International Journal of Computer Applications (0975 – 8887)
Volume 152 – No.6, October 2016

Simulation and Investigation on "Effect of Dependency in under Pipelining"

Renuka Patel
Department of Computer Science,
Pt. Ravishankar Shukla University,
Raipur, Chhattisgarh,
492010, India

Sanjay Kumar
Department of Computer Science,
Pt. Ravishankar Shukla University,
Raipur, Chhattisgarh,
492010, India

ABSTRACT

Instruction level parallelism is the most common technique to achieve speedup and Pipelining is one of the techniques to achieve Instruction level parallelism. Pipelining is of 5 types – Scalar pipelining, Superscalar pipelining, Super pipelining, Under pipelining and Super Scalar Super pipelining. In pipelining technique more than one instruction can issue simultaneously into different functional unit. But, dependency is most common problem in pipelining. This paper shows the development of simulator using 'C' language to study the effect of dependency in under pipelining. This paper also calculates some pipelining parameters like CPI, IPC etc.

General Terms

Pipelining.

Keywords

Instruction Level Parallelism, Dependency, Pipelining, Simulation, CPI, IPC, MIPS

1. INTRODUCTION

In pipelining technique, instruction execution process is divided into number of stages called pipelining stage (Load, Decode, Fetch, Execute and Write), and each stage of instruction is executed by different functional unit (Load unit, Decode unit, Fetch Unit, Decoder and Write Unit) of processor[1-3]. Figure 1 shows 5 stage pipelining.

Pipelining is the technique in which instructions are executed into overlapped cascaded manner [4-5]. Pipelining are of 5 types – Scalar pipelining, Superscalar pipelining, Super pipelining, Under pipelining and Super Scalar Super pipelining. When instruction issue latency is more than 1 clock cycle then it comes under under-pipelining architecture [6]. Processor is utilized fully when 1 instruction is issued in each clock cycle, but because of various practical reasons instruction issue latency is more than one. When instruction issue latency is more than one, then the pipeline is underutilized and this pipeline is known as under-pipeline [7]. Under pipelining is shown in figure 2. In figure 2 x axis shows clock cycles and y axis shows number of instructions. There are 2 instructions i_1 and i_2, both instructions are getting processed parallel but first instruction i1 is loading in 1st clock cycle and second instruction i_2 is loading on 3rd clock cycle because this is under-pipelining architecture. First instruction i_1 is completing in 5th clock cycle and second instruction i_2 is completing on 7th clock cycle. So total number of clock cycle to complete 2 instructions is 7 clock cycles. Here instruction issue latency is 2 clock cycles. Following are the various practical reasons for instruction issues latency is being more than one clock cycle [8-9]-

- True data dependency
- Procedural dependency
- Resource Dependence
- Output dependency

True data dependency also called write read dependency means an instruction cannot be executed until all required operands are available [10]. Instructions having branch is called procedural dependent instruction, in which the instruction cannot be completely executed until the branch is executed [9]. Resource dependence means two or more than two instructions require same resource same time. Here resource means integer units (such as integer adder), floating point units, registers, memory areas etc [7]. Output dependency also called write after write dependency. Output dependency means two instructions write into same output variable simultaneously [11]. If dependency is not handled properly then incorrect result will be generated. There are various methods available to deal with dependency. One of the simple method is pipeline stalling. Pipeline stalling means giving time delay [12]. In the present work the effect of dependency in under pipelining is studied.

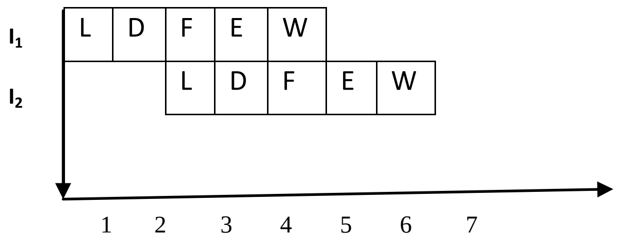


Fig. 2 Under-pipelining

2. EXPERIMENTAL METHOLOGY

This section describes the methodology used to study the effect of dependency in under pipelining. This paper considered mainly 2 conditions. First condition is current instruction is not dependent on any previous instruction and second condition is current instruction is dependent on previous instruction.

