ENGG 3380: Computer Organization and Design

Project Phase Two

Group #: 6	
Name	**Student ID**
Elijah Johnston	0961152
Thomas Green	1048389
Lorent Aliu	1058897

Contents

Overview

To begin, before designing the control path it was necessary to design the overall microprocessor, as this higher-level design dictates what control signals exists and hence the contents/layout of the microcode store; Figure 1 shows the block diagram of such. Evidently, this design is quite similar to that of the MIPS 32 architecture. There are a few changes, however. To begin, there exists “special register file” and “B selector”, as will be further discussed under “Changes from Phase 1” section. Furthermore, there is no need for an ALU control unit, as all its function can be implemented using the microcode store. Finally, there is no logic needed for the computation of the jump address, as the 12 bit “target” field represents an address directly. This design uses 8-bit registers and 16-bit memory, as outlined in phase 1.

ALU Configuration

High Level Design

The ALU designed for the purpose of this project is very similar to that created for lab 4. To begin, it is split into the following blocks: An arithmetic unit, a logic unit, a conditional unit, and a “master mux” (figure 2). The three main units compute their respective results on any given clock cycle, while the master mux chooses which one to output provided what the select lines are set to.

The Arithmetic Unit

The arithmetic unit uses cascading 1-bit units to perform all its main operations (figure 3). For addition, the unit operates as a typical ripple carry adder would. For subtraction operations, operand B is complemented (as it is XOR’d with S0 = 1), and S0 acts as a carry in. The overflow detector is much simpler than the one implemented for lab 4. Carry 7 and 8 are XOR’d together to detect whether there is overflow for signed operations (this means that the result’s sign is different from that of the operands, which have identical signs). For unsigned operations, we are simply interested in whether the final carry out is 1. Finally, status signal zero (used in BEQ) is implemented by NORing all of the result bits.

The Logic Unit

The logic unit takes advantage of eight 1-bit logic units to perform its operations (figure 6). Each 1-bit unit is simply a 3-1 mux (figure 7). The inputs of the mux represent each operation (ADD, OR and XOR). The select lines S1 and S0 choose which operation to perform. Each of the outputs from the eight 1-bit logic units are then sent to the master mux (figure 2).

The Conditional Unit

The conditional unit is simply responsible for SLT.

The Master Mux

The master MUX is in charge of selecting the final output between the three buses from the other main units. This is done with the help of the 3 select lines.

Control Unit Configuration

The control unit is also very similar to the one designed for lab 5 (figure 6). It is comprised of the following components: Microprogram address decoder, 8-bit ripple carry adder, 3-1 MUX, microcode store, and micro PC. The main components will be further discussed in the subsequent sections.

Instruction Register

The instruction register (IR) is loaded with the contents of the first 4 bits of an instruction from memory (also known as the Opcode).

Microprogram Address Decoder

This component takes in the contents of the IR, concatenates 4 zeros and outputs this value to the 3-1 MUX. This represents the address of the first microinstruction of a given microprogram and is loaded into micro PC when appropriate.

3-1 MUX

The 3-1 MUX dictates the contents of the micro PC for the subsequent clock cycle. Part of the microinstruction layout is dedicated to acting as a select for this MUX. Given this, microinstructions that have a subsequent instruction make the MUX select output from the 8-bit ripple carry adder (PC + 1). The last microinstruction of a microprogram makes the MUX select output “00000000”, which allows the control unit to undergo the fetch phase. The very first address of the microcode store is known as the fetch instruction, and makes the MUX output the signal from the Microprogram address decoder (the address of the first micro instruction).

8-bit Ripple Carry Adder

This component simply increments the contents of micro PC by 1, which represents the address of a subsequent micro instruction.

Microcode Store

The microcode store contains the sequence of control bits for every micro program (i.e machine instructions). For the purpose of this project, it will have 3 bits for internal control signals, and 17 control signals for the Datapath.

