Indian Journal of Science and Technology, *Vol 8(19), DOI: 10.17485/ijst/2015/v8i19/76865, August 2015* *ISSN (Print) : 0974-6846*
ISSN (Online) : 0974-5645

Canonic Signed Digit Recoding based RISC Processor Design

Ajintha Elsa Abraham*, N. R. Sangeetha and P. Reena Monica

SENSE Department, VIT University, Chennai Campus, Chennai - 600127,
Tamil Nadu, India; ajintha.elsa2013@vit.ac.in, sangeetha.nr2013@vit.ac.in, reenamonica@vit.ac.in

Abstract

Objective: The objective of this work is to design a Reduced Instruction Set Computing (RISC) processor with Canonic Signed Digit (CSD) recoding. **Methods:** The incorporation of the CSD recording reduces the number of non-zero bits in the constant word length coefficient. The processor has a dedicated processing unit for the manipulation of floating point numbers. It uses CSD recoded number for the execution of the arithmetic operations such as multiplication. This novel technique reduces the switching activity and in turn resulting in reduction in the power consumed by the processor. **Findings:** The simulation was carried out using XILINX 14.3 and CADENCE NCLAUNCH. The RISC processor was synthesized with and without the CSD recoding. Although there is a slight increase in the area overhead, the use of ternary number representation in the processor design brought in a power reduction of 56.23%. **Conclusions:** The CSD recoding was found to be effective in terms of power consumption, making the RISC processor power efficient.

Keywords: CSD Recoding, Floating Point Number, Power Reduction, RISC Processor

1. Introduction

The advancement in the Si technology and the reduction in the cost of integrated circuits have resulted in wide spread use of RISC processors in all fields. The RISC processor as the name suggests executes the instructions in lesser machine cycles in comparison with Complex Instruction Set Computer (CISC) processor. The highly optimized and simple sets of instructions are the added advantages of the RISC processor. The RISC processor usually follows the Harvard architecture, i.e., the instruction and the data stream are separated. Thus the alterations made in the memory, doesn't affect the execution of the instruction. Some of the features of the RISC processor include: uniform format of the instruction which makes the decoding process easier. The RISC processor also has higher throughput per cycle. The RISC processor consumes lesser hardware in terms of number of transistors. This work enhances the advantages of the RISC processor with the incorporation of CSD recoding. The CSD coding is a signed digit coding technique. It finds application in low power, area efficient and high speed signal processing systems[1]. CSD is a ternary number system with digit set {1, 0, ī}, Where ī represents -1. The CSD recoding has two important properties: 1. Number of non-zero digits is minimal; this property reduces the number of addition in the arithmetic operation. The reduction in the number of adders reduces the area and power consumed by the processor as a whole. 2. No two adjacent digits are non-zero; thus the hamming weight of the number is reduced[2, 7].

R. Hashemian proposes a Binary Signed (BS) number system. The CSD is presented as a special case of BS numbering. The BS system is observed to be an efficient number system as it doesn't add extra sign bit to the number. A bit 1 in the i^{th} position contributes 2^i to the number and similarly a -1 in the i^{th} position adds a 2^i to the number[3].

B. Phillips et al. in their work on minimal weight digit conversion deals with a recoding scheme which is intended to reduce the hamming weight of the number. The hamming weight stands for the number of 1's in the number[4].

**Author for correspondence*

Floating point numbers are omnipresent in the computer systems. Therefore its representation and manipulation is an inevitable task. IEEE 754 is the widely accepted standard for the floating point number representation. The single precision IEEE 754 representation occupies 32 bits. The format has a sign bit, 8 bit exponent and 23 bit mantissa. The IEEE 754 representation is officially known as binary 32. The sign bit tells whether the number is a positive or negative number. The exponent is an 8 bit signed integer from -128 to 127[5].

2. Architecture of RISC Processor

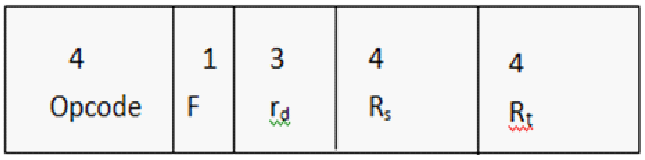
The RISC processor presented in the work has the following blocks: instruction fetch unit, instruction memory, decoder, Arithmetic and Logical Unit (ALU), general purpose registers, data memory, flag, program counter. Figure 1 shows the architecture of the proposed processor.

