.h.

Constructor & Destructor Documentation

bnssassembler::InvalidDataDefinitionException::InvalidDataDefinitionException (std::string *data*) `[explicit]`, `[noexcept]`

Constructs an **InvalidDataDefinitionException** object.

Parameters:

data	String containing the invalid data

Definition at line 5 of file InvalidDataDefinitionException.cpp.

```
    5 : MessageException(data + " can not be parsed as data") {}
```

The documentation for this class was generated from the following files:

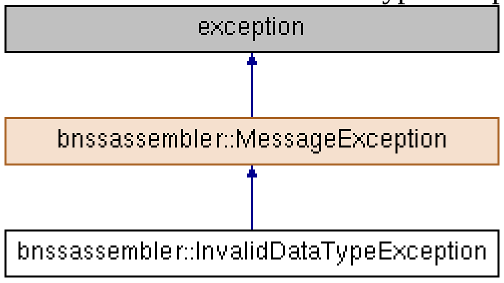
- Code/Assembler/Include/**InvalidDataDefinitionException.h**
- Code/Assembler/Source/**InvalidDataDefinitionException.cpp**

bnssassembler::InvalidDataTypeException Class Reference

Exception representing the invalid data type.

`#include <InvalidDataTypeException.h>`

Inheritance diagram for bnssassembler::InvalidDataTypeException:


exception
bnssassembler::MessageException
bnssassembler::InvalidDataTypeException

Public Member Functions

- **InvalidDataTypeException** (std::string data_type) noexcept
  *Constructs an **InvalidDataTypeException**.*

Detailed Description

Exception representing the invalid data type.

Definition at line 10 of file InvalidDataTypeException.h.

Constructor & Destructor Documentation

bnssassembler::InvalidDataTypeException::InvalidDataTypeException (std::string *data_type*) `[explicit]`, `[noexcept]`

Constructs an **InvalidDataTypeException**.

Parameters:

data_type	String containing the invalid DataType

Definition at line 5 of file InvalidDataTypeException.cpp.

```
    5 : MessageException(data_type + " is not a valid data type") {}
```

The documentation for this class was generated from the following files:

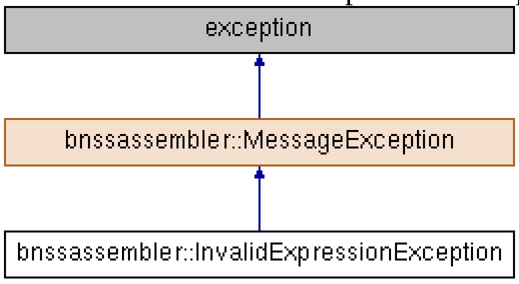
- Code/Assembler/Include/**InvalidDataTypeException.h**
- Code/Assembler/Source/**InvalidDataTypeException.cpp**

bnssassembler::InvalidExpressionException Class Reference

Exception representing the invalid expression.

`#include <InvalidExpressionException.h>`

Inheritance diagram for bnssassembler::InvalidExpressionException:


exception
bnssassembler::MessageException
bnssassembler::InvalidExpressionException

Public Member Functions

- **InvalidExpressionException** () noexcept
  *Constructs an **InvalidExpressionException** object.*

Detailed Description

Exception representing the invalid expression.

Definition at line 10 of file InvalidExpressionException.h.

Constructor & Destructor Documentation

bnssassembler::InvalidExpressionException::InvalidExpressionException () [noexcept]

Constructs an **InvalidExpressionException** object.

Definition at line 5 of file InvalidExpressionException.cpp.

```
    5 : MessageException("The expression is invalid") {}
```

The documentation for this class was generated from the following files:

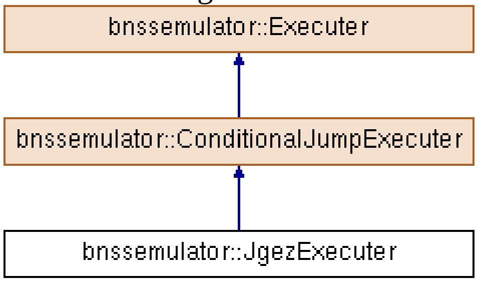
- Code/Assembler/Include/**InvalidExpressionException.h**
- Code/Assembler/Source/**InvalidExpressionException.cpp**

bnssemulator::JgezExecuter Class Reference

Class representing the executer for the jgez instruction.

`#include <JgezExecuter.h>`

Inheritance diagram for bnssemulator::JgezExecuter:


bnssemulator::Executer
bnssemulator::ConditionalJumpExecuter
bnssemulator::JgezExecuter

Protected Member Functions

- bool **test** (bool negative, bool zero, bool overflow, bool carry) const noexcept override
  Tests whether the jump should happen.

Additional Inherited Members

Detailed Description

Class representing the executer for the jgez instruction.

Definition at line 10 of file JgezExecuter.h.

Member Function Documentation

bool bnssemulator::JgezExecuter::test (bool *negative*, bool *zero*, bool *overflow*, bool *carry*) const`[override]`, `[protected]`, `[virtual]`, `[noexcept]`

Tests whether the jump should happen.

Parameters:

negative	Negative flag of the register
zero	Zero flag of the register
overflow	Overflow flag of the register
carry	Carry flag of the register

Returns:

Whether the jump should happen

Implements **bnssemulator::ConditionalJumpExecuter** (*p.127*).

Definition at line 5 of file JgezExecuter.cpp.

```
{
            return negative == overflow;
        }
```

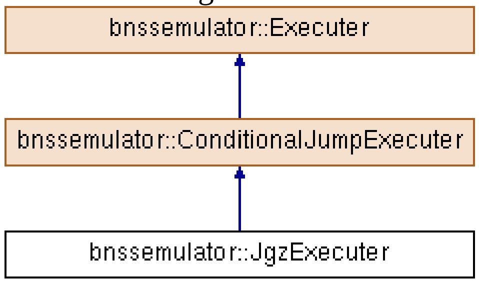
The documentation for this class was generated from the following files:

- Code/Emulator/Include/**JgezExecuter.h**

- Code/Emulator/Source/**JgezExecuter.cpp**

bnssemulator::JgzExecuter Class Reference

Class representing the executer for the jgz instruction.
`#include <JgzExecuter.h>`
Inheritance diagram for bnssemulator::JgzExecuter:


bnssemulator::Executer
bnssemulator::ConditionalJumpExecuter
bnssemulator::JgzExecuter

Protected Member Functions

- bool **test** (bool negative, bool zero, bool overflow, bool carry) const noexcept override
  Tests whether the jump should happen.

Additional Inherited Members

Detailed Description

Class representing the executer for the jgz instruction.

Definition at line 10 of file JgzExecuter.h.

Member Function Documentation

bool bnssemulator::JgzExecuter::test (bool *negative*, bool *zero*, bool *overflow*, bool *carry*) const`[override]`, `[protected]`, `[virtual]`, `[noexcept]`

Tests whether the jump should happen.

Parameters:

negative	Negative flag of the register
zero	Zero flag of the register
overflow	Overflow flag of the register
carry	Carry flag of the register

Returns:

Whether the jump should happen

Implements **bnssemulator::ConditionalJumpExecuter** (*p.127*).

Definition at line 5 of file JgzExecuter.cpp.

```
{
            return (negative == overflow) && !zero;
        }
```

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**JgzExecuter.h**

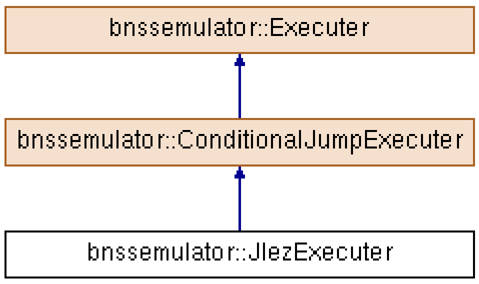
- Code/Emulator/Source/**JgzExecuter.cpp**

bnssemulator::JlezExecuter Class Reference

Class representing the executer for the jlez instruction.

`#include <JlezExecuter.h>`

Inheritance diagram for bnssemulator::JlezExecuter:


bnssemulator::Executer
bnssemulator::ConditionalJumpExecuter
bnssemulator::JlezExecuter

Protected Member Functions

- bool **test** (bool negative, bool zero, bool overflow, bool carry) const noexcept override
  Tests whether the jump should happen.

Additional Inherited Members

Detailed Description

Class representing the executer for the jlez instruction.

Definition at line 10 of file JlezExecuter.h.

Member Function Documentation

bool bnssemulator::JlezExecuter::test (bool *negative*, bool *zero*, bool *overflow*, bool *carry*) const `[override]`, `[protected]`, `[virtual]`, `[noexcept]`

Tests whether the jump should happen.

Parameters:

negative	Negative flag of the register
zero	Zero flag of the register
overflow	Overflow flag of the register
carry	Carry flag of the register

Returns:

Whether the jump should happen

Implements **bnssemulator::ConditionalJumpExecuter** (*p.127*).

Definition at line 5 of file JlezExecuter.cpp.

```
{
        return negative != overflow;
    }
```

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**JlezExecuter.h**

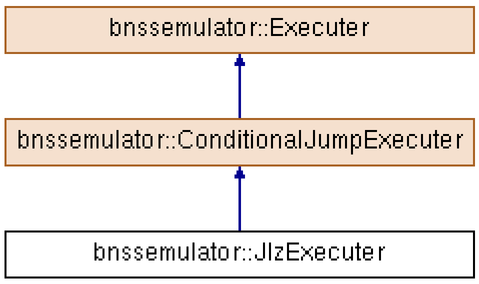
- Code/Emulator/Source/**JlezExecuter.cpp**

bnssemulator::JlzExecuter Class Reference

Class representing the executer for the jlz instruction.

`#include <JlzExecuter.h>`

Inheritance diagram for bnssemulator::JlzExecuter:


bnssemulator::Executer
bnssemulator::ConditionalJumpExecuter
bnssemulator::JlzExecuter

Protected Member Functions

- bool **test** (bool negative, bool zero, bool overflow, bool carry) const noexcept override
  Tests whether the jump should happen.

Additional Inherited Members

Detailed Description

Class representing the executer for the jlz instruction.

Definition at line 10 of file JlzExecuter.h.

Member Function Documentation

bool bnssemulator::JlzExecuter::test (bool *negative*, bool *zero*, bool *overflow*, bool *carry*) const`[override]`, `[protected]`, `[virtual]`, `[noexcept]`

Tests whether the jump should happen.

Parameters:

negative	Negative flag of the register
zero	Zero flag of the register
overflow	Overflow flag of the register
carry	Carry flag of the register

Returns:

Whether the jump should happen

Implements **bnssemulator::ConditionalJumpExecuter** (*p.127*).

Definition at line 5 of file JlzExecuter.cpp.

```
{
        return (negative != overflow) && !zero;
    }
```

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**JlzExecuter.h**

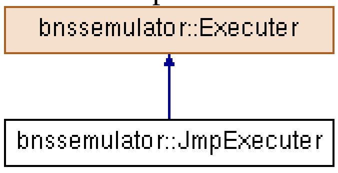
- Code/Emulator/Source/**JlzExecuter.cpp**

bnssemulator::JmpExecuter Class Reference

Class representing the executer for the jmp instruction.

`#include <JmpExecuter.h>`

Inheritance diagram for bnssemulator::JmpExecuter:



Public Member Functions

- void **execute** (**InstructionBitField** instruction, **Context** &context) const override
  Executes the instruction.

Detailed Description

Class representing the executer for the jmp instruction.

Definition at line 10 of file JmpExecuter.h.

Member Function Documentation

void bnssemulator::JmpExecuter::execute (InstructionBitField *instruction*, Context & *context*) const `[override]`, `[virtual]`

Executes the instruction.

Parameters:

instruction	Instruction
context	**Processor** context

Implements **bnssemulator::Executer** (*p.163*).

Definition at line 5 of file JmpExecuter.cpp.

References bnssemulator::Context::getOperandAddress(), and bnssemulator::Context::jumpTo().

```
{
        auto address = context.getOperandAddress(instruction, 0);
        context.jumpTo(address);
    }
```

The documentation for this class was generated from the following files:

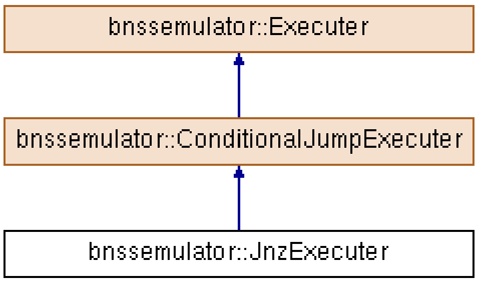
- Code/Emulator/Include/**JmpExecuter.h**
- Code/Emulator/Source/**JmpExecuter.cpp**

bnssemulator::JnzExecuter Class Reference

Class representing the executer for the jnz instruction.

`#include <JnzExecuter.h>`

Inheritance diagram for bnssemulator::JnzExecuter:


bnssemulator::Executer
bnssemulator::ConditionalJumpExecuter
bnssemulator::JnzExecuter

Protected Member Functions

- bool **test** (bool negative, bool zero, bool overflow, bool carry) const noexcept override
  Tests whether the jump should happen.

Additional Inherited Members

Detailed Description

Class representing the executer for the jnz instruction.

Definition at line 10 of file JnzExecuter.h.

Member Function Documentation

bool bnssemulator::JnzExecuter::test (bool *negative*, bool *zero*, bool *overflow*, bool *carry*) const`[override]`, `[protected]`, `[virtual]`, `[noexcept]`

Tests whether the jump should happen.

Parameters:

negative	Negative flag of the register
zero	Zero flag of the register
overflow	Overflow flag of the register
carry	Carry flag of the register

Returns:

Whether the jump should happen

Implements **bnssemulator::ConditionalJumpExecuter** (*p.127*).

Definition at line 5 of file JnzExecuter.cpp.

```
{
        return !zero;
    }
```

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**JnzExecuter.h**

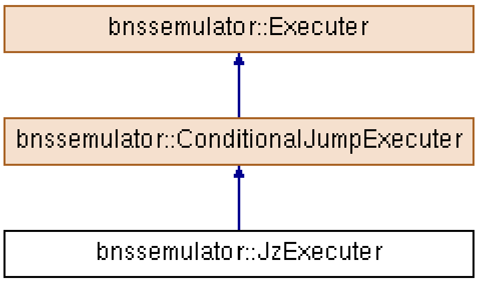
- Code/Emulator/Source/**JnzExecuter.cpp**

bnssemulator::JzExecuter Class Reference

Class representing the executer for the jz instruction.

`#include <JzExecuter.h>`

Inheritance diagram for bnssemulator::JzExecuter:



Protected Member Functions

- bool **test** (bool negative, bool zero, bool overflow, bool carry) const noexcept override
  Tests whether the jump should happen.

Additional Inherited Members

Detailed Description

Class representing the executer for the jz instruction.

Definition at line 10 of file JzExecuter.h.

Member Function Documentation

bool bnssemulator::JzExecuter::test (bool *negative*, bool *zero*, bool *overflow*, bool *carry*) const`[override]`, `[protected]`, `[virtual]`, `[noexcept]`

Tests whether the jump should happen.

Parameters:

negative	Negative flag of the register
zero	Zero flag of the register
overflow	Overflow flag of the register
carry	Carry flag of the register

Returns:

Whether the jump should happen

Implements **bnssemulator::ConditionalJumpExecuter** (*p.127*).

Definition at line 5 of file JzExecuter.cpp.

```
{
        return zero;
    }
```

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**JzExecuter.h**

- Code/Emulator/Source/**JzExecuter.cpp**

bnssemulator::KeyboardListener Class Reference

Class representing the keyboard listener thread.
`#include <KeyboardListener.h>`

Static Public Member Functions

- static void **listen** (**Context** *context)
  Listens to keyboard interrupts and sets the context flag every time they fire.

Detailed Description

Class representing the keyboard listener thread.

Definition at line 10 of file KeyboardListener.h.

Member Function Documentation

void bnssemulator::KeyboardListener::listen (Context * *context*) `[static]`

Listens to keyboard interrupts and sets the context flag every time they fire.

Definition at line 6 of file KeyboardListener.cpp.

References consoleio::getCharacter(), consoleio::keyboardHit(), bnssemulator::Context::pressCharacter(), and bnssemulator::Context::programFinished().

Referenced by bnssemulator::Processor::executeProgram().

```
    6                                                 {
    7         while (!context->programFinished()) {
    8             while (!consoleio::keyboardHit()) {
    9                 if (context->programFinished()) {
   10                     return;
   11                 }
   12             }
   13 
   14             auto character = consoleio::getCharacter();
   15             context->pressCharacter(character);
   16         }
   17     }
```

The documentation for this class was generated from the following files:

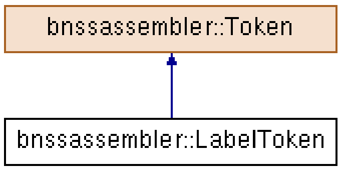
- Code/Emulator/Include/**KeyboardListener.h**
- Code/Emulator/Source/**KeyboardListener.cpp**

bnssassembler::LabelToken Class Reference

Class representing the label token.

`#include <LabelToken.h>`

Inheritance diagram for bnssassembler::LabelToken:



Public Member Functions

- **LabelToken** (std::string label, size_t line_number, std::string **line**) noexcept
  *Constructs a **LabelToken** object.*
- void **firstPass** (**FirstPassData** &data) const override
  Executes the first pass over the token.
- void **secondPass** (**SecondPassData** &data) const override
  Executes the second pass over the token.

Private Attributes

- std::string **label_**

Detailed Description

Class representing the label token.

Definition at line 10 of file LabelToken.h.

Constructor & Destructor Documentation

bnssassembler::LabelToken::LabelToken (std::string *label*, size_t *line_number*, std::string *line*) [explicit], [noexcept]

Constructs a **LabelToken** object.

Parameters:

label	Label
line_number	Number of the line where the label is
line	Line where the label is

Definition at line 5 of file LabelToken.cpp.

```
    5 : Token(line_number, line), label_(label) {}
```

Member Function Documentation

void bnssassembler::LabelToken::firstPass (FirstPassData & *data*) const [override], [virtual]

Executes the first pass over the token.

Parameters:

data	**Data** that the token will modify

Implements **bnssassembler::Token** (*p.510*).

Definition at line 7 of file LabelToken.cpp.

References bnssassembler::FirstPassData::insertSymbol(), and label_.

```
                                                    {
        data.insertSymbol(label_);
    }
```

void bnssassembler::LabelToken::secondPass (SecondPassData & *data*) const[override], [virtual]

Executes the second pass over the token.

Parameters:

data	**Data** that the token will modify

Implements **bnssassembler::Token** (*p.512*).

Definition at line 11 of file LabelToken.cpp.

```
                                                     {
        // Do nothing
    }
```

Member Data Documentation

std::string bnssassembler::LabelToken::label_[private]

Definition at line 23 of file LabelToken.h.

Referenced by firstPass().

The documentation for this class was generated from the following files:

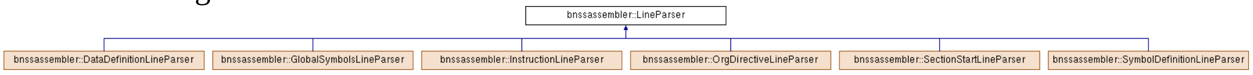
- Code/Assembler/Include/**LabelToken.h**
- Code/Assembler/Source/**LabelToken.cpp**

bnssassembler::LineParser Class Reference

Chain of command abstract class used for parsing one line of file.

`#include <LineParser.h>`

Inheritance diagram for bnssassembler::LineParser:


bnssassembler::LineParser
bnssassembler::DataDefinitionLineParser
bnssassembler::GlobalSymbolsLineParser
bnssassembler::InstructionLineParser
bnssassembler::OrgDirectiveLineParser
bnssassembler::SectionStartLineParser
bnssassembler::SymbolDefinitionLineParser

Public Member Functions

- std::shared_ptr< **Token** > **tryParse** (const std::string &line, size_t line_number, std::string initial_line) const
  Tries to parse one line of the file. Calls the next parser in chain if it fails.
- virtual **~LineParser** ()=default
- void **next** (std::shared_ptr< **LineParser** > next) noexcept
  Sets the next parser in the chain of parsers.

Protected Member Functions

- virtual std::shared_ptr< **Token** > **parse** (const std::string &line, size_t line_number, std::string initial_line) const =0
  Parses one line of the file. Does not call the next parser in chain.

Private Attributes

- std::shared_ptr< **LineParser** > **next_** = nullptr
  The next parser in the chain.

Detailed Description

Chain of command abstract class used for parsing one line of file.

Definition at line 13 of file LineParser.h.

Constructor & Destructor Documentation

virtual bnssassembler::LineParser::~LineParser () `[virtual]`, `[default]`

Member Function Documentation

void bnssassembler::LineParser::next (std::shared_ptr< LineParser > *next*) `[noexcept]`

Sets the next parser in the chain of parsers.

Parameters:

next	The next parser

Definition at line 18 of file LineParser.cpp.

References next_.

```
18                                                                        {
        next_ = next;
    }
```

virtual std::shared_ptr<Token> bnssassembler::LineParser::parse (const std::string & *line*, size_t *line_number*, std::string *initial_line*) const`[protected]`, `[pure virtual]`

Parses one line of the file. Does not call the next parser in chain.

Parameters:

line	Line to parse
line_number	Number of the line that is parsed
initial_line	Initial line that is parsed

Returns:

Extracted token from line or nullptr if the parser failed parsing the line

Exceptions:

Throws	if the parser failed and identified the error

Implemented in **bnssassembler::InstructionLineParser** (*p.223*), **bnssassembler::DataDefinitionLineParser** (*p.145*), **bnssassembler::GlobalSymbolsLineParser** (*p.192*), **bnssassembler::OrgDirectiveLineParser** (*p.348*), **bnssassembler::SectionStartLineParser** (*p.430*), and **bnssassembler::SymbolDefinitionLineParser** (*p.496*).

Referenced by tryParse().

std::shared_ptr< Token > bnssassembler::LineParser::tryParse (const std::string & *line*, size_t *line_number*, std::string *initial_line*) const

Tries to parse one line of the file. Calls the next parser in chain if it fails.

Parameters:

line	Line to parse
line_number	Number of the line that is parsed
initial_line	Initial line that is parsed

Returns:

Extracted token from line or nullptr if the whole chain failed parsing

Exceptions:

Throws	if the chain failed and the parser identified the error

Definition at line 5 of file LineParser.cpp.

References next_, and parse().

```
{
        auto ret = parse(line, line_number, initial_line);
        if (ret != nullptr) {
            return ret;
        }

        if (next_ == nullptr) {
            return nullptr;
        }

        return next_->tryParse(line, line_number, initial_line);
    }
```

Member Data Documentation

std::shared_ptr<LineParser> bnssassembler::LineParser::next_ = nullptr `[private]`

The next parser in the chain.

Definition at line 46 of file LineParser.h.

Referenced by next(), and tryParse().

The documentation for this class was generated from the following files:

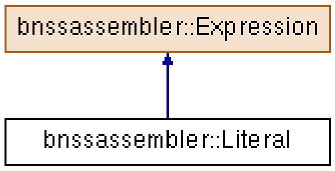
- Code/Assembler/Include/**LineParser.h**
- Code/Assembler/Source/**LineParser.cpp**

bnssassembler::Literal Class Reference

Class representing the literal value.

`#include <Literal.h>`

Inheritance diagram for bnssassembler::Literal:



Public Member Functions

- **Literal** (int32_t **value**) noexcept
  *Constructs a **Literal** object.*
- int32_t **value** () const override
  Evaluates the expression.

Private Attributes

- int32_t **value_**

Detailed Description

Class representing the literal value.

Definition at line 11 of file Literal.h.

Constructor & Destructor Documentation

bnssassembler::Literal::Literal (int32_t *value*)`[explicit]`, `[noexcept]`

Constructs a **Literal** object.

Parameters:

value	Value of the

Definition at line 5 of file Literal.cpp.

```
    5 : value_(value) {}
```

Member Function Documentation

int32_t bnssassembler::Literal::value () const`[override]`, `[virtual]`

Evaluates the expression.

Exceptions:

Throws	if the expression has variables or could not be evaluated (for example, division by zero)

Implements **bnssassembler::Expression** (*p.167*).

Definition at line 7 of file Literal.cpp.

References value_.

```
    7                                {
    8          return value_;
    9      }
```

Member Data Documentation

int32_t bnssassembler::Literal::value_ `[private]`

Definition at line 20 of file Literal.h.

Referenced by value().

The documentation for this class was generated from the following files:

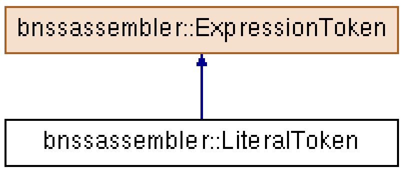
- Code/Assembler/Include/**Literal.h**
- Code/Assembler/Source/**Literal.cpp**

bnssassembler::LiteralToken Class Reference

Token class representing a math literal value.

`#include <LiteralToken.h>`

Inheritance diagram for bnssassembler::LiteralToken:



Public Member Functions

- **LiteralToken** (std::string value)
- int **inputPriority** () const noexcept override
  Gets the input priority of the token.
- int **stackPriority** () const noexcept override
  Gets the stack priority of the token.
- int **rank** () const noexcept override
  Gets the rank of the token.
- void **process** (std::list< std::shared_ptr< **ExpressionToken** >> &output, std::stack< std::shared_ptr< **ExpressionToken** >> &stack, int &expression_rank) const override
  Processes the current token.
- std::shared_ptr< **Expression** > **create** () const override
  Creates an expression object out of the token.

Protected Member Functions

- std::shared_ptr< **ExpressionToken** > **clone** (std::string param) const override
  Clones the current object, using the string provided.

Private Attributes

- int32_t **value_**

Detailed Description

Token class representing a math literal value.

Definition at line 10 of file LiteralToken.h.

Constructor & Destructor Documentation

bnssassembler::LiteralToken::LiteralToken (std::string *value*) `[explicit]`

Definition at line 8 of file LiteralToken.cpp.

References cxxopts::value(), and value_.

```
                                          {
        value_ = StringHelper::parseNumber<int32_t>(value);
    }
```

Member Function Documentation

std::shared_ptr< ExpressionToken > bnssassembler::LiteralToken::clone (std::string *param*) const `[override]`, `[protected]`, `[virtual]`

Clones the current object, using the string provided.

Parameters:

param	String that will be used to construct the new object

Returns:

Pointer to the cloned object

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 21 of file LiteralToken.cpp.

```
   21
{
   22         return std::make_shared<LiteralToken>(param);
   23     }
```

std::shared_ptr< Expression > bnssassembler::LiteralToken::create () const `[override]`, `[virtual]`

Creates an expression object out of the token.

Returns:

Pointer to the expression

Exceptions:

Throws	if the token has no corresponding expression object

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 17 of file LiteralToken.cpp.

References value_.

```
   17                                                       {
   18         return std::make_shared<Literal>(value_);
   19     }
```

int bnssassembler::LiteralToken::inputPriority () const `[override]`, `[virtual]`, `[noexcept]`

Gets the input priority of the token.

Returns:

Input priority of the token

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 25 of file LiteralToken.cpp.

```
   25                                                  {
   26         return INT_MAX;
   27     }
```

void bnssassembler::LiteralToken::process (std::list< std::shared_ptr< ExpressionToken >> & *output*, std::stack< std::shared_ptr< ExpressionToken >> & *stack*, int & *expression_rank*) const `[override]`, `[virtual]`

Processes the current token.

Parameters:

output	Output list of tokens
stack	Helper stack of tokens
expression_rank	Rank of the expression

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 12 of file LiteralToken.cpp.

References rank().

```
{
        output.push_back(std::make_shared<LiteralToken>(*this));
        expression_rank += rank();
    }
```

int bnssassembler::LiteralToken::rank () const `[override]`, `[virtual]`, `[noexcept]`

Gets the rank of the token.

Returns:

Rank of the token

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 33 of file LiteralToken.cpp.

Referenced by process().

```
                                         {
        return 1;
    }
```

int bnssassembler::LiteralToken::stackPriority () const `[override]`, `[virtual]`, `[noexcept]`

Gets the stack priority of the token.

Returns:

Stack priority of the token

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 29 of file LiteralToken.cpp.

```
                                                     {
        return INT_MAX;
    }
```

Member Data Documentation

int32_t bnssassembler::LiteralToken::value_ `[private]`

Definition at line 23 of file LiteralToken.h.

Referenced by create(), and LiteralToken().

The documentation for this class was generated from the following files:

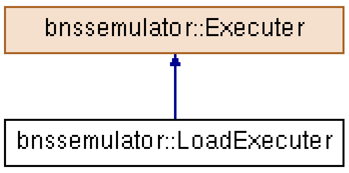
- Code/Assembler/Include/**LiteralToken.h**
- Code/Assembler/Source/**LiteralToken.cpp**

bnssemulator::LoadExecuter Class Reference

Class representing the executer for the load instruction.

`#include <LoadExecuter.h>`

Inheritance diagram for bnssemulator::LoadExecuter:



Public Member Functions

- void **execute** (**InstructionBitField** instruction, **Context** &context) const override
  Executes the instruction.

Detailed Description

Class representing the executer for the load instruction.

Definition at line 10 of file LoadExecuter.h.

Member Function Documentation

void bnssemulator::LoadExecuter::execute (InstructionBitField *instruction*, Context & *context*) const `[override]`, `[virtual]`

Executes the instruction.

Parameters:

instruction	Instruction
context	**Processor** context

Implements **bnssemulator::Executer** (*p.163*).

Definition at line 42 of file LoadExecuter.cpp.

References bnssemulator::fill(), bnssemulator::Context::getOperand(), bnssemulator::Context::getRegister(), bnssemulator::InstructionBitField::register0, bnssemulator::REGULAR_DOUBLE_WORD, bnssemulator::SIGNED_WORD, bnssemulator::InstructionBitField::type, bnssemulator::UNSIGNED_WORD, and bnssemulator::Register::value().

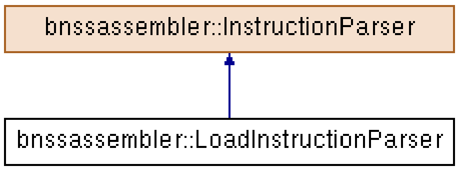
```
{
        auto num_of_bytes = instruction.type == REGULAR_DOUBLE_WORD ? 4 :
instruction.type == UNSIGNED_WORD || instruction.type == SIGNED_WORD ? 2 : 1;
        auto operand = context.getOperand(instruction, 1, num_of_bytes);
        auto &reg = context.getRegister(instruction.register0);
        reg.value(fill(static_cast<OperandType>(instruction.type), operand));
    }
```

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**LoadExecuter.h**
- Code/Emulator/Source/**LoadExecuter.cpp**

bnssassembler::LoadInstructionParser Class Reference

Class representing the load instruction parser.
`#include <LoadInstructionParser.h>`
Inheritance diagram for bnssassembler::LoadInstructionParser:


bnssassembler::InstructionParser
bnssassembler::LoadInstructionParser

Public Member Functions

- **LoadInstructionParser** () noexcept
  *Constructs a **LoadInstructionParser** object.*

Additional Inherited Members

Detailed Description

Class representing the load instruction parser.

Definition at line 10 of file LoadInstructionParser.h.

Constructor & Destructor Documentation

bnssassembler::LoadInstructionParser::LoadInstructionParser () [noexcept]

Constructs a **LoadInstructionParser** object.

Definition at line 10 of file LoadInstructionParser.cpp.

References bnssassembler::InstructionParser::operands_.

```
                                                  {
        operands_.push_back(std::make_shared<RegisterDirectParser>());

        auto immed = std::make_shared<ImmediateParser>();
        auto regdir = std::make_shared<RegisterDirectParser>();
        auto memdir = std::make_shared<MemoryDirectParser>();
        auto regindpom = std::make_shared<RegisterIndirectOffsetParser>();
        auto regind = std::make_shared<RegisterIndirectParser>();

        immed->next(regdir);
        regdir->next(memdir);
        memdir->next(regindpom);
        regindpom->next(regind);

        operands_.push_back(immed);
    }
```

The documentation for this class was generated from the following files:

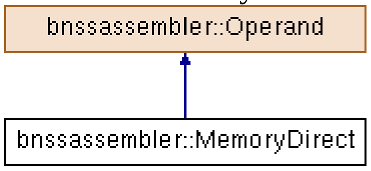
- Code/Assembler/Include/**LoadInstructionParser.h**
- Code/Assembler/Source/**LoadInstructionParser.cpp**

bnssassembler::MemoryDirect Class Reference

Class representing the memory direct operand.

`#include <MemoryDirect.h>`

Inheritance diagram for bnssassembler::MemoryDirect:



Public Member Functions

- **MemoryDirect** (**MicroRiscExpression** address) noexcept
  *Constructs a **MemoryDirect** object.*
- void **packToInstruction** (**InstructionBitFieldUnion** &instruction, **uint32_t** &second_word, std::list< **RelocationRecord** > &relocations) const override
  Packs the operand into the instruction.
- void **resolveSymbols** (std::unordered_set< **SymbolDefinition** > symbols) noexcept override
  Resolves the defined symbols in the expressions.
- void **resolveSymbolTable** (const **SymbolTable** &symbol_table) noexcept override
  Resolves the symbols from the symbol table and updates the relocation info.
- void **resolveImports** (std::unordered_set< std::string > imported_symbols) noexcept override
  Resolves the imported symbols and updates the relocation info.
- void **resolveCurrentPcSymbol** (size_t section_index, size_t offset) noexcept override
  Resolves the current PC symbol and sets the relocation info.
- **AddressMode addressMode** () const noexcept override
  Gets the address mode of the operand.

Private Attributes

- **MicroRiscExpression address_**

Detailed Description

Class representing the memory direct operand.

Definition at line 11 of file MemoryDirect.h.

Constructor & Destructor Documentation

bnssassembler::MemoryDirect::MemoryDirect (MicroRiscExpression *address*) [explicit], [noexcept]

Constructs a **MemoryDirect** object.

Parameters:

address	Address of the memory direct operand

Definition at line 5 of file MemoryDirect.cpp.

```
    5 : address_(address) {}
```

Member Function Documentation

AddressMode bnssassembler::MemoryDirect::addressMode () const`[override]`, `[virtual]`, `[noexcept]`

Gets the address mode of the operand.

Returns:

Address mode of the operand

Implements **bnssassembler::Operand** (*p.302*).

Definition at line 31 of file MemoryDirect.cpp.

References bnssassembler::MEMORY_DIRECT.

```
                                                        {
        return MEMORY_DIRECT;
    }
```

void bnssassembler::MemoryDirect::packToInstruction (InstructionBitFieldUnion & *instruction*, uint32_t & *second_word*, std::list< RelocationRecord > & *relocations*) const`[override]`, `[virtual]`

Packs the operand into the instruction.

Parameters:

instruction	Reference to the first word of the instruction containing the instruction info
second_word	Reference to the second word of the instruction containing the address/value/displacement
relocations	Reference to the list of relocation records

Implements **bnssassembler::Operand** (*p.303*).

Definition at line 7 of file MemoryDirect.cpp.

References address_, bnssassembler::InstructionBitField::address_mode, bnssassembler::InstructionBitFieldUnion::bit_field, bnssassembler::MicroRiscExpression::generateRelocations(), bnssassembler::MEMORY_DIRECT, and bnssassembler::MicroRiscExpression::value().

```
{
        instruction.bit_field.address_mode = MEMORY_DIRECT;
        second_word = address_.value();
        relocations.splice(relocations.end(),
address_.generateRelocations());
    }
```

void bnssassembler::MemoryDirect::resolveCurrentPcSymbol (size_t *section_index*, size_t *offset*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the current PC symbol and sets the relocation info.

Parameters:

section_index	Current PC section
offset	PC address in relation to the current section beginning

Reimplemented from **bnssassembler::Operand** (*p.303*).

Definition at line 27 of file MemoryDirect.cpp.

References address_, and bnssassembler::MicroRiscExpression::resolveCurrentPcSymbol().

```
   27
{
   28         address_.resolveCurrentPcSymbol(section_index, offset);
   29     }
```

void bnssassembler::MemoryDirect::resolveImports (std::unordered_set< std::string > *imported_symbols*) `[override]`, `[virtual]`, `[noexcept]`

Resolves the imported symbols and updates the relocation info.

Parameters:

imported_symbols	Collection of imported symbols

Reimplemented from **bnssassembler::Operand** (*p.303*).

Definition at line 23 of file MemoryDirect.cpp.

References address_, and bnssassembler::MicroRiscExpression::resolveImports().

```
   23
{
   24         address_.resolveImports(imported_symbols);
   25     }
```

void bnssassembler::MemoryDirect::resolveSymbols (std::unordered_set< SymbolDefinition > *symbols*) `[override]`, `[virtual]`, `[noexcept]`

Resolves the defined symbols in the expressions.

Parameters:

symbols	Collection of symbol definitions

Reimplemented from **bnssassembler::Operand** (*p.303*).

Definition at line 13 of file MemoryDirect.cpp.

References address_, and bnssassembler::MicroRiscExpression::setValue().

```
   13
{
   14         for (auto &symbol : symbols) {
   15             address_.setValue(symbol.name(), symbol.expression());
   16         }
   17     }
```

void bnssassembler::MemoryDirect::resolveSymbolTable (const SymbolTable & *symbol_table*) `[override]`, `[virtual]`, `[noexcept]`

Resolves the symbols from the symbol table and updates the relocation info.

Parameters:

symbol_table	**Symbol** table

Reimplemented from **bnssassembler::Operand** (*p.304*).

Definition at line 19 of file MemoryDirect.cpp.

References address_, and bnssassembler::MicroRiscExpression::resolveSymbolTable().

```
   19
{
   20         address_.resolveSymbolTable(symbol_table);
   21     }
```

Member Data Documentation

MicroRiscExpression bnssassembler::MemoryDirect::address_ `[private]`

Definition at line 26 of file MemoryDirect.h.

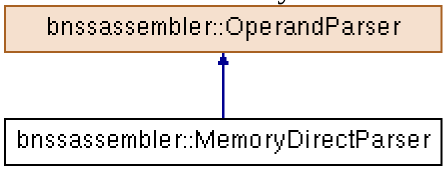
Referenced by packToInstruction(), resolveCurrentPcSymbol(), resolveImports(), resolveSymbols(), and resolveSymbolTable().

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**MemoryDirect.h**
- Code/Assembler/Source/**MemoryDirect.cpp**

bnssassembler::MemoryDirectParser Class Reference

Class representing the parser for the memory direct operand.
`#include <MemoryDirectParser.h>`
Inheritance diagram for bnssassembler::MemoryDirectParser:


bnssassembler::OperandParser
bnssassembler::MemoryDirectParser

Protected Member Functions

- std::shared_ptr< **Operand** > **parse** (std::string str) const override
  Parses one operand. Does not call the next parser if it fails.

Additional Inherited Members

Detailed Description

Class representing the parser for the memory direct operand.

Definition at line 10 of file MemoryDirectParser.h.

Member Function Documentation

std::shared_ptr< Operand > bnssassembler::MemoryDirectParser::parse (std::string *str*) const `[override]`, `[protected]`, `[virtual]`

Parses one operand. Does not call the next parser if it fails.

Parameters:

str	**Operand** which should be parsed

Returns:

Pointer to the operand or nullptr, if the parser failed parsing

Exceptions:

Throws	if the parser fails but identifies the error

Implements **bnssassembler::OperandParser** (*p.306*).

Definition at line 9 of file MemoryDirectParser.cpp.

References bnssassembler::ExpressionBuilder::build(), and bnssassembler::CONSTANT_TERM_REGEX.

```
                                                                        {
        if (!regex_match(str, CONSTANT_TERM_REGEX)) {
            return nullptr;
        }

        try {
            auto expression = ExpressionBuilder::build(str);
            return std::make_shared<MemoryDirect>(expression);
        }
        catch (...) {
            return nullptr;
        }
```

```
}
```

The documentation for this class was generated from the following files:

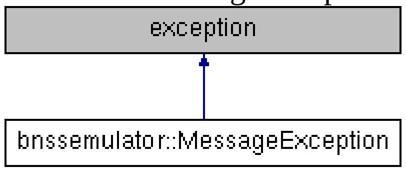
- Code/Assembler/Include/**MemoryDirectParser.h**
- Code/Assembler/Source/**MemoryDirectParser.cpp**

bnssemulator::MessageException Class Reference

Represents an exception with a string message.

```
#include <MessageException.h>
```

Inheritance diagram for bnssemulator::MessageException:



Public Member Functions

- **MessageException** (std::string **message**) noexcept
  *Constructs a **MessageException** object with the message.*
- std::string **message** () const noexcept
  Gets the message of the exception.
- const char * **what** () const noexcept override

Private Attributes

- std::string **message_**

Detailed Description

Represents an exception with a string message.

Definition at line 11 of file MessageException.h.

Constructor & Destructor Documentation

bnssemulator::MessageException::MessageException (std::string *message*) [explicit], [noexcept]

Constructs a **MessageException** object with the message.

Parameters:

message	Message

Definition at line 5 of file MessageException.cpp.

```
    5 : message_(message) {}
```

Member Function Documentation

std::string bnssemulator::MessageException::message () const [noexcept]

Gets the message of the exception.

Definition at line 7 of file MessageException.cpp.

References message_.

```
    7                                                        {
        return message_;
    }
```

const char * bnssemulator::MessageException::what () const`[override]`, `[noexcept]`

Definition at line 11 of file MessageException.cpp.

References message_.

```
                                                     {
        return message_.c_str();
    }
```

Member Data Documentation

std::string bnssemulator::MessageException::message_`[private]`

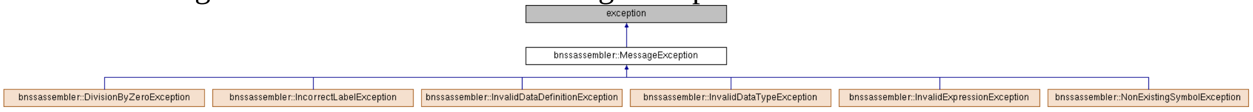
Definition at line 25 of file MessageException.h.

Referenced by message(), and what().

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**MessageException.h**
- Code/Emulator/Source/**MessageException.cpp**

bnssassembler::MessageException Class Reference

Represents an exception with a string message.
`#include <MessageException.h>`
Inheritance diagram for bnssassembler::MessageException:

exception
bnssassembler::MessageException
bnssassembler::DivisionByZeroException
bnssassembler::IncorrectLabelException
bnssassembler::InvalidDataDefinitionException
bnssassembler::InvalidDataTypeException
bnssassembler::InvalidExpressionException
bnssassembler::NonExistingSymbolException

Public Member Functions

- **MessageException** (std::string **message**) noexcept
  *Constructs a **MessageException** object with the message.*
- std::string **message** () const noexcept
  Gets the message of the exception.
- const char * **what** () const noexcept override

Private Attributes

- std::string **message_**

Detailed Description

Represents an exception with a string message.

Definition at line 11 of file MessageException.h.

Constructor & Destructor Documentation

bnssassembler::MessageException::MessageException (std::string *message*) `[explicit]`, `[noexcept]`

Constructs a **MessageException** object with the message.

Parameters:

message	Message

Definition at line 5 of file MessageException.cpp.

```
    5 : message_(message) {}
```

Member Function Documentation

std::string bnssassembler::MessageException::message () const `[noexcept]`

Gets the message of the exception.

Definition at line 7 of file MessageException.cpp.

References message_.

Referenced by bnssassembler::FirstPass::execute(), bnssassembler::SecondPass::execute(), and bnssassembler::Parser::parse().

```
    7                                                                 {
        return message_;
    }
```

const char * bnssassembler::MessageException::what () const `[override]`, `[noexcept]`

Definition at line 11 of file MessageException.cpp.

References message_.

```
                                                        {
        return message_.c_str();
    }
```

Member Data Documentation

std::string bnssassembler::MessageException::message_ `[private]`

Definition at line 25 of file MessageException.h.

Referenced by message(), and what().

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**MessageException.h**
- Code/Assembler/Source/**MessageException.cpp**

bnssassembler::MicroRiscExpression Class Reference

Adapter class for **Expression**.
`#include <MicroRiscExpression.h>`

Public Member Functions

- **MicroRiscExpression** (std::shared_ptr< **Expression** > expression) noexcept
  *Constructs a **MicroRiscExpression** object.*
- int32_t **value** () const
  Get the value of the expression.
- bool **setValue** (std::string **name**, **MicroRiscExpression** expression) const noexcept
  Sets the value for the symbol.
- void **resolveSymbolTable** (const **SymbolTable** &symbol_table) const noexcept
  Resolves the symbols from the symbol table and sets the relocation info.
- void **resolveImports** (std::unordered_set< std::string > imported_symbols) const noexcept
  Resolves the imported symbols and sets the relocation info.
- void **resolveCurrentPcSymbol** (size_t section_index, size_t offset) const noexcept
  Resolves the current PC symbol and sets the relocation info.
- std::list< **RelocationRecord** > **generateRelocations** () const
  Validates the tree and generates the relocation records for the expression.

Private Attributes

- std::shared_ptr< **Expression** > **expression_**

Detailed Description

Adapter class for **Expression**.

Definition at line 11 of file MicroRiscExpression.h.

Constructor & Destructor Documentation

bnssassembler::MicroRiscExpression::MicroRiscExpression (std::shared_ptr< Expression > *expression*) `[explicit]`, `[noexcept]`

Constructs a **MicroRiscExpression** object.

Parameters:

expression	Pointer to **Expression** object that this object will adapt

Definition at line 6 of file MicroRiscExpression.cpp.

```
    6 : expression_(expression) {}
```

Member Function Documentation

std::list< RelocationRecord > bnssassembler::MicroRiscExpression::generateRelocations () const

Validates the tree and generates the relocation records for the expression.

Returns:

Collection of relocation records

Definition at line 28 of file MicroRiscExpression.cpp.

References expression_.

Referenced by bnssassembler::Immediate::packToInstruction(), bnssassembler::MemoryDirect::packToInstruction(), bnssassembler::RegisterIndirectOffset::packToInstruction(), and bnssassembler::OrgDirectiveToken::secondPass().

```
   28 
{
   29         expression_->validate();
   30         auto ret = expression_->generateRelocations();
   31         for (auto &element : ret) {
   32             if (element.opposite()) {
   33                 throw MessageException((element.section() ? "Symbols from " + 
std::to_string(element.sectionIndex()) + "th section are " : "Symbol " + 
element.symbolName() + " is ") + "subtracted more times than added");
   34             }
   35         }
   36 
   37         return ret;
   38     }
```

void bnssassembler::MicroRiscExpression::resolveCurrentPcSymbol (size_t *section_index*, size_t *offset*) const `[noexcept]`

Resolves the current PC symbol and sets the relocation info.

Parameters:

section_index	Current PC section
offset	PC address in relation to the current section beginning

Definition at line 24 of file MicroRiscExpression.cpp.

References expression_.

Referenced by bnssassembler::Immediate::resolveCurrentPcSymbol(), bnssassembler::MemoryDirect::resolveCurrentPcSymbol(), and bnssassembler::RegisterIndirectOffset::resolveCurrentPcSymbol().

```
   24 
{
   25         expression_->resolveCurrentPcSymbol(section_index, offset);
   26     }
```

void bnssassembler::MicroRiscExpression::resolveImports (std::unordered_set< std::string > *imported_symbols*) const `[noexcept]`

Resolves the imported symbols and sets the relocation info.

Parameters:

imported_symbols	Collection of imported symbols

Definition at line 20 of file MicroRiscExpression.cpp.

References expression_.

Referenced by bnssassembler::Immediate::resolveImports(), bnssassembler::MemoryDirect::resolveImports(), bnssassembler::OrgDirectiveToken::resolveImports(), and bnssassembler::RegisterIndirectOffset::resolveImports().

```
{
        expression_->resolveImports(imported_symbols);
    }
```

void bnssassembler::MicroRiscExpression::resolveSymbolTable (const SymbolTable & *symbol_table*) const`[noexcept]`

Resolves the symbols from the symbol table and sets the relocation info.

Parameters:

symbol_table	**Symbol** table

Definition at line 16 of file MicroRiscExpression.cpp.

References expression_.

Referenced by bnssassembler::Immediate::resolveSymbolTable(), bnssassembler::MemoryDirect::resolveSymbolTable(), bnssassembler::OrgDirectiveToken::resolveSymbolTable(), and bnssassembler::RegisterIndirectOffset::resolveSymbolTable().

```
{
        expression_->resolveSymbolTable(symbol_table);
    }
```

bool bnssassembler::MicroRiscExpression::setValue (std::string *name*, MicroRiscExpression *expression*) const`[noexcept]`

Sets the value for the symbol.

Parameters:

name	Name of the symbol
expression	**Expression** of the symbol

Definition at line 12 of file MicroRiscExpression.cpp.

References expression_, and bnssassembler::name().

Referenced by bnssassembler::Immediate::resolveSymbols(), bnssassembler::MemoryDirect::resolveSymbols(), and bnssassembler::RegisterIndirectOffset::resolveSymbols().

```
{
        return expression_->setValue(name, expression.expression_);
    }
```

int32_t bnssassembler::MicroRiscExpression::value () const

Get the value of the expression.

Returns:

Value of the expression

Exceptions:

Throws	if the value can not be calculated

Definition at line 8 of file MicroRiscExpression.cpp.

References expression_.

Referenced by bnssassembler::Immediate::packToInstruction(), bnssassembler::MemoryDirect::packToInstruction(), bnssassembler::RegisterIndirectOffset::packToInstruction(), and bnssassembler::OrgDirectiveToken::secondPass().

```
                                        {
        return expression_->value();
    }
```

Member Data Documentation

std::shared_ptr<Expression> bnssassembler::MicroRiscExpression::expression_ `[private]`

Definition at line 58 of file MicroRiscExpression.h.

Referenced by generateRelocations(), resolveCurrentPcSymbol(), resolveImports(), resolveSymbolTable(), setValue(), and value().

The documentation for this class was generated from the following files:

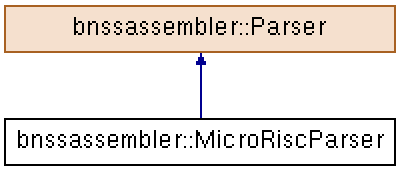
- Code/Assembler/Include/**MicroRiscExpression.h**
- Code/Assembler/Source/**MicroRiscExpression.cpp**

bnssassembler::MicroRiscParser Class Reference

Class representing the parser for the MicroRISC assembly.

`#include <MicroRiscParser.h>`

Inheritance diagram for bnssassembler::MicroRiscParser:



Static Public Member Functions

- static **MicroRiscParser** & **instance** () noexcept
  Static method for getting the singleton object.

Protected Member Functions

- std::vector< std::string > **oneLineCommentDelimiters** () const noexcept override
  Returns all strings that start the comment to the end of the line.
- std::vector< std::string > **labelDelimiters** () const noexcept override
  Returns all strings that end the label at the start of the line.
- bool **isEnd** (std::string line) const noexcept override
  Checks if the parser should stop parsing the file.
- std::shared_ptr< **LineParser** > **chain** () const noexcept override
  *Returns the first **LineParser** in chain.*

Private Member Functions

- **MicroRiscParser** ()
- **MicroRiscParser** (**MicroRiscParser** &)=delete
- void **operator=** (**MicroRiscParser** &)=delete

Private Attributes

- std::shared_ptr< **LineParser** > **head_**

Additional Inherited Members

Detailed Description

Class representing the parser for the MicroRISC assembly.

Definition at line 10 of file MicroRiscParser.h.

Constructor & Destructor Documentation

bnssassembler::MicroRiscParser::MicroRiscParser () `[private]`

Definition at line 34 of file MicroRiscParser.cpp.

References head_.

```
                                {
        auto instructions = std::make_shared<InstructionLineParser>();
        auto data = std::make_shared<DataDefinitionLineParser>();
        auto sections = std::make_shared<SectionStartLineParser>();
        auto global = std::make_shared<GlobalSymbolsLineParser>();
        auto org = std::make_shared<OrgDirectiveLineParser>();
        auto symbol = std::make_shared<SymbolDefinitionLineParser>();

        instructions->next(data);
        data->next(sections);
        sections->next(global);
        global->next(org);
        org->next(symbol);

        head_ = instructions;
    }
```

bnssassembler::MicroRiscParser::MicroRiscParser (MicroRiscParser &) `[private]`, `[delete]`

Member Function Documentation

std::shared_ptr< LineParser > bnssassembler::MicroRiscParser::chain () const `[override]`, `[protected]`, `[virtual]`, `[noexcept]`

Returns the first **LineParser** in chain.

Returns:

Pointer to the first parser

Implements **bnssassembler::Parser** (*p.353*).

Definition at line 30 of file MicroRiscParser.cpp.

References head_.

```
30                                                                   {
31        return head_;
32    }
```

MicroRiscParser & bnssassembler::MicroRiscParser::instance () `[static]`, `[noexcept]`

Static method for getting the singleton object.

Returns:

Reference to the singleton **MicroRiscParser** object

Definition at line 12 of file MicroRiscParser.cpp.

Referenced by main().

```
12                                                       {
13        static MicroRiscParser instance;
14        return instance;
15    }
```

bool bnssassembler::MicroRiscParser::isEnd (std::string *line*) const `[override]`, `[protected]`, `[virtual]`, `[noexcept]`

Checks if the parser should stop parsing the file.

Parameters:

line	Line to check

Implements **bnssassembler::Parser** (*p.354*).

Definition at line 25 of file MicroRiscParser.cpp.

```
                                                                 {
        static std::regex regex("[[:space:]]*[.][Ee][Nn][Dd].*[[:space:]]*");
        return regex_match(line, regex);
    }
```

std::vector< std::string > bnssassembler::MicroRiscParser::labelDelimiters () const[override], [protected], [virtual], [noexcept]

Returns all strings that end the label at the start of the line.

Returns:

Vector of such strings

Implements **bnssassembler::Parser** (*p.354*).

Definition at line 21 of file MicroRiscParser.cpp.

```
                                                                                {
        return { ":" };
    }
```

std::vector< std::string > bnssassembler::MicroRiscParser::oneLineCommentDelimiters () const[override], [protected], [virtual], [noexcept]

Returns all strings that start the comment to the end of the line.

Returns:

Vector of such strings

Implements **bnssassembler::Parser** (*p.354*).

Definition at line 17 of file MicroRiscParser.cpp.

```
{
        return { ";", "//" };
    }
```

void bnssassembler::MicroRiscParser::operator= (MicroRiscParser &)[private], [delete]

Member Data Documentation

std::shared_ptr<LineParser> bnssassembler::MicroRiscParser::head_[private]

Definition at line 23 of file MicroRiscParser.h.

Referenced by chain(), and MicroRiscParser().

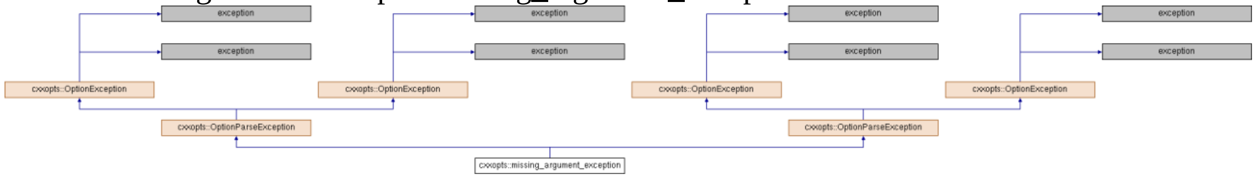
The documentation for this class was generated from the following files:

- Code/Assembler/Include/**MicroRiscParser.h**
- Code/Assembler/Source/**MicroRiscParser.cpp**

cxxopts::missing_argument_exception Class Reference

`#include <cxxopts.h>`

Inheritance diagram for cxxopts::missing_argument_exception:


exception
exception
cxxopts::OptionException
cxxopts::OptionParseException
cxxopts::missing_argument_exception

Public Member Functions

- **missing_argument_exception** (const std::string &option)
- **missing_argument_exception** (const std::string &option)

Detailed Description

Definition at line 340 of file cxxopts.h.

Constructor & Destructor Documentation

cxxopts::missing_argument_exception::missing_argument_exception (const std::string & *option*) `[inline]`

Definition at line 343 of file cxxopts.h.

```
            : OptionParseException("Option '" + option + "' is missing an argument")
        {
        }
```

cxxopts::missing_argument_exception::missing_argument_exception (const std::string & *option*) `[inline]`

Definition at line 343 of file cxxopts.h.

```
            : OptionParseException("Option '" + option + "' is missing an argument")
        {
        }
```

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

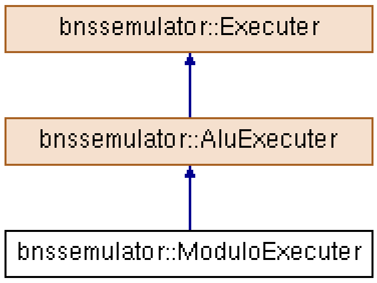
- Code/Assembler/Include/**cxxopts.h**

bnssemulator::ModuloExecuter Class Reference

Class representing the executer for the modulo instruction.

`#include <ModuloExecuter.h>`

Inheritance diagram for bnssemulator::ModuloExecuter:



Protected Member Functions

- void **execute** (**Register** &dst, const **Register** &lhs, const **Register** &rhs) const override
  Executes the ALU instruction.

Additional Inherited Members

Detailed Description

Class representing the executer for the modulo instruction.

Definition at line 10 of file ModuloExecuter.h.

Member Function Documentation

void bnssemulator::ModuloExecuter::execute (Register & *dst*, const Register & *lhs*, const Register & *rhs*) const `[override]`, `[protected]`, `[virtual]`

Executes the ALU instruction.

Parameters:

dst	Reference to the destination register
lhs	Left operand register
rhs	Right operand register

Implements **bnssemulator::AluExecuter** (*p.102*).

Definition at line 6 of file ModuloExecuter.cpp.

```
{
        if (rhs == 0) {
            throw MessageException("Modulo by zero");
        }

        dst = lhs % rhs;
    }
```

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**ModuloExecuter.h**

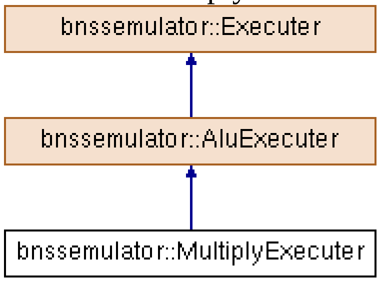
- Code/Emulator/Source/**ModuloExecuter.cpp**

bnssemulator::MultiplyExecuter Class Reference

Class representing the executer for the multiply instruction.

`#include <MultiplyExecuter.h>`

Inheritance diagram for bnssemulator::MultiplyExecuter:



Protected Member Functions

- void **execute** (**Register** &dst, const **Register** &lhs, const **Register** &rhs) const override
  Executes the ALU instruction.

Additional Inherited Members

Detailed Description

Class representing the executer for the multiply instruction.

Definition at line 10 of file MultiplyExecuter.h.

Member Function Documentation

void bnssemulator::MultiplyExecuter::execute (Register & *dst*, const Register & *lhs*, const Register & *rhs*) const `[override]`, `[protected]`, `[virtual]`

Executes the ALU instruction.

Parameters:

dst	Reference to the destination register
lhs	Left operand register
rhs	Right operand register

Implements **bnssemulator::AluExecuter** (*p.102*).

Definition at line 5 of file MultiplyExecuter.cpp.

```
{
        dst = lhs * rhs;
    }
```

The documentation for this class was generated from the following files:

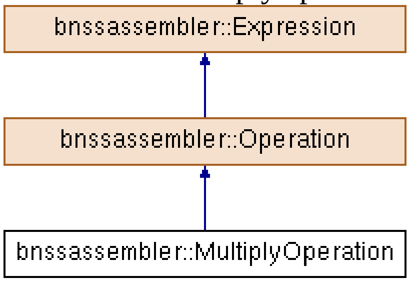
- Code/Emulator/Include/**MultiplyExecuter.h**
- Code/Emulator/Source/**MultiplyExecuter.cpp**

bnssassembler::MultiplyOperation Class Reference

Class implementing the behaviour of the * operator in expressions.

`#include <MultiplyOperation.h>`

Inheritance diagram for bnssassembler::MultiplyOperation:



Public Member Functions

- bool **validate** () const noexcept override
  Validates the expression.

Protected Member Functions

- int32_t **calculate** (int32_t lhs, int32_t rhs) const noexcept override
  Calculates the value of the subexpression.

Detailed Description

Class implementing the behaviour of the * operator in expressions.

Definition at line 10 of file MultiplyOperation.h.

Member Function Documentation

int32_t bnssassembler::MultiplyOperation::calculate (int32_t *lhs*, int32_t *rhs*) const`[override]`, `[protected]`, `[virtual]`, `[noexcept]`

Calculates the value of the subexpression.

Parameters:

lhs	Left side of the operator
rhs	Right side of the operator

Returns:

Result of the operation

Exceptions:

Throws	if the expression can not be evaluated (example: division by zero)

Implements **bnssassembler::Operation** (*p.309*).

Definition at line 9 of file MultiplyOperation.cpp.

```
{
        return lhs * rhs;
    }
```

bool bnssassembler::MultiplyOperation::validate () const`[override]`, `[virtual]`, `[noexcept]`

Validates the expression.

Returns:

Boolean value indicating whether the expression is correct

Reimplemented from **bnssassembler::Expression** (*p.167*).

Definition at line 5 of file MultiplyOperation.cpp.

References bnssassembler::Operation::containsSymbol().

```
                                               {
        return !containsSymbol();
    }
```

The documentation for this class was generated from the following files:

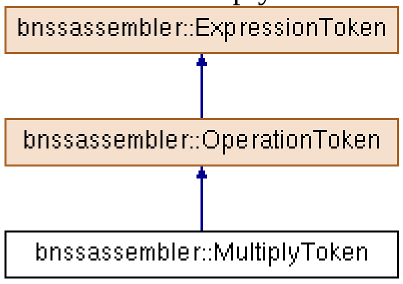
- Code/Assembler/Include/**MultiplyOperation.h**
- Code/Assembler/Source/**MultiplyOperation.cpp**

bnssassembler::MultiplyToken Class Reference

Token class representing the * operation.

`#include <MultiplyToken.h>`

Inheritance diagram for bnssassembler::MultiplyToken:


bnssassembler::ExpressionToken
bnssassembler::OperationToken
bnssassembler::MultiplyToken

Public Member Functions

- int **inputPriority** () const noexcept override
  Gets the input priority of the token.
- int **stackPriority** () const noexcept override
  Gets the stack priority of the token.
- int **rank** () const noexcept override
  Gets the rank of the token.
- std::string **operation** () const noexcept override
- std::shared_ptr< **Expression** > **create** () const override
  Creates an expression object out of the token.

Protected Member Functions

- std::shared_ptr< **ExpressionToken** > **clone** (std::string param) const override
  Clones the current object, using the string provided.

Detailed Description

Token class representing the * operation.

Definition at line 10 of file MultiplyToken.h.

Member Function Documentation

std::shared_ptr< ExpressionToken > bnssassembler::MultiplyToken::clone (std::string *param*) const `[override]`, `[protected]`, `[virtual]`

Clones the current object, using the string provided.

Parameters:

param	String that will be used to construct the new object

Returns:

Pointer to the cloned object

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 26 of file MultiplyToken.cpp.

```
   26
{
   27         return std::make_shared<MultiplyToken>();
   28     }
```

std::shared_ptr< Expression > bnssassembler::MultiplyToken::create () const[override], [virtual]

Creates an expression object out of the token.

Returns:

Pointer to the expression

Exceptions:

Throws	if the token has no corresponding expression object

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 22 of file MultiplyToken.cpp.

```
   22                                                          {
   23         return std::make_shared<MultiplyOperation>();
   24     }
```

int bnssassembler::MultiplyToken::inputPriority () const[override], [virtual], [noexcept]

Gets the input priority of the token.

Returns:

Input priority of the token

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 6 of file MultiplyToken.cpp.

```
    6                                                     {
    7         return 3;
    8     }
```

std::string bnssassembler::MultiplyToken::operation () const[override], [virtual], [noexcept]

Implements **bnssassembler::OperationToken** (*p.314*).

Definition at line 18 of file MultiplyToken.cpp.

```
   18                                                        {
   19         return "*";
   20     }
```

int bnssassembler::MultiplyToken::rank () const[override], [virtual], [noexcept]

Gets the rank of the token.

Returns:

Rank of the token

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 14 of file MultiplyToken.cpp.

```
   14                                          {
   15         return -1;
   16     }
```

int bnssassembler::MultiplyToken::stackPriority () const `[override]`, `[virtual]`, `[noexcept]`

Gets the stack priority of the token.

Returns:

Stack priority of the token

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 10 of file MultiplyToken.cpp.

```
                                                          {
        return 3;
    }
```

The documentation for this class was generated from the following files:

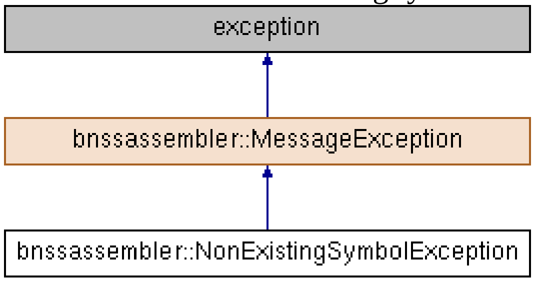
- Code/Assembler/Include/**MultiplyToken.h**
- Code/Assembler/Source/**MultiplyToken.cpp**

bnssassembler::NonExistingSymbolException Class Reference

Exception representing the non existing symbol.

`#include <NonExistingSymbolException.h>`

Inheritance diagram for bnssassembler::NonExistingSymbolException:


exception
bnssassembler::MessageException
bnssassembler::NonExistingSymbolException

Public Member Functions

- **NonExistingSymbolException** (std::string symbol) noexcept
  *Constructs a **NonExistingSymbolException** object.*

Detailed Description

Exception representing the non existing symbol.

Definition at line 11 of file NonExistingSymbolException.h.

