Applying 2-way Superscalar Technique to a 32-bit RISC Microprocessor

- **Do Ngoc Quynh**
- **Hau Nguyen Thanh Hoang**

Integrated Circuit Design Research and Education Center

(Manuscript Received on September 25th, 2013, Manuscript Revised October 15th, 2013)

ABSTRACT:

In one-way microprocessor, the program code is executed at the maximum (ideal) rate of one instruction per cycle. In practice, due to the occurrence of branch instruction, this rate is less than 1. Superscalar architecture, when applied to a 32-bit RISC microprocessor, enables the handling of two instructions in a single machine cycle. To further increase the processing speed, the out-of-order execution is also applied to process an instruction that its operands are ready. As a result, the microprocessor which can complete two instructions per cycle is obtained.

***Keywords**: Microprocessor, advanced computer, RISC computer, superscalar.*

INTRODUCTION

In the present paper, we show how to apply superscalar technique to a 32-bit RISC microprocessor (MCU) to enable a simultaneous processing of multiple instructions. In our design, the maximum number of instructions is 2. The out-of-order execution is also incorporated for more efficient use of hardware resources. An instruction is executed immediately when its operands are ready. A branch prediction block with high accuracy is indispensable to reduce waste in the case when branch instructions exist.

RELATED WORKS

To improve the performance of a single microprocessor, designers can increase the pipeline stages to raise the operating frequency of the system. With this method, the maximum upper limit processing speed of the system is still 1 instruction per 1 cycle. The complexity of the design is manifold due to this increase of stages accordingly many times; and high frequency can be more costly in term of power consumption. This is a disadvantage when the MCU is used in a system that requires less power consumption.

In superscalar technique, it is not necessary to increase the operating frequency, but the capability of processing multiple instructions in one cycle is permitted. The difficulty of this technique is to make sure the relationship between the result of the previous instruction and the operands of the current instruction (data dependence). Thus the design of CPUs with multiple instruction issues has recently become a trend of the microprocessor industry.

SUPERSCALAR HARDWARE

The superscalar techniques require more resources to handle multiple instructions in parallel at the same time. These circuits are added at all pipeline stages.

Fetch and Decode stages

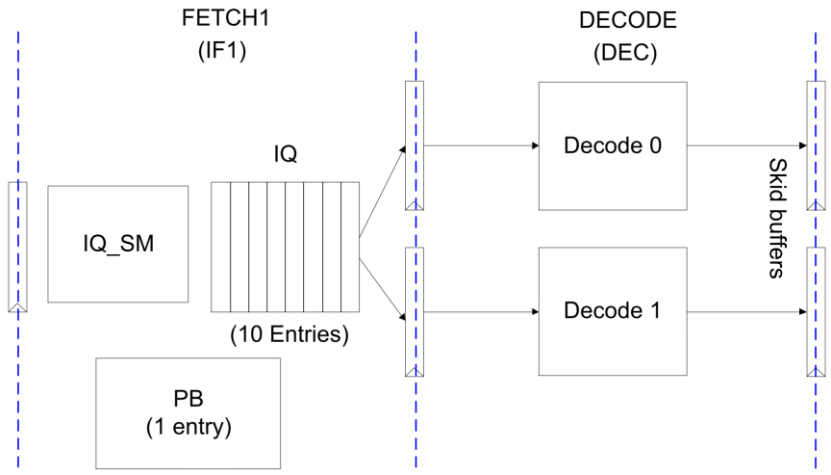

Fig.1. 2-way parallel decode block

In the Fetch stage, the Instruction Queue (IQ) is able to send 2 instructions to Decode stage. Two read pointers are used to choose Instruction_0 and Instruction_1 from IQ.

The main functions of IF1:

Checking Instruction Cache (IC) Tag for IC_hit

Checking Prefetch Buffer (PB) Address for PB_hit

Pushing instruction into IQ and queue there

Sending to fetch request to lower level memory if the IC and the PB are both missed

Checking Branch Target Buffer (BTB) data, Local History, Global History for predicting branch

Controlling the operation of GHR

Controlling the operation of RS

Pushing branch predicted information into Prediction Queue (PQ)

Holding BTB_PLRU value in BTB_lookup buffer

The continualness of providing two instructions is a key issue when designing Fetch stage. The pipeline process will reach the highest efficiency while input instruction is always ready. The state machine IQ_SM (as shown in Fig. 1) generates an instruction request to prefetch 4 instructions of a L1 cache line (the address bits from 31 to 4 are the same). First, there is a comparison circuit to check the address request and the contents of Array IC; the instruction is read from IC cache and brought to IQ in the hit case. In the miss case, the instruction must be read from lower memory (UL2 cache). These instructions are queued in IQ and waited for sending to Decode stage. IQ can accommodate up to 10 instructions; this value is sufficient for optimal IQ with this 2-way superscalar design.