Changes from Phase 1

Special Register File

Since phase 1, it has become apparent that there is no reason to have 3 bits for the function field in R-type instructions, as control signals for the ALU can be stored directly in the microcode store:

OpCode	Rs	Rt	Rd	Func
4 bits	3 bits	3 bits	3 bits	3 bits

Given this, it is possible to use these 3 bits for something else. For the updated design, these 3 bits will be used to address registers within a "special" register file (figure 7). These registers will be read-only and will contain the following values: register 0 will contain 0, register 1 will contain 1, register 2 will contain 2^2 = 4 and so on. The new R-type instruction is the following:

OpCode	Rs	Rt	Rd	Rspecial
4 bits	3 bits	3 bits	3 bits	3 bits

In order to access this special register file, Rt will have to be set to "111", thereby replacing the contents going into register B with the specified special register. This is possible through the component "B selector" (figure 1). This means that Rt will not be able to access register 7 in the register file but will be able to access the special register file. Contrary to this, Rs will not be able to access the special register file but will be able to access register 7

Other Special Purpose Registers

Because the overall microprocessor design is based off the MIPS architecture presented in class, there will be the following special purpose registers for storing values between clock cycles: PC IR, MDR, A, B, and ALUout. The contents of PC will point to the current instruction stored in memory and will either be incremented by 2, incremented by 2 + branch, or set to the target field in J type instruction. The instruction register (IR) will store the current instruction that PC is pointing to in memory. Registers A and B will store the values of Rs and Rt/Rspecial respectively. Finally, ALUout will store the output from the ALU. Figure 7 goes into more detail in terms of what these registers are set to on given microinstructions.

General Purpose Registers

In this design, there is a total of eight general purpose registers. Named R0-R7, they reside in the main register file and all have read/write access depending on the register file's control signals (RegWrite).

Appendix

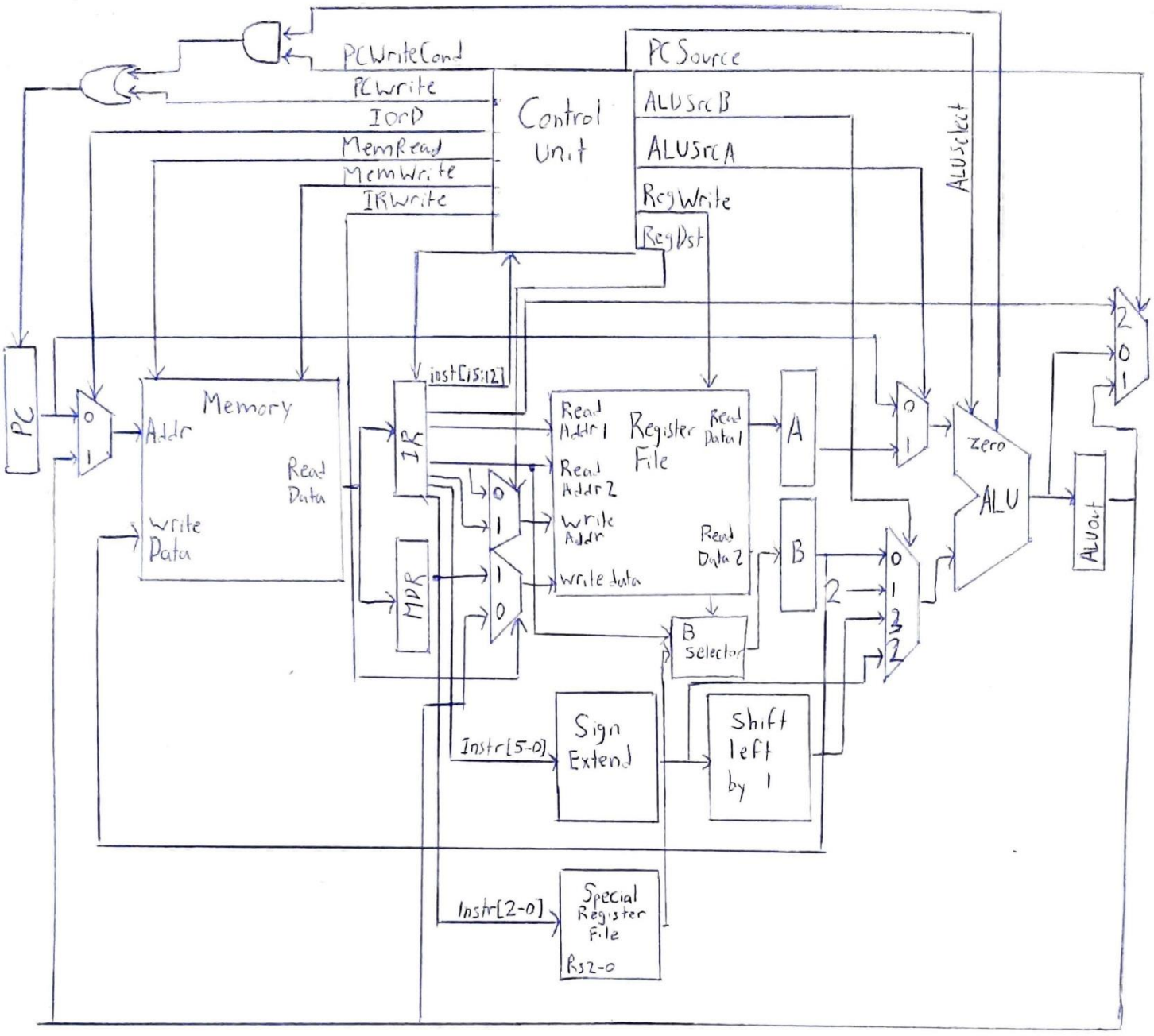

Figure 1: High level block diagram for the microprocessor.