Constructor & Destructor Documentation

bnssassembler::NonExistingSymbolException::NonExistingSymbolException (std::string *symbol*)`[explicit]`, `[noexcept]`

Constructs a **NonExistingSymbolException** object.

Parameters:

symbol	Non existing symbol

Definition at line 5 of file NonExistingSymbolException.cpp.

```
    5 : MessageException("The symbol \"" + symbol + "\" is not defined") {}
```

The documentation for this class was generated from the following files:

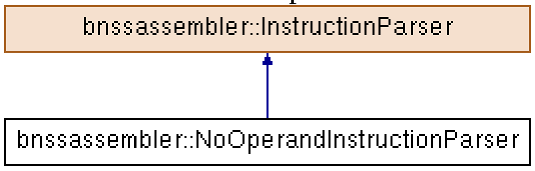
- Code/Assembler/Include/**NonExistingSymbolException.h**
- Code/Assembler/Source/**NonExistingSymbolException.cpp**

bnssassembler::NoOperandInstructionParser Class Reference

Class representing the parser for the instruction without operands.

`#include <NoOperandInstructionParser.h>`

Inheritance diagram for bnssassembler::NoOperandInstructionParser:


bnssassembler::InstructionParser
bnssassembler::NoOperandInstructionParser

Additional Inherited Members

Detailed Description

Class representing the parser for the instruction without operands.

Definition at line 10 of file NoOperandInstructionParser.h.

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

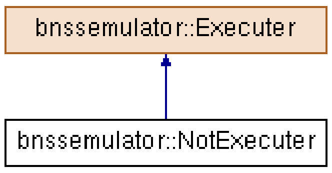
- Code/Assembler/Include/**NoOperandInstructionParser.h**

bnssemulator::NotExecuter Class Reference

Class representing the executer for the not instruction.

`#include <NotExecuter.h>`

Inheritance diagram for bnssemulator::NotExecuter:



Public Member Functions

- void **execute** (**InstructionBitField** instruction, **Context** &context) const override
  Executes the instruction.

Detailed Description

Class representing the executer for the not instruction.

Definition at line 10 of file NotExecuter.h.

Member Function Documentation

void bnssemulator::NotExecuter::execute (InstructionBitField *instruction*, Context & *context*) const `[override]`, `[virtual]`

Executes the instruction.

Parameters:

instruction	Instruction
context	**Processor** context

Implements **bnssemulator::Executer** (*p.163*).

Definition at line 5 of file NotExecuter.cpp.

References bnssemulator::Context::getRegister(), bnssemulator::InstructionBitField::register0, and bnssemulator::InstructionBitField::register1.

```
{
            auto &dst = context.getRegister(instruction.register0);
            auto &src = context.getRegister(instruction.register1);

            dst = ~src;
        }
```

The documentation for this class was generated from the following files:

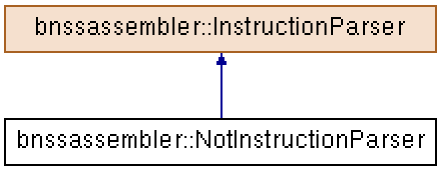
- Code/Emulator/Include/**NotExecuter.h**
- Code/Emulator/Source/**NotExecuter.cpp**

bnssassembler::NotInstructionParser Class Reference

Class representing the parser for the not instruction.

`#include <NotInstructionParser.h>`

Inheritance diagram for bnssassembler::NotInstructionParser:


bnssassembler::InstructionParser
bnssassembler::NotInstructionParser

Public Member Functions

- **NotInstructionParser** () noexcept
  *Constructs a **NotInstructionParser** object.*

Additional Inherited Members

Detailed Description

Class representing the parser for the not instruction.

Definition at line 10 of file NotInstructionParser.h.

Constructor & Destructor Documentation

bnssassembler::NotInstructionParser::NotInstructionParser () `[noexcept]`

Constructs a **NotInstructionParser** object.

Definition at line 6 of file NotInstructionParser.cpp.

References bnssassembler::InstructionParser::operands_.

```
                                                {
        operands_.push_back(std::make_shared<RegisterDirectParser>());
        operands_.push_back(std::make_shared<RegisterDirectParser>());
    }
```

The documentation for this class was generated from the following files:

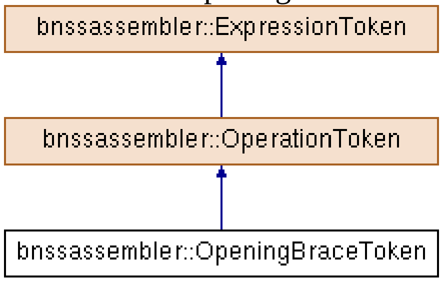
- Code/Assembler/Include/**NotInstructionParser.h**
- Code/Assembler/Source/**NotInstructionParser.cpp**

bnssassembler::OpeningBraceToken Class Reference

Token class representing the opening brace.

`#include <OpeningBraceToken.h>`

Inheritance diagram for bnssassembler::OpeningBraceToken:


bnssassembler::ExpressionToken
bnssassembler::OperationToken
bnssassembler::OpeningBraceToken

Public Member Functions

- int **inputPriority** () const noexcept override
  Gets the input priority of the token.
- int **stackPriority** () const noexcept override
  Gets the stack priority of the token.
- int **rank** () const noexcept override
  Gets the rank of the token.
- std::string **operation** () const noexcept override
- std::shared_ptr< **Expression** > **create** () const override
  Creates an expression object out of the token.

Protected Member Functions

- std::shared_ptr< **ExpressionToken** > **clone** (std::string param) const override
  Clones the current object, using the string provided.

Detailed Description

Token class representing the opening brace.

Definition at line 10 of file OpeningBraceToken.h.

Member Function Documentation

std::shared_ptr< ExpressionToken > bnssassembler::OpeningBraceToken::clone (std::string *param*) const `[override]`, `[protected]`, `[virtual]`

Clones the current object, using the string provided.

Parameters:

param	String that will be used to construct the new object

Returns:

Pointer to the cloned object

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 27 of file OpeningBraceToken.cpp.

```
{
        return std::make_shared<OpeningBraceToken>();
    }
```

std::shared_ptr< Expression > bnssassembler::OpeningBraceToken::create () const[override], [virtual]

Creates an expression object out of the token.

Returns:

Pointer to the expression

Exceptions:

Throws	if the token has no corresponding expression object

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 23 of file OpeningBraceToken.cpp.

```
                                                        {
        throw MessageException("Error - opening brace without closing brace");
    }
```

int bnssassembler::OpeningBraceToken::inputPriority () const[override], [virtual], [noexcept]

Gets the input priority of the token.

Returns:

Input priority of the token

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 7 of file OpeningBraceToken.cpp.

```
                                                  {
        return INT_MAX - 1;
    }
```

std::string bnssassembler::OpeningBraceToken::operation () const[override], [virtual], [noexcept]

Implements **bnssassembler::OperationToken** (*p.314*).

Definition at line 19 of file OpeningBraceToken.cpp.

```
                                                    {
        return "(";
    }
```

int bnssassembler::OpeningBraceToken::rank () const[override], [virtual], [noexcept]

Gets the rank of the token.

Returns:

Rank of the token

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 15 of file OpeningBraceToken.cpp.

```
                                          {
```

```
        return 0;
    }
```

int bnssassembler::OpeningBraceToken::stackPriority () const `[override]`**,** `[virtual]`**,** `[noexcept]`

Gets the stack priority of the token.

Returns:

Stack priority of the token

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 11 of file OpeningBraceToken.cpp.

```
                                                    {
        return 0;
    }
```

The documentation for this class was generated from the following files:

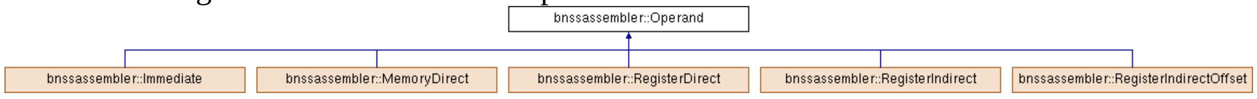
- Code/Assembler/Include/**OpeningBraceToken.h**
- Code/Assembler/Source/**OpeningBraceToken.cpp**

bnssassembler::Operand Class Reference

Class representing one operand in an instruction.

`#include <Operand.h>`

Inheritance diagram for bnssassembler::Operand:


bnssassembler::Operand
bnssassembler::Immediate
bnssassembler::MemoryDirect
bnssassembler::RegisterDirect
bnssassembler::RegisterIndirect
bnssassembler::RegisterIndirectOffset

Public Member Functions

- virtual void **packToInstruction** (**InstructionBitFieldUnion** &instruction, **uint32_t** &second_word, std::list< **RelocationRecord** > &relocations) const =0
  Packs the operand into the instruction.
- virtual void **resolveSymbols** (std::unordered_set< **SymbolDefinition** > symbols) noexcept
  Resolves the defined symbols in the expressions.
- virtual void **resolveSymbolTable** (const **SymbolTable** &symbol_table) noexcept
  Resolves the symbols from the symbol table and updates the relocation info.
- virtual void **resolveImports** (std::unordered_set< std::string > imported_symbols) noexcept
  Resolves the imported symbols and updates the relocation info.
- virtual void **resolveCurrentPcSymbol** (size_t section_index, size_t offset) noexcept
  Resolves the current PC symbol and sets the relocation info.
- virtual **AddressMode** **addressMode** () const noexcept=0
  Gets the address mode of the operand.
- virtual **~Operand** ()=default

Detailed Description

Class representing one operand in an instruction.

Definition at line 13 of file Operand.h.

Constructor & Destructor Documentation

virtual bnssassembler::Operand::~Operand () `[virtual]`, `[default]`

Member Function Documentation

virtual AddressMode bnssassembler::Operand::addressMode () const `[pure virtual]`, `[noexcept]`

Gets the address mode of the operand.

Returns:

Address mode of the operand

Implemented in **bnssassembler::RegisterIndirectOffset** (*p.382*), **bnssassembler::Immediate** (*p.206*), **bnssassembler::MemoryDirect** (*p.269*), **bnssassembler::RegisterDirect** (*p.373*), and **bnssassembler::RegisterIndirect** (*p.378*).

virtual void bnssassembler::Operand::packToInstruction (InstructionBitFieldUnion & *instruction*, uint32_t & *second_word*, std::list< RelocationRecord > & *relocations*) const`[pure virtual]`

Packs the operand into the instruction.

Parameters:

instruction	Reference to the first word of the instruction containing the instruction info
second_word	Reference to the second word of the instruction containing the address/value/displacement
relocations	Reference to the list of relocation records

Implemented in **bnssassembler::RegisterIndirectOffset** (*p.382*), **bnssassembler::Immediate** (*p.206*), **bnssassembler::MemoryDirect** (*p.269*), **bnssassembler::RegisterDirect** (*p.374*), and **bnssassembler::RegisterIndirect** (*p.379*).

void bnssassembler::Operand::resolveCurrentPcSymbol (size_t *section_index*, size_t *offset*)`[virtual]`, `[noexcept]`

Resolves the current PC symbol and sets the relocation info.

Parameters:

section_index	Current PC section
offset	PC address in relation to the current section beginning

Reimplemented in **bnssassembler::RegisterIndirectOffset** (*p.383*), **bnssassembler::Immediate** (*p.206*), and **bnssassembler::MemoryDirect** (*p.269*).

Definition at line 18 of file Operand.cpp.

```
{
        // Default: Do nothing
    }
```

void bnssassembler::Operand::resolveImports (std::unordered_set< std::string > *imported_symbols*)`[virtual]`, `[noexcept]`

Resolves the imported symbols and updates the relocation info.

Parameters:

imported_symbols	Collection of imported symbols

Reimplemented in **bnssassembler::RegisterIndirectOffset** (*p.383*), **bnssassembler::Immediate** (*p.207*), and **bnssassembler::MemoryDirect** (*p.270*).

Definition at line 14 of file Operand.cpp.

```
{
        // Default: Do nothing
    }
```

void bnssassembler::Operand::resolveSymbols (std::unordered_set< SymbolDefinition > *symbols*)`[virtual]`, `[noexcept]`

Resolves the defined symbols in the expressions.

Parameters:

symbols	Collection of symbol definitions

Reimplemented in **bnssassembler::RegisterIndirectOffset** (*p.384*), **bnssassembler::Immediate** (*p.207*), and **bnssassembler::MemoryDirect** (*p.270*).

Definition at line 6 of file Operand.cpp.

```
{
        // Default: Do nothing
    }
```

void bnssassembler::Operand::resolveSymbolTable (const SymbolTable & *symbol_table*)[virtual], [noexcept]

Resolves the symbols from the symbol table and updates the relocation info.

Parameters:

symbol_table	**Symbol** table

Reimplemented in **bnssassembler::RegisterIndirectOffset** (*p.384*), **bnssassembler::Immediate** (*p.207*), and **bnssassembler::MemoryDirect** (*p.270*).

Definition at line 10 of file Operand.cpp.

```
{
        // Default: Do nothing
    }
```

The documentation for this class was generated from the following files:

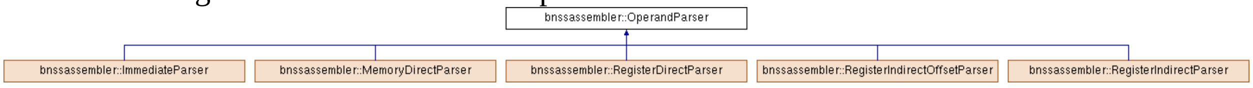
- Code/Assembler/Include/**Operand.h**
- Code/Assembler/Source/**Operand.cpp**

bnssassembler::OperandParser Class Reference

Chain of command class used to parse operands of the instructions.

`#include <OperandParser.h>`

Inheritance diagram for bnssassembler::OperandParser:


bnssassembler::OperandParser
bnssassembler::ImmediateParser
bnssassembler::MemoryDirectParser
bnssassembler::RegisterDirectParser
bnssassembler::RegisterIndirectOffsetParser
bnssassembler::RegisterIndirectParser

Public Member Functions

- std::shared_ptr< **Operand** > **tryParse** (std::string str) const
  Tries to parse one operand. Calls the next parser in the chain if it fails.
- void **next** (std::shared_ptr< **OperandParser** > next) noexcept
  Sets the next parser in the chain.
- virtual **~OperandParser** ()=default

Protected Member Functions

- virtual std::shared_ptr< **Operand** > **parse** (std::string str) const =0
  Parses one operand. Does not call the next parser if it fails.

Private Attributes

- std::shared_ptr< **OperandParser** > **next_**
  The next parser in the chain.

Detailed Description

Chain of command class used to parse operands of the instructions.

Definition at line 12 of file OperandParser.h.

Constructor & Destructor Documentation

virtual bnssassembler::OperandParser::~OperandParser () `[virtual]`, `[default]`

Member Function Documentation

void bnssassembler::OperandParser::next (std::shared_ptr< OperandParser > *next*) `[noexcept]`

Sets the next parser in the chain.

Parameters:

next	Next parser in the chain

Definition at line 18 of file OperandParser.cpp.

References next_.

```
                                                                                         {
          next_ = next;
     }
```

virtual std::shared_ptr<Operand> bnssassembler::OperandParser::parse (std::string *str*) const `[protected], [pure virtual]`

Parses one operand. Does not call the next parser if it fails.

Parameters:

str	**Operand** which should be parsed

Returns:

Pointer to the operand or nullptr, if the parser failed parsing

Exceptions:

Throws	if the parser fails but identifies the error

Implemented in **bnssassembler::ImmediateParser** (*p.209*), **bnssassembler::MemoryDirectParser** (*p.272*), **bnssassembler::RegisterDirectParser** (*p.376*), **bnssassembler::RegisterIndirectOffsetParser** (*p.386*), and **bnssassembler::RegisterIndirectParser** (*p.388*).

Referenced by tryParse().

std::shared_ptr< Operand > bnssassembler::OperandParser::tryParse (std::string *str*) const

Tries to parse one operand. Calls the next parser in the chain if it fails.

Parameters:

str	**Operand** which should be parsed

Returns:

Pointer to the operand or nullptr, if the whole chain failed parsing

Exceptions:

Throws	if the chain fails but identifies the error

Definition at line 5 of file OperandParser.cpp.

References next_, and parse().

```
                                                                        {
        auto ret = parse(str);
        if (ret != nullptr) {
            return ret;
        }

        if (next_ == nullptr) {
            return nullptr;
        }

        return next_->tryParse(str);
    }
```

Member Data Documentation

std::shared_ptr<OperandParser> bnssassembler::OperandParser::next_ `[private]`

The next parser in the chain.

Definition at line 40 of file OperandParser.h.

Referenced by next(), and tryParse().

The documentation for this class was generated from the following files:

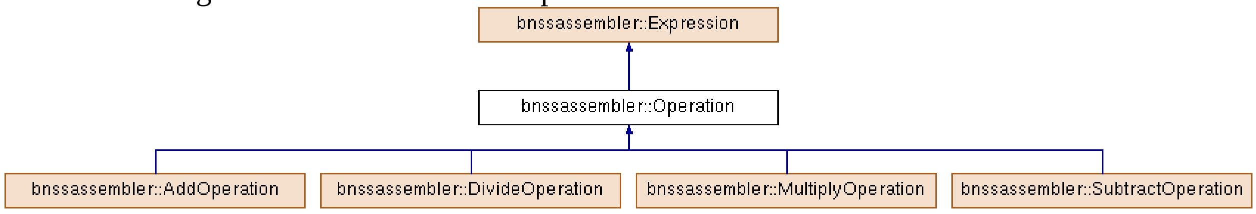
- Code/Assembler/Include/**OperandParser.h**
- Code/Assembler/Source/**OperandParser.cpp**

bnssassembler::Operation Class Reference

Class representing the mathematical operation with two operands.

```
#include <Operation.h>
```

Inheritance diagram for bnssassembler::Operation:


bnssassembler::Expression
bnssassembler::Operation
bnssassembler::AddOperation
bnssassembler::DivideOperation
bnssassembler::MultiplyOperation
bnssassembler::SubtractOperation

Public Member Functions

- int32_t **value** () const override
  Evaluates the expression.
- bool **setValue** (std::string symbol, std::shared_ptr< **Expression** > **value**) noexcept override
  Traverses the subtree and sets the value for the symbol.
- bool **containsSymbol** () const noexcept override
  Tests whether the expression contains a ***Symbol****.*
- int **symbolCount** () const noexcept override
  Counts the symbols in the expression.
- void **pushChildren** (std::stack< std::reference_wrapper< std::shared_ptr< **Expression** >>> &stack) const noexcept override
  Pushes the children to the stack.
- void **resolveSymbolTable** (const **SymbolTable** &symbol_table) noexcept override
  Resolves the symbols from the symbol table and sets the relocation info.
- void **resolveImports** (std::unordered_set< std::string > imported_symbols) noexcept override
  Resolves the imported symbols and sets the relocation info.
- void **resolveCurrentPcSymbol** (size_t section_index, size_t offset) noexcept override
  Resolves the current PC symbol and sets the relocation info.
- std::list< **RelocationRecord** > **generateRelocations** () const override
  Generates the relocation records for the subtree.
- void **left** (std::shared_ptr< **Expression** > left) noexcept
  Sets the left side of the operator.
- void **right** (std::shared_ptr< **Expression** > right) noexcept
  Sets the right side of the operator.

Protected Member Functions

- virtual int32_t **calculate** (int32_t lhs, int32_t rhs) const =0
  Calculates the value of the subexpression.
- std::shared_ptr< **Expression** > **left** () const noexcept
  Gets the left side of the operator.
- std::shared_ptr< **Expression** > **right** () const noexcept
  Gets the right side of the operator.

Private Attributes

- std::shared_ptr< **Expression** > **left_**
- std::shared_ptr< **Expression** > **right_**

Detailed Description

Class representing the mathematical operation with two operands.

Definition at line 11 of file Operation.h.

Member Function Documentation

virtual int32_t bnssassembler::Operation::calculate (int32_t *lhs*, int32_t *rhs*) const`[protected]`, `[pure virtual]`

Calculates the value of the subexpression.

Parameters:

lhs	Left side of the operator
rhs	Right side of the operator

Returns:

Result of the operation

Exceptions:

Throws	if the expression can not be evaluated (example: division by zero)

Implemented in **bnssassembler::SubtractOperation** (*p.473*), **bnssassembler::AddOperation** (*p.87*), **bnssassembler::DivideOperation** (*p.156*), and **bnssassembler::MultiplyOperation** (*p.290*).

Referenced by value().

bool bnssassembler::Operation::containsSymbol () const`[override]`, `[virtual]`, `[noexcept]`

Tests whether the expression contains a **Symbol**.

Returns:

Boolean value indicating whether the expression contains a **Symbol**

Reimplemented from **bnssassembler::Expression** (*p.165*).

Reimplemented in **bnssassembler::SubtractOperation** (*p.474*).

Definition at line 28 of file Operation.cpp.

References left_, and right_.

Referenced by bnssassembler::SubtractOperation::containsSymbol(), bnssassembler::DivideOperation::validate(), and bnssassembler::MultiplyOperation::validate().

```
28                                              {
29         return left_->containsSymbol() || right_->containsSymbol();
30     }
```

std::list< RelocationRecord > bnssassembler::Operation::generateRelocations () const`[override]`, `[virtual]`

Generates the relocation records for the subtree.

Returns:

Collection of relocation records

Reimplemented from **bnssassembler::Expression** (*p.165*).

Reimplemented in **bnssassembler::SubtractOperation** (*p.474*).

Definition at line 56 of file Operation.cpp.

References left(), left_, right(), and right_.

Referenced by bnssassembler::AddOperation::generateRelocations().

```
                                                                    {
        auto left = left_->generateRelocations();
        auto right = right_->generateRelocations();
        left.splice(left.end(), move(right));
        return left;
    }
```

void bnssassembler::Operation::left (std::shared_ptr< Expression > *left*)`[noexcept]`

Sets the left side of the operator.

Parameters:

left	Pointer to the expression on the left side

Definition at line 12 of file Operation.cpp.

References left(), and left_.

```
                                                                 {
        left_ = left;
    }
```

std::shared_ptr< Expression > bnssassembler::Operation::left () const`[protected]`, `[noexcept]`

Gets the left side of the operator.

Returns:

Pointer to the expression on the right side

Definition at line 20 of file Operation.cpp.

References left_.

Referenced by bnssassembler::AddOperation::generateRelocations(), bnssassembler::SubtractOperation::generateRelocations(), generateRelocations(), left(), and bnssassembler::SubtractOperation::symbolCount().

```
                                                              {
        return left_;
    }
```

void bnssassembler::Operation::pushChildren (std::stack< std::reference_wrapper< std::shared_ptr< Expression >>> & *stack*) const`[override]`, `[virtual]`, `[noexcept]`

Pushes the children to the stack.

Parameters:

stack	Reference to the stack

Reimplemented from **bnssassembler::Expression** (*p.165*).

Definition at line 36 of file Operation.cpp.

References left_, and right_.

```
{
        stack.push(const_cast<std::shared_ptr<Expression>&>(left_));
        stack.push(const_cast<std::shared_ptr<Expression>&>(right_));
    }
```

void bnssassembler::Operation::resolveCurrentPcSymbol (size_t *section_index*, size_t *offset*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the current PC symbol and sets the relocation info.

Parameters:

section_index	Current PC section
offset	PC address in relation to the current section beginning

Reimplemented from **bnssassembler::Expression** (*p.165*).

Definition at line 51 of file Operation.cpp.

References left_, and right_.

```
{
        left_->resolveCurrentPcSymbol(section_index, offset);
        right_->resolveCurrentPcSymbol(section_index, offset);
    }
```

void bnssassembler::Operation::resolveImports (std::unordered_set< std::string > *imported_symbols*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the imported symbols and sets the relocation info.

Parameters:

imported_symbols	Collection of imported symbols

Reimplemented from **bnssassembler::Expression** (*p.166*).

Definition at line 46 of file Operation.cpp.

References left_, and right_.

```
{
        left_->resolveImports(imported_symbols);
        right_->resolveImports(imported_symbols);
    }
```

void bnssassembler::Operation::resolveSymbolTable (const SymbolTable & *symbol_table*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the symbols from the symbol table and sets the relocation info.

Parameters:

symbol_table	**Symbol** table

Reimplemented from **bnssassembler::Expression** (*p.166*).

Definition at line 41 of file Operation.cpp.

References left_, and right_.

```
{
        left_->resolveSymbolTable(symbol_table);
        right_->resolveSymbolTable(symbol_table);
    }
```

void bnssassembler::Operation::right (std::shared_ptr< Expression > *right*)`[noexcept]`

Sets the right side of the operator.

Parameters:

right	Pointer to the expression on the right side

Definition at line 16 of file Operation.cpp.

References right(), and right_.

```
                                                                    {
        right_ = right;
    }
```

std::shared_ptr< Expression > bnssassembler::Operation::right () const`[protected]`, `[noexcept]`

Gets the right side of the operator.

Returns:

Pointer to the expression on the right side

Definition at line 24 of file Operation.cpp.

References right_.

Referenced by bnssassembler::AddOperation::generateRelocations(), bnssassembler::SubtractOperation::generateRelocations(), generateRelocations(), right(), and bnssassembler::SubtractOperation::symbolCount().

```
                                                                {
        return right_;
    }
```

bool bnssassembler::Operation::setValue (std::string *symbol*, std::shared_ptr< Expression > *value*)`[override]`, `[virtual]`, `[noexcept]`

Traverses the subtree and sets the value for the symbol.

Parameters:

symbol	Name of the symbol
value	New value of the symbol

Returns:

Whether the symbol was found and the value was set

Reimplemented from **bnssassembler::Expression** (*p.166*).

Definition at line 8 of file Operation.cpp.

References left_, right_, and value().

```
{
        return left_->setValue(symbol, value) || right_->setValue(symbol, value);
    }
```

int bnssassembler::Operation::symbolCount () const`[override]`, `[virtual]`, `[noexcept]`

Counts the symbols in the expression.

Returns:

Number of symbols in the expression

Reimplemented from **bnssassembler::Expression** (*p.167*).

Reimplemented in **bnssassembler::SubtractOperation** (*p.475*).

Definition at line 32 of file Operation.cpp.

References left_, and right_.

```
                                          {
        return left_->symbolCount() + right_->symbolCount();
    }
```

int32_t bnssassembler::Operation::value () const`[override]`, `[virtual]`

Evaluates the expression.

Exceptions:

Throws	if the expression has variables or could not be evaluated (for example, division by zero)

Implements **bnssassembler::Expression** (*p.167*).

Definition at line 4 of file Operation.cpp.

References calculate(), left_, and right_.

Referenced by setValue().

```
                                 {
        return calculate(left_->value(), right_->value());
    }
```

Member Data Documentation

std::shared_ptr<Expression> bnssassembler::Operation::left_`[private]`

Definition at line 58 of file Operation.h.

Referenced by containsSymbol(), generateRelocations(), left(), pushChildren(), resolveCurrentPcSymbol(), resolveImports(), resolveSymbolTable(), setValue(), symbolCount(), and value().

std::shared_ptr<Expression> bnssassembler::Operation::right_`[private]`

Definition at line 59 of file Operation.h.

Referenced by containsSymbol(), generateRelocations(), pushChildren(), resolveCurrentPcSymbol(), resolveImports(), resolveSymbolTable(), right(), setValue(), symbolCount(), and value().

The documentation for this class was generated from the following files:

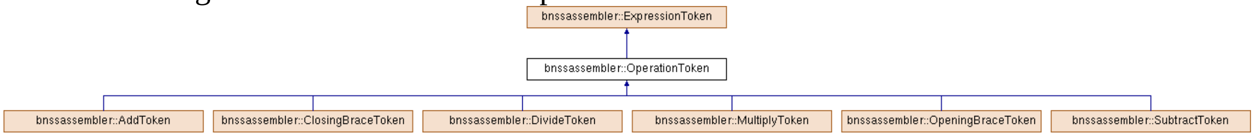
- Code/Assembler/Include/**Operation.h**
- Code/Assembler/Source/**Operation.cpp**

bnssassembler::OperationToken Class Reference

Token class representing a math operator.

`#include <OperationToken.h>`

Inheritance diagram for bnssassembler::OperationToken:


bnssassembler::ExpressionToken
bnssassembler::OperationToken
bnssassembler::AddToken
bnssassembler::ClosingBraceToken
bnssassembler::DivideToken
bnssassembler::MultiplyToken
bnssassembler::OpeningBraceToken
bnssassembler::SubtractToken

Public Member Functions

- void **process** (std::list< std::shared_ptr< **ExpressionToken** >> &output, std::stack< std::shared_ptr< **ExpressionToken** >> &stack, int &expression_rank) const override
  Processes the current token.
- virtual std::string **operation** () const noexcept=0

Protected Member Functions

- virtual bool **isClosingBrace** () const noexcept
  Checks if the operator is the closing brace (closing brace should not be on the stack)

Detailed Description

Token class representing a math operator.

Definition at line 10 of file OperationToken.h.

Member Function Documentation

bool bnssassembler::OperationToken::isClosingBrace () const `[protected]`, `[virtual]`, `[noexcept]`

Checks if the operator is the closing brace (closing brace should not be on the stack)

Returns:

Whether the operator is the closing brace

Reimplemented in **bnssassembler::ClosingBraceToken** (*p.118*).

Definition at line 24 of file OperationToken.cpp.

Referenced by process().

```
24                                                        {
25          return false;
26      }
```

virtual std::string bnssassembler::OperationToken::operation () const `[pure virtual]`, `[noexcept]`

Implemented in **bnssassembler::AddToken** (*p.99*), **bnssassembler::ClosingBraceToken** (*p.118*), **bnssassembler::DivideToken** (*p.159*), **bnssassembler::MultiplyToken** (*p.293*), **bnssassembler::OpeningBraceToken** (*p.300*), and **bnssassembler::SubtractToken** (*p.478*).

void bnssassembler::OperationToken::process (std::list< std::shared_ptr< ExpressionToken >> & *output*, std::stack< std::shared_ptr< ExpressionToken >> & *stack*, int & *expression_rank*) const`[override]`, `[virtual]`

Processes the current token.

Parameters:

output	Output list of tokens
stack	Helper stack of tokens
expression_rank	Rank of the expression

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 6 of file OperationToken.cpp.

References bnssassembler::ExpressionToken::clone(), bnssassembler::ExpressionToken::inputPriority(), isClosingBrace(), and bnssassembler::ExpressionBuilder::popToPostfix().

```
{
        while (!stack.empty() && stack.top()->stackPriority() >= inputPriority()) {
            ExpressionBuilder::popToPostfix(output, stack, expression_rank);
        }

        if (isClosingBrace()) {
            if (!stack.empty()) {
                stack.pop();
            }
            else {
                throw MessageException("The opening brace is missing");
            }
        }
        else {
            stack.push(clone("dummy"));
        }
    }
```

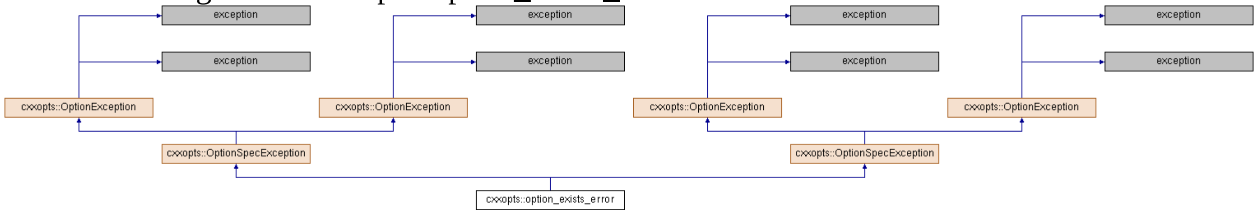
The documentation for this class was generated from the following files:

- Code/Assembler/Include/**OperationToken.h**
- Code/Assembler/Source/**OperationToken.cpp**

cxxopts::option_exists_error Class Reference

`#include <cxxopts.h>`

Inheritance diagram for cxxopts::option_exists_error:



Public Member Functions

- **option_exists_error** (const std::string &option)
- **option_exists_error** (const std::string &option)

Detailed Description

Definition at line 313 of file cxxopts.h.

Constructor & Destructor Documentation

cxxopts::option_exists_error::option_exists_error (const std::string & *option*) `[inline]`

Definition at line 316 of file cxxopts.h.

```
            : OptionSpecException("Option '" + option + "' already exists")
        {
        }
```

cxxopts::option_exists_error::option_exists_error (const std::string & *option*) `[inline]`

Definition at line 316 of file cxxopts.h.

```
            : OptionSpecException("Option '" + option + "' already exists")
        {
        }
```

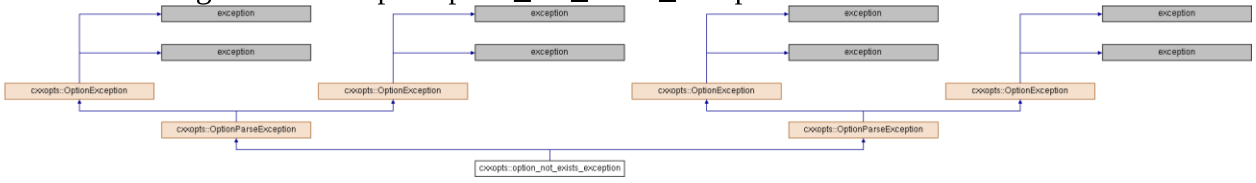
The documentation for this class was generated from the following file:

- Code/Assembler/Include/**cxxopts.h**

cxxopts::option_not_exists_exception Class Reference

`#include <cxxopts.h>`

Inheritance diagram for cxxopts::option_not_exists_exception:


exception
exception
cxxopts::OptionException
cxxopts::OptionParseException
cxxopts::option_not_exists_exception

Public Member Functions

- **option_not_exists_exception** (const std::string &option)
- **option_not_exists_exception** (const std::string &option)

Detailed Description

Definition at line 331 of file cxxopts.h.

Constructor & Destructor Documentation

cxxopts::option_not_exists_exception::option_not_exists_exception (const std::string & *option*)`[inline]`

Definition at line 334 of file cxxopts.h.

```
            : OptionParseException("Option '" + option + "' does not exist")
        {
        }
```

cxxopts::option_not_exists_exception::option_not_exists_exception (const std::string & *option*)`[inline]`

Definition at line 334 of file cxxopts.h.

```
            : OptionParseException("Option '" + option + "' does not exist")
        {
        }
```

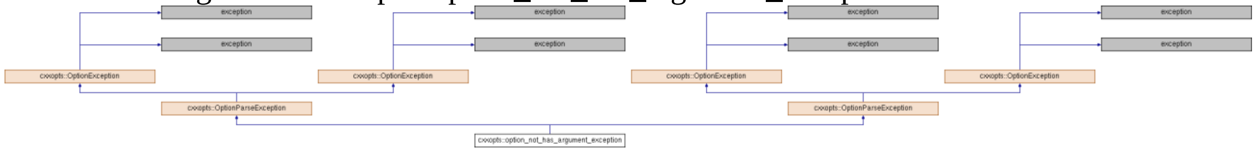
The documentation for this class was generated from the following file:

- Code/Assembler/Include/**cxxopts.h**

cxxopts::option_not_has_argument_exception Class Reference

`#include <cxxopts.h>`

Inheritance diagram for cxxopts::option_not_has_argument_exception:


exception
exception
cxxopts::OptionException
cxxopts::OptionParseException
cxxopts::option_not_has_argument_exception

Public Member Functions

- **option_not_has_argument_exception** (const std::string &option, const std::string &arg)
- **option_not_has_argument_exception** (const std::string &option, const std::string &arg)

Detailed Description

Definition at line 358 of file cxxopts.h.

Constructor & Destructor Documentation

cxxopts::option_not_has_argument_exception::option_not_has_argument_exception (const std::string & *option*, const std::string & *arg*)`[inline]`

Definition at line 362 of file cxxopts.h.

```
        : OptionParseException(
          "Option '" + option + "' does not take an argument, but argument'"
          + arg + "' given")
    {
    }
```

cxxopts::option_not_has_argument_exception::option_not_has_argument_exception (const std::string & *option*, const std::string & *arg*)`[inline]`

Definition at line 362 of file cxxopts.h.

```
        : OptionParseException(
          "Option '" + option + "' does not take an argument, but argument'"
          + arg + "' given")
    {
    }
```

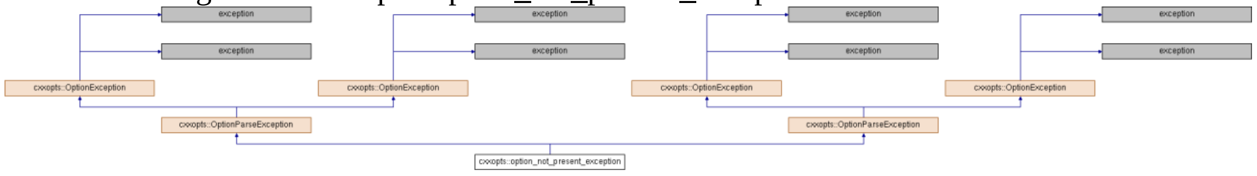
The documentation for this class was generated from the following file:

- Code/Assembler/Include/**cxxopts.h**

cxxopts::option_not_present_exception Class Reference

`#include <cxxopts.h>`

Inheritance diagram for cxxopts::option_not_present_exception:


exception
exception
cxxopts::OptionException
cxxopts::OptionParseException
cxxopts::option_not_present_exception

Public Member Functions

- **option_not_present_exception** (const std::string &option)
- **option_not_present_exception** (const std::string &option)

Detailed Description

Definition at line 373 of file cxxopts.h.

Constructor & Destructor Documentation

cxxopts::option_not_present_exception::option_not_present_exception (const std::string & *option*) `[inline]`

Definition at line 376 of file cxxopts.h.

```
                : OptionParseException("Option '" + option + "' not present")
            {
            }
```

cxxopts::option_not_present_exception::option_not_present_exception (const std::string & *option*) `[inline]`

Definition at line 376 of file cxxopts.h.

```
                : OptionParseException("Option '" + option + "' not present")
            {
            }
```

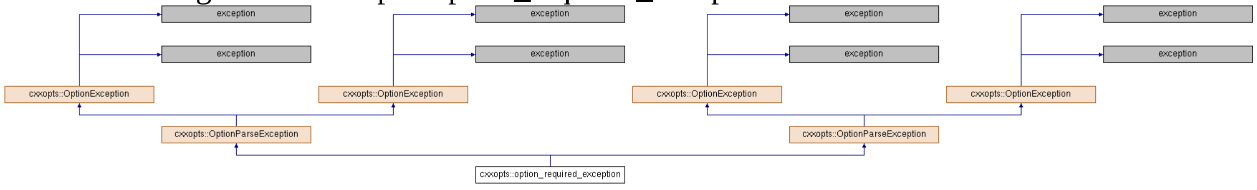
The documentation for this class was generated from the following file:

- Code/Assembler/Include/**cxxopts.h**

cxxopts::option_required_exception Class Reference

`#include <cxxopts.h>`

Inheritance diagram for cxxopts::option_required_exception:



exception
cxxopts::OptionException
cxxopts::OptionParseException
cxxopts::option_required_exception

Public Member Functions

- **option_required_exception** (const std::string &option)
- **option_required_exception** (const std::string &option)

Detailed Description

Definition at line 396 of file cxxopts.h.

Constructor & Destructor Documentation

cxxopts::option_required_exception::option_required_exception (const std::string & *option*) `[inline]`

Definition at line 399 of file cxxopts.h.

```
            : OptionParseException
            (
                "Option '" + option + "' is required but not present"
            )
        {
        }
```

cxxopts::option_required_exception::option_required_exception (const std::string & *option*) `[inline]`

Definition at line 399 of file cxxopts.h.

References cxxopts::values::parse_value(), and cxxopts::value().

```
            : OptionParseException
            (
                "Option '" + option + "' is required but not present"
            )
        {
        }
```

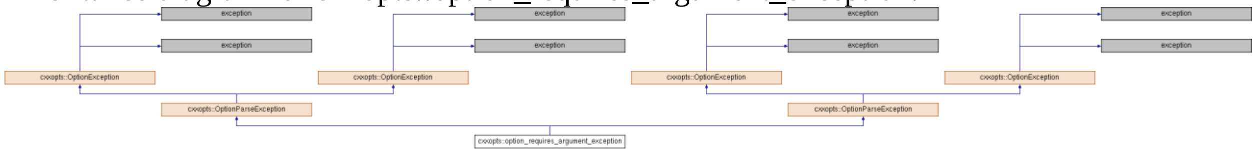
The documentation for this class was generated from the following file:

- Code/Assembler/Include/**cxxopts.h**

cxxopts::option_requires_argument_exception Class Reference

`#include <cxxopts.h>`

Inheritance diagram for cxxopts::option_requires_argument_exception:



Public Member Functions

- **option_requires_argument_exception** (const std::string &option)
- **option_requires_argument_exception** (const std::string &option)

Detailed Description

Definition at line 349 of file cxxopts.h.

Constructor & Destructor Documentation

cxxopts::option_requires_argument_exception::option_requires_argument_exception (const std::string & *option*) `[inline]`

Definition at line 352 of file cxxopts.h.

```
            : OptionParseException("Option '" + option + "' requires an argument")
        {
        }
```

cxxopts::option_requires_argument_exception::option_requires_argument_exception (const std::string & *option*) `[inline]`

Definition at line 352 of file cxxopts.h.

```
            : OptionParseException("Option '" + option + "' requires an argument")
        {
        }
```

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

- Code/Assembler/Include/**cxxopts.h**

cxxopts::OptionAdder Class Reference

`#include <cxxopts.h>`

Public Member Functions

- **OptionAdder** (**Options** &options, std::string group)
- **OptionAdder** & **operator()** (const std::string &opts, const std::string &desc, std::shared_ptr< const **Value** > **value**=::**cxxopts::value**< bool >(), std::string arg_help="")
- **OptionAdder** (**Options** &options, std::string group)
- **OptionAdder** & **operator()** (const std::string &opts, const std::string &desc, std::shared_ptr< const **Value** > **value**=::**cxxopts::value**< bool >(), std::string arg_help="")

Private Attributes

- **Options** & **m_options**
- std::string **m_group**

Detailed Description

Definition at line 820 of file cxxopts.h.

Constructor & Destructor Documentation

cxxopts::OptionAdder::OptionAdder (Options & *options*, std::string *group*)`[inline]`

Definition at line 824 of file cxxopts.h.

References cxxopts::value().

```
825             : m_options(options), m_group(std::move(group))
826         {
827         }
```

cxxopts::OptionAdder::OptionAdder (Options & *options*, std::string *group*)`[inline]`

Definition at line 824 of file cxxopts.h.

References cxxopts::Options::add_one_option(), cxxopts::Options::add_option(), cxxopts::Options::add_options(), cxxopts::Options::add_to_option(), cxxopts::HelpOptionDetails::arg_help, cxxopts::check_required(), cxxopts::Options::checked_parse_arg(), cxxopts::Options::consume_positional(), cxxopts::Options::count(), cxxopts::HelpOptionDetails::default_value, cxxopts::HelpOptionDetails::desc, cxxopts::empty(), cxxopts::anonymous_namespace{cxxopts.h}::format_description(), cxxopts::anonymous_namespace{cxxopts.h}::format_option(), cxxopts::Options::generate_all_groups_help(), cxxopts::Options::generate_group_help(), cxxopts::Options::group_help(), cxxopts::Options::groups(), cxxopts::HelpOptionDetails::has_arg, cxxopts::HelpOptionDetails::has_default, cxxopts::HelpOptionDetails::has_implicit, cxxopts::Options::help(), cxxopts::Options::help_one_group(), cxxopts::HelpOptionDetails::implicit_value, cxxopts::HelpOptionDetails::l, bnssassembler::name(), cxxopts::anonymous_namespace{cxxopts.h}::OPTION_DESC_GAP, cxxopts::anonymous_namespace{cxxopts.h}::OPTION_LONGEST, cxxopts::anonymous_namespace{cxxopts.h}::option_matcher(), cxxopts::anonymous_namespace{cxxopts.h}::option_specifier(), cxxopts::Options::parse(), cxxopts::Options::parse_option(), cxxopts::Options::parse_positional(), cxxopts::HelpOptionDetails::s, cxxopts::stringAppend(), cxxopts::stringLength(), cxxopts::toLocalString(), cxxopts::toUTF8String(), and cxxopts::value().

```
          : m_options(options), m_group(std::move(group))
        {
        }
```

Member Function Documentation

OptionAdder & cxxopts::OptionAdder::operator() (const std::string & *opts*, const std::string & *desc*, std::shared_ptr< const Value > *value* = `::cxxopts::value<bool>()`, std::string *arg_help* = `""`)`[inline]`

Definition at line 988 of file cxxopts.h.

References cxxopts::anonymous_namespace{cxxopts.h}::option_specifier(), and cxxopts::Options::parse_option().

```
    {
        std::match_results<const char*> result;
        std::regex_match(opts.c_str(), result, option_specifier);

        if (result.empty())
        {
            throw invalid_option_format_error(opts);
        }

        const auto& short_match = result[2];
        const auto& long_match = result[3];

        if (!short_match.length() && !long_match.length())
        {
            throw invalid_option_format_error(opts);
        }
        else if (long_match.length() == 1 && short_match.length())
        {
            throw invalid_option_format_error(opts);
        }

        auto option_names = []
        (
            const std::sub_match<const char*>& short_,
            const std::sub_match<const char*>& long_
            )
        {
            if (long_.length() == 1)
            {
                return std::make_tuple(long_.str(), short_.str());
            }
            else
            {
                return std::make_tuple(short_.str(), long_.str());
            }
        }(short_match, long_match);

        m_options.add_option
        (
            m_group,
            std::get<0>(option_names),
            std::get<1>(option_names),
            desc,
            value,
            std::move(arg_help)
        );

        return *this;
    }
```

OptionAdder& cxxopts::OptionAdder::operator() (const std::string & *opts*, const std::string & *desc*, std::shared_ptr< const Value > *value* = `::cxxopts::value< bool >()`, std::string *arg_help* = `""`)`[inline]`

Member Data Documentation

std::string cxxopts::OptionAdder::m_group `[private]`

Definition at line 842 of file cxxopts.h.

Options & cxxopts::OptionAdder::m_options `[private]`

Definition at line 841 of file cxxopts.h.

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

- Code/Assembler/Include/**cxxopts.h**

cxxopts::OptionDetails Class Reference

```
#include <cxxopts.h>
```

Public Member Functions

- **OptionDetails** (const **String** &desc, std::shared_ptr< const **Value** > val)
- const **String** & **description** () const
- bool **has_arg** () const
- void **parse** (const std::string &text)
- void **parse_default** () const
- int **count** () const
- const **Value** & **value** () const
- template<typename T > const T & **as** () const
- **OptionDetails** (const **String** &desc, std::shared_ptr< const **Value** > val)
- const **String** & **description** () const
- bool **has_arg** () const
- void **parse** (const std::string &text)
- void **parse_default** () const
- int **count** () const
- const **Value** & **value** () const
- template<typename T > const T & **as** () const

Private Attributes

- **String m_desc**
- std::shared_ptr< const **Value** > **m_value**
- int **m_count**

Detailed Description

Definition at line 581 of file cxxopts.h.

Constructor & Destructor Documentation

cxxopts::OptionDetails::OptionDetails (const String & *desc*, std::shared_ptr< const Value > *val*)`[inline]`

Definition at line 585 of file cxxopts.h.

```
            : m_desc(desc)
            , m_value(val)
            , m_count(0)
        {
        }
```

cxxopts::OptionDetails::OptionDetails (const String & *desc*, std::shared_ptr< const Value > *val*)`[inline]`

Definition at line 585 of file cxxopts.h.

```
            : m_desc(desc)
            , m_value(val)
            , m_count(0)
        {
        }
```

Member Function Documentation

template<typename T > const T& cxxopts::OptionDetails::as () const `[inline]`

Definition at line 631 of file cxxopts.h.

References cxxopts::values::standard_value< T >::get().

```
        {
#ifdef CXXOPTS_NO_RTTI
            return static_cast<const values::standard_value<T>&>(*m_value).get();
#else
            return dynamic_cast<const values::standard_value<T>&>(*m_value).get();
#endif
        }
```

template<typename T > const T& cxxopts::OptionDetails::as () const `[inline]`

Definition at line 631 of file cxxopts.h.

References cxxopts::values::standard_value< T >::get(), and bnssassembler::name().

```
        {
#ifdef CXXOPTS_NO_RTTI
            return static_cast<const values::standard_value<T>&>(*m_value).get();
#else
            return dynamic_cast<const values::standard_value<T>&>(*m_value).get();
#endif
        }
```

int cxxopts::OptionDetails::count () const `[inline]`

Definition at line 620 of file cxxopts.h.

```
        {
            return m_count;
        }
```

int cxxopts::OptionDetails::count () const `[inline]`

Definition at line 620 of file cxxopts.h.

```
        {
            return m_count;
        }
```

const String& cxxopts::OptionDetails::description () const `[inline]`

Definition at line 596 of file cxxopts.h.

```
        {
            return m_desc;
        }
```

const String& cxxopts::OptionDetails::description () const `[inline]`

Definition at line 596 of file cxxopts.h.

```
        {
            return m_desc;
```

```
599         }
```

bool cxxopts::OptionDetails::has_arg () const `[inline]`

Definition at line 602 of file cxxopts.h.

```
603         {
604             return m_value->has_arg();
605         }
```

bool cxxopts::OptionDetails::has_arg () const `[inline]`

Definition at line 602 of file cxxopts.h.

```
603         {
604             return m_value->has_arg();
605         }
```

void cxxopts::OptionDetails::parse (const std::string & *text*) `[inline]`

Definition at line 608 of file cxxopts.h.

```
609         {
610             m_value->parse(text);
611             ++m_count;
612         }
```

void cxxopts::OptionDetails::parse (const std::string & *text*) `[inline]`

Definition at line 608 of file cxxopts.h.

```
609         {
610             m_value->parse(text);
611             ++m_count;
612         }
```

void cxxopts::OptionDetails::parse_default () const `[inline]`

Definition at line 615 of file cxxopts.h.

```
615                                   {
616             m_value->parse();
617         }
```

void cxxopts::OptionDetails::parse_default () const `[inline]`

Definition at line 615 of file cxxopts.h.

```
615                                   {
616             m_value->parse();
617         }
```

const Value& cxxopts::OptionDetails::value () const `[inline]`

Definition at line 625 of file cxxopts.h.

```
625                                    {
626             return *m_value;
627         }
```

const Value& cxxopts::OptionDetails::value () const `[inline]`

Definition at line 625 of file cxxopts.h.

```
                                      {
            return *m_value;
        }
```

Member Data Documentation

int cxxopts::OptionDetails::m_count `[private]`

Definition at line 643 of file cxxopts.h.

String cxxopts::OptionDetails::m_desc `[private]`

Definition at line 641 of file cxxopts.h.

std::shared_ptr< const Value > cxxopts::OptionDetails::m_value `[private]`

Definition at line 642 of file cxxopts.h.

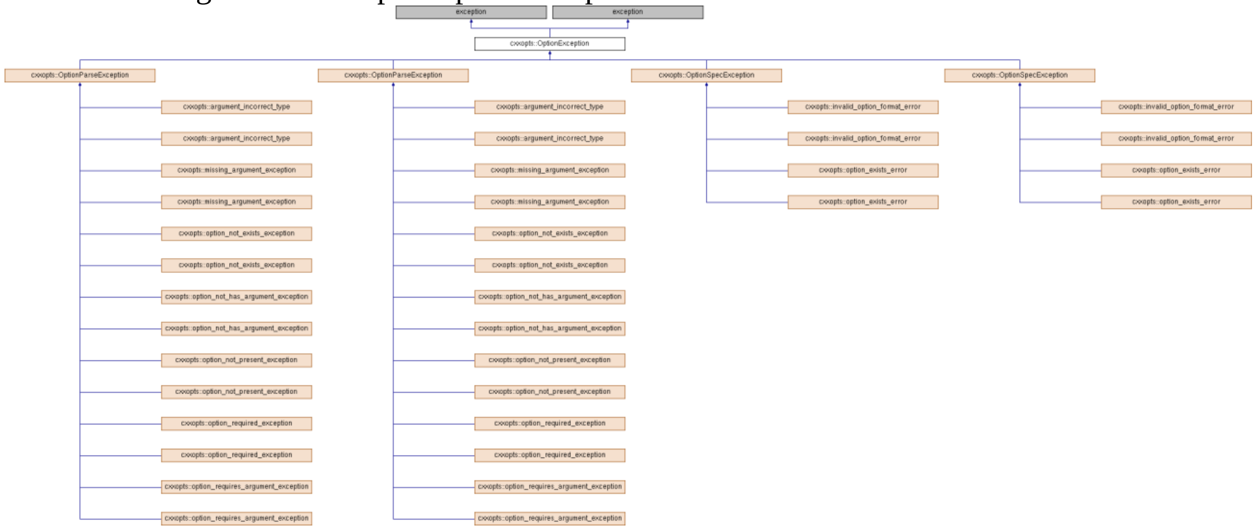
The documentation for this class was generated from the following file:

- Code/Assembler/Include/**cxxopts.h**

cxxopts::OptionException Class Reference

`#include <cxxopts.h>`

Inheritance diagram for cxxopts::OptionException:


exception
exception
cxxopts::OptionException
cxxopts::OptionParseException
cxxopts::OptionParseException
cxxopts::OptionSpecException
cxxopts::OptionSpecException
cxxopts::argument_incorrect_type
cxxopts::missing_argument_exception
cxxopts::option_not_exists_exception
cxxopts::option_not_has_argument_exception
cxxopts::option_not_present_exception
cxxopts::option_required_exception
cxxopts::option_requires_argument_exception
cxxopts::invalid_option_format_error
cxxopts::option_exists_error

Public Member Functions

- **OptionException** (const std::string &message)
- const char * **what** () const noexcept override
- **OptionException** (const std::string &message)
- const char * **what** () const noexcept override

Private Attributes

- std::string **m_message**

Detailed Description

Definition at line 277 of file cxxopts.h.

Constructor & Destructor Documentation

cxxopts::OptionException::OptionException (const std::string & *message*) `[inline]`, `[explicit]`

Definition at line 280 of file cxxopts.h.

```
          : m_message(message)
      {
      }
```

cxxopts::OptionException::OptionException (const std::string & *message*) `[inline]`, `[explicit]`

Definition at line 280 of file cxxopts.h.

```
          : m_message(message)
      {
      }
```

Member Function Documentation

const char* cxxopts::OptionException::what () const `[inline]`, `[override]`, `[noexcept]`

Definition at line 286 of file cxxopts.h.

```
                                         {
                return m_message.c_str();
            }
```

const char* cxxopts::OptionException::what () const `[inline]`, `[override]`, `[noexcept]`

Definition at line 286 of file cxxopts.h.

```
                                         {
                return m_message.c_str();
            }
```

Member Data Documentation

std::string cxxopts::OptionException::m_message `[private]`

Definition at line 291 of file cxxopts.h.

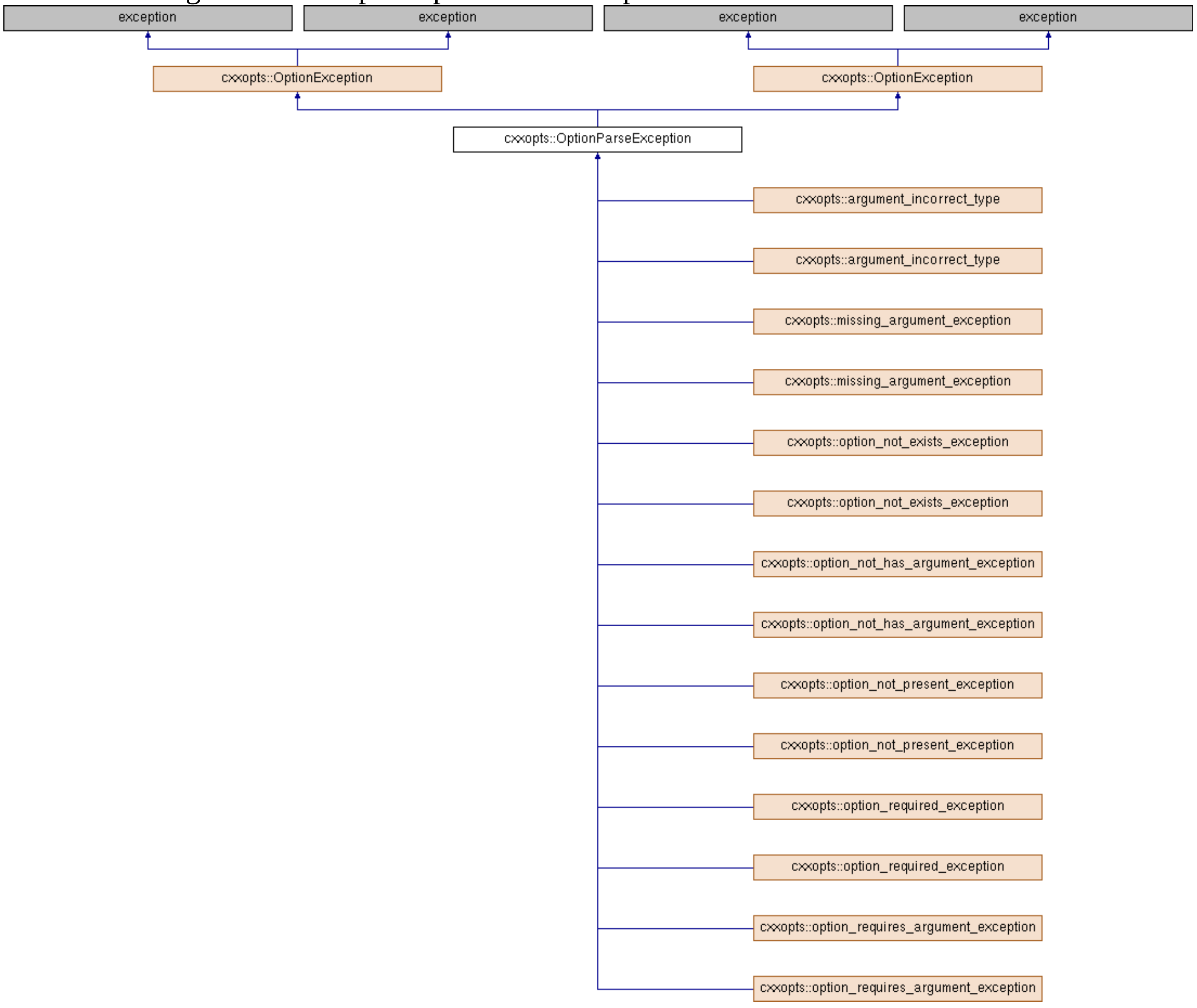
The documentation for this class was generated from the following file:

- Code/Assembler/Include/**cxxopts.h**

cxxopts::OptionParseException Class Reference

`#include <cxxopts.h>`

Inheritance diagram for cxxopts::OptionParseException:


exception
exception
exception
exception
cxxopts::OptionException
cxxopts::OptionException
cxxopts::OptionParseException
cxxopts::argument_incorrect_type
cxxopts::argument_incorrect_type
cxxopts::missing_argument_exception
cxxopts::missing_argument_exception
cxxopts::option_not_exists_exception
cxxopts::option_not_exists_exception
cxxopts::option_not_has_argument_exception
cxxopts::option_not_has_argument_exception
cxxopts::option_not_present_exception
cxxopts::option_not_present_exception
cxxopts::option_required_exception
cxxopts::option_required_exception
cxxopts::option_requires_argument_exception
cxxopts::option_requires_argument_exception

Public Member Functions

- **OptionParseException** (const std::string &message)
- **OptionParseException** (const std::string &message)

Detailed Description

Definition at line 304 of file cxxopts.h.

Constructor & Destructor Documentation

cxxopts::OptionParseException::OptionParseException (const std::string & *message*) `[inline]`

Definition at line 307 of file cxxopts.h.

```
            : OptionException(message)
        {
        }
```

cxxopts::OptionParseException::OptionParseException (const std::string & *message*)`[inline]`

Definition at line 307 of file cxxopts.h.

```
            : OptionException(message)
        {
        }
```

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

- Code/Assembler/Include/**cxxopts.h**

cxxopts::Options Class Reference

```
#include <cxxopts.h>
```

Public Member Functions

- **Options** (std::string program, std::string help_string="")
- **Options** & **positional_help** (std::string help_text)
- void **parse** (int &argc, char **&argv)
- **OptionAdder** **add_options** (std::string group="")
- void **add_option** (const std::string &group, const std::string &s, const std::string &l, std::string desc, std::shared_ptr< const **Value** > **value**, std::string arg_help)
- int **count** (const std::string &o) const
- const **OptionDetails** & **operator[]** (const std::string &option) const
- void **parse_positional** (std::string option)
- void **parse_positional** (std::vector< std::string > options)
- std::string **help** (const std::vector< std::string > &**groups**={ "" }) const
- std::vector< std::string > **groups** () const
- const **HelpGroupDetails** & **group_help** (const std::string &group) const
- **Options** (std::string program, std::string help_string="")
- **Options** & **positional_help** (std::string help_text)
- void **parse** (int &argc, char **&argv)
- **OptionAdder** **add_options** (std::string group="")
- void **add_option** (const std::string &group, const std::string &s, const std::string &l, std::string desc, std::shared_ptr< const **Value** > **value**, std::string arg_help)
- int **count** (const std::string &o) const
- const **OptionDetails** & **operator[]** (const std::string &option) const
- void **parse_positional** (std::string option)
- void **parse_positional** (std::vector< std::string > options)
- std::string **help** (const std::vector< std::string > &**groups**={ "" }) const
- std::vector< std::string > **groups** () const
- const **HelpGroupDetails** & **group_help** (const std::string &group) const

Private Member Functions

- void **add_one_option** (const std::string &option, std::shared_ptr< **OptionDetails** > details)
- bool **consume_positional** (std::string a)
- void **add_to_option** (const std::string &option, const std::string &arg)
- **String** **help_one_group** (const std::string &group) const
- void **generate_group_help** (**String** &result, const std::vector< std::string > &**groups**) const
- void **generate_all_groups_help** (**String** &result) const
- void **add_one_option** (const std::string &option, std::shared_ptr< **OptionDetails** > details)
- bool **consume_positional** (std::string a)
- void **add_to_option** (const std::string &option, const std::string &arg)
- **String** **help_one_group** (const std::string &group) const
- void **generate_group_help** (**String** &result, const std::vector< std::string > &**groups**) const
- void **generate_all_groups_help** (**String** &result) const

Static Private Member Functions

- static void **parse_option** (std::shared_ptr< **OptionDetails** > **value**, const std::string &name, const std::string &arg="")
- static void **checked_parse_arg** (int argc, char *argv[], int &current, std::shared_ptr< **OptionDetails** > **value**, const std::string &name)
- static void **parse_option** (std::shared_ptr< **OptionDetails** > **value**, const std::string &name, const std::string &arg="")

- static void **checked_parse_arg** (int argc, char *argv[], int &current, std::shared_ptr< **OptionDetails** > **value**, const std::string &name)

Private Attributes

- std::string **m_program**
- **String m_help_string**
- std::string **m_positional_help**
- std::map< std::string, std::shared_ptr< **OptionDetails** > > **m_options**
- std::vector< std::string > **m_positional**
- std::vector< std::string >::iterator **m_next_positional**
- std::unordered_set< std::string > **m_positional_set**
- std::map< std::string, **HelpGroupDetails** > **m_help**

Detailed Description

Definition at line 667 of file cxxopts.h.

Constructor & Destructor Documentation

cxxopts::Options::Options (std::string *program*, std::string *help_string* = "")[inline]

Definition at line 671 of file cxxopts.h.

```
            : m_program(std::move(program))
            , m_help_string(toLocalString(std::move(help_string)))
            , m_positional_help("positional parameters")
            , m_next_positional(m_positional.end())
        {
        }
```

cxxopts::Options::Options (std::string *program*, std::string *help_string* = "")[inline]

Definition at line 671 of file cxxopts.h.

```
            : m_program(std::move(program))
            , m_help_string(toLocalString(std::move(help_string)))
            , m_positional_help("positional parameters")
            , m_next_positional(m_positional.end())
        {
        }
```

Member Function Documentation

void cxxopts::Options::add_one_option (const std::string & *option*, std::shared_ptr< OptionDetails > *details*)[inline], [private]

Definition at line 1347 of file cxxopts.h.

Referenced by add_option(), and cxxopts::OptionAdder::OptionAdder().

```
    {
        auto in = m_options.emplace(option, details);

        if (!in.second)
        {
            throw option_exists_error(option);
        }
    }
```

void cxxopts::Options::add_one_option (const std::string & *option*, std::shared_ptr< OptionDetails > *details*)`[inline]`, `[private]`

void cxxopts::Options::add_option (const std::string & *group*, const std::string & *s*, const std::string & *l*, std::string *desc*, std::shared_ptr< const Value > *value*, std::string *arg_help*)`[inline]`

void cxxopts::Options::add_option (const std::string & *group*, const std::string & *s*, const std::string & *l*, std::string *desc*, std::shared_ptr< const Value > *value*, std::string *arg_help*)`[inline]`

Definition at line 1312 of file cxxopts.h.

References add_one_option(), cxxopts::HelpOptionDetails::has_arg, cxxopts::toLocalString(), and cxxopts::value().

Referenced by cxxopts::OptionAdder::OptionAdder(), and parse().

```
    {
        auto stringDesc = toLocalString(std::move(desc));
        auto option = std::make_shared<OptionDetails>(stringDesc, value);

        if (s.size() > 0)
        {
            add_one_option(s, option);
        }

        if (l.size() > 0)
        {
            add_one_option(l, option);
        }

        //add the help details
        auto& options = m_help[group];

        options.options.emplace_back(HelpOptionDetails{ s, l, stringDesc,
            value->has_arg(),
            value->has_default(), value->get_default_value(),
            value->has_implicit(), value->get_implicit_value(),
            std::move(arg_help),
            value->is_container() });
    }
```

OptionAdder cxxopts::Options::add_options (std::string *group* = `""`)`[inline]`

Definition at line 981 of file cxxopts.h.

