**Processor Design Project**

**Interim Report**

**EE480**

Team Members: Alex Hendren & Zachary Sean McFeely

3/25/2013

Table of Contents

[1. Introduction 3](#_Toc351824127)

[2. Instruction Set Architecture Development 3](#_Toc351824128)

[2.1. Implementation 3](#_Toc351824129)

[2.1.1. Add 4](#_Toc351824130)

[2.1.2. Sub 4](#_Toc351824131)

[2.1.3. Logical OR 5](#_Toc351824132)

[2.1.4. Logical AND 5](#_Toc351824133)

[2.1.5. Logical Complement (COMP) 6](#_Toc351824134)

[2.1.6. Multiply and Divide 7](#_Toc351824135)

[2.1.7. Arithmetic Left/Right Shift 8](#_Toc351824136)

[2.1.8. Conditional Branch 9](#_Toc351824137)

[2.1.9. Unconditional Jump 9](#_Toc351824138)

[2.1.10. Jump/Return to/from a Subroutine 10](#_Toc351824139)

[2.1.11. Return from Interrupt Service Routine 10](#_Toc351824140)

[2.1.12. LOAD Accumulator 11](#_Toc351824141)

[2.1.13. STORE Accumulator to RAM 12](#_Toc351824142)

[2.1.14. LOAD A Register from RAM 12](#_Toc351824143)

[2.1.15. STORE A Register to RAM 13](#_Toc351824144)

[2.1.16. LOAD B Register from RAM 14](#_Toc351824145)

[2.1.17. STORE B Register to RAM 15](#_Toc351824146)

[2.1.18. INPUT Data Word to RAM 15](#_Toc351824147)

[2.1.19. OUTPUT Data Word from RAM 16](#_Toc351824148)

[2.1.20. LOAD Mask Register of HVPI 16](#_Toc351824149)

[2.1.21. NOP 17](#_Toc351824150)

[3. Accumulator Architecture Development 17](#_Toc351824151)

[3.1. Architecture Overview 18](#_Toc351824152)

[3.2. Instruction Implementation 19](#_Toc351824153)

[3.2.1. Add 20](#_Toc351824154)

[3.2.2. Sub 21](#_Toc351824155)

[3.2.3. Logical OR 22](#_Toc351824156)

[3.2.4. Logical AND 23](#_Toc351824157)

[3.2.5. Logical Complement (COMP) 23](#_Toc351824158)

[3.2.6. Multiply and Divide 24](#_Toc351824159)

[3.2.7. Arithmetic Left/Right Shift 26](#_Toc351824160)

[3.2.8. Conditional Branch 27](#_Toc351824161)

[3.2.9. Unconditional Jump 28](#_Toc351824162)

[3.2.10. Jump/Return to/from a Subroutine 28](#_Toc351824163)

[3.2.11. LOAD Accumulator from RAM 29](#_Toc351824164)

[3.2.12. STORE Accumulator to RAM 30](#_Toc351824165)

[3.2.13. LOAD A Register from RAM 31](#_Toc351824166)

[3.2.14. STORE A Register to RAM 32](#_Toc351824167)

[3.2.15. LOAD B Register from RAM 33](#_Toc351824168)

[3.2.16. STORE B Register to RAM 34](#_Toc351824169)

[3.2.17. INPUT Data Word to RAM 35](#_Toc351824170)

[3.2.18. OUTPUT Data Word from RAM 36](#_Toc351824171)

[3.2.19. LOAD Mask Register of HVPI 36](#_Toc351824172)

[3.2.20. NOP 37](#_Toc351824173)

[3.3. Direct Mapped Cache 37](#_Toc351824174)

[4. Conclusion 38](#_Toc351824175)

[5. References 38](#_Toc351824176)

Table of Figures

[Figure 1 - Top Level Architecture Overview 19](#_Toc351824183)

[Figure 2 - ADD Flow Diagram 21](#_Toc351824184)

[Figure 3 - SUB Flow Diagram 22](#_Toc351824185)

[Figure 4 - OR Flow Diagram 23](#_Toc351824186)

[Figure 5 - AND Flow Diagram 24](#_Toc351824187)

