ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, *Vol 8(20), DOI: 10.17485/ijst/2015/v8i20/78320, August 2015* ISSN (Online) : 0974-5645

Design of a 16 Bit RISC Processor

K. Vishnuvardhan Rao*, A. Anita Angeline and V. S. Kanchana Bhaaskaran

School of Electronics Engineering Department, VIT University Chennai, 632014, Tamil Nadu, India;
vishnu.vardhan2013@vit.ac.in, anitaangeline.a@vit.ac.in, kanchana.vs@vit.ac.in

Abstract

Objectives: This paper presents the design of a 16 bit Reduced Instruction Set Computing (RISC) processor using the custom design approach. **Statistical Analysis:** The type of processor employed in a system claims its efficiency. The compressed instruction incorporated in the design reduces the area and power dissipation of the processor. **Findings:** Various functional blocks of the processor such as the Control Unit, Instruction Decoder, Instruction Register unit and Arithmetic and Logical Unit (ALU) are designed using the Cadence® Virtuoso tool and the simulations are carried out using Cadence® ADE_L Tool using 180nm technology library from TSMC. The integration of the various functional bocks is done based on the finite states arrived at, for the execution of each instruction. **Conclusion:** The RISC processor is found to consume 68.9mW of power for the execution of the AND instruction with a delay of 1600ns. It consumes 77.6mW of power dissipation for the execution of the ADD instruction with a delay of 1900ns.

Keywords: Data Driven Dynamic Logic (D3L), Domino Logic, Introduction, Low Power ALU, RISC Processor Custom Design

1. Introduction

Microprocessor is a profound example of the Very Large Scale Integration (VLSI) industry, which takes the input data in the form of 0's and 1's and processes it according to the instructions. It is expected to yield the output according to the specified instruction at the maximally possible speed. A set of instructions (program) or group of programmes (software) are written to the microprocessor, to perform the task and compute the output[1]. A physical set of hardware modules accomplish the said purpose. The primarily used such modules are the Arithmetic Logic Unit (ALU), Control Unit, Registers and Instruction Execution Unit. The design of an efficient hardware architecture involves the capability to operate with maximum performance even while consuming lower power and reduced silicon area.

The emergence of the Reduced Instruction Set Computing (RISC) processor was an evolution in the computing research platform during the recent years[2]. It has a very simple and defined architecture with fewer fixed length instructions, as compared to the Complex Instruction Set Computing (CISC) which, on the other hand had complex architecture with more number of instructions in the instruction set.

The RISC processor has the following special features:

- All the instructions are of fixed length
- All the instructions are executed in single clock cycle
- Microcode is not allowed and explicit instructions are used

The Advanced RISC Machine (ARM) is one of the Central Processing Unit (CPU) families based on the RISC architecture. The ARM designs are majorly of 32 bit and 64 bit RISC processors for typical applications. The custom design methodology of the processor offers many ways to achieve low power and high speed. One such methodology is by designing the functional blocks in different CMOS logic styles in adaptation to the expected type of arithmetic and logical operations, and integrating them together. The focus of this paper is the design of a RISC machine using custom design static CMOS logic style.

This paper is organized as follows. Section 2 elaborates on the architecture of the RISC processor. Section 3 deals with the functional blocks employed in the processor. Section 4 deals with the integration of processor sub blocks

* *Author for correspondence*

and cites the principal using the typical instructions AND and ADD. Section 5 deals with the performance analysis of the processor and discussion of the results. Section 6 concludes the paper.

2. Processor Architecture

The architecture of the 16 bit RISC processor as shown in Figure 1 contains the Arithmetic and Logical Unit (ALU), Instruction Register, Instruction Decoder and Controller unit. It is based on the RISC machine architecture[1].

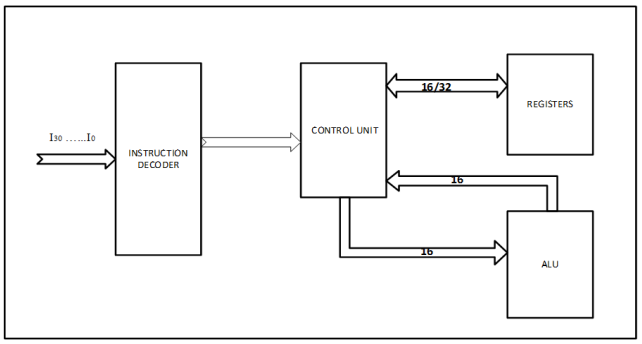

Figure 1. Schematic of the 16 bit RISC processor.

