Project Outline

Ruben Saunders

August 2024

Contents

1 Overview **2**
1.1 Motivation & Goal 2
1.2 Feasability Plan 2

2 The Processor **3**
2.1 Emulator Requirements 3
2.2 Processor Requirements 4

3 The Assembler **5**
3.1 Assembler Requirements 6
3.2 Assembly Language Requirements 6

4 The Compiler **7**
4.1 Compiler Requirements 7
4.2 High-Level Language Requirements 7

1 Overview

This project aims to design and build all three "layers" of operation down from a high-level language to the processor itself. Specifically, my third year project has three component:

- Designing a RISC processor and writing an emulator for it, the "processor" stage;
- Designing an assembly language and writing an assembler to output a binary for the above processor, the "assembler" stage; and
- Designing a custom high-level language and writing a compiler to output the above assembly code, the "compiler" stage.

1.1 Motivation & Goal

The project has the goal of being primarily educational, both to me and to users. The desired simply design has the intention of being easy to understand at all stages and therefore a resource for education and exploration on the topic of CPU design, assembly language, compiler design, and learning a high-level language, as well as the interaction between all three stages. The educational value to me, in addition to above, will be managing and documentation a large project, and the integration of three smaller sub-systems into one workflow.

Hence stems the decision to fully design and build all three stages; ot fully explore and consider the entire pipeline, from user code to machine code.

1.2 Feasability Plan

This project is divided into three sub-systems, each requiring a different, but relatively large, amount of research and work. Each sub-system is completely dependent on the layer below it, so it is essential that each sub-system is completed in a reasonable time to allow for the next sub-system to be worked on. This is only for writing; research, documentation, and design can be written beforehand, and ideally so as to ensure each sub-system will interface well.

The high-level language has been identified to be the most complex sub-system, both in terms of design and creation. Therefore, this sub-system should have the most time reserved for it. Furthermore, due to the presence of the compiler design module, this sub-system has the most to benefit from term-time. With this in mind, the following rough time-frames and targets have been deduced.

		Processor	**Assembler**	**Compiler**
Summer	Design	100%	100%	
	Implementation	100%	100% core	

Term 1	Design			100% core
	Implementation		100%	$\sim 30\%$
Term 2	Design			100%
	Implementation			$> 90\%$
Easter	Design			
	Implementation			100%

The development of each sub-system is plan-driven, meaning that documentation will be full or mostly written during design, with minimal changes during implementation. This makes sense with a larger project and timeframes, and the reliable and consistent documentation will ensure the project is kept on-track and to specification.

2 The Processor

Core to the project, a RISC-based processor will be designed and an emulator written. The emulator is but a program which will implement the processor specification.

2.1 Emulator Requirements

> "The emulator must be able to accept a binary file, read it, and execute the contents according to a RISC processor specification."

R1 The emulator shall be able to accept, read, and execute a binary.

- R1.1 The emulator shall be able to accept a file name as an argument.
- R1.2 The emulator shall be able to open and read a file in binary mode.

R2 The emulator shall execute a binary according to a pre-defined RISC processor specification.

- R2.1 The emulator shall implement a processor specification.
- R2.2 The specification shall define a RISC-based processor.
  - R2.2.1 The specification shall fully outline the ISA.
  - R2.2.2 The specification shall fully outline expected memory structure.
  - R2.2.3 The specification shall fully outline all available registers.

R3 The emulator shall be able to indicate its past and current state.

- R3.1 The emulator shall be able to print to the console.
- R3.2 The emulator shall be able have a configurable debug/verbose mode to provide further details into its operation and state.

2.2 Processor Requirements

"The processor must be RISC based with a minimal but expressive instruction set."

R1 The processor's instruction set shall be reduced and minimal.

- R1.1 Every instruction shall be encoded in a fixed-size.
- R1.2 Each instruction shall be atomic.
- R1.3 Each instruction shall be necessary.

R2 The processor's instruction set shall minimally be Turing-complete and expressive.

- R2.1 The processor shall operate primary through registers.
- R2.2 The ISA shall provide instructions for storing and retrieving data.
  - R2.2.1 The ISA shall provide operations to load data into registers.
  - R2.2.2 The ISA shall provide operations to store data into memory.
- R2.3 The processor shall support both positive and negative integers, as well as floating-point numbers.
  - R2.3.1 The ISA shall support the conversion between integers and floating-point values.
- R2.4 The ISA shall support arithmetic operations for both integers and floats.
  - R2.4.1 The ISA shall support a method to specify the datatype for an operation.
    - R2.4.1.1 The ISA shall be able to specify a (signed?) word.
    - R2.4.1.2 The ISA shall be able to specify a (signed?) half-word.
    - R2.4.1.3 The ISA shall be able to specify an IEEE754 float and double.
  - R2.4.2 The ISA shall support addition.
  - R2.4.3 The ISA shall support subtraction.
  - R2.4.4 The ISA shall support multiplication.
  - R2.4.5 The ISA shall support division.
- R2.5 The ISA shall support conditional branching and jumps.
  - R2.5.1 The ISA shall support the comparison between registers and values.
  - R2.5.2 The processor shall be able to store the result of comparisons for future use.
  - R2.5.3 The ISA shall support branching conditional on the result of a comparison.
- R2.6 The ISA shall support a complete set of logical operations.
  - R2.6.1 The ISA shall support bitwise-not.

