Partitioned Branch Condition Resolution Logic

Aamir Farooqui[1], K. Wayne Current[2], Vojin G. Oklobdzija[2]

[1]*Synopsys Inc.*
Synopsys Module Compiler Group
700 Middlefield Road,
Mountain View
CA 94043-4033
(650) 584-5689
(650) 584-1227 FAX
aamirf@synopsys.com
http://aamir.homepage.com

[2]*Advanced Computer System Engineering Laboratory*
Department of Electrical and Computer Engineering
University of California
Davis
CA 95616
(510) 486-8171
(510) 486-0790 FAX
vojin, current @ece.ucdavis.edu
http://www.ece.ucdavis.edu/acsel

Abstract

This paper presents the design of an early condition resolution circuit. The proposed circuit works in parallel with the arithmetic unit, and calculates the Equal to (EQ), Greater-than (GT), Less-than (LT), Overflow (OV), Underflow (UF), and Carry-out (C_{out}), conditions. The proposed logic is reconfigurable, and can calculate all the conditions (mentioned above) for one 64-bit, two 32-bit, four 16-bit, or eight 8-bit signed and unsigned operands. The reconfigurability of the logic is achieved using only four control signals. Two of the control signals (Part0, Part1) are used to partition the logic into 64, 32, 16, or 8-bit independent condition resolvers and the Sign control signal is used to control the signed/unsigned operation. The Add-Sub control signal specifies and controls the Add/Subtract operation for the condition resolution. In order to achieve high speed with reconfigurability and minimum area, new techniques are developed for calculating the conditions before the results are available. The proposed logic is designed for VLIW or Media processors, which require a high degree of reconfigurability, and high speed operation. It can also be used in MIPS family of processors for early branch condition resolution, to avoid branch stalls. Simulations of realizations in a standard digital CMOS fabrication technology show a 30% improvement in speed and a 25% savings in area using the approaches presented here.

1. Introduction

Conditional branches play an important role in controlling the program execution on a digital computer. Implementing conditional branches typically involves two steps: first step is the condition resolution, and the second step is the actual branch operation based on the result of the condition resolution. The execution latency of conditional branches depends on the availability of the condition resolution results. If the conditional branch instruction occurs before the condition is resolved, either the branch instruction must be stalled or executed speculatively, both of which significantly diminish the overall performance of the processor.

In VLIW processors, the conditional branches are avoided using predicated instruction execution; i.e., the condition is embedded within the instruction. If the condition is satisfied, the result is written to output register, otherwise no operation is performed. Using this technique, branch stalls are avoided [1].

However, it either requires very fast circuits to resolve the condition (before the result is available) in a single cycle, or performs the operation in two cycles. Therefore, VLIW processors require novel early branch condition resolution circuits to perform predicated instruction execution

Similarly, in Media processors, in order to perform saturation; one needs to evaluate carry, overflow, and underflow conditions before the result of an operation is available. The carry, overflow, and underflow signals are then used to select the proper output. Moreover, in multi-media operations, equal-to, greater-than, and smaller-than conditions are used to implement the Absolute Sum of Difference, Min, and Max operations. Therefore, Multimedia processors with wider and varying word lengths require re-configurable condition re-solvers that can be reconfigured for variety of data formats.

The design presented here has been optimized for speed and area. Pass-transistor-based multiplexers, the fastest circuit elements in the standard CMOS logic, are

employed in this design. In this design, special techniques are developed for partitioning the condition resolution circuit without producing any significant delay and area penalty. The proposed circuit evaluates the Equal-to (EQ), Greater-than (GT), Less-than (LT), and Carry-out (C_{out}) conditions in parallel with the arithmetic unit.

Previous work on the early detection of branch conditions is described in [2], [3], [4], [5], [6]. The solutions presented in [2] and [3] are not implemented, while the solutions presented in [7] are not practical due to large fan-in requirements. A very high-speed condition code resolution circuit, used in PowerPC, is presented in [6]. This circuit achieves high speed using dynamic logic implementation.

