# A stack-based RISC architecture for control applications

by Timothy Dale Thompson

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering
Montana State University


Abstract:
Automatic control systems are assuming an increasingly important role in the advancement of modern civilization and technology. This paper focuses on the development of a very simple stack-based computer architecture tailored to execute a core subset of the Forth programming language, with consideration toward use in real-time control applications.

The instruction set developed is composed of 27 instructions, which operate on a processor with a dual-stack architecture that has a datapath width of 16 bits. Instruction op-codes are 16 bits wide. The limitations of this instruction set prevent the processor from being used in applications when extensive floating point arithmetic calculations are necessary, but the instruction set is adequate for processor applications such as direct memory access control.

# A STACK-BASED RISC ARCHITECTURE FOR CONTROL APPLICATIONS

by

Timothy Dale Thompson

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Electrical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

December 1991

APPROVAL

of a thesis submitted by

Timothy Dale Thompson

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

11/18/91
Date

Chairperson, Graduate Committee

Approved for the Major Department

11/26/91
Date

D. L. Price for V. Lorenz
Head, Major Department

Approved for the College of Graduate Studies

December 17, 1991
Date

Graduate Dean

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his/her absence, by the Dean of Libraries when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Timothy D. Thompson

Date 11/26/91

ACKNOWLEDGEMENTS

I would like to thank the members of my committee: Kel Winters, Roy Johnson and Harley Leach, for their guidance during my graduate work. I would also like to thank Bob Wall and Diane Mathews for their help in developing this thesis. Finally, thanks to Jaye Mathisen for his help with UNIX and the computer system.

TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

## ABSTRACT

Automatic control systems are assuming an increasingly important role in the advancement of modern civilization and technology. This paper focuses on the development of a very simple stack-based computer architecture tailored to execute a core subset of the Forth programming language, with consideration toward use in real-time control applications.

The instruction set developed is composed of 27 instructions, which operate on a processor with a dual-stack architecture that has a datapath width of 16 bits. Instruction op-codes are 16 bits wide. The limitations of this instruction set prevent the processor from being used in applications when extensive floating point arithmetic calculations are necessary, but the instruction set is adequate for processor applications such as direct memory access control.

# CHAPTER 1

## INTRODUCTION

Automatic control systems are assuming an increasingly important role in the advancement of modern civilization and technology. The phrase automatic control system is self explanatory: the word system implies not just one component but a number of components which work together to achieve a particular goal; that goal is the control of some physical quantity; and the control is to be accomplished in an automatic fashion, without the aid of human supervision.

Practically every aspect of everyone's day-to-day activities is affected by some type of control system. For example, automatic controls in heating and air-conditioning systems regulate the temperature of office buildings and homes. There are control systems in robotics, power systems, weapon systems, aircraft, automobiles, washers and dryers, toasters, and many other common items. Control systems are an integral component of any industrial society, and are necessary for the production of goods required by the people of the world.

### Types of Control Systems

Control systems can be separated into two classes: open-loop control systems, and closed-loop control systems. An

open-loop control system is characterized by the fact that the output quantity has no effect upon the input quantity. An example of an open-loop control system is the toaster. In this case the input quantity is the timer which controls the length of time that heat is applied. The output quantity, which is the darkness of the toast, can in no way alter the length of time that heat is applied. Open-loop control systems have practical use in numerous situations, due to their simplicity and economy.

Although open-loop control systems have many applications, in a large number of systems there is a need for more accurate and adaptive control. A necessary factor in this case is the feedback of the output of the system to the input of the system, where it is compared to the desired output value and adjusted accordingly. This type of control system, where the output has an effect on the input quantity, is known as a closed-loop control system. The thermostat in a home or office building is a common example of a closed-loop control system. A coil in the thermostat is affected by the desired temperature setting and the actual room temperature. If the room temperature is less than the chosen temperature, the coil changes shape, which in turn activates a relay, which causes the furnace to produce heat in the room. When the room reaches the desired temperature, the coil again changes shape, deactivating the relay, which turns off the furnace. Most industrial control systems are closed-loop control systems,

since the quality of a manufactured product is dependent on the accuracy of the systems which are used to fabricate it.[D'Azzo88]

A control system can be implemented as an analog system, or as a digital system. An analog signal is a continuous and time varying signal which describes most physical quantities such as light or sound. In a control system, these quantities are usually represented by a time varying voltage. A digital system represents continuous values, to a given accuracy, as digital values. A digital system uses binary numbers, or a base 2 counting system, to represent digital quantities, since the binary states 1 and 0 are easily represented by digital electronic devices.