R2.6.2 The ISA shall support bitwise-and.

R2.6.3 The ISA shall support bitwise-or.

R2.6.4 The ISA shall support both logical left– and right-shifting.

The following additional requirements extend the above requirements by adding more expressiveness to the processor:

AR1 The ISA shall provide pseudo-instructions for common tasks.

AR2 The ISA shall provide instructions for common operations which are covered but may be complex or multi-stepped to do so.

AR2.1 The ISA shall provide a modulo operation.

AR2.2 The ISA shall provide a bitwise-exclusive-or operation.

AR3 The ISA shall support functionality to call and return from procedures.

AR3.1 The processor shall support the implementation and operation of a global stack structure.

AR3.1.1 The processor shall have a stack pointer register.

AR3.1.2 The ISA shall support pushing values to the stack.

AR3.1.3 The ISA shall support popping values off the stack.

AR3.2 The processor shall support the concept of stack frames.

AR3.2.1 The processor shall have a frame pointer register.

AR4 The processor shall support configurable interrupts.

AR4.1 The processor shall have an interrupt status register.

AR4.2 The processor shall have an interrupt mask register.

AR4.3 The processor shall check for the presence of an interrupt every fetch-execute cycle.

AR5 The processor shall provide functionality syscall functionality for additional useful behaviours.

AR5.1 The processor shall provide syscalls for printing data to stdout.

AR5.2 The processor shall provide syscalls for inputting data to stdin.

AR5.3 The processor shall provide syscalls for halting the current process.

3 The Assembler

An intermediary between machine code for the emulator and the high-level language, the assembler's role is to take generated assembly code and output a binary.

3.1 Assembler Requirements

"The assembler must convert an assembly source to a machine-code binary executable on the emulator."

R1 The assembler shall be able to accept, read, assemble a text file, and output a binary.

- R1.1 The assembler shall be able to accept a file name as an argument.
- R1.2 The assembler shall be able to open and read a file in text mode.
- R1.3 The assembler shall be able to write to a file in binary mode.

R2 The assembler shall transform a text source to a binary according to a specification.

- R2.1 The assembler shall implement an assembly language specification.
  - R2.1.1 The specification shall specify the overall structure and syntax of the assembly code.
  - R2.1.2 The specification shall outline and describe mnemonics to implement the processor specification.
  - R2.1.3 The specification shall outline argument formats.
- R2.2 The output assembly shall be executable with no non-user errors by the processor.

3.2 Assembly Language Requirements

"The assembly language must be able to express all the processor's operations and functions, serving as a more abstract but equally powerful medium for programming the processor."

R1 The assembly shall be at least as powerful as the processor.

- R1.1 The assembly shall implement all instructions stated in the processor specification.
- R1.2 The assembly shall support access to all available memory regions of the processor.

R2 The assembly shall provide a layer of abstraction over raw processor binary.

- R2.1 Assembly mnemonics shall be short, easy to remember, and relate to the instruction.
- R2.2 Assembly arguments shall be explicit and use of different formats obvious to recognise.
- R2.3 The assembly shall support labels as placeholders for addresses to ensure ease of programming.

4 The Compiler

Arguably the entire reason for the project, a high-level language will be designed and documented. The compiler will simply implement this specification, outputting one or more assembly files for the assembler.

4.1 Compiler Requirements

> "The compiler must be able to accept one or more files, read them, and compile the contents to one or more assembly files."

R1 The compiler shall be able to accept, read, assemble a text file, and output an assembly file.

- R1.1 The compiler shall be able to accept a file name as an argument.
- R1.2 The compiler shall be able to open and read a file in text mode.
- R1.3 The compiler shall be able to write to a file in text mode.

R2 The compiler shall transform a text source from the language to a text assembly source according to a specification.

- R2.1 The compiler shall implement a language specification.
  - R2.1.1 The specification shall specify the overall structure and syntax of the code.
  - R2.1.2 The specification shall specify how to create, read, and write to variables.
  - R2.1.3 The specification shall outline control structures.
- R2.2 The compiler shall implement a language specification.

4.2 High-Level Language Requirements

> "The language must be able to provide structures or methods to fully exercise the assembly language, but provide sufficient abstraction as to reveal no design intricacies or details, and be easy to read and understand."

R1 The language shall be at least as powerful as the assembly language.

- R1.1 The language shall implement or provide structures for all assembly instructions.

R2 The language shall provide a layer of abstraction over assembly code.

- R2.1 No assembly mnemonic shall be necessary to directly reference.
  - R2.1.1 The language shall provide standard infix operators to achieve operations between data.

R2.1.2 The language shall provide semantic structures to express branching operations.

R2.2 The language shall provide the ability to address registers and memory locations without explicit knowledge of the programmer.