The Decode stage identifies an instruction and generates the corresponding control signals. Two instructions from IF1 are processed in parallel and flopped at the output of the stage. It will be the income of the next stage. In some cases, the system must handle a task too long. It will pause receiving a new instruction to be able to complete the old one. At this point, the Decode stage will activate the 'stall' signal to prevent new instruction from Fetch stage. A Skid buffer will save the control signals in this case to reuse as

soon as the system exits 'stall' state. After that, the control signals are selected from the Skid Buffer instead of the outgoing values of the 2 decode units.

Dispatch stage

An important block in the Dispatch stage is ROB (in Fig. 2). An instruction is processed in the order in Fetch, Decode, Issue stages and out-of-order in the following stages. The ROB will rearrange the order of instructions and retire them in the program flow. An instruction is not actually completed until ROB retires it and writes to the register file. When the program branches to a new location, the following instructions after the branch will be ignored all, and a new instruction at the new location is fetched and written to ROB. So, ROB is to control the flow of microprocessor operation.

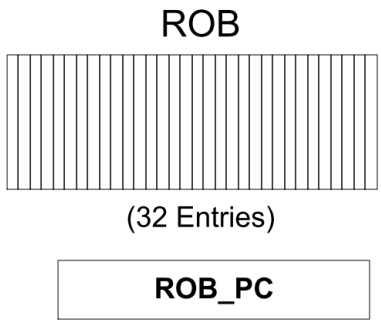

Fig.2. ROB

The main functions of Dispatch (DP):

Containing Register Files (RF)

Containing PC at Dispatch Stage (Dispatch_PC)

Containing Reorder Buffer (ROB). The CPU pipeline executes instructions out-of-order, ROB reorders the executed instructions and keeps them retiring in-order.

Containing PC at ROB (ROB_PC)

Dispatching instructions to the next stage (up to 2 instructions/cycle).

Essential information of Dispatch PC needs to calculate and save in this stage because the PC of instructions is not in order after this stage. There are 2 Dispatch PC0 and PC1 values, they correspond to instruction 0 and instruction 1. In normal operation, the value of Dispatch_PC increases 8 for each cycle. For branch instructions, the PC is the target of the branch instruction.

Issue stage

Each RS holds control signals of each type of instruction. When instruction operands are ready, the instruction will be issued to the next stage. The ALU0RS and ALU1RS are issued out-of-order. And the LSRS, BRRS, and MULTRS are issued in order. These blocks must be done in order to ensure the order of instructions.

The main functions of Issue (ISU):

There are 5 Reservation Stations (RS): 2 ALU Reservation Stations (ALU0RS, ALU1RS), Load/Store Reservation Station (LSRS), Branch Reservation Station (BRRS), Multiply Reservation Station (MULTRS)

Each RS holds control signals of each type of instruction

When instruction operands are ready, the instruction will be issued to the next stage.

