Multicycle MIPS Processor

Computer Architecture

6/16/16

Contents

Controller:

A controller is a composite of Main Decoder and ALU Decoder, as shown in figure 1. To implement the controller, a structure entity that has the highlighted input and output signals shown in table 1. The controller then maps its input and output signals to the main decoder and ALU decoder.

Code Structure:

Entity Input / Output Signals:

INPUT SIGNALS	OUTPUT SIGNALS	INTERNAL SIGNALS
OPCODE (5 DOWNTO 0)	MEMTOREG	BRANCH
FUNCT (5 DOWNTO 0)	REGDST	INTSIGNAL = BRANCH AND ZERO
ZERO	IORD	
	PCSRC	
	ALUSRCA	
	ALUSRCB (1 DOWNTO 0)	
	IRWRITE	
	MEMWRITE	
	PCENABLE	
	REGWRITE	

TABLE 2: MULTI CYCLE CONTROLLER INPUT/ OUTPUT SIGNALS

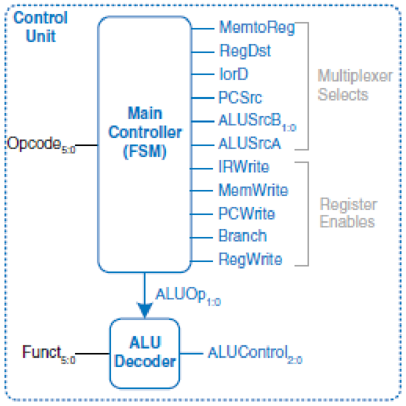


FIGURE 1: MULTI CYCLE CONTROLLER

Controller Architecture Structure

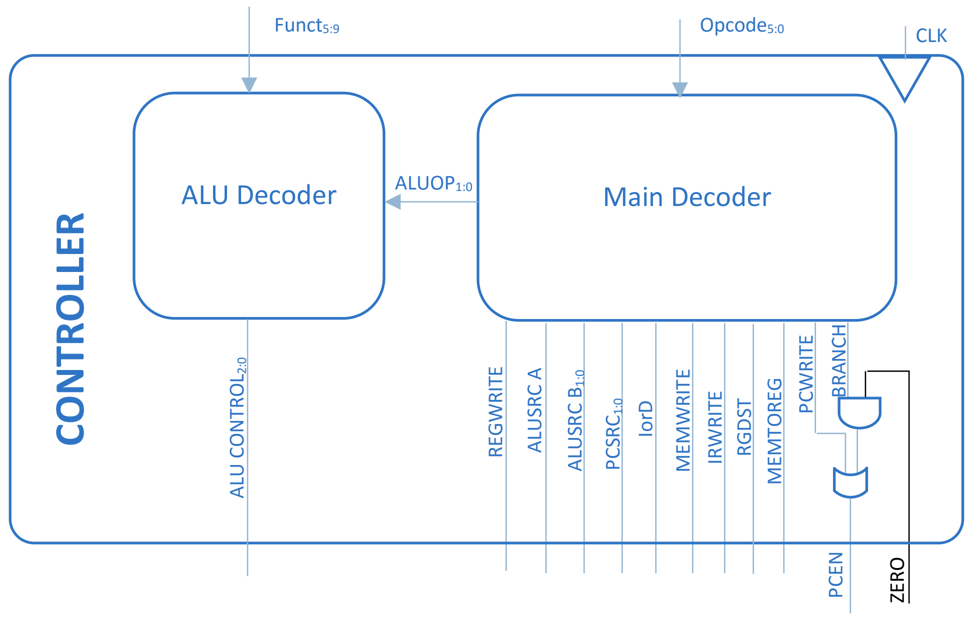
The controller defines a component for its constituents' input and output signals; the main decoder and ALU decoder.

```
component MainDec
    port(
    --Clock and Reset Signals
    clk, reset: in STD_logic;
    --Input OPCODE
    op: in STD_Logic_vector (5 downto 0);
    --Register Enable Signals
    Branch, MemWrite, IRWrite, RegWrite, PCWrite: out STD_logic;
    --MUX Select Signals
    RegDst, MemtoReg, IorD,ALUSrcA: out STD_Logic;
    ALUSrcB: out STD_logic_vector (1 downto 0);
    PCSrc: out STD_logic_vector (1 downto 0);
    --ALUDEC Output
    ALUOp: out STD_logic_vector (1 downto 0)
    );
end component;
```

Then the controller maps the components to the suitable decoder structure file.

```
md: MainDec port map (clk, reset,op,
                      Branch, MemWrite, IRWrite, RegWrite, PCWrite,
                      RegDst, MemtoReg, IorD, ALUSrcA, ALUSrcB, PCSrc,
                      ALUOp);
```

Controller Diagram:



Main Decoder:

In a MIPS Multicycle Processor, a Main Decoder is a finite state machine; that is, its current state depends on its previous state. The following diagram represents the given MIPS State Diagram, and table 2 represents State table. In the table, red bits represents the changed states, while black bits represents don't care states.

