BAEJ Architecture v0.1

Team 3B - Alex Dripchak, Joshua Eckels, Bailey Morgan, Eric Tu

Rose-Hulman Institute of Technology
CSSE232 Computer Architecture – Dr. Micah Taylor

Contents

1. Project Scope

The BAEJ v0.1 processor is a 16-bit, dual-port memory, load-store inspired and accumulator-influenced, multi-cycle architecture. The purpose of the design is to allow for a simple arithmetic and memory operation instruction set. The capabilities of the processor include function calling and returning, recursive calls, basic arithmetic operations, memory caching, data storage and retrieval, and basic register-mapped input and output. The performance of the processor was evaluated based on clock speed, execution time, and the cycles per instruction required to execute a relative prime program with a given input.

2. Design Overview

2.1 Instruction Set

BAEJ architecture supports two instruction types: I-types and G-types. I-types are 2-word instructions, with the first 16-bit word specifying the op-code and register addresses, and the second 16-bit word representing an immediate value to be used with the instruction. G-type instructions are general-purpose, single-word instructions that follow the same format as I-types, but do not require an immediate value.

BAEJ architecture currently supports 15 instructions:

Category	Type	Mnemonic	Instruction
Arithmetic	G	add	add (+)
	G	sub	subtract (-)
	G	and	bitwise and (&)
	G	orr	bitwise or (|)
	G	slt	set on less than
	I	sft	bit shift
	G	cop	copy
Memory operation	I	ldi	load immediate
	I	lda	load address
	I	str	store
Program Control	I	bop	bop (branching)
	I	beq	bop on equal
	I	bne	bop on not equal
	I	cal	function call
	G	ret	function return

Table 2.1.1: BAEJ Instruction Set

See the Instruction section in Appendix A for more detail on the instruction set architecture.

2.2 Implementation

The goal of this section is to clearly identify each hardware component of the processor and provide a brief overview of the datapath. Table 2.2.1 shows a list of all hardware components used in the processor. See the Hardware section in Appendix A for a more comprehensive list.

Component	Quantity	Description
Adder	2	Takes in two 16-bit inputs and outputs the 16-bit addition result
1-bit multiplexer	8	1 selector bit for two 16-bit inputs and one 16-bit output
2-bit multiplexer	1	2 selector bits for four 16-bit inputs and one 16-bit output
ALU	1	Performs all basic arithmetic operations
Memory Unit	1	Dual-port memory 16x1024 (width x depth)
Register File	1	Read and write dual-port register file 16x64 (width x depth)
Register	10	16-bit rising-edge flip-flop
Fcache	1	Memory cache 256x1024 (width x depth)
Comparator	1	Performs equal (=) and not equal (!=) comparisons
Control Unit	1	Implemented as a state machine (combinational logic)

