VTX1 Assembler Documentation

Table of Contents

Chapter 1. Introduction

This document provides comprehensive documentation for the VTX1 assembler, which is designed to convert assembly language code into binary machine code for the VTX1 ternary processor.

The VTX1 assembler supports the complete instruction set of the VTX1 architecture, including VLIW operations with parallel execution capabilities, balanced ternary numeric representation, and various addressing modes.

Chapter 2. Assembler Components

Chapter 3. VTX1 Assembler Architecture

3.1. Overview

The VTX1 assembler is designed as a multi-pass assembler to convert VTX1 assembly language into binary machine code for the VTX1 ternary processor. This document outlines the architecture and design principles of the assembler.

3.2. Architecture Components

3.2.1. 1. Lexer (Tokenization)

The lexer processes the input source code and converts it into a stream of tokens. Each token represents a meaningful element in the assembly language such as:

- Mnemonics (e.g., ADD, SUB, LD)
- Registers (e.g., T0, TA, VA)
- Literals (decimal, binary, ternary)
- Labels and symbols
- Directives and special characters

3.2.2. 2. Parser

The parser processes the token stream and builds an abstract syntax tree (AST) that represents the structure of the program. It handles:

- Instruction recognition
- Operand validation
- VLIW grouping for parallel operations
- Directive processing
- Label definition and usage

3.2.3. 3. Symbol Table Manager

Manages symbol resolution including:

- Label definitions and references
- Global and local symbols
- Address calculation
- Forward references

3.2.4. 4. Code Generator

Converts the parsed instructions into binary code according to the VTX1 instruction encoding:

- 96-bit VLIW instruction words (3 operations per word)
- 36-bit ternary word encoding

- Balanced ternary literal encoding
- Address resolution

3.2.5. 5. Error Handler

Provides comprehensive error detection and reporting:

- Syntax errors
- Semantic errors (e.g., invalid register usage)
- Symbol resolution errors
- Range and value validation

3.2.6. 6. Output Formatter

Generates the final output in the required format:

- Binary file format
- Hexadecimal listing
- Debug information
- Symbol table output

3.3. Processing Flow

The assembler operates in multiple passes:

1. **Pass 1**: Lexical analysis and parsing, build initial symbol table
2. **Pass 2**: Resolve symbols and calculate addresses
3. **Pass 3**: Generate code and produce binary output

3.4. Implementation Choices

3.4.1. Programming Language

The initial prototype will be implemented in Python for rapid development and ease of maintenance. Key considerations:

- Excellent string processing capabilities
- Rich library ecosystem
- Easy to extend and modify
- Good performance for an assembler of this scale
- Cross-platform compatibility

3.4.2. Error Handling Strategy

The assembler will use a comprehensive error handling strategy:

- Continue parsing after errors when possible

- Collect multiple errors before halting
- Provide clear error messages with line/column information
- Suggest potential fixes when appropriate

3.4.3. VLIW Handling

VLIW instructions (multiple operations per instruction word) will be handled through:

- Explicit grouping syntax in the assembly language
- Automatic validation of illegal operation combinations
- Resource conflict detection
- Pipeline hazard warnings

Chapter 4. Assembly Language

Chapter 5. VTX1 Assembly Language Instruction Syntax

5.1. Overview

This document describes the syntax for VTX1 assembly language instructions. The VTX1 assembler supports a sophisticated instruction set with VLIW (Very Long Instruction Word) capabilities that allow for parallel execution of multiple operations.

5.2. Basic Instruction Format

Individual instructions follow this general format:

```
[label:] mnemonic [operand1][, operand2][, operand3] [; comment]
```

Where:

- `label` (optional): A symbol used to reference a memory location
- `mnemonic`: The instruction name (e.g., ADD, LD, JMP)
- `operand1, operand2, operand3` (optional, instruction-dependent): Registers, literals, or symbols
- `comment` (optional): Text following a semicolon, ignored by the assembler

5.3. VLIW Instruction Format

The VTX1 supports VLIW instructions with up to 3 operations executed in parallel. VLIW instructions use the following syntax:

```
[op1] [op2] [op3]
```

Where each operation follows the basic instruction format. Operations are enclosed in square brackets and separated by spaces.