Analog electronic systems are typically made up of discrete transistors, analog integrated circuits, and resistors, capacitors, and inductors, interconnected to perform a specific function. Analog systems are often used when high-frequency signals must be processed, or when simple functions such as signal amplification, comparison, or filtering are required.

Digital systems can be separated into two classes: dedicated digital systems, and computer-based systems. Dedicated digital systems are often used when rapid response to inputs is necessary, but the processing of analog signals is not necessary. Digital logic elements, such as combinational logic gates and storage elements, and the way

they are interconnected define the function of a dedicated digital system. These systems are high speed, but the application must be simple, and function of the system must not be expected to change, since changing the function of the system would require that the system be redesigned. Often a digital system is composed of discrete components, but for high performance applications, custom VLSI (very large scale integrated) circuits are usually required.

Although the preceding two types of systems are characterized by high performance, they lack flexibility. Computer-based control systems have the advantage that they can be designed for a general case, and then programmed for a specific task. This allows mass production of these systems, which reduces the cost of the systems, and makes them more available for use by system designers. Programs can be easily revised during the development cycle of the system, and later, while the system is in use, the program can be modified to adjust to a changing environment. Another advantage of computer-based systems is that they are compatible with both analog and digital inputs and outputs. The flexibility in a computer-based control system does not come without some functional cost. Operation of this type of system is slower than that of hard wired analog and dedicated digital systems. In many cases, however, the speed of a computer-based system is sufficient, and the number of applications continues to increase. Other factors in the increased use of this type of

system include speed improvements in microprocessors, special purpose architectures, cost reductions in conjunction with increased functionality, ease of testing, and the growing number of designers in industry who are familiar with microcomputer technology.[Lawrence87]

## Real-Time Control Systems

An important class of computer-based control systems is known as real-time control systems. A real-time system is defined as any information processing activity or system which has to respond to externally-generated input stimuli within a finite and specified period.[Young82] The lag time between an input signal and the necessary output changes greatly as applications vary. In some systems, a failure to respond to an input signal can be tolerated, but in others, a failure to respond can be as bad as an incorrect response. This leads to the conclusion that the correctness of the response of a real time system depends not only on the result of processing the input data, but also on the time at which the result is produced. To distinguish between systems in which a failure to respond can be accepted and those in which a failure to respond may be disastrous, real-time systems may be classified as hard real-time systems or soft real-time systems. Hard real-time systems are those in which a response within a given time window is critical, while soft real-time systems are those in which response times are important, but the system can still function if a response time deadline is missed.

Another important requirement for a real time system is that it must have a high degree of reliability. Since many computer-based systems operate without human supervision, it is important that if a failure occurs, the failure does not have destructive effects on the system being controlled. The system should detect the failure and execute a controlled recovery.[Burns90]

## System Hardware and Software

A real-time computer-based control system is a combination of hardware and software. There is little to distinguish real-time computer system hardware from that of a general purpose computer system, but in addition to the computer hardware there are other hardware components necessary for interaction with the world in a real-time system. Input signals to the system are furnished by sensors, which are devices that produce an output signal, usually an analog voltage level, which is a function of the change in a physical variable being represented.

Once a change is sensed and represented as an analog signal, it may be necessary to modify the signal so that it may be sampled and interfaced with the processor. Signal conditioning includes the matching of sensor output voltages to voltages acceptable by input interfaces, filtering of unwanted high frequency or power supply noise in the signal, conversion of signal forms, such as the conversion of an analog signal into a series of pulses for transmission in

which the frequency or duration of the pulses is proportional to the sensor voltage level, and electrical isolation to prevent noise and voltage fluctuations which may damage the system.