FSM Diagram:

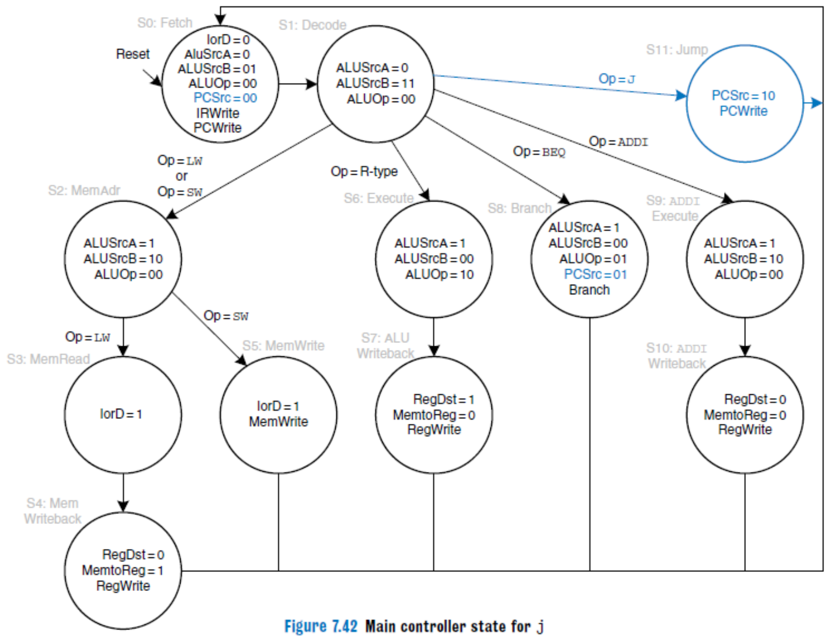


Figure 7.42 Main controller state for j

FSM Table:

OPCODE	INSTRUCTION	PCWRITE	MEMWRITE	IRWRITE	REGWRITE	ALUSRCA	BRANCH	IORD	MEMTOREG	REGDST	ALUSRCB[1:0]		PCSRC		ALUOP		FSM WORD Control.
		0	13	12	11	7	14	8	9	10	6	5	4	3	2	1	
	FETCH	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0x5010
	DECODE	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0x0030
	LOAD WORD EXECUTE(MEM ADR)	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0x0420
100011	LOAD WORD MEMORY (MEMRD)	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0x0100
	LOAD WORD WRITE BACK (MEMWB)	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0x0880
101011	STORE WORD MEMORY (MEMWR)	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0x2100
000000	R-TYPE EXECUTE	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0x0402
	R-TYPE WRITE BACK	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0x0840
000100	BEQ EXECUTE	0	0	0	0	1	1	0	0	0	0	0	0	1	0	1	0x0605
001000	ADDI EXECUTE	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0x0420
	ADDI WRITE BACK	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0x0800
000010	J EXECUTE	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0x4008

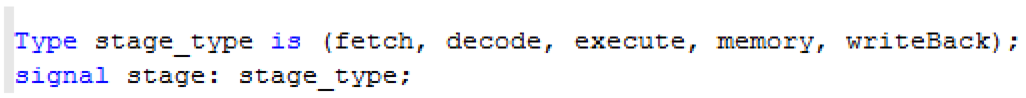
Code Structure:

Input / Output Signals

INPUT SIGNALS	OUTPUT SIGNALS	INTERNAL SIGNALS
CLK	MEMTOREG	STAGE_TYPE
RESET	REGDST	CONTROLSIGNALS
OP (5 DOWNTO 0)	IORD	
	PCSRC (1 DOWNTO 0)	
	ALUSRCA	
	ALUSRCB (1 DOWNTO 0)	
	IRWRITE	
	MEMWRITE	
	PCWRITE	
	REGWRITE	
	BRANCH	
	ALUOP (1 DOWNTO 0)	

Main Decoder Architecture Structure

1. Defined a type for each stage type for the multicycle FSM (Fetch, Decode, Execute, Memory, Write Back)



```
Type stage_type is (fetch, decode, execute, memory, writeBack);
signal stage: stage_type;
```

2. Using process statement on reset and clock, FSM implementation is modeled. On Reset, when a new instruction is read, the stage is set to FETCH. On the rising edge of the clock, the processor defines which stage it is currently holding using the CASE Statement, and then outputs the corresponding control signals to the opcode provided.