In this paper we present the design of a branch condition resolution circuit using standard cells. The proposed circuit works in parallel with the arithmetic unit, and calculates the Equal to (EQ), Greater-than (GT), Less-than (LT), Overflow (OV), Underflow (UF), and Carry-out (C_{out}), result Equal-to Zero (ZE) conditions with or even earlier than the result. The proposed logic is reconfigurable, and can calculate all the conditions (mentioned above) for one 64-bit, two 32-bit, four 16-bit, or eight 8-bit signed and unsigned operands. The reconfigurability of the logic is achieved using only four control signals.

This paper is organized as organized as follows. In section 2, we will explain the condition resolution based Carry signal. In section 3, we present the design of the partitioned branch condition resolution circuit [8]. Finally, the VLSI implementation and comparisons are presented in section 4.

2. Condition Resolution based on Carry Signal

2.1. Overflow, Underflow

In multimedia arithmetic, we deal mostly with 8- or 16-bit pixel data, that represent the color intensity and luminous information. To correctly represent this data, we cannot use modulo arithmetic in which the overflow is ignored, because a small change in color intensity or luminous may result a huge a change in the resulting information. For example, a change from FFHex to 00Hex (is a change from white to black). Therefore, in media signal processing, we use saturated arithmetic. In the case of saturated arithmetic, it is necessary to clamp overflowing results to a high or low value.

There are two modes of saturation: asymmetric and symmetric. For unsigned integers, the two modes are the same. While, for signed values, symmetric mode saturates positive and negative numbers to the same absolute value: for instance, signed byte saturation values of -127 and +127. The asymmetric mode will saturate the negative value to a number, which is an absolute value of one greater than the positive value; for instance, -128 and +127. The conditions for asymmetric mode of saturation resolution based on overflow/underflow are summarized for Add and Subtract operations in Table 1 and Table 2, respectively. In these tables, A and B are the input operands, and As and Bs are the sign bits of these operands. In the case of unsigned addition, an overflow occurs when carry out is '1' and the result is saturated to the maximum value. While, in the case of unsigned subtraction, an underflow occurs when carry out (C_{out}) is '0' and the result is saturated to the minimum value. Similarly, in the case of signed addition, an overflow occurs when both the input operands are positive and carry out and carry out-1 (C_{out-1}) are '1'. In this case, the result is saturated to the maximum positive value. These rules are summarized in Table 1 and Table 2.

Table 1. Overflow and underflow detection for Add operation

Data Type	**MSB**	**Oper-ation**	**Overflow**	**Underflow**
Unsign		A + B	$C_{out} = 1$	
Sign	$A_s = 0$, $B_s = 0$	A + B	C_{out} XOR C_{out-1}	
	$A_s = 1$, $B_s = 1$	A + B		C_{out} XOR C_{out-1}
	$A_s = 1$, $B_s = 0$	A + B		
	$A_s = 0$, $B_s = 1$	A + B		

Table 2. Overflow and underflow detection for Subtract operation

Data Type	**MSB**	**Oper-ation**	**Overflow**	**Underflow**
Unsign		A - B		$C_{out} = 0$ (A<B)
Sign	As = 0, Bs = 0	A - B		
	As = 1, Bs = 1	A - B		
	As = 1, Bs = 0	A - B		C_{out} XOR C_{out-1}
	As = 0, Bs = 1	A - B	C_{out} XOR C_{out-1}	

2.2. Min, Max, Compare

The major operation required in Min, and Max operation is the generation of flags for the conditions A > B

(Greater-than (GT)), A < B (Less-than (LT)), or A = B (equal-to (EQ)). Based on these signals the selection of Maximum and Minimum operands is performed. Subtracting B from A and looking at the Carry_out signal of the result generates these conditions. Once the conditions are resolved, then the output multiplexers select of correct result. Therefore, by calculating the Carry_out signal, the implementation of three operations Compare, Min, and Max is straightforward as shown in Table 3.