2.1 ALU

The ALU performs the 16 bit arithmetic operations, such as the Addition, Subtraction, Multiplication and Division, and 16 bit logical operations, such as AND, OR and exclusive OR. The ALU consists of the following blocks.

- 16 bit AND operation unit
- 16 bit OR operation unit
- 16 bit XOR operation unit
- 16 bit Carry Look Ahead (CLA) unit
- 16 bit Wallace Tree Multiplier unit
- 16 bit Subtracterunit
- 16 bit Barrel Shifter

2.2 Control Unit

The instruction register brings in the instruction, which is decoded by the instruction decoder. The output of the instruction decoder is fed to the control unit. The control part of the processor asserts the necessary control signals to the ALU and the appropriate registers and ports. It also initiates the program counter to fetch the next instruction.

2.3 Instruction Decoder

The instruction decoder decodes the 32 bit instruction and it enables the respective functional units for execution[3]. There are three types of instructions, namely, Data Transfer type, Data Processing type, and Multiplication instructions. As a typical RISC processor, the design employs the instructions of 32 bit word size. On decoding, the category of instruction is identified. In the instruction, the most significant 4-bits are formed to identify the type of condition codes of the processor. It is depicted in Figure 2 and Figure 3.

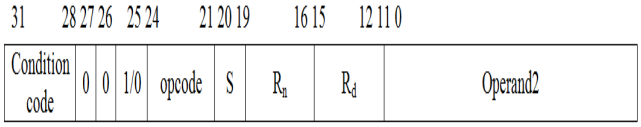

Figure 2. Data processing instruction format.

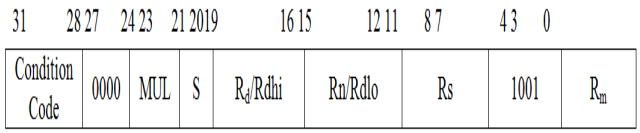

Figure 3. Multiply instruction format.

The following 4 bits (bit positions 27-24) represents the op code and it determines the type of operation or the data processing to be executed on the data, as listed in Table 1.

Table 1. Data processing instructions

OPCODE	Mnemonic	Operation	Execution
0000	AND	Logical bit wise AND	Rd←Rn AND Operand2
0001	OR	Logical bit wise OR	Rd←Rn OR Operand2
0010	EXOR	Logical bit wise XOR	Rd←Rn XOR Operand2
0011	ADD	Arithmetic Addition	Rd←Rn + Operand2
0100	RIGHTROTATE	Logical Right Rotate	Rd←Rn >> N
0101	RIGHTSHIFT	Logical Right Shift	Rd←Rn >> N
0110	LEFTSHIFT	Logical Left Rotate	Rd←Rn << N
0111	LEFTROTATE	Logical Left Shift	Rd←Rn << N
1000	SBC	Substract with Carry	Rd←Rn – Operand2 + Carry -1
1001	ADC	Add with Carry	Rd←Rn + Operand2 + Carry
1010	SUB	Substraction	Rd←Rn – Operand2

The instruction format for the data Processing instructions format is as shown in Figure 2[4]. The operand 2 is shifted or rotated by N bits based on the 25th bit value. If the 25th bit is 1 then the least significant 8 bits are rotated according to bit positions 8 to 11. If the 25th bit is zero the value in the R_m register is shifted by the value specified in R_s. In this the R_d refers to the destination register, R_n refers to the first source register and R_m refers to the second source register. Thus, all the operations are executed in a single clock cycle.

The format of the multiplication instruction is shown in Figure 3. Similar to that of the data processing instructions, the operand 2 is shifted or rotated based on the 25th bit of the instruction. This operand value is then multiplied with the value made available at the source register R_s. Thus, the final result will be either a 32 bit value or 16 bit value, which will be stored in the two registers each of 16 bit width, viz., R_{dlo} and R_{dhi}. For the data transfer instructions, such as the MOVE operations, the value in the source register is copied into the destination register.