[Figure 6 - COMP Flow Diagram 24](#_Toc351824188)

[Figure 7 - MULT Flow Diagram 25](#_Toc351824189)

[Figure 8 - DIV Flow Diagram 26](#_Toc351824190)

[Figure 9 - SHFT Flow Diagram 27](#_Toc351824191)

[Figure 10 - BRA Flow Diagram 28](#_Toc351824192)

[Figure 11 - JMP Flow Diagram 29](#_Toc351824193)

[Figure 12 - RTS Flow Diagram 29](#_Toc351824194)

[Figure 13 - LOAD Flow Diagram 30](#_Toc351824195)

[Figure 14 - STORE Flow Diagram 31](#_Toc351824196)

[Figure 15 - LDA Flow Diagram 32](#_Toc351824197)

[Figure 16 - STA Flow Diagram 33](#_Toc351824198)

[Figure 17 - LDB Flow Diagram 34](#_Toc351824199)

[Figure 18 - STB Flow Diagram 35](#_Toc351824200)

[Figure 19 - INPUT Flow Diagram 36](#_Toc351824201)

[Figure 20 - OUTPUT Flow Diagram 37](#_Toc351824202)

[Figure 21 - NOP Flow Diagram 38](#_Toc351824203)

[Figure 22 - Direct Mapped Cache Flow Diagram 38](#_Toc351824204)

# Introduction

The purpose of this project is to design, develop, test, and validate a high performing accumulator based processor capable of efficiently implementing an Instruction Set Architecture (ISA) that we specifically designed for it. By definition, the accumulator based architecture has only one register, called the accumulator, in the data path and for all ALU operations. For every ALU instruction, one operand will always be in the ACC register and the other operand comes from RAM. The process of developing the accumulator processor begins with defining the ISA and designing the physical layout of the architecture. After the architecture is designed, each clock cycle operation of the processor is defined as to show the processors operation and functionality as a system. From this, the steps of each instruction are diagramed on a cycle by cycle basis to define their exact operation by the system on the architecture. Instruction Set Architecture Development

The instruction set architecture defines the computer architecture and capabilities by detailing the data types, possible instructions, registers, memory addressing modes, interrupt control, and data input/output control. Defining the instruction set architecture enables programming the computer architecture, assembly instructions and associated parameters are translated to machine code.

## Implementation

This accumulator computer architecture includes 16 opcodes with multiple flags, which describe addressing modes and specific opcode operation. Each instruction is implemented as 16-bits wide (one word) for this accumulator processor, which features an 8-bit data bus. 5-bits are reserved for the opcode, 3-bits are reserved for the opcode flag, and 8-bits are allocated for the operand. Table 1 provides an overview of the instruction structure. The size of the operand is limited by the width of the data bus (5-bits).

Table 1 - Instruction Structure

|  |  |  |  |
| --- | --- | --- | --- |
| Bits | 15:11 | 10:8 | 7:0 |
| Field | OPCODE | FLAG | OPERAND |

Each of the 16 opcodes are detailed in the following sections by OPCode, flags, assembly format, machine code format, description of operation, architecture level operation, and memory addressing.

### Add

|  |  |
| --- | --- |
| Instruction | Add |
| OPCode | 00000 |
| Flags | 00: Direct 01: Indirect 10: Immediate |
| Format | ADD FLAG OPERAND |
| Machine Format | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 00000 | xxx | RAM ADDR / INTEGER | |
| Description | Adds the specified operand to the current contents of the ACC register. The Output from the ALU will be returned to the ACC register. |
| Operation | ACC ← ACC + OPERAND |
| Memory Addressing | Immediate, Direct, and Indirect |

### Sub

|  |  |
| --- | --- |
| Instruction | SUB |
| OPCode | 00001 |
| Flags | 00: Direct 01: Indirect 10: Immediate |
| Format | SUB FLAG OPERAND |
| Machine Format | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 00001 | xxx | RAM ADDR / INTEGER | |
| Description | Subtracts the specified operand from the current contents of the ACC register. The Output from the ALU will be returned to the ACC register. |
| Operation | ACC ← ACC - OPERAND |
| Memory Addressing | Immediate, Direct, and Indirect |