Table 3. GT (Max), LT (Min). and Equal to Condition Resolution using A - B operation. As, Bs are the MSB of the inputs A and B, C_{out} is the carry generated in A - B Operation, and != is not Equal to

Data Type		**A < B**	**A > B**	**A = B**
Unsign		C_{out} = 0	C_{out} = 1 And A != B	A XNOR B
Sign	A_s = 0, B_s = 0	C_{out} = 0	C_{out} = 1 And A != B	A XNOR B
	A_s = 1, B_s = 1	C_{out} = 0	C_{out} = 1 And A != B	A XNOR B
	A_s = 1, B_s = 0	C_{out} = 1		A XNOR B
	A_s = 0, B_s = 1		C_{out} = 0	A XNOR B

3. Design

In this section we present the design of the basic condition resolution logic without any partition. In the following, we explain their interfacing in order to obtain the 64-bit partitioned condition resolution circuit. Let A = $(a_{n-1} .... a_0)$, and B = $(b_{n-1} .... b_0)$ be the two inputs, with a_{n-1} and b_{n-1} be the sign bits. The conditions that need to be tested are Equal-to (EQ), Greater-than (GT), Less-than (LT), Overflow (OV), Underflow (UF) and Carry-out (C_{out}).

The GT, LT conditions are calculated by subtracting B from A, and looking at the carry_out, and sign bits of A and B (see Table 3). The UF, OV and C_{out} conditions are calculated using the sign bits of A and B, C_{out}, and C_{out-1} produced during the ADD or Subtract operation. The calculation of these conditions is performed according Tables 1 and 2, for ADD and Subtract operations respectively. The EQ (A = B) condition is calculated using an XNOR-AND tree. It can be seen from the above that all the conditions can be calculated using the Carry-out (C_{out}) signal. Hence we design a fast carry detection circuit.

In this design we have used a modified Carry Look-ahead scheme. It is based on the fact that the carry generated (G) at bit position i ($G_i = a_i \bullet b_i$, where $\bullet$ is the AND operation) can be dropped in the computation of the group carry, if the two input bits at any position j ($j > i$) are zero. We call this function NZ (no zero), and it's value at bit position i is, $NZ_i = a_i + b_i$, where + is the OR operation.

Since, the functions G_i and NZ_i (i= *n-1 to 0*) can be generated simultaneously from A and B, the group carry C_{out} can be computed in 3 logic levels (if there is no limitation on fan in/out).

$$C_{out} = NZ_{n-1} \bullet \ldots\ldots \bullet NZ_2 \bullet NZ_1 \bullet G_0 + NZ_{n-1} \bullet \bullet NZ_2 \bullet G_1 + \ldots\ldots + G_{n-1} \qquad \text{Equation 1}$$

Based on Equation 1 a circuit (as shown in Fig. 1) has been designed for the detection of C_{out}. Due to fan in/out limitations the circuit has been designed for four bits only. The C_{out} (group carry) is produced in 3-gate delays without any XOR gate or tri-state buffer as required in [2] [3]. The NZ_{out} (group not zero) is produced after 2-gate delays. The evaluation of A>B, or A<B, based on an-1, bn-1, C_{out} and A=B, requires a delay of two more additional XOR gates for all methods [2]. The cascading of the 4-bit circuit for 16-bit is shown in Fig. 2.