Now if current instruction is not dependent on any other previous instruction then loading of current instruction takes place on (l[i-1]+2)th clock cycle. Decoding takes place on (L[i]+1)th clock cycle. Fetching will be taken place on

(D[i]+1)th. Execution cycle depends upon type of instruction. If current instruction is addition or subtraction then, execution stage is completed on (F[i]+2)th clock cycle. If current instruction is multiplication or division then execution stage is completed on (F[i]+3)th clock cycle. If current instruction is of else category then execution stage is completed on (F[i]+1)th clock cycle. After execution, writing of ith instruction takes place on (E[i]+1)th clock cycle.

If instruction i, is dependent on only one of the previous instructions, then fetching of ith instruction will be taken place on (W[i-1]+1)th clock cycle, where W[i-1] is a dynamic array element which contains write clock cycle of previous instruction, upon which current instruction i is dependent as shown in Fig. 2. In Fig. 3 two instructions are processing I_1 and I_2 in parallel manner. I_1 is one instruction where 2 variables 'b' and 'c' are multiplying and result is stored in variable 'a', now I_2 is another instruction which need value of variable 'a' from instruction I_1 i.e. output of instruction I_1 become input of I_2, in another word I_2 is true data dependent on I_1. Then fetching of I_2 is taking place on 8th clock cycle (here 'i' is 2 so w[i-1] is w[1], w[1] is 7 so (W[i-1]+1) is 7+1 i.e. 8).

When instruction i is dependent on more than one (i-1) instructions, then take maximum of all W[i-1] (here W[i-1] contains all write clock cycles of previous instructions, in which current instruction i is dependent) and denote it as (max W[i-1]). Now fetching of ith instruction will be taken place on (max W[i-1]+1)th clock cycle provided there is no resource conflicts.

In both the conditions for every stage of every instruction, resource conflicts (Load conflict, decode unit conflicts, fetch unit conflicts, execute unit conflicts and write unit conflicts) is checked. For example before fetching of ith instruction on (e[i]+1)th clock cycle, it checks that at (e[i]+1)th clock cycle fetching of another instruction is taking place on same clock cycle or not i.e. it check fetch unit is free or not. If fetching of another instruction is not taking place on same clock cycle which means fetch unit is free then, fetching takes place on (e[i]+1)th clock cycle. If fetching of another instruction is taking place on same clock cycle then it increase the clock cycle for fetching (i.e. (e[i]+2)th clock cycle) and again it check same thing, this process is repeated until fetch unit is not free.

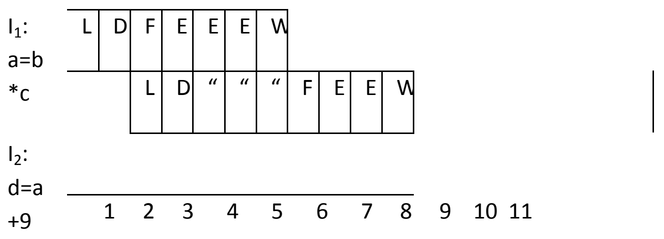


Fig. 3 Data Dependency

(I_2 Dependent on I_1)

3. RESULT AND DISCUSSION

As shown in tables (Table 1, and Table 2) as increases the number of instructions Total cycle also increases, CPI decreases and in high dependency case when increase the level of dependency CPI increases. IPC increases with number of instructions, IPC is highest for no dependency, least for highest dependency. Reason behind this is that number of idle cycles increase as level of dependency increases and also shown in table and graph1 also. For no dependency idle cycle is zero because in this situation instructions are need not to wait.

Sample program code segments for which carrying out simulation is given in Appendix A. It may be noted that when program code will be changed, values in tables will be also changed but nature of graph will not be changed.

Table 1 and Table 2 shows values of CPI, IPC, total clock cycles, idle cycles MIPS in various situations like no dependency, high dependency respectively, and subsequently graph is shown in graph 1, and graph 2.

4. CONCLUSION

In this paper, effect of dependency in under pipeline is studied. The simulation result shows that, when dependency increases, number of clock cycles taken for executing instructions also increases. This is well exhibited by the simulator as well. This simulator also shows the effect of different types of conflicts.