Example:

```
[ADD T0, T1, T2] [LD T3, [TB+1]] [NOP]
```

5.4. Register Naming

The VTX1 supports several register types:

1. **General Purpose Registers (GPRs)**: T0-T6
2. **Special Registers**:
   - TA: Accumulator
   - TB: Base Pointer

- TC: Program Counter
- TS: Status Register
- TI: Instruction Register

3. **Vector Registers**: VA, VT, VB
4. **Floating-Point Registers**: FA, FT, FB

5.5. Addressing Modes

The VTX1 supports the following addressing modes:

1. **Register Direct**: The operand is a register

```
ADD T0, T1, T2  ; T0 = T1 + T2
```

2. **Immediate**: The operand is a constant value

```
ADD T0, T1, 5   ; T0 = T1 + 5
```

3. **Base + Offset**: The operand is a memory location specified by a base register plus offset

```
LD T0, [TB+4]   ; T0 = Memory[TB+4]
```

4. **Program Counter Relative**: The operand is an address relative to the current program counter

```
JMP label       ; PC = label_address
```

5. **Vector Indexed**: The operand is a vector register with an index

```
VLD VA, [TB+T0] ; VA = Memory[TB+T0]
```

5.6. Literal Formats

The VTX1 supports multiple literal formats:

1. **Decimal**: Standard decimal notation

```
ADD T0, T1, 42  ; Decimal 42
```

2. **Hexadecimal**: Prefixed with `0x`

```
ADD T0, T1, 0x2A  ; Hexadecimal 2A (42 decimal)
```

3. **Binary**: Prefixed with `0b`

```
ADD T0, T1, 0b101010  ; Binary 101010 (42 decimal)
```

4. **Ternary**: Prefixed with `0t` (balanced ternary notation using `-`, `0`, `+`)

```
ADD T0, T1, 0t+--+0  ; Balanced ternary +--+0 (42 decimal)
```

5.7. Operation Categories

The VTX1 instruction set is organized into the following categories:

1. **ALU Operations**: ADD, SUB, MUL, AND, OR, NOT, etc.
2. **Memory Operations**: LD, ST, VLD, VST, FLD, FST, etc.
3. **Control Operations**: JMP, JAL, BEQ, BNE, CALL, RET, etc.
4. **Vector Operations**: VADD, VSUB, VMUL, VAND, VOR, etc.
5. **Floating-Point Operations**: FADD, FSUB, FMUL, etc.
6. **System Operations**: NOP, WFI, etc.

5.8. Example Assembly Code

```
; Simple VTX1 assembly program example
        LD T0, 0x1000        ; Load value from address 0x1000 into T0
        LD T1, 0x1004        ; Load value from address 0x1004 into T1
loop:   ADD T2, T0, T1        ; T2 = T0 + T1
        [ADD T0, T1, 0] [SUB T1, T2, T0] [NOP]  ; VLIW instruction with 3 operations
        BNE T0, 0, loop       ; Branch to loop if T0 != 0
        ST T2, 0x1008        ; Store result to address 0x1008
        WFI                  ; Wait for interrupt
```

Chapter 6. Instruction Encoding

Chapter 7. VTX1 VLIW Instruction Encoding

7.1. Overview

The VTX1 processor implements a VLIW (Very Long Instruction Word) architecture that allows multiple operations to be executed in parallel. This document describes the encoding format for VLIW instructions in the VTX1 assembler.