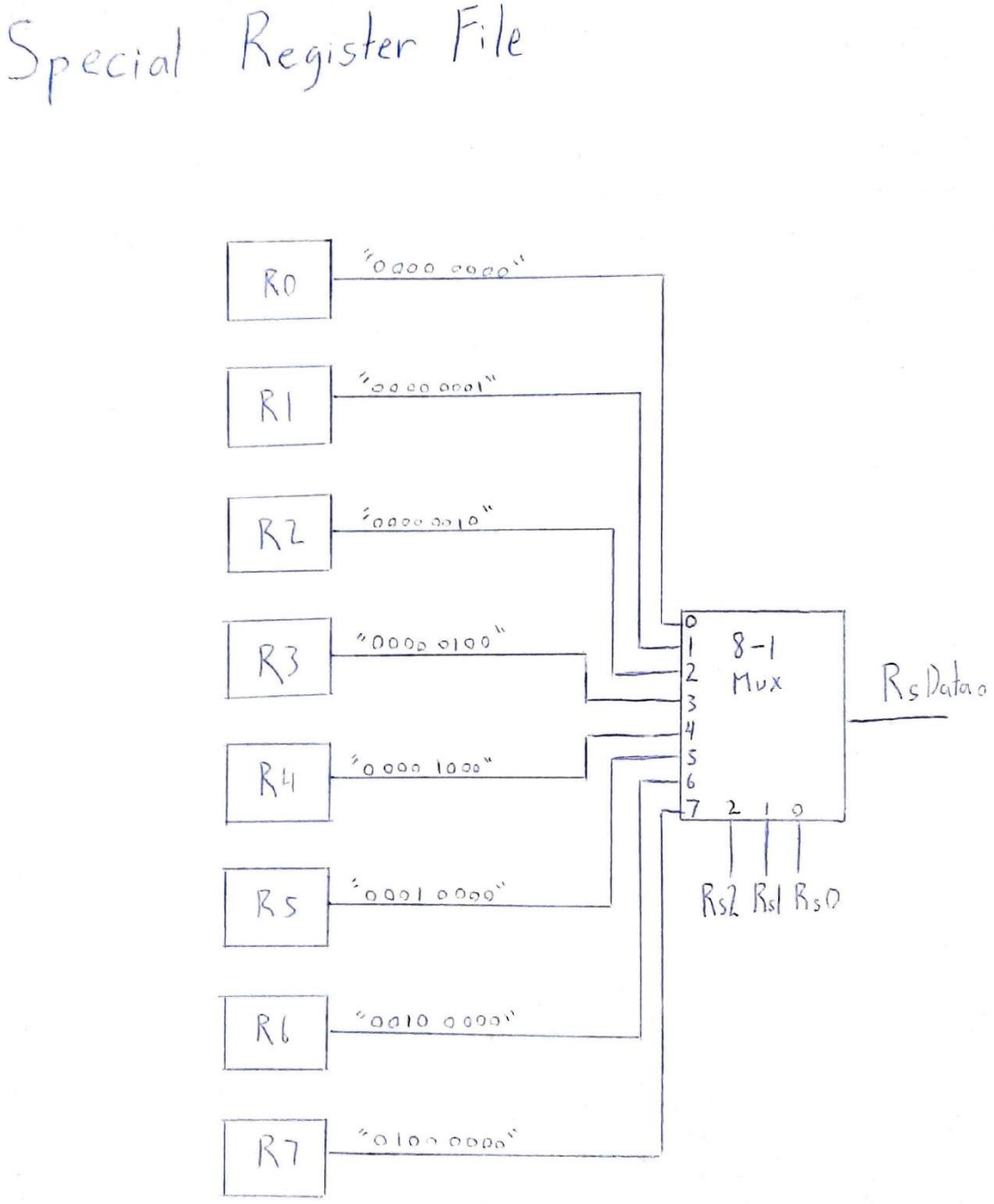

Figure 2: Block diagram for the special register file. These registers are read only. The 8-1 mux outputs the register specified by the Rspecial field.