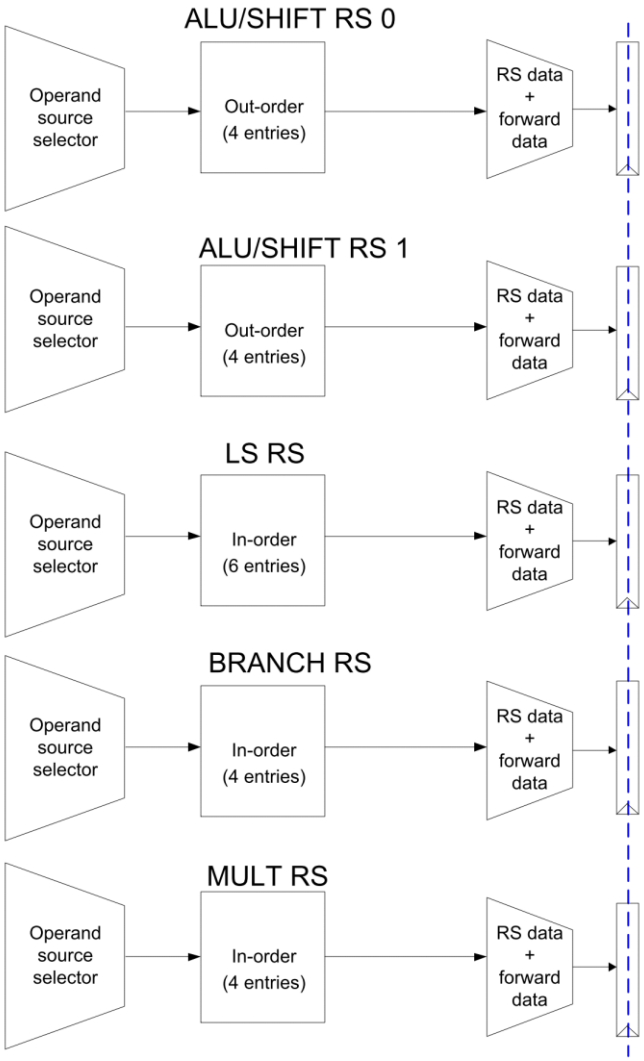

Fig.3. Issue stage

Normally, data is written into the RSs from DP. In the case the data is the result of preceding instructions, the result will be forwarded into the RSs from the stage which firstly has valid result.

Execute stage

All ALU instructions are executed in one clock cycle. The following instructions are executed in ALU0/ALU1: ADDx (addition), SUBx (subtraction), CLTx (comparison), ANDx, ORx, XORx, NOR, SLLx (left shift), SRx (right shift). The isu_alu0_control or isu_alu1_control signals select which operator will be executed. Two input operands of ALUx have the 32-bit width. For the signed addition or subtraction, if the result of ALUx is incorrect to return, the alu0_overflow or alu1_overflow will assign to 1.

In the Execute stage, the load/store instruction is only calculated the load/store address and byte valid bits. The isu_ls_opa is a base address, and isu_ls_offset is an offset address of a load/store request. The offset address is the 16-bit width, so it is signed-extend to become 32-bit width. The offset address is added to the base address including a sign bit. It can be shown in Fig. 4.

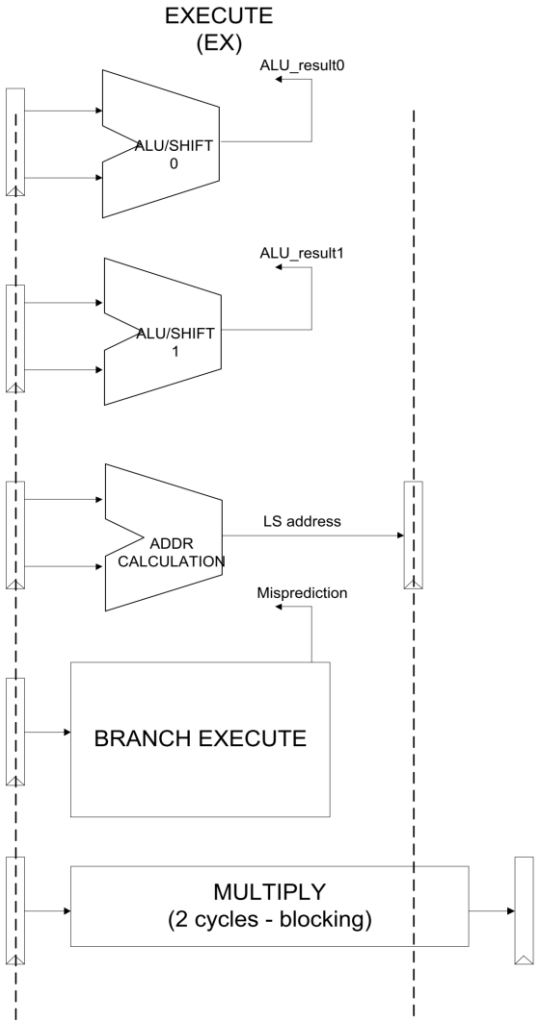

Fig.4. Execute stage

A multiplication executes in 2 clock cycles. The product is stored in the pair of registers ***mult_result_hi*** and ***mult_result_lo***. In order to read the contents of the high / low product, the user must use MFUP/MFLP instructions. The isu_mult_unsigned signal indicates that the multiplication is signed or unsigned. When it is 1, this is an unsigned multiplication and vice versa.