2.1 Module Description

The various blocks and their functionalities are discussed in detail in the following sections:

2.1.1 Instruction Fetch Unit

The instruction fetch unit procures a 16 bit instruction. The instruction format is shown in Figure 2. Opcode is the 4 bit data which determines the operation to be performed such as addition, multiplication etc. The F bit determines whether the number is floating point or fixed point. F = 0 means that the number is fixed point and otherwise it is floating point operand. So according to the F bit, the operands are handled by either fixed point unit or floating point unit. R_d stands for the destination register, i.e., the location at which the computed results have to be stored. The R_s and R_t stands for the source registers from where the ALU acquires the data to carry out necessary computation.



4 Opcode	1 F	3 r_d	4 R_s	4 R_t

Figure 2. 16 bit instruction.

2.1.2 Instruction Register

The instruction register accumulates the data acquired by the instruction fetch unit, and stores it in a sequential format. The PC points towards that location of the instruction register whose content is to be executed. After the execution of each instruction the PC increments, and points towards the next location of the instruction register and the next instruction gets executed.

2.1.3 Instruction Decoder

The instruction decoder unit decodes the data that is available in the instruction register. This module instructs the ALU to perform the requisite task. It directs the ALU to acquire data for the arithmetic operation and shows the location of the results to be stored. The instruction fetch unit, instruction register and the decoder unit form the control unit which performs the wholesome controlling operation. The FSM that governs the control unit is shown in Figure 3.

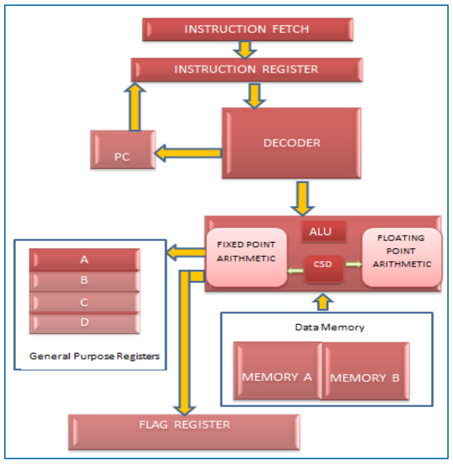


Figure 1. RISC processor architecture.

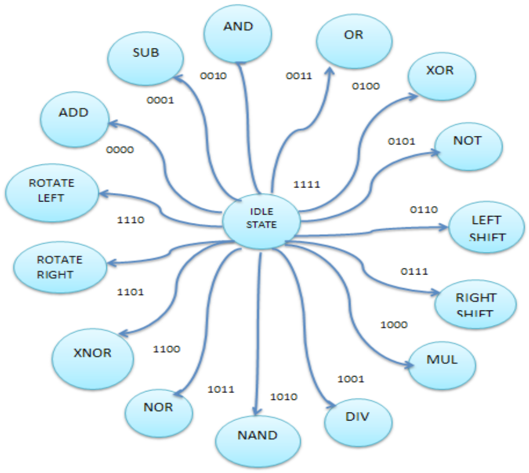


Figure 3. State Diagram of the control unit.

2.1.4 ALU

The ALU performs the arithmetic and logical operations as directed by the decoder unit. The ALU has two units in it: fixed point arithmetic unit and floating point arithmetic unit. The 12th bit of the instruction determines if the data is to be operated by the floating point unit or the fixed point unit.

2.1.4.1 Floating Point Arithmetic Unit

Floating point arithmetic unit is incorporated to manipulate the floating point data that is input to the system. The floating point arithmetic unit takes in two sets of data: the integer part and fraction part. Both the integer and the fractional part are each 32 bit data. The inputs have to be consolidated to form a 32 bit effective operand[4]. The algorithm for the conversion of the input data to effective operand is as follows:

Step 1: The sign bit of the integer input is taken as the sign bit of the effective operand.

Step 2: Binary search algorithm is employed to find out the occurrence of the first '1' in the integer input. The position of the first '1' bit is stored in a register named 'position'.

Step 3: All the bits in the integer input from the position bit to the end is stored in the effective operand.

Step 4: If there are vacant bits left back in the effective operand, it is filled with operands from fractional input.

The FPU unit is designed to be in compliance with the IEEE 754 format. So the effective operand needs to be converted into the format. The IEEE 754 is shown in Figure 4. IEEE754 single precision is encoded in a 32 bit format. The first bit is the sign bit and the next 8 bits forms the exponent, the rest of the 23 bits represents the mantissa. The algorithm for the conversion is as follows:

Step1: Sign bit of the effective operand acts the sign bit for the IEEE format input.