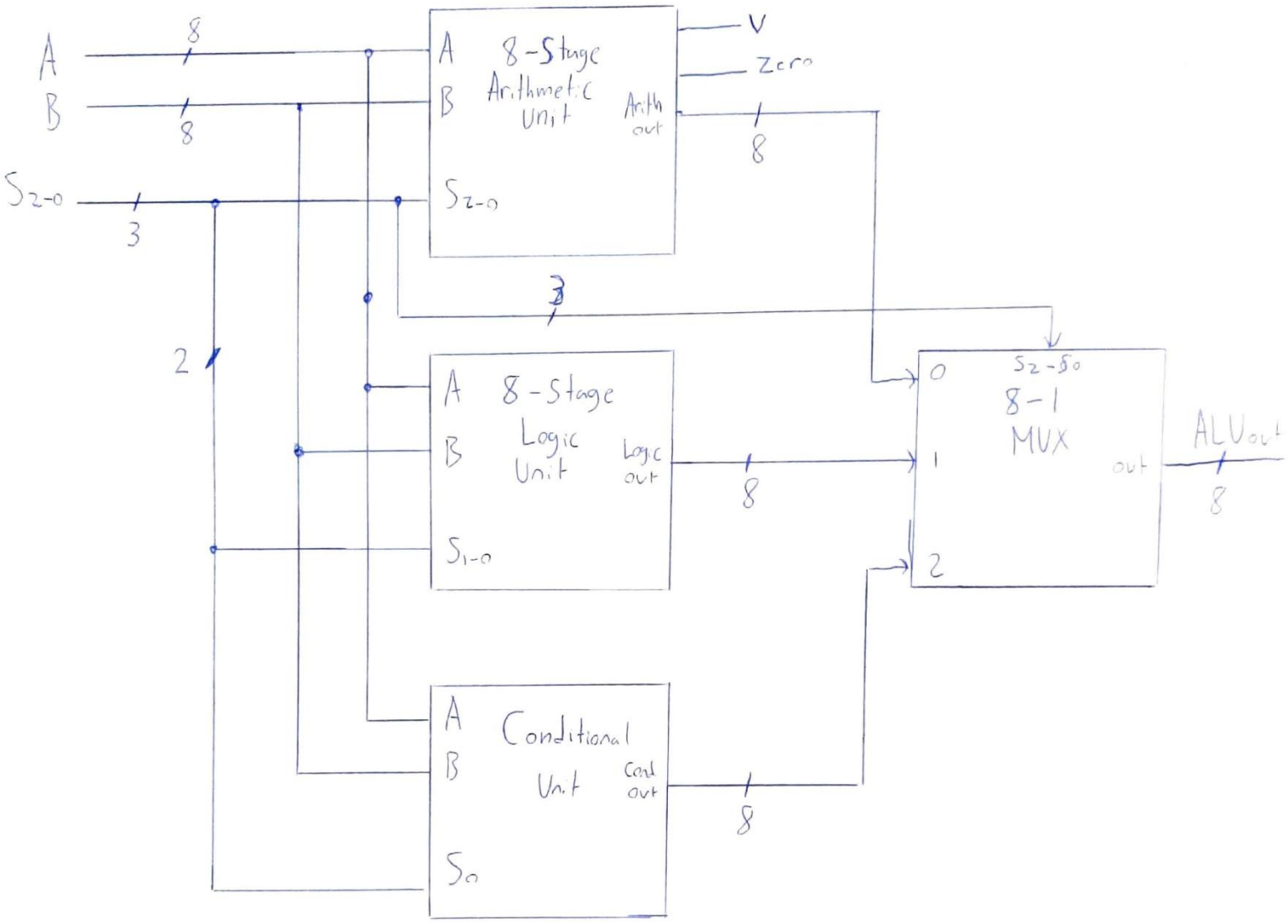

Figure 3: Block diagram for the ALU.