Referenced by cxxopts::OptionAdder::OptionAdder(), bnssemulator::CommandLineHelper::parse(), and bnssassembler::CommandLineHelper::parse().

```
    {
        return OptionAdder(*this, std::move(group));
    }
```

OptionAdder cxxopts::Options::add_options (std::string *group* = `""`)`[inline]`

void cxxopts::Options::add_to_option (const std::string & *option*, const std::string & *arg*)`[inline]`, `[private]`

void cxxopts::Options::add_to_option (const std::string & *option*, const std::string & *arg*)`[inline]`, `[private]`

Definition at line 1091 of file cxxopts.h.

Referenced by cxxopts::OptionAdder::OptionAdder().

```
    {
        auto iter = m_options.find(option);

        if (iter == m_options.end())
        {
            throw option_not_exists_exception(option);
        }

        parse_option(iter->second, option, arg);
    }
```

void cxxopts::Options::checked_parse_arg (int *argc*, char * *argv*[], int & *current*, std::shared_ptr< OptionDetails > *value*, const std::string & *name*)`[inline]`, `[static]`, `[private]`

Definition at line 1057 of file cxxopts.h.

Referenced by cxxopts::OptionAdder::OptionAdder(), and parse_option().

```
    {
        if (current + 1 >= argc)
        {
            if (value->value().has_implicit())
            {
                parse_option(value, name,
value->value().get_implicit_value());
            }
            else
            {
                throw missing_argument_exception(name);
            }
        }
        else
        {
            if (argv[current + 1][0] == '-' && value->value().has_implicit())
            {
                parse_option(value, name,
value->value().get_implicit_value());
            }
            else
            {
                parse_option(value, name, argv[current + 1]);
                ++current;
            }
        }
    }
```

static void cxxopts::Options::checked_parse_arg (int *argc*, char * *argv*[], int & *current*, std::shared_ptr< OptionDetails > *value*, const std::string & *name*)`[inline]`, `[static]`, `[private]`

bool cxxopts::Options::consume_positional (std::string *a*)`[inline]`, `[private]`

bool cxxopts::Options::consume_positional (std::string *a*)`[inline]`, `[private]`

Definition at line 1104 of file cxxopts.h.

Referenced by cxxopts::OptionAdder::OptionAdder().

```
    {
        while (m_next_positional != m_positional.end())
        {
            auto iter = m_options.find(*m_next_positional);
            if (iter != m_options.end())
            {
                if (!iter->second->value().is_container())
                {
                    if (iter->second->count() == 0)
                    {
                        add_to_option(*m_next_positional, a);
                        ++m_next_positional;
                        return true;
                    }
                    else
                    {
                        ++m_next_positional;
                        continue;
                    }
                }
                else
                {
                    add_to_option(*m_next_positional, a);
                    return true;
                }
            }
            ++m_next_positional;
        }

        return false;
    }
```

int cxxopts::Options::count (const std::string & *o*) const`[inline]`

Definition at line 708 of file cxxopts.h.

Referenced by cxxopts::check_required(), cxxopts::OptionAdder::OptionAdder(), bnssassembler::CommandLineHelper::parse(), and bnssemulator::CommandLineHelper::parse().

```
        {
            auto iter = m_options.find(o);
            if (iter == m_options.end())
            {
                return 0;
            }

            return iter->second->count();
        }
```

int cxxopts::Options::count (const std::string & *o*) const`[inline]`

Definition at line 708 of file cxxopts.h.

```
        {
            auto iter = m_options.find(o);
            if (iter == m_options.end())
            {
                return 0;
            }

            return iter->second->count();
        }
```

void cxxopts::Options::generate_all_groups_help (String & *result*) const`[inline]`, `[private]`

Definition at line 1453 of file cxxopts.h.

Referenced by cxxopts::OptionAdder::OptionAdder().

```
    {
        std::vector<std::string> all_groups;
        all_groups.reserve(m_help.size());

        for (auto& group : m_help)
        {
            all_groups.push_back(group.first);
        }

        generate_group_help(result, all_groups);
    }
```

void cxxopts::Options::generate_all_groups_help (String & *result*) const`[inline]`, `[private]`

void cxxopts::Options::generate_group_help (String & *result*, const std::vector< std::string > & *groups*) const`[inline]`, `[private]`

void cxxopts::Options::generate_group_help (String & *result*, const std::vector< std::string > & *groups*) const`[inline]`, `[private]`

Definition at line 1432 of file cxxopts.h.

References cxxopts::empty().

Referenced by help_one_group(), and cxxopts::OptionAdder::OptionAdder().

```
    {
        for (size_t i = 0; i != print_groups.size(); ++i)
        {
            const String& group_help_text = help_one_group(print_groups[i]);
            if (empty(group_help_text))
            {
                continue;
            }
            result += group_help_text;
            if (i < print_groups.size() - 1)
            {
                result += '\n';
            }
        }
    }
```

const HelpGroupDetails & cxxopts::Options::group_help (const std::string & *group*) const`[inline]`

Definition at line 1509 of file cxxopts.h.

Referenced by cxxopts::OptionAdder::OptionAdder().

```
    {
        return m_help.at(group);
    }
```

const HelpGroupDetails& cxxopts::Options::group_help (const std::string & *group*) const`[inline]`

std::vector< std::string > cxxopts::Options::groups () const`[inline]`

Definition at line 1491 of file cxxopts.h.

Referenced by cxxopts::OptionAdder::OptionAdder().

```
    {
        std::vector<std::string> g;

        std::transform(
            m_help.begin(),
            m_help.end(),
            std::back_inserter(g),
            [](const std::map<std::string, HelpGroupDetails>::value_type&
pair)
        {
            return pair.first;
        }
        );

        return g;
    }
```

std::vector<std::string> cxxopts::Options::groups () const`[inline]`

std::string cxxopts::Options::help (const std::vector< std::string > & *groups* = `{ "" }`) const`[inline]`

Definition at line 1467 of file cxxopts.h.

References cxxopts::toLocalString(), and cxxopts::toUTF8String().

Referenced by cxxopts::OptionAdder::OptionAdder(), bnssemulator::CommandLineHelper::parse(), and bnssassembler::CommandLineHelper::parse().

```
    {
        String result = m_help_string + "\nUsage:\n  " +
            toLocalString(m_program) + " [OPTION...]";

        if (m_positional.size() > 0) {
            result += " " + toLocalString(m_positional_help);
        }

        result += "\n\n";

        if (help_groups.size() == 0)
        {
            generate_all_groups_help(result);
        }
        else
        {
            generate_group_help(result, help_groups);
        }

        return toUTF8String(result);
    }
```

std::string cxxopts::Options::help (const std::vector< std::string > & *groups* = `{ "" }`) const`[inline]`

String cxxopts::Options::help_one_group (const std::string & *group*) const`[inline]`, `[private]`

String cxxopts::Options::help_one_group (const std::string & *group*) const`[inline]`, `[private]`

Definition at line 1361 of file cxxopts.h.

References cxxopts::anonymous_namespace{cxxopts.h}::format_description(), cxxopts::anonymous_namespace{cxxopts.h}::format_option(), generate_group_help(), cxxopts::anonymous_namespace{cxxopts.h}::OPTION_DESC_GAP, cxxopts::stringLength(), and cxxopts::toLocalString().

Referenced by cxxopts::OptionAdder::OptionAdder().

```
    {
        typedef std::vector<std::pair<String, String>> OptionHelp;

        auto group = m_help.find(g);
        if (group == m_help.end())
        {
            return "";
        }

        OptionHelp format;

        size_t longest = 0;

        String result;

        if (!g.empty())
        {
            result += toLocalString(" " + g + " options:\n");
        }

        for (const auto& o : group->second.options)
        {
            if (o.is_container && m_positional_set.find(o.l) != m_positional_set.end())
            {
                continue;
            }

            auto s = format_option(o);
            longest = std::max(longest, stringLength(s));
            format.push_back(std::make_pair(s, String()));
        }

        longest = std::min(longest, static_cast<size_t>(OPTION_LONGEST));

        //widest allowed description
        auto allowed = size_t{ 76 } -longest - OPTION_DESC_GAP;

        auto fiter = format.begin();
        for (const auto& o : group->second.options)
        {
            if (o.is_container && m_positional_set.find(o.l) != m_positional_set.end())
            {
                continue;
            }

            auto d = format_description(o, longest + OPTION_DESC_GAP, allowed);

            result += fiter->first;
            if (stringLength(fiter->first) > longest)
            {
                result += '\n';
                result += toLocalString(std::string(longest + OPTION_DESC_GAP, ' '));
            }
            else
            {
                result += toLocalString(std::string(longest + OPTION_DESC_GAP -
                    stringLength(fiter->first),
                    ' '));
            }
            result += d;
            result += '\n';

            ++fiter;
        }

        return result;
    }
```

const OptionDetails& cxxopts::Options::operator[] (const std::string & *option*) const`[inline]`

Definition at line 720 of file cxxopts.h.

References bnssassembler::name().

```
        {
            auto iter = m_options.find(option);

            if (iter == m_options.end())
            {
                throw option_not_present_exception(option);
            }

            return *iter->second;
        }
```

const OptionDetails& cxxopts::Options::operator[] (const std::string & *option*) const`[inline]`

Definition at line 720 of file cxxopts.h.

References bnssassembler::name().

```
        {
            auto iter = m_options.find(option);

            if (iter == m_options.end())
            {
                throw option_not_present_exception(option);
            }

            return *iter->second;
        }
```

void cxxopts::Options::parse (int & *argc*, char **& *argv*)`[inline]`

void cxxopts::Options::parse (int & *argc*, char **& *argv*)`[inline]`

Definition at line 1153 of file cxxopts.h.

References add_option(), bnssassembler::name(), and cxxopts::anonymous_namespace{cxxopts.h}::option_matcher().

Referenced by cxxopts::OptionAdder::OptionAdder(), bnssemulator::CommandLineHelper::parse(), and bnssassembler::CommandLineHelper::parse().

```
    {
        int current = 1;

        int nextKeep = 1;

        bool consume_remaining = false;

        while (current != argc)
        {
            if (strcmp(argv[current], "--") == 0)
            {
                consume_remaining = true;
                ++current;
                break;
            }

            std::match_results<const char*> result;
            std::regex_match(argv[current], result, option_matcher);

            if (result.empty())
            {
                //not a flag

                //if true is returned here then it was consumed, otherwise it
is
                //ignored
                if (consume_positional(argv[current]))
                {
                }
                else
                {
                    argv[nextKeep] = argv[current];
                    ++nextKeep;
                }
                //if we return from here then it was parsed successfully, so
continue
            }
            else
            {
                //short or long option?
                if (result[4].length() != 0)
                {
                    const std::string& s = result[4];

                    for (std::size_t i = 0; i != s.size(); ++i)
                    {
                        std::string name(1, s[i]);
                        auto iter = m_options.find(name);

                        if (iter == m_options.end())
                        {
                            throw option_not_exists_exception(name);
                        }

                        auto value = iter->second;

                        //if no argument then just add it
                        if (!value->has_arg())
                        {
                            parse_option(value, name);
                        }
                        else
                        {
                            //it must be the last argument
                            if (i + 1 == s.size())
                            {
                                checked_parse_arg(argc, argv, current, value, name);
                            }
                            else if (value->value().has_implicit())
                            {
                                parse_option(value, name, value->value().get_implicit_value());
                            }
                            else
                            {
                                //error
                                throw option_requires_argument_exception(name);
                            }
                        }
                    }
                }
                else if (result[1].length() != 0)
                {
                    const std::string& name = result[1];

                    auto iter = m_options.find(name);

                    if (iter == m_options.end())
                    {
                        throw option_not_exists_exception(name);
                    }

                    auto opt = iter->second;

                    //equals provided for long option?
                    if (result[3].length() != 0)
                    {
                        //parse the option given

                        //but if it doesn't take an argument, this is an error
                        if (!opt->has_arg())
                        {
                            throw option_not_has_argument_exception(name, result[3]);
                        }

                        parse_option(opt, name, result[3]);
                    }
                    else
                    {
                        if (opt->has_arg())
                        {
                            //parse the next argument
                            checked_parse_arg(argc, argv, current, opt, name);
                        }
                        else
                        {
                            //parse with empty argument
                            parse_option(opt, name);
                        }
                    }
                }

            }

            ++current;
        }

        for (auto& opt : m_options)
        {
            auto& detail = opt.second;
            auto& value = detail->value();

            if (!detail->count() && value.has_default()) {
                detail->parse_default();
            }
        }

        if (consume_remaining)
        {
            while (current < argc)
            {
                if (!consume_positional(argv[current])) {
                    break;
                }
                ++current;
            }

            //adjust argv for any that couldn't be swallowed
            while (current != argc) {
                argv[nextKeep] = argv[current];
                ++nextKeep;
                ++current;
            }
        }

        argc = nextKeep;

    }
```

static void cxxopts::Options::parse_option (std::shared_ptr< OptionDetails > *value*, const std::string & *name*, const std::string & *arg* = `""`)`[inline]`, `[static]`, `[private]`

void cxxopts::Options::parse_option (std::shared_ptr< OptionDetails > *value*, const std::string & *name*, const std::string & *arg* = `""`)`[inline]`, `[static]`, `[private]`

Definition at line 1046 of file cxxopts.h.

References checked_parse_arg().

Referenced by cxxopts::OptionAdder::operator()(), and cxxopts::OptionAdder::OptionAdder().

```
    {
        value->parse(arg);
    }
```

void cxxopts::Options::parse_positional (std::string *option*)`[inline]`

Definition at line 1138 of file cxxopts.h.

Referenced by cxxopts::OptionAdder::OptionAdder().

```
    {
        parse_positional(std::vector<std::string>{option});
    }
```

void cxxopts::Options::parse_positional (std::string *option*) [inline]

void cxxopts::Options::parse_positional (std::vector< std::string > *options*) [inline]

void cxxopts::Options::parse_positional (std::vector< std::string > *options*) [inline]

Definition at line 1144 of file cxxopts.h.

```
    {
        m_positional = std::move(options);
        m_next_positional = m_positional.begin();

        m_positional_set.insert(m_positional.begin(), m_positional.end());
    }
```

Options& cxxopts::Options::positional_help (std::string *help_text*) [inline]

Definition at line 681 of file cxxopts.h.

References cxxopts::value().

```
        {
            m_positional_help = std::move(help_text);
            return *this;
        }
```

Options& cxxopts::Options::positional_help (std::string *help_text*) [inline]

Definition at line 681 of file cxxopts.h.

References cxxopts::value().

```
        {
            m_positional_help = std::move(help_text);
            return *this;
        }
```

Member Data Documentation

std::map< std::string, HelpGroupDetails > cxxopts::Options::m_help [private]

Definition at line 817 of file cxxopts.h.

String cxxopts::Options::m_help_string [private]

Definition at line 808 of file cxxopts.h.

std::vector< std::string >::iterator cxxopts::Options::m_next_positional [private]

Definition at line 813 of file cxxopts.h.

std::map< std::string, std::shared_ptr< OptionDetails > > cxxopts::Options::m_options [private]

Definition at line 811 of file cxxopts.h.

std::vector< std::string > cxxopts::Options::m_positional `[private]`

Definition at line 812 of file cxxopts.h.

std::string cxxopts::Options::m_positional_help `[private]`

Definition at line 809 of file cxxopts.h.

std::unordered_set< std::string > cxxopts::Options::m_positional_set `[private]`

Definition at line 814 of file cxxopts.h.

std::string cxxopts::Options::m_program `[private]`

Definition at line 807 of file cxxopts.h.

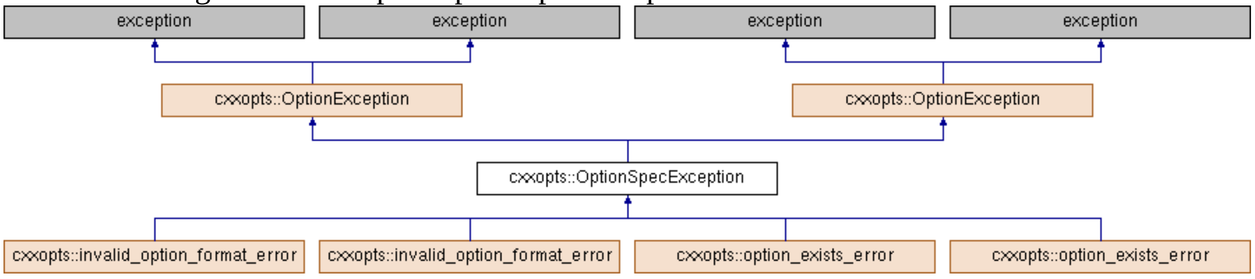
The documentation for this class was generated from the following file:

- Code/Assembler/Include/**cxxopts.h**

cxxopts::OptionSpecException Class Reference

`#include <cxxopts.h>`

Inheritance diagram for cxxopts::OptionSpecException:


exception
exception
exception
exception
cxxopts::OptionException
cxxopts::OptionException
cxxopts::OptionSpecException
cxxopts::invalid_option_format_error
cxxopts::invalid_option_format_error
cxxopts::option_exists_error
cxxopts::option_exists_error

Public Member Functions

- **OptionSpecException** (const std::string &message)
- **OptionSpecException** (const std::string &message)

Detailed Description

Definition at line 294 of file cxxopts.h.

Constructor & Destructor Documentation

cxxopts::OptionSpecException::OptionSpecException (const std::string & *message*) `[inline]`

Definition at line 298 of file cxxopts.h.

```
              : OptionException(message)
          {
          }
```

cxxopts::OptionSpecException::OptionSpecException (const std::string & *message*) `[inline]`

Definition at line 298 of file cxxopts.h.

```
              : OptionException(message)
          {
          }
```

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

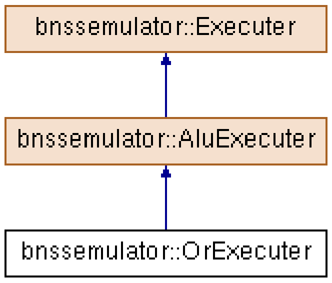
- Code/Assembler/Include/**cxxopts.h**

bnssemulator::OrExecuter Class Reference

Class representing the executer for the or instruction.

`#include <OrExecuter.h>`

Inheritance diagram for bnssemulator::OrExecuter:



Protected Member Functions

- void **execute** (**Register** &dst, const **Register** &lhs, const **Register** &rhs) const override
  Executes the ALU instruction.

Additional Inherited Members

Detailed Description

Class representing the executer for the or instruction.

Definition at line 10 of file OrExecuter.h.

Member Function Documentation

void bnssemulator::OrExecuter::execute (Register & *dst*, const Register & *lhs*, const Register & *rhs*) const `[override]`, `[protected]`, `[virtual]`

Executes the ALU instruction.

Parameters:

dst	Reference to the destination register
lhs	Left operand register
rhs	Right operand register

Implements **bnssemulator::AluExecuter** (*p.102*).

Definition at line 5 of file OrExecuter.cpp.

```
{
            dst = lhs | rhs;
        }
```

The documentation for this class was generated from the following files:

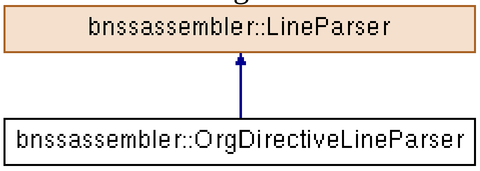
- Code/Emulator/Include/**OrExecuter.h**
- Code/Emulator/Source/**OrExecuter.cpp**

bnssassembler::OrgDirectiveLineParser Class Reference

Class representing a line parser for the origin directive.

`#include <OrgDirectiveLineParser.h>`

Inheritance diagram for bnssassembler::OrgDirectiveLineParser:


bnssassembler::LineParser
bnssassembler::OrgDirectiveLineParser

Protected Member Functions

- std::shared_ptr< **Token** > **parse** (const std::string &line, size_t line_number, std::string initial_line) const override
  Parses one line of the file. Does not call the next parser in chain.

Additional Inherited Members

Detailed Description

Class representing a line parser for the origin directive.

Definition at line 10 of file OrgDirectiveLineParser.h.

Member Function Documentation

std::shared_ptr< Token > bnssassembler::OrgDirectiveLineParser::parse (const std::string & *line*, size_t *line_number*, std::string *initial_line*) const `[override]`, `[protected]`, `[virtual]`

Parses one line of the file. Does not call the next parser in chain.

Parameters:

line	Line to parse
line_number	Number of the line that is parsed
initial_line	Initial line that is parsed

Returns:

Extracted token from line or nullptr if the parser failed parsing the line

Exceptions:

Throws	if the parser failed and identified the error

Implements **bnssassembler::LineParser** (*p.257*).

Definition at line 9 of file OrgDirectiveLineParser.cpp.

References bnssassembler::ExpressionBuilder::build(), bnssassembler::CONSTANT_TERM, and bnssassembler::ORG_DIRECTIVE.

```
    9
{
   10          static std::regex regex("[[:space:]]*" + ORG_DIRECTIVE + "(" +
CONSTANT_TERM + ")");
   11
   12          if (!regex_match(line, regex)) {
```

```
            return nullptr;
        }

        auto expression_string = regex_replace(line, regex, "$1");
        auto expression = ExpressionBuilder::build(expression_string);
        return std::make_shared<OrgDirectiveToken>(expression, line_number,
initial_line);
    }
```

The documentation for this class was generated from the following files:

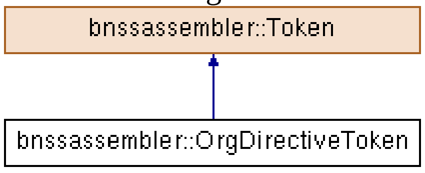
- Code/Assembler/Include/**OrgDirectiveLineParser.h**
- Code/Assembler/Source/**OrgDirectiveLineParser.cpp**

bnssassembler::OrgDirectiveToken Class Reference

Class representing the origin directive token.

`#include <OrgDirectiveToken.h>`

Inheritance diagram for bnssassembler::OrgDirectiveToken:



Public Member Functions

- **OrgDirectiveToken** (**MicroRiscExpression** expression, size_t line_number, std::string **line**) noexcept
  *Constructs an **OrgDirectiveToken** object.*
- void **resolveSymbolDefinitions** (std::unordered_set< **SymbolDefinition** > symbols) noexcept override
  Resolves symbol definitions in a token.
- void **firstPass** (**FirstPassData** &data) const override
  Executes the first pass over the token.
- void **secondPass** (**SecondPassData** &data) const override
  Executes the second pass over the token.
- void **resolveSymbolTable** (const **SymbolTable** &symbol_table) noexcept override
  Resolves the symbols from the symbol table and updates relocation info.
- void **resolveImports** (std::unordered_set< std::string > imported_symbols) noexcept override
  Resolves the imported symbols and updates relocation info.

Private Attributes

- **MicroRiscExpression expression_**

Detailed Description

Class representing the origin directive token.

Definition at line 11 of file OrgDirectiveToken.h.

Constructor & Destructor Documentation

bnssassembler::OrgDirectiveToken::OrgDirectiveToken (MicroRiscExpression *expression*, size_t *line_number*, std::string *line*) `[noexcept]`

Constructs an **OrgDirectiveToken** object.

Parameters:

expression	**Expression** of this origin directive
line_number	Number of the line where this directive is located
line	Line where this directive is located

Definition at line 7 of file OrgDirectiveToken.cpp.

```
    7 : Token(line_number, line), expression_(expression) {}
```

Member Function Documentation

void bnssassembler::OrgDirectiveToken::firstPass (FirstPassData & *data*) const`[override]`, `[virtual]`

Executes the first pass over the token.

Parameters:

data	**Data** that the token will modify

Implements **bnssassembler::Token** (*p.510*).

Definition at line 15 of file OrgDirectiveToken.cpp.

```
                                                            {
        // Do nothing
    }
```

void bnssassembler::OrgDirectiveToken::resolveImports (std::unordered_set< std::string > *imported_symbols*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the imported symbols and updates relocation info.

Parameters:

imported_symbols	Collection of imported symbols

Reimplemented from **bnssassembler::Token** (*p.511*).

Definition at line 31 of file OrgDirectiveToken.cpp.

References expression_, and bnssassembler::MicroRiscExpression::resolveImports().

```
{
        expression_.resolveImports(imported_symbols);
    }
```

void bnssassembler::OrgDirectiveToken::resolveSymbolDefinitions (std::unordered_set< SymbolDefinition > *symbols*)`[override]`, `[virtual]`, `[noexcept]`

Resolves symbol definitions in a token.

Parameters:

symbols	Vector od symbol definitions that should be resolved

Reimplemented from **bnssassembler::Token** (*p.511*).

Definition at line 9 of file OrgDirectiveToken.cpp.

```
{
        for (auto &symbol : symbols) {
            expression_.setValue(symbol.name(), symbol.expression());
        }
    }
```

void bnssassembler::OrgDirectiveToken::resolveSymbolTable (const SymbolTable & *symbol_table*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the symbols from the symbol table and updates relocation info.

Parameters:

symbol_table	**Symbol** table

Reimplemented from **bnssassembler::Token** (*p.511*).

Definition at line 27 of file OrgDirectiveToken.cpp.

References expression_, and bnssassembler::MicroRiscExpression::resolveSymbolTable().

```
{
        expression_.resolveSymbolTable(symbol_table);
    }
```

void bnssassembler::OrgDirectiveToken::secondPass (SecondPassData & *data*) const`[override]`, `[virtual]`

Executes the second pass over the token.

Parameters:

data	**Data** that the token will modify

Implements **bnssassembler::Token** (*p.512*).

Definition at line 19 of file OrgDirectiveToken.cpp.

References expression_, bnssassembler::MicroRiscExpression::generateRelocations(), bnssassembler::SecondPassData::org(), and bnssassembler::MicroRiscExpression::value().

```
                                                            {
        if (!expression_.generateRelocations().empty()) {
            throw MessageException("ORG directive expression can not have labels");
        }

        data.org(static_cast<uint32_t>(expression_.value()));
    }
```

Member Data Documentation

MicroRiscExpression bnssassembler::OrgDirectiveToken::expression_`[private]`

Definition at line 27 of file OrgDirectiveToken.h.

Referenced by resolveImports(), resolveSymbolTable(), and secondPass().

The documentation for this class was generated from the following files:

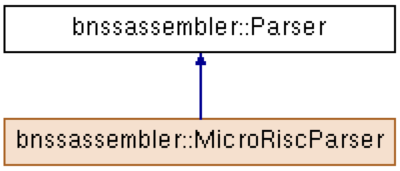
- Code/Assembler/Include/**OrgDirectiveToken.h**
- Code/Assembler/Source/**OrgDirectiveToken.cpp**

bnssassembler::Parser Class Reference

Abstract class representing a text parser.

`#include <Parser.h>`

Inheritance diagram for bnssassembler::Parser:



Public Member Functions

- std::vector< std::shared_ptr< **Token** > > **parse** (std::vector< std::string > body) const
  Parses the File into tokens.
- virtual **~Parser** ()=default
  Virtual destructor needed for polymorphic class.

Protected Member Functions

- virtual std::vector< std::string > **oneLineCommentDelimiters** () const noexcept=0
  Returns all strings that start the comment to the end of the line.
- virtual std::vector< std::string > **labelDelimiters** () const noexcept=0
  Returns all strings that end the label at the start of the line.
- virtual bool **isEnd** (std::string line) const noexcept=0
  Checks if the parser should stop parsing the file.
- virtual std::shared_ptr< **LineParser** > **chain** () const noexcept=0
  *Returns the first **LineParser** in chain.*

Detailed Description

Abstract class representing a text parser.

Definition at line 14 of file Parser.h.

Constructor & Destructor Documentation

virtual bnssassembler::Parser::~Parser () `[virtual]`, `[default]`

Virtual destructor needed for polymorphic class.

Member Function Documentation

virtual std::shared_ptr<LineParser> bnssassembler::Parser::chain () const `[protected]`, `[pure virtual]`, `[noexcept]`

Returns the first **LineParser** in chain.

Returns:

Pointer to the first parser

Implemented in **bnssassembler::MicroRiscParser** (*p.283*).

Referenced by parse().

virtual bool bnssassembler::Parser::isEnd (std::string *line*) const`[protected]`, `[pure virtual]`, `[noexcept]`

Checks if the parser should stop parsing the file.

Parameters:

line	Line to check

Implemented in **bnssassembler::MicroRiscParser** (*p.283*).

Referenced by parse().

virtual std::vector<std::string> bnssassembler::Parser::labelDelimiters () const`[protected]`, `[pure virtual]`, `[noexcept]`

Returns all strings that end the label at the start of the line.

Returns:

Vector of such strings

Implemented in **bnssassembler::MicroRiscParser** (*p.284*).

Referenced by parse().

virtual std::vector<std::string> bnssassembler::Parser::oneLineCommentDelimiters () const`[protected]`, `[pure virtual]`, `[noexcept]`

Returns all strings that start the comment to the end of the line.

Returns:

Vector of such strings

Implemented in **bnssassembler::MicroRiscParser** (*p.284*).

Referenced by parse().

std::vector< std::shared_ptr< Token > > bnssassembler::Parser::parse (std::vector< std::string > *body*) const

Parses the File into tokens.

Parameters:

body	Collection of all lines in the file

Returns:

Collection of tokens

Exceptions:

Throws	if the file can not be parsed

Definition at line 53 of file Parser.cpp.

References chain(), bnssassembler::extractLabel(), bnssassembler::StringHelper::isAllWhiteSpace(), isEnd(), labelDelimiters(), bnssassembler::MessageException::message(), oneLineCommentDelimiters(), and bnssassembler::stripComment().

Referenced by main().

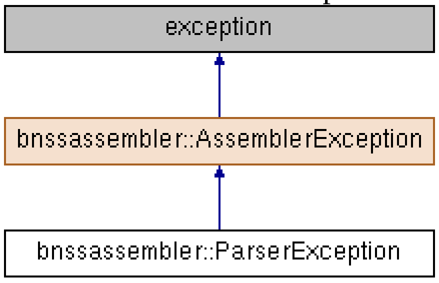
```
    {
        std::vector<std::shared_ptr<Token>> ret;

        for (size_t i = 0; i < body.size(); i++) {
            auto &line = body[i];
            auto initial_line = line;

            try {
                // Strip the comments
                stripComment(line, oneLineCommentDelimiters());

                // Extract the label (if it exists) and insert it into the Token vector
                auto label = extractLabel(line, labelDelimiters());
                if (label != "") {
                    ret.push_back(std::make_shared<LabelToken>(label, i + 1, initial_line));
                }

                // Skip if the line contains no data
                if (StringHelper::isAllWhiteSpace(line)) {
                    continue;
                }

                // Check if the file should still be parsed
                if (isEnd(line)) {
                    break;
                }

                // Parse the line
                auto token = chain()->tryParse(line, i + 1, initial_line);
                if (token == nullptr) {
                    throw MessageException("The line can not be parsed");
                }

                ret.push_back(token);
            }
            catch (MessageException &e) {
                throw ParserException(i + 1, initial_line, e.message());
            }
        }

        return ret;
    }
```

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**Parser.h**
- Code/Assembler/Source/**Parser.cpp**

bnssassembler::ParserException Class Reference

Represents an exception that happend during the parsing of the file.
`#include <ParserException.h>`
Inheritance diagram for bnssassembler::ParserException:


exception
bnssassembler::AssemblerException
bnssassembler::ParserException

Public Member Functions

- **ParserException** (size_t line_number, std::string line, std::string specific_message) noexcept
  *Constructs a **ParserException** object.*

Protected Member Functions

- std::string **messageBody** () const noexcept override
  Returns the actual message body of the exception.

Private Attributes

- std::string **specific_message_**

Detailed Description

Represents an exception that happend during the parsing of the file.

Definition at line 10 of file ParserException.h.

Constructor & Destructor Documentation

bnssassembler::ParserException::ParserException (size_t *line_number*, std::string *line*, std::string *specific_message*) `[noexcept]`

Constructs a **ParserException** object.

Parameters:

line_number	Number of the line where the error happened
line	Line where the error happened
specific_message	Specific message about the error that happened

Definition at line 8 of file ParserException.cpp.

```
    8 : AssemblerException(line_number, line), specific_message_(specific_message) {}
```

Member Function Documentation

std::string bnssassembler::ParserException::messageBody () const `[override]`, `[protected]`, `[virtual]`, `[noexcept]`

Returns the actual message body of the exception.

Implements **bnssassembler::AssemblerException** (*p.109*).

Definition at line 4 of file ParserException.cpp.

References specific_message_.

```
                                                  {
        return "Error during the parsing phase:\n" + specific_message_;
    }
```

Member Data Documentation

std::string bnssassembler::ParserException::specific_message_ `[private]`

Definition at line 22 of file ParserException.h.

Referenced by messageBody().

The documentation for this class was generated from the following files:

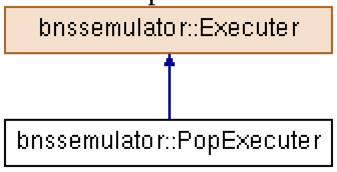
- Code/Assembler/Include/**ParserException.h**
- Code/Assembler/Source/**ParserException.cpp**

bnssemulator::PopExecuter Class Reference

Class representing the executer for the pop instruction.

`#include <PopExecuter.h>`

Inheritance diagram for bnssemulator::PopExecuter:



Public Member Functions

- void **execute** (**InstructionBitField** instruction, **Context** &context) const override
  Executes the instruction.

Detailed Description

Class representing the executer for the pop instruction.

Definition at line 10 of file PopExecuter.h.

Member Function Documentation

void bnssemulator::PopExecuter::execute (InstructionBitField *instruction*, Context & *context*) const `[override]`, `[virtual]`

Executes the instruction.

Parameters:

instruction	Instruction
context	**Processor** context

Implements **bnssemulator::Executer** (*p.163*).

Definition at line 5 of file PopExecuter.cpp.

References bnssemulator::Context::getRegister(), bnssemulator::Context::popFromStack(), and bnssemulator::InstructionBitField::register0.

```
{
        auto &reg = context.getRegister(instruction.register0);
        reg = context.popFromStack();
    }
```

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**PopExecuter.h**
- Code/Emulator/Source/**PopExecuter.cpp**

bnssemulator::Processor Class Reference

Class representing the processor.

`#include <Processor.h>`

Classes

- struct **ProcessorStaticData**

Static Public Member Functions

- static void **executeProgram** (**Context** &context)
  Executes the program using the given processor context.

Static Private Member Functions

- static void **executeInstruction** (**Context** &context)
  Executes one instruction of the program using the given processor context.
- static **ProcessorStaticData** & **staticData** () noexcept

---

Detailed Description

Class representing the processor.

Definition at line 13 of file Processor.h.

Member Function Documentation

void bnssemulator::Processor::executeInstruction (Context & *context*) `[static]`, `[private]`

Executes one instruction of the program using the given processor context.

Parameters:

context	Reference to the processor context

Definition at line 76 of file Processor.cpp.

References bnssemulator::Context::getInstruction(), bnssemulator::Processor::ProcessorStaticData::map, bnssemulator::opcode(), staticData(), and bnssemulator::StringHelper::toHexString().

Referenced by executeProgram().

```
                                                    {
        auto instruction = context.getInstruction();
        if (staticData().map.count(opcode(instruction)) == 0) {
            throw MessageException("Invalid operation code: " +
StringHelper::toHexString(instruction.operation_code));
        }

        auto &executer = staticData().map.at(opcode(instruction));
        executer->execute(instruction, context);
    }
```

void bnssemulator::Processor::executeProgram (Context & *context*) `[static]`

Executes the program using the given processor context.

Parameters:

context	Reference to the processor context

Definition at line 35 of file Processor.cpp.

References executeInstruction(), bnssemulator::Context::finishProgram(), bnssemulator::Context::hasCharacters(), bnssemulator::Context::insideInterrupt(), bnssemulator::Context::jumpToErrorInterrupt(), bnssemulator::Context::jumpToKeyboardInterrupt(), bnssemulator::Context::jumpToTimerInterrupt(), bnssemulator::TimerListener::listen(), bnssemulator::KeyboardListener::listen(), bnssemulator::Context::programFinished(), and bnssemulator::Context::timerTriggered().

Referenced by main().

```
                                                    {
        std::thread keyboard_listener(KeyboardListener::listen, &context);
        std::thread timer_thread(TimerListener::listen, &context);

        try {
            while (!context.programFinished()) {
                try {
                    executeInstruction(context);
                }
                catch (...) {
                    if (context.insideInterrupt()) {
                        throw;
                    }

                    context.jumpToErrorInterrupt();
                }

                if (context.hasCharacters() && !context.insideInterrupt()) {
                    context.jumpToKeyboardInterrupt();
                }

                if (context.timerTriggered() && !context.insideInterrupt()) {
                    context.jumpToTimerInterrupt();
                }
            }
        }
        catch (...) {
            context.finishProgram();
            keyboard_listener.join();
            timer_thread.join();
            throw;
        }

        keyboard_listener.join();
        timer_thread.join();
    }
```

Processor::ProcessorStaticData & bnssemulator::Processor::staticData () `[static]`, `[private]`, `[noexcept]`

Definition at line 114 of file Processor.cpp.

Referenced by executeInstruction().

```
                                                                  {
        static ProcessorStaticData static_data;
        return static_data;
    }
```

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**Processor.h**
- Code/Emulator/Source/**Processor.cpp**

bnssemulator::Processor::ProcessorStaticData Struct Reference

Public Member Functions

- **ProcessorStaticData** ()

Public Attributes

- std::unordered_map< **InstructionCode**, std::shared_ptr< **Executer** > > **map**

Detailed Description

Definition at line 27 of file Processor.h.

Constructor & Destructor Documentation

bnssemulator::Processor::ProcessorStaticData::ProcessorStaticData ()

Definition at line 86 of file Processor.cpp.

References bnssemulator::ADD, bnssemulator::AND, bnssemulator::ASL, bnssemulator::ASR, bnssemulator::CALL, bnssemulator::DIV, bnssemulator::INT, bnssemulator::JGEZ, bnssemulator::JGZ, bnssemulator::JLEZ, bnssemulator::JLZ, bnssemulator::JMP, bnssemulator::JNZ, bnssemulator::JZ, bnssemulator::LOAD, bnssemulator::MOD, bnssemulator::MUL, bnssemulator::NOT, bnssemulator::OR, bnssemulator::POP, bnssemulator::PUSH, bnssemulator::RET, bnssemulator::STORE, bnssemulator::SUB, and bnssemulator::XOR.

```
   86                                                          {
   87           map[INT] = std::make_shared<IntExecuter>();
   88           map[RET] = std::make_shared<RetExecuter>();
   89           map[JMP] = std::make_shared<JmpExecuter>();
   90           map[CALL] = std::make_shared<CallExecuter>();
   91           map[JZ] = std::make_shared<JzExecuter>();
   92           map[JNZ] = std::make_shared<JnzExecuter>();
   93           map[JGZ] = std::make_shared<JgzExecuter>();
   94           map[JGEZ] = std::make_shared<JgezExecuter>();
   95           map[JLZ] = std::make_shared<JlzExecuter>();
   96           map[JLEZ] = std::make_shared<JlezExecuter>();
   97           map[LOAD] = std::make_shared<LoadExecuter>();
   98           map[STORE] = std::make_shared<StoreExecuter>();
   99           map[PUSH] = std::make_shared<PushExecuter>();
  100           map[POP] = std::make_shared<PopExecuter>();
  101           map[ADD] = std::make_shared<AddExecuter>();
  102           map[SUB] = std::make_shared<SubtractExecuter>();
  103           map[MUL] = std::make_shared<MultiplyExecuter>();
  104           map[DIV] = std::make_shared<DivideExecuter>();
  105           map[MOD] = std::make_shared<ModuloExecuter>();
  106           map[AND] = std::make_shared<AndExecuter>();
  107           map[OR] = std::make_shared<OrExecuter>();
  108           map[XOR] = std::make_shared<XorExecuter>();
  109           map[ASL] = std::make_shared<AslExecuter>();
  110           map[ASR] = std::make_shared<AsrExecuter>();
  111           map[NOT] = std::make_shared<NotExecuter>();
  112       }
```

Member Data Documentation

std::unordered_map<InstructionCode, std::shared_ptr<Executer> > bnssemulator::Processor::ProcessorStaticData::map

Definition at line 28 of file Processor.h.

Referenced by bnssemulator::Processor::executeInstruction().

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

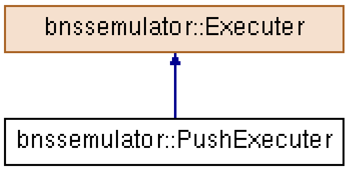
- Code/Emulator/Include/**Processor.h**
- Code/Emulator/Source/**Processor.cpp**

bnssemulator::PushExecuter Class Reference

Class representing the executer for the push instruction.

`#include <PushExecuter.h>`

Inheritance diagram for bnssemulator::PushExecuter:



Public Member Functions

- void **execute** (**InstructionBitField** instruction, **Context** &context) const override
  Executes the instruction.

Detailed Description

Class representing the executer for the push instruction.

Definition at line 10 of file PushExecuter.h.

Member Function Documentation

void bnssemulator::PushExecuter::execute (InstructionBitField *instruction*, Context & *context*) const `[override]`, `[virtual]`

Executes the instruction.

Parameters:

instruction	Instruction
context	**Processor** context

Implements **bnssemulator::Executer** (*p.163*).

Definition at line 5 of file PushExecuter.cpp.

References bnssemulator::Context::getRegister(), bnssemulator::Context::pushToStack(), and bnssemulator::InstructionBitField::register0.

```
{
            auto &reg = context.getRegister(instruction.register0);
            context.pushToStack(reg);
        }
```

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**PushExecuter.h**
- Code/Emulator/Source/**PushExecuter.cpp**

bnssemulator::Register Class Reference

Class representing the register.
`#include <Register.h>`

Public Member Functions

- **Register** ()=default
  *Constructs a **Register** object.*
- int32_t **value** () const noexcept
  Gets the value of the register.
- void **value** (int32_t value) noexcept
  Sets the value of the register.
- bool **negativeFlag** () const noexcept
  Gets the negative flag of the register.
- bool **zeroFlag** () const noexcept
  Gets the zero flag of the register.
- bool **carryFlag** () const noexcept
  Gets the Carry flag of the register.
- bool **overflowFlag** () const noexcept
  Gets the overflow flag of the register.
- **Register** (int32_t **value**) noexcept
  *Constructs a **Register** object.*
- **Register** (int32_t **value**, bool carry_flag, bool overflow_flag) noexcept
  *Constructs a **Register** object.*
- **operator int32_t** () const noexcept
- **Register operator-** () const noexcept
- **Register operator~** () const noexcept
- **Register** & **operator+=** (const **Register** &reg) noexcept
- **Register** & **operator-=** (const **Register** &reg) noexcept
- **Register** & **operator*=** (const **Register** &reg) noexcept
- **Register** & **operator/=** (const **Register** &reg) noexcept
- **Register** & **operator%=** (const **Register** &reg) noexcept
- **Register** & **operator &=** (const **Register** &reg) noexcept
- **Register** & **operator|=** (const **Register** &reg) noexcept
- **Register** & **operator^=** (const **Register** &reg) noexcept
- **Register** & **operator<<=** (const **Register** &reg) noexcept
- **Register** & **operator>>=** (const **Register** &reg) noexcept

Private Attributes

- int32_t **value_** = 0
- bool **carry_flag_** = false
- bool **overflow_flag_** = false

Friends

- **Register operator+** (const **Register** &lhs, const **Register** &rhs) noexcept
- **Register operator-** (const **Register** &lhs, const **Register** &rhs) noexcept
- **Register operator*** (const **Register** &lhs, const **Register** &rhs) noexcept
- **Register operator/** (const **Register** &lhs, const **Register** &rhs) noexcept
- **Register operator%** (const **Register** &lhs, const **Register** &rhs) noexcept
- **Register operator &** (const **Register** &lhs, const **Register** &rhs) noexcept
- **Register operator|** (const **Register** &lhs, const **Register** &rhs) noexcept

- **Register operator^** (const **Register** &lhs, const **Register** &rhs) noexcept
- **Register operator<<** (const **Register** &lhs, const **Register** &rhs) noexcept
- **Register operator>>** (const **Register** &lhs, const **Register** &rhs) noexcept

Detailed Description

Class representing the register.

Definition at line 10 of file Register.h.

Constructor & Destructor Documentation

bnssemulator::Register::Register () `[default]`

Constructs a **Register** object.

Referenced by bnssemulator::operator &(), bnssemulator::operator%(), bnssemulator::operator*(), bnssemulator::operator+(), operator-(), bnssemulator::operator/(), bnssemulator::operator<<(), bnssemulator::operator>>(), bnssemulator::operator^(), bnssemulator::operator|(), and operator~().

bnssemulator::Register::Register (int32_t *value*) `[noexcept]`

Constructs a **Register** object.

Parameters:

value	Starting value of the register

Definition at line 31 of file Register.cpp.

```
31 : value_(value) {}
```

bnssemulator::Register::Register (int32_t *value*, bool *carry_flag*, bool *overflow_flag*) `[noexcept]`

Constructs a **Register** object.

Parameters:

value	Starting value of the register
carry_flag	Starting carry flag of the register
overflow_flag	Starting overflow flag of the register

Definition at line 37 of file Register.cpp.

```
37 : value_(value), carry_flag_(carry_flag), overflow_flag_(overflow_flag) {}
```

Member Function Documentation

bool bnssemulator::Register::carryFlag () const `[noexcept]`

Gets the Carry flag of the register.

Returns:

Carry flag

Definition at line 23 of file Register.cpp.

References carry_flag_.

```
                                        {
        return carry_flag_;
    }
```

bool bnssemulator::Register::negativeFlag () const`[noexcept]`

Gets the negative flag of the register.

Returns:

Negative flag

Definition at line 15 of file Register.cpp.

References value_.

```
                                           {
        return value_ < 0;
    }
```

Register& bnssemulator::Register::operator&= (const Register & *reg*)`[noexcept]`

Referenced by operator%=().

bnssemulator::Register::operator int32_t () const`[noexcept]`

Definition at line 33 of file Register.cpp.

References value_.

```
                                          {
        return value_;
    }
```

Register & bnssemulator::Register::operator%= (const Register & *reg*)`[noexcept]`

Definition at line 67 of file Register.cpp.

References operator &=().

```
                                                            {
        *this = *this % reg;
        return *this;
    }
```

Register & bnssemulator::Register::operator*= (const Register & *reg*)`[noexcept]`

Definition at line 57 of file Register.cpp.

```
                                                            {
        *this = *this * reg;
        return *this;
    }
```

Register & bnssemulator::Register::operator+= (const Register & *reg*)`[noexcept]`

Definition at line 47 of file Register.cpp.

```
                                                            {
        *this = *this + reg;
```

```
    49          return *this;
    50      }
```

Register bnssemulator::Register::operator- () const `[noexcept]`

Definition at line 39 of file Register.cpp.

References Register(), and value_.

```
    39                                          {
    40          return Register(-value_);
    41      }
```

Register & bnssemulator::Register::operator-= (const Register & *reg*) `[noexcept]`

Definition at line 52 of file Register.cpp.

```
    52                                                                  {
    53          *this = *this - reg;
    54          return *this;
    55      }
```

Register & bnssemulator::Register::operator/= (const Register & *reg*) `[noexcept]`

Definition at line 62 of file Register.cpp.

```
    62                                                                  {
    63          *this = *this / reg;
    64          return *this;
    65      }
```

Register & bnssemulator::Register::operator<<= (const Register & *reg*) `[noexcept]`

Definition at line 87 of file Register.cpp.

```
    87                                                                   {
    88          *this = *this << reg;
    89          return *this;
    90      }
```

Register & bnssemulator::Register::operator>>= (const Register & *reg*) `[noexcept]`

Definition at line 92 of file Register.cpp.

```
    92                                                                   {
    93          *this = *this >> reg;
    94          return *this;
    95      }
```

Register & bnssemulator::Register::operator^= (const Register & *reg*) `[noexcept]`

Definition at line 82 of file Register.cpp.

```
    82                                                                  {
    83          *this = *this ^ reg;
    84          return *this;
    85      }
```

Register & bnssemulator::Register::operator|= (const Register & *reg*) `[noexcept]`

Definition at line 77 of file Register.cpp.

```
    77                                                                  {
    78          *this = *this | reg;
    79          return *this;
    80      }
```

Register bnssemulator::Register::operator~ () const `[noexcept]`

Definition at line 43 of file Register.cpp.

References Register(), and value_.

```
   43                                         {
   44         return Register(~value_);
   45     }
```

bool bnssemulator::Register::overflowFlag () const `[noexcept]`

Gets the overflow flag of the register.

Returns:

Overflow flag

Definition at line 27 of file Register.cpp.

References overflow_flag_.

```
   27                                        {
   28         return overflow_flag_;
   29     }
```

int32_t bnssemulator::Register::value () const `[noexcept]`

Gets the value of the register.

Returns:

Value of the register

Definition at line 7 of file Register.cpp.

References value_.

Referenced by bnssemulator::Context::Context(), bnssemulator::LoadExecuter::execute(), and value().

```
    7                                  {
    8         return value_;
    9     }
```

void bnssemulator::Register::value (int32_t *value*) `[noexcept]`

Sets the value of the register.

Parameters:

value	Value of the register

Definition at line 11 of file Register.cpp.

References value(), and value_.

```
   11                                        {
   12         value_ = value;
   13     }
```

bool bnssemulator::Register::zeroFlag () const `[noexcept]`

Gets the zero flag of the register.

Returns:

Zero flag

Definition at line 19 of file Register.cpp.

References value_.

```
                                    {
        return value_ == 0;
    }
```

Friends And Related Function Documentation

Register operator& (const Register & *lhs*, const Register & *rhs*)[friend]

Definition at line 131 of file Register.cpp.

```
                                                                        {
        return Register(lhs.value_ & rhs.value_);
    }
```

Register operator% (const Register & *lhs*, const Register & *rhs*)[friend]

Definition at line 127 of file Register.cpp.

```
                                                                        {
        return Register(lhs.value_ % rhs.value_);
    }
```

Register operator* (const Register & *lhs*, const Register & *rhs*)[friend]

Definition at line 112 of file Register.cpp.

```
                                                                        {
        auto result_value = static_cast<int64_t>(lhs.value_) + static_cast<int64_t>(rhs.value_);
        auto left = static_cast<bool>(lhs.value_ & INT32_MIN);
        auto right = static_cast<bool>(rhs.value_ & INT32_MIN);
        auto result = static_cast<bool>(result_value & INT32_MIN);

        auto flags = ((result_value & TOP_32_BITS) != 0) || (!left && !right && result);

        return Register(static_cast<int32_t>(result_value), flags, flags);
    }
```

Register operator+ (const Register & *lhs*, const Register & *rhs*)[friend]

Definition at line 97 of file Register.cpp.

```
                                                                        {
        auto result_value = static_cast<int64_t>(lhs.value_) + static_cast<int64_t>(rhs.value_);
        auto left = static_cast<bool>(lhs.value_ & INT32_MIN);
        auto right = static_cast<bool>(rhs.value_ & INT32_MIN);
        auto result = static_cast<bool>(result_value & INT32_MIN);

        auto flags = (left && right && !result) || (!left && !right && result);

        return Register(static_cast<int32_t>(result_value), flags, flags);
    }
```

Register operator- (const Register & *lhs*, const Register & *rhs*)[friend]

Definition at line 108 of file Register.cpp.

```
                                                                        {
```

```
        return lhs + -rhs;
    }
```

Register operator/ (const Register & *lhs*, const Register & *rhs*) `[friend]`

Definition at line 123 of file Register.cpp.

```
                                                                   {
        return Register(lhs.value_ / rhs.value_);
    }
```

Register operator<< (const Register & *lhs*, const Register & *rhs*) `[friend]`

Definition at line 143 of file Register.cpp.

```
                                                                    {
        auto shift = rhs.value_ % 32;
        auto result = lhs.value_ << shift;

        auto carry = (result & TOP_32_BITS) != 0;

        return Register(result, carry, false);
    }
```

Register operator>> (const Register & *lhs*, const Register & *rhs*) `[friend]`

Definition at line 152 of file Register.cpp.

```
                                                                    {
        auto shift = rhs.value_ % 32;
        auto result = lhs.value_ >> shift;

        auto back = result << shift;
        auto carry = lhs.value_ != back;

        return Register(result, carry, false);
    }
```

Register operator^ (const Register & *lhs*, const Register & *rhs*) `[friend]`

Definition at line 139 of file Register.cpp.

```
                                                                   {
        return Register(lhs.value_ ^ rhs.value_);
    }
```

Register operator| (const Register & *lhs*, const Register & *rhs*) `[friend]`

Definition at line 135 of file Register.cpp.

```
                                                                   {
        return Register(lhs.value_ | rhs.value_);
    }
```

Member Data Documentation

bool bnssemulator::Register::carry_flag_ = false `[private]`

Definition at line 104 of file Register.h.

Referenced by carryFlag().

bool bnssemulator::Register::overflow_flag_ = false `[private]`

Definition at line 105 of file Register.h.

Referenced by overflowFlag().

int32_t bnssemulator::Register::value_ = 0 `[private]`

Definition at line 102 of file Register.h.

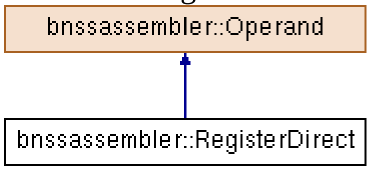
Referenced by negativeFlag(), operator int32_t(), operator-(), bnssemulator::operator<<(), bnssemulator::operator>>(), operator~(), value(), and zeroFlag().

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**Register.h**
- Code/Emulator/Source/**Register.cpp**

bnssassembler::RegisterDirect Class Reference

Class representing the register direct operand.
`#include <RegisterDirect.h>`
Inheritance diagram for bnssassembler::RegisterDirect:



Public Member Functions

- **RegisterDirect** (**Register** reg) noexcept
  Constructs a register direct object.
- void **packToInstruction** (**InstructionBitFieldUnion** &instruction, **uint32_t** &second_word, std::list< **RelocationRecord** > &relocations) const override
  Packs the operand into the instruction.
- **AddressMode addressMode** () const noexcept override
  Gets the address mode of the operand.

Private Attributes

- **Register reg_**

Detailed Description

Class representing the register direct operand.

Definition at line 11 of file RegisterDirect.h.

Constructor & Destructor Documentation

bnssassembler::RegisterDirect::RegisterDirect (Register *reg*) `[explicit]`, `[noexcept]`

Constructs a register direct object.

Parameters:

reg	Register

Definition at line 5 of file RegisterDirect.cpp.

```
    5 : reg_(reg) {}
```

Member Function Documentation

AddressMode bnssassembler::RegisterDirect::addressMode () const `[override]`, `[virtual]`, `[noexcept]`

Gets the address mode of the operand.

Returns:

Address mode of the operand

Implements **bnssassembler::Operand** (*p.302*).

Definition at line 23 of file RegisterDirect.cpp.

References bnssassembler::REGISTER_DIRECT.

```
                                                          {
        return REGISTER_DIRECT;
    }
```

void bnssassembler::RegisterDirect::packToInstruction (InstructionBitFieldUnion & *instruction*, uint32_t & *second_word*, std::list< RelocationRecord > & *relocations*) const[override], [virtual]

Packs the operand into the instruction.

Parameters:

instruction	Reference to the first word of the instruction containing the instruction info
second_word	Reference to the second word of the instruction containing the address/value/displacement
relocations	Reference to the list of relocation records

Implements **bnssassembler::Operand** (*p.303*).

Definition at line 7 of file RegisterDirect.cpp.

References bnssassembler::InstructionBitFieldUnion::bit_field, bnssassembler::NONE, reg_, bnssassembler::InstructionBitField::register0, bnssassembler::InstructionBitField::register1, and bnssassembler::InstructionBitField::register2.

```
{
        if (instruction.bit_field.register0 == NONE) {
            instruction.bit_field.register0 = reg_;
            return;
        }

        if (instruction.bit_field.register1 == NONE) {
            instruction.bit_field.register1 = reg_;
            return;
        }

        if (instruction.bit_field.register2 == NONE) {
            instruction.bit_field.register2 = reg_;
        }
    }
```

Member Data Documentation

Register bnssassembler::RegisterDirect::reg_[private]

Definition at line 22 of file RegisterDirect.h.

Referenced by packToInstruction().

The documentation for this class was generated from the following files:

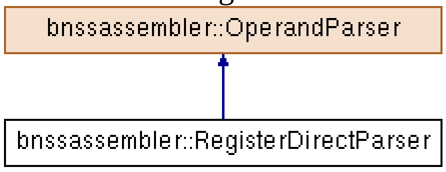
- Code/Assembler/Include/**RegisterDirect.h**
- Code/Assembler/Source/**RegisterDirect.cpp**

bnssassembler::RegisterDirectParser Class Reference

Class representing the parser for the register direct operand.

`#include <RegisterDirectParser.h>`

Inheritance diagram for bnssassembler::RegisterDirectParser:


bnssassembler::OperandParser
bnssassembler::RegisterDirectParser

Protected Member Functions

- std::shared_ptr< **Operand** > **parse** (std::string str) const override
  Parses one operand. Does not call the next parser if it fails.

Additional Inherited Members

Detailed Description

Class representing the parser for the register direct operand.

Definition at line 10 of file RegisterDirectParser.h.

Member Function Documentation

std::shared_ptr< Operand > bnssassembler::RegisterDirectParser::parse (std::string *str*) const [override], [protected], [virtual]

Parses one operand. Does not call the next parser if it fails.

Parameters:

str	**Operand** which should be parsed

Returns:

Pointer to the operand or nullptr, if the parser failed parsing

Exceptions:

Throws	if the parser fails but identifies the error

Implements **bnssassembler::OperandParser** (*p.306*).

Definition at line 7 of file RegisterDirectParser.cpp.

References bnssassembler::RegisterParser::parse().

```
                                                                        {
        try {
            return
std::make_shared<RegisterDirect>(RegisterParser::parse(str));
        }
        catch (...) {
            return nullptr;
        }
    }
```

The documentation for this class was generated from the following files:

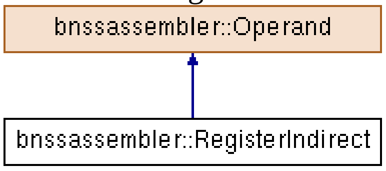
- Code/Assembler/Include/**RegisterDirectParser.h**
- Code/Assembler/Source/**RegisterDirectParser.cpp**

bnssassembler::RegisterIndirect Class Reference

Class representing the register indirect operand.

`#include <RegisterIndirect.h>`

Inheritance diagram for bnssassembler::RegisterIndirect:



Public Member Functions

- **RegisterIndirect** (**Register** reg) noexcept
  *Constructs a **RegisterIndirect** object.*
- void **packToInstruction** (**InstructionBitFieldUnion** &instruction, **uint32_t** &second_word, std::list< **RelocationRecord** > &relocations) const override
  Packs the operand into the instruction.
- **AddressMode addressMode** () const noexcept override
  Gets the address mode of the operand.

Private Attributes

- **Register reg_**

Detailed Description

Class representing the register indirect operand.

Definition at line 11 of file RegisterIndirect.h.

Constructor & Destructor Documentation

bnssassembler::RegisterIndirect::RegisterIndirect (Register *reg*) `[explicit]`, `[noexcept]`

Constructs a **RegisterIndirect** object.

Parameters:

reg	Register

Definition at line 5 of file RegisterIndirect.cpp.

```
    5 : reg_(reg) {}
```

Member Function Documentation

AddressMode bnssassembler::RegisterIndirect::addressMode () const `[override]`, `[virtual]`, `[noexcept]`

Gets the address mode of the operand.

Returns:

Address mode of the operand

Implements **bnssassembler::Operand** (*p.302*).

Definition at line 25 of file RegisterIndirect.cpp.

References bnssassembler::REGISTER_INDIRECT.

```
                                                              {
        return REGISTER_INDIRECT;
    }
```

void bnssassembler::RegisterIndirect::packToInstruction (InstructionBitFieldUnion & *instruction*, uint32_t & *second_word*, std::list< RelocationRecord > & *relocations*) const`[override]`, `[virtual]`

Packs the operand into the instruction.

Parameters:

instruction	Reference to the first word of the instruction containing the instruction info
second_word	Reference to the second word of the instruction containing the address/value/displacement
relocations	Reference to the list of relocation records

Implements **bnssassembler::Operand** (*p.303*).

Definition at line 7 of file RegisterIndirect.cpp.

References bnssassembler::InstructionBitField::address_mode, bnssassembler::InstructionBitFieldUnion::bit_field, bnssassembler::NONE, reg_, bnssassembler::InstructionBitField::register0, bnssassembler::InstructionBitField::register1, bnssassembler::InstructionBitField::register2, and bnssassembler::REGISTER_INDIRECT.

```

{
        instruction.bit_field.address_mode = REGISTER_INDIRECT;

        if (instruction.bit_field.register0 == NONE) {
            instruction.bit_field.register0 = reg_;
            return;
        }

        if (instruction.bit_field.register1 == NONE) {
            instruction.bit_field.register1 = reg_;
            return;
        }

        if (instruction.bit_field.register2 == NONE) {
            instruction.bit_field.register2 = reg_;
        }
    }
```

Member Data Documentation

Register bnssassembler::RegisterIndirect::reg_`[private]`

Definition at line 22 of file RegisterIndirect.h.

Referenced by packToInstruction().

The documentation for this class was generated from the following files:

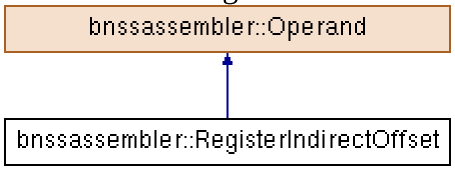
- Code/Assembler/Include/**RegisterIndirect.h**
- Code/Assembler/Source/**RegisterIndirect.cpp**

bnssassembler::RegisterIndirectOffset Class Reference

Class representing the register indirect operand with offset.

`#include <RegisterIndirectOffset.h>`

Inheritance diagram for bnssassembler::RegisterIndirectOffset:



Public Member Functions

- **RegisterIndirectOffset** (**Register** reg, **MicroRiscExpression** offset_or_address, bool absolute)
  *Constructs a **RegisterIndirectOffset** object.*
- void **packToInstruction** (**InstructionBitFieldUnion** &instruction, **uint32_t** &second_word, std::list< **RelocationRecord** > &relocations) const override
  Packs the operand into the instruction.
- void **resolveSymbols** (std::unordered_set< **SymbolDefinition** > symbols) noexcept override
  Resolves the defined symbols in the expressions.
- void **resolveSymbolTable** (const **SymbolTable** &symbol_table) noexcept override
  Resolves the symbols from the symbol table and updates the relocation info.
- void **resolveImports** (std::unordered_set< std::string > imported_symbols) noexcept override
  Resolves the imported symbols and updates the relocation info.
- void **resolveCurrentPcSymbol** (size_t section_index, size_t offset) noexcept override
  Resolves the current PC symbol and sets the relocation info.
- **AddressMode addressMode** () const noexcept override
  Gets the address mode of the operand.

Private Attributes

- **Register reg_**
- **MicroRiscExpression offset_or_address_**
- bool **absolute_** = false
- size_t **pc_section_index_** = 0
- size_t **pc_offset_** = 0

Detailed Description

Class representing the register indirect operand with offset.

Definition at line 12 of file RegisterIndirectOffset.h.

Constructor & Destructor Documentation

bnssassembler::RegisterIndirectOffset::RegisterIndirectOffset (Register *reg*, MicroRiscExpression *offset_or_address*, bool *absolute*)

Constructs a **RegisterIndirectOffset** object.

Parameters:

reg	Register
offset_or_address	Offset or absolute address of the operand
absolute	Whether the address is absolute

Definition at line 6 of file RegisterIndirectOffset.cpp.

References bnssassembler::PC.

```
: reg_(reg), offset_or_address_(offset_or_address), absolute_(absolute) {
        if (absolute && reg != PC) {
            throw MessageException("Only PC relative address can be absolute");
        }
    }
```

Member Function Documentation

AddressMode bnssassembler::RegisterIndirectOffset::addressMode () const`[override]`, `[virtual]`, `[noexcept]`

Gets the address mode of the operand.

Returns:

Address mode of the operand

Implements **bnssassembler::Operand** (*p.302*).

Definition at line 78 of file RegisterIndirectOffset.cpp.

References bnssassembler::REGISTER_INDIRECT_OFFSET.

```
                                                                    {
        return REGISTER_INDIRECT_OFFSET;
    }
```

void bnssassembler::RegisterIndirectOffset::packToInstruction (InstructionBitFieldUnion & *instruction*, uint32_t & *second_word*, std::list< RelocationRecord > & *relocations*) const`[override]`, `[virtual]`

Packs the operand into the instruction.

Parameters:

instruction	Reference to the first word of the instruction containing the instruction info
second_word	Reference to the second word of the instruction containing the address/value/displacement
relocations	Reference to the list of relocation records

Implements **bnssassembler::Operand** (*p.303*).

Definition at line 12 of file RegisterIndirectOffset.cpp.

