Journal of Alasmarya University: Basic and Applied Sciences
Vol. 6, No. 5, December 2021, Special Issue for Forth Conference on Engineering Science and Technology (CEST-2021)
مجلة الجامعة الأسمرية: العلوم الأساسية والتطبيقية
المجلد 6 ،العدد 5، ديسمبر 2021، عدد خاص بالمؤتمر الرابع للعلوم الهندسية والتقنية (CEST-2021)

Combining the Forwarding with Delay Slots Operations to Avoid the Branch Misprediction Penalty in Superscalar Processors

Ali Hudoud

Department of Computer, Faculty of Engineering, Azzytuna University, Tarhuna, Libya

Keywords:
Branch Misprediction Penalty.
Clock Cycle Per Instruction (CPI).
Control hazard Dynamic Branch Prediction
Million Instruction Per Second (MIPS).

ABSTRACT

Using pipeline system in modern possessors has contributed significantly to the development of processors performance by increasing its speed faster than before, where CPI approaches to 1, however, this technique accompanied many problems, one of them called Branch Misprediction penalty due to control hazard, most of them occurs when implementing dynamic branch prediction. For every five jump commands of a program there is jump command, which causes interruption of the execution of orders through pipeline systems. There are many proposed previews studies, such as Dynamic Branch Prediction and Control Speculation, NTB Branch Predictor: Dynamic Branch Predictor for High-Performance Embedded Processors. This paper presents a new mechanism to combine forwarding and delay slots together to avoid a Branch Misprediction Penalty in Superscalar Processors, it's got better results.

Author Email: ali.amary81@yahoo.com

1 INTRODUCTION

In order to make the system acts as a sustainable pipeline, it must be fetched with new instruction in every new clock cycle. whereas within these instructions there is a branch instruction must be knowing its direction in the second stage" ID" [1,2,3,4,5]. However, until the branch is resolved, the problem occurred due to unknowing from where the instruction will be fetched. This delay of calculating the address of the next instruction is known as a control hazard [6,7,8,9]. It occurs due to the dependency in control operations. The instructions that depend on the branch cannot be moved in front of this branch, that is, they cannot come in the line to execute orders until the direction of the branch can be controlled. Nor can these non-branch-dependent instructions be moved for the same reason. In this research, work has been done to limiting the problems of control hazard, the most important of which is known as branch misperception. This paper is organized as follows: Section II, presented related work includes the most important methods of solutions, previous treatments and some recent scientific papers were mentioned. Section III, presented research method, which will explain one of the most important problems and then the most important elements used in the mechanism of this research. Section IV, presented proposal methodology, during

ISSN: 2706-9524 (Print) ISSN: 2706-9532 (Online)

which the proposed mechanism was implemented is to combining both forwarding and delay slots operations in order to obtain the best result. Section V, presented results and discussion. Finally, section VI, presented conclusion.

2 RELATED WORK

A lot of the researches has been done to find solutions to problem branch misprediction penalty in superscalar processors and all pipelined systems, focusing on improvements on the physical side, such as research entitled "Dynamic Branch Prediction and Control Speculation"[2]. In the last two decades in general has focused on improving prediction through the most important of them as prediction scheme, aliasing in gglobal ppredictors, hhybrid ppredictors, third-level of aadaptively. One of these researches entitled " combing static and dynamic branch prediction to reduce destructive aliasing", the prediction rate was 75% for a simple branch prediction and 14% for a very aggressive hybrid predictor for a certain program [10]. In other, direct relevant research in recent years such as comparative study on behaviour based dynamic branch prediction using machine learning, January 2019, it concludes that the use of machine learning provides competitive results. However, the use of machine learning doesn't help predict all conditional branch behaviours, and looking at the results of this research, most of them work to improve the accuracy of the dynamic prediction by developing the physical part, however, these improvements remain undefinite results to avoid all predictive errors. Other studies have also found software solutions that have the advantage of being less expensive than research requiring physical development, most important of these software-developed methods is what's known as forwarding operation and delay slots operation, They were used separately and were effective, but only used to avoid data hazard, it has achieved excellent results by taking advantage of the time it takes for a dynamic forecaster to know its purpose and execute orders that have nothing to do with jumping. Based on the results of previous studies, this research has been conducted, both forwarding operation and delay slots operation has been combined to get better result, that's by avoiding control hazard.

3 RESEARCH METHOD

This section was divided into three parts as follows.

