IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 2, Issue 2 (Mar. – Apr. 2013), PP 01-04
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

Design, Implementation and Testing of 16 bit RISC Processor

V. R. Gaikwad
(Department of Electronics Engineering, Walchand College of Engineering, Sangli, India)

Abstract : *Nowadays Embedded Systems have became a part of human life. The most important part of an embedded system is the embedded processor. The performance of embedded processor determines the performance of embedded system. An embedded processor is a Reduced Instruction Set Computer (RISC). In this paper the procedure for designing, implementing and testing a 16 bit RISC processor is presented. This processor was implemented in XC3S400 Field Programmable Gate Array (FPGA) and tested on XC3S400 FPGA development board. This processor is useful for demonstrating hazards in pipeline and the techniques used to solve them.*

Keywords - *Arithmetic and logical unit (ALU), Input output block (IOB), Integrated software environment (ISE), Look up table (LUT), Very high speed integrated circuit hardware descriptive language (VHDL)*

I. INTRODUCTION

A RISC processor uses load-store architecture, fix length instructions and pipelining. In load-store architecture, load instruction reads data from memory and writes it to a register, data processing instructions process data available in registers and write result to a register, and store instruction copies data from register to memory. Pipelining is an implementation technique in which multiple instructions are overlapped in execution [1]. Pipelining improves performance by increasing instruction throughput. To demonstrate how instructions are executed in pipelined processor and the problems created by pipelining, a 16 bit RISC processor supporting eight instructions was designed. The terms byte, half word, word and double are used for 8, 16, 32 and 64 bits data respectively. In mnemonic for an instruction the alphabets b, h, w and d are used for byte, half word, word and double respectively.

II. INSTRUCTION SET

The first step in design of a RISC processor is the design of instruction set. An instruction set contains instructions supported by the processor. Load instruction reads data from memory and writes it to a register. If the address of memory location is specified in load instruction, the instruction length will exceed the number of bits used to address memory. An alternative to this is, to write the memory address in a register and specify the register in the instruction. In addition to this, if an offset is specified in the instruction then an operand at a known offset from a memory location can be accessed. So, load instruction capable of copying data from the memory location whose address is sum of base register (Rs) content and offset (Imm) to a destination register (Rt), was selected. Store instruction capable of copying data from register (Rt) to the memory location whose address is sum of base register (Rs) and offset (Imm), was selected. Arithmetic and logical instructions process data present in registers. Both arithmetic and logical operations produce a result after processing two operands. In two operands arithmetic and logical instructions one of the two source registers is the destination register. If the destination register is a fixed register then, before performing the next arithmetic or logical operation the result must be copied to other register. If the destination register isn't a fixed register then, its content will be lost after performing arithmetic or logical operation. In three operands arithmetic and logical instructions, the result is directly written to a register other than or one of the two source registers. So, arithmetic and logical instructions capable of reading data from two source registers (Rs and Rt) and writing result to destination register (Rd), was selected. The normal sequence of program execution can be changed by using branch instruction. An unconditional branch instruction loads program counter (PC) with the value specified in the instruction. Conditional branch instructions modify PC, if the branch condition is true, by an amount equal to the offset specified in the instruction. Table 1 lists the instructions supported by this processor.

III. INSTRUCTION FORMAT

The second step is design of instruction format. Each instruction is assigned a unique code, known as operation code (Opcode). For eight instructions 3 bits opcode field is required. The opcode field can be reduced to 2 bits by using same opcode for arithmetic and logical instructions and another code, known as function code, to specify the intended arithmetic or logical operations. For five arithmetic and logical instructions 3 bits function field is sufficient. Each register, in register file, is assigned a unique address. To address eight registers, 3 bits address field is required. The systematic placement of these fields in the instruction is referred to as instruction format.

