Improving Cache Hit Rate Using The Control Flow Graph

Musbah Mohammed Elahresh
Department of post graduate studies
College of Electronic Technology, Tripoli–Libya
E-mail:**Algab1402@gmail.com**

Abedelatie Ali Elaraby
Department of Electrical and Computer Engineering
Elmergib University, Alkhoms–Libya
Email: **Alatie2001@yahoo.com**

Abstract

This paper provides a technique for designing a cache control unit that speeds up program execution time. This feature is highly required for modern computers to enhance system performance and efficiency. The technique focuses on solving the problem of cache misses by utilizing the control flow graph of the program behavior during its loading from the main memory and executing from the cache by the processor.The proposed cache control unit performs its task in two stages that work in parallel. These stages are implemented by the following circuits:

1- Loader circuit that loads program blocks from main memory into cache lines.
2- Replacement circuit that manages the cache lines by placing the coming program blocks into the proper cache lines and performing the replacement without misses.

This solution required that a program has to be logically partitioned according to its control flow graph into basic blocks with one exit point. This results in variable-sized program blocks to be loaded into the cache. There by in the cache there exists a block with its two successors blocks. The selection of next block to be executed from these two successors depends on the condition of the exit point of the parent block (taken or not taken branch). Thus always the next block to be executed is available in the cache. The design of the loader circuit and the replacement circuit are given in details and their functionalities are simulated. Program partitioning and the relations between program blocks are assumed to be collected from other job in a form of profile data. This data is used by the proposed circuit to control its operations and synchronizing its functions.

I. INTRODUCTION

Computers nowadays play an important role in our everyday life. Many factor increases the people to depend in computer. One of these factors is its performance represented by the speed of programs execution. Many technique where developed and are still used to enhance computer performance. One of these techniques is the use of cache memory of small capacity and less access time. This introduced many techniques and methodology to map program block between the main memory and the cache memory and which blocks should be available in cache for the processor to execute next. All that solutions utilize which so called locality principles.[1][2]

As it is known any program has many execution paths. The program blocks and the execution paths are modeled by what is so called ***Control Flow Graph CFG***. Nodes of the graph represent the program blocks and the edges represent the execution paths [5].The solution of the above mentioned problem is based on portioning the program into blocks of fixed number of exit points. In this paper the implementation of the hardware circuit of cache memory management unit is introduced and simulated.

II. CFG Example

Consider the following of code:

Program

```
x = z-2 ;
y = 2*z;
if (c) {
x = x+1;
y = y+1;
}
else {
x = x-1;
y = y-1;
}
z = x+y;
```

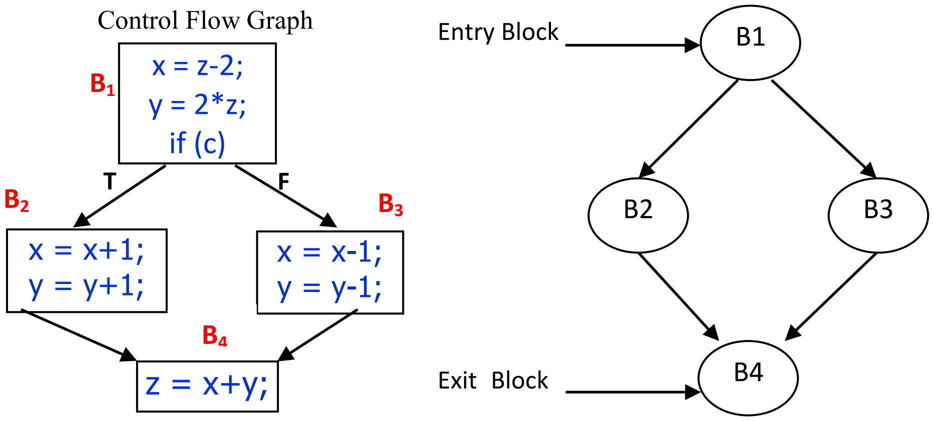


Figure (1) Control Flow Graph Example

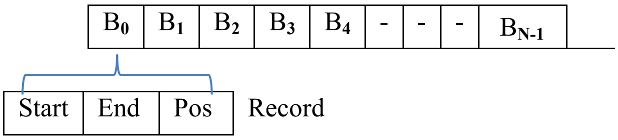
In this example, we have 4 basic blocks in particular, in this case, B1 is the **"entry block ",** B4 the **" exit block ".**A graph for this fragment has edges from B1 to B2, B1 to B3, B2 to B4, and B3 to B4 as shown in figure (1).

Possible execution =path in the graph

Possible Execution 1:
– c is true
– Program executes
- basic blocks B1, B2, B4

Possible Execution 2:
– c is false
– Program executes
- basic blocks B1, B3, B4

A. CFG Profile Representation



Start: starting address in maim memory.
End: end address in maim memory.
Pos: index in CFG data structure of the next block if jump is taken, if jump is not taken then follow next record.

B. CFG Organization

CFG organization conceptually a binary tree and physical as a graph.

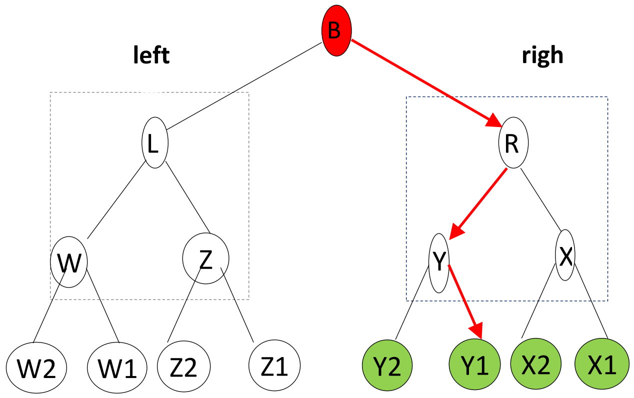


Figure (2) CFG A Binary Tree

The tree is chosen to represent the control flow graph since each basic block has at most two successors; one is entered by a control transfer instruction and the other one is the continuation block (next adjacent block followed in the main memory).[5]

D. Cache Organization

Blocks are staff in the cache start root, level one right, left and level two right , left , etc.., of the binary tree from 7 nodes tree.

Buffer address	Physical address	data	
000	0	B0	root
001	1	R0	Level one
010	2	L0	
011	3	X	Level two
100	4	Z	
101	5	Y	
110	6	W	

Figure (3) CFG Cache Organization

At first load = saturation

Next only 4 blocks are loaded, 2 for each sub tree. Staff here from the right left and last one position B0

III. Functional Units

The general block diagram is illustrated in Figure (6) below

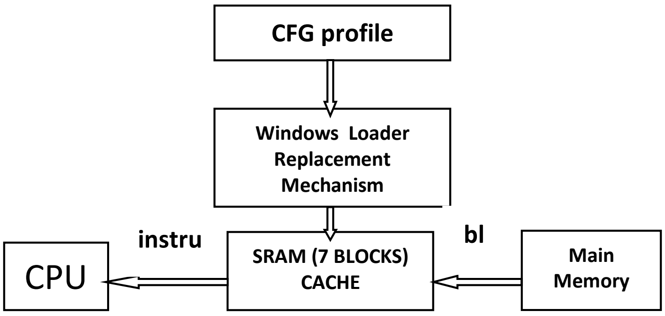


Figure (4) General Block Diagram for CFG Cache Organization

This block diagram shows idea for replacement and loader blocks from main memory to the cache memory. Execution program is divided into blocks, CFG profile contains these address blocks. At first start program execution, CPU loads these blocks in main memory, loader mechanism loads address for the needed four blocks (right or left sub tree) to main memory, main memory loads these block to the cache memory in free location, and at the same time the replacement mechanism provides free location in the cache. All needed execution blocks are ready in cache memory previously by loader mechanism.