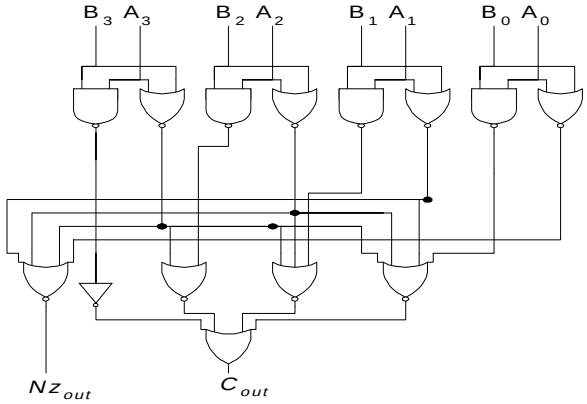

Fig. 1. Schematic diagram of the 4-bit fast carry and equal to detector.

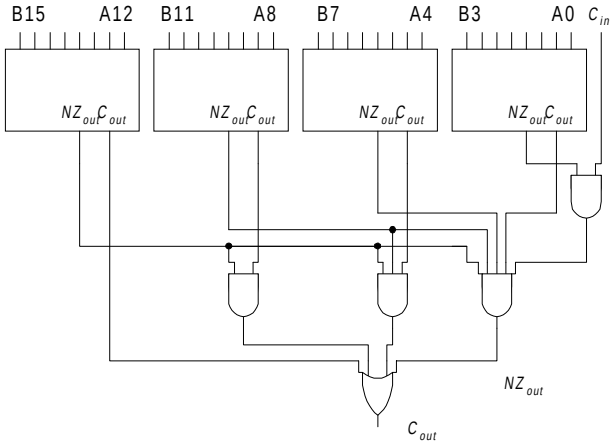

Fig. 2. Cascading of 4-bit fast carry and equal to modules to obtain 16-bit module.

3.1. Partitioned Carry Logic

The main motivation behind the design of this circuit is to calculate multiple independent flags/conditions (OV, UF, GT, LT, EQ, C_{out}) with different word-lengths using only one circuit. This is the key requirement for Media processors. Although it seems very simple to control the partition by just controlling the carry-in of each partitioned block, in reality this is not feasible for high-speed operation, because the carry chain is on the critical path of the circuit. If we insert the control gate on the carry chain, then the delay of the largest word size is increased in proportion to the number of partitions.

In this section we provide a very simple solution for partitioning the adder without producing any delay in the critical path. In the following, we will show the partition of a 16-bit circuit into four 4-bit circuits, but it can be extended to any partition in multiples of 4. Consider a 16 - bit carry generate logic (Fig. 2) formed using the 4-bit groups. The group carry and *NZ* signals are C_{0-3} and NZ_{0-3}, for the first group, C_{4-7} and NZ_{4-7} for the second group and so on. The carries out of each block are calculated using Equation 1.

$$C_{0-7} \quad C_{4-7} + NZ_{4-7}\, C_{0-3}$$

$$C_{0-11} \quad C_{8-11} + NZ_{8-11}\, C_{0-7}$$

$$C_{8-11} + NZ_{8-11}\, C_{4-7} + NZ_{8-11}\, NZ_{4-7} C_{0-3}$$

$$C_{0-15} \quad C_{8-15} + NZ_{8-15}\, C_{0-7}$$

$$C_{12-15} + NZ_{12-15}\, C_{8-11} + NZ_{12-15}\, NZ_{8-11}\, C_{0-7}$$

$$C_{12-15} + NZ_{12-15}\, C_{8-11} + NZ_{12-15}\, NZ_{8-11} C_{4-7} + NZ_{12-15}\, NZ_{8-11}\, NZ_{4-7}\, C_{0-3}$$

Now if we want to divide this adder into 4 carry generate circuits of 4-bit each, then we need to make C_3 equal zero for the block generating C_{0-7}, and the rest of the blocks. Similarly C_7 equal zero for the block generating C_{0-11}, and the rest of the blocks. Finally C_{11} equal zero for the block generating C_{0-15}. At the same time we need these carries for the generation of OV, UF, GT, LT conditions. A simple solution to solve this problem is to make NZ_{4-7} = 0 (NZ_{8-11}, NZ_{12-15} for the other partitions). By making this NZ signals = 0, all the carries dependent on this propagate signal will become '0' and at the same time the C_{out} generated by a block remains valid for the calculation of OV, UF, GT, LT conditions. Since the NZ signal is not on the critical path (one gate less than C in the path), its control is easy and does not require any delay. We have extended the same partitioning technique for partitioning the carry logic into byte, half-word, word, and double word. The partitioning of the circuit has been performed using two signals (Part0 and Part1). The circuit operation is based on these signals as shown in Table 4.