The next stage, if one is following a signal produced by an event through the control system from the time it is sensed until a correcting signal is produced, is the computer input stage. There are two types of computer interfaces: analog input interfaces, and digital input interfaces. In an analog interface, the analog signal is passed through an analog-to-digital (A/D) converter. An A/D converter samples the analog input voltage at one instant of time and converts it into an equivalent digital value. The accuracy of the digital value is dependent upon the number of binary digits (bits) used. For example, an A/D converter which has a 16 bit resolution can represent $2^{16}$ = 65,536 different values between the minimum and maximum input voltages allowed. A digital input interface is much simpler. Since the input data is already in binary form, it can simply be transferred directly into the computer when the information is needed.

The most important part of a real-time control system is the processor. In the broadest sense, the processor acts on the information that it receives from the input interfaces. This involves fetching the instructions of the controller program from memory and executing them, fetching data from memory and input interfaces, writing data to memory and output

interfaces, and responding to external requests for action.

Output interfaces also can be separated into two categories. Analog output interfaces take a digital value, and by means of an digital-to-analog (D/A) converter, generate an analog output signal. A 16-bit D/A converter with output voltage range of 0 - 15 Volts has 65,536 possible levels of output voltage in that range, which gives a resolution of approximately $2.3 \times 10^{-4}$ volts. Digital output interfaces are again very simple. The output data is written to a storage register, which drives the output signal lines, and the data remains on those lines until new data is written to the register.

It may be necessary to perform some conditioning on the output signals, which is similar to the conditioning done on input signals, before they reach the devices that the signals are controlling. Often the devices that are controlled are actuators. An actuator is any device which produces a motion. Electric actuators include relays, motors, and solenoids. Non-electric actuators, such as hydraulic rams or pneumatic devices, must have some sort of electrical interface in order for the devices to be computer-controlled.[Savitzky85]

The other part of a real-time computer-based system is the system software. The basic requirement for real-time software is that it must execute quickly and not take up much storage space. Computer languages can be divided into two classes: assembly languages, and high-level languages. The

most optimal software for real-time systems is written in the assembly language of the processor which is the core of the system. Assembly language is a method of programming a processor in which the instructions, the actual binary input bit patterns, that the processor will execute directly are given alphabetic names. Using assembly language allows the programmer to take advantage of every time-saving trick that the machine allows, and this means that the software will execute quicker than if it is programmed in another language. Assembly language programs execute approximately twice as fast as programs written in high-level languages.[Lawrence87] The main problem with assembly language is that programming is time consuming and difficult, and requires a programmer who is familiar with the particular processor being used to produce quality programs.

High-level languages are programming languages for which the statements do not have a simple, direct mapping into the internal binary representation of the processor's instructions, but are instead closer to the terms in which the user thinks of a problem. It is much easier to write programs in a high-level language, since a high-level language will support a variety of control and data structures without consideration of the processor on which the program will be executed. This allows the program development cycle to be shortened. Programs written in a high-level language are converted into assembly language by other programs known as

compilers or interpreters. A compiler converts the whole program into an executable assembly language program, while an interpreter decodes a single instruction, executes it, and then continues on to the next instruction. For this reason, interpreted languages are generally slow, but are very good for program development. The sophistication of the conversion program will drastically affect the size and efficiency of the assembly language generated, and hence impact the speed that the program will execute. Unfortunately, the speed that the program will execute is very important in real-time systems.

Often, real-time systems take advantage of both high-level and assembly languages. Time critical portions of the software are written in assembly code, while non-critical portions are written in high-level language and compiled into assembly code. This allows software to be developed rapidly, while still meeting timing requirements. The main problem with this approach is integrating the code written in assembly language with that created by the compiler.

Other than the speed and size of the software in a real-time system, other important characteristics of a real-time computer language include the ability to directly control external devices and custom hardware, efficient interrupt handling, and support for system testing.

Examples of compiled languages used in real time systems include BASIC, Fortran, PL/M, Pascal, Modula-2, Ada, and C. BASIC has the advantages that it is easy to learn, simple to

use, and is inexpensive, but it also has the disadvantages that it supports unstructured programming techniques, but does not easily support the direct control of external devices.

Fortran was developed in the mid-1950s as a scientific and engineering applications programming language. It suffers from the same problems as BASIC when applied to real-time systems, but if the real-time system requires extensive numerical processing capability, the disadvantages of Fortran may be overcome by its excellent data processing capabilities.