References absolute_, bnssassembler::InstructionBitField::address_mode, bnssassembler::InstructionBitFieldUnion::bit_field, bnssassembler::MicroRiscExpression::generateRelocations(), bnssassembler::NONE, offset_or_address_, pc_offset_, pc_section_index_, reg_, bnssassembler::InstructionBitField::register0, bnssassembler::InstructionBitField::register1, bnssassembler::InstructionBitField::register2, bnssassembler::REGISTER_INDIRECT_OFFSET, and bnssassembler::MicroRiscExpression::value().

```
{
        instruction.bit_field.address_mode = REGISTER_INDIRECT_OFFSET;
        second_word = offset_or_address_.value();
        if (absolute_) {
```

```
   16             second_word -= 4;
   17             auto rels = offset_or_address_.generateRelocations();
   18             if (rels.empty()) {
   19                 throw MessageException("PC Relative address must contain at
least one label");
   20             }
   21 
   22             auto found_same_section = false;
   23 
   24             for (auto &rel : rels) {
   25                 if (rel.sectionIndex() == pc_section_index_) {
   26                     found_same_section = true;
   27                     second_word -= pc_offset_ + 4;
   28                     rels.remove(rel);
   29                     break;
   30                 }
   31             }
   32 
   33             if (!found_same_section) {
   34                 rels.front().absolute(false);
   35             }
   36 
   37             relocations.splice(relocations.end(), rels);
   38         }
   39         else {
   40             relocations.splice(relocations.end(),
offset_or_address_.generateRelocations());
   41         }
   42 
   43         if (instruction.bit_field.register0 == NONE) {
   44             instruction.bit_field.register0 = reg_;
   45             return;
   46         }
   47 
   48         if (instruction.bit_field.register1 == NONE) {
   49             instruction.bit_field.register1 = reg_;
   50             return;
   51         }
   52 
   53         if (instruction.bit_field.register2 == NONE) {
   54             instruction.bit_field.register2 = reg_;
   55         }
   56     }
```

void bnssassembler::RegisterIndirectOffset::resolveCurrentPcSymbol (size_t *section_index*, size_t *offset*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the current PC symbol and sets the relocation info.

Parameters:

section_index	Current PC section
offset	PC address in relation to the current section beginning

Reimplemented from **bnssassembler::Operand** (*p.303*).

Definition at line 72 of file RegisterIndirectOffset.cpp.

References offset_or_address_, pc_offset_, pc_section_index_, and bnssassembler::MicroRiscExpression::resolveCurrentPcSymbol().

```
   72 
{
   73         offset_or_address_.resolveCurrentPcSymbol(section_index, offset);
   74         pc_section_index_ = section_index;
   75         pc_offset_ = offset;
   76     }
```

void bnssassembler::RegisterIndirectOffset::resolveImports (std::unordered_set< std::string > *imported_symbols*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the imported symbols and updates the relocation info.

Parameters:

imported_symbols	Collection of imported symbols

Reimplemented from **bnssassembler::Operand** (*p.303*).

Definition at line 68 of file RegisterIndirectOffset.cpp.

References offset_or_address_, and bnssassembler::MicroRiscExpression::resolveImports().

```
{
        offset_or_address_.resolveImports(imported_symbols);
    }
```

void bnssassembler::RegisterIndirectOffset::resolveSymbols (std::unordered_set< SymbolDefinition > *symbols*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the defined symbols in the expressions.

Parameters:

symbols	Collection of symbol definitions

Reimplemented from **bnssassembler::Operand** (*p.303*).

Definition at line 58 of file RegisterIndirectOffset.cpp.

References offset_or_address_, and bnssassembler::MicroRiscExpression::setValue().

```
{
        for (auto &symbol : symbols) {
            offset_or_address_.setValue(symbol.name(), symbol.expression());
        }
    }
```

void bnssassembler::RegisterIndirectOffset::resolveSymbolTable (const SymbolTable & *symbol_table*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the symbols from the symbol table and updates the relocation info.

Parameters:

symbol_table	**Symbol** table

Reimplemented from **bnssassembler::Operand** (*p.304*).

Definition at line 64 of file RegisterIndirectOffset.cpp.

References offset_or_address_, and bnssassembler::MicroRiscExpression::resolveSymbolTable().

```
{
        offset_or_address_.resolveSymbolTable(symbol_table);
    }
```

Member Data Documentation

bool bnssassembler::RegisterIndirectOffset::absolute_ = false `[private]`

Definition at line 31 of file RegisterIndirectOffset.h.

Referenced by packToInstruction().

MicroRiscExpression bnssassembler::RegisterIndirectOffset::offset_or_address_ `[private]`

Definition at line 30 of file RegisterIndirectOffset.h.

Referenced by packToInstruction(), resolveCurrentPcSymbol(), resolveImports(), resolveSymbols(), and resolveSymbolTable().

size_t bnssassembler::RegisterIndirectOffset::pc_offset_ = 0 `[private]`

Definition at line 33 of file RegisterIndirectOffset.h.

Referenced by packToInstruction(), and resolveCurrentPcSymbol().

size_t bnssassembler::RegisterIndirectOffset::pc_section_index_ = 0 `[private]`

Definition at line 32 of file RegisterIndirectOffset.h.

Referenced by packToInstruction(), and resolveCurrentPcSymbol().

Register bnssassembler::RegisterIndirectOffset::reg_ `[private]`

Definition at line 29 of file RegisterIndirectOffset.h.

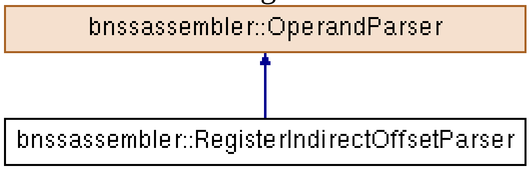
Referenced by packToInstruction().

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**RegisterIndirectOffset.h**
- Code/Assembler/Source/**RegisterIndirectOffset.cpp**

bnssassembler::RegisterIndirectOffsetParser Class Reference

Class representing the parser for the register indirect operand with offset.
`#include <RegisterIndirectOffsetParser.h>`
Inheritance diagram for bnssassembler::RegisterIndirectOffsetParser:


bnssassembler::OperandParser
bnssassembler::RegisterIndirectOffsetParser

Protected Member Functions

- std::shared_ptr< **Operand** > **parse** (std::string str) const override
  Parses one operand. Does not call the next parser if it fails.

Additional Inherited Members

Detailed Description

Class representing the parser for the register indirect operand with offset.

Definition at line 10 of file RegisterIndirectOffsetParser.h.

Member Function Documentation

std::shared_ptr< Operand > bnssassembler::RegisterIndirectOffsetParser::parse (std::string *str*) const `[override]`, `[protected]`, `[virtual]`

Parses one operand. Does not call the next parser if it fails.

Parameters:

str	**Operand** which should be parsed

Returns:

Pointer to the operand or nullptr, if the parser failed parsing

Exceptions:

Throws	if the parser fails but identifies the error

Implements **bnssassembler::OperandParser** (*p.306*).

Definition at line 29 of file RegisterIndirectOffsetParser.cpp.

References bnssassembler::ExpressionBuilder::build(), bnssassembler::RegisterParser::parse(), and bnssassembler::parsePcrel().

```
   29 
{
   30         static std::regex 
regex("[[:space:]]*\\[[[:space:]]*([A-Za-z0-9]*)[[:space:]]*(.*)\\][[:space:]]*");
   31         if (!regex_match(str, regex)) {
   32             return parsePcrel(str);
   33         }
   34 
   35         auto reg_str = regex_replace(str, regex, "$1");
   36         auto off_str = "0" + regex_replace(str, regex, "$2");
   37 
   38         // This would be register indirect without offset
```

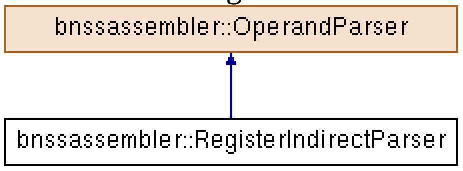
```
        if (off_str == "0") {
            return nullptr;
        }

        auto reg = RegisterParser::parse(reg_str);
        auto off = ExpressionBuilder::build(off_str);
        return std::make_shared<RegisterIndirectOffset>(reg, off, false);
    }
```

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**RegisterIndirectOffsetParser.h**
- Code/Assembler/Source/**RegisterIndirectOffsetParser.cpp**

bnssassembler::RegisterIndirectParser Class Reference

Class representing the parser for the register indirect operand.
`#include <RegisterIndirectParser.h>`
Inheritance diagram for bnssassembler::RegisterIndirectParser:


bnssassembler::OperandParser
bnssassembler::RegisterIndirectParser

Protected Member Functions

- std::shared_ptr< **Operand** > **parse** (std::string str) const override
  Parses one operand. Does not call the next parser if it fails.

Additional Inherited Members

Detailed Description

Class representing the parser for the register indirect operand.

Definition at line 10 of file RegisterIndirectParser.h.

Member Function Documentation

std::shared_ptr< Operand > bnssassembler::RegisterIndirectParser::parse (std::string *str*) const `[override]`, `[protected]`, `[virtual]`

Parses one operand. Does not call the next parser if it fails.

Parameters:

str	**Operand** which should be parsed

Returns:

Pointer to the operand or nullptr, if the parser failed parsing

Exceptions:

Throws	if the parser fails but identifies the error

Implements **bnssassembler::OperandParser** (*p.306*).

Definition at line 8 of file RegisterIndirectParser.cpp.

References bnssassembler::RegisterParser::parse().

```
{
        static std::regex
regex("[[:space:]]*\\[[[:space:]]*(.*)[[:space:]]*\\][[:space:]]*");
        if (!regex_match(str, regex)) {
            return nullptr;
        }

        auto reg_str = regex_replace(str, regex, "$1");

        auto reg = RegisterParser::parse(reg_str);
        return std::make_shared<RegisterIndirect>(reg);
    }
```

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**RegisterIndirectParser.h**
- Code/Assembler/Source/**RegisterIndirectParser.cpp**

bnssassembler::RegisterParser Class Reference

Utility class used for parsing registers.
`#include <RegisterParser.h>`

Classes

- struct **RegisterParserStaticData**

Static Public Member Functions

- static **Register parse** (std::string str)
  Parses the register.

Private Member Functions

- **RegisterParser** ()=delete
- **RegisterParser** (**RegisterParser** &)=delete
- void **operator=** (**RegisterParser** &)=delete

Static Private Member Functions

- static **RegisterParserStaticData** & **staticData** () noexcept

---

Detailed Description

Utility class used for parsing registers.

Definition at line 11 of file RegisterParser.h.

Constructor & Destructor Documentation

bnssassembler::RegisterParser::RegisterParser () `[private]`, `[delete]`

bnssassembler::RegisterParser::RegisterParser (RegisterParser &) `[private]`, `[delete]`

Member Function Documentation

void bnssassembler::RegisterParser::operator= (RegisterParser &) `[private]`, `[delete]`

Register bnssassembler::RegisterParser::parse (std::string *str*) `[static]`

Parses the register.

Parameters:

str	String representing the register

Returns:

Register

Exceptions:

Throws	if the register is not valid

Definition at line 8 of file RegisterParser.cpp.

References bnssassembler::RegisterParser::RegisterParserStaticData::map, and staticData().

Referenced by bnssassembler::RegisterDirectParser::parse(), bnssassembler::RegisterIndirectParser::parse(), and bnssassembler::RegisterIndirectOffsetParser::parse().

```
                                                 {
        static std::regex regex("[[:space:]]*([a-z0-9]*)[[:space:]]*");
        transform(str.begin(), str.end(), str.begin(), tolower);

        if (!regex_match(str, regex)) {
            throw MessageException("Invalid register: " + str);
        }

        str = regex_replace(str, regex, "$1");

        if (staticData().map.count(str) == 0) {
            throw MessageException("Invalid register: " + str);
        }

        return staticData().map[str];
    }
```

RegisterParser::RegisterParserStaticData & bnssassembler::RegisterParser::staticData () `[static]`, `[private]`, `[noexcept]`

Definition at line 25 of file RegisterParser.cpp.

Referenced by parse().

```
{
        static RegisterParserStaticData static_data;
        return static_data;
    }
```

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**RegisterParser.h**
- Code/Assembler/Source/**RegisterParser.cpp**

bnssassembler::RegisterParser::RegisterParserStaticData Struct Reference

Public Member Functions

- **RegisterParserStaticData** () noexcept

Public Attributes

- std::unordered_map< std::string, **Register** > **map**

Detailed Description

Definition at line 21 of file RegisterParser.h.

Constructor & Destructor Documentation

bnssassembler::RegisterParser::RegisterParserStaticData::RegisterParserStaticData ()[noexcept]

Definition at line 30 of file RegisterParser.cpp.

References bnssassembler::NUM_OF_REGISTERS, bnssassembler::PC, and bnssassembler::SP.

```
{
        for (size_t i = 0; i < NUM_OF_REGISTERS; i++) {
            map["r" + std::to_string(i)] = static_cast<Register>(i);
        }

        map["pc"] = PC;
        map["sp"] = SP;
    }
```

Member Data Documentation

std::unordered_map<std::string, Register> bnssassembler::RegisterParser::RegisterParserStaticData::map

Definition at line 22 of file RegisterParser.h.

Referenced by bnssassembler::RegisterParser::parse().

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

- Code/Assembler/Include/**RegisterParser.h**
- Code/Assembler/Source/**RegisterParser.cpp**

bnssassembler::RelocationRecord Class Reference

Class representing one relocation record.

```
#include <RelocationRecord.h>
```

Public Member Functions

- **RelocationRecord** ()=default
- **RelocationRecord** (bool **absolute**, size_t section_index) noexcept
  Create a relocation record using a section.
- **RelocationRecord** (bool **absolute**, std::string symbol_name) noexcept
  Create a relocation record using a symbol.
- void **offset** (size_t offset) noexcept
  Sets the offset of the relocation.
- void **toggleOpposite** () noexcept
  Toggles the opposite flag.
- void **absolute** (bool absolute) noexcept
  Sets the absolute flag.
- size_t **sectionIndex** () const noexcept
  Gets the section index of the relocation.
- std::string **symbolName** () const noexcept
  Gets the symbol name of the relocation.
- bool **section** () const noexcept
  Test whether the relocation is by section or by symbol.
- bool **opposite** () const noexcept
  Gets the opposite flag.

Private Attributes

- size_t **offset_** = 0
- bool **absolute_**
- size_t **section_index_** = 0
- std::string **symbol_name_**
- bool **section_**
- bool **opposite_** = false

Friends

- std::ostream & **operator<<** (std::ostream &os, const **RelocationRecord** &record)
  Writes the content of the object to a stream.
- bool **operator==** (const **RelocationRecord** &lhs, const **RelocationRecord** &rhs)
- bool **operator!=** (const **RelocationRecord** &lhs, const **RelocationRecord** &rhs)

Detailed Description

Class representing one relocation record.

Definition at line 10 of file RelocationRecord.h.

Constructor & Destructor Documentation

bnssassembler::RelocationRecord::RelocationRecord () `[default]`

bnssassembler::RelocationRecord::RelocationRecord (bool *absolute*, size_t *section_index*) `[noexcept]`

Create a relocation record using a section.

Parameters:

absolute	Boolean value indicating whether the relocation is absolute or relative
section_index	Index of relocation section

Definition at line 8 of file RelocationRecord.cpp.

```
    8 : absolute_(absolute), section_index_(section_index), section_(true) {}
```

bnssassembler::RelocationRecord::RelocationRecord (bool *absolute*, std::string *symbol_name*) `[noexcept]`

Create a relocation record using a symbol.

Parameters:

absolute	Boolean value indicating whether the relocation is absolute or relative
symbol_name	Name of the relocation symbol

Definition at line 9 of file RelocationRecord.cpp.

```
    9 : absolute_(absolute), symbol_name_(symbol_name), section_(false) {}
```

Member Function Documentation

void bnssassembler::RelocationRecord::absolute (bool *absolute*) `[noexcept]`

Sets the absolute flag.

Parameters:

absolute	Absolute flag

Definition at line 15 of file RelocationRecord.cpp.

```
   15                                                        {
   16          absolute_ = absolute;
   17      }
```

void bnssassembler::RelocationRecord::offset (size_t *offset*) `[noexcept]`

Sets the offset of the relocation.

Parameters:

offset	New offset

Definition at line 11 of file RelocationRecord.cpp.

```
   11                                                      {
   12          offset_ = offset;
   13      }
```

bool bnssassembler::RelocationRecord::opposite () const [noexcept]

Gets the opposite flag.

Returns:

Opposite flag

Definition at line 35 of file RelocationRecord.cpp.

References opposite_.

```
                                                   {
          return opposite_;
      }
```

bool bnssassembler::RelocationRecord::section () const [noexcept]

Test whether the relocation is by section or by symbol.

Returns:

Whether the relocation is by section

Definition at line 31 of file RelocationRecord.cpp.

References section_.

```
                                                  {
          return section_;
      }
```

size_t bnssassembler::RelocationRecord::sectionIndex () const [noexcept]

Gets the section index of the relocation.

Returns:

Section index

Definition at line 23 of file RelocationRecord.cpp.

References section_index_.

```
                                                         {
          return section_index_;
      }
```

std::string bnssassembler::RelocationRecord::symbolName () const [noexcept]

Gets the symbol name of the relocation.

Returns:

Symbol name

Definition at line 27 of file RelocationRecord.cpp.

References symbol_name_.

```
                                                          {
          return symbol_name_;
      }
```

void bnssassembler::RelocationRecord::toggleOpposite () [noexcept]

Toggles the opposite flag.

Definition at line 19 of file RelocationRecord.cpp.

References opposite_.

```
   19                                                    {
   20          opposite_ = !opposite_;
   21      }
```

Friends And Related Function Documentation

bool operator!= (const RelocationRecord & *lhs*, const RelocationRecord & *rhs*) `[friend]`

Definition at line 69 of file RelocationRecord.cpp.

```
   69
{
   70          return !(lhs == rhs);
   71      }
```

std::ostream& operator<< (std::ostream & *os*, const RelocationRecord & *record*) `[friend]`

Writes the content of the object to a stream.

Parameters:

os	Stream where the content will be written
record	**Data** that will be written

Definition at line 39 of file RelocationRecord.cpp.

```
   39
{
   40          os << record.offset_ << std::endl;
   41          os << record.absolute_ << std::endl;
   42          os << record.section_ << std::endl;
   43          if (record.section_) {
   44              os << record.section_index_ << std::endl;
   45          }
   46          else {
   47              os << record.symbol_name_ << std::endl;
   48          }
   49
   50          std::cout << VERTICAL << " " << std::setw(7) << std::left <<
record.offset_ << VERTICAL << " " << (record.absolute_ ? "Absolute" : "Relative") <<
" " << VERTICAL << " ";
   51          if (record.section_) {
   52              std::cout << std::setw(8) << std::left <<
std::to_string(record.section_index_) + "." << VERTICAL << std::setw(51) << " " <<
VERTICAL << std::endl;
   53          }
   54          else {
   55              std::cout << std::setw(8) << " " << VERTICAL << std::setw(51) <<
std::left << record.symbol_name_ << VERTICAL << std::endl;
   56          }
   57
   58          return os;
   59      }
```

bool operator== (const RelocationRecord & *lhs*, const RelocationRecord & *rhs*) `[friend]`

Definition at line 61 of file RelocationRecord.cpp.

```
{
        return
            lhs.offset_ == rhs.offset_ &&
            lhs.absolute_ == rhs.absolute_ &&
            lhs.section_ == rhs.section_ &&
            (lhs.section_ ? lhs.section_index_ == rhs.section_index_ : lhs.symbol_name_ == rhs.symbol_name_);
    }
```

Member Data Documentation

bool bnssassembler::RelocationRecord::absolute_ `[private]`

Definition at line 82 of file RelocationRecord.h.

Referenced by bnssassembler::operator<<(), and bnssassembler::operator==().

size_t bnssassembler::RelocationRecord::offset_ = 0 `[private]`

Definition at line 81 of file RelocationRecord.h.

Referenced by bnssassembler::operator<<(), and bnssassembler::operator==().

bool bnssassembler::RelocationRecord::opposite_ = false `[private]`

Definition at line 86 of file RelocationRecord.h.

Referenced by opposite(), and toggleOpposite().

bool bnssassembler::RelocationRecord::section_ `[private]`

Definition at line 85 of file RelocationRecord.h.

Referenced by bnssassembler::operator<<(), bnssassembler::operator==(), and section().

size_t bnssassembler::RelocationRecord::section_index_ = 0 `[private]`

Definition at line 83 of file RelocationRecord.h.

Referenced by bnssassembler::operator<<(), bnssassembler::operator==(), and sectionIndex().

std::string bnssassembler::RelocationRecord::symbol_name_ `[private]`

Definition at line 84 of file RelocationRecord.h.

Referenced by bnssassembler::operator<<(), bnssassembler::operator==(), and symbolName().

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**RelocationRecord.h**
- Code/Assembler/Source/**RelocationRecord.cpp**

bnssemulator::RelocationRecord Class Reference

Class representing one relocation record.
`#include <RelocationRecord.h>`

Public Member Functions

- size_t **offset** () const noexcept
  Gets the offset of the relocation record.
- bool **absolute** () const noexcept
  Checks whether the relocation is absolute or relative.
- size_t **sectionIndex** () const noexcept
  Gets the section index of the relocation.
- std::string **symbolName** () const noexcept
  Gets the symbol name of the relocation.
- bool **section** () const noexcept
  Test whether the relocation is by section or by symbol.

Private Attributes

- size_t **offset_** = 0
- bool **absolute_** = false
- size_t **section_index_** = 0
- std::string **symbol_name_**
- bool **section_** = false

Friends

- std::istream & **operator**>> (std::istream &is, **RelocationRecord** &data)
  Loads the object from stream.

Detailed Description

Class representing one relocation record.

Definition at line 11 of file RelocationRecord.h.

Member Function Documentation

bool bnssemulator::RelocationRecord::absolute () const [noexcept]

Checks whether the relocation is absolute or relative.

Returns:

Whether the relocation is absolute or relative

Definition at line 23 of file RelocationRecord.cpp.

References absolute_.

```
{
        return absolute_;
    }
```

size_t bnssemulator::RelocationRecord::offset () const `[noexcept]`

Gets the offset of the relocation record.

Returns:

Offset of the relocation record

Definition at line 19 of file RelocationRecord.cpp.

References offset_.

```
                                                   {
        return offset_;
    }
```

bool bnssemulator::RelocationRecord::section () const `[noexcept]`

Test whether the relocation is by section or by symbol.

Returns:

Whether the relocation is by section

Definition at line 35 of file RelocationRecord.cpp.

References section_.

```
                                                  {
        return section_;
    }
```

size_t bnssemulator::RelocationRecord::sectionIndex () const `[noexcept]`

Gets the section index of the relocation.

Returns:

Section index

Definition at line 27 of file RelocationRecord.cpp.

References section_index_.

```
                                                         {
        return section_index_;
    }
```

std::string bnssemulator::RelocationRecord::symbolName () const `[noexcept]`

Gets the symbol name of the relocation.

Returns:

Symbol name

Definition at line 31 of file RelocationRecord.cpp.

References symbol_name_.

```
                                                          {
        return symbol_name_;
    }
```

Friends And Related Function Documentation

std::istream& operator>> (std::istream & *is*, RelocationRecord & *data*) `[friend]`

Loads the object from stream.

Parameters:

is	Input stream
data	Reference to the object that should be loaded

Returns:

Input stream

Definition at line 5 of file RelocationRecord.cpp.

```
                                                                    {
       is >> data.offset_;
       is >> data.absolute_;
       is >> data.section_;
       if (data.section_) {
           is >> data.section_index_;
       }
       else {
           is >> data.symbol_name_;
       }

       return is;
   }
```

Member Data Documentation

bool bnssemulator::RelocationRecord::absolute_ = false `[private]`

Definition at line 52 of file RelocationRecord.h.

Referenced by absolute(), and bnssemulator::operator>>().

size_t bnssemulator::RelocationRecord::offset_ = 0 `[private]`

Definition at line 51 of file RelocationRecord.h.

Referenced by offset(), and bnssemulator::operator>>().

bool bnssemulator::RelocationRecord::section_ = false `[private]`

Definition at line 55 of file RelocationRecord.h.

Referenced by bnssemulator::operator>>(), and section().

size_t bnssemulator::RelocationRecord::section_index_ = 0 `[private]`

Definition at line 53 of file RelocationRecord.h.

Referenced by bnssemulator::operator>>(), and sectionIndex().

std::string bnssemulator::RelocationRecord::symbol_name_ `[private]`

Definition at line 54 of file RelocationRecord.h.

Referenced by bnssemulator::operator>>(), and symbolName().

The documentation for this class was generated from the following files:

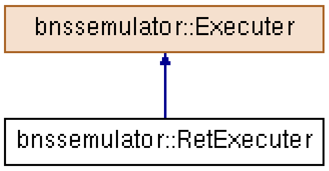
- Code/Emulator/Include/**RelocationRecord.h**
- Code/Emulator/Source/**RelocationRecord.cpp**

bnssemulator::RetExecuter Class Reference

Class representing the executer for ret instruction.

`#include <RetExecuter.h>`

Inheritance diagram for bnssemulator::RetExecuter:



Public Member Functions

- void **execute** (**InstructionBitField** instruction, **Context** &context) const override
  Executes the instruction.

Detailed Description

Class representing the executer for ret instruction.

Definition at line 10 of file RetExecuter.h.

Member Function Documentation

void bnssemulator::RetExecuter::execute (InstructionBitField *instruction*, Context & *context*) const `[override]`, `[virtual]`

Executes the instruction.

Parameters:

instruction	Instruction
context	**Processor** context

Implements **bnssemulator::Executer** (*p.163*).

Definition at line 5 of file RetExecuter.cpp.

References bnssemulator::Context::returnFromSubroutine().

```
{
        context.returnFromSubroutine();
    }
```

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**RetExecuter.h**
- Code/Emulator/Source/**RetExecuter.cpp**

bnssassembler::SecondPass Class Reference

Utility class executing the second pass.
`#include <SecondPass.h>`

Static Public Member Functions

- static **SecondPassData execute** (std::vector< std::shared_ptr< **Token** >> tokens, **FirstPassData** &&first_pass_data)
  Executes the second pass using the first pass.

Private Member Functions

- **SecondPass** ()=delete
- **SecondPass** (**SecondPass** &)=delete
- void **operator=** (**SecondPass** &)=delete

Detailed Description

Utility class executing the second pass.

Definition at line 11 of file SecondPass.h.

Constructor & Destructor Documentation

bnssassembler::SecondPass::SecondPass () `[private]`, `[delete]`

bnssassembler::SecondPass::SecondPass (SecondPass &) `[private]`, `[delete]`

Member Function Documentation

SecondPassData bnssassembler::SecondPass::execute (std::vector< std::shared_ptr< Token >> *tokens*, FirstPassData && *first_pass_data*) `[static]`

Executes the second pass using the first pass.

Parameters:

tokens	Vector of parsed tokens
first_pass_data	**Data** generated from the first pass

Definition at line 7 of file SecondPass.cpp.

References bnssassembler::SecondPassData::currentSectionIndex(), bnssassembler::SecondPassData::currentSectionOffset(), bnssassembler::SecondPassData::importedSymbols(), bnssassembler::MessageException::message(), bnssassembler::SecondPassData::orgValid(), and bnssassembler::SecondPassData::symbolTable().

Referenced by main().

```
{
        SecondPassData ret(std::move(first_pass_data));

        for (auto &token : tokens) {
            try {
                if (ret.orgValid() && !token->usesAddress()) {
                    throw MessageException("ORG directive must be followed by a section start");
                }

                token->resolveCurrentPcSymbol(ret.currentSectionIndex(), ret.currentSectionOffset());
                token->resolveSymbolTable(ret.symbolTable());
                token->resolveImports(ret.importedSymbols());
                token->secondPass(ret);
            } catch (MessageException &exception) {
                throw SecondPassException(token->lineNumber(), token->line(), exception.message());
            }
        }

        return ret;
    }
```

void bnssassembler::SecondPass::operator= (SecondPass &)`[private]`, `[delete]`

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**SecondPass.h**
- Code/Assembler/Source/**SecondPass.cpp**

bnssassembler::SecondPassData Class Reference

Class representing the data that will be updated during the second pass.
`#include <SecondPassData.h>`

Public Member Functions

- **SecondPassData** (**FirstPassData** &&first_pass_data) noexcept
  Constructs the object.
- void **nextSection** () noexcept
  Increasses the current section counter.
- bool **orgValid** () const noexcept
  Check if the ORG address is valid.
- bool **contains** (std::string symbol) const noexcept
  Check whether the symbol exists in the symbol table.
- void **insertImported** (std::string symbol)
  Inserts an imported symbol into the set.
- void **exportSymbol** (std::string symbol)
  Exports a symbol.
- **SectionType currentSectionType** () const noexcept
  Gets the current section type.
- void **addData** (uint8_t data, std::list< **RelocationRecord** > relocations)
  Adds 8 bits of data to the current section.
- void **addData** (uint16_t data, std::list< **RelocationRecord** > relocations)
  Adds 16 bits of data to the current section.
- void **addData** (**uint32_t** data, std::list< **RelocationRecord** > relocations)
  Adds 32 bits of data to the current section.
- void **org** (**uint32_t** address)
  Sets the ORG address.
- **SymbolTable** & **symbolTable** () noexcept
  Gets the symbol table.
- const **SymbolTable** & **symbolTable** () const noexcept
  Gets the symbol table.
- std::unordered_set< std::string > & **importedSymbols** ()
  Gets the collection of imported symbols.
- const std::unordered_set< std::string > & **importedSymbols** () const
  Gets the collection of imported symbols.
- size_t **currentSectionIndex** () const noexcept
  Gets the index of the current section.
- size_t **currentSectionOffset** () const noexcept
  Gets the current offset inside the current section.

Private Attributes

- std::unordered_set< std::string > **imported_symbols_**
- **SymbolTable symbol_table_**
- **SectionTable section_table_**
- int32_t **org_address_** = 0
- bool **org_valid_** = false

Friends

- std::ostream & **operator<<** (std::ostream &os, const **SecondPassData** &data)
  Writes the content of the object to a stream.

Detailed Description

Class representing the data that will be updated during the second pass.

Definition at line 11 of file SecondPassData.h.

Constructor & Destructor Documentation

bnssassembler::SecondPassData::SecondPassData (FirstPassData && *first_pass_data*)`[explicit]`, `[noexcept]`

Constructs the object.

Parameters:

first_pass_data	**Data** generated by the first pass

Definition at line 8 of file SecondPassData.cpp.

```
    8 : symbol_table_(move(first_pass_data.symbol_table_)),
section_table_(move(first_pass_data.section_table_)) {}
```

Member Function Documentation

void bnssassembler::SecondPassData::addData (uint8_t *data*, std::list< RelocationRecord > *relocations*)

Adds 8 bits of data to the current section.

Parameters:

data	**Data** to be addded
relocations	List of relocation records for the data

Definition at line 38 of file SecondPassData.cpp.

References bnssassembler::SectionTable::addData(), and section_table_.

Referenced by bnssassembler::DataDefinitionToken::secondPass(), and bnssassembler::InstructionToken::secondPass().

```
   38
{
   39         section_table_.addData(data, relocations);
   40     }
```

void bnssassembler::SecondPassData::addData (uint16_t *data*, std::list< RelocationRecord > *relocations*)

Adds 16 bits of data to the current section.

Parameters:

data	**Data** to be addded
relocations	List of relocation records for the data

Definition at line 42 of file SecondPassData.cpp.

References bnssassembler::SectionTable::addData(), and section_table_.

```
   42                                                              
{
   43         section_table_.addData(data, relocations);
   44     }
```

void bnssassembler::SecondPassData::addData (uint32_t *data*, std::list< RelocationRecord > *relocations*)

Adds 32 bits of data to the current section.

Parameters:

data	**Data** to be addded
relocations	List of relocation records for the data

Definition at line 46 of file SecondPassData.cpp.

References bnssassembler::SectionTable::addData(), and section_table_.

```
   46                                                              
{
   47         section_table_.addData(data, relocations);
   48     }
```

bool bnssassembler::SecondPassData::contains (std::string *symbol*) const`[noexcept]`

Check whether the symbol exists in the symbol table.

Definition at line 22 of file SecondPassData.cpp.

References bnssassembler::SymbolTable::contains(), and symbol_table_.

Referenced by bnssassembler::GlobalSymbolsToken::secondPass().

```
   22                                                              {
   23         return symbol_table_.contains(symbol);
   24     }
```

size_t bnssassembler::SecondPassData::currentSectionIndex () const`[noexcept]`

Gets the index of the current section.

Definition at line 71 of file SecondPassData.cpp.

References bnssassembler::SectionTable::currentIndex(), and section_table_.

Referenced by currentSectionOffset(), and bnssassembler::SecondPass::execute().

```
   71                                                          {
   72         return section_table_.currentIndex();
   73     }
```

size_t bnssassembler::SecondPassData::currentSectionOffset () const`[noexcept]`

Gets the current offset inside the current section.

Definition at line 75 of file SecondPassData.cpp.

References bnssassembler::SectionTable::current(), currentSectionIndex(), section_table_, and bnssassembler::SectionData::size().

Referenced by bnssassembler::SecondPass::execute().

```
                                                        {
        if (currentSectionIndex() == static_cast<size_t>(-1)) {
            return 0;
        }

        return section_table_.current().size();
    }
```

SectionType bnssassembler::SecondPassData::currentSectionType () const[noexcept]

Gets the current section type.

Returns:

Current section type

Definition at line 34 of file SecondPassData.cpp.

References bnssassembler::SectionTable::currentSectionType(), and section_table_.

Referenced by bnssassembler::DataDefinitionToken::secondPass(), and bnssassembler::InstructionToken::secondPass().

```
                                                            {
        return section_table_.currentSectionType();
    }
```

void bnssassembler::SecondPassData::exportSymbol (std::string *symbol*)

Exports a symbol.

Parameters:

symbol	**Symbol** to be exported

Definition at line 30 of file SecondPassData.cpp.

References bnssassembler::SymbolTable::exportSymbol(), and symbol_table_.

Referenced by bnssassembler::GlobalSymbolsToken::secondPass().

```
                                                {
        symbol_table_.exportSymbol(symbol);
    }
```

std::unordered_set< std::string > & bnssassembler::SecondPassData::importedSymbols ()

Gets the collection of imported symbols.

Returns:

Collection of imported symbols

Definition at line 63 of file SecondPassData.cpp.

References imported_symbols_.

Referenced by bnssassembler::SecondPass::execute(), and importedSymbols().

```
                                                            {
        return imported_symbols_;
    }
```

const std::unordered_set< std::string > & bnssassembler::SecondPassData::importedSymbols () const

Gets the collection of imported symbols.

Returns:

Collection of imported symbols

Definition at line 67 of file SecondPassData.cpp.

References importedSymbols().

```
{
        return const_cast<SecondPassData &>(*this).importedSymbols();
    }
```

void bnssassembler::SecondPassData::insertImported (std::string *symbol*)

Inserts an imported symbol into the set.

Parameters:

symbol	**Symbol** to be imported

Definition at line 26 of file SecondPassData.cpp.

References imported_symbols_.

Referenced by bnssassembler::GlobalSymbolsToken::secondPass().

```
                                                {
        imported_symbols_.insert(symbol);
    }
```

void bnssassembler::SecondPassData::nextSection () `[noexcept]`

Increasses the current section counter.

Definition at line 10 of file SecondPassData.cpp.

References bnssassembler::SectionTable::current(), bnssassembler::SectionTable::nextSection(), bnssassembler::SectionData::org(), org_address_, org_valid_, and section_table_.

Referenced by bnssassembler::SectionStartToken::secondPass().

```
                                          {
        section_table_.nextSection();
        if (org_valid_) {
            org_valid_ = false;
            section_table_.current().org(org_address_);
        }
    }
```

void bnssassembler::SecondPassData::org (uint32_t *address*)

Sets the ORG address.

Parameters:

address	ORG address

Definition at line 50 of file SecondPassData.cpp.

References org_address_, and org_valid_.

Referenced by bnssassembler::OrgDirectiveToken::secondPass().

```
                                        {
        org_address_ = address;
        org_valid_ = true;
    }
```

bool bnssassembler::SecondPassData::orgValid () const`[noexcept]`

Check if the ORG address is valid.

Definition at line 18 of file SecondPassData.cpp.

References org_valid_.

Referenced by bnssassembler::SecondPass::execute().

```
                                            {
        return org_valid_;
    }
```

SymbolTable & bnssassembler::SecondPassData::symbolTable ()`[noexcept]`

Gets the symbol table.

Returns:

Symbol table

Definition at line 55 of file SecondPassData.cpp.

References symbol_table_.

Referenced by bnssassembler::SecondPass::execute(), and symbolTable().

```
                                                   {
        return symbol_table_;
    }
```

const SymbolTable & bnssassembler::SecondPassData::symbolTable () const`[noexcept]`

Gets the symbol table.

Returns:

Symbol table

Definition at line 59 of file SecondPassData.cpp.

References symbolTable().

```
                                                                 {
        return const_cast<SecondPassData &>(*this).symbolTable();
    }
```

Friends And Related Function Documentation

std::ostream& operator<< (std::ostream & *os*, const SecondPassData & *data*)`[friend]`

Writes the content of the object to a stream.

Parameters:

os	Stream where the content will be written

data	**Data** that will be written

Definition at line 83 of file SecondPassData.cpp.

```
   83                                                                   {
   84         os << data.imported_symbols_.size() << std::endl;
   85
   86         std::cout << UPPER_LEFT << multiple(HORIZONTAL, 81) << UPPER_RIGHT <<
std::endl;
   87         std::cout << VERTICAL << UPPER_LEFT << multiple(HORIZONTAL, 79) <<
UPPER_RIGHT << VERTICAL << std::endl;
   88         std::cout << VERTICAL << VERTICAL << std::setw(79) << std::left << "
Imported symbols:" << VERTICAL << VERTICAL << std::endl;
   89         std::cout << VERTICAL << LOWER_LEFT << multiple(HORIZONTAL, 79) <<
LOWER_RIGHT << VERTICAL << std::endl;
   90         std::cout << T_RIGHT << multiple(HORIZONTAL, 81) << T_LEFT << std::endl;
   91
   92         for (auto &symbol : data.imported_symbols_) {
   93             os << symbol << std::endl;
   94             std::cout << VERTICAL << " " << std::setw(80) << std::left << symbol
<< VERTICAL << std::endl;
   95         }
   96
   97         std::cout << LOWER_LEFT << multiple(HORIZONTAL, 81) << LOWER_RIGHT <<
std::endl << std::endl << std::endl;
   98         os << data.section_table_ << std::endl;
   99         os << data.symbol_table_ << std::endl;
  100
  101         return os;
  102     }
```

Member Data Documentation

std::unordered_set<std::string> bnssassembler::SecondPassData::imported_symbols_ `[private]`

Definition at line 120 of file SecondPassData.h.

Referenced by importedSymbols(), insertImported(), and bnssassembler::operator<<().

int32_t bnssassembler::SecondPassData::org_address_ = 0 `[private]`

Definition at line 124 of file SecondPassData.h.

Referenced by nextSection(), and org().

bool bnssassembler::SecondPassData::org_valid_ = false `[private]`

Definition at line 125 of file SecondPassData.h.

Referenced by nextSection(), org(), and orgValid().

SectionTable bnssassembler::SecondPassData::section_table_ `[private]`

Definition at line 122 of file SecondPassData.h.

Referenced by addData(), currentSectionIndex(), currentSectionOffset(), currentSectionType(), nextSection(), and bnssassembler::operator<<().

SymbolTable bnssassembler::SecondPassData::symbol_table_ `[private]`

Definition at line 121 of file SecondPassData.h.

Referenced by contains(), exportSymbol(), bnssassembler::operator<<(), and symbolTable().

The documentation for this class was generated from the following files:

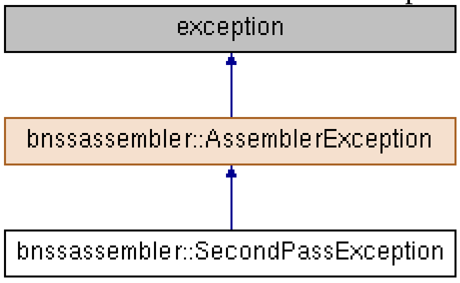
- Code/Assembler/Include/**SecondPassData.h**
- Code/Assembler/Source/**SecondPassData.cpp**

bnssassembler::SecondPassException Class Reference

Represents an exception that happened during the assembler second pass.

`#include <SecondPassException.h>`

Inheritance diagram for bnssassembler::SecondPassException:


exception
bnssassembler::AssemblerException
bnssassembler::SecondPassException

Public Member Functions

- **SecondPassException** (size_t line_number, std::string line, std::string specific_message) noexcept
  *Constructs a **SecondPassException** object.*

Protected Member Functions

- std::string **messageBody** () const noexcept override
  Returns the actual message body of the exception.

Private Attributes

- std::string **specific_message_**

Detailed Description

Represents an exception that happened during the assembler second pass.

Definition at line 10 of file SecondPassException.h.

Constructor & Destructor Documentation

bnssassembler::SecondPassException::SecondPassException (size_t *line_number*, std::string *line*, std::string *specific_message*) `[noexcept]`

Constructs a **SecondPassException** object.

Parameters:

line_number	Number of the line where the error happened
line	Line where the error happened
specific_message	Specific message about the error that happened

Definition at line 5 of file SecondPassException.cpp.

```
    5 : AssemblerException(line_number, line), specific_message_(specific_message) {}
```

Member Function Documentation

std::string bnssassembler::SecondPassException::messageBody () const`[override]`, `[protected]`, `[virtual]`, `[noexcept]`

Returns the actual message body of the exception.

Implements **bnssassembler::AssemblerException** (*p.109*).

Definition at line 7 of file SecondPassException.cpp.

References specific_message_.

```
                                                            {
        return "Error during the second pass\n" + specific_message_;
    }
```

Member Data Documentation

std::string bnssassembler::SecondPassException::specific_message_`[private]`

Definition at line 22 of file SecondPassException.h.

Referenced by messageBody().

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**SecondPassException.h**
- Code/Assembler/Source/**SecondPassException.cpp**

bnssassembler::SectionData Class Reference

Class representing the data about one section.

`#include <SectionData.h>`

Public Member Functions

- **SectionData** (**SectionType type**) noexcept
  *Construct a non-indexed **SectionData** object with the specified type.*
- **SectionData** (**SectionType type**, size_t **index**) noexcept
  *Construct an indexed **SectionData** object with the specified type and index.*
- void **incLocationCounter** (size_t offset) noexcept
  Increases the location counter by an offset.
- **SectionType type** () const noexcept
  Get the type of the section.
- bool **indexed** () const noexcept
  Check whether the section is indexed.
- size_t **index** () const noexcept
  Get the index of the section.
- size_t **locationCounter** () const noexcept
  Gets the value of the location counter.
- size_t **hash** () const noexcept
  Hash the sectionData object.
- void **addData** (uint8_t data, std::list< **RelocationRecord** > &relocations)
  Adds 8 bits of data to the data array.
- void **addData** (uint16_t data, std::list< **RelocationRecord** > &relocations)
  Adds 16 bits of data to the data array.
- void **addData** (**uint32_t** data, std::list< **RelocationRecord** > &relocations)
  Adds 32 bits of data to the data array.
- void **org** (**uint32_t** address)
  Sets the ORG address.
- size_t **size** () const noexcept
  Gets the current size of the data.

Private Member Functions

- void **addData** (uint8_t data)
  Adds 8 bits of data to the data array, without relocation records.

Private Attributes

- **SectionType type_**
- bool **indexed_** = false
- size_t **index_** = 0
- size_t **location_counter_** = 0
- **uint32_t org_address_** = 0
- bool **org_valid_** = false
- std::vector< uint8_t > **data_**
- std::list< **RelocationRecord** > **relocation_records_**

Friends

- bool **operator==** (const **SectionData** &lhs, const **SectionData** &rhs) noexcept

- bool **operator!=** (const **SectionData** &lhs, const **SectionData** &rhs) noexcept
- bool **operator<** (const **SectionData** &lhs, const **SectionData** &rhs) noexcept
- bool **operator>** (const **SectionData** &lhs, const **SectionData** &rhs) noexcept
- bool **operator<=** (const **SectionData** &lhs, const **SectionData** &rhs) noexcept
- bool **operator>=** (const **SectionData** &lhs, const **SectionData** &rhs) noexcept
- std::ostream & **operator<<** (std::ostream &os, const **SectionData** &data)
  Writes the content of the object to a stream.

Detailed Description

Class representing the data about one section.

Definition at line 18 of file SectionData.h.

Constructor & Destructor Documentation

bnssassembler::SectionData::SectionData (SectionType *type*) `[explicit]`, `[noexcept]`

Construct a non-indexed **SectionData** object with the specified type.

Parameters:

type	Type of the section

Definition at line 8 of file SectionData.cpp.

```
    8 : type_(type) {}
```

bnssassembler::SectionData::SectionData (SectionType *type*, size_t *index*) `[explicit]`, `[noexcept]`

Construct an indexed **SectionData** object with the specified type and index.

Parameters:

type	Type of the section
index	Index of the section

Definition at line 9 of file SectionData.cpp.

```
    9 : type_(type), indexed_(true), index_(index) {}
```

Member Function Documentation

void bnssassembler::SectionData::addData (uint8_t *data*, std::list< RelocationRecord > & *relocations*)

Adds 8 bits of data to the data array.

Parameters:

data	**Data** to be addded
relocations	List of relocation records for the data

Definition at line 39 of file SectionData.cpp.

Referenced by addData().

```
   39
{
   40         addData(data);
   41         if (!relocations.empty()) {
   42             throw MessageException("Only 32bit data definitions can contain
labels");
   43         }
   44     }
```

void bnssassembler::SectionData::addData (uint16_t *data*, std::list< RelocationRecord > & *relocations*)

Adds 16 bits of data to the data array.

Parameters:

data	**Data** to be addded
relocations	List of relocation records for the data

Definition at line 46 of file SectionData.cpp.

References addData().

```
   46
{
   47         addData(static_cast<uint8_t>(data & 0x00FF));
   48         addData(static_cast<uint8_t>((data & 0xFF00) >> 8));
   49         if (!relocations.empty()) {
   50             throw MessageException("Only 32bit data definitions can contain
labels");
   51         }
   52     }
```

void bnssassembler::SectionData::addData (uint32_t *data*, std::list< RelocationRecord > & *relocations*)

Adds 32 bits of data to the data array.

Parameters:

data	**Data** to be addded
relocations	List of relocation records for the data

Definition at line 54 of file SectionData.cpp.

References addData(), data_, and relocation_records_.

```
   54
{
   55         addData(static_cast<uint8_t>(data & 0x000000FF));
   56         addData(static_cast<uint8_t>((data & 0x0000FF00) >> 8));
   57         addData(static_cast<uint8_t>((data & 0x00FF0000) >> 16));
   58         addData(static_cast<uint8_t>((data & 0xFF000000) >> 24));
   59         for (auto &relocation : relocations) {
   60             relocation.offset(data_.size() - 4);
   61         }
   62
   63         relocation_records_.splice(relocation_records_.end(),
move(relocations));
   64     }
```

void bnssassembler::SectionData::addData (uint8_t *data*)`[private]`

Adds 8 bits of data to the data array, without relocation records.

Parameters:

data	**Data** to be added

Definition at line 75 of file SectionData.cpp.

References data_.

```
75                                  {
76          data_.push_back(data);
77      }
```

size_t bnssassembler::SectionData::hash () const `[noexcept]`

Hash the sectionData object.

Returns:

Hashed value

Definition at line 31 of file SectionData.cpp.

References index_, indexed_, and type_.

Referenced by std::hash< bnssassembler::SectionData >::operator()().

```
31                                      {
32          if (indexed_) {
33              return 4 + index_ * 4 + type_;
34          }
35
36          return type_;
37      }
```

void bnssassembler::SectionData::incLocationCounter (size_t *offset*) `[noexcept]`

Increases the location counter by an offset.

Parameters:

offset	Offset

Definition at line 11 of file SectionData.cpp.

```
11                                                              {
12          location_counter_ += offset;
13      }
```

size_t bnssassembler::SectionData::index () const `[noexcept]`

Get the index of the section.

Returns:

Index of the section

Undefined when the section is not indexed

Definition at line 23 of file SectionData.cpp.

References index_.

Referenced by bnssassembler::SectionTable::operator+=().

```
23                                       {
24          return index_;
25      }
```

bool bnssassembler::SectionData::indexed () const `[noexcept]`

Check whether the section is indexed.

Returns:

Whether the section is indexed

Definition at line 19 of file SectionData.cpp.

References indexed_.

Referenced by bnssassembler::SectionTable::operator+=().

```
                                             {
            return indexed_;
        }
```

size_t bnssassembler::SectionData::locationCounter () const [noexcept]

Gets the value of the location counter.

Returns:

Value of the location counter

Definition at line 27 of file SectionData.cpp.

References location_counter_.

```
                                                       {
            return location_counter_;
        }
```

void bnssassembler::SectionData::org (uint32_t *address*)

Sets the ORG address.

Parameters:

address	ORG address

Definition at line 66 of file SectionData.cpp.

References org_address_, and org_valid_.

Referenced by bnssassembler::SecondPassData::nextSection().

```
                                           {
            org_address_ = address;
            org_valid_ = true;
        }
```

size_t bnssassembler::SectionData::size () const [noexcept]

Gets the current size of the data.

Definition at line 71 of file SectionData.cpp.

References data_.

Referenced by bnssassembler::SecondPassData::currentSectionOffset().

```
                                            {
            return data_.size();
        }
```

SectionType bnssassembler::SectionData::type () const [noexcept]

Get the type of the section.

Returns:

Type of the section

Definition at line 15 of file SectionData.cpp.

References type_.

Referenced by bnssassembler::SectionTable::operator+=().

```
                                                     {
        return type_;
    }
```

Friends And Related Function Documentation

bool operator!= (const SectionData & *lhs*, const SectionData & *rhs*)`[friend]`

Definition at line 83 of file SectionData.cpp.

```
{
        return !(lhs == rhs);
    }
```

bool operator< (const SectionData & *lhs*, const SectionData & *rhs*)`[friend]`

Definition at line 87 of file SectionData.cpp.

```
{
        if (lhs.type_ < rhs.type_) {
            return true;
        }

        if (lhs.type_ > rhs.type_) {
            return false;
        }

        if (!lhs.indexed_ && rhs.indexed_) {
            return true;
        }

        if (lhs.indexed_ && !rhs.indexed_) {
            return false;
        }

        if (lhs.indexed_) {
            return lhs.index_ < rhs.index_;
        }

        return false;
    }
```

std::ostream& operator<< (std::ostream & *os*, const SectionData & *data*)`[friend]`

Writes the content of the object to a stream.

Parameters:

os	Stream where the content will be written
data	**Data** that will be written

Definition at line 156 of file SectionData.cpp.

```
                                                                        {
        os << data.type_ << std::endl;
        os << data.indexed_ << std::endl;
        if (data.indexed_) {
            os << data.index_ << std::endl;
        }

        os << data.org_valid_ << std::endl;
        if (data.org_valid_) {
            os << data.org_address_ << std::endl;
        }

        os << data.location_counter_ << std::endl;
        os << data.data_.size() << std::endl;
        for (auto &entry : data.data_) {
            os << StringHelper::numberFormat(entry) << std::endl;
        }

        writeDescription(data.type_, data.indexed_, data.index_, data.org_valid_, data.org_address_, data.location_counter_);

        std::cout << VERTICAL << " ";

        size_t i;
        for (i = 0; i < data.data_.size(); i++) {
            auto entry = data.data_[i];
            if (i % 16 == 0 && i != 0) {
                std::cout << VERTICAL << std::endl << VERTICAL << " ";
            }

            std::cout << StringHelper::toHexString(entry) << " ";
        }

        for (; i % 16 != 0 || i == 0; i++) {
            std::cout << "     ";
        }

        std::cout << VERTICAL << std::endl;

        std::cout << T_RIGHT << multiple(HORIZONTAL, 81) << T_LEFT << std::endl;
        std::cout << VERTICAL << std::setw(81) << std::left << " Relocation table:" << VERTICAL << std::endl;
        std::cout << T_RIGHT << multiple(HORIZONTAL, 8) << T_DOWN << multiple(HORIZONTAL, 10) << T_DOWN << multiple(HORIZONTAL, 9) << T_DOWN << multiple(HORIZONTAL, 51) << T_LEFT << std::endl;

        std::cout << VERTICAL << " Offset " << VERTICAL << " Absolute " << VERTICAL << " Section " << VERTICAL << "                      Symbol                       " << VERTICAL << std::endl;
        std::cout << T_RIGHT << multiple(HORIZONTAL, 8) << ALL_FOUR << multiple(HORIZONTAL, 10) << ALL_FOUR << multiple(HORIZONTAL, 9) << ALL_FOUR << multiple(HORIZONTAL, 51) << T_LEFT << std::endl;

        os << data.relocation_records_.size() << std::endl;
        for (auto &record : data.relocation_records_) {
            os << record << std::endl;
        }

        std::cout << LOWER_LEFT << multiple(HORIZONTAL, 8) << T_UP << multiple(HORIZONTAL, 10) << T_UP << multiple(HORIZONTAL, 9) << T_UP << multiple(HORIZONTAL, 51) << LOWER_RIGHT << std::endl;

        return os;
    }
```

bool operator<= (const SectionData & *lhs*, const SectionData & *rhs*) `[friend]`

Definition at line 115 of file SectionData.cpp.

```
{
        return !(lhs > rhs);
    }
```

bool operator== (const SectionData & *lhs*, const SectionData & *rhs*) `[friend]`

Definition at line 79 of file SectionData.cpp.

```
   79
{
   80         return lhs.type_ == rhs.type_ && lhs.indexed_ == rhs.indexed_ &&
(lhs.indexed_ ? lhs.index_ == rhs.index_ : true);
   81     }
```

bool operator> (const SectionData & *lhs*, const SectionData & *rhs*) `[friend]`

Definition at line 111 of file SectionData.cpp.

```
  111
{
  112         return !(lhs < rhs || lhs == rhs);
  113     }
```

bool operator>= (const SectionData & *lhs*, const SectionData & *rhs*) `[friend]`

Definition at line 119 of file SectionData.cpp.

```
  119
{
  120         return !(lhs < rhs);
  121     }
```

Member Data Documentation

std::vector<uint8_t> bnssassembler::SectionData::data_ `[private]`

Definition at line 126 of file SectionData.h.

Referenced by addData(), bnssassembler::operator<<(), and size().

size_t bnssassembler::SectionData::index_ = 0 `[private]`

Definition at line 121 of file SectionData.h.

Referenced by hash(), index(), and bnssassembler::operator<<().

bool bnssassembler::SectionData::indexed_ = false `[private]`

Definition at line 120 of file SectionData.h.

Referenced by hash(), indexed(), and bnssassembler::operator<<().

size_t bnssassembler::SectionData::location_counter_ = 0 `[private]`

Definition at line 122 of file SectionData.h.

Referenced by locationCounter(), and bnssassembler::operator<<().

uint32_t bnssassembler::SectionData::org_address_ = 0 `[private]`

Definition at line 123 of file SectionData.h.

Referenced by bnssassembler::operator<<(), and org().

bool bnssassembler::SectionData::org_valid_ = false `[private]`

Definition at line 124 of file SectionData.h.

Referenced by bnssassembler::operator<<(), and org().

std::list<RelocationRecord> bnssassembler::SectionData::relocation_records_ `[private]`

Definition at line 127 of file SectionData.h.

Referenced by addData(), and bnssassembler::operator<<().

SectionType bnssassembler::SectionData::type_ `[private]`

Definition at line 119 of file SectionData.h.

Referenced by hash(), bnssassembler::operator<<(), and type().

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**SectionData.h**
- Code/Assembler/Source/**SectionData.cpp**

bnssemulator::SectionData Class Reference

Class representing the data about one section.
`#include <SectionData.h>`

Public Member Functions

- bool **hasAddress** () const noexcept
  Checks whether the section already has a starting address.
- **uint32_t address** () const noexcept
  Gets the starting address of the section.
- void **address** (**uint32_t** address) noexcept
  Sets the starting address of the section.
- size_t **size** () const noexcept
  Gets the size of the sections.
- **SectionType type** () const noexcept
  Gets the type of the section.
- std::vector< uint8_t > & **data** () noexcept
  Gets the data vector of the section.
- const std::vector< uint8_t > & **data** () const noexcept
  Gets the data vector of the section.
- std::vector< **RelocationRecord** > & **relocations** () noexcept
  Gets the relocation records for the section.
- const std::vector< **RelocationRecord** > & **relocations** () const noexcept
  Gets the relocation records for the section.

Private Attributes

- **SectionType type_** = **TEXT**
- bool **indexed_** = false
- size_t **index_** = 0
- size_t **location_counter_** = 0
- **uint32_t org_address_** = 0
- bool **org_valid_** = false
- std::vector< uint8_t > **data_**
- std::vector< **RelocationRecord** > **relocation_records_**

Friends

- std::istream & **operator>>** (std::istream &is, **SectionData** &**data**)
  Loads the object from stream.

Detailed Description

Class representing the data about one section.

Definition at line 13 of file SectionData.h.

Member Function Documentation

uint32_t bnssemulator::SectionData::address () const `[noexcept]`

Gets the starting address of the section.

Returns:

Starting address of the section

Definition at line 44 of file SectionData.cpp.

References org_address_.

Referenced by address().

```
                                            {
        return org_address_;
    }
```

void bnssemulator::SectionData::address (uint32_t *address*) `[noexcept]`

Sets the starting address of the section.

Returns:

Starting address of the section

Definition at line 48 of file SectionData.cpp.

References address(), org_address_, and org_valid_.

```
                                                  {
        org_valid_ = true;
        org_address_ = address;
    }
```

std::vector< uint8_t > & bnssemulator::SectionData::data () `[noexcept]`

Gets the data vector of the section.

Returns:

Data vector of the section

Definition at line 61 of file SectionData.cpp.

References data_.

Referenced by data().

```
                                               {
        return data_;
    }
```

const std::vector< uint8_t > & bnssemulator::SectionData::data () const `[noexcept]`

Gets the data vector of the section.

Returns:

Data vector of the section

Definition at line 65 of file SectionData.cpp.

References data().

```
   65                                                           {
   66         return const_cast<SectionData &>(*this).data();
   67     }
```

bool bnssemulator::SectionData::hasAddress () const `[noexcept]`

Checks whether the section already has a starting address.

Returns:

Whether the section already has a starting address

Definition at line 40 of file SectionData.cpp.

References org_valid_.

```
   40                                              {
   41         return org_valid_;
   42     }
```

std::vector< RelocationRecord > & bnssemulator::SectionData::relocations () `[noexcept]`

Gets the relocation records for the section.

Returns:

Relocation records for the section

Definition at line 69 of file SectionData.cpp.

References relocation_records_.

Referenced by relocations().

```
   69                                                                {
   70         return relocation_records_;
   71     }
```

const std::vector< RelocationRecord > & bnssemulator::SectionData::relocations () const `[noexcept]`

Gets the relocation records for the section.

Returns:

Relocation records for the section

Definition at line 73 of file SectionData.cpp.

References relocations().

```
   73
{
   74         return const_cast<SectionData &>(*this).relocations();
   75     }
```

size_t bnssemulator::SectionData::size () const `[noexcept]`

Gets the size of the sections.

Returns:

Size of the section

Definition at line 53 of file SectionData.cpp.

References location_counter_.

```
                                          {
        return location_counter_;
    }
```

SectionType bnssemulator::SectionData::type () const `[noexcept]`

Gets the type of the section.

Returns:

Type of the section

Definition at line 57 of file SectionData.cpp.

References type_.

```
                                               {
        return type_;
    }
```

Friends And Related Function Documentation

std::istream& operator>> (std::istream & *is*, SectionData & *data*) `[friend]`

Loads the object from stream.

Parameters:

is	Input stream
data	Reference to the object that should be loaded

Returns:

Input stream

Definition at line 6 of file SectionData.cpp.

```
                                                                {
        int type;
        is >> type;
        data.type_ = static_cast<SectionType>(type);
        is >> data.indexed_;
        if (data.indexed_) {
            is >> data.index_;
        }

        is >> data.org_valid_;
        if (data.org_valid_) {
            is >> data.org_address_;
        }

        is >> data.location_counter_;
        size_t data_size;
        is >> data_size;
        for (size_t i = 0; i < data_size; i++) {
            int data_byte;
            is >> data_byte;
            data.data_.push_back(static_cast<int8_t>(data_byte));
        }

        size_t relocation_records_size;
        is >> relocation_records_size;
        for (size_t i = 0; i < relocation_records_size; i++) {
            RelocationRecord relocation_record;
            is >> relocation_record;
            data.relocation_records_.push_back(relocation_record);
        }

        return is;
    }
```

Member Data Documentation

std::vector<uint8_t> bnssemulator::SectionData::data_ `[private]`

Definition at line 83 of file SectionData.h.

Referenced by data(), and bnssemulator::operator>>().

size_t bnssemulator::SectionData::index_ = 0 `[private]`

Definition at line 79 of file SectionData.h.

Referenced by bnssemulator::operator>>().

bool bnssemulator::SectionData::indexed_ = false `[private]`

Definition at line 78 of file SectionData.h.

Referenced by bnssemulator::operator>>().

size_t bnssemulator::SectionData::location_counter_ = 0 `[private]`

Definition at line 80 of file SectionData.h.

Referenced by bnssemulator::operator>>(), and size().

uint32_t bnssemulator::SectionData::org_address_ = 0 `[private]`

Definition at line 81 of file SectionData.h.

Referenced by address(), and bnssemulator::operator>>().

bool bnssemulator::SectionData::org_valid_ = false `[private]`

Definition at line 82 of file SectionData.h.

Referenced by address(), hasAddress(), and bnssemulator::operator>>().

std::vector<RelocationRecord> bnssemulator::SectionData::relocation_records_ `[private]`

Definition at line 84 of file SectionData.h.

Referenced by bnssemulator::operator>>(), and relocations().

SectionType bnssemulator::SectionData::type_ = TEXT `[private]`

Definition at line 77 of file SectionData.h.

Referenced by bnssemulator::operator>>(), and type().

The documentation for this class was generated from the following files:

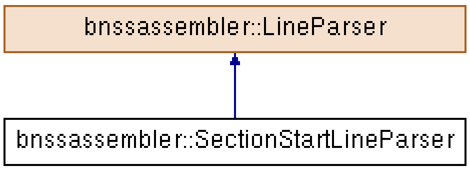
- Code/Emulator/Include/**SectionData.h**
- Code/Emulator/Source/**SectionData.cpp**

bnssassembler::SectionStartLineParser Class Reference

Class used for parsing section start definitions.

`#include <SectionStartLineParser.h>`

Inheritance diagram for bnssassembler::SectionStartLineParser:


bnssassembler::LineParser
bnssassembler::SectionStartLineParser

Protected Member Functions

- std::shared_ptr< **Token** > **parse** (const std::string &line, size_t line_number, std::string initial_line) const override
  Parses one line of the file. Does not call the next parser in chain.

Additional Inherited Members

Detailed Description

Class used for parsing section start definitions.

Definition at line 10 of file SectionStartLineParser.h.

Member Function Documentation

std::shared_ptr< Token > bnssassembler::SectionStartLineParser::parse (const std::string & *line*, size_t *line_number*, std::string *initial_line*) const `[override]`, `[protected]`, `[virtual]`

Parses one line of the file. Does not call the next parser in chain.

Parameters:

line	Line to parse
line_number	Number of the line that is parsed
initial_line	Initial line that is parsed

Returns:

Extracted token from line or nullptr if the parser failed parsing the line

Exceptions:

Throws	if the parser failed and identified the error

Implements **bnssassembler::LineParser** (*p.257*).

Definition at line 9 of file SectionStartLineParser.cpp.

References bnssassembler::SectionTypeParser::parse().

```
{
        static std::regex
regex("[[:space:]]*[.]([a-zA-Z]*)([.]([0-9][0-9]*))?[[:space:]]*");

        if (!regex_match(line, regex)) {
            return nullptr;
        }

        auto section_name_string = regex_replace(line, regex, "$1");
        auto section_number_string = regex_replace(line, regex, "$3");

        auto section = SectionTypeParser::parse(section_name_string);

        if (section_number_string.empty()) {
            return std::make_shared<SectionStartToken>(section, line_number, 
initial_line);
        }

        auto number = 
StringHelper::parseNumber<size_t>(section_number_string);
        return std::make_shared<SectionStartToken>(section, line_number, 
initial_line, number);
    }
```

The documentation for this class was generated from the following files:

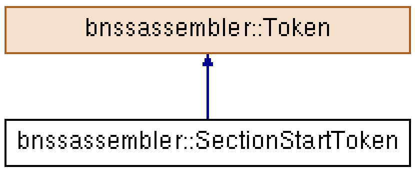
- Code/Assembler/Include/**SectionStartLineParser.h**
- Code/Assembler/Source/**SectionStartLineParser.cpp**

bnssassembler::SectionStartToken Class Reference

Class representing the section start token.

`#include <SectionStartToken.h>`

Inheritance diagram for bnssassembler::SectionStartToken:



Public Member Functions

- **SectionStartToken** (**SectionType** type, size_t line_number, std::string **line**) noexcept
  Constructs a non-indexed section start token.
- **SectionStartToken** (**SectionType** type, size_t line_number, std::string **line**, size_t index) noexcept
  Constructs an indexed section start token.
- void **firstPass** (**FirstPassData** &data) const override
  Executes the first pass over the token.
- void **secondPass** (**SecondPassData** &data) const override
  Executes the second pass over the token.
- bool **usesAddress** () const noexcept override
  Check whether the token can use the ORG address.

Private Attributes

- **SectionType type_**
- bool **indexed_** = false
- size_t **index_** = 0

Detailed Description

Class representing the section start token.

Definition at line 11 of file SectionStartToken.h.

Constructor & Destructor Documentation

bnssassembler::SectionStartToken::SectionStartToken (SectionType *type*, size_t *line_number*, std::string *line*) `[noexcept]`

Constructs a non-indexed section start token.

Parameters:

type	Type of the section
line_number	Line number where the section was defined
line	Line where the section was defined

Definition at line 6 of file SectionStartToken.cpp.

```
    6 : Token(line_number, line), type_(type) {}
```

bnssassembler::SectionStartToken::SectionStartToken (SectionType *type*, size_t *line_number*, std::string *line*, size_t *index*)`[noexcept]`

Constructs an indexed section start token.

Parameters:

type	Type of the section
line_number	Line number where the section was defined
line	Line where the section was defined
index	Index of section

Definition at line 8 of file SectionStartToken.cpp.

```
: Token(line_number, line), type_(type), indexed_(true), index_(index) {}
```

Member Function Documentation

void bnssassembler::SectionStartToken::firstPass (FirstPassData & *data*) const`[override]`, `[virtual]`

Executes the first pass over the token.