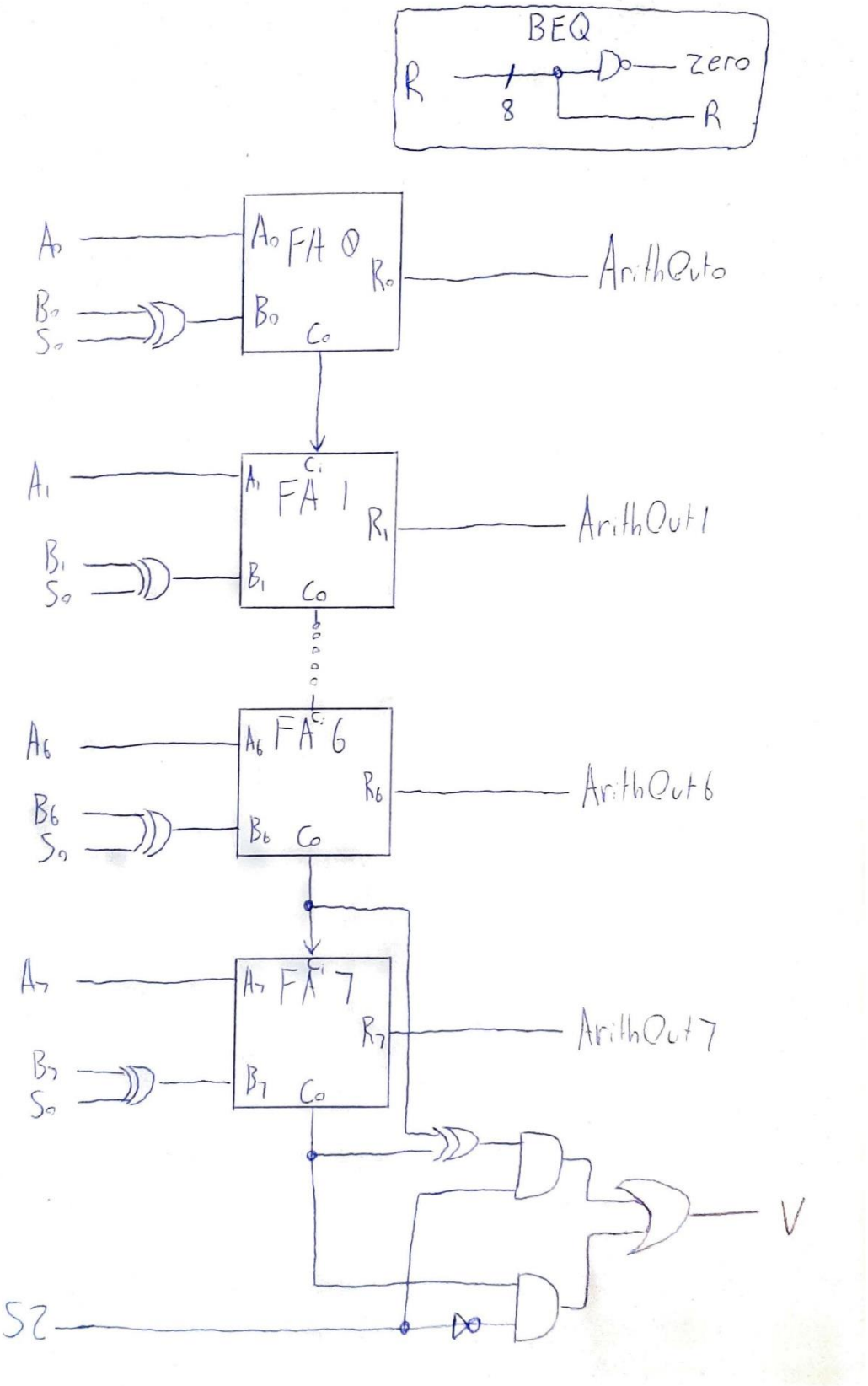

Figure 4: Block diagram for the arithmetic unit. Included is the logic for the zero signal as well as the overflow signal.