3.1 PIPELINE HAZARDS AND STALLING OPERATION

The flow of data values takes place between the instructions that accomplish the result and the instructions require the result, since it is the instructions that determine the means from which the required data will be brought through a dynamic guidance mechanism for branch [3]. In the MIPS system and all of systems that operate as a pipeline mechanism, three cycles will be lost in the time after the implementation phase, thus the direction of the branch can be accurately known as this example described in figure (1). The branch penalty is analysed as a

function of a relative number of branch instructions executed and probability that a branch is taken [8]. The penalty for the direction phase can be reduced in the implementation phase if the hardware is developed.

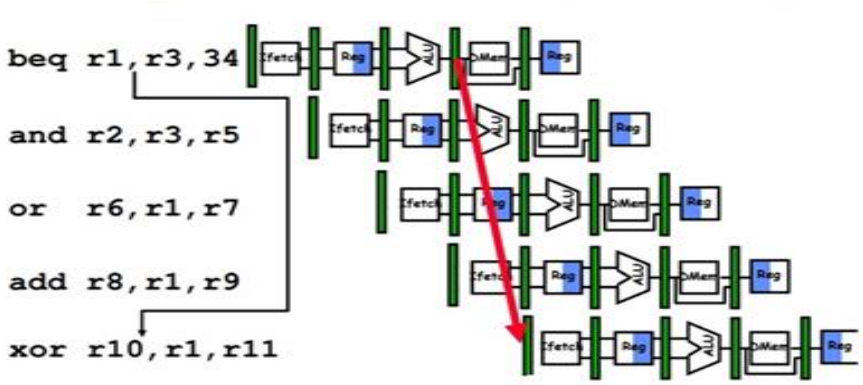

Figure 1. Control hazard in branches three stage (3cycles).

Memory protection violation may be accrued at executing instruction xor before **beq** result. The jump to the L1 address after checking the zero address is made by the branch instruction, and another exception will occur if the command continues to be executed. This needs two operations early, first one is “computing the branch target address" and "evaluating the branch decision earlier than usual".

Change the location of the branch adder to Instruction Decoding stage (ID) instead of Execution stage (EX), because the Program Counter (PC) value in the “IF-ID” pipeline register already exist; address computation of the branch target for all instructions will be accomplished by comparing the two registers in ID by different ways such as ORing to see if equal, to use it when required. To support the improvement of the pipeline performance, forwarding mechanism and hazard detection hardware has been added [4].

To get rid of control hazard problems, there’re two ways:

1. Predicting branch behavior (taken or not taken).
   a. Recruitment of compiler for Static prediction.
   b. Adding hardware solution for dynamic prediction.
2. Stall should be performed until the direction of the branch is known.

To implement the first way, the next steps must be followed [4,11]:

- not taken

if not taken branch is assumed: the condition evaluated as false, where the registers read a next instruction and known if it is "branch or no" during 2rd stage "ID" Instruction Decoding stage of the next instruction, and clearing the pipeline if the branch is assessed as true.

- taken

If taken branch: the condition evaluated as true, where the registers read a next instruction and known if it's "branch or no" during 1rd stage "IF" Instruction Fetch stage of the next instruction, just worked with especial processors. i.e., no delay, however, it cannot work with MIPS systems (5 stages) [12].

- Delay branch.

For more operant solution.

A processing can be performed to a conditional branch instructions as in the following steps:

1- Recognition of conditional branch.
2- Determine whether branch (taken - not taken)
3- Determine the branch target.
4- If branch taken will be redirected instruction fetch.

3.2 FORWARDING OPERATION

Data forwarding is employed as an optimizer in pipelined processors in order to limit a deficits that has been caused by pipeline stalls which would potentially arise because of data hazard as the current progression is timely linked with results of previous unfinished process [7]. The result is forwarded directly to Arithmetic Logic Unit directly, so that, the results can be used, and there is no need to wait for the results from the register file when it is restored. Forwarding mechanism operation as follows [9, 12, and 13]:

a) Control starts in IF/ID buffer.
b) The EX, MEM, and WB stages controlled by ID/EX buffer, while executing control for the EX-stage.
c) The MEM and WB stages controlled by *EX/MEM buffer*, while executing control for the MEM stage.
d) The WB stage controlled by MEM/WB buffer.

Example: sequence of instructions in pipeline before forwarding operation.