III. Load Mechanism

This circuit used to load mechanism is illustrated in figure (5) below, this part is used to load address blocks from CFG profile to main memory.

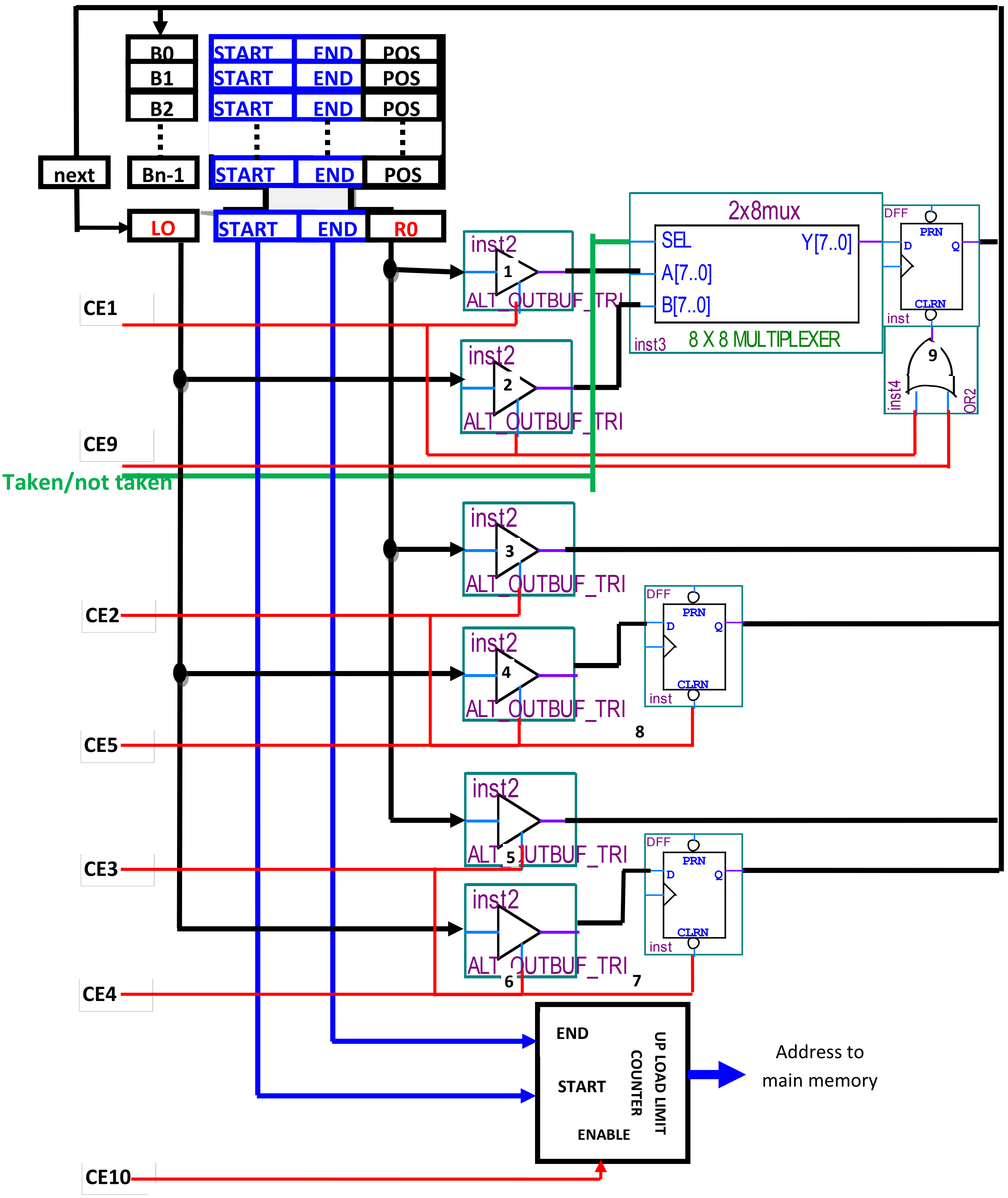


Figure (5) The Loader Mechanism Circuit

IV. Loader Circuit Operation

When the program start execution , CPU load blocks for program from CFG to main memory and load address for these block in CFG profile.

The load circuit operate is illustrated in flow chart in figure (6) below.