PL/M is a language developed by Intel for real-time systems using Intel processors. It allows direct access to the input/output of Intel processors, and also has capabilities which improve interrupt handling. PL/M has the disadvantage that it does not support non-Intel processors.

Pascal was developed as a language for teaching structured programming to students. It has little application as a real-time language because it does not allow direct control of the computer hardware.

Modula-2 was created by the designer of Pascal for use as a real-time programming language, and it corrects for the deficiencies that Pascal has in these applications while maintaining the advantage of the strong structure of Pascal.

Ada was developed with the support of the Department of Defense as a language for all its embedded computer systems. An embedded computer system is a computer system which is an integral part of a larger system. Ada has all of the

capabilities required of a real-time computer language, but it has the problem that it is very large and complex, which makes it difficult to learn.

C is a language developed at Bell Labs, and is probably the most popular language in real time control and many other applications at this time. It has the advantages that it allows direct access to the computer hardware, and is usually very efficient when compiled.

The only interpreted language that is used extensively in real time control is Forth. Forth has all the capabilities of a good real-time language, and application programs are generally very compact. Forth's biggest advantage is that since it is a stack-based language, it provides fast context switching. Although interpreted languages tend to be slow, Forth runs quite efficiently, while also providing the program development advantages of other interpreted languages. After program development is finished, the Forth program can be compiled into a assembly language program which can be burned into a read-only memory (ROM) just like compiled languages. Other advantages of Forth are that it supports development of the software on the computer system that will be used in the real-time system, and that software-development packages are inexpensive. The biggest disadvantages of Forth are that the programs developed tend to be cryptic and hard to maintain, and that postfix notation, which is the syntax of the language, tends to be harder to understand than standard

syntax.[Lawrence87]

In a real-time system, there must be tradeoffs made between software and hardware, depending on costs, computation speed required, development time, and other factors. In low-speed applications, functions which are usually performed in hardware in time critical systems are performed in software, and the performance of the software is also sacrificed for budget reasons. Medium-speed applications require system software to be time efficient, with prioritized event handling, but some hardware functions may still be performed in software. In high-speed applications, the required response time is nearly equal to the computer's capacity. All hardware speed improvements are used, and the software must be optimized so that no processing capacity is wasted. High-speed systems are usually exceptionally simple, due to the fact that there is not time to execute anything complicated.

One approach to improving the marriage of hardware and software in a computer system is to tailor the architecture of a processor to a particular high-level language. This allows the programmer to take advantage of ease of programming in a high-level language, while increasing the efficiency of mapping the high-level language instructions into the assembly code of the processor. This has been realized by several design teams, but the results tend to be more complex than is necessary for control applications.[Fraeman86][Golden85] [Hayes87][Hayes86][Jones87][Koopman88][Williams86]

The rest of this document addresses the development of a very simple hardware architecture tailored to execute a core subset of the Forth programming language, with consideration toward use in real-time control applications.

The first three chapters give background information concerning control systems, the Forth programming language, and stack-based architectures. The fourth chapter focuses on the instruction set and architecture developed. The fifth chapter discusses the architectural decisions which were made, and the register transfer level modelling of the system. The final chapter discusses work yet to be accomplished and some packaging considerations and performance trade-offs.

# CHAPTER 2

# THE FORTH PROGRAMMING LANGUAGE

## A Brief History of Forth

Unlike most other languages, Forth is not the creation of a committee or design team, but that of a single person, Charles Moore. During the 1960s, he became frustrated with the time and effort required to write programs in the programming languages which existed at the time, so he created Forth. In doing so, he disregarded most of the conventions of other programming languages and included capabilities which he believed were needed for productive programming. The most important of these is the extensibility of Forth, so that it may be tailored to solving the problem at hand.

In 1971, Moore was hired by the National Radio Astronomy Observatory to program a data-acquisition system for a radio telescope, which he wrote in Forth. This led to the acceptance of Forth as a major applications language by the community of astronomers. By 1973, the demand for Forth had increased such that Moore, in partnership with Elizabeth Rather, formed the company FORTH, Inc., with the goal of expanding Forth applications to various mainframe, mini- and microcomputers.