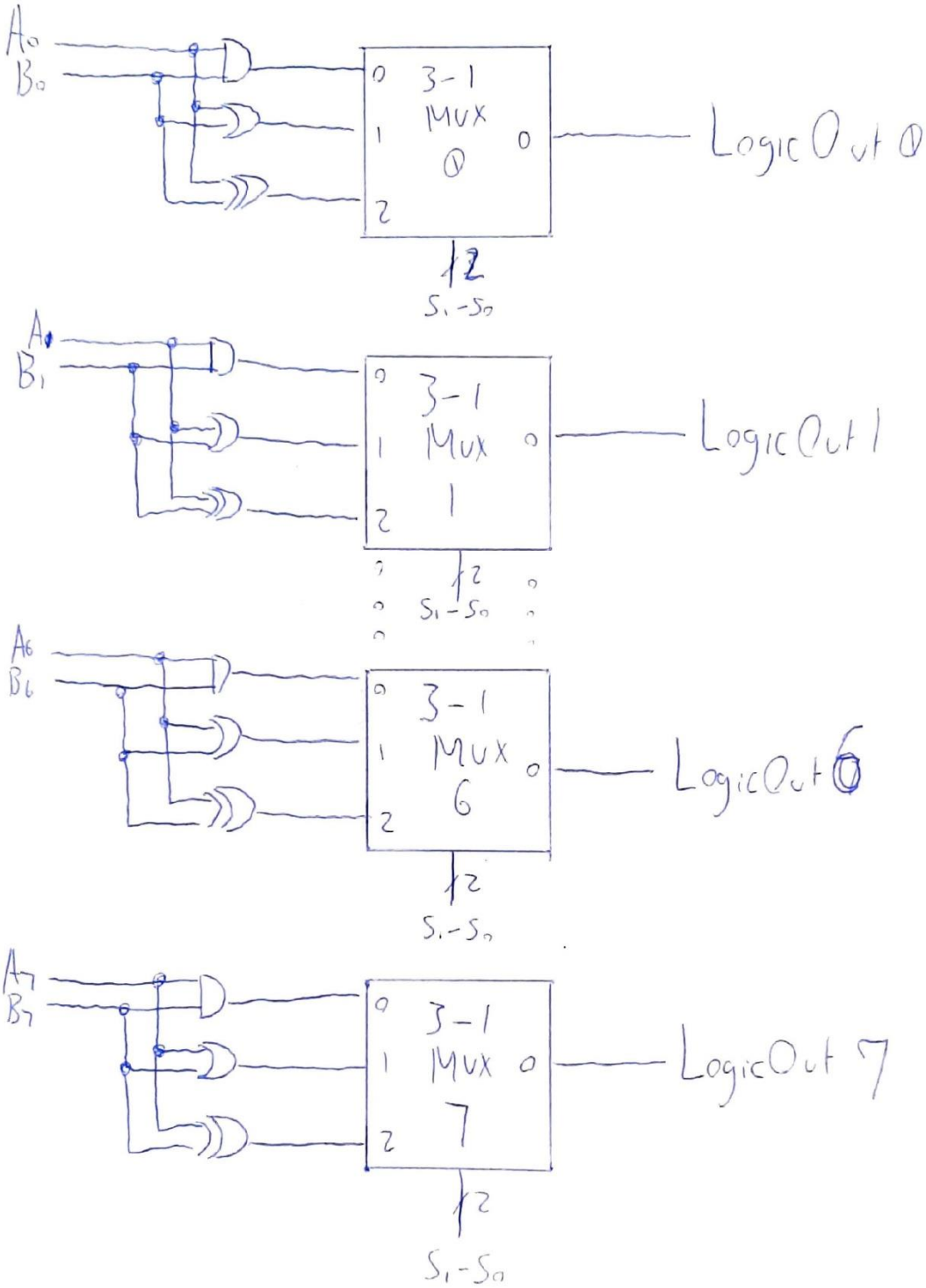

Figure 5: Block diagram for the logic unit.