Table 1. Before forwarding

		1	2	3	4	5	6	7
ADD	R1, R3, R2	IF	$ID_{\_add}$	EX	MEM	WB		
SUB	R5, R4, R1		IF	$ID_{\_sub}$	EX	MEM	WB	
AND	R8, R1, R7			IF	$ID_{\_and}$	EX	MEM	WB

The main idea of the forwarding in this example is that SUB does not need to ADD result until the result is actually obtained, while it has to be actual when the subtraction requires this result. i.e. if the result can be passed from (EX/MEM register) for ADD to (ALU input latch) for SUB then, no need to use a stall, and method it works, as this example turns out, which is as follows [4]:

- The result of EX/MEM always fed back from ALU input latches.
- If the hardware of forwarding discovers that the previous ALU value matches the current, the forwarded result to ALU input instead of the value from the register file.

There are tree paths to the new input, and three extra inputs to perform this operation on each ALU multiplexer.

The path corresponds to forwarding of:
(a) End of EX depends on the ALU output.
(b) End of memory depends on ALU output.

(c) End of MEM depends on the memory output.

The following example will be executed correctly using only stalls, and without forwarding.

Table 2. without forwarding

		1	2	3	4	5	6	7	8	9
ADD	R1, R3, R2	IF	$ID_{\_add}$	EX	MEM	WB				
SUB	R5, R4, R1		IF	stall	Stall	$ID_{\_sub}$	EX	MEM	WB	
AND	R8, R1, R7			stall	Stall	IF	$ID_{\_and}$	EX	MEM	WB

As in the above example, the results are required to be forwarded not just from the previous instruction immediately, but also instructions could have started three cycles earlier. Since the forwarding can be as follows "from MEM/WB latch to ALU input ", so it can execute the code sequence without stalls:

- First forwarding is for **R1** value from $\mathbf{EX_{\_add}}$ to $\mathbf{EX_{\_sub}}$.
- Second forwarding is for R**1** value from $\mathbf{MEM_{\_add}}$ to $\mathbf{EX_{\_and}}$

Table 3. forwarding without stalls

		12	3	4	5	6	7	
ADD	R1, R3, R2	IF	$ID_{\_add}$	EX	MEM	WB		
SUB	R5, R4, R1		IF	ID	$EX_{\_sub}$	MEM	WB	
AND	R8, R1, R7			IF	ID	$EX_{\_and}$	MEM	WB

The results can be forwarded from the output of one unit to the input of other units instead of the output of one unit to the input of the same unit only.

3.3 BRANCH DELAY SLOTS

Placing branch instructions in the instruction set is one way to reduce the number of idle cycles associated with a branch with a case taken in a pipeline processor, so that the next instructions are in case they follow up immediately after the branch instructions, regardless of whether the branch result has been taken or not. [4,13,14,15].

To put instructions in so-called the delay slots, there are three ways [4], as explained in the following:

(a) When branch is not taken.
(b) Put it before branch instruction in ordering implantation.
(c) From the target address, if a branch is taken.

If the conditional branch instructions are taken, they do not execute the instructions in the delay period, and MIPS processors execute the branch or jump instruction and the delay time instruction as an indivisible unit. [12]. The jump or branch instructions are not implemented in the event of an exception as a result of the execution of the delay slots instructions, and it appears that the exception was due to jumping or branch instructions. The compiler places no operations in the delay slot when it does not find appropriate instructions. In the pipeline processor system, the compiler sets a specific number of delays for each type of jump. The number of delay slots remains unspecified and as few as possible for each type of jump. The compatible processor uses a number compatible with branch delays to exploit the difference in predictability of different types of jumps and branches. Many types of jumps have switched their target addresses with varying numbers of delay slots. Intelligent translators/compilers have the potential to generate more efficient code than processors that have a constant number of delays for all types of jumps, leading to better processor performance. [4]. If the interpreter does not find the appropriate instructions to fill the delay period. It's going to fill it no_op [6].

