TILE

Threaded • Interpretive • Language • Environment

St. Andrews University

Department of Computational Science

Senior Honours Project 1989-90

Implementation & Documentation by

Steven James

University of St. Andrews,

North Haugh,

St. Andrews,

Fife.

Preface

This manuscript is the outcome of a sequence of events that began in October 1989. It was originally intended that the honours project should take the form of a major programming assignment undertaken by a team of senior student working over a period of two terms. However due to unforeseen circumstances the original team broken up after only one term with little in the way of completed work. The remaining term was spent designing and implementing the threaded interpretive language outlined in this document.

The documentation contained herein is therefore not as extensive as could have been expected had the project run its normal course. It had originally been my intention in include a full project report and user manual, however the pressure of closing deadlines has regrettably forced me to omit these section leaving only those parts which I consider essential. Because of this decision I am now inclined to suggest that the documentation is aimed at the reader with some knowledge of a concepts and techniques underlying the design of a threaded interpretive language and not the novice.

Fortunately the deign and implementation is complete and stands as example of how much can be accomplished by one person in so short a time. In addition to the standard language I also though it necessary to incorporate the source code of some working examples of real applications. For this reason the source listing section contains a set of language extensions including such constructs as vectors, abstract data structures and a case statement as well as a complete and working basic compiler.

Table of Contents

1

Preliminary Specification

1.1 Introduction

The preliminary specification of a threaded interpretive language (hereafter refereed to as a **TIL**) is far less critical in terms of the influence it will have over the eventual shape of the language, than would be the case were it for a more conventional language such as **PASCAL** or **S-algol**. This is in part the result of a design decision that allows a **TIL** to be extended and retracted to such a degree that it no longer resembles or has any relation to its initial form. The specification is never the less an import and essential part of the design process, in that it provides an initial description of the project as a whole, and gives direction to the forth coming design effort.

1.2 Overview

The initial specification was to design and implement a **TIL** drawing upon the ideas and concepts from existing thread interpretive languages such as **FORTH** and **IPS**, while maintaining a high degree of independence from both, by incorporating many new ideas and implementation techniques.

1.3 Design Considerations

The specification of any large software project can seen to consist of two separate and distinct phases. The initial phase is concerned with the overall design considerations for the project as a whole, with little or no attention to specific design issues or fine detail. The design considerations for a **TIL** or indeed any computer language fall naturally into two categories.

1.3.1 Language Elements

A **TIL** can be described in terms of a collection of functionally dependent modules, each representing one unique aspect of the language.

1.3.1.1 The Interpreter

The interpreter is the primary interface between the user and the abstract machine. Broadly speaking its function can be described as the mapping of

lexemes for an input stream via the dictionary structure to fragments of abstract machine code. The source of such lexemes can be either the terminal keyboard or a file.

1.3.1.2 The Abstract Machine

The abstract machine is the underlying execution mechanism, and can be view as the very essence of a **TIL**. Its structure is more or less fixed, and is dependent upon the type of treaded execution implemented. The instruction set of the abstract machine is based around the functions provided by the **TIL**, each primitive within the language represents a single abstract machine operation.

1.3.1.3 The Dictionary

The dictionary provides the primary heap management system for a **TIL**. Its structure is hierarchically based around a system of linked lists organised into a tree structure, each leaf node representing a separate list, or vocabulary of secondary function definitions.

1.3.2 Language Constructs

Having described the elements of **TIL**, it now remains to describe the constructs supported by those elements.

1.3.2.1 Arithmetic

The arithmetic is based upon single length (16 bit), and double length (32 bit) signed integers. Elementary Arithmetic operations are provided to perform single and double length addition and subtraction, and single length multiplication and division.

1.3.2.2 Logic

The logic is based upon single length (16 bit) bitwise operations, and single length (16 bit) comparative operations. Elementary bitwise operations are provided to perform **and**, **or** , **xor**, and **not**. together with the equality operations (=) equal, (<) small than and (>) greater than.

1.3.2.3 Stack Manipulation

The stack manipulation is based upon the four elementary stack operations **swap**, **dup**, **over**, and **rot** .

1.3.2.4 I/O

The I/O is based around elementary stream manipulation operations, and provides a stream selection operation, and read and write character operations upon the selected stream.

1.3.2.5 Memory Manipulation

The memory manipulation is based around the storage and retrieval of byte quantities from 64K of addressable memory. single length (16 bit) operations may be implemented with two byte operations.

1.3.2.6 Dictionary Manipulation

The dictionary manipulation is based around the creation and subsequent location of dictionary headers and associated data.

1.3.2.7 Numeric Manipulation

The numeric manipulation is based around the **constant** and **literal** operations upon single (16 bit) integers.

1.3.2.8 String Manipulation

The string manipulation is based around the parsing of lexemes from the input stream, and the subsequent conversion of such lexemes into their integer equivalent.