Table 2.2.1: Hardware Components

2.3 Final Datapath

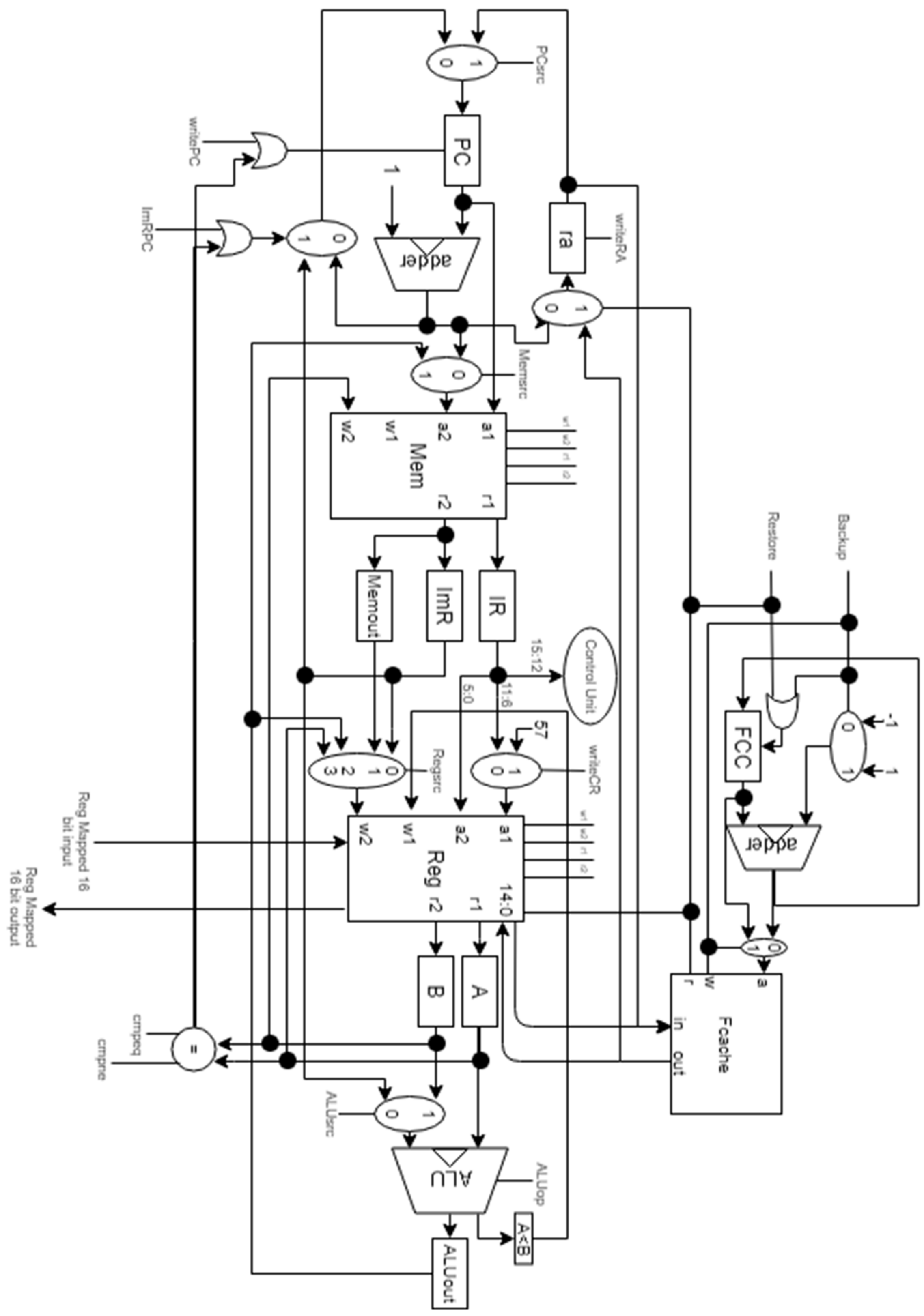
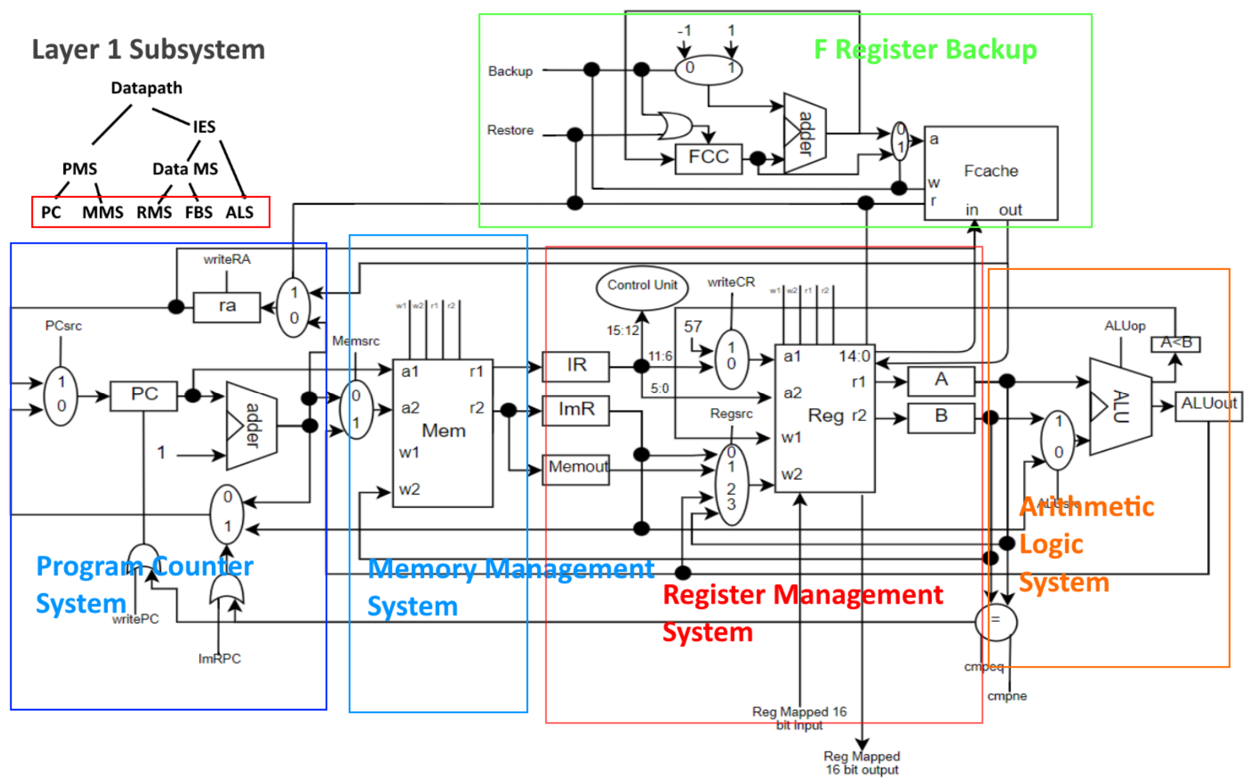