2.4 Registers

This processor accommodates 16 number of general purpose registers, which are 16 bit wide. They are identified as B, C, D, E, F, G and H[1]. These registers are primarily used to store the data temporarily during the execution of a program and are accessible to the user through instructions. The last two registers can be made to be employed as the program counter and the stack pointer. Thus, a total of 14 registers are used to help for the internal data operations. The instruction register contains one write and one read port.

2.5 Program Counter

The program counter keeps track of the instructions being executed. It holds the address of the next instruction to be executed. Once the current instruction execution is started, the program counter gets incremented by one. Thus, it is made to always point out to the next instruction to be executed in order.

3. Design of Processor Subblocks

The processor comprises of an ALU unit which executes various the arithmetic and logical operations. In order to realize this, various sub blocks have been designed. The remaining part of this section presents the modules.

3.1 Carry Look Ahead Adder (CLA)

Figure 4 shows the CLA block with input bits represented by A_0 to A_{15} and B_0 to B_{15} respectively, as shown. The SUM bits S_0 to S_{15} with carry bit $C_{15}$are made availble from the CLA, The general form of computation using the equation of a full adder is expressed by

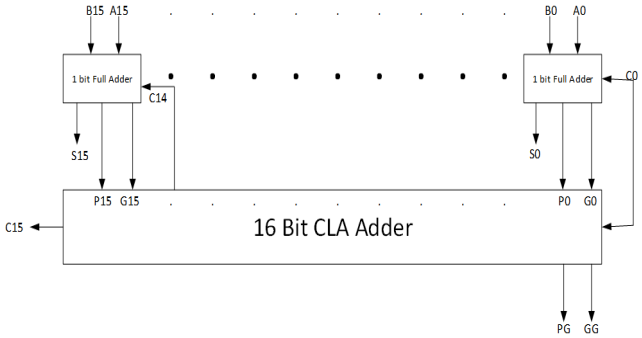

Figure 4. Implementation of 16-bit CLA.

$SUM = A \oplus B \oplus Cin$

In place of the carry bit propagating from one stage to another, with the time length of the process determined by the number of bits of the addition process, the carry bit of each stage is calculated separately by computing the *propagate* and *generate* bits. These bits computations are carried out as follows.

$c(i + 1) = gi + (pi. ci)$

Here, $c(i + 1)$ refers to the carry output of ith stage. The generate and propagate bits of the i[th] stage are computed depending on the input Ai as given by,

$gi = Ai. Bi$

$pi = Ai + Bi$

This makes the CLA perform the addition in a faster manner as the carry is computed quicker rather than waiting for the carry to be propagated through the whole length. Especially, while adding more number of bits or larger words, the CLA computes the sum and carry bits at a higher speed. It may however be noted that the CLA incurs additional silicon area overhead due to the redundant circuitry, which can by itself be an acceptable trade-off.

3.2 Wallace Tree Multiplier

In a Wallace Tree Multiplier, the number of partial products to be added is made available as intermediate results which are then summed up. It comprises of half adder and full adder structures to perform the summing operation. Hence, this makes the computation time to get reduced.

3.3 Barrel Shifter Implementation

The shifting operations are performed by the barrel shifter. The shifter is designed to support arithmetic rotate and logical rotate operations of the processor. The shifting could be done in two ways, namely, 1. Left Shifting (LS) and 2. Right Shifting (RS). The various types of shifting operations possible are tabulated in Table 2. Here, X refers to the 8-bit data on which the shifting operation is to be done and Y is the number, which represents the number of bit positions by which the shifting is to be carried out.

Table 2. Shift and rotate examples for the data X = $a_7a_6a_5a_4a_3a_2a_1a_0$ and Y = 3.

Operation	Output
3-bit shift right logical	000 a_7 a_6 a_5 a_4 a_3
3-bit shift right arithmetic	a_7 a_7 a_7 a_7a_6 a_5 a_4 a_3
3-bit rotate right	a_2 a_1 a_0 a_7 a_6 a_5 a_4 a_3
3-bit shift left logical	a_4 a_3 a_2 a_1 a_0 000
3-bit shift left arithmetic	a_7 a_3 a_2 a_1 a_0 000
3-bit rotate left	a_4 a_3 a_2 a_1 a_0 a_7 a_6 a_5

Figure 5 shows an 8 bit Logical Right Shifter implemented using layers of 2 X 1 multiplexers[5,6]. The selection input bits S2, S1 and S0 identify the number of bits to be shifted. The selection bits are defined by the instruction bits 8 to 11. For example, for a Logical Right implementation if the bit S2 is zero, and the next two bits S1 = S0 = 1, then it indicates that the shifting operation by 3 bit positions is to be performed. This makes the first 3 MSBs of the data as zeros as shown in Table 2.

