ENGG 3380: Computer Organization and Design

Project Phase One

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

Contents

Register Configuration

For this project, the microprocessor will contain 8 general purpose registers. To easily implement the components designed in the labs, the registers will be 8 bits wide. Given that there are 8 registers, this means that 3 address lines will be used when addressing registers in the machine code, as 2^(3 address line) = 8 destinations. Also, this means that an 8-bit data bus will be designed, as this is the length of each register.

Memory Configuration

If the design were to be consistent with the MIPS architecture, this would mean that the memory would be 8 bits wide, as are the registers. However, given that 8 bits will not be long enough to store instructions, 16-bit memory will be used in this design. In this architecture, we will simplify the jump format such that the 12-bit immediate is a direct address. Given this, and that the J format's immediate is 12 bits, we have 2^12 = 4K memory addresses. Given that each memory address is 16 bits, we have a memory size of 4K * 2B = 8KB.

Defining the Machine Code Format

Like MIPS, there will be 3 different machine code formats, namely: R, I, and J formats. The following are diagrams that help show how each machine code format will be organized:

R format

Opcode	Rs	Rt	Rd	Funct
4 bits	3 bits	3 bits	3 bits	3 bits

I format

Opcode	Rs	Rt	Immediate
4 bits	3 bits	3 bits	6 bits

J format

Opcode	Target
4 bits	12 bits

Defining the Instruction Set

The following table describes the instructions that this microprocessor will be able to execute. These instructions will allow for the implementation of the code that generates the Fibonacci sequence.

Category	Mnemonic	Syntax of Operation	RTL	Machine Format	Status signals
Arithmetic	ADD	add Rd, Rs, Rt	Rd <- Rs + Rt	R	OVF
Arithmetic	SUB	sub Rd, Rs, Rt	Rd <- Rs – Rt	R	OVF
Arithmetic	ADDI	addi Rd, Rs, #6 bits	Rd <- Rs + #6 bits	I	OVF
Arithmetic	SUBI	subi Rd, Rs, #6 bits	Rd <- Rs - #6 bits	I	OVF
Arithmetic	ADDU	addu Rd, Rs, Rt	Rd <- Rs + Rt	R	OVF
Arithmetic	SUBU	subu Rd, Rs, Rt	Rd <- Rs - Rt	R	OVF
Arithmetic	ADDIU	addi Rd, Rs, #6 bits	Rd <- Rs + #6 bits	I	OVF
Arithmetic	SUBIU	subi Rd, Rs, #6 bits	Rd <- Rs - #6 bits	I	OVF
Logical	AND	and Rd, Rs, Rt	Rd <- Rs ^ Rt	R	
Logical	OR	or Rd, Rs, Rt	Rd <- Rs ᐯ Rt	R	
Logical	XOR	xor Rd, Rs, Rt	Rd <- Rs ⊕ Rt	R	
Logical	ANDI	andi Rd, Rs, #6 bits	Rd <- Rs ^ #6 bits	I	
Logical	ORI	ori Rd, Rs, #6 bits	Rd <- Rs ᐯ #6 bits	I	
Logical	XORI	xori Rd, Rs, #6 bits	Rd <- Rs ⊕ #6 bits	I	
Memory Reference	LW	lw Rd, #6 bits(Rs)	Rd <- offset(Rs)	I	
Memory Reference	SW	sw Rd, #6 bits(Rs)	offset(Rs)<- Rd	I	
Control	BEQ	beq Rd, Rs, #6 bits	Cond <- Rs – Rt If (Cond eq 0) PC <- PC + 4 + (Immediate)	I	ZERO

			else PC <- PC + 4		
Control	SLT	slt Rd, Rs, Rt	Cond <- Rs – Rt If (Cond < $Zero) Rd <- 0X F Else Rd <- 0X 0	R	
Control	SLTI	slti Rd, Rs, #6 bits	Cond <- Rs – Immediate If (Cond < $Zero) Rd <- 0X F Else Rd <- 0X 0	I	
Control	SLTU	sltu Rd, Rs, Rt	Cond <- Rs – Rt If (Cond < $Zero) Rd <- 0X F Else Rd <- 0X 0	R	
Control	SLTIU	sltiu Rd, Rs, #6 bits	Cond <- Rs – Immediate If (Cond < $Zero) Rd <- 0X F Else Rd <- 0X 0	I	
Control	J	j #12 bits	PC <- Target	J	

ALU Configuration

A similar ALU to that designed in lab 4 will be implemented. The ALU will be able to undergo the following functions, given the specified control signals:

ALU Control Line	Function
000	AND
001	OR
010	Unsigned Add
011	XOR
100	Signed Add
101	Signed Subtract
110	Set on Less Than
111	Branch if equal

Memory Mapped I/O

The CPU and I/O devices will share the same address space within the 4KB memory. Hence, I/O operations will be done through instructions of type “memory reference” (see above) performed within the memory map. The following is a diagram that represents how memory for I/O will be mapped:

Address	Contents
7998	Allocated for otherwise
…	Allocated for otherwise
8	Allocated for otherwise
6	LEDs
4	Push Buttons
2	Switches
0	Seven Segment Display