Step2: The 30th bit to the 8th bit of the effective operand acts as the mantissa.

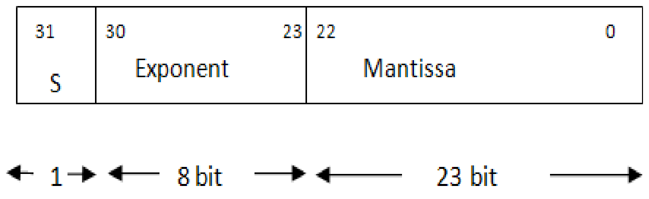


Figure 4. IEEE754 format.

Step3: The exponent is calculated by subtracting the position of the first '1' that occurred in the integer input explained in the previous algorithm. Then a bias of 127 is added to the exponent.

The IEEE 754 format input is used for performing the arithmetic and logical operations. Floating point multiplication block diagram representation[5] is shown in Figure 5.

The sign bits of the two inputs are XOR ed to get the sign bit of the product. The exponent of the two numbers represents the already biased value. Hence it doesn't represent the original exponent value. Thus the exponent needs to be subtracted once otherwise the bias 127 gets added twice.

Let Ea and Eb be the exponents of the two operands. The exponent of the resulting product will be:

$$E = Ea + Eb - 127$$

The mantissa part of the multiplicand is taken as it is and the mantissa of the multiplier is converted into CSD. The CSD recoded data will have lesser number of 1s hence the number of shift and add operations get reduced. This can further reduce the power consumed by the processor. The result gets stored in the destination register as specified in the instruction.

2.1.4.2 Fixed Point Arithmetic Unit

The block manipulates the fixed point data. The data is procured from the location specified in the instruction and the arithmetic and logical operations are done according to the requirement.

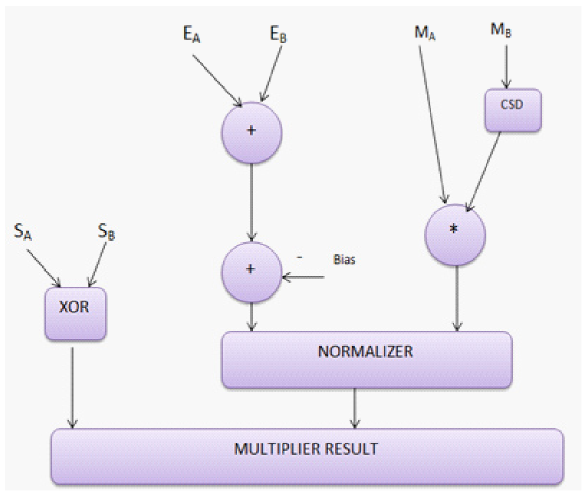


Figure 5. Floating point multiplication block diagram.

2.1.5 *Data Memory and General Purpose Register*

The data memory consists of 16 bit memory locations which are used to store the operands. The general purpose registers consists of four sixteen bit locations which hold the results from the ALU. The flag register consists of two flags, zero-flag and carry-flag. The zero flag is set to one when the output of the ALU is 16 bit zero value and the carry flag is set to one when there is a carry value after the arithmetic value.

3. CSD Recoding

Algorithm for conversion of a 2's complement number into CSD is given below[6, 8]:

Let 'a' be the 2's complement representation of a binary number.

```
A = aw-1 aw-2 ....... a1 a0
Let
    a-1 = 0;
    g-1 = 0;
    aw = aw-1;
For (i=0 to w-1)
    {
    ti = ai ^ai-1 ;
    gi =~(gi-1) * ti;
    ci = (1-2ai+1) gi;
    }
```

$C = C_W \, C_{W-1} \ldots\ldots\ldots\, C_0$

C is the generated CSD number with less number 1's and no adjacent non zero digits.

For example, the binary number 1111 when recoded into CSD it will be 000 ī. Thus the number of non-zero digits gets drastically reduced and therefore the number of additions in the arithmetic operations also gets reduced.

In the proposed processor, the CSD recoding unit has been incorporated into the ALU part. So that, the operand passes through the CSD recoding unit and then it goes for the corresponding operation.

4. Simulation Results

The simulation was done using XILINX 14.3 and CADENCE NCLAUNCH. To estimate the variation in power by the incorporation of the CSD recoding, the simulations were performed with and without the CSD recoding unit.