The popularity of Forth also has spread due to the

availability of public-domain versions of the language. These have been distributed by the Forth Interest Group (FIG), and are available for most computers. FIG also publishes FORTH Dimensions, a bimonthly magazine which addresses the modification and extension of Forth as well as its application to programming problems, and sponsors a conference called the Forth Modification Laboratory, the purpose of which is to allow users and system developers to meet and discuss the development of the language. A counterpart to this conference is the Rochester Forth Application Conference, sponsored by the Institute for Applied Forth Research, Inc.

Forth applications are wide and varied. To name a few, it is used in data acquisition and analysis, expert systems, graphics, medicine, process control, and robotics.

## Basic Structure of Forth

The core of Forth is a dictionary or list of approximately 300 operations known as words. These words are essentially subroutines, and can be combined to define other operations, which can then be added to the dictionary and used in more complicated definitions. These definitions can be combined until a single word completely describes and executes a given task. Forth is highly structured, in that every program is a list of these words, each of which is defined in terms of other words, and so on until at the bottom level, everything is defined in terms of the core operations of Forth. This type of code is known as threaded code, and is

efficient in terms of speed and memory.

The use of Forth complements the practice of a top-down design methodology. A problem is broken down into functional blocks, which in turn are broken down further until each function is described in basic steps. Forth allows each of these basic steps to be defined as words, and then combined into functional blocks until the solution of the problem is complete.[McCabe83][Brodie87]

## Forth Difficulties

Forth's largest pitfall is the fact that programs written in the language are difficult to read and comprehend, and it often requires special effort and documentation for the program to be understood by anyone, including the original programmer. The main problem concerning the readability of Forth code is that the language is stack-based, which requires that the syntax of the language be postfix, similar to the notation used on Hewlett-Packard calculators.

Forth programs are stored in a non-standard form. Information is stored in mass-storage as numbered blocks of 1,024 characters, which are arranged into 16 lines of 64 characters each. When a program is larger than a block long, special procedures are required to load and execute it. This format was reasonable when Forth was invented, since computer memory was limited and expensive, but these problems do not exist to the scale that they did originally.

Forth also lacks several basic functions which are

standard in other high-level languages. Error checking is virtually nonexistent, and there is no standard way of manipulating strings of data, working with files of data, or using graphics.

The counter-argument to these complaints is that most commercial versions of Forth provide functions which overcome these limitations, and that Forth allows a programmer to easily extend the dictionary of the system to include words which correct for limitations as they are discovered. [Lawrence87]

Forth also uses scaled-integer arithmetic instead of the more standard floating-point arithmetic, and if floating point functions are needed, then those functions need to be written and integrated into the system. Scaled-integer arithmetic is a method of storing numbers in memory without having to keep track of each number's decimal point. All numbers are treated as integers, which are all of the same scale. Examples of scaled-integer and floating-point representations are shown in Table 1.

Table 1. Scaled-Integer Arithmetic

| Real-World Value | Scaled-Integer Representation | Floating-Point Representation |
|---|---|---|
| 1.23 | 123 | $123 \times 10^{-2}$ |
| 10.98 | 1098 | $1098 \times 10^{-2}$ |
| 100.00 | 10000 | $1 \times 10^{2}$ |
| 58.60 | 5860 | $586 \times 10^{-1}$ |

The arguments for the use of scaled-integer arithmetic are based on the concept that it is much faster to execute a

calculation in scaled-integer arithmetic than it is in floating-point, and the fact that Forth supports special operators which improve the ease of using scaled-integer math.[Brodie87]

Although all these arguments, both positive and negative, bring up valid points, it must be considered that the ability of Forth to be tailored toward a particular problem, the speed of the programs written, and the compact size of the code may be the deciding factors when choosing a language to write real-time software.

# CHAPTER 3

## STACK-BASED ARCHITECTURES

Most conventional processors are optimized for office automation systems or computer-aided design applications. The memory management and general-purpose data processing requirements of these applications require design complexity in the processor which is usually not necessary in control applications. Microcontrollers, on the other hand, have been developed for specific applications and do not require the complexity of general-purpose processors, but tend to have much slower instruction execution rates than general-purpose processors, which hinders their use in real-time systems.

