CS4380 Assembler v1.0.0.0

Andrew Todd | November 6, 2023

Assembler Framework

Directive represents a language supported directive mnemonic and what the assembler should do when encountering that mnemonic/directive

Language() Directives and Instructions store instances of Directive and Instruction respectively. These hash tables are serched by a pass to allow for the machine code to be produced for a directive or instruction, or to verify the existance of a mnemonic

All "Directives" are realizations of *ASMDirective*

ASMDirective

+ mnemonic: string
# decimalPattern: const regex {static}
# hexPattern: const regex {static}
# asciiPattern: const regex {static}
# asciiEscape: const regex {static}
# stringPattern: const regex {static}

+ Implementation(const Workpiece* const&, const vector<string>&) const: size_t {virtual}

ASMDirective(string&&)

template<std::integral InputType, std::integral TargetType>
SerializeToBuffer(const InputType&, Workpiece* const&): void {static} {inline}

LanguageDefinition

m_languageDirectives: const unordered_map<string, const ASMDirective*>
m_languageInstructions: const unordered_map<string, const ASMInstruction*>
m_reservedKeywords: const unordered_set<string>
m_registerMnemonics: const unordered_map<string>

LanguageDefinition()
~LanguageDefinition()

SetDirectives(const DerivedFromDirective auto&...): void
SetInstructions(const DerivedFromInstruction auto&...): void
SetKeywords(StringType auto&&...): void
SetRegisterMnemonics(StringType auto&&...): void)

+ LanguageDefinition(const LanguageDefinition&) {delete}
+ operator=(const LanguageDefinition&) {delete}
+ LanguageDefinition(LanguageDefinition&&) {delete}
+ operator=(LanguageDefinition&&) {delete}

+ GetDirective(const string&) const: const ASMDirective* {inline}
+ TryGetDirective(const string&, const ASMDirective*&) const: bool {inline}
+ ContainsDirective(const string&) const: bool {inline}

+ GetInstruction(const string&) const: const ASMInstruction* {inline}
+ TryGetInstruction(const string&, const ASMInstruction*&) const: bool {inline}
+ ContainsInstruction(const string&) const: bool {inline}

+ ContainsKeyword(const string&) const: bool {inline}

template<std::integral TargetType>
+ GetRegisterID(const string&) const: TargetType {inline}

Instruction represents a language supported instruction mnemonic and what the assembler should do when encountering that mnemonic/instruction

All "Instructions" are realizations of *ASMInstruction*

ASMInstruction

+ mnemonic: string
# registerPattern: const regex {static}
# decimalPattern: const regex {static}
# hexPattern: const regex {static}
# asciiPattern: const regex {static}
# asciiEscape: const regex {static}

+ Implementation(vector<uint8_t>&, const Workpiece* const&, const LanguageDefinition* const&, const vector<string>&) const: size_t {virtual}

ASMInstruction(string&&)

GetLabelOffset(const string&, Workpiece* const&): size_t& {static} {inline}

template<std::integral TargetType>
AddUnresolvedLabel(const string&, vector<uint8_t>&, Workpiece* const&): void {static} {inline}

template<std::integral TargetType, std::integral... InputType>
SerializeToBuffer(vector<uint8_t>&, const InputType&...): void {static} {inline}

template<std::integral TargetType>
GetImmediateValue(const string&): TargetType {static} {inline}

Note: SetDirectives uses concept (c++ 20) called DerivedFromDirective to create a variadic parameter to allow passing in any number of Directiv(s) as constructor arguments. DerivedFromDirective is defined as follows.
template<typename T>
concept DerivedFromDirective = std::derived_from<T, ASMDirective>;

Note: SetInstructions uses concept (c++ 20) called DerivedFromInstruction to create a variadic parameter to allow passing in any number of Instruction(s) as constructor arguments. DerivedFromInstruction is defined as follows.
template<typename T>
concept DerivedFromInstruction = std::derived_from<T, ASMInstruction>;

Note: SetKeywords and SetregisterMnemonics uses concept (c++ 20) called StringType to create a variadic parameter to allow passing in any number of Instruction(s) as constructor arguments. StringType is defined as follows.
template<typename T>
concept StringType = std::is_same_v<T, Instruction>;

Derived: typename

SingletonWrapper

s_instance: shared_ptr<Derived> {inline} {static}
_sharedMutex: shared_mutex {inline} {static}

+ GetInstance(): shared_ptr<Derived> {static}
+ ShutDown(): void {virtual}

+ SingletonWrapper(const SingletonWrapper&) {delete}
+ operator=(const SingletonWrapper&) {delete}
+ SingletonWrapper(SingletonWrapper&&) {delete}
+ operator=(SingletonWrapper&&) {delete}

The Assembler can be thought of as an assebmly line. It contains a collection of "Passes" that work upon the "Workpiece" until all the work needing to be done has been conpleted and the finished product has emerged

Note: Assembler derived types will be singleton, deriving also from SingletonWrapper