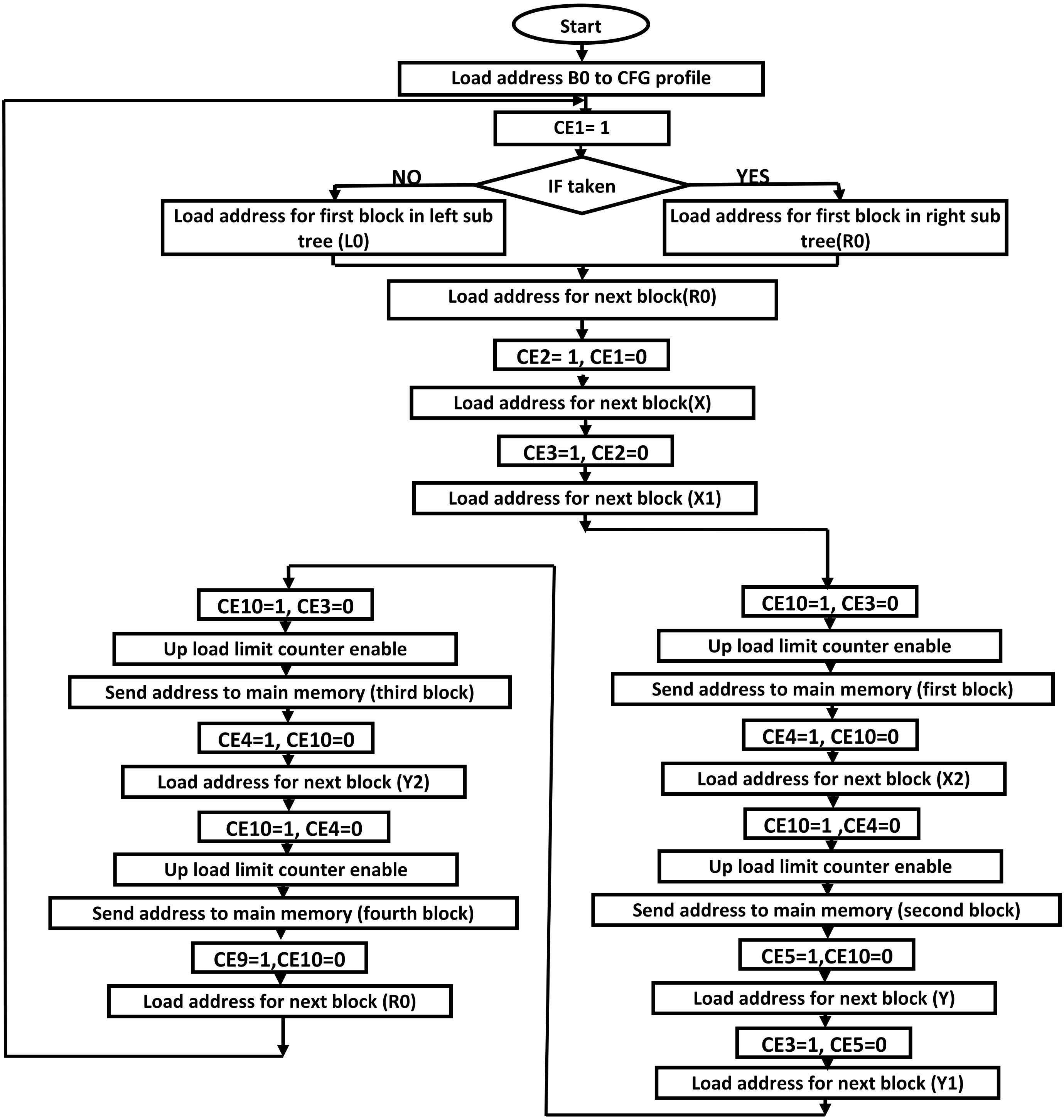

Start
Load address B0 to CFG profile
CE1= 1
IF taken
NO
YES
Load address for first block in left sub tree (L0)
Load address for first block in right sub tree(R0)
Load address for next block(R0)
CE2= 1, CE1=0
Load address for next block(X)
CE3=1, CE2=0
Load address for next block (X1)
CE10=1, CE3=0
Up load limit counter enable
Send address to main memory (third block)
CE4=1, CE10=0
Load address for next block (Y2)
CE10=1, CE4=0
Up load limit counter enable
Send address to main memory (fourth block)
CE9=1,CE10=0
Load address for next block (R0)
CE10=1, CE3=0
Up load limit counter enable
Send address to main memory (first block)
CE4=1, CE10=0
Load address for next block (X2)
CE10=1 ,CE4=0
Up load limit counter enable
Send address to main memory (second block)
CE5=1,CE10=0
Load address for next block (Y)
CE3=1, CE5=0
Load address for next block (Y1)

Figure (6) Flow Chart For Load Circuit

V. Replacement Mechanism

This circuit used in replacement mechanism is illustrated in figure (7) below.

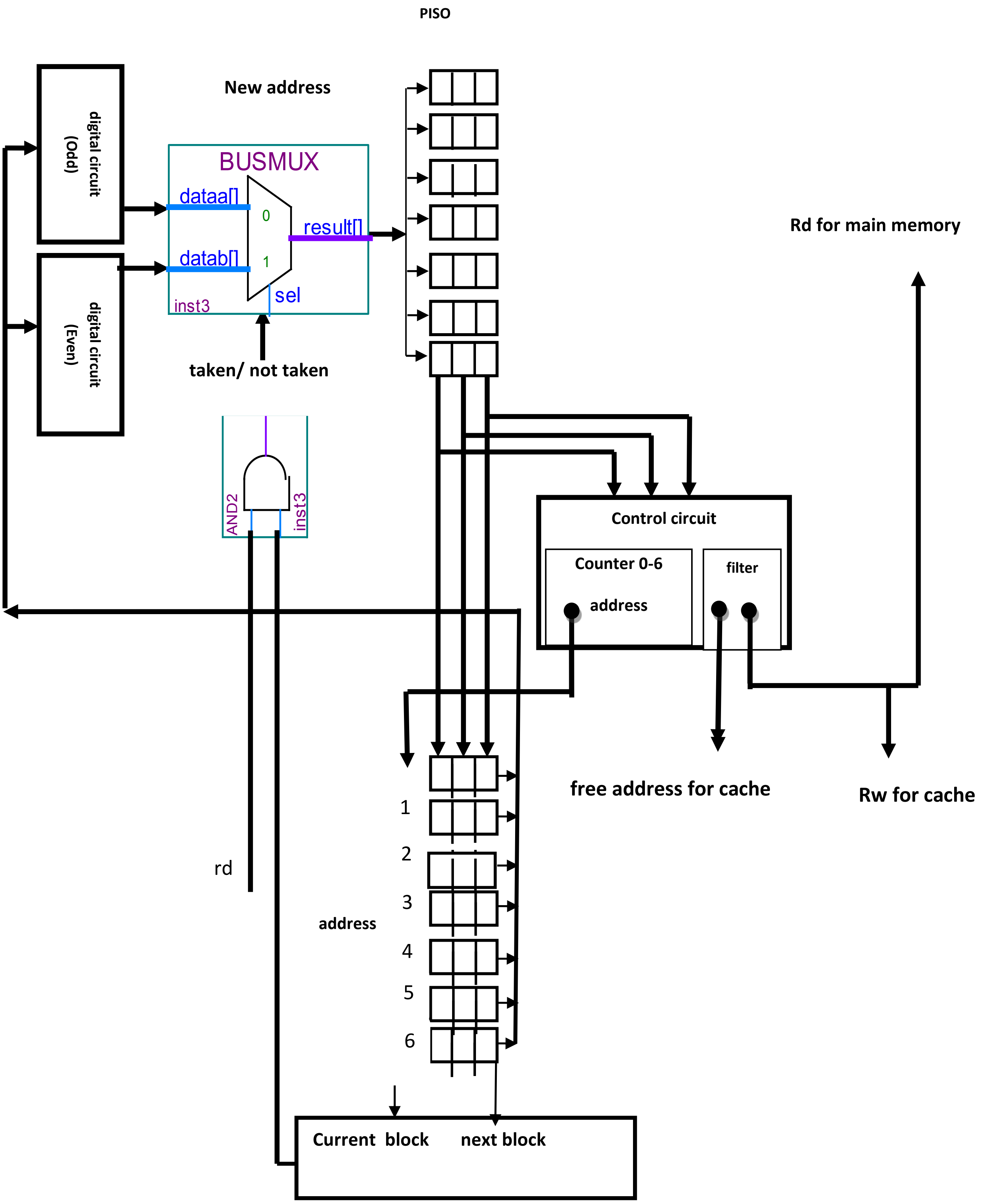


Figure (7) The Replacement Mechanism Circuit

VI. Replacement Circuit Operation

The main idea for the operation replacement circuit is shown in figure (7). It converts addresses **1,3,5 to 0,1,2** and saves its contents, because it can be used in the future, and **addresses 0,2,4,6 are converted to 3,5,4,6** and considered free position. We reload it by new data from main memory if the control signal is taken. If the control signal is not taken, the proposed replacement circuit converts **addresses 2,4,6 to 0,1,2** and saves its content because it can be used in the future. And the **addresses 0,1,3,5 convert to 3,5,4,6** and considered free position. We reload it by new data from main memory. The replacement circuit operate is illustrated in flow chart in figure (8) below