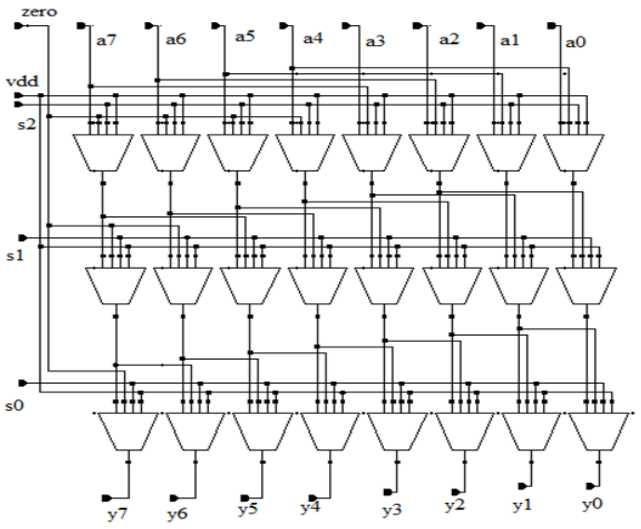

Figure 5. 8 Bit logical right implementation using 2X1 MUX.

On the other hand, note that for the Right Rotate operation for the same selection input combinations, the data will rotate right by 3 bits i.e. from LSB to the higher bits of MSB. This makes the Barrel Shifter to perform (N-1) bits shifts, in a single clock pulse duration[7,8]. The advantage of the barrel shifter is its capability to perform the logical/arithmetic operations in one clock cycle.

4. Integration of the Functional Sub-blocks

The integration of the modules discussed above is presented below in this section. The 32 bit binary instruction is given as the input to the instruction decoder. It decodes the instruction based on the binary bit pattern. The decoded signals are given to the control unit. The control unit initiates the states execution as per the decoded signals. Starting from S0, based on the input, the control unit enables the different units to perform accordingly in each of the required states.

For example, consider the following instruction.

AL213A24B

The binary encoding for above instruction will result in the bit combination shown below, with space between each nibble for easier understanding.

1110 0010 0001 0011 1010 0010 0100 1011

As indicated in Figure 2, the op code 0000 (bits 21 to 24) signifies that the operation to be carried out is the Logical AND and the bits 16 to 19 indicate that the value present in the source register is to be ANDed with the operand 2 as depicted by bits 0 to 11. Further, the 25th bit being a 0, the 8 bit immediate value is rotated by the value 0010 i.e., by 2. The value present in the first source register and the rotated value being the two operands for the AND operation, the Logical AND operation is carried out by the execution of this instruction and the result is stored in the destination register as indicated by the bits 1010.

Figure 6 demonstrates the AND instruction execution flow as a flowchart. Before the execution of the instruction, random values might be present in the registers. Since these registers are to be employed while executing these instructions, both the register data and the instruction in the binary format are simultaneously loaded and the initial state, S0 is made to identify enabling of the register module. The next state S1 writes the data into the destination register. In the state S2, based on the

instruction decoder output, the respective states in the controller execute their conditions. The process continues in a successive manner.

5. Performance Analysis

The basic gates, cells and modules to be used in the processor are designed using the static CMOS logic using Cadence® Virtuoso schematic tool. The 180 nm technology library files from TSMC have been employed. The power dissipation and the delay incurred while operating each of the modules and various arithmetic and logical operations are tabulated in Table 3.

The structure of barrel shifter is so designed to perform the four operations, namely, 1. right shifting, 2. left shifting, 3. right rotate and 4. left rotate. The power dissipation, delay incurred in the execution of each of the operations and the power delay product are represented in the Table 3.