Table 4. Adder operation controlled by part0 and part1

Part1	Part0	Adder Operation
0	0	Byte
0	1	Half-word (16-bit)
1	0	Word (32-bit)
1	1	Double-word (64-bit)

As shown earlier, the basic blocks required for condition resolution are the carry generation C_{out}, blocks (see Fig. 1), these blocks generate the C_{out} (and C_{out-1}), and NZ signals respectively. The proposed logic can resolve the conditions for eight 8-bit operands packed as 64 bits and produce eight independent sets of GT, LT, and C_{out} for 64-bit input. The partition of all the blocks is performed using part0, and part1 control signals, and signed / unsigned operands are specified by the sign signal.

In order to produce the correct carries and sign of the input operands for different data types, a MUX based block is used. This block produces the sign, and carries for each 8-bit group, and part0 and part1 control its function. The output of MUX block for different data types is shown in Table 5 and Table 6. Finally, the outputs of the MUX block, and EQ (Equal_to) signals are passed to eight independent logic blocks to generate the GT and LT conditions.

Table 5. The C_{out} of the MUX-block for different data types

Double-word	Word	Half-word	Byte	C_{out}
$C_{out63-0}$	$C_{out63-32}$	$C_{out63-48}$	$C_{out63-56}$	C_{out7}
$C_{out63-0}$	$C_{out63-32}$	$C_{out63-48}$	$C_{out55-48}$	C_{out6}
$C_{out63-0}$	$C_{out63-32}$	$C_{out47-32}$	$C_{out47-40}$	C_{out5}
$C_{out63-0}$	$C_{out63-32}$	$C_{out47-32}$	$C_{out39-32}$	C_{out4}
$C_{out63-0}$	$C_{out31-0}$	$C_{out31-16}$	$C_{out31-24}$	C_{out3}
$C_{out63-0}$	$C_{out31-0}$	$C_{out31-16}$	$C_{out23-16}$	C_{out2}
$C_{out63-0}$	$C_{out31-0}$	$C_{out15-0}$	$C_{out15-8}$	C_{out1}
$C_{out63-0}$	$C_{out31-0}$	$C_{out15-0}$	C_{out7-0}	C_{out0}

Table 6. The Sign of the MUX-block for different data types

Double-word	Word	Half-word	Byte	C_{out}
A_{63}	A_{63}	A_{63}	A_{63}	A_{s7}
A_{63}	A_{63}	A_{63}	A_{55}	A_{s6}
A_{63}	A_{63}	A_{47}	A_{47}	A_{s5}
A_{63}	A_{63}	A_{47}	A_{39}	A_{s4}
A_{63}	A_{31}	A_{31}	A_{31}	A_{s3}
A_{63}	A_{31}	A_{31}	A_{23}	A_{s2}
A_{63}	A_{31}	A_{15}	A_{15}	A_{s1}
A_{63}	A_{31}	A_{15}	A_{7}	A_{s0}