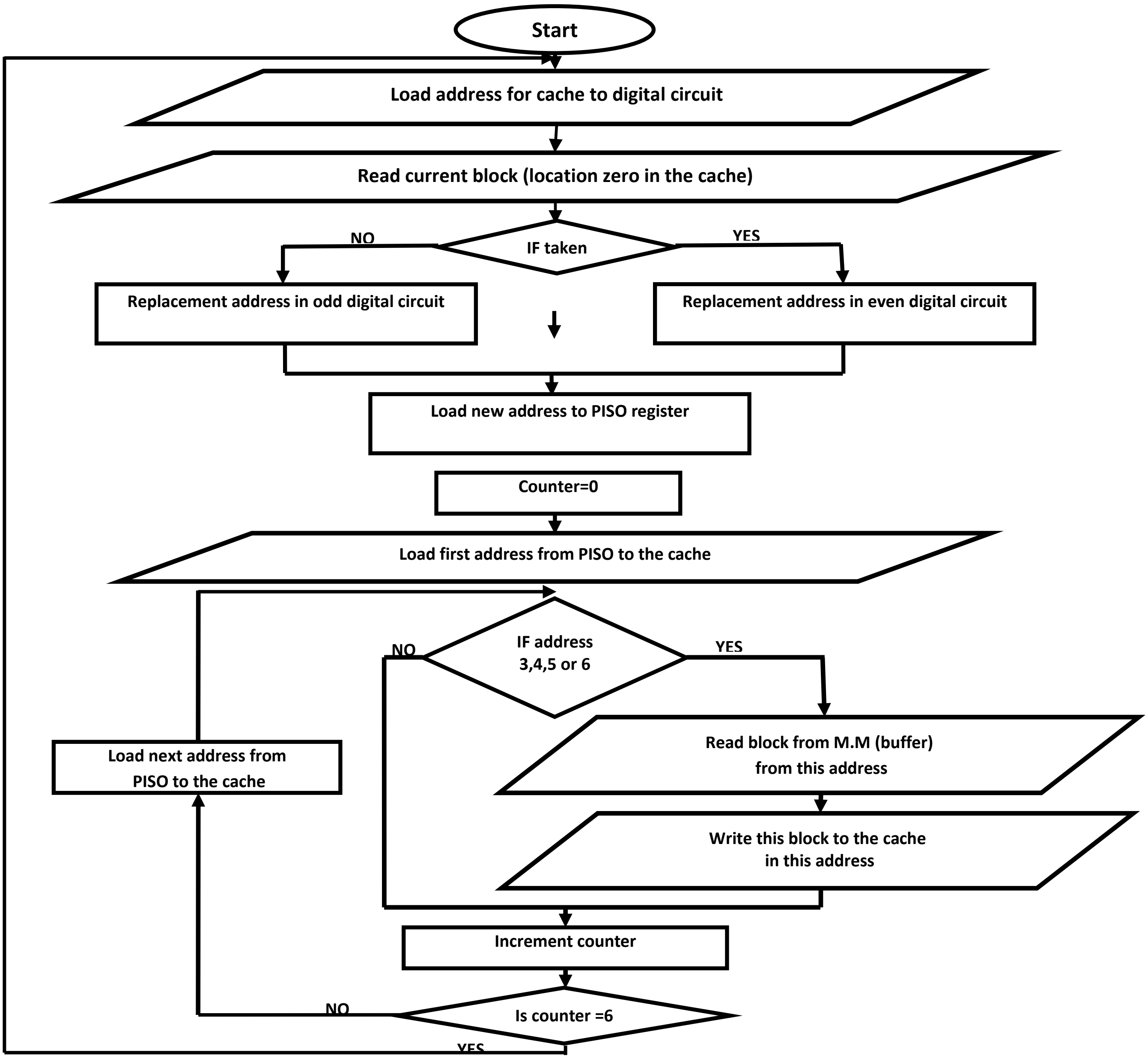

Start
Load address for cache to digital circuit
Read current block (location zero in the cache)
NO
IF taken
YES
Replacement address in odd digital circuit
Replacement address in even digital circuit
Load new address to PISO register
Counter=0
Load first address from PISO to the cache
NO
IF address 3,4,5 or 6
YES
Read block from M.M (buffer) from this address
Write this block to the cache in this address
Load next address from PISO to the cache
Increment counter
NO
Is counter =6
YES

Figure (8) Flow Chart for Replacement Circuit

Results and Discussion

In figure (9), **path of execution in red color (path 1)**, current block transfers from B0 to R0, which means the branch is taken and the window is shifted to right on the tree and odd logic circuit converts addresses 0,1,2,3,4,5,6 to 3,0,5,1,4,2,6 and blocks B0,L0,Z,W are replaced with new loaded blocks X1,X2,Y1,Y2. In the next step, current block transfers from R0 to Y, the branch is not taken and the window is shifted to left on the tree and even logic circuit converts addresses 3,0,5,1,4,2,6 to 4,3,6,5,1,0,2 and blocks R0,X,X1,X2 are replaced with new loaded blocks Y3,Y4,Y5,Y6. In the next step, current block transfers from Y to Y1, the branch is taken and the window is shifted to right on the tree and odd logic circuit converts addresses 4,3,6,5,1,0,2 to 4,1,6,2,0,3,5 and blocks Y,Y2,Y5,Y6 are replaced with new loaded blocks Y7,Y8,Y9,Y10. In the following, current block transfers from Y1 to Y4, the branch is not taken and the window is shifted to left on the tree and even logic circuit converts addresses 4,1,6,2,0,3,5 to 1,5,2,0,3,4,6 and blocks Y1,Y3,Y7,Y8 are replaced with new loaded blocks Y19,Y20,Y21,Y22 And etc…

