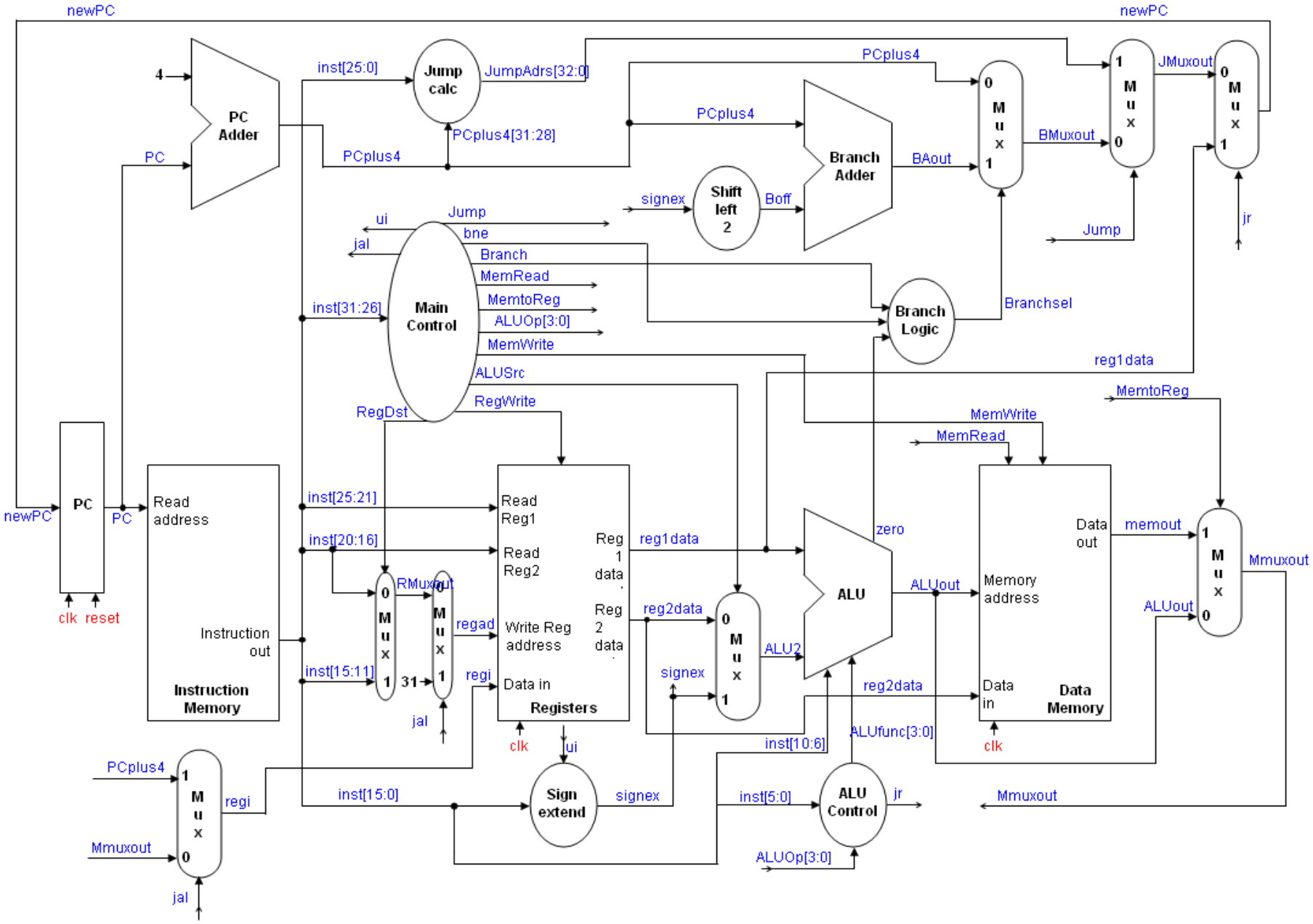
Project: 32-bit Single Cycle Processor Development and Testing