Table 3. Performance of various ALU blocks

	Power(uW)	Delay(pS)	PDP
Control Unit			
2:1MUX	42.56	77.32	3290
4:1 MUX	4.46	242.2	1080
8:1 MUX	8.88	325.7	2892.2
16:1MUX	19.12	373.9	7143.2
32:1MUX	39.41	504.6	19886.2
4:2 Encoder	1.24	113.6	140.86
8:3 Encoder	3.6	181.9	653.7
16:4 Encoder	5.1	168.5	859.35
32:5 Encoder	9.53	227	2163.3
3:8 Decoder	3.79	209.9	795.52
4:16 Decoder	110.05	369	40608
5:32 Decoder	142.5	367	52297
Full Adder	25.2	110	2772
Logical unit			
NAND(2)	5.44	34.17	185.8
NAND(3)	4.77	61.64	294
NAND(4)	7.22	134.4	970.3
NAND(5)	2.55	72.92	185.9
AND (2)	20	93.36	1867
AND (3)	12.26	76.94	943.2
AND (4)	6.12	110.3	675.0
AND (5)	3.7	95.08	351.7
16 bit Logical Right Rotate	112.9	144.9	16359.21
16 bit Bitwise AND	3.37	62.71	211.332
16 bit Bitwise OR	5.84	106.7	623.12
16 bit Bitwise XOR	18.82	119.9	2256.518
16 bit Barrel Right shifting	79.97	276.9	22143.69
16 bit Barrel right rotation	112.9	144.9	16359.21
16 bit Barrel Left shifter	75.5	277.2	20928.6
16 bit Barrel Left rotation	113.2	231.7	26228.44
Arithmetic Unit			
16 bit CLA	62.44	143.7	8972.6
16 bit Wallace tree Multiplier	515.5	1600	824800
16 bit Subtracter	86.06	439.3	37806.16
Instruction Register Unit	10.36mW	90	932400
Instruction Decoder	12.91mW	77	994070