5. FUTURE WORK

In future, effect of dependency can be measured in other types of pipelines like scalar pipeline, superscalar pipeline, and superscalar super pipeline for analysing the effect of dependency. And can also make simulator for other scheduling policies like In-order issue with in-order completion and out-of-order issue with out-of-order completion because in this paper simulator for In-order issue with out-of-order completion is only presented.

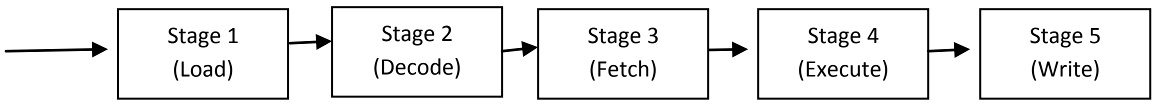


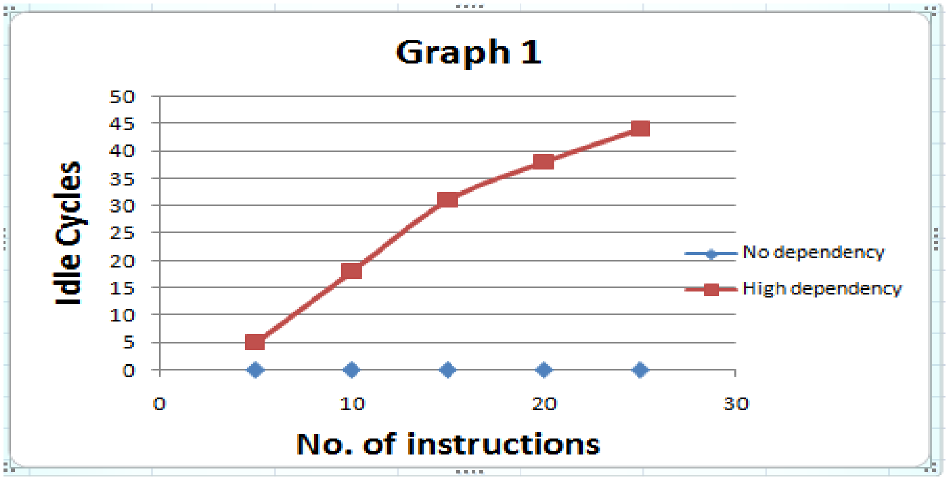
Fig1. Five Stage Pipelining

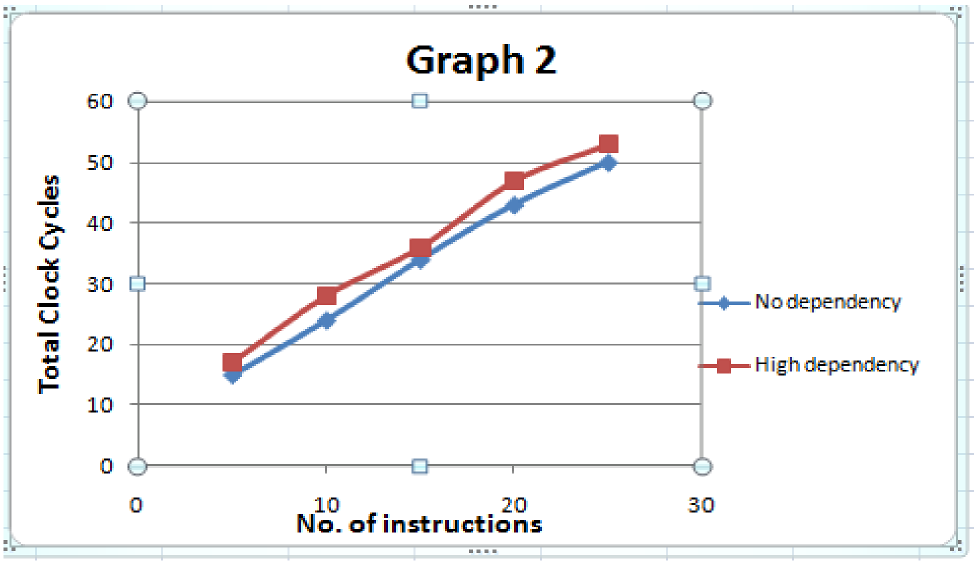
Table 1.No Dependency

No. of instructions	Total clock cycles	Idle cycles	CPI	IPC	MIPS (for 100 MHz processor)
5	15	0	3.0	0.3333	33.33333
10	24	0	2.40	0.4167	41.66664
15	34	0	2.26666	0.441176	44.117649
20	43	0	2.150	0.465716	46.57163
25	50	0	2	0.48454	48.454548