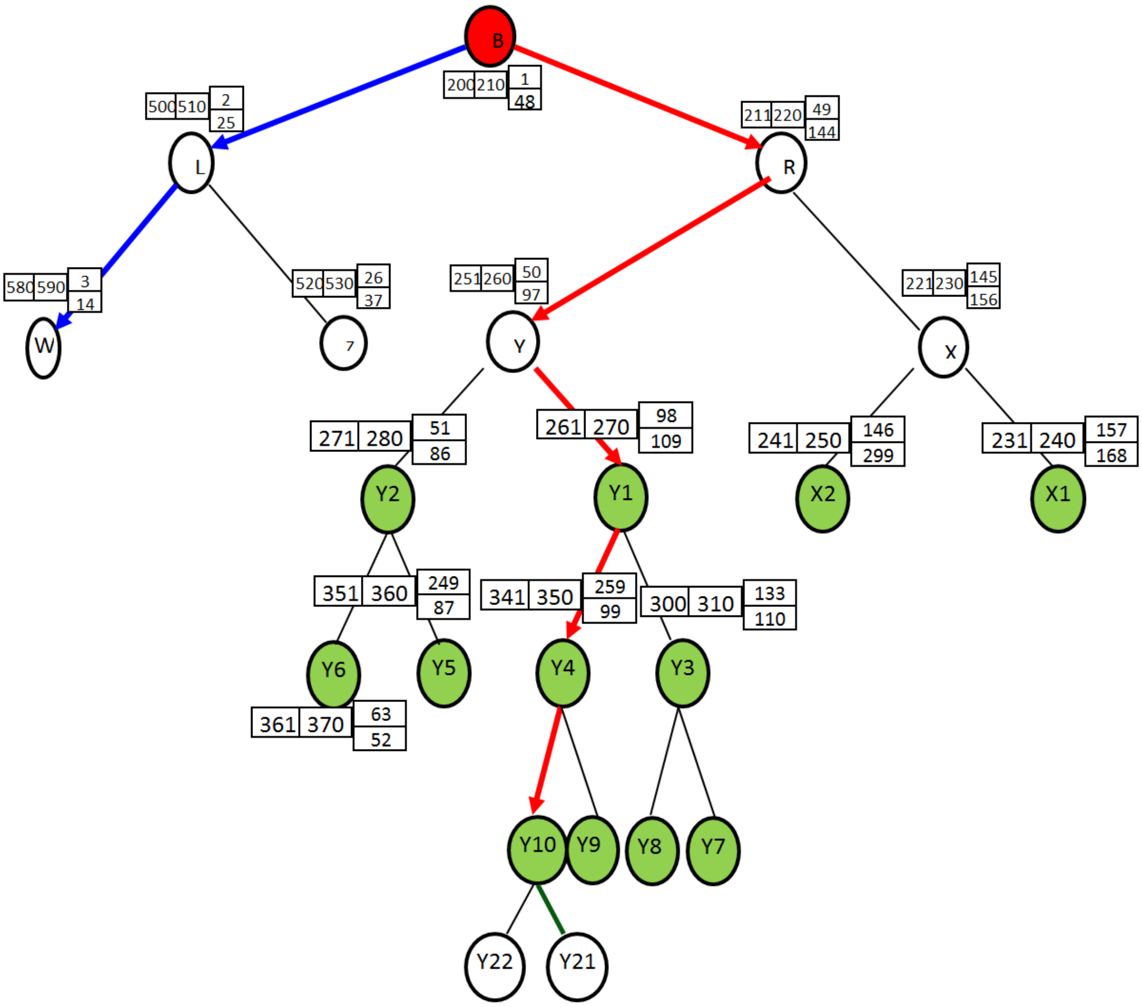
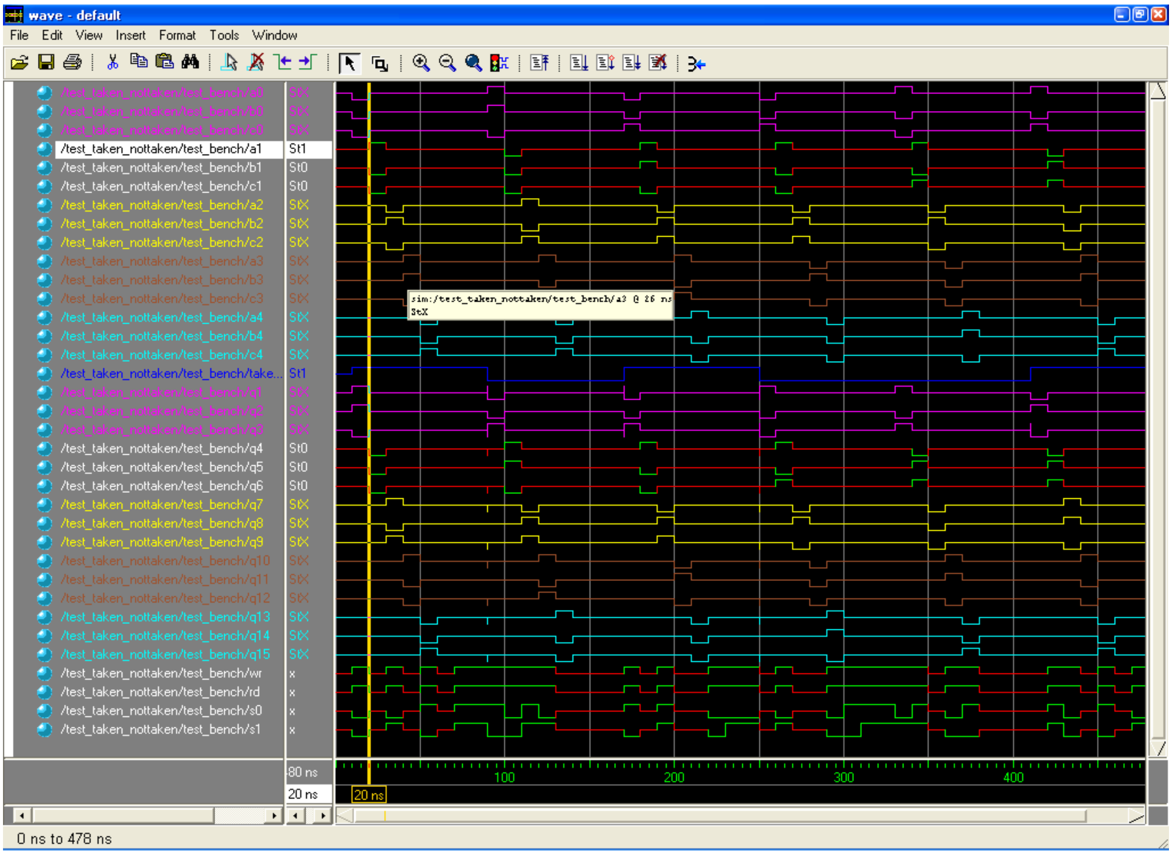