7.2. VLIW Instruction Format

Each VTX1 VLIW instruction is 96 bits wide and can contain up to 3 operations (32 bits each):

```
+----------------------------------+----------------------------------
+----------------------------------+
|          Operation 1 (32 bits)    |          Operation 2 (32 bits)    |          Operation 3 (32
bits)    |
+----------------------------------+----------------------------------
+----------------------------------+
```

7.3. Operation Field Encoding

Each operation field is encoded as follows:

```
+--------+------+------+------+--------------------+------+------+
| Opcode | Reg1 | Reg2 | Reg3 | Immediate/Offset   | Type | Par  |
+--------+------+------+------+--------------------+------+------+
| 6 bits | 3b   | 3b   | 3b   | 11 bits            | 3b   | 3b   |
+--------+------+------+------+--------------------+------+------+
```

Where:

- **Opcode** (6 bits): Instruction operation code
- **Reg1, Reg2, Reg3** (3 bits each): Register specifiers
- **Immediate/Offset** (11 bits): Immediate value or address offset
- **Type** (3 bits): Operation type (ALU, Memory, Control, Vector, FPU, System)
- **Par** (3 bits): Parallel execution flags

7.4. Operation Type Encoding

The "Type" field uses the following encoding:

- `000`: ALU Operation
- `001`: Memory Operation
- `002`: Control Operation
- `003`: Vector Operation
- `004`: FPU Operation

- `005`: System Operation
- `006`: Microcode Operation
- `007`: Reserved

7.5. Parallel Flags Encoding

The "Par" field uses the following encoding:

- `000`: Serial Execution
- `001`: Parallel with ALU
- `002`: Parallel with Memory
- `003`: Parallel with Control
- `004`: Full Parallel
- `005-007`: Reserved

7.6. Binary Representation

In memory, the VLIW instruction is stored in little-endian format (lower address bits first).

Example encoding for the VLIW instruction `[ADD T0, T1, T2] [LD T3, [TB+1]] [NOP]`:

```
+--------+------+------+------+--------------------+------+------+
| 000001 | 000  | 001  | 010  | 00000000000        | 000  | 004  | Operation 1: ADD T0, T1, T2
+--------+------+------+------+--------------------+------+------+
| 010000 | 011  | 100  | 000  | 00000000001        | 001  | 001  | Operation 2: LD T3, [TB+1]
+--------+------+------+------+--------------------+------+------+
| 000000 | 000  | 000  | 000  | 00000000000        | 005  | 000  | Operation 3: NOP
+--------+------+------+------+--------------------+------+------+
```

7.7. Assembler Representation

In assembly language, VLIW instructions are represented by enclosing operations in square brackets:

```
[ADD T0, T1, T2] [LD T3, [TB+1]] [NOP]
```

7.8. Instruction Packing Rules

Not all operations can be combined in a VLIW instruction. The assembler enforces the following rules:

1. Maximum of 3 operations per VLIW instruction
2. Maximum of 1 memory operation per VLIW instruction
3. Maximum of 1 control operation per VLIW instruction
4. No register conflicts between operations
5. No pipeline hazards between operations

7.9. Register Encoding

Registers are encoded as follows:

- `T0`-`T6`: `000`-`110` (General Purpose Registers)
- `TA`: `111` (Accumulator)
- `TB`: `000` (Base Pointer, accessed via type field)
- `TC`: `001` (Program Counter, accessed via type field)
- `TS`: `010` (Status Register, accessed via type field)
- `TI`: `011` (Instruction Register, accessed via type field)
- `VA`, `VT`, `VB`: Accessed via vector operations
- `FA`, `FT`, `FB`: Accessed via FPU operations

7.10. Error Handling

The assembler will report errors if:

1. A VLIW instruction contains more than 3 operations
2. Operations within a VLIW instruction have conflicts
3. Register usage violates architectural constraints
4. Parallel execution flags are incompatible

Chapter 8. Appendix

8.1. References

- VTX1 CPU Architecture Documentation
- VTX1 Instruction Set Reference
- Balanced Ternary Logic Implementation Guide

8.2. Version History

Version	Date	Changes
0.1.0	2025-06-16	Initial documentation structure