ECE5480/4480 Computer Architecture and Organizations

Overview

- A single cycle processor was designed and implemented previously
  - The Verilog models can be simulated as is in ModelSim without any modifications
  - Only a small portion of the MIPS ISA was implemented
- The block diagram and instruction set implemented are given in more details in the three slides that follows


newPC
newPC
4
PC Adder
PC
inst[25:0]
Jump calc
JumpAdrs[32:0]
PCplus4[31:28]
PCplus4
PCplus4
PCplus4
Branch Adder
BAout
signex
Shift left 2
Boff
0 M u x 1
BMuxout
1 M u x 0
JMuxout
0 M u x 1
Jump
jr
ui
jal
Jump
bne
Branch
MemRead
MemtoReg
ALUOp[3:0]
MemWrite
ALUSrc
RegWrite
RegDst
inst[31:26]
Main Control
Branch Logic
Branchsel
reg1data
MemtoReg
MemWrite
MemRead
PC
newPC
PC
clk reset
Read address
Instruction out
Instruction Memory
inst[25:21]
inst[20:16]
inst[15:11]
0 M u x 1
RMuxout
0 M u x 1
31
regad
regi
jal
Read Reg1
Read Reg2
Write Reg address
Data in
Registers
Reg 1 data
Reg 2 data
reg1data
reg2data
clk
ui
0 M u x 1
ALU2
signex
ALU
zero
ALUout
Memory address
Data in
Data Memory
Data out
memout
clk
reg2data
ALUfunc[3:0]
inst[10:6]
1 M u x 0
Mmuxout
ALUout
PCplus4
1 M u x 0
regi
Mmuxout
jal
inst[15:0]
Sign extend
signex
inst[5:0]
ALU Control
jr
ALUOp[3:0]
Mmuxout

List of implemented instructions

R-format	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	syntax
	op						rs					rt					rd					shamt					funct						
addu	0	0	0	0	0	0	operand addr					operand addr					destination					shamt					1	0	0	0	0	1	addu $rd, $rs, $rt
subu	0	0	0	0	0	0	operand addr					operand addr					destination					shamt					1	0	0	0	1	1	subu $rd, $rs, $rt
and	0	0	0	0	0	0	operand addr					operand addr					destination					shamt					1	0	0	1	0	0	and $rd, $rs, $rt
or	0	0	0	0	0	0	operand addr					operand addr					destination					shamt					1	0	0	1	0	1	or $rd, $rs, $rt
xor	0	0	0	0	0	0	operand addr					operand addr					destination					shamt					1	0	0	1	1	0	xor $rd, $rs, $rt
sll	0	0	0	0	0	0						operand addr					destination					shamt					0	0	0	0	0	0	sll $rd, $rt, shamt
sra	0	0	0	0	0	0						operand addr					destination					shamt					0	0	0	0	1	1	sra $rd, $rt, shamt
srl	0	0	0	0	0	0						operand addr					destination					shamt					0	0	0	0	1	0	srl $rd, $rt, shamt
slt	0	0	0	0	0	0	operand addr					operand addr					destination					shamt					1	0	1	0	1	0	slt $rd, $rs, $rt
sltu	0	0	0	0	0	0	operand addr					operand addr					destination					shamt					1	0	1	0	1	1	sltu $rd, $rs, $rt
jr	0	0	0	0	0	0	$ra (11111)															shamt					0	0	1	0	0	0	jr

I-format	op						rs	rt	address or immediate value	
lui	0	0	1	1	1	1		destination	16 bits to be placed in upper half of reg	lui $rt, imm
addui	0	0	1	0	0	1	operand addr	destination	immediate value	andi $rt, $rs, imm
andi	0	0	1	1	0	0	operand addr	destination	immediate value	addui $rt, $rs, imm
ori	0	0	1	1	0	1	operand addr	destination	immediate value	ori $rt, $rs, imm
xori	0	0	1	1	1	0	operand addr	destination	immediate value	xori $rt, $rs, imm
slti	0	0	1	0	1	0	operand addr	destination	immediate value	slti $rt, $rs, imm
sltiu	0	0	1	0	1	1	operand addr	destination	immediate value	sltiu $rt, $rs, imm
beq	0	0	0	1	0	0	operand addr	operand addr	offset to base address/4	beq $rs, $rt, name
bne	0	0	0	1	0	1	operand addr	operand addr	offset to base address/4	bne $rs, $rt, name
lw	1	0	0	0	1	1	base address	destination	immediate value	lw $rt, imm($rs)
sw	1	0	1	0	1	1	base address	source	immediate value	sw $rt, imm($rs)

J-format	op						target address or offset amount	
j	0	0	0	0	1	0	26 bits to be manipulated to determine new address	j name
jal	0	0	0	0	1	1	26 bits to be manipulated to determine new address	jal name

