EE 320

Computer Organization

Course Project Final Report

Submitted to

Dr. Jahangir Ikram, Shahrukh Athar

{jikram, shahrukh.athar}@lums.edu.pk

Submitted by

Muhammad Ahmed Riaz, Muhammad Khurram Shahzad,

Muhammad Safdar Iqbal, Muhammad Aqeel Raza

{12100035, 12100001, 12100005, 12100079}@lums.edu.pk

Original Proposal

Instructions Implemented

R-type

- ADD
- AND
- OR
- SUB
- JR

I-type

- ADDI
- ANDI
- ORI
- LW
- SW
- BEQ
- BNE

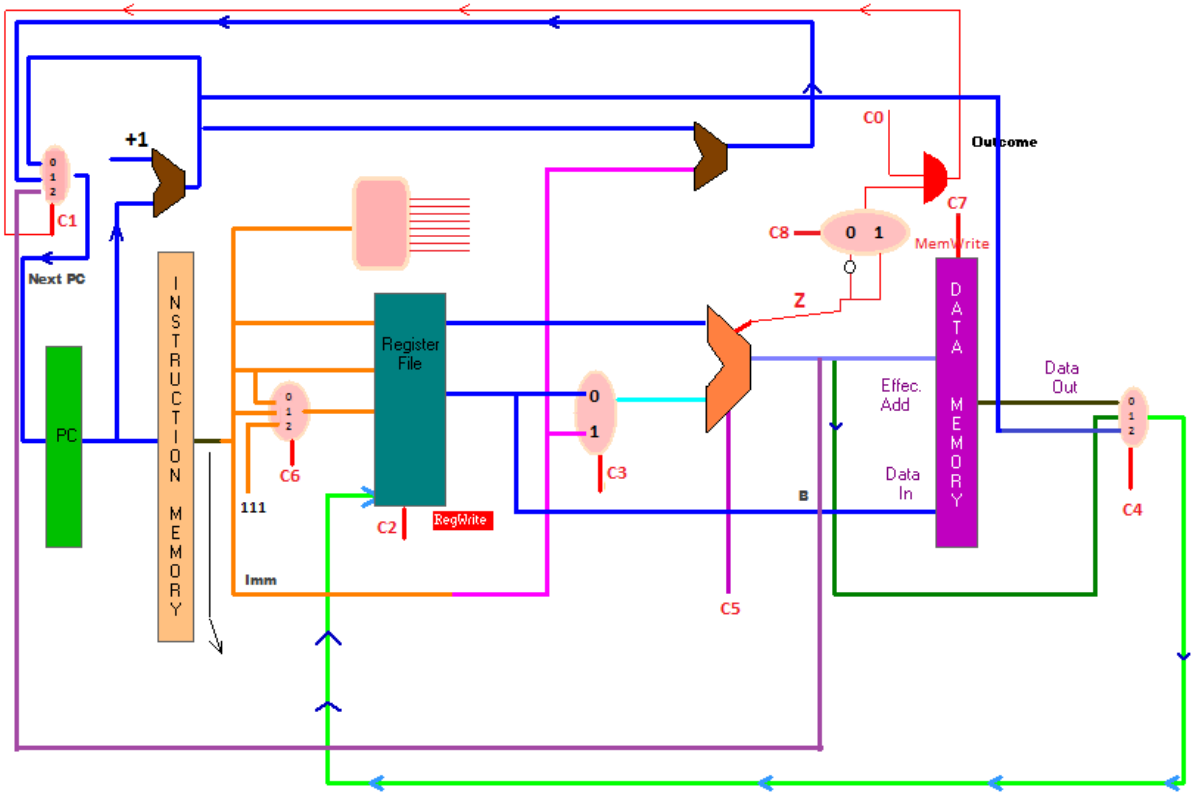
J-type

- JAL
- J

Instruction Detail

ADD Rd, Rs, Rt	Rd=Rs+Rt	0000	R-Type
AND Rd, Rs, Rt	Rd=Rs & Rt	0001	R-Type
SUB Rd, Rs, Rt	Rd=Rs – Rt	0010	R-Type
OR Rd, Rs, Rt	Rd=Rs | Rt	0011	R-Type
JR Rt	PC=Rt ; Rs=X ; Rd=X	0100	R-Type
ADDI Rt, Rs, Imm	Rt=Rs+Imm	0111	I-Type
ANDI Rt,Rs,Imm	Rt=Rs & Imm	1000	I-Type
ORI Rt, Rs, Imm	Rt=Rs | Imm	1001	I-Type
LW Rt, Rs, Imm	Rt=Mem(Rs+Imm)	1010	I-Type
SW Rt, Rs, Imm	Mem(Rs+Imm)=Rt	1011	I-Type
BEQ Rt, Rs, Imm	If Rt==Rs: PC=PC+1+Imm	1100	I-Type
BNE Rt, Rs, Imm	If Rt!=Rs: PC=PC+1+Imm	1101	I-Type
JAL JumpAddr	PC=JumpAddr ; Ra=PC+1	0101	J-Type
J JumpAddr	PC=JumpAddr	0110	J-Type

Block Diagram


+1
C0
Outcome
C1
Next PC
C7
C8
0 1
MemWrite
Z
INSTRUCTION MEMORY
Register File
DATA MEMORY
PC
C6
111
C2
RegWrite
Imm
C3
B
C5
Effec. Add
Data In
Data Out
C4

Instruction Format

R-Type Instruction Set				
Opcode	**RS**	**RT**	**RD**	**DON'T CARE**
4-bit	3-bit	3-bit	3-bit	11-bit
I-Type Instruction Set				
Opcode	**RS**	**RT**	**Imm**	**DON'T CARE**
4-bit	3-bit	3-bit	8-bit	6-bit
J-Type Instruction Set				
Opcode	**DON'T CARE**		**Imm**	**DON'T CARE**
4-bit	6-bit		8-bit	6-bit

Control Unit Table

OPCODE	**Function**	**C0**	**C1**	**C2** RegWrite	**C3**	**C4**	**C5** ALU Function Select	**C6**	**C7** ~MemWrite	**C8**