Parameters:

data	**Data** that the token will modify

Implements **bnssassembler::Token** (*p.510*).

Definition at line 10 of file SectionStartToken.cpp.

References index_, indexed_, bnssassembler::FirstPassData::insertSection(), and type_.

```
                                                                {
        if (indexed_) {
            data.insertSection(type_, index_);
        }
        else {
            data.insertSection(type_);
        }
    }
```

void bnssassembler::SectionStartToken::secondPass (SecondPassData & *data*) const`[override]`, `[virtual]`

Executes the second pass over the token.

Parameters:

data	**Data** that the token will modify

Implements **bnssassembler::Token** (*p.512*).

Definition at line 19 of file SectionStartToken.cpp.

References bnssassembler::SecondPassData::nextSection().

```
                                                                  {
        data.nextSection();
    }
```

bool bnssassembler::SectionStartToken::usesAddress () const`[override]`, `[virtual]`, `[noexcept]`

Check whether the token can use the ORG address.

Reimplemented from **bnssassembler::Token** (*p.512*).

Definition at line 23 of file SectionStartToken.cpp.

```
                                                                  {
        return true;
    }
```

Member Data Documentation

size_t bnssassembler::SectionStartToken::index_ = 0 `[private]`

Definition at line 36 of file SectionStartToken.h.

Referenced by firstPass().

bool bnssassembler::SectionStartToken::indexed_ = false `[private]`

Definition at line 35 of file SectionStartToken.h.

Referenced by firstPass().

SectionType bnssassembler::SectionStartToken::type_ `[private]`

Definition at line 34 of file SectionStartToken.h.

Referenced by firstPass().

The documentation for this class was generated from the following files:

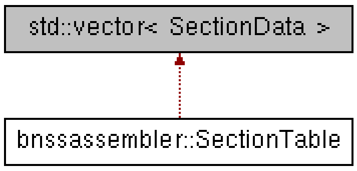
- Code/Assembler/Include/**SectionStartToken.h**
- Code/Assembler/Source/**SectionStartToken.cpp**

bnssassembler::SectionTable Class Reference

Class representing the table of sections.

`#include <SectionTable.h>`

Inheritance diagram for bnssassembler::SectionTable:



Public Member Functions

- **SectionTable** & **operator+=** (const **SectionData** &data)
  Insert new section into the table.
- void **nextSection** () noexcept
  Increasses the current section index.
- **SectionType currentSectionType** () const noexcept
  Gets the current section type.
- void **addData** (uint8_t data, std::list< **RelocationRecord** > &relocations)
  Adds 8 bits of data to the current section.
- void **addData** (uint16_t data, std::list< **RelocationRecord** > &relocations)
  Adds 16 bits of data to the current section.
- void **addData** (**uint32_t** data, std::list< **RelocationRecord** > &relocations)
  Adds 32 bits of data to the current section.
- **SectionData** & **current** () noexcept
  Gets the current section.
- const **SectionData** & **current** () const noexcept
  Gets the current section.
- size_t **currentIndex** () const noexcept
  Gets the index of current section.

Private Attributes

- std::unordered_set< **SectionData** > **set_**
- size_t **current_index_** = static_cast<size_t>(-1)

Friends

- std::ostream & **operator<<** (std::ostream &os, const **SectionTable** &section_table)
  Writes the content of the object to a stream.

Detailed Description

Class representing the table of sections.

Definition at line 12 of file SectionTable.h.

Member Function Documentation

void bnssassembler::SectionTable::addData (uint8_t *data*, std::list< RelocationRecord > & *relocations*)

Adds 8 bits of data to the current section.

Parameters:

data	**Data** to be addded
relocations	List of relocation records for the data

Definition at line 29 of file SectionTable.cpp.

References current_index_.

Referenced by bnssassembler::SecondPassData::addData().

```
   29
{
   30         (*this)[current_index_].addData(data, relocations);
   31     }
```

void bnssassembler::SectionTable::addData (uint16_t *data*, std::list< RelocationRecord > & *relocations*)

Adds 16 bits of data to the current section.

Parameters:

data	**Data** to be addded
relocations	List of relocation records for the data

Definition at line 33 of file SectionTable.cpp.

References current_index_.

```
   33
{
   34         (*this)[current_index_].addData(data, relocations);
   35     }
```

void bnssassembler::SectionTable::addData (uint32_t *data*, std::list< RelocationRecord > & *relocations*)

Adds 32 bits of data to the current section.

Parameters:

data	**Data** to be addded
relocations	List of relocation records for the data

Definition at line 37 of file SectionTable.cpp.

References current_index_.

```
   37
{
   38         (*this)[current_index_].addData(data, relocations);
   39     }
```

SectionData & bnssassembler::SectionTable::current () `[noexcept]`

Gets the current section.

Returns:

Current section

Definition at line 41 of file SectionTable.cpp.

References current_index_.

Referenced by current(), bnssassembler::SecondPassData::currentSectionOffset(), and bnssassembler::SecondPassData::nextSection().

```
                                         {
        return (*this)[current_index_];
    }
```

const SectionData & bnssassembler::SectionTable::current () const `[noexcept]`

Gets the current section.

Returns:

Current section

Definition at line 45 of file SectionTable.cpp.

References current().

```
                                                  {
        return const_cast<SectionTable &>(*this).current();
    }
```

size_t bnssassembler::SectionTable::currentIndex () const `[noexcept]`

Gets the index of current section.

Returns:

Index of current section

Definition at line 49 of file SectionTable.cpp.

References current_index_.

Referenced by bnssassembler::SecondPassData::currentSectionIndex().

```
                                            {
        return current_index_;
    }
```

SectionType bnssassembler::SectionTable::currentSectionType () const `[noexcept]`

Gets the current section type.

Returns:

Current section type

Definition at line 25 of file SectionTable.cpp.

References current_index_.

Referenced by bnssassembler::SecondPassData::currentSectionType().

```
                                                       {
        return (*this)[current_index_].type();
    }
```

void bnssassembler::SectionTable::nextSection () `[noexcept]`

Increasses the current section index.

Definition at line 21 of file SectionTable.cpp.

References current_index_.

Referenced by bnssassembler::SecondPassData::nextSection().

```
   21                                    {
   22         current_index_++;
   23     }
```

SectionTable & bnssassembler::SectionTable::operator+= (const SectionData & *data*)

Insert new section into the table.

Parameters:

data	Section data to be inserted

Returns:

Reference to this section table after the insertion

Definition at line 10 of file SectionTable.cpp.

References bnssassembler::SectionData::index(), bnssassembler::SectionData::indexed(), set_, bnssassembler::SectionTypeParser::toString(), and bnssassembler::SectionData::type().

```
   10                                                                  {
   11         if (set_.count(data) > 0) {
   12             auto section_string = "." +
SectionTypeParser::toString(data.type()) + (data.indexed() ? "." + data.index() : "");
   13             throw MessageException("Section " + section_string + " already
exists");
   14         }
   15 
   16         set_.insert(data);
   17         push_back(data);
   18         return *this;
   19     }
```

Friends And Related Function Documentation

std::ostream& operator<< (std::ostream & *os*, const SectionTable & *section_table*)`[friend]`

Writes the content of the object to a stream.

Parameters:

os	Stream where the content will be written
section_table	**Data** that will be written

Definition at line 53 of file SectionTable.cpp.

```
   53 
{
   54         os << section_table.size() << std::endl;
   55 
   56         std::cout << UPPER_LEFT << multiple(HORIZONTAL, 81) << UPPER_RIGHT <<
std::endl;
   57         std::cout << VERTICAL << UPPER_LEFT << multiple(HORIZONTAL, 79) <<
UPPER_RIGHT << VERTICAL << std::endl;
   58         std::cout << VERTICAL << VERTICAL << std::setw(79) << std::left << "
Section table:" << VERTICAL << VERTICAL << std::endl;
   59         std::cout << VERTICAL << LOWER_LEFT << multiple(HORIZONTAL, 79) <<
LOWER_RIGHT << VERTICAL << std::endl;
   60         std::cout << LOWER_LEFT << multiple(HORIZONTAL, 81) << LOWER_RIGHT <<
std::endl;
```

```

        for (auto &section : section_table) {
            os << section << std::endl;
        }

        std::cout << std::endl << std::endl;

        return os;
    }
```

Member Data Documentation

size_t bnssassembler::SectionTable::current_index_ = static_cast<size_t>(-1) [private]

Definition at line 83 of file SectionTable.h.

Referenced by addData(), current(), currentIndex(), currentSectionType(), and nextSection().

std::unordered_set<SectionData> bnssassembler::SectionTable::set_ [private]

Definition at line 82 of file SectionTable.h.

Referenced by operator+=().

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**SectionTable.h**
- Code/Assembler/Source/**SectionTable.cpp**

bnssassembler::SectionTypeParser Class Reference

Utility class representing the parser for the section types.
`#include <SectionTypeParser.h>`

Classes

- struct **SectionTypeParserData**

Static Public Member Functions

- static **SectionType parse** (std::string type)
  Parses the section type.
- static std::string **toString** (**SectionType** type) noexcept
  Converts a SectionType to string.

Static Private Member Functions

- static **SectionTypeParserData** & **staticData** () noexcept

---

Detailed Description

Utility class representing the parser for the section types.

Definition at line 12 of file SectionTypeParser.h.

Member Function Documentation

SectionType bnssassembler::SectionTypeParser::parse (std::string *type*) `[static]`

Parses the section type.

Parameters:

type	String representing the section

Returns:

SectionType enum

Exceptions:

Throws	if the section is invalid

Definition at line 6 of file SectionTypeParser.cpp.

References bnssassembler::SectionTypeParser::SectionTypeParserData::map, and staticData().

Referenced by bnssassembler::SectionStartLineParser::parse().

```
                                                    {
        transform(type.begin(), type.end(), type.begin(), tolower);

        if (staticData().map.count(type) == 0) {
            throw MessageException(type + " is not a valid section");
        }

        return staticData().map[type];
    }
```

SectionTypeParser::SectionTypeParserData & bnssassembler::SectionTypeParser::staticData () `[static]`, `[private]`, `[noexcept]`

Definition at line 20 of file SectionTypeParser.cpp.

Referenced by parse(), and toString().

```
{
        static SectionTypeParserData static_data;
        return static_data;
    }
```

std::string bnssassembler::SectionTypeParser::toString (SectionType *type*) `[static]`, `[noexcept]`

Converts a SectionType to string.

Parameters:

type	SectionType object

Returns:

String representation of the type

Definition at line 16 of file SectionTypeParser.cpp.

References bnssassembler::SectionTypeParser::SectionTypeParserData::reverse, and staticData().

Referenced by bnssassembler::SectionTable::operator+=().

```
                                                                     {
        return staticData().reverse[type];
    }
```

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**SectionTypeParser.h**
- Code/Assembler/Source/**SectionTypeParser.cpp**

bnssassembler::SectionTypeParser::SectionTypeParserData Struct Reference

Public Member Functions

- **SectionTypeParserData** ()

Public Attributes

- std::unordered_map< std::string, **SectionType** > **map**
- std::unordered_map< **SectionType**, std::string > **reverse**

Detailed Description

Definition at line 29 of file SectionTypeParser.h.

Constructor & Destructor Documentation

bnssassembler::SectionTypeParser::SectionTypeParserData::SectionTypeParserData ()

Definition at line 25 of file SectionTypeParser.cpp.

References bnssassembler::BSS, bnssassembler::DATA, bnssassembler::RODATA, and bnssassembler::TEXT.

```
                                                           {
        map["text"] = TEXT;
        map["data"] = DATA;
        map["rodata"] = RODATA;
        map["bss"] = BSS;

        reverse[TEXT] = "text";
        reverse[DATA] = "data";
        reverse[RODATA] = "rodata";
        reverse[BSS] = "bss";
    }
```

Member Data Documentation

std::unordered_map<std::string, SectionType> bnssassembler::SectionTypeParser::SectionTypeParserData::map

Definition at line 30 of file SectionTypeParser.h.

Referenced by bnssassembler::SectionTypeParser::parse().

std::unordered_map<SectionType, std::string> bnssassembler::SectionTypeParser::SectionTypeParserData::reverse

Definition at line 31 of file SectionTypeParser.h.

Referenced by bnssassembler::SectionTypeParser::toString().

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

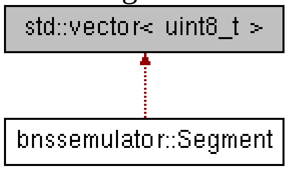
- Code/Assembler/Include/**SectionTypeParser.h**
- Code/Assembler/Source/**SectionTypeParser.cpp**

bnssemulator::Segment Class Reference

Class representing one segment of memory.

`#include <Segment.h>`

Inheritance diagram for bnssemulator::Segment:



Public Member Functions

- **Segment** (**uint32_t** address, size_t length, **SectionType** type, vector< uint8_t > &&data)
  Constructs a segment.
- **InstructionBitField getInstruction** (**uint32_t** address) const
  Gets the instruction at the specified address.
- int32_t **getSecondWordOfInstruction** (**uint32_t** address) const
  Gets the second word of the instruction at the specified address.
- uint8_t **readData** (**uint32_t** address) const
  Reads a byte of data at the specified address.
- void **writeData** (**uint32_t** address, uint8_t data)
  Writes a byte of data at the specified address.
- void **relocate** (**uint32_t** address, **uint32_t** relocation)
  Fixes the value at the address by a relocation.

Private Attributes

- **uint32_t address_**
- size_t **length_**
- **SectionType type_**

Detailed Description

Class representing one segment of memory.

Definition at line 13 of file Segment.h.

Constructor & Destructor Documentation

bnssemulator::Segment::Segment (uint32_t *address*, size_t *length*, SectionType *type*, vector< uint8_t > && *data*)

Constructs a segment.

Parameters:

address	Starting address of the segment
length	Length of the segment
type	Type of the segment
data	Data contained in the segment

Definition at line 8 of file Segment.cpp.

```
: vector(move(data)), address_(address), length_(length), type_(type) {
        if (size() == 0) {
            resize(length);
        }
    }
```

Member Function Documentation

InstructionBitField bnssemulator::Segment::getInstruction (uint32_t *address*) const

Gets the instruction at the specified address.

Parameters:

address	Address

Returns:

Instruction

Definition at line 14 of file Segment.cpp.

References address_, bnssemulator::InstructionBitFieldUnion::bit_field, bnssemulator::InstructionBitFieldUnion::data, length_, bnssemulator::TEXT, bnssemulator::StringHelper::toHexString(), and type_.

Referenced by bnssemulator::AddressSpace::getInstruction().

```
                                                                       {
        if (type_ != TEXT) {
            throw MessageException("No execute permission at address " +
StringHelper::toHexString(address));
        }

        if (address < address_ || address + 4 > address_ + length_) {
            throw MessageException("Address " +
StringHelper::toHexString(address) + " is out of range");
        }

        auto offset = address - address_;
        InstructionBitFieldUnion ret;
        ret.data = 0;

        ret.data |= (*this)[offset];
        ret.data |= (*this)[offset + 1] << 8;
        ret.data |= (*this)[offset + 2] << 16;
        ret.data |= (*this)[offset + 3] << 24;

        return ret.bit_field;
    }
```

int32_t bnssemulator::Segment::getSecondWordOfInstruction (uint32_t *address*) const

Gets the second word of the instruction at the specified address.

Parameters:

address	Address

Returns:

Second word of the instruction

Definition at line 35 of file Segment.cpp.

References address_, length_, bnssemulator::TEXT, bnssemulator::StringHelper::toHexString(), and type_.

Referenced by bnssemulator::AddressSpace::getSecondWordOfInstruction().

```
   35                                                                      {
   36         if (type_ != TEXT) {
   37             throw MessageException("No execute permission at address " +
StringHelper::toHexString(address));
   38         }
   39 
   40         if (address < address_ || address + 4 > address_ + length_) {
   41             throw MessageException("Address " +
StringHelper::toHexString(address) + " is out of range");
   42         }
   43 
   44         auto offset = address - address_;
   45         // ReSharper disable once CppUseAuto
   46         int32_t ret = 0;
   47 
   48         ret |= (*this)[offset];
   49         ret |= (*this)[offset + 1] << 8;
   50         ret |= (*this)[offset + 2] << 16;
   51         ret |= (*this)[offset + 3] << 24;
   52 
   53         return ret;
   54     }
```

uint8_t bnssemulator::Segment::readData (uint32_t *address*) const

Reads a byte of data at the specified address.

Parameters:

address	Address

Returns:

Byte of read data

Definition at line 56 of file Segment.cpp.

References address_, length_, and bnssemulator::StringHelper::toHexString().

Referenced by bnssemulator::AddressSpace::get8bitData().

```
   56                                                     {
   57         if (address < address_ || address > address_ + length_) {
   58             throw MessageException("Address " +
StringHelper::toHexString(address) + " is out of range");
   59         }
   60 
   61         auto offset = address - address_;
   62         return (*this)[offset];
   63     }
```

void bnssemulator::Segment::relocate (uint32_t *address*, uint32_t *relocation*)

Fixes the value at the address by a relocation.

Parameters:

address	Address
relocation	Value to be added to the value at the address

Definition at line 78 of file Segment.cpp.

References address_, length_, and bnssemulator::StringHelper::toHexString().

```
   78                                                                {
   79         if (address < address_ || address + 4 > address_ + length_) {
   80             throw MessageException("Address " +
StringHelper::toHexString(address) + " is out of range");
   81         }
   82 
   83         auto offset = address - address_;
   84
   85         uint32_t data = 0;
   86
   87         data |= (*this)[offset];
   88         data |= (*this)[offset + 1] << 8;
   89         data |= (*this)[offset + 2] << 16;
   90         data |= (*this)[offset + 3] << 24;
   91
   92         data += relocation;
   93
   94         (*this)[offset] = static_cast<uint8_t>(data & 0x000000ff);
   95         (*this)[offset + 1] = static_cast<uint8_t>((data & 0x0000ff00) >> 8);
   96         (*this)[offset + 2] = static_cast<uint8_t>((data & 0x00ff0000) >> 16);
   97         (*this)[offset + 3] = static_cast<uint8_t>((data & 0xff000000) >> 24);
   98     }
```

void bnssemulator::Segment::writeData (uint32_t *address*, uint8_t *data*)

Writes a byte of data at the specified address.

Parameters:

address	Address
data	Byte of data to be written

Definition at line 65 of file Segment.cpp.

References address_, bnssemulator::BSS, bnssemulator::DATA, length_, bnssemulator::StringHelper::toHexString(), and type_.

Referenced by bnssemulator::AddressSpace::set8bitData().

```
   65                                                          {
   66         if (type_ != DATA && type_ != BSS) {
   67             throw MessageException("No write permission at address " +
StringHelper::toHexString(address));
   68         }
   69
   70         if (address < address_ || address > address_ + length_) {
   71             throw MessageException("Address " +
StringHelper::toHexString(address) + " is out of range");
   72         }
   73
   74         auto offset = address - address_;
   75         (*this)[offset] = data;
   76     }
```

Member Data Documentation

uint32_t bnssemulator::Segment::address_ `[private]`

Definition at line 62 of file Segment.h.

Referenced by getInstruction(), getSecondWordOfInstruction(), readData(), relocate(), and writeData().

size_t bnssemulator::Segment::length_ `[private]`

Definition at line 63 of file Segment.h.

Referenced by getInstruction(), getSecondWordOfInstruction(), readData(), relocate(), and writeData().

SectionType bnssemulator::Segment::type_ `[private]`

Definition at line 64 of file Segment.h.

Referenced by getInstruction(), getSecondWordOfInstruction(), and writeData().

The documentation for this class was generated from the following files:

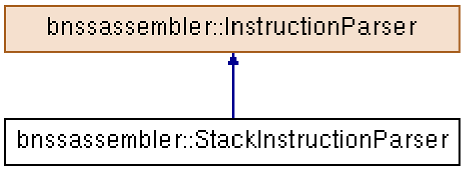
- Code/Emulator/Include/**Segment.h**
- Code/Emulator/Source/**Segment.cpp**

bnssassembler::StackInstructionParser Class Reference

Class representing the parser for stack instructions.

`#include <StackInstructionParser.h>`

Inheritance diagram for bnssassembler::StackInstructionParser:


bnssassembler::InstructionParser
bnssassembler::StackInstructionParser

Public Member Functions

- **StackInstructionParser** () noexcept
  *Constructs a **StackInstructionParser** object.*

Additional Inherited Members

Detailed Description

Class representing the parser for stack instructions.

Definition at line 10 of file StackInstructionParser.h.

Constructor & Destructor Documentation

bnssassembler::StackInstructionParser::StackInstructionParser () `[noexcept]`

Constructs a **StackInstructionParser** object.

Definition at line 6 of file StackInstructionParser.cpp.

References bnssassembler::InstructionParser::operands_.

```
                                                            {
        operands_.push_back(std::make_shared<RegisterDirectParser>());
    }
```

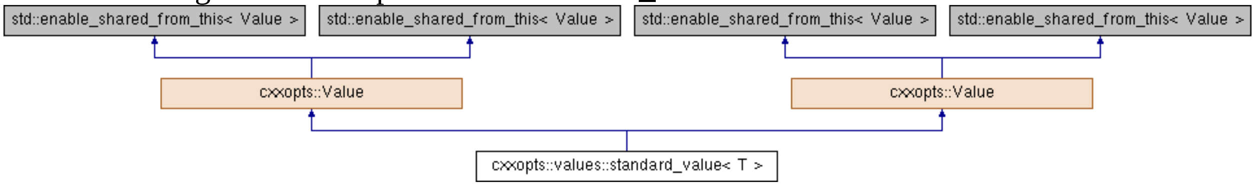
The documentation for this class was generated from the following files:

- Code/Assembler/Include/**StackInstructionParser.h**
- Code/Assembler/Source/**StackInstructionParser.cpp**

cxxopts::values::standard_value< T > Class Template Reference

`#include <cxxopts.h>`

Inheritance diagram for cxxopts::values::standard_value< T >:


std::enable_shared_from_this< Value >
std::enable_shared_from_this< Value >
std::enable_shared_from_this< Value >
std::enable_shared_from_this< Value >
cxxopts::Value
cxxopts::Value
cxxopts::values::standard_value< T >

Public Member Functions

- **standard_value** ()
- **standard_value** (T *t)
- void **parse** (const std::string &text) const override
- bool **is_container** () const override
- void **parse** () const override
- bool **has_arg** () const override
- bool **has_default** () const override
- bool **has_implicit** () const override
- std::shared_ptr< **Value** > **default_value** (const std::string &**value**) override
- std::shared_ptr< **Value** > **implicit_value** (const std::string &**value**) override
- std::string **get_default_value** () const override
- std::string **get_implicit_value** () const override
- const T & **get** () const
- **standard_value** ()
- **standard_value** (T *t)
- void **parse** (const std::string &text) const override
- bool **is_container** () const override
- void **parse** () const override
- bool **has_arg** () const override
- bool **has_default** () const override
- bool **has_implicit** () const override
- std::shared_ptr< **Value** > **default_value** (const std::string &**value**) override
- std::shared_ptr< **Value** > **implicit_value** (const std::string &**value**) override
- std::string **get_default_value** () const override
- std::string **get_implicit_value** () const override
- const T & **get** () const

Protected Attributes

- std::shared_ptr< T > **m_result**
- T * **m_store**
- bool **m_default** = false
- std::string **m_default_value**
- bool **m_implicit** = false
- std::string **m_implicit_value**

Detailed Description

template<typename T>

class cxxopts::values::standard_value< T >

Definition at line 474 of file cxxopts.h.

Constructor & Destructor Documentation

template<typename T > cxxopts::values::standard_value< T >::standard_value ()[inline]

Definition at line 477 of file cxxopts.h.

```
      : m_result(std::make_shared<T>())
      , m_store(m_result.get())
    {
    }
```

template<typename T > cxxopts::values::standard_value< T >::standard_value (T * *t*)[inline], [explicit]

Definition at line 483 of file cxxopts.h.

```
      : m_store(t)
    {
    }
```

template<typename T > cxxopts::values::standard_value< T >::standard_value ()[inline]

Definition at line 477 of file cxxopts.h.

```
      : m_result(std::make_shared<T>())
      , m_store(m_result.get())
    {
    }
```

template<typename T > cxxopts::values::standard_value< T >::standard_value (T * *t*)[inline], [explicit]

Definition at line 483 of file cxxopts.h.

```
      : m_store(t)
    {
    }
```

Member Function Documentation

template<typename T > std::shared_ptr<Value> cxxopts::values::standard_value< T >::default_value (const std::string & *value*)[inline], [override], [virtual]

Implements **cxxopts::Value** (*p.517*).

Definition at line 519 of file cxxopts.h.

References cxxopts::value().

```
                                                            {
                m_default = true;
                m_default_value = value;
                return shared_from_this();
            }
```

template<typename T > std::shared_ptr<Value> cxxopts::values::standard_value< T >::default_value (const std::string & *value*)`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.517*).

Definition at line 519 of file cxxopts.h.

References cxxopts::value().

```
                                                            {
                m_default = true;
                m_default_value = value;
                return shared_from_this();
            }
```

template<typename T > const T& cxxopts::values::standard_value< T >::get () const`[inline]`

Definition at line 543 of file cxxopts.h.

Referenced by cxxopts::OptionDetails::as().

```
            {
                if (m_store == nullptr)
                {
                    return *m_result;
                }
                else
                {
                    return *m_store;
                }
            }
```

template<typename T > const T& cxxopts::values::standard_value< T >::get () const`[inline]`

Definition at line 543 of file cxxopts.h.

References cxxopts::value().

```
            {
                if (m_store == nullptr)
                {
                    return *m_result;
                }
                else
                {
                    return *m_store;
                }
            }
```

template<typename T > std::string cxxopts::values::standard_value< T >::get_default_value () const`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.517*).

Definition at line 533 of file cxxopts.h.

```
                                                {
                return m_default_value;
            }
```

template<typename T > std::string cxxopts::values::standard_value< T >::get_default_value () const`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.517*).

Definition at line 533 of file cxxopts.h.

```
533                                            {
534                 return m_default_value;
535             }
```

template<typename T > std::string cxxopts::values::standard_value< T >::get_implicit_value () const`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.517*).

Definition at line 538 of file cxxopts.h.

```
538                                             {
539                 return m_implicit_value;
540             }
```

template<typename T > std::string cxxopts::values::standard_value< T >::get_implicit_value () const`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.517*).

Definition at line 538 of file cxxopts.h.

```
538                                             {
539                 return m_implicit_value;
540             }
```

template<typename T > bool cxxopts::values::standard_value< T >::has_arg () const`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.517*).

Definition at line 504 of file cxxopts.h.

```
504                               {
505                 return value_has_arg<T>::value;
506             }
```

template<typename T > bool cxxopts::values::standard_value< T >::has_arg () const`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.517*).

Definition at line 504 of file cxxopts.h.

```
504                               {
505                 return value_has_arg<T>::value;
506             }
```

template<typename T > bool cxxopts::values::standard_value< T >::has_default () const`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.518*).

Definition at line 509 of file cxxopts.h.

```
509                                   {
510                 return m_default;
511             }
```

template<typename T > bool cxxopts::values::standard_value< T >::has_default () const`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.518*).

Definition at line 509 of file cxxopts.h.

```
                                              {
                    return m_default;
                }
```

template<typename T > bool cxxopts::values::standard_value< T >::has_implicit () const`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.518*).

Definition at line 514 of file cxxopts.h.

```
                                               {
                    return m_implicit;
                }
```

template<typename T > bool cxxopts::values::standard_value< T >::has_implicit () const`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.518*).

Definition at line 514 of file cxxopts.h.

```
                                               {
                    return m_implicit;
                }
```

template<typename T > std::shared_ptr<Value> cxxopts::values::standard_value< T >::implicit_value (const std::string & *value*)`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.518*).

Definition at line 526 of file cxxopts.h.

References cxxopts::value().

```
                                                                   {
                    m_implicit = true;
                    m_implicit_value = value;
                    return shared_from_this();
                }
```

template<typename T > std::shared_ptr<Value> cxxopts::values::standard_value< T >::implicit_value (const std::string & *value*)`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.518*).

Definition at line 526 of file cxxopts.h.

References cxxopts::value().

```
                                                                   {
                    m_implicit = true;
                    m_implicit_value = value;
                    return shared_from_this();
                }
```

template<typename T > bool cxxopts::values::standard_value< T >::is_container () const`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.518*).

Definition at line 494 of file cxxopts.h.

```
494                                     {
495                 return type_is_container<T>::value;
496             }
```

template<typename T > bool cxxopts::values::standard_value< T >::is_container () const`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.518*).

Definition at line 494 of file cxxopts.h.

```
494                                     {
495                 return type_is_container<T>::value;
496             }
```

template<typename T > void cxxopts::values::standard_value< T >::parse (const std::string & *text*) const`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.518*).

Definition at line 489 of file cxxopts.h.

References cxxopts::values::parse_value().

```
489                                                     {
490                 parse_value(text, *m_store);
491             }
```

template<typename T > void cxxopts::values::standard_value< T >::parse (const std::string & *text*) const`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.518*).

Definition at line 489 of file cxxopts.h.

References cxxopts::values::parse_value().

```
489                                                     {
490                 parse_value(text, *m_store);
491             }
```

template<typename T > void cxxopts::values::standard_value< T >::parse () const`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.519*).

Definition at line 499 of file cxxopts.h.

References cxxopts::values::parse_value().

```
499                             {
500                 parse_value(m_default_value, *m_store);
501             }
```

template<typename T > void cxxopts::values::standard_value< T >::parse () const`[inline]`, `[override]`, `[virtual]`

Implements **cxxopts::Value** (*p.519*).

Definition at line 499 of file cxxopts.h.

References cxxopts::values::parse_value().

```
499                             {
500                 parse_value(m_default_value, *m_store);
501             }
```

Member Data Documentation

template<typename T > bool cxxopts::values::standard_value< T >::m_default = false`[protected]`

Definition at line 558 of file cxxopts.h.

template<typename T > std::string cxxopts::values::standard_value< T >::m_default_value`[protected]`

Definition at line 559 of file cxxopts.h.

template<typename T > bool cxxopts::values::standard_value< T >::m_implicit = false`[protected]`

Definition at line 560 of file cxxopts.h.

template<typename T > std::string cxxopts::values::standard_value< T >::m_implicit_value`[protected]`

Definition at line 561 of file cxxopts.h.

template<typename T > std::shared_ptr< T > cxxopts::values::standard_value< T >::m_result`[protected]`

Definition at line 556 of file cxxopts.h.

template<typename T > T * cxxopts::values::standard_value< T >::m_store`[protected]`

Definition at line 557 of file cxxopts.h.

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

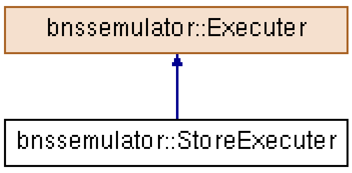
- Code/Assembler/Include/**cxxopts.h**

bnssemulator::StoreExecuter Class Reference

Class representing the executer for the store instruction.

`#include <StoreExecuter.h>`

Inheritance diagram for bnssemulator::StoreExecuter:



Public Member Functions

- void **execute** (**InstructionBitField** instruction, **Context** &context) const override
  Executes the instruction.

Detailed Description

Class representing the executer for the store instruction.

Definition at line 10 of file StoreExecuter.h.

Member Function Documentation

void bnssemulator::StoreExecuter::execute (InstructionBitField *instruction*, Context & *context*) const `[override]`, `[virtual]`

Executes the instruction.

Parameters:

instruction	Instruction
context	**Processor** context

Implements **bnssemulator::Executer** (*p.163*).

Definition at line 9 of file StoreExecuter.cpp.

References bnssemulator::InstructionBitField::address_mode, bnssemulator::Context::addressSpace(), bnssemulator::Context::getOperandAddress(), bnssemulator::Context::getRegister(), bnssemulator::InstructionBitField::register0, bnssemulator::InstructionBitField::register1, bnssemulator::REGISTER_DIRECT, bnssemulator::REGULAR_BYTE, bnssemulator::REGULAR_DOUBLE_WORD, bnssemulator::REGULAR_WORD, bnssemulator::AddressSpace::set16bitData(), bnssemulator::AddressSpace::set32bitData(), bnssemulator::AddressSpace::set8bitData(), bnssemulator::StringHelper::toHexString(), and bnssemulator::InstructionBitField::type.

```
    9
{
   10         auto &src = context.getRegister(instruction.register0);
   11
   12         if (instruction.address_mode == REGISTER_DIRECT) {
   13             auto &dst = context.getRegister(instruction.register1);
   14             dst = src;
   15         }
   16         else {
   17             auto address = context.getOperandAddress(instruction, 1);
   18             switch (instruction.type) {
   19             case REGULAR_BYTE:
```

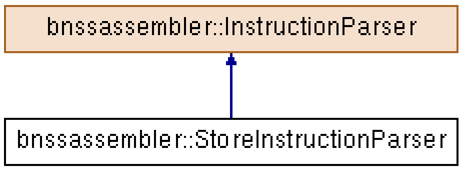
```
                context.addressSpace().set8bitData(address, src);
                break;
            case REGULAR_WORD:
                context.addressSpace().set16bitData(address, src);
                break;
            case REGULAR_DOUBLE_WORD:
                context.addressSpace().set32bitData(address, src);
                break;
            default:
                throw MessageException("Invalid instruction type: " + StringHelper::toHexString(instruction.type));
            }
        }
    }
```

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**StoreExecuter.h**
- Code/Emulator/Source/**StoreExecuter.cpp**

bnssassembler::StoreInstructionParser Class Reference

Class representing the parser for the store instruction.
`#include <StoreInstructionParser.h>`
Inheritance diagram for bnssassembler::StoreInstructionParser:


bnssassembler::InstructionParser
bnssassembler::StoreInstructionParser

Public Member Functions

- **StoreInstructionParser** () noexcept
  *Constructs a **StoreInstructionParser** object.*

Additional Inherited Members

Detailed Description

Class representing the parser for the store instruction.

Definition at line 10 of file StoreInstructionParser.h.

Constructor & Destructor Documentation

bnssassembler::StoreInstructionParser::StoreInstructionParser () `[noexcept]`

Constructs a **StoreInstructionParser** object.

Definition at line 9 of file StoreInstructionParser.cpp.

References bnssassembler::InstructionParser::operands_.

```
                                                    {
        operands_.push_back(std::make_shared<RegisterDirectParser>());

        auto memdir = std::make_shared<MemoryDirectParser>();
        auto regindpom = std::make_shared<RegisterIndirectOffsetParser>();
        auto regind = std::make_shared<RegisterIndirectParser>();
        auto regdir = std::make_shared<RegisterDirectParser>();

        memdir->next(regindpom);
        regindpom->next(regind);
        regind->next(regdir);

        operands_.push_back(memdir);

    }
```

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**StoreInstructionParser.h**
- Code/Assembler/Source/**StoreInstructionParser.cpp**

bnssemulator::StringHelper Class Reference

Utility class providing helper methods for std::string class.
`#include <StringHelper.h>`

Static Public Member Functions

- static std::string **fileToString** (std::string file_name)
  Reads the whole file into a string.
- static std::vector< std::string > **split** (std::string string, std::string delimiters) noexcept
  Splits the string using the specified delimiters.
- static std::string **join** (std::vector< std::string > strings, std::string delimiter) noexcept
  Joins the strings from a vector using the specified delimiter.
- template<typename Num > static std::string **numberFormat** (Num number) noexcept
  Converts the number to its string representation.
- template<typename Num > static std::string **toHexString** (Num number) noexcept
  Converts the number to its hex string representation.
- static std::string **toHexString** (unsigned char number) noexcept
  Converts the number to its hex string representation.
- static std::string **toHexString** (signed char number) noexcept
  Converts the number to its hex string representation.
- template<typename Num > static Num **parseNumber** (std::string number)
  Parses the number from its string representation.
- static bool **isAllWhiteSpace** (const std::string &string) noexcept
  Checks if the string contains only whitespace characters.

Private Member Functions

- **StringHelper** ()=delete
- **StringHelper** (**StringHelper** &)=delete
- void **operator=** (**StringHelper** &)=delete

Detailed Description

Utility class providing helper methods for std::string class.

Definition at line 16 of file StringHelper.h.

Constructor & Destructor Documentation

bnssemulator::StringHelper::StringHelper () `[private]`, `[delete]`

bnssemulator::StringHelper::StringHelper (StringHelper &) `[private]`, `[delete]`

Member Function Documentation

std::string bnssemulator::StringHelper::fileToString (std::string *file_name*) `[static]`

Reads the whole file into a string.

Parameters:

file_name	Name of the file

Returns:

String containing the content of the file

Exceptions:

Throws	if the file does not exist or could not be opened for reading

Definition at line 10 of file StringHelper.cpp.

Referenced by bnssemulator::FileReader::parse().

```
                                                          {
        std::ifstream file(file_name);
        if (!file.is_open()) {
            throw std::invalid_argument("File " + file_name + " does not exist");
        }

        std::stringstream ss;
        ss << file.rdbuf();
        auto ret = ss.str();
        file.close();
        return ret;
    }
```

bool bnssemulator::StringHelper::isAllWhiteSpace (const std::string & *string*)`[static]`, `[noexcept]`

Checks if the string contains only whitespace characters.

Parameters:

string	String to be checked

Returns:

Boolean value indicating whether the string contains only whitespace characters

Definition at line 72 of file StringHelper.cpp.

```
                                                                      {
        for (auto &ch : string) {
            if (!isspace(ch)) return false;
        }

        return true;
    }
```

std::string bnssemulator::StringHelper::join (std::vector< std::string > *strings*, std::string *delimiter*)`[static]`, `[noexcept]`

Joins the strings from a vector using the specified delimiter.

Parameters:

strings	Vector of strings to be joined
delimiter	Delimiter to be joined

Definition at line 41 of file StringHelper.cpp.

Referenced by bnssemulator::Context::Context().

```
{
        switch (strings.size())
        {
        case 0:
            return "";
        case 1:
            return strings[0];
        default:
            std::ostringstream os;
            copy(strings.begin(), strings.end() - 1,
std::ostream_iterator<std::string>(os, delimiter.c_str()));
            os << *strings.rbegin();
            return os.str();
        }
    }
```

template<typename Num > std::string bnssemulator::StringHelper::numberFormat (Num *number*)`[static]`, `[noexcept]`

Converts the number to its string representation.

Template Parameters:

Num	Type of the number

Parameters:

number	Number to be converted

Returns:

String representation of the number

Definition at line 97 of file StringHelper.h.

Referenced by bnssemulator::getRegisterIndex().

```
                                                            {
        return std::to_string(number);
    }
```

void bnssemulator::StringHelper::operator= (StringHelper &)`[private]`, `[delete]`

template<typename Num > Num bnssemulator::StringHelper::parseNumber (std::string *number*)`[static]`

Parses the number from its string representation.

Template Parameters:

Num	Type of the number

Parameters:

number	String representation of the number to be parsed

Returns:

Parsed number

Definition at line 111 of file StringHelper.h.

References bnssemulator::BINARY_REGEX, bnssemulator::CHARACTER_REGEX, bnssemulator::DECIMAL_REGEX, bnssemulator::HEX_REGEX, bnssemulator::OCT_REGEX, and bnssemulator::ZERO_REGEX.

```
                                                  {
        long long long_long;

        try {
            if (regex_match(number, ZERO_REGEX)) {
                long_long = 0;
            }
            else if (regex_match(number, DECIMAL_REGEX)) {
                long_long = stoll(number);
            }
            else if (regex_match(number, HEX_REGEX)) {
                long_long = stoll(number.substr(2), nullptr, 16);
            }
```

```
  124             else if (regex_match(number, OCT_REGEX)) {
  125                 long_long = stoll(number, nullptr, 8);
  126             }
  127             else if (regex_match(number, BINARY_REGEX)) {
  128                 long_long = stoll(number.substr(2), nullptr, 2);
  129             }
  130             else if (regex_match(number, CHARACTER_REGEX)) {
  131                 long_long = static_cast<long long>(number[1]);
  132             }
  133             else {
  134                 throw MessageException("The number " + number + " could not be
parsed");
  135             }
  136         }
  137         catch (std::invalid_argument&) {
  138             throw MessageException("The number " + number + " could not be
parsed");
  139         }
  140         catch (std::out_of_range&) {
  141             throw MessageException("The number " + number + " is out of range");
  142         }
  143
  144         auto ret = static_cast<Num>(long_long);
  145
  146         if (ret != long_long) {
  147             throw MessageException("The number + " + number + " is out of range");
  148         }
  149
  150         return ret;
  151     }
```

std::vector< std::string > bnssemulator::StringHelper::split (std::string *string*, std::string *delimiters*) `[static]`, `[noexcept]`

Splits the string using the specified delimiters.

Parameters:

string	String to be split
delimiters	Delimiter characters in the string

Definition at line 23 of file StringHelper.cpp.

```
   23
{
   24         std::vector<std::string> ret;
   25
   26         std::string::size_type last_pos = 0;
   27         auto pos = string.find_first_of(delimiters, last_pos);
   28
   29         while (std::string::npos != pos && std::string::npos != last_pos)
   30         {
   31             ret.push_back(string.substr(last_pos, pos - last_pos));
   32             last_pos = pos + 1;
   33             pos = string.find_first_of(delimiters, last_pos);
   34         }
   35
   36         ret.push_back(string.substr(last_pos, pos - last_pos));
   37
   38         return ret;
   39     }
```

template<typename Num > std::string bnssemulator::StringHelper::toHexString (Num *number*) `[static]`, `[noexcept]`

Converts the number to its hex string representation.

Template Parameters:

Num	Type of the number

Parameters:

number	Number to be converted

Returns:

Hex string representation of the number

Definition at line 102 of file StringHelper.h.

Referenced by bnssemulator::StoreExecuter::execute(), bnssemulator::Processor::executeInstruction(), bnssemulator::fill(), bnssemulator::Segment::getInstruction(), bnssemulator::Context::getOperand(), bnssemulator::Context::getOperandAddress(), bnssemulator::Segment::getSecondWordOfInstruction(), bnssemulator::Segment::readData(), bnssemulator::Segment::relocate(), bnssemulator::AddressSpace::segment(), and bnssemulator::Segment::writeData().

```
                                                            {
        std::stringstream stream;
        stream << "0x"
            << std::setfill('0') << std::setw(sizeof(Num) * 2)
            << std::hex << number;
        return stream.str();
    }
```

std::string bnssemulator::StringHelper::toHexString (unsigned char *number*)`[static]`, `[noexcept]`

Converts the number to its hex string representation.

Template Parameters:

Num	Type of the number

Parameters:

number	Number to be converted

Returns:

Hex string representation of the number

Definition at line 56 of file StringHelper.cpp.

```
                                                                       {
        std::stringstream stream;
        stream << "0x"
            << std::setfill('0') << std::setw(2)
            << std::hex << static_cast<unsigned int>(number);
        return stream.str();
    }
```

std::string bnssemulator::StringHelper::toHexString (signed char *number*)`[static]`, `[noexcept]`

Converts the number to its hex string representation.

Template Parameters:

Num	Type of the number

Parameters:

number	Number to be converted

Returns:

Hex string representation of the number

Definition at line 64 of file StringHelper.cpp.

```
                                                                    {
        std::stringstream stream;
        stream << "0x"
            << std::setfill('0') << std::setw(2)
            << std::hex << static_cast<signed int>(number);
        return stream.str();
    }
```

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**StringHelper.h**
- Code/Emulator/Source/**StringHelper.cpp**

bnssassembler::StringHelper Class Reference

Utility class providing helper methods for std::string class.
`#include <StringHelper.h>`

Static Public Member Functions

- static std::string **fileToString** (std::string file_name)
  Reads the whole file into a string.
- static std::vector< std::string > **split** (std::string string, std::string delimiters) noexcept
  Splits the string using the specified delimiters.
- static std::string **join** (std::vector< std::string > strings, std::string delimiter) noexcept
  Joins the strings from a vector using the specified delimiter.
- template<typename Num > static std::string **numberFormat** (Num number) noexcept
  Converts the number to its string representation.
- template<typename Num > static std::string **toHexString** (Num number) noexcept
  Converts the number to its hex string representation.
- static std::string **toHexString** (unsigned char number) noexcept
  Converts the number to its hex string representation.
- static std::string **toHexString** (signed char number) noexcept
  Converts the number to its hex string representation.
- template<typename Num > static Num **parseNumber** (std::string number)
  Parses the number from its string representation.
- static bool **isAllWhiteSpace** (const std::string &string) noexcept
  Checks if the string contains only whitespace characters.

Private Member Functions

- **StringHelper** ()=delete
- **StringHelper** (**StringHelper** &)=delete
- void **operator=** (**StringHelper** &)=delete

Detailed Description

Utility class providing helper methods for std::string class.

Definition at line 16 of file StringHelper.h.

Constructor & Destructor Documentation

bnssassembler::StringHelper::StringHelper () `[private]`, `[delete]`

bnssassembler::StringHelper::StringHelper (StringHelper &) `[private]`, `[delete]`

Member Function Documentation

std::string bnssassembler::StringHelper::fileToString (std::string *file_name*) `[static]`

Reads the whole file into a string.

Parameters:

file_name	Name of the file

Returns:

String containing the content of the file

Exceptions:

Throws	if the file does not exist or could not be opened for reading

Definition at line 10 of file StringHelper.cpp.

Referenced by bnssassembler::FileReader::readAllLines().

```
                                                     {
        std::ifstream file(file_name);
        if (!file.is_open()) {
            throw std::invalid_argument("File " + file_name + " does not exist");
        }

        std::stringstream ss;
        ss << file.rdbuf();
        auto ret = ss.str();
        file.close();
        return ret;
    }
```

bool bnssassembler::StringHelper::isAllWhiteSpace (const std::string & *string*)`[static]`, `[noexcept]`

Checks if the string contains only whitespace characters.

Parameters:

string	String to be checked

Returns:

Boolean value indicating whether the string contains only whitespace characters

Definition at line 72 of file StringHelper.cpp.

Referenced by bnssassembler::Parser::parse().

```
                                                                    {
        for (auto &ch : string) {
            if (!isspace(ch)) return false;
        }

        return true;
    }
```

std::string bnssassembler::StringHelper::join (std::vector< std::string > *strings*, std::string *delimiter*)`[static]`, `[noexcept]`

Joins the strings from a vector using the specified delimiter.

Parameters:

strings	Vector of strings to be joined
delimiter	Delimiter to be joined

Definition at line 41 of file StringHelper.cpp.

Referenced by bnssassembler::extractLabel(), and bnssassembler::stripComment().

```
{
        switch (strings.size())
```

```
   43         {
   44         case 0:
   45             return "";
   46         case 1:
   47             return strings[0];
   48         default:
   49             std::ostringstream os;
   50             copy(strings.begin(), strings.end() - 1,
std::ostream_iterator<std::string>(os, delimiter.c_str()));
   51             os << *strings.rbegin();
   52             return os.str();
   53         }
   54     }
```

template<typename Num > std::string bnssassembler::StringHelper::numberFormat (Num *number*)`[static]`, `[noexcept]`

Converts the number to its string representation.

Template Parameters:

Num	Type of the number

Parameters:

number	Number to be converted

Returns:

String representation of the number

Definition at line 95 of file StringHelper.h.

Referenced by bnssassembler::AssemblerException::message(), and bnssassembler::operator<<().

```
   95                                                           {
   96         return std::to_string(number);
   97     }
```

void bnssassembler::StringHelper::operator= (StringHelper &)`[private]`, `[delete]`

template<typename Num > Num bnssassembler::StringHelper::parseNumber (std::string *number*)`[static]`

Parses the number from its string representation.

Template Parameters:

Num	Type of the number

Parameters:

number	String representation of the number to be parsed

Returns:

Parsed number

Definition at line 109 of file StringHelper.h.

References bnssassembler::BINARY_REGEX, bnssassembler::CHARACTER_REGEX, bnssassembler::DECIMAL_REGEX, bnssassembler::HEX_REGEX, bnssassembler::OCT_REGEX, and bnssassembler::ZERO_REGEX.

```
  109                                                 {
  110         long long long_long;
  111 
  112         try {
  113             if (regex_match(number, ZERO_REGEX)) {
  114                 long_long = 0;
  115             }
  116             else if (regex_match(number, DECIMAL_REGEX)) {
  117                 long_long = stoll(number);
```

```
            }
            else if (regex_match(number, HEX_REGEX)) {
                long_long = stoll(number.substr(2), nullptr, 16);
            }
            else if (regex_match(number, OCT_REGEX)) {
                long_long = stoll(number, nullptr, 8);
            }
            else if (regex_match(number, BINARY_REGEX)) {
                long_long = stoll(number.substr(2), nullptr, 2);
            }
            else if (regex_match(number, CHARACTER_REGEX)) {
                long_long = static_cast<long long>(number[1]);
            }
            else {
                throw MessageException("The number " + number + " could not be parsed");
            }
        }
        catch (std::invalid_argument&) {
            throw MessageException("The number " + number + " could not be parsed");
        }
        catch (std::out_of_range&) {
            throw MessageException("The number " + number + " is out of range");
        }

        auto ret = static_cast<Num>(long_long);

        if (ret != long_long) {
            throw MessageException("The number + " + number + " is out of range");
        }

        return ret;
    }
```

std::vector< std::string > bnssassembler::StringHelper::split (std::string *string*, std::string *delimiters*) `[static]`, `[noexcept]`

Splits the string using the specified delimiters.

Parameters:

string	String to be split
delimiters	Delimiter characters in the string

Definition at line 23 of file StringHelper.cpp.

Referenced by bnssassembler::GlobalSymbolsLineParser::parse(), and bnssassembler::FileReader::readAllLines().

```
   23
{
   24        std::vector<std::string> ret;
   25
   26        std::string::size_type last_pos = 0;
   27        auto pos = string.find_first_of(delimiters, last_pos);
   28
   29        while (std::string::npos != pos && std::string::npos != last_pos)
   30        {
   31            ret.push_back(string.substr(last_pos, pos - last_pos));
   32            last_pos = pos + 1;
   33            pos = string.find_first_of(delimiters, last_pos);
   34        }
   35
   36        ret.push_back(string.substr(last_pos, pos - last_pos));
   37
   38        return ret;
   39    }
```

template<typename Num > std::string bnssassembler::StringHelper::toHexString (Num *number*) `[static]`, `[noexcept]`

Converts the number to its hex string representation.

Template Parameters:

Num	Type of the number

Parameters:

number	Number to be converted

Returns:

Hex string representation of the number

Definition at line 100 of file StringHelper.h.

Referenced by bnssassembler::operator<<(), and bnssassembler::writeDescription().

```
                                                          {
        std::stringstream stream;
        stream << "0x"
            << std::setfill('0') << std::setw(sizeof(Num) * 2)
            << std::hex << number;
        return stream.str();
    }
```

std::string bnssassembler::StringHelper::toHexString (unsigned char *number*)`[static]`, `[noexcept]`

Converts the number to its hex string representation.

Parameters:

number	Number to be converted

Returns:

Hex string representation of the number

Definition at line 56 of file StringHelper.cpp.

```
                                                                    {
        std::stringstream stream;
        stream << "0x"
            << std::setfill('0') << std::setw(2)
            << std::hex << static_cast<unsigned int>(number);
        return stream.str();
    }
```

std::string bnssassembler::StringHelper::toHexString (signed char *number*)`[static]`, `[noexcept]`

Converts the number to its hex string representation.

Parameters:

number	Number to be converted

Returns:

Hex string representation of the number

Definition at line 64 of file StringHelper.cpp.

```
                                                                  {
        std::stringstream stream;
        stream << "0x"
            << std::setfill('0') << std::setw(2)
            << std::hex << static_cast<signed int>(number);
        return stream.str();
    }
```

The documentation for this class was generated from the following files:

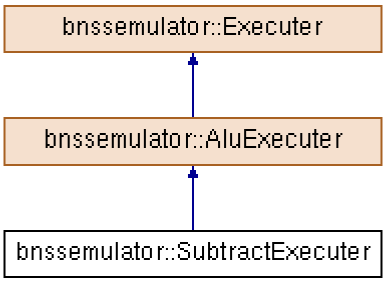
- Code/Assembler/Include/**StringHelper.h**
- Code/Assembler/Source/**StringHelper.cpp**

bnssemulator::SubtractExecuter Class Reference

Class representing the executer for the subtract instruction.

`#include <SubtractExecuter.h>`

Inheritance diagram for bnssemulator::SubtractExecuter:



Protected Member Functions

- void **execute** (**Register** &dst, const **Register** &lhs, const **Register** &rhs) const override
  Executes the ALU instruction.

Additional Inherited Members

Detailed Description

Class representing the executer for the subtract instruction.

Definition at line 10 of file SubtractExecuter.h.

Member Function Documentation

void bnssemulator::SubtractExecuter::execute (Register & *dst*, const Register & *lhs*, const Register & *rhs*) const `[override]`, `[protected]`, `[virtual]`

Executes the ALU instruction.

Parameters:

dst	Reference to the destination register
lhs	Left operand register
rhs	Right operand register

Implements **bnssemulator::AluExecuter** (*p.102*).

Definition at line 5 of file SubtractExecuter.cpp.

```
{
        dst = lhs - rhs;
    }
```

The documentation for this class was generated from the following files:

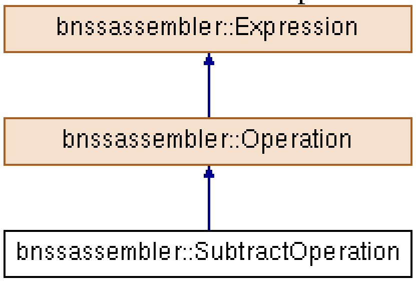
- Code/Emulator/Include/**SubtractExecuter.h**
- Code/Emulator/Source/**SubtractExecuter.cpp**

bnssassembler::SubtractOperation Class Reference

Class implementing the behaviour of the - operator in expressions.

`#include <SubtractOperation.h>`

Inheritance diagram for bnssassembler::SubtractOperation:



Public Member Functions

- bool **containsSymbol** () const noexcept override
  *Tests whether the expression contains a **Symbol**.*
- int **symbolCount** () const noexcept override
  Counts the symbols in the expression.
- std::list< **RelocationRecord** > **generateRelocations** () const override
  Generates the relocation records for the subtree.

Static Public Member Functions

- static std::list< **RelocationRecord** > **generateRelocations** (std::list< **RelocationRecord** > left_list, std::list< **RelocationRecord** > right_list)

Protected Member Functions

- int32_t **calculate** (int32_t lhs, int32_t rhs) const noexcept override
  Calculates the value of the subexpression.

Detailed Description

Class implementing the behaviour of the **-** operator in expressions.

Definition at line 12 of file SubtractOperation.h.

Member Function Documentation

int32_t bnssassembler::SubtractOperation::calculate (int32_t *lhs*, int32_t *rhs*) const`[override]`, `[protected]`, `[virtual]`, `[noexcept]`

Calculates the value of the subexpression.

Parameters:

lhs	Left side of the operator
rhs	Right side of the operator

Returns:

Result of the operation

Exceptions:

Throws	if the expression can not be evaluated (example: division by zero)

Implements **bnssassembler::Operation** (*p.309*).

Definition at line 125 of file SubtractOperation.cpp.

```
  125
{
  126         return lhs - rhs;
  127     }
```

bool bnssassembler::SubtractOperation::containsSymbol () const `[override]`, `[virtual]`, `[noexcept]`

Tests whether the expression contains a **Symbol**.

Returns:

Boolean value indicating whether the expression contains a **Symbol**

Reimplemented from **bnssassembler::Operation** (*p.309*).

Definition at line 6 of file SubtractOperation.cpp.

References bnssassembler::Operation::containsSymbol(), and symbolCount().

```
    6                                                         {
    7         if (Operation::containsSymbol()) {
    8             return symbolCount() == 0;
    9         }
   10
   11         return false;
   12     }
```

std::list< RelocationRecord > bnssassembler::SubtractOperation::generateRelocations () const `[override]`, `[virtual]`

Generates the relocation records for the subtree.

Returns:

Collection of relocation records

Reimplemented from **bnssassembler::Operation** (*p.309*).

Definition at line 68 of file SubtractOperation.cpp.

References bnssassembler::Operation::left(), and bnssassembler::Operation::right().

Referenced by bnssassembler::AddOperation::generateRelocations().

```
   68                                                                            {
   69         auto left_list = left()->generateRelocations();
   70         auto right_list = right()->generateRelocations();
   71         return generateRelocations(left_list, right_list);
   72     }
```

std::list< RelocationRecord > bnssassembler::SubtractOperation::generateRelocations (std::list< RelocationRecord > *left_list*, std::list< RelocationRecord > *right_list*) `[static]`

Definition at line 74 of file SubtractOperation.cpp.

References bnssassembler::exchange(), and bnssassembler::generateMaps().

```
   74
{
   75         std::unordered_map<size_t, std::pair<RelocationRecord, size_t>>
left_sections;
```

```
        std::unordered_map<size_t, std::pair<RelocationRecord, size_t>> right_sections;

        std::unordered_map<std::string, std::pair<RelocationRecord, size_t>> left_symbols;
        std::unordered_map<std::string, std::pair<RelocationRecord, size_t>> right_symbols;

        exchange(left_list, right_list);

        generateMaps(left_list, left_sections, left_symbols);
        generateMaps(right_list, right_sections, right_symbols);

        std::list<RelocationRecord> ret;

        for (auto &element : left_sections) {
            if (right_sections.count(element.first) < left_sections.count(element.first)) {
                for (size_t i = 0; i < element.second.second - right_sections[element.first].second; i++) {
                    ret.push_back(element.second.first);
                }
            }
        }

        for (auto &element : right_sections) {
            if (left_sections.count(element.first) < right_sections.count(element.first)) {
                for (size_t i = 0; i < element.second.second - left_sections[element.first].second; i++) {
                    element.second.first.toggleOpposite();
                    ret.push_back(element.second.first);
                }
            }
        }

        for (auto &element : left_symbols) {
            if (right_symbols.count(element.first) < left_symbols.count(element.first)) {
                for (size_t i = 0; i < element.second.second - right_symbols[element.first].second; i++) {
                    ret.push_back(element.second.first);
                }
            }
        }

        for (auto &element : right_symbols) {
            if (left_symbols.count(element.first) < right_symbols.count(element.first)) {
                for (size_t i = 0; i < element.second.second - right_symbols[element.first].second; i++) {
                    element.second.first.toggleOpposite();
                    ret.push_back(element.second.first);
                }
            }
        }

        return ret;
    }
```

int bnssassembler::SubtractOperation::symbolCount () const`[override]`, `[virtual]`, `[noexcept]`

Counts the symbols in the expression.

Returns:

Number of symbols in the expression

Reimplemented from **bnssassembler::Operation** (*p.312*).

Definition at line 14 of file SubtractOperation.cpp.

References bnssassembler::Operation::left(), and bnssassembler::Operation::right().

Referenced by containsSymbol().

```
                                                {
        return left()->symbolCount() - right()->symbolCount();
    }
```

The documentation for this class was generated from the following files:

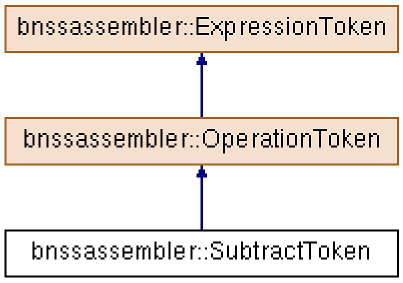
- Code/Assembler/Include/**SubtractOperation.h**
- Code/Assembler/Source/**SubtractOperation.cpp**

bnssassembler::SubtractToken Class Reference

Token class representing the - operation.

`#include <SubtractToken.h>`

Inheritance diagram for bnssassembler::SubtractToken:


bnssassembler::ExpressionToken
bnssassembler::OperationToken
bnssassembler::SubtractToken

Public Member Functions

- int **inputPriority** () const noexcept override
  Gets the input priority of the token.
- int **stackPriority** () const noexcept override
  Gets the stack priority of the token.
- int **rank** () const noexcept override
  Gets the rank of the token.
- std::string **operation** () const noexcept override
- std::shared_ptr< **Expression** > **create** () const override
  Creates an expression object out of the token.

Protected Member Functions

- std::shared_ptr< **ExpressionToken** > **clone** (std::string param) const override
  Clones the current object, using the string provided.

Detailed Description

Token class representing the - operation.

Definition at line 10 of file SubtractToken.h.

Member Function Documentation

std::shared_ptr< ExpressionToken > bnssassembler::SubtractToken::clone (std::string *param*) const `[override]`, `[protected]`, `[virtual]`

Clones the current object, using the string provided.

Parameters:

param	String that will be used to construct the new object

Returns:

Pointer to the cloned object

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 26 of file SubtractToken.cpp.

```
   26
{
   27         return std::make_shared<SubtractToken>();
   28     }
```

std::shared_ptr< Expression > bnssassembler::SubtractToken::create () const`[override]`, `[virtual]`

Creates an expression object out of the token.

Returns:

Pointer to the expression

Exceptions:

Throws	if the token has no corresponding expression object

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 22 of file SubtractToken.cpp.

```
   22                                                         {
   23         return std::make_shared<SubtractOperation>();
   24     }
```

int bnssassembler::SubtractToken::inputPriority () const`[override]`, `[virtual]`, `[noexcept]`

Gets the input priority of the token.

Returns:

Input priority of the token

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 6 of file SubtractToken.cpp.

```
    6                                                  {
    7         return 2;
    8     }
```

std::string bnssassembler::SubtractToken::operation () const`[override]`, `[virtual]`, `[noexcept]`

Implements **bnssassembler::OperationToken** (*p.314*).

Definition at line 18 of file SubtractToken.cpp.

```
   18                                                    {
   19         return "-";
   20     }
```

int bnssassembler::SubtractToken::rank () const`[override]`, `[virtual]`, `[noexcept]`

Gets the rank of the token.

Returns:

Rank of the token

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 14 of file SubtractToken.cpp.

```
   14                                       {
        return -1;
    }
```

int bnssassembler::SubtractToken::stackPriority () const`[override]`, `[virtual]`, `[noexcept]`

Gets the stack priority of the token.

Returns:

Stack priority of the token

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 10 of file SubtractToken.cpp.

```
                                                    {
        return 2;
    }
```

The documentation for this class was generated from the following files:

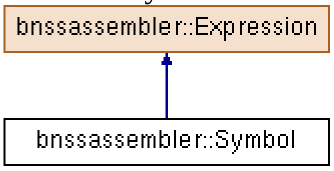
- Code/Assembler/Include/**SubtractToken.h**
- Code/Assembler/Source/**SubtractToken.cpp**

bnssassembler::Symbol Class Reference

Class representing a symbol inside an expression.

`#include <Symbol.h>`

Inheritance diagram for bnssassembler::Symbol:



Public Member Functions

- **Symbol** (std::string **name**) noexcept
- int32_t **value** () const override
  Evaluates the expression.
- bool **setValue** (std::string symbol, std::shared_ptr< **Expression** > **value**) noexcept override
  Traverses the subtree and sets the value for the symbol.
- bool **containsSymbol** () const noexcept override
  *Tests whether the expression contains a **Symbol**.*
- int **symbolCount** () const noexcept override
  Counts the symbols in the expression.
- void **resolveSymbolTable** (const **SymbolTable** &symbol_table) noexcept override
  Resolves the symbols from the symbol table and sets the relocation info.
- void **resolveImports** (std::unordered_set< std::string > imported_symbols) noexcept override
  Resolves the imported symbols and sets the relocation info.
- void **resolveCurrentPcSymbol** (size_t section_index, size_t offset) noexcept override
  Resolves the current PC symbol and sets the relocation info.
- std::list< **RelocationRecord** > **generateRelocations** () const override
  Generates the relocation records for the subtree.

Private Attributes

- std::string **name_**
- std::shared_ptr< **Expression** > **value_** = nullptr
- bool **assigned_** = false
- int32_t **relocatable_value_** = 0
- bool **relocatable_** = false
- size_t **section_index_** = 0
- bool **section_** = false
- bool **absolute_** = false

Detailed Description

Class representing a symbol inside an expression.

Definition at line 10 of file Symbol.h.

Constructor & Destructor Documentation

bnssassembler::Symbol::Symbol (std::string *name*)`[explicit]`, `[noexcept]`

Definition at line 5 of file Symbol.cpp.

```
    5 : name_(name) {}
```

Member Function Documentation

bool bnssassembler::Symbol::containsSymbol () const`[override]`, `[virtual]`, `[noexcept]`

Tests whether the expression contains a **Symbol**.

Returns:

Boolean value indicating whether the expression contains a **Symbol**

Reimplemented from **bnssassembler::Expression** (*p.165*).

Definition at line 29 of file Symbol.cpp.

References assigned_, relocatable_, and value_.

```
   29                                            {
   30         if (relocatable_) {
   31             return true;
   32         }
   33 
   34         if (assigned_) {
   35             return value_->containsSymbol();
   36         }
   37 
   38         return false;
   39     }
```

std::list< RelocationRecord > bnssassembler::Symbol::generateRelocations () const`[override]`, `[virtual]`

Generates the relocation records for the subtree.

Returns:

Collection of relocation records

Reimplemented from **bnssassembler::Expression** (*p.165*).

Definition at line 89 of file Symbol.cpp.

References absolute_, name_, relocatable_, section_, and section_index_.

```
   89                                                                  {
   90         if (relocatable_) {
   91             if (section_) {
   92                 return std::list<RelocationRecord> {
RelocationRecord(absolute_, section_index_) };
   93             }
   94 
   95             return std::list<RelocationRecord> { RelocationRecord(absolute_,
name_) };
   96         }
   97 
   98         return std::list<RelocationRecord>();
   99     }
```

void bnssassembler::Symbol::resolveCurrentPcSymbol (size_t *section_index*, size_t *offset*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the current PC symbol and sets the relocation info.

Parameters:

section_index	Current PC section
offset	PC address in relation to the current section beginning

Reimplemented from **bnssassembler::Expression** (*p.165*).

Definition at line 79 of file Symbol.cpp.