### Logical OR

|  |  |
| --- | --- |
| Instruction | OR |
| OPCode | 00011 |
| Flags | 00: Direct 01: Indirect 10: Immediate |
| Format | OR FLAG OPERAND |
| Machine Format | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 00011 | xxx | RAM ADDR / INTEGER | |
| Description | Perform a LOGICAL OR operation of contents in Accumulator and OPERAND. The result of the operation will be stored in the accumulator. |
| Operation | ACC ← ACC | OPERAND |
| Memory Addressing | Immediate, Direct, and Indirect |

### Logical AND

|  |  |
| --- | --- |
| Instruction | AND |
| OPCode | 00100 |
| Flags | 00: Direct 01: Indirect 10: Immediate |
| Format | AND FLAG OPERAND |
| Machine Format | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 00100 | xxx | RAM ADDR / INTEGER | |
| Description | Perform a LOGICAL AND operation of contents in Accumulator and OPERAND. The result of the operation will be stored in the accumulator. |
| Operation | ACC ← ACC & OPERAND |
| Memory Addressing | Immediate, Direct, and Indirect |

### Logical Complement (COMP)

|  |  |
| --- | --- |
| Instruction | COMP |
| OPCode | 10000 |
| Flags | 00 |
| Format | COMP FLAG OPERAND |
| Machine Format | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 10000 | 000 | 00000000 | |
| Description | Perform LOGICAL complement/Negation on the contents of the accumulator register. The complement will be stored in the accumulator register. |
| Operation | ACC ← NOT ACC |
| Memory Addressing | N/A |

### Multiply and Divide

|  |  |
| --- | --- |
| Instruction | MUL |
| OPCode | 00010 |
| Flags | 00: Direct 01: Indirect |
| Format | MUL FLAG OPERAND [Lower 4 bits] |
| Machine Format | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 00010 | 000 001 | RAM ADDR | |
| Description | Performs multiplication on lower 4-bits of Accumulator and lower 4-bits of the operand. |
| Operation | ACC ← ACC \* OPERAND |
| Memory Addressing | Direct and Indirect |

|  |  |
| --- | --- |
| Instruction | DIV |
| OPCode | 00010 |
| Flags | 10: Direct 11: Indirect |
| Format | DIV FLAG OPERAND [Lower 4 bits] |
| Machine Format | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 00010 | 010 011 | RAM ADDR | |
| Description | Performs division on lower 4-bits of Accumulator and lower 4-bits of the operand. |
| Operation | ACC ← ACC / OPERAND |
| Memory Addressing | Direct and Indirect |

### Arithmetic Left/Right Shift

|  |  |
| --- | --- |
| Instruction | SHFT |
| OPCode | 00101 |
| Flags | 00: Left Shift in 0 01: Left Shift in 1 |
| Format | SHFT FLAG OPPERAND |
| Machine Format | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 00101 | 000 001 | INTEGER < 7 | |
| Description | Performs a LEFT SHIFT operation on the contents of the accumulator register. The value of the bit shifted in is dependent upon the provided flag. Any number of bits less than 7. |
| Operation | ACC ← ACC [N-2:0 + (FLAG BIT)\*OPERAND] |
| Memory Addressing | N/A |

|  |  |
| --- | --- |
| Instruction | SHFT |
| OPCode | 00101 |
| Flags | 10: Right Shift in 0 11: Right Shift in 1 |
| Format | SHFT FLAG OPERAND |
| Machine Format | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 00101 | 010 011 | INTEGER < 7 | |
| Description | Performs a RIGHT SHIFT operation on the contents of the accumulator register. The value of the bit shifted in is dependent upon the provided flag. Shift by any number of bits less than 7. |
| Operation | ACC ← ACC [OPERAND\*(FLAG BIT) + N-1:1] |
| Memory Addressing | N/A |

### Conditional Branch

|  |  |
| --- | --- |
| Instruction | BRA |
| OPCode | 00110 |
| Flags | 00: Branch if Equal 01: Branch if Not Equal |
| Format | BRA FLAG OPERAND |
| Machine Format | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 00110 | 000 001 | MEM ADDR | |
| Description | Performs a comparison between the Accumulator and Operand. If true, jump to instruction pointed to by OPERAND.  The comparison is performed on the ACC and A registers. |
| Operation | PC ←OPERAND |
| Memory Addressing | Direct |