List of the control signals for implemented instructions

R-format	ALUOp	ALU Control	ALU function	ALUSrc	RegDst	Reg Write	Mem Write	Mem toReg	Mem Read	Branch	bne	Jump	jr	jal	ui
addu	0010	0010	add	0	**1**	**1**	0	0	0	0	0	0	0	0	0
subu	0010	0110	sub	0	**1**	**1**	0	0	0	0	0	0	0	0	0
and	0010	0000	AND	0	**1**	**1**	0	0	0	0	0	0	0	0	0
or	0010	0001	OR	0	**1**	**1**	0	0	0	0	0	0	0	0	0
xor	0010	**1001**	XOR	0	**1**	**1**	0	0	0	0	0	0	0	0	0
sll	0010	**1010**	shift left log	0	**1**	**1**	0	0	0	0	0	0	0	0	0
sra	0010	**1011**	shift right arth	0	**1**	**1**	0	0	0	0	0	0	0	0	0
srl	0010	**1100**	shift right log	0	**1**	**1**	0	0	0	0	0	0	0	0	0
slt	0010	0111	sub 2's comp	0	**1**	**1**	0	0	0	0	0	0	0	0	0
sltu	0010	**1110**	sub & set	0	**1**	**1**	0	0	0	0	0	0	0	0	0
jr	0010	xxxx	none	x	x	0	0	0	0	0	0	0	**1**	0	0

I-format															
lui	**0011**	**1101**	shift left 16	**1**	0	**1**	0	0	0	0	0	0	0	0	0
addui	0000	0010	add	**1**	0	**1**	0	0	0	0	0	0	0	0	**1**
andi	**0101**	0000	AND	**1**	0	**1**	0	0	0	0	0	0	0	0	**1**
ori	**0110**	0001	OR	**1**	0	**1**	0	0	0	0	0	0	0	0	**1**
xori	**0111**	**1001**	XOR	**1**	0	**1**	0	0	0	0	0	0	0	0	**1**
slti	**0100**	0111	sub 2's comp	**1**	0	**1**	0	0	0	0	0	0	0	0	0
sltiu	**1001**	**1110**	sub & set	**1**	0	**1**	0	0	0	0	0	0	0	0	**1**
beq	0001	0110	sub	0	x	0	0	0	0	**1**	0	0	0	0	0
bne	0001	0110	sub	0	x	0	0	0	0	0	**1**	0	0	0	0
lw	0000	0010	add	**1**	0	**1**	0	**1**	**1**	0	0	0	0	0	0
sw	0000	0010	add	**1**	x	0	**1**	0	0	0	0	0	0	0	0

J-format															
j	xxx	xxxx	none	x	x	0	0	0	0	0	0	**1**	0	0	0
jal	xxx	xxxx	none	x	x	**1**	0	0	0	0	0	**1**	0	**1**	**1**

Development and testing tasks

- Learn and familiarize with the design
  - Convert the design into one file per module, add comments
- Model improvements
  - All the modules of the design should be fully synthesizable
  - Other needed improvements
- Write a test bench to test one instruction of each type from the implemented R, I, and J instructions
  - Demonstrate the tested instructions are implemented correctly
- Implement one instruction from MIPS ISA that is not yet implemented in the design

Verilog file: upc.v, embedded in the slides

Testbench and test examples: next a few slides

Note: you can use this example as one of your test, but not recommended

Rough estimation of working hours needed:

1) Multiple files: 2 hour
2) Familiar with the design: 3 hours
3) Verilog improvement and move non-synthesizable structure into testbench: 3 hours
4) Implementing the R, I and J instruction testbench: 6 hours
5) Simulations and run implemented testbench: 9 hours
6) Implement an instruction of your choice: 5 hours

Total: about 28 working hours, adjust this estimation based on your own familiarities with Verilog and working habits

Testing the instruction addu
addu: Add unsigned, R type instruction

It simply add $r1 to $r2 and place the results in $r3, which will be a code of:

000000_00001_00010_00011_00000_100001

set $r1 = 01010101….., and $r2 = 10101010…..

At the first positive edge of the clock *reg1add* = 1, *reg2add* = 2, and *RegWrite* = 1, *writeadd* = 3, and the ALU must have performed the addition because *regin* = 11111111….. Further, on the next positive edge of the clock you can see that the result is written to register[3], as intended.
Run the second clock just to verify the results.

addu testing simulation waveform

Where both the addresses for $rs, $st and the $sd are shown
Also shown is the result data that was transferred back to the register