DIGITAL LOGIC DESIGN TERM PROJECT REPORT

AUTHORS

EKİN NOHUTÇU 150116067
FURKAN KUSE 150117041
METEHAN ERTAN 150117051

INTRODUCTION

PROBLEM DESCRIPTION:

We have designed a processors that can do AND, OR, ADD, LD, ST, ANDI, ORI, ADDI, XOR, XORI, JUMP, BEQ, BGT, BLT, BGE, BLE instructions. Our processor has 18 bits address width and 18 bits data width. Our processor has 16 registers. Data memory has 10 bits address width and 18 bits data width. Arithmetic Logic Unit computes arithmetic operations ADD, OR, XOR, AND, ADDI, ORI, XORI, ANDI.

For the first iteration of this project, we designed our Instruction Set Architecture (ISA). After creating ISA table, we wrote a Java code for assembler. Assembler should convert given mnemonics to the binary and hexadecimal codes.

ISATABLE:

[17:14]	[13:10]	[9:6]	[5:2]	[1:0]
AND 0000	R0-15 (DEST)	R0-15 (SRC1)	R0-15 (SRC2)	00 (UNUSED)
OR 0001	R0-15 (DEST)	R0-15 (SRC1)	R0-15 (SRC2)	00 (UNUSED)
ADD 0010	R0-15 (DEST)	R0-15 (SRC1)	R0-15 (SRC2)	00 (UNUSED)
XOR 0011	R0-15 (DEST)	R0-15 (SRC1)	R0-15 (SRC2)	00 (UNUSED)
ANDI 0100	R0-15 (DEST)	R0-15 (SRC1)	IMM (6 bits)	
ORI 0101	R0-15 (DEST)	R0-15 (SRC1)	IMM (6 bits)	
ADDI 0110	R0-15 (DEST)	R0-15 (SRC1)	IMM (6 bits)	
XORI 0111	R0-15 (DEST)	R0-15 (SRC1)	IMM (6 bits)	
BEQ 1000	R0-15 (OP1)	R0-15 (OP2)	ADDR (6 bits)	
BGT 1001	R0-15 (OP1)	R0-15 (OP2)	ADDR (6 bits)	
BLT 1010	R0-15 (OP1)	R0-15 (OP2)	ADDR (6 bits)	
BGE 1011	R0-15 (OP1)	R0-15 (OP2)	ADDR (6 bits)	
BLE 1100	R0-15 (OP1)	R0-15 (OP2)	ADDR (6 bits)	
LD 1101	R0-15 (DEST)	ADDR (10 bits)		
ST 1110	R0-15 (SRC1)	ADDR (10 bits)		
JUMP 1111	ADDR (14 bits)			

DESIGN EXPLANATION:

We have 16 different operands. We choose 4 bits to show opcode for each operand. Their opcodes are different than each other.

ASSEMBLER

First of all we imported libraries to read files and write into a file.

We used File and BufferWriter for reading from file and write into a file.

First, we checked the AND, OR, ADD and XOR operations and assign them opcodes which we designed in the ISA table. Then we split the instructions in the file and send them into a FirstCase method.

FirstCase method:

Takes instruction as separated strings, then converts them into binary form and concatenates them to the 18-bit binary output. At last, converts this 18-bit output to hexadecimal form and writes this value to the output file as string.

Then, we checked ANDI, ORI, ADDI and XORI instructions and assign them different opcodes as we designed in the ISA table.

After assigning them opcodes, we send instruction to a SecondCase method.

SecondCase method:

Takes instruction as separated strings, then converts them into binary form and concatenates them to the 18-bit binary output. At last, converts this 18-bit output to hexadecimal form and writes this value to the output file as string.

Then, we checked the BEQ, BGT, BLT, BGE and BLE instructions and assign them different opcodes.

After assigning them opcodes, we send the instruction into a ThirdCase method.

ThirdCase method:

Takes instruction as separated strings, then converts them into binary form and concatenates them to the 18-bit binary output. At last, converts this 18-bit output to hexadecimal form and writes this value to the output file as string.

Then, we checked the LD instruction and assign an opcode to this instruction.

After assign an opcode, we send the instruction into a FourthCase method.

FourthCase method:

Takes instruction as separated strings, then converts them into binary form and concatenates them to the 18-bit binary output. At last, converts this 18-bit output to hexadecimal form and writes this value to the output file as string.

After that, we checked the ST instruction. We assign a different opcode into this instruction.

After we assign an opcode into a ST instruction, we send the instruction into a FifthCase method.

FifthCase method:

Takes instruction as separated strings, then converts them into binary form and concatenates them to the 18-bit binary output. At last, converts this 18-bit output to hexadecimal form and writes this value to the output file as string.

Finally, we assigned a different opcode into a JUMP instruction as we designed in our ISA table.

Then we send the instruction into a SixthCase method.

SixthCase method:

Takes instruction as separated strings, then converts them into binary form and concatenates them to the 18-bit binary output. At last, converts this 18-bit output to hexadecimal form and writes this value to the output file as string.

concatBinary method:

It adds 0 till the given strings' length is the wanted length.

LOGISIM

HALF ADDER:

Adds two bits and carry-in. Returns sum of them and carry-out.

FULL ADDER:

Adds two bits. Returns sum of them and carry-out.

18-BIT ADDER:

Adds two 18-bit inputs bit by bit using full adders and a half adder. Returns the sum of the inputs and carry-out.

HALF COMPARATOR:

Compares two 1bit inputs returns the output.

- Returns G=1 if a is greater.
- Returns E=1 if a and b is equal.
- Returns L=1 if b is greater.

FULL COMPARATOR:

Compares two 1bit inputs. Compares them using the previous process input

- Returns G=1 if a is greater.
- Returns E=1 if a and b is equal.
- Returns L=1 if b is greater.

18-BIT COMPARATOR:

Compares two 18-bit inputs using full comparator and half comparator and sets the flags.

REGISTER FILE:

Sets or returns a registers value. Only if write enable is 1, sets wanted registers value to given immediate value.

ALU (ARITHMETIC LOGIC UNIT):

Takes two 18-bit inputs and makes arithmetic logic operations using those inputs. Returns the one that is needed.

- ADD operation is bit-wise and.
- AND operation is bit-wise and.
- OR operation is bit-wise and.
- XOR operation is bit-wise and.

DECODER INPUT:

For our system to run as a finite state machine there needs to be an input. This circuit takes current state of the control unit and based on that returns the next state to the control unit.

PC SELECTOR:

Based on the input called BChoice this circuit returns the return value of the comparison value.

14 BIT ADDER:

Adds two 14-bit inputs bit by bit using full adders and a half adder. Returns the sum of the inputs and carry-out.

6 TO 18 EXTENDER:

Turns 6-bit input to 18-bit output using twos' complement.

6 TO 14 EXTENDER:

Turns 6-bit input to 18-bit output. Sets empty bits to 0 because the input we are converting is address. And address can't be less than zero.

CONTROL UNIT:

Based on its state it generates the input that needed for other components to perform current instruction. This circuit can count as the brain of the system.

PROGRAM COUNTER:

Program counter is a register that contains the address of the instruction being executed at the current time. As each instruction gets fetched, the program counter increases its stored value by 1.

It has pcSelect for jump address or next address. If instruction is jump instruction, it will use the jump address. If not, it will go to the next instruction.

CPU:

In this circuit we basicly connect the whole system. Instruction are inserted in the RAM and directed the output of the RAM to the control unit by looking at program counter.