### Unconditional Jump

|  |  |
| --- | --- |
| Instruction | JMP |
| OPCode | 00111 |
| Flags | NULL |
| Format | JUMP OPERAND |
| Machine Format | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 00111 | 000 | MEM ADDR | |
| Description | Executes an unconditional branch. When encountered the instruction pointer is adjusted to the operand target memory address. |
| Operation | IP ←IP + OPERAND |
| Memory Addressing | N/A |

### Jump/Return to/from a Subroutine

|  |  |
| --- | --- |
| Instruction | RTS |
| OPCode | 01000 |
| Flags | NULL |
| Format | RTS |
| Machine Format | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 01000 | 000 | 00000000 | |
| Description | Transfer program control to the address located in the implied return address. Return is made to the top element in the PC stack. Should also pop the ACC value from the ACC stack. |
| Operation | PC ← Implied Address (From Stack) |
| Memory Addressing | NULL |

### Return from Interrupt Service Routine

|  |  |
| --- | --- |
| Instruction | RTI |
| OPCode | 01001 |
| Flags | NULL |
| Format | RTI |
| Machine Format | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 01001 | 000 | 00000000 | |
| Description | Return from an Interrupt Subroutine to the position the PC was at before the interrupt. The PC location to return to will be the top element in the PC stack. |
| Operation | PC ← Return Address (From Stack) |
| Memory Addressing | NULL |

### LOAD Accumulator

|  |  |
| --- | --- |
| Instruction | LOAD |
| OPCode | 01010 |
| Flags | 00: Direct 01: Indirect  10: Immediate |
| Format | LOAD FLAG OPERAND |
| Format Machine | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 01010 | 000 001 010 | MEM ADDR MEM ADDR Integer | |
| Description | **Direct Mode** Copy the contents of the address in RAM specified by the source operand to the accumulator.  **Indirect Mode:** Copy the accumulator to the destination memory address found at the memory address of the operand  **Immediate Mode:** Copy the 8-bit immediate source operand to the accumulator. |
| Operation | **Direct Mode** ACC ← memory (OPERAND)  **Indirect Mode:**  ACC ← Memory{ memory( OPERAND) }   **Immediate Mode:** ACC ← IMMED OPERAND |
| Memory Addressing | Direct, Immediate |

### STORE Accumulator to RAM

|  |  |
| --- | --- |
| Instruction | STOR |
| OPCode | 01011 |
| Flags | 00: Direct 01: Indirect |
| Format | STORE FLAG OPERAND |
| Format Machine | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 01011 | 000 001 | MEM ADDR | |
| Description | **Direct Mode** Copy the accumulator to the destination memory address specified by the operand **Indirect Mode:** Copy the accumulator to the destination memory address found at the memory address of the operand |
| Operation | **Direct Mode** memory(OPERAND) ← ACC **Indirect Mode:** Memory{ memory( OPERAND) } ← ACC |
| Memory Addressing | Direct, Indirect |

### LOAD A Register from RAM

|  |  |
| --- | --- |
| Instruction | LDA |
| OPCode | 10001 |
| Flags | 00: Direct 01: Indirect  10: Immediate |
| Format | LOAD FLAG OPERAND |
| Format Machine | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 10001 | 000 001 010 | MEM ADDR MEM ADDR Integer | |
| Description | **Direct Mode** Copy the contents of the address in RAM specified by the source operand to the A register.  **Indirect Mode:** Copy the memory data found at the memory address of the operand into the A register.   **Immediate Mode:** Copy the 8-bit immediate source operand to the A register. |
| Operation | **Direct Mode** A ← memory (OPERAND)  **Indirect Mode:**  A ← Memory{ memory( OPERAND) }   **Immediate Mode:** A ← IMMED OPERAND |
| Memory Addressing | Direct, Immediate |