For arithmetic and logical instructions, an opcode field, three 3 bits address fields and a function field is required. The resulting instruction format is known as R format.

opcode (2 bits)	rs (3 bits)	rt (3 bits)	rd (3 bits)	unused (2 bits)	function (3 bits)

For load, store and branch instructions, an opcode field, two 3 bits address fields and an immediate field is required. The resulting instruction format is known as I format.

opcode (2 bits)	rs (3 bits)	rt (3 bits)	imm (8 bits)

Table 1 Instruction Set

Format	Opcode	Instruction	Operation
R	$(00)_2$	add Rd, Rs, Rt	Rd ← Rs + Rt
	$(00)_2$	sub Rd, Rs, Rt	Rd ← Rs – Rt
	$(00)_2$	and Rd, Rs, Rt	Rd ← Rs & Rt
	$(00)_2$	or Rd, Rs, Rt	Rd ← Rs | Rt
	$(00)_2$	slt Rd, Rs, Rt	if (Rs < Rt) then Rd ← 1 else Rd ← 0
	$(11)_2$	beq Rs, Rt, Imm	if (Rs = Rt) then PC ← PC + 1 + Imm else PC ← PC + 1
I	$(01)_2$	lh Rt, Imm(Rs)	Rt ← Mem[Rs + Imm]
	$(10)_2$	sh Rt, Imm(Rs)	Mem[Rs + Imm] ← Rt

Table 2 Control Signals

Control signal	Value	Function
RegRd	0	Selects rt as destination register address
	1	Selects rd as destination register address
RegWr	0	Disables write to register file
	1	Enables write to register file
ALUsrc	0	Selects Rt as 2nd source for ALU
	1	Selects Imm as 2nd source for ALU
ALUop	$(00)_2$	Instruction is load or store
	$(01)_2$	Instruction is branch
	$(10)_2$	Instruction is arithmetic or logical
Branch	0	Instruction isn't a branch
	1	Instruction is branch
MemRd	0	No memory read operation
	1	Memory read operation
MemWr	0	No memory write operation
	1	Memory write operation
MemtoReg	0	Selects output of ALU
	1	Selects output of Memory

Table 3 Truth Table for Control Unit

Opcode	RegRd	RegWr	ALUsrc	ALUop	Branch	MemRd	MemWr	MemtoReg
$(00)_2$	1	1	0	$(10)_2$	0	0	0	0
$(01)_2$	0	1	1	$(00)_2$	0	1	0	1
$(10)_2$	0	0	1	$(00)_2$	0	0	1	0
$(11)_2$	0	0	0	$(01)_2$	1	0	0	0

IV. DATAPATH AND CONTROL

The third step is design of datapath and control unit. Datapath is a systematic arrangement of hardware components and their interconnection for performing an operation. The instruction set is divided into two or more parts with each part containing instructions which require the same logic or hardware for implementation. In this case the instruction set is divided into three classes, arithmetic and logical, memory reference and branch instructions. The datapath for each instruction class is designed and combined to get the final datapath. While combining datapaths a hardware resource is shared by using multiplexer at its input.

Table 4 Truth Table for ALU

ALUop	Function field	ALU function	Operation
$(00)_2$	$(XXX)_2$	$(010)_2$	Addition
$(01)_2$	$(XXX)_2$	$(110)_2$	Subtraction
$(10)_2$	$(000)_2$	$(000)_2$	Bitwise AND
$(10)_2$	$(001)_2$	$(001)_2$	Bitwise OR
$(10)_2$	$(010)_2$	$(010)_2$	Addition
$(10)_2$	$(011)_2$	$(110)_2$	Subtraction
$(10)_2$	$(111)_2$	$(011)_2$	Set on less than