1.3.2.9 Flow Control

The flow control is based around conditional and unconditional, forward and backward branch operations.

1.4 Implementation Considerations

The final phase of the specification process is concerned with the implementation considerations for the project as a whole, with little or not attention to to specific implantation issues or fine detail. The implementation considerations for a thread interpretive language fall naturally into two categories.

1.4.1 Hardware

The hardware considerations are primarily concerned with the selection of implementation hardware. The economics of departmental funding limit the choice to one of two machines.

1.4.1.1 Apple Macintosh Plus

The Apple Macintosh is an extremely successful microcomputer with an outstanding user interface, but a non standard operating system. The basic hardware comprises of a Motorola 68000 microprocessor and 1 Megabyte of ram. Peripheral are available via the Apple Talk™ network.

1.4.1.2 Sun 3/60

The Sun 3/60 is an integrated development environment with an outstanding user interface, and standard unix™ operating system. The basic hardware comprises of a Motorola 68020 microprocessor and 8 Megabytes of internal ram, and an indefinite amount of virtual memory. Peripherals are available via the Sun Eithernet™ network.

The **Sun 3/60** promises both better performance, and a greater degree of portability, with the additional advantage of the support of a major operating system.

1.4.2 Software

The software considerations are primarily concerned with the selection of an implementation language. The possible alternatives can be short listed to a choice of just two, **ANSI 'C'** and **PS-Algol**.

1.4.2.1 ANSI 'C'

ANSI 'C' is rapidly establishing itself as the standard systems programming language. It offers both efficient execution, and a high degree of portability between machines and operating systems.

1.4.2.2 PS-Algol

PS-Algol is relatively new programming language little known outside the university environment. It offers many outstanding feature such as persistent database management and first class procedures, but suffers in terms of execution and portability because of its relative obscurity.

ANSI 'C' promises both better performance and a greater degree of portability than **PS-Algol**. There also exist a large number software tools to aid in the development and debugging of **'C'** programs.

1.5 Summary

To recap, the project is to design and implement a **TIL** based around an interpreter to parse lexemes from the input stream, a dictionary structure to map lexemes to abstract machine code fragments, and an abstract machine to execute the code fragments. The project will be implemented upon a **Sun 3/60** using **ANSI 'C'**.

2

The TILE Dictionary

2.1 Introduction

The dictionary constitutes the largest single data structure used within a **TIL**. It not only provides the memory management mechanism, but is also an essential and integral part of the interpreter. Its function with respect to the interpreter, can be broadly described as the mapping of lexemes from the input stream, to machine code fragments and data, stored within the dictionary structure.

2.2 Dictionary Structure

The dictionary is based around a linked list, with each dictionary entry as a node within the list structure. Each dictionary entry consists of two parts, the head and the body. The head of the entry contains some information about the name of the word (**the name field**), a pointer to the previous word in the dictionary (**the link field**) and a pointer to the machine code associated with the word (**the code field**). Figure 2.2.1 show two typical dictionary entries, a code primitive (left) and a secondary (right). The name field address is prefixed with a header byte, containing information related to the nature of the word. The individual bits within the byte are used for different purposes by the interpreter. Bits 1 to 6 inclusive contain the length in bytes of the word name (up to a maximum size of 63 bytes), bit 7 indicates whether or not the word can be found via a dictionary search (used to hide words during compilation) and bit 8 indicates if the word is immediate (executable during compilation). Figure 2.2.2 shows the bit-by-bit format of the header byte.

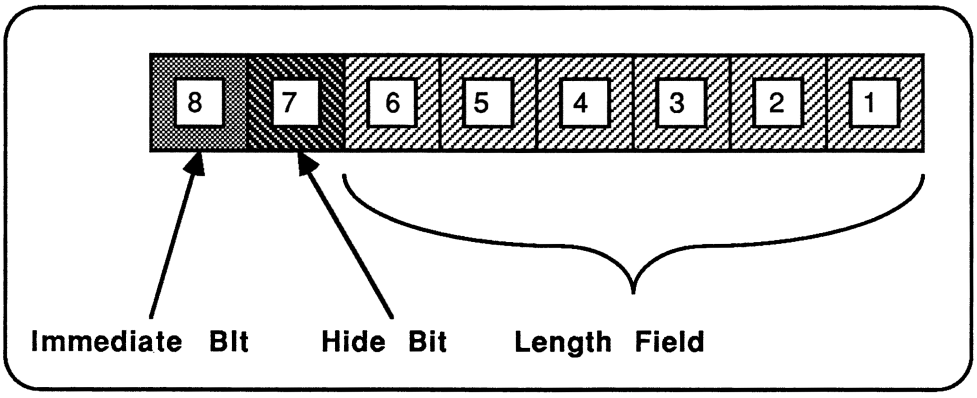

Figure 2.2.2 - Diagram of the header byte.