References absolute_, name_, relocatable_, relocatable_value_, section_, and section_index_.

```
{
        if (name_ == "$") {
            relocatable_value_ = static_cast<int32_t>(offset);
            relocatable_ = true;
            section_index_ = section_index;
            absolute_ = true;
            section_ = true;
        }
    }
```

void bnssassembler::Symbol::resolveImports (std::unordered_set< std::string > *imported_symbols*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the imported symbols and sets the relocation info.

Parameters:

imported_symbols	Collection of imported symbols

Reimplemented from **bnssassembler::Expression** (*p.166*).

Definition at line 67 of file Symbol.cpp.

References absolute_, assigned_, name_, relocatable_, relocatable_value_, section_, and value_.

```
{
        if (imported_symbols.count(name_) > 0) {
            relocatable_value_ = static_cast<int32_t>(0);
            relocatable_ = true;
            absolute_ = true;
            section_ = false;
        }
        else if (assigned_) {
            value_->resolveImports(imported_symbols);
        }
    }
```

void bnssassembler::Symbol::resolveSymbolTable (const SymbolTable & *symbol_table*)`[override]`, `[virtual]`, `[noexcept]`

Resolves the symbols from the symbol table and sets the relocation info.

Parameters:

symbol_table	**Symbol** table

Reimplemented from **bnssassembler::Expression** (*p.166*).

Definition at line 53 of file Symbol.cpp.

References absolute_, assigned_, name_, relocatable_, relocatable_value_, section_, section_index_, and value_.

```
{
        if (symbol_table.contains(name_)) {
            auto symbol_entry = symbol_table.at(name_);
            relocatable_value_ = static_cast<int32_t>(symbol_entry.offset());
            relocatable_ = true;
            section_index_ = symbol_entry.sectionIndex();
            absolute_ = true;
            section_ = true;
        }
        else if (assigned_) {
            value_->resolveSymbolTable(symbol_table);
        }
    }
```

bool bnssassembler::Symbol::setValue (std::string *symbol*, std::shared_ptr< Expression > *value*)`[override]`, `[virtual]`, `[noexcept]`

Traverses the subtree and sets the value for the symbol.

Parameters:

symbol	Name of the symbol
value	New value of the symbol

Returns:

Whether the symbol was found and the value was set

Reimplemented from **bnssassembler::Expression** (*p.166*).

Definition at line 19 of file Symbol.cpp.

References assigned_, name_, value(), and value_.

```
{
        if (symbol == name_) {
            value_ = value;
            assigned_ = true;
            return true;
        }

        return false;
    }
```

int bnssassembler::Symbol::symbolCount () const`[override]`, `[virtual]`, `[noexcept]`

Counts the symbols in the expression.

Returns:

Number of symbols in the expression

Reimplemented from **bnssassembler::Expression** (*p.167*).

Definition at line 41 of file Symbol.cpp.

References assigned_, relocatable_, and value_.

```
                                      {
        if (relocatable_) {
            return 1;
        }

        if (assigned_) {
            return value_->symbolCount();
        }

        return 0;
    }
```

int32_t bnssassembler::Symbol::value () const [override], [virtual]

Evaluates the expression.

Exceptions:

Throws	if the expression has variables or could not be evaluated (for example, division by zero)

Implements **bnssassembler::Expression** (*p.167*).

Definition at line 7 of file Symbol.cpp.

References assigned_, name_, relocatable_, relocatable_value_, and value_.

Referenced by setValue().

```
                            {
        if (assigned_) {
            return value_->value();
        }

        if (relocatable_) {
            return relocatable_value_;
        }

        throw NonExistingSymbolException(name_);
    }
```

Member Data Documentation

bool bnssassembler::Symbol::absolute_ = false [private]

Definition at line 32 of file Symbol.h.

Referenced by generateRelocations(), resolveCurrentPcSymbol(), resolveImports(), and resolveSymbolTable().

bool bnssassembler::Symbol::assigned_ = false [private]

Definition at line 26 of file Symbol.h.

Referenced by containsSymbol(), resolveImports(), resolveSymbolTable(), setValue(), symbolCount(), and value().

std::string bnssassembler::Symbol::name_ [private]

Definition at line 23 of file Symbol.h.

Referenced by generateRelocations(), resolveCurrentPcSymbol(), resolveImports(), resolveSymbolTable(), setValue(), and value().

bool bnssassembler::Symbol::relocatable_ = false [private]

Definition at line 29 of file Symbol.h.

Referenced by containsSymbol(), generateRelocations(), resolveCurrentPcSymbol(), resolveImports(), resolveSymbolTable(), symbolCount(), and value().

int32_t bnssassembler::Symbol::relocatable_value_ = 0 `[private]`

Definition at line 28 of file Symbol.h.

Referenced by resolveCurrentPcSymbol(), resolveImports(), resolveSymbolTable(), and value().

bool bnssassembler::Symbol::section_ = false `[private]`

Definition at line 31 of file Symbol.h.

Referenced by generateRelocations(), resolveCurrentPcSymbol(), resolveImports(), and resolveSymbolTable().

size_t bnssassembler::Symbol::section_index_ = 0 `[private]`

Definition at line 30 of file Symbol.h.

Referenced by generateRelocations(), resolveCurrentPcSymbol(), and resolveSymbolTable().

std::shared_ptr<Expression> bnssassembler::Symbol::value_ = nullptr `[private]`

Definition at line 25 of file Symbol.h.

Referenced by containsSymbol(), resolveImports(), resolveSymbolTable(), setValue(), symbolCount(), and value().

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**Symbol.h**
- Code/Assembler/Source/**Symbol.cpp**

bnssemulator::SymbolData Class Reference

Class representing data about one symbol.
`#include <SymbolData.h>`

Public Member Functions

- std::string **name** () const noexcept
  Gets the name of the symbol.
- size_t **sectionIndex** () const noexcept
  Gets the index of the section where the symbol is defined.
- size_t **offset** () const noexcept
  Gets the offset of the symbol from the start of the section.

Private Attributes

- std::string **name_**
- size_t **section_index_** = 0
- size_t **offset_** = 0
- bool **local_** = false

Friends

- std::istream & **operator**>> (std::istream &is, **SymbolData** &data)
  Loads the object from stream.

Detailed Description

Class representing data about one symbol.

Definition at line 11 of file SymbolData.h.

Member Function Documentation

std::string bnssemulator::SymbolData::name () const `[noexcept]`

Gets the name of the symbol.

Returns:

Name of the symbol

Definition at line 5 of file SymbolData.cpp.

References name_.

```
                                     {
        return name_;
    }
```

size_t bnssemulator::SymbolData::offset () const `[noexcept]`

Gets the offset of the symbol from the start of the section.

Returns:

Offset of the symbol from the start of the section

Definition at line 13 of file SymbolData.cpp.

References offset_.

```
                                        {
        return offset_;
    }
```

size_t bnssemulator::SymbolData::sectionIndex () const `[noexcept]`

Gets the index of the section where the symbol is defined.

Returns:

Index of the section where the symbol is defined

Definition at line 9 of file SymbolData.cpp.

References section_index_.

```
                                                  {
        return section_index_;
    }
```

Friends And Related Function Documentation

std::istream& operator>> (std::istream & *is*, SymbolData & *data*) `[friend]`

Loads the object from stream.

Parameters:

is	Input stream
data	Reference to the object that should be loaded

Returns:

Input stream

Definition at line 17 of file SymbolData.cpp.

```
                                                                  {
        is >> data.name_;
        is >> data.section_index_;
        is >> data.offset_;
        is >> data.local_;

        return is;
    }
```

Member Data Documentation

bool bnssemulator::SymbolData::local_ = false `[private]`

Definition at line 42 of file SymbolData.h.

Referenced by bnssemulator::operator>>().

std::string bnssemulator::SymbolData::name_ `[private]`

Definition at line 39 of file SymbolData.h.

Referenced by name(), and bnssemulator::operator>>().

size_t bnssemulator::SymbolData::offset_ = 0 `[private]`

Definition at line 41 of file SymbolData.h.

Referenced by offset(), and bnssemulator::operator>>().

size_t bnssemulator::SymbolData::section_index_ = 0 `[private]`

Definition at line 40 of file SymbolData.h.

Referenced by bnssemulator::operator>>(), and sectionIndex().

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**SymbolData.h**
- Code/Emulator/Source/**SymbolData.cpp**

bnssassembler::SymbolData Class Reference

Class representing data about one symbol.
`#include <SymbolData.h>`

Public Member Functions

- **SymbolData** ()=default
- **SymbolData** (std::string **name**, size_t section_index, size_t **offset**, bool **local**) noexcept
  *Construct a **SymbolData** object.*
- std::string **name** () const noexcept
  Gets the name of the symbol.
- size_t **sectionIndex** () const noexcept
  Get the index of the section where the symbol is located.
- size_t **offset** () const noexcept
  Get the symbol offset from the start of the section.
- bool **local** () const noexcept
  Get whether the symbol is local or global.
- void **exportSymbol** () noexcept
  Exports the symbol.

Private Attributes

- std::string **name_**
- size_t **section_index_**
- size_t **offset_**
- bool **local_**

Friends

- std::ostream & **operator<<** (std::ostream &os, const **SymbolData** &data)
  Writes the content of the object to a stream.

Detailed Description

Class representing data about one symbol.

Definition at line 10 of file SymbolData.h.

Constructor & Destructor Documentation

bnssassembler::SymbolData::SymbolData () `[default]`

bnssassembler::SymbolData::SymbolData (std::string *name*, size_t *section_index*, size_t *offset*, bool *local*) `[noexcept]`

Construct a **SymbolData** object.

Definition at line 8 of file SymbolData.cpp.

```
    8 : name_(name), section_index_(section_index), offset_(offset), local_(local) {}
```

Member Function Documentation

void bnssassembler::SymbolData::exportSymbol () `[noexcept]`

Exports the symbol.

Definition at line 26 of file SymbolData.cpp.

References local_.

```
   26                                        {
   27         local_ = false;
   28     }
```

bool bnssassembler::SymbolData::local () const `[noexcept]`

Get whether the symbol is local or global.

Returns:

Boolean value indicating whether the symbol is local

Definition at line 22 of file SymbolData.cpp.

References local_.

```
   22                                       {
   23         return local_;
   24     }
```

std::string bnssassembler::SymbolData::name () const `[noexcept]`

Gets the name of the symbol.

Returns:

Name of the symbol

Definition at line 10 of file SymbolData.cpp.

References name_.

Referenced by bnssassembler::SymbolTable::operator+=().

```
   10                                            {
   11         return name_;
   12     }
```

size_t bnssassembler::SymbolData::offset () const `[noexcept]`

Get the symbol offset from the start of the section.

Returns:

Offset from the start of the section

Definition at line 18 of file SymbolData.cpp.

References offset_.

```
   18                                          {
   19         return offset_;
   20     }
```

size_t bnssassembler::SymbolData::sectionIndex () const `[noexcept]`

Get the index of the section where the symbol is located.

Returns:

Index of section

Definition at line 14 of file SymbolData.cpp.

References section_index_.

```
                                                    {
          return section_index_;
      }
```

Friends And Related Function Documentation

std::ostream& operator<< (std::ostream & *os*, const SymbolData & *data*)`[friend]`

Writes the content of the object to a stream.

Parameters:

os	Stream where the content will be written
data	**Data** that will be written

Definition at line 30 of file SymbolData.cpp.

```
                                                                        {
          os << data.name_ << std::endl;
          os << data.section_index_ << std::endl;
          os << data.offset_ << std::endl;
          os << data.local_ << std::endl;

          std::cout << VERTICAL << " " << std::setw(46) << std::left << data.name_
<< VERTICAL << " " << std::setw(8) << std::left << data.section_index_ << VERTICAL <<
" " << std::setw(7) << std::left << data.offset_ << VERTICAL << std::setw(14) <<
std::left << (data.local_ ? " Local" : " Global") << VERTICAL << std::endl;

          return os;
      }
```

Member Data Documentation

bool bnssassembler::SymbolData::local_`[private]`

Definition at line 59 of file SymbolData.h.

Referenced by exportSymbol(), local(), and bnssassembler::operator<<().

std::string bnssassembler::SymbolData::name_`[private]`

Definition at line 56 of file SymbolData.h.

Referenced by name(), and bnssassembler::operator<<().

size_t bnssassembler::SymbolData::offset_`[private]`

Definition at line 58 of file SymbolData.h.

Referenced by offset(), and bnssassembler::operator<<().

size_t bnssassembler::SymbolData::section_index_`[private]`

Definition at line 57 of file SymbolData.h.

Referenced by bnssassembler::operator<<(), and sectionIndex().

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**SymbolData.h**
- Code/Assembler/Source/**SymbolData.cpp**

bnssassembler::SymbolDefinition Class Reference

Class representing a symbol definition.
`#include <SymbolDefinition.h>`

Public Member Functions

- **SymbolDefinition** (std::string **name**, **MicroRiscExpression expression**) noexcept
  Constructs a symbol definition.
- std::string **name** () const noexcept
  Get the name of the symbol.
- **MicroRiscExpression expression** () const noexcept
  Get the expression.

Private Attributes

- std::string **name_**
- **MicroRiscExpression expression_**

Friends

- bool **operator==** (const **SymbolDefinition** &lhs, const **SymbolDefinition** &rhs)
- bool **operator!=** (const **SymbolDefinition** &lhs, const **SymbolDefinition** &rhs)
- bool **operator<** (const **SymbolDefinition** &lhs, const **SymbolDefinition** &rhs)
- bool **operator>** (const **SymbolDefinition** &lhs, const **SymbolDefinition** &rhs)
- bool **operator<=** (const **SymbolDefinition** &lhs, const **SymbolDefinition** &rhs)
- bool **operator>=** (const **SymbolDefinition** &lhs, const **SymbolDefinition** &rhs)

Detailed Description

Class representing a symbol definition.

Definition at line 10 of file SymbolDefinition.h.

Constructor & Destructor Documentation

bnssassembler::SymbolDefinition::SymbolDefinition (std::string *name*, MicroRiscExpression *expression*)`[noexcept]`

Constructs a symbol definition.

Parameters:

name	Name of the symbol
expression	**Expression** representing the symbol

Definition at line 5 of file SymbolDefinition.cpp.

```
    5 : name_(name), expression_(expression) {}
```

Member Function Documentation

MicroRiscExpression bnssassembler::SymbolDefinition::expression () const`[noexcept]`

Get the expression.

Returns:

Expression

Definition at line 11 of file SymbolDefinition.cpp.

References expression_.

```
                                                    {
        return expression_;
    }
```

std::string bnssassembler::SymbolDefinition::name () const`[noexcept]`

Get the name of the symbol.

Returns:

Name of the symbol

Definition at line 7 of file SymbolDefinition.cpp.

References name_.

Referenced by bnssassembler::FirstPassData::insertSymbolDefinition(), and std::hash< bnssassembler::SymbolDefinition >::operator()().

```
                                        {
        return name_;
    }
```

Friends And Related Function Documentation

bool operator!= (const SymbolDefinition & *lhs*, const SymbolDefinition & *rhs*)`[friend]`

Definition at line 19 of file SymbolDefinition.cpp.

```
{
        return !(lhs == rhs);
    }
```

bool operator< (const SymbolDefinition & *lhs*, const SymbolDefinition & *rhs*)`[friend]`

Definition at line 23 of file SymbolDefinition.cpp.

```
{
        return lhs.name_ < rhs.name_;
    }
```

bool operator<= (const SymbolDefinition & *lhs*, const SymbolDefinition & *rhs*)`[friend]`

Definition at line 31 of file SymbolDefinition.cpp.

```
   31
{
   32         return !(lhs > rhs);
   33     }
```

bool operator== (const SymbolDefinition & *lhs*, const SymbolDefinition & *rhs*) `[friend]`

Definition at line 15 of file SymbolDefinition.cpp.

```
   15
{
   16         return lhs.name_ == rhs.name_;
   17     }
```

bool operator> (const SymbolDefinition & *lhs*, const SymbolDefinition & *rhs*) `[friend]`

Definition at line 27 of file SymbolDefinition.cpp.

```
   27
{
   28         return !(lhs < rhs || lhs == rhs);
   29     }
```

bool operator>= (const SymbolDefinition & *lhs*, const SymbolDefinition & *rhs*) `[friend]`

Definition at line 35 of file SymbolDefinition.cpp.

```
   35
{
   36         return !(lhs < rhs);
   37     }
```

Member Data Documentation

MicroRiscExpression bnssassembler::SymbolDefinition::expression_ `[private]`

Definition at line 42 of file SymbolDefinition.h.

Referenced by expression().

std::string bnssassembler::SymbolDefinition::name_ `[private]`

Definition at line 41 of file SymbolDefinition.h.

Referenced by name(), bnssassembler::operator<(), and bnssassembler::operator==().

The documentation for this class was generated from the following files:

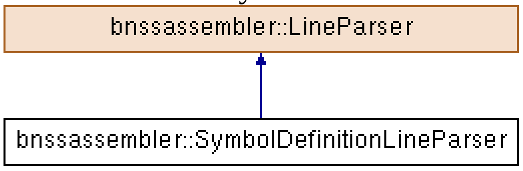
- Code/Assembler/Include/**SymbolDefinition.h**
- Code/Assembler/Source/**SymbolDefinition.cpp**

bnssassembler::SymbolDefinitionLineParser Class Reference

Class used for parsing symbol definitions.

`#include <SymbolDefinitionLineParser.h>`

Inheritance diagram for bnssassembler::SymbolDefinitionLineParser:


bnssassembler::LineParser
bnssassembler::SymbolDefinitionLineParser

Protected Member Functions

- std::shared_ptr< **Token** > **parse** (const std::string &line, size_t line_number, std::string initial_line) const override
  Parses one line of the file. Does not call the next parser in chain.

Additional Inherited Members

Detailed Description

Class used for parsing symbol definitions.

Definition at line 10 of file SymbolDefinitionLineParser.h.

Member Function Documentation

std::shared_ptr< Token > bnssassembler::SymbolDefinitionLineParser::parse (const std::string & *line*, size_t *line_number*, std::string *initial_line*) const `[override]`, `[protected]`, `[virtual]`

Parses one line of the file. Does not call the next parser in chain.

Parameters:

line	Line to parse
line_number	Number of the line that is parsed
initial_line	Initial line that is parsed

Returns:

Extracted token from line or nullptr if the parser failed parsing the line

Exceptions:

Throws	if the parser failed and identified the error

Implements **bnssassembler::LineParser** (*p.257*).

Definition at line 9 of file SymbolDefinitionLineParser.cpp.

References bnssassembler::ExpressionBuilder::build(), bnssassembler::CONSTANT_TERM, bnssassembler::name(), bnssassembler::SYMBOL, and bnssassembler::SYMBOL_DEFINITION.

```
{
        static std::regex regex("[[:space:]]*(" + SYMBOL + ")[[:space:]]*" +
SYMBOL_DEFINITION + "(" + CONSTANT_TERM + ")");

        if (!regex_match(line, regex)) {
            return nullptr;
        }

        auto name = regex_replace(line, regex, "$1");
        auto expression_string = regex_replace(line, regex, "$4");
        auto expression = ExpressionBuilder::build(expression_string);

        return std::make_shared<SymbolDefinitionToken>(name, expression,
line_number, initial_line);
    }
```

The documentation for this class was generated from the following files:

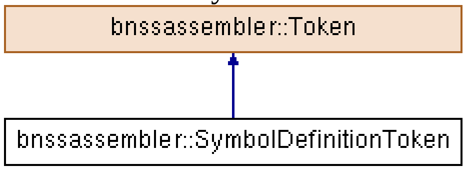
- Code/Assembler/Include/**SymbolDefinitionLineParser.h**
- Code/Assembler/Source/**SymbolDefinitionLineParser.cpp**

bnssassembler::SymbolDefinitionToken Class Reference

Class representing the symbol definition token.

`#include <SymbolDefinitionToken.h>`

Inheritance diagram for bnssassembler::SymbolDefinitionToken:



Public Member Functions

- **SymbolDefinitionToken** (std::string **name**, **MicroRiscExpression** value, size_t line_number, std::string **line**) noexcept
  *Constructs a **SymbolDefinitionToken** object.*
- void **resolveSymbolDefinitions** (std::unordered_set< **SymbolDefinition** > symbols) noexcept override
  Resolves symbol definitions in a token.
- void **firstPass** (**FirstPassData** &data) const override
  Executes the first pass over the token.
- void **secondPass** (**SecondPassData** &data) const override
  Executes the second pass over the token.

Private Attributes

- std::string **name_**
- **MicroRiscExpression value_**

Detailed Description

Class representing the symbol definition token.

Definition at line 11 of file SymbolDefinitionToken.h.

Constructor & Destructor Documentation

bnssassembler::SymbolDefinitionToken::SymbolDefinitionToken (std::string *name*, MicroRiscExpression *value*, size_t *line_number*, std::string *line*) `[noexcept]`

Constructs a **SymbolDefinitionToken** object.

Parameters:

name	Name of the symbol
value	Value of the symbol
line_number	Number of the line where the definition is located
line	Line where the definition is located

Definition at line 5 of file SymbolDefinitionToken.cpp.

```
    5 : Token(line_number, line), name_(name), value_(value) {}
```

Member Function Documentation

void bnssassembler::SymbolDefinitionToken::firstPass (FirstPassData & *data*) const[override], [virtual]

Executes the first pass over the token.

Parameters:

data	**Data** that the token will modify

Implements **bnssassembler::Token** (*p.510*).

Definition at line 13 of file SymbolDefinitionToken.cpp.

References bnssassembler::FirstPassData::insertSymbolDefinition(), name_, and value_.

```
                                                             {
        data.insertSymbolDefinition(SymbolDefinition(name_, value_));
    }
```

void bnssassembler::SymbolDefinitionToken::resolveSymbolDefinitions (std::unordered_set< SymbolDefinition > *symbols*)[override], [virtual], [noexcept]

Resolves symbol definitions in a token.

Parameters:

symbols	Vector od symbol definitions that should be resolved

Reimplemented from **bnssassembler::Token** (*p.511*).

Definition at line 7 of file SymbolDefinitionToken.cpp.

```
{
        for (auto &symbol : symbols) {
            value_.setValue(symbol.name(), symbol.expression());
        }
    }
```

void bnssassembler::SymbolDefinitionToken::secondPass (SecondPassData & *data*) const[override], [virtual]

Executes the second pass over the token.

Parameters:

data	**Data** that the token will modify

Implements **bnssassembler::Token** (*p.512*).

Definition at line 17 of file SymbolDefinitionToken.cpp.

```
                                                                 {
        // TODO: Implementation
    }
```

Member Data Documentation

std::string bnssassembler::SymbolDefinitionToken::name_[private]

Definition at line 25 of file SymbolDefinitionToken.h.

Referenced by firstPass().

MicroRiscExpression bnssassembler::SymbolDefinitionToken::value_ `[private]`

Definition at line 26 of file SymbolDefinitionToken.h.

Referenced by firstPass().

The documentation for this class was generated from the following files:

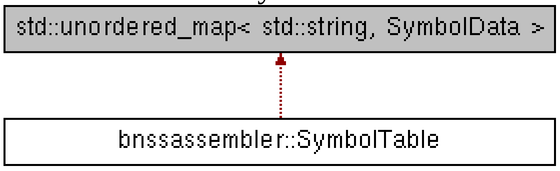
- Code/Assembler/Include/**SymbolDefinitionToken.h**
- Code/Assembler/Source/**SymbolDefinitionToken.cpp**

bnssassembler::SymbolTable Class Reference

Class representing the symbol table.

`#include <SymbolTable.h>`

Inheritance diagram for bnssassembler::SymbolTable:


std::unordered_map< std::string, SymbolData >
bnssassembler::SymbolTable

Public Member Functions

- **SymbolTable** & **operator+=** (const **SymbolData** &data)
  Inserts a symbol into the table.
- bool **contains** (std::string symbol) const noexcept
  Check if the table contains a symbol.
- void **exportSymbol** (std::string symbol) noexcept
  Export a symbol.

Friends

- std::ostream & **operator<<** (std::ostream &os, const **SymbolTable** &table)
  Writes the content of the object to a stream.

Detailed Description

Class representing the symbol table.

Definition at line 11 of file SymbolTable.h.

Member Function Documentation

bool bnssassembler::SymbolTable::contains (std::string *symbol*) const `[noexcept]`

Check if the table contains a symbol.

Parameters:

symbol	**Symbol** to be checked

Returns:

Whether the symbol exists in the table

Definition at line 14 of file SymbolTable.cpp.

Referenced by bnssassembler::SecondPassData::contains(), and bnssassembler::FirstPassData::insertSymbol().

```
                                                            {
          return count(symbol) > 0;
    }
```

void bnssassembler::SymbolTable::exportSymbol (std::string *symbol*) `[noexcept]`

Export a symbol.

Parameters:

symbol	**Symbol** to be exported

Definition at line 18 of file SymbolTable.cpp.

Referenced by bnssassembler::SecondPassData::exportSymbol().

```
   18                                                   {
   19         (*this)[symbol].exportSymbol();
   20     }
```

SymbolTable & bnssassembler::SymbolTable::operator+= (const SymbolData & *data*)

Inserts a symbol into the table.

Parameters:

data	**Symbol** to be inserted

Returns:

Reference to this **SymbolTable** object after the insertion

Definition at line 9 of file SymbolTable.cpp.

References bnssassembler::SymbolData::name().

```
    9                                                       {
   10         insert(make_pair(data.name(), data));
   11         return *this;
   12     }
```

Friends And Related Function Documentation

std::ostream& operator<< (std::ostream & *os*, const SymbolTable & *table*)`[friend]`

Writes the content of the object to a stream.

Parameters:

os	Stream where the content will be written
table	**Data** that will be written

Definition at line 22 of file SymbolTable.cpp.

```
   22                                                                       {
   23         std::cout << UPPER_LEFT << multiple(HORIZONTAL, 81) << UPPER_RIGHT <<
std::endl;
   24         std::cout << VERTICAL << UPPER_LEFT << multiple(HORIZONTAL, 79) <<
UPPER_RIGHT << VERTICAL << std::endl;
   25         std::cout << VERTICAL << VERTICAL << std::setw(79) << std::left << "
Symbol table:" << VERTICAL << VERTICAL << std::endl;
   26         std::cout << VERTICAL << LOWER_LEFT << multiple(HORIZONTAL, 79) <<
LOWER_RIGHT << VERTICAL << std::endl;
   27         std::cout << T_RIGHT << multiple(HORIZONTAL, 47) << T_DOWN <<
multiple(HORIZONTAL, 9) << T_DOWN << multiple(HORIZONTAL, 8) << T_DOWN <<
multiple(HORIZONTAL, 14) << T_LEFT << std::endl;
   28         std::cout << VERTICAL << "                     Name
" << VERTICAL << " Section " << VERTICAL << " Offset " << VERTICAL << " Global/Local
" << VERTICAL << std::endl;
   29         std::cout << T_RIGHT << multiple(HORIZONTAL, 47) << ALL_FOUR <<
multiple(HORIZONTAL, 9) << ALL_FOUR << multiple(HORIZONTAL, 8) << ALL_FOUR <<
multiple(HORIZONTAL, 14) << T_LEFT << std::endl;
   30
   31         os << table.size() << std::endl;
   32         for (auto &entry : table) {
```

```
            os << entry.second << std::endl;
        }

        std::cout << LOWER_LEFT << multiple(HORIZONTAL, 47) << T_UP << 
multiple(HORIZONTAL, 9) << T_UP << multiple(HORIZONTAL, 8) << T_UP << 
multiple(HORIZONTAL, 14) << LOWER_RIGHT << std::endl;

        return os;
    }
```

The documentation for this class was generated from the following files:

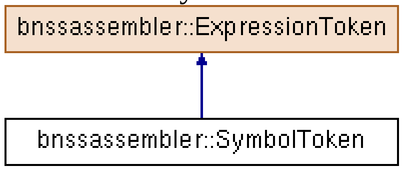
- Code/Assembler/Include/**SymbolTable.h**
- Code/Assembler/Source/**SymbolTable.cpp**

bnssassembler::SymbolToken Class Reference

Token class representing a math symbol.

`#include <SymbolToken.h>`

Inheritance diagram for bnssassembler::SymbolToken:



Public Member Functions

- **SymbolToken** (std::string **name**) noexcept
- int **inputPriority** () const noexcept override
  Gets the input priority of the token.
- int **stackPriority** () const noexcept override
  Gets the stack priority of the token.
- int **rank** () const noexcept override
  Gets the rank of the token.
- void **process** (std::list< std::shared_ptr< **ExpressionToken** >> &output, std::stack< std::shared_ptr< **ExpressionToken** >> &stack, int &expression_rank) const override
  Processes the current token.
- std::shared_ptr< **Expression** > **create** () const override
  Creates an expression object out of the token.

Protected Member Functions

- std::shared_ptr< **ExpressionToken** > **clone** (std::string param) const override
  Clones the current object, using the string provided.

Private Attributes

- std::string **name_**

Detailed Description

Token class representing a math symbol.

Definition at line 10 of file SymbolToken.h.

Constructor & Destructor Documentation

bnssassembler::SymbolToken::SymbolToken (std::string *name*) `[explicit]`, `[noexcept]`

Definition at line 7 of file SymbolToken.cpp.

```
: name_(name) {}
```

Member Function Documentation

std::shared_ptr< ExpressionToken > bnssassembler::SymbolToken::clone (std::string *param*) const `[override]`, `[protected]`, `[virtual]`

Clones the current object, using the string provided.

Parameters:

param	String that will be used to construct the new object

Returns:

Pointer to the cloned object

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 18 of file SymbolToken.cpp.

```
                                                                        {
        return std::make_shared<SymbolToken>(param);
    }
```

std::shared_ptr< Expression > bnssassembler::SymbolToken::create () const `[override]`, `[virtual]`

Creates an expression object out of the token.

Returns:

Pointer to the expression

Exceptions:

Throws	if the token has no corresponding expression object

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 14 of file SymbolToken.cpp.

References name_.

```
                                                    {
        return std::make_shared<Symbol>(name_);
    }
```

int bnssassembler::SymbolToken::inputPriority () const `[override]`, `[virtual]`, `[noexcept]`

Gets the input priority of the token.

Returns:

Input priority of the token

Implements **bnssassembler::ExpressionToken** (*p.171*).

Definition at line 22 of file SymbolToken.cpp.

```
                                                {
        return INT_MAX;
    }
```

void bnssassembler::SymbolToken::process (std::list< std::shared_ptr< ExpressionToken >> & *output*, std::stack< std::shared_ptr< ExpressionToken >> & *stack*, int & *expression_rank*) const `[override]`, `[virtual]`

Processes the current token.

Parameters:

output	Output list of tokens
stack	Helper stack of tokens
expression_rank	Rank of the expression

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 9 of file SymbolToken.cpp.

References rank().

```
{
        output.push_back(std::make_shared<SymbolToken>(*this));
        expression_rank += rank();
    }
```

int bnssassembler::SymbolToken::rank () const [override], [virtual], [noexcept]

Gets the rank of the token.

Returns:

Rank of the token

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 30 of file SymbolToken.cpp.

Referenced by process().

```
                                    {
        return 1;
    }
```

int bnssassembler::SymbolToken::stackPriority () const [override], [virtual], [noexcept]

Gets the stack priority of the token.

Returns:

Stack priority of the token

Implements **bnssassembler::ExpressionToken** (*p.172*).

Definition at line 26 of file SymbolToken.cpp.

```
                                              {
        return INT_MAX;
    }
```

Member Data Documentation

std::string bnssassembler::SymbolToken::name_ [private]

Definition at line 23 of file SymbolToken.h.

Referenced by create().

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**SymbolToken.h**
- Code/Assembler/Source/**SymbolToken.cpp**

bnssemulator::TimerListener Class Reference

Class representing a listener for the timer events.
`#include <TimerListener.h>`

Static Public Member Functions

- static void **listen** (**Context** *context)
  Listens to timer interrupts and sets the context flag every time it should.

Detailed Description

Class representing a listener for the timer events.

Definition at line 10 of file TimerListener.h.

Member Function Documentation

void bnssemulator::TimerListener::listen (Context * *context*) `[static]`

Listens to timer interrupts and sets the context flag every time it should.

Definition at line 7 of file TimerListener.cpp.

References bnssemulator::Context::programFinished(), and bnssemulator::Context::timerTriggered().

Referenced by bnssemulator::Processor::executeProgram().

```
                                        {
        using namespace std::literals::chrono_literals;

        while (!context->programFinished()) {
            std::this_thread::sleep_for(100ms);
            context->timerTriggered(true);
        }
    }
```

The documentation for this class was generated from the following files:

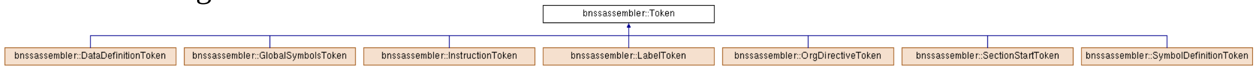
- Code/Emulator/Include/**TimerListener.h**
- Code/Emulator/Source/**TimerListener.cpp**

bnssassembler::Token Class Reference

Class representing one token of the assembler source file.

`#include <Token.h>`

Inheritance diagram for bnssassembler::Token:


bnssassembler::Token
bnssassembler::DataDefinitionToken
bnssassembler::GlobalSymbolsToken
bnssassembler::InstructionToken
bnssassembler::LabelToken
bnssassembler::OrgDirectiveToken
bnssassembler::SectionStartToken
bnssassembler::SymbolDefinitionToken

Public Member Functions

- **Token** (size_t line_number, std::string **line**) noexcept
  Constructs a token.
- virtual void **resolveSymbolDefinitions** (std::unordered_set< **SymbolDefinition** > symbols) noexcept
  Resolves symbol definitions in a token.
- virtual void **firstPass** (**FirstPassData** &data) const =0
  Executes the first pass over the token.
- virtual void **secondPass** (**SecondPassData** &data) const =0
  Executes the second pass over the token.
- virtual bool **usesAddress** () const noexcept
  Check whether the token can use the ORG address.
- virtual void **resolveSymbolTable** (const **SymbolTable** &symbol_table) noexcept
  Resolves the symbols from the symbol table and updates relocation info.
- virtual void **resolveImports** (std::unordered_set< std::string > imported_symbols) noexcept
  Resolves the imported symbols and updates relocation info.
- virtual void **resolveCurrentPcSymbol** (size_t section_index, size_t offset) noexcept
  Resolves the current PC symbol and sets the relocation info.
- size_t **lineNumber** () const noexcept
  Get the line number of the token.
- std::string **line** () const noexcept
  Get the line of the token.
- virtual **~Token** ()=default

Private Attributes

- size_t **line_number_**
- std::string **line_**

Detailed Description

Class representing one token of the assembler source file.

Definition at line 13 of file Token.h.

Constructor & Destructor Documentation

bnssassembler::Token::Token (size_t *line_number*, std::string *line*) `[noexcept]`

Constructs a token.

Parameters:

line_number	Number of the line in the assembler source file
line	Line in the assembler source file

Definition at line 5 of file Token.cpp.

```
    5 : line_number_(line_number), line_(line) {}
```

virtual bnssassembler::Token::~Token () `[virtual]`, `[default]`

Member Function Documentation

virtual void bnssassembler::Token::firstPass (FirstPassData & *data*) const `[pure virtual]`

Executes the first pass over the token.

Parameters:

data	**Data** that the token will modify

Implemented in **bnssassembler::SectionStartToken** (*p.433*), **bnssassembler::InstructionToken** (*p.229*), **bnssassembler::DataDefinitionToken** (*p.148*), **bnssassembler::OrgDirectiveToken** (*p.351*), **bnssassembler::SymbolDefinitionToken** (*p.499*), **bnssassembler::GlobalSymbolsToken** (*p.194*), and **bnssassembler::LabelToken** (*p.254*).

std::string bnssassembler::Token::line () const `[noexcept]`

Get the line of the token.

Returns:

line

Definition at line 32 of file Token.cpp.

References line_.

```
   32                                      {
   33          return line_;
   34      }
```

size_t bnssassembler::Token::lineNumber () const `[noexcept]`

Get the line number of the token.

Returns:

Line number

Definition at line 28 of file Token.cpp.

References line_number_.

```
   28                                          {
   29          return line_number_;
   30      }
```

void bnssassembler::Token::resolveCurrentPcSymbol (size_t *section_index*, size_t *offset*) `[virtual]`, `[noexcept]`

Resolves the current PC symbol and sets the relocation info.

Parameters:

section_index	Current PC section
offset	PC address in relation to the current section beginning

Reimplemented in **bnssassembler::InstructionToken** (*p.230*).

Definition at line 24 of file Token.cpp.

```
{
        // Default: Do nothing
    }
```

void bnssassembler::Token::resolveImports (std::unordered_set< std::string > *imported_symbols*)[virtual], [noexcept]

Resolves the imported symbols and updates relocation info.

Parameters:

imported_symbols	Collection of imported symbols

Reimplemented in **bnssassembler::InstructionToken** (*p.230*), **bnssassembler::DataDefinitionToken** (*p.148*), and **bnssassembler::OrgDirectiveToken** (*p.351*).

Definition at line 20 of file Token.cpp.

```
{
        // Default: Do nothing
    }
```

void bnssassembler::Token::resolveSymbolDefinitions (std::unordered_set< SymbolDefinition > *symbols*)[virtual], [noexcept]

Resolves symbol definitions in a token.

Parameters:

symbols	Vector od symbol definitions that should be resolved

Reimplemented in **bnssassembler::InstructionToken** (*p.231*), **bnssassembler::DataDefinitionToken** (*p.149*), **bnssassembler::OrgDirectiveToken** (*p.351*), and **bnssassembler::SymbolDefinitionToken** (*p.499*).

Definition at line 7 of file Token.cpp.

```
{
        // Default: Do nothing
    }
```

void bnssassembler::Token::resolveSymbolTable (const SymbolTable & *symbol_table*)[virtual], [noexcept]

Resolves the symbols from the symbol table and updates relocation info.

Parameters:

symbol_table	**Symbol** table

Reimplemented in **bnssassembler::InstructionToken** (*p.231*), **bnssassembler::DataDefinitionToken** (*p.149*), and **bnssassembler::OrgDirectiveToken** (*p.351*).

Definition at line 16 of file Token.cpp.

```
                                                            {
        // Default: Do nothing
    }
```

virtual void bnssassembler::Token::secondPass (SecondPassData & *data*) const[pure virtual]

Executes the second pass over the token.

Parameters:

data	**Data** that the token will modify

Implemented in **bnssassembler::SectionStartToken** (*p.433*), **bnssassembler::InstructionToken** (*p.231*), **bnssassembler::DataDefinitionToken** (*p.149*), **bnssassembler::OrgDirectiveToken** (*p.352*), **bnssassembler::SymbolDefinitionToken** (*p.499*), **bnssassembler::GlobalSymbolsToken** (*p.195*), and **bnssassembler::LabelToken** (*p.255*).

bool bnssassembler::Token::usesAddress () const[virtual], [noexcept]

Check whether the token can use the ORG address.

Reimplemented in **bnssassembler::SectionStartToken** (*p.433*).

Definition at line 11 of file Token.cpp.

```
                                    {
        // Default: Do not use address
        return false;
    }
```

Member Data Documentation

std::string bnssassembler::Token::line_[private]

Definition at line 79 of file Token.h.

Referenced by line().

size_t bnssassembler::Token::line_number_[private]

Definition at line 78 of file Token.h.

Referenced by lineNumber().

The documentation for this class was generated from the following files:

- Code/Assembler/Include/**Token.h**
- Code/Assembler/Source/**Token.cpp**

cxxopts::values::type_is_container< T > Struct Template Reference

`#include <cxxopts.h>`

Static Public Attributes

- static constexpr bool **value** = false

Detailed Description

template<typename T>

struct cxxopts::values::type_is_container< T >

Definition at line 462 of file cxxopts.h.

Member Data Documentation

template<typename T > static constexpr bool cxxopts::values::type_is_container< T >::value = false `[static]`

Definition at line 464 of file cxxopts.h.

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

- Code/Assembler/Include/**cxxopts.h**

cxxopts::values::type_is_container< std::vector< T > > Struct Template Reference

`#include <cxxopts.h>`

Static Public Attributes

- static constexpr bool **value** = true

Detailed Description

template<typename T>

struct cxxopts::values::type_is_container< std::vector< T > >

Definition at line 468 of file cxxopts.h.

Member Data Documentation

template<typename T > static constexpr bool cxxopts::values::type_is_container< std::vector< T > >::value = true `[static]`

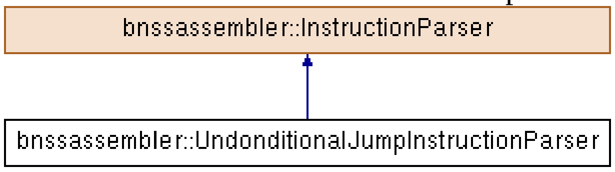
Definition at line 470 of file cxxopts.h.

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

- Code/Assembler/Include/**cxxopts.h**

bnssassembler::UndonditionalJumpInstructionParser Class Reference

Class representing the parser for the unconditional jump instructions.
`#include <UnconditionalJumpInstructionParser.h>`
Inheritance diagram for bnssassembler::UndonditionalJumpInstructionParser:


bnssassembler::InstructionParser
bnssassembler::UndonditionalJumpInstructionParser

Public Member Functions

- **UndonditionalJumpInstructionParser** () noexcept
  Constructs an UnconditionalJumpInstructionParser object.

Additional Inherited Members

Detailed Description

Class representing the parser for the unconditional jump instructions.

Definition at line 10 of file UnconditionalJumpInstructionParser.h.

Constructor & Destructor Documentation

bnssassembler::UndonditionalJumpInstructionParser::UndonditionalJumpInstructionParser () `[noexcept]`

Constructs an UnconditionalJumpInstructionParser object.

Definition at line 8 of file UnconditionalJumpInstructionParser.cpp.

References bnssassembler::InstructionParser::operands_.

```
{
        auto memdir = std::make_shared<MemoryDirectParser>();
        auto regindpom = std::make_shared<RegisterIndirectOffsetParser>();
        auto regind = std::make_shared<RegisterIndirectParser>();

        memdir->next(regindpom);
        regindpom->next(regind);

        operands_.push_back(memdir);
    }
```

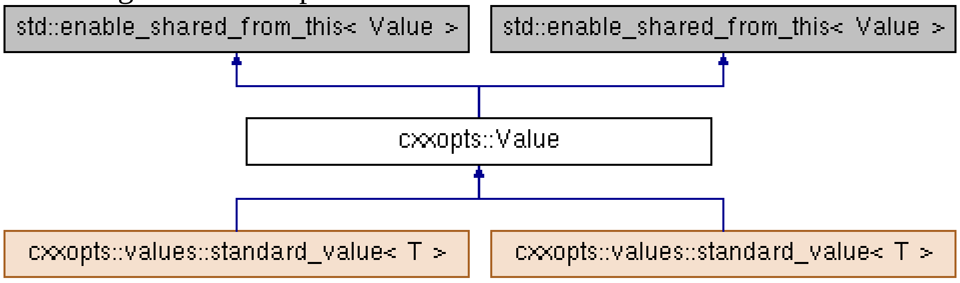
The documentation for this class was generated from the following files:

- Code/Assembler/Include/**UnconditionalJumpInstructionParser.h**
- Code/Assembler/Source/**UnconditionalJumpInstructionParser.cpp**

cxxopts::Value Class Reference

`#include <cxxopts.h>`

Inheritance diagram for cxxopts::Value:


std::enable_shared_from_this< Value >
std::enable_shared_from_this< Value >
cxxopts::Value
cxxopts::values::standard_value< T >
cxxopts::values::standard_value< T >

Public Member Functions

- virtual ~**Value** ()=default
- virtual void **parse** (const std::string &text) const =0
- virtual void **parse** () const =0
- virtual bool **has_arg** () const =0
- virtual bool **has_default** () const =0
- virtual bool **is_container** () const =0
- virtual bool **has_implicit** () const =0
- virtual std::string **get_default_value** () const =0
- virtual std::string **get_implicit_value** () const =0
- virtual std::shared_ptr< **Value** > **default_value** (const std::string &**value**)=0
- virtual std::shared_ptr< **Value** > **implicit_value** (const std::string &**value**)=0
- virtual ~**Value** ()=default
- virtual void **parse** (const std::string &text) const =0
- virtual void **parse** () const =0
- virtual bool **has_arg** () const =0
- virtual bool **has_default** () const =0
- virtual bool **is_container** () const =0
- virtual bool **has_implicit** () const =0
- virtual std::string **get_default_value** () const =0
- virtual std::string **get_implicit_value** () const =0
- virtual std::shared_ptr< **Value** > **default_value** (const std::string &**value**)=0
- virtual std::shared_ptr< **Value** > **implicit_value** (const std::string &**value**)=0

Detailed Description

Definition at line 241 of file cxxopts.h.

Constructor & Destructor Documentation

virtual cxxopts::Value::~Value () `[virtual]`, `[default]`

virtual cxxopts::Value::~Value () `[virtual]`, `[default]`

Member Function Documentation

virtual std::shared_ptr<Value> cxxopts::Value::default_value (const std::string & *value*)`[pure virtual]`

Implemented in **cxxopts::values::standard_value< T >** (*p.452*), and **cxxopts::values::standard_value< T >** (*p.452*).

virtual std::shared_ptr<Value> cxxopts::Value::default_value (const std::string & *value*)`[pure virtual]`

Implemented in **cxxopts::values::standard_value< T >** (*p.452*), and **cxxopts::values::standard_value< T >** (*p.452*).

virtual std::string cxxopts::Value::get_default_value () const`[pure virtual]`

Implemented in **cxxopts::values::standard_value< T >** (*p.453*), and **cxxopts::values::standard_value< T >** (*p.453*).

virtual std::string cxxopts::Value::get_default_value () const`[pure virtual]`

Implemented in **cxxopts::values::standard_value< T >** (*p.453*), and **cxxopts::values::standard_value< T >** (*p.453*).

virtual std::string cxxopts::Value::get_implicit_value () const`[pure virtual]`

Implemented in **cxxopts::values::standard_value< T >** (*p.453*), and **cxxopts::values::standard_value< T >** (*p.453*).

virtual std::string cxxopts::Value::get_implicit_value () const`[pure virtual]`

Implemented in **cxxopts::values::standard_value< T >** (*p.453*), and **cxxopts::values::standard_value< T >** (*p.453*).

virtual bool cxxopts::Value::has_arg () const`[pure virtual]`

Implemented in **cxxopts::values::standard_value< T >** (*p.453*), and **cxxopts::values::standard_value< T >** (*p.453*).

virtual bool cxxopts::Value::has_arg () const`[pure virtual]`

Implemented in **cxxopts::values::standard_value< T >** (*p.453*), and **cxxopts::values::standard_value< T >** (*p.453*).

virtual bool cxxopts::Value::has_default () const`[pure virtual]`

Implemented in **cxxopts::values::standard_value< T >** (*p.454*), and **cxxopts::values::standard_value< T >** (*p.454*).

virtual bool cxxopts::Value::has_default () const`[pure virtual]`

Implemented in **cxxopts::values::standard_value< T >** (*p.454*), and **cxxopts::values::standard_value< T >** (*p.454*).

virtual bool cxxopts::Value::has_implicit () const`[pure virtual]`

Implemented in **cxxopts::values::standard_value< T >** (*p.454*), and **cxxopts::values::standard_value< T >** (*p.454*).

virtual bool cxxopts::Value::has_implicit () const`[pure virtual]`

Implemented in **cxxopts::values::standard_value< T >** (*p.454*), and **cxxopts::values::standard_value< T >** (*p.454*).

virtual std::shared_ptr<Value> cxxopts::Value::implicit_value (const std::string & *value*)`[pure virtual]`

Implemented in **cxxopts::values::standard_value< T >** (*p.454*), and **cxxopts::values::standard_value< T >** (*p.454*).

virtual std::shared_ptr<Value> cxxopts::Value::implicit_value (const std::string & *value*)`[pure virtual]`

Implemented in **cxxopts::values::standard_value< T >** (*p.454*), and **cxxopts::values::standard_value< T >** (*p.454*).

virtual bool cxxopts::Value::is_container () const`[pure virtual]`

Implemented in **cxxopts::values::standard_value< T >** (*p.455*), and **cxxopts::values::standard_value< T >** (*p.455*).

virtual bool cxxopts::Value::is_container () const`[pure virtual]`

Implemented in **cxxopts::values::standard_value< T >** (*p.455*), and **cxxopts::values::standard_value< T >** (*p.455*).

virtual void cxxopts::Value::parse (const std::string & *text*) const`[pure virtual]`

Implemented in **cxxopts::values::standard_value< T >** (*p.455*), and **cxxopts::values::standard_value< T >** (*p.455*).

virtual void cxxopts::Value::parse (const std::string & *text*) const`[pure virtual]`

Implemented in **cxxopts::values::standard_value< T >** (*p.455*), and **cxxopts::values::standard_value< T >** (*p.455*).

virtual void cxxopts::Value::parse () const`[pure virtual]`

Implemented in **cxxopts::values::standard_value**< **T** > (*p.455*), and **cxxopts::values::standard_value**< **T** > (*p.455*).

virtual void cxxopts::Value::parse () const`[pure virtual]`

Implemented in **cxxopts::values::standard_value**< **T** > (*p.455*), and **cxxopts::values::standard_value**< **T** > (*p.455*).

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

- Code/Assembler/Include/**cxxopts.h**

cxxopts::values::value_has_arg< T > Struct Template Reference

`#include <cxxopts.h>`

Static Public Attributes

- static constexpr bool **value** = true

Detailed Description

template<typename T>

struct cxxopts::values::value_has_arg< T >

Definition at line 450 of file cxxopts.h.

Member Data Documentation

template<typename T > static constexpr bool cxxopts::values::value_has_arg< T >::value = true`[static]`

Definition at line 452 of file cxxopts.h.

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

- Code/Assembler/Include/**cxxopts.h**

cxxopts::values::value_has_arg< bool > Struct Template Reference

`#include <cxxopts.h>`

Static Public Attributes

- static constexpr bool **value** = false

Detailed Description

template<>

struct cxxopts::values::value_has_arg< bool >

Definition at line 456 of file cxxopts.h.

Member Data Documentation

static constexpr bool cxxopts::values::value_has_arg< bool >::value = false `[static]`

Definition at line 458 of file cxxopts.h.

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

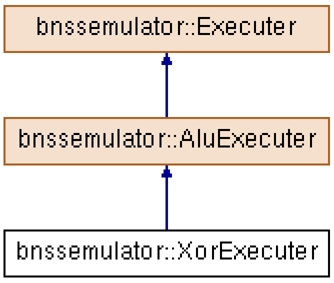
- Code/Assembler/Include/**cxxopts.h**

bnssemulator::XorExecuter Class Reference

Class representing the executer for the xor instruction.

`#include <XorExecuter.h>`

Inheritance diagram for bnssemulator::XorExecuter:



Protected Member Functions

- void **execute** (**Register** &dst, const **Register** &lhs, const **Register** &rhs) const override
  Executes the ALU instruction.

Additional Inherited Members

Detailed Description

Class representing the executer for the xor instruction.

Definition at line 10 of file XorExecuter.h.

Member Function Documentation

void bnssemulator::XorExecuter::execute (Register & *dst*, const Register & *lhs*, const Register & *rhs*) const `[override]`, `[protected]`, `[virtual]`

Executes the ALU instruction.

Parameters:

dst	Reference to the destination register
lhs	Left operand register
rhs	Right operand register

Implements **bnssemulator::AluExecuter** (*p.102*).

Definition at line 5 of file XorExecuter.cpp.

```
{
            dst = lhs ^ rhs;
        }
```

The documentation for this class was generated from the following files:

- Code/Emulator/Include/**XorExecuter.h**
- Code/Emulator/Source/**XorExecuter.cpp**

File Documentation

Code/Assembler/Include/AddOperation.h File Reference

```
#include "Operation.h"
```

Classes

- class **bnssassembler::AddOperation**

Class implementing the behaviour of the + operator in expressions.

Namespaces

- **bnssassembler**

Code/Assembler/Include/AddressMode.h File Reference

Namespaces

- **bnssassembler**

Enumerations

- enum **bnssassembler::AddressMode** { **bnssassembler::IMMEDIATE** = 0b100, **bnssassembler::REGISTER_DIRECT** = 0b000, **bnssassembler::MEMORY_DIRECT** = 0b110, **bnssassembler::REGISTER_INDIRECT** = 0b010, **bnssassembler::REGISTER_INDIRECT_OFFSET** = 0b111 }*Enum representing the address mode.*

Code/Emulator/Include/AddressMode.h File Reference

```
#include <cstdint>
```

Namespaces

- **bnssemulator**

Enumerations

- enum **bnssemulator::AddressMode** : uint32_t { **bnssemulator::IMMEDIATE** = 0b100, **bnssemulator::REGISTER_DIRECT** = 0b000, **bnssemulator::MEMORY_DIRECT** = 0b110, **bnssemulator::REGISTER_INDIRECT** = 0b010, **bnssemulator::REGISTER_INDIRECT_OFFSET** = 0b111 }*Enum representing the address mode.*

Code/Assembler/Include/AddToken.h File Reference

```
#include "OperationToken.h"
```

Classes

- class **bnssassembler::AddToken**

Token ***class representing the + operation.*** Namespaces

- **bnssassembler**

Code/Assembler/Include/AluInstructionParser.h File Reference

`#include "InstructionParser.h"`

Classes

- class **bnssassembler::AluInstructionParser**

Class representing the parser for ALU instructions. Namespaces

- **bnssassembler**

Code/Assembler/Include/AssemblerException.h File Reference

```
#include <string>
```

Classes

- class **bnssassembler::AssemblerException**

Class representing the custom exception for the assembler.

Namespaces

- **bnssassembler**

Code/Assembler/Include/ClosingBraceToken.h File Reference

`#include "OperationToken.h"`

Classes

- class **bnssassembler::ClosingBraceToken**

Token ***class representing the opening brace.*** Namespaces

- **bnssassembler**

Code/Assembler/Include/CommandLineHelper.h File Reference

```
#include <utility>
#include <string>
```

Classes

- class **bnssassembler::CommandLineHelper**

Utility class used to parse the command line. Namespaces

- **bnssassembler**

Code/Emulator/Include/CommandLineHelper.h File Reference

```
#include <utility>
#include <string>
```

Classes

- class **bnssemulator::CommandLineHelper**

Utility class used for parsing the command line. Namespaces

- **bnssemulator**

Code/Assembler/Include/CommonRegexes.h File Reference

```
#include <regex>
```

Namespaces

- **bnssassembler**

Variables

- const std::string **bnssassembler::ZERO** = "0"
- const std::string **bnssassembler::DECIMAL** = "[1-9][0-9]*"
- const std::string **bnssassembler::HEX** = "0x[0-9a-fA-F][0-9a-fA-F]*"
- const std::string **bnssassembler::BINARY** = "0b[01][01]*"
- const std::string **bnssassembler::OCT** = "0[0-7][0-7]*"
- const std::string **bnssassembler::CHARACTER** = "'[[:print:]]'"
- const std::string **bnssassembler::LITERAL** = "(" + ZERO + "|" + DECIMAL + "|" + HEX + "|" + BINARY + "|" + OCT + "|" + CHARACTER + ")"
- const std::string **bnssassembler::OPERATOR** = "[-+*/()]"
- const std::string **bnssassembler::SYMBOL** = "(([a-zA-Z_][a-zA-Z_0-9]*)|\\$)"
- const std::string **bnssassembler::LABEL** = SYMBOL
- const std::string **bnssassembler::CONSTANT_TERM** = "([[:space:]]*(" + LITERAL + "|" + OPERATOR + "|" + SYMBOL + ")[[:space:]]*)*"
- const std::string **bnssassembler::ORG_DIRECTIVE** = "[Oo][Rr][Gg]"
- const std::string **bnssassembler::SYMBOL_DEFINITION** = "[Dd][Ee][Ff]"
- const std::string **bnssassembler::DUPLICATE_DIRECTIVE** = "[Dd][Uu][Pp]"
- const std::string **bnssassembler::GLOBAL_DIRECTIVE** = "[.][Gg][Ll][Oo][Bb][Aa][Ll]"
- const std::string **bnssassembler::COMMA_TOKENIZER** = "[[:space:]]*(.*?)[[:space:]]*,(.*)"
- const std::string **bnssassembler::LAST_COMMA_TOKEN** = "[[:space:]]*(.*)[[:space:]]*"
- const std::regex **bnssassembler::ZERO_REGEX** = std::regex(ZERO)
- const std::regex **bnssassembler::DECIMAL_REGEX** = std::regex(DECIMAL)
- const std::regex **bnssassembler::HEX_REGEX** = std::regex(HEX)
- const std::regex **bnssassembler::BINARY_REGEX** = std::regex(BINARY)
- const std::regex **bnssassembler::OCT_REGEX** = std::regex(OCT)
- const std::regex **bnssassembler::CHARACTER_REGEX** = std::regex(CHARACTER)
- const std::regex **bnssassembler::LITERAL_REGEX** = std::regex(LITERAL)
- const std::regex **bnssassembler::OPERATOR_REGEX** = std::regex(OPERATOR)
- const std::regex **bnssassembler::SYMBOL_REGEX** = std::regex(SYMBOL)
- const std::regex **bnssassembler::LABEL_REGEX** = std::regex(LABEL)
- const std::regex **bnssassembler::CONSTANT_TERM_REGEX** = std::regex(CONSTANT_TERM)
- const std::regex **bnssassembler::ORG_DIRECTIVE_REGEX** = std::regex(ORG_DIRECTIVE)
- const std::regex **bnssassembler::SYMBOL_DEFINITION_REGEX** = std::regex(SYMBOL_DEFINITION)
- const std::regex **bnssassembler::DUPLICATE_DIRECTIVE_REGEX** = std::regex(DUPLICATE_DIRECTIVE)
- const std::regex **bnssassembler::GLOBAL_DIRECTIVE_REGEX** = std::regex(GLOBAL_DIRECTIVE)
- const std::regex **bnssassembler::COMMA_TOKENIZER_REGEX** = std::regex(COMMA_TOKENIZER)
- const std::regex **bnssassembler::LAST_COMMA_TOKEN_REGEX** = std::regex(LAST_COMMA_TOKEN)

Code/Emulator/Include/CommonRegexes.h File Reference

```
#include <regex>
```

Namespaces

- **bnssemulator**

Variables

- const std::string **bnssemulator::ZERO** = "0"
- const std::string **bnssemulator::DECIMAL** = "[1-9][0-9]*"
- const std::string **bnssemulator::HEX** = "0x[0-9a-fA-F][0-9a-fA-F]*"
- const std::string **bnssemulator::BINARY** = "0b[01][01]*"
- const std::string **bnssemulator::OCT** = "0[0-7][0-7]*"
- const std::string **bnssemulator::CHARACTER** = "'[[:print:]]'"
- const std::string **bnssemulator::LITERAL** = "(" + ZERO + "|" + DECIMAL + "|" + HEX + "|" + BINARY + "|" + OCT + "|" + CHARACTER + ")"
- const std::string **bnssemulator::OPERATOR** = "[-+*/()]"
- const std::string **bnssemulator::SYMBOL** = "(([a-zA-Z_][a-zA-Z_0-9]*)|\\$)"
- const std::string **bnssemulator::LABEL** = SYMBOL
- const std::string **bnssemulator::CONSTANT_TERM** = "([[:space:]]*(" + LITERAL + "|" + OPERATOR + "|" + SYMBOL + ")[[:space:]]*)*"
- const std::string **bnssemulator::ORG_DIRECTIVE** = "[Oo][Rr][Gg]"
- const std::string **bnssemulator::SYMBOL_DEFINITION** = "[Dd][Ee][Ff]"
- const std::string **bnssemulator::DUPLICATE_DIRECTIVE** = "[Dd][Uu][Pp]"
- const std::string **bnssemulator::GLOBAL_DIRECTIVE** = "[.][Gg][Ll][Oo][Bb][Aa][Ll]"
- const std::string **bnssemulator::COMMA_TOKENIZER** = "[[:space:]]*(.*?)[[:space:]]*,(.*)"
- const std::string **bnssemulator::LAST_COMMA_TOKEN** = "[[:space:]]*(.*)[[:space:]]*"
- const std::regex **bnssemulator::ZERO_REGEX** = std::regex(ZERO)
- const std::regex **bnssemulator::DECIMAL_REGEX** = std::regex(DECIMAL)
- const std::regex **bnssemulator::HEX_REGEX** = std::regex(HEX)
- const std::regex **bnssemulator::BINARY_REGEX** = std::regex(BINARY)
- const std::regex **bnssemulator::OCT_REGEX** = std::regex(OCT)
- const std::regex **bnssemulator::CHARACTER_REGEX** = std::regex(CHARACTER)
- const std::regex **bnssemulator::LITERAL_REGEX** = std::regex(LITERAL)
- const std::regex **bnssemulator::OPERATOR_REGEX** = std::regex(OPERATOR)
- const std::regex **bnssemulator::SYMBOL_REGEX** = std::regex(SYMBOL)
- const std::regex **bnssemulator::LABEL_REGEX** = std::regex(LABEL)
- const std::regex **bnssemulator::CONSTANT_TERM_REGEX** = std::regex(CONSTANT_TERM)
- const std::regex **bnssemulator::ORG_DIRECTIVE_REGEX** = std::regex(ORG_DIRECTIVE)
- const std::regex **bnssemulator::SYMBOL_DEFINITION_REGEX** = std::regex(SYMBOL_DEFINITION)
- const std::regex **bnssemulator::DUPLICATE_DIRECTIVE_REGEX** = std::regex(DUPLICATE_DIRECTIVE)
- const std::regex **bnssemulator::GLOBAL_DIRECTIVE_REGEX** = std::regex(GLOBAL_DIRECTIVE)
- const std::regex **bnssemulator::COMMA_TOKENIZER_REGEX** = std::regex(COMMA_TOKENIZER)
- const std::regex **bnssemulator::LAST_COMMA_TOKEN_REGEX** = std::regex(LAST_COMMA_TOKEN)

Code/Assembler/Include/ConditionalJumpInstructionParser.h File Reference

```
#include "InstructionParser.h"
```

Classes

- class **bnssassembler::ConditionalJumpInstructionParser**

Class representing the parser for conditional jump instructions.