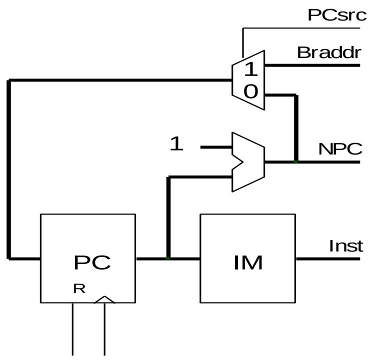


Figure 1 Instruction Fetch (IF) Stage

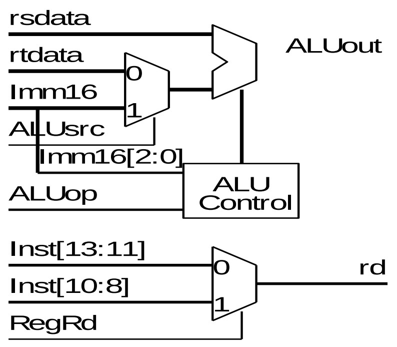


Figure 3 Execution (EX) Stage

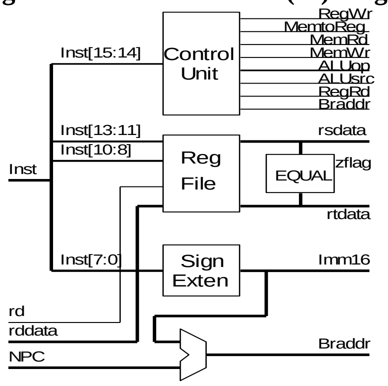


Figure 2 Instruction Decode (ID) Stage

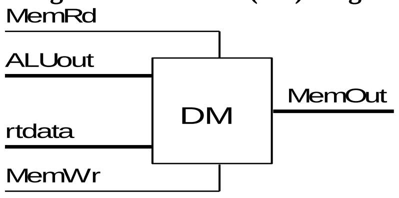


Figure 4 Memory (MEM) Stage

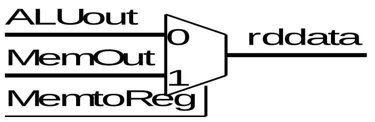


Figure 5 Write Back (WB) Stage

For each instruction to perform the expected operation control signals are required. The control signals required and the function performed by them is listed in table 2. Control signals are generated by the control unit according to the opcode of instruction. The value of control signal for each instruction is listed in table 3. The ALUop control signal and function field in instruction are used to generate ALU function select signal. The truth table for ALU function select signal is shown in table 4. Depending on the operation performed by each section of datapath, during execution of an instruction, the datapath can be divided into five stages. The five stages are, instruction fetch (IF) (Fig. 1), instruction decode (ID) (Fig. 2), execution (EX) (Fig. 3), data memory access (MEM) (Fig. 4) and write back (WB) (Fig. 5) stage. The operation of each stage was verified by writing VHDL code and simulating it using ISE simulator [2]-[3]. The propagation delay of each stage, as observed in Post Route simulation, is listed in table 5.

Table 5 Propagation Delay

Stage	Propagation Delay
IF	13.1 ns
ID	12.5 ns
EX	16.6 ns
MEM	13.6 ns
WB	12.3 ns

Table 6 XC3S400 Device Utilization

Resource	Percentage
Number of Slices used	15 %
Number of Slice flip flops used	7 %
Number of 4 input LUT used	10 %
Number of bounded IOB's used	32 %
Number of GCLK's used	25 %

Table 7 Sample Code

Address	Instruction	Machine code
$(0000)_2$	lh $1, 8($0)	$(4108)_{16}$
$(0001)_2$	sh $1, 9($0)	$(8109)_{16}$
$(0010)_2$	and $0, $0, $0	$(0000)_{16}$
$(0011)_2$	lh $2, 8($0)	$(4208)_{16}$
$(0100)_2$	sh $2, 9($0)	$(8209)_{16}$
$(0101)_2$	add $3, $1, $2	$(0A62)_{16}$
$(0110)_2$	sh $3, 9($0)	$(8309)_{16}$
$(0111)_2$	sub $4, $1, $2	$(0A83)_{16}$
$(1000)_2$	sh $4, 9($0)	$(8409)_{16}$
$(1001)_2$	and $5, $1, $2	$(0AA0)_{16}$
$(1010)_2$	sh $5, 9($0)	$(8509)_{16}$
$(1011)_2$	slt $6, $5, $4	$(2CC7)_{16}$
$(1100)_2$	beq $0, $0, -11	$(C0F5)_{16}$
$(1101)_2$	sh $6, 9($0)	$(8609)_{16}$

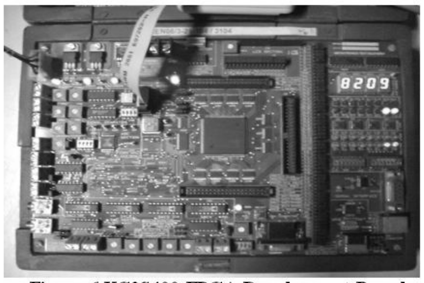
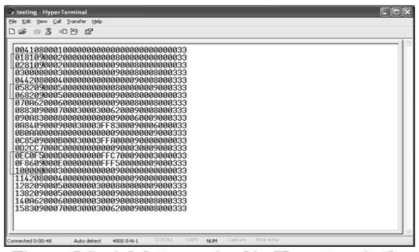