Figure (9) Binary Tree Represents The Control Flow Graph

Table (1) CFG profile information

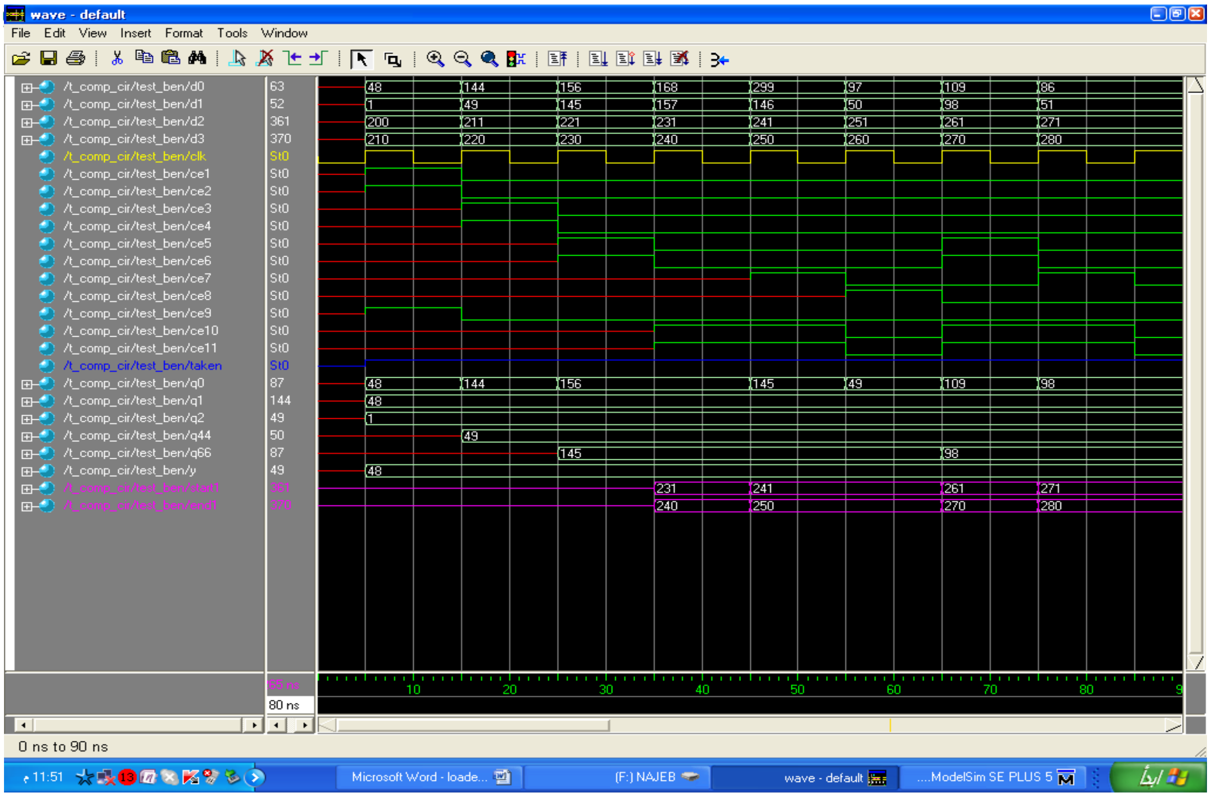
Block Address	Block Name	Start	End	Jump	NEXT
0	B0	200	210	48	1
1	L0	500	510	25	2
2	W	580	590	14	3
3	W2	610	620	179	4
4	W6	690	700	…	5
14	W1	590	600	189	15
15	W4	650	660	290	16
16	W10	770	780	…	17
25	Z	520	530	37	26
26	Z2	560	570	199	27
27	Z6	…	…	…	28
65	Y30	…	…	…	66
75	Y19	…	…	239	76
76	Y28	…	…	…	77
86	Y5	351	360	249	87
87	Y12	411	420	…	88
97	Y1	261	270	109	98
98	Y4	341	350	259	99
99	Y10	860	870	310	100
109	Y3	300	310	133	110
110	Y8	820	830	122	111
111	Y18	…	…	269	
158	X10	…	…	…	159
168	X3	…	…	319	169
169	X8	…	…	…	170
179	W5	670	680	…	180
189	W3	630	640	280	190
190	W8	730	740	…	191
199	Z5	…	…	…	200
209	Z3	…	…	…	210
219	Y21	…	…	…	220

The result obtained from execution program for the replacement circuit by ModelSim-Altera 6.1g (Quartus II 7.2), is illustrated in figure (10) below.



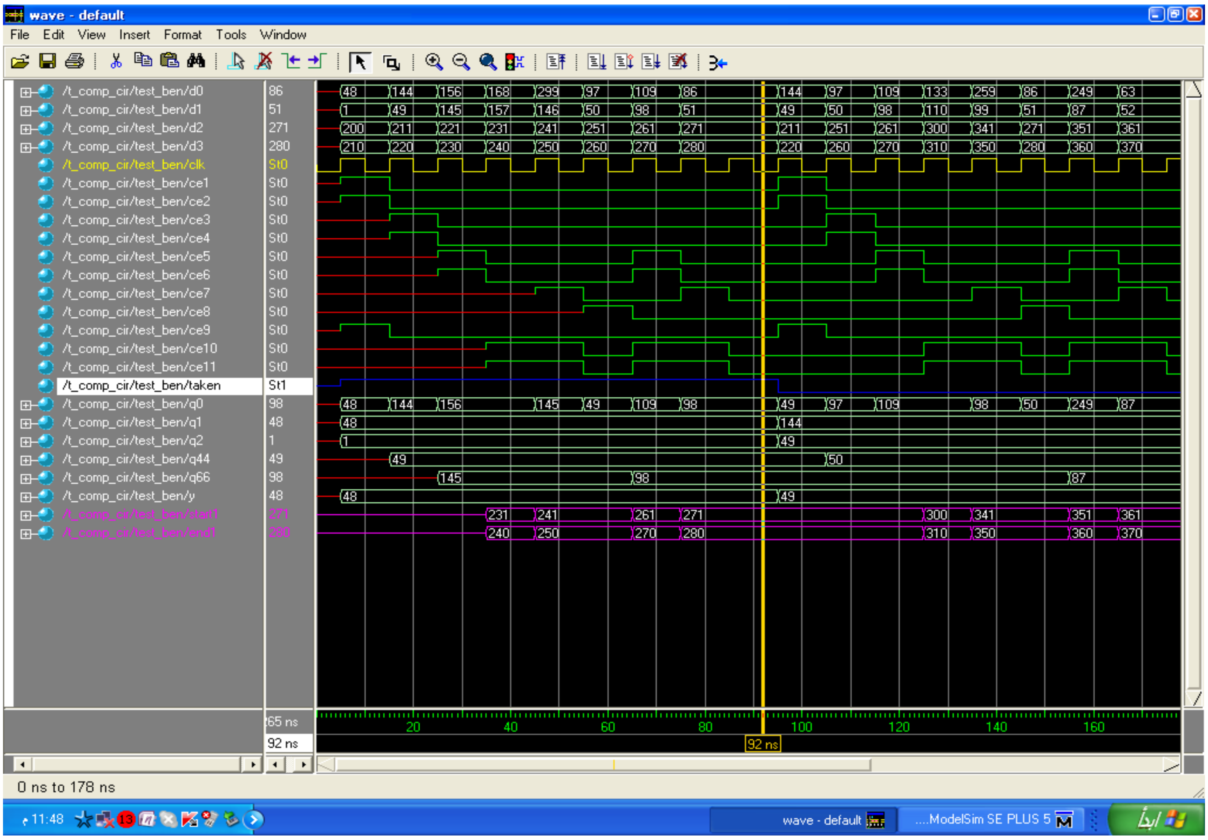
```
/test_taken_nottaken/test_bench/a0       StX
/test_taken_nottaken/test_bench/b0       StX
/test_taken_nottaken/test_bench/c0       StX
/test_taken_nottaken/test_bench/a1       St1
/test_taken_nottaken/test_bench/b1       St0
/test_taken_nottaken/test_bench/c1       St0
/test_taken_nottaken/test_bench/a2       StX
/test_taken_nottaken/test_bench/b2       StX
/test_taken_nottaken/test_bench/c2       StX
/test_taken_nottaken/test_bench/a3       StX
/test_taken_nottaken/test_bench/b3       StX
/test_taken_nottaken/test_bench/c3       StX
/test_taken_nottaken/test_bench/a4       StX
/test_taken_nottaken/test_bench/b4       StX
/test_taken_nottaken/test_bench/c4       StX
/test_taken_nottaken/test_bench/take...  St1
/test_taken_nottaken/test_bench/q1       StX
/test_taken_nottaken/test_bench/q2       StX
/test_taken_nottaken/test_bench/q3       StX
/test_taken_nottaken/test_bench/q4       St0
/test_taken_nottaken/test_bench/q5       St0
/test_taken_nottaken/test_bench/q6       St0
/test_taken_nottaken/test_bench/q7       StX
/test_taken_nottaken/test_bench/q8       StX
/test_taken_nottaken/test_bench/q9       StX
/test_taken_nottaken/test_bench/q10      StX
/test_taken_nottaken/test_bench/q11      StX
/test_taken_nottaken/test_bench/q12      StX
/test_taken_nottaken/test_bench/q13      StX
/test_taken_nottaken/test_bench/q14      StX
/test_taken_nottaken/test_bench/q15      StX
/test_taken_nottaken/test_bench/wr       x
/test_taken_nottaken/test_bench/rd       x
/test_taken_nottaken/test_bench/s0       x
/test_taken_nottaken/test_bench/s1       x
                                         80 ns
                                         20 ns
sim:/test_taken_nottaken/test_bench/a3 @ 26 ns
StX
20 ns
0 ns to 478 ns
```