Namespaces

- **bnssassembler**

Code/Assembler/Include/cxxopts.h File Reference

```
#include <map>
#include <memory>
#include <regex>
#include <sstream>
#include <unordered_set>
#include <vector>
```

Classes

- class **cxxopts::Value**
- class **cxxopts::OptionException**
- class **cxxopts::OptionSpecException**
- class **cxxopts::OptionParseException**
- class **cxxopts::option_exists_error**
- class **cxxopts::invalid_option_format_error**
- class **cxxopts::option_not_exists_exception**
- class **cxxopts::missing_argument_exception**
- class **cxxopts::option_requires_argument_exception**
- class **cxxopts::option_not_has_argument_exception**
- class **cxxopts::option_not_present_exception**
- class **cxxopts::argument_incorrect_type**
- class **cxxopts::option_required_exception**
- struct **cxxopts::values::value_has_arg< T >**
- struct **cxxopts::values::value_has_arg< bool >**
- struct **cxxopts::values::type_is_container< T >**
- struct **cxxopts::values::type_is_container< std::vector< T > >**
- class **cxxopts::values::standard_value< T >**
- class **cxxopts::OptionDetails**
- struct **cxxopts::HelpOptionDetails**
- struct **cxxopts::HelpGroupDetails**
- class **cxxopts::Options**
- class **cxxopts::OptionAdder**

Namespaces

- **cxxopts**
- **cxxopts::values**
- **cxxopts::anonymous_namespace{cxxopts.h}**

Typedefs

- typedef std::string **cxxopts::String**

Functions

- template<typename T > T **cxxopts::toLocalString** (T &&t)
- size_t **cxxopts::stringLength** (const String &s)
- String & **cxxopts::stringAppend** (String &s, String a)
- String & **cxxopts::stringAppend** (String &s, size_t n, char c)
- template<typename Iterator > String & **cxxopts::stringAppend** (String &s, Iterator begin, Iterator end)
- template<typename T > std::string **cxxopts::toUTF8String** (T &&t)
- bool **cxxopts::empty** (const std::string &s)
- template<typename T > void **cxxopts::values::parse_value** (const std::string &text, T &value)
- void **cxxopts::values::parse_value** (const std::string &, bool &value)
- void **cxxopts::values::parse_value** (const std::string &text, std::string &value)

- template<typename T > void **cxxopts::values::parse_value** (const std::string &text, std::vector< T > &value)
- template<typename T > std::shared_ptr< Value > **cxxopts::value** ()
- template<typename T > std::shared_ptr< Value > **cxxopts::value** (T &t)
- void **cxxopts::check_required** (const Options &options, const std::vector< std::string > &required)
- std::basic_regex< char > **cxxopts::anonymous_namespace{cxxopts.h}::option_matcher** ("--([[:alnum:]][-_[:alnum:]]+)(=(.*))?|-([[:alnum:]]+)")
- std::basic_regex< char > **cxxopts::anonymous_namespace{cxxopts.h}::option_specifier** ("(([[:alnum:]]),)?([[:alnum:]][-_[:alnum:]]*)?")
- String **cxxopts::anonymous_namespace{cxxopts.h}::format_option** (const HelpOptionDetails &o)
- String **cxxopts::anonymous_namespace{cxxopts.h}::format_description** (const HelpOptionDetails &o, size_t start, size_t width)

Variables

- constexpr int **cxxopts::anonymous_namespace{cxxopts.h}::OPTION_LONGEST** = 30
- constexpr int **cxxopts::anonymous_namespace{cxxopts.h}::OPTION_DESC_GAP** = 2

Code/Emulator/Include/cxxopts.h File Reference

```
#include <map>
#include <memory>
#include <regex>
#include <sstream>
#include <unordered_set>
#include <vector>
```

Classes

- class **cxxopts::Value**
- class **cxxopts::OptionException**
- class **cxxopts::OptionSpecException**
- class **cxxopts::OptionParseException**
- class **cxxopts::option_exists_error**
- class **cxxopts::invalid_option_format_error**
- class **cxxopts::option_not_exists_exception**
- class **cxxopts::missing_argument_exception**
- class **cxxopts::option_requires_argument_exception**
- class **cxxopts::option_not_has_argument_exception**
- class **cxxopts::option_not_present_exception**
- class **cxxopts::argument_incorrect_type**
- class **cxxopts::option_required_exception**
- struct **cxxopts::values::value_has_arg< T >**
- struct **cxxopts::values::value_has_arg< bool >**
- struct **cxxopts::values::type_is_container< T >**
- struct **cxxopts::values::type_is_container< std::vector< T > >**
- class **cxxopts::values::standard_value< T >**
- class **cxxopts::OptionDetails**
- struct **cxxopts::HelpOptionDetails**
- struct **cxxopts::HelpGroupDetails**
- class **cxxopts::Options**
- class **cxxopts::OptionAdder**

Namespaces

- **cxxopts**
- **cxxopts::values**
- **cxxopts::anonymous_namespace{cxxopts.h}**

Functions

- template<typename T > T **cxxopts::toLocalString** (T &&t)
- size_t **cxxopts::stringLength** (const String &s)
- String & **cxxopts::stringAppend** (String &s, String a)
- String & **cxxopts::stringAppend** (String &s, size_t n, char c)
- template<typename Iterator > String & **cxxopts::stringAppend** (String &s, Iterator begin, Iterator end)
- template<typename T > std::string **cxxopts::toUTF8String** (T &&t)
- bool **cxxopts::empty** (const std::string &s)
- template<typename T > void **cxxopts::values::parse_value** (const std::string &text, T &value)
- void **cxxopts::values::parse_value** (const std::string &, bool &value)
- void **cxxopts::values::parse_value** (const std::string &text, std::string &value)
- template<typename T > void **cxxopts::values::parse_value** (const std::string &text, std::vector< T > &value)
- template<typename T > std::shared_ptr< Value > **cxxopts::value** ()
- template<typename T > std::shared_ptr< Value > **cxxopts::value** (T &t)

- void **cxxopts::check_required** (const Options &options, const std::vector< std::string > &required)
- std::basic_regex< char > **cxxopts::anonymous_namespace{cxxopts.h}::option_matcher** ("--([[:alnum:]][-_[:alnum:]]+)(=(.*))?|-([[:alnum:]]+)")
- std::basic_regex< char > **cxxopts::anonymous_namespace{cxxopts.h}::option_specifier** ("(([[:alnum:]]),)?([[:alnum:]][-_[:alnum:]]*)?")
- String **cxxopts::anonymous_namespace{cxxopts.h}::format_option** (const HelpOptionDetails &o)
- String **cxxopts::anonymous_namespace{cxxopts.h}::format_description** (const HelpOptionDetails &o, size_t start, size_t width)

Code/Assembler/Include/Data.h File Reference

```
#include "DataType.h"
#include "MicroRiscExpression.h"
```

Classes

- class **bnssassembler::Data**

Class representing the MicroRISC data. Namespaces

- **bnssassembler**

Code/Assembler/Include/DataDefinitionLineParser.h File Reference

```
#include "LineParser.h"
```

Classes

- class **bnssassembler::DataDefinitionLineParser**

Class used for parsing data definitions. Namespaces

- **bnssassembler**

Code/Assembler/Include/DataDefinitionToken.h File Reference

```
#include <vector>
#include "Data.h"
#include "Token.h"
```

Classes

- class **bnssassembler::DataDefinitionToken**

Class representing the data definition token. Namespaces

- **bnssassembler**

Code/Assembler/Include/DataType.h File Reference

Namespaces

- **bnssassembler**

Enumerations

- enum **bnssassembler::DataType** { **bnssassembler::DOUBLE_WORD** = 0, **bnssassembler::WORD**, **bnssassembler::BYTE** }*Enum representing a data type.*

Code/Emulator/Include/DataType.h File Reference

Namespaces

- **bnssemulator**

Enumerations

- enum **bnssemulator::DataType** : int8_t { **bnssemulator::DOUBLE_WORD** = 0, **bnssemulator::WORD**, **bnssemulator::BYTE** }*Enum representing a data type.*

Code/Assembler/Include/DataTypeParser.h File Reference

```
#include "DataType.h"
#include <unordered_map>
```

Classes

- class **bnssassembler::DataTypeParser**
- *Utility class used for parsing data types.* struct **bnssassembler::DataTypeParser::DataTypeParserStaticData**

Namespaces

- **bnssassembler**

Code/Assembler/Include/DivideOperation.h File Reference

```
#include "Operation.h"
```

Classes

- class **bnssassembler::DivideOperation**

Class implementing the behaviour of the / operator in expressions.

Namespaces

- **bnssassembler**

Code/Assembler/Include/DivideToken.h File Reference

`#include "OperationToken.h"`

Classes

- class **bnssassembler::DivideToken**

Token ***class representing the / operation.*** Namespaces

- **bnssassembler**

Code/Assembler/Include/DivisionByZeroException.h File Reference

```
#include "MessageException.h"
```

Classes

- class **bnssassembler::DivisionByZeroException**

Exception class representing division by zero. Namespaces

- **bnssassembler**

Code/Assembler/Include/Expression.h File Reference

```
#include <cstdint>
#include <memory>
#include <stack>
#include "SymbolTable.h"
#include <unordered_set>
#include <list>
#include "RelocationRecord.h"
```

Classes

- class **bnssassembler::Expression**

Class representing the math expression. Namespaces

- **bnssassembler**

Code/Assembler/Include/ExpressionBuilder.h File Reference

```
#include "MicroRiscExpression.h"
#include "ExpressionToken.h"
```

Classes

- class **bnssassembler::ExpressionBuilder**

Utility class used for building math expressions. Namespaces

- **bnssassembler**

Code/Assembler/Include/ExpressionToken.h File Reference

```
#include <memory>
#include <stack>
#include <list>
#include "Expression.h"
```

Classes

- class **bnssassembler::ExpressionToken**

Class representing the token found in infix and postfix expressions.

Namespaces

- **bnssassembler**

Code/Assembler/Include/ExpressionTokenFactory.h File Reference

```
#include <memory>
#include "ExpressionToken.h"
#include <unordered_map>
```

Classes

- class **bnssassembler::ExpressionTokenFactory**
- *Utility class used for creating the **ExpressionToken** objects.* struct **bnssassembler::ExpressionTokenFactory::ExpressionTokenFactoryData**

Namespaces

- **bnssassembler**

Code/Assembler/Include/FileReader.h File Reference

```
#include <vector>
#include <string>
```

Classes

- class **bnssassembler::FileReader**

Utility class providing methods for reading the file. Namespaces

- **bnssassembler**

Code/Emulator/Include/FileReader.h File Reference

```
#include "AssemblerOutput.h"
#include <string>
```

Classes

- class **bnssemulator::FileReader**

Utility class used for reading assembler output from the file.

Namespaces

- **bnssemulator**

Code/Assembler/Include/FileWriter.h File Reference

```
#include <string>
#include "SecondPassData.h"
```

Classes

- class **bnssassembler::FileWriter**

Utility class used to write the assembler result to a file. Namespaces

- **bnssassembler**

Code/Assembler/Include/FirstPass.h File Reference

```
#include "FirstPassData.h"
#include "Token.h"
```

Classes

- class **bnssassembler::FirstPass**

Class representing the executor of the first pass. Namespaces

- **bnssassembler**

Code/Assembler/Include/FirstPassData.h File Reference

```
#include "SymbolTable.h"
#include "SectionTable.h"
#include "SymbolDefinition.h"
#include <unordered_set>
```

Classes

- class **bnssassembler::FirstPassData**

Class representing the data that the two-pass assembler will modify in the first pass. Namespaces

- **bnssassembler**

Code/Assembler/Include/FirstPassException.h File Reference

```
#include "AssemblerException.h"
```

Classes

- class **bnssassembler::FirstPassException**

Represents an exception that happend during the assembler first pass.

Namespaces

- **bnssassembler**

Code/Assembler/Include/GlobalSymbolsLineParser.h File Reference

```
#include "LineParser.h"
```

Classes

- class **bnssassembler::GlobalSymbolsLineParser**

Class used for parsing information about global symbols. Namespaces

- **bnssassembler**

Code/Assembler/Include/GlobalSymbolToken.h File Reference

```
#include "Token.h"
#include <vector>
```

Classes

- class **bnssassembler::GlobalSymbolsToken**

Class representing the global symbols token. Namespaces

- **bnssassembler**

Code/Assembler/Include/Immediate.h File Reference

```
#include "Operand.h"
#include "MicroRiscExpression.h"
```

Classes

- class **bnssassembler::Immediate**

Class representing the immediate operand. Namespaces

- **bnssassembler**

Code/Assembler/Include/ImmediateParser.h File Reference

```
#include "OperandParser.h"
```

Classes

- class **bnssassembler::ImmediateParser**

Class representing the parser for the immediate operands. Namespaces

- **bnssassembler**

Code/Assembler/Include/IncorrectLabelException.h File Reference

```
#include <string>
#include "MessageException.h"
```

Classes

- class **bnssassembler::IncorrectLabelException**

Exception representing the incorrect label. Namespaces

- **bnssassembler**

Code/Assembler/Include/InstructionBitField.h File Reference

`#include <cstdint>`

Classes

- struct **bnssassembler::InstructionBitField**

Bit field that enables easier manipulation of instructions. Namespaces

- **bnssassembler**

Code/Emulator/Include/InstructionBitField.h File Reference

```
#include <cstdint>
```

Classes

- struct **bnssemulator::InstructionBitField**

Bit field that enables easier manipulation of instructions. Namespaces

- **bnssemulator**

Code/Assembler/Include/InstructionBitFieldUnion.h File Reference

```
#include "InstructionBitField.h"
```

Classes

- union **bnssassembler::InstructionBitFieldUnion**

Union that enables easier manipulation of the instruction bit field.

Namespaces

- **bnssassembler**

Code/Emulator/Include/InstructionBitFieldUnion.h File Reference

```
#include "InstructionBitField.h"
```

Classes

- union **bnssemulator::InstructionBitFieldUnion**

Union that enables easier manipulation of the instruction bit field.

Namespaces

- **bnssemulator**

Code/Assembler/Include/InstructionCode.h File Reference

```
#include <cstdint>
#include <utility>
```

Classes

- struct **std::hash< bnssassembler::InstructionCode >**

Namespaces

- **bnssassembler**
- **std**

Enumerations

- enum **bnssassembler::InstructionCode** : int8_t { **bnssassembler::INT** = 0x00, **bnssassembler::JMP** = 0x02, **bnssassembler::CALL** = 0x03, **bnssassembler::RET** = 0x01, **bnssassembler::JZ** = 0x04, **bnssassembler::JNZ** = 0x05, **bnssassembler::JGZ** = 0x06, **bnssassembler::JGEZ** = 0x07, **bnssassembler::JLZ** = 0x08, **bnssassembler::JLEZ** = 0x09, **bnssassembler::LOAD** = 0x10, **bnssassembler::STORE** = 0x11, **bnssassembler::PUSH** = 0x20, **bnssassembler::POP** = 0x21, **bnssassembler::ADD** = 0x30, **bnssassembler::SUB** = 0x31, **bnssassembler::MUL** = 0x32, **bnssassembler::DIV** = 0x33, **bnssassembler::MOD** = 0x34, **bnssassembler::AND** = 0x35, **bnssassembler::OR** = 0x36, **bnssassembler::XOR** = 0x37, **bnssassembler::NOT** = 0x38, **bnssassembler::ASL** = 0x39, **bnssassembler::ASR** = 0x3A }*Enum representing the instruction code.*

Code/Emulator/Include/InstructionCode.h File Reference

```
#include <cstdint>
#include <utility>
```

Classes

- struct **std::hash< bnssemulator::InstructionCode >**

Namespaces

- **bnssemulator**
- **std**

Enumerations

- enum **bnssemulator::InstructionCode** : int8_t { **bnssemulator::INT** = 0x00, **bnssemulator::JMP** = 0x02, **bnssemulator::CALL** = 0x03, **bnssemulator::RET** = 0x01, **bnssemulator::JZ** = 0x04, **bnssemulator::JNZ** = 0x05, **bnssemulator::JGZ** = 0x06, **bnssemulator::JGEZ** = 0x07, **bnssemulator::JLZ** = 0x08, **bnssemulator::JLEZ** = 0x09, **bnssemulator::LOAD** = 0x10, **bnssemulator::STORE** = 0x11, **bnssemulator::PUSH** = 0x20, **bnssemulator::POP** = 0x21, **bnssemulator::ADD** = 0x30, **bnssemulator::SUB** = 0x31, **bnssemulator::MUL** = 0x32, **bnssemulator::DIV** = 0x33, **bnssemulator::MOD** = 0x34, **bnssemulator::AND** = 0x35, **bnssemulator::OR** = 0x36, **bnssemulator::XOR** = 0x37, **bnssemulator::NOT** = 0x38, **bnssemulator::ASL** = 0x39, **bnssemulator::ASR** = 0x3A }*Enum representing the instruction code.*

Code/Assembler/Include/InstructionCodeParser.h File Reference

```
#include <unordered_map>
#include "InstructionCode.h"
```

Classes

- class **bnssassembler::InstructionCodeParser**
- *Utility class used for parsing instruction codes.* struct **bnssassembler::InstructionCodeParser::InstructionCodeParserStaticData**

Namespaces

- **bnssassembler**

Code/Assembler/Include/InstructionLineParser.h File Reference

```
#include "LineParser.h"
#include "InstructionCode.h"
#include "InstructionParser.h"
#include <memory>
#include <unordered_map>
```

Classes

- class **bnssassembler::InstructionLineParser**

Class used for parsing instructions. Namespaces

- **bnssassembler**

Code/Assembler/Include/InstructionParser.h File Reference

```
#include <memory>
#include "InstructionToken.h"
#include "OperandParser.h"
```

Classes

- class **bnssassembler::InstructionParser**

Abstract lass used for parsing one instruction. Namespaces

- **bnssassembler**

Code/Assembler/Include/InstructionToken.h File Reference

```
#include "Token.h"
#include <vector>
#include "Operand.h"
#include "InstructionCode.h"
#include <memory>
#include "OperandType.h"
```

Classes

- class **bnssassembler::InstructionToken**

Class representing the instruction in an assembler source file.

Namespaces

- **bnssassembler**

Code/Assembler/Include/InterruptInstructionParser.h File Reference

```
#include "InstructionParser.h"
```

Classes

- class **bnssassembler::InterruptInstructionParser**

Class representing the parser for the interrupt instruction. Namespaces

- **bnssassembler**

Code/Assembler/Include/InvalidDataDefinitionException.h File Reference

```
#include "MessageException.h"
```

Classes

- class **bnssassembler::InvalidDataDefinitionException**

Exception representing invalid data definition. Namespaces

- **bnssassembler**

Code/Assembler/Include/InvalidDataTypeException.h File Reference

```
#include "MessageException.h"
```

Classes

- class **bnssassembler::InvalidDataTypeException**

Exception representing the invalid data type. Namespaces

- **bnssassembler**

Code/Assembler/Include/InvalidExpressionException.h File Reference

```
#include "MessageException.h"
```

Classes

- class **bnssassembler::InvalidExpressionException**

Exception representing the invalid expression. Namespaces

- **bnssassembler**

Code/Assembler/Include/LabelToken.h File Reference

```
#include "Token.h"
```

Classes

- class **bnssassembler::LabelToken**

Class representing the label token. Namespaces

- **bnssassembler**

Code/Assembler/Include/LineParser.h File Reference

```
#include "Token.h"
#include <functional>
#include <memory>
```

Classes

- class **bnssassembler::LineParser**

Chain of command abstract class used for parsing one line of file.

Namespaces

- **bnssassembler**

Code/Assembler/Include/Literal.h File Reference

```
#include "Expression.h"
#include <cstdint>
```

Classes

- class **bnssassembler::Literal**

Class representing the literal value. Namespaces

- **bnssassembler**

Code/Assembler/Include/LiteralToken.h File Reference

```
#include "ExpressionToken.h"
```

Classes

- class **bnssassembler::LiteralToken**

Token ***class representing a math literal value.*** Namespaces

- **bnssassembler**

Code/Assembler/Include/LoadInstructionParser.h File Reference

```
#include "InstructionParser.h"
```

Classes

- class **bnssassembler::LoadInstructionParser**

Class representing the load instruction parser. Namespaces

- **bnssassembler**

Code/Assembler/Include/MemoryDirect.h File Reference

```
#include "Operand.h"
#include "MicroRiscExpression.h"
```

Classes

- class **bnssassembler::MemoryDirect**

Class representing the memory direct operand. Namespaces

- **bnssassembler**

Code/Assembler/Include/MemoryDirectParser.h File Reference

```
#include "OperandParser.h"
```

Classes

- class **bnssassembler::MemoryDirectParser**

Class representing the parser for the memory direct operand.

Namespaces

- **bnssassembler**

Code/Assembler/Include/MessageException.h File Reference

```
#include <string>
#include <exception>
```

Classes

- class **bnssassembler::MessageException**

Represents an exception with a string message. Namespaces

- **bnssassembler**

Code/Emulator/Include/MessageException.h File Reference

```
#include <string>
#include <exception>
```

Classes

- class **bnssemulator::MessageException**

Represents an exception with a string message. Namespaces

- **bnssemulator**

Code/Assembler/Include/MicroRiscExpression.h File Reference

```
#include <memory>
#include "Expression.h"
```

Classes

- class **bnssassembler::MicroRiscExpression**

Adapter class for Expression*.* Namespaces

- **bnssassembler**

Code/Assembler/Include/MicroRiscParser.h File Reference

```
#include "Parser.h"
```

Classes

- class **bnssassembler::MicroRiscParser**

Class representing the parser for the MicroRISC assembly. Namespaces

- **bnssassembler**

Code/Assembler/Include/MultiplyOperation.h File Reference

```
#include "Operation.h"
```

Classes

- class **bnssassembler::MultiplyOperation**

*Class implementing the behaviour of the * operator in expressions.*

Namespaces

- **bnssassembler**

Code/Assembler/Include/MultiplyToken.h File Reference

`#include "OperationToken.h"`

Classes

- class **bnssassembler::MultiplyToken**

Token ***class representing the * operation.*** Namespaces

- **bnssassembler**

Code/Assembler/Include/NonExistingSymbolException.h File Reference

```
#include <string>
#include "MessageException.h"
```

Classes

- class **bnssassembler::NonExistingSymbolException**

Exception representing the non existing symbol. Namespaces

- **bnssassembler**

Code/Assembler/Include/NoOperandInstructionParser.h File Reference

```
#include "InstructionParser.h"
```

Classes

- class **bnssassembler::NoOperandInstructionParser**

Class representing the parser for the instruction without operands.
Namespaces

- **bnssassembler**

Code/Assembler/Include/NotInstructionParser.h File Reference

```
#include "InstructionParser.h"
```

Classes

- class **bnssassembler::NotInstructionParser**

Class representing the parser for the not instruction. Namespaces

- **bnssassembler**

Code/Assembler/Include/OpeningBraceToken.h File Reference

```
#include "OperationToken.h"
```

Classes

- class **bnssassembler::OpeningBraceToken**

Token ***class representing the opening brace.*** Namespaces

- **bnssassembler**

Code/Assembler/Include/Operand.h File Reference

```
#include "SymbolDefinition.h"
#include "AddressMode.h"
#include <unordered_set>
#include "InstructionBitFieldUnion.h"
```

Classes

- class **bnssassembler::Operand**

Class representing one operand in an instruction. Namespaces

- **bnssassembler**

Code/Assembler/Include/OperandParser.h File Reference

```
#include <memory>
#include "Operand.h"
#include <string>
```

Classes

- class **bnssassembler::OperandParser**

Chain of command class used to parse operands of the instructions.

Namespaces

- **bnssassembler**

Code/Assembler/Include/OperandType.h File Reference

`#include <cstdint>`

Namespaces

- **bnssassembler**

Enumerations

- enum **bnssassembler::OperandType** : int8_t { **bnssassembler::DEFAULT** = 0b000, **bnssassembler::UNSIGNED_BYTE** = 0b011, **bnssassembler::SIGNED_BYTE** = 0b111, **bnssassembler::REGULAR_BYTE** = 0b111, **bnssassembler::UNSIGNED_WORD** = 0b001, **bnssassembler::SIGNED_WORD** = 0b101, **bnssassembler::REGULAR_WORD** = 0b101, **bnssassembler::REGULAR_DOUBLE_WORD** = 0b000 }*Enum representing the operand type.*

Code/Emulator/Include/OperandType.h File Reference

```
#include <cstdint>
```

Namespaces

- **bnssemulator**

Enumerations

- enum **bnssemulator::OperandType** : int8_t { **bnssemulator::DEFAULT** = 0b000, **bnssemulator::UNSIGNED_BYTE** = 0b011, **bnssemulator::SIGNED_BYTE** = 0b111, **bnssemulator::REGULAR_BYTE** = 0b111, **bnssemulator::UNSIGNED_WORD** = 0b001, **bnssemulator::SIGNED_WORD** = 0b101, **bnssemulator::REGULAR_WORD** = 0b101, **bnssemulator::REGULAR_DOUBLE_WORD** = 0b000 }*Enum representing the operand type.*

Code/Assembler/Include/Operation.h File Reference

```
#include "Expression.h"
#include <memory>
```

Classes

- class **bnssassembler::Operation**

Class representing the mathematical operation with two operands.

Namespaces

- **bnssassembler**

Code/Assembler/Include/OperationToken.h File Reference

```
#include "ExpressionToken.h"
```

Classes

- class **bnssassembler::OperationToken**

Token ***class representing a math operator.*** Namespaces

- **bnssassembler**

Code/Assembler/Include/OrgDirectiveLineParser.h File Reference

```
#include "LineParser.h"
```

Classes

- class **bnssassembler::OrgDirectiveLineParser**

Class representing a line parser for the origin directive. Namespaces

- **bnssassembler**

Code/Assembler/Include/OrgDirectiveToken.h File Reference

```
#include "Token.h"
#include "MicroRiscExpression.h"
```

Classes

- class **bnssassembler::OrgDirectiveToken**

Class representing the origin directive token. Namespaces

- **bnssassembler**

Code/Assembler/Include/Parser.h File Reference

```
#include <vector>
#include "Token.h"
#include <memory>
#include "LineParser.h"
```

Classes

- class **bnssassembler::Parser**

Abstract class representing a text parser. Namespaces

- **bnssassembler**

Code/Assembler/Include/ParserException.h File Reference

```
#include "AssemblerException.h"
```

Classes

- class **bnssassembler::ParserException**

Represents an exception that happend during the parsing of the file.

Namespaces

- **bnssassembler**

Code/Assembler/Include/PrintHelpers.h File Reference

`#include <string>`

Namespaces

- **bnssassembler**

Functions

- std::string **bnssassembler::multiple** (unsigned char c, size_t times)
  Returns a string containing multiple of the same characters.
- std::string **bnssassembler::multiple** (std::string s, size_t times)
  Returns a string containing multiple of the same strings.

Variables

- const std::string **bnssassembler::UPPER_LEFT** = "\u2554"
- const std::string **bnssassembler::UPPER_RIGHT** = "\u2557"
- const std::string **bnssassembler::LOWER_LEFT** = "\u255a"
- const std::string **bnssassembler::LOWER_RIGHT** = "\u255d"
- const std::string **bnssassembler::HORIZONTAL** = "\u2550"
- const std::string **bnssassembler::VERTICAL** = "\u2551"
- const std::string **bnssassembler::T_LEFT** = "\u2563"
- const std::string **bnssassembler::T_RIGHT** = "\u2560"
- const std::string **bnssassembler::T_UP** = "\u2569"
- const std::string **bnssassembler::T_DOWN** = "\u2566"
- const std::string **bnssassembler::ALL_FOUR** = "\u256c"

Code/Assembler/Include/Register.h File Reference

`#include <cstddef>`

Namespaces

- **bnssassembler**

Enumerations

- enum **bnssassembler::Register** { **bnssassembler::R0** = 0x00, **bnssassembler::R1**, **bnssassembler::R2**, **bnssassembler::R3**, **bnssassembler::R4**, **bnssassembler::R5**, **bnssassembler::R6**, **bnssassembler::R7**, **bnssassembler::R8**, **bnssassembler::R9**, **bnssassembler::R10**, **bnssassembler::R11**, **bnssassembler::R12**, **bnssassembler::R13**, **bnssassembler::R14**, **bnssassembler::R15**, **bnssassembler::SP** = 0x10, **bnssassembler::PC** = 0x11, **bnssassembler::NONE** = 0x1F }*Enum representing a register.*

Variables

- const size_t **bnssassembler::NUM_OF_REGISTERS** = 16
  Number of all purpose registers (excluding PC and SP)

Code/Emulator/Include/Register.h File Reference

`#include <cstdint>`

Classes

- class **bnssemulator::Register**

Class representing the register. Namespaces

- **bnssemulator**

Code/Assembler/Include/RegisterDirect.h File Reference

```
#include "Register.h"
#include "Operand.h"
```

Classes

- class **bnssassembler::RegisterDirect**

Class representing the register direct operand. Namespaces

- **bnssassembler**

Code/Assembler/Include/RegisterDirectParser.h File Reference

```
#include "OperandParser.h"
```

Classes

- class **bnssassembler::RegisterDirectParser**

Class representing the parser for the register direct operand.

Namespaces

- **bnssassembler**

Code/Assembler/Include/RegisterIndirect.h File Reference

```
#include "Operand.h"
#include "Register.h"
```

Classes

- class **bnssassembler::RegisterIndirect**

Class representing the register indirect operand. Namespaces

- **bnssassembler**

Code/Assembler/Include/RegisterIndirectOffset.h File Reference

```
#include "Operand.h"
#include "Register.h"
#include "MicroRiscExpression.h"
```

Classes

- class **bnssassembler::RegisterIndirectOffset**

Class representing the register indirect operand with offset.

Namespaces

- **bnssassembler**

Code/Assembler/Include/RegisterIndirectOffsetParser.h File Reference

```
#include "OperandParser.h"
```

Classes

- class **bnssassembler::RegisterIndirectOffsetParser**

Class representing the parser for the register indirect operand with offset. Namespaces

- **bnssassembler**

Code/Assembler/Include/RegisterIndirectParser.h File Reference

```
#include "OperandParser.h"
```

Classes

- class **bnssassembler::RegisterIndirectParser**

Class representing the parser for the register indirect operand.

Namespaces

- **bnssassembler**

Code/Assembler/Include/RegisterParser.h File Reference

```
#include "Register.h"
#include <unordered_map>
```

Classes

- class **bnssassembler::RegisterParser**
- *Utility class used for parsing registers.* struct **bnssassembler::RegisterParser::RegisterParserStaticData**

Namespaces

- **bnssassembler**

Code/Assembler/Include/RelocationRecord.h File Reference

```
#include <string>
```

Classes

- class **bnssassembler::RelocationRecord**

Class representing one relocation record. Namespaces

- **bnssassembler**

Code/Emulator/Include/RelocationRecord.h File Reference

```
#include <string>
#include <istream>
```

Classes

- class **bnssemulator::RelocationRecord**

Class representing one relocation record. Namespaces

- **bnssemulator**

Code/Assembler/Include/SecondPass.h File Reference

```
#include "SecondPassData.h"
#include "Token.h"
```

Classes

- class **bnssassembler::SecondPass**

Utility class executing the second pass. Namespaces

- **bnssassembler**

Code/Assembler/Include/SecondPassData.h File Reference

```
#include "FirstPassData.h"
#include "SectionType.h"
```

Classes

- class **bnssassembler::SecondPassData**

Class representing the data that will be updated during the second pass.

Namespaces

- **bnssassembler**

Code/Assembler/Include/SecondPassException.h File Reference

```
#include "AssemblerException.h"
```

Classes

- class **bnssassembler::SecondPassException**

Represents an exception that happened during the assembler second pass. Namespaces

- **bnssassembler**

Code/Assembler/Include/SectionData.h File Reference

```
#include "SectionType.h"
#include <functional>
#include <cstddef>
#include <utility>
#include <vector>
#include <list>
#include "RelocationRecord.h"
```

Classes

- class **bnssassembler::SectionData**
- *Class representing the data about one section.* struct **std::hash< bnssassembler::SectionData >**

Namespaces

- **bnssassembler**
- **std**

Code/Emulator/Include/SectionData.h File Reference

```
#include "SectionType.h"
#include "RelocationRecord.h"
#include <vector>
#include <istream>
```

Classes

- class **bnssemulator::SectionData**

Class representing the data about one section. Namespaces

- **bnssemulator**

Code/Assembler/Include/SectionStartLineParser.h File Reference

```
#include "LineParser.h"
```

Classes

- class **bnssassembler::SectionStartLineParser**

Class used for parsing section start definitions. Namespaces

- **bnssassembler**

Code/Assembler/Include/SectionStartToken.h File Reference

```
#include "Token.h"
#include "SectionType.h"
```

Classes

- class **bnssassembler::SectionStartToken**

Class representing the section start token. Namespaces

- **bnssassembler**

Code/Assembler/Include/SectionTable.h File Reference

```
#include <vector>
#include "SectionData.h"
#include <unordered_set>
```

Classes

- class **bnssassembler::SectionTable**

Class representing the table of sections. Namespaces

- **bnssassembler**

Code/Assembler/Include/SectionType.h File Reference

```
#include <cstdint>
#include <functional>
```

Classes

- struct **std::hash< bnssassembler::SectionType >**

Namespaces

- **bnssassembler**
- **std**

Enumerations

- enum **bnssassembler::SectionType** : int8_t { **bnssassembler::TEXT** = 0, **bnssassembler::DATA**, **bnssassembler::RODATA**, **bnssassembler::BSS** }*Enum representing the type of the section.*

Code/Emulator/Include/SectionType.h File Reference

```
#include <cstdint>
#include <functional>
```

Classes

- struct **std::hash< bnssemulator::SectionType >**

Namespaces

- **bnssemulator**
- **std**

Enumerations

- enum **bnssemulator::SectionType** : int8_t { **bnssemulator::TEXT** = 0, **bnssemulator::DATA**, **bnssemulator::RODATA**, **bnssemulator::BSS** }*Enum representing the type of the section.*

Code/Assembler/Include/SectionTypeParser.h File Reference

```
#include "SectionType.h"
#include <string>
#include <unordered_map>
```

Classes

- class **bnssassembler::SectionTypeParser**
- *Utility class representing the parser for the section types.* struct **bnssassembler::SectionTypeParser::SectionTypeParserData**

Namespaces

- **bnssassembler**

Code/Assembler/Include/StackInstructionParser.h File Reference

```
#include "InstructionParser.h"
```

Classes

- class **bnssassembler::StackInstructionParser**

Class representing the parser for stack instructions. Namespaces

- **bnssassembler**

Code/Assembler/Include/StoreInstructionParser.h File Reference

`#include "InstructionParser.h"`

Classes

- class **bnssassembler::StoreInstructionParser**

Class representing the parser for the store instruction. Namespaces

- **bnssassembler**

Code/Assembler/Include/StringHelper.h File Reference

```
#include <vector>
#include "CommonRegexes.h"
#include <regex>
#include "MessageException.h"
#include <iomanip>
#include <sstream>
```

Classes

- class **bnssassembler::StringHelper**

Utility class providing helper methods for std::string class. Namespaces

- **bnssassembler**

Code/Emulator/Include/StringHelper.h File Reference

```
#include <vector>
#include "CommonRegexes.h"
#include <regex>
#include "MessageException.h"
#include <iomanip>
#include <sstream>
```

Classes

- class **bnssemulator::StringHelper**

Utility class providing helper methods for std::string class. Namespaces

- **bnssemulator**

Code/Assembler/Include/SubtractOperation.h File Reference

```
#include "Operation.h"
#include "RelocationRecord.h"
#include <list>
```

Classes

- class **bnssassembler::SubtractOperation**

Class implementing the behaviour of the - operator in expressions.

Namespaces

- **bnssassembler**

Code/Assembler/Include/SubtractToken.h File Reference

`#include "OperationToken.h"`

Classes

- class **bnssassembler::SubtractToken**

Token ***class representing the - operation.*** Namespaces

- **bnssassembler**

Code/Assembler/Include/Symbol.h File Reference

```
#include "Expression.h"
```

Classes

- class **bnssassembler::Symbol**

Class representing a symbol inside an expression. Namespaces

- **bnssassembler**

Code/Assembler/Include/SymbolData.h File Reference

```
#include <string>
```

Classes

- class **bnssassembler::SymbolData**

Class representing data about one symbol. Namespaces

- **bnssassembler**

Code/Emulator/Include/SymbolData.h File Reference

```
#include <string>
#include <istream>
```

Classes

- class **bnssemulator::SymbolData**

Class representing data about one symbol. Namespaces

- **bnssemulator**

Code/Assembler/Include/SymbolDefinition.h File Reference

```
#include "MicroRiscExpression.h"
```

Classes

- class **bnssassembler::SymbolDefinition**
- *Class representing a symbol definition.* struct **std::hash< bnssassembler::SymbolDefinition >**

Namespaces

- **bnssassembler**
- **std**

Code/Assembler/Include/SymbolDefinitionLineParser.h File Reference

```
#include "LineParser.h"
```

Classes

- class **bnssassembler::SymbolDefinitionLineParser**

Class used for parsing symbol definitions. Namespaces

- **bnssassembler**

Code/Assembler/Include/SymbolDefinitionToken.h File Reference

```
#include "Token.h"
#include "MicroRiscExpression.h"
```

Classes

- class **bnssassembler::SymbolDefinitionToken**

Class representing the symbol definition token. Namespaces

- **bnssassembler**

Code/Assembler/Include/SymbolTable.h File Reference

```
#include "SymbolData.h"
#include <unordered_map>
```

Classes

- class **bnssassembler::SymbolTable**

Class representing the symbol table. Namespaces

- **bnssassembler**

Code/Assembler/Include/SymbolToken.h File Reference

`#include "ExpressionToken.h"`

Classes

- class **bnssassembler::SymbolToken**

Token ***class representing a math symbol.*** Namespaces

- **bnssassembler**

Code/Assembler/Include/Token.h File Reference

```
#include "FirstPassData.h"
#include "SecondPassData.h"
```

Classes

- class **bnssassembler::Token**

Class representing one token of the assembler source file. Namespaces

- **bnssassembler**

Code/Assembler/Include/UnconditionalJumpInstructionParser .h File Reference

`#include "InstructionParser.h"`

Classes

- class **bnssassembler::UndonditionalJumpInstructionParser**

Class representing the parser for the unconditional jump instructions.

Namespaces

- **bnssassembler**

Code/Assembler/Include/z85.h File Reference

```
#include <stddef.h>
```

Functions

- size_t **Z85_encode_with_padding** (const char *source, char *dest, size_t inputSize)

  *Encodes 'inputSize' bytes from 'source' into 'dest'. If 'inputSize' is not divisible by 4 with no remainder, 'source' is padded. Destination buffer must be already allocated. Use **Z85_encode_with_padding_bound()** to evaluate size of the destination buffer.*
- size_t **Z85_decode_with_padding** (const char *source, char *dest, size_t inputSize)

  *Decodes 'inputSize' printable symbols from 'source' into 'dest', encoded with **Z85_encode_with_padding()**. Destination buffer must be already allocated. Use **Z85_decode_with_padding_bound()** to evaluate size of the destination buffer.*
- size_t **Z85_encode_with_padding_bound** (size_t size)

  *Evaluates a size of output buffer needed to encode 'size' bytes into string of printable symbols using **Z85_encode_with_padding()**.*
- size_t **Z85_decode_with_padding_bound** (const char *source, size_t size)

  *Evaluates a size of output buffer needed to decode 'size' symbols into binary string using **Z85_decode_with_padding()**.*
- size_t **Z85_encode** (const char *source, char *dest, size_t inputSize)

  *Encodes 'inputSize' bytes from 'source' into 'dest'. If 'inputSize' is not divisible by 4 with no remainder, 0 is retured. Destination buffer must be already allocated. Use **Z85_encode_bound()** to evaluate size of the destination buffer.*
- size_t **Z85_decode** (const char *source, char *dest, size_t inputSize)

  *Decodes 'inputSize' printable symbols from 'source' into 'dest'. If 'inputSize' is not divisible by 5 with no remainder, 0 is returned. Destination buffer must be already allocated. Use **Z85_decode_bound()** to evaluate size of the destination buffer.*
- size_t **Z85_encode_bound** (size_t size)

  *Evaluates a size of output buffer needed to encode 'size' bytes into string of printable symbols using **Z85_encode()**.*
- size_t **Z85_decode_bound** (size_t size)

  *Evaluates a size of output buffer needed to decode 'size' symbols into binary string using **Z85_decode()**.*
- char * **Z85_encode_unsafe** (const char *source, const char *sourceEnd, char *dest)

  Encodes bytes from [source;sourceEnd) range into 'dest'. It can be used for implementation of your own padding scheme. Preconditions:
- char * **Z85_decode_unsafe** (const char *source, const char *sourceEnd, char *dest)

  Decodes symbols from [source;sourceEnd) range into 'dest'. It can be used for implementation of your own padding scheme. Preconditions:

Function Documentation

size_t Z85_decode (const char * *source*, char * *dest*, size_t *inputSize*)

Decodes 'inputSize' printable symbols from 'source' into 'dest'. If 'inputSize' is not divisible by 5 with no remainder, 0 is returned. Destination buffer must be already allocated. Use **Z85_decode_bound()** to evaluate size of the destination buffer.

Parameters:

source	in, input buffer (printable string to be decoded)

dest	out, destination buffer
inputSize	in, number of symbols to be decoded

Returns:

number of bytes written into 'dest' or 0 if something goes wrong

Definition at line 146 of file z85.cpp.

Referenced by z85::decode().

```
{
    if (!source || !dest || inputSize % 5)
    {
        assert(!"wrong source, destination or input size");
        return 0;
    }

    return Z85_decode_unsafe(source, source + inputSize, dest) - dest;
}
```

size_t Z85_decode_bound (size_t *size*)

Evaluates a size of output buffer needed to decode 'size' symbols into binary string using **Z85_decode()**.

Parameters:

size	in, number of symbols to be decoded

Returns:

minimal size of output buffer in bytes

Definition at line 130 of file z85.cpp.

Referenced by z85::decode().

```
{
    return size * 4 / 5;
}
```

char* Z85_decode_unsafe (const char * *source*, const char * *sourceEnd*, char * *dest*)

Decodes symbols from [source;sourceEnd) range into 'dest'. It can be used for implementation of your own padding scheme. Preconditions:

- (sourceEnd - source) % 5 == 0
- destination buffer must be already allocated

Parameters:

source	in, begin of input buffer
sourceEnd	in, end of input buffer (not included)
dest	out, output buffer

Returns:

a pointer immediately after last byte written into the 'dest'

Definition at line 100 of file z85.cpp.

```
{
    byte* src = (byte*)source;
    byte* end = (byte*)sourceEnd;
    byte* dst = (byte*)dest;
    uint32_t value;

    for (; src != end; src += 5, dst += 4)
    {
        value = base256[(src[0] - 32) & 127];
        value = value * 85 + base256[(src[1] - 32) & 127];
        value = value * 85 + base256[(src[2] - 32) & 127];
        value = value * 85 + base256[(src[3] - 32) & 127];
        value = value * 85 + base256[(src[4] - 32) & 127];

        // pack big-endian frame
        dst[0] = value >> 24;
        dst[1] = (byte)(value >> 16);
        dst[2] = (byte)(value >> 8);
        dst[3] = (byte)(value);
    }

    return (char*)dst;
}
```

size_t Z85_decode_with_padding (const char * *source*, char * *dest*, size_t *inputSize*)

Decodes 'inputSize' printable symbols from 'source' into 'dest', encoded with **Z85_encode_with_padding()**. Destination buffer must be already allocated. Use **Z85_decode_with_padding_bound()** to evaluate size of the destination buffer.

Parameters:

source	in, input buffer (printable string to be decoded)
dest	out, destination buffer
inputSize	in, number of symbols to be decoded

Returns:

number of bytes written into 'dest' or 0 if something goes wrong

Definition at line 203 of file z85.cpp.

Referenced by z85::decode_with_padding().

```
{
    char*       dst = dest;
    size_t      tailBytes;
    char        tailBuf[4] = { 0 };
    const char* end = source + inputSize;

    assert(source && dest && (inputSize == 0 || (inputSize - 1) % 5 == 0));

    // zero length string is not padded
    if (!source || !dest || inputSize == 0 || (inputSize - 1) % 5)
    {
        return 0;
    }

    tailBytes = (source++)[0] - '0'; // possible values: 1, 2, 3 or 4
    if (tailBytes - 1 > 3)
    {
        assert(!"wrong tail bytes count");
        return 0;
    }

    end -= 5;
    if (source != end)
    {
        // decode body
        dst = Z85_decode_unsafe(source, end, dst);
    }

    // decode last 5 bytes chunk
    Z85_decode_unsafe(end, end + 5, tailBuf);

    switch (tailBytes)
    {
    case 4:
        dst[3] = tailBuf[3];
    case 3:
        dst[2] = tailBuf[2];
    case 2:
        dst[1] = tailBuf[1];
    case 1:
        dst[0] = tailBuf[0];
    }

    return dst - dest + tailBytes;
}
```

size_t Z85_decode_with_padding_bound (const char * *source*, size_t *size*)

Evaluates a size of output buffer needed to decode 'size' symbols into binary string using **Z85_decode_with_padding()**.

Parameters:

source	in, input buffer (first symbol is read from 'source' to evaluate padding)
size	in, number of symbols to be decoded

Returns:

minimal size of output buffer in bytes

Definition at line 164 of file z85.cpp.

Referenced by z85::decode_with_padding().

```
{
    if (size == 0 || !source || (byte)(source[0] - '0' - 1) > 3) return 0;
    return Z85_decode_bound(size - 1) - 4 + (source[0] - '0');
}
```

size_t Z85_encode (const char * *source*, char * *dest*, size_t *inputSize*)

Encodes 'inputSize' bytes from 'source' into 'dest'. If 'inputSize' is not divisible by 4 with no remainder, 0 is retured. Destination buffer must be already allocated. Use **Z85_encode_bound()** to evaluate size of the destination buffer.

Parameters:

source	in, input buffer (binary string to be encoded)
dest	out, destination buffer
inputSize	in, number of bytes to be encoded

Returns:

number of printable symbols written into 'dest' or 0 if something goes wrong

Definition at line 135 of file z85.cpp.

Referenced by z85::encode().

```
{
    if (!source || !dest || inputSize % 4)
    {
        assert(!"wrong source, destination or input size");
        return 0;
    }

    return Z85_encode_unsafe(source, source + inputSize, dest) - dest;
}
```

size_t Z85_encode_bound (size_t *size*)

Evaluates a size of output buffer needed to encode 'size' bytes into string of printable symbols using **Z85_encode()**.

Parameters:

size	in, number of bytes to be encoded

Returns:

minimal size of output buffer in bytes

Definition at line 125 of file z85.cpp.

Referenced by z85::encode().

```
126 {
127     return size * 5 / 4;
128 }
```

char* Z85_encode_unsafe (const char * *source*, const char * *sourceEnd*, char * *dest*)

Encodes bytes from [source;sourceEnd) range into 'dest'. It can be used for implementation of your own padding scheme. Preconditions:

- (sourceEnd - source) % 4 == 0
- destination buffer must be already allocated

Parameters:

source	in, begin of input buffer
sourceEnd	in, end of input buffer (not included)
dest	out, output buffer

Returns:

a pointer immediately after last symbol written into the 'dest'

Definition at line 77 of file z85.cpp.

```
 78 {
 79     byte* src = (byte*)source;
 80     byte* end = (byte*)sourceEnd;
 81     byte* dst = (byte*)dest;
 82     uint32_t value;
 83     uint32_t value2;
 84 
 85     for (; src != end; src += 4, dst += 5)
 86     {
 87         // unpack big-endian frame
 88         value = (src[0] << 24) | (src[1] << 16) | (src[2] << 8) | src[3];
 89 
 90         value2 = DIV85(value); dst[4] = base85[value - value2 * 85]; value = value2;
 91         value2 = DIV85(value); dst[3] = base85[value - value2 * 85]; value = value2;
 92         value2 = DIV85(value); dst[2] = base85[value - value2 * 85]; value = value2;
 93         value2 = DIV85(value); dst[1] = base85[value - value2 * 85];
 94         dst[0] = base85[value2];
 95     }
 96 
 97     return (char*)dst;
 98 }
```

size_t Z85_encode_with_padding (const char * *source*, char * *dest*, size_t *inputSize*)

Encodes 'inputSize' bytes from 'source' into 'dest'. If 'inputSize' is not divisible by 4 with no remainder, 'source' is padded. Destination buffer must be already allocated. Use **Z85_encode_with_padding_bound()** to evaluate size of the destination buffer.

Parameters:

source	in, input buffer (binary string to be encoded)
dest	out, destination buffer
inputSize	in, number of bytes to be encoded

Returns:

number of printable symbols written into 'dest' or 0 if something goes wrong

Definition at line 170 of file z85.cpp.

Referenced by z85::encode_with_padding().

```
{
    size_t      tailBytes = inputSize % 4;
    char        tailBuf[4] = { 0 };
    char*       dst = dest;
    const char* end = source + inputSize - tailBytes;

    assert(source && dest);

    // zero length string is not padded
    if (!source || !dest || inputSize == 0)
    {
        return 0;
    }

    (dst++)[0] = (tailBytes == 0 ? '4' : '0' + (char)tailBytes); // write tail
bytes count
    dst = Z85_encode_unsafe(source, end, dst);                   // write body

                                                                  // write
tail
    switch (tailBytes)
    {
    case 3:
        tailBuf[2] = end[2];
    case 2:
        tailBuf[1] = end[1];
    case 1:
        tailBuf[0] = end[0];
        dst = Z85_encode_unsafe(tailBuf, tailBuf + 4, dst);
    }

    return dst - dest;
}
```

size_t Z85_encode_with_padding_bound (size_t *size*)

Evaluates a size of output buffer needed to encode 'size' bytes into string of printable symbols using **Z85_encode_with_padding()**.

Parameters:

size	in, number of bytes to be encoded

Returns:

minimal size of output buffer in bytes

Definition at line 157 of file z85.cpp.

Referenced by z85::encode_with_padding().

```
{
    if (size == 0) return 0;
    size = Z85_encode_bound(size);
    return size + (5 - size % 5) % 5 + 1;
}
```

Code/Emulator/Include/z85.h File Reference

```
#include <stddef.h>
```

Functions

- size_t **Z85_encode_with_padding** (const char *source, char *dest, size_t inputSize)

  *Encodes 'inputSize' bytes from 'source' into 'dest'. If 'inputSize' is not divisible by 4 with no remainder, 'source' is padded. Destination buffer must be already allocated. Use **Z85_encode_with_padding_bound()** to evaluate size of the destination buffer.*
- size_t **Z85_decode_with_padding** (const char *source, char *dest, size_t inputSize)

  *Decodes 'inputSize' printable symbols from 'source' into 'dest', encoded with **Z85_encode_with_padding()**. Destination buffer must be already allocated. Use **Z85_decode_with_padding_bound()** to evaluate size of the destination buffer.*
- size_t **Z85_encode_with_padding_bound** (size_t size)

  *Evaluates a size of output buffer needed to encode 'size' bytes into string of printable symbols using **Z85_encode_with_padding()**.*
- size_t **Z85_decode_with_padding_bound** (const char *source, size_t size)

  *Evaluates a size of output buffer needed to decode 'size' symbols into binary string using **Z85_decode_with_padding()**.*
- size_t **Z85_encode** (const char *source, char *dest, size_t inputSize)

  *Encodes 'inputSize' bytes from 'source' into 'dest'. If 'inputSize' is not divisible by 4 with no remainder, 0 is retured. Destination buffer must be already allocated. Use **Z85_encode_bound()** to evaluate size of the destination buffer.*
- size_t **Z85_decode** (const char *source, char *dest, size_t inputSize)

  *Decodes 'inputSize' printable symbols from 'source' into 'dest'. If 'inputSize' is not divisible by 5 with no remainder, 0 is returned. Destination buffer must be already allocated. Use **Z85_decode_bound()** to evaluate size of the destination buffer.*
- size_t **Z85_encode_bound** (size_t size)

  *Evaluates a size of output buffer needed to encode 'size' bytes into string of printable symbols using **Z85_encode()**.*
- size_t **Z85_decode_bound** (size_t size)

  *Evaluates a size of output buffer needed to decode 'size' symbols into binary string using **Z85_decode()**.*
- char * **Z85_encode_unsafe** (const char *source, const char *sourceEnd, char *dest)

  Encodes bytes from [source;sourceEnd) range into 'dest'. It can be used for implementation of your own padding scheme. Preconditions:
- char * **Z85_decode_unsafe** (const char *source, const char *sourceEnd, char *dest)

  Decodes symbols from [source;sourceEnd) range into 'dest'. It can be used for implementation of your own padding scheme. Preconditions:

Function Documentation

size_t Z85_decode (const char * *source*, char * *dest*, size_t *inputSize*)

Decodes 'inputSize' printable symbols from 'source' into 'dest'. If 'inputSize' is not divisible by 5 with no remainder, 0 is returned. Destination buffer must be already allocated. Use **Z85_decode_bound()** to evaluate size of the destination buffer.

Parameters:

source	in, input buffer (printable string to be decoded)

dest	out, destination buffer
inputSize	in, number of symbols to be decoded

Returns:

number of bytes written into 'dest' or 0 if something goes wrong

Definition at line 146 of file z85.cpp.

References Z85_decode_unsafe().

```
{
    if (!source || !dest || inputSize % 5)
    {
        assert(!"wrong source, destination or input size");
        return 0;
    }

    return Z85_decode_unsafe(source, source + inputSize, dest) - dest;
}
```

size_t Z85_decode_bound (size_t *size*)

Evaluates a size of output buffer needed to decode 'size' symbols into binary string using **Z85_decode()**.

Parameters:

size	in, number of symbols to be decoded

Returns:

minimal size of output buffer in bytes

Definition at line 130 of file z85.cpp.

Referenced by Z85_decode_with_padding_bound().

```
{
    return size * 4 / 5;
}
```

char* Z85_decode_unsafe (const char * *source*, const char * *sourceEnd*, char * *dest*)

Decodes symbols from [source;sourceEnd) range into 'dest'. It can be used for implementation of your own padding scheme. Preconditions:

- (sourceEnd - source) % 5 == 0
- destination buffer must be already allocated

Parameters:

source	in, begin of input buffer
sourceEnd	in, end of input buffer (not included)
dest	out, output buffer

Returns:

a pointer immediately after last byte written into the 'dest'

Definition at line 100 of file z85.cpp.

References base256, and cxxopts::value().

Referenced by Z85_decode(), and Z85_decode_with_padding().

```
{
    byte* src = (byte*)source;
    byte* end = (byte*)sourceEnd;
    byte* dst = (byte*)dest;
    uint32_t value;

    for (; src != end; src += 5, dst += 4)
    {
        value = base256[(src[0] - 32) & 127];
        value = value * 85 + base256[(src[1] - 32) & 127];
        value = value * 85 + base256[(src[2] - 32) & 127];
        value = value * 85 + base256[(src[3] - 32) & 127];
        value = value * 85 + base256[(src[4] - 32) & 127];

        // pack big-endian frame
        dst[0] = value >> 24;
        dst[1] = (byte)(value >> 16);
        dst[2] = (byte)(value >> 8);
        dst[3] = (byte)(value);
    }

    return (char*)dst;
}
```

size_t Z85_decode_with_padding (const char * *source*, char * *dest*, size_t *inputSize*)

Decodes 'inputSize' printable symbols from 'source' into 'dest', encoded with **Z85_encode_with_padding()**. Destination buffer must be already allocated. Use **Z85_decode_with_padding_bound()** to evaluate size of the destination buffer.

Parameters:

source	in, input buffer (printable string to be decoded)
dest	out, destination buffer
inputSize	in, number of symbols to be decoded

Returns:

number of bytes written into 'dest' or 0 if something goes wrong

Definition at line 203 of file z85.cpp.

References Z85_decode_unsafe().

```
{
    char*       dst = dest;
    size_t      tailBytes;
    char        tailBuf[4] = { 0 };
    const char* end = source + inputSize;

    assert(source && dest && (inputSize == 0 || (inputSize - 1) % 5 == 0));

    // zero length string is not padded
    if (!source || !dest || inputSize == 0 || (inputSize - 1) % 5)
    {
        return 0;
    }

    tailBytes = (source++)[0] - '0'; // possible values: 1, 2, 3 or 4
    if (tailBytes - 1 > 3)
    {
        assert(!"wrong tail bytes count");
        return 0;
    }

    end -= 5;
    if (source != end)
    {
        // decode body
        dst = Z85_decode_unsafe(source, end, dst);
    }

    // decode last 5 bytes chunk
    Z85_decode_unsafe(end, end + 5, tailBuf);

    switch (tailBytes)
    {
    case 4:
        dst[3] = tailBuf[3];
    case 3:
        dst[2] = tailBuf[2];
    case 2:
        dst[1] = tailBuf[1];
    case 1:
        dst[0] = tailBuf[0];
    }

    return dst - dest + tailBytes;
}
```

size_t Z85_decode_with_padding_bound (const char * *source*, size_t *size*)

Evaluates a size of output buffer needed to decode 'size' symbols into binary string using **Z85_decode_with_padding()**.

Parameters:

source	in, input buffer (first symbol is read from 'source' to evaluate padding)
size	in, number of symbols to be decoded

Returns:

minimal size of output buffer in bytes

Definition at line 164 of file z85.cpp.

References Z85_decode_bound().

```
{
    if (size == 0 || !source || (byte)(source[0] - '0' - 1) > 3) return 0;
    return Z85_decode_bound(size - 1) - 4 + (source[0] - '0');
}
```

size_t Z85_encode (const char * *source*, char * *dest*, size_t *inputSize*)

Encodes 'inputSize' bytes from 'source' into 'dest'. If 'inputSize' is not divisible by 4 with no remainder, 0 is retured. Destination buffer must be already allocated. Use **Z85_encode_bound()** to evaluate size of the destination buffer.

Parameters:

source	in, input buffer (binary string to be encoded)
dest	out, destination buffer
inputSize	in, number of bytes to be encoded

Returns:

number of printable symbols written into 'dest' or 0 if something goes wrong

Definition at line 135 of file z85.cpp.

References Z85_encode_unsafe().

```
{
    if (!source || !dest || inputSize % 4)
    {
        assert(!"wrong source, destination or input size");
        return 0;
    }

    return Z85_encode_unsafe(source, source + inputSize, dest) - dest;
}
```

size_t Z85_encode_bound (size_t *size*)

Evaluates a size of output buffer needed to encode 'size' bytes into string of printable symbols using **Z85_encode()**.

Parameters:

size	in, number of bytes to be encoded

Returns:

minimal size of output buffer in bytes

Definition at line 125 of file z85.cpp.

Referenced by Z85_encode_with_padding_bound().

```
{
    return size * 5 / 4;
}
```

char* Z85_encode_unsafe (const char * *source*, const char * *sourceEnd*, char * *dest*)

Encodes bytes from [source;sourceEnd) range into 'dest'. It can be used for implementation of your own padding scheme. Preconditions:

- (sourceEnd - source) % 4 == 0
- destination buffer must be already allocated

Parameters:

source	in, begin of input buffer
sourceEnd	in, end of input buffer (not included)
dest	out, output buffer

Returns:

a pointer immediately after last symbol written into the 'dest'

Definition at line 77 of file z85.cpp.

References base85, DIV85, and cxxopts::value().

Referenced by Z85_encode(), and Z85_encode_with_padding().

```
{
    byte* src = (byte*)source;
    byte* end = (byte*)sourceEnd;
    byte* dst = (byte*)dest;
    uint32_t value;
    uint32_t value2;

    for (; src != end; src += 4, dst += 5)
    {
        // unpack big-endian frame
        value = (src[0] << 24) | (src[1] << 16) | (src[2] << 8) | src[3];

        value2 = DIV85(value); dst[4] = base85[value - value2 * 85]; value = value2;
        value2 = DIV85(value); dst[3] = base85[value - value2 * 85]; value = value2;
        value2 = DIV85(value); dst[2] = base85[value - value2 * 85]; value = value2;
        value2 = DIV85(value); dst[1] = base85[value - value2 * 85];
        dst[0] = base85[value2];
    }

    return (char*)dst;
}
```

size_t Z85_encode_with_padding (const char * *source*, char * *dest*, size_t *inputSize*)

Encodes 'inputSize' bytes from 'source' into 'dest'. If 'inputSize' is not divisible by 4 with no remainder, 'source' is padded. Destination buffer must be already allocated. Use **Z85_encode_with_padding_bound()** to evaluate size of the destination buffer.

Parameters:

source	in, input buffer (binary string to be encoded)
dest	out, destination buffer
inputSize	in, number of bytes to be encoded

Returns:

number of printable symbols written into 'dest' or 0 if something goes wrong

Definition at line 170 of file z85.cpp.

References Z85_encode_unsafe().

```
{
    size_t      tailBytes = inputSize % 4;
    char        tailBuf[4] = { 0 };
    char*       dst = dest;
    const char* end = source + inputSize - tailBytes;

    assert(source && dest);

    // zero length string is not padded
    if (!source || !dest || inputSize == 0)
    {
        return 0;
    }

    (dst++)[0] = (tailBytes == 0 ? '4' : '0' + (char)tailBytes); // write tail bytes count
    dst = Z85_encode_unsafe(source, end, dst);                  // write body

                                                                 // write tail
    switch (tailBytes)
    {
    case 3:
        tailBuf[2] = end[2];
    case 2:
        tailBuf[1] = end[1];
    case 1:
        tailBuf[0] = end[0];
        dst = Z85_encode_unsafe(tailBuf, tailBuf + 4, dst);
    }

    return dst - dest;
}
```

size_t Z85_encode_with_padding_bound (size_t *size*)

Evaluates a size of output buffer needed to encode 'size' bytes into string of printable symbols using **Z85_encode_with_padding()**.

Parameters:

size	in, number of bytes to be encoded

Returns:

minimal size of output buffer in bytes

Definition at line 157 of file z85.cpp.

References Z85_encode_bound().

```
{
    if (size == 0) return 0;
    size = Z85_encode_bound(size);
    return size + (5 - size % 5) % 5 + 1;
}
```

Code/Assembler/Include/z85_cpp.h File Reference

```
#include <stddef.h>
#include <string>
```

Namespaces

- **z85**

Macros

- #define **Z85_DELETE_FUNCTION_DEFINITION**

Functions

- std::string **z85::encode_with_padding** (const char *source, size_t inputSize)
  Encodes 'inputSize' bytes from 'source'. If 'inputSize' is not divisible by 4 with no remainder, 'source' is padded.
- std::string **z85::encode_with_padding** (const std::string &source)
- std::string **z85::encode_with_padding** (const char *) **Z85_DELETE_FUNCTION_DEFINITION**
- std::string **z85::decode_with_padding** (const char *source, size_t inputSize)
  *Decodes 'inputSize' printable symbols from 'source', encoded with **encode_with_padding()**.*
- std::string **z85::decode_with_padding** (const std::string &source)
- std::string **z85::decode_with_padding** (const char *) **Z85_DELETE_FUNCTION_DEFINITION**
- std::string **z85::encode** (const char *source, size_t inputSize)
  Encodes 'inputSize' bytes from 'source'. If 'inputSize' is not divisible by 4 with no remainder, empty string is retured.
- std::string **z85::encode** (const std::string &source)
- std::string **z85::encode** (const char *) **Z85_DELETE_FUNCTION_DEFINITION**
- std::string **z85::decode** (const char *source, size_t inputSize)
  Decodes 'inputSize' printable symbols from 'source'. If 'inputSize' is not divisible by 5 with no remainder, empty string is returned.
- std::string **z85::decode** (const std::string &source)
- std::string **z85::decode** (const char *) **Z85_DELETE_FUNCTION_DEFINITION**

Macro Definition Documentation

#define Z85_DELETE_FUNCTION_DEFINITION

Definition at line 40 of file z85_cpp.h.

Code/Emulator/Include/z85_cpp.h File Reference

```
#include <stddef.h>
#include <string>
```

Namespaces

- **z85**

Macros

- #define **Z85_DELETE_FUNCTION_DEFINITION**

Functions

- std::string **z85::encode_with_padding** (const char *source, size_t inputSize)
  Encodes 'inputSize' bytes from 'source'. If 'inputSize' is not divisible by 4 with no remainder, 'source' is padded.
- std::string **z85::encode_with_padding** (const std::string &source)
- std::string **z85::encode_with_padding** (const char *) **Z85_DELETE_FUNCTION_DEFINITION**
- std::string **z85::decode_with_padding** (const char *source, size_t inputSize)
  *Decodes 'inputSize' printable symbols from 'source', encoded with **encode_with_padding()**.*
- std::string **z85::decode_with_padding** (const std::string &source)
- std::string **z85::decode_with_padding** (const char *) **Z85_DELETE_FUNCTION_DEFINITION**
- std::string **z85::encode** (const char *source, size_t inputSize)
  Encodes 'inputSize' bytes from 'source'. If 'inputSize' is not divisible by 4 with no remainder, empty string is retured.
- std::string **z85::encode** (const std::string &source)
- std::string **z85::encode** (const char *) **Z85_DELETE_FUNCTION_DEFINITION**
- std::string **z85::decode** (const char *source, size_t inputSize)
  Decodes 'inputSize' printable symbols from 'source'. If 'inputSize' is not divisible by 5 with no remainder, empty string is returned.
- std::string **z85::decode** (const std::string &source)
- std::string **z85::decode** (const char *) **Z85_DELETE_FUNCTION_DEFINITION**

Macro Definition Documentation

#define Z85_DELETE_FUNCTION_DEFINITION

Definition at line 40 of file z85_cpp.h.

Code/Assembler/Source/AddOperation.cpp File Reference

```
#include "AddOperation.h"
#include "SubtractOperation.h"
#include "RelocationRecord.h"
```

Namespaces

- **bnssassembler**

Functions

- static void **bnssassembler::split** (std::list< RelocationRecord > &original, std::list< RelocationRecord > &left, std::list< RelocationRecord > &right)

Code/Assembler/Source/AddToken.cpp File Reference

```
#include "AddToken.h"
#include "AddOperation.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/AluInstructionParser.cpp File Reference

```
#include "AluInstructionParser.h"
#include "RegisterDirectParser.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/AssemblerException.cpp File Reference

```
#include "AssemblerException.h"
#include "StringHelper.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/ClosingBraceToken.cpp File Reference

```
#include "ClosingBraceToken.h"
#include "MessageException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/CommandLineHelper.cpp File Reference

```
#include "CommandLineHelper.h"
#include <iostream>
#include "cxxopts.h"
```

Namespaces

- **bnssassembler**

Code/Emulator/Source/CommandLineHelper.cpp File Reference

```
#include "CommandLineHelper.h"
#include <iostream>
#include "cxxopts.h"
```

Namespaces

- **bnssemulator**

Code/Assembler/Source/ConditionalJumpInstructionParser.cpp File Reference

```
#include "ConditionalJumpInstructionParser.h"
#include "RegisterDirectParser.h"
#include "MemoryDirectParser.h"
#include "RegisterIndirectParser.h"
#include "RegisterIndirectOffsetParser.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/Data.cpp File Reference

```
#include "Data.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/DataDefinitionLineParser.cpp File Reference

```
#include "DataDefinitionLineParser.h"
#include <regex>
#include "CommonRegexes.h"
#include "Data.h"
#include "InvalidDataDefinitionException.h"
#include "DataDefinitionToken.h"
#include "InvalidDataTypeException.h"
#include "DataTypeParser.h"
#include "ExpressionBuilder.h"
```

Namespaces

- **bnssassembler**

Functions

- Data **bnssassembler::parseData** (std::string str)
  Parses the data from the string.

Code/Assembler/Source/DataDefinitionToken.cpp File Reference

```
#include "DataDefinitionToken.h"
#include "DataTypeParser.h"
#include "SecondPassData.h"
#include "MessageException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/DataTypeParser.cpp File Reference

```
#include "DataTypeParser.h"
#include <algorithm>
#include <locale>
#include "InvalidDataTypeException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/DivideOperation.cpp File Reference

```
#include "DivideOperation.h"
#include "DivisionByZeroException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/DivideToken.cpp File Reference

```
#include "DivideToken.h"
#include "DivideOperation.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/DivisionByZeroException.cpp File Reference

```
#include "DivisionByZeroException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/Expression.cpp File Reference

```
#include "Expression.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/ExpressionBuilder.cpp File Reference

```
#include "ExpressionBuilder.h"
#include <list>
#include "InvalidExpressionException.h"
#include <regex>
#include "CommonRegexes.h"
#include "ExpressionToken.h"
#include "ExpressionTokenFactory.h"
```

Namespaces

- **bnssassembler**

Functions

- static void **bnssassembler::fixUnaryMinusStart** (std::string &infix_expression, std::regex token_extractor)
  Fixes the expression that starts with an unary minus sign.
- static std::list< std::shared_ptr< ExpressionToken > > **bnssassembler::infixToPostfix** (std::string infix_expression)
  Builds a postfix expression from the infix string.
- static std::shared_ptr< Expression > **bnssassembler::postfixToTree** (const std::list< std::shared_ptr< ExpressionToken >> &postfix_expression)
  Builds a tree from the postfix expression.