A move-from result can be selected from 2 sources: high/low product of multiplication or CP0 register contents. It depends on isu_mfmult_val signal. When it is 1, ex_mf_result is the contents of the high / low product. Otherwise, ex_mf_result is a value of CP0 register.

PERFORMANCE IMPROVEMENT

Superscalar technique

At one point, the pipeline stages can handle up to 2 instructions at the same time. Especially Execute stage can handle up to 5 instructions simultaneously because it has 5 independent execution blocks.

An instruction will be distributed to the Issue blocks corresponding to its instruction type after decoding is completed. When the instruction

goes to Issue stage, it also is written to the ROB in accordance with the program order. And in the final stage of the pipeline, the instructions are sequentially retired in the program order. The function of ROB is to re-arrange the order of instructions and writes the result of execution to the system registers. The rearrangement is required because the instructions in Issue and Execute stages are performed out-of-order.

After the instruction has been dispatched, it will be in turn placed in the Reservation Station (RS) to wait for executing. The RS in Issue stage will check all operand sources continuously. As soon as the result of the operand appears, the ready bits of the operand are updated to 1. It means that the content of the operand is valid. When all operands of instruction are ready, operator will be executed in the Execute stage and the calculation result is saved into the Reorder buffer (ROB).

If the ROB contains the branch instruction that the branch prediction target is wrong, then the following instructions of a branch that will be ignored because the flow of the program was no longer true. The same manner is in the case of an exception.

Branch Predictor

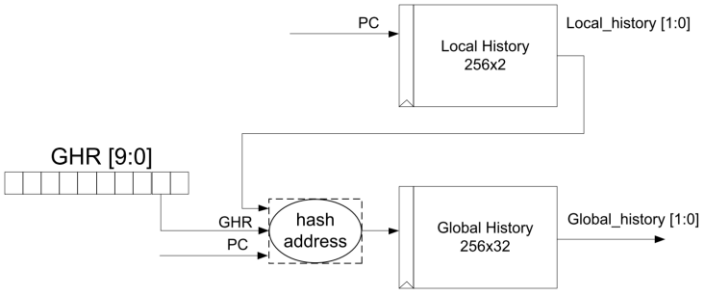

Fig. 5. Branch Predictor

Two techniques will be used in Branch prediction block: Local history and Global history.

- Local history based on the address of the PC of jump and branch instructions to predict the branch target.

- GHR saves branch results of 10 previous branch instructions.

- Global history is based on the results of the previous branch instructions to form the bit index to read the contents of the Global History and get branch prediction result.

The implement of branch prediction is applied in this 32-bit RISC microprocessor that combined Local and Global predictors (as shown in Fig. 5). In particular, the address of the branch instruction PC will be brought to the Local History SRAM to read the 2-bit saturating counter. The 2-bit saturating counter, the PC address bits of branch instructions and GHR registers are mixed to generate the index bits to read 2-bit Counter of Global History. Global values are read from the user to predict the direction of the branch instruction. The bit 1 is the direction of the branch and bit 0 is the strength of the prediction. The index address of global history is compiled from various sources that contain more information about jump/branch prediction. All information of branch instruction and the history of branch flow are stored in SRAMs (in Fig. 6). The target addresses of branch instructions are in Branch Predictor – Entries (in Fig. 7).

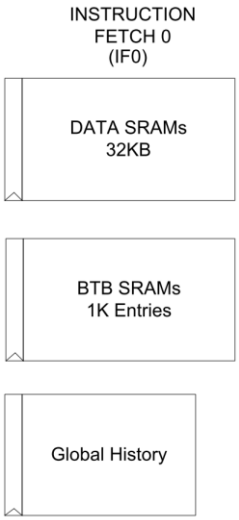

Fig. 6. Branch predictor - SRAM (IF0 stage)

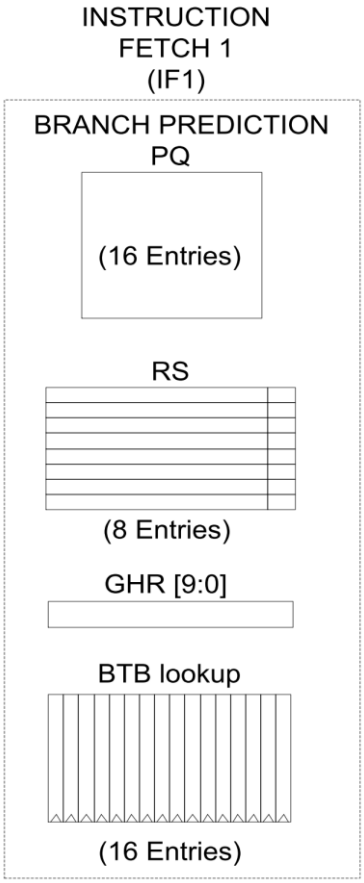

Fig. 7. Branch predictor - Entries (IF1 stage)