ALU Decoder

ALU Decoder structure doesn't change from that in the Single Cycle, and it follows the same Truth Table, which is shown below:

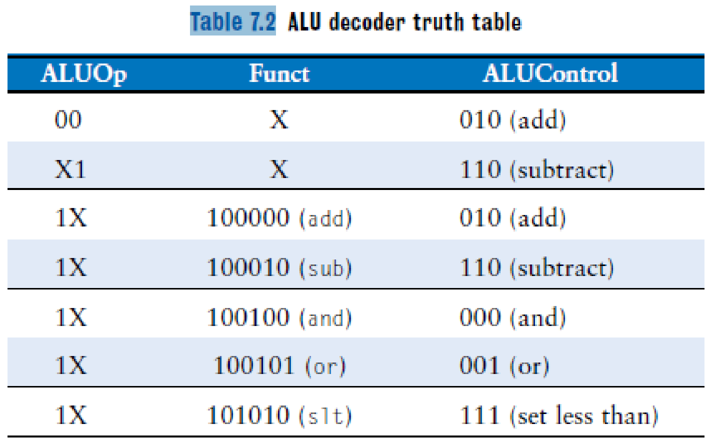


Table 7.2 ALU decoder truth table

ALUOp	Funct	ALUControl
00	X	010 (add)
X1	X	110 (subtract)
1X	100000 (add)	010 (add)
1X	100010 (sub)	110 (subtract)
1X	100100 (and)	000 (and)
1X	100101 (or)	001 (or)
1X	101010 (slt)	111 (set less than)

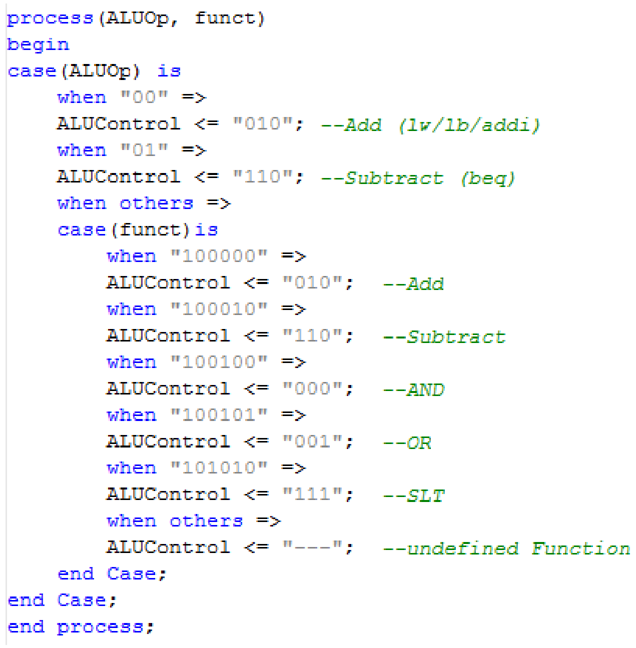
Code Structure

Input/ Output Signals

INPUT SIGNALS	OUTPUT SIGNALS
FUNCT	ALUCONTROL
ALUOP	

ALU Decoder Architecture Structure

Using a process statement on (ALUOP) input from the Main Decoder, and (FUNCT) from instruction address, the ALU Decoder uses a CASE Statement to navigate the correct ALUCONTROL Signal from the given bits.



```
process(ALUOp, funct)
begin
case(ALUOp) is
    when "00" =>
    ALUControl <= "010"; --Add (lw/lb/addi)
    when "01" =>
    ALUControl <= "110"; --Subtract (beq)
    when others =>
    case(funct)is
        when "100000" =>
        ALUControl <= "010";  --Add
        when "100010" =>
        ALUControl <= "110";  --Subtract
        when "100100" =>
        ALUControl <= "000";  --AND
        when "100101" =>
        ALUControl <= "001";  --OR
        when "101010" =>
        ALUControl <= "111";  --SLT
        when others =>
        ALUControl <= "---";  --undefined Function
    end Case;
end Case;
end process;
```

Data Path

Code Structure

Entity Input / Output Signal

INPUT SIGNALS	OUTPUT SIGNALS	INTERNAL SIGNALS
Clk	Zero	Instr
Reset	$Op_{5:0}$	Data
PCEn	$Func_{5:0}$	ImmExt
IorD	$Adr_{31:0}$	SrcA, SrcB, ALUResult
IRWrite	$WD_{31:0}$	ALUOut
RegDst		writeReg
MemtoReg		regInput
RegWrite		RD1, RD2
ALUSrcA		rdA, rdB
$ALUSrcB_{1:0}$		PCJump
$PCSrc_{1:0}$		PCNext
$ALUControl_{2:0}$		PC
$RD_{31:0}$		jumpShift