Figure (10) Simulation for Replacement Circuit



```
/t_comp_cir/test_ben/d0      63
/t_comp_cir/test_ben/d1      52
/t_comp_cir/test_ben/d2      361
/t_comp_cir/test_ben/d3      370
/t_comp_cir/test_ben/clk     St0
/t_comp_cir/test_ben/ce1     St0
/t_comp_cir/test_ben/ce2     St0
/t_comp_cir/test_ben/ce3     St0
/t_comp_cir/test_ben/ce4     St0
/t_comp_cir/test_ben/ce5     St0
/t_comp_cir/test_ben/ce6     St0
/t_comp_cir/test_ben/ce7     St0
/t_comp_cir/test_ben/ce8     St0
/t_comp_cir/test_ben/ce9     St0
/t_comp_cir/test_ben/ce10    St0
/t_comp_cir/test_ben/ce11    St0
/t_comp_cir/test_ben/taken   St0
/t_comp_cir/test_ben/q0      87
/t_comp_cir/test_ben/q1      144
/t_comp_cir/test_ben/q2      49
/t_comp_cir/test_ben/q44     50
/t_comp_cir/test_ben/q66     87
/t_comp_cir/test_ben/y       49
/t_comp_cir/test_ben/start1  361
/t_comp_cir/test_ben/end1    370
                             65 ns
                             80 ns
0 ns to 90 ns
```

The result obtained from execution program for the loader circuit by ModelSim-Altera 6.1g (Quartus II 7.2), is illustrated in figure (11 a,b) below.

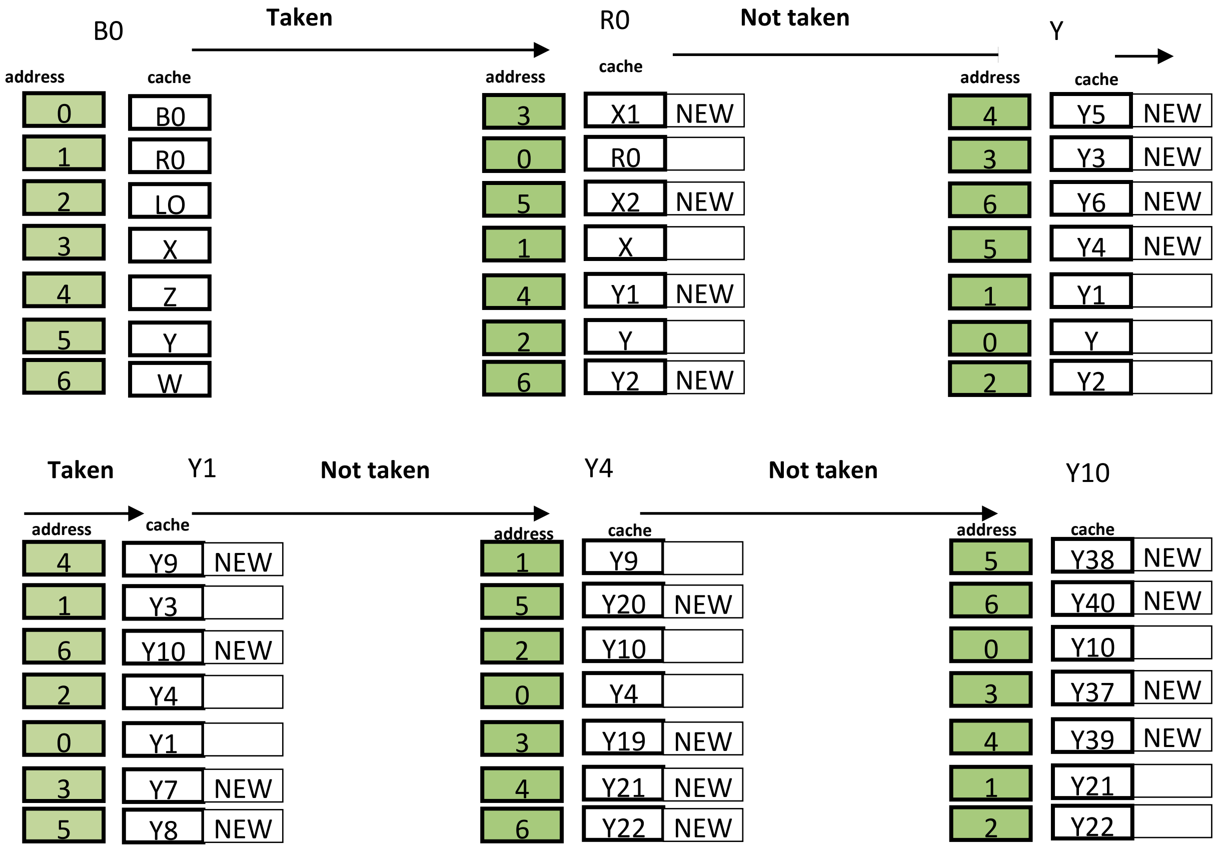


```
/t_comp_cir/test_ben/d0      86
/t_comp_cir/test_ben/d1      51
/t_comp_cir/test_ben/d2      271
/t_comp_cir/test_ben/d3      280
/t_comp_cir/test_ben/clk     St0
/t_comp_cir/test_ben/ce1     St0
/t_comp_cir/test_ben/ce2     St0
/t_comp_cir/test_ben/ce3     St0
/t_comp_cir/test_ben/ce4     St0
/t_comp_cir/test_ben/ce5     St0
/t_comp_cir/test_ben/ce6     St0
/t_comp_cir/test_ben/ce7     St0
/t_comp_cir/test_ben/ce8     St0
/t_comp_cir/test_ben/ce9     St0
/t_comp_cir/test_ben/ce10    St0
/t_comp_cir/test_ben/ce11    St0
/t_comp_cir/test_ben/taken   St1
/t_comp_cir/test_ben/q0      98
/t_comp_cir/test_ben/q1      48
/t_comp_cir/test_ben/q2      1
/t_comp_cir/test_ben/q44     49
/t_comp_cir/test_ben/q66     98
/t_comp_cir/test_ben/y       48
/t_comp_cir/test_ben/start1  271
/t_comp_cir/test_ben/end1    280
                             65 ns
                             92 ns
0 ns to 178 ns
```

Figure (11 a ,b) Simulation for Loader Circuit

The table below (2) shows new address for cache memory after transferring current block from B0 to Y10 (Path 1).