0000	ADD	0	0	1	0	01	100110	01	1	0
0001	AND	0	0	1	0	01	10110X	01	1	0
0011	OR	0	0	1	0	01	1110X1	01	1	0
0010	SUB	0	0	1	0	01	011000	01	1	0
0111	ADDI	0	0	1	1	01	100110	00	1	0
1001	ORI	0	0	1	1	01	1110X1	00	1	0
1000	ANDI	0	0	1	1	01	10110X	00	1	0
1010	LW	0	0	1	1	00	100110	00	1	0
1011	SW	0	0	0	1	0X	100110	0X	0	0
1100	BEQ	1	0	0	0	0X	011000	0X	1	0
1101	BNE	1	0	0	0	0X	011000	0X	1	1
0101	J	0	1	0	1	0X	101001	0X	1	X
0110	JALR	0	1	0	0	0X	101001	0X	1	X

Changes during Implementation

Instructions Added/Modified

Instruction	Action	Opcode	Type
J JumpAddr	PC=JumpAddr	0101	J-Type
JAL JumpAddr	Ra=PC+1; PC=JumpAddr	1111	J-Type
JR Rt	PC=Rt;	1110	R-Type
JALR Rt	Ra=PC+1; PC=Rt	0110	R-Type

Conventions followed:

The following specials registers were reserved for several features we implemented

$R0	-	Temporary 1	($t0)
$R1	-	Temporary 2	($t1)
$R2	-	Temporary 3	($t2)
$R3	-	Function Argument 1	($a0)
$R4	-	Function Argument 2	($a1)
$R5	-	Function Output	($v0)
$R6	-	Stack Pointed	($sp)
$R7	-	Return Address	($ra)

Changes to Instruction Format

BYTE 1								BYTE 2								BYTE 3							
R-type format																							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Rt		Rs		Opcode				Don't Care			Rd			Rt		Don't Care							
0	2	1	0	3	2	1	0				2	1	0	2	1								
I-type format																							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Rt		Rs		Opcode				Immediate						Rt		Don't Care						Imm	
0	2	1	0	3	2	1	0	5	4	3	2	1	0	2	1							7	6
J-type format																							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Don't Care				Opcode				Immediate						DC		Don't Care						Imm	
				3	2	1	0	5	4	3	2	1	0									7	6