Assembler

m_passes: const vector<Pass*>

m_workpiece: Workpiece*

m_filePath: const filesystem::path

_languageSpec: const LanguageDefinition*

Assembler(Workpiece*, const LanguageDefinition*, const DerivedFromPass auto*...)
~Assembler()

+ ProcessASM(const char*): void {virtual}
+ ProcessASM(filesystem::path&): void {virtual}

+ Assembler(const Assembler&) {delete}
+ operator=(const Assembler&) {delete}
+ Assembler(Assembler&&) {delete}
+ operator=(Assembler&&) {delete}

Note: Constructor uses concept (c++ 20) called DerivedFromPass to create a variadic parameter to allow passing in any number of Pass(s) as constructor arguments. DerivedFromPass is defined as follows.
template<typename T>
concept DerivedFromPass = std::derived_from<T, Pass>;

The Workpiece is the program under assembly. It can be though of as a "*workpiece*" traveling through an assembly line from start to finish. Each pass preforming some work on the piece and passign it on to the next stage of the line

Workpiece

+ _sharedMutex: shared_mutex

+ _symbolTable: unordered_map<string, size_t>
+ _unresolvedLables: unordered_map<string, pair<vector<uint8_t>*, size_t>>>

+ _dataSegmentItems: vector<std::pair<string, DataSegmentItem>>
+ _dataSegmentBin: vector<uint8_t>

+ _codeSegmentItems: vector<std::pair<string, vector<CodeSegmentItem>>>
+ _codeSegmentBins: list<std::pair<<string, vector<uint8_t>>>

DataSegmentItem

+ _lineNum: size_t
+ _directive: const ASMDirective*
+ _dirArgs: vector<string>
+ _lineComment: string

CodeSegmentItem

+ _lineNum: size_t
+ _instruction: const ASMInstruction*
+ _instrArgs: vector<string>
+ _lineComment: string

A Pass should be though of as a station, or section, of an assembly line to preform some work on the workpiece. This can be as granular as needed

Pass

+ Execute(Workpiece* const&, const filesystem::path&, const LanguageDefinition* const&) const: void {virtual}

All "Passes" are realizations of *Pass*

Directives

BYT_DIR
CSTR_DIR
INT_DIR
STR_DIR

Instructions

ADD_INS | ADI_INS | DIVI_INS | DIV_INS | MULI_INS
MUL_INS | SUB_INS | CMPI_INS | CMP_INS | ALCI_INS
ALLC_INS | FREE_INS | BGT_INS | BLT_INS | BNZ_INS
BRZ_INS | JMP_INS | JMR_INS | AND_INS | NOT_INS
OR_INS | LDA_INS | LDB_INS | LDR_INS | MOVI_INS
MOV_INS | STB_INS | STR_INS | BLK_INS | END_INS
LCK_INS | RUN_INS | ULK_INS | TRP_INS

ASM_4380

All concrete "Instructions" are contained within *LanguageDefinition_4380*

All concrete "Directives" are contained within *LanguageDefinition_4380*

LanguageDefinition_4380

- *all directives, and all instructions*

ASM4380

- _passOne: PassOne_Tokenization
- _passTwo: PassTwo_Assemble
- _passThree: PassThree_Serialize

- _workpiece: Workpiece

- _languageDef: LanguageDefinition_4380

+ _trapZero: bool {static}

+ ShutDown: void {override}
+ ProcessASM(const char*): void {override}
+ ProcessASM(filesystem::path&&): void {override}

All concrete "Passes" are contained within *ASM4380*

Passes

PassOne_Tokenization

+ Execute(Workpiece* const&, const filesystem::path&, const LanguageDefinition* const&) const: void {override}

- AddLabel(const string&, Workpiece* const&)const: void

- ProcessDataSegment(size_t&, Workpiece* const&, ifstream&, const LanguageDefinition* const&) const: void

- HandleDirective(const size_t&, string&&, const string&, const vector<string>&, string&&, Workpiece* const&, const LanguageDefinition* const&) const: void

- ProcessCodeSegment(size_t&, Workpiece* const&, ifstream&, const LanguageDefinition* const&) const: void

- HandleInstruction(const size_t&, const string&, const vector<string>&, string&&, vector<CodeSegmentItem>* const&, const LanguageDefinition* const&) const: void

- HandleCodeSegmentItem(const string&, const vector<CodeSegmentItem>&, Workpiece* const&) const: void

PassTwo_Assemble

- _fail: bool {mutable}

+ Execute(Workpiece* const&, const filesystem::path&, const LanguageDefinition* const&) const: void {override}

- ProcessDataSegment(size_t&, Workpiece* const&, const LanguageDefinition* const&) const: void
- ProcessCodeSegment(size_t&, Workpiece* const&, const LanguageDefinition* const&) const: void
- ProcessUnresolvedLabels(Workpiece* const&) const: void

PassThree_Serialize

+ Execute(Workpiece* const&, const filesystem::path&, const LanguageDefinition* const&) const: void {override}

- SerializeInitialPC(ofstream&, Workpiece* const&) const: size_t
- SerializeDataSegment(ofstream&, Workpiece* const&) const: size_t
- SerializeCodeSegment(ofstream&, Workpiece* const&) const: size_t