Branch Prediction flow:

Data from BTB Arrays, Local History, and Global History (Fig. 6) will be checked to detect branch instructions, and predict whether the branch is taken. The necessary information will be written to PQ. The information in the PQ will be useful for the next stage.

The status of current predicted branch (taken/not taken) is shifted to GHR[9:0] (LSB to MSB). The value of GHR is then used to access Global History for later prediction.

Return Stack (ReS): there are 3 blocks of ReSs: (1) ReS at Fetch (Fetch_ReS), (2) ReS at Execute (Execute_ReS), and (3) ReS when retiring (arch_ReS). When CALL, return PC will be written to ReS at ReS_pointer and the pointer is increased by 1. This pointer is decreased by 1 when return PC will be read out (RET instruction).

BTB_lookup store the BTB_PLRU value of the BTB. This value will be used for replacing the BTB.

Prediction Queue (PQ): hold all necessary information for branch prediction. Each PQ entry is setup at IF1. PQ entries can be read and updated at other stages such as DP, EX, ROB (when retiring) .

Memory hierarchy

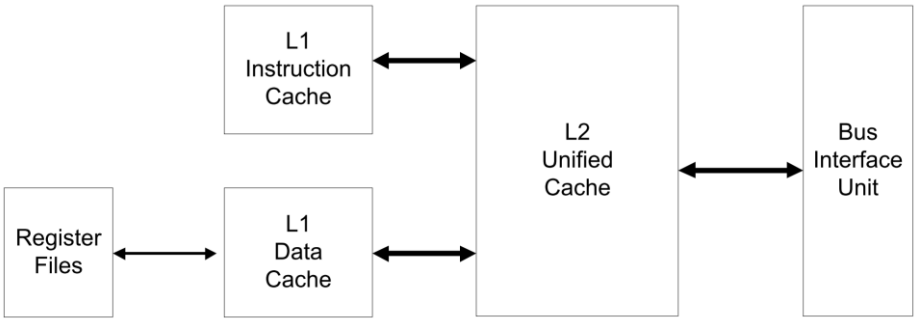

Fig. 8. System memory hierarchy

The memory hierarchy of this microprocessor is designed to operate efficiently with 2 levels. In the Fig. 8, at the level 1, it consists of L1 instruction cache (IL1) and L1 data cache (DL1) separately, the Harvard memory model supports to access instruction and data simultaneously. At the level 2, there is only an L2 Unified cache (UL2), it is shared between Instruction and Data. The Pseudo Least Recent Use (PLRU) replacement policy is applied to achieve a high hit ratio for the cache.

The IL1 and DL1 sizes are small 32KB each and simple circuit implementation for fast operation (two clock cycles). The hit rate of L1 caches only reaches about 85%. UL2 cache is larger than the L1 cache 8 times, 256KB. It has a slower return data rate (s cycles) and higher hit rate 95%. With a high hit rate, the performance of the system will increase significantly, due to missing request in UL2 cache will be taken directly from the system bus with long latency (52 cycles for the fastest case: 1 request cycle + 3 cycles for the first address + 16 data x 3 cycles for each (the clock rate ratio of CPU clock / bus clock = 1/3).

The DL1 cache can operate as a non-blocking data cache. The PB block in DL1 cache has 2 entries and can handle up to 2 miss loads. This is really a valuable feature when a miss load is followed by another load. The second load will start running before the end of the first load. The second load can be continued to process and

return read data if it hits in DL1, and it is written to PB in the miss case. For third load, it must be re-sent in the miss case. Because, at that time, the PB is full and can't receive more miss loads.

The UL2 cache supports Store Gathering Buffer block. This one collects all store requests with the same L2 cache line. An L2 cache line contains 64 bytes, so 2 store addresses are considered the same cache line if address bits from 31 to 6 are equal. An L2 cache line in SGB is written AHB bus when all byte of cache line is collected. This helps to optimize the access to the system bus, and bus bandwidth is used more efficiently. In some special cases, the entry of the SGB will be evicted to the bus before it collected enough one L2 cache line, but these cases will not be presented in detail.