### STORE A Register to RAM

|  |  |
| --- | --- |
| Instruction | STA |
| OPCode | 10010 |
| Flags | 00: Direct 01: Indirect |
| Format | STORE FLAG OPERAND |
| Format Machine | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 10010 | 000 001 | MEM ADDR | |
| Description | **Direct Mode** Copy the A register contents to the destination memory address specified by the operand **Indirect Mode:** Copy the A register contents to the destination memory address found at the memory address of the operand |
| Operation | **Direct Mode** memory(OPERAND) ← A **Indirect Mode:** Memory{ memory( OPERAND) } ← A |
| Memory Addressing | Direct, Indirect |

### LOAD B Register from RAM

|  |  |
| --- | --- |
| Instruction | LDB |
| OPCode | 10011 |
| Flags | 00: Direct 01: Indirect  10: Immediate |
| Format | LOAD FLAG OPERAND |
| Format Machine | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 10011 | 000 001 010 | MEM ADDR MEM ADDR Integer | |
| Description | **Direct Mode** Copy the contents of the address in RAM specified by the source operand to the B register.  **Indirect Mode:** Copy the memory data found at the memory address of the operand into the B register.   **Immediate Mode:** Copy the 8-bit immediate source operand to the B register. |
| Operation | **Direct Mode** B ← memory (OPERAND)  **Indirect Mode:**  B ← Memory{ memory( OPERAND) }   **Immediate Mode:** B ← IMMED OPERAND |
| Memory Addressing | Direct, Immediate |

### STORE B Register to RAM

|  |  |
| --- | --- |
| Instruction | STB |
| OPCode | 10100 |
| Flags | 00: Direct 01: Indirect |
| Format | STORE FLAG OPERAND |
| Format Machine | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 10100 | 000 001 | MEM ADDR | |
| Description | **Direct Mode** Copy the B register contents to the destination memory address specified by the operand **Indirect Mode:** Copy the B register contents to the destination memory address found at the memory address of the operand |
| Operation | **Direct Mode** memory(OPERAND) ← B **Indirect Mode:** Memory{ memory( OPERAND) } ← B |
| Memory Addressing | Direct, Indirect |

### INPUT Data Word to RAM

|  |  |
| --- | --- |
| Instruction | INPUT |
| OPCode | 01100 |
| Flags | 00: Direct 01: Indirect |
| Format | INPUT FLAG OPERAND |
| Format Machine | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 01100 | 000 001 | MEM ADDR | |
| Description | Input a data word to RAM |
| Operation | **Direct Mode** memory(OPERAND) ← I/O Port  **Indirect Mode:** Memory{ memory( OPERAND) } ← I/O Port |
| Memory Addressing | Direct, Indirect |

### OUTPUT Data Word from RAM

|  |  |
| --- | --- |
| Instruction | OUTPUT |
| OPCode | 01101 |
| Flags | 00: Direct 01: Indirect |
| Format | OUTPUT FLAG OPERAND |
| Format Machine | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 01101 | 000 001 | MEM ADDR | |
| Description | Output data word from RAM |
| Operation | **Direct Mode** memory(OPERAND) → I/O Port  **Indirect Mode:** Memory{ memory( OPERAND) } → I/O Port |
| Memory Addressing | Direct, Indirect |

### LOAD Mask Register of HVPI

IN PROGRESS

|  |  |
| --- | --- |
| Instruction | LOAD Mask Register of HVPI |
| OPCode | 01110 |
| Flags |  |
| Format |  |
| Description |  |
| Operation |  |
| Memory Addressing |  |

### NOP

|  |  |
| --- | --- |
| Instruction | NOP |
| OPCode | 01111 |
| Flags | NULL |
| Format | NOP |
| Format Machine | |  |  |  |  | | --- | --- | --- | --- | | Bits | 15:11 | 10:8 | 7:0 | | Field | OPCODE | FLAG | OPERAND | | Values | 01111 | 000 | 00000000 | |
| Description | Performs no operation. |
| Operation | NULL |
| Memory Addressing | NULL |

# Accumulator Architecture Development

## Architecture Overview

Figure 2 provides a single cycle diagram highlighting the accumulator processor. The hardware architecture has been designed to support the operations available in the instruction set architecture. Each of the data paths (wires) in Figure 2 are 8-bits wide. Control point and select lines are not drawn out in Figure 2 in interest of preserving space. Two memories (RAM) are used to store program data and the instruction data (programs to be executed).