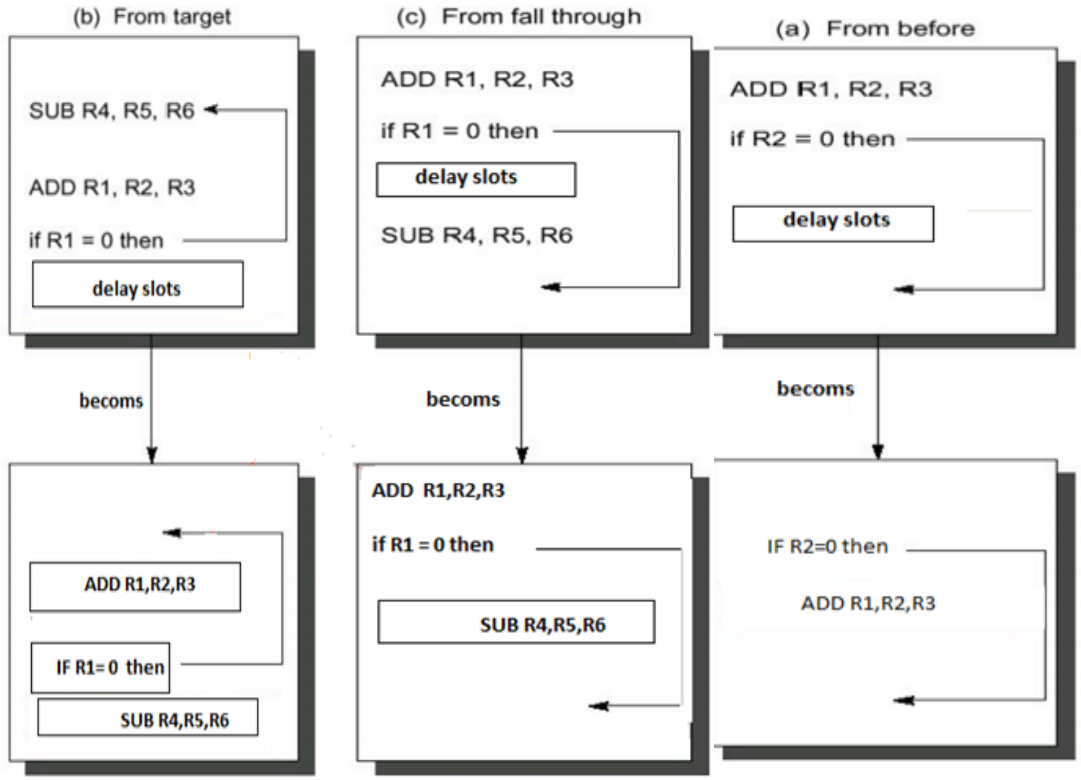

Figure 2. Branch delay slots cases

- In the case (a), when a branch is not taken, the Branch Always helpful when possible.

But it's not possible if ADD R2, R1, R3.

- In the case (b) from the target: helps whenever the branch is taken.

- In case (c) from fall through: helps whenever branch is not taken.

4 PROPOSAL METHODOLOGY

At a conditional branch in a taken case, the dynamic branch prediction mechanism acts as main mechanism that is used to determine the target to which the branch prediction will be launched. Branch Misprediction penalty maybe be accrue. This problem is costing us more time which negatively affects the performance of the processor. Previously, a lot of researches introduced different methods to limiting this problem, most of them devoted to improve the prediction value (taken or not taken), through improving and developing branch predictors that employed to improve the flow in the instruction pipeline such as

[16,17,18,19,20]. In this research, introduced a new effective mechanism and described as bellow.

4.1 FRAMEWORK

This mechanism plan relies mainly on integrating both (forwarding and delay slots operations) as explained in part (3.2) and (3.3) is performed to achieve an outcome in improving the accuracy of the dynamic branch prediction, and thus avoiding branch misprediction penalty. The general form for this mechanism illustrated in figure (3).

Predicting whether the majority of branches are taken or not, it will be very useful as proved in the most pervious solutions. The developed compiler does this software mechanism, using intelligent compiler, they can also be done at the time of execution. Emphasis has been placed on high-use barrel techniques (being closer to the translator). We can say that, to avoid the Branch Misprediction Penalty, we need the following compound mechanism to achieve the best possible results:

- Forwarding whenever possible to handle data hazards without incurring any delays.
- If it's not sufficient, the compiler automatically performs a supporting operation by

  employing branch delay slots.
- You may handle jumps either in the same way as branches.