High-performance stack-based architectures have been proposed to improve the performance of microcontrollers in real-time applications, citing advantages in algebraic problems and internal addressing schemes. This idea is not new, since stack-based architectures were proposed as early as 1962, when the KDF.9 computer system was designed by the English Electric Company.[Haley62]

### Stacks

A stack, also known as a push-down stack, is a one dimensional array used for temporary storage of data. Items are entered and removed from stacks one at a time, such that

the last item placed on the stack is the first one to be removed. This method of storing and removing data on the stack is generally known as a last in, first out (LIFO) method. The process of adding an item to a stack is called a push, and the act of removing an item from the stack is called a pop. A common example of the operation of a stack is that of a rifle magazine, where a spring is used to keep the ammunition at the top, and the last shell to be pushed into the magazine is the first to leave the magazine, just like the data in a LIFO stack.

## Advantages

One advantage of a stack-based architecture is that the addresses of operands are implicit in the stack, which minimizes control overhead. A stack uses only a single pointer register to keep track of accessible data. All arithmetic operations execute on data from the stack and store the result of the operation on the stack. This means that no addresses are required for arithmetic operations, since they are implied by the function.

Stack-based architectures use postfix notation, more commonly known as reverse Polish notation, which is the notation used in Hewlett-Packard calculators. When using postfix notation, the operator is specified only after the operands have been placed on the stack. Stack operations do not depend on the manner in which the operands reach the stack. Operands may be placed on the stack explicitly, or

they may be the result of an earlier operation. The stack serves as a common location where data may be passed between operations or manipulated. Examples of postfix notation are shown in Table 2, where they are compared to the more common algebraic or infix notation.

Table 2. Algebraic and Postfix Notation

| Algebraic Notation | Postfix Notation |
|---|---|
| 4 + 8 | 4 8 + |
| 21 / 7 | 21 7 / |
| 5 * (2 + 7) | 5 2 7 + * |
| (5 - (6 * 7))/8 | 5 6 7 * - 8 / |

Keeping track of the order and magnitude of stack contents requires that the programmer pay attention to what is happening in the program, but overall, postfix notation is often easier to work with than infix notation.

Compilers also use postfix notation to express high-level language calculations as an intermediate step in translation to machine language, so the use of a stack-based architecture removes some complexity from the compiler.[Kelly86]

A stack-based architecture also has an important advantage in the event of an interrupt. In a conventional processor, when an interrupt occurs, the program counter and the contents of all registers in the processor must be saved before the interrupt can be serviced. In a stack-based processor, all registers are already saved on the stack, so all that has to be preserved is the program counter before entering the interrupt service routine.

There must be some consideration taken with respect to how the program counter is saved in the occurrence of an interrupt and also on subroutine calls. In a conventional processor the program counter is saved on a memory stack and then popped back into the program counter register on the return from subroutine. Stack-based processors take the same approach, but the program counter is saved on a second hardware stack usually called the return stack, which is used for saving the return address for subroutines and interrupts and for temporary storage of data in normal processing operations.

## Special Requirements

The use of a stack-based architecture and postfix notation require some special data manipulation instructions which have no equivalent counterparts in other systems. In a calculation it may be necessary to reverse the order of the top two locations in the stack, to duplicate the top location in the stack, or to discard the top location in the stack, and instructions must be defined to accomplish these operations.[Haley62]

## Other Stack-Based Designs

Stack-based designs generally focus on tailoring their architecture to the high-level language Forth, since it is stack-based and therefore directly is supported by the architecture. Recent examples of stack-based architectures include the Johns Hopkins University FRISC machines

[Fraeman86][Hayes87][Hayes86][Williams86], the Novix machines[Golden85][Ting86], and the RTX machines, which use the FORCE architecture[Jones87][Koopman88]. All of these machines support a very fast subroutine call and return, since almost every Forth word is a subroutine. Of these machines the Novix is the simplest, with only the datapath and control on the chip, with stacks, program, and data memory off-chip. The Novix processor allows only 256 word stacks, and expects the programmer to watch for overflow problems. One feature in the Novix processor which improves the throughput of the system is the return from subroutine. A flag bit in the op-code of the last instruction in a subroutine indicates the return, and the return is executed concurrently with the instruction, effectively reducing overhead to zero.