Table (2) Cache Memory Status When Execute Path1



B0

address	cache
0	B0
1	R0
2	LO
3	X
4	Z
5	Y
6	W

R0

address	cache	
3	X1	NEW
0	R0	
5	X2	NEW
1	X	
4	Y1	NEW
2	Y	
6	Y2	NEW

Y

address	cache	
4	Y5	NEW
3	Y3	NEW
6	Y6	NEW
5	Y4	NEW
1	Y1	
0	Y	
2	Y2	

Y1

address	cache	
4	Y9	NEW
1	Y3	
6	Y10	NEW
2	Y4	
0	Y1	
3	Y7	NEW
5	Y8	NEW

Y4

address	cache	
1	Y9	
5	Y20	NEW
2	Y10	
0	Y4	
3	Y19	NEW
4	Y21	NEW
6	Y22	NEW

Y10

address	cache	
5	Y38	NEW
6	Y40	NEW
0	Y10	
3	Y37	NEW
4	Y39	NEW
1	Y21	
2	Y22	

VII. Summary of Results

After the designing odd logic circuit and even logic circuit and testing it and evaluating by ModelSim-Altera 6.1g (Quartus II 7.2), the results obtained from simulation are equal to the predicted results. These results are illustrated in table (2), and when testing the detection circuit and comparing the results obtained from simulation with the predicted results, we have the same results. These results are illustrated in figure (10). After that, we connect these circuits with each other by multiplexer and parallel input serial output (PISO) register, and testing it many times for random paths. These circuits always give the same predicted results. This circuit is connected to cache memory and main memory to get replacement circuit in figure (7). After design all parts in complete circuits, The testing, evaluating and investigating the loader circuit in figure (5) by ModelSim-Altera 6.1g (Quartus II 7.2), and when comparing between the predicted results in the table (2) with the obtained results from simulation in experment1 for loader circuit in figure (10), equal results are obtained. And when testing, evaluating and investigating the replacement circuit in figure (7) by ModelSim-Altera 6.1g (Quartus II 7.2), and when comparing between the predicted results in table (2) with the obtained results from simulation in experment1(path1-red color) for replacement circuit in figure (10), equal results are obtained.

VIII. Conclusion

During the work on this paper the previous techniques used for cache memory mapping and replacement strategies were surveyed to highlight their functionalities, capabilities and limitations for minimizing cache misses. It was found that all those techniques work on the principles of locality: Spatial locality and temporal locality. Those two principles do not put into account the prediction of next block to be loaded into the cache and the next block to be entered and executed. This results in that each technique works good with some programs to minimize cache misses and poor with other programs. This depends on the nature and structure of the program being executed.

By utilizing the behavior of a program represented by its control flow graph, a new technique for cache memory mapping and replacement is proposed. Its corresponding digital circuit was designed and simulated. The circuit consists of two parts that work in parallel. One part performance the task of loading program blocks from the main memory into the cache lines. The other part performance the required replacement. The circuit was simulated using Verilog hardware simulation language and tested using Quartus II 7.2. The results obtained from the simulation verified the proposed idea and it was found that always there exist in the cache successor blocks organized as a binary tree. This eliminating processor wait states and hence a program is executed continuously without cache miss.

REFERENCES:

[1] Ranjith Subramanian, Yannis Smaragdakis, Gabriel H. Loh; " **Adaptive Caches**"; IEEE Computer Society Washington, DC, USA, 2006.

[2] Jim Handy; " **The cache memory book (2nd ed)**" ; Academic Press, Inc. Orlando, FL, USA, 1998.

[3] Steven A. Przybylski; " **Cache and Memory Hierarchy Design"** ; Morgan Kaufmann (30 Jun 1990**).**

[4] Y. Thomas Hou , Jianping Pan; " **Analysis and evaluation of expiration-based hierarchical caching systems** " ; Elsevier Science Publishers B. V. Amsterdam, The Netherlands, The Netherlands, 2004.

[5] Malek M. Kream; " **The Impact Of Program Control Flow on Cache Performance**"; M.Sc. Thesis , Academy of Higher Study; Tripoli-Libya, 2008

[6] Wei Ding, Mahmut Kandemir, Diana Guttman, Adwait Jog, Chita R. Das, Praveen Yedlapalli , **Trading Cache Hit Rate for Memory Performance**, Department of Computer Science and Engineering , The Pennsylvania State University, University Park, Pennsylvania, USA, 2014.

[7] W. Ding, J. Liu, K. Mahmut, and M. J. Irwin, **Reshaping cache misses to improve row-bu_er locality, in multicore systems**," In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques, 2013.

[8] J. Meza, J. Li, and O. Mutlu, **Evaluating row bu_er locality in future non-volatile main memories,**"SAFARI Technical Report, 2012.

[9] H. Yoon, J. Meza, R. Ausavarungnirun, R. Harding, and O. Mutlu, **Row bu_er locality-aware data placement in hybrid memories**," SAFARI Technical Report, 2011.

[10] K. Pingali, D. Nguyen, M. Kulkarni, M. Burtscher, M. A. Hassaan, R. Kaleem, T.-H. Lee, A. Lenharth, R. Manevich, M. M_endez-Lojo, D. Prountzos, and X. Sui, **The tao**

of parallelism in algorithms," In Proceedings of the Conference on Programming ,Language Design and Implementation., 2011.

[11] C. J. Lee, V. Narasiman, E. Ebrahimi, O. Mutlu, and Y. N. Patt, **Dram-aware last-level cache writeback: Reducing write-caused interference in memory ,systems**," HPS Technical Report, 2010.

[12] Y. Kim, D. Han, O. Mutlu, and M. Harchol-balter, \ATLAS: **A scalable and high-performance scheduling algorithm for multiple memory controllers**," In Proceedings of the International Symposium On High Performance Computer Architecture, 2010.

[13] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-balter, **Thread cluster memory scheduling: Exploiting di_erences in memory access behavior**," In Proceedings of the International, Symposium on Microarchitecture, 2010.

[14] G. L. Yuan, A. Bakhoda, and T. M. Aamodt, **Complexity e_ective memory access scheduling for many-core accelerator architectures**," In Proceedings of the International Symposium on Microarchitecture, 2009.

[15] M. Kulkarni, M. Burtscher, R. Inkulu, K. Pingali, and C. Cas_caval, **How much parallelism is there in irregular applications?**" In Proceedings of the ACM ,SIGPLAN symposium on Principles and practice of parallel programming, pp. 3{14, 2009.

[16] **Static Analysis for Fast and Accurate ,Design Space Exploration of Caches**, Yun Liang, Tulika Mitra, Department of Computer Science, National University of Singapore, {liangyun,tulika}@comp.nus.edu.sg, 2008.

[17] Nimrod Megiddo and Dharmendra S. Modha. ARC: A Self-Tuning, Low Overhead Replacement Cache. FAST, 2003.

[18] Sing, Joel. Computer Technology – Cache Memory. http://ironbark.bendigo.latrobe.edu.au/subjects/int11ct/2002/lectures/l17/cache.html

[19] Hamid R. Zarandi, Seyed Ghassem Miremadi; " **Hierarchical Multiple Associative Mapping in Cache Memories** " IEEE Computer Society Washington, DC, USA 2005.

[20] Stephen Hines, David Whalley, Gary Tyson; " **Guaranteeing Hits to Improve the Efficiency of a Small Instruction Cache**" ; IEEE Computer Society Washington, DC, USA ,2007.