Figure 2.3.1: BAEJ Architecture Datapath

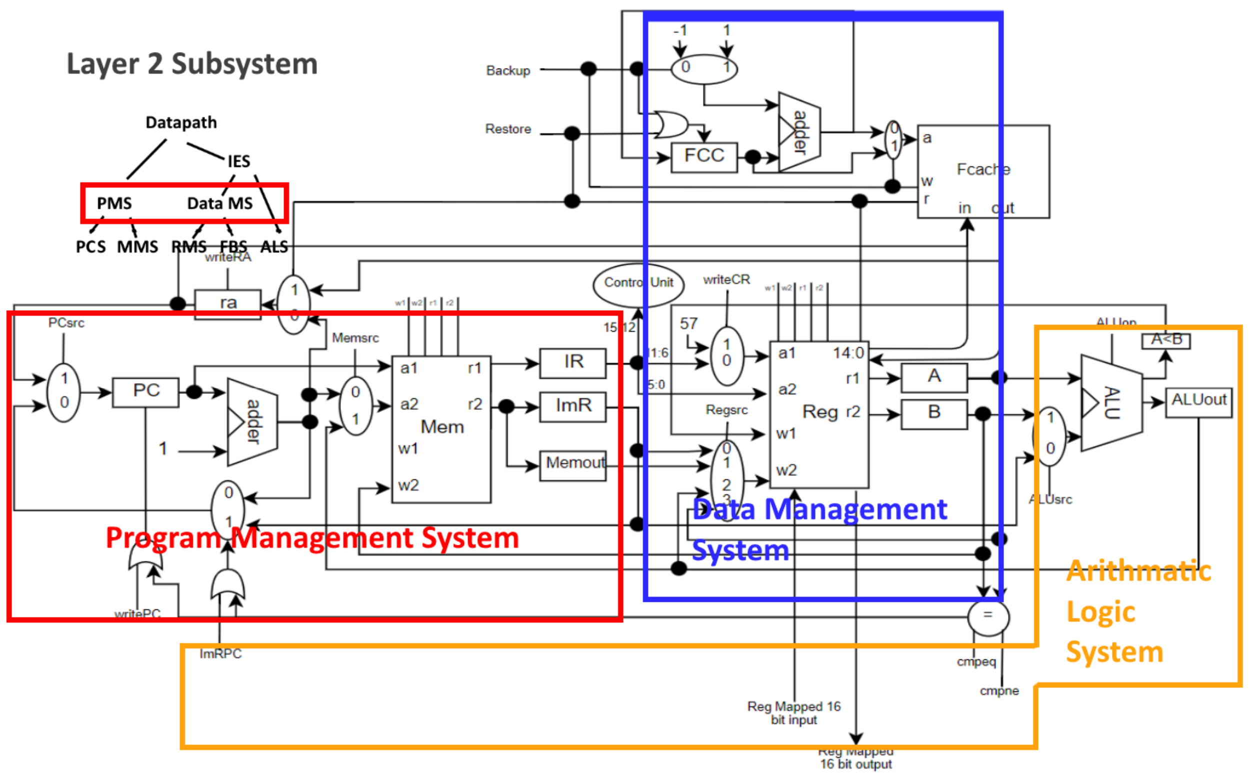
2.4 Integration and Testing

The datapath was assembled in three distinct integration layers, as summarized in the tables and figures below. Each integration subsystem was extensively tested by executing several permutations of inputs and control signals and verifying the correct outputs. See the Testing section of Appendix A for the detailed testing plan of each subsystem.



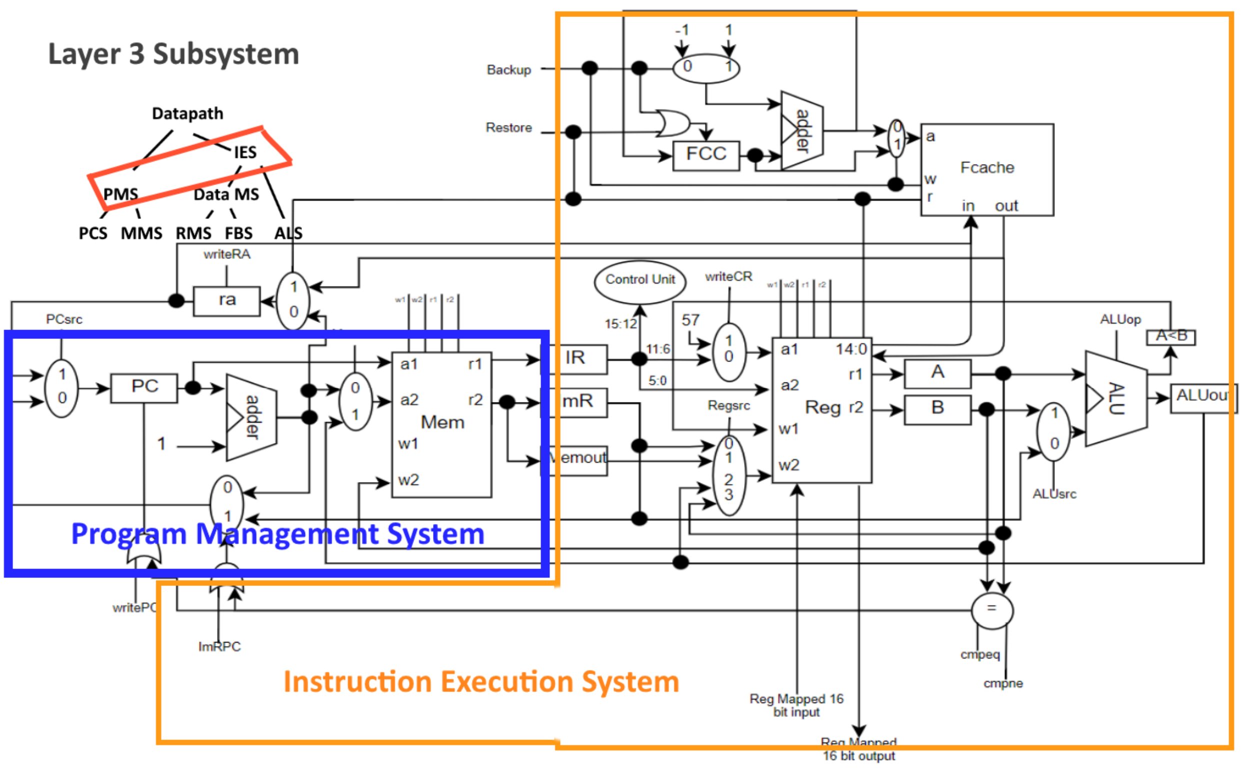
Subsystem	Description
Program Counter System (PCS)	Tracks and increments program counter (PC)
Memory Management System (MMS)	Manages input/output from memory unit
Register Management System (RMS)	Manages input/output from register file
Arithmetic Logic System (ALS)	Performs arithmetic operations
F Register Backup System (FBS)	Manages Fcache backup and restoring