The FRISC (Forth Reduced Instruction Set Computer) machines are set apart from the other architectures by the stack caching implemented in the chip. The caching algorithm implemented in FRISC 3 uses a ring buffer for the on-chip stack cache. A stack pointer is used to indicate position in the ring, and two sliding pointers mark the overflow and underflow points on the ring. If the stack pointer reaches the overflow or underflow point, then data is written to or read from external memory, and the overflow or underflow point is adjusted accordingly. The problem with caching in a real-time system is that the cache introduces uncertainty in critical timing paths. If the cache has to be serviced during

a critical operation it may cause the system to fail.

The RTX (Real-Time Express) machines use an architecture which was called the FORCE (Forth Optimized RISC Computing Engine) architecture when it was designed, but the name was changed to avoid confusion with the FORCE computer company. These processors use a building block approach to systems. They supply the FORCE processor core and a library of macrocells which can be interfaced on a single VLSI chip to create a system. This library includes timers, interrupt controllers, clock generators, I/O controllers, stack controllers, RAM, and a 16 X 16 bit multiplier. This approach allows each system to be adapted for a particular application. The RTX machines also support the same type of return from subroutine as the Novix processors, which results in zero overhead for returns.

Each of these systems has its advantages, but they appear to be more complex than is necessary for high-speed control applications.

# CHAPTER 4

# THE FTCP

As was mentioned in previous chapters, the key to high-speed controllers is often extreme simplicity. With this in mind, a limited instruction set which is a core subset of the Forth programming language was chosen as the executable instructions for a very simple architecture, with a few modifications to support the needs of control applications.

## The FTCP Instruction Set

A large portion of the FTCP instruction set was taken from the instruction set of the Bridger SIMD processor array[Winters88]. Additions to that instruction set include the greater than, less than, and equal comparison instructions, the interrupt control instructions, and the external data storage instructions. The instruction set, which is composed of 27 instructions, is shown in Figure 1.

Figure 1. The FTCP Instruction Set
(continued on next page)

**FTCP Instruction Set**
Revision 6
11/5/91

| Stack Functions | | |
|---|---|---|
| DUP | (n - - n n) | Duplicates the top word of the data stack. |
| DROP | (n - - ) | Discards the top word of the data stack. |
| SWAP | (n1 n2 - - n2 n1) | Reverses the order of the top two stack words. |

## Figure 1. (cont.) The FTCP Instruction Set

(continued on next page)

| | | |
|---|---|---|
| >R | (n - - )( - - r) | Pops the top word of the data stack onto the return stack. |
| R> | (r - - )( - - n) | Pops the top word of the return stack onto the data stack. |
| **Arithmetic Functions** | | |
| + | (n1 n2 - - n1+n2) | Adds the top two words of the data stack. |
| - | (n1 n2 - - n1-n2) | Subtracts the top word of the data stack from the word below it in the data stack. |
| 2* | (n - - 2*n) | Multiplies the top word of the data stack by 2. (Note: The least significant bit of the word is replaced with a 0) |
| 2/ | (n - - n/2) | Divides the top word of the data stack by 2. (Note: The most significant bit of the word is fed back into the register so that the sign of the number is retained) |
| SHIFTR | (n - - n) | Shifts the top word of the data stack right one bit. (Note: The leftmost bit of the word is replaced with a 0) |
| **Boolean Functions** | | |
| NOT | (n - - n') | Performs one's complement on the top word of the data stack. |
| NAND | (n1 n2 - - n) | Performs bitwise NAND of the top two words of the data stack. |
| XOR | (n1 n2 - - n) | Performs bitwise XOR of the top two words of the data stack. |
| **Comparisons** | | |
| > | (n1 n2 - - n) | Pushes -1 on the top of the data stack if n1>n2. If the condition is false a 0 is pushed on the top of the data stack. |
| < | (n1 n2 - - n) | Pushes -1 on the top of the data stack if n1<n2. If the condition is false a 0 is pushed on the top of the data stack. |
| = | (n1 n2 - - n) | Pushes -1 on the top of the data stack if n1=n2. If the condition is false a 0 is pushed on the top of the data stack. |
| **Control Functions** | | |
| CALL | ( - - r) | Subroutine call performed by pushing the program counter on the top of the return stack and placing the subroutine address in the program counter. |