Multiplexers are used to select what data exists on varying data busses at a given time. It is not desirable to have multiple data trying to output onto one bus at the same time, the current value would be unknown. Each of the multiplexer’s select lines are attached to the controller via control point lines (CP). Depending upon the opcode interpreted by the controller, the control points are set to enable the correct data to be passed to the desired function units.

It is preferred for each instruction to have a minimal cycle time, enabling the processor to execute each instruction more quickly. Every use of the ALU function unit leaves the new result stored in the accumulator register, ACC. Operations to use, load, and store the ACC register contents should be easily accessible, meaning available in very few clock cycles. The layout of the single cycle processor enables very short paths to the next functional unit. Memory access is an expensive operation, to simulate this – memory operations loop 8 clock cycles before returning results. To avoid constantly running realizing the burden of memory accesses, a direct-mapped cache is implemented. The 4 word cache mimics the L1 or L2 cache on a modern system. If the target memory address exists in the cache, the data can be returned without experiencing the 8 clock cycle delay.



Figure 2 - ACC Processor Single Cycle Diagram

One ALU exists in the single cycle accumulator processor design. ALU operations must always include the ACC register as an input, and results are stored back into the ACC register. Operations enabling two inputs (Add, Subtract, etc.) allow input from the A, B, and MDR registers. A, B, and ACC may be loaded three different ways – direct memory addressing, indirect memory addressing, and using immediate addressing. A shifter is connected to the ACC register and enables the CPU to be programmed to shift the contents of the ACC register to the left or right by bits. Current contents of the A, B, and ACC registers may be saved into the processor’s data RAM using direct and indirect addressing schemes.

Interrupts are emitted and handled using a Masked Hardware Vectored Priority Interrupt System (MHVPIS). Divide by zero, overflow, input/output, and illegal Op-code instructions are all occurrences handled by the MHVPIS. When an interrupt is encountered, the MHVPIS points the program counter to the memory address for the interrupt service routine (ISR) to handle the interrupt.

Entering subroutines and interrupt service routines requires that the current value of the ACC register and PC be saved. Two stacks are introduced, PC Stack and Accumulator Stack, to handle pushing the current register contents before entering the subroutine. When an instruction to return from subroutine is encountered the top of the stack is popped off and placed into the respective register.

## Instruction Implementation



### Add



Figure 3 - ADD Flow Diagram

### Sub



Figure 4 - SUB Flow Diagram

### Logical OR



Figure 5 - OR Flow Diagram

### Logical AND



Figure 6 - AND Flow Diagram

### Logical Complement (COMP)



Figure 7 - COMP Flow Diagram

### Multiply and Divide



Figure 8 - MULT Flow Diagram



Figure 9 - DIV Flow Diagram

### Arithmetic Left/Right Shift



Figure 10 - SHFT Flow Diagram

### Conditional Branch



Figure 11 - BRA Flow Diagram

### Unconditional Jump



Figure 12 - JMP Flow Diagram

### Jump/Return to/from a Subroutine



Figure 13 - RTS Flow Diagram

### LOAD Accumulator from RAM



Figure 14 - LOAD Flow Diagram

### STORE Accumulator to RAM



Figure 15 - STORE Flow Diagram

### LOAD A Register from RAM



Figure 16 - LDA Flow Diagram

### STORE A Register to RAM



Figure 17 - STA Flow Diagram

### LOAD B Register from RAM



Figure 18 - LDB Flow Diagram

### STORE B Register to RAM



Figure 19 - STB Flow Diagram

### INPUT Data Word to RAM



Figure 20 - INPUT Flow Diagram

### OUTPUT Data Word from RAM



Figure 21 - OUTPUT Flow Diagram

### LOAD Mask Register of HVPI

IN PROGRESS

### NOP



Figure 22 - NOP Flow Diagram

## Direct Mapped Cache



Figure 23 - Direct Mapped Cache Flow Diagram

# Conclusion

# References