Table 2.High Dependency

No. of instructions	Total clock cycles	Idle cycles	CPI	IPC	MIPS (for 100 MHz processor)
5	17	5	3.400000	0.294118	29.411766
10	28	18	2.800000	0.35	35.71
15	36	31	2.40	0.4167	41.66667
20	47	38	2.35	0.4255	42.553192
25	53	44	2.12	0.471	47.169811





6. REFERENCES

[1] Anish Gupta,Vinayak Kini, Prathik Shetty "Five staged pipelined processor with self clocking mechanism", ISBN: 978-1-4673-7910-6 USB ISBN: 978-1-4673-7909-0, IEEE Xplore, JAN 14 2016.

[2] M. Flynn, Computer Architecture—Pipelined and Parallel Processor Design, Jones and Bartlett Publishers, Boston, 1995

[3] Martti forsell" Implementation of Instruction Level and Thread Level Parallelism in Computers" ISSN 1238-6944, ISBN 951-708-557-5, pp 3.

[4] L John. Hennessy ,"VLSI Processor Architecture", IEEE Transactions on Computers, VOL. c-33, No. 12, December 1984

[5] Simran Rana Rajesh Mehra "Hyper Pipelined RISC Processor Implementation- A Review",International Journal of Engineering Research & Technology (IJERT) Vol. 2 Issue 10, October - 2013 IJERTIJERT ISSN: 2278-0181

[6] Jouppi N. P. and Wall D. W., "Available Instruction-Level Parallelism for Superscalar and Superpipelined Machines" Digital Western Research Laboratory, Tech. Rep. 89/7, Jul. 1989

[7] Hwang K.,"Advanced Computer Architecture" Tata Mc Graw Hill 2001, pp 160, 54,

[8] Stallings W. ,"Computer Organization and Architecture " Pearson Education 2010, ISBN 978-81-7758-993-1, pp 504-510

[9] Johnson W.M., "Super-Scalar Processor Design" Technical Report No.CSL-TR-89-383, June 1989, pp 8-9.

[10] Andreas Moshovos, Scott E. Breach, T. N. Vijaykumar, Gurindar S. Sohi "Dynamic Speculation and Synchronization of Data Dependences", Proceedings of the 24th Annual International Symposium on Computer Architecture.

[11] Henry Styles, David Barrie Thomas and Wayne Luk ,"Pipelining Designs with Loop-Carried Dependencies", http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.134.8880&rep=rep1&type=pdf

[12] Saravanan V., Kothari D. P. and Woungang I.," An optimizing pipeline stall reduction algorithm for power and performance on multi-core CPUs" Human-centric Computing and Information Sciences , SpringerOpen

7. APPENDIX A

Program code for No dependency case:

1. Print "hello"
2. Print "this is second instruction"
3. a = 8679-456
4. b = 8+2
5. c=d*5
6. a1=98+09
7. e=55*5
8. y=p+2
9. h=i*5
10. j=33.7-3
11. z=9+57
12. l=69-5
13. m=78*4
14. b1=78+b7
15. h=15-10
16. n=56-9
17. y=78+8
18. o=P+q
19. r=Q+3
20. Print "This is 20 cc"
21. t=10-4
22. u=100*765
23. g1=h1+9
24. X=88*82
25. S=98*90

Program code for High dependency case :

1. a=b*c
2. d=5+87
3. e=10
4. f=d*e
5. Print "f"
6. g=h-i
7. j=g*e
8. k=j
9. m=k+1
10. l=d
11. n=12
12. o=k+m
13. p=k-m
14. Print "p"
15. Q=15
16. r= s/t
17. Print "r"
18. U=a-d
19. V= Q*100
20. W=V+59
21. X=V*34
22. Y=500
23. Print "Y"
24. Z=X
25. Print