Changes to Control Unit

OPCODE	Function	C0	C1	C2 RegWrite	C3	C4	C5 ALU Function Select	C6	C7 ~MemWrite	C8
0000	ADD	0	0	1	0	01	100110	01	1	0
0001	AND	0	0	1	0	01	10110X	01	1	0
0011	OR	0	0	1	0	01	1110X1	01	1	0
0010	SUB	0	0	1	0	01	011000	01	1	0
0111	ADDI	0	0	1	1	01	100110	00	1	0
1001	ORI	0	0	1	1	01	1110X1	00	1	0
1000	ANDI	0	0	1	1	01	10110X	00	1	0
1010	LW	0	0	1	1	00	100110	00	1	0
1011	SW	0	0	0	1	0X	100110	0X	0	0
1100	BEQ	1	0	0	0	0X	011000	0X	1	0
1101	BNE	1	0	0	0	0X	011000	0X	1	1
0101	J	0	1	0	1	0X	101001	0X	1	X
0110	JALR	0	1	1	0	10	101001	10	1	X
1110	JR	0	1	0	0	0X	101001	0X	1	X
1111	JAL	0	1	1	1	10	101001	10	1	X

Problems

We faced the following problems.

PCB Procurement

We decided to build the processor hardware on PCBs instead of Veroboards. We were encouraged by Dr Jahangir Ikram as well. But the required double-sided PCBs were not available in the lab; neither were they procured in time. In the end, we had to procure them our self. (We are thankful to Dr Ikram for personally financing the procurement.)

Etching Machine problems

We tried to use the etching machine from the Physics Lab for fabrication of the double-sided PCB, but the machine was not properly calibrated with software used for designing the PCB. Hence we had to give up on the machine, after wasting 4 to 5 days trying to get it to work.

Conventional double-sided PCB etching (ironing method)

1. Aligning both the sides of the PCB while printing the circuit onto the PCB by ironing is a real challenge and it cost a lost effort and time.
2. Keeping the size of the circuit reasonable while using an appropriate width for tracks and pads are opposing goals.

Soldering on a PCB

While soldering jumpers, the copper wiring on the manually-etched PCB wore off. Therefore, we decided to give up using PCBs for the project, just 10 days before the deadline.

How to avoid PCB-related problems?

1. **Listen to Sir Shahrukh Athar. (He tried to stop us from using PCBs.)**
2. For PCB design, DipTrace is good software to use. Its auto-routing feature is really good.
3. Do not use Proteus for PCB design, especially its (horrible) auto-routing feature.
4. Tracks and pads in your design should be wider than at least 80 Th in the software.
5. If you plan to use an etching machine, make sure you know its use inside out *before* starting the project.

Dry Joints

We had to make sure each and every soldered connection was properly covered with metal, i.e. there were no dry joints in the circuit. This point cannot be over-emphasized.

ALU

During the testing of the ALU component, we were having a peculiar problem in which the ADD and AND instructions were working properly, but we were not able to do SUB and OR. The reason being that there is 1-bit difference between the OPCODEs of ADD and SUB (or AND and OR). That bit was FLOAT, and since float is always interpreted as HIGH, this was causing us problems.

ENABLE signals of MUXes

The ENABLE signals of some MUXes were not properly grounded due to dry joints, causing them to FLOAT.

Extra shorts

There were many extra shorts in the circuits (obviously causing many varied problems). This calls for proper short testing prior to interfacing.

Using Connectors

Instead of soldering GROUND, V_{CC} and CONTROL signal wires onto the Veroboard directly, we used connectors. The directly soldered wires repeatedly break out, causing GROUND, V_{CC} the CONTROL signals to FLOAT.

Labeling

During the interfacing phase, we usually kept forgetting what different sets of male pins and connectors did and how did they interconnect. Hence, we decided to label each input and output in every component of our processor, including labels for LSBs and MSBs.

Faulty trainers and Problems in Clocks

Due to prolonged use, many trainers in the EE lab have become faulty in many ways; the most common problems being faulty clock switches. They sometimes cause panic by sending spurious clocks to the circuit.

Synchronizing Register File clock with Rest of the Processor

We had given the main processor clock directly to every component and for register file we had ANDed the clock with RegWrite and then NOTed it. But NOT gate resulted in some delays that caused the

current instruction output to be written to the Register file in the current clock edge instead of the opposite edge as originally intended. So we gave the NOTed clock to the other components and the original one to the Reg file to fix this problem.