The arithmetic operations were performed with and without CSD conversion and the results were noted and compared.

In Figure 6., the processor procures a 16 bit instruction, 1000_1100_0000_0001. The decoder unit identifies, the opcode, destination register, and the source registers. The opcode 1000 stands for multiplication. Then the ALU fetches the 16 bit data from the R_s and R_t (source registers) which are at locations 0000 and 00001 respectively. The data bits 0001110011011011 and 0000001001101101 are multiplied and the result is stored at the destination location 1100.

Figure 7 shows the multiplication with the CSD recoding technique. The operands are being converted into the CSD value and then the multiplication is carried out. As mentioned earlier, the CSD conversion reduces the number of nonzero digits in the number; the number of addition operations involved will be reduced. The multiplication of the same operands as in the example above gives the result 1ī10ī0001000ī00ī. So there is a considerable reduction in number of 1s. Thus the switching activity reduction achieved which in turn leads to reduction in power.

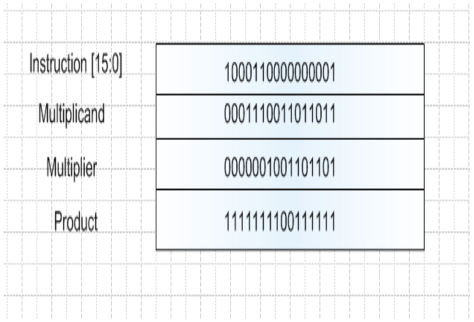


Figure 6. Fixed point Multiplication output without CSD recoding.

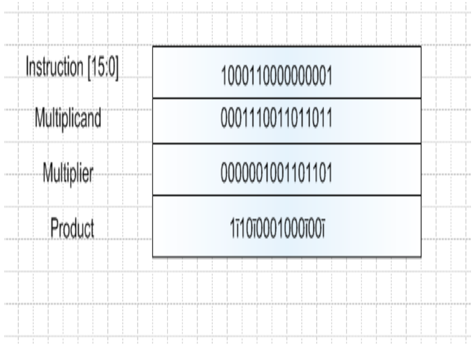


Figure 7. Fixed point multiplication output with CSD recoding.

Figure 8 shows the floating point multiplication. The processor takes in two operands a and b. Each operand a and b has two parts a1, a2 and b1, b2 respectively. a1 and b1 corresponds to the integer part of the input. a2 and b2 are the fractional parts. o1 and o2 are the operands that have been converted into IEEE 754 format.

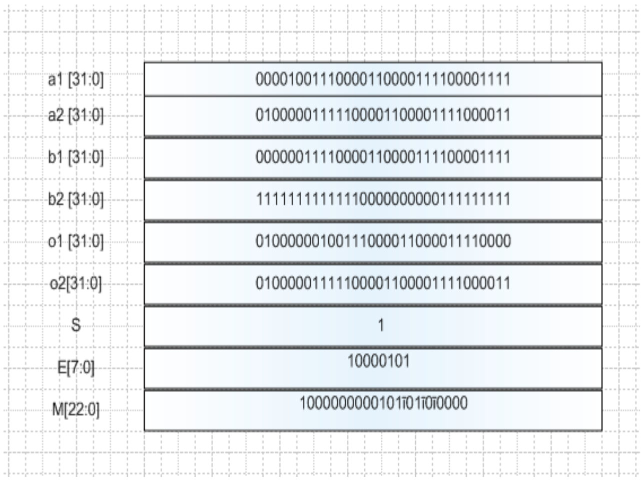
Now the multiplication algorithm is executed using o1and o2. The product is a 32 bit data with the 31st bit (S) representing the sign bit, the 8 bits from 30 to 23 represents the exponent (E) and the final 23 bits represents the mantissa (M).

The power consumption of the ALU block with CSD recoding was found to be lesser than that without CSD recoding. Even though there was a slight increase in the area and the timing, it is very menial in comparison with the amount of the power reduced. Table 1. gives the ALU performance parameters.

4.1 ASIC Implementation of the Processor

The ASIC implementation of the processor was accomplished by the CADENCE –ENCOUNTER tool. The tool first of all does the floor plan and placement for the design wherein it places the cells in the most appropriate position. The floor plan and placement are two critical processes as it determines the overall processor performance. The effective placement of the cells will boost the performance in terms of timing area etc. It also helps in reduction of congestion. Constraints have to be defined to perform the placement in compliance with the timing requirements. The clock tree insertion is done by the tool after the cell placement. The routing is done in two steps global routing and detailed routing. This further optimises the placement. Fial step done by the layout tool is the detailed routing. The design then undergoes Design Rule Check (DRC) and Layout Versus Schematic (LVS) before the generation of the GDS II file. Figure 9 shows the final layout of the 16bit RISC processor obtained using the CADENCE tool.