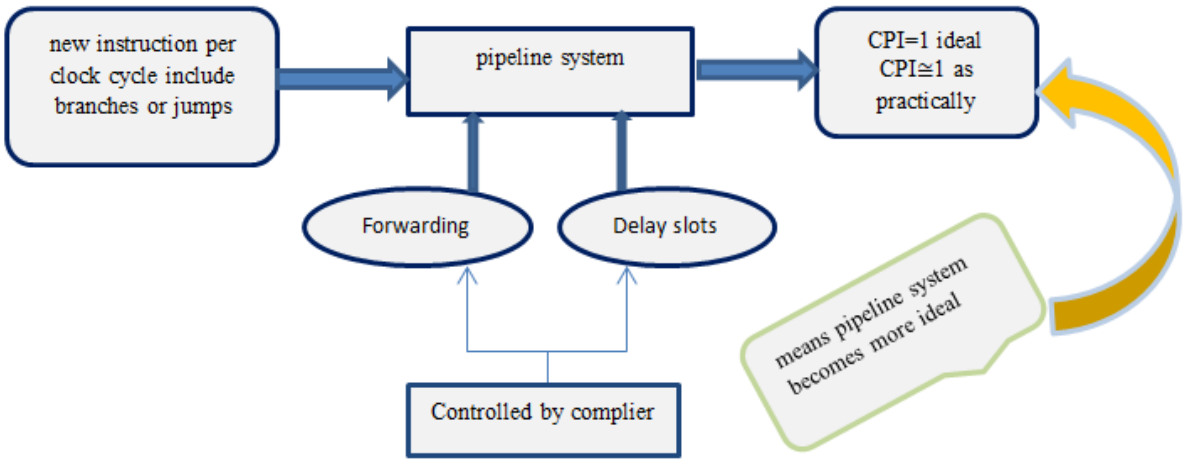

Figure (3) overall format of the research plan

4.2 PERFORMANCE MEASURES

In this research, the following factors has been adopted as criteria for demonstrating the importance of this mechanism in improving computer performance by improving these factors:

- Clock Cycle Per Instruction (CPI), the closer the CPI value to 1, the better the pipeline system works and so we get the higher performance speed.
- Number of stalls in the following cases:
  * Branch Taken Stalls
  * Branch Misprediction Stalls.

Its possibility to add another standard, which is structural hazard.

5 RESULTS AND DISCUSSION

In the following experiment a program was used containing 140-byte code size.

Where:

EB = Enable BTB.

EF = Enable Forwarding.

EDS = Enable Delay Slots.

SS = Structural Stalls

BTS = Branch Taken Stalls

BMS = Branch Misprediction Stalls

Table 1. Enable BTB and forwarding factors

Config	Exec	Stalls
EB	428 cycles	0 SS
&	371 inst	2 BTS
EF	1.154 CPI	2 BMS

Table 2. enable delay slots and forwarding factors

Config	Exec	Stalls
EF	40 cycles	0 SS
&	35 inst	0 BTS
EDS	1.43 CPI	0 BMS

Table 3. enable only delay slots

Config	Exec	Stalls
	59 cycles	0 SS
EDS	35 inst	0 BTS
	1.686 CPI	0 BMS

From the above three tables, it is noted that when the forwarding and delay slots factors are used together, better results are obtained than using only delay slots. We can say that these results may have a small change when we use another model program, but this result shows that our new mechanism of combining the forwarding and delay slots operations is the best solution as it prevents control hazard that occurs during the dynamic branch prediction. Why did we say that the result in table (2) better than the results in table (3)? Because the results in table (2) (59 cycles) i.e. less execution time than table (3), and CPI in table (2) is closed to 1, i.e. more perfect result. These results, which were obtained as shown in Table (3) values of (SS, BTS, BMS) after combining two operations, and which clearly demonstrated the effectiveness of this mechanism in which the combining of both delay slots and forwarding operation, and that's where we get a very positive result by completely avoided any branch misprediction penalty as indicated by the value of BMS in the same table.

6 CONCLUSIONS

Forwarding and delay slots operations were used together to avoid occurrence of branch misprediction penalty that occur due to control hazard between instructions of certain program. This technique produces excellent results as the occurrence of control hazard with a high degree of reliability is obtained. Also, the mechanism of this method does not work only for superscalar processors but also it can work on all types of processors that operate on the pipeline system. In future work, studying the effectiveness of this mechanism is the utilization

of cash memory, as well as using the fusion mechanism in this research to improve how this mechanism is more ideal and to avoid the floating-point operation problem.

7 References

[1] Ali S. Al-Khalid, Safaa S. Omran. August 2020, Hybrid branch Prediction for pipelined MIPS processor, Vol. 10, No. 4.

[2] Jurij Silc, Theo Ungerer & Borut Robic, 2007, Dynamic branch prediction and control speculation.

[3] Cong Thuan Do, Hong Jun Choi, Dong Oh Son, Jong Myon Kim & Cheol Hong Kim. 2014,
NTB Branch Predictor: Dynamic Branch Predictor For High-Performance Embedded Processors.