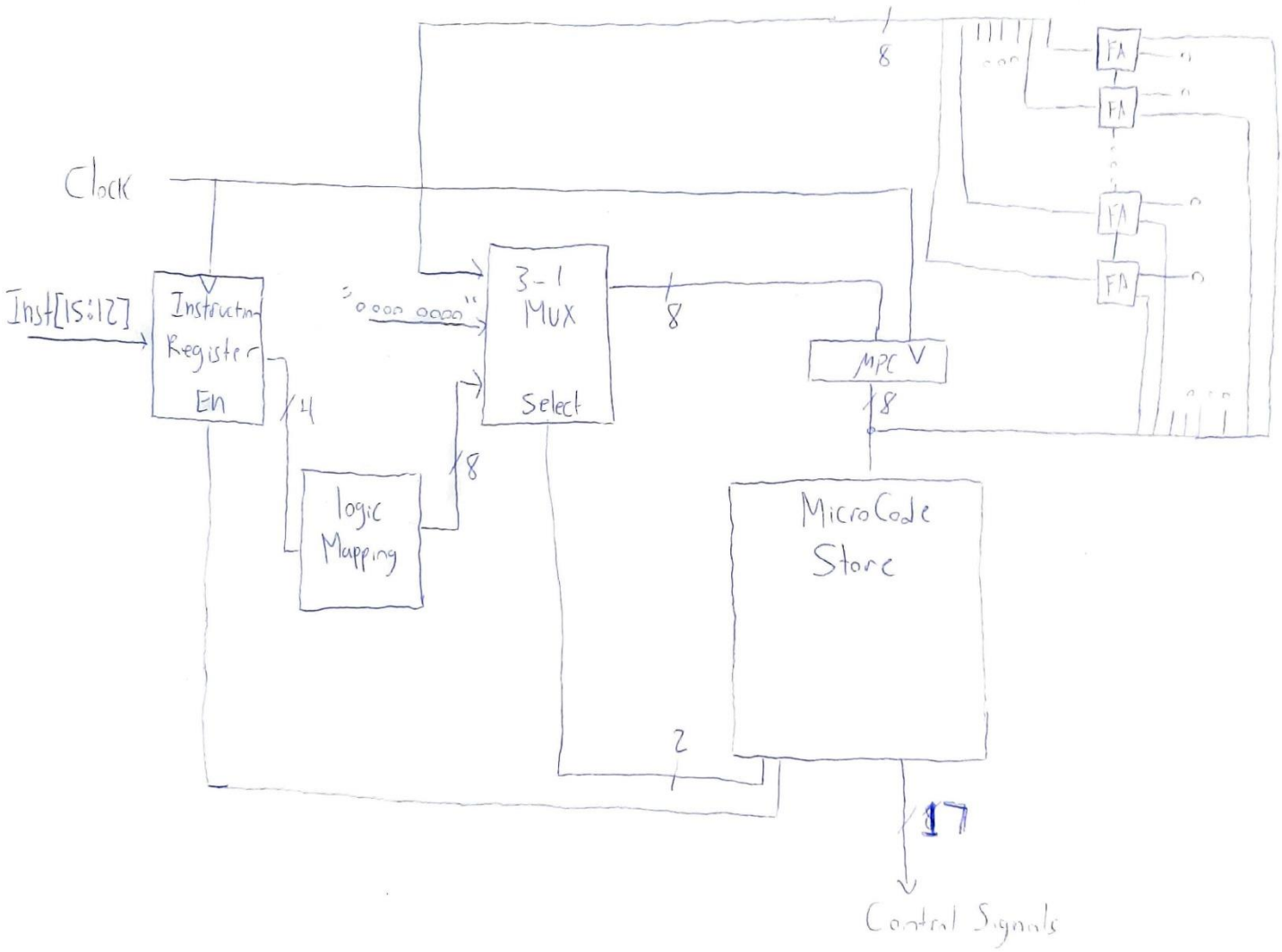

Figure 6: Block diagram for the control unit.

Opcode	Function/Step	Micro IR enable (for intern	PcSource	ALU OP	ALUSourceA	ALUSourceB	RegWrite	RegDest	PCWriteCond	PCWrite	IorD	MemRead	MemWrite	MemToReg	IRWrite
0000	INSTRUCTION FETCH	1	00	010 (add)	0	01	0	X	X	1	0	1	0	X	1
	DECODE	0	X	010 (add)	0	11	0	x	0	0	X	X	0	X	0
0001	OR - Compute	0	X	001	1	00	0	X	0	0	X	X	0	X	0
	OR - Store	0	X	XXX	X	XX	1	1	0	0	X	X	0	0	0
0010	Unsigned Add - Compute	0	X	010	1	00	0	X	0	0	X	X	0	X	0
	Unsigned Add - Store	0	X	XXX	X	XX	1	1	0	0	X	X	0	0	0
0011	XOR - Compute	0	X	011	1	00	0	X	0	0	X	X	0	X	0
	XOR - Store	0	X	XXX	X	XX	1	1	0	0	X	X	0	0	0
0100	Signed Add - Compute	0	X	100	1	00	0	X	0	0	X	X	0	X	0
	Signed Add - Store	0	X	XXX	X	XX	1	1	0	0	X	X	0	0	0
0101	Signed Subtract - Compute	0	X	101	1	00	0	X	0	0	X	X	0	X	0
	Signed Subtract - Store	0	X	XXX	X	XX	1	1	0	0	X	X	0	0	0
0110	SLT - Compute	0	X	110	1	00	0	X	0	0	X	X	0	X	0
	SLT - Store	0	X	XXX	X	XX	1	1	0	0	X	X	0	0	0
0111	BEQ	0	01	111	1	00	0	X	1	0	X	X	0	X	0
1000	AND - Compute	0	X	000	1	00	0	X	0	0	X	X	0	X	0
	AND - Store	0	X	XXX	X	XX	1	1	0	0	X	X	0	0	0
1001	J	0	10	XXX	X	XX	0	X	X	1	X	X	0	X	0
1010	ADDIU - Compute	0	X	010	1	10	0	X	0	0	X	X	0	X	0
	ADDIU - Store	0	X	XXX	X	XX	1	1	0	0	X	X	0	0	0
1011	SW - Excecute	0	X	011	1	10	0	X	0	0	X	X	0	X	0
	SW - Memory Access	0	X	XXX	X	XX	0	X	0	0	1	X	1	X	0
1100	ADDI - Compute	0	X	100	1	10	0	X	0	0	X	X	0	X	0
	ASSI - Store	0	X	XXX	X	XX	1	1	0	0	X	X	0	0	0
1101	SUBI - Compute	0	X	101	1	10	0	X	0	0	X	X	0	X	0
	SUBI - Store	0	X	XXX	X	XX	1	1	0	0	X	X	0	0	0
1110	SLTI - Compute	0	X	110	1	10	0	X	0	0	X	X	0	X	0
	SLTI - Store	0	X	XXX	X	XX	1	1	0	0	X	X	0	0	0
1111	LW - Excecute	0	X	111	1	10	0	X	0	0	X	X	0	X	0
	LW - Memory Access	0	X	XXX	X	XX	0	X	0	0	1	1	0	X	0
	LW - Write Back	0	X	XXX	X	XX	1	0	0	0	X	X	0	1	0