Architecture Structure

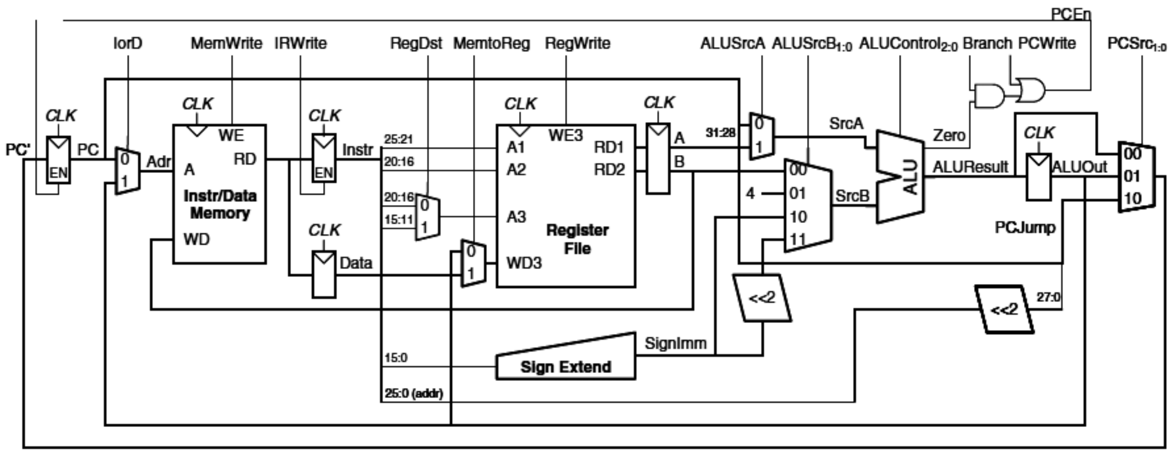


Figure 7.41 Multicycle MIPS datapath enhanced to support the j instruction

The Data path entity constitutes of the following components:

- Instruction Memory
- Data Memory
- Sign Extender
- Register File
- ALU
- MUX2, MUX4

The architecture is divided into 3 logics:

PC Increment and Select Logic

1. The PC Latch takes clk, reset, EN, PCNext and outputs it to PC Signal.

2. The PC Signal is then taken as an input along with ALUOut to to the PCMux to determine the output Address to the Memory

3. The signExtender takes an input Memory Output from (bits: 15 to 0) and extends the sign to ImmExt.

4. ImmExt is taken as input to shft32, which is a component that shifts a 32 bit address by 2 to be an input for SrcB, the output signal is ImmExtSl2

5. ImmExt is also taken as input to shft26, which shifts the Instr (bits: 25 to 0) to prepare it for the jump instruction, the output signal is jumpshift

6. PCJump is the final signal for the Jump instruction, it takes the shifted amount from above and value in PC(bits: 31 to 28) to give the final address of the jump instruction.

Memory Select Logic

1. InstrLatch transfers the data from the RD of Memory to Instr Signal on the Clk and Enable signals.

2. Datalatch transfers the data from the RD of Memory to the Data Signal to be transferred to the Register File.

3. Op and Func are assigned their respective bits from the instruction.

Register File Logic

1. The RegFile Component takes its input from the respective Instr bits (A1 bits 25: 21) and (A2 bits 20:16), WriteReg (the output signal from the mux, which indicates the destination register), Reg Input (the actual data to be written to the destination register), and finally the two signals RD1 and RD2 hold the final outputs of the register.

2. The WriteRegMux Component, chooses between the destination of R-type instruction (Instr bits: 20 to 16), and lw Data (Instr bits: 15 to 11) using the RegDst Selection bit.

3. The regInputMux Component, chooses between the actual data source, from ALU or from Memory, using the MemtoReg Selector bit.

4. SrcALatch transfers RD1 to rdA on clock cycle , and srcBLatch likewise.

ALU Logic

1. srcAMux2 chooses between the PC and input from clocked register (rdA) to provide source for ALU A.

2. srcBMux2 chooses between clocked register(rdB), the number 4 to add to the PC, ImmExt to calculate immediate value of Addi and shifted Extended Immediate to provide it as a source for ALU B.