3.2. Partitioned EQ (Equal-To) Logic

In this section, we present the design of a reconfigurable Equal-to Detector. This circuit can detect result-equal-zero for 64, 32, 16, or 8-bit independent operands for add and subtract operations. The Equal-to logic is obtained by XNORing of the two inputs (A and B) and then ANDing the XNOR output using an n-input AND gate (obtained using a tree of 2-input AND gate). The input to the 64-bit partitioned equal-to detector are two 64-bit vectors A and B, partition control signals part0, and part1. The outputs are EQ $_{0...7}$. The partition of the logic is performed according to Table 4. The equal-to logic produces 8-independent outputs and they represent different values for different data types (see Table **7**). In the case of byte operation all the outputs are different, while in the case of half-word operation EQ_7 and EQ_6; EQ_5 and EQ_4 ; EQ_3 and EQ_2 ; and EQ_1 and EQ_0 are the same. In the case of word operation, EQ_7 , EQ_6 , EQ_5 , and EQ_4; and EQ_3 , EQ_2 , EQ_1, and EQ_0 are all the same. In case of double word, all the outputs are the same.

Table 7. The outputs of the EQ logic for different data types. Note: a_{63} is the 64-bit input operand Data

Double-word	Word	Half-word	Byte	Equal-To
a_{63-0}	a_{63-32}	a_{63-48}	a_{63-56}	EQ_7
a_{63-0}	a_{63-32}	a_{63-48}	a_{55-48}	EQ_6
a_{63-0}	a_{63-32}	a_{47-32}	a_{47-40}	EQ_5
a_{63-0}	a_{63-32}	a_{47-32}	a_{39-32}	EQ_4
a_{63-0}	a_{31-0}	a_{31-16}	a_{31-24}	EQ_3
a_{63-0}	a_{31-0}	a_{31-16}	a_{16-23}	EQ_2
a_{63-0}	a_{31-0}	a_{15-0}	a_{15-8}	EQ_1
a_{63-0}	a_{31-0}	a_{15-0}	a_{0-7}	EQ_0

4. VLSI Implementation and Comparison

The VLSI implementation was done in order to compare the chip area required by [2] and the proposed circuit, and to obtain more realistic device characteristics for accurate HSPICE simulation. The complete chip layout is shown in Fig. 3. The simulations on the extracted layout using HSPICE for the case A=11111111, B=0000000x (b_0=1/0), and C_{in} = 0 are presented in Fig. 4. The delay and area of the two circuits are shown in Table 8. Simulations performed for 2-mm n-well CMOS technology with a 2.5V supply, at 25^{o}C show a 30% increase in speed and 25% decrease in area in comparison to [2].

Table 8. Area and delay (with pad frame) of the chip for 8-bit operands

Circuit	Area	Delay
[1]	417ux618u	1.04 ns
Proposed	402ux509u	0.78 ns

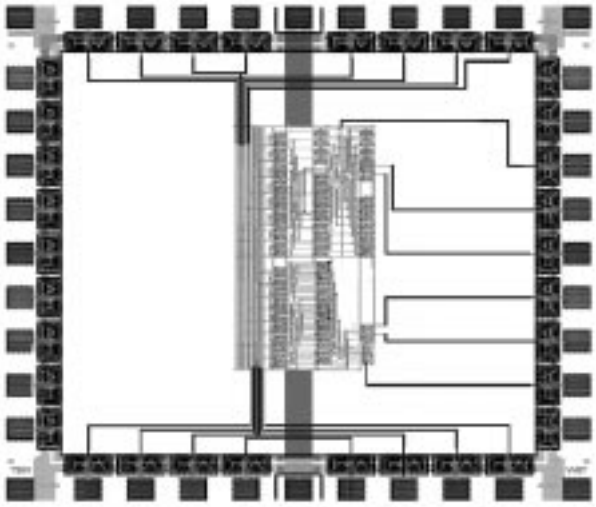

Fig. 3. VLSI Layout of the chip in 2-µm n-well CMOS technology.

Comparisons of the cost and delay of the four algorithms for the detection of C_{out} has been made by [2]. We have used the same criteria and values, to compare our design with the existing designs.

Table 9 shows the asymptotic analysis for n ≤ 64. We have also simulated our design for n = 4, 16, 32, and 64 bits, the simulation results are presented in Table 10. The simulations are performed for 0.2-µm technology, with 2.5V supply and 25^{o}C. These results show a 64-bit carry generation in 679ps.