Out-of-Order Execution

The instructions are processed with the correct program order (in order) in the IF0, IF1, and DP stages and out-of-order in the ISU, EX, M0 and M1. But, at Commit stage (the last stage), the instructions are rearranged to write the result of calculation to the register file with the exact program flow.

Most instructions go through ISU and EX stage out-of-order. At the ISU stage, as shown in Fig. 3, the instructions will be classified and placed in the reservation stations (RS) in preparation for the execution of the instruction. These stations check all operands of instruction and send the instruction having ready operands to EX stage. Two ALU/SHIFT RSs issue instruction out-of-order, the remaining RSs: LOAD/STORE, BRANCH, MULT is in order. An instruction consumes a one cycle to give the operator result in EX stage, except MULT (multiplier) instruction which must take two cycles to complete. Then, the results of execution will be saved in ROB to wait for retiring.

CONCLUSION

The Fetch stage is made to achieve high performance. When instructions are already in the cache, the 2 consecutive instructions are put to the Decode in a cycle. The 2-way superscalar fulfills 2 instructions in parallel. The number of instructions, which is executed simultaneously in the pipeline, is 2. An instruction is executed out-of-order, and it will start whenever all operands are ready. This helps an efficient use of hardware resources.

The branch prediction block eliminates wasted cycles to change the program flow in the correct prediction cases. When the prediction is wrong, waste cycles are required. Since the miss prediction rate is low, the performance of microprocessor increases significantly.

The two-level cache hides the long latency due to waiting for response data from the main memory. The fast L1 caches can return the load data after 2 cycles. The high hit rate of UL2 cache is to minimize the number of times when the system reads data directly from the main memory. The bottleneck issue between the processor and the memory is reduced and the load/store instructions will be processed at the speed of the microprocessor.

Áp dụng kỹ thuật Superscalar vào một vi xử lý RISC 32 bit

- **Đỗ Ngọc Quỳnh**
- **Hầu Nguyên Thanh Hoàng**

Trung tâm đào tạo nghiên cứu thiết kế vi mạch ICDREC

TÓM TẮT:

Trong vi xử lý một luồng mã lệnh, mã lệnh chương trình được thực thi với khả năng lý tưởng là một mã lệnh trên một chu kỳ. Nhưng với các ứng dụng thực tế, tỷ lệ này sẽ giảm thấp hơn 1 bởi vì sự xuất hiện của các lệnh rẽ nhánh. Kiến trúc superscalar khi được sử dụng trong một vi xử lý RISC 32 bít sẽ cho phép xử lý hai mã lệnh trong một chu kỳ máy. Để tăng thêm tốc độ xử lý, phương thức thực hiện mã lệnh không theo thứ tự cũng được sử dụng. Các mã lệnh sẽ ngay lập tức được thực thi khi các toán hạng ngõ vào sẵn sang. Kết quả nghiên cứu đã đạt được một vi xử lý có thể hoàn tất hai mã lệnh trong một chu kỳ.

REFERENCES

[1]. J. L. Hennessy and D. A. Patterson, *Computer Architecture - A Quantitative Approach*, 4th ed.: Morgan Kaufmann, (2007).

[2]. J. P. Shen and M. H. Lipasti, *Modern Processor Design - Fundamental of Superscalar Processor*.: McGraw-Hill, (2005).

[3]. M. C. Chang, Y. W. Chau, Branch prediction using both global and local branch history information, *IEEE Proc.-Comput. Digit. Tech.*, vol. 149, Mar. (2002).

[4]. K. Skadron, M. Martonosi, D. W. Clark, *Speculative Updates of Local and Global Branch History - A Quantitative Analysis.*

[5]. H. Ghasemzadeh, S. Mazrouee, and M. R. Kakoee, Modified Pseudo LRU Replacement Algorithm, *Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS'06)*, (2006).

[6]. K. Aasaraai and A. Moshovos, An Efficient Non-Blocking Data Cache for Soft Processors, *International Conference on Reconfigurable Computing and FPGAs (ReConFig), 2010*, 19-24, December 2010.

[7]. D. Kroft, Lockup-free instruction fetch/prefetch cache organization, *Proceedings of the 8th annual symposium on Computer Architecture (ISCA '81)*, pp. 81-87, 1981.