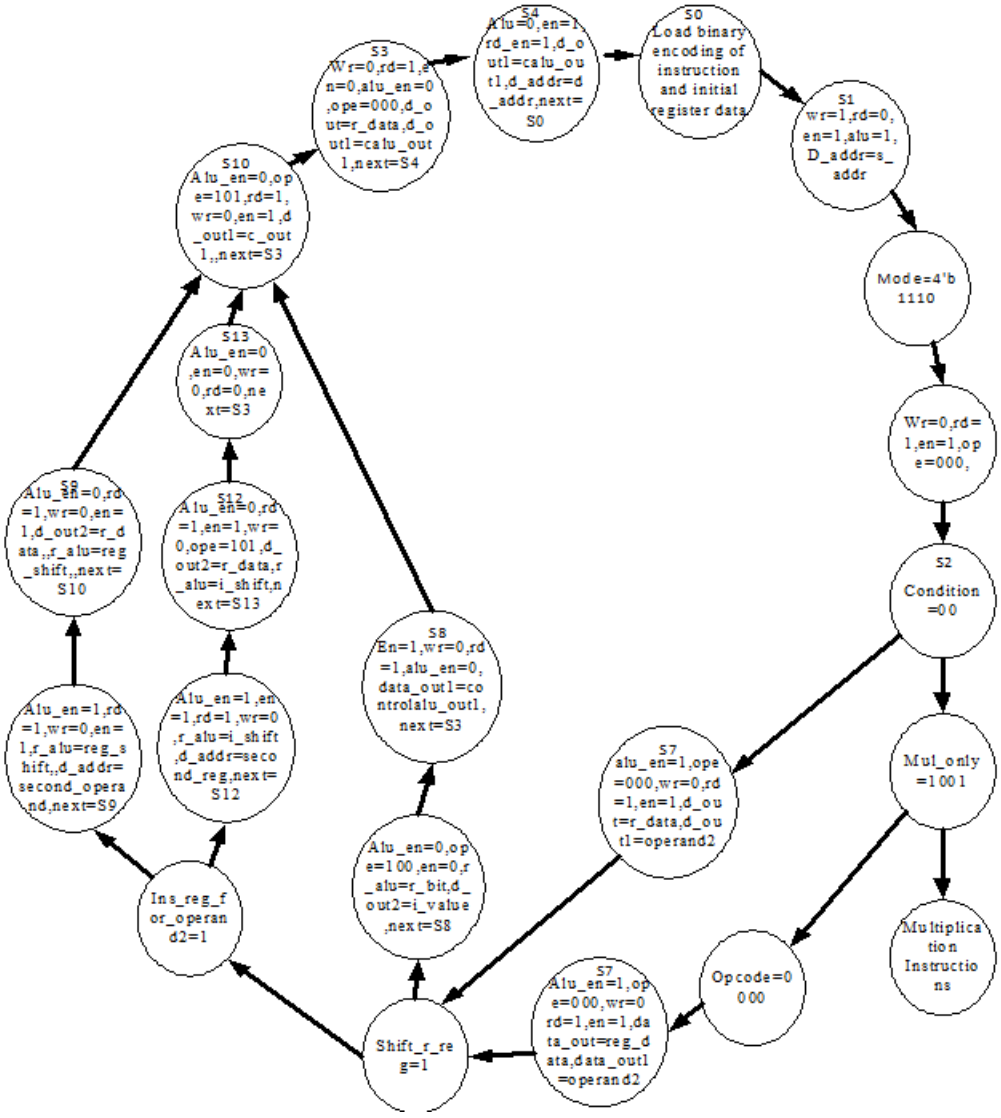

Figure 6. Logical AND instruction execution flow.

In the arithmetic unit of the processor, the CLA is found to consume 62.44 uW and the Wallace tree multiplier consumes 515.5uW of power. As the instruction decoder comprises of D Flip flops and multiplexers, it is found to incur increased power dissipation. To achieve sufficient driving strength of the signals, buffers are inserted at different stages.

Table 4 shows the processor performance specifications while executing the instructions AND and ADD considered for example. The power consumption of indicated include those values incurred by various sub blocks enabled during the execution of the above instructions. To cite a typical case under consideration, for the execution of the ADD instruction, the CLA had been enabled. Hence, the power contributed by the CLA is the primary component of total power during the execution of the ADD instruction.

Table 4. Processor performance on execution of instruction

Instruction	Power (mW)	Delay (nS)
AND instruction	68.9	1.2
ADD instruction	77.6	1.9

6. Conclusion

The design of a 16 bit RISC processor using the static CMOS logic is presented. The processor is designed using various functional blocks which are controlled sequentially by a control unit, according to the states as realized for each of the instructions. The processor handles 32 bit instruction format and execution of various arithmetic and logical operations have been tested exhaustively. For aset of typical cases of logical and arithmetic operations, the AND instruction has been found to consume a power of 68.9 mW incurring a delay of 1.2ns and the ADD instruction consumes a power of 77.6 mW incurring a delay of 1.9ns. The processor has been custom designed and the use of appropriate logic styles for various functional blocks can also be attempted, while striving for low power operation for the processor.

7. References

1. Gaonkar R. Microprocessor architecture, programming and applications with the 8085. PENRAM International Publishing Pvt Ltd; 2010.
2. Hohl W, Hinds C. ARM assembly language fundamentals and techniques. CRC Press; 2014.
3. Saambhavi VB, Kanchana VSB. A 16-bit RISC microprocessor using DCPAL circuits. International Journal of Advanced Engineering Technology (IJAET). 2011 Jan–Mar; 2(1):1–9.
4. Furber S. ARM System-on-chip Architecture. Pearson Education Limited; 2000.
5. Pillmeier MR, Schulte MJ, Walters EG. Design alternatives for barrel shifters. Proceedings of the SPIE; 2002 Dec; USA: Lehigh University Bethlehem, PA 18015. 4791: 436–47.
6. Acken KP, Irwin MJ, Owens RM. Power comparisons for barrel shifters. International Symposium on Low Power Electronics and Design. IEEE Conference Publications; 1996 Aug 12-14; Monterey, CA; p. 209–12.
7. Mandal P, Malani S, Gudepkar Y, Singhi S, Palsodkar P M. VLSI implementation of a barrel shifter. Proceedings of SPIT-IEEE Colloquium and International Conference; Mumbai, India. 2. p. 150–4.
8. Neema V, Gupta P. Design strategy for barrel shifter using mux at 180nm technology node. International Journal of Science and Modern Engineering (IJISME). 2013 July; 1(8):7–11. ISSN: 2319-6386.