Code/Assembler/Source/ExpressionTokenFactory.cpp File Reference

```
#include "ExpressionTokenFactory.h"
#include <regex>
#include "CommonRegexes.h"
#include "MessageException.h"
#include "LiteralToken.h"
#include "SymbolToken.h"
#include "AddToken.h"
#include "SubtractToken.h"
#include "MultiplyToken.h"
#include "DivideToken.h"
#include "OpeningBraceToken.h"
#include "ClosingBraceToken.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/FileReader.cpp File Reference

```
#include "FileReader.h"
#include "StringHelper.h"
```

Namespaces

- **bnssassembler**

Code/Emulator/Source/FileReader.cpp File Reference

```
#include "FileReader.h"
#include "StringHelper.h"
#include "z85_cpp.h"
#include <sstream>
```

Namespaces

- **bnssemulator**

Code/Assembler/Source/FileWriter.cpp File Reference

```
#include "FileWriter.h"
#include "fstream"
#include "SecondPassData.h"
#include <sstream>
#include "z85_cpp.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/FirstPass.cpp File Reference

```
#include "FirstPass.h"
#include "MessageException.h"
#include "FirstPassException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/FirstPassData.cpp File Reference

```
#include "FirstPassData.h"
#include "MessageException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/FirstPassException.cpp File Reference

```
#include "FirstPassException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/GlobalSymbolsLineParser.cpp File Reference

```
#include "GlobalSymbolsLineParser.h"
#include <regex>
#include "CommonRegexes.h"
#include "StringHelper.h"
#include "GlobalSymbolToken.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/GlobalSymbolToken.cpp File Reference

```
#include "GlobalSymbolToken.h"
#include "SecondPassData.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/Immediate.cpp File Reference

```
#include "Immediate.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/ImmediateParser.cpp File Reference

```
#include "ImmediateParser.h"
#include <regex>
#include "CommonRegexes.h"
#include "ExpressionBuilder.h"
#include "Immediate.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/IncorrectLabelException.cpp File Reference

```
#include "IncorrectLabelException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/InstructionCodeParser.cpp File Reference

```
#include "InstructionCodeParser.h"
#include <locale>
#include <algorithm>
#include "MessageException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/InstructionLineParser.cpp File Reference

```
#include "InstructionLineParser.h"
#include <regex>
#include "InstructionCodeParser.h"
#include "InterruptInstructionParser.h"
#include "NoOperandInstructionParser.h"
#include "ConditionalJumpInstructionParser.h"
#include "UnconditionalJumpInstructionParser.h"
#include "StackInstructionParser.h"
#include "AluInstructionParser.h"
#include "NotInstructionParser.h"
#include "LoadInstructionParser.h"
#include "StoreInstructionParser.h"
#include "MessageException.h"
#include <unordered_map>
```

Namespaces

- **bnssassembler**

Functions

- static void **bnssassembler::loadStoreFixup** (std::string &instruction, OperandType &type)
  Hack to fix the load and store instructions which can have various operands.

Code/Assembler/Source/InstructionParser.cpp File Reference

```
#include "InstructionParser.h"
#include <regex>
#include "MessageException.h"
#include "CommonRegexes.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/InstructionToken.cpp File Reference

```
#include "InstructionToken.h"
#include "InstructionBitFieldUnion.h"
#include "SecondPassData.h"
#include "Register.h"
#include "MessageException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/InterruptInstructionParser.cpp File Reference

```
#include "InterruptInstructionParser.h"
#include "RegisterDirectParser.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/InvalidDataDefinitionException.cpp File Reference

```
#include "InvalidDataDefinitionException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/InvalidDataTypeException.cpp File Reference

```
#include "InvalidDataTypeException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/InvalidExpressionException.cpp File Reference

```
#include "InvalidExpressionException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/LabelToken.cpp File Reference

```
#include "LabelToken.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/LineParser.cpp File Reference

```
#include "LineParser.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/Literal.cpp File Reference

`#include "Literal.h"`

Namespaces

- **bnssassembler**

Code/Assembler/Source/LiteralToken.cpp File Reference

```
#include "LiteralToken.h"
#include "StringHelper.h"
#include "Literal.h"
#include <climits>
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/LoadInstructionParser.cpp File Reference

```
#include "LoadInstructionParser.h"
#include "RegisterDirectParser.h"
#include "MemoryDirectParser.h"
#include "RegisterIndirectOffsetParser.h"
#include "RegisterIndirectParser.h"
#include "ImmediateParser.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/Main.cpp File Reference

```
#include <iostream>
#include "FileReader.h"
#include <string>
#include "StringHelper.h"
#include "MicroRiscParser.h"
#include "AssemblerException.h"
#include "FirstPass.h"
#include "SecondPass.h"
#include "FileWriter.h"
#include "CommandLineHelper.h"
```

Functions

- int **main** (int argc, char *argv[])

Function Documentation

int main (int *argc*, char * *argv*[])

Definition at line 12 of file Main.cpp.

References bnssassembler::FirstPass::execute(), bnssassembler::SecondPass::execute(), bnssassembler::MicroRiscParser::instance(), bnssassembler::AssemblerException::message(), bnssassembler::CommandLineHelper::parse(), bnssassembler::Parser::parse(), bnssassembler::FileReader::readAllLines(), and bnssassembler::FileWriter::write().

```
   12                                  {
   13     try {
   14         auto in_out = bnssassembler::CommandLineHelper::parse(argc, argv);
   15         auto lines = bnssassembler::FileReader::readAllLines(in_out.first);
   16         auto parsed = bnssassembler::MicroRiscParser::instance().parse(lines);
   17         auto first = bnssassembler::FirstPass::execute(parsed);
   18         auto second = bnssassembler::SecondPass::execute(parsed,
std::move(first));
   19         bnssassembler::FileWriter::write(in_out.second, second);
   20     }
   21     catch (bnssassembler::AssemblerException &e) {
   22         std::cerr << e.message() << std::endl;
   23         return EXIT_FAILURE;
   24     }
   25     catch (std::exception &e) {
   26         std::cerr << e.what() << std::endl;
   27         return EXIT_FAILURE;
   28     }
   29     catch (...) {
   30         std::cerr << "Unknown error" << std::endl;
   31         return EXIT_FAILURE;
   32     }
   33 
   34     return EXIT_SUCCESS;
   35 }
```

Code/Emulator/Source/Main.cpp File Reference

```
#include <iostream>
#include "CommandLineHelper.h"
#include "FileReader.h"
#include "AddressSpace.h"
#include "Context.h"
#include "Processor.h"
```

Functions

- int **main** (int argc, char *argv[])

Function Documentation

int main (int *argc*, char * *argv*[])

Definition at line 8 of file Main.cpp.

References bnssemulator::Processor::executeProgram(), bnssemulator::FileReader::parse(), and bnssemulator::CommandLineHelper::parse().

```
                                  {

    try {
        auto input = bnssemulator::CommandLineHelper::parse(argc, argv);
        auto data = bnssemulator::FileReader::parse(input);
        bnssemulator::Context context(std::move(data));
        bnssemulator::Processor::executeProgram(context);
    }
    catch (const std::exception &exception) {
        std::cerr << exception.what() << std::endl;
        return EXIT_FAILURE;
    }
    catch (...) {
        std::cerr << "Unknown exception" << std::endl;
        return EXIT_FAILURE;
    }

    return EXIT_SUCCESS;
}
```

Code/Assembler/Source/MemoryDirect.cpp File Reference

```
#include "MemoryDirect.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/MemoryDirectParser.cpp File Reference

```
#include "MemoryDirectParser.h"
#include <regex>
#include "CommonRegexes.h"
#include "ExpressionBuilder.h"
#include "MemoryDirect.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/MessageException.cpp File Reference

```
#include "MessageException.h"
```

Namespaces

- **bnssassembler**

Code/Emulator/Source/MessageException.cpp File Reference

```
#include "MessageException.h"
```

Namespaces

- **bnssemulator**

Code/Assembler/Source/MicroRiscExpression.cpp File Reference

```
#include "MicroRiscExpression.h"
#include "MessageException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/MicroRiscParser.cpp File Reference

```
#include "MicroRiscParser.h"
#include "InstructionLineParser.h"
#include "SectionStartLineParser.h"
#include "OrgDirectiveLineParser.h"
#include "SymbolDefinitionLineParser.h"
#include "DataDefinitionLineParser.h"
#include "GlobalSymbolsLineParser.h"
#include <regex>
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/MultiplyOperation.cpp File Reference

```
#include "MultiplyOperation.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/MultiplyToken.cpp File Reference

```
#include "MultiplyToken.h"
#include "MultiplyOperation.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/NonExistingSymbolException.cpp File Reference

`#include "NonExistingSymbolException.h"`

Namespaces

- **bnssassembler**

Code/Assembler/Source/NotInstructionParser.cpp File Reference

```
#include "NotInstructionParser.h"
#include "RegisterDirectParser.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/OpeningBraceToken.cpp File Reference

```
#include "OpeningBraceToken.h"
#include "MessageException.h"
#include <climits>
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/Operand.cpp File Reference

```
#include "Operand.h"
#include <unordered_set>
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/OperandParser.cpp File Reference

```
#include "OperandParser.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/Operation.cpp File Reference

```
#include "Operation.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/OperationToken.cpp File Reference

```
#include "OperationToken.h"
#include "MessageException.h"
#include "ExpressionBuilder.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/OrgDirectiveLineParser.cpp File Reference

```
#include "OrgDirectiveLineParser.h"
#include <regex>
#include "CommonRegexes.h"
#include "ExpressionBuilder.h"
#include "OrgDirectiveToken.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/OrgDirectiveToken.cpp File Reference

```
#include "OrgDirectiveToken.h"
#include "MessageException.h"
#include "SecondPassData.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/Parser.cpp File Reference

```
#include "Parser.h"
#include <regex>
#include "StringHelper.h"
#include "LabelToken.h"
#include <string>
#include <iostream>
#include "IncorrectLabelException.h"
#include "CommonRegexes.h"
#include "ParserException.h"
```

Namespaces

- **bnssassembler**

Functions

- static void **bnssassembler::stripComment** (std::string &line, std::vector< std::string > one_line_comment_delimiters)
  Strips the comment from one line of the file.
- static std::string **bnssassembler::extractLabel** (std::string &line, std::vector< std::string > label_delimiters)
  Extracts the label (if any) from the line.

Code/Assembler/Source/ParserException.cpp File Reference

```
#include "ParserException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/RegisterDirect.cpp File Reference

```
#include "RegisterDirect.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/RegisterDirectParser.cpp File Reference

```
#include "RegisterDirectParser.h"
#include "RegisterParser.h"
#include "RegisterDirect.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/RegisterIndirect.cpp File Reference

```
#include "RegisterIndirect.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/RegisterIndirectOffset.cpp File Reference

```
#include "RegisterIndirectOffset.h"
#include "MessageException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/RegisterIndirectOffsetParser.cpp File Reference

```
#include "RegisterIndirectOffsetParser.h"
#include <regex>
#include "RegisterParser.h"
#include "ExpressionBuilder.h"
#include "RegisterIndirectOffset.h"
#include "CommonRegexes.h"
```

Namespaces

- **bnssassembler**

Functions

- std::shared_ptr< Operand > **bnssassembler::parsePcrel** (std::string str)
  Parses the input as a PC relative address.

Code/Assembler/Source/RegisterIndirectParser.cpp File Reference

```
#include "RegisterIndirectParser.h"
#include <regex>
#include "RegisterParser.h"
#include "RegisterIndirect.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/RegisterParser.cpp File Reference

```
#include "RegisterParser.h"
#include "MessageException.h"
#include <algorithm>
#include <regex>
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/RelocationRecord.cpp File Reference

```
#include "RelocationRecord.h"
#include <iostream>
#include <iomanip>
#include "PrintHelpers.h"
```

Namespaces

- **bnssassembler**

Functions

- std::ostream & **bnssassembler::operator<<** (std::ostream &os, const RelocationRecord &record)
- bool **bnssassembler::operator==** (const RelocationRecord &lhs, const RelocationRecord &rhs)
- bool **bnssassembler::operator!=** (const RelocationRecord &lhs, const RelocationRecord &rhs)

Code/Emulator/Source/RelocationRecord.cpp File Reference

```
#include "RelocationRecord.h"
```

Namespaces

- **bnssemulator**

Functions

- std::istream & **bnssemulator::operator**>> (std::istream &is, RelocationRecord &data)

Code/Assembler/Source/SecondPass.cpp File Reference

```
#include "SecondPass.h"
#include "MessageException.h"
#include "SecondPassException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/SecondPassData.cpp File Reference

```
#include "SecondPassData.h"
#include "StringHelper.h"
#include <iostream>
#include "PrintHelpers.h"
```

Namespaces

- **bnssassembler**

Functions

- std::ostream & **bnssassembler::operator<<** (std::ostream &os, const SecondPassData &data)

Code/Assembler/Source/SecondPassException.cpp File Reference

```
#include "SecondPassException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/SectionData.cpp File Reference

```
#include "SectionData.h"
#include "StringHelper.h"
#include <iostream>
#include "PrintHelpers.h"
```

Namespaces

- **bnssassembler**

Functions

- bool **bnssassembler::operator==** (const SectionData &lhs, const SectionData &rhs) noexcept
- bool **bnssassembler::operator!=** (const SectionData &lhs, const SectionData &rhs) noexcept
- bool **bnssassembler::operator<** (const SectionData &lhs, const SectionData &rhs) noexcept
- bool **bnssassembler::operator>** (const SectionData &lhs, const SectionData &rhs) noexcept
- bool **bnssassembler::operator<=** (const SectionData &lhs, const SectionData &rhs) noexcept
- bool **bnssassembler::operator>=** (const SectionData &lhs, const SectionData &rhs) noexcept
- static std::string **bnssassembler::name** (SectionType type, bool indexed, size_t index)
- static void **bnssassembler::writeDescription** (SectionType type, bool indexed, size_t index, bool org_valid, **uint32_t** org_address, size_t size)
- std::ostream & **bnssassembler::operator<<** (std::ostream &os, const SectionData &data)

Code/Emulator/Source/SectionData.cpp File Reference

```
#include "SectionData.h"
#include <istream>
```

Namespaces

- **bnssemulator**

Functions

- std::istream & **bnssemulator::operator**>> (std::istream &is, SectionData &data)

Code/Assembler/Source/SectionStartLineParser.cpp File Reference

```
#include "SectionStartLineParser.h"
#include <regex>
#include "SectionTypeParser.h"
#include "StringHelper.h"
#include "SectionStartToken.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/SectionStartToken.cpp File Reference

```
#include "SectionStartToken.h"
#include "SecondPassData.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/SectionTable.cpp File Reference

```
#include "SectionTable.h"
#include "MessageException.h"
#include "SectionTypeParser.h"
#include <iostream>
#include "PrintHelpers.h"
#include <iomanip>
```

Namespaces

- **bnssassembler**

Functions

- std::ostream & **bnssassembler::operator<<** (std::ostream &os, const SectionTable &section_table)

Code/Assembler/Source/SectionTypeParser.cpp File Reference

```
#include "SectionTypeParser.h"
#include "MessageException.h"
#include <algorithm>
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/StackInstructionParser.cpp File Reference

```
#include "StackInstructionParser.h"
#include "RegisterDirectParser.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/StoreInstructionParser.cpp File Reference

```
#include "StoreInstructionParser.h"
#include "MemoryDirectParser.h"
#include "RegisterIndirectOffsetParser.h"
#include "RegisterIndirectParser.h"
#include "RegisterDirectParser.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/StringHelper.cpp File Reference

```
#include "StringHelper.h"
#include <fstream>
#include <sstream>
#include <iterator>
#include <stdexcept>
#include <iomanip>
```

Namespaces

- **bnssassembler**

Code/Emulator/Source/StringHelper.cpp File Reference

```
#include "StringHelper.h"
#include <fstream>
#include <sstream>
#include <iterator>
#include <stdexcept>
#include <iomanip>
```

Namespaces

- **bnssemulator**

Code/Assembler/Source/SubtractOperation.cpp File Reference

```
#include "SubtractOperation.h"
#include "StringHelper.h"
```

Namespaces

- **bnssassembler**

Functions

- static void **bnssassembler::generateMaps** (const std::list< RelocationRecord > &source, std::unordered_map< size_t, std::pair< RelocationRecord, size_t >> &sections, std::unordered_map< std::string, std::pair< RelocationRecord, size_t >> &symbols)
- static void **bnssassembler::exchange** (std::list< RelocationRecord > &left, std::list< RelocationRecord > &right)

Code/Assembler/Source/SubtractToken.cpp File Reference

```
#include "SubtractToken.h"
#include "SubtractOperation.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/Symbol.cpp File Reference

```
#include "Symbol.h"
#include "NonExistingSymbolException.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/SymbolData.cpp File Reference

```
#include "SymbolData.h"
#include <iostream>
#include "PrintHelpers.h"
#include <iomanip>
```

Namespaces

- **bnssassembler**

Functions

- std::ostream & **bnssassembler::operator<<** (std::ostream &os, const SymbolData &data)

Code/Emulator/Source/SymbolData.cpp File Reference

`#include "SymbolData.h"`

Namespaces

- **bnssemulator**

Functions

- std::istream & **bnssemulator::operator**>> (std::istream &is, SymbolData &data)

Code/Assembler/Source/SymbolDefinition.cpp File Reference

```
#include "SymbolDefinition.h"
```

Namespaces

- **bnssassembler**

Functions

- bool **bnssassembler::operator==** (const SymbolDefinition &lhs, const SymbolDefinition &rhs)
- bool **bnssassembler::operator!=** (const SymbolDefinition &lhs, const SymbolDefinition &rhs)
- bool **bnssassembler::operator<** (const SymbolDefinition &lhs, const SymbolDefinition &rhs)
- bool **bnssassembler::operator>** (const SymbolDefinition &lhs, const SymbolDefinition &rhs)
- bool **bnssassembler::operator<=** (const SymbolDefinition &lhs, const SymbolDefinition &rhs)
- bool **bnssassembler::operator>=** (const SymbolDefinition &lhs, const SymbolDefinition &rhs)

Code/Assembler/Source/SymbolDefinitionLineParser.cpp File Reference

```
#include "SymbolDefinitionLineParser.h"
#include <regex>
#include "CommonRegexes.h"
#include "SymbolDefinitionToken.h"
#include "ExpressionBuilder.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/SymbolDefinitionToken.cpp File Reference

```
#include "SymbolDefinitionToken.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/SymbolTable.cpp File Reference

```
#include "SymbolTable.h"
#include "SymbolData.h"
#include "PrintHelpers.h"
#include <iostream>
#include <iomanip>
```

Namespaces

- **bnssassembler**

Functions

- std::ostream & **bnssassembler::operator<<** (std::ostream &os, const SymbolTable &table)

Code/Assembler/Source/SymbolToken.cpp File Reference

```
#include "SymbolToken.h"
#include "Symbol.h"
#include <climits>
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/Token.cpp File Reference

```
#include "Token.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/UnconditionalJumpInstructionParser.cpp File Reference

```
#include "UnconditionalJumpInstructionParser.h"
#include "MemoryDirectParser.h"
#include "RegisterIndirectOffsetParser.h"
#include "RegisterIndirectParser.h"
```

Namespaces

- **bnssassembler**

Code/Assembler/Source/z85.cpp File Reference

```
#include <assert.h>
#include <limits.h>
#include "z85.h"
```

Macros

- #define **DIV85_MAGIC**   3233857729ULL
- #define **DIV85**(number)   ((**uint32_t**)((**DIV85_MAGIC** * number) >> 32) >> 6)

Typedefs

- typedef unsigned int **uint32_t**
- typedef unsigned char **byte**
- typedef char **Z85_uint32_t_static_assert**[(sizeof(**uint32_t**) *CHAR_BIT==32) *2 - 1]
- typedef char **Z85_div85_magic_static_assert**[(sizeof(**DIV85_MAGIC**) *CHAR_BIT==64) *2 - 1]

Functions

- char * **Z85_encode_unsafe** (const char *source, const char *sourceEnd, char *dest)
  Encodes bytes from [source;sourceEnd) range into 'dest'. It can be used for implementation of your own padding scheme. Preconditions:
- char * **Z85_decode_unsafe** (const char *source, const char *sourceEnd, char *dest)
  Decodes symbols from [source;sourceEnd) range into 'dest'. It can be used for implementation of your own padding scheme. Preconditions:
- size_t **Z85_encode_bound** (size_t size)
  *Evaluates a size of output buffer needed to encode 'size' bytes into string of printable symbols using **Z85_encode()**.*
- size_t **Z85_decode_bound** (size_t size)
  *Evaluates a size of output buffer needed to decode 'size' symbols into binary string using **Z85_decode()**.*
- size_t **Z85_encode** (const char *source, char *dest, size_t inputSize)
  *Encodes 'inputSize' bytes from 'source' into 'dest'. If 'inputSize' is not divisible by 4 with no remainder, 0 is retured. Destination buffer must be already allocated. Use **Z85_encode_bound()** to evaluate size of the destination buffer.*
- size_t **Z85_decode** (const char *source, char *dest, size_t inputSize)
  *Decodes 'inputSize' printable symbols from 'source' into 'dest'. If 'inputSize' is not divisible by 5 with no remainder, 0 is returned. Destination buffer must be already allocated. Use **Z85_decode_bound()** to evaluate size of the destination buffer.*
- size_t **Z85_encode_with_padding_bound** (size_t size)
  *Evaluates a size of output buffer needed to encode 'size' bytes into string of printable symbols using **Z85_encode_with_padding()**.*
- size_t **Z85_decode_with_padding_bound** (const char *source, size_t size)
  *Evaluates a size of output buffer needed to decode 'size' symbols into binary string using **Z85_decode_with_padding()**.*
- size_t **Z85_encode_with_padding** (const char *source, char *dest, size_t inputSize)
  *Encodes 'inputSize' bytes from 'source' into 'dest'. If 'inputSize' is not divisible by 4 with no remainder, 'source' is padded. Destination buffer must be already allocated. Use **Z85_encode_with_padding_bound()** to evaluate size of the destination buffer.*
- size_t **Z85_decode_with_padding** (const char *source, char *dest, size_t inputSize)
  *Decodes 'inputSize' printable symbols from 'source' into 'dest', encoded with **Z85_encode_with_padding()**. Destination buffer must be already allocated. Use **Z85_decode_with_padding_bound()** to evaluate size of the destination buffer.*

Variables

- static const char * **base85**
- static **byte base256** []

Macro Definition Documentation

#define DIV85(number)   ((uint32_t)((DIV85_MAGIC * number) >> 32) >> 6)

Definition at line 46 of file z85.cpp.

Referenced by Z85_encode_unsafe().

#define DIV85_MAGIC   3233857729ULL

Definition at line 42 of file z85.cpp.

Typedef Documentation

typedef unsigned char byte

Definition at line 37 of file z85.cpp.

typedef unsigned int uint32_t

Definition at line 36 of file z85.cpp.

typedef char Z85_div85_magic_static_assert[(sizeof(DIV85_MAGIC) *CHAR_BIT==64) *2 - 1]

Definition at line 44 of file z85.cpp.

typedef char Z85_uint32_t_static_assert[(sizeof(uint32_t) *CHAR_BIT==32) *2 - 1]

Definition at line 40 of file z85.cpp.

Function Documentation

size_t Z85_decode (const char * *source*, char * *dest*, size_t *inputSize*)

Decodes 'inputSize' printable symbols from 'source' into 'dest'. If 'inputSize' is not divisible by 5 with no remainder, 0 is returned. Destination buffer must be already allocated. Use **Z85_decode_bound()** to evaluate size of the destination buffer.

Parameters:

source	in, input buffer (printable string to be decoded)
dest	out, destination buffer

inputSize	in, number of symbols to be decoded

Returns:

number of bytes written into 'dest' or 0 if something goes wrong

Definition at line 146 of file z85.cpp.

```
{
    if (!source || !dest || inputSize % 5)
    {
        assert(!"wrong source, destination or input size");
        return 0;
    }

    return Z85_decode_unsafe(source, source + inputSize, dest) - dest;
}
```

size_t Z85_decode_bound (size_t *size*)

Evaluates a size of output buffer needed to decode 'size' symbols into binary string using **Z85_decode()**.

Parameters:

size	in, number of symbols to be decoded

Returns:

minimal size of output buffer in bytes

Definition at line 130 of file z85.cpp.

Referenced by Z85_decode_with_padding_bound().

```
{
    return size * 4 / 5;
}
```

char* Z85_decode_unsafe (const char * *source*, const char * *sourceEnd*, char * *dest*)

Decodes symbols from [source;sourceEnd) range into 'dest'. It can be used for implementation of your own padding scheme. Preconditions:

- (sourceEnd - source) % 5 == 0
- destination buffer must be already allocated

Parameters:

source	in, begin of input buffer
sourceEnd	in, end of input buffer (not included)
dest	out, output buffer

Returns:

a pointer immediately after last byte written into the 'dest'

Definition at line 100 of file z85.cpp.

Referenced by Z85_decode(), and Z85_decode_with_padding().

```
{
    byte* src = (byte*)source;
    byte* end = (byte*)sourceEnd;
    byte* dst = (byte*)dest;
    uint32_t value;

    for (; src != end; src += 5, dst += 4)
    {
        value = base256[(src[0] - 32) & 127];
        value = value * 85 + base256[(src[1] - 32) & 127];
        value = value * 85 + base256[(src[2] - 32) & 127];
        value = value * 85 + base256[(src[3] - 32) & 127];
        value = value * 85 + base256[(src[4] - 32) & 127];

        // pack big-endian frame
        dst[0] = value >> 24;
        dst[1] = (byte)(value >> 16);
        dst[2] = (byte)(value >> 8);
        dst[3] = (byte)(value);
    }

    return (char*)dst;
}
```

size_t Z85_decode_with_padding (const char * *source*, char * *dest*, size_t *inputSize*)

Decodes 'inputSize' printable symbols from 'source' into 'dest', encoded with **Z85_encode_with_padding()**. Destination buffer must be already allocated. Use **Z85_decode_with_padding_bound()** to evaluate size of the destination buffer.

Parameters:

source	in, input buffer (printable string to be decoded)
dest	out, destination buffer
inputSize	in, number of symbols to be decoded

Returns:

number of bytes written into 'dest' or 0 if something goes wrong

Definition at line 203 of file z85.cpp.

```
{
    char*       dst = dest;
    size_t      tailBytes;
    char        tailBuf[4] = { 0 };
    const char* end = source + inputSize;

    assert(source && dest && (inputSize == 0 || (inputSize - 1) % 5 == 0));

    // zero length string is not padded
    if (!source || !dest || inputSize == 0 || (inputSize - 1) % 5)
    {
        return 0;
    }

    tailBytes = (source++)[0] - '0'; // possible values: 1, 2, 3 or 4
    if (tailBytes - 1 > 3)
    {
        assert(!"wrong tail bytes count");
        return 0;
    }

    end -= 5;
    if (source != end)
    {
        // decode body
        dst = Z85_decode_unsafe(source, end, dst);
    }

    // decode last 5 bytes chunk
    Z85_decode_unsafe(end, end + 5, tailBuf);

    switch (tailBytes)
    {
    case 4:
        dst[3] = tailBuf[3];
    case 3:
        dst[2] = tailBuf[2];
    case 2:
        dst[1] = tailBuf[1];
    case 1:
        dst[0] = tailBuf[0];
    }

    return dst - dest + tailBytes;
}
```

size_t Z85_decode_with_padding_bound (const char * *source*, size_t *size*)

Evaluates a size of output buffer needed to decode 'size' symbols into binary string using **Z85_decode_with_padding()**.

Parameters:

source	in, input buffer (first symbol is read from 'source' to evaluate padding)
size	in, number of symbols to be decoded

Returns:

minimal size of output buffer in bytes

Definition at line 164 of file z85.cpp.

```
{
    if (size == 0 || !source || (byte)(source[0] - '0' - 1) > 3) return 0;
    return Z85_decode_bound(size - 1) - 4 + (source[0] - '0');
}
```

size_t Z85_encode (const char * *source*, char * *dest*, size_t *inputSize*)

Encodes 'inputSize' bytes from 'source' into 'dest'. If 'inputSize' is not divisible by 4 with no remainder, 0 is retured. Destination buffer must be already allocated. Use **Z85_encode_bound()** to evaluate size of the destination buffer.

Parameters:

source	in, input buffer (binary string to be encoded)
dest	out, destination buffer
inputSize	in, number of bytes to be encoded

Returns:

number of printable symbols written into 'dest' or 0 if something goes wrong

Definition at line 135 of file z85.cpp.

```
{
    if (!source || !dest || inputSize % 4)
    {
        assert(!"wrong source, destination or input size");
        return 0;
    }

    return Z85_encode_unsafe(source, source + inputSize, dest) - dest;
}
```

size_t Z85_encode_bound (size_t *size*)

Evaluates a size of output buffer needed to encode 'size' bytes into string of printable symbols using **Z85_encode()**.

Parameters:

size	in, number of bytes to be encoded

Returns:

minimal size of output buffer in bytes

Definition at line 125 of file z85.cpp.

Referenced by Z85_encode_with_padding_bound().

```
{
    return size * 5 / 4;
}
```

char* Z85_encode_unsafe (const char * *source*, const char * *sourceEnd*, char * *dest*)

Encodes bytes from [source;sourceEnd) range into 'dest'. It can be used for implementation of your own padding scheme. Preconditions:

- (sourceEnd - source) % 4 == 0
- destination buffer must be already allocated

Parameters:

source	in, begin of input buffer
sourceEnd	in, end of input buffer (not included)
dest	out, output buffer

Returns:

a pointer immediately after last symbol written into the 'dest'

Definition at line 77 of file z85.cpp.

Referenced by Z85_encode(), and Z85_encode_with_padding().

```
{
    byte* src = (byte*)source;
    byte* end = (byte*)sourceEnd;
    byte* dst = (byte*)dest;
    uint32_t value;
    uint32_t value2;

    for (; src != end; src += 4, dst += 5)
    {
        // unpack big-endian frame
        value = (src[0] << 24) | (src[1] << 16) | (src[2] << 8) | src[3];

        value2 = DIV85(value); dst[4] = base85[value - value2 * 85]; value = value2;
        value2 = DIV85(value); dst[3] = base85[value - value2 * 85]; value = value2;
        value2 = DIV85(value); dst[2] = base85[value - value2 * 85]; value = value2;
        value2 = DIV85(value); dst[1] = base85[value - value2 * 85];
        dst[0] = base85[value2];
    }

    return (char*)dst;
}
```

size_t Z85_encode_with_padding (const char * *source*, char * *dest*, size_t *inputSize*)

Encodes 'inputSize' bytes from 'source' into 'dest'. If 'inputSize' is not divisible by 4 with no remainder, 'source' is padded. Destination buffer must be already allocated. Use **Z85_encode_with_padding_bound()** to evaluate size of the destination buffer.

Parameters:

source	in, input buffer (binary string to be encoded)
dest	out, destination buffer
inputSize	in, number of bytes to be encoded

Returns:

number of printable symbols written into 'dest' or 0 if something goes wrong

Definition at line 170 of file z85.cpp.

```
{
    size_t      tailBytes = inputSize % 4;
    char        tailBuf[4] = { 0 };
    char*       dst = dest;
    const char* end = source + inputSize - tailBytes;

    assert(source && dest);

    // zero length string is not padded
    if (!source || !dest || inputSize == 0)
    {
        return 0;
    }

    (dst++)[0] = (tailBytes == 0 ? '4' : '0' + (char)tailBytes); // write tail bytes count
    dst = Z85_encode_unsafe(source, end, dst);                   // write body

                                                                  // write tail
    switch (tailBytes)
    {
    case 3:
        tailBuf[2] = end[2];
    case 2:
        tailBuf[1] = end[1];
    case 1:
        tailBuf[0] = end[0];
        dst = Z85_encode_unsafe(tailBuf, tailBuf + 4, dst);
    }

    return dst - dest;
}
```

size_t Z85_encode_with_padding_bound (size_t *size*)

Evaluates a size of output buffer needed to encode 'size' bytes into string of printable symbols using **Z85_encode_with_padding()**.

Parameters:

size	in, number of bytes to be encoded

Returns:

minimal size of output buffer in bytes

Definition at line 157 of file z85.cpp.

```
{
    if (size == 0) return 0;
    size = Z85_encode_bound(size);
    return size + (5 - size % 5) % 5 + 1;
}
```

Variable Documentation

byte base256[] `[static]`

```
Initial value:=
{
    0x00, 0x44, 0x00, 0x54, 0x53, 0x52, 0x48, 0x00,
    0x4B, 0x4C, 0x46, 0x41, 0x00, 0x3F, 0x3E, 0x45,
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
    0x08, 0x09, 0x40, 0x00, 0x49, 0x42, 0x4A, 0x47,
    0x51, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2A,
    0x2B, 0x2C, 0x2D, 0x2E, 0x2F, 0x30, 0x31, 0x32,
    0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A,
    0x3B, 0x3C, 0x3D, 0x4D, 0x00, 0x4E, 0x43, 0x00,
    0x00, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10,
    0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18,
    0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F, 0x20,
    0x21, 0x22, 0x23, 0x4F, 0x00, 0x50, 0x00, 0x00
}
```

Definition at line 61 of file z85.cpp.

Referenced by Z85_decode_unsafe().

const char* base85 `[static]`

```
Initial value:=
{
    "0123456789"
    "abcdefghij"
    "klmnopqrst"
    "uvwxyzABCD"
    "EFGHIJKLMN"
    "OPQRSTUVWX"
    "YZ.-:+=^!/"
    "*?&<>()[]{"
    "}@%$#"
}
```

Definition at line 48 of file z85.cpp.

Referenced by Z85_encode_unsafe().

Code/Emulator/Source/z85.cpp File Reference

```
#include <assert.h>
#include <limits.h>
#include "z85.h"
```

Macros

- #define **DIV85_MAGIC**   3233857729ULL
- #define **DIV85**(number)   ((**uint32_t**)((**DIV85_MAGIC** * number) >> 32) >> 6)

Typedefs

- typedef unsigned int **uint32_t**
- typedef unsigned char **byte**
- typedef char **Z85_uint32_t_static_assert**[(sizeof(**uint32_t**) *CHAR_BIT==32) *2 - 1]
- typedef char **Z85_div85_magic_static_assert**[(sizeof(**DIV85_MAGIC**) *CHAR_BIT==64) *2 - 1]

Functions

- char * **Z85_encode_unsafe** (const char *source, const char *sourceEnd, char *dest)
  Encodes bytes from [source;sourceEnd) range into 'dest'. It can be used for implementation of your own padding scheme. Preconditions:
- char * **Z85_decode_unsafe** (const char *source, const char *sourceEnd, char *dest)
  Decodes symbols from [source;sourceEnd) range into 'dest'. It can be used for implementation of your own padding scheme. Preconditions:
- size_t **Z85_encode_bound** (size_t size)
  *Evaluates a size of output buffer needed to encode 'size' bytes into string of printable symbols using **Z85_encode()**.*
- size_t **Z85_decode_bound** (size_t size)
  *Evaluates a size of output buffer needed to decode 'size' symbols into binary string using **Z85_decode()**.*
- size_t **Z85_encode** (const char *source, char *dest, size_t inputSize)
  *Encodes 'inputSize' bytes from 'source' into 'dest'. If 'inputSize' is not divisible by 4 with no remainder, 0 is retured. Destination buffer must be already allocated. Use **Z85_encode_bound()** to evaluate size of the destination buffer.*
- size_t **Z85_decode** (const char *source, char *dest, size_t inputSize)
  *Decodes 'inputSize' printable symbols from 'source' into 'dest'. If 'inputSize' is not divisible by 5 with no remainder, 0 is returned. Destination buffer must be already allocated. Use **Z85_decode_bound()** to evaluate size of the destination buffer.*
- size_t **Z85_encode_with_padding_bound** (size_t size)
  *Evaluates a size of output buffer needed to encode 'size' bytes into string of printable symbols using **Z85_encode_with_padding()**.*
- size_t **Z85_decode_with_padding_bound** (const char *source, size_t size)
  *Evaluates a size of output buffer needed to decode 'size' symbols into binary string using **Z85_decode_with_padding()**.*
- size_t **Z85_encode_with_padding** (const char *source, char *dest, size_t inputSize)
  *Encodes 'inputSize' bytes from 'source' into 'dest'. If 'inputSize' is not divisible by 4 with no remainder, 'source' is padded. Destination buffer must be already allocated. Use **Z85_encode_with_padding_bound()** to evaluate size of the destination buffer.*
- size_t **Z85_decode_with_padding** (const char *source, char *dest, size_t inputSize)
  *Decodes 'inputSize' printable symbols from 'source' into 'dest', encoded with **Z85_encode_with_padding()**. Destination buffer must be already allocated. Use **Z85_decode_with_padding_bound()** to evaluate size of the destination buffer.*

Variables

- static const char * **base85**
- static **byte base256** []

Macro Definition Documentation

#define DIV85(number)   ((uint32_t)((DIV85_MAGIC * number) >> 32) >> 6)

Definition at line 46 of file z85.cpp.

Referenced by Z85_encode_unsafe().

#define DIV85_MAGIC   3233857729ULL

Definition at line 42 of file z85.cpp.

Typedef Documentation

typedef unsigned char byte

Definition at line 37 of file z85.cpp.

typedef unsigned int uint32_t

Definition at line 36 of file z85.cpp.

typedef char Z85_div85_magic_static_assert[(sizeof(DIV85_MAGIC) *CHAR_BIT==64) *2 - 1]

Definition at line 44 of file z85.cpp.

typedef char Z85_uint32_t_static_assert[(sizeof(uint32_t) *CHAR_BIT==32) *2 - 1]

Definition at line 40 of file z85.cpp.

Function Documentation

size_t Z85_decode (const char * *source*, char * *dest*, size_t *inputSize*)

Decodes 'inputSize' printable symbols from 'source' into 'dest'. If 'inputSize' is not divisible by 5 with no remainder, 0 is returned. Destination buffer must be already allocated. Use **Z85_decode_bound()** to evaluate size of the destination buffer.

Parameters:

source	in, input buffer (printable string to be decoded)
dest	out, destination buffer

inputSize	in, number of symbols to be decoded

Returns:

number of bytes written into 'dest' or 0 if something goes wrong

Definition at line 146 of file z85.cpp.

References Z85_decode_unsafe().

Referenced by z85::decode().

```
{
    if (!source || !dest || inputSize % 5)
    {
        assert(!"wrong source, destination or input size");
        return 0;
    }

    return Z85_decode_unsafe(source, source + inputSize, dest) - dest;
}
```

size_t Z85_decode_bound (size_t *size*)

Evaluates a size of output buffer needed to decode 'size' symbols into binary string using **Z85_decode()**.

Parameters:

size	in, number of symbols to be decoded

Returns:

minimal size of output buffer in bytes

Definition at line 130 of file z85.cpp.

Referenced by z85::decode().

```
{
    return size * 4 / 5;
}
```

char* Z85_decode_unsafe (const char * *source*, const char * *sourceEnd*, char * *dest*)

Decodes symbols from [source;sourceEnd) range into 'dest'. It can be used for implementation of your own padding scheme. Preconditions:

- (sourceEnd - source) % 5 == 0
- destination buffer must be already allocated

Parameters:

source	in, begin of input buffer
sourceEnd	in, end of input buffer (not included)
dest	out, output buffer

Returns:

a pointer immediately after last byte written into the 'dest'

Definition at line 100 of file z85.cpp.

References base256, and cxxopts::value().

```
{
    byte* src = (byte*)source;
    byte* end = (byte*)sourceEnd;
    byte* dst = (byte*)dest;
    uint32_t value;

    for (; src != end; src += 5, dst += 4)
    {
        value = base256[(src[0] - 32) & 127];
        value = value * 85 + base256[(src[1] - 32) & 127];
        value = value * 85 + base256[(src[2] - 32) & 127];
        value = value * 85 + base256[(src[3] - 32) & 127];
        value = value * 85 + base256[(src[4] - 32) & 127];

        // pack big-endian frame
        dst[0] = value >> 24;
        dst[1] = (byte)(value >> 16);
        dst[2] = (byte)(value >> 8);
        dst[3] = (byte)(value);
    }

    return (char*)dst;
}
```

size_t Z85_decode_with_padding (const char * *source*, char * *dest*, size_t *inputSize*)

Decodes 'inputSize' printable symbols from 'source' into 'dest', encoded with **Z85_encode_with_padding()**. Destination buffer must be already allocated. Use **Z85_decode_with_padding_bound()** to evaluate size of the destination buffer.

Parameters:

source	in, input buffer (printable string to be decoded)
dest	out, destination buffer
inputSize	in, number of symbols to be decoded

Returns:

number of bytes written into 'dest' or 0 if something goes wrong

Definition at line 203 of file z85.cpp.

References Z85_decode_unsafe().

Referenced by z85::decode_with_padding().

```
{
    char*       dst = dest;
    size_t      tailBytes;
    char        tailBuf[4] = { 0 };
    const char* end = source + inputSize;

    assert(source && dest && (inputSize == 0 || (inputSize - 1) % 5 == 0));

    // zero length string is not padded
    if (!source || !dest || inputSize == 0 || (inputSize - 1) % 5)
    {
        return 0;
    }

    tailBytes = (source++)[0] - '0'; // possible values: 1, 2, 3 or 4
    if (tailBytes - 1 > 3)
    {
        assert(!"wrong tail bytes count");
        return 0;
    }

    end -= 5;
    if (source != end)
    {
        // decode body
        dst = Z85_decode_unsafe(source, end, dst);
    }

    // decode last 5 bytes chunk
    Z85_decode_unsafe(end, end + 5, tailBuf);

    switch (tailBytes)
    {
    case 4:
        dst[3] = tailBuf[3];
    case 3:
        dst[2] = tailBuf[2];
    case 2:
        dst[1] = tailBuf[1];
    case 1:
        dst[0] = tailBuf[0];
    }

    return dst - dest + tailBytes;
}
```

size_t Z85_decode_with_padding_bound (const char * *source*, size_t *size*)

Evaluates a size of output buffer needed to decode 'size' symbols into binary string using **Z85_decode_with_padding()**.

Parameters:

source	in, input buffer (first symbol is read from 'source' to evaluate padding)
size	in, number of symbols to be decoded

Returns:

minimal size of output buffer in bytes

Definition at line 164 of file z85.cpp.

References Z85_decode_bound().

Referenced by z85::decode_with_padding().

```
{
    if (size == 0 || !source || (byte)(source[0] - '0' - 1) > 3) return 0;
    return Z85_decode_bound(size - 1) - 4 + (source[0] - '0');
}
```

size_t Z85_encode (const char * *source*, char * *dest*, size_t *inputSize*)

Encodes 'inputSize' bytes from 'source' into 'dest'. If 'inputSize' is not divisible by 4 with no remainder, 0 is retured. Destination buffer must be already allocated. Use **Z85_encode_bound()** to evaluate size of the destination buffer.

Parameters:

source	in, input buffer (binary string to be encoded)
dest	out, destination buffer
inputSize	in, number of bytes to be encoded

Returns:

number of printable symbols written into 'dest' or 0 if something goes wrong

Definition at line 135 of file z85.cpp.

References Z85_encode_unsafe().

Referenced by z85::encode().

```
{
    if (!source || !dest || inputSize % 4)
    {
        assert(!"wrong source, destination or input size");
        return 0;
    }

    return Z85_encode_unsafe(source, source + inputSize, dest) - dest;
}
```

size_t Z85_encode_bound (size_t *size*)

Evaluates a size of output buffer needed to encode 'size' bytes into string of printable symbols using **Z85_encode()**.

Parameters:

size	in, number of bytes to be encoded

Returns:

minimal size of output buffer in bytes

Definition at line 125 of file z85.cpp.

Referenced by z85::encode().

```
{
    return size * 5 / 4;
}
```

char* Z85_encode_unsafe (const char * *source*, const char * *sourceEnd*, char * *dest*)

Encodes bytes from [source;sourceEnd) range into 'dest'. It can be used for implementation of your own padding scheme. Preconditions:

- (sourceEnd - source) % 4 == 0
- destination buffer must be already allocated

Parameters:

source	in, begin of input buffer
sourceEnd	in, end of input buffer (not included)
dest	out, output buffer

Returns:

a pointer immediately after last symbol written into the 'dest'

Definition at line 77 of file z85.cpp.

References base85, DIV85, and cxxopts::value().

```
{
    byte* src = (byte*)source;
    byte* end = (byte*)sourceEnd;
    byte* dst = (byte*)dest;
    uint32_t value;
    uint32_t value2;

    for (; src != end; src += 4, dst += 5)
    {
        // unpack big-endian frame
        value = (src[0] << 24) | (src[1] << 16) | (src[2] << 8) | src[3];

        value2 = DIV85(value); dst[4] = base85[value - value2 * 85]; value = value2;
        value2 = DIV85(value); dst[3] = base85[value - value2 * 85]; value = value2;
        value2 = DIV85(value); dst[2] = base85[value - value2 * 85]; value = value2;
        value2 = DIV85(value); dst[1] = base85[value - value2 * 85];
        dst[0] = base85[value2];
    }

    return (char*)dst;
}
```

size_t Z85_encode_with_padding (const char * *source*, char * *dest*, size_t *inputSize*)

Encodes 'inputSize' bytes from 'source' into 'dest'. If 'inputSize' is not divisible by 4 with no remainder, 'source' is padded. Destination buffer must be already allocated. Use **Z85_encode_with_padding_bound()** to evaluate size of the destination buffer.

Parameters:

source	in, input buffer (binary string to be encoded)
dest	out, destination buffer
inputSize	in, number of bytes to be encoded

Returns:

number of printable symbols written into 'dest' or 0 if something goes wrong

Definition at line 170 of file z85.cpp.

References Z85_encode_unsafe().

Referenced by z85::encode_with_padding().

```
{
    size_t      tailBytes = inputSize % 4;
    char        tailBuf[4] = { 0 };
    char*       dst = dest;
    const char* end = source + inputSize - tailBytes;

    assert(source && dest);

    // zero length string is not padded
    if (!source || !dest || inputSize == 0)
    {
        return 0;
    }

    (dst++)[0] = (tailBytes == 0 ? '4' : '0' + (char)tailBytes); // write tail bytes count
    dst = Z85_encode_unsafe(source, end, dst);                   // write body

                                                                  // write tail
    switch (tailBytes)
    {
    case 3:
        tailBuf[2] = end[2];
    case 2:
        tailBuf[1] = end[1];
    case 1:
        tailBuf[0] = end[0];
        dst = Z85_encode_unsafe(tailBuf, tailBuf + 4, dst);
    }

    return dst - dest;
}
```

size_t Z85_encode_with_padding_bound (size_t *size*)

Evaluates a size of output buffer needed to encode 'size' bytes into string of printable symbols using **Z85_encode_with_padding()**.

Parameters:

size	in, number of bytes to be encoded

Returns:

minimal size of output buffer in bytes

Definition at line 157 of file z85.cpp.

References Z85_encode_bound().

Referenced by z85::encode_with_padding().

```
{
    if (size == 0) return 0;
    size = Z85_encode_bound(size);
    return size + (5 - size % 5) % 5 + 1;
}
```

Variable Documentation

byte base256[] [static]

```
Initial value:=
{
    0x00, 0x44, 0x00, 0x54, 0x53, 0x52, 0x48, 0x00,
    0x4B, 0x4C, 0x46, 0x41, 0x00, 0x3F, 0x3E, 0x45,
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
    0x08, 0x09, 0x40, 0x00, 0x49, 0x42, 0x4A, 0x47,
    0x51, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2A,
    0x2B, 0x2C, 0x2D, 0x2E, 0x2F, 0x30, 0x31, 0x32,
    0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A,
    0x3B, 0x3C, 0x3D, 0x4D, 0x00, 0x4E, 0x43, 0x00,
    0x00, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x10,
    0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17, 0x18,
    0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F, 0x20,
    0x21, 0x22, 0x23, 0x4F, 0x00, 0x50, 0x00, 0x00
}
```

Definition at line 61 of file z85.cpp.

Referenced by Z85_decode_unsafe().

const char* base85 [static]

```
Initial value:=
{
    "0123456789"
    "abcdefghij"
    "klmnopqrst"
    "uvwxyzABCD"
    "EFGHIJKLMN"
    "OPQRSTUVWX"
    "YZ.-:+=^!/"
    "*?&<>()[]{"
    "}@%$#"
}
```

Definition at line 48 of file z85.cpp.

Referenced by Z85_encode_unsafe().

Code/Assembler/Source/z85_impl.cpp File Reference

```
#include "z85_cpp.h"
#include <cassert>
#include "z85.h"
```

Namespaces

- **z85**

Functions

- std::string **z85::encode_with_padding** (const char *source, size_t inputSize)
  Encodes 'inputSize' bytes from 'source'. If 'inputSize' is not divisible by 4 with no remainder, 'source' is padded.
- std::string **z85::encode_with_padding** (const std::string &source)
- std::string **z85::decode_with_padding** (const char *source, size_t inputSize)
  *Decodes 'inputSize' printable symbols from 'source', encoded with **encode_with_padding()**.*
- std::string **z85::decode_with_padding** (const std::string &source)
- std::string **z85::encode** (const char *source, size_t inputSize)
  Encodes 'inputSize' bytes from 'source'. If 'inputSize' is not divisible by 4 with no remainder, empty string is retured.
- std::string **z85::encode** (const std::string &source)
- std::string **z85::decode** (const char *source, size_t inputSize)
  Decodes 'inputSize' printable symbols from 'source'. If 'inputSize' is not divisible by 5 with no remainder, empty string is returned.
- std::string **z85::decode** (const std::string &source)

Code/Emulator/Source/z85_impl.cpp File Reference

```
#include "z85_cpp.h"
#include <cassert>
#include "z85.h"
```

Namespaces

- **z85**

Functions

- std::string **z85::encode_with_padding** (const char *source, size_t inputSize)
  Encodes 'inputSize' bytes from 'source'. If 'inputSize' is not divisible by 4 with no remainder, 'source' is padded.
- std::string **z85::encode_with_padding** (const std::string &source)
- std::string **z85::decode_with_padding** (const char *source, size_t inputSize)
  Decodes 'inputSize' printable symbols from 'source', encoded with ***encode_with_padding()****.*
- std::string **z85::decode_with_padding** (const std::string &source)
- std::string **z85::encode** (const char *source, size_t inputSize)
  Encodes 'inputSize' bytes from 'source'. If 'inputSize' is not divisible by 4 with no remainder, empty string is retured.
- std::string **z85::encode** (const std::string &source)
- std::string **z85::decode** (const char *source, size_t inputSize)
  Decodes 'inputSize' printable symbols from 'source'. If 'inputSize' is not divisible by 5 with no remainder, empty string is returned.
- std::string **z85::decode** (const std::string &source)

Code/Emulator/Include/AddExecuter.h File Reference

```
#include "AluExecuter.h"
```

Classes

- class **bnssemulator::AddExecuter**

Class representing the executer for the add instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/Address.h File Reference

```
#include <cstddef>
#include <cstdint>
```

Namespaces

- **bnssemulator**

Variables

- const size_t **bnssemulator::BLOCK_BITS** = 16
- const **uint32_t bnssemulator::PAGE_MASK** = ~0 << BLOCK_BITS
- const **uint32_t bnssemulator::OFFSET_MASK** = ~PAGE_MASK
- const size_t **bnssemulator::BLOCK_SIZE** = OFFSET_MASK + 1

Code/Emulator/Include/AddressSpace.h File Reference

```
#include "Segment.h"
#include <map>
#include "SectionData.h"
#include "InstructionBitField.h"
#include "SymbolData.h"
#include <unordered_map>
#include <queue>
```

Classes

- class **bnssemulator::AddressSpace**

Class representing the address space of the emulator. Namespaces

- **bnssemulator**

Code/Emulator/Include/AluExecuter.h File Reference

`#include "Executer.h"`

Classes

- class **bnssemulator::AluExecuter**

Base class used for executing ALU instructions. Namespaces

- **bnssemulator**

Code/Emulator/Include/AndExecuter.h File Reference

```
#include "AluExecuter.h"
```

Classes

- class **bnssemulator::AndExecuter**

Class representing the executer for the and instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/AslExecuter.h File Reference

`#include "AluExecuter.h"`

Classes

- class **bnssemulator::AslExecuter**

Class representing the executer for the asl instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/AsrExecuter.h File Reference

```
#include "AluExecuter.h"
```

Classes

- class **bnssemulator::AsrExecuter**

Class representing the executer for the asr instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/AssemblerOutput.h File Reference

```
#include <istream>
#include "SectionData.h"
#include "SymbolData.h"
#include <unordered_set>
#include <vector>
#include <unordered_map>
```

Classes

- class **bnssemulator::AssemblerOutput**

Class representing the output from the assembler. Namespaces

- **bnssemulator**

Code/Emulator/Include/CallExecuter.h File Reference

```
#include "Executer.h"
```

Classes

- class **bnssemulator::CallExecuter**

Class representing the executer for the call instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/ConditionalJumpExecuter.h File Reference

```
#include "Executer.h"
```

Classes

- class **bnssemulator::ConditionalJumpExecuter**

Base executer for conditional jump instructions. Namespaces

- **bnssemulator**

Code/Emulator/Include/ConsoleInputOutput.h File Reference

Namespaces

- **consoleio**

Functions

- bool **consoleio::keyboardHit** ()
- int **consoleio::getCharacter** ()

Code/Emulator/Include/Context.h File Reference

```
#include "Register.h"
#include <vector>
#include "AssemblerOutput.h"
#include "AddressSpace.h"
#include <queue>
#include <mutex>
```

Classes

- class **bnssemulator::Context**

Class representing the context of the processor. Namespaces

- **bnssemulator**

Code/Emulator/Include/DivideExecuter.h File Reference

```
#include "AluExecuter.h"
```

Classes

- class **bnssemulator::DivideExecuter**

Class representing the executer of the divide instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/Executer.h File Reference

```
#include "InstructionBitField.h"
#include "Context.h"
```

Classes

- class **bnssemulator::Executer**

Base class used for executing instructions. Namespaces

- **bnssemulator**

Code/Emulator/Include/IntExecuter.h File Reference

```
#include "Executer.h"
```

Classes

- class **bnssemulator::IntExecuter**

Class representing the executer for the int instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/JgezExecuter.h File Reference

`#include "ConditionalJumpExecuter.h"`

Classes

- class **bnssemulator::JgezExecuter**

Class representing the executer for the jgez instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/JgzExecuter.h File Reference

`#include "ConditionalJumpExecuter.h"`

Classes

- class **bnssemulator::JgzExecuter**

Class representing the executer for the jgz instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/JlezExecuter.h File Reference

```
#include "ConditionalJumpExecuter.h"
```

Classes

- class **bnssemulator::JlezExecuter**

Class representing the executer for the jlez instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/JlzExecuter.h File Reference

`#include "ConditionalJumpExecuter.h"`

Classes

- class **bnssemulator::JlzExecuter**

Class representing the executer for the jlz instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/JmpExecuter.h File Reference

```
#include "Executer.h"
```

Classes

- class **bnssemulator::JmpExecuter**

Class representing the executer for the jmp instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/JnzExecuter.h File Reference

```
#include "ConditionalJumpExecuter.h"
```

Classes

- class **bnssemulator::JnzExecuter**

Class representing the executer for the jnz instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/JzExecuter.h File Reference

```
#include "ConditionalJumpExecuter.h"
```

Classes

- class **bnssemulator::JzExecuter**

Class representing the executer for the jz instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/KeyboardListener.h File Reference

```
#include "Context.h"
```

Classes

- class **bnssemulator::KeyboardListener**

Class representing the keyboard listener thread. Namespaces

- **bnssemulator**

Code/Emulator/Include/LoadExecuter.h File Reference

```
#include "Executer.h"
```

Classes

- class **bnssemulator::LoadExecuter**

Class representing the executer for the load instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/ModuloExecuter.h File Reference

```
#include "AluExecuter.h"
```

Classes

- class **bnssemulator::ModuloExecuter**

Class representing the executer for the modulo instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/MultiplyExecuter.h File Reference

```
#include "AluExecuter.h"
```

Classes

- class **bnssemulator::MultiplyExecuter**

Class representing the executer for the multiply instruction.

Namespaces

- **bnssemulator**

Code/Emulator/Include/NotExecuter.h File Reference

```
#include "Executer.h"
```

Classes

- class **bnssemulator::NotExecuter**

Class representing the executer for the not instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/OrExecuter.h File Reference

```
#include "AluExecuter.h"
```

Classes

- class **bnssemulator::OrExecuter**

Class representing the executer for the or instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/PopExecuter.h File Reference

```
#include "Executer.h"
```

Classes

- class **bnssemulator::PopExecuter**

Class representing the executer for the pop instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/Processor.h File Reference

```
#include "Executer.h"
#include "InstructionCode.h"
#include <unordered_map>
#include <memory>
```

Classes

- class **bnssemulator::Processor**
- *Class representing the processor.* struct **bnssemulator::Processor::ProcessorStaticData**

Namespaces

- **bnssemulator**

Code/Emulator/Include/PushExecuter.h File Reference

```
#include "Executer.h"
```

Classes

- class **bnssemulator::PushExecuter**

Class representing the executer for the push instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/RetExecuter.h File Reference

```
#include "Executer.h"
```

Classes

- class **bnssemulator::RetExecuter**

Class representing the executer for ret instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/Segment.h File Reference

```
#include "SectionType.h"
#include <cstdint>
#include <vector>
#include "InstructionBitField.h"
```

Classes

- class **bnssemulator::Segment**

Class representing one segment of memory. Namespaces

- **bnssemulator**

Code/Emulator/Include/StlHelper.h File Reference

`#include <functional>`

Classes

- struct **bnssemulator::compare_pair_first< T1, T2, Pred >**
- struct **bnssemulator::compare_pair_second< T1, T2, Pred >**
- struct **bnssemulator::compare_pair_difference< T, Pred >**

Namespaces

- **bnssemulator**

Code/Emulator/Include/StoreExecuter.h File Reference

```
#include "Executer.h"
```

Classes

- class **bnssemulator::StoreExecuter**

Class representing the executer for the store instruction. Namespaces

- **bnssemulator**

Code/Emulator/Include/SubtractExecuter.h File Reference

```
#include "AluExecuter.h"
```

Classes

- class **bnssemulator::SubtractExecuter**

Class representing the executer for the subtract instruction.

Namespaces

- **bnssemulator**

Code/Emulator/Include/TimerListener.h File Reference

```
#include "Context.h"
```

Classes

- class **bnssemulator::TimerListener**

Class representing a listener for the timer events. Namespaces

- **bnssemulator**

Code/Emulator/Include/XorExecuter.h File Reference

```
#include "AluExecuter.h"
```

Classes

- class **bnssemulator::XorExecuter**

Class representing the executer for the xor instruction. Namespaces

- **bnssemulator**

Code/Emulator/Source/AddExecuter.cpp File Reference

```
#include "AddExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/AddressSpace.cpp File Reference

```
#include "AddressSpace.h"
#include "StringHelper.h"
#include "StlHelper.h"
#include <list>
#include <set>
#include "SymbolData.h"
#include <unordered_map>
#include <iostream>
```

Namespaces

- **bnssemulator**

Functions

- static void **bnssemulator::removeEmpty** (std::vector< SectionData > &section_table)
- static bool **bnssemulator::checkOverlaps** (const std::vector< SectionData > &section_table)
- static std::list< std::pair< **uint32_t**, **uint32_t** > > **bnssemulator::getAvailable** (const std::vector< SectionData > &section_table)
- static void **bnssemulator::generateAddresses** (std::vector< SectionData > &section_table, std::list< std::pair< **uint32_t**, **uint32_t** >> &available)

Code/Emulator/Source/AluExecuter.cpp File Reference

```
#include "AluExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/AndExecuter.cpp File Reference

```
#include "AndExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/AslExecuter.cpp File Reference

```
#include "AslExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/AsrExecuter.cpp File Reference

```
#include "AsrExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/AssemblerOutput.cpp File Reference

```
#include "AssemblerOutput.h"
#include <string>
#include "MessageException.h"
```

Namespaces

- **bnssemulator**

Functions

- std::istream & **bnssemulator::operator**>> (std::istream &is, AssemblerOutput &data)

Code/Emulator/Source/CallExecuter.cpp File Reference

```
#include "CallExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/ConditionalJumpExecuter.cpp File Reference

```
#include "ConditionalJumpExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/ConsoleInputOutput.cpp File Reference

```
#include "ConsoleInputOutput.h"
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/select.h>
#include <termios.h>
```

Namespaces

- **consoleio**

Functions

- static void **consoleio::restore** ()
- static void **consoleio::initialize** ()
- bool **consoleio::keyboardHit** ()
- int **consoleio::getCharacter** ()

Variables

- static struct termios **consoleio::old_termios**
- static bool **consoleio::initialized** = false
- static const int **consoleio::STDIN_DESCRIPTOR_ID** = 0

Code/Emulator/Source/Context.cpp File Reference

```
#include "Context.h"
#include "MessageException.h"
#include "StringHelper.h"
#include "AddressMode.h"
```

Namespaces

- **bnssemulator**

Functions

- static size_t **bnssemulator::getRegisterIndex** (InstructionBitField instruction, size_t register_index)
  Gets the index of the register in the array of registers.

Code/Emulator/Source/DivideExecuter.cpp File Reference

```
#include "DivideExecuter.h"
#include "MessageException.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/IntExecuter.cpp File Reference

`#include "IntExecuter.h"`

Namespaces

- **bnssemulator**

Code/Emulator/Source/JgezExecuter.cpp File Reference

```
#include "JgezExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/JgzExecuter.cpp File Reference

```
#include "JgzExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/JlezExecuter.cpp File Reference

```
#include "JlezExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/JlzExecuter.cpp File Reference

```
#include "JlzExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/JmpExecuter.cpp File Reference

```
#include "JmpExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/JnzExecuter.cpp File Reference

```
#include "JnzExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/JzExecuter.cpp File Reference

```
#include "JzExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/KeyboardListener.cpp File Reference

```
#include "KeyboardListener.h"
#include "ConsoleInputOutput.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/LoadExecuter.cpp File Reference

```
#include "LoadExecuter.h"
#include "OperandType.h"
#include "MessageException.h"
#include "StringHelper.h"
```

Namespaces

- **bnssemulator**

Functions

- static **uint32_t bnssemulator::fill** (OperandType type, int32_t operand)

Variables

- static const int32_t **bnssemulator::UNSIGNED_BYTE_MASK** = 0x000000ff
- static const int32_t **bnssemulator::UNSIGNED_WORD_MASK** = 0x0000ffff
- static const int32_t **bnssemulator::SIGNED_BYTE_TEST** = 0x00000080
- static const int32_t **bnssemulator::SIGNED_WORD_TEST** = 0x00008000
- static const int32_t **bnssemulator::SIGNED_BYTE_FILL** = 0xffffff00
- static const int32_t **bnssemulator::SIGNED_WORD_FILL** = 0xffff0000

Code/Emulator/Source/ModuloExecuter.cpp File Reference

```
#include "ModuloExecuter.h"
#include "MessageException.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/MultiplyExecuter.cpp File Reference

```
#include "MultiplyExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/NotExecuter.cpp File Reference

```
#include "NotExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/OrExecuter.cpp File Reference

```
#include "OrExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/PopExecuter.cpp File Reference

```
#include "PopExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/Processor.cpp File Reference

```
#include "Processor.h"
#include "IntExecuter.h"
#include "RetExecuter.h"
#include "JmpExecuter.h"
#include "CallExecuter.h"
#include "JzExecuter.h"
#include "JnzExecuter.h"
#include "JgzExecuter.h"
#include "JgezExecuter.h"
#include "JlzExecuter.h"
#include "JlezExecuter.h"
#include "LoadExecuter.h"
#include "StoreExecuter.h"
#include "PushExecuter.h"
#include "PopExecuter.h"
#include "AddExecuter.h"
#include "SubtractExecuter.h"
#include "MultiplyExecuter.h"
#include "DivideExecuter.h"
#include "ModuloExecuter.h"
#include "AndExecuter.h"
#include "OrExecuter.h"
#include "XorExecuter.h"
#include "AslExecuter.h"
#include "AsrExecuter.h"
#include "NotExecuter.h"
#include "MessageException.h"
#include "StringHelper.h"
#include "KeyboardListener.h"
#include <thread>
#include "TimerListener.h"
```

Namespaces

- **bnssemulator**

Functions

- static InstructionCode **bnssemulator::opcode** (InstructionBitField instruction)

Code/Emulator/Source/PushExecuter.cpp File Reference

```
#include "PushExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/Register.cpp File Reference

`#include "Register.h"`

Namespaces

- **bnssemulator**

Functions

- Register **bnssemulator::operator+** (const Register &lhs, const Register &rhs) noexcept
- Register **bnssemulator::operator-** (const Register &lhs, const Register &rhs) noexcept
- Register **bnssemulator::operator*** (const Register &lhs, const Register &rhs) noexcept
- Register **bnssemulator::operator/** (const Register &lhs, const Register &rhs) noexcept
- Register **bnssemulator::operator%** (const Register &lhs, const Register &rhs) noexcept
- Register **bnssemulator::operator &** (const Register &lhs, const Register &rhs) noexcept
- Register **bnssemulator::operator|** (const Register &lhs, const Register &rhs) noexcept
- Register **bnssemulator::operator**^ (const Register &lhs, const Register &rhs) noexcept
- Register **bnssemulator::operator**<< (const Register &lhs, const Register &rhs) noexcept
- Register **bnssemulator::operator**>> (const Register &lhs, const Register &rhs) noexcept

Variables

- static const uint64_t **bnssemulator::TOP_32_BITS** = ~static_cast<uint64_t>(0) << 32

Code/Emulator/Source/RetExecuter.cpp File Reference

```
#include "RetExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/Segment.cpp File Reference

```
#include "Segment.h"
#include "MessageException.h"
#include "StringHelper.h"
#include "InstructionBitFieldUnion.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/StoreExecuter.cpp File Reference

```
#include "StoreExecuter.h"
#include "AddressMode.h"
#include "OperandType.h"
#include "MessageException.h"
#include "StringHelper.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/SubtractExecuter.cpp File Reference

```
#include "SubtractExecuter.h"
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/TimerListener.cpp File Reference

```
#include "TimerListener.h"
#include <chrono>
#include <thread>
```

Namespaces

- **bnssemulator**

Code/Emulator/Source/XorExecuter.cpp File Reference

```
#include "XorExecuter.h"
```

Namespaces

- **bnssemulator**

README.md File Reference

Index

INDEX