3. ALUComp, computes the sources of the ALU and provides the result at ALUResult.

4. The ALULatch, provides a clocked ALUResult that outputs to ALUOut.

5 the ALUResult is then taken as input into the ALUOutMux4, along with ALUOut, and shifted PC Jump.

Top Frame

The TopFrame encapsulates the whole design, it consist of MIPS (Controller and Datapath) and the Memory and it associates them together through signals and in and out ports.

The Top Frame defines two Components for its constituents: the Mips, takes Clk, reset, RD, MemWrite, as inputs from the Memory and provides Adr and WD as outputs to Memory. The Memory in return, takes Adr and WD as input Data and WE and Clk as input signals, and writes its data to the RD.

Input Signal	**Output Signal**	**Internal Signal**
Clk	MemWrite (Buffer)	RD
Reset	Adr (Buffer)	
	WD (Buffer)	

Memory

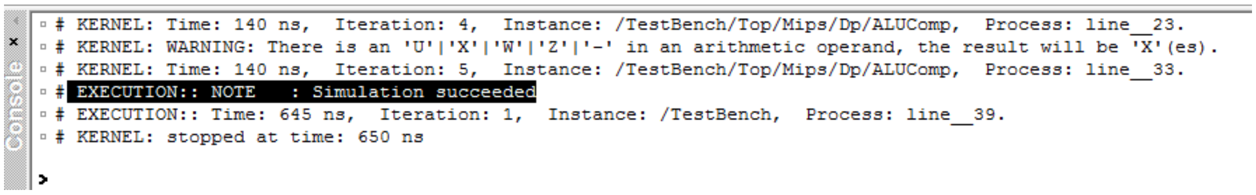
The MIPS Multcycle Memory has the Instruction and Data logics combined. The Multicycle Memory reads the Object Code from File, and outputs the Object Code into an Instruction/Data Memory (MemD) of 4 MB Capacity (32x128)

The Memory is in Write Mode when WE and Clk Events are enabled, and stores inside the Memory the Data given from the Register File.

Test Bench

Putting all the components together and testing the following Components, results in a successful simulation of the given Memfile Object Code in Book Page: 428.

The Following is the Successful Report Message when the integer 7 is written in Address 84.

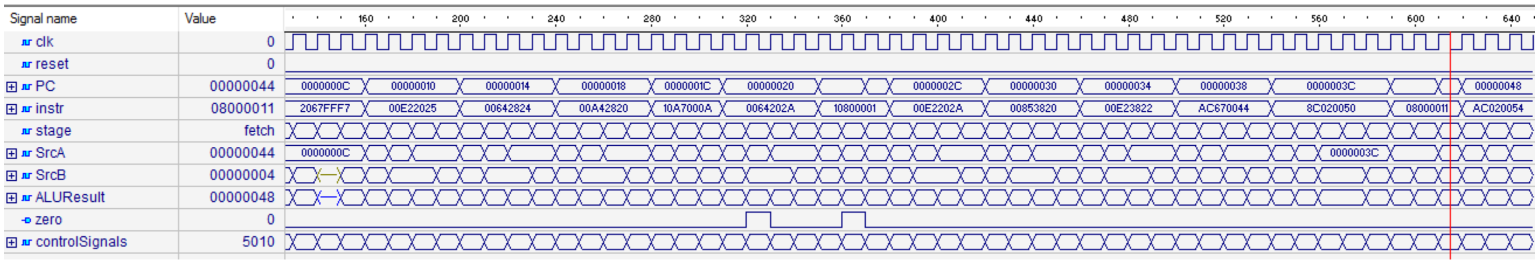


```
# KERNEL: Time: 140 ns,  Iteration: 4,  Instance: /TestBench/Top/Mips/Dp/ALUComp,  Process: line__23.
# KERNEL: WARNING: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
# KERNEL: Time: 140 ns,  Iteration: 5,  Instance: /TestBench/Top/Mips/Dp/ALUComp,  Process: line__33.
# EXECUTION:: NOTE    : Simulation succeeded
# EXECUTION:: Time: 645 ns,  Iteration: 1,  Instance: /TestBench,  Process: line__39.
# KERNEL: stopped at time: 650 ns

>
```

SIMULATION: ATTACHED

The required signals can be found in the attached excel file (Table1.xlsx) and below is the required waveform simulation for the specified signals in the document, which can be found more clearly in jpg format attached to the report file.



Signal name	Value
clk	0
reset	0
PC	00000044
instr	08000011
stage	fetch
SrcA	00000044
SrcB	00000004
ALUResult	00000048
zero	0
controlSignals	5010

FIGURE 3: SIMULATION WAVEFORM FOR MEMFILE