Opcode	Function/Step	Comment
0000	INSTRUCTION FETCH	Read instruction from memory (MemRead = 1), load it into IR)IRWrite = 1), increment PC by 2 (ALUsource = 01(2), ALUOP = 010 (ADDU), PCSource = 00 (ALUout), PCWrite = 1)
	DECODE	Read register rs and rt into A and B, as well as address for branch into ALUout **IN CASE** it is needed for later
0001	OR - Compute	Perform the specified R type instruction (using operands A and B), store the result into ALUout regiter for next step
	OR - Store	Store the contents from ALUout (from the previous step) into the specified register
0010	Unsigned Add - Compute	Perform the specified R type instruction (using operands A and B), store the result into ALUout regiter for next step
	Unsigned Add - Store	Store the contents from ALUout (from the previous step) into the specified register
0011	XOR - Compute	Perform the specified R type instruction (using operands A and B), store the result into ALUout regiter for next step
	XOR - Store	Store the contents from ALUout (from the previous step) into the specified register
0100	Signed Add - Compute	Perform the specified R type instruction (using operands A and B), store the result into ALUout regiter for next step
	Signed Add - Store	Store the contents from ALUout (from the previous step) into the specified register
0101	Signed Subtract - Compute	Perform the specified R type instruction (using operands A and B), store the result into ALUout regiter for next step
	Signed Subtract - Store	Store the contents from ALUout (from the previous step) into the specified register
0110	SLT - Compute	Perform the specified R type instruction (using operands A and B), store the result into ALUout regiter for next step
	SLT - Store	Store the contents from ALUout (from the previous step) into the specified register
0111	BEQ	Branch using the contents of ALUout (from decode stage) if the contents of A and B are the same.
1000	AND - Compute	Perform the specified R type instruction (Using operand A and the value in the specified special register), store the result into ALUout regiter for next step
	AND - Store	Store the contents from ALUout (from the previous step) into the specified register
1001	J	Load PC with the contents of INSTR[11-0]
1010	ADDIU - Compute	Perform the specified I type instruction (using operands A signextended immediate), store the result into ALUout regiter for next step
	ADDIU - Store	Store the contents from ALUout (from the previous step) into the specified register
1011	SW - Excecute	Compute the addition of the base address (A) and immediate, store it in ALUout
	SW - Memory Access	Store the contents of ALUout (from previous step) into the address stored in register B.
1100	ADDI - Compute	Perform the specified I type instruction (using operands A signextended immediate), store the result into ALUout regiter for next step
	ASSI - Store	Store the contents from ALUout (from the previous step) into the specified register
1101	SUBI - Compute	Perform the specified I type instruction (using operands A signextended immediate), store the result into ALUout regiter for next step
	SUBI - Store	Store the contents from ALUout (from the previous step) into the specified register
1110	SLTI - Compute	Perform the specified R type instruction (using operands A and B), store the result into ALUout regiter for next step
	SLTI - Store	Store the contents from ALUout (from the previous step) into the specified register
1111	LW - Excecute	Compute the addition of the base address (A) and immediate, store it in ALUout
	LW - Memory Access	Store the contents of ALUout (from previous step) into the MDR register
	LW - Write Back	Store the contents inside the MDR register into the specified register (as specified by Rs)

Figure 7: Contents of the microcode store (above). Comments outlining the function of each micro instruction (below).