Figure 6 XC3S400 FPGA Development Board


testing - HyperTerminal
File Edit View Call Transfer Help
Connected 0:00:48 Auto detect 4800 8-N-1 SCROLL CAPS NUM

```
00410800010000000000000000000000000033
01810900020000000000000800000000000033
02810900020000000000000900080000000033
03000000030000000000000900080008000333
04420800040000000000000000090008000333
05820900050000000000000800000009000333
06820900050000000000000900080000000033
070A6200060000000000000900080008000333
08830900070003000300062000900080000333
090A8300080000000000000900060009000333
0A840900090003000003FF8300090006000033
0B0AA0000A0000000000000900000009000333
0C850900080003000003FFA000090000000033
0D2CC7000C0000000000000900030009000333
0EC0F5000D000000000FFC7000900030000033
0F8609000E000000000FFF5000000009000333
10000000030000000000000900000000000033
11420800040000000000000000900000000033
12820900050000000030008000000009000333
13820900050000000030009000800000000033
140A6200060000000030009000800080000333
15830900070003000300062000900080000333
```

Figure 7 Serial data received in Hyperterminal

V. PIPELINING

Pipelining is an implementation technique in which multiple instructions are overlapped in execution. The datapath is pipelined by inserting a register, known as pipeline register, between two stages. The stage enclosed between two pipeline registers is known as pipe stage. The PC and pipeline registers are driven by the same clock and reset signal. For pipelined datapath to work properly, clock with period greater than or equal to maximum of propagation delays of pipe stages, is required. For this processor, according to table 5, the minimum clock period is 20 ns. Problems associated with pipelined datapath are referred to as pipeline hazards. Data dependence between instructions results in data hazards. Data hazards are solved by compiler by inserting no operation (NOP) instruction between the instruction producing the result and the instruction using it. In hardware, techniques like forwarding and stalling are used. In data forwarding, the data forwarding logic creates a path from the location in the pipeline where the required data is available to the location where it is required. In stalling, the hazard detection logic detects data hazard and stalls the pipeline. In case of branch instructions, the delay in starting execution of instruction at branch target address, when branch condition is true, results in control hazards. This delay is reduced by modifying the datapath. VHDL codes for pipelined datapath, pipelined datapath with forwarding logic, pipelined datapath with forwarding and stalling logic, and pipelined datapath modified to handle control hazard were written and tested.

VI. TESTING

The processor was implemented in XC3S400 FPGA. The device utilization is presented in table 6. The XC3S400 development board, from Mechatronics, has 16 input pins, 16 output pins, 4 seven segment LED displays, RS232 interface, USB interface, VGA interface, ADC and DAC. A memory mapped input port at address 8, memory mapped output port at address 9, 4 digit multiplexed display controller and serial transmitter were added to the design. The 4 digit multiplexed display controller was used to display the current instruction fetched from IM (Fig. 6). The serial transmitter was used to transmit, PC contents, instruction fetched, next PC calculated, Rs contents, Rt contents, Imm16, ALU output, ALU output bypassed, memory output, end of line indicator (33), carriage return and line feed, during each clock cycle with 8-N-1 format and 4800 bps. Before transmitting the 16 bit data, it was divided into four 4 bit data and each 4 bit data was converted from binary to ASCII. A test code (table 7) containing the instructions supported by the processor was written to IM, to test the processor. This code contains both data hazard and control hazard. The data obtained in hyperterminal clearly indicates the activities inside the processor. The data dependency between the first and second instruction and fourth and fifth instruction stalls the pipeline. Stalls and one clock cycle branch delay are indicated by square in Fig. 7.

VII. CONCLUSION

This processor is similar to the 32 bit MIPS processor explained in [1] however, this processor uses word addressing and supports only eight instructions. This processor is useful for demonstrating pipeline hazards and the techniques used to solve them.

REFERENCES

[1] David A. Patterson and John L. Hennessy, Computer organization and design (Morgan Kaufmann, 1998)
[2] Douglas Perry, VHDL Programming (Tata McGraw Hill)
[3] John Wakerly, Digital design (PHI)

BIBLIOGRAPHY

Mr. V. R. Gaikwad born in Maharashtra, in India on December 29, 1977. He graduated from Dr. J. J. Magdum College of Engineering, Jaysingpur and post graduated from Walchand College of Engineering, Sangli. He received B.E and M.Tech. degree in Electronics Engineering from Shivaji University, Kolhapur. He is working as Assistant Professor in the department of electronics engineering at walchand college of engineering, sangli. His fields of interest include circuits and systems, embedded systems and VLSI.