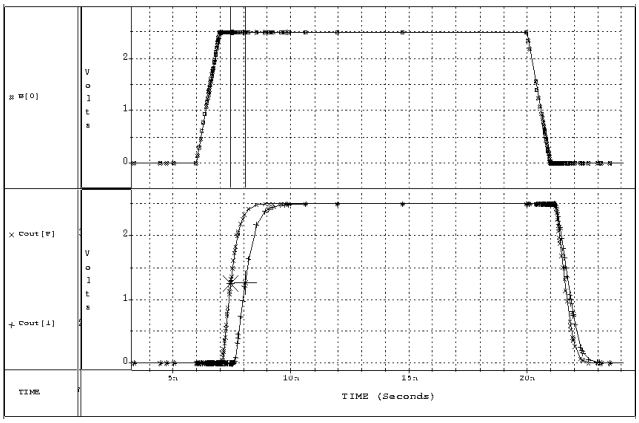

Fig. 4. HSPICE Simulation results of the chip, top input, and bottom C_{out}[P] (proposed, legend ^{x}) and C_{out}[2] (legend +).

Table 9. Asymptotic cost of different schemes for the detection of carry

Method	**Delay**	**Gates**
Carry-lookahead	2 log n + 1	5n-3
Brent	log n + O($\sqrt{\log n}$)	<7n
Bit prefix-and	log n + 3	<7n
Group prefix-and	log n + 3	5n-4
Proposed	$2\lceil \log n/2 \rceil + 1$	4n

Table 10. HSPICE simulation results for n=4, 16, 32, and 64 bit

	n = 4	n=16	n=32	n=64
C_{out}	289 ps	483 ps	642 ps	679 ps

5. Conclusion

This paper presents the design of a branch condition resolution circuit. The proposed circuit works in parallel with the arithmetic unit, and calculates the Equal to (EQ), Greater-than (GT), Less-than (LT), Overflow (OV), Underflow (UF), and Carry-out (C_{out}), result Equal-to Zero (ZE) conditions with or even earlier than the result. The proposed logic is reconfigurable, and can calculate all the conditions (mentioned above) for one 64-bit, two 32-bit, four 16-bit, or eight 8-bit signed and unsigned operands. The reconfigurability of the logic is achieved using only four control signals.

5. Acknowledgments

The authors thank the reviewers for their valuable comments.

References

[1] Hennessy, John L. and Patterson, David A, *Computer Architecture: A Quantitative Approach: Second Edition*, Publisher Morgan Kaufman Publishers, Inc. San Francisco, California.

[2] David R. Lutz and D. N. Jayasimha, "The half adder form and early branch condition resolution", *13th Symposium on Computer arithmetic;* 266-273, 1997.

[3] David R. Lutz and D. N. Jayasimha, "Comparison of two's complement numbers", *International Journal of Electronics;* 513-523, April 1996.

[4] Arnold Weinberger, "High Speed zero sum detection", *4th Symposium on Computer arithmetic*; 200-207, 1975.

[5] M. Putrino, and S. Vassiliades, "Resolution of branching with prediction", *International Journal of Electronics*; 163-172, February 1989.

[6] Burns J. L. and K. J. Nowka, "Parallel condition code generation for high frequency PowerPC Microprocessors", *1998 Symposium on VLSI Technology and VLSI Circuits.*

[7] Jordi Cortadella, and M. Llaberia, "Evaluation of A+B=K conditions without carry propagate", *IEEE Transactions on Computer*; 1484-1488, 1982.

[8] Farooqui, A.A.; Oklobdzija, V.G.,"VLSI implementation of early branch prediction circuits for high performance computing", Proceedings. *9th Great Lakes Symposium on VLSI*, 1999 , Page(s): 30 -33.