Table 2. lists the performance parameters of the RISC processor. The processor was found to be operating at 11.2MHz frequency. The power consumption was found to be in the milli-watt range.



```
a1 [31:0]   00001001110000110000111100001111
a2 [31:0]   01000001111100001100001111000011
b1 [31:0]   00000011110000110000111100001111
b2 [31:0]   11111111111110000000000111111111
o1 [31:0]   01000000100111000011000011110000
o2[31:0]    01000001111100001100001111000011
S           1
E[7:0]      10000101
M[22:0]     1000000000101ī01ī0ī0000
```

Figure 8. Floating point multiplication output with CSD recoding.

Table 1. ALU performance parameters

Properties	With CSD recoding	Without CSD recoding
Cell area(um^2)	10416	10380
Timing (pS)	56618	56624
Power (mW)	1.185	3.987

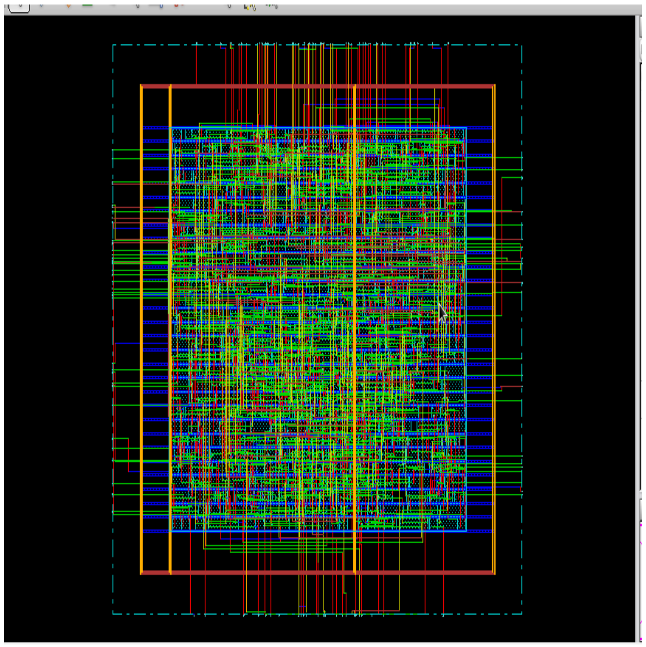


Figure 9. Layout of the 16bit RISC processor.

Table 2. Processor performance parameters

Properties	Values
Frequency of operation	11.2MHz
Total cell area occupied	31456 um^2
Power consumption	10.32048mW

5. Conclusion

The RISC processor with CSD recoding was designed and implemented. The processor can be given both fixed point and floating point operands. This makes the processor to be employed in digital signal processing systems. The CSD recoding of the multiplier bit in the multiplication operation could reduce the power consumption by 2mW. The CSD recoding was found to be effective in terms of power consumption with a very slight overhead in the area occupied, making the RISC processor power efficient.

6. References

1. Anjana R, Gandhi K. VHDL implementation of a MIPS RISC processor. International Journal of Advanced Research in Computer Science and Software Engineering. 2012 Aug; 2(8):83–8.
2. Ruiz GA, Granda M. Efficient Canonic signed digit recoding. Microelectronics Journal. 2011; 42(9):1090–7.
3. Karam LJ, Hasan YM, Falkinberg M. Canonic signed digit FIR filter design. IEEE Conference; Pacific Grove, CA, USA; 2000 Oct 29-Nov 1; vol 2. p. 1653–6.
4. Phillips B, Burgess N. Minimal weight digit set conversions. IEEE Transactions on Computers. 2004 Jun; 53(6):666–77.
5. Rao VN, Swathi V. Normalization on floating point multiplication using verilog HDL. International Journal of VLSI and Embedded Systems. 2013 Aug; 04:586–91.
6. Harini Sharma D, Ramesh AP. Floating point multiplier using Canonical Signed Digit. International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE). 2013 Nov; 2(11):872–4.
7. Parhi KK. VLSI Digital Signal Processing System 6th ed. Microelectronic Research Center. 2007.