[4] DR A. P. Shanthi, Handling Control Hazards.

[5] Charles Price. 1995, MIPS IV Instruction Set Revision 3.2.

[6] Yihui He , Han Wan, Bo Jiang and Xiaopeng Gao. A Method to Detect Hazards in Pipeline Processor, MATEC Web of Conferences 139, 00085 (2017).

[7] S.A.Hudoud and A.M.Mosbah. 2014, Limiting The Data Hazards by Combining The Forwarding with Delay Slots Operations to Improve Dynamic Branch Prediction in Superscalar Processor.

[8] David J.Lilja. Reducing the branch penalty in pipelined processors.

[9] M.S. Schmalz. Organization of Computer Systems.

[10] Harich Patil and Joel S. February 2000, Emer.Combing Static and Dynamic Branch Prediction to Reduce Destructive Aliasing.

[11] Gurpur M. Prabhu. Computer architecture tutorial

[12] JAMES E. SMITH, and GURINDAR S. SOHI. The Microarchitecture of Superscalar Processors, IEEE.

[13] David Money Harris and Sarah L. 2013, Harris.Digital Design and Computer Architecture (second edition).

[14] Arthur Perais. 2016, Increasing the performance of superscalar processors through value
prediction.

[15] Craig Zilles and Gurindar Sohi. July, 2001, Execution-based Prediction Using Speculative Slices.

[16] Ali S. Al-Khalid, Safaa S. Omran. August 2020 , Hybrid branch prediction for pipelined MIPS processor Vol. 10, No. 4, pp.3476~3482.

[17] L. Hennessy and D. Patterson, 2019,Computer Architecture. A Quantitative Approach.

[18] Arthur Perais . Mar 2015, Increasing the performance of superscalar processors through value prediction.

[19] Jimenez DA, Lin C. Neural methods for dynamic branch prediction. ACM Transactions on Computer Systems, ACM Transactions on Computer Systems, Vol. 20, No. 4, November 2002..

[20] Joan Puiggali , Boleslaw K.Szymanski, Teo Jové , Jose L Marzo. Dynamic Branch Speculation in a Speculative Parallelization Architecture for Computer Clusters.

دمج عمليتي التأخير الزمني والإرسال لتجنب حدوث عواقب الخطأ التنبئي لأوامر القفز في المعالجات الفائقة

علي حدود

ali.amary81@yahoo.com قسم الحاسب الآلي، كلية الهندسة، جامعة الزيتونة، ليبيا

الملخص

إن استخدام نظام الانبوب في انظمة المعالجات الحديثة ساهمت الي حد كبير في تحسين اداء هذه المعالجات ولذلك من خلال بزيادة سرعتها بشكل أكبر بكثير من ذي قبل حيث ان معدل تنفيذ الاوامر في الدورة الزمنية الواحدة يؤول الي 1، ولكن هذه التقنية لا تخلوا من عدة مشاكل مصاحبة لهذه العملية، أحد هذه المشاكل هي العواقب الناتجة عن التنبؤ الخاطئ عند استخدام الية التنبؤ التي يتم من خلالها توجيه عنوان القفز من نقطة زمنية ما اثناء تنفيذ العمليات الي نقطة أخري حيث أن حوالي 20 % أي أمر واحد من بين خمسة أوامر يتم تنفيذها في أي برنامج هي أوامر قفز والتي توجه بألية عمل معينة للوصول الي الهدف المطلوب، والتي تقوم بعملية مقاطعة تدفق تنفيذ الاوامر التي تمر خلال هذا الانبوب. هناك عدة محاولات تم القيام بها في دراسات سابقة منها" تنبأ القفز الديناميكي وتوقع التحكم " و " متنبأ قفز ال (ان تي بي): متنبأ القفز الديناميكي لأداء المعالجات المدمجة.
في هذه الورقة قمنا بتقديم الية جديدة تعتمد علي دمج عمليتي اعادة التوجيه والتأخير في المواقع الشاغرة لتجنب اكبر قدر ممكن من فقد للزمن نتيجة التبؤ الخاطئ في المعالجات الفائقة.

الكلمات الدالة:
عواقب التنبؤ الخاطئ.
دورة ساعة لكل أمر.
مخاطر التحكم.
التنبؤ الديناميكي للفرع.
مليون أمر لكل ثانية.

*البريد الإلكتروني للباحث المراسل: *ali.amary81@yahoo.com*