Figure 2.4.1: Layer 1 Sub-systems



Subsystem	Description
Program Management System (PMS)	Obtains values from memory at PC
Data Management System (DMS)	Manages register file backing up and restoring
Arithmetic Logic System (ALS)	SEE LAYER 1

Figure 2.4.2: Layer 2 Sub-systems



Subsystem	Description
Program Management System (PMS)	SEE LAYER 2
Instruction Execution System (IES)	Performs arithmetic operations on values obtained from register file

Figure 2.4.3: Layer 3 Sub-systems

3. Features

3.1 Caching

Conventionally, in a load-store based architecture, the user is required to backup variables and the return address on the stack to preserve them across function calls. The BAEJ architecture eliminates this responsibility from the user, instead having all necessary backup protocols occur automatically within the hardware. Upon function calling, the hardware sends the data in the designated "f registers", along with the return address, up to the memory cache (Fcache), where it is safely stored in a 256-bit word of cache memory. Upon function returning, these values are safely returned to the user in the "f registers" and to the return address register. This backup and restore caching system can support multiple function calls at a time, (up to the designated Fcache memory depth of 1024), without any need of stack memory.

3.2 BAEJ Assembler

The BAEJ assembler included is written in C++ and converts a baej source file to machine code. The assembler makes two passes over the source code. In the first pass, the source is read from the file provided. It looks up all the symbols in the source and maps them into a symbol table. In the second pass, the symbol table is used to convert the source code to machine code which is then written to an output file, baej.out. The current implementation of the assembler outputs the machine code as a text file rather than binary so that it can easily be used with the verilog test bench and the simulator.

3.3 Performance Simulation

The BAEJ simulator is written in C++ as well and it does the job of simulating algorithms implemented in the BAEJ language. It must be given a machine code file, such as the output from the assembler, and an input to the system. The simulator will then simulate the given program and return the output of the program (corresponding to the given input), as well as performance feedback. Performance metrics returned by the simulator include bytes transferred to and from memory, program size in memory, number of instructions and cycles executed, average CPI, and execution time of the program. On top of providing these performance metrics, the simulator is a means of testing the language itself, as well as programs and algorithms written in the BAEJ language. Where Xilinx test benches test the hardware implementation of the datapath, the BAEJ simulator tests the correctness of the code being ran.

4. Conclusion

The BAEJ architecture provides simple and robust functionality for the execution of algorithms such as relative prime and recursive summation. The architecture was designed for ease of use on the part of the user in its straightforward and intuitive interface. Its versatility is evidenced by its use of caching, its dual-port memory functionality, and its compact instruction set.

The BAEJ architecture comes equipped with an assembler, as well as a simulator for its instruction set. The high performance of the architecture is evidenced by low execution times and correct results provided by the simulator.

On top of its current excellent performance, BAEJ architecture has the potential for further growth and development. Opportunities presented by exploring a larger instruction set, a larger memory size, and increased performance optimizations leave BAEJ with much potential in continued